Penerapan Algoritma *Divide and Conquer* untuk Meningkatkan Kecepatan Deteksi dari Mesin *Client Honeypots*

Nugraha 13509096
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
nugraha.siburian@students.itb.ac.id

***Abstrak*—Pada makalah ini akan dibahas bagaimana cara meningkatkan kecepatan untuk mendeteksi dan mengidentifikasi *server* berbahaya dalam suatu jaringan internet pada *client honeyspot* dengan menggunakan algoritma devide and conquer. Penggunaan algortima *Divide and Conquer* akan meningkatkan performansi dibandingkan menggunakan algoritma *brute force*. Peningkatan performansi menyebabkan *client honeypots* memeriksa lebih banyak *server* dan *delay* untuk klarifikasi lebih cepat dengan kemampuan untuk memeriksa dan mengidentifikasi sejumlah *server* menggunakan *resource* yang sama dan memprediksi serangan yang tertunda oleh *server* berbahaya.**

Kata kunci—*Divide and Conquer*, *server* berbahaya, *client honeypots*, performansi.

1 Pendahuluan

Internet telah menjadi kebutuhan yang cukup penting bagi masyarakat dunia saat ini. Internet telah menjadi sumber informasi, hiburan, dan menjadi alat komunikasi utama baik di lingkungan rumah atau di kantor. Namun demikian, konektivitas internet menimbulkan ancaman keamanan. Pengaksesan secara tidak sah, penolakan layanan, atau penguasaan sepenuhnya terhadap komputer oleh *user* berbahaya adalah contoh dari ancaman keamanan yang menyerang internet.

Karena internet adalah jaringan global, serangan dapat dilakukan dari tempat dimanapun di dunia dengan anonimitas yang tinggi. Oleh karena itu pada umumnya, komputer yang terhubung dengan internet setidaknya telah memiliki *antivirus* dan *firewall*. Tindakan ini terbukti telah cukup sukses untuk menangkal ancaman-ancaman keamanan di internet. Setelah media penyerangan dihambat oleh aplikasi pertahanan, *user* berbahaya selanjutnya mencari jalur yang tidak terlindungi untuk menyerang. Serangan ini disebut *client-side attack* yang memiliki target aplikasi *client*.

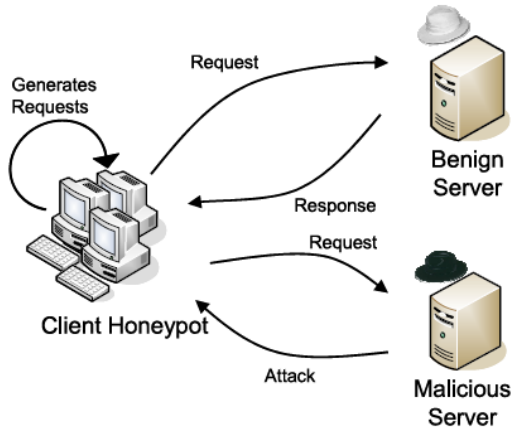
Setelah *client* mengakses *server* berbahaya, *server* selanjutnya mengirimkan serangan kepada pihak *client* sebagai bagian dari *request* oleh *client*. Contoh yang biasa dari serangan ini adalah *server web* yang menyerang *web browser*. Setelah *request* dari *web browser* untuk mendapatkan konten, *server* akan mengirimkan halaman-halaman berbahaya yang akan menyerang *browser*. Jika berhasil, pertahanan tradisional seperti *antivirus* dan *firewall* pun akan terinfeksi. Untuk mengembangkan metoda pertahanan baru terhadap ancaman-ancaman tersebut, dibutuhkan kajian yang lebih mendalam terhadap *server* berbahaya. Kajian utama terhadap *server-server* tersebut adalah untuk mencari *server* mana saja yang berbahaya dalam suatu jaringan, khususnya internet. Untungnya, *server* berbahaya perlu *accessable* agar penyerangannya berhasil. Peluang inilah yang memungkinkan untuk menemukan *server* berbahaya.

Client honeypots adalah teknologi baru yang digunakan untuk proses pencarian tersebut. Setelah *server* berbahaya diidentifikasi, akses terhadapnya dapat dihambat atau aturan dapat digunakan sebagai bantuan untuk menutup element ini dari jaringan. Namun demikian, *client honeypots* dihadapkan oleh tantangan-tantangan dari sisi internalnya. Pertama, harus"meraba-raba" internet yang memilik jutaan *server*. Mencari *server* berbahaya akan sama halnya dengan mencari jarum di tumpukan jerami. Kecepatan akan menjadi aspek yang sangan krusial untuk mengidentifikasi *server* berbahaya secara cepat dan melakukan pertahanan terhadapnya.

Client honeypots yang ada saat ini memiliki kinerja yang lambat. Algoritma-algoritma pendeteksi yang berdasarkan pada pemantauan perubahan kondisi yang sah dari *client honeypots* sangatlah "mahal". Algoritma-algoritma tersebut membutuhkan waktu dan *resource* yang besar. Berdasarkan pada penelitian *client honeypots* yang dikembangkan Microsoft Research, kebanyakan *website* berbahaya harus menginstall terlebih dahulu *file* berbahaya antara 30 detik. Berhadapan dengan kuantitas *server* di internet, durasi 30 detik untuk mengklasifikasi *server* sebagai *server* berbahaya membuat pencarian komprehensif di internet menjadi tidak mungkin.

Lalu, bagaimana untuk meningkatkan performansi *client honeypots*? Salah satu caranya adalah dengan menerapkan algoritma *Divide and Conquer* sebagai jalan *client honeypots* untuk berinteraksi dengan *server* berbahaya dan mengidentifikasi *server* tersebut.

2 *Client Honeypots*



Gambar 2.1 Arsitektur *Client Honeypots*

Client honeypots dapat bekerja menemukan *server* berbahaya pada jaringan. Mereka melakukannya dengan menghasilkan antiran permintaan *server*, menerbitkan permintaan ini ke *server* satu per satu dan mengkonsumsi tanggapan dari *server* seperti yang ditunjukkan pada Gambar 2.1. Setelah tanggapan dikonsumsi, maka *honeypots client* dapat melakukan analisis yang menentukan apakah *server* tersebut berbahaya atau baik.

