

**TUGAS AKHIR MATA KULIAH
KEAMANAN SISTEM LANJUT (EC.7010)**

Dosen
Dr. Ir. Budi Rahardjo

***SESSION HIJACKING*
DAN CARA PENCEGAHANNYA**

Oleh

Hendri Murti Susanto
NIM. 23203109



**PROGRAM MAGISTER TEKNIK ELEKTRO
BIDANG KHUSUS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI BANDUNG
2004**

ABSTRAKSI

Istilah sesi pembajakan (*session hijacking*) umumnya digunakan untuk menggambarkan proses sebuah koneksi TCP yang diambil alih oleh sebuah rangkaian serangan yang sudah dapat diprediksi sebelumnya. Pada serangan seperti itu, penyerang memperoleh kendali melalui koneksi TCP yang sudah ada. Bila diterapkan pada keamanan aplikasi web, *session hijacking* mengacu pada pengambilalihan sebuah *session* aplikasi web.

Aksi ini melakukan pengambilan kendali *session* milik user lain setelah sebelumnya “pembajak” berhasil memperoleh autentifikasi ID *session*. *Session hijacking* menggunakan metode *captured*, *brute forced* atau *reserve engineered* guna memperoleh ID *session* yang untuk selanjutnya memegang kendali atas *session* yang dimiliki oleh user lain tersebut selama *session* berlangsung.

Pokok masalah serangan ini semata-mata pada cara desain dan pengembangan aplikasi. Kelalaian dalam mendesain atau dalam mengimplementasikan mekanisme *session tracking* pada aplikasi akan menghasilkan kelemahan yang dapat berakibat fatal. Tak satupun *patch* sistem operasi, firewall atau konfigurasi web server dapat mencegah serangan *session hijacking*. Tiap pengembang web harus mengerjakan secara cermat desain dan implementasi *session* dan *state tracking*.

Ada beberapa aturan yang dapat digunakan untuk menerapkan *session* dan *state tracking* secara benar. Aturan-aturan tersebut sama sekali tidak melengkapi atau mengikat suatu aplikasi. Tetapi sebaliknya, aturan-aturan ini dapat menjadi petunjuk yang berguna untuk mendesain sebuah *session* dan mekanisme *state tracking*.

Kata kunci : *session hijacking*, *session tracking*, *state tracking*.

KATA PENGANTAR

Puji syukur penulis panjatkan ke hadirat Allah Tuhan Yang Maha Esa, karena hanya dengan rahmatNya penulis dapat menyelesaikan tugas mata kuliah Keamanan Sistem Lanjut (EC 7010), yang berjudul *SESSION HIJACKING* dan CARA PENCEGAHANNYA.

Rasa terimakasih yang tak terhingga, penulis sampaikan kepada :

1. Bapak Dr. Ir. Budi Rahardjo, selaku dosen mata kuliah Keamanan Sistem Lanjut (EC 7010) yang telah membimbing dan mengarahkan penulis sehingga tugas ini dapat diselesaikan,
2. Semua pihak yang telah membantu dalam penulisan tugas mata kuliah ini.

Semoga Allah SWT senantiasa memberikan balasan dari semua kebaikan dengan pembalasan yang sebaik-baiknya, karena sesungguhnya Allah SWT adalah sebaik-baik pemberi pembalasan.

Tidak ada gading yang tak retak, penulis menyadari bahwa masih banyak kekurangan dan kelemahan dalam laporan ini. Oleh karena itu saran, kritik dan koreksi sangat diharapkan. Semoga buku ini bermanfaat bagi kita semua dan merupakan amalan baik bagi penulis. Amien.

Bandung, Desember 2004

Penulis

DAFTAR ISI

ABSTRAKSI	i
KATA PENGANTAR	ii
DAFTAR ISI.....	iii
DAFTAR GAMBAR	iv
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang	1
1.2 Tujuan	1
BAB II PENGERTIAN <i>SESSION HIJACKING</i>	2
2.1 Pengertian Cookies.....	2
2.2 <i>Session Hijacking</i>	4
BAB III AKSI <i>SESSION HIJACKING</i>	5
3.1 Identitas yan Dicuri dan Kencan yang Batal.....	5
3.2 Tanggal 5 Maret Pukul 7:00 – di Rumah Alice	5
3.3 Pukul 8:30 – di Kantor Alice	6
3.4 Pukul 10:00 – di Kantor Bob	7
3.5 Pukul 11:00 – di Kantor Bob	9
3.6 Pukul 12:30 – di Kantor Alice	11
3.7 Pukul 21:30 – di Restoran Italia Bertolini	12
3.8 Postmortem Serangan <i>Session Hijacking</i>	13
3.9 Diagram Aplikasi State	14
3.10 HTTP dan <i>Session Tracking</i>	15
3.11 Aplikasi Stateless vs Aplikasi Stateful.....	18
3.12 Cookie dan Field Tersembunyi	20
3.12.1 Cookie	20
3.12.2 Field Tersebunyi	21
3.13 Mengimplementasikan <i>Session</i> dan <i>State Tracking</i>	21
3.13.1 <i>Session Identifier</i> Harus Unik	21
3.13.2 <i>Session Identifier</i> Tidak Boleh “Mudah Diterka”	22
3.13.3 <i>Session Identifier</i> Harus Independen.....	22
3.13.4 <i>Session Identifier</i> Harus Dapat Dipetakan dengan Koneksi Client Side	23
BAB IV KESIMPULAN	24
DAFTAR PUSTAKA	25

DAFTAR GAMBAR

Gambar 1. <i>Kotak pop-up Cookie Pal</i>	7
Gambar 2. <i>Diagram state dari eWebMail</i>	14
Gambar 3. <i>Session Alice pada eWebMail</i>	16
Gambar 5. <i>Interaksi Bob dengan eWebMail</i>	19

BAB I

PENDAHULUAN

1.1 Latar Belakang

Peniruan identitas, atau kemampuan seseorang untuk mengambil identitas orang lain dan dengan menggunakan identitas itu dia dapat masuk ke suatu ruang lingkup masyarakat, dialami oleh kurang lebih 750.000 orang setiap tahunnya. Hal tersebut, termasuk juga kenyataan bahwa peniruan identitas merupakan keluhan nomor satu pada Komisi Dagang Federal Amerika Serikat, menimbulkan pertanyaan tidakkah mengherankan bila peniruan *cyber* memiliki resiko tinggi?

Nyatanya, tindakan mengambil identitas seseorang sangat mudah dilakukan lewat internet daripada lewat dunia fisik. Hal itu karena hanya beberapa orang memahami resikonya dan bahkan sedikit pula yang berusaha mencegah resiko-resiko tersebut. Ketika e-commerce melebarkan sayapnya, pentingnya berusaha mendapatkan identitas seseorang secara akurat pada internet menjadi hal yang sangat vital bagi kerahasiaan *online* pelanggan dan pelaku bisnis.

1.2 Tujuan

Tujuan dari penulisan makalah ini adalah untuk memahami bagaimana session hijacking dilakukan serta mengetahui cara pencegahannya.

