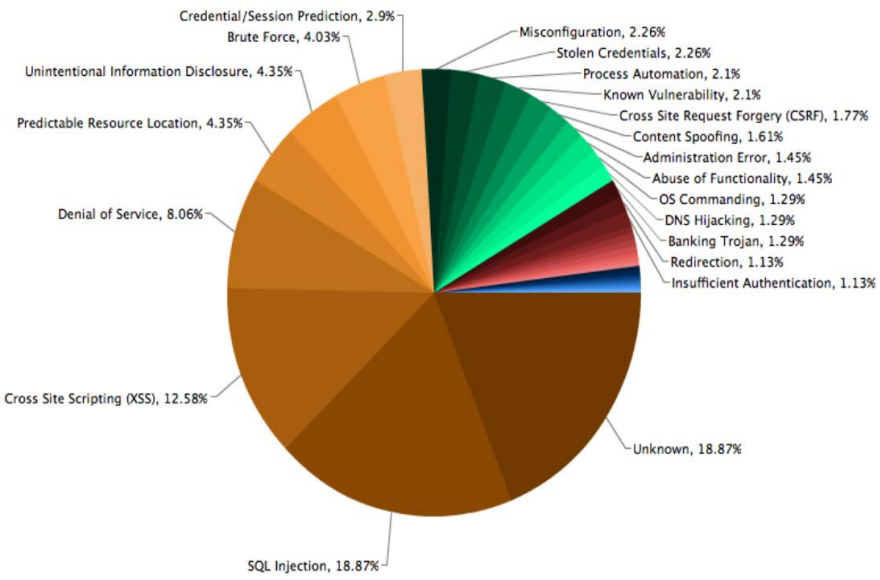
BAB 3
PEMBAHASAN

3.1 Analisis Sistem

Analisis Sistem adalah penguraian dari suatu masalah yang utuh ke dalam bagian-bagian komponennya dengan maksud untuk mengidentifikasikan dan mengevaluasi permasalahan-permasalahan, kesempatan-kesempatan, hambatan-hambatan yang terjadi dan kebutuhan-kebutuhan yang diharapkan sehingga dapatdiusulkan perbaikan-perbaikannya.

3.1.1 Analisis Masalah

Berdasarkan data dari *Web Hacking Incidents Database* (WHID) pada tahun 2013, dari 75 jenis ancaman terhadap aplikasi web, sebanyak 18,87% merupakan ancaman *SQL Injection* dan 12,58% adalah ancaman *Cross Site Scripting*.



Gambar 3.1 Metode Ancaman Yang Terjadi

Ancaman *SQL Injection* merupakan ancaman yang terjadi pada database aplikasi web. Database digunakan sebagai media penyimpanan data yang

berisi informasi berkaitan dengan aplikasi. Dalam penanganan ancaman *SQL Injection*, metode yang digunakan adalah *Base64_Encode*, namun metode ini masih rentan terhadap manipulasi *string*. Ancaman kedua adalah *Cross Site Scripting* yang terjadi pada tampilan aplikasi web. Salah satu penyebab ancaman ini adalah kurangnya metode *Input Encoding* dalam proses penyaringan karakter sehingga mengizinkan karakter tertentu dieksekusi. Kedua ancaman di atas menyulitkan para perancang aplikasi web untuk mengetahui mekanisme ancaman yang terjadi.

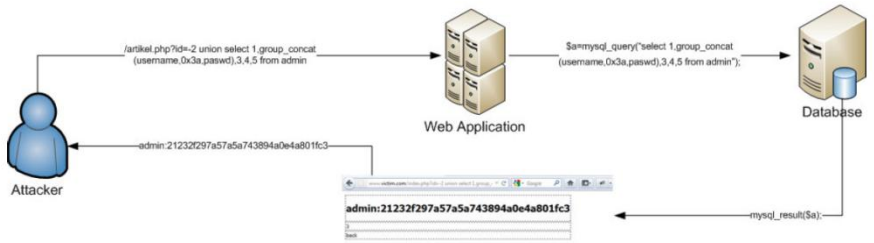
Berikut ini adalah hal-hal yang akan dianalisis, yaitu Analisis Metode Ancaman dan Analisis Metode Pertahanan.

3.1.2 Analisis Metode Ancaman

Metode ancaman yang digunakan dalam penelitian ini adalah:

3.1.2.1 SQL Injection

SQL injection adalah kelemahan yang diterjadi ketika penyerang mampu mengubah *Structured Query Language* (SQL) di dalam database. SQL injection merupakan salah satu kelemahan keamanan yang sangat mempengaruhi sistem bisnis, karena dapat menyebabkan pembongkaran semua informasi sensitif yang tersimpan dalam sebuah aplikasi *database*, termasuk informasi yang berguna seperti *username*, *password*, nama, alamat, nomor telepon, dan rincian kartu kredit. Dengan mempengaruhi *database*, penyerang dapat memanfaatkan sintaks, kemampuan, kekuatan dan fleksibilitas dari SQL yang mendukung fungsi database dan fungsi sistem operasi yang hanya dilakukan dalam database.



Gambar 3.2 Mekanisme *SQL Injection*

a. Cara Kerja SQL Injection

Aplikasi web menjadi lebih canggih dengan teknis yang kompleks. Sifat aplikasi web dengan desain yang memiliki beragam fitur dan kemampuan web untuk menyusun, memproses, dan menyerbarkan informasi melalui internet atau dari dalam intranet membuat aplikasi web tersebut menjadi target yang populer untuk diserang. Penyebab lain Aplikasi web menjadi target penyerangan adalah karena keamanan jaringan pada pasar teknologi bisa menembus sistem informasi melalui kelemahan networkbased.

SQL injection merupakan serangan dimana kode SQL dimasukkan atau ditambahkan ke dalam aplikasi/user input parameter yang kemudian diteruskan ke SQL server back-end untuk parsing dan eksekusi. Setiap prosedur yang membangun pernyataan SQL berpotensi untuk mudah diserang.

Bentuk injeksi utama SQL terdiri dari penyisipan langsung kode ke dalam parameter yang digabungkan dengan perintah SQL dan kemudian dieksekusi. Serangan langsung dilakukan dengan menyuntikan kode ke dalam string yang ditujukan ke penyimpanan di dalam tabel atau sebagai metadata. String yang tersimpan akhirnya digabungkan ke dalam perintah SQL dinamis untuk dieksekusi. Jika aplikasi web gagal untuk membersihkan paramater dengan benar, maka dilewatkan ke pernyataan SQL yang dibuat secara dinamis, hal ini memungkinkan untuk seorang penyerang untuk mengubah mengubah konstruksi back-end SQL.

Ketika seorang penyerang dapat memodifikasi pernyataan SQL, pernyataan tersebut akan dieksekusi dengan hak yang sama dengan aplikasi pengguna, saat menggunakan server SQL untuk mengeksekusi perintah yang berinteraksi dengan sistem operasi, proses akan berjalan dengan hak akses yang sama dengan komponen yang mengeksekusi perintah (misalnya database server, aplikasi server, atau web server).

