ISSN 2355-3286

Implementasi Fuzzy Hashing untuk Signature Malware

Aditia Rinaldi
Program Studi Teknik Informatika, Universitas Multimedia Nusantara, Tangerang, Indonesia
aditia.rinaldi@outlook.com

Diterima 11 April 2014
Disetujui 20 Juni 2014

***Abstract*—Cryptographic hash value has long been used as a database of signatures to identify malware. The most widely used is the MD5 and/or SHA256. In addition, there are fuzzy hashing that slightly different from the traditional hash: length hash value is not fixed and hash value can be used to calculate the degree of similarity of some malware that may still be a variant. This research use ssdeep tool to calculate fuzzy hash. Database signature with fuzzy hash is smaller than SHA256 and larger than MD5. The level of accuracy for the detection of script-based malware variants is greater than the executable-based malware variants.**

***Index Terms*—file signature, fuzzy hashing, malware signature, rolling hashing, sha**

I. PENDAHULUAN

Malicious Software atau biasa disingkat *malware* merupakan sekumpulan instruksi atau program yang berjalan pada suatu sistem komputer yang membuat sistem tersebut melakukan sesuatu yang diinginkan penyerang [1]. *Malware* memiliki banyak jenis, diantaranya *virus*, *worm*, dan *trojan*. Umumnya, *malware* memiliki ukuran yang relatif kecil, tetapi dengan akibat yang besar, seperti pencurian dan penghapusan data. Pertumbuhan *malware* berlangsung sangat cepat. Dalam laporan "The State of Malware 2013" dari McAfee, menyatakan ada 100000 sampel *malware* baru per harinya atau 69 *malware* setiap menitnya [2].

Di sisi antivirus, pertumbuhan *malware* yang sangat cepat ini berdampak pada semakin besarnya ukuran *database signature*. *Database signature* sendiri merupakan basis data yang digunakan *antivirus* yang berisi penanda *malware* yang telah diketahui; *database* ini digunakan untuk memeriksa *file* lain apakah terinfeksi *malware* dengan membandingkan *signature file*nya apakah sama dengan *signature* yang ada di *database*. *Signature* suatu *file* umumnya berupa nilai *hash* yang merupakan nilai unik yang merepresentasikan data dalam *file* tersebut. Nilai *hash* dihasilkan oleh suatu *Hash Function*, seperti MD5, SHA1, dan SHA256. Ketiga *hash function* tersebut memiliki algoritma yang berbeda-beda sehingga menghasilkan nilai *hash* yang berbeda untuk *file* yang sama.

Setiap 1 *malware* memiliki 1 *signature* yang unik. Contohnya untuk mengenali 2 juta *malware* dibutuhkan 2 juta *signature* dan jika ada 100000 sampel *malware* baru per harinya [2], maka dalam waktu 10 hari jumlah *signature* akan mencapai 3 juta. Diasumsikan 3 juta *signature* tersebut menggunakan MD5 (nilai *hash*nya 32 Byte), maka ukuran yang diperlukan sekitar 32 x 3000000 = 9600000 Bytes = 92 MBytes [3].

Dalam laporan lain tentang pertumbuhan *malware* yang dipublikasi oleh Kaspersky Lab 2013 menyatakan ada 315000 *malware* baru setiap harinya [4]. Jika menggunakan laporan Kaspersky Lab [4] untuk perkiraan jumlah *signature malware* yang dibutuhkan, maka: untuk mengenali 2 juta *malware* dibutuhkan 2 juta *signature* dan ada 315000 sampel *malware* baru per harinya [4], dalam waktu 10 hari jumlah *signature* akan mencapai 5,15 juta. Diasumsikan 5,15 juta *signature* tersebut menggunakan MD5 (nilai *hash*nya 32 Byte), maka ukuran yang diperlukan sekitar 32 x 5150000 = 164800000 Bytes ≈ 157, 2 MBytes.

Penghitungan sederhana sebelumnya menggunakan asumsi *file signature* tidak dikompresi dan mengabaikan varian atau turunan yang umumnya dimiliki *malware*. Sebagai contoh, *worm Conficker* memiliki 5 varian (A-E) [5]. Jika varian *malware* diperhitungkan, varian tersebut tentu memiliki nilai *hash* yang berbeda walaupun perubahan dari induknya hanya sedikit, misalnya mengganti beberapa *string* dalam *source code*nya Hal ini membuat ukuran *database signature* menjadi lebih besar lagi.

Dalam penelitian ini, *fuzzy hashing* sebagai salah satu metode *hash* diimplementasikan untuk membuat *signature malware* sederhana dan dilakukan pengujian efisiensi ukuran dan ketepatan deteksi.