BAB II

PENGERTIAN *SESSION HIJACKING*

2.1 Pengertian Cookies

Cookies merupakan data file yang ditulis ke dalam hard disk komputer oleh web server yang digunakan untuk mengidentifikasi diri user pada situs tersebut sehingga sewaktu user kembali mengunjungi situs tersebut, situs itu akan dapat mengenalinya. Jadi dapat dikatakan bahwa cookies merupakan semacam *ID card* user saat koneksi pada situs [1]. Tiap-tiap website pada umumnya mengeluarkan / membuat cookies sesuai karakteristiknya. Ada web yang dapat menyapa user tiap kali mengunjungi situs tersebut selayaknya teman lama karena menggunakan cookies.

Jadi secara umum cookies berfungsi untuk.

1. Membantu web site untuk "mengingat" siapa kita dan mengatur *preferences* yang sesuai sehingga apabila user kembali mengunjungi web site tersebut akan langsung dikenali.
2. Menghilangkan kebutuhan untuk *re-register* ulang di web site tersebut saat mengakses lagi tersebut (site tertentu saja), cookies membantu proses *login* user ke dalam web server tersebut.
3. Memungkinkan web site untuk menelusuri pola web *surfing* user dan mengetahui situs favorit yang sering dikunjunginya.

Meskipun sekilas cookies itu seakan banyak gunanya, akan tetapi sampai sekarang masih menjadi bahan perdebatan mengenai keberadaan cookies ini, Karena selain membuat sebuah web site terlihat *user friendly*, cookie juga menghadirkan isu melanggar privasi pengakses web dan masalah keamanan.

Saat user mengunjungi situs yang ada cookiesnya, server akan mencari informasi yang dibuat sebelumnya dan browser membaca informasi di cookies dan menampilkannya. Cookies di simpan di salah satu direktori di dalam hard disk, salah satu cara cepat untuk mencari tahu di mana cookies disimpan yaitu dengan

cara *search / find* 'cookies'. Misalnya user menggunakan Windows 2000 versi Profesional dengan browser Internet Explorer dan *login* ke network dengan *account* "hendri". Maka cookies itu akan disimpan di direktori (*profiles*) yaitu C:\Documents and settings\hendri\cookies. Atau di windows 95/98 disimpan di direktori C:\Windows\cookies. Jika menggunakan Netscape, maka file cookies akan disimpan di cookies.txt.

Cookies dapat dibedakan menjadi 2 jenis yaitu.

1. ***Non persistent (session) cookies***. Suatu cookie yang akan hilang sewaktu user menutup browser dan biasanya digunakan pada '*shopping carts*' di toko belanja *online* untuk menelusuri item-item yang dibeli,
2. ***Persistent cookies***. Diatur oleh situs-situs portal, *banner* / media iklan situs dan lainnya yang ingin tahu ketika user kembali mengunjungi site mereka. (misal dengan cara memberikan opsi "Remember Me" saat login). File file ini tersimpan di hardisk user.

Kedua tipe cookies ini menyimpan informasi mengenai URL atau *domain name* dari situs yang dikunjungi user dan beberapa kode yang mengindikasikan halaman apa saja yang sudah dikunjungi. Cookies dapat berisi informasi pribadi user, seperti nama dan alamat email, Akan tetapi dapat juga user memberikan informasi ke *website* tersebut melalui proses registrasi. Dengan kata lain, cookies tidak akan dapat "mencuri" nama dan alamat email kecuali diberikan oleh user. Namun demikian, ada kode tertentu (*malicious code*) yang dibuat misalnya dengan *ActiveX control*, yang dapat mengambil informasi dari PC tanpa sepengetahuan user.

Cookies umumnya kurang dari 100 bytes sehingga tidak akan mempengaruhi kecepatan browsing. tetapi karena umumnya browser diatur secara default untuk menerima cookies maka user tidak akan tahu bahwa cookies sudah ada di komputer. Cookies dapat berguna terutama pada situs yang memerlukan registrasi, sehingga setiap kali mengunjungi situs tersebut, cookies akan *me-login*-kan user tanpa harus memasukkan *user name* dan *password* lagi.

Untuk keperluan bisnis, seperti situs amazon.com, menggunakan cookies dapat membantu menghubungkan ke data pembelian yang terdahulu ke basis data yang berisi *unique* ID dan historikal pembelian. Sehingga mampu merekomendasikan buku yang sesuai dengan selera user. Ini merupakan hal yang menarik, sehingga pembeli akan dengan senang hati untuk kembali ke situs amazon. Situs-situs lain juga menggunakan cookies untuk mengetahui berapa orang yang mengakses mereka setiap harinya. Sehingga angka yang dihasilkan oleh cookies tersebut menunjukkan seberapa sibuknya situs mereka.

Bagaimana caranya agar dapat mengatasi dan memblokir cookies? Di masing-masing browser baik netscape maupun IE dapat diatur untuk *enable* maupun *disable* cookies. Misalnya IE, dapat diatur pada bagian Internet Options | Security.

2.2 Session Hijacking

Merupakan aksi pengambilan kendali *session* milik user lain setelah sebelumnya “pembajak” berhasil memperoleh autentifikasi ID *session* yang biasanya tersimpan dalam cookies. *Session hijacking* menggunakan metode *captured*, *brute forced* atau *reverse engineered* guna memperoleh ID *session*, yang untuk selanjutnya memegang kendali atas *session* yang dimiliki oleh user lain tersebut selama *session* berlangsung [2].

HTTP merupakan protokol yang *stateless*, sehingga perancang aplikasi mengembangkan suatu cara untuk menelusuri suatu *state* diantara user-user yang koneksi secara *multiple*. Aplikasi menggunakan *session* untuk menyimpan parameter-parameter yang relevan terhadap user. *Session* akan terus ada pada server selama user masih aktif / terkoneksi. *Session* akan otomatis dihapus jika user logout atau melampaui batas waktu koneksi. Karena sifatnya ini, *session* dapat dimanfaatkan oleh seorang *hacker* untuk melakukan *session hijacking*.

BAB III

AKSI SESSION HIJACKING

Guna memberikan gambaran yang jelas mengenai bagaimana *session hijacking* dilakukan, cerita berikut ini [3] menunjukkan bagaimana aksi tersebut berlangsung dengan menggunakan "teknik pembajakan" yang disesuaikan dengan kelemahan sistem aplikasi web.

3.1 Identitas yang Dicuri dan Kencan yang Batal

Alice bertemu dengan Charles dalam ruang *chatting online*. Mereka saling bertukar informasi dan memahami bahwa mereka beralamat tidak berjauhan dan memiliki banyak persamaan. Charles mengajak Alice berkencan. Alice merasa gembira atas undangan Charles. Setelah ia membalas beberapa email, Alice langsung berangkat ke restoran Italia tempat ia akan bertemu dengan Charles. Namun, Charles tidak pernah muncul. Alice kesal dan sakit hati, mengapa pria seperti Charles yang kelihatannya baik dapat mempermainkannya. Pulang ke rumah, Alice memutuskan tidak ingin bicara lagi dengan Charles. Tetapi tetap ada sesuatu yang mengganjal dalam hati Alice, apa yang salah ?