Simbol-simbol yang digunakan untuk memodifikasi memiliki banyak jenis, berikut simbol (*pattern*) yang digunakan untuk memodifikasi pernyataan SQL:

Tabel 3.1 Daftar simbol penyerang *SQL Injection*

Daftar Simbol Penyerangan *SQL Injection*					
‘	“	#	-	--	‘ --
--‘;	‘ ;	= ‘	= ;	= --	\x23
\x27	\x3D \X3B’	\x3D \x27	\x27\x4F\x52 SELECT *	\x27\x6F\x72 SELECT *	‘or select’
Admin ‘--	‘;shutdown--	<>”’%;(&+	‘ or ‘ ‘=’	‘ or ‘ ‘x’=’x	“ or “x”=”x
‘) or (‘x’=’x	0 or 1=1	‘ or 0=0 --	“ or 0=0 --	or 0=0 --	‘ or 0=0 #
“ or 0=0 #	or 0=0 #	‘ or 1=1 --	“ or 1=1 --	‘ or ‘1’=’1’ --	“ ’ or 1 –‘ “
or 1=1 --	Or % 201=1	Or % 201=1 --	‘ or 1=1 or ‘ ‘ = ‘	“ or 1=1 or “ “ = “	‘ or a=a--
“ or “a”=”a	‘) or (‘a’=’a	“) or (“a”=”a	Hi” or “a”=”a	Hi” or 1=1 --	Hi’ or 1=1 --
Hi’ or ‘a’ =’a	Hi’) or (‘a’=’a	Hi”) or (“a”=”a	‘hi’ or ‘x’=’x’;	@variable	,@variable
PRINT	PRINT @@variable	select	insert	as	or
or	procedure	limit	Order by	asc	desc
delete	Update	distinct	having	truncate	replace
like	handler	bfilename	‘ or username like ‘%	Exec xp	Exec sp
‘; exec master..xp_cmdshell	‘; exec xp_regread	-- sp_password	!	/	//
//*	*/*	@*	*|	|	%2A%7C

b. Baris Perintah SQL Injection

1. Pemakaian Double Minus (--)

Tanda ini menunjukkan akhir dari sebuah statement SQL, sehingga perintah dibelakangnya tidak akan dieksekusi lagi. Sehingga tanda ini berfungsi untuk mengabaikan sisa query setelah tanda double minus tersebut.

Contoh pemakaian : DROP namatabel;--

2. Pemakaian ORDER BY

Untuk menemukan banyaknya kolom dalam query SELECT . Contoh Syntax :

ORDER BY 1--

ORDER BY 2--

ORDER BY 3--

3. Menggunakan UNION

Perintah UNION Select berfungsi untuk menggabungkan hasil query pertama dan query kedua. Syntax

UNION SELECT * FROM nama tabel

c. Step by Step SQL Injection

1. Mencari Celah Keamanan

Ancaman SQL Injection dilakukan pada URL yang mempunyai ID. Mekanisme ancaman dilakukan sebagai berikut :

http://www.target.com/index.php?pageID=1&#8242

Tambahkan tanda petik satu (‘) di akhir id, sehingga menjadi :

http://www.target.com/index.php?pageID=1&#8242’

maka akan tampil pesan kesalahan seperti "*You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right...*", itu menadakan website mempunyai potensi celah keamanan *SQL Injection*.

Tanda petik satu (‘) pada akhir id digunakan untuk mengecek apakah variabel id difilter atau tidak. Apabila variabel tersebut tidak difilter maka akan menampilkan pesan kesalahan yang menyatakan kesalahan pada *query* SQL. Gambar berikut ini menjelaskan mengapa tanda petik satu (‘) pada akhir id dapat menghasilkan pesan kesalahan.

2. Menentukan Jumlah Kolom

Penentuan jumlah kolom perlu dilakukan untuk mengetahui kolom mana dari sebuah tabel yang bisa dimanfaatkan. Hal ini bertujuan supaya kita bisa memasukkan perintah SQL Injection pada lokasi yang tepat. Sebab kalau kita

memasukkan di tempat yang salah maka kita tidak akan memperoleh apapun. Untuk melakukan hal ini agak bersifat "*trial and error*", dimana perintah yang digunakan adalah ORDER BY.

3. Mencari Kolom yang Vunerable

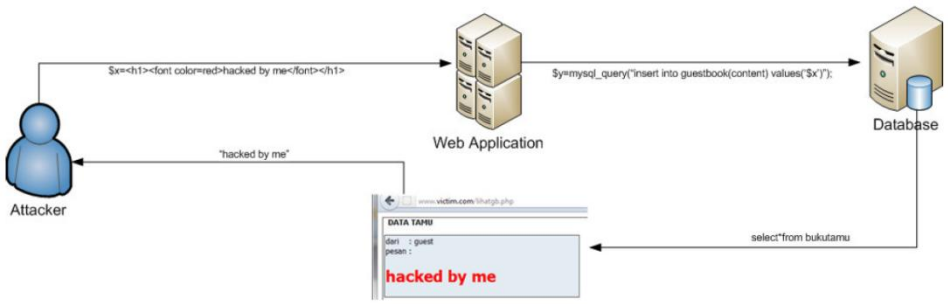
Selanjutnya adalah mencari kolom yang memiliki celah untuk dilakukan *SQL Injection*. Istilah lainnya adalah angka error.

4. Menentukan Nama Database

Pada dasarnya ini bukanlah sesuatu yang penting tapi akan lebih baik agar bisa dimanfaatkan untuk kepentingan lain.

3.1.2.2 Cross Site Scripting

Cross Site Scripting (XSS) adalah salah satu serangan aplikasi-lapisan yang paling umum web. XSS biasanya target script tertanam di halaman yang dijalankan pada sisi-klien (di web browser pengguna) daripada pada sisi-server. XSS adalah sebuah ancaman yang disebabkan oleh kelemahan keamanan internet bahasa *script* sisi klien, dengan HTML dan JavaScript (lainnya adalah VBScript, ActiveX, HTML, atau Flash) sebagai penyebab utama untuk mengeksploitasi. Konsep XSS adalah untuk memanipulasi *script* sisi klien-aplikasi web untuk mengeksekusi dengan cara yang diinginkan oleh pengguna jahat. Seperti manipulasi bisa menanamkan script di halaman yang dapat dijalankan setiap kali halaman di-load, atau kapan sebuah peristiwa yang terkait dilakukan.



