PENERAPAN ALGORITMA DIVIDE AND CONQUER UNTUK MENINGKATKAN KECEPATAN DETEKSI DARI CLIENT HONEYPOTS BERINTERAKSI TINGGI

James Filipus (13507087)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Alamat e-mail: if17087@students.if.itb.ac.id, james_filipus_12@hotmail.com

ABSTRAK

Di jaringan atau Internet kita dapat saling terhubung satu sama lain begitu pula dengan server jahat, maka diperlukan pertahanan terhadap server jahat tersebut yaitu client honeypot. Kecepatan merupakan faktor penting untuk meningkatkan performa client honeypot interaksi tinggi dalam mendeteksi server-server jahat. Oleh karena itu diperlukan cara untuk meningkatkan performa dalam mendeteksi server jahat yaitu dengan mengaplikasikan algoritma *divide and conquer* untuk menggantikan algoritma sekuensial yang digunakan sebelumnya dalam metode pendeteksian server jahat. Pada algoritma *divide and* conquer, antrian server-server yang akan diperiksa dimasukkan ke dalam buffer, kemudian membagi buffer tersebut menjadi dua dan menerapkan algoritma *divide and conquer* secara rekursif pada kedua bagian buffer untuk mengidentifikasi server jahat. Demikian selanjutnya buffer-buffer tersebut terus dibagi sampai semua server jahat teridentifikasi. Peningkatan performa ini tidak hanya berakibat membuat client honeypot dapat memeriksa lebih banyak server dari sekumpulan sumber yang identik, tetapi juga memungkinkan peneliti untuk meningkatkan delay klasifikasi yang dibutuhkan untuk menyelidiki adanya server negatif palsu dengan penggunaan waktu penundaan tiruan dalam solusi yang sedang digunakan. Dengan peningkatan performa client honeypot maka dapat melindungi client dengan lebih baik. Selain itu kecepatan klasifikasi server pun meningkat sehingga mesin client dapat terhindar dari pengaruh server jahat.

Kata kunci: kecepatan, client honeypot, server jahat, divide and conquer, rekursif, negative palsu, buffer, delay klasifikasi.

1. PENDAHULUAN

Konektivitas broadband dan berbagai jenis layanan yang ditawarkan melalui Internet sudah banyak berperan dalam suksesnya *World Wide Web*. Hal ini membuat manusia menjadi lebih terhubung satu sama lain. Internet telah menjadi sumber informasi, hiburan dan komunikasi yang penting baik di rumah maupun di tempat kerja. Film dengan panjang penuh, suara melalui IP, dan toko-toko besar berbasis web adalah ebebrapa contoh layanan yang dapat ditemukan di internet.

Dengan konektivitas pada internet, mendatangkan ancaman yang berhubungan dengan keamanan. Akses tanpa izin, penolakan layanan, atau kendali penuh atas mesin oleh user jahat adalah contoh ancaman yang berhubungan dengan keamananan yang dihadapi di Internet. Karena Internet adalah jaringan global, sebuah serangan dapat dikirim dari mana saja di dunia dan dengan anonimitas yang tinggi. Para professional di bidang keamanan telah merespon terhadap ancaman ini dan menawarkan strategi pencegahan dan *mitigasi* berjangkauan luas. Mesin yang terhubung pada jaringan biasanya dilengkapi dengan paling tidak perangkat lunak antivirus dan *firewall*. Pertahanan tersebut biasanya sangat berhasil dalam melawan kebanyakan serangan-serangan tersebut.

