ISSN: 1907-7912 EISSN: 2622-8092

OPTIMASI METODE OTOMATISASI PENGHILANGAN KERENTANAN TERHADAP SERANGAN XSS PADA APLIKASI WEB

M. Faizal Kurniawan[1], Wahyu Setianto[2]
STMIK Widya Pratama Jl. Patriot 25 Pekalongan
faizal@stmik-wp.ac.id, kian@stmik-wp.ac.id

ABSTRAKSI

Cross site scripting (XSS) adalah salah satu bentuk serangan pada aplikasi web. XSS adalah bentuk serangan injeksi. Serangan injeksi dilakukan dengan menyisipkan kode javascript melalui fasilitas interaksi yang diberikan oleh aplikasi web kepada pengguna. Terdapat tiga jenis XSS: Reflected XSS, Stored XSS dan DOM-Based XSS. OWASP, CWE/SANS dan Accunetix menempatkan serangan XSS pada peringkat 5 besar untuk serangan yang sering dilakukan oleh peretas dan berakibat fatal. Berbagai tema penelitian telah diusulkan untuk mengeliminasi serangan ini. Salah satu paper tersebut mengusulkan cara untuk mengotomatisasi penghilangan kode sumber dengan cara mengoreksi kode sumber dan mendeklarasikan OWASP ESAPI untuk mengencoding semua deklarasi yang berkaitan dengan inputan pengguna. Penelitian tersebut diuusulkan untuk digunakan pada bahasa pemrograman Java namun dapat pula diterapkan pada bahasa pemrograman lain. Penelitian ini mengusulkan metode untuk mengoptimal metode yang diusulkan sebelumnya jika diterapkan pada bahasa pemrograman PHP. Jika penelitian sebelumnya mengusulkan untuk mengoreksi semua baris yang mengandung deklarasi inputan pengguna maka penelitian ini mengoptimalkannya dengan hanya menambahkan beberapa baris kode program dibagian header. Penelitian ini juga mengusulkan untuk tidak lagi menggunakan OWASP ESAPI namun menggunakan fungsi native PHP. Hasil penelitian ini menunjukkan bahwa dibandingkan dengan metode sebelumnya didapatkan peningkatan berupa jumlah baris kode yang diubah atau ditambahkan pada kode sumber aplikasi menjadi lebih sedikit namun dengan hasil yang sama.

Kata Kunci: *Cross site scripting (XSS), PHP , Keamanan Aplikasi Web, Fungsi Encoding PHP*

1. PENDAHULUAN

Cross site scripting (XSS) adalah salah satu masalah keamanan pada aplikasi web (Hydara, 2014). XSS ditemukan pada permulaan tahun 1990 oleh World Wide Web. XSS merupakan jenis masalah injeksi, dimana memungkinkan kode berbahaya diinjeksikan pada website (OWASP, 2013). Hal ini dapat terjadi ketika aplikasi gagal melakukan validasi dengan baik terhadap masukan pengguna.

Serangan XSS yang berhasil dilakukan dapat mengakibatkan pelanggaran yang serius baik pada website itu sendiri maupun pada pengguna lain. Peretas

dapat menginjeksikan kode berbahaya melalui aplikasi web yang mengijinkan pengguna melakukan masukan yang tidak divalidasi dengan baik. Akibat dari serangan tersebut peretas dapat mencuri cookie, mengambil informasi yang sifatnya pribadi, membajak akun pengguna, memanipulasi konten web, melakukan serangan danial of service dan berbagai macam aktifitas berbahaya lainnya (Bathia, 2011). XSS dapat juga dijadikan sebagai fasilitas untuk melakukan serangan dalam bentuk lain seperti phishing dan drive-by-download (Johari, 2012).