Gambar 3.3 *Mekanisme Cross Site Scripting*

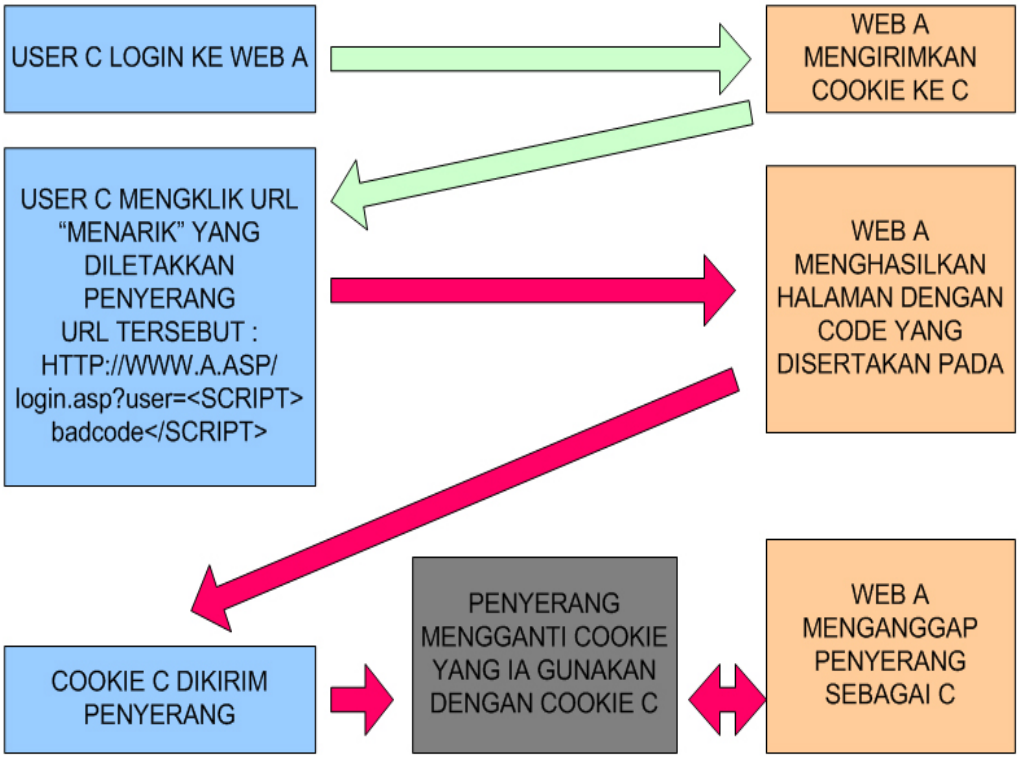
a. Cara Kerja *Cross Site Scripting*

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* yang baik dari si penipu. *Social engineering* merupakan elemen terpenting yang menentukan keberhasilan penipuan yang akan dilakukan. *Cross site scripting* memampukan seseorang yang tidak bertanggung jawab melakukan penyalahgunaan informasi penting.

Sebelum sampai pada proses penyalahgunaan tersebut, penyerang mengambil langkah-langkah dengan mengikuti pola tertentu. Langkah pertama, penyerang melakukakan pengamatan untuk mencari web-web yang memiliki kelemahan *cross site scripting*. Langkah kedua, sang penyerang 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, sang penyerang membujuk korban untuk mengikuti sebuah link yang mengandung kode, ditujukan untuk mendapatkan informasi yang telah disebutkan sebelumnya. Kemampuan *social engineering* dari sang penyerang diuji disini. Setelah mendapatkan informasi tersebut, sang penyerang melakukan langkah terakhir, pencurian maupun pengubahan informasi vital.

Berikut merupakan contoh kasus dari *cross site scripting*. Anggap sebuah penyedia layanan *message board*, A, memiliki kelemahan *cross site scripting*. Web tersebut juga menghasilkan *cookie*. *Cookie* tersebut bertujuan agar pengguna 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 pengguna, 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 cookienya dengan

cookie yang ia dapatkan dari C. Gambar dibawah ini akan menunjukkan alur dari kejadian tersebut.



Gambar 3.4 Contoh kasus cross site scripting

b. Penemuan Titik Rawan Aplikasi

Serangan XSS berdampak pada ketidaksempurnaan aplikasi berbasis web dan berasal pada validasi masukan pengguna (*user*) yang tidak begitu baik. Hal ini dapat memungkinkan penyerang menyisipkan kode HTML tambahan dimana mereka dapat mengendalikan eksekusi pada halaman tersebut dibawah izin yang diberikan oleh situs itu sendiri. Sebuah contoh sederhana dari halaman yang dapat digunakan untuk cross site scripting seperti dibawah ini :

<?php echo "Hello, {$HTTP_GET_VARS['name']}!"; ?>

Sekali halaman tersebut dapat terakses, variabel tersebut dikirim melalui metode GET langsung menuju halaman yang dituju. Dimana masukan tidak

ditandai sebagai variabel masukan, pengguna dapat memasukkan beberapa karakter yang diinterpretasikan sebagai perintah meta karakter, cukup mirip dengan injeksi SQL.

Dengan memasukkan HTML metacharacter memungkinkan keluaran yang tidak diinginkan: Dimana masukan tidak divalidasi terlebih dahulu sebelum keluaran dikirimkan ke browser klien. Hal di atas memberikan pengguna untuk mengendalikan HTML dengan menyisipkan script ke halaman tersebut.

Beberapa titik dimana biasanya XSS 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.

c. Serangan

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 dengan metode GET merupakan cara termudah 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 URI dan biasanya tercatat di server HTTP. Contoh serangan dengan metode ini dapat dilihat seperti berikut:

http://www.microsoft.com/education/?ID=MCTN&target=http://www.microsoft.com/education/?ID=MCTN&target=”><script>alert(document.cookie)</script>

http://hotwired.lycos.com/webmonkey/00/18/index3a_page2.html?tw=<script>alert(‘Test’);</script>

http://www.shopnbc.com/listing.asp?qu=<script>alert(document.cookie)</script>&frompage=4&page=1&ct=VVTV&mh=0&sh=0&RN=1

http://www.oracle.co.jp/mts_sem_owa/MTS_SEM/im_search_exe?search_text=%22%3E%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E

Karena karakter dari XSS, penyerang tidak dapat secara langsung memanfaatkan kerawanan untuk keuntungan sendiri. Korban mesti melihat kode yang disisipkan agar dapat dieksekusi. Dan begitu korban melihat kode tersebut, dan mengeksekusi, informasi tentang korban dapat diketahui oleh penyerang.

Dibanding serangan dengan metode GET, dengan metode POST sedikit lebih rumit, dimana variabel POST dikirim secara mandiri dari permintaan URI, sehingga butuh halaman transisi untuk memaksa korban mengeksekusi permintaan POST (POST request) yang berisi kode XSS.

Baru-baru ini metode serangan XSS telah menggunakan metode TRACE, serangan ini memanfatkan activeX, dimana informasi tentang user dikirimkan melalui TRACE request melalui browser klien, informasi ini juga dapat diubah ke dalam bentuk XML, sehingga klien dan situs web tersebut tidak menyadarinya. Metode ini mampu melewati keamanan HTTP Only.

d. Eksploitasi

Setelah melakukan serangan, penyerang dapat melakukan eksploitasi atas situs web yang menjadi target. Umumnya dilakukan dengan membajak sesi koneksi dari korban dan melakukan aksi seperti layaknya si korban mengakses situs web target. Kerugian yang muncul dapat merugikan situs web itu sendiri ataupun korban. Setelah eksploitasi ini, sesi serangan XSS dapat dikatakan telah sempurna.

e. Deteksi & Pencegahan serangan

1. **Deteksi kerawanan**

Mendeteksi apakah suatu aplikasi rawan terhadap adanya penyisipan kode adalah persoalan yang tidak sulit. Deteksi ini dapat dilakukan untuk menemukan titik rawan serangan pada aplikasi yang akan di-upload ke server. Kuncinya terletak pada analisis terhadap konten HTML dinamis pada sisi klien. Untuk bisa mendeteksi titik rawan tersebut, dibuat suatu skenario melalui proses-proses berikut ini :

1.1 Untuk setiap field input yang terlihat (yang mungkin terletak dalam suatu form HTML atau direpresentasikan dalam URL sebagai variabel), gunakan format script berikut ini :

<script>alert('CSS Vulnerable')</script>

<img csstest=javascript:alert('CSS Vulnerable')>

&{alert('CSS Vulnerable')};

1.2 Jika cek script di atas menyebabkan HTML menampilkan halaman yang tidak benar, komponen aplikasi masih dalam kondisi rawan.