II. DASAR TEORI

A. File Signature

File Signature merupakan data yang merepresentasikan suatu *file* yang digunakan untuk mengidentifikasi dan/atau memverifikasi suatu isi *file*. Ada dua tipe *file signature*: *file magic number* dan *file*

checksum.

File magic number merupakan penanda berupa deretan *byte* yang berada di awal *file*. *Magic number* ini digunakan untuk mengidentifikasi *format file* [6].

Sebagai contoh, *file executable* Windows memiliki *magic number* "4D 5A" [7]. Sedangkan *file checksum* merupakan penanda yang berasal dari hasil suatu *hash function* terhadap isi suatu *file* yang digunakan untuk memverifikasi integritas *file* terhadap serangan dan/ atau kesalahan transmisi data [6]. Nilai *checksum* ini biasanya disimpan di akhir *file* atau di *file* terpisah.

B. Malware Signature

Signature malware menggunakan salah satu atau kombinasi dari *file magic number* dan *file checksum* ditambah dengan *string signature. String signature* merupakan deretan *bytes* tertentu (bukan *file magic number*) yang terdapat dalam tubuh *malware* [3]. Beberapa *malware* memiliki *string signature* yang biasanya berupa informasi *file Dynamic Link Library* yang digunakan, alamat *registry* yang coba diakses, dan nama pembuat *malware*. Walaupun *malware*nya tidak sama, selama deretan *bytes* tersebut tidak berubah atau berbeda, maka *malware* tetap akan terdeteksi. Hal ini membuat risiko untuk menghasilkan *false positive* lebih besar karena *file* yang bersih bisa saja kebetulan memiliki deretan *string* yang sama dengan *malware* [3].

C. Traditional Hashing

Dalam *traditional hashing* (MD5 dan SHA256 termasuk di dalamnya) [8], secara garis besar, nilai *hash* didapat dari langkah-langkah berikut:

1. mulai dari *initial state*;
2. pecah input ke dalam *block* yang berukuran tetap dan lakukan proses:
   2a. lakukan penghitungan matematis terhadap *current state* dengan *current block*;
   2b. dapatkan *new state* dari hasil 2a;
3. pindah ke *block* selanjutnya;
4. ulangi langkah 2 dan 3 hingga semua *block* diproses;
5. *state* akhir, keluarkan hasil.

Nilai *hash* sama untuk setiap *file* yang identik. Perubahan susunan *byte* pada isi *file* walau sedikit saja akan menghasilkan nilai *hash* yang berbeda [9], sehingga *file* hasil modifikasi walau sedikit tidak akan terdeteksi dengan *hash* yang lama. *Malware* juga dimodifikasi (menghasilkan banyak varian) agar tidak terdeteksi *antivirus* karena *signature*nya telah berbeda. Bisa dibayangkan betapa lemahnya *antivirus* jika pendeteksian *malware* hanya mengandalkan pembandingan *signature* yang telah diketahui: jika *signature*nya sama dengan di *database*, maka terdeteksi, jika tidak, akan dibiarkan. Bukan berarti semua pendeteksian *malware* yang menggunakan nilai *hashnya* itu jelek, tetapi perlu dipilih algoritma *hash function* yang lebih baik, salah satunya *Fuzzy Hashing*.

D. Fuzzy Hashing

Fuzzy Hashing (disebut juga *Context Triggered Piecewise Hash*) merupakan *hash function* yang menggabungkan *Rolling Hash* dengan *hash* tradisional. *Signature* dihasilkan dari kumpulan *Least Significant Bit* (LSB) yang berasal dari hasil *hash* tradisional per bagian *file* yang pengambilan *hash*nya dipicu oleh *Rolling Hash* [9]. Secara garis besar, nilai *hash* dari *fuzzy hashing* didapat dari langkah-langkah berikut:

1. tetapkan sebuah *trigger* atau pemicu sebagai penanda *rolling* isi *file* ke dalam *block* (ukuran *block* bisa bersifat tetap dan berubah-ubah);
2. baca *file*;
3. lakukan *rolling* dan cari pemicu:
   3a. jika pemicu ditemukan, hitung nilai *hash* dari *block* ini dengan *hash* tradisional;
   3b. ambil LSB dari hasil *hash* tradisional;
4. ulangi langkah 2 hingga *file* berakhir;
5. ketika selesai, gabungkan LSB-LSB untuk membuat *signature*. [9]

