TUGAS AKHIR EC7010
KEAMANAN SISTEM LANJUT
Dr. Ir. Budi Rahardjo

Tatyantoro Andrasto
(2 3 2 0 2 0 1 6)

MIKROELEKTRONIKA
DEPARTEMEN TEKNIK ELEKTRO
INSTITUT TEKNOLOGI BANDUNG

20 NOPEMBER 2003

DAFTAR ISI

DAFTAR ISI 1

DAFTAR GAMBAR & TABEL 2

Abstrak 3

1. Pendahuluan 3

2. TCP/IP Stack Fingerprinting 6

3. Fingerprinting Scrubber 12

3.1 Tujuan dan Keinginan Menggunakan Fingerprinting Scrubber 13

3.2 Desain dan Implementasi Fingerprinting Scrubber 13

3.2.1. IP scrubbing 15

3.2.2 ICMP scrubbing 17

3.2.3 TCP scrubbing 17

3.2.4 Timing attacks 18

4. Evaluasi terhadap Fingerprint Scrubber 21

4.1 Menangkal Fingerprint Scan 22

4.2 Throughput 23

4.3. Parameter yang dapat diukur 26

5. Perbandingan 28

6. Pengembangan 29

7. Kesimpulan 30

8. Daftar Pustaka 31

DAFTAR GAMBAR & TABEL

Gambar 1: Output dari `Nmap` scan 7

Gambar 2: Aliran data melalui FreeBSD kernel yang dimodifikasi 14

Gambar 3: Code fragment untuk normalize IP header flag 16

Gambar 4: *ICMP rate limiting* dari pantulan ICMP yang ditangkap dengan `tcpdump`. 19

Gambar 5: Set up eksperimen untuk pengukuran unjuk kerja dari *fingerprint scrubber*. 21

Gambar 6: Sistem Operasi *guess* 22

Gambar 7: Sambungan tiap detik melalui *gateway*. 27

Table 1. Throughput untuk single *untrusted* host ke *trusted* host menggunakan TCP (Mbps, +-2.5% at 99% CI) 24

Table 2. Throughput untuk single *trusted* host ke *untrusted* host menggunakan TCP (Mbps, +-2.5% at 99% CI). 25

Table 3: Throughput untuk single *untrusted* host ke *trusted* host menggunakan UDP (Mbps, +-2.5% at 99% CI). 25

Table 4. Throughput for a single *trusted* host to an *untrusted* host using UDP 26

TCP/IP Stack Fingerprinting Scrubber

Abstrak

*Paper ini menguraikan tentang desain dan implementasi dari sebuah TCP/IP stack fingerprint scrubber. Fingerprint scrubber adalah sebuah tools untuk menghadapi kemampuan remote user dalam mendeteksi operating sistem dari host yang lain dalam suatu jaringan. Membiarkan seluruh subnetwork melakukan remote scanning dan karakterisasi akan berakibat terbukanya sistem kekebalan keamanan (**security vulverability)**. Khususnya eksploitasi operating sistem merupakan langkah yang effisien dalam pre-scanning, karena eksploitasi lebih dalam biasanya dilakukan dalam software yang sama. Fingerprint scrubber bekerja pada network dan transport layer untuk mengubah kedua traffic dari sebuah group yang heterogen dari host menjadi paket-paket yang sudah diakali sehingga isinya bukan lagi petunjuk-petunjuk tertentu dari operating sistem host. Paper ini mengevaluasi kemampuan fingerprint scrubber yang diterapkan pada FreeBSD kernel dan keterbatasan-keterbatasan dari pendekatan ini.*

1. Pendahuluan

TCP/IP stack fingerprinting adalah proses penentuan identitas operating sistem dari sebuah remote host dengan menganalisa paket-paket dari host. *Tools-tools* yang tersedia bebas (seperti *Nmap* dan *queso*) digunakan untuk melakukan scan TCP/IP stack secara effisien dengan melakukan matching dengan cepat dari hasil-hasil pertanyaan dengan sebuah database dari operating sistem yang telah diketahui. Hasil identifikasi ini disebut *fingerprinting*, proses ini sama dengan identifikasi seseorang yang belum dikenal dengan mengambil ciri khusus sidik jari dan kemudian membandingkannya dengan sebuah database sidik jari. Perbedaanya adalah dalam sidik jari yang nyata, penegak hukum menggunakan sidik jari untuk melacak tersangka kriminal. Sedangkan dalam jaringan komputer, sidik jari digunakan oleh para penyerang untuk menyusun data-data dari targetnya dengan cepat.

Kami berpikir bahwa *fingerprinting tools* dapat digunakan untuk membantu pemakai yang berniat tidak baik dalam usahanya untuk menghancurkan sistem komputer. Seseorang dapat membuat sebuah *IP address profile* dan menghubungkan sistem operasi setelah penyusupan. *Nmap* dapat melakukan scan sebuah subnetwork yang terdiri dari

254 host hanya dalam beberapa detik. Hasil-hasil report dapat di-*compile* lebih dari berminggu-minggu atau bahkan bulan dan mencakup sebagian besar dari perilaku sebuah network. Saat seseorang menemukannya maka akan ada sebuah ide penyerangan baru untuk suatu sistem operasi tertentu, hal ini mempermudah penyerang untuk melakukan *generate script* kemudian melakukan penguasaan terhadap host yang menggunakan sistem operasi tersebut. Sebuah contoh perlawanan adalah pemasangan *code* pada sebuah mesin untuk ambil bagian pada sebuah penangkisan serangan. *Fingerprinting scan* memiliki potensial digunakan untuk hal-hal yang tidak semestinya dari *network resources* termasuk bandwidth dan waktu pengolahan oleh deteksi penyusupan sistem dan router.

Fingerprinting menyediakan ciri-ciri pasti yang menentukan dari sebuah sistem operasi. Sebagai contoh, *Nmap* memiliki pengetahuan dari 21 ciri dan perbedaan-perbedaan dari versi Linux. Metode-metode lain dari penentuan suatu sistem operasi umumnya menggunakan perkiraan sebab mereka menggunakan metode pada level aplikasi. Contohnya adalah *banner message* yang diterima user saat mereka menggunakan telnet untuk disambungkan ke mesin. Sebagian besar sistem bebas (gratis) mengiklankan sistem operasi mereka dengan cara ini. Paper ini tidak setuju dengan penghentian *application-level fingerprinting* karena hal ini harus diselesaikan pula pada level aplikasi.

Hampir semua sistem yang terhubung dengan internet tidak kebal terhadap *fingerprinting*. Bagian yang penting dari sistem operasi tidak hanya *TCP/IP stack* yang diidentifikasi oleh *fingerprinting tools*. Router, switch, hub, bridge, embedded system, printer, firewall, web camera, dan bahkan game console memungkinkan untuk diidentifikasi. Sebagian besar dari sistem-sistem itu seperti router, adalah bagian penting dari infrastruktur internet, dan kesatuan infrastruktur adalah masalah yang lebih serius daripada kesatuan *end host*. Cara proteksi tiap sistem sangat diperlukan.

Sebagian orang mungkin mempertimbangkan *stack fingerprinting* adalah gangguan yang lebih ringan dibandingkan dengan serangan terhadap sistem security. Seperti halnya sebagian besar *tools*, *fingerprinting* memiliki sisi baik dan sisi jahat dalam pemakaiannya. Network administrator harus mampu mengendalikan fingerprint machine dibawah kontrol mereka dan untuk mendeteksi kelemahannya. *Stack fingerprinting* tidak memerlukan ijin atau indikasi dalam melakukan kejahatan, tetapi kami percaya bahwa

jumlah scan akan meningkat seiring dengan jumlah orang yang mengakses internet dan menemukan kemudahan untuk menggunakan *tools* seperti Nmap. Seperti diketahui, administratur jaringan tidak ingin membuang-buang waktu dan uang untuk melacak hal-hal kecurangan yang terjadi dari waktu ke waktu. Sebagai gantinya mereka memilih membuat back up atas semua resource-nya untuk menghadapi penyusupan yang merajalela. Atau terdapat jaringan yang memiliki lebih dari satu otoritas untuk pengendali administratif. Atau juga sebuah *tools* yang dapat mendeteksi *fingerprinting scan* tetapi tidak memerlukan perhatian dari administratur untuk melacak dan mampu menjaga mereka dari penetrasi dalam jaringan lokal.

Paper ini menjelaskan desain dan implementasi dari suatu *tools* untuk menangkal *TCP/IP stack fingerprinting*. Kita sebut *tool*s ini dengan istilah *fingerprinting scrubber*. *Fingerprint scrubber* adalah sela transparan yang terletak antara internet dan jaringan yang dilindungi. Cara pemakaian *scrubber* adalah meletakannya di depan suatu rangkaian host atau suatu rangkaian komponen infrastruktur dari sebuah jaringan. Tujuan dari *tools* ini adalah untuk menghadang sebagian besar teknik-teknik *stack fingerprinting* pada umumnya secara cepat, terukur dan transparan.

Kami akan paparkan evaluasi hasil eksperimen dari *tools* dan memperlihatkan bahwa implementasi usaha penghadangan terhadap fingerprint scan yang dikenali disiapkan juga untuk menghadapi scan masa mendatang. Kami juga perlihatkan bahwa *fingerprint scrubber* ini sesuai dengan kemampuan dari *IP forwarding gateway* pada hardware yang sama dan diusahakan sesuai dengan rentang yang lebih terukur daripada *transport-level firewall*.