Klasifikasi ini didasarkan pada pemantauan sistem pada kondisi tidak sah yang terjadi pada sistem setelah *client honeypots* telah berinteraksi dengan *server*. *Client honeypots* merupakan mesin yang berdedikasi karena tidak ada aktifitas lain yang terjadi pada mereka, kondisi perubahan tidak sah seperti proses baru, *file* baru diinstal, dll, dapat dideteksi oleh *client honeypot*.

Honeypot dapat menjalankan bermacam service dan dapat berjalan pada bermacam system operasi, honeypot dibedakan menjadi ;

Low-interaction Mengemulasi system operasi dan service	**High-interaction** Sistem operasi dan service sungguhan tanpa emulasi
• Mudah diinstall dan deploy, konfigurasi biasanya sederhana • Resiko minimal, emulasi mengontrol apa yang bisa dilakukan penyusup • Menangkap jumlah informasi terbatas	• Menangkap informasi lebih banyak • Bisa cukup kompleks • Resiko tinggi, penyusup bisa berinteraksi dengan system operasi sungguhan

Honeypot juga dapat dibedakan menjadi :

- Physical : Mesin sungguhan dalam jaringan dengan alamat ip sendiri
- Virtual : Disimulasikan oleh mesin lain yang berespon pada traffic jaringan yang dikirim ke virtual honeypot.

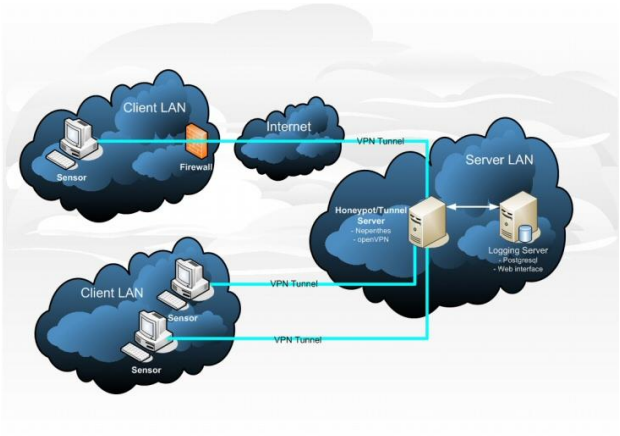
Suatu honeypots merupakan sumber system informasi yang menghasilkan nilai palsu pada saat terjadi penggunaan sumber daya yang tidak sah atau tidak diijinkan

Untuk mengenali sebuah serangan yang dilakukan oleh hacker atau cracker, digunakan data yang diperoleh. Pendekatan yang sering digunakan untuk mengenali serangan antara lain :

- Anomaly detection
  Anomaly detection mengidentifikasi perilaku tak lazim yang terjadi dalam host atau network.
- Misuse detection
  Detector tersebut melakukan analisa terhadap aktivitas system, mencari event yang cocok dengan pola perilaku yang dikenali sebagai serangan.

Setelah perubahan kondisi terdeteksi dan klasifikasi telah dibuat, mesin perlu diatur ulang dalam keadaan bersih sebelum dapat berinteraksi dengan *server* lain. *Client honeypots* yang menggunakan pendekatan ini juga disebut sebagai *client honeypots* dengan interaksi yang tinggi. Mencari *server web* yang berbahaya, misalnya, orang akan mengambil halaman *web* dengan *browser*. Hal ini akan menyebabkan berbagai perubahan kondisi terjadi pada sistem, seperti *file* yang ditulis ke dalam *cache*. Ini adalah peristiwa yang berwenang kita akan mengabaikan untuk membuat klasifikasi pada apakah halaman *web* yang berbahaya atau tidak.

Jika exe baru *file* muncul dalam folder *start-up*, maka diklasifikasikan bahwa halaman *web* sebagai halaman yang berbahaya karena hanya serangan yang berasal dari bahwa halaman *web* dapat menyebabkan penempatan *file* ini dalam folder *start-up*. *Client honeypots* fokus pada *server web* berbahaya yang berinteraksi menggunakan *browser web* pada sistem *honeypot* khusus. *Client honeypot* mendeteksi serangan yang sukses dengan memantau perubahan pada daftar *file*, direktori, dan sistem konfigurasi setelah *client honeypot* telah berinteraksi dengan *server*. *client honeypot* juga mendeteksi intrusi oleh perubahan daftar *file* dan entri registry, tetapi melangkah lebih jauh dengan menambahkan pemantauan proses anak untuk mendeteksi serangan sisi *client*. Selain kemampuan untuk mendeteksi perubahan kondisi tambahan, *client honeypot* juga memungkinkan untuk mendeteksi perubahan kondisi yang terjadi.

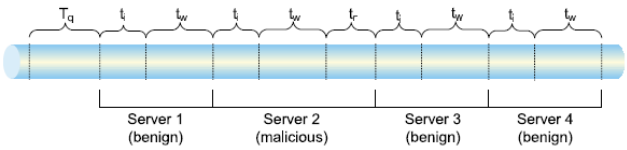


Gambar 2.2 Ilustrasi *Client Honeypot*

Pendeteksian kecepatan dari *server* berbahaya oleh *client honeypots* dipengaruhi oleh berbagai faktor. Pertama, ada teknologi yang mendasari tentang bagaimana perubahan kondisi terdeteksi. *Client honeypot* misalnya, menggunakan *snapshot* yang memerlukan waktu lama untuk menciptakan sedangkan implementasi lainnya menggunakan pemicu yang memungkinkan deteksi dari perubahan kondisi yang terjadi. Ketidakbergantungan dari impelementasi, terdapat faktor-faktor tambahan yang mempengaruhi waktu total tt untuk memeriksa satu set *server* n. *Bandwidth* jaringan b dan rata-rata ukuran permintaan / respon, yang mempengaruhi waktu ti untuk mengambil respon *server*, *overhead* dari *client honeypot* menjadi kondisi bersih setelah *server* berbahaya telah telah ditemukan tr, yang secara keseluruhan dipengaruhi oleh persentase *server* berbahaya yang ada pada jaringan pm, dan klasifikasi terakhir waktu penundaan tw.

