ANCAMAN TERHADAP KEAMANAN INFORMASI OLEH SERANGAN *CROSS-SITE SCRIPTING (XSS)* DAN METODE PENCEGAHANNYA

Suroto Suroto[1)], Asman Asman[2)]
suroto@univbatam.ac.id[1)], asmantroops05@gmail.com[2)]
Teknik Sistem Informasi. Fakultas Teknik, Universitas Batam
Jl. Uniba No 5, Batam Center, Kota Batam, 29432, Indonesia

Abstract

Information stored on websites or the internet makes it easier for users to exchange information. Of course, only authorized users have access to this information. The amount of information that is spread and stored on the internet invites people who are not entitled to get that information. Various methods are used to obtain access rights and information. One of them is to carry out an attack called Cross Site Scripting Attack. Websites become the target of this type of attack. In this paper, an explanation of what is meant by a Cross Site Scripting attack, how this attack is carried out, and ways to prevent it. Knowledge of this type of attack is indispensable for developers of Web-based information systems. Because this attack works by exploiting programming errors or writing program code in an information system.

Keyword: *CSS Attack, Cross-site Scripting, XSS, Cyber Security*

Pendahuluan

Website merupakan salah satu sarana untuk menyimpan dan menyebarkan informasi. Pengguna yang berhak dapat memperoleh data dan informasi yang ada didalamnya. Umumnya, data dan informasi tersimpan dalam sebuah database server. Website hanyalah merupakan front end, sedangkan sebagai back end nya adalah database server.

Banyaknya data dan informasi yang tersebar di Internet / website mengundang pengguna internet yang tidak memiliki hak akses untuk mendapatkan data tersebut. Tentunya secara ilegal pula. Salah satu cara yang dilakukan adalah melakukan serangan *Cross Site Scripting.*

Tujuan dari paper ini adalah untuk mengkaji pengetahuan tentang Cross Site Scripting Attack, cara kerja serangan Cross Site Scripting (XSS), evaluasi kerentanan website dan cara pencegahan terhadap serangan XSS.

Tinjauan Teori

Cross Site Scripting

Cross-site scripting (XSS) merupakan sebuah eksploitasi keamanan dimana penyerang menyisipkan kode berbahaya (biasanya Javascript) di sisi klien ke suatu halaman web. Ketika kode berbahaya ini bersama dengan halaman web asli akan ditampilkan dalam klien web (browser seperti IE, Mozilla dll), memungkinkan Hacker untuk mendapatkan akses yang lebih besar dari halaman tersebut. Jenis serangan keamanan komputer ini telah ada sejak tahun 1990-an. Banyak website telah dipengaruhi oleh kelemahan *cross-site scripting (XSS)* di beberapa titik . Serangan eksploitasi kerentanan XSS dapat mencuri data, mengendalikan sesi pengguna, menjalankan kode berbahaya, atau digunakan sebagai bagian dari *phishing scam*.

Banyak yang berpikir Web 2.0 telah menciptakan putaran terbaru dari serangan XSS. Pada kenyataannya mereka hanya variasi pada tema lama. Sementara itu, teknologi Ajax (*asynchronous JavaScript dan XML*) mengubah lanskap ancaman bahwa mereka memungkinkan penyerang untuk mengeksploitasi kerentanan *cross-site scripting* dalam cara yang lebih rahasia. Aplikasi Ajax cenderung sangat kompleks, lebih banyak interaksi antara browser dan server, bahkan sebuah halaman dapat menarik konten dari situs lain. Pengaturan ini membuat sulit untuk menguji banyak kemungkinan permutasi dari pengguna dan