Bab-bab selanjutrnya diatur sebagai berikut. *TCP/IP stack fingerprinting* dijelaskan lebih detil di bab 2. Di bab 3 kami jelaskan tentang desain dan implementasi dari *fingerprint scrubber*. Di bab 4 kita evaluasi tentang validitas dan kemampuan dari *scrubber*. Bab 5 akan membicarakan hal-hal pekerjaan yang ada hubungan dengan *fingerprinting scrubber* dan bab 6 membicarakan arah perkembangan ke depan. Akhirnya kesimpulan dari pekerjaan ini akan dimuat di bab 7.

2. TCP/IP Stack Fingerprinting

Saat ini *tools* untuk *TCP/IP fingerprinting* yang paling lengkap dan luas pemakaiannya adalah *Nmap*. Dengan menggunakan suatu database yang lebih dari 450 ciri untuk dibandingkan ke TCP/IP stack dengan operating sistem tertentu atau hardware tertentu. Database ini termasuk sistem operasi yang komersial, router, switch, firewall dan sistem-sistem yang lain. Setiap sistem yang percaya bahwa TCP/IP adalah potensial untuk masuk database akan meng-update-nya secara priodik. *Nmap* merupakan software yang bebas untuk didownload dan mudah untuk digunakan. Dengan alasan ini, akan kita batasi pembahasan kita tentang *tools fingerprinting* ke *Nmap*.

Nmap fingerprint adalah sebuah sistem dengan 3 step. Pertama, kemampuannya sebagai port scan untuk menemukan satu set open dan closed TCP dan UDP port. Kedua, kemampuan untuk melakukan generate bentuk paket-paket kemudian mengirimnya ke *remote host* dan mendengar tanggapannya. Ketiga, menggunakan hasilnya untuk diolah dan menemukan masukan yang sesuai dengan database yang dimilikinya.

Nmap menggunakan satu set yang terdiri dari 9 macam tes untuk menentukan pilihan dari operating sistem. Setiap tes terdiri dari satu atau lebih paket-paket dan tanggapan yang diterimanya. Delapan tes yang dilakukan *Nmap* ditujukan pada TCP layer dan satu ditujukan pada UDP layer. Tes terhadap TCP adalah yang paling penting karena TCP memiliki beberapa option dan variabel dalam implementasinya. *Nmap* melihat perintah dari option-option TCP, pola dari sejumlah initial sequence, IP-level flag seperti fragment bit, TCP flag seperti RST, ukuran windownya dan masih banyak hal-hal lain. Untuk lebih detilnya termasuk option-option spesific dalam paket tes dapat dilihat di homepage yang memuat tentang *Nmap*.

Gambar 1 adalah contoh hasil keluaran dari *Nmap* saat melakukan scanning pada sebuah web server dan yang satunya scanning terhadap perangkat printer. Hasil perkiraan dari tes beruntun terhadap TCP didapat dari kemampuan *Nmap* mendeteksi tentang bagaimana sebuah host menggerakan sejumlah initial sequence untuk tiap *TCP connection*. Banyak sistem operasi komersial menggunakan model acak, positive increment, tetapi sistem yang lebih sederhana mengarah kepada penggunaan fixed increment atau increment berdasar pada waktu membuat sambungan.

Gambar 1: Output dari `Nmap` scan

(a) web server running Linux

```
TCP Sequence Prediction:
            Class=truly random
            Difficulty=9999999 (Good luck!)
Remote operating system guess:
            Linux 2.0.35-37
```

(b) shared printer

```
TCP Sequence Prediction:
            Class=trivial time dependency
            Difficulty=1 (Trivial joke)
Remote operating system guess:
            Xerox DocuPrint N40
```

Sementara *Nmap* terdiri dari beberapa fungsional dan bekerja dengan baik untuk kemampuan sebagai *fingerprinting* yang sangat presisi, tetapi ini bukan peralatan yang dapat dipakai untuk semua teknik/cara. Bermacam scan dapat dijalankan dalam waktu bersamaan. Sebagai contoh, untuk menentukan apakah sebuah host menggunakan *TCP Tahoe* atau *TCP Reno* dengan membuat paket hilang tiruan dan melihat bagaimana caranya melakukan perbaikan (recovery). Kita bicarakan perlakuan ini dan bagaimana penyelesaiannya pada bagian 3.2.4. Juga seseorang dapat menggunakan metode seperti *social engineering* atau teknik pada tingkatan aplikasi untuk menentukan sistem operasi yang digunakan host. Seperti teknik-teknik di luar dari cakupan pembicaraan ini, akan selalu dibutuhkan cara untuk menghadang scan yang dilakukan *TCP/IP fingerprinting* selama *tools* untuk aplikasi *fingerprinting* terus berkembang. Selama ini *TCP/IP fingerprinting* adalah metode yang paling cepat dan paling mudah untuk mengenali sistem operasi yang digunakan remote host.

METODOLOGI FINGERPRINT

Ada banyak teknik yang dapat digunakan untuk melakukan *fingerprint stack networking*. Dasarnya, hanya melihat benda yang berbeda diantara sistem operasi dan menulis perbedannya. Kemudian kombinasikan, berikutnya mempersempitnya secara tepat. Contohnya, *Nmap* dapat membedakan Solaris 2.4 vs Solaris 2.5-2.51 vs Solaris 2.6.

Juga dapat memberitahu kernel Linux 2.0.30 dari 2.0.31-34 atau 2.0.35. Berikut ini beberapa tekniknya:

FIN probe

Di sini kita mengirim paket FIN (atau paket tanpa flag ACK atau SYN) untuk membuka port dan menunggu respon. Tindakan RFC 793 yang benar adalah tidak ada respon tapi implementasi yang buruk seperti MS Windows, BSDI, CISCO, HP/UX, MVS, dan IRIX mengirim kembali RESET. Banyak peralatan menggunakan teknik ini.

Flag BOGUS probe

Queso adalah scanner pertama yang menggunakan tes pintar ini. Idenya adalah mengeset flag TCP undefined (64 atau 128) pada header TCP dari paket SYN. Mesin Linux terutama 2.0.35 menjaga flag ini tetap di keadaan set pada responnya. Saya tidak menemukan OS lain mempunyai bug ini. Tapi, beberapa OS tampak mereset koneksi ketika mereka dapat paket SYN+BOGUS. Tingkah laku ini dapat berguna untuk mengidentifikasi mereka.

Sampling TCP ISN

Ide di sini untuk menemukan pola pada angka urutan inisial yang dipilih oleh implementasi TCP ketika merespon permintaan koneksi. Dapat dikategorikan ke dalam banyak grup misalnya tradisional 64K(kotak UNIX lama), peningkatan random (versi terbaru Solaris, IRIX, FreeBSD,Digital UNIX, Cray, dsb), random asli (Linux 2.0.*, OpenVMS, AIX terbaru, dsb). Kotak Windows (dan sedikit yang lain) menggunakan model *time dependent* (tergantung waktu) dimana ISN dinaikkan sedikit setiap waktu periode, ini mudah diserang seperti 64K lama. Mesin selalu menggunakan ISN yang sama, dapat dilihat pada 3Com hubs(memakai 0x803) dan printer AppleLaserWriter(memakai 0x7001). Grup ini dapat dibagi seperti peningkatan random dengan variasi perhitungan, pembagi terbesar, dan fungsi lain pada set angka urutan dan selisihnya. Dicatat bahwa penciptaan ISN mempunyai implikasi keamanan yang penting. Untuk informasi yang lebih banyak , hubungi "ahli keamanan" Tsutomu "Shimmy" Shimomura di SDSC. *Nmap* adalah program pertama yang menggunakan ini untuk identifikasi OS.

Bit "don't fragment"

Banyak OS memulai untuk mengeset bit IP *Don't Fragment* pada beberapa paket yang mereka kirim. Ini memberikan keuntungan performasi yang beragam(meskipun ini dapat

diabaikan-- inilah mengapa scan fragmentasi *nmap* tidak bekerja dari kotak Solaris). Pada beberapa kasus. tidak semua OS melakukan ini dan beberapa melakukannya pada kasus berbeda, jadi dengan memperhatikan bit ini kita dapat informasi tentang OS target. Saya belum melihat yang satu ini sebelumnya.

Window inisial TCP

Ini termasuk mengecek ukuran window pada paket yang dikembalikan. Scanner lama menggunakan window non-zero pada paket RST untuk mengartikan "BSD 4.4 turunan". Scanner terbaru seperti queso dan *Nmap* menjaga jejak window asli karena merupakan konstanta sesuai tipe OS. Tes ini memberikan kita banyak informasi karena beberapa OS dapat diidentifikasi secara unik lewat window sendiri(misalnya, AIX adalah OS yang menggunakan 0x3F25). Pada stack TCP untuk NT5, Microsoft menggunakan 0x402E. Menariknya, itu adalah angka yang digunakan OpenBSD dan FreeBSD.