Klasifikasi penundaan sengaja diperkenalkan menunggu periode setelah respon dari sebuah *server* telah diterima sebelum klasifikasi dibuat. Klasifikasi diperkenalkan karena beberapa waktu berlalu sebelum banyak eksploitasi memicu. Hal ini mungkin disebabkan karena sifat dari eksploitasi atau sengaja diperkenalkan oleh penyerang untuk menghindari deteksi. Dalam pengaturan di mana *server web* diperiksa, *delay* klasifikasi mengkonsumsi sebagian besar waktu ketika memeriksa *server*. Selain durasi untuk mengambil dan menganalisis respon *server* dilengkapi biaya menciptakan antrian *server* permintaan untuk mengeluarkan Tq, yang biasanya konstan. Tq, ti, tw, tr adalah empat faktor yang kita mempertimbangkan untuk menentukan kompleksitas komputasi dari berbagai algoritma.

Untuk meyakinkan respon ke *server* berbahaya terhadap permintaan, contoh *client honeypot* membutuhkan interaksi dengan *server* berurutan dan membuat klasifikasi setelah *server* masing-masing telah dikunjungi. Jika respon *server* itu harus diambil dalam paralel, yang harus didukung oleh *bandwidth* yang tersedia ke *client honeypot*.



Gambar 2.3 Ilustrasi Durasi Waktu pada Algortima *Brute force*

Gambar 2.3 berisi kode pseudo dari algoritma ini. Setelah antrian permintaan *server* telah dibuat, masing-masing *server* dikunjungi. Setelah masing-masing dikunjungi, *client honeypot* menunggu sebelum memeriksa perubahan kondisi pada sistem untuk mengklasifikasikan *server* sebagai berbahaya atau jinak. Jika *server* itu memang berbahaya, keadaan dari sistem di-*reset*. Kompleksitas waktu komputasi adalah O (n) sebagai waktu untuk memeriksa kenaikan *server* dipengaruhi faktor CSeq konstan dengan setiap *server* tambahan sebagai ditunjukkan dalam Gambar 2.2. Total durasi untuk memeriksa satu set *server* n diberikan oleh persamaan berikut:

tt = Tq + CSeqn
Tq = + (am (ti + + tw tr) + (1 - am) (ti + tw)) n,
dimana ti = s/b.

Algoritma ini diimplementasikan oleh sebagian besar implementasi *client honeypot* yang tinggi, seperti Honeymonkey, Capture-HPC, dan *client honeypot* dari University of Washington (UW).

```
def examine_servers_sequentially()
    server_queue = create_server_queue()
    while(server_queue.next?())
        server = server_queue.next()
        visit(server)
        wait()
        classification = check_state()
        if(classification == MALICIOUS)
            //found malicious server
            report_state_change()
            reset_state()
        end
    end
end
```

Gambar 2.4 Algoritma *Brute force*

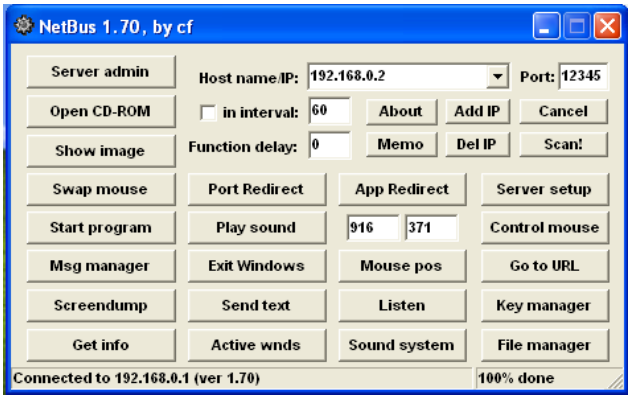
Untuk lebih memahami kerja dari metode honeypot, digunakan aplikasi yang mengemulasikan kerja dari metode honeypot. Lalu untuk menguji aplikasi tersebut dibutuhkan suatu Trojan yang berfungsi ntuk mnginfeksi system tersebut. Seperti yang kita ketahui fungsi utama

Trojan ialah melakukan remote server, sehingga seseorang yang tidak berhak mengakses dapat bertindak sebagai administrator dari system tersebut.

Sering sekali system kita terinfeksi oleh berapa Trojan, sebagaimana kita ketahui program tersebut akan menginfeksi system yang telah mengaktifkan patch.exe. Patch.exe tersebut merupakan file milik Trojan yang apabila telah masuk kedalam system kita, maka system tersebut akan otomatis menjadi sasaran remote. Sistem tersebut akan mudah dikendalikan oleh hacker yang memanfaatkan program Trojan tersebut. Pada contoh penetrasi tersebut, saya mencoba memakai Trojan Netbus.

Trojan yang digunakan kali ini ialah NetBus, adapun fitur yang terdapat pada aplikasi tersebut dalam me-remote server ialah :

- buka tutup CDROM
- menggerakkan mouse
- swap mouse button
- start aplikasi
- memainkan file WAV
- menampilkan pesan di layar
- shutdown, reboot,
- dll
- Berikut merupakan gambaran dari infeksi yang dilakukan oleh Trojan, yaitu seseorang yang meremote system serta efek yang didapat dari system tersebut :