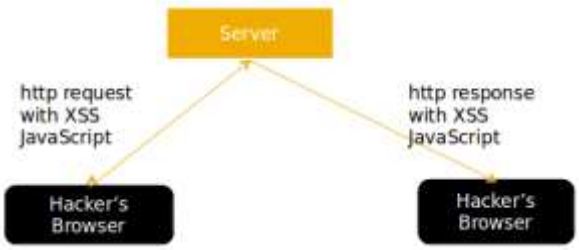
interaksi layanan, memungkinkan kerentanan lama, seperti kelemahan XSS.
Situs terus menjadi mangsa serangan XSS karena sebagian harus interaktif, menerima dan mengembalikan data dari pengguna. Ini berarti penyerang, juga dapat berinteraksi langsung dengan proses aplikasi ini, melewati data yang dirancang untuk menyamar permintaan aplikasi sebagai yang sah atau perintah melalui saluran permintaan yang normal seperti script, URL dan data formulir. Komunikasi ini pada lapisan aplikasi dapat mengeksploitasi aplikasi yang ditulis secara buruk dengan cara bypass pertahanan keamanan perimeter tradisional.

Cara Kerja Serangan XSS

Serangan cross-site scripting berbeda dari kebanyakan serangan lapisan aplikasi, seperti *SQL injection*, karena mereka menyerang pengguna aplikasi ini, bukan aplikasi atau server. Web server mendapat data dari web client (*POST, GET, COOKIES* dan lain lain) dengan sebuah permintaan. Sehingga malicious user dapat menyerang dengan menyuntikkan kode (biasanya script sisi klien seperti JavaScript) ke dalam output aplikasi Web. Misal, sebuah potongan kode ini:

<script>alert ('this site has been hacked') ;</script>

Kebanyakan website memiliki banyak titik injeksi, seperti *search fields, feedback forms, cookies* dan forum yang rentan terhadap cross-site scripting.



Gambar 1. Cara Kerja Secara Umum XSS Attack

Tujuan yang paling umum dari serangan XSS adalah untuk mengumpulkan data cookie, cookie biasanya dan secara teratur digunakan secara tidak benar untuk menyimpan informasi seperti ID sesi, preferensi pengguna atau informasi login. Meskipun script sisi klien tidak dapat secara langsung mempengaruhi informasi server-side, mereka masih bisa kompromi keamanan sebuah situs, sering menggunakan manipulasi *Document Object Model* untuk mengubah nilai-nilai bentuk atau beralih bentuk tindakan untuk mengirim data yang diajukan ke situs penyerang.

Tipe Serangan XSS

Terdapat 3 jenis model serangan XSS yang dilakukan oleh malicious user, yaitu :

i) Persistent
ii) Replected XSS
iii) DOM Based XSS

Persistent.

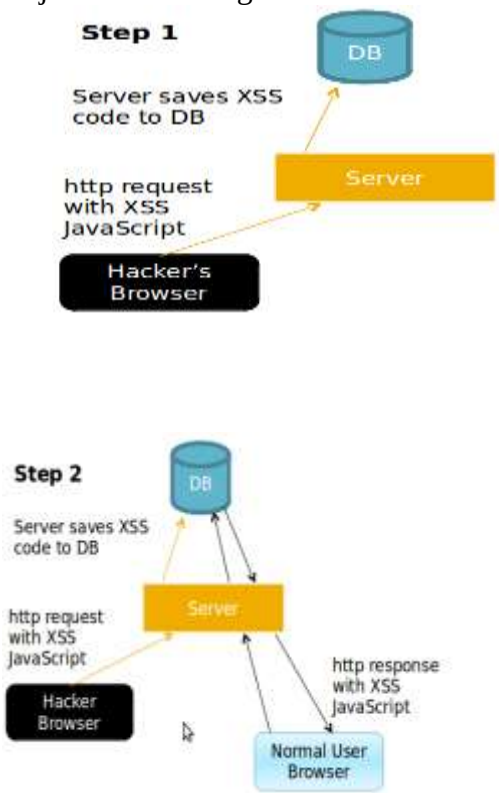
Dalam jenis persisten serangan XSS, kode XSS akan disimpan ke dalam penyimpanan persisten seperti database dengan data lain dan kemudian terlihat oleh pengguna lain juga.
Salah satu contoh, papan pesan klub sepak bola XYZ memungkinkan anggota klub untuk menulis komentar tentang tim dan kinerjanya. Komentar disimpan dalam sebuah database online dan ditampilkan kepada anggota lain tanpa pernah divalidasi atau dikodekan. Seorang anggota *malicious* dapat memposting komentar yang mengandung script tertutup <script> tag. Penyerang kemudian menunggu anggota lain untuk melihat komentar. Karena teks di dalam tag <script> umumnya tidak ditampilkan, anggota lainnya bahkan mungkin tidak menyadari bahwa script telah dijalankan, hanya melihat komentar akan mengeksekusi script.
Jenis serangan ini lebih rentan, karena Hacker dapat mencuri cookies dan dapat melakukan modifikasi pada halaman. Risiko dengan jenis serangan ini adalah setiap hacker dapat menggunakan celah ini untuk melakukan beberapa tindakan atas nama pengguna lain. Berikut contoh *script injections* yang dimasukkan dalam form komentar:

abc<script>window.location =
"http://www.hackers.com?yid=" +

document.cookie; </script>

Gambar 2: Injeksi Script Dalam Form Komentar

Sistem kerja jenis serangan XSS ini dapat disajikan dalam diagram berikut ini :

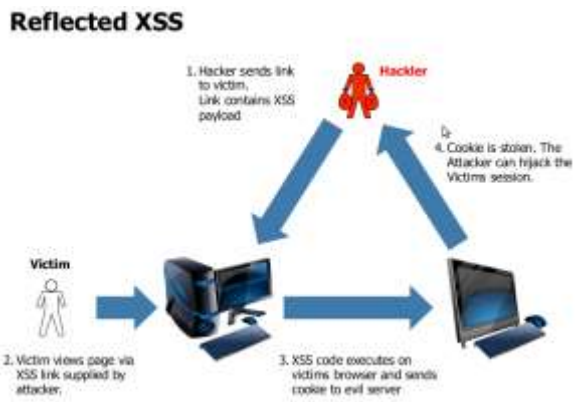


Gambar 3: Sistem Kerja Serangan XSS Tipe Persistent

Replected XSS.

Kode XSS hanya akan ditampilkan di halaman berikutnya untuk pengguna yang sama dan tidak akan disimpan ke dalam penyimpanan persisten seperti basis data. Jenis serangan ini kurang rentan, karena Hacker hanya dapat melihat cookie mereka sendiri dan dapat melakukan modifikasi di halaman yang saat ini dibuka mereka sendiri. Risiko dengan jenis lubang XSS ini adalah bahwa hal itu membuka jalan bagi *Cross Site Request Forgery* (CSRF). CSRF memungkinkan hacker untuk menempatkan beberapa link.

Jenis serangan XSS ini bekerja bahkan jika situs tersebut dilihat melalui koneksi SSL, karena script dijalankan dalam konteks situs "aman", dan browser tidak bisa membedakan antara konten sah dan berbahaya yang disajikan oleh aplikasi Web. Tapi penyerang tidak harus bergantung pada suntikan kode mereka ke situs Halaman komentar. Mereka dapat mengelabui korban agar mengklik URL dalam email *phishing*, yang kemudian menyuntikkan kode ke halaman yang dilihat, memberikan penyerang akses penuh ke konten halaman ini. Berikut diagram kerja serangan *Replected XSS:*



Gambar 4. Sistem Kerja Serangan Replected XSS

Encoding URL sering digunakan dalam serangan tersebut untuk menyamarkan link dan membuat pengguna lebih cenderung untuk mengikutinya. Pada contoh di bawah ini, link tersebut ke URL https aman untuk situs terpercaya.

https://www.userstrustedbank.com/script/LoginServlet?function=”><script>document.write(String.fromCharCode(60,105,102,114,97,

109,
101,32,115,114,99,61,104,
116,116,112,58,47,47,
119,119,119,46,97,98,97,
100,98,97,110,107,
46,99,111,109,47,108,
111,103,105,110,32,112,104,112,62))</script>

Pengguna melihat bahwa link tersebut untuk www.userstrustedbank.com dan melalui koneksi SSL; terlihat cukup asli karena link sering memiliki panjang, teks yang tampaknya tidak berarti di akhir. pengguna mengklik link. Namun, kode antara <script> tag ketika diterjemahkan oleh browser berbunyi:
<iframe
src=http://www.abadbank.com/login.php>

String Serangan ini membuat IFRAME dalam konteks situs yang sebenarnya *usertrustedbank*.
Halaman login.php penyerang akan terlihat persis seperti halaman login usertrustedbank, menipu pengguna untuk memasuk dan mengirim nama login dan password ke server bank yang salah, sumber IFRAME, sementara sepanjang waktu berada di situs userstrustedbank.com nyata.