Nilai ACK

Meskipun hal ini standar, implementasinya berbeda pada nilai yang digunakan untuk field ACK pada kasus tertentu. Sebagai contoh, kita mengirim FIN|PSH|URG ke port TCP yang tertutup. Banyak implementasi akan mengeset ACK sama seperti angka urutan inisial, meskipun Windows dan beberapa printer yang bodoh akan mengirim urutan+1. Jika kita mengirim SYN|FIN|URG|PSH ke port terbuka, Windows sangat tidak konsisten. Kadang-kadang mengirim kembali urutannya, lain waktu mengirim S++ atau nilai yang acak. Seseorang mesti bertanya-tanya kode apa yang ditulis MS sehingga berubah seperti itu.

ICMP Error Message Quenching

Beberapa OS mengikuti RFC 1812, saran untuk membatasi kecepatan mengirim pesan error. Misalnya, kernel Linux(pada net/ipv4/icmp.h) membatasinya 80 per 4 detik, dengan 1/4 detik penalti jika melewati. Satu jalan untuk mengetesnya adalah mengirim segudang paket ke beberapa port UDP secara acak dan menghitung yang tidak sampai diterima. Saya tidak menambah ini pada *Nmap* (kecuali untuk scan port UDP). Tes ini membuat deteksi OS menjadi lebih lama karena perlu mengirim segudang paket dan menunggunya kembali. Juga kemungkinan adanya paket yang drop akan menyakitkan.

ICMP Message Quoting

RFC memberi tahu pesan error ICMP perlu sejumlah kecil pesan ICMP yang menyebabkan error yang beragam. Untuk pesan "port tidak dapat dicapai", hampir semua implementasi mengirim IP header + 8 bytes kembali. Tapi, Solaris mengirim lebih banyak bit dan Linux lebih dari itu. Cantiknya, ini mengijinkan *Nmap* mengenali host Linux dan Solaris bahkan bila tidak ada port yang listening.

ICMP Error message echoing integrity

Saya dapat ide dari Theo De Raadt (developer OpenBSD) pada comp.security.unix. Mesin mesti mengirim kembali bagian pesan yang asli dengan error port yang tidak dapat dicapai. Beberapa mesin memakai header sebagai *scratch space* selama proses sehingga terjadi *bitwarp* oleh waktu yang didapat kembali. Misal , AIX dan BSDI mengirim kembali field IP 'panjang total' yang 20 byte terlalu tinggi. Beberapa BSDI, FreeBSD, OpenBSD, ULTRIX, dan VAXen mengirim IP ID yang kita kirim ke mereka. Ketika *checksum* berubah karena perubahan TTL, beberapa mesin (AIX, FreeBSD, dsb) mengirim kembali checksum inkonsisten atau 0. Beberapa berjalan dengan checksum UDP. *Nmap* melakukan 9 test berbeda pada error ICMP untuk mengendus perbedaan seperti itu.

Tipe Service

Untuk pesan port ICMP yang tidak dapat dicapai saya melihat nilai *Type Of Service* (TOS) dari paket yang kembali. Hampir semua implementasi memakai 0 untuk error ICMP ini meskipun Linux memakai 0xC0. Ini tidak mengindikasikan satu nilai standar TOS, tapi merupakan bagian field yang tidak terpakai (AFAIK). Saya tidak tahu mengapa ini diset, tapi jika mereka merubahnya ke 0 kita akan dapat mengidentifikasi versi lama dan baru.

Penanganan Fragmentasi

Ini merupakan teknik favorit Thomas H. Ptacek dari Secure Networks, Inc(sekarang milik user Windows di NAI). Ini mengambil keuntungan dari fakta bahwa implementasi berbeda sering menangani fragmen IP yang overlapping secara berbeda. Beberapa mengganti yang baru, lainnya masih suka yang lama. Di sana banyak perbedaan yang digunakan untuk menentukan bagaimana paket disusun kembali. Saya tidak menambahkan kemampuan ini karena saya tahu tidak ada jalan portable untuk mengirim

fragmen IP (khususnya, di Solaris). Untuk informasi lebih jauh tentang fragmen overlapping kamu dapat membaca catatan IDS (http://www.secnet.com/).

Pilihan TCP

Di sini ada sebuah tambang emas pembocoran informasi. Kecantikan pilihan ini adalah bahwa:

1) Mereka secara umum opsional, jadi tidak semua host mengimplementasikannya.
2) Jika host mengimplementasikannya dengan mengirim urutan dengan pilihan set, target menunjukkan dukungan pilihan itu dengan mensetnya di tanggapan.
3) Seluruh pilihan di satu paket dapat dimiliki untuk mengetes semuanya sekali saja

Nmap mengirim pilihan ini di hampir semua paket probe:

Window Scale=10; NOP; Max Segment Size = 265; Timestamp; End of Ops;

Ketika dapat respon , dapat dilihat pilihan mana yang dikembalikan dan didukung. Beberapa OS seperti FreeBSD saat ini mendukung semua, sedang yang lain seperti Linux 2.0.X mendukung sedikit. Linux 2.1.x kernel mendukung semuanya. Dengan kata lain, mereka dapat diserang prediksi urutan TCP. Bahkan jika beberapa OS mendukung pilihan sama, kamu dapat membedakannya lewat nilai pilihan itu.Contoh, jika kamu mengirim nilai MSS kecil ke kotak Linux, ini akan mengembalikan MSS kembali ke kamu. Host yang lain akan memberi nilai yang beda. Dan jika kamu dapat set pilihan yang sama DAN nilai sama, kamu tetap data membedakannya lewat pesan pilihan yang diberikan, dan dimana padding dipakai.

Contoh :

Solaris mengembalikan 'NNTNWME' yang berarti:

<no op><no op><timestamp><no op><window scale><echoed MSS>

Sedang Linux 2.1.122 mengembalikan MENNTW.

Pilihan sama, nilai sama, tapi pesan berbeda!

Kronologi Exploit

Bahkan dengan semua tes diatas, *nmap* tidak dapat membedakan antara stack TCP Win95, WinNT atau Win98. Ini agak mengejutkan, khususnya sejak Win98 datang 4 tahun setelah Win95. Kita mungkin berpikir mereka mengembangkan stack dengan jalan yang sama (seperti mendukung lebih banyak pilihan TCP) dan kita dapat mendeteksi perubahan dan membedakan OS. Sayangnya ini bukan kasusnya. Stack NT sama berantakan dengan stack '95 dan mereka tidak mengupgradenya untuk '98. Tapi jangan menyerah, ada solusinya. Kita dapat memulai dengan *Windows DOS attacks* (*Ping of Death*, *Winnuke*, dsb) dan pindah ke serangan yang lebih seperti *Teardrop* dan *Land*. Setelah serangan ping mereka untuk melihat apakah mereka *crash*. Ketika sudah *crash* kamu akan mempersempit apa yang mereka jalankan ke satu *service pack* atau *hotfix*, saya tidak menambahkan fungsi ini ke *nmap*.

Resistansi SYN Flood

Beberapa OS akan berhenti menerima koneksi baru jika kamu mengirim terlalu banyak paket SYN. Banyak OS hanya dapat menangani 8 paket. Linux kernel sekarang mengijinkan metode beragam seperti *SYN cookie* untuk mencegah menjadi masalah serius. Kemudian kamu dapat belajar tentang OS sasaran dengan mengirimkan 8 paket dari sumber ke port terbuka dan mengetes apakah kita dapat membuat koneksi ke port itu sendiri. Ini tidak diimplementasikan di *nmap* karena orang sebal jika kita membanjiri mereka SYN. Bahkan penjelasan bahwa kita melakukan sedikit percobaan untuk menentukan OS yang meraka pakai tidak dapat membantu untuk membuat mereka tenang.

3. Fingerprinting Scrubber

Kami perkenalkan sebuah *tools* yang disebut *fingerprinting scrubber* untuk membuang ambiguitas dari TCP/IP traffic yang memberikan petunjuk sebuah sistem operasi dari host. Pada bagian ini kami membicarakan tentang tujuan dan keinginan pemakaian dari *scrubber* serta desain dan implementasinya. Kami perlihatkan validitas *scrubber* dalam menghadapi *fingerprinting* scan yang dikenali dan memberi hasil unjuk kerjanya pada bagian berikutnya.

3.1 Tujuan dan Keinginan Menggunakan Fingerprinting Scrubber

Tujuan *fingerprint scrubber* adalah menangkal (block) teknik stack *fingerprinting* pada umumnya yang diketahui secara cepat, terukur dan caranya jelas. *Tools* ini diharapkan secara umum mampu untuk menangkal berbagai jenis scan, tidak hanya scan tertentu yang dilakukan oleh *fingerprinting* yang diketahui. *Scrubber* tidak harus sangat rahasia dan harus dapat digunakan untuk beberapa hubungan TCP secara *concurrent*. Juga *fingerprint scrubber* tidak boleh menyebabkan penampilannya menarik perhatian atau perbedaan sifat pada host-nya. Sebagai contoh, penggunaan *scrubber* ini selayaknya memiliki efek yang seminimal mungkin pada terhambatnya mekanisme kontrol TCP akibat delay atau dropping paket secara tidak efektif.