Misteri ini akan terbongkar jika kita menyaksikan tindakan Bob beberapa jam sebelum Alice berangkat kencan. Bob adalah seorang administrator keamanan pada perusahaan tempat Alice bekerja. Bob sebenarnya diam-diam naksir Alice dan ingin mengajaknya kencan tapi belum pernah kesampaian. Seorang karyawan lain memberi tahu Bob bahwa Alice sedang siap-siap kencan dengan "teman *online*-nya". Bob cemburu mendengar hal itu dan membalasnya. Berikut ini ceritanya.

3.2 Tanggal 5 Maret Pukul 7:00 – di Rumah Alice

Alice masuk ke email di <http://ewebmail.example.com/> dan memutuskan menulis pesan untuk Charles yang berisi menerima undangan kencannya untuk makan malam dan nonton film. Hatinya tergetar karena sekarang ia bisa berjumpa dengan Charles! Layanan email basis web yang dipakai Alice, yaitu eWebmail

serupa dengan Hotmail atau Yahoo!Mail. Mekanisme loginnya serupa, yaitu mencakup penerimaan *Username* dan *password* melalui form HTML.

Alice kemudian segera masuk ke bagian “*compose*” pada email itu untuk menulis pesannya kepada Charles. Ia memberi tahu Charles jam dan tempat kencan malam nanti. Setelah mengirimkan pesannya, Alice baru sadar kalau dia hampir terlambat untuk pertemuan pagi di kantornya. Dia segera berganti pakaian dan bersiap berangkat ke kantor melewati lalu lintas pagi yang sibuk.

3.3 Pukul 8:30 – di Kantor Alice

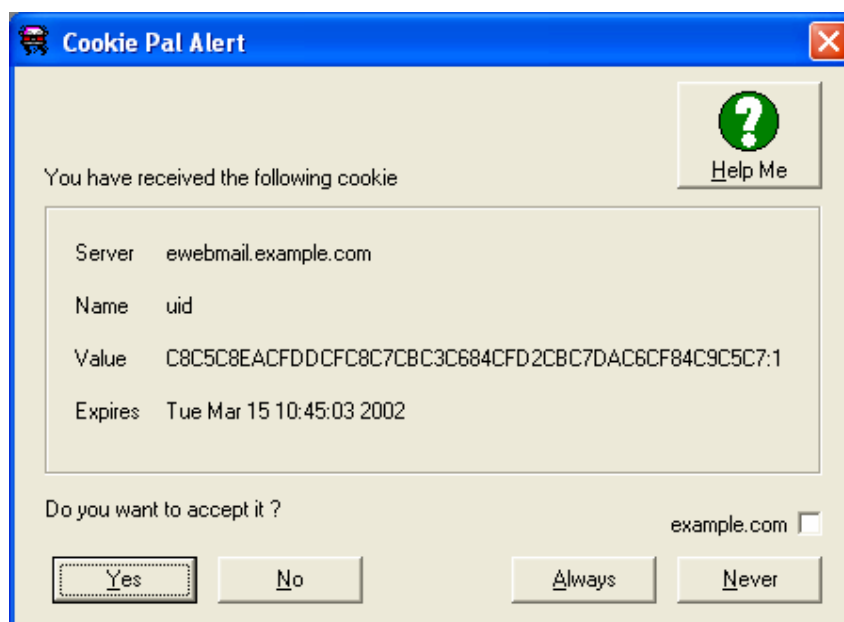
Alice berjalan masuk ke lobi kantor perusahaannya dan disapa oleh Nicole, si resepsionis. Nicole dan Alice berteman akrab. Dengan gembira Alice mengatakan kepada Nicole bahwa akhirnya ia bisa bertemu muka dengan Charles nanti malam. Beberapa saat setelah Alice memasuki ruang pertemuan, karyawan lain, Bob, masuk ke lobby. Setelah saling bertegur sapa, Nicole memberi tahu Bob mengenai kencan Alice nanti malam. Bob memasuki ruang kantornya dan berusaha berkonsentrasi untuk kerja. Dia kecewa karena seseorang lebih dahulu mengajak Alice kencan. Satu jam kemudian Bob akhirnya memutuskan untuk mencoba apakah ia bisa memasuki email Alice dan mencari tahu apa yang sedang terjadi di antara keduanya.

Bob tahu bahwa Alice menggunakan layanan eWbMail untuk menyimpan email-email pribadinya. Dia berpikir bahwa jika ingin mengakses email Alice, ia perlu membuat *account* pada layanan email yang sama dengan milik Alice. Bulan lalu, Bob menghadiri sebuah pertemuan mengenai sekuriti. Dua orang pembicaranya membahas soal “Web hacking”, dan ia tertarik dengan pembahasan mengenai keamanan pada daerah khusus tersebut. Membuat *account* pada eWbMail sangat mudah, dan Bob dapat segera *login* ke <http://ewbmail.example.com>.

3.4 Pukul 10:00 – di Kantor Bob

Bob secara teliti mempelajari cara kerja eWebMail. Aplikasi ini ternyata dibuat dalam servlet Java dan Java Server Page. Fitur-fitur yang ditawarkan oleh eWebMail sama dengan yang ditawarkan oleh banyak layanan email gratis lain di web.

Sewaktu Bob *login*, sebuah program bernama Cookie Pal muncul di layar dalam bentuk kotak *pop-up*, seperti ditunjukkan dalam Gambar 1 berikut.



Gambar 1. Kotak *pop-up* Cookie Pal

Cookie Pal adalah aplikasi *shareware* yang tersedia pada <http://www.kbura.net/>. Bob menggunakannya untuk memonitor dan mengontrol cookie yang dikirim oleh situs web ke browsernya. Pada kotak *pop-up* itu kelihatannya eWebMail mengirimkan cookie *string* yang panjang, dengan nama “uid”. Nilai “uid” kelihatannya di-*encode* dalam heksadesimal. Cookie ini menarik perhatian Bob, ia pernah mendengar bahwa seringkali aplikasi web menggunakan cookie untuk melewati *session identifier* selama berinteraksi dengan web. Mungkin cookie ini juga semacam *session identifier* dari eWebMail, pikir Bob.

Tindakan Bob berikutnya adalah membuat beberapa *account* baru pada eWebMail. Dia membuat *account* bernama “bob1”, “bob2”, “bob3”. Setiap kali *login* ia menerima *string* cookie yang serupa. Setelah login sebanyak empat kali pada user yang berbeda. Bob mengumpulkan empat *string* cookie :

bob@ewebmail.example.com	C8C5C8EACFDDFC8C7CBC3C684CFD2CBC7DAC6CF84C9C5C7:1
bob1@ewebmail.example.com	C8C5C89BEACFDDFC8C7CBC3C684CFD2CBC7DAC6CF84C9C5C7:1
bob2@ewebmail.example.com	C8C5C898EACFDDFC8C7CBC3C684CFD2CBC7DAC6CF84C9C5C7:1
bob3@ewebmail.example.com	C8C5C899EACFDDFC8C7CBC3C684CFD2CBC7DAC6CF84C9C5C7:1

Sewaktu cookie itu diurutkan secara membujur, dia menemukan dua perkiraan :

1. jumlah “byte” heksadesimal sama persis dengan jumlah karakter pada alamat email,
2. pada tiga alamat terakhir, semua nilai *cookie* berbeda satu sama lain, yaitu pada byte keempat dari awal. Ini mewakili angka 1,2 dan 3 pada alamat email.