DOM Based
XSS Berbasis DOM (atau jenis-0 XSS) adalah serangan XSS dimana payload serangan dijalankan sebagai akibat dari memodifikasi "lingkungan" DOM di browser korban yang digunakan oleh script sisi klien asli, sehingga kode sisi klien berjalan dalam sebuah cara "tak terduga". Artinya, halaman itu sendiri (respon HTTP tersebut) tidak berubah, tapi kode sisi klien yang terkandung dalam halaman mengeksekusi berbeda karena modifikasi berbahaya yang telah terjadi di lingkungan DOM.

Hal ini berbeda dengan serangan lain XSS (disimpan atau dipantulkan), dimana payload serangan ditempatkan di halaman respon (karena cacat sisi server).
var pos = document.URL.indexOf ("name =") + 5;
document.write (document.URL.substring (pos, document.URL.length));
...

http://www.vulnerable.site/welcome.html?name=Joe

Menguji Kerentanan XSS Injection

Kita dapat menentukan apakah sebuah aplikasi berbasis web rentan terhadap serangan XSS dengan sangat mudah. Sebuah tes yang sederhana adalah dengan mengambil parameter saat itu, yang dikirim dalam permintaan HTTP GET dan memodifikasinya. Ambil contoh permintaan berikut di URL bar alamat browser. URL ini akan mengambil parameter name yang kita masukkan dalam textbox dan cetak sesuatu di halaman. Seperti "Hello Suroto, terima kasih untuk datang ke situs saya"

http://www.yoursite.com/index.html?name=suroto

Dan memodifikasinya sehingga menambah ekstra beberapa informasi tambahan ke parameter. Misalnya mencoba memasuki sesuatu yang mirip dengan permintaan berikut di URL bar alamat browser.

http://www.yoursite.com/index.html?name=<script>alert('Ada kerentanan XSS di Website ini ') </ script>

Jika muncul kotak pesan peringatan yang menyatakan "Ada kerentanan XSS di Website ini", maka kita tahu parameter ini rentan terhadap serangan XSS. Parameter name tidak sedang memvalidasi, itu memungkinkan apa pun untuk diproses sebagai name, termasuk script berbahaya yang disuntikkan ke dalam parameter. Pada dasarnya apa yang terjadi adalah biasanya di mana nama Suroto akan dimasukkan pada halaman yang < / script> </ script> pesan bukannya sedang ditulis ke halaman dinamis. Pesan peringatan hanya merupakan contoh bagaimana untuk menguji kerentanan XSS. Seorang hacker jahat akan jauh lebih licik

untuk jenis kerentanan keamanan.

Implementasi Pencegahan Serangan XSS

Aplikasi web yang dikembangkan menggunakan beberapa bentuk siklus pengembangan keamanan (SDL). Tujuan mereka adalah untuk mengurangi jumlah desain yang berhubungan dengan keamanan dan kesalahan coding dalam aplikasi, dan mengurangi keparahan dari setiap kesalahan yang tidak terdeteksi.

Validasi dan Encode Input Dari User

Ketika mengembangkan aplikasi aman adalah dengan mengasumsikan bahwa semua data yang diterima oleh aplikasi tersebut dari sumber yang tidak dipercaya. Hal ini berlaku untuk setiap data yang diterima oleh aplikasi - data, cookies, email, file atau gambar - bahkan jika data dari pengguna yang telah login ke account mereka dan dikonfirmasi sendiri.