Kami menginginkan *fingerprinting scrubber* ditempatkan di depan dari satu set sistem dengan manggunakan hanya satu sambungan untuk sebuah jaringan yang besar. Kami berpikir *fingerprinting scrubber* akan menjadi implementasi utama sebuah mesin gateway dari sebuah sistem LAN yang heterogen (misal : Windows, Solaris, MacOS, printers, switches) untuk sebuah jaringan perusahaan yang besar. Tempat yang sesuai pada sebuah sistem untuk aplikasi ini adalah menjadi bagian *firewall* yang sudah terpasang. Pemakaian yang lain adalah meletakkan sebuah *scrubber* di depan kontrol hubungan (connection) dari router. Jaringan yang diproteksi harus dilindungi dari hubungan keluar karena semua paket yang lewat baik dari dan ke sebuah host harus melewati *scrubber*.

Karena *scrubber* mengharuskan traffic melewatinya maka administrator sebagai pihak yang dipercaya jaringan masih memiliki hak untuk melakukan scan terhadap jaringan. Alternatif lainnya adalah dengan menambahkan daftar akses IP atau mekanisme *authentication* lain kepada *fingerprint scrubber* untuk memberi ijin kepada host tertentu untuk melakukan bypass terhadap *scrubber*.

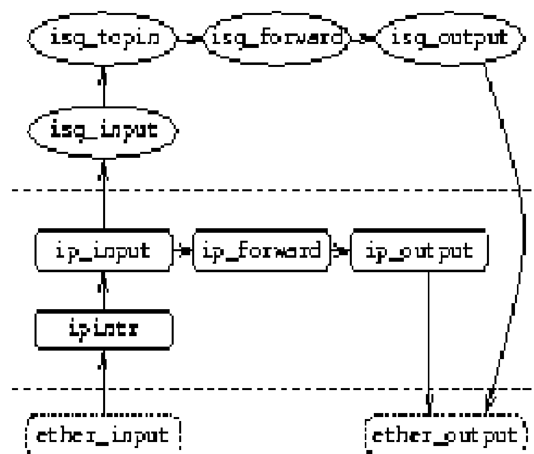
3.2 Desain dan Implementasi Fingerprinting Scrubber

Fingerprinting dirancang untuk diletakkan antara sebuah jaringan yang sudah dipercaya dari sistem yang heterogen dengan jaringan/hubungan yang belum dapat

dipercaya (misalnya internet). Jadi *scrubber* memiliki dua interface, yaitu interface yang dirancang untuk jaringan yang sudah dipercaya (*trusted*) dan interface yang dirancang untuk jaringan yang belum dipercaya (*untrusted*). Sebuah paket datang dari *untrusted* interface dikirim ke *trusted* interface secara bolak balik. Prinsip dasar perancangannya adalah data datang dari *untrusted* interface diperlakukan berbeda dari data yang dikirim ke *untrusted* interface.

Fingerprint scrubber beroperasi pada IP dan TCP layer. Agar dapat memiliki pengetahuan yang luas tentang scan *fingerprinting* yang berbahaya, kami mencoba implementasi sederhana. Sebagian dari tekniknya akan dibicarakan pada bagian yang membicarakan tentang bagaimana menangkal *Nmap*. Akan tetapi tujuan dari pekerjaan ini adalah bagaimana agar perkembangan *fingerprinting tools* dapat selalu dipantau. Dengan membuat *scrubber* beroperasi pada level generik untuk IP dan TCP, kami yakin mampu mengatasi rintangan yang cukup berat.

Gambar 2: Aliran data melalui FreeBSD
kernel yang dimodifikasi



Fingerprint scrubber berdasarkan pada protocol *scrubber* yang dibuat oleh *Malan*. Protocol *scrubber* beroperasi pada IP dan TCP layer dari protocol stack. Ini adalah satu set modifikasi kernel yang memungkinkan aliran fast TCP menghindari penyusupan TCP dan penghapusan serangan seperti yang dipaparkan oleh *Ptacek dan Newsham*. Protocol *scrubber* mengikuti perubahan status TCP dengan menjaga sebagian

kecil dari status untuk setiap hubungan, tetapi ini membiarkan bagian paling besar perawatan dari proses TCP dan statusnya sampai ke host tujuan. Ini membiarkan sebuah *tradeoff* antara kemampuan dalam penyelesaikan masalah tanpa melihat status dengan kontrol dari *full transport-layer proxy*. Protocol *scrubber* dimplementasikan dengan FreeBSD dan dibangun dengan FreeBSD 2.2.8 sebagai kelanjutannya.

Paket-paket yang datang melalui kedua arah (*trusted* interface dan *untrusted* interface) akan melewati Ethernet driver. IP paket yang masuk diarahkan ke *ip_input* melalui sebuah *software interrupt*, selanjutnya akan berjalan normal. Sebuah filter pada *ip_input* akan memutuskan apakah sebuah paket harus dikirim ke *TCP scrubbing code*. Jika tidak maka paket ini akan diijinkan untuk menjalani proses normal IP forwarding ke *ip_output*. Jika ini dijalankan maka *isg_input* (ISG : *Internet Scrubbing Gateway*) menampilkan *IP fragment reassembly* jika diperlukan dan melewatkan paket ke *isg_tcpin*. Di dalam isg_tcpin *scrubber* menjaga alur dari status hubungan TCP. Kemudian paket akan melewati *isg_forward* untuk menampilkan *TCP-level processing*. Akhirnya *isg_output* melakukan modifikasi terhadap *next-hop link level address* dan *isg_output* atau *ip_output* membawa paket langsung ke *device driver interface* yang sesuai untuk *trusted* atau *untrusted* link.

Kita juga harus yakin tentang perbedaan paket yang dikirim oleh *trusted* host ke *untrusted* host, jangan lupa ciri-cirinya. Pemeriksaan-pemeriksaan dan modifikasi-modifikasi yang dikerjakan di dalam *isg_forward* untuk modifikasi TCP, *isg_output* untuk modifikasi IP terhadap *TCP segment*, dan *ip_output* modifikasi IP terhadap paket-paket non-TCP.

3.2.1. IP scrubbing

Ambiguitas IP-level muncul terutama di *IP header flag* dan bagian *reassembly algorithm*. Modifikasi flag tidak membutuhkan status tetapi membutuhkan adjustment *header checksum*. Pada proses pembentukan kembali (Reassembly) membutuhkan bagian-bagian untuk disimpan di *scrubber*. Sesaat setelah *IP datagram* yang lengkap terbentuk, ada kemungkinan membutuhkan untuk dilakukan pemecahan (re-fragmented) pada jalan keluar (output) interface.

Fingerprint scrubber menggunakan code seperti pada gambar 3 untuk jenis pelayanan normalisasi IP. Ini terjadi untuk semua ICMP, IGMP, UDP, TCP, dan paket-paket lain untuk protocol yang dibuat pada puncak IP. Kombinasi yang tidak biasa dan umumnya tidak dipakai dari bit TOS dibuang. Untuk kasus dimana bit-bit tersebut dibutuhkan untuk digunakan (misal : eksperimen modifikasi IP) maka fungsi ini juga terbuang. Sebagian terbesar implementasi TCP/IP telah dicoba untuk mengabaikan fragment bit cadangan dan me-reset-nya ke 0 jika posisi awalnya set (1) tetapi kami menginginkan keamanan maka kami menutupinya dari luar. *Don't fragment bit* di-reset jika MTU jaringan berikutnya cukup besar untuk paket. Proses pemeriksaan ini tidak perlihatkan pada gambar.

Gambar 3: Code fragment untuk normalize IP
header flag

```
  /*
   * Normalize IP type-of-service flags
   */
  switch (ip->ip_tos)
  {
      case IPTOS_LOWDELAY:
      case IPTOS_THROUGHPUT:
      case IPTOS_RELIABILITY:
      case IPTOS_MINCOST:
      case
IPTOS_LOWDELAY|IPTOS_THROUGHPUT:
          break;
      default:
          ip->ip_tos = 0;
  }

  /*
   * Mask out reserved fragment flag.
   * The MTU of the next downstream link
   * is large enough for the packet so
   * clear the don't fragment flag.
   */
  ip->ip_off &= ~(IP_RF|IP_DF);
```

Modifikasi terhadap don't fragment dapat menghentikan MTU dalam menemukan jalur *scrubber*. Satu hal yang menjadi alasan bahwa kami harus meletakkan *fingerprint scrubber* di suatu tempat untuk menyembunyikan informasi tentang sistem di baliknya. Ini termasuk informasi topologi dan bandwidth. Namun demikian sebuah modifikasi adalah sesuatu yang masih diperdebatkan. Kami membiarkan keputusan untuk

membuang atau tidak *don't fragment bit* kepada pemakai dengan memberikan option untuk turn off.

Fragment reassembly code adalah sebuah modifikasi versi *slightly* (ringkas) dari implementasi standar FreeBSD 2.2.8 kernel. Ini untuk menjaga bagian-bagian dari satu set dari penggandaan daftar. Ini adalah perhitungan pertama sebuah pengacakan untuk menentukan daftar yang mana yang menjadi bagian dari fragment. Sebuah pelacakan linier dikerjakan melalui daftar ini untuk menemukan *IP datagram* yang menjadi tujuan fragment dan tempatnya dalam datagram. Data yang lama dalam antrean fragment selalu dipilih lebih dahulu daripada data yang baru.