Bob belum dapat menemukan penjelasan atas “:1” pada akhir *string cookie*. Pada saat itu kelihatannya karakter itu tidak terlalu penting. Bob berpikir, bila alamat email di-*encode* secara langsung dengan cara heksadesimal pasti terlalu mudah. Memang benar, alamat email tidak ter-*encode* dengan cara itu, sebab jika benar di-*encode* langsung ke heksadesimal pasti byte heksadesimal itu akan sama persis dengan nilai ASCII dari karakter pada string alamat email. Bob menyimpulkan bahwa ada suatu teknik enkripsi sederhana yang digunakan untuk mengenkripsi alamat email dan memperoleh *string cookie* darinya. Sewaktu melihat pola string yang ter-*encode*, Bob mengetahui bahwa *developer* pada eWebMail menggunakan bentuk enkripsi XOR yang lemah. Segera ia memasukkan sedikit *script* Perl untuk mencoba mengenkripsi dengan XOR *string cookie* dalam 256 kombinasi byte dan melihat apakah salah satunya menampilkan alamat email yang sebenarnya. Bob bergembira ketika mencapai karakter 0xAA, *string cookie* diperoleh dengan meng-XOR-kan tiap karakter dalam string alamat email dengan 0xAA, yang pola bitnya adalah 01010101.

Bersenjatakan pengetahuan ini, Bob siap berusaha meng-*hack account* email Alice. Bob juga menyadari bahwa eWebMail mengatur tanggal dan waktu *expire cookie* selama 1 jam sejak *cookie* tersebut digunakan. Dengan kata lain, sesi *login* tipikal dari eWebMail berlangsung selama satu jam selama tidak ada aktivitas. Jika

browser tetap *idle* selama satu jam dan kemudian muncul beberapa aktivitas, *cookie* tidak akan diulangi dan eWebMail otomatis *me-logoff* user. Untuk meng-*hack* email Alice, Bob membuat sebuah string cookie terenkripsi XOR dari alamat email Alice `alice@ewbmail.example.com`. Gambaran Operasi XOR dapat dilihat pada Tabel 1.

3.5 Pukul 11:00 – di Kantor Bob

Bob keluar dari ruangan untuk minum. Dia melihat Alice masih sibuk dengan pertemuannya di ruang konferensi. Dia tersenyum tipis dan kembali ke ruang kerjanya dan menutup pintu. Dia membersihkan browser-nya dari semua cookie dan login ke eWebMail sebagai bob@ewebmailexample.com. Cookie “uid” di-set, dan segeralah ia dapat melihat inbox-nya. Dia memiliki satu buah pesan email. Email ini berasal dari service eWebMail.

Dia melihat lagi ke inbox dan melihatnya sebentar, kemudian menutup window Netscapenya. Kemudian dia membuka file cookie Netscape, `cookie.txt`, dan mencari cookie yang di-set oleh `ewebmail.example.com`. Cookie itu berisi nilai “uid” terenkripsi.

```
Ewebmail.example.com    FALSE / FALSE1020114192 uid  
C8C5C8EACFDDFC8C7CBC3C684CFD2CBC7DAC6CF84C9C5C7:1
```

Tabel 1. Hasil Operasi XOR alamat email

Huruf	a	l	i	c	e	@	e	w	e	b	m	a	i	l	.	e	x	a	m	p	l	e	.	c	o	m	
Kode	61	6C	69	63	65	40	65	77	65	62	6D	61	69	6C	2E	65	78	61	6D	70	6C	65	2E	63	6F	6D	
XOR	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA	AA
hasil	CB	C6	C3	C9	CF	EA	CF	DD	CF	C8	C7	CB	C3	C6	84	CF	D2	CB	C7	DA	C6	CF	84	C9	C5	C7	

Dia kemudian mengganti nilai cookie dengan enkripsi XOR dari `alice@ewbmail.example.com`. Dia membiarkan akhiran `":1"` sebagaimana adanya.

```
Ewebmail.example.com FALSE / FALSE1020114192 uid  
CBC6C3C9CFEACFDDCF8C7CBC3C684CFD2CBC7DAC6CF84C9C7DAC6CF8  
4C9C5C7:1
```

Kemudian dibukanya Netscape dan mengirim *request* <http://ewebmail.example.com/>. Ini dia! Kini dia dapat melihat layar *inbox* Alice pada browser-nya. Dia melihat bahwa Alice menerima tiga email baru hari itu. Bob mengklik *link inbox* untuk melihat daftar pesan email Alice. Salah satu email yang menarik adalah dari Charles Le Tan “Re : Makan malam dan nonton” menarik baginya. Dia membuka pesan itu dan membacanya. Pesan Charles Le Tan sampai pada pukul 10:16 pagi, dan ada di *inbox*. Rupanya Alice belum ada kesempatan untuk memeriksa email pribadinya itu. Pada pesan itu, Charles menawarkan tempat restoran yang lain. Dalam keadaan sebal dan kecewa untuk sesaat, Bob menghapus balasan email Charles kepada Alice dan *logoff* dari aplikasi email. Alice tidak akan pernah lagi mengetahui pesan Charles untuk bertemu di restoran lain.

3.6 Pukul 12:30 – di Kantor Alice

Pertemuan itu berlangsung lama daripada biasanya. Alice kembali ke ruang kantornya dan memeriksa emailnya cepat-cepat sebelum dia makan siang. Dia *login* ke *account* emailnya dan melihat dua pesan yang belum dibuka. Pesan pertama berasal dari temannya Tammy di Kanada yang mengiriminya beberapa foto anak-anaknya. Pesan yang lain berasal dari CNET, sebuah tabloid elektronik mingguan yang digunakannya untuk memperoleh berita-berita baru mengenai perkembangan PC. Dia kemudian *logoff* lalu makan siang.

3.7 Pukul 21:30 – di Restoran Italia Bertolini

Setelah menunggu selama dua jam, Alice mulai menyadari bahwa ia dipermainkan oleh Charles. Padahal Charles pada saat yang sama menyerah juga menunggu Alice sampai pukul 21:00 di Las Brisas. Kedua orang itu saling tidak mengetahui mengapa pasangannya tidak datang.

Istilah sesi pembajakan (*session hijacking*) umumnya digunakan untuk menggambarkan proses sebuah koneksi TCP yang diambil alih oleh sebuah rangkaian serangan yang sudah dapat diprediksi sebelumnya. Pada serangan seperti itu, penyerang memperoleh kendali melalui koneksi TCP yang sudah ada. Bila diterapkan pada keamanan aplikasi web, *session hijacking* mengacu pada pengambilalihan sebuah *session* aplikasi web.