Rolling merupakan proses penelusuran isi *file* untuk membagi isi *file* ke dalam *block-block* dengan ukuran *block* tergantung dari *trigger* (pemicu) yang ditetapkan; jika sebuah *trigger* ditemukan, maka isi *file* sampai *trigger* merupakan satu *block* dan dilakukan penghitungan nilai *hash* dengan *hashing* tradisional dan setelah melewati *trigger* merupakan *block* selanjutnya. Berikut simulasi mendapatkan nilai *hash fuzzy* dari suatu teks dengan *trigger string* "ou" dan "re" (teks diambil dari kutipan novel "The Fault In Our Stars" – John Green):

*"I'm in love with y**ou**, and I'm not in the business of denying myself the simple pleasu**re** of saying true things. I'm in love with y**ou**, and I know that love is just a sh**ou**t into the void, and that oblivion is inevitable, and that we'**re** all doomed and that th**ere** will come a day when all **ou**r labor has been **re**turned to dust, and I know the sun will swallow the only earth we'll ever have, and I am in love with y**ou**"*

Dipecah menjadi 9 *block* dan masing-masing dihitung *hash*nya, contoh di sini menggunakan MD5 :

1. *I'm in love with y**ou***
2. *, and I'm not in the business of denying myself the simple pleasu**re***
3. *of saying true things. I'm in love with y**ou**,*
4. *and I know that love is just a sh**ou***
5. *t into the void, and that oblivion is inevitable, and that we'**re***
6. *all doomed and that th**ere***
7. *will come a day when all **ou***
8. *r labor has been **re***
9. *turned to dust, and I know the sun will swallow the only earth we'll ever have, and I am in love with y**ou***

Tabel 1. Tabel kumpulan *hash* per *block* dengan MD5

No. block	Nilai Hash
1	cae79be2a441e4f9c333075807460a9**5**
2	8afbe93cc25582868957d7f7388c244**5**
3	ef25df7fb4679fd73306cfa29e1d161**2**
4	0fd61c1f75fb81883d48d05d8448847**c**
5	bdc56038f27b202acf57c207d7ecf8c**d**
6	0ba1f78a08c5cc98815cce6ab095d88**c**
7	547e9a90b1391daeb5c8512958baafe**3**
8	949b28d4911bf9c1893c754c120976b**3**
9	6f77406a3e7767294191dceeecd6de8**0**

Sehingga didapat nilai *fuzzy hash* dari kumpulan LSB-LSB dari masing-maing nilai *hash* per bagian = **552cdc330**. Jika dilakukan sedikit modifikasi bagian ke-3 dan ke-4 (selama tidak mengandung *trigger* yang ditentukan) menjadi :

3. *of saying false things. I hate with y**ou**,*
4. *and I know that hate is just a sh**ou***

Nilai MD5 untuk bagian 3 dan 4 berubah menjadi :

3. 2bf3a00618d0e844be398ec8a8b9134**2**
4. caf444a15906b9e551610f1993521a6**1**

Sehingga didapat perubahan nilai *fuzzy hash* menjadi **5521dc330** dari nilai *fuzzy hash* sebelumnya **552cdc330**. Dari hasil ini dapat diamati, modifikasi kecil teks di tengah-tengah kalimat membuat hasil dari *fuzzy hashing* berbeda, tetapi tetap terlihat kemiripannya. Bandingkan dengan nilai *hash* MD5 untuk keseluruhan teks sebelum dimodifikasi (**4587e3fd3cc0b3285cf4c3e47d6fb801**) dan setelah dimodifikasi (**101cf5b466ac6941c59d67c6103a2653**).

III. DEMO FUZZY HASHING DENGAN SSDEEP

Ssdeep merupakan salah satu *freeware tools* yang diilhami dari *spam detector* yang diciptakan Andrew Tridgell [3]. Ssdeep digunakan untuk menghitung *fuzzy hash* (disebut juga *Context Triggered Piecewise Hash*) suatu *file*, menyimpan nilai *hash* hasil penghitungan dan melakukan pencocokan kemiripan antara suatu nilai *hash* dengan *file* yang diuji [10]. Secara umum, ssdeep dapat mencocokkan input berupa *file* yang memiliki *homologies*, yaitu *file* yang memiliki serangkaian *bytes* yang identik dalam urutan yang sama walau *byte* lain yang berada di antara rangkaian *bytes* yang sama memiliki konten dan ukuran yang berbeda-beda [11].

