# Studi Mengenai Serangan pada Fungsi *Hash* yang Digunakan pada Otentikasi di Aplikasi Web

Khairul Fahmi, 13507125
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*If17125@students.if.itb.ac.id*

***Abstract*—Penggunaan aplikasi berbasis web sudah sangat luas diadopsi. Beberapa aplikasi membutuhkan layanan yang terpersonalisasi, yaitu layanan yang unik untuk masing-masing pengguna. Untuk memungkinkan layanan tersebut, digunakan otentikasi menggunakn pasangan username dan *password*. Pada aplikasi web, *password* dan username disimpan dalam basisdata. *Password* biasanya disimpan dalam bentuk terenkripsi dengan fungsi *hash*(fungsi satu arah). Fungsi *hash* tidak memungkinkan penyerang membuat fungsi inverse untuk menemukan plainteks dari nilai *hash* yang ada. Kesulitan ini diatasi oleh penyerang dengan metode penyerangan *table lookup*. Membuat *table lookup* yang jumlah entrynya sangat besar membutuhkan computasi yang besar. Penggunaan aplikasi tersebar yang berkolaborasi dapat memperkecil waktu yang dibutuhkan proses pembuatan *table lookup*. Google dapat digunakan untuk mempermudah pencarian pada file *table lookup* yang ukurannya sangat besar.**

***Index Terms*—Aplikasi web, fungsi *hash*, Google search, otentikasi, *table lookup*.**

## I. PENDAHULUAN

### 1. Web Aplikasi

Dalam rekayasa perangkat lunak, suatu aplikasi web (bahasa Inggris: web application atau sering disingkat webapp) adalah suatu aplikasi yang diakses menggunakan penjelajah web melalui suatu jaringan seperti Internet atau intranet. Ia juga merupakan suatu aplikasi perangkat lunak komputer yang dikodekan dalam bahasa yang didukung penjelajah web (seperti HTML, JavaScript, AJAX, Java, dll) dan bergantung pada penjelajah tersebut untuk menampilkan aplikasi.

Aplikasi web menjadi populer karena kemudahan tersedianya aplikasi klien untuk mengaksesnya, penjelajah web, yang kadang disebut sebagai suatu thin client (klien tipis). Kemampuan untuk memperbarui dan memelihara aplikasi web tanpa harus mendistribusikan dan menginstalasi perangkat lunak pada kemungkinan ribuan komputer klien merupakan alasan kunci popularitasnya. Aplikasi web yang umum misalnya webmail, toko ritel daring, lelang daring, wiki, papan diskusi, weblog, serta MMORPG.

### 2. Fungsi *Hash*

Fungsi *hash* Kriptografis adalah fungsi *hash* yang memiliki beberapa sifat keamanan tambahan sehingga dapat dipakai untuk tujuan keamanan data. Umumnya digunakan untuk keperluan autentikasi dan integritas data. Fungsi *hash* adalah fungsi yang secara efisien mengubah string input dengan panjang berhingga menjadi string output dengan panjang tetap yang disebut nilai *hash*.

Sifat-Sifat Fungsi *Hash* Kriptografi

- Tahan preimej (Preimage resistant): bila diketahui nilai *hash* h maka sulit (secara komputasi tidak layak) untuk mendapatkan m dimana h = *hash*(m).
- Tahan preimej kedua (Second preimage resistant): bila diketahui input m1 maka sulit mencari input m2 (tidak sama dengan m1) yang menyebabkan *hash*(m1) = *hash*(m2).
- Tahan tumbukan (Collision-resistant): sulit mencari dua input berbeda m1 dan m2 yang menyebabkan *hash*(m1) = *hash*(m2)

Beberapa contoh algoritma fungsi *hash* Kriptografi:

- MD4
- MD5
- SHA-0
- SHA-1
- SHA-256
- SHA-512

### 3. Otentikasi

Log masuk (login, juga biasa disebut sebagai log in, log on, logon, signon, sign on, signin, sign in) adalah proses untuk mengakses komputer dengan memasukkan identitas dari akun pengguna dan kata sandi guna mendapatkan hak akses menggunakan sumber daya komputer tujuan.

Untuk melakukan log masuk ke sistem biasanya membutuhkan:

- Rekening pengguna; digunakan sebagai

identitas berupa runtutan karakter yang secara unik merujuk ke pengguna tertentu.

