Studi Serangan Terhadap Digital Signature dengan Memanfaatkan MD5 Message Digest

Glen Christian (13505098)
Teknik Informatika ITB, Bandung 40132, e-mail: if15098@students.if.itb.ac.id

Abstrak – Dalam melakukan hal ini digunakan pengetahuan mengenai MD5 Colission. Untuk menujukkan contoh dari pasangan binary self-extract packages denga MD5 checksum yang sesuai, dimana hasil ekstraksi memiliki arti yang berbeda secara mendasar. Kedua, akan dilakukan demonstrasi dimana seorang penyerang dapat membuat pasangan paket data yang terdiri dari file-file yang ditentukan oleh pnyerang tersebut dengan MD5 checksum yang sama diaman tiap paket data diekstraksi menjadi file yang berbeda. Setelah algoritma untuk mencari MD5 collisions dipublikasikan, serangan yang jauh lebih efektif dapat dilakukan. Suatu scenario yang nyata dari serangan disusun secara teratur dan sistematis.
Kata Kunci: collision, hash, MD5

1. Pendahuluan

MD5 adalah fungsi hash yang paling umum dipakai pada saat ini. MD5 didesain oleh Ronald Rivest. Pada saat sekarang ini, MD5 banyak sekali digunakan dalam melakukan integrity checking sebagai message digest dalam dunia kriptografi khususnya digital signature.

Pada tahun 2004, pada saat konferensi CRYPTO 2004 dilakukan, sebuah tim kriptographer dari China menunjukkan bahwa mereka mengetahui suatu algoritma untuk mencari sepasang colliding message, maksudnya adalah pesan yang berbeda dengan checksum MD5 yang sama. Dan yang lebih mengagetkan lagi, algoritma tersebut hanya membutuhkan waktu 1 jam untuk menemukan hal tersebut. Hal ini memberikan suatu konklusi bahwa MD5 tidak lagi merupakan cara yang aman untuk digunakan sebagai integrity checking, karena dua colliding file dapat dibuat tanpa mengubah nilai hash.

Dibawah ini akan diberi contoh dua pasang colliding string dari tim kriptografi dari China yang memiliki nilai hash yang sama. Selain itu pada bagian selanjutnya akan dilakukan demonstrasi dampaknya pada dunia digital signature, terutama dengan menggunakkan GPG.

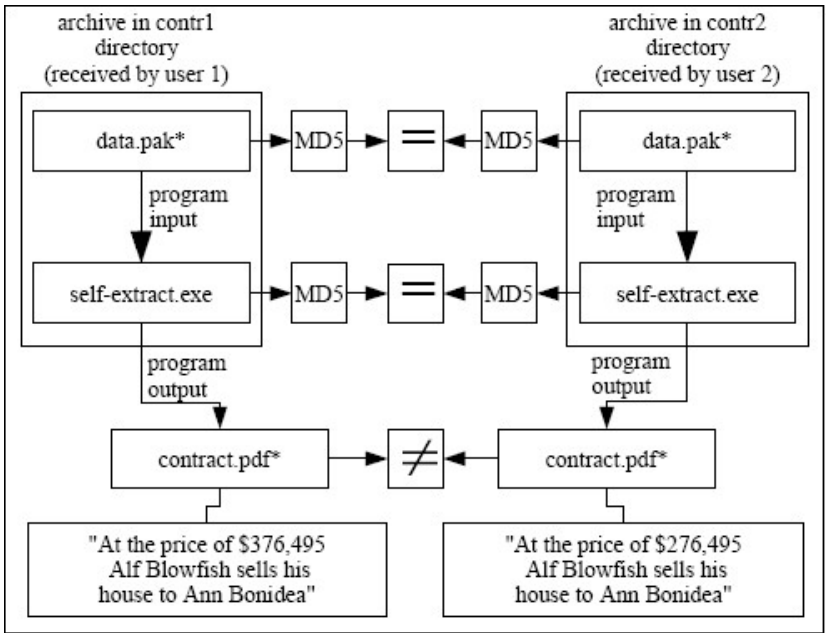
1.1 Cara Kerja

Dalam contoh ini digunakan sebuah self-extract archive untuk demonstrasi dari collisions. File zip atau rar dan ekstensi laninnya sangat umum digunakan untuk mereduksi ukurang file.