HTTP adalah protokol berciri *stateless* (tidak punya tempat yang tetap) karena ia sebenarnya berfungsi untuk menyebarkan informasi. Klien *me-request* sumber tertentu yang kebetulan dikirim oleh server yang *meng-host* sumber tertentu itu. Awal mulanya, tujuan World Wide Web adalah menjadi media penyatu untuk penyebaran informasi HTTP dan menterjemahkan informasi itu melalui HTML. Informasi juga dapat berciri rujuk silang dengan menggunakan hyperlink. Selama berjalannya waktu, banyak server dikembangkan dengan kemampuan untuk menangani isi web yang dinamis dan mengeksekusi program yang membangkitkan HTML. Akhirnya, kebutuhan akan interaksi makin meningkat. Oleh karena kemampuannya menangani teks dan grafik, browser pun mengambil peran pada tempat klien secara universal. Aplikasi berskala kecil mulai di-*host* pada web server dengan menggunakan *script* CGI yang memperluas kemampuan partisipasi secara universal ke seluruh pemakai internet yang memiliki browser. Sistem operasi yang mendasari semua itu tidak lagi menjadi topik utama. Perkembangan aplikasi bergerak mulai dari konsep berbasis *central mainframe-terminal* ke model *client-server*, dan kembali ke konsep pusat berbasis web server-browser lagi.

Dewasa ini, server aplikasi web melakukan *hosting* aplikasi yang kompleks, misalnya keseluruhan aplikasi produktivitas kantor. Microsoft Outlook untuk web merupakan contoh dari pengiriman *full* fitur klien email melalui browser web. Server Lotus Domino menyediakan antarmuka web yang membuat *User* dapat menjalankan kurang lebih *task* yang sama seperti dijalankan melalui kliennya, Lotus Notes.

Semua aplikasi *multi-User* menerapkan konsep dari *session* user. Setiap user berinteraksi dengan aplikasi melalui *session* user yang berbeda. Aplikasi tetap melacak semua yang sedang menggunakan aplikasi tersebut melalui *session*. Kemampuan ini merupakan pokok dari pemisahan aktivitas user.

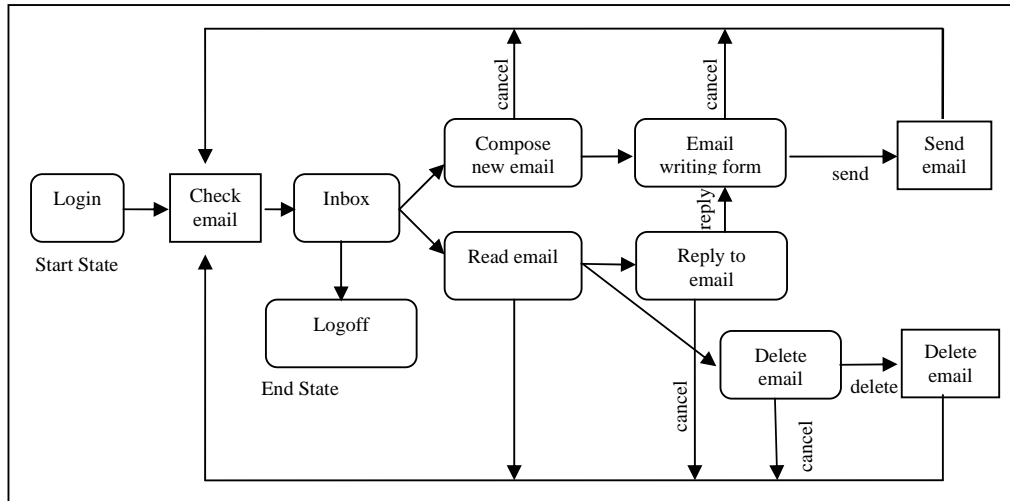
Walaupun teknologi web server mengalami perubahan drastis, tetapi protokol HTTP tetap saja sama. Saat ini, protokol HTTP 1.1. tetap yang paling banyak dipakai. Rintangan terberat dalam mendesain dan meng-*host* aplikasi berbasis web adalah seputar HTTP yang *stateless* itu. Tak ada standar yang mengatur bagaimana sebuah aplikasi berbasis web harus menyediakan mekanisme *state-maintaining*-nya sendiri melalui HTTP. Developern melakukan penetapan *state* dengan cara yang berbeda. Ada pendekatan yang baik dan buruk untuk masalah ini, walaupun kedua pendekatan itu menghasilkan aplikasi yang dapat bekerja normal. Teknik penerapan *session state* yang kurang baik dapat membawa kepada *session hijacking*.

3.8 Postmortem Serangan *Session Hijacking*

Mari meninjau kembali serangan Alice-Bob-Charles, Bob memulai *session* aplikasi pada eWbMail dengan identitasnya sendiri. Ia tidak mengetahui password Alice. Tetapi dia kemudian dapat mengetahui password itu dan akhirnya memperdaya mekanisme *session state* yang digunakan oleh eWebMail. Sewaktu *session* miliknya setengah jalan, dia mengganti password-nya dengan password milik Alice dan menirunya. Kekeliruan atau kurangnya pemahaman mengenai masalah yang disebabkan oleh buruknya *session management* menghasilkan serangan seperti itu.

3.9 Diagram Aplikasi State

Akan lebih baik menggambarkan apa yang salah dari aplikasi eWebMail dengan menggunakan diagram *state*, seperti ditunjukkan pada Gambar 2. Diagram *state* adalah bagian dari cabang matematika diskrit yang disebut otomata *state* terbatas atau teori mesin *state* terbatas.



Gambar 2. Diagram state dari eWebMail

Setiap aplikasi sedikitnya memiliki dua *state* : *state* permulaan dan *state* akhir. Sewaktu menjalankan aplikasi, *User* dikatakan berada pada salah satu dari sekian banyak *state* aplikasi. Pada Gambar 2, kotak bersudut melengkung menunjukkan *state*. Kotak yang bersudut lancip menunjukkan proses yang ada pada bagian internal dari aplikasi. Masng-masing *state* terhubung satu sama lain oleh *path* transisi yang membawa *User* dari satu *state* ke *state* yang lainnya. Transisi diantara dua *state* tidak dimungkinkan bila *path* transisi tidak ada diantaranya. Pernyataan terakhir ini sanga penting, sebab pada hakikatnya serangan sesi pembajakan mempergunakan kecacatan dalam bentuk transisi ilegal.