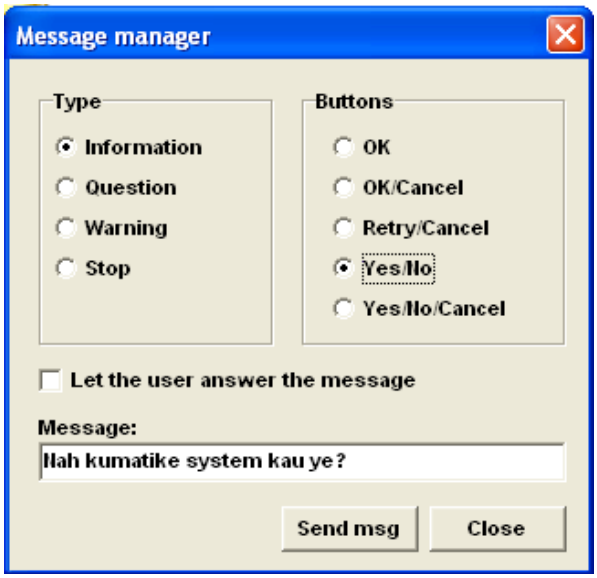


Gambar 2.5 Netbus 1.70

Pada gambar diatas terlihat dari beberapa button tersebut ialah gambaran dari fitur yang disediakan oleh netbus. Untuk mengaktifkan program tersebut, saya menonaktifkan antivirus. karena program tersebut tidak akan running pada system yang memiliki antivirus didalamnya. Program tersebut dijalankan di computer client, dimana computer yang akan diserang diasumsikan sebagai server. Untuk menjalankan fungsi dari Trojan tersebut, terlebih dahulu kita harus menginfeksi system server dengan patch.exe yang merupakan file bawaan dari Trojan. Apabila patch.exe telah aktif di computer server tersebut, proses peremotan akan berjalan.

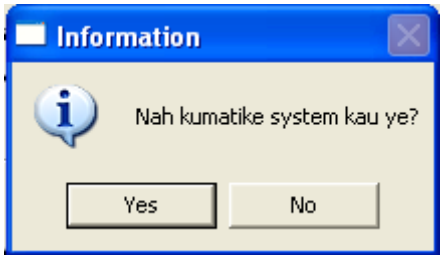
Seperti yang terlihat pada gambar,kita dapat melakukan apa saja dengan cara menginputkan ip dari computer server dan melakukan koneksi dengan menekan tombol connect. Apabila koneksi berjalan, maka akan keluar informasi pada sudut bawah program.

Dibawah ini merupakan salah satu fitur yang dijalankan oleh Trojan.



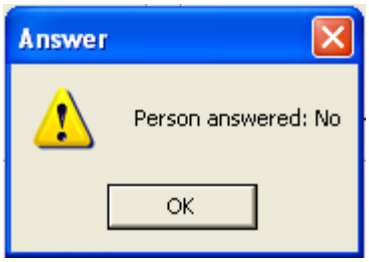
Gambar 2.6 Message manager

Saya telah menjalankan salah satu fitur yang terdapat pada Trojan yaitu Message manager. Dengan fitur tersebut kita dapat mengirimkan suatu message kepada computer server yang telah kita remote. Lalu disini terlihat beberapa opsi yang kita bisa gunakan untuk format dari message yang akan dikirim. Kemudian message yang dikirim akan tampil pada computer server sebagai berikut.



Gambar 2.7 Information

Computer server diharuskan merespon dari message yang telah dikirim oleh Trojan tersebut.

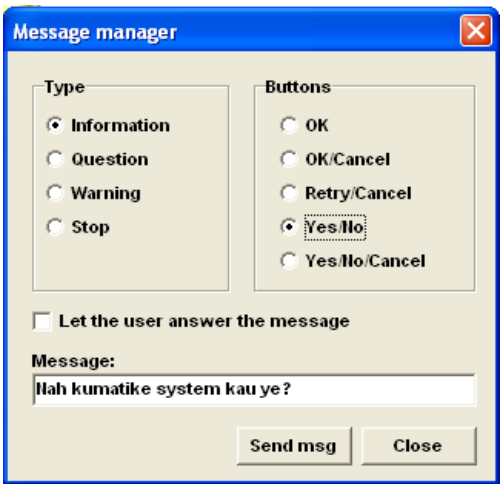


Gambar 2.8 Information

Respon yang dijawab oleh server akan dikirim kembali, kemudian akan tampil message respon tersebut pada computer yang meremote.

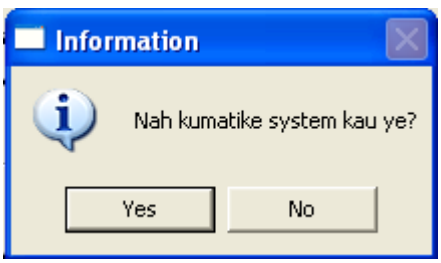
Ini merupakan bagian kecil dari segala fitur yang terdapat pada Trojan. Banyak hal yang bias kita lakukan yaitu men shutdown system tersebut, atau yang lebih parah merubah susunan dari directory yang terdapat pada system, bahkan menghapus sebagian data padasistem server tersebut.

Dibawah ini merupakan salah satu fitur yang dijalankan oleh Trojan.



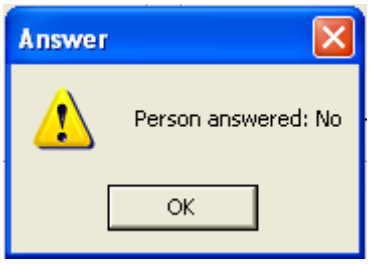
Gambar 2.9 Message Manager

Saya telah menjalankan salah satu fitur yang terdapat pada Trojan yaitu Message manager. Dengan fitur tersebut kita dapat mengirimkan suatu message kepada computer server yang telah kita remote. Lalu disini terlihat beberapa opsi yang kita bisa gunakan untuk format dari message yang akan dikirim. Kemudian message yang dikirim akan tampil pada computer server sebagai berikut.



Gambar 2.10 Information

Computer server diharuskan merespon dari message yang telah dikirim oleh Trojan tersebut.



Gambar 2.11 Answer

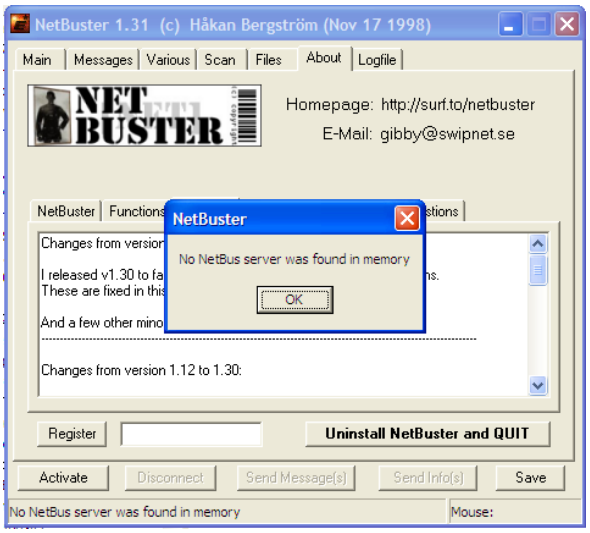
Respon yang dijawab oleh server akan dikirim kembali, kemudian akan tampil message respon tersebut pada computer yang meremote.