3.2.2 ICMP scrubbing

Pada bagian ini kami akan menjelaskan modifikasi yang dibuat pada *fingerprint scrubber* ke ICMP message. Kami hanya mengembalikan pesan modifikasi ICMP dari sisi *trusted* kembali ke sisi *untrusted* karena *fingerprinting* bertumpu pada respon ICMP bukan meminta. Khususnya kami memodifikasi pesan error ICMP dan batas laju semua pesan keluar ICMP.

Pesan error ICMP dimaksud adalah termasuk IP header yang terakhir ditambah 8 byte dari data yang menyebabkan error. Sesuai dengan RFC 1812 sebanyak mungkin byte yang mungkin sampai panjang total paket ICMP adalah 576 byte masih diijinkan. Akan tetapi *Nmap* mengambil keuntungan dari kenyataan bahwa beberapa operating sistem memiliki batas jumlah data yang berbeda. Untuk menangkal hal ini kami paksa semua pesan error yang datang dari sisi *trusted* untuk memiliki beban data hanya 8 byte dengan memampatkan beban data yang lebih besar. Alternative lainnya adalah kita dapat melihat ke dalam pesan error ICMP untuk menentukan apakah ada pemakaian *IP tunneling*. Jika ya, maka kita membiarkan jumlah datanya lebih dari 8 byte.

3.2.3 TCP scrubbing

TCP protocol *scrubber* didasarkan pada *fingerprint scrubber* dalam mengubah aliran TCP ke dalam aliran yang tidak mendua dengan menjaga sejumlah kecil dari status tiap sambungan. Protocol *scrubber* menjaga arah aliran dari sambungan TCP

menggunakan sebuah TCP state diagram yang sederhana. Pada dasarnya ini menjaga arah aliran dari sambungan terbuka dengan mengikuti standar TCP *three-way handshake* (3WHS). Hal ini memungkinkan *fingerprint scrubber* untuk menghadang scan terhadap TCP yang tidak dimulai dengan 3WHS. Kenyataannya langkah pertama dalam *fingerprinting* sebuah sistem adalah menjalankan port scanning untuk menentukan port-port yang terbuka dan tertutup. Penyelidikan rahasia, berarti sulit untuk dideteksi, akan tetapi teknik untuk port scanning tidak menunjukan 3WHS karena itu akan dihadang. Hanya scan yang menggunakan 3WHS yang akan dibiarkan lewat.

Sejumlah besar informasi dapat dikumpulkan dari pilihan-pilihan TCP. Kami tidak ingin menghadang pilihan-pilihan tertentu karena option-option tersebut membantu unjuk kerja dari TCP (misal : SACK) sehingga akibatnya tidak dapat berkembang. Karena itu kami batasi modifikasi ini untuk dapat meminta kembali option-option di dalam TCP header. Kami hanya menyediakan permintaan tertentu dari TCP option yang kita ketahui. Option-option yang tidak dikenal dimasukkan didalamnya setelah semua option yang dikenal. Penanganan option-option yang tidak dikenal dan permintaannya dapat diatur oleh pemakai.

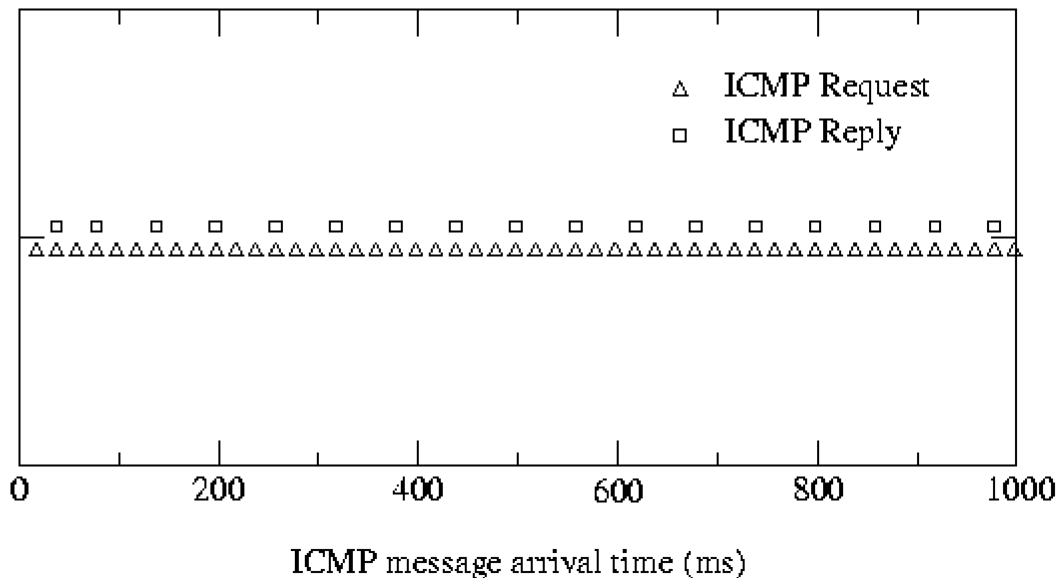
Kami juga menangkal penyusupan-penyusupan pada perkiraan jumlah TCP sequence dengan memodifikasi jumlah sequence normal dari sambungan-sambungan TCP yang baru. *Fingerprint scrubber* menyimpan jumlah random saat sebuah sambungan dikenali. Masing-masing TCP segment untuk perjalanan sambungan dari *trusted* interface ke *untrusted* interface akan memiliki jumlah yang meningkat pada nilai ini. Setiap segment memiliki perjalanan sambungan dalam arah yang berlawanan dengan jumlah penurunannya pada nilai ini.

3.2.4 Timing attacks

Fingerprinting scan yang ada dibuatkan *fingerprint scrubber*-nya untuk menghadang semua yang bersifat statis, model penyelidikan query-response. Sebuah host dengan hati-hati membuat pertanyaan-pertanyaan kemudian mengirimnya ke sebuah host dan menganalisa tanggapan (respons) atau kelemahan dari tanggapan. Bentuk lain yang mungkin dari scan adalah satu, yaitu mengacu pada tanggapan waktu (timing respons).

Sebagai contoh, scanning tehadap host dapat membuka sambungan TCP kemudian melakukan simulasi paket hilang dan melihat bagaimana sebuah host lain melakukan perbaikan (recovery).

Gambar 4: *ICMP rate limiting* dari pantulan ICMP yang ditangkap dengan `tcpdump`.



Cukup sulit dalam membuat sebuah metode generic untuk menangkal timing-related scan khususnya scan yang tidak dikenal. Satu pendekatan yang mungkin dapat ditambahkan sedikit jumlah random dari delay terhadap perjalanan keluar paket-paket menuju *untrusted* interface, *scrubber* dapat mengirim paket-paket di luar order. Akan tetapi pendekatan ini dapat membuat meningkatnya jumlah antrian delay dan mungkin menurunkan unjuk kerjanya dan cara ini tidak menjamin mampu menghadang scan. Sebagai contoh dengan menambah sedikit delay secara random, ini relatif mudah untuk diketahui TCP stack menggunakan TCP Tahoe atau TCP Reno berdasarkan pada simulasi penghilangan paket. Karena sebuah paket akan dikirim kembali setelah sebuah RTO memiliki delay yang lebih besar daripada satu pengiriman kembali yang menggunakan *fast retransmit*.

Kami terapkan proteksi untuk melawan satu timing-related scan yang mungkin. Beberapa operating sistem menerapkan pembatasan laju ICMP tetapi mereka menerapkan pada laju yang berbeda dan beberapa diantara yang lain tidak menerapkan pembatasan laju. Kami menambahkan sebuah parameter untuk pembatasan laju ICMP terhadap *fingerprinting scrubber* untuk menangkal scan. *Scrubber* merekam sebuah waktu khusus saat sebuah pesan ICMP melewatinya dari *trusted* interface ke *untrusted* interface. Waktu khusus ini disimpan dalam sebuah tabel referensi hash kecil dengan kombinasi dari IP address sumber dan tujuan. Sebelum sebuah pesan ICMP dikirim ke luar menuju *untrusted* interface, ini diperiksa kecocokannya dengan waktu khusus yang dirahasiakan. Paket ini akan di-drop jika jumlah waktunya tidak sesuai dengan pesan ICMP yang pernah dikirim ke tujuan tersebut dari sumber yang sama (dirahasiakan).

Gambar 4 memperlihatkan *fingerprint scrubber* yang membatasi laju ICMP untuk sinyal permintaan (request) dan sinyal jawaban (reply). Misalnya sebuah *untrusted* host sedang mengirim sinyal permintaan ICMP setiap 20 milli detik (milli second) menggunakan flag *ping* (terus-menerus). *Scrubber* membiarkan request lewat tanpa melakukan modifikasi selama kita tidak mencoba menyembunyikan identitas *untrusted* host dari *trusted* host. Saat sinyal ICMP echo reply datang maka *fingerprinting scrubber* harus memastikan bahwa hanya reply yang datang pada selisih waktu 50 ms adalah sinyal yang yang harus diteruskan untuk dikirim. Jika sinyal request yang datang memiliki jarak 20 ms , maka setiap 3 sinyal request akan melewati *scrubber* satu kali. Sehingga *untrusted* host akan menerima satu kali reply setiap 60 ms.