Format nilai *hash* yang dihasilkan *tool* ssdeep sebagai berikut :

blocksize:hash:hash,filename

A. *Pengujian pada Plaintext file(txt)*

Dilakukan penghitungan *hash* menggunakan ssdeep terhadap sebuah *file plaintext* dengan nama playlist.txt berukuran 14KB. *Hash* disimpan dalam sebuah *file* signplaylist.txt. Hasilnya sebagai berikut :

192:5xgEpn4GyzphzLmPgh7CwFCQW0IfhS2/woVQ/xB1+ylQNkl:5xgEpedhzLmPgh7CIY0IfhSb/xvlJ,"playlist.txt"

File playlist.txt di*rename* menjadi playlist2.txt tanpa mengubah isinya, nilai *hash*nya pun masih sama seperti sebelumnya. Begitu juga bila *file* dimodifikasi, lalu disimpan, dan dimodifikasi kembali seperti semula, nilai *hash*nya pun masih sama. Hal ini berarti perubahan nama *file* dan *metadata* sebuah *file* tidak memengaruhi *hash* dari *file*.

File playlist.txt diduplikat menjadi playlist-o.txt dan dilakukan perubahan 1 karakter. Nilai *hash*nya sebagai berikut:

192:5xgEpn4GyzphzLmPgh7CwFCQ40IfhS2/woVQ/xB1+ylQNkl:5xgEpedhzLmPgh7CI20IfhSb/xvlJ,"playlist-o.txt"

File playlist.txt diduplikat menjadi playlist-r.txt dan dilakukan penghapusan 2 karekter. Nilai *hash*nya sebagai berikut:

192:5xgE pn4GyzphzLmPgh7CwFCQW0IfhS2kwJVQ/xB1+ylQNkl:5xgEpedhzLmPgh7CIY0IfhSd/xvlJ,"playlist-r.txt"

File playlist.txt diduplikat menjadi playlist-a.txt dan dilakukan penambahan 3 *string* "hello": di awal *file*, tengah *file*, dan akhir *file*. Nilai *hash*nya sebagai berikut:

192:/xgEpn4GyzphzLmPgh7CwFaQW0IfhS2/woVQ/xB1+ylQNkH:/xgEpedhzLmPgh7CIw0IfhSb/xvlj,"playlist-a.txt"

Setelah itu dilakukan penghitungan berapakah derajat kemiripan setiap *file* terhadap *file* lainnya menggunakan nilai *hash* masing-masing. Berdasarkan *screenshot* di halaman berikutnya, setelah diuji menggunakan ssdeep, semua *file* mempunyai tingkat kemiripan lebih dari 90% terhadap *file* lain.

Hal ini, berarti modifikasi kecil berupa menghapusan beberapa karakter, pengubahan beberapa karakter dan penambahan sejumlah kecil *string* tidak terlalu memengaruhi secara besar hasil dari *fuzzy hashing*. Sehingga, ssdeep bisa mengenali kemiripan *file*nya berdasarkan nilai *hash* yang dihasilkan.

```
F:\Projects\C\Ssdeep\bin>ssdeep.exe -bm signplaylist.txt -r test
playlist-a.txt matches signplaylist.txt:playlist.txt (94)
playlist-o.txt matches signplaylist.txt:playlist.txt (99)
playlist-r.txt matches signplaylist.txt:playlist.txt (97)
playlist.txt matches signplaylist.txt:playlist.txt (100)

F:\Projects\C\Ssdeep\bin>ssdeep.exe -bm signplaylist-o.txt -r test
playlist-a.txt matches signplaylist-o.txt:playlist-o.txt (93)
playlist-o.txt matches signplaylist-o.txt:playlist-o.txt (100)
playlist-r.txt matches signplaylist-o.txt:playlist-o.txt (94)
playlist.txt matches signplaylist-o.txt:playlist-o.txt (99)

F:\Projects\C\Ssdeep\bin>ssdeep.exe -bm signplaylist-r.txt -r test
playlist-a.txt matches signplaylist-r.txt:playlist-r.txt (91)
playlist-o.txt matches signplaylist-r.txt:playlist-r.txt (94)
playlist-r.txt matches signplaylist-r.txt:playlist-r.txt (100)
playlist.txt matches signplaylist-r.txt:playlist-r.txt (97)

F:\Projects\C\Ssdeep\bin>ssdeep.exe -bm signplaylist-a.txt -r test
playlist-a.txt matches signplaylist-a.txt:playlist-a.txt (100)
playlist-o.txt matches signplaylist-a.txt:playlist-a.txt (93)
playlist-r.txt matches signplaylist-a.txt:playlist-a.txt (91)
playlist.txt matches signplaylist-a.txt:playlist-a.txt (94)
```

Gambar 1. Hasil pengujian tingkat kemiripan beberapa *plaintext* menggunakan ssdeep.