Ini merupakan bagian kecil dari segala fitur yang terdapat pada Trojan. Banyak hal yang bias kita lakukan yaitu men shutdown system tersebut, atau yang lebih parah merubah susunan dari directory yang terdapat pada system, bahkan menghapus sebagian data padasistem server tersebut.

Berikut ini merupakan solusi yang bisa dilakukan oleh seorang admin yang servernya telah terjangkit Trojan yaitu dengan menggunakan honeypot. Honeypot tersebut berfungsi sebagai suatu sisteminformasi yang memiliki nilai kebohongan bagi siapa saja yang secara tidak sah atau tidak memiliki hak unutk akses ke sumber system informasi tersebut. Sehingga hacker akan terkecoh, sekaligus admin dapat mengetahui trik-trik yang dilakukan oleh hacker tersebut.

Aplikasi yang saya gunakan untuk menjalankan metode dari honeypot tersebut ialah NetBuster. Aplikasi tersebut selain bisa merecord segala bentuk serangan dari Trojan tersebut, aplikasi ini juga dapat melakukan serangan balik untuk melakukan pembalasan dendam ke hacker tersebut.

Tool netbuster tersebut pada saat dijalankan akan menghapus semua server netbus, baik di memori ataupun di registri. Akan tetapi computer yang meremote tidak akan terputus koneksinya terhadap computer server, sehingga hacker tidak akan sadar akan hilanganya file yang Trojan yang telah aktif dikomputer server. Karena netbuster akan menggantikan peran dari netbus, dimana setiap command yang dilakukan oleh Trojan akan direcord oleh netbuster dalam sebuah file log. Perlu diingat, karena keseluruhan file dari netbus telah diremove dari system maka command yang dieksekusi oleh Trojan tidak akan berfungsi di server.



Gambar 2.12 Netbuster 1.31

3 Algoritma Divide and Conquer

Algoritma Divide and Conquer dapat digunakan oleh *client honeypot* untuk berinteraksi dan mengklasifikasi *server* yang berbahaya. Demikian pula pada algoritma *brute force*, algoritma ini mampu mengidentifikasi respon tertentu dari *server* yang berbahaya yang dapat menyebabkan perubahan keadaan yang tidak sah, tetapi algoritma ini dirancang untuk menjadi lebih cepat. Hal ini didasarkan pada membagi dan menaklukkan paradigma algoritma.

Kompleksitas waktu dari algoritma divide and conquer jauh lebih baik daripada pendekatan brute force. Brute force memiliki kompleksitas waktu O (n), sedangkan divide and conquer memiliki kompleksitas O (log (n)).

```
def examine_servers_dac()
  server_queue = create_server_queue()
  while(server_queue.next?())
      buffer = get_buffer(server_queue, SIZE)
      cache_buffer(buffer)
      visit_and_analyze_buffer(buffer)
  end
end

def visit_and_analyze_buffer(buffer)
  while(buffer.next?())
      server = buffer.next()
      visit(server)
  end

  wait(INTERVAL)
  classification = check_state()
  if(classification == MALICIOUS)
      reset_state()
      if(buffer.size == 1)
          //found malicious server
          report_state_change()
      else
          first_half = buffer.get_first_half
          visit_and_analyze_buffer(first_half)
          sec_half = buffer.get_sec_half
          visit_and_analyze_buffer(sec_half)
      end
  end
end
```

Gambar 3.1 Algoritma Divide and Conquer

4 Implementasi

Pada algoritma Divide and Conquer, dilakuakn pembagian kumpulan permintaan *server* ke dalam *buffer* ukuran k dan mengirimkan permintaan *server* pada *buffer*. Perubahan kondisi hanya diperiksa setelah semua respon *server* sudah diambil. Seharusnya respon dari *server* berbahaya terdeteksi, *buffer* dibagi menjadi dua bagian dan algoritma rekursif diterapkan untuk masing-masing setengah bagian. Jika *buffer* menyusut ke ukuran satu dan klasifikasi berbahaya telah dibuat, algoritma mengidentifikasi respon *server* yang menyebabkan klasifikasi berbahaya. *Overhead* dari *server* yang mengambil tanggapan berulang kali selama jaringan dapat diatasi dengan *caching* respon *server* pada *server proxy*. Membagi kumpulan *server* ke dalam k-kelompok untuk memilih ukuran *server* berbahaya menurut distribusi binomial. Tergantung pada berapa banyak *server* berbahaya telah dipilih, jumlah operasi untuk mengidentifikasi setiap *server* berbahaya menggunakan algoritma di atas.

Dengan pemilihan nol *server* berbahaya, algoritma akan beroperasi sekali pada *buffer* dan keluar. Jika satu *server* berbahaya muncul dalam kumpulan, algoritma akan beroperasi 2log2 (k) +1 kali pada *buffer* untuk mengidentifikasi *server* berbahaya. Jika dua atau m lebih *server* berbahaya muncul dalam kumpulan, algoritma akan melintasi pohon biner untuk setiap respon *server* m berbahaya, sehingga akan ada m (2log2 (k) + 1) operasi.