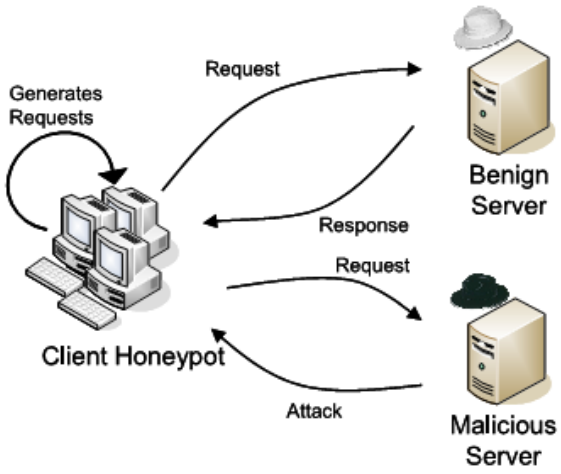
Karena vektor serangan dilarang oleh pertahanan, maka user jahat mencari jalan yang tidak terlindungi untuk menyerang. Salah satu tipe serangan besar adalah *client-side attack*, yang menetapkan aplikasi client sebagai target. Saat client mengakses sebuah server jahat, server tersebut mengirimkan serangan pada client sebagai bagian dari respon server terhadam permintaan client. Contoh umum dari serangan ini adalah server web yang menyerang web browser. Jika berhasil, sebagai contoh, server web dapat menginstall program sembarang pada mesin client. Pertahanan tradisional, seperti *firewall*s dan perangkat lunak antivirus, tidak efektif untuk melawan ancaman baru ini.

Untuk mengembangkan mekanisme pertahanan baru melawan ancaman-ancaman tersebut, maka kita perlu mempelajari server jahat. Tujuan dari mempelajari server jahat yaitu agar bisa menemukan server jahat pada suatu jaringan, yang merupakan perhatian utama. Untungnya

server jahat harus dapat diakses agar berhasil, sehingga memungkinkan kita untuk mencarinya. Client honeypot adalah teknologi baru yang bisal melakukan pencarian seperti demikian. Sekali server jahat sudah diidentifikasi, akses kepadanya dapat ditolak atau penegakan hukum dapat dilibatkan untuk membantu menutup elemen buruk ini pada jaringan. Sebagai tambahan, kita dapat mempelajari bagaimana server jahat beroperasi, yang akan membantu penelity keamanan untuk merancang mekanisme pertahanan yang lebih terarah dan efektif.

Client honeypot, dihadapkan pada beberapa tantangannya sendiri. Pertama, menjelajahi Internet dengan jutaan server. Mencari server jahat serupa dengan mencari jarum di tumpukan jerami. Kecepatan menjadi aspek penting untuk mengidentifikasi server jahat dengan cepar dan melindungi terhadapnya. Teknologi client honeypot yang digunakan sekarang lambat. Sayangnya, algoritma pendeteksian yang didasari pada pemantauan perubahan kondisi client honeypot yang tidak sah memakan banyak niaya, serta memakan waktu dan sumber daya. Mengacu pada peneliti terdahulu, Wang of Honeymongkey(client honeypot yang dikembangkan pada penelitian Microsoft), kebanyakan server jahat menginstall file jahat pertama dalam 30 detik. Dihadapkan dengan jumlah server di Internet, durasi 30 detik untuk mengklasifikasi sebuah server [1] sebagai server jahat membuat pencarian komprehensif di Internet tidak layak. Pembahasan pada makalah ini dimaksudkan untuk meningkatkan performa client honeypot dengan megaplikasikan startegi *divide-and-conquer* dalam cara client honeypot berinteraksi dengan dan mengkasifikasi server jahat.

2. CLIENT HONEYPOT



Gambar 1 Client Honeypot Architecture

Client honeypot menemukan server jahat pada jaringan. Mereka melakukannya dengan membuat antrian permintaan server, menerbitkan permintaan ini ke server satu per satu dan memakan respon server seperti ditunjukkan pada Gambar 1. Setelah respon dimanakan, client honeypot dapat melakukan analisa yang menentukan apakan server tersebut jahat atau tidak.