Kami memilih selisih 50 ms untuk adalah untuk memudahkan karena flag ping (*ping-f*) men-generate sebuah aliran ICMP echo request dengan jarak 20 ms dan kami menginginkan adanya kemungkinan pembatasan laju (rate limiting). Nilai pasti dari sebuah sistem produksi akan ditentukan oleh administrator atau berdasarkan pada batas aliran serangan ICMP sebelumnya. Tujuannya adalah untuk menyamakan laju traffic ICMP yang berjalan dari *untrusted* interface ke *trusted* interface karena laju operating sistem memaksa pesan-pesan ICMP untuk memiliki laju yang berbeda. Metode yang lain untuk mengacaukan adalah fingerprinter akan menambahkan sebuah delay random yang kecil untuk setiap pesan ICMP. Sebuah pendekatan akan dibutuhkan untuk menjaga kekurangan status. Kita dapat menambahkan delay kepada reply ICMP, sebagai

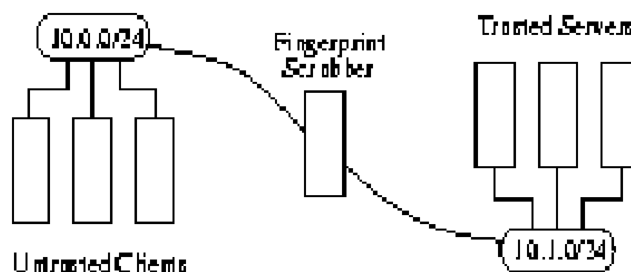
kebalikan dari TCP segment, karena mereka tidak menginginkan unjuk kerja jaringan terganggu.

4. Evaluasi terhadap Fingerprint Scrubber

Bagian ini membahas hasil-hasil dari serangkaian eksperimen yang kita buat untuk menentukan validitas, throughput dan ukuran *fingerprinting* scraubber. Mereka memperlihatkan bahwa blok-blok implementasi kita sekarang berusaha mengenali scan fingerprint. Percobaan-percobaan ini dilakukan dengan menggunakan satu set kernel yang option-option *fingerprint scrubber*-nya berbeda sehingga dapat dipakai sebagai perbandingan.

Scrubber dan semua host, masing-masing menggunakan CPU Pentium III 500 MHz dengan memory utama 256 megabyte. Setiap host menggunakan satu 3Com 3c905B Fast Etherlink XL 10/100BaseTX Ethernet card (`xl` device driver). Gateway memiliki dua Intel EtherExpress Pro 10/100B Ethernet cards (`fxp` device driver). Konfigurasi jaringan dapat bekerja untuk traffic dari 10.0.0/24 sampai dengan 10.1.0/24 melalui mesin gateway. Gambar 5 memperlihatkan bagaimana 3 buah mesin dihubungkan sebaik mungkin sebagai *trusted* dan *untrusted* domain.

Gambar 5: Set up eksperimen untuk pengukuran unjuk kerja dari *fingerprint scrubber*.



4.1 Menangkal Fingerprint Scan

Untuk memeriksa bahwa *fingerprint scrubber* benar-benar mampu menangkal scan yang dikenalinya, kami menempatkan diantara gateway dan satu set mesin yang berjalan dengan menggunakan sistem operasi yang berbeda. Sistem operasi yang dijalankan untuk di-scan dibawah pengawasan adalah FreeBSD 2.2.8, Solaris 2.7 x86, Windows NT 4.0 SP 3 dan Linux 2.2.12. Kami juga menjalankan scan terhadap beberapa website yang populer, campus workstation, server dan printer.

Nmap mampu menentukan semua sistem operasi dari host tanpa meletakkan *fingerprint scrubber* diantaranya. Akan tetapi terbukti benar-benar *Nmap* tidak mampu mendeteksi jika kita menghadang tamu dengan menggunakan *fingerprint scrubber*. Kenyataannya *Nmap* tidak mampu membedakan sebagian besar host yang ada. Sebagai contoh, jika tanpa *scrubber Nmap* mampu dengan akurat mengidentifikasi sistem FreeBSD 2.2.8. Dengan meletakkan *scrubber*, *Nmap* mendatangi 14 operating sistem yang berbeda, semuanya salah deteksi. Gambar 6 memperlihatkan sebuah kesimpulan yang dibuat dari *Nmap* saat menyerang FreeBSD sebelum dan setelah meletakkan FreeBSD.

Gambar 6: Sistem Operasi *guess*

(a) sebelum fingerprint scrubbing
(b) Setelah fingerprint scrubbing untuk `Nmap` scan menyerang mesin yang menjalankan FreeBSD 2.2.8.

(a)
```
Remote operating system guess:
   FreeBSD 2.2.1 - 3.2
```

(b)
```
Remote OS guesses:
   AIX 4.0 - 4.1, AIX 4.02.0001.0000,
   AIX 4.1, AIX 4.1.5.0, AIX 4.2,
   AIX 4.3.2.0 on an IBM RS/*,
   Raptor Firewall 6 on Solaris 2.6,
   Solaris 2.5, 2.5.1, Solaris 2.6 - 2.7,
   Solaris 2.6 - 2.7 X86,
   Solaris 2.6 - 2.7 with tcp_strong_iss=0,
   Solaris 2.6 - 2.7 with tcp_strong_iss=2,
   Sun Solaris 8 early acces beta (5.8)
     Beta_Refresh February 2000
```

Dua komponen utama yang membantu melakukan penghadangan *Nmap* adalah kemampuan dalam penggunaan *three-way handshake* untuk TCP dan kemampuan untuk meminta kembali option-option TCP. Sebagian besar prinsip kerja *Nmap* adalah dengan mengirim alat deteksi (probe) tanpa SYN flag sehingga host yang didatangi membuangnya. Sementara sistem operasi sangat bervariasi dalam memanggil kembali option-option TCP. Karena itu *Nmap* menderita kehilangan besar dari informasi yang mungkin didapat.

Kami berharap *tools* ini mampu untuk menghadang scan-scan baru dan potensial. Kami percaya bahwa keterbukaan dari IP header flag normalization dan IP fragmet reassembly membantu mencapai tujuan untuk mampu menembus apa yang kita belum tahu dari *tools* yang melakukan eksploitasi yang sedemikian berbeda.

4.2 Throughput

Kami melakukan eksperimen untuk testing *raw throughput* yang melewati *fingerprint scrubber*. Throughput dihitung dengan menggunakan *netperf benchmark*. Tiga buah mesin tes dihubungkan dengan switch 100 MBps.

Kami mengukur througput dari kedua sisi, yaitu keluaran sisi *trusted* ke sisi *untrusted* dan dari sisi *untrusted* ke sisi *trusted*. Hal ini untuk menghitung asymmetric filtering dari traffic. Tujuan menjalankan eksperimen TCP traffic adalah untuk memperlihatkan pengaruh sebuah tansfer TCP yang besar dan untuk mencoba fragment reassemby code dari UDP. Dengan menggunakan 3 kernel pada mesin gateway bertujuan untuk mencoba perbedaan fungsi dari *fingerprint scrubber*. *IP forwarding kernel* adalah *unmodified FreeBSD kernel*, yang dipakai sebagai baseline untuk perbandingan. Fingerprint scrubbing kernel termasuk *TCP option reordering*, *IP header flag normalization*, *ICMP modification*, dan *TCP sequence number modification* tetapi tidak termasuk *IP fragment reassembly*. *Last kernel* adalah *full fingerprint scrubber* termasuk menyalakan *fragment reassembly code*. Kami juga membandingkan *fingerprint scrubber* pada sebuah *full application-level proxy*. TIS Firewall Toolkit's plug-gw proxy adalah sebuah contoh komponen firewall yang beroperasi pada level user untuk menjalankan *transport-layer proxying*. Saat sebuah sambungan baru TCP dibuat ke proxy, *plug-gw* membuat sebuah sambungan kedua dari proxy ke server. Satu-satunya tugas proxy adalah

membaca dan mengkopi data dari satu sambungan kepada yang lain. Sebuah firewall dengan fasilitas (feature) yang lebih lengkap akan mengolah header dan data yang lebih lengkap, yang diberi tambahan dan status lebih banyak. Karena itu unjuk kerja dari plug-gw mewakili sebuah jumlah minimum yang harus mampu dilakukan oleh firewall pada application-level proxiy. *Original plug-gw code* dimodifikasi sedemikian sehingga tanpa logging dan tanpa resolusi DNS, yang berakibat pada sebagian besar unjuk kerja meningkat. Kernel proxy juga dimodifikasi sehingga sebagian besar dari proses dapat diakomodasi. *Custom user-space proxy* dioptimasi agar kecepatan pada bagian-bagian tertentu bekerja lebih baik (plug-gw proxy membentuk anak-anak cabang untuk setiap sambungan yang masuk). Akan tetapi kerja simultan mengkopi data dan mengatur hubungan switching akan selalu mengurangi implementasi user-space yang akan berakibat menurunkan unjuk kerja dibandingkan dengan *in-kernel approach*.