B. Pengujian pada file Binary berupa Executable yang dicompile dengan GCC di sistem operasi Windows

Isi *file* prog.c sebagai berikut :

```
#include <stdio.h>
#include <conio.h>

int main()
{
printf("hello world\n");
return 0;
}
```

Source code C tersebut kemudian di*compile* menggunakan GCC dan menghasilkan *file* prog.exe berukuran 16KB. *Hash* disimpan dalam sebuah *file* signprog.txt. Hasilnya sebagai berikut :

192:ODHKC75Zr3yqMPTRqz9SAPrOkjyFoqcCDq:M75ZrCqMdqhffeaqrW,"prog.exe"

File prog.c dimodifikasi dengan menambahkan "#include <stdlib.h>". Kemudian di*compile* ulang menjadi prog-a.exe. Nilai *hash*nya sebagai berikut:

192:WDHKC75Zr3yqMPTRqz9SAPrOkjyFoqcCDq:U75ZrCqMdqhffeaqrW,"prog-a.exe"

File prog.c dimodifikasi lagi dengan menambahkan beberapa deklarasi *variable integer*, melakukan kalkulasi sederhana, dan menampilkan hasilnya bersama dengan "hello world" sebelumnya. Kemudian di*compile* ulang menjadi prog-b.exe. Nilai *hash*nya sebagai berikut:

192:MTHa7kMMZr3yqMM444xKzJWAhOkjyFoqcCDq:LkFZrCqM6iKdVeaqrW,"prog-b.exe"

File prog.c dimodifikasi lagi dengan mengembalikannya seperti semula (prog.exe) hanya ditambahkan beberapa komentar. Kemudian di*compile* ulang menjadi prog-c.exe. Nilai *hash*nya sebagai berikut:

192:cDHKC75Zr3yqMPTRqz9SAPrOkjyFoqcCDq:i75ZrCqMdqhffeaqrW,"prog-c.exe"

File prog.exe, diduplikat menjadi prog-d.exe dan dikompres menggunakan UPX. Ukuran *file* berkurang dari 16KB menjadi 14 KB. Nilai *hash*nya sebagai berikut:

192:bu4u8eZMciT6oTDQ+PTRqz9SAPrOkjyFoqcCDq:/ugciT6oTM+dqhffeaqrW,"prog-d.exe"

Setelah itu dilakukan penghitungan berapakah derajat kemiripan setiap *file* terhadap *file* lainnya menggunakan nilai *hash* masing-masing. Setelah diuji menggunakan ssdeep, semua *file* mempunyai tingkat kemiripan lebih dari 60% terhadap *file* lain.

```
C:\Users\HIMASIKOM\Downloads\upx\bin>ssdeep.exe -bm signprog.txt -r test
prog-a.exe matches signprog.txt:prog.exe (99)
prog-b.exe matches signprog.txt:prog.exe (61)
prog-c.exe matches signprog.txt:prog.exe (99)
prog-d.exe matches signprog.txt:prog.exe (65)
prog.exe matches signprog.txt:prog.exe (100)

C:\Users\HIMASIKOM\Downloads\upx\bin>ssdeep.exe -bm signprog-a.txt -r test
prog-a.exe matches signprog-a.txt:prog-a.exe (100)
prog-b.exe matches signprog-a.txt:prog-a.exe (61)
prog-c.exe matches signprog-a.txt:prog-a.exe (99)
prog-d.exe matches signprog-a.txt:prog-a.exe (65)
prog.exe matches signprog-a.txt:prog-a.exe (99)

C:\Users\HIMASIKOM\Downloads\upx\bin>ssdeep.exe -bm signprog-b.txt -r test
prog-a.exe matches signprog-b.txt:prog-b.exe (61)
prog-b.exe matches signprog-b.txt:prog-b.exe (100)
prog-c.exe matches signprog-b.txt:prog-b.exe (61)
prog-d.exe matches signprog-b.txt:prog-b.exe (44)
prog.exe matches signprog-b.txt:prog-b.exe (61)

C:\Users\HIMASIKOM\Downloads\upx\bin>ssdeep.exe -bm signprog-c.txt -r test
prog-a.exe matches signprog-c.txt:prog-c.exe (99)
prog-b.exe matches signprog-c.txt:prog-c.exe (61)
prog-c.exe matches signprog-c.txt:prog-c.exe (100)
prog-d.exe matches signprog-c.txt:prog-c.exe (65)
prog.exe matches signprog-c.txt:prog-c.exe (99)

C:\Users\HIMASIKOM\Downloads\upx\bin>ssdeep.exe -bm signprog-d.txt -r test
prog-a.exe matches signprog-d.txt:prog-d.exe (65)
prog-b.exe matches signprog-d.txt:prog-d.exe (44)
prog-c.exe matches signprog-d.txt:prog-d.exe (65)
prog-d.exe matches signprog-d.txt:prog-d.exe (100)
prog.exe matches signprog-d.txt:prog-d.exe (65)
```

Gambar 2. Hasil pengujian tingkat kemiripan beberapa *Windows executable file* menggunakan ssdeep.