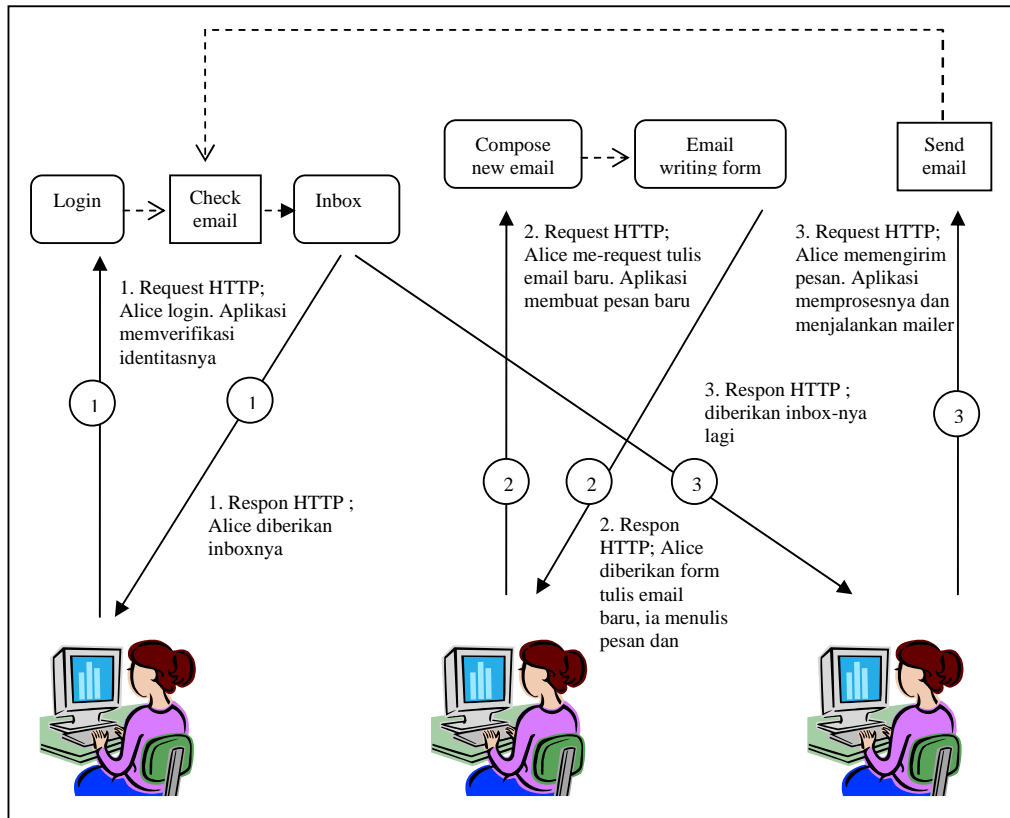
Dalam hal eWebMail, mari lihat *state* bernama Read email. *User* dapat mencapai *state* ini hanya jika ia sudah berada pada *state* Inbox. *User* mengindikasikan kepada aplikasi bahwa ia ingin berganti sesi (transisi) dari *state* Inbox ke *state* Read email dengan mengklik hyperlink pada inbox. Secara internal, aplikasi memproses *request* user ini dan menampilkan data email yang ada pada

inbox. Tindakan ini menyebabkan user bertransisi ke *state* Read email. Dari *state* Read email, dia memiliki tiga kemungkinan, kembali ke *state* Inbox, ke *state* Reply email, atau ke *state* Delete email.

Titik masuk ke aplikasi diatur oleh *state* Start. Dalam hal ini, adalah *state* Login, yang diperbolehkan hanya jika user memasukkan ID dan *password* yang benar. Sewaktu dimasukkan dengan ID dan *password* yang benar, aplikasi membuat *logical session*, *Session* mengikat user kepada aplikasi dan bertanggung jawab melacak *state* user dalam aplikasi. *Session* ini berlangsung sampai *state* End dicapai atau ketika muncul *error*. Dalam kasus ini *state* Log-Off menterminasi user *session*. Pada titik ini, user tidak berhubungan lagi dengan aplikasi.

3.10 HTTP dan *Session Tracking*

Untuk melihat bagaimana *session tracking* dijalankan melalui HTTP, sebagai contoh Alice menggunakan eWebMail untuk mengirim pesan email ke Charles. Gambar 3 menunjukkan bagaimana Alice menggunakan eWebMail, dalam hubungan dengan diagram *state* dan sebuah *session*.



Gambar 3. *Session Alice pada eWebMail*

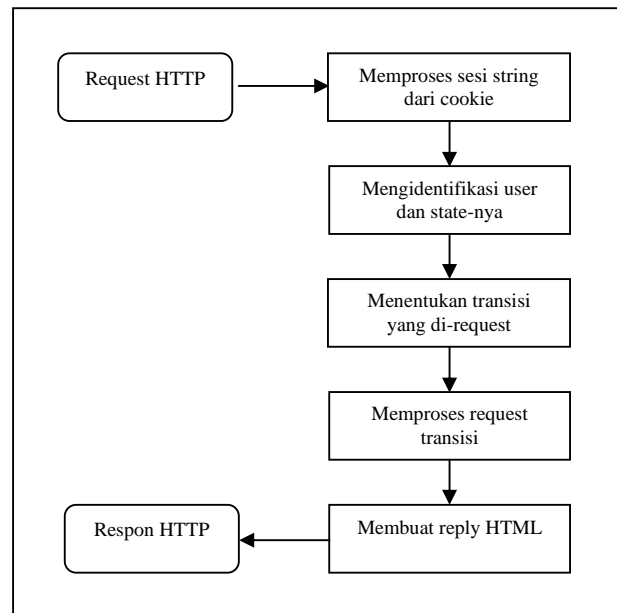
Alice mulai dengan *state* Login. Dia memasukkan *username* dan *password*nya dan berhasil keluar dari *state* Login. Pada gilirannya, aplikasi membuat “uid” bagi Alice dan mengirimkannya sebagai cookie browser ke browser Alice. Aplikasi mengatur waktu akhir selama satu jam terhitung sejak saat dikeluarkan cookie tersebut. Dengan demikian, setiap kali browser mengirimkan *request* ke <http://ewebmail.example.com/>, “uid” juga dikirim bersama dengan *request* HTTP, sampai cookie itu habis masa berlakunya. Setelah Alice melewati *state* Login, aplikasi kemudian membawanya ke *state* Inbox. Sekarang aplikasi mengirim respon ke *request* HTTP yang diterimanya sementara Alice ada pada *state* Login. Respon ini terdiri dari View Inbox yang muncul pada browser Alice, melengkapi transaksi HTTP. Browser dan web server tidak memiliki koneksi jaringan yang sedang bekerja di antara mereka. Pada titik ini mereka hanya terkoneksi secara *logical*.

Pada *state* Inbox, Alice ingin menulis pesan email baru dengan meng-klik link Compose, browser mengirim *request* HTTP itu beserta cookienya ke <http://ewebmail.example.com/>. Sewaktu aplikasi menerima *request*, ia pertamanya mendeskripsi string cookie dengan teknik XOR dan mengetahui *request* itu berasal dari Alice. Beginilah aplikasi itu membuat *binding logical* dengan identitas user ke *request* HTTP. Bagaimana aplikasi itu dapat mengetahui bahwa Alice ada pada *state* Inbox ? Itulah gunanya “:1” yang terletak pada bagian akhir string cookie. Bob tidak mengetahui bahwa “:1” merupakan nomor *state*. Dalam kasus eWebMail, setiap *state* memiliki nomor *state*-nya sendiri yang dilewatkan antara browser dan server. Nomor 1 merupakan *state* Inbox, nomor 2 merupakan *state* Read email, dan nomor 3 merupakan *state* Buat email baru, dan seterusnya. Dengan mendeteksi “:1”, aplikasi mengetahui bahwa Alice berada pada *state* 1, atau *state* Inbox, sebelum ia me-*request* transisi ke *state* 3, atau *state* Buat email baru. Secara internal, aplikasi sekarang mengatur nomor *state*-nya menjadi 3, dan mentransisi *session*nya ke *state* Compose email. HTTP me-*reply* dengan memberikan form untuk membuat pesan email baru, seperti yang ditunjukkan pada Gambar..... String cookie sekarang berisi “:3” pada bagian akhirnya. Browser mengganti cookie yang lama dengan yang baru beserta waktu kadaluarsanya. Browser dan web server sekali lagi diputuskan dari koneksi jaringan.