Terdapat tiga jenis serangan XSS yaitu: reflected, stored dan DOM-based. Reflected XSS dieksekusi melalui browser dan terjadi jika website menyediakan tempat bagi pengguna untuk melakukan masukan. Stored XSS terjadi ketika kode berbahaya berhasil tersimpan ke dalam database dan dapat dieksekusi oleh pengguna lain (biasanya melalui sebuah link yang yang telah diinjeksi dengan kode berbahaya). DOM-based XSS terjadi ketika peretas dapat melakukan serangan XSS di sisi client side melalui DOM (Document Object Model), serangan ini dilakukan dengan memanfaatkan celah keamanan website pada bagian client side (Gundy, 2012). DOM sendiri adalah cara standar dari W3C (World Wide Web Consortium) untuk mengakses dan memanipulasi document HTML atau XML (Mozilla, 2014).

Open Web Application Security Project (OWASP) menempatkan XSS pada peringkat ketiga sebagai salah satu serangan yang paling berakibat fatal terhadap aplikasi web pada OWASP Top Ten 2013 (OWASP, 2013). CWE/SANS menempatkan XSS di posisi keempat pada CWE/SANS Top 25 Most Dangerous Software Errors 2011 (CWE/SANS, 2011). Sedangkan Acunetix menempatkan XSS sebagai peringkat ketiga dengan persentase 12,58% sebagai salah satu metode yang paling sering digunakan oleh peretas dalam melakukan serangan terhadap aplikasi web (Acunetix, 2014).

Berbagai tema penelitian dan berbagai metode telah diusulkan oleh banyak peneliti untuk menangani masalah kerentanan aplikasi web terhadap serangan XSS (Hydara, 2014). Namun, solusi paling ideal untuk mengeliminasi kerentanan aplikasi web terhadap serangan XSS dari akarnya adalah dengan memperbaiki kode sumber aplikasi dari kemungkinan terserang oleh XSS (Barnett, 2011).

Berikut ini adalah ringkasan dari hasil penelitian sebelumnya yang berfokus pada metode untuk menghilangkan kerentanan kode sumber dari serangan XSS:

a.) Penilitian pertama mengusulkan sebuah metode untuk mendeteksi dan menghilangkan pada kode sumber aplikasi web sejak pertama kali ditulis. Penelitian ini mengusulkan sebuah plugin Eclipse yang mampu mendeteksi dan menghilangkan kerentan kode sumber dari kerentanan terhadap serangan XSS pada bahasa pemrograman Java. Metode ini cocok diterapkan saat kode program baru akan ditulis, namun tidak cocok diterapkan pada kode sumber yang sudah ada (Bathia, 2011).

b.) Penelitian kedua mengusulkan metode otomatisasi pendeteksian dan penghilangan kode sumber dari serangan XSS pada aplikasi web. Yaitu sebuah metode yang mampu mendeteksi dan menghilangkan kerentanan kode sumber yang sudah pernah ada dari serangan XSS secara otomatis. Yaitu dengan cara menscan semua file kode program yang ada dan secara otomatis mendeteksi kode yang dianggap rentan dari serangan XSS untuk kemudian direplace dengan mendeklasikan fungsi encoding/escaping dari pustaka OWASP ESAPI. Encoding/escaping dimaksudkan agar apa yang diinjeksikan peretas kedalam program tidak dapat tereksekusi, dimana kode seperti single quote (‘), double quote (“) dan lain sebagainya dikonversi menjadi kode yang tidak tereksekusi. Contoh penerapan dilakukan pada bahasa pemrograman Java, namun dapat diterapkan pada bahasa pemrograman lain seperti PHP, Perl, Phyton dan lain sebagainya (Shar, 2011).

Pada dasarnya tujuan akhir yang dicapai dari kedua penelitian tersebut diatas adalah sama, yaitu melakukan encoding/escaping semua inputan pengguna dari kemungkinan tersisipnya kode javascript “jahat” sehingga aplikasi menjadi rentan terhadap serangan XSS (Shar, 2011). Karena sesuai dengan jenisnya, XSS adalah jenis serangan injeksi. Inti dari serangan jenis ini adalah berhasilnya peretas menyisipkan kode berbahaya ke dalam aplikasi. Sedangkan kode berbahaya hanya akan tereksekusi jika apa yang diinputkan oleh peretas tidak disanitasi dengan baik oleh aplikasi (Shar, 2011).