Dari sudut pandang pengguna dalam melihat situasi ini, pengguna mendapat dua file yaitu self-extract.exe dan data.pak. Pengguna dapat melakukan integrity checking dengan cara mengecek nilai MD5 dari kedua buah file. Pengguna akan melakukan eksekusi self-extract.exe dan program menggunakkan data.pak untuk mengekstraksi dirinya, tetapi dengan file data.pak yang berbeda. Kedua file data.pak diciptakan sehingga kedua MD5 checksum yang dimiliki keduanya identik. Lebih jauh lagi, kedua pengguna berpikir isi file yang terekstraksi adalah sama pada kedua kasus tersebut.

Dalam archive homepage[3], terdapat dua buah directory yaitu contr1 dan contr2. Kedua hal tersebut menunjukkan file terkirim dari kedua perspektif pengguna yang berbeda. Untuk melakukan pengetesan terhadap program, harus dilakukan simulasi masuk kepada kedua buah direktori dan melakukan eksekusi file self-extract.exe.

Program self-extract.exe melakukan pembacaan terhadap file data.pak menggunakkan bit yang spesifik dari colliding block sebagai indeks yang mengacu kepada suatu tabel untuk memutuskan file manakah yang diekstraksi. Block data yang menyebabkan collision berlokasi pada awal tiap file dari data.pak. menggunakkan satu atau file lain data.pak pada archive (pada direktori contr1 dan contr2), pada gambar kita akan mendapat dua buah isi contract.pdf yang berbeda. Dapat dilihat pada gambar di bawah, kedua file executable dan data file memiliki nilai MD5 yang sama.



2. Perangkat untuk membuat Sepasang Archive

Bagian ini menjelaskan langkah bagaimana seorang dapat membuat sepasang archive berpasangan yang mempunyai isi dokumen yang berbeda, namu memiliki nilai MD5 sum yang sama

Dokumen pada homepage[3] memiliki sebuah program bernama create-package. Untuk platfor UNIX, dokumen tersebut dapat decompile dari sumbernya demgan menggunakkan Makefile, untuk platform Windows, terdapat sebuah program biner bernama create-package.exe. Penggunaan dari program ini contohnya:

```
create-package outfile infile1 infile2
```

Dimana: outfile menyatakan bagaimana file tersebut harus diberi nama (setelah menggunakkan program self-extract), infile1 dan infile2 menyatakan dua buah arbitrary file yang disimpan dalam package tersebut. Program ini akan membuat file data1.pak dan data2.pak yang disimpan kedalam direktori contr1 dan contr2 pada archive tersebut, dan diganti namanya menjadi data.pak. Program self-extract dijalankan setelahnya akan membuat file yang diberi nama outfile, berisi data infile1 atau infile2, contohnya:

```
create-package contract.pdf
contract1.pdf contract2.pdf
```

akan mengambil dua buah file yaitu contract1.pdf dan contract2.pdf, memasukan mereka kedalam file data1.pak dan data2.pak. Kedua file tersebut ketika digunakan oleh program self-extract, akan membuat file baru yaitu contract.pdf, yang pertama terdiri dari data contract1.pdf, yang kedua terdiri dari data contract2.pdf. Dapat dilihat bahwa kedua file akan memiliki nilai MD5 sum yang sama

Table 1. Layout of data.pak file

Size in bytes	Data stored
128	colliding block
1	filename length - *fnamelen*
fnamelen	filename to be extracted
4	32-bit integer size of first stored file - *filesize1*
4	32-bit integer size of second stored file - *filesize2*
filesize1	data of file1
filesize2	data of file2

Hal ini bekerja dengan menggunakkan layout dari file data.pak yang diberikan di tabel 1. Setiap file data.pak memiliki colliding block yang berbeda. Hanya satu pasang binary strings yang diberikan oleh kriptografer dari China[2]. Sisanya dari kedual buah file data.pak selalu sama.

Ketika melakukan komputasi nilai MD5 pada file data.pak, colliding block yang pertama (1024bit pertama) akan menyebabkan nilai hash menjadi identik.