Sewaktu Alice selesai membuat pesan email, dia meng-klik tombol “Send”. Dengan demikian *request* HTTP lain dikirimkan ke eWebMail. Cookie baru dikirimkan, dan Alice berpindah dari form Buat email baru ke proses Send email. Proses Send email merupakan *state* temporer yang memiliki transisi sendiri. Sekali email diserahkan ke *mailer* program, aplikasi mentransisi Alice ke *state* Inbox. Sewaktu menuju ke *state* Inbox, cookie mengirimkan responnya yang berakhiran “:1”. Sewaktu Alice meng-klik Log-Off, aplikasi akan mengirimkan string cookie kosong, yang akan menghapus cookie dari browser Alice. Berakhirlah *session* Alice pada eWebMail.

3.11 Aplikasi Stateless vs Aplikasi Stateful

Pada bagian sebelumnya, telah diuraikann cara melacak *session* Alice melalui aplikasi eWebMail dan menguji bagaimana aplikasi tetap membuntuti *state* Alice. Tetapi, bisakah dikatakan bahwa aplikasi itu sungguh-sungguh merupakan aplikasi *stateful* ? Setiap kali *request* HTTP dikirim, aplikasi mengetahui identitas user dan *state*-nya dari cookie. Aplikasi kemudian menangani *request* dan memberikan *reply* HTTP. Gambar 4 menunjukkan bagaimana aplikasi itu dipanggil setiap kali *request* HTTP muncul.

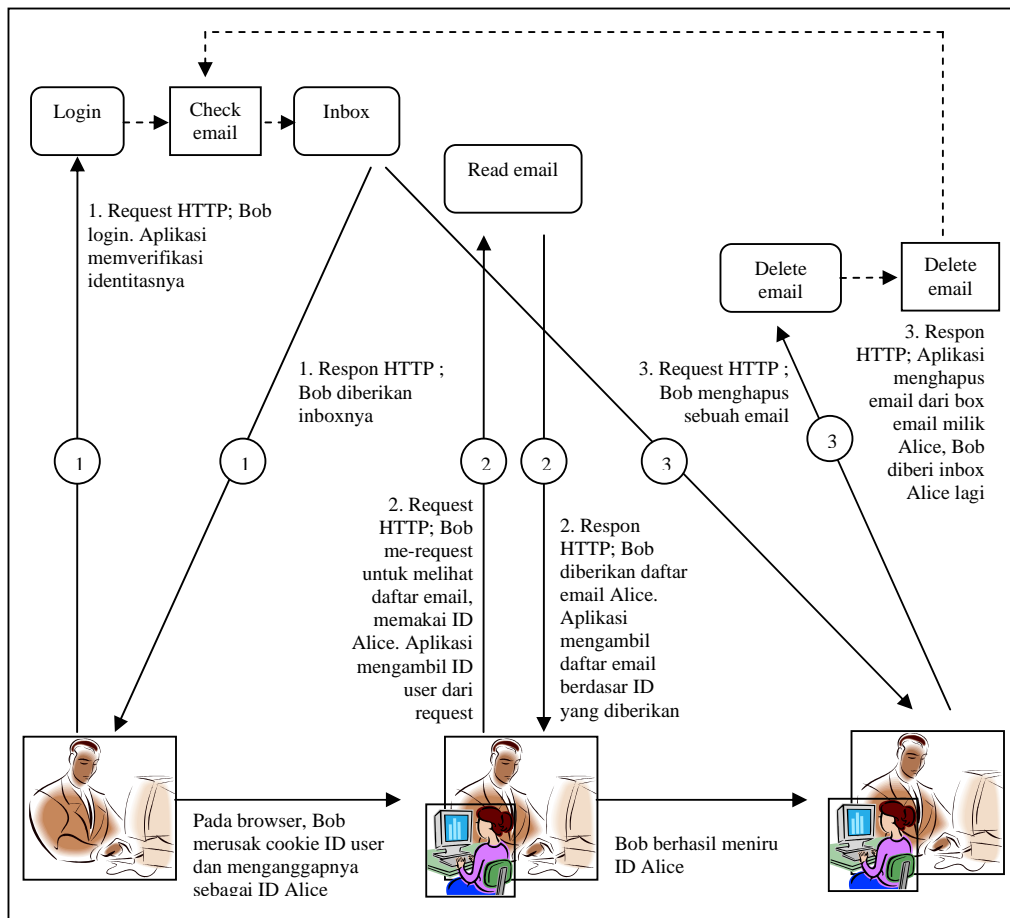


Gambar 4. Diagram alur dari aplikasi eWebMail

Setiap *request* yang datang ke aplikasi mengikuti *path* yang sama. Dengan demikian, aplikasi itu bukanlah aplikasi yang sungguh-sungguh aplikasi *stateful*. Istilah sungguh-sungguh *stateful* berarti aplikasi tetap melacak user *session* dan *state*-nya secara independen. Dalam term web server, hal itu berarti *session tracking* dijalankan pada server side.

Di sini keseluruhan batasan dari *session tracking* diserahkan ke browser klien. Menggunakan cookie merupakan cara yang mudah dalam melewati informasi ke browser hingga pada *request* berikutnya, web server akan menerima

kembali informasi yang sama. *Session tracking* pada client side mempermudah pemrograman aplikasi. Tetapi developer sering cenderung melupakan peraturan utama mengenai keamanan aplikasi web : setiap informasi yang datang dari luar batasan web server tersebut dapat dirusak atau diubah-ubah. Inilah yang terjadi pada serangan Bob-Alice-Charles. Bob merusak dengan cara mengganti cookie yang berisi sesi informasi, yang menyebabkan aplikasi menyerahkan data Alice kepada Bob. Gambar 5 meringkas serangan *session hijacking* yang dilakukan oleh Bob.



Gambar 5. Interaksi Bob dengan eWebMail

3.12 Cookie dan Field Tersembunyi

Walaupun sebuah aplikasi sungguh-sungguh *stateful*, ia perlu mengirimkan dan mengembalikan informasi *session identifier* antara browser dan dirinya sendiri. Hal ini diperlukan karena HTTP umumnya memutuskan hubungan TCP setelah setiap respon telah dilakukan. Sewaktu user mengeluarkan *request* baru, aplikasi harus memiliki cara untuk menentukan bahwa user yang bersangkutan sudah ada di *session* aplikasi.

Ada dua cara untuk mengirimkan dan mengembalikan informasi antara browser dan web server, yaitu cookie dan field tersembunyi. Uraian berikut ini menjabarkan keuntungan dan kerugian kedua pendekatan tersebut.