Klasifikasi ini didasarkan pada pemantauan sistem untuk perubahan kondisi yang tidak sah mengacu pada sistem setelah client honeypot sudah berinteraksi dengan sebuah server. Client honeypot adalah mesin berdedikasi dan karena tidak ada aktivitas lain yang mengacu padanya, perubahan kondisi yang tidak sah seperti proses baru, file yang baru diinstall, dll., bias dideteksi oleh client honeypot. Sekali perubahan kondisi terdeteksi dan klasifikasi telah dibuat , mesin perlu dikembalikan ke kondisi bersih sebelum dapat berinteraksi kembali dengan server lain.client honeypot yang menggunakan pendekatan ini juga dikenal sebagai client honeypot berinteraksi tinggi. Mencari web server jahat, sebagai contoh, seseorang akan mengambil halaman web dengan browser. Hal ini akan menyebabkan beragam perubahan kondisi yang akan terjadi pada sistem, seperti file yang ditulis ke dalam cache. Hal ini adalah peristiwa yang sah sehingga akan diacuhkan untuk diklasifikasi baik halamn web tersebut jahat atau tidak. Akan tetapi, bila file exe baru muncul di folder start-up, maka halamn web akan diklasifikasi sebagai halaman web jahat karena hanya serangan yang berasal dari halaman page itu yang dapat menyebabkan pergantian file ini dalam folder start-up.

Sebagian kecil implementasi dari client honeypot berinteraksi tinggi yang ada saat ini : HoneyClient, Honeymongkey, client honeypot University of Washington (UW), the CHP System, dan Capture-HPC. Client-client honeypot tersebut fokus pada web server jahat, yang berinteraksi dengan mereka dengan cara mengarahkan web browser pada sistem honeypot yang didedikasikan. HoneyClient mendeteksi serangan yang

[1] Lebih tepatnya, respon server. Agar lebih mudahnya pada makalah ini digunakan istilah server karenarspon server jahat menunjukkan bahwa server tersebut jahat. Client honeypoy,selalu berurusan dengan respon server.

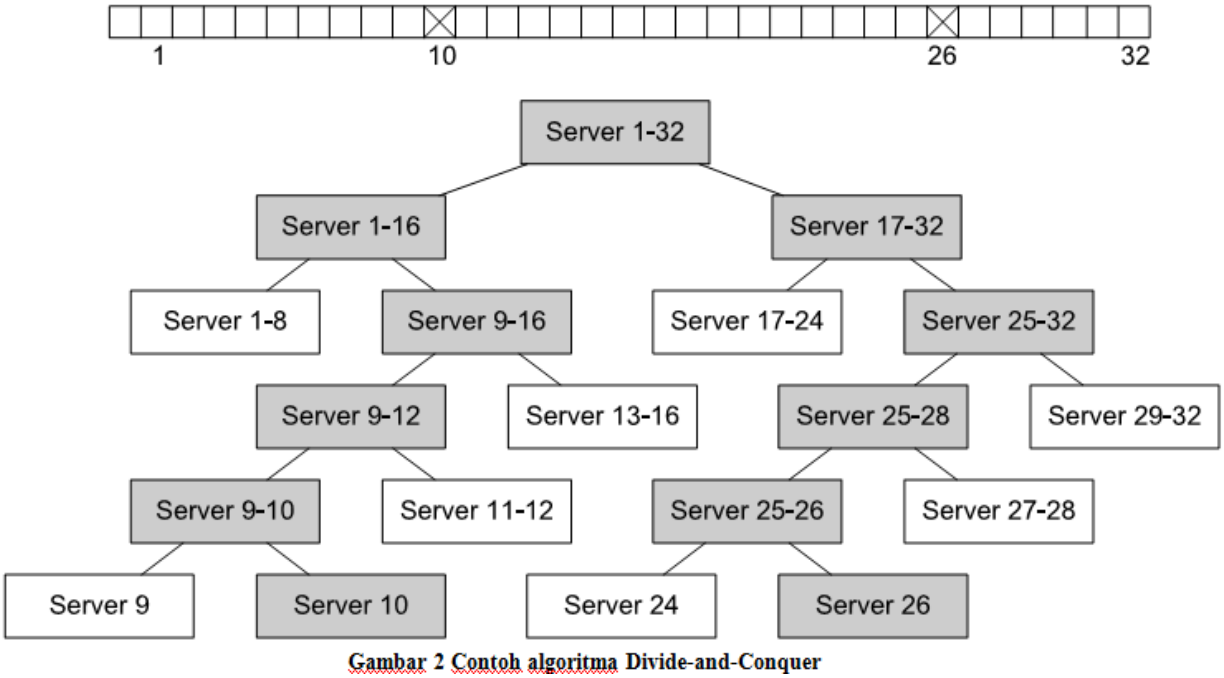
berhasil dengan memantau perubahan pada daftar file, direktori dan konfigurasi sistem setelah HoneyClient berinteraksi dengan sebuah server. Honeymonkey juga mendeteksi intrusi dengan memantau perubahan pada daftar file dan catatan registry, tapi Honeymonkey maju satu langkah lebih jauh dengan membantu memantau proses anak untuk mendeteksi serangan lain terhadap client. Selain kemampuan untuk mendeteksi perubahan kondisi tambahan, Honeymonkey juga berbasis peristiwa, dimana memungkinkannya untuk mendeteksi perubahan konsisi saat terjadi. UW client honeypot dan Capture-HPC menggunakan *event triggers* dari aktivitas file sistem,pembuatan proses, aktivitas registry, dan egagalan browser untuk mengidentifikasi serangan lain pada client. Sistem CHP, yang menggunakan CWSandbox sebagai mekanisme dasar untuk mendeteksi perubahan kondisi, dapat memantau tipe tambahan dari perubahan kondisi pada *event triggers*, seperti akses ke area memori virtual, area penyimpanan protektif, dan menginisiasi koneksi jaringan.