Tidak percaya input pengguna berarti memvalidasinya terhadap jenis, panjang, format dan jangkauan (range) setiap kali data yang melewati batas kepercayaan, misal, dari sebuah form Web hingga script aplikasi, dan kemudian meng-encodingnya sebelum menampilkan kembali di halaman dinamis. Dalam prakteknya, ini berarti bahwa kita perlu meninjau setiap titik di situs kita dimana data yang disediakan pengguna diproses dan memastikan bahwa, sebelum dilewatkan kembali ke pengguna, nilai-nilai yang diterima dari sisi klien diperiksa, disaring dan dikodekan.
Berikut contoh dalam DOJO framework

```
<label for="firstName">First Name:
</label>
<input type="text" id="firstName"
name="firstName"

        dojoType="dijit.form.ValidationTextB
ox"
        required="true"
        propercase="true"
        promptMessage="Enter first name."
        invalidMessage="First name is
required."
        trim="true"/><br>
```

Contoh Validasi lain pada inputan alamat email dari pengguna. Validasi tersebut dapat dilakukan seperti kode di bawah ini:

```
String email =
request.getParameter("email");
String expression =
"^\w+((-\w+)|(\.\w+))*\@[A-Za-z0-9]+((\.|-
)[A-Za-z0-9]+)*\.[A-Za-z0-9]+$";

Pattern pattern =
Pattern.compile(expression,Pattern.CASE_I
NSENSITIVE);
Matcher matcher = pattern.matcher(email);
if (macher.maches())
{
    out.println("Your email address is: " +
Encode.forHTML(email));

}
else
{
//log & throw a specific validation exception
and fail safely
}
```

Selain itu, kita dapat melakukan blacklist, misal, tolak <*script*> dan atribut lain seperti *onload, onclick, onmouseover* dan lain-lain.
Namun validasi yang dilakukan sisi klien (JavaScript), sebelum mengirimkan data ke server, hanya membantu dalam aspek kegunaan dari situs web. Hal ini tidak dapat memberikan keamanan yang sebenarnya, karena pengguna dapat menonaktifkan JavaScript.

Batasi charset.

Kita dapat menggunakan ekspresi reguler untuk mencari dan mengganti input pengguna untuk memastikan itu tidak berbahaya. Pembersihan ini dan validasi harus dilakukan pada semua data sebelum diteruskan ke proses lain.
Dengan sanitasi input data, kita dapat mencegah kode berbahaya untuk masuk dalam sistem.
Pemeriksaan tipe data yang tepat membantu dalam membersihkan data. Pertama-tama kita

harus membatasi data numerik untuk fields numerik dan karakter alfanumerik untuk fields teks. Misalnya, field nomor telepon tidak boleh menerima tanda baca selain kurung dan garis.

Kita juga perlu untuk mengkodekan karakter khusus seperti "<" dan ">" sebelum mereka ditampilkan kembali jika mereka diterima dari input pengguna. Misalnya, encoding tag script memastikan browser akan menampilkan <script> tapi tidak melaksanakannya. Dalam hubungannya dengan encoding, adalah penting bahwa halaman Web kita selalu menentukan set karakter sehingga browser tidak akan menafsirkan pengkodean karakter khusus dari set karakter lainnya.

Mengingat bahwa browser tidak dimaksudkan untuk mengasumsikan setiap nilai default untuk charset halaman, dan beberapa server tidak mengizinkan atau tidak dikonfigurasi untuk memungkinkan parameter charset dikirim. Penting bahwa kita tidak kehilangan tag meta ini keluar dari halaman Web kita. Ini akan sangat mengurangi jumlah kemungkinan bentuk injeksi skrip yang dapat dilakukan.

Jadi jika aplikasi Web kita tidak perlu menampilkan karakter diluar set karakter ISO-8859-1, yang cukup untuk bahasa Inggris dan kebanyakan Eropa, setiap halaman harus menggunakan meta tag berikut untuk menyatakan karakter-karakternya :

<META http-equiv="Content-Type" content="text/html; charset= ISO-8859-1">

Aplikasi web yang tidak perlu menerima data banyak (rich data) dapat menggunakan escaping untuk menghilangkan risiko XSS. Ada, tentu saja, saat ketika aplikasi harus menerima karakter HTML khusus, seperti "<" dan ">", misalnya di situs jejaring sosial di mana format font diterima. Pengamanan pengkodean masukan tersebut dapat menjadi rumit karena fleksibilitas dan kompleksitas HTML.