3.12.1 Cookie

Cookie ditangani melalui browser. Browser mengirimkan cookie yang diperlukan ke web server bersama dengan *request* HTTP jika sebelumnya ada cookie yang diterima dari server yang sama. Browser terkenal, seperti Netscape, Internet Explorer, dan Opera menangani cookie secara baik.

Segi kerugiannya adalah kebanyakan situs menggunakan cookie untuk melacak tingkah laku user. Situs yang menampilkan *banner* iklan diketahui melanggar *privacy* user dengan cara mengumpulkan informasi tentang user secara berlebihan melalui pelacakan aktivitas user via cookie dan acuan-acuan HTTP. Sayangnya, browser tidak memiliki mekanisme *built-in* yang memadai untuk secara selektif hanya memilih cookie-cookie tertentu saja. Untuk maksud ini program seperti Cookie Pal dapat digunakan sebagai alat bantu.

Cookie lebih menguntungkan daripada field tersembunyi. Field tersembunyi selalu memerlukan halaman form HTML untuk dikirim kembali ke server, sedangkan cookie tidak memerlukan form HTML apapun.

3.12.2 Field Tersebunyi

Field tersembunyi di dalam form HTML dapat juga digunakan untuk mengirimkan dan mengembalikan informasi antara browser dan web server. Keuntungan field tersembunyi dibandingkan cookie adalah field tersebut tetap dapat bekerja walaupun browser telah diatur untuk menolak semua cookie.

Tetapi, baik cookie maupun field tersembunyi sewaktu ditransmisikan sangat terbuka untuk dirusak dan dimanfaatkan oleh pihak lain. User dapat mengubah nilai pada cookie maupun field tersembunyi sementara ia berinteraksi dengan aplikasi web. Oleh karena itu, sangat perlu dipikirkan sebuah cara untuk menangkal perusakan session identifier dan mekanisme session tracking.

3.13 Mengimplementasikan *Session* dan *State Tracking*

Session hijacking dimungkinkan terutama jika aplikasi web tergantung sepenuhnya pada client side untuk melacak *session* dan *state* informasi. Dalam beberapa hal, walaupun *session tracking* dijalankan pada server side, *session identifier* yang dinyatakan sukses bisa saja mengakibatkan *session hijacking*. Dalam kasus serangan Bob-Alice-Charles, Bob dengan mudah memperdaya *session identifier* dengan *reserve engineering*. *Session identifier* yang menghasilkan sendiri rangkaian prediksi serangan dapat menjadi korban serangan *session hijacking*. Oleh karena itu sangat penting untuk menerapkan metode yang memadai untuk *session* dan *state tracking*.

Ada beberapa aturan berikut ini yang dapat digunakan untuk menerapkan *session* dan *state tracking* secara benar. Aturan-aturan tersebut sama sekali tidak melengkapi atau mengikat suatu aplikasi. Tetapi sebaliknya, aturan-aturan ini dapat menjadi petunjuk untuk mendesain sebuah *session* dan mekanisme *state tracking*.

3.13.1 *Session Identifier* Harus Unik

Sekali lagi, dalam tiap aplikasi web, sebuah *logical session* harus ditetapkan antara browser dan web server. Untuk tujuan itu, suatu string atau sebuah angka

dari sekumpulan data yang disebut *session identifier* perlu dikirimkan dan dikembalikan antara web server dan browser. Lebih baik, tiap user *session* pada aplikasi harus diidentifikasi oleh sebuah *session identifier* yang unik yang tidak dapat digunakan kembali dari satu *session* ke *session* yang lain. Sekalipun user yang sama logon kembali, suatu *session identifier* yang baru harus tetap dibuat.

3.13.2 *Session Identifier* Tidak Boleh “Mudah Diterka”

Cara yang mudah untuk mengatasi *session identifier* yang lemah adalah dengan banyak membuat user *session* secara cepat atau terus menerus membuat dan menghapus user *session* dalam selang waktu yang pendek. *Session identifier* yang berpola sekuensial, atau mengikuti selang waktu atau pola tertentu rawan terhadap serangan karena pola seperti ini dapat diprediksi. Saat penyerang dapat memprediksi pola tersebut, maka ia dapat membuat *session identifier* yang baru dengan menggunakan identitas user lain padahal ia sedang logon di aplikasi yang bersangkutan.

Cara mudah untuk membuat *session identifier* yang berpola acak atau tidak berpola sekuensial adalah menggunakan kombinasi angka acak, menggunakan *time stamp* (pencatat waktu), dan angka rahasia untuk membuat campuran kombinasi. Campuran itu juga memiliki properti *low-crash*, yang membantu mengalahkan serangan rangkaian prediksi (*sequence prediction*).

3.13.3 *Session Identifier* Harus Independen

Session identifier tidak boleh diambil dari *username*, *password* dan *state* aplikasi. Tetapi, dari sebuah tabel pencarian yang harus disediakan pada server side. Tabel ini memetakan *session identifier* dengan data rahasia user dan *state* aplikasi.

3.13.4 *Session Identifier* Harus Dapat Dipetakan dengan Koneksi Client Side

Untuk mencegah pihak lain mendeteksi dan menggunakan kembali *session identifier* milik user lain pada jaringan dan aplikasi yang sama, aplikasi ini harus tetap dapat melacak alamat IP klien dan waktu pembuatan *session* pada server side. *Session identifier* harus dapat dipetakan dengan informasi koneksi klien. Setiap kali sebuah *request* diterima dari klien, informasi koneksi klien yang ada harus dapat dikomparasi dengan informasi yang disimpan. Jika ada ketidaksesuaian, *session* tersebut harus secara otomatis dibatalkan.

Lebih baik, *session identifier* dikirimkan ke browser melalui SSL setelah *session* selesai dibuat pada server side. Tindakan ini juga mencegah penggunaan ulang serangan yang ditujukan pada *session hijacking*.

BAB IV

KESIMPULAN

Serangan *session hijacking* dilakukan tidak semudah serangan aplikasi web lainnya. Tetapi efeknya bisa sangat merusak. Pokok masalah serangan ini semata-mata pada cara desain dan pengembangan aplikasi. Kelalaian dalam mendesain atau dalam mengimplementasikan mekanisme *session tracking* pada aplikasi akan menghasilkan kelemahan-kelemahan seperti yang telah dibahas pada bagian sebelumnya. Tak satupun *patch* sistem operasi, firewall atau konfigurasi web server dapat mencegah serangan *session hijacking*. Tiap pengembang web harus mengerjakan secara cermat desain dan implementasi *session* dan *state tracking*. Server-server komersial kelas menengah sampai *high-end* memiliki mekanisme *session tracking built-in* dan menyediakan sebuah API untuk membantu developer dalam mendesain aplikasi web.

DAFTAR PUSTAKA

¹ <http://www.jasakom.com/Artikel.asp?ID=8>, 12/25/2004 7:17:50 AM

² http://www.iss.net/security_center/advice/Exploits/TCP/session_hijacking/default.htm, 12/25/2004 6:43:21 AM

³ Mc.Clare, Stuart; Shah, Saumil; Shah, Shreejah, *Attacks and Defense*, Pearson Education Inc, 2003