Namun, beberapa operasi di bagian atas pohon biner dibagi sebagai percabangan terjadi turun lebih rendah dari pohon. Jumlah operasi bersama harus dikurangi, sehingga kasus terburuk jumlah total operasi untuk mengidentifikasi *server* menggunakan pendekatan ini adalah:

$$op(k,m) = \begin{cases} 1 & \text{if } m = 0, \\ m(2log_2(k)+1) & \\ (2mlog_2(m) - m + 1) & \text{if } m > 0. \end{cases} \quad (3)$$

Jumlah operasi eksekusi yang tidak mengidentifikasi *server* berbahaya adalah :

$$op_b(k,m) = \begin{cases} 1 & \text{if } m = 0, \\ ((log_2(k)+1) & \\ (log_2(m)+1))m & \text{if } m > 0. \end{cases} \quad (4)$$

dan jumlah operasi eksekusi yang mengidentifikasi *server* berbahaya adalah:

$$op_m(k,m) = m(log_2(k)log_2(m) + 2)1 \quad (5)$$

dimana m>0.

Gambar 3.2 menunjukkan sebuah *buffer* dari 32 *server* diperiksa oleh *client honeypot* menggunakan algoritma Divide and Conquer yang dijelaskan di atas. Pertama, *server* 1-32 diperiksa. Sebuah *server*

berbahaya terdeteksi di *buffer* ini, jadi *buffer* dibagi dalam dua dan *server* 1-16 dan 17-32 diperiksa. Karena kedua bagian mengindikasikan bahwa *server* berbahaya ada, *buffer* dibagi lagi dan diperiksa (*server* 1-8, 9-16, 17-24, dan 25-32). Dalam *buffer* dengan *server* 1-8 dan 17-24, tidak ada *server* berbahaya yang diidentifikasi. Dengan demikian, tidak akan investigasi lebih pada *buffer* tersebut. Pada *buffer* sisanya, *server* berbahaya sekali lagi identikasi dan algoritma rekursif diterapkan sampai *server* berbahaya 10 dan 26 diidentifikasi. Pohon ditraversal sebanyak dua kali untuk mengidentifikasi setiap *server*, namun percabangan terjadi setelah *buffer* dengan *server* 1-16 dan 16-32 mengidentifikasikan *server* berbahaya. Total 19 operasi dihitung, dimana 11 operasi memerlukan keadaan dari *client honeypot* untuk diatur ulang.

Seperti dijelaskan di atas, jumlah operasi untuk mengidentifikasi *server* berbahaya dalam *buffer* ditentukan oleh jumlah aktual *server* berbahaya dalam *buffer*. Jumlah *server* berbahaya m yang dipilih dengan ukuran *buffer* k dan kemungkinan memilih *server* berbahaya pm diberikan oleh distribusi binomial:

$$f(m;k;p_m) = \binom{k}{m} p_m^m (1-p_m)^{k-m} \qquad (6)$$

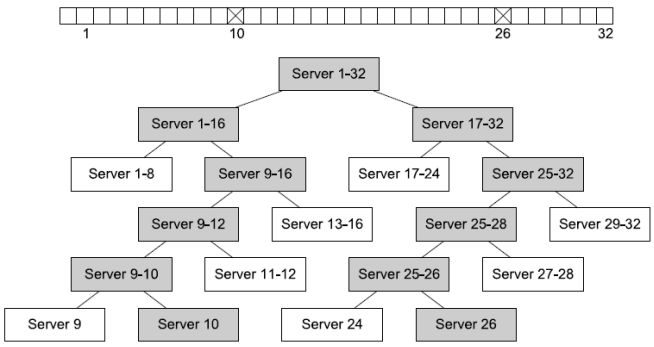
Distribusi binomial perlu diperhitungkan ketika menghitung total waktu untuk memeriksa *server*:

$$\begin{aligned} t_t &= T_q + C_{DAC} n \\ &= T_q + t_i n + \frac{n}{k}(f(0;k;p_m)t_w \\ &\quad + \sum_{0<m\leq k} f(m;k;p_m(op_b t_w + op_m(t_w + t_r)) \end{aligned}$$

dimana ti = s/b dan k adalah ukuran *buffer* yang dinamis. Total waktu dihitung dengan menambahkan waktu untuk membuat antrian *server* dan mengambil respon ditambah menunggu dan waktu ulang menurut distribusi binomial dan jumlah operasi yang diperlukan untuk mengidentifikasi berbagai respon *server* berbahaya.

Kompleksitas waktu keseluruhan dari algoritma ini adalah O (n). Hal ini identik dengan kompleksitas komputasi algoritma *brute force*. Namun, konstanta CDAC lebih kecil dari CSeq milik algoritma *brute force* memberikan persentase kecil dari *server* berbahaya dalam kumpulan, meningkatkan performansi.

Ukuran *buffer* k dapat diatur ke nilai apapun sesuai dengan Persamaan 3. Namun, ada nilai baik dan buruk bagi k.



Gambar 4.1 Contoh Algoritma Divide and Conquer pada *Client Honeypots*

Jika k terlalu kecil, algoritma akan berperilaku sama seperti algoritma *brute force*. Jika *buffer* terlalu besar, pilihan *server* berbahaya terlalu banyak dalam *buffer*, menyebabkan identifikasi kurang efisien dibandingkan *buffer* dibagi menjadi potongan kecil. Nilai optimal k diberikan oleh minimum global dari fungsi yang menangkap total jumlah operasi pada *binomial distribution*.

p_m	Optimum buffer size k
0.005	80
0.010	40
0.015	27
0.020	20
0.025	16
0.030	13
0.035	11
0.040	10
0.045	09
0.050	08

Tabel 4.1 Nilai optimum untuk ukuran dari *buffer* bergantung pada pm.

Tabel 3.1 menunjukkan nilai optimum ukuran *buffer* dengan variasi nilai pm. Algoritma yang disajikan dibatasi oleh pengaturan yang sebenarnya. Pertama, *bandwidth* membatasi jumlah permintaan yang dapat diperoleh. B*andwidth* menurun akan meningkatkan kemungkinan faktor utama yang menentukan waktu total tt untuk memeriksa kumpulan *server*. Kedua, asumsikan bahwa pemrosesan tanggapan dari satu set *server* mahal sebagai pengolahan satu respon *server* jika semua respon *server* telah di-*cache*. Ini adalah asumsi aman ketika kasus beberapa klien mengambil respon *server* dari *cache*.