Penulis akan merekomendasikan untuk memanfaatkan sebuah *security encoding library*. Microsoft ASP.NET menyediakan server kontrol validasi yang dapat memvalidasi input pengguna. Aplikasi web yang berjalan pada server Apache atau menggunakan bahasa pemrograman Perl juga dapat menggunakan modul Apache :: TaintRequest atau PerlTaintcheck untuk mengotomatisasi proses penanganan data eksternal. Pengembang software harus memahami bagaimana untuk menggabungkan fitur-fitur keamanan tambahan ke dalam kode mereka.

Escaping Output

Salah satu teknik umum untuk mencegah kerentanan XSS adalah "escaping". Tujuan dari 'escaping' karakter dan string adalah untuk memastikan bahwa setiap bagian dari string ditafsirkan sebagai string primitif, bukan sebagai karakter kontrol atau kode. Beberapa karakter bermasalah diantaranya : <, >, ", ‘, \ dan &. Karakter ini dapat diganti dengan entitas karakter HTML. Contoh, '**&lt**;' merupakan karakter HTML encoding untuk karakter '<'. Jika kita meletakan script ini:

<script>alert('testing')</script>

dalam sebuah halaman HTML, maka script akan mengekskusi. Tetapi jika kita include seperti ini:

<script>alert('testing')</script>

dalam halaman HTML, ia akan menampilkan teks *"<script>alert('testing')</script>"*, tetapi ia tidak menjalankan script secara aktual.

Berikut contoh kode *Output Encode :*

```
StringBuffer buff = new StringBuffer();
if ( value == null ) {
return null;
}
for(int i=0; i<value.length(); i++) {
    char ch = value.charAt(i);
    if ( ch == '&' ) {
            buff.append("&amp;");
    } else if ( ch == '<') {
           buff.append("&lt;");
    } else if ( ch == '>') {
          buff.append("&gt;");
    } else if ( Character.isWhitespace(ch ) ) {
```

```
        buff.append(ch);
    } else if ( Character.isLetterOrDigit(ch) ) {
        buff.append(ch);
    } else if ( Integer.valueOf(ch).intValue()
>= 20 &&
          Integer.valueOf(ch).intValue()   <=
126 ) {
        buff.append( "&#" + (int)ch + ";" );
    }
}
return buff.toString();
```

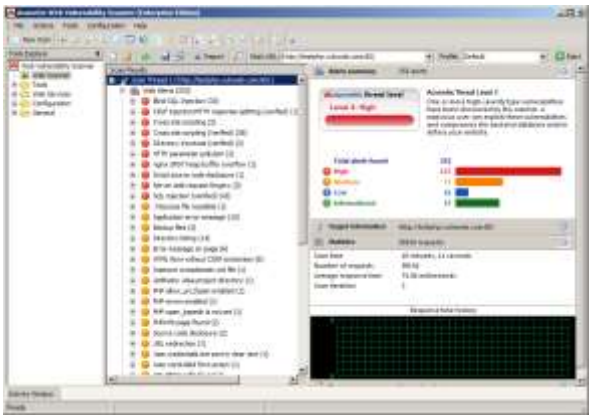
Terdapat 5 aturan (*Rules*) untuk *escaping output* :
#1 - HTML Escape sebelum memasukkan ke konten elemen.
#2 - Attribute Escape sebelum memasukkan ke *attributes*
#3 - JavaScript Escape sebelum memasukkan ke nilai-nilai data JavaScript.
#4 - CSS Escape sebelum memasukkan ke nilai *style property.*
#5 - URL Escape sebelum memasukkan ke *URL attributes*

Crossing Boundaries
Penghalang lain untuk serangan XSS adalah kebijakan"*crossing boundaries*" dimana pengguna otentik harus memasukkan kembali password mereka sebelum mengakses layanan tertentu. Misalnya, bahkan jika pengguna memiliki cookie yang secara otomatis akan masuk ke dalam situs kita, mereka harus dipaksa untuk memasukkan username dan password mereka ketika mereka mencoba untuk mengakses informasi akun sensitif. Batas tambahan ini dapat membatasi kemungkinan sesi yang dibajak oleh serangan XSS.