PHP adalah bahasa pemrograman server side yang paling banyak digunakan saat ini (W3Tech, 2015). Hasil survey menunjukkan bahwa persentase penggunaan bahasa pemrograman PHP sebesar 81.5% mengungguli bahasa pemrograman lain. Bahasa pemrograman yang dikembangkan oleh Rasmus Lerdorf sejak tahun 1994 dan terus berkembang hingga sekarang ini telah menyediakan fungsi native untuk melakukan encoding yaitu htmlspecialchars, dimana fungsi ini akan mengencoding string parameter yang mengandung karakter spesial seperti: &,‘,”,<, dan > menjadi kode yang masih dapat dibaca di browser namun tidak tereksekusi (PHP, 2013).

PHP juga menyediakan fungsi native untuk melakukan looping array yaitu foreach, fungsi ini dapat dimanfaatkan untuk mendapatkan semua nilai inputan yang dilakukan oleh pengguna secara recursive (PHP, 2013). Karena inputan yang dikirim pengguna akan diterima dalam bentuk array, maka fungsi foreach dapat dimanfaatkan untuk mengencoding lebih dulu semua inputan pengguna sebelum dieksekusi oleh baris-baris program dibawahnya.

Tujuan dari penelitian ini adalah mengoptimalkan metode yang diusulkan oleh Shar, et., al untuk diterapkan pada aplikasi web yang dibangun dengan bahasa pemrograman PHP. Perbaikan yang diusulkan adalah dengan tidak lagi menggunakan pustaka tambahan, namun menggunakan fungsi PHP native yaitu htmlspecialchars sehingga diharapkan tidak perlu ada konfigurasi tambahan serta tidak diperlukan jeda waktu untuk meload pustaka tersebut.

Selain itu perubahan metode untuk mengencoding semua inputan pengguna juga diusulkan untuk diubah, alih-alih mengubah semua kode yang dianggap rentan pada setiap baris kodenya, metode ini mengusulkan untuk hanya cukup menginjeksikan sedikit kode dibagian header program dengan memanfaatkan fungsi foreach PHP untuk mengencoding semua deklarasi kode yang mengandung inputan pengguna.

2. METODE PENELITIAN

2.1 Pengumpulan Data

Penelitian ini diawali dengan melakukan pengumpulan data. Dataset yang akan dibuat terdiri dari 2 aplikasi sederhana yang masing-masing terindikasi rentan terhadap serangan XSS.

Aplikasi pertama adalah aplikasi yang rentan terhadap jenis serangan reflected XSS

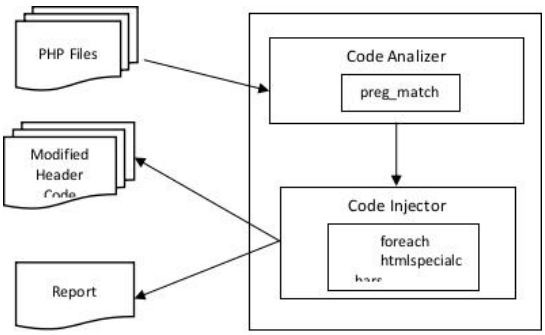
Aplikasi kedua adalah aplikasi yang rentan terhadap jenis serangan stored XSS

Model aplikasi yang dibangun merujuk pada Demo Application - Google Application Security Learning bagian Cross-site Scripting.

2.2 Pengolahan Data Awal

Pengolahan data awal dilakukan dengan meniru dan menulis kedua aplikasi pada Demo Application - Google Application Security Learning bagian Cross-site Scripting ke dalam bahasa pemrograman PHP. Penulisan ulang dilakukan karena kedua aplikasi tersebut hanyalah demo dan berjalan dilingkungan virtual serta dibangun dengan bahasa pemrograman Phyton (Google, 2014).