Sebagai contoh, membuka beberapa tab dalam Internet Explorer untuk mengambil konten *web* tidak menimbulkan suatu *overhead*. Namun, membuka puluhan dokumen PowerPoint secara banyak pada satu mesin

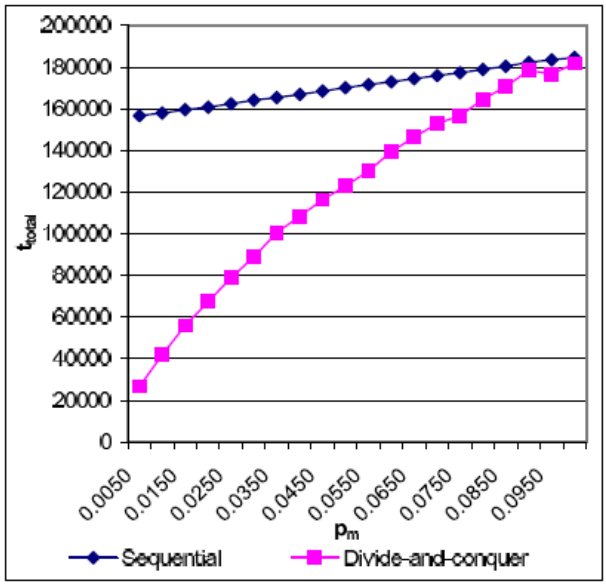
cenderung lebih intensif dan cepat dijalankan kedalam memory.

5 Perbadingan Algoritma Divide and Conquer dengan Algoritma Brute Force

Pada bagian ini akan dibandingkan antara algoritma Divide and Conquer dengan algoritma *brute force* yang digunakan oleh banyak *client honeypot* saat ini. Berikut disajikan perbandingan karakteristik kinerja dalam kondisi yang berbeda-beda melalui simulasi. Berikut nilai-nilai yang digunakan, yang ditujukan untuk mewakili keadaan nyata:

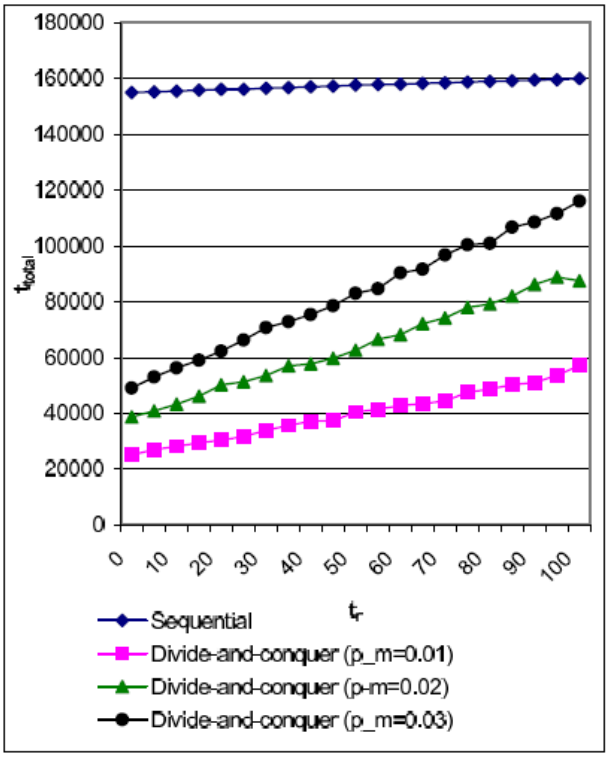
n = 5000,
pm = 0,01, ti = 1,2, tw = 30, tr = 60, tq = 500 dan ukuran *buffer* k = 40 diidentifikasi dari Tabel 3.1.

Nilai-nilai ini diperoleh terutama dari pengamatan penelitian sebelumnya [4, 8, 9] dan diturunkan dari kinerja perangkat keras yang mendasarinya.



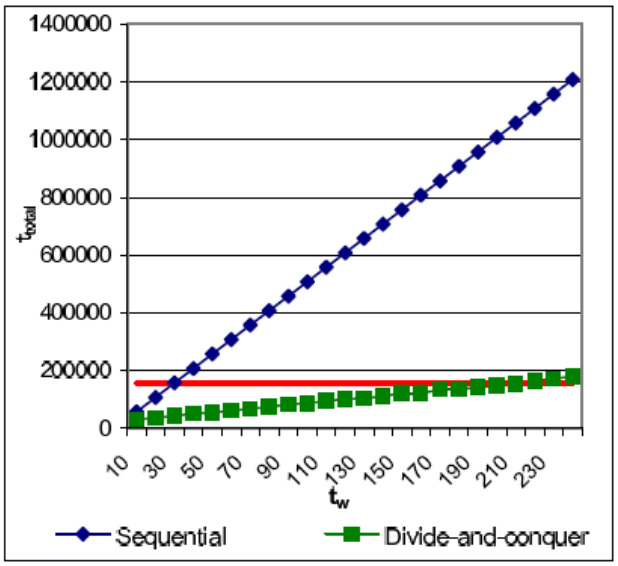
Gambar 5.1 Simulasi ttotal dengan variasi pm

Gambar 4.1 membandingkan kedua algoritma(Divide and Conquer dan brute force) dengan berbagai nilai-nilai pm, mulai dari pm = 0,005 untuk pm = 0,10. *Buffer* dengan ukuran k disesuaikan dengan Tabel 3.1. Ditunjukkan bahwa dalam konfigurasi ini, peningkatan kinerja dari Divide and Conquer berkisar dari 82,98% untuk nilai terkecil dari pm sampai kenaikan 1,51% untuk pm = 0,1. Peningkatan kinerja 68,61% terdapat pada pm = 0,0127. Gambar 4.1 menampilkan keefektifan dari algoritma Divide and Conquer. Pada algoritma ini hanya disediakan peningkatan kinerja dengan nilai kecil pm, yang umumnya kasus untuk *server* berbahaya dalam sampel acak.