Penggunaan ID Session
Teknik lain yang sederhana namun efektif adalah untuk segera berakhir sesi jika mesin di dua alamat IP yang terpisah mencoba menggunakan data sesi yang sama. kita dapat membuat ID sesi menggunakan informasi spesifik untuk pengguna seperti timestamp dan alamat IP. Meskipun teknik ini dapat diatasi dengan IP spoofing, itu memberikan lapisan tambahan keamanan terhadap serangan otomatis.

v) *Security Penetration Testing*
Kita harus melihat dengan menggunakan alat pemindaian kode sumber secara otomatis dan Web vulnerability scanners (scanner kerentanan Web) selama pengembangan aplikasi kita. Sebuah scanner vulnerability Web yang baik akan melihat kelemahan teknis yang umum, seperti kerentanan terhadap cross-site scripting. Jika kita menggunakan paket pihak ketiga seperti mesin pencari di situs kita, kita harus selalu memeriksa kerentanan yang diketahui atau masalah konfigurasi dengan vendor. Ini harus diikuti dengan tes menyeluruh tentang bagaimana mereka menangani input yang tidak diinginkan. Tidak pernah menganggap mereka aman.
Sebelum menempatkan aplikasi Web kita online, kita harus melakukan tes penetrasi. Dengan simulasi serangan, kita dapat mengevaluasi apakah situs kita masih memiliki potensi kerentanan XSS yang dihasilkan dari konfigurasi sistem yang buruk atau tidak benar, kekurangan perangkat keras atau perangkat lunak atau kelemahan dalam pertahanan perlindungan situs.

Beberapa tool untuk test keamanan *Web Application* , diantaranya :
Acunetix Web Vulnerability Scanner (Acunetix WVS).
Acunetix Web Vulnerability Scanner adalah sebuah alat pengujian keamanan aplikasi web otomatis yang mengaudit aplikasi web dengan memeriksa kerentanan seperti *SQL Injection*, *Cross site scripting*, dan kerentanan lainnya yang dapat dieksploitasi. Acunetix menawarkan layanan produk komersil dan free (online scanner).
Contoh hasil scan Acunetix, tampak seperti gambar berikut :



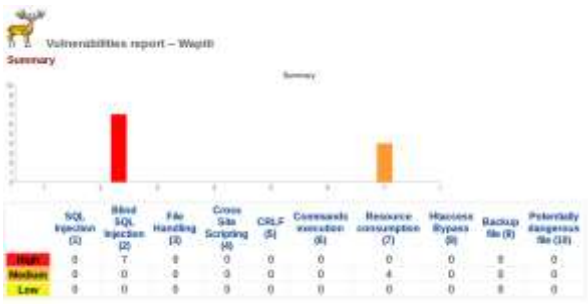
Gambar 5: Contoh Hasil Acunetix WVS

Wapiti

Wapiti memungkinkan kita untuk mengaudit keamanan aplikasi web Anda. Ia melakukan "black-box" scan, yaitu tidak mempelajari kode sumber dari aplikasi tetapi akan memindai halaman web, mencari script dan bentuk di mana ia dapat menyuntikkan data. Setelah mendapat daftar ini, Wapiti bertindak seperti *fuzzer*, melakukan injeksi kode untuk melihat apakah script rentan (vulnerable).

Wapiti dapat mendeteksi kerentanan berikut :

- *File Handling Errors (Local and remote include/require, fopen, readfile…)*
- *Database Injection (PHP/JSP/ASP SQL Injections and XPath Injections)*
- *XSS (Cross Site Scripting) Injection*
- *LDAP Injection*
- *Command Execution detection (eval (), system (), passtru()…)*
- *CRLF Injection (HTTP Response Splitting, session fixation…)*