Hal ini, berarti modifikasi kecil berupa penambahan *library* dan pemberian komentar tidak terlalu memengaruhi secara besar hasil dari *fuzzy hashing*. Modifikasi yang menyebabkan perubahan alur program (prog-b.exe) menyebabkan tingkat kemiripan yang dideteksi menjadi lebih rendah (61%). Dalam kasus ini, *file executable* yang telah dikompres menggunakan UPX memiliki tingkat kemiripan dengan *file* asli yang lebih tinggi dari *file* yang telah dimodifikasi alur programnya.

Sangat penting untuk menetapkan batas bawah toleransi tingkat kemiripan yang bisa diterima. Semakin besar toleransinya (semakin lebar rentang tingkat kemiripan yang diterima, misal 30%-100%), maka kemungkinan *false positive* atau salah deteksi terhadap suatu *file* semakin besar.

Semakin kecil toleransinya (semakin kecil rentang tingkat kemiripan yang diterima, misal 95% - 100%), maka ketepatan deteksi semakin besar tetapi dengan kemungkinan semakin banyak *file* yang tidak dideteksi atau diabaikan.

IV. PENGUJIAN UKURAN DAN TINGKAT DETEKSI DENGAN SIGNATURE FUZZY hASHING

Beberapa *malware* berikut yang diuji dalam pembahasan ini merupakan *malware* simulasi buatan penulis yang digunakan untuk tujuan penelitian ini. Terdapat 6 *malware* (3 *malware* berjenis *VBScript*, yaitu virus Rat0, ILOVEYOU, TAGALIPAARE; dan sisanya berjenis *Windows executable* dibuat menggunakan bahasa C dan C++) yang diuji untuk dihitung nilai *fuzzy hash*nya, kemudian kumpulan *hash* tersebut disimpan dalam *file* signatures-fuzzy.txt.

Sebagai pembanding ukuran, dibuat juga *signatures* dengan *hash function* lain berupa MD5 dan SHA256.

A. Signature Malware asli tanpa varian

Signature MD5 disimpan dalam *file* signatures-md5.txt dengan format nama_file_asli#hash. *Signature* SHA256 disimpan dalam *file* signatures-sha.txt dengan format yang sama dengan MD5. *Signature fuzzy* disimpan dalam *file* signatures-fuzzy.txt dengan format *default* dari ssdeep (sudah termasuk nama *file* asli). blocksize:hash:hash,filename. Berikut *screenshot* untuk masing-masing signature:

```
c-1.exe#230ce3bd689199d8df9b23cc6efa14f8
c-2.exe#d8deb204ad8e494703a6282add0c3b69
cpp-1.exe#eac461d815c3bbe9776e21ccb16330cf
vbs-1.vbs#09770f11a662b6b10426d97ae4538833
vbs-2.vbs#7b6630db25a7a329d93d2560568c5045
vbs-3.vbs#900e66d886d4d98136a1877c18ea72d2
```

Gambar 3. *Signature MD5*.

```
c-1.exe#ecdab5e485ad4453da3a6a7621499f1296b1bc819bcd6a23a4bb6153e10f864c
c-2.exe#7183c2104861179 73a9093a8d3e0e7be605190fc9714bc663217992d7628dcfc
cpp-1.exe#2bfc3c1ff756a2ce41ccfbaf50ea19c7d9f996cc0673470a7a03d50d09bdf67a
vbs-1.vbs#71e6467876d39a1e74003e4b5146af7120fe4035bac295c21d8b3af33dbeae1e
vbs-2.vbs#d2315291cc2ce66674a7cd3c0424aa15d24bbbcac233f42177e848d6863ef6ad
vbs-3.vbs#dae5888ce349c1e1d80452113572 97b728b7cc3200a67732b3afffbd5fe8000a
```