2.3 Metode yang Diusulkan



Gambar 2.1 Metode yang Diusulkan

File-file PHP terlebih dulu akan diproses oleh Code Analizer, tahapan ini dilakukan untuk mencari jenis kode yang dideklarasikan untuk mendapatkan inputan dari pengguna. Untuk mendapatkan inputan dari pengguna, PHP perlu mendeklaraskan klausa $_POST, $_GET dan $_REQUEST. Klausa inilah yang dideteksi oleh Code Analizer. Output yang dihasilkan pada tahap ini adalah jenis deklarasi kode apa saja yang terdapat pada file kode sumber.

Output yang dihasilkan oleh Code Analizer adalah input untuk Code Injector. Code Injector akan secara otomatis menuliskan kode tambahan dibagian atas kode program berdasarkan laporan yang dihasilkan oleh Code Analizer.

2.4 Evaluasi dan Validasi

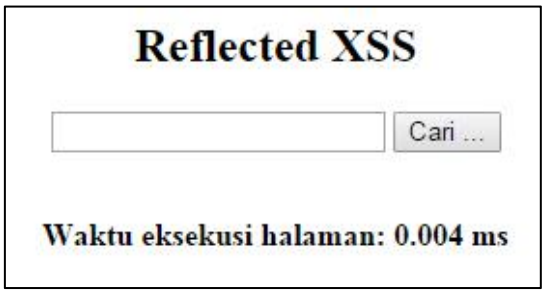
Untuk mengetahui adanya peningkatan efisiensi dari metode yang diusulkan dengan metode yang diusulkan oleh Shar,

et., al. maka sesuai dengan desain penelitian yang digunakan yaitu one group pretest postest, maka pada tahap ini akan membanding hasil modifikasi kode program saat menggunakan metode yang diusulkan sebelumnya dengan metode yang diusulkan pada penelitian ini. Variabel yang akan diukur adalah jumlah baris kode yang dimodifikasi pada kode sumber target serta waktu eksekusi target setelah kode programnya dimodifikasi.

3. HASIL DAN PEMBAHASAN

3.1 Hasil Pembuatan Data Uji

Guna mendapatkan pembuktian yang baik, maka pengujian dilakukan dengan terlebih dahulu membuat aplikasi yang benar-benar rentan terhadap serangan XSS. Model aplikasi yang dibuat merujuk pada pembelajaran XSS pada Google Application Security. Dibagian bawah aplikasi akan muncul informasi mengenai seberapa lama halaman ini ditampilkan oleh browser (waktu eksekusi), kode ini ditambahkan untuk kepentingan pengujian.



Gambar 3.1 Aplikasi 1

Gambar 3.2 Aplikasi 2

3.2 Hasil Uji Kerentanan Aplikasi Sebelum Dimodifikasi

Guna mengetahui bahwa aplikasi yang digunakan sebagai dataset benar-benar rentan terhadap serangan XSS, maka kedua aplikasi tersebut diuji dengan menyisipkan kode javascript sederhana pada form yang disediakan oleh aplikasi yaitu :

```
<script>alert('Hello World')</script>
```

apabila kode tersebut tereksekusi maka aplikasi tersebut rentan terhadap serangan XSS.

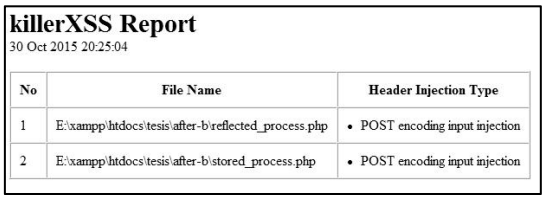
	Aplikasi 1	Aplikasi 2
Kode: <script>alert('Hello World')</script>	Tereksekusi	Tereksekusi
Gambar Tampilkan Hasil Eksekusi		

Gambar 3.4 Hasil uji kerentanan XSS pada aplikasi sebelum dimodifikasi

3.3 Hasil Pembuatan Prototype

Sebuah prototype telah dibangun sebagai sarana untuk menerapkan metode yang diusulkan pada penelitian ini. Hal utama yang dilakukan oleh prototype ini adalah men-scan dan mengoreksi semua kode pada direktori target yang dianggap rentan terhadap serangan XSS.