Kecepatan pendeteksian server jahat dari client-client honeypot ini sipengaruhi oleh bemacam-macam factor. Pertama, ada teknologi yang mendasari cara pendeteksian perubahan kondisi. Sebagai contoh HoneyClient menggunakan pemotretan yang memerlukan banyak waktu untuk dibuat sedangkan implementasi lainnya menggunakan *event triggers* yang mengizinkan pendeteksian dari perubahan konsdisi saat terjadi. Pada kenyataannya, ada factor tambahan yang mempengaruhi waktu total t_t untuk memeriksa n buah server. *Bandwidth* jaringan b dan ukiran rata-rata dari permintaan / respon s, yang mempengaruhi waktu t_i untuk mengambil respon server, waktu yang dibutuhkan dari menembalikan client honeypot ke kondisi bersih setelah server jahat ditemukan adalah t_r, dimana semuanya terkena dampak oleh persentase dari server jahat yang ada di jaringan p_m, dan berlangsung selama delay klasifikasi t_w. Delay klasifikasi adalah periode waktu menunggu yang diperkenalkan dengan suatu tujuan setelah respon server telah diterima sebelum klasifikasi dibuat. Hal ini diperkenalkan karena beberapa waktu terlewati sebelum banyak eksploitasi dipicu. Hal ini mungkin karena sifat asli dari eksploitasi atau diperkenalkan dengan sebgaja oleh penyerang untuk menghindari pendeteksian. Dalam suatupengaturan dimana server web diperiksa, delay klasifikasi memakan sebagian besar dari waktu saat memeriksa server. Sebagai tambahan pada durasi untuk mengambil dan menganalisa respon server yaitu biaya dalam membuat antrian permintaan server untuk diterbitkan T_q, yang biasanya konstan. T_q, t_i, t_w, t_r adalah empat factor yang diambil ke dalam *perhitungan* untuk menentukan kompeksitas komputasional dari beragam algoritma.