Program self-extract akan memutuskan file manakah yang akan di-extract berdasarkan satu bit dari colliding block (yang akan menyebabkan perbedaan pada file data.pak), bit yang digunakan di define sebagai:

```
#define MD5_COLLISION_OFFSET 19
#define MD5_COLLISION_BITMASK 0x80
```

Kedua nilai di atas menyatakan posisi dan mask dari differing bit. *MD5_COLLISION_OFFSET* adalah indeks dari differing byte pada file data.pak. Seperti yang dapat dilihat, hal ini terdapat pada 128bytes dari colliding block. File executable self-extract.exe itu sendiri sama untuk kedua buah package, karena itu memiliki nilai hash MD5 yang sama.

Kode yang relevan adalah:

```
uint8_t colliding_byte;
//seek to and read the byte where
MD5 collision occurs
packedfile.seekg(MD5_COLLISION_OFF
SET, ios::beg);
packedfile.read((char
*)&colliding_byte, 1);
```

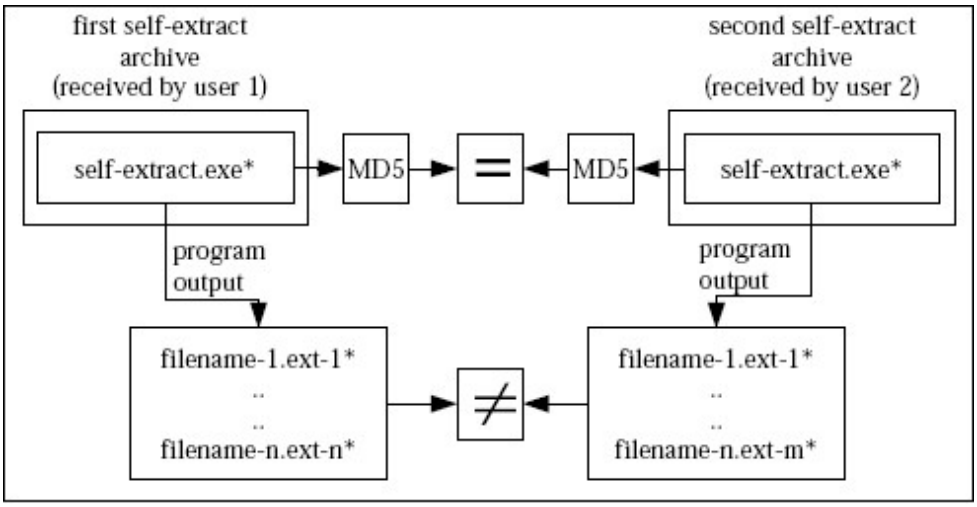
Hal yang menentukan file yang akan diekstraksi:

```
//use boolean value of colliding
byte vs. bitmask
//comparison as index to filetable
unsigned int fileindex =
(colliding_byte &
MD5_COLLISION_BITMASK) ? 1 : 0;
```

Sekarang program akan mengetahui, file manakah yang akan diekstraksi, berdasarkan posisi yang benar dalam file package dan melakukan ekstraksi sesuai dengan jumlah byte yang benar menjadi sebuah output file.

3. Meningkatkan Serangan

Pada bagian sebelumnya telah dijelaskan bagaimana melakukan serangan dengan menggunakkan dua buah file. Kita harus menggunakkan string yang disebut sebagai collision pada MD5 hash. Hal tersebut membatasi kita untuk meletakkan colliding block pada awal file. Ketika algoritma untuk mencari MD5 collision untuk initialization vector apapun diketahui, colliding block dapat diletakkan pada posisi manapun pada file. Pengetahuan seperti ini menyebabkan dua buah file executable dapat dibuat, keduanya memiliki nilai MD5 sum yang sama, biarpun setiap file melakukan hal yang berbeda bergantung pada pilihan si penyerang. Dua pengguna menerima dua file self extract dengan nilai MD5 sum yang sama. Setiap mereka akan melakukan ekstraksi file yang berbeda (gambar 2). Biasanya digunakan self-extract executable pada sebuah file sehingga tidak akan dicurigai. Satu-satunya hal yang harus dipenuhi adalah, blok harus dibatasi 512bit. (kebutuhan ini adalah konsekuensi dari algoritma MD5).