1.3 Untuk setiap variabel yang terlihat, kirim atau ganti string berikut ini : **';!–"<CSS_Check>=&{()}** (Note that the string begins with two single-quotes) pada halaman hasil, cari string "<CSS_Check>". Jika anda menemukan "<CSS_Check>", masih ada kemungkinan komponen aplikasi dalam kondisi rawan. Namun demikian, jika kata CSS_Check tidak lagi diapit dengan %ltCSS_Check%gt, berarti sudah tidak rawan lagi. Jika input ditanmpilkan pada sembarang posisi dalam suatu dokumen, input akan dapat digunakan untuk mengalihkan aliran eksekusi ke penyerang.

1.4 Setelah mendapatkan lokasi word CSS_Check, verifikasi apakah (jika ada) karakter lain yang ditambahkan atau difilter dari string asal "**';!–"<CSS_Check>= &{()}**". Berdasarkan karakter yang terfilter, komponen aplikasi masih dikatakan rawan.

1.5 Melihat lebih detail hasil kode HTML, identikasikan string khusus yang diperlukan oleh penyerang untuk memisahkan tag HTML atau urutan kode. Jika ditemukan karakter yang dimaksud, tanpa mengaktifkan filter, diamati respon terhadap test string seperti pada point 3), sehingga dapat dikatakan komponen aplikasi yang ada masih dalam kondisi rawan.

1.6 Mengulangi proses-proses di atas untuk semua field-field yang tidak dapat diedit secara normal di sisi klien. Server Proxy memberikan fasilitas untuk editing request HTTP setelah request tersebut meninggalkan aplikasi pada klien sebelum akhirnya dikirim ke aplikasi server.

1.7 Pada banyak kasus, data akan dikirimkan melalui HTTP GET request. Melalui suatu investigasi, perlu juga dicari kerawanan pada komponen aplikasi yang memerlukan perintah HTTP POST untuk mengirim data.

2. **Pencegahan terhadap serangan**

Sebab tidak ada dua aplikasi yang identik, pengembang aplikasi perlu menyelaraskan antisipasi pengamanannya sebagaimana ditetapkan oleh kebutuhan bisnis. Kunci dalam mencegah aplikasi yang rawan terhadap injeksi kode dan serangan tipikal CSS ialah dengan memastikan bahwa halaman yang membangkitkan konten secara dinamis tidak mengandung tag yang tidak diinginkan.

Sumber-sumber yang umumnya dijadikan titik masuk antara lain :

a. Query string
b. URL’s dan bagian universal locator
c. Data yang dikirimkan
d. Cookies
e. Data persisten yang dipasok oleh user dan ditemu-kembali saat waktu setelahnya.

Solusi untuk pencegahan adanya serangan ini secara garis besar dapat disediakan pada sisi klien dan pada sisi server

Antisipasi atas interaksi dengan sisi klien, solusi yang ditawarkan dapat berupa :

1. Filtering

Menapis masukan dari klien browser dengan mewaspadai karakter-karakter khusus. Karakter-karakter khusus yang harus diwaspadai (menurut CERT) :

a) terletak di dalam isi dari element blok level

i. "<" : pembuka sebuah tag
ii. "&" : penanda sebuah entitas
iii. ">" : penutup sebuah tag

b) terletak di dalam nilai suatu atribut

i. di dalam nilai suatu atribut, diapit tanda petik ganda
ii. di dalam nilai suatu atribut, diapit dengan petik tunggal
iii. di dalam nilai suatu atirbut, tanpa diapit tanda petik
iv. "&" dengan conjunction

c) terletak pada URL

i. spasi, tabulasi dan pindah baris sebagai penanda akhir dari suatu URL
ii. "&" menandai suatu entitas, atau batas parameter CGI
iii. Karakter yang bukan ASCII : tidak ada karakter non-ASCII di URL
iv. "%" :

d) Terletak di antara <SCRIPT> dengan </SCRIPT> : titik koma, tanda kurung, kurung kurawal dan ganti garis.
e) Terletak di dalam server side script yang akan mengkonversikan tanda seru di input menjadi tanda petik ganda di output

2. Encoding

Selain menggunakan filtering, melakukan pengkodean karakter yang dinilai membahayakan menjadi karakter yang bisa diabaikan merupakan tujuan dari encoding. Dibandingkan dengan filtering, encoding juga memiliki keuntungan, yaitu tidak mengakibatkan hilangnya data.

Pengkodean dilakukan pada saat data disampaikan kembali pada user. Sebagai contoh, karakter "<" akan dikodekan sebagai karakter "&lt", meskipun akan nampak oleh user sebagai karakter "<" tetapi tidak akan diterjemahkan oleh aplikasi klien sebagai penanda awal dari tag HTML.

3. Validasi

Teknik ini dilakukan untuk menjamin hanya input yang tepat yang akan dipilih. .

Sedangkan solusi yang diterapkan pada interaksi di sisi server adalah sebagai berikut :

a. Membatasi respon server

Dalam banyak kasus, dimungkinkan untuk membatasi jumlah personalised data" yang dikirim ke browser klien melalui penggunaan respon. Sebagai contoh, sebuah situs yang menampilkan sapaan pada user saat pertama kali login sebagai response atas http://trusted.org/greeting.jsp?name=User, demi pertimbangan keamanan, perlu untuk mengorbankan respon dinamis seperti http://trusted.org/greeting.jsp?name=User ke dalam bentuk "hard code".

b. Membatasi panjang respon string

Bagi aplikasi utama, pengembang seharusnya dapat membatasi panjang maksimum string yang dipasok oleh user, walaupun awalnya dipaksakan di sisi klien, pasokan string tersebut juga harus diperiksa di sisi server, dimana

sangat memungkinkan ditegakkannya batas maksimum panjang string yang dilewatkan pada satu response.

c. HTTP referer

Memanfaatkan standar HTTP dengan menggunakan bagian *field header* "referer", ketika klien browser mengikuti sebuah link atau mengirimkan data. Isi referer harus berisi URL dari halaman asal data, ini memungkinkan aplikasi web untuk memeriksa field referer dan menolak bila bukan berasal dari asal host yang dibenarkan.