Gambar 3.5 Tampilan awal prototype



killerXSS Report

30 Oct 2015 20:25:04

No	File Name	Header Injection Type
1	E:\xampp\htdocs\tesis\after-b\reflected_process.php	• POST encoding input injection
2	E:\xampp\htdocs\tesis\after-b\stored_process.php	• POST encoding input injection

Gambar 3.6 Report yang Dihasilkan oleh Prototype

3.4 Hasil Uji Kerentanan Aplikasi Setelah Dimodifikasi

Guna mengetahui bahwa metode yang diusulkan sebelumnya oleh Shar, et.al., dan metode yang diusulkan pada penelitian ini benar-benar mampu mencegah tereksekusinya kode javascript oleh aplikasi, maka setelah dimodifikasi aplikasi 1 dan 2 diuji kembali dengan kode javascript yang sama. File yang dianggap rentan oleh kedua metode adalah berkas pemroses (reflected_proses.php dan stored_proses.php) hal ini disebabkan karena hanya pada dua berkas itulah terdapat deklarasi kode untuk mendapatkan inputan dari pengguna.

	Metode Sebelumnya		Metode yang Diusulkan	
	Aplikasi 1	Aplikasi 2	Aplikasi 1	Aplikasi 2
Kode: <script> alert('H ello World')< /script>	Tidak Tereksekusi	Tidak Tereksekusi	Tidak Tereksekusi	Tidak Tereksekusi
Hasil Encoding	<script&gt ;alert(' Hello World ');</ script>	<script&gt ;alert(' Hello World ');</ script>	<script&gt ;alert(' Hello World ');</ script>	<script&gt ;alert(' Hello World ');</ script>

Gambar 3.7 Hasil uji kerentanan XSS pada aplikasi setelah dimodifikasi

Gambar 3.7 menunjukkan hasil eksekusi kode javascript setelah dimodifikasi oleh kedua metode. Dari gambar 3.7 tersebut menunjukan bahwa kedua metode mampu mengencoding kode javascript sehingga tidak dieksekusi oleh browser.

Metode sebelumnya menggunakan fungsi encodeForHTML dari OWASP ESAPI dan metode yang diusulkan menggunakan fungsi native php yaitu htmlspecialchar. Hal ini membuktikan bahwa fungsi native php yaitu htmlspecialchars berfungsi selayaknya pustaka OWASP ESAPI.

3.5 Hasil Pengukuran Jumlah Perubahan Baris Kode Aplikasi Setelah Dikoreksi Oleh Kedua Metode

	Jumlah Perubahan Kode Program	
	Metode Sebelumnya	Metode yang Diusulkan
Aplikasi 1 (reflected_proses.php)	8	4
Aplikasi 2 (stored_proses.php)	13	4
Rata-Rata	**10.5**	**4**

Gambar 3.7 Hasil pengukuran jumlah baris program yang dimodifikasi atau ditambahkan

Gambar 3.7 menunjukkan hasil pengukuran jumlah baris yang

dimodifikasi atau ditambahkan ke kode sumber target. Terlihat bahwa baris yang dimodifikasi atau ditambahkan ke kode sumber target oleh metode yang diusulkan lebih sedikit dari pada metode sebelumnya, artinya metode yang diusulkan lebih efisien karena hanya menambahkan sedikit perubahan pada kode sumber target.