Sekarang, hanya satu file executable self-extract yang dibutuhkan. Kedua self-extract executable memiliki nilai MD5 hash yang sama, biarpun kedua program tersebut akan memberikan output

yang berbeda saat dijalankan, nama file sama tetapi isi berbeda.

3.1 Struktur File Executable

Ketika sebuah source code dikompilasi dan dijalankan, hasil keluaran yang diberikan memiliki keluaran berupa:

Table 2. Layout of an executable file

Header	Static data	Machine code

Header biasanya berisi data yang bersifat platform-spesific, biasanya digunakan oleh loader operating system (contohnya relocations, code start position). Data yang static berisi variable dan hasil yang dikenal sebagai linking time. Kode mesin dibangkitkan oleh compiler dan linker dari sebuah source code.

Kunci dari ide ini adalah meletakkan colliding block di dalam data section yang bersifat static dimana dapat diakses oleh program code. Asumsi, colliding block memiliki panjang 1024 bits. Untuk mencapai kondisi tersebut, seorang code snippet melakukan hal ini:

```
#define MD5_COLLISION_BLOCK_ALIGN
512
#define MAGIC_OFFSET 0x41424344
#define MAGIC_BITMASK 0x45464748
#define MAGIC_BLOCK 0x494a4b4c

unsigned char MD5_CollisionBlock[
(MD5_COLLISION_BLOCK_BITS+MD5_COLL
ISION_BLOCK_ALIGN)/8
] = {MAGIC_BLOCK, 0, 0, 0};
unsigned int MD5_CollisionOffset =
MAGIC_OFFSET;
unsigned int MD5_CollisionBitMask
= MAGIC_BITMASK;
```

Catatan pertama adalah yang disebut dengan “magic” constants. Tujuannya adalah agar dapat menemukan posisi secara cepat pada executable file (beberapa linker mungkin dapat menemukan lokasi dari hal ini secara langsung, beberapa lainnya tidak). Variable tersebut static dan linker akan meletakkan mereka dalam static data section.

Variabel MD5_CollisionBlock adalah sebuah array dengan panjang 1024+512bit. Hanya perlu digunakan 1024bit dari mereka, sisanya adalah padding block sehingga colliding block akan memenuhi constraints 512-bit.

MD5_CollisionOffset dan MD5_CollisionBitMask menyatakan posisi unik dari differing bit dalam array MD5_CollisionBlock.
Setelah itu bit tsersebut dapat diakses dan membuat keputusan dengan cara:

```
bool decision()
{ return (
MD5_CollisionBlock[MD5_CollisionOf
fset] &
MD5_CollisionBitMask);
}
void some_function()
{
if (decision()) do_good_thing();
else do_bad_thing();
}
```

Setelah dikompilasi dan dilakukan linking pada program, kita dapat menemukan posisi dari tiga variable dalam hasil executable file dan melakukan update ketika diperlukan. Pertama-tama kita melakukan komputasi dari colliding block untuk diletakkan pada array MD5_CollisionBlock:

1. Misalkan I menyatakan indeks dari bit pertama pada file executable (dihitugn dari 0) dimana colliding blocks harus ditimpa.
2. Kita mengambil byte dari indeks 0..I-1 dari file executable dan melakukan komputasi MD5. Hasil komputasi tersebut disebut dengan H.
3. H digunakan sebagai nilai initialization vector ketika kita melakukan komputasi dari collision block.
4. Colliding block diletakkan sebagai indeks I dari file executable.

Setelah itu, nilai dari variable MD5_CollisionOffset dan MD5_CollisionMask diganti ke lokasi yang benar. Diasumsikan linker akan meletakkan variable kedalam file executable sesuai dengan urutan saat kita melakukan deklarasi variable. Hal ini akan memungkinkan penggantian offset dan bitmask tanpa melakukan penghitungan ulang collision blocks.

Hal ini sangat simple dan mudah dilakukan. Suatu kode dapat mengundang kecurigaan. Bagaimanapun, menghindari hal ini tidak begitu sulit dan dapat dilakukan dengan cara-cara yang berbeda: banyak perusahaan me-release hanya software saya tanpa source code, kode dapat dimanfaatkan untuk hal-hal negative contohnya menggunakkan Buffer Overflow untuk membaca variable-variable lainnya.