Tabel 1 memperlihatkan hasil transfer TCP yang besar untuk sebuah hubungan dari *untrusted* host ke *trusted* host. Tabel 2 memperlihatkan hasil untuk sebuah hubungan dari *trusted* host ke *untrusted* host. Hasil pertama adalah memperlihatkan bahwa pada kedua arah throughput-nya sama. Kedua, yang merupakan hal lebih penting bahwa saat semua fungsi *fingerprinting scrubber* dijalankan maka dapat dilihat throughput hampir pasti sama dengan plain IP yang dikirimkan. Bandwidth dari link adalah jelas faktor yang critical untuk semua eksperimen throughput, karena itu kami akan melakukan eksperimen pada jaringan yang lebih cepat pada waktu yang akan datang.

Table 1. Throughput untuk single *untrusted* host ke *trusted* host menggunakan TCP (Mbps, +-2.5% at 99% CI)

IP Forwarding	87.06
Fingerprint Scrubbing	86.86
Fingerprint Scrub. + Frag. Reas.	87.00
Application-level Transport Proxy	86.53

Table 2. Throughput untuk single *trusted* host ke *untrusted* host menggunakan TCP (Mbps, +-2.5% at 99% CI).

IP Forwarding	87.06
Fingerprint Scrubbing	86.79
Fingerprint Scrub. + Frag. Reas.	86.84
Application-level Transport Proxy	86.53

Kami menjalankan eksperimen UDP dengan menggunakan *IP forwarding kernel* dan *fingerprint scrubbing kernel* dengan *IP fragment reassembly*. Selanjutnya kita ukur kedua sisi dari *untrusted* ke *trusted* dan sebaliknya. Untuk mengukur efek dari fragmentasi dijalankan tes dengan ukuran yang bervariasi pada MTU Ethernet link ke atas. Perhatikan bahwa data UDP maksimum yang dapat dimuat adalah 1472 byte yang dapat ditransmisikan jika UDP dan IP header ditambahkan sepanjang 28 byte sehingga akan didapat 1500 byte MTU dari link. Tes byte sepanjang 2048 byte disambung ke 2 fragment dan tes byte sepanjang 8192 byte dihubungkan ke 5 fragment.

Tabel 3 memperlihatkan hasil transfer UDP untuk sebuah sambungan *untrusted* host ke *trusted* host. Tabel 4 memperlihatkan hasil sebuah hubungan *trusted* host ke *untrusted* host. Sekali lagi memperlihatkan untuk hubungan kedua arah akan menghasilkan throughput yang sama. Dapat dilihat juga bahwa throughput dari *fingerprint scrubber* dengan IP fragment reassembly hampir pasti sama dengan IP plain yang dikirimkan. Hal ini terbukti pada kasus 8192 byte tes dimana bagian-bagiannya harus di-reassemly pada gateway dan kemudian harus di-refragment sebelum dikirimkan.

Table 3**:** Throughput untuk single *untrusted* host ke *trusted* host menggunakan UDP (Mbps, +-2.5% at 99% CI).

	64 bytes	1472 bytes	2048 bytes	8192 bytes
IP Forwarding	14.39	89.39	92.76	90.11
Fingerprint Scrubbing + Frag. Reas.	14.48	89.35	92.76	90.11

Table 4. Throughput for a single *trusted* host to an *untrusted* host using UDP (Mbps, +-2.5% at 99% CI).

	64 bytes	1472 bytes	2048 bytes	8192 bytes
IP Forwarding	14.39	89.39	92.76	90.11
Fingerprint Scrubbing + Frag. Reas.	14.40	89.37	92.76	90.12

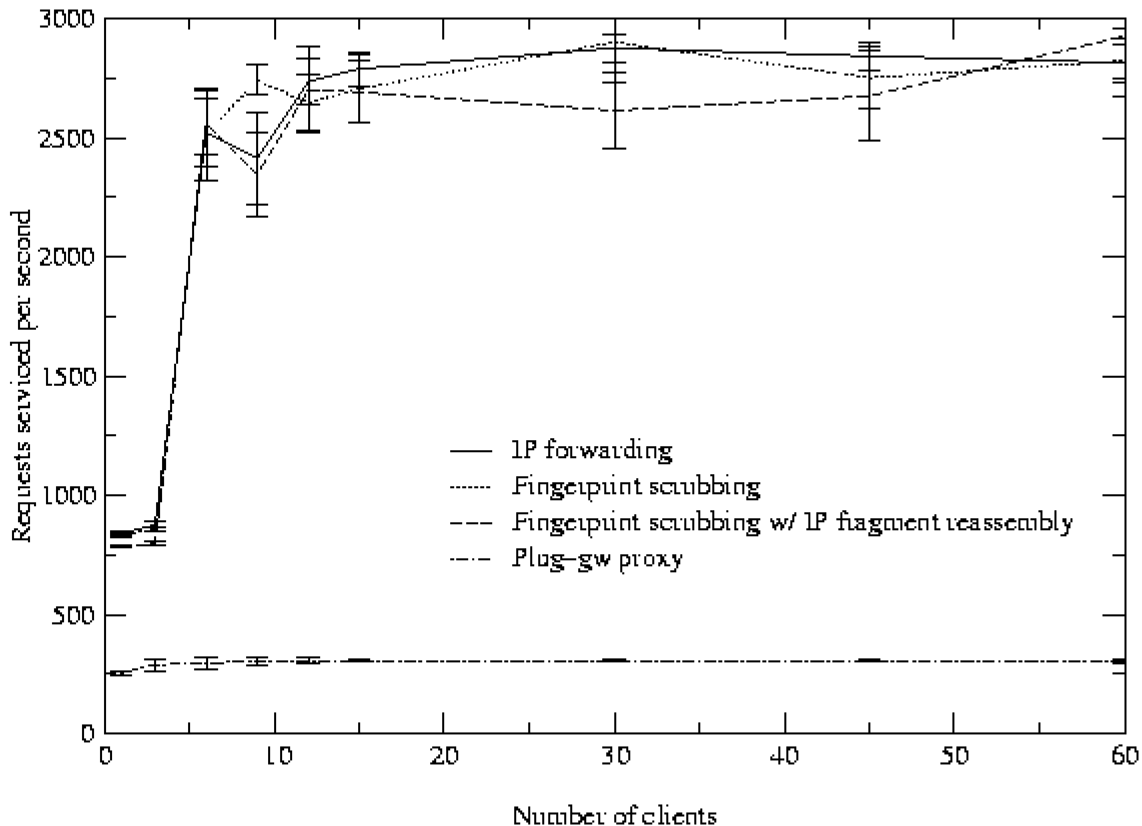
4.3. Parameter yang dapat diukur

Kami juga menjalankan eksperimen untuk mengukur keterukuran (scalability) dari *fingerprint scrubber*. Berapa sambungan TCP secara concurrent yang dapat di-support oleh *fingerprint scrubber* gateway? Dengan mengatur 3 mesin sebagai web server berfungsi sebagai sink untuk request HTTP. Pada 3 mesin lain dijalankan penambahan jumlah client secara berulang meminta file sebanyak 1 KB dari web server. Pilihan angka 1 KB untuk menjaga CPU server dari faktor pembatas. Sebagai gantinya bandwidth dari link akan mengalami penyumbatan (bottleneck). Client-client tersebut dihubungkan dengan hub 100 Mbps dan server dihubungkan dengan switch 100 Mbps. Jumlah sambungan tiap detik dibuat melalui *fingerprint scrubber* yang diukur pada hub.

Gambar 7 memperlihatkan jumlah sambungan yang tertahan tiap detik diukur untuk plain IP forwarding, TCP/IP fingerprint scrubbing, fingerprint scrubbing dengan IP fragment reassembly dan the plug-gw application-level proxy. Grafik bar dari error mewakili standar deviasi tiap detiknya. Hasil-hasil eksperimen adalah ukuran perbandingan *fingerprint scrubber* terhadap unmodified IP forwarder dan memperlihatkan seberapa besar kelebihannya dibandingkan dengan transport proxy. *Fingerprint scrubber* mencapai laju sekitar 2700 sambungan tiap detik, yang diperkirakan cukup untuk semua LAN. Sebagai perbandingan plug-gw proxy hanya mencapai laju sekitar 300 sambungan tiap detik, yang memperlihatkan nilai yang lebih jelek daripada *scrubber*. Unjuk kerja yang tidak terukur dari application-level proxy dapat dijelaskan dengan jumlah interrupt, data yang dikopi dan switch yang berhubungan

ditentukan oleh proses pada level user. Untuk setiap sambungan TCP, proxy harus menjaga arah aliran dari 2 buah mesin status TCP secara lengkap dan copy data dari kernel kemudian kembali lagi. Sistem yang menggunakan plug-gw adalah membatasi CPU untuk semua pekerjaan akan tetapi hanya untuk beberapa client yang berjalan concurrent.

Gambar 7: Sambungan tiap detik melalui *gateway*.



Kecepatan penuh lintasan yang dapat dicapai *fingerprint scrubber* untuk link bandwidth yang lebih besar membutuhkan hardware yang teruji. Sebuah platform yang sama seperti *Intel's Internet Exchange Architecture* dapat membantu. Sebagian kecil dari tabel TCP status digunakan *fingerprint scrubber* dengan mudah untuk sistem tersebut.