Pada umumnya client honeypot menggunakan algoritma sekuensial dimana setiap server harus dikunjungi secara sekuensial dan diklasifikasi satu per satu apakah server tersebut jahat atau tidak. Kompleksitas algoritma ini yaitu O(n). Ada pula client honeypot dengan algoritma bulk yang berinteraksi langsungdengan banyak server sekaligus. Algoritma ini tetap memiliki kompleksitan O(n).

3. ALGORITMA DIVIDE AND CONQUER

Pada bagian ini akan djelaskan algoritma baru untuk interaksi dan klasifikasi client honeypot terhadap server yang berpotensi sebagai server jahat. Sama denganalgoritma sekuensial, algoritma ini dapat mengidentifikasi respon server jahat yang spesifik dapat menyebabkan perubahan kondisi yang tidak sah, tetapi algoritma ini didesain agar leebih cepat. Algoritma ini didasarkan pada paradigm algoritma divide-and-conquer. Kompleksitas algoritma komputasional dari algoritma ini lebih baik dari pendekatan linier. Daripada O(n), algoritma ini mempunyai kompleksitas O(log(n)).
Divide and conquer diimplementasikan pada client



Gambar 2 Contoh algoritma Divide-and-Conquer

honeypot sebagaimana ditampilkan pada Gambar 2. Sama dengan algoritma bulk, set permintaan server dibagi-bagi ke dalam buffer dengan ukuran *k* dan menerbitkan permintaan server pada buffer. Perubahan kondisi hanya diperiksa setelah semua respon server sudah diambil. Pendeteksian server jahat dilakukan, buffer dibagi menjadi dua bagian dan algoritma diaplikasika secera rekursif pada tiap bagian.bila buffer menyusut menjadi satu ukuran dan klasifikasi server jahat telah dibuat, maka algoritma ini telah mengidentifikasi respon server yang menyebabkan

klasifikasi server jahat. Kerugian dari mengambil respon server secara berturut-turut melalui jaringan dapat diatasi dengan melakukan *caching* respon server pada server *proxy*.