Untuk mengetahui peningkatan efisiensi metode yang diusulan terhadap metode sebelumnya, maka nilai rata-rata jumlah modifikasi atau penambahan kode program dimasukkan ke dalam rumus : $\frac{r1-r2}{r1} * 100\%$ dimana r1 adalah rata-rata jumlah modifikasi atau penambahan oleh Metode A dan r2 adalah rata-rata jumlah modifikasi atau penambahan oleh Metode B. Berikut adalah hasil perhitungannya:

$$\frac{10.5 - 4}{10.5} * 100\% = 61.90\%$$

Dari hasil perhitungan diatas dapat disimpulkan bahwa ada peningkatan efisiensi jumlah modifikasi atau penambahan antara metode yang diusulkan terhadap metode sebelumnya sebesar **61.90%**.

4. KESIMPULAN

Dari hasil penelitian yang telah dilakukan mulai dari tahap hingga proses pengujian, dapat disimpulkan bahwa:

4.1 Berdasarkan hasil eksperimen yang dilakukan, didapatkan fakta kedua metode dapat mengencoding inputan pengguna, sehingga apabila pengguna menginjeksikan kode javascript maka kode tersebut tidak akan dieksekusi oleh browser.

4.2 Berdasarkan hasil pengukuran jumlah kode sumber yang diubah atau ditambahkan pada kode sumber target, dapat disimpulkan bahwa metode yang diusulkan mampu meminimalkan perubahan sebesar 61.90%

DAFTAR PUSTAKA

I. Hydara, A. B. M. Sultan, H. Zulzalil, and N. Admodisastro, (2014), "Current state of research on cross-site scripting (XSS) – A systematic literature review," Inf. Softw. Technol.

L. K. Shar and H. B. K. Tan, (2011), "Automated removal of cross site scripting vulnerabilities in web applications," Inf. Softw. Technol., vol. 54, no. 5, pp. 467–478.

OWASP, (2014, September 2014) "OWASP Top Ten project 2013," 2013. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-Top_10.

P. Bathia, B. R. Beerelli, and M.-A. Laverdière, (2011), "Assisting Programmers Resolving Vulnerabilities in Java Web Applications," CCIST 2011 Commun. Comput. Inf. Sci., vol. 133, no. 1, pp. 268–279.

R. Johari and P. Sharma, (2012), "A Survey on Web Application Vulnerabilities (SQLIA, XSS) Exploitation and Security Engine for SQL Injection," 2012 Int. Conf. Commun. Syst. Netw. Technol., pp. 453–458.

M. Van Gundy and H. Chen, (2012), "Noncespaces: Using randomization to defeat cross-site scripting attacks," Comput. Secur., vol. 31, no. 4, pp. 612–628, Jun.

H. Shahriar and M. Zulkernine, (2009), "MUTEC : Mutation-based Testing of Cross Site Scripting School of Computing," pp. 47–53.

Google, "Cross-site scripting," (2014, Oktober 2014), https://www.google.com/about/appsecurity/learning/xss/.

Mozilla, "DOM," (2014, Oktober 2014). https://developer.mozilla.org/en-US/docs/Glossary/DOM.

w3schools, "JavaScript HTML DOM," (2014), http://www.w3schools.com/js/js_htmldom.asp.

CWE/SANS, "Top 25 Most Dangerous Programming Errors 2011," (2011), http://cwe.mitre.org/top25/.

Acunetix, "Cross-site Scripting (XSS) Attack." (2014, November 2014). http://www.acunetix.com/websitesecurity/cross-site-scripting/.

R. Barnett, (2011), "XSS Street-Fight : The Only Rule Is There Are No Rules".

L. K. Shar and H. B. K. Tan, (2011), "Defending against cross-site scripting attacks," Computer (Long. Beach. Calif)., vol. 3, pp. 55–62.

S. Esser, (2008), "Lesser Known Security Problems in PHP Applications," Zend Conf.

W3Tech, (2015), "World Wide Web Technology Surveys," http://w3techs