3.1.3. Analisis Metode Pertahanan

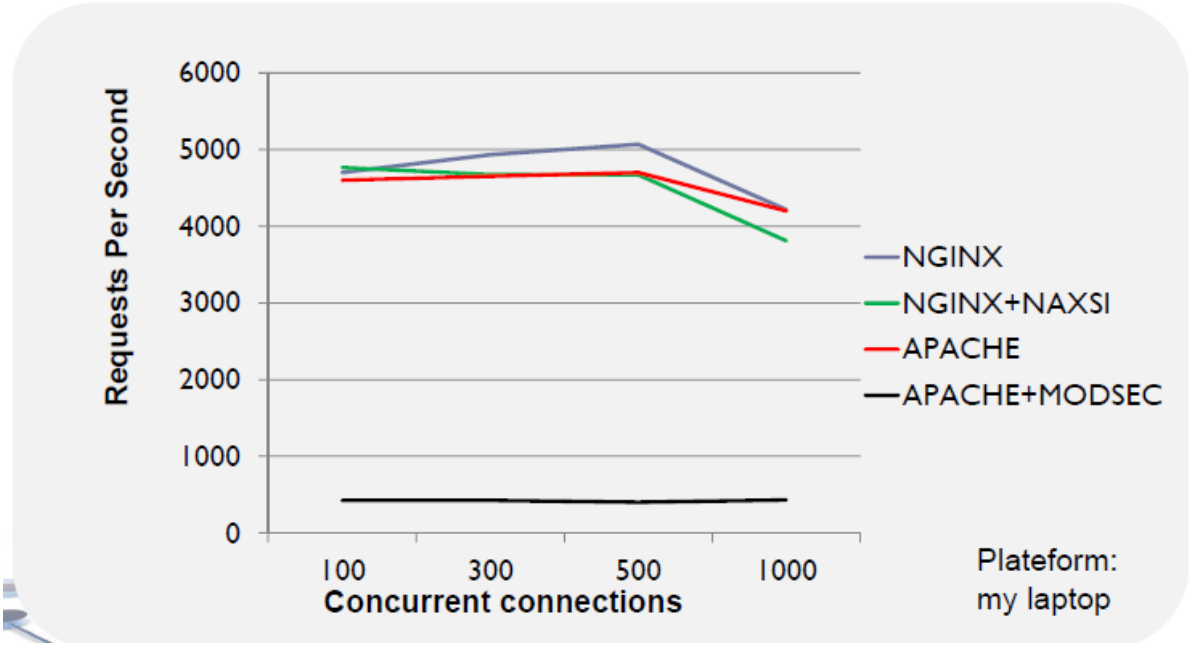
***Web application firewall*:** *Firewall* aplikasi web adalah alat *hardware server plug-in* , atau saringan perangkat lunak lain yang akan menerapkan serangkaian aturan untuk percakapan HTTP . Aturan semacam itu umumnya disesuaikan untuk aplikasi sehingga banyak serangan dapat diidentifikasi dan diblokir. Ada 2 jenis *web application firewall* yang akan dijelaskan disini yaitu Nginx dengan *web application firewall*-nya yaitu naxsi dan apache dengan *web application firewall*-nya yaitu Mod_Security.

a. Naxsi (Nginx Anti Xss Sql Injection)

Naxsi merupakan web server yang berbasis open source, memiliki kinerja yang tinggi, dan memiliki aturan pemeliharaan yang rendah. Naxsi adalah *Web Application Firewall* untuk Nginx. Naxsi bertujuan untuk mengamankan aplikasi web dari serangan-serangan seperti *SQL Injection, Cross Site Scripting, Cross Site Request Forgery, Local & Remote file inclusions*.

Perbedaan Naxsi dengan sebagian WAF (Web Application Firewall) yang lain adalah Naxsi tidak bergantung pada tanda-tanda untuk mendeteksi dan memblokir serangan. Naxsi menggunakan model sederhana untuk mendeteksi serangan, yaitu dengan cara mendeteksi karakter tidak dikenal yang ada di dalam HTTP *request*/argumen.

Setiap jenis karakter yang tidak biasa akan meningkatkan nilai *request*. Jika request mencapai nilai paling tinggi, permintaan akan ditolak, dan pengguna akan diarahkan ke sebuah halaman "Forbidden". Performa Naxsi ditunjukan dengan diagram berikut ini:



Gambar 3.5 Performa Web Application Firewall

b. Mod_Security

Mod Security adalah *web application firewall* yang dapat bekerja baik sebagai *reverse proxy*. *Mod Security* memberikan perlindungan dari berbagai serangan terhadap aplikasi web dan memungkinkan untuk pemantauan lalu lintas HTTP, *logging* dan analisis *real-time*. *Mod Security* umumnya menggunakan model keamanan negatif dan memiliki beberapa fitur yang dapat membantu meningkatkan keamanan.

Mod Security Core Rules Set adalah kumpulan aturan yang akan mendeteksi serangan web. Pendeteksian serangan dilakukan dengan 2 mode yaitu dengan mode deteksi tradisional (*traditional detection mode*) dan mode deteksi nilai anomaly (*anomaly score detection mode*). kedua mode tersebut memiliki deteksi serangan yang berbeda.

1. Mode Deteksi Tradisional

   Mode deteksi tradisional (IDS/IPS Mode) adalah mode operasi default terbaru dalam Mod Security. Mode ini merupakan mode paling dasar dalam pendeteksian serangan dimana semua aturan bersifat “mandiri”. Hal tersebut dapat diartikan bahwa mode ini memiliki aturan yang tidak dapat dibagikan terhadap aturan lain sehingga setiap aturan tidak memiliki informasi terhadap aturan lain.

   Cara kerja aturan ini sangat sederhana, jika *pattern* serangan dan *pattern* CRS cocok maka akan ditolak dan data serangan akan disimpan di file apache error_log dan Mod Security audit_log.

   a. Keunggulan mode deteksi tradisional

      Penggunaan mode ini lebih mudah dipahami dan memiliki performa yang baik dalam penghentian serangan.

   b. Kekurangan mode deteksi tradisional

      Sangat sulit dalam merubah aturan yang lebih kompleks.

2. Mode Deteksi Nilai Anomali

   Mode deteksi nilai anomali menggabungkan konsep *collaborative detection* dan *delayed blocking*. Hal ini mambuat mode ini tidak akan langsung menghentikan serangan tetapi mode ini akan mengalihkan setiap serangan yang cocok terhadap aturan pada nilai anomali. Jika nilai anomaly mencapai batas yang ditentukkan oleh aturan maka serangan tersebut akan di hentikan dan jika tidak serangan tersebut akan menghasilkan peringatan saja.

   a. Keunggulan mode deteksi nilai anomali

      Meningkatkan penghentian ancaman serangan dan memperbolehkan pengguna mengatur batas nilai untuk penghentian serangan.