Membagi sekumpulan server ke dalam kelompok-kelompok dengan ukuran *k* akan memilih server jahat mengacu pada distribusi binomial. Tergantung pada berapa banyak server jahat yang sudah dipilih, jumlah operasi untuk mengidentifikasi setiap server jahat menggunakan algoritma ini berbeda. Dengan tidak ada pemilihan server jahat, algoritma ini akan beroperasi sekali pada buffer dan tetap ada. Jika satu server jahat muncul di dalam kelompok, algoritma akan beroperasi $2log_2(k) + 1$ kali pada buffer untuk mengidentifikasi

```
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 Algoritma Divide and Conquer

server jahat. Apabila dua atau lebih server jahat *m* muncul di dalam kelompok, algoritma ini akan mentraversal pohon biner menurun untuk setiap respon server jahat *m* , maka akan ada $m(2log_2(k) + 1)$ operasi. Akan tetapi, beberapa operasi di puncak pohon biner dibagikan sebagaimana percabangan muncul menuruni pohon. Jumlah operasi yang dibagikan harus dikurangii, jadi kasus terburuk dari julah total operasi untuk mengidentifikasi server dengan pendekatan ini yaitu :

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

Jumlah operasi yang dieksekusi dan tidak mengidentifikasi server jahat yaitu :

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

dan jumlah operasi yang dieksekusi dan mengidentifikasi server jahat yaitu :

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

dimana m > 0.

Gambar 7 menunjukkan contoh. Sebuah buffer yang terdiri dari 32 server diperiksa oleh client honeypot menggunakan algoritma divide-and-conquer seperti yang digambarkan di atas. Pertama, server 1-32 diperiksa. Sebuah server jahat dideteksi di buffer ini, jadi buffer dibagi lagi menjadi dua dan server 1-16 serta 17-32 diperiksa. Karena kedua bagian mengindikasikan bahwa ada server jahat, maka buffer dibagi lagi lebih lanjut dan diperiksa (server 1-8, 9-16, 17-24, dan 25-32). Dalam buffer dengan server 1-8 dan 17-24, tidak ada server jahat yang diidentifikasi, oleh karena itu tidak ada investigasi lebih lanjut yang akan dilakukan pada buffer-buffer tersebut. Dalam buffer yang tersisa, server jahat sekali lagi teridentifikasi dan algoritma divide-and-conquer diaplikasikan secara rekursif sampai server jahat 10 dan 26 teridentifikasi. Pohon ditraversal dua kali untuk mengidentifikasi setiap server, tetapi percabangan terdapat setelah buffer dengan server 1-16 dan 16-32 mengidentifikasi server jahat. Total 19 operasi terhitung, di antaranya 11 operasi membutuhkan kondisi client honeypot untuk dikembalikan seperti semula.

Seperti dijelaskan di atas, jumlah operasi untuk mengidentifikasi sebuah server jahat dalam sebuah buffer ditentukan oleh jumlah aktual dari server jahat dalam buffer tersebut. Jumlah server jahat *m* ini yang dipilih dengan buffer berukuran *k* dan kemungkinan memilih server jahat p_m dapat diketahui dengan ditribusi binomial :

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

Ditribusi binomial perlu dibawa ke dalam perhitungan saat mengkalkulasi waktu total untuk memeriksa server :

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