4. Contoh Serangan Digital Signature dengan GPG

Contoh dari bagian dua. Setiap file (self-extract dan data.pak) diberi tanda tangan digital yang berbeda

secara terpisah dengan algoritma MD5 sebagai message digest. GPG 1.2.4 untuk Linux digunakan dalam serangan ini.

Pertama-tama dilakukan pembangkitan kunci privat:

```
gpg --gen-key
```

Nilai dari data ini digunakan pada RSA:

```
Real name: Collision test key
Email address: collision@md5.abc
Comment: Used to demonstrate
signature collision when
using MD5 algorithm
```

Akan dilakukan pemberian tanda tangan digital pada kedua buah file yaitu data.pak dan file self-extract. Pada dua buah direktori yaitu contr1 dan contr2 dilakukan tanda tangan digital:

```
gpg -u collision --digest-algo md5
-ab -s data.pak
gpg -u collision --digest-algo md5
-ab -s self-extract
```

Maksud parameter diatas adalah:
-u memilih kunci untuk melakukan sign
-digest-algo memilih algoritma pemberian tanda tangan digital
-a berarti output berupa ASCII
-b berarti untuk membuat “detached signature” (hasilnya adalah file hanya berisi signature tanpa file original)
-s berarti command “sign”
Parameter terakhir adalah file yang akan diberi tanda tangan digital.

Sekarang, dilakukan verifikasi terhadap file yang diberi tanda tangan digital, dan berhasil:

```
gpg -v --verify data.pak.asc
data.pak
```

Parameter di atas berarti:
-v memberitahu GPG untuk bersifat “verbose”
--verify adalah perintah untuk melakukan verifikasi
Nama file pertama adalah file signature
Nama file kedua adalah file yang diverifikasi

Hasil:

```
gpg: armor header: Version: GnuPG
v1.2.4 (GNU/Linux)
gpg: Signature made Tue Nov 23
21:54:28 2004 CET using
RSA key ID 3E200834
gpg: Good signature from
"Collision test key (Used to
demonstrate signature collision
when using MD5
algorithm) <collision@md5.abc>"
gpg: binary signature, digest
algorithm MD5
```

Bagaimanapun, apabila kita melakukan pengetesan terhadap file data.pak lainnya pada direktori contr2, akan memberikan hasil verifikasi berhasil biarpun sebenarnya file tersebut berbeda.

```
gpg -v --verify data.pak.md5.asc
../contr2/data.pak
```

Hasilnya:

```
gpg: armor header: Version: GnuPG
v1.2.4 (GNU/Linux)
gpg: Signature made Tue Nov 23
21:54:29 2004 CET using
RSA key ID 3E200834
gpg: Good signature from
"Collision test key (Used to
demonstrate signature collision
when using MD5
algorithm) <collision@md5.abc>"
gpg: binary signature, digest
algorithm MD5
```

Melakukan pengetesan file self-extract pada kedua direktori, akan berhasil diverifikasi karena mereka identik:

```
gpg -v --verify self-extract.asc
self-extract
```

Hasilnya:

```
gpg: armor header: Version: GnuPG
v1.2.4 (GNU/Linux)
gpg: Signature made Tue Nov 23
21:55:02 2004 CET using
RSA key ID 3E200834
gpg: Good signature from
"Collision test key (Used to
demonstrate signature collision
when using MD5
algorithm) <collision@md5.abc>"
gpg: binary signature, digest
algorithm MD5
```

Melakukan pengetesan pada file self-extract lainnya pada direktori contr2 menggunakkan signature pada direktori contr1:

```
gpg -v --verify self-extract.asc
../contr2/self-extract
```

Output:

```
gpg: armor header: Version: GnuPG
v1.2.4 (GNU/Linux)
gpg: Signature made Tue Nov 23
21:55:04 2004 CET using
RSA key ID 3E200834
gpg: Good signature from
"Collision test key (Used to
demonstrate signature collision
when using MD5
algorithm) <collision@md5.abc>"
gpg: binary signature, digest
algorithm MD5
```

5. Serangan Pada Dunia Nyata

Sekarang ini jelas tidak dapat dipungkiri bahwa membuat dua file self-extract executable yang memiliki nilai komputasi MD5 yang sama adalah mungkin dilakukan. Contohya adalah Software Web Browser.