5. Perbandingan

Teknologi firewall sekarang ini sesuai dengan *fingerprint scrubber*. Firewall berada pada batas sebuah jaringan untuk menyediakan kontrol akses. Keduanya membutuhkan paket-paket untuk berjalan melewatinya sampai dengan tujuan akhirnya dan dapat menolak beberapa jenis paket tertentu. Firewall yang lebih tua seperti TIS Firewall Toolkit, menggunakan application-level proxy dan tidak terukur dengan baik karena mesti menjaga dua sambungan TCP agar terbuka tiap session. Demikian sebuah firewall akan menghadang scan TCP *fingerprinting* karena ini mengisolasi tingkah laku dari implementasi TCP pada sisi yang dilindungi oleh proses copy data. Akan tetapi kami perlihatkan bahwa unjuk kerja firewall memiliki nilai yang lebih buruk dari *fingerprint scrubber*. Juga application-level proxy tidak memperdulikan ambiguitas IP-level. Firewall yang baru, seperti gauntlet melakukan identifikasi aliran yang legal dengan menguji bagian-bagian dari kepala paket dan daya tampung data atau menggunakan metode authentification yang lebih canggih. Firewall mengantarkan paket-paket sebuah lintasan yang cepat, dimana aliran dirancang untuk meningkatkan throughput dan scalability (keterukuran parameter). Firewall tidak menyediakan pengamanan terhadap serangan scan *fingerprinting* secara terus menerus saat sebuah sambungan dijalankan. Sebaliknya *fingerprint scrubber* membuang scan sepanjang aliran dengan lebih terukur dengan menjaga jumlah minimal dari status tiap sambungan.

Bermacam *tools* yang dapat digunakan untuk mengamankan sebuah mesin dari serangan *Nmap fingerprinting* terhadap sistem operasi. *TCP/IP traffic logger iplog* dapat mendeteksi sebuah *Nmap* fingerprint scan dan mengirim paket yang dirancang untuk membingungkan hasil scan. *Tools* yang lain dan modifikasi sistem operasi mempermudah penggunaan status agar menyatu dalam implementasi TCP untuk menghadang tipe scan tertentu. Akan tetapi *tools* tersebut tidak ada yang dapat digunakan untuk melindungi jaringan sekitarnya dari sistem yang heterogen. Sebagai tambahan metode-metode ini tidak akan bekerja untuk jaringan yang tidak di bawah satu kontrol administratif, tidak seperti *fingerprint scrubber*.

Vern Paxson mempersembahkan sebuah *tools* untuk menganalisa tingkah laku implementasi TCP yang disebut *tepanaly*. Saat offline *tcpdump* melacak untuk mencoba membedakan jika sebuah pola traffic tertentu tetap berjalan pada suatu implementasi.

Pada cara ini, *tools* melakukan sebuah pelacakan dari TCP *fingerprinting*. Akan tetapi tepanaly membiarkan beberapa ketidakpastian yang membuatnya tidak kelihatan sebagai *fingerprinting tools*. Hal ini juga menjaga keterbukaan tentang pengetahuan beberapa implementasi TCP/IP. Sebaliknya pada *fingerprint scrubber* tidak memiliki pengetahuan tentang implementasi yang lain. Kontribusi utama tepanaly membuatnya tidak sebagai *fingerprinting* tetapi sebagai analisa untk koreksi implementasi TCP dan membantu menentukan apakah sebuah implementasi keliru.

Malan memaparkan ide bahwa tidak hanya perlu transport-level scrubbing tetapi juga pada application-level scrubbing. Dengan jelas bahwa pekerjaan ini membutuhkan spesialisasi, Fokus utama adalah pada HTTP traffic untuk melindungi web server. Dasar utama pemikirannya adalah melindungi komponen infrastruktur, misalnya : router, dengan melakukan scrubbing terhadap RIP, OSPF dan BGP.

6. Pengembangan

Seperti dijelaskan di bagian 4.2 dan 4.3, orde pertama yang menjadi faktor pembatas unjuk kerja *fingerprint scrubber* adalah ketersediaan link bandwidth. Maka dapat dirancang pengujian *scrubber* malalui sambungan ethernet gigabit. Untuk mendukung berlipatnya bandwidth sampai 10 kali, dapat dilakukan dengan melihat pada berkurangnya jumlah data yang di-copy dengan menggunakan incremental checksum saat melakukan modifikasi bit header dan menggunakan sebuah fragment reassembly algorithm yang lebih cepat.

Karena hubungan yang sangat dekat antara firewall dan *fingerprint scrubber*, maka memungkinkan untuk melakukan kombinasi kedua teknologi. *Scrubber* dapat digunakan sebagai substrat dan menambahkan beberapa layanan (feature), seperti authentication, membutuhkan fungsi firewall secara penuh. Kami yakin bahwa sebuah sistem akan terdiri dari kombinasi kelebihan-kelebihan keamanan dari firewall modern yang memiliki karakteristik unjuk kerja dan kelebihan-kelebihan *fingerprint scrubber*.

Kita juga harus menguji bagaimana IP security mempengaruhi YCP/IP stack *fingerprinting* dan penemuan sistem operasi. Jika sebuah host menerapkan IP security yang tidak mengijinkan host-host yang tidak dikenal untuk menjalin hubungan sehingga

host-host tersebut tidak akan mampu sistem operasi host karena semua paket akan dihadang. Jika sebuah host membiarkan host yang tidak dikenali untuk berhubungan dalam mode tunnel maka *fingerprint scrubber* tidak akan effektif. *Scrubber* tidak akan sanggup menguji dan memodifikasi pengacakan dan penyandian TCP & IP header. Membiarkan setiap host untuk membuat sebuah sambungan yang aman ke sebuah IPsec host bukan merupakan prosedur yang standar akan tetapi ini adalah sebuah public server. Porsi yang lain dari IPsec yang dapat di-eksploitasi adalah kunci-kunci exchange protokol, seperti ISAKMP/IKE. Jika sistem yang berbeda memiliki perbedaan-perbedaan kecil pada implementasinya, sebuah scanner mungkin untuk melihat operating sistem host.

Pemikiran lain yang dapat dicoba adalah membuat *fingerprint scrubber* spoof sebuah fingerprint sistem operasi untuk membongkar kerahasiaanya. Sebagai contoh, ini mungkin menarik untuk memiliki semua komputer pada jaringan kita dijalankan pada *secure operating system OpenBSD*. Hal ini sulit untuk dikerjakan daripada cara sederhana membuang ambiguitas karena kita harus mengenal cukup luas untuk membuat sebuah tipuan yang meyakinkan.

Sebagai komponen infrastruktur jaringan harus mampu meningkatkan kecepatan, *tools* seperti *fingerprint scrubber* harus disesuaikan dengan kebutuhannya. Untuk mencoba mencapai line-speed dapat dicoba menerapkan komponen-komponen utama dari *fingerprint scrubber* di hardware. Sebuah contoh akan membangun mesin status TCP minimal dengan menggunakan satu platform seperti Intel's Internet Exchange Architecture (IXA).

7. Kesimpulan

Kami telah menjelaskan sebuah *tools* baru yang disebut *fingerprint scrubber* yang berguna untuk membuang ciri-ciri identifikasi sistem operasi dari host tujuan. *Scrubber* akan menghadang *fingerprinting* scan yang dikenalinya, sebagai pembanding unjuk kerjanya digunakan untuk mengirim plain IP lewat gateway dan yang lebih penting lebih terukur daripada full transport-layer firewall.

Fingerprint scrubber berhasil dengan lengkap menghadang scan dengan membuang beberapa ciri dari TCP dan IP layer. Karena desain umum akan dengan efektif menghadapi evolusi dan pengembangan fingerprint scanner. Hal ini akan melindungi jaringan sekitarnya menghadapi scan yang dirancang dengan profile sistem kekebalan. Scan selalu merupakan langkah yang pertama dalam melakukan serangan terhadap kontrol komputer yang exploitable. Saat terjadi penggabungan dari beberapa sistem maka dibutuhkan distribusi terhadap penangkal serangan. Dengan melakukan penghadangan sebagai langkah pertama maka *fingerprint scrubber* akan meningkatkan keamanan dari jaringan yang heterogen.

8. Daftar Pustaka

1. G. Robert Malan, David Watson, Farnam Jahanian, and Paul Howell. *Transport and Application Protocol Scrubbing*. In Proceedings of the IEEE INFOCOM 2000 Conference, Tel Aviv, Israel, March 2000.
2. David Maltz and Pravin Bhagwat. *TCP Splicing for Application Layer Proxy Performance*. Technical Report RC 21139, IBM Research Division, March 1998.
3. D. Maughan, M. Schertler, M. Schneider, and J. Turner. *Internet Security Association and Key Mangement Protocol (ISAKMP)*. RFC 2408, November 1998.
4. Deteksi OS melalui Fingerprint TCP/IP oleh Fyodor <fyodor@insecure.org> (http://www.insecure.org/).
5. Thomas H. Ptacek and Timothy N. Newsham. *Insertion, evasion, and denial of service: Eluding network intrusion detection*. Originally Secure Networks, Inc., now available as a white paper at the Network Associates.
6. Oliver Spatscheck, Jorgen S. Hansen, John H. Hartman, and Larry L. Peterson. *Optimizing TCP Forwarder Performance*. Technical Report TR98-01, Dept. of Computer Science, University of Arizona, February 1998.