dimana t_i = *s/b* dan *k* adalah ukuran buffer sembarang. Waktu total dikalkulasi dengan menambahkan waktu untuk membuat antrian server dan mengambil responnya ditambah waktu tunggu dan waktu untuk mengembalikan ke kondisi semula mengacu pada distribusi binomial dan jumlah operasi yang diperlukan untuk mengidentifikasi respon-respon server jahat. Kompleksitas komputasional keseluruhan dari algoritma ini adalah O(n). ini identik dengan kompleksitas komputasional dari algoritma sekuensial. Akan tetapi, konstanta C_{DAC} lebih kecil dari konstatnta C_{Seq} pada algoritma sequensial sehingga memberikan persentase server jahat dalam set, menyediakan kenaikan performa.

Ukuran buffer *k* dapat diset menjadi nilai berapa pun mengacu pada persamaan 3. Ada nilai baik dan buruk untuk k. jika k terlalu kecil, algoritma akan bertindak serupa dengan algoritma sekuensial. Jika buffer terlalu besar, bias terlalu banyak server jahat yang dipilih dalam buffer yang mengarah ke indentifikasi yang kurang efisien

Table 1Nilai optimal untuk ukuran buffer k bergantung pada p_m

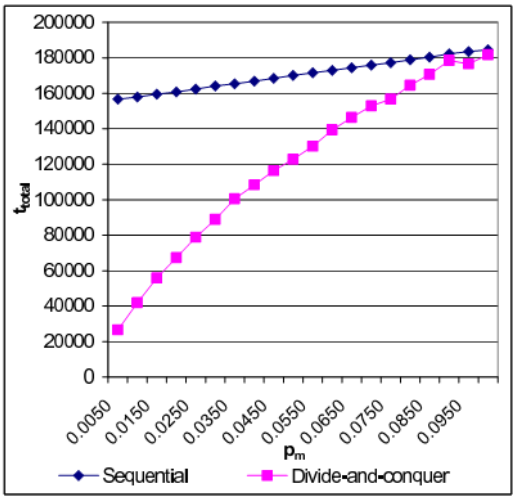
p_m	Optimum buffer size k
0.005	80
0.010	40
0.015	27
0.020	20
0.025	16
0.030	13
0.035	11
0.040	10
0.045	09
0.050	08

dibandingkan jika buffer dibagi menjadi bagian yang lebih kecil. Nilai *k* optimal yang diberikan oleh minimum global dr fungsi yang menghasilkan jumlah total operasi yang dibawa ke dalam perhitungan distribusi binomial.

Algoritma ini dibatasi oleh bebrapa pengaturan di dunia nyata. Pertama, bandwidth membatasi jumlah permintaan yang dapat diperoleh. Seiring dengan penurunan bandwidth, t_i akan meningkat bersama kemungkinan menjadi faktor utama yang menentukan waktu total t_t untuk memeriksa sekumpulan server. Kedua, diasumsikan bahwa memproses sekumpulan respon server sama mahalnya dengan memproses satu respon server jika semua respon server sudah di *cache*. Ini adalah asumsi aman bila sedikit turunan client mengambil respon server dari cache. Sebagai contoh, membuka beberapa tab dala Internet Explorer untuk mengambil konten webhampir tidak mendatangkan kerugian tambahan. Akan tetapi, membuka banyak presentasi Power Point pada satu mesin seperti lebih banyak sumber intensif dan akan cepat menemui batasan memori.

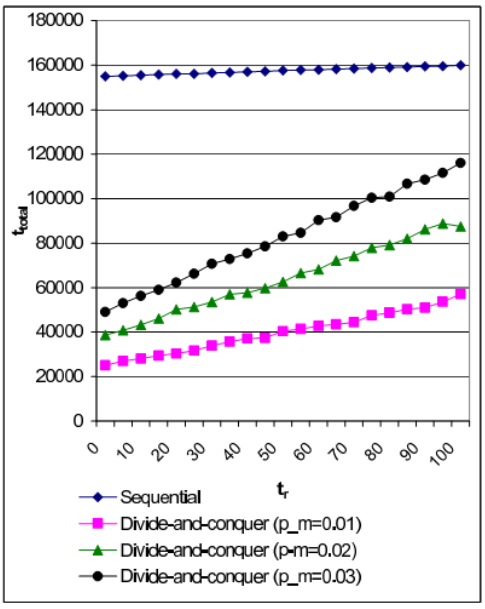
4. PERBANDINGAN DENGAN ALGORITMA SEKUENSIAL

Pada bagian ini, akan dibandingkan algoritma *divide-and-conquer* dengan algoritma sekuensial yang digunakan oleh kebanyakan implementasi client honeypot sekarang ini. Berikut adalah perbandingan performa karakteristik di bawah beragam kondisi melalui simulasi.



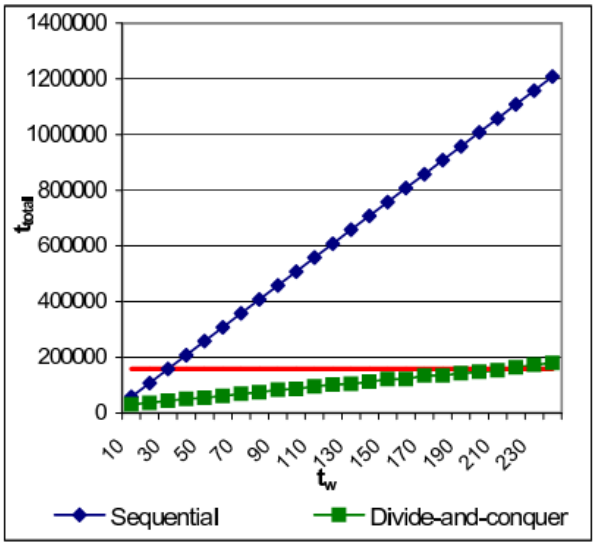
Gambar 4 Simulasi t_{total} dengan p_m yang beragam

Gambar 3 membandingkan kedua algoritma dengan beragam nilai pm mulai dari pm =0.005 sampai pm =0.10. Ukuran buffer k disesuaikan mengacu pada Table 1. Ini menunjukkan bahwa dalam konfigurasi khusus ini, kenaikan performa dari *divide-and-conquer* berkisar 82.98% unntuk nilai terkecil dari p_m sampai kenaikan 1.51 untuk p_m = 0.1. Kenaikan performa sebesar 68.61 % untuk pm =0.0127, sebuah nilai yang dioservasi oleh Wang's HoneyMonkey saat memeriksa URL yang mencurigakan [9]. Gambar di atas memperjelas asumsi mengenai efektivitas dari algoritma divide-and-conquer. Ini hanya menyediakan kenaikan performa dengan nilai p_m yang kecil, yang secara umum merupakan kasus server jahat dalam sampel acak.