Misalkan, sebuah package dari perusahaan tertentu membuat package self-extract yang bertujuan untuk distribusi. Setelah pengembangan dari web-browsers tersebut selesai, semua file akan diberikan kepada program packager untuk membuat installations script dan membuat package yang sudah siap untuk didistribusikan. Packager membuat scripts dan package itu sendiri. Package akan dikirimkan untuk dilakukan testing pada departemen apakah melewati test tertentu dan dapat dilakukan instalasi. Apabila semua test dilewati, package tersebut dinyatakan baik dan dilakukan tanda tangan digital. Departemen testing akan melakukan tanda tangan digital pada package tersebut untuk menyatakan bahwa program tersebut sudah dites. Kedua package web browser, hasil MD5 sum dan tanda tangan digital diletakkan pada FTP perusahaan tersebut atau web page agar bisa didownload.

Misalkan saja seorang packager tidak jujur (seorang penyerang). Penyerang tersebut membuat pasangan package dengan nilai MD5 sum yang sama, pertama berisi file yang original, satu lagi berisi package yang sudah dimodifikasi. Package yang benar dikirim untuk testing. Sedangkan package yang sudah dimodifikasi akan dipakai untuk mereplace package yang sudah dites tersebut.

Hasil komputasi MD5, digital signature akan tetap sesuai walaupun package tersebut telah di replace dengan paket hasil modifikasi. Apabila penyerang tersebut pandai, ia akan melakukan modifikasi software original tersebut secara sesedikit mungkin dan dengan hati-hati. Hanya penyerang yang mengetahui kondisi untuk melakukan eksekusi program hasil modifikasi miliknya. Package browser yang sudah dimodifikasi akan di download dan di install. Karena nilai MD5 sum dan signature sesuai, dan software tidak melakukan hal-hal yang mencurigakan, akan memakan waktu yang lama untuk mengetahui terjadinya pengubahan tersebut.

Sekarang, penyerang contohnya dapat melakukan hal seperti, menjual kebocoran tersebut pada spammers, penulis virus, membuat web-pages yang akan menginfeksi computer menggunakkan browser modifikasi tersebut atau melakukan hacking terhadap web server yang vulnerable dan hal lainnya.

6. Konklusi

Pada saat ini, Chinese attack tidak dipikirkan secara serius, karena tidak dapat membangkitkan preimage collisions kedua. Tetapi telah ditunjukkan diatas bahwa MD5 collision yang dipublikasikan sangat berbahaya.

Dan telah terbukti bahwa dapat dibuat dua buah dokumen yang memiliki digital signature yang identik. Sehingga dapat disimpulkan bahwa perlu ada pengganti fungsi hash dari MD5 karena sudah tidak aman lagi pada saat ini.

7. Referensi

1. Rivest, R.: "The MD5 Message-Digest Algorithm", RFC 1321, April 1992, ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt

2. X. Wang, D. Feng, X. Lai, H. Yu, "Collisions for Hash Functions MD4, MD5, HAVAL- 128 and RIPEMD", rump session, CRYPTO 2004, *Cryptology ePrint Archive*, Report 2004/199, http://eprint.iacr.org/2004/199

3. Homepage of this project: http://cryptography.hyperlink.cz/2004/collisions.htm

4. Klima, V.: "Hash functions, MD5 and Chinese attack", Seminar "Security of Information Systems", Faculty of Mathematics and Physics, Charles University in Prague, November 22, 2004, http://cryptography.hyperlink.cz/2004/Hasovaci_funkce_a_cinsky_utok_MFFUK_2004.pdf

5. The GNU Privacy Guard - GnuPG, http://www.gnupg.org

6. Dobbertin, H.: "Cryptanalysis of MD4", Journal of Cryptology, 1998, p. 253-271, http://www.ruhr-uni-bochum.de/itsc/download/KryptIISS04/Cryptanalysis_MD4.pdf