- Kata sandi; runtutan karakter berupa kunci yang dijaga kerahasiaannya terhadap orang lain.

Kedua pasang runtutan karakter itu harus tepat dan keduanya adalah pasangan yang tidak bisa dipisahkan. Kata sandi dapat berubah sesuai dengan kebutuhan, sedangkan akun pengguna tidak pernah diubah karena berupa identitas unik yang merujuk ke pengguna tertentu.

Proses ini akan membuat sesi pada mesin tujuan untuk pengguna yang melakukan log masuk. Dalam kasus pengaksesan situs Internet, situs acapkali meletakkan cookies pada komputer pengguna.

### 4. Tabel *lookup*

Tabel *lookup* merupakan sebuah daftar yang berisi kumpulan pasangan

index → *value*

Tabel *lookup* digunakan dalam cryptanalysis dengan menggunakan prinsip Time-Memory-*Processor* Tradeoffs [1]. Cryptanalyst melakukan *lookup* pada *table* untuk melakukan dekripsi dari sebuah chipertext.

## II. Isi

Saat ini aplikasi berbasis web sudah diadaptasi secara luas. Aplikasi web bisa ditemukan di bank, rumah sakit, pemerintah, bisnis berbasis internet, dunia pendidikan, dunia entertainment, dan tempat lain yang tidak mungkin dituliskan semuanya disini. Lebih dari 50% aplikasi web yang pernah saya gunakan memberikan layanan yang unik terhadap masing-masing pengguna. Untuk menyediakan kemampuan personalisasi layanan terhadap pengguna, pengembang aplikasi bisa memanfaatkan mekanisme otentikasi yang menggunakan username dan passoword.i Fungsi otentikasi yang kedua adalah untuk membatasi hak akses. Pengguna hanya boleh melakukan operasi atau menggunakan resources yang menjadi haknya.

Cara otentikasi paling umum digunakan pada aplikasi adalah dengan membuat session untuk sebelum pengguna memanfaatkan layanan. Username dan *password* yang diberikan oleh pengguna dibandingkan dengan yang ada di basisdata. Jika berhasil maka session akan tercipta. Sekarang pengguna bisa menggunakan layanan yang disediakan oleh aplikasi. *Password* disimpan dalam basisdata tidak dalam bentuk plainteks, melainkan terenkripsi dalam bentuk *hash value*. Jika *password* ini jatuh ke tangan orang yang tidak bertanggung jawab, penyerang(selanjutnya kita sebut *cracker*) tidak bisa berbuat apa-apa terhadap *password*. Jaminan keamanan ini diperoleh dari desain fungsi *hash* yang satu arah(yaitu jika diketahui sebuah nilai *hash* h, tidak mungkin membuat fungsi inversi sehingga f(h) = p, dimana p adalah plaintext yang bersesuaian dengan nilai *hash* h).

Fungsi *hash* secara teori tidak bisa dipecahkan dengan membuat fungsi inverse *hash*. Sebuah cara yang bisa dilakukan untuk melakukan *password cracking* adalah dengan menggunakan *table lookup*. *Cracker* membangkitkan nilai-nilai *hash* untuk masukan berupa kombinasi alfa numeric dan simbol-simbol dengan panjang 1-n. Hasil pembangkitan ini disimpan dalam sebuah daftar berupa daftar pemetaan dari pasangan

*Hash value* → plaintext.

Untuk melakukan *password cracking*, *cracker* cukup melakukan *lookup* terhadap nilai *hash* yang dia punya. Dari hasil *lookup* ini akan diperoleh plaintext yang bersesuaian dengan *hash value*(yaitu h = *hash*(p)) tanpa harus membuat fungsi inverse dari fungsi *hash*.

*Password cracking* dengan cara ini jauh lebih cepat dibandingkan melakukan exhaustive search atau brute force setiap saat dibutuhkan melakukan *cracker* melakukan *password cracking*. Ide dari penggunaan *table lookup* ini berasal dari [1]. Cara yang ditawarkan pada [1] masih berupa teori yang belum dioptimasi. Sumber[2] mencoba meberikan solusi yang telah teroptimasi sehingga pembangkitan *table lookup* lebih handal.