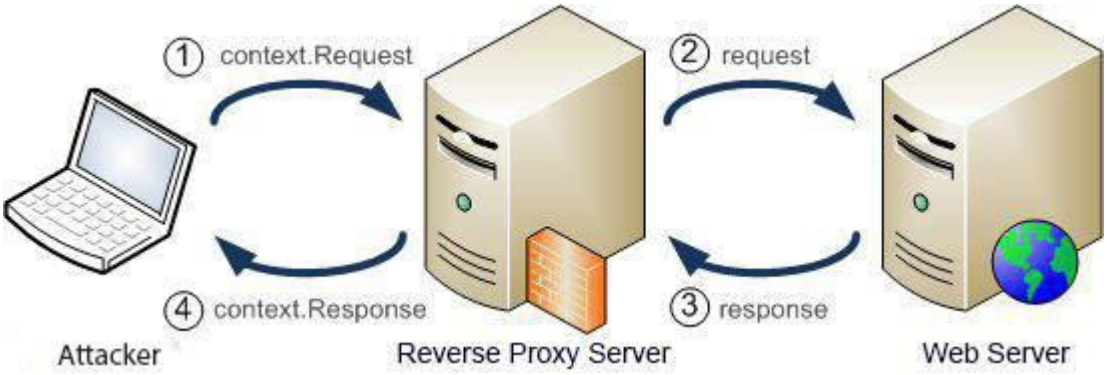
   b. Kekurangan mode deteksi nilai anomali

      Mode ini lebih kompleks dibandingkan mode tradisonal dan mode ini harus memperbaharui monitoring log untuk dianalisis

3.2. Implementasi Mod Security

3.2.1.Skema Akses Web

Xecureit menggunakan *reverse proxy* untuk menghubungkan pengakses data ke server, untuk mengakses data-data web terdapat pada server. Sehingga ketika ada *maintenance* di luar data website tidak akan mengganggu *performance* pada website tersebut. Berikut adalah skema yang akan dilalui oleh para pengakses website termasuk penyerang website (*attacker*):



Gambar 3.6. Skema akses website www.xecureit.com

Sistem operasi yang digunakan dari ketiga mesin di atas adalah sebagai berikut:

a. *Attacker* = Windows 7
b. Reverse Proxy Server (*firewall*) = Ubuntu Desktop 13.04
c. Web Server = Turnkey Lamp 12.0

3.2.2.Kebutuhan Sistem

a. Pada mesin *attacker* dan *firewall* harus di pasang web server apache versi 2.0 atau di atasnya
b. PHP5
c. Modul-modul yang ada pada apache seperti :
    a. mod_uniqueid
    b. libapr and libapr-util
    c. libpcre
    d. libxml2

3.2.3.Instalasi Mod Security

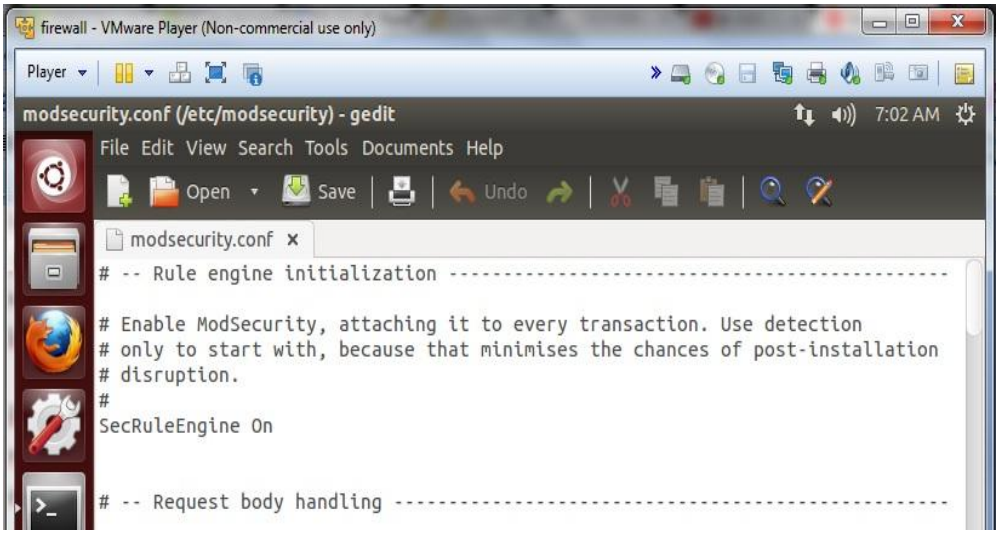
Mod Security dapat digunakan secara *embeded program* atau dengan cara *reverse proxy* pada apache minimal versi 2.0. Berikut langkah-langkah dalam instalasi mod security dalam *reverse proxy*:

a. install beberapa *library* dengan *command* berikut ini:
   sudo apt-get install libxml2 libxml2-dev libxml2-utils
   sudo apt-get install libaprutil1 libaprutil1-dev

b. kemudian instal *library* untuk *apache* dengan *command* berikut:
   sudo apt-get install libapache-mod-security

c. setelah instalasi berhasil hal selanjutnya yang harus di lakukan adalah konfigurasi mod_security agar dapat menggunakan *rules*. Instalasi *rules mod security* dengan mengetik *command* di bawah ini:
   Sudo mv /etc/modsecurity/modsecurity.conf-recommended/etc/modsecurity/ modsecurity.conf
   sudo gedit /etc/modsecurity/modsecurity.conf

ubah konfigurasi ***SecRuleEngine*** menjadi ***On*** seperti berikut



Gambar 3.7 konfigurasi Mod Security

d. Instalasi Rules Set

Mod Security membutuhkan *rules* atau aturan untuk medeskripsikan tipe-tipe serangan yang dapat ditangani oleh *firewall* sehingga terhindar dari serangan-serangan yang telah di definisikan sebelumnya pada *rules* yang telah diaktifkan.

Berikut adalah langkah-langkah instalasi *rules set* OWASP pada *Mod Security* :

cd /tmp

sudo wget https://github.com/root25/MODSEC/raw/master/modsecurity-crs_2.2.5.tar.gz

sudo tar -zxvf modsecurity-crs_2.2.5.tar.gz

sudo cp -R modsecurity-crs_2.2.5/ /etc/modsecurity/ sudo rm modsecurity-crs_2.2.5.tar.gz*

sudo rm -R modsecurity-crs_2.2.5

sudo mv /etc/modsecurity/modsecurity_crs_10_setup.conf.example /etc/modsecurity/modsecurity_crs_10_setup.conf

Setelah langkah-langkah diatas dilakukan selanjutnya adalah mengaktifkan *rules* yang telah diinstal dengan langkah-langkah di bawah ini:

cd /etc/modsecurity/base_rules

*for f in * ; do*

sudo ln -s /etc/modsecurity/base_rules/$f /etc/modsecurity/activated_rules/$f ; done

cd /etc/modsecurity/optional_rules

*for f in * ; do*

sudo ln -s /etc/modsecurity/optional_rules/$f

/etc/modsecurity/activated_rules/$f ; done

Sekarang tambahkan *script* untuk melakukan pengaktifan *rules*:

sudo gedit /etc/apache2/mods-available/mod-security.conf

Tambahkan *script* di bawah ini sebelum tulisan </IfModule> dan simpan fille:

Include "/etc/modsecurity/activated_rules/.conf"*

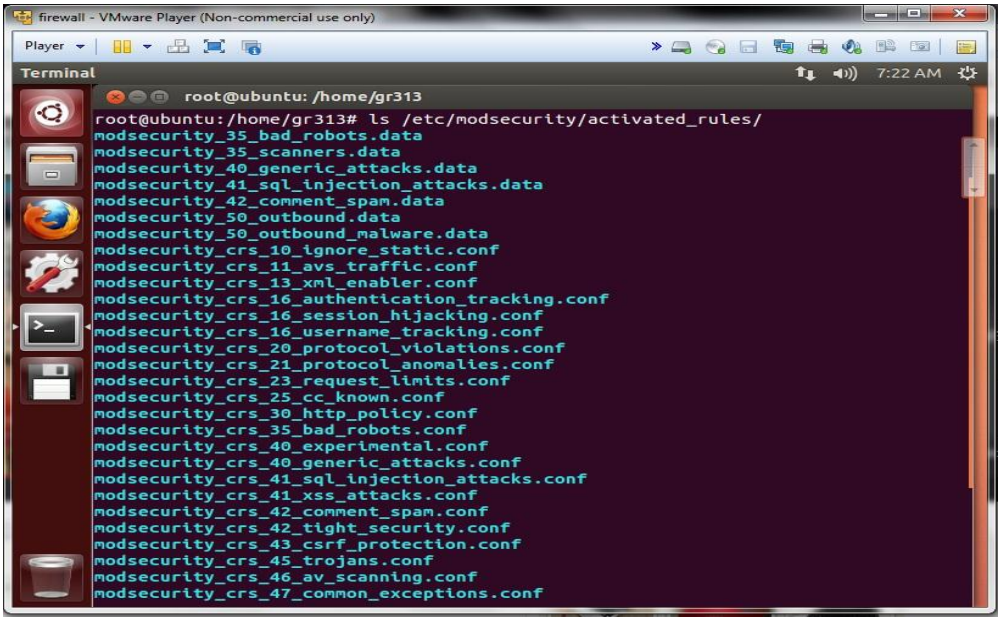
enable headers module dengan *script* berikut:

sudo a2enmod headers

Setelah langkah-langkah diatas di lakukan ModSecurity dan Rules-nya telah berhasil diinstal dalam keadaan *default*. Keadaan *default* diartikan bahwa pengaturan keamanan *firewall* yang ada pada modsecurity telah di atur oleh *system*.

Untuk mengetahui rules apa saja yang telah aktif pada *mod security* lakukan *command* berikut ini

Sudo ls /etc/modsecurity/activated_rules/

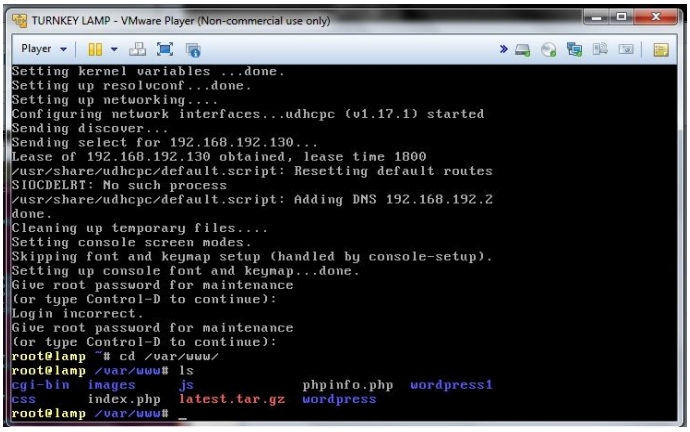


Gambar 3.8 Daftar rules set yang terinstal

4. Konfigurasi Reverse Proxy

a. Upload Data Website ke Server

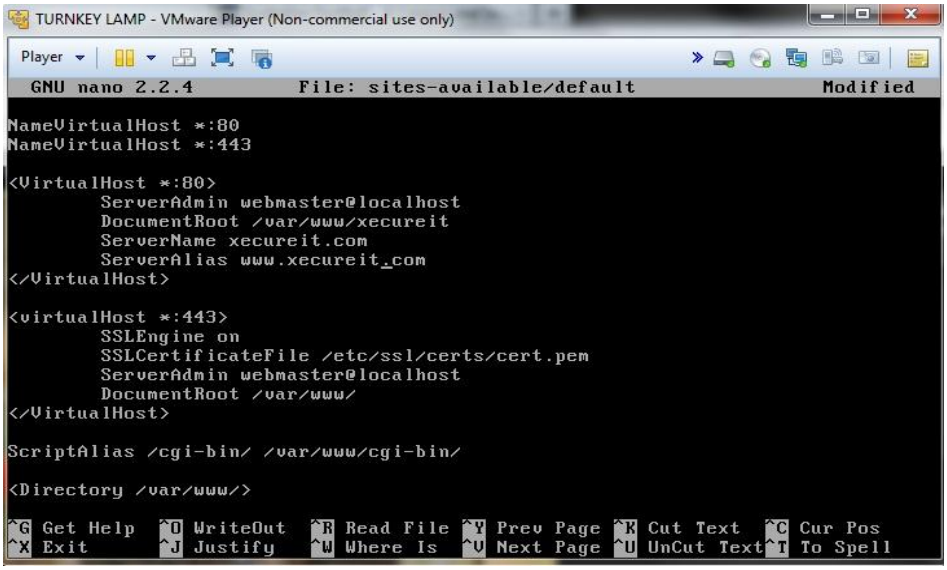
masukkan data website pada localhost yang telah ada di server



Gambar 3.9 Tampilan localhost pada server

b. Konfigurasi Situs di Server

setelah data website ada selanjutnya adalah mengecek situs yang aktif pada server dengan cara menuliskan alamat website dan path yang ditujukan pada folder data website yang telah di upload. Dari konfigurasi ini digunakan nama website www.xecureit.com dan menggunakan port 80 untuk mengakses website tersebut.



Gambar 3.10 Tampilan konfigurasi situs yang aktif pada server

c. Konfigurasi *Reverse Proxy*

konfigurasi yang ada pada server dilakukan agar website yang berada di localhost server bisa di akses oleh firewall dan user. Dengan tujuan itu firewall harus tersambung lebih dulu pada server agar skema pada gambar 3. Dapat berjalan tanpa ada kesalahan.

Konfigurasi yang akan dilakukan pada firewall hampir sama dengan konfigurasi yang telah di lakukan pada server. Dengan mendeklarasikan nama website yang telah dimasukkan pada konfigurasi pada server sebelumnya, seperti pada gambar dibawah ini.

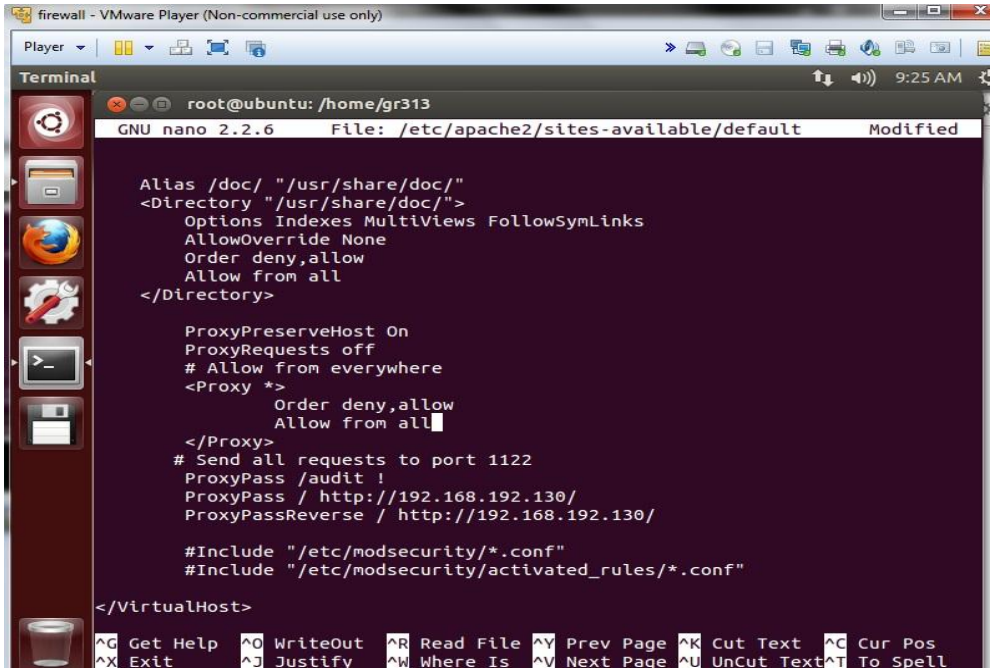
```
<VirtualHost *:80>
        ServerAdmin administrator@localhost
        ServerName xecureit.com
        ServerAlias     www.xecureit.com

        DocumentRoot /var/www

        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
        </Directory>
```