Gambar 5 Simulasi t_{total} dengan t_r dan p_m yang beragam

Gambar 5 di atas menunjukkan bahwa dalam situasi dengan jangkauan yang luas, algoritma divide-and conquer menyediakan kenaikan performa.



Gambar 6 Simulasi t_{total} dengan t_w yang beragam

Berdasarkan gambar 6 kenaikan performa meningkat bersamaan dengan meningkatnya nilai t_w karena klasifikasi adalah operasi yang sudah diminimalisasi dengan algoritma *divide-and-conquer*.

Simulasi di atas mendemonstrasikan bahwa algoritma *divide-and-conquer* mempunyai karakteristik performa yang menguntungkan dibandingkan dengan algoritma sekuensial standar, membawa ke dalam perhitungan nilai performa yang khas dan karekteristik client honeypot dan server jahat. Algoritma ini tidak hanya memungkinkan kita untuk mngetahui alamat terbitan performa dari client honeypot, tetapi dengan kenaikan performa, kita juga dimungkinkan untuk mengetahui alamat dari negattif palsu pada eksploitasi yang memicu setelah delay yang lebih panjang, karena kita mampu meningkatkan delay kalsifikasi tanpa pinalti.

5. KESIMPULAN

Pada makalah ini telah dibahas mengenai penerapan algoritma *divide-and-conquer* untuk meningkatkan performa dari client honeypot berinteraksi tinggi. Pada konfigurasi khusus, algoritma ini menyediakan kenaikan performa sebeesar kira-kira 72%. Seperti dijelaskan di atas, kenaikan ini muncul pada beragam kondisi dari implementasi client honeypot dan sifat asli dari server yang diperiksa. Client honeypot yang dihasilkan akan tetap lambat dibandingkan dengan teknologi lain seperti client honeypot berinteraksi rendah, kenaikan performa yang berarti dapat diperoleh. Hal ini tidak hanya mengarah kepada pemeriksaan server yang lebih cepat, tetapi juga memungkinkan untuk menghemat biaya peragkat keras / perizinan.

Kemampuan untuk memeriksa sekumpulan server lebih cepat dengan algoritma ini juga memungkinkan kita untuk mengetahui alamat dari terbitan negatif palsu yang dikeluarkan. Dengan perangkat keras yang ada, kita dapat meningkatkan delay klasifikasi dengan kemampuan untuk memeriksa server dengan sumber yang identik dan mengakses terbitan dari serangan yang tertunda dengan jumlah yang identik. Sejauh yang diketahui, belum ada seorangpun yang menginvestigasi delay waktu pada level yang komprehensif. Algoritma ini membuka kesempatan untuk itu.

REFERENSI

[1] http://en.wikipedia.org/wiki/Client_honeypot, diakses 26 Desember 2009, pk 23.52.

[2] http://en.wikipedia.org/wiki/HoneyMonkey, diakses 26 Desember 2009, pk 23.52.

[3] http://en.wikipedia.org/wiki/Honeypot_(computing), diakses 26 Desember 2009, pk. 23.57.

[4] Ir. Rinaldi Munir, M.T., "Diktat Kuliah IF3051 Strategi Algoritma", Program Studi Informatika ITB, 2009

[5] Aladdin eSafe CSRT, 2005, "Malicious Code Report: The big Threats Shift" , 2006.

[6] Y.-M. Wang, "PersonalCommunication". 2006.

[7] http://www.honeyclient.org/trac, diakses 27 Desember 2009, pk 18.15.