Gambar 4. *Signature SHA256*.

```
192:lHH7c5w/acO5aeAK0cTDzOayfKYbLJzh3xwpfmNbPh2JM3RgL3Z3YjcFyF1USLCR:yhrFp9hwpfQ2Y2Lp3YYUPUSe40,"c-1.exe"
384:ET8pjV+hRh7psVEl70txAAxFV36mNJs5NV8qnbehhO5RJR/c7yQB:ET8pV+hv7eVLH3V36mNJs5NqqPe+,"c-2.exe"
1228B:brXLBbCBGLJYr9IjgigMCZX/H+E23JIueUYFT+u2kSfcEiP/3IWVB/oxAcwzD:bjLBbCBGLJbq/H+53JKhlE/8,"cpp-1.exe"
96:VWko53CyCtD2EK1XGMHao+JpGyhAX1vHMTMcvAB5TGDJ40ED19b:VWko8CyCtD2EK1XGMHao+J0yhAwMTHi3,"vbs-1.vbs"
192:p4o6L5E4tZfPFKb/7MGMQMLMIMmMuM5MmMwM4M79MHjXJ02AHGQNfqc8s0jSe8cv:p4/NRvfPFKb/7PlmzxX4xZTe9oj5lnQM,"vbs-2.vbs"
48:1iJ6eV2UqhILvpUxObib1U3D3svqLlmMQJvmLuNqWk+uvvy:lMkOer1sv9q+z,"vbs-3.vbs"
```

Gambar 5. *Signature fuzzy hash* dengan ssdeep.

Dapat diketahui bahwa panjang *hash* yang dihasilkan *fuzzy hashing* berbeda-beda tergantung *file*nya, tidak seperti MD5 dan SHA256 yang mempunyai panjang *hash* yang tetap.

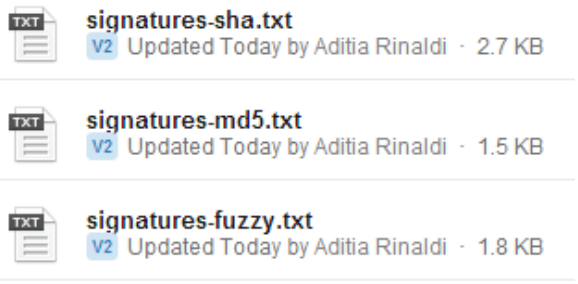
Berikut hasil perbandingan ukuran ketiga *signature*:

Gambar 6. Perbandingan ukuran ketiga*signature*.

Dari *screenshot* di atas terlihat bahwa MD5 memiliki ukuran yang paling kecil, diikuti *fuzzy* dan yang paling besar SHA256.

B. Signature Malware asli dengan varian

File signature dari masing-masing *hash* sebelumnya ditambah dengan 2 varian dari masing-masing *malware*. Varian merupakan hasil modifikasi yang berupa penambahan *string* dan penghapusan *string* secara acak dari *malware* asli. Berikut hasil perbandingan ukuran ketiga *signature*:



Gambar 7. Perbandingan ukuran ketiga *signature* setelah masing-masing *malware* dibuat 2 buah variannya (total 18 *malware*) .

Dari *screenshot* di atas terlihat bahwa MD5 tetap memiliki ukuran yang paling kecil, diikuti *fuzzy* dan yang paling besar SHA256.

C. Pengujian tingkat deteksi malware dengan fuzzy hashing

Dalam pengujian ini, *database signature* yang digunakan adalah *signature-fuzzy.txt* versi Gambar 5 yang belum ditambahkan *signature* varian (dengan ukuran 606 Bytes). Hal ini untuk menguji sejauh mana *malware* dan variannya dapat dengan tepat dideteksi walau hanya menggunakan *signature* malware aslinya yang belum dimodifikasi. Atau dengan kata lain menguji juga *trade-off* antara menjaga ukuran *database signature* sekecil mungkin dengan tingkat deteksi yang masih cukup memadai. Berikut hasil penghitungan tingkat deteksi menggunakan ssdeep.

Tabel 2. Tabel tingkat deteksi malware dengan hanya menggunakan *signature malware* asli.

Nama File	Jenis Virus	Tingkat Deteksi
c-1.exe	asli	100%
c-1a.exe	varian	99%
c-1b.exe	varian	50%
c-2.exe	asli	100%
c-2a.exe	varian	99%
c-2b.exe	varian	54%
cpp-1.exe	asli	100%
cpp-1a.exe	varian	41%
cpp-1b.exe	varian	43%
vbs-1.vbs	asli	100%
vbs-1a.vbs	varian	74%
vbs-1b.vbs	varian	74%
vbs-2.vbs	asli	100%
vbs-2a.vbs	varian	97%
vbs-2b.vbs	varian	94%
vbs-3.vbs	asli	100%
vbs-3a.vbs	varian	97%
vbs-3b.vbs	varian	94%
Rata -rata		**84,22%**

Pada *malware* yang berjenis *plaintext* seperti *VBScript* yang digunakan dalam penelitian ini, tingkat deteksi variannya lebih baik daripada *executable malware*. Tingkat deteksi terendah yang dihasilkan yaitu 41% pada varian cpp-1a.exe terhadap *malware* aslinya (cpp-1.exe). Rata-rata yang dihasilkan cukup

baik (84,22%).