Penggunaan metode *table lookup* untuk melakukan *password cracking* pada *password* yang dienkripsi dengan menggunakan fungsi *hash* bukanlah tanpa masalah. Permasalahan pertama pada metode *table lookup* ada pada proses pembangkitan *table*. Untuk kunci dengan panjang 4-10 dengan domain input

{A-Z,a-z,0-9,space, dash}

Kemungkinan jumlah *hash* yang harus dibangkitkan adalah

$$n = \sum_{i=4}^{10} 64^i \qquad (1)$$

Untuk domain input 64 buah kemungkinan dan panjang *password* yang akan dipecahkan 4-14, total jumlah pembangitan nilai *hash* adalah sekitar

***n = 6.51461E18 = 6514610000000000000 kali***

Jika panjang input adalah 4 sampai 10 karakter, maka panjang bit input rata-rata adalah

***(10+4)*0.5 * 8 = 56 bit.***

Total digest yang harus di-*generate* untuk input seperti diatas adalah

$$\begin{aligned} \text{Total size} &= 6.5146 \times 10^{18} \cdot 56 bit \\ &= 3{,}6481 \times 10^{19} \text{ bit} \\ &= 4.25 \times 10^{10} GB \end{aligned} \qquad (2)$$

Untuk salah satu algoritma *hash* terbaru, yaitu MD6, untuk panjang *diggest* 256 bit, sebuah computer dengan *processor* 16 core mampu melakukan operasi 1GB/s.[3]

Pada pengujian lain dengan menggunakan hardware PS3, dan fungsi md5[4], throughput yang dihasilkan adalah 88GB/s.

Untuk menyelesaikan pembangkitan semua nilai *hash* pada kasus di atas dengan menggunakan PS3(untuk algoritma MD5) dibutuhkan waktu sekitar

$$Waktu = \frac{4.25 \times 10^{10} GB}{3600 \cdot 24 \cdot 365 \cdot 1 GB/s} \quad (3)$$
$$= 1.3467 \times 10^{3} tahun$$

Jika pembangkitan ini dilakukan secara sendiri, waktu ini sangatlah tidak feasible. Sampai matipun *table lookup* tidak akan selesai dibangkitkan. Akan tetapi jika dilakukan bersama-sama dengan membangun aplikasi terdistribusi, misalnya dengan memanfaatkan 10.000 PS3 yang saling berkolaborasi dengan memggunakan aplikasi terdistribusi, waktu yang dibutuhkan adalah lebih-kurang

$$Waktu = \frac{1.3467 \times 10^{3} tahun}{10000} \quad (4)$$
$$\cong 50 hari$$

Begitu *table* berhasil dibangkitkan, *password cracking* dapat dilakukan dengan cara *lookup* pada *table lookup*.

Permasalahan kedua adalah metode *lookup*. Dengan ukuran *table lookup* yang sangat besar (ukuran *table lookup* yang disimpan lebih kecil dari hasil perhitungan pada (2), yaitu dengan menggunakan metode penyimpanan yang lebih efisien), dibutuhkan sebuah algoritma pencarian string yang sangat bagus. Permasalahan ini dapat diatasi dengan memanfaatkan mesin pencari Google.

Hal ini dimungkinkan dengan penggunaan aplikasi terdistribusi untuk melakukan proses pembangkitan *table lookup*.

Berikut ini arsitektur sistem yang saya rancang untuk melakukan tugas diatas.

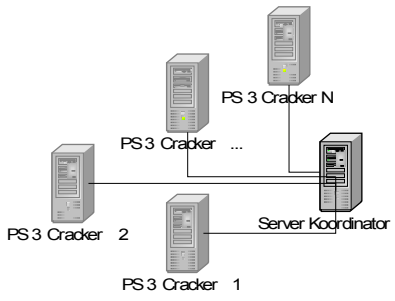


**Fig 1 Arsitektur Sistem Pembangkitan**

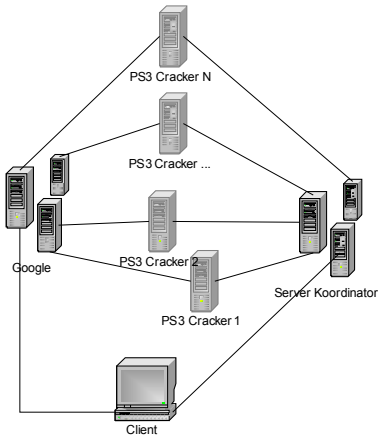


**Fig 2 Arsitektur Lengkap**

Langkah-langkah untuk melakukan ini adalah sebagai berikut:

- *Cracker* ke-N melakukan pembangkitan *table lookup*. Pembagian kerja diatur oleh server coordinator. Setelah *table lookup* (partial) berhasil dibangkitkan, Craker-N melaporkan ke server pusat lokasi url penyimpanan file *table lookup*.
- Carcker ke-N membuat file *table lookup* yang dibangkitkan bisa di-*index* oleh Google. Dengan memanfaatkan Google, *cracker* tidak perlu membuat tool pencarian khusus. Selain itu file yang disebar di internet memberikan kemudahan dalam penyimpanan. *Cracker* tidak harus manyimpan semua *table lookup* pada satu tempat terpisah. Untuk memudahkan pencarian, client bisa melakukan pencarian lewat mesin pencarian Google biasa atau server pusat membuat *Custom Search* yang disediakan oleh Google. Custom Search ini memiliki skop tempat file-file *table lookup*
- Pengguna dari system dapat melakukan *query* di *Google Search* dengan kata kata kunci nilai *hash* yang akan dicari plainteksnya atau dengan melakukan query pencarian pada Custom Search pada server pusat.

## III. PEMBAHASAN

Sistem yang dirancang merupakan sebuah framework system aplikasi terdistribusi untuk melakukan *cracking password* yang terenkripsi dengan fungsi arah. Framework dirancang *generic*. Untuk memecahkan fungsi

*hash* yang berbeda, framework ini masih dapat digunakan.

Fungsi *hash* yang berbeda bisa diimplementasikan dengan menyesuaikan aplikasi terdistribusi yang digunakan, sementara bagian lain masih tetap.

Untuk memperoleh system pembangkitan yang efisien dibutuhkan koordinasi yang bagus antar participant(disini adalah *Cracker* dengan infrastructure yang tergabung dalam system). Sebuah server coordinator berguna untuk mengatur *siapa* yang mengerjakan *apa*. Jika *Cracker* ke-i sudah membangkitkan *table lookup* dengan masukan *xxxxx*, maka *cracker* ke-j tidak perlu melakukan proses yang sama. Disinilah peran dari coordinator. Server coordinator menyimpan daftar *siapa* mengerjakan *apa*, dan *apa* saja yang telah selesai dikerjakan sehingga begitu ada contributor baru yang bergabung, dia tahu *apa* yang harus dikerjakan.

Dari sisi hardware, implementasi system tidak harus menggunakan PS3. PS3 dipilih karena saat ini hardware yang telah teruji handal yang penulis ketahui salah satunya adalah PS3. Seiring dengan berjalannya waktu, tentu saja suatu saat nanti akan bermunculan hardware yang jauh lebih handal. Sistem dapat diupgrade untuk memperoleh performansi yang lebih bagus.

Pada paper ini, panjang kunci yang digunakan masih terbatas(maksimal 10 karakter). Kedepannya jika teknologi hardware sudah lebih bagus dan teknologi penyimpanan lebih handal dari sekarang, panjang kunci yang dibangkitkan bisa lebih panjang dan kombinasi masukan lebih banyak. Bukan mustahil sebuah fungsi satu arah yang bisa dipecahakan dalam hitungan hari setelah fungsi tersebut di-publish ke public.

Pada system ini untuk melakukan *lookup* digunakan *Google Search Engine(GSE)*. Pemilihan penggunan GSE adalah karena Google telah terbukti memiliki teknologi pencarian yang bagus.

## IV. Kesimpulan

Dengan menggunakan system yang telah dijelaskan pada paper, sebuah fungsi satu arah dapat dipecahkan dalam waktu yang relative cepat. Teknologi yang digunakan sangat mempengaruhi kehandalan dari system.

## References


H. R. Amirazizi , M. E. Hellman, "Time-Memory-*Processor* Tradeoffs," IEEE Trans. on Info. Theory, Vol. 34, pp. 505-512, May 1988. published

Philippe Oechslin, *Making a Faster Crytanalytical Time-Memory Trade-off*. Proceedings. Lecture Notes in Computer Science 2729 Springer 2003, ISBN 3-540-40674-3, published

Ronald L. Rives, *The MD6 hash function, A proposal to NIST for SHA-3* http://cado.gforge.inria.fr/workshop/slides/bos.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, 16 Mei 2010

Khairul Fahmi
13507125