Gambar 3.11 Konfigurasi website pada *firewall*

Konfigurasi diatas dilakukan untuk menyamakan nama website pada server dan *firewall* sehingga kedua mesin tersebut dapat terhubung. Namun dengan menyamakan nama website saja tidak cukup karena *firewall* masih belum mengenal alamat (IP Address) mesin server sehingga *firewall* harus mendeskripsikan alamat server yang dituju dimana data website tersimpan.



Gambar 3.12 Konfigurasi *reverse proxy*

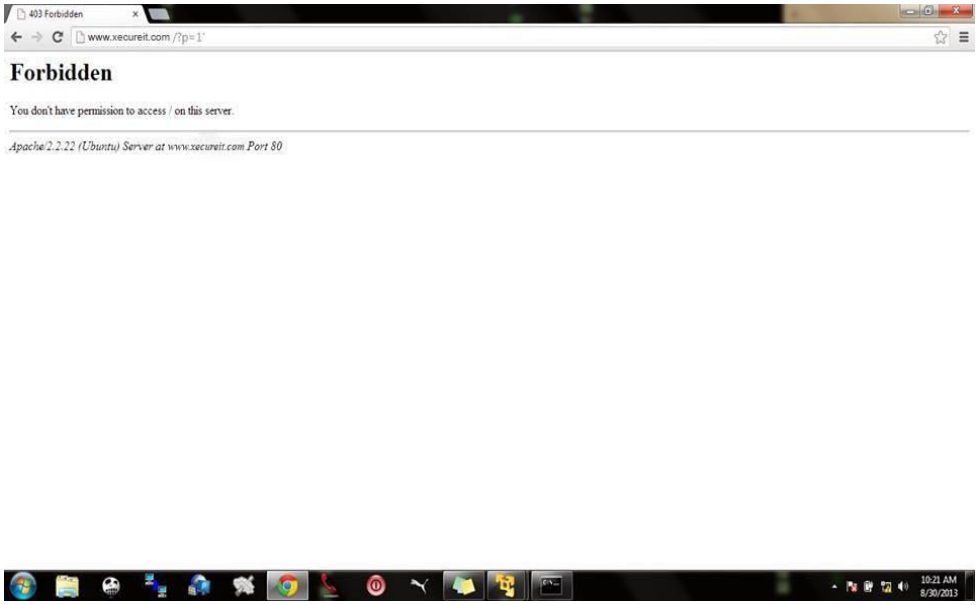
Dari gambar diatas diketahui bahwa alamat server yang dituju oleh *reverse proxy* ini adalah 192.168.192.130. Alamat IP tersebut merupakan alamat IP yang dimiliki oleh server.

Setelah konfigurasi di atas selesai maka akses yang dilakukan oleh user dapat di lakukan ke mesin firewall kemudian mesin *firewall* melanjutkan akses tersebut pada server sehingga website dapat diakses oleh para *user*.

5. Testing

Firewall yang telah terpasang dapat melindungi website dari serangan-serangan seperti *SQL injection* maupun *XSS attacking*. *Web Application Firewall* akan terbukti ampuh jika *firewall* bisa mengatasi serangan.

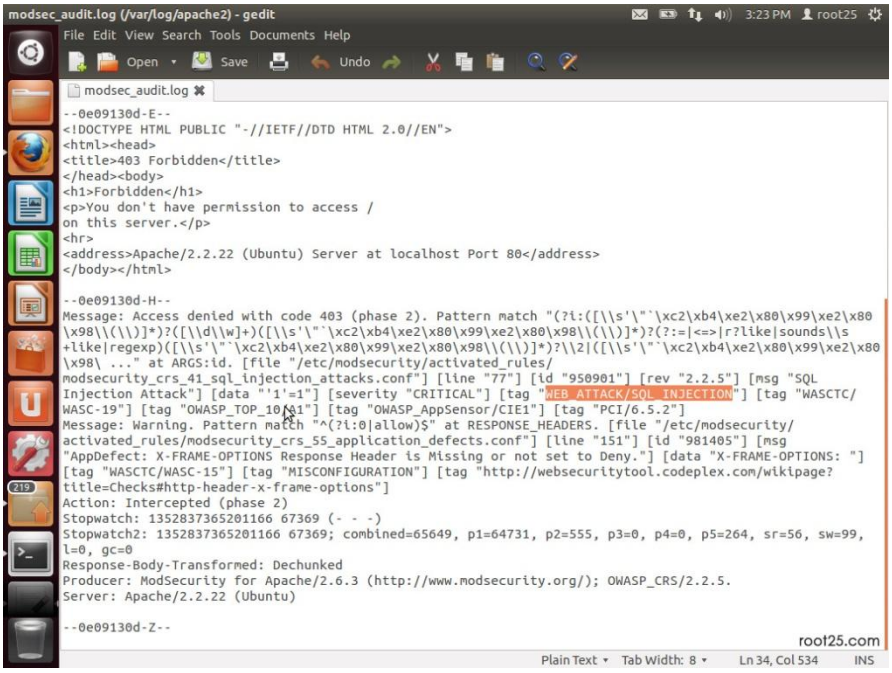
Mod Security yang telah di pasang akan di uji oleh *SQL injection*, dengan cara menyerang website yang memiliki ID pada *database* dan tidak di *filter*. Serangan dilakukan dengan menambahkan tanda (‘) setelah alamat ID website.Setelah melakukan serangan tampilan yang akan muncul adalah sebagai berikut :



Gambar 3.13 Hasil testing *SQL injection* pada website

Serangan dapat di *block* oleh Mod Security sehingga muncul tampilan "Forbidden". Dari serangan tersebut Mod Security menyimpan log serangan di sebuah file yang ada di mesin *firewall.* Untuk melihat log tersebut dapat dilakukan dengan mengetikkan *command* berikut ini:

Sudo gedit /var/log/apache2/modsec_audit.log



Gambar 3.14 Tampilan Log Mod Security

Dari log diatas diketahui bahwa serangan yang dilakukan merupakan serangan dengan tipe *SQL injection* dan diketahui pula data-data akses pada website.