Gambar 6: Contoh Hasil Wapiti Scanner

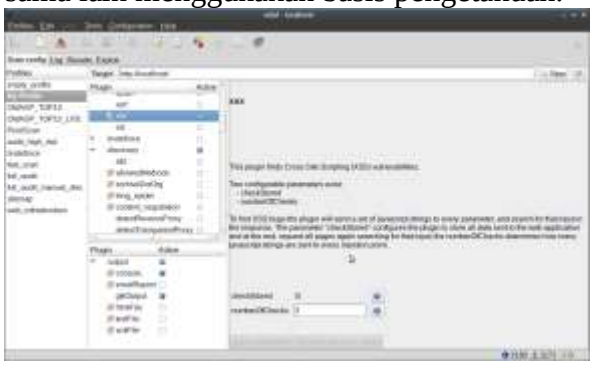
w3af

w3af (singkatan dari *web application attack and audit framework)* adalah scanner keamanan aplikasi web yang *open source*. Proyek ini menyediakan scanner kerentanan dan alat eksploitasi untuk aplikasi Web. Tool ini memberikan informasi tentang kerentanan keamanan dan membantu dalam upaya pengujian penetrasi.

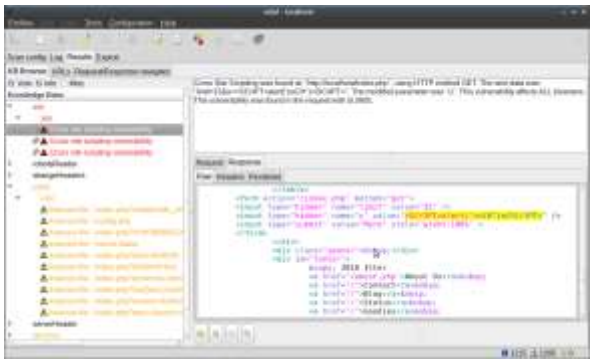
Tool lintas platform ini tersedia di semua sistem operasi populer seperti Microsoft Windows, Linux, Mac OS X, FreeBSD dan OpenBSD dan ditulis dalam bahasa pemrograman Python. Pengguna memiliki pilihan antara antarmuka pengguna grafis dan antarmuka baris perintah.

w3af mengidentifikasi paling banyak kerentanan aplikasi web menggunakan lebih dari 130 plug-in. Setelah identifikasi, kerentanan seperti *SQL injections, OS commanding, remote file inclusions (PHP), cross-site scripting (XSS)* dan upload file yang tidak aman, dapat dimanfaatkan untuk mendapatkan berbagai jenis akses ke sistem remote.

w3af dibagi menjadi dua bagian utama, inti dan plug-in. Inti koordinat proses dan menyediakan fitur yang dikonsumsi oleh plug-in, yang menemukan kerentanan dan mengeksploitasi mereka. Plug-in yang terhubung dan berbagi informasi dengan satu sama lain menggunakan basis pengetahuan.



Gambar 7: Konfigurasi W3af



Gambar 8: Hasil W3af Scanner

KESIMPULAN

Sebuah studi rinci dan terfokus telah dilakukan dalam paper ini dengan metodologi dan pendekatan pada profil keamanan website. Studi detil ini pasti akan diteruskan lebih lanjut pada desain framework dan realisasinya, untuk website yang aman. Desain dan pengembangan sistem informasi atau aplikasi basis web yang aman dapat membantu untuk mengurangi ancaman serangan terhadap website di dunia nyata.

Daftar Pustaka

Anonymous. Reviewing_Code_for_Cross-site_scripting. https://www.owasp.org/index.php/Reviewing_Code_for_Cross-site_scripting. Diakses November-2020

MacKenzie, Thomas. "ScriptAlert1.com – Concise Cross-Site Scripting Explanation in Multiple Languages". Diakses November-2020

Manico, Jim. **Advanced XSS Defence**. http://Secappdev.org. Diakses Desember 2020

Neha Gupta. 2015. "**XSS Defense: An Approach for Detecting and Preventing Cross Site Scripting Attacks '.** COMPUSOFT, An international journal of advanced computer technology, (Volume-IV, Issue-III) March-2015

Robinson, Memen Akbar dan M.F. Ridha. 2018. "SQL Injection and Cross Site Scripting Prevention using OWASP ModSecurity Web Application Firewall". International Journal on Informatics Visualization Vol.2. No. 4 2018; 286 - 292