Gambar 5.2 Simulasi ttotal dengan variasi tr dan pm

Gambar 4.2 menggambarkan total waktu yang diperlukan untuk memeriksa *server* dengan algoritma *brute force* dan algoritma Divide and Conquer dengan variasi nilai tr, waktu untuk me-*reset* keadaan sistem dalam kasus *server* berbahaya telah diidentifikasi, mulai dari tr = 0 sampai tr = 100. Gambar tersebut menunjukkan bahwa dalam berbagai macam situasi, algoritma Divide and Conquer memberikan keuntungan dalam performasi. Jika *client honeypot* memanfaatkan teknologi virtualisasi *sandboxing* yang tidak memerlukan *reset* keadaan *client honeypot* setelah permintaan *server* berbahaya telah ditemukan, keuntungan kinerja 83,78% masih dapat diamati. Untuk *client honeypots* yang membutuhkan waktu intensif membersihkan keadaan, seperti pencitraan kembali dan me-restart *client honeypot* misalnya, keuntungan kinerja 64,29% dapat diamati pada tr = 100s. Hal ini berlaku untuk nilai-nilai kecil dari pm, namun, jika pm menaik, kinerja menjadi menyusut dan dipengaruhi oleh nilai-nilai besar dari tr karena melakukan reset ulang dari *client honeypot* terjadi lebih sering.



Gambar 5.3 Simulasi ttotal dengan variasi tw

Gambar 4.3 menunjukkan perbandingan algoritma *brute force* dan algoritma Divide and Conquer dengan variasi nilai tw. Tw adalah waktu delay klasifikasi yang diperlukan untuk mendeteksi serangan yang mungkin sengaja atau sengaja tertunda sebelum memicu. Kinerja keuntungan meningkat dengan meningkatnya nilai tw sebagai klasifikasi merupakan sebuah operasi yang telah diminimalkan dengan algoritma Divide and Conquer. Penundaan klasifikasi selama 30 detik membawa keuntungan kinerja 72,08% sedangkan penundaan 240 detik membawa keuntungan sebesar 85,08%.

Perhatikan bahwa algoritma Divide and Conquer memungkinkan untuk meningkatkan delay klasifikasi tanpa melebihi waktu yang diperlukan untuk memeriksa *server* dengan algoritma brute force yang diwakili oleh garis horizontal yang memotong garis Divide and Conquer line di sekitar 210 detik. Hal ini berarti bahwa *client honeypot* dengan algoritma Divide and Conquer dapat menerapkan delay klasifikasi 210 detik dan sama baiknya dengan menggunakan *client honeypot* yang menggunakan algoritma *brute force* dengan delay klasifikasi 30 detik.

Simulasi pada gambar di atas menunjukkan bahwa algoritma Divide and Conquer memiliki karakteristik kinerja yang menguntungkan dibandingkan dengan algoritma *brute force*, dengan mempertimbangkan kinerja khas dan karakteristik nilai *client honeypots* dan *server* berbahaya. Algoritma Divide and Conquer tidak hanya mengatasi masalah kinerja *client honeypots*, tetapi dengan peningkatan kinerja memungkinkan untuk mengatasi masalah negatif pada eksploitasi yang memicu setelah penundaan waktu yang lebih lama karena mampu meningkatkan klasifikasi penundaan tanpa penalti.

6 Kesimpulan

Client honeypot membuat tugas dari deteksi menjadi lebih sederhana, efektif dan murah. Konsepnya sendiri sangat mudah dipahami dan diimlementasikan. *Client honetpot* sendiri ditujukan untuk mendeteksi serangan ayng dilakukan oleh *hacker* dengan mengecoh *hacker* tersebut dengan fasilitas server bohongan.

Pada makalah ini telah dijelaskan penggunaan algoritma Divide and Conquer untuk meningkatkan kinerja *client honeypots*. Pada konfigurasi biasa, algoritma Divide and Conquer memberikan peningkatan kinerja sekitar 72%. Seperti ditunjukkan di atas, keuntungan terdapat pada berbagai kondisi implementasi *client honeypot* dan kondisi dari *server* yang diperiksa. Sementara *client honeypot* yang dihasilkan akan tetap lambat dibandingkan dengan teknologi lainnya seperti *client honeypots* dengan interaksi rendah, keuntungan kinerja yang cukup dapat dicapai. Hal ini tidak hanya mengarah ke pemeriksaan lebih cepat dari *server*, tetapi juga memungkinkan seseorang untuk menyimpan pada perangkat keras / biaya lisensi.

Kemampuan untuk memeriksa sekumpulan *server* lebih cepat dengan algoritma Divide and Conquer dan memungkinkan seseorang untuk mengatasi masalah negatif yang terjadi. Dengan hardware yang ada, salah satu mampu meningkatkan delay klasifikasi dengan kemampuan untuk memeriksa sejumlah *server* yang identik dan menilai isu serangan tertunda. Sejauh ini, belum ada yang telah menyelidiki penundaan waktu pada tingkat yang komprehensi. Dengan menggunakan algoritma Divide and Conquer, hal ini dapat dilakukan.

Referensi

[1] http://en.wikipedia.org/wiki/Knuth-Morris-Pratt_algorithm
diakses tanggal 30 Nopember 2011, 15.00

[2] http://en.wikipedia.org/wiki/Client_honeypot
diakses tanggal 30 Nopember 2011, 15.00

[3] http://www.scribd.com/doc/73861935/Algoritma-Devide-and-Conquer
diakses tanggal 30 Nopember 2011, 15.00

[4] http://www.citeulike.org/user/ianwelch/article/2811949
diakses tanggal 30 Nopember 2011, 15.00

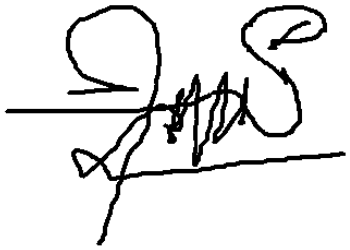
[5] http://arnetminer.org/viewpub.do?pid=569409
diakses tanggal 30 Nopember 2011, 15.00

[6] Munir, Rinaldi, "Diktat Kuliah IF2251 Strategi Algoritmik", Program Studi Teknik Informatika, ITB, 2007.

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, 30 Nopember 2011



Nugraha
13509096