Setelah itu, dilakukan pengujian sekali lagi dengan melibatkan berbagai macam jenis *file* yang sehat (tidak terinfeksi) untuk melihat apakah terjadi salah deteksi (*false positive*). Dari hasil yang didapat, tidak ditemukan *false positive* (dengan menunjukkan tingkat deteksi 0%).

V. SIMPULAN

Fuzzy Hashing dapat dipilih untuk menghitung *signature malware* yang cukup baik. Karena nilai *hash* yang dihasilkan dari beberapa *file* yang bersumber dari *file* yang sama tetapi dengan modifikasi kecil yang berbeda-beda tetap dapat terlihat kemiripannya dibandingkan dengan nilai hash MD5 atau SHA256 yang menghasilkan nilai *hash* yang tidak mirip walau ternyata *file-file* yang dibandingkan dan dihitung *hash* nya hanya berbeda beberapa *byte*.

Sehingga, dari kemiripan nilai *hash* dapat dilakukan pembandingan antara *file* yang dideteksi dengan *signature* yang ada di *database* yang menghasilkan derajat kemiripan (0%-100%). Hal ini memberi keuntungan lain, yaitu *signature fuzzy hash* dari satu *file*, dapat digunakan untuk mendeteksi beberapa hasil modifikasi kecil atau variannya. Secara teoritis, hal ini dapat mengurangi ukuran *database signature* karena jika modifikasi kecil atau varian dari *file* masih dikenali kemiripannya dengan *signature* yang sudah dikenal, kita tidak perlu membuat *entry hash* baru untuk seluruh *file* modifikasinya.

Namun, bila dilihat dari cara kerjanya, *fuzzy hashing* melakukan penghitungan *hash* tradisional per *block* untuk setiap *trigger* yang ditemukan; kemudian, dari nilai LSB-LSB dari *hash* yang dihitung itulah dihasilkan suatu *hash fuzzy*. Hal ini menyebabkan waktu penghitungan *hash fuzzy* menjadi sedikit lebih lama dibandingkan *hash* tradisional lain. Selain itu, bila kita memilih *trigger* yang kurang tepat, maka waktu penghitungan menjadi lebih lama lagi.

Oleh karena itu, diperlukan penelitian lebih lanjut mengenai kecepatan *fuzzy hashing* dengan nilai *hash* yang tetap menjaga similiaritasnya dan memilih *trigger* yang tepat untuk melakukan penghitungan dengan tools (selain ssdeep) yang lebih banyak.

Daftar Pustaka

[1] Ed Skoudis dan Lenny Zeltser, Malware: Fighting Malicious Code, New Jersey: Prentice Hall, 2004.

[2] McAfee. Infographic: The State of Malware 2013 [online], 2013. Tersedia dalam: http://www.mcafee.com/us/security-awareness/articles/state-of-malware-2013.aspx. Diakses 10 Maret 2014.

[3] Joko Nurjadi,"Signature Malware", Majalah PCMedia, vol. 10/2013, hal.78, Oktober 2013.

[4] Kaspersky Lab, Number of the year: Kaspersky Lab is detecting 315,000 new malicious files every day [online], 2013. Tersedia dalam: http://www.kaspersky.com/about/news/virus/2013/number-of-the-year. Diakses 10 Maret 2014.

[5] Dave Piscitello, "Conficker summary and review", ICANN News & Press, May 2010.

[6] T. Sammes dan B. Jenkinson, Forensic Computing: A Practitioner's Guide, Springer, 2000.

[7] Gary Kessler, File Signatures Table [online], 2014. Tersedia dalam: http://www.garykessler.net/library/file_sigs.html. Diakses 10 Maret 2014.

[8] Ken Dunham. "A Fuzzy future in malware research", ISSA Journal, hal.17, Agustus 2013.

[9] Jesse Kornblum, "Fuzzy Hashing" [presentation], Conference on Digital Forensics, Security and the Law, 2007.

[10] Dustin Hurlbut,"Fuzzy hashing for digital forensic investigators", AccessData, 9 Januari 2009.

[11] ----------, "Identifying almost identical files using context triggered piecewise hashing", Digital Investigation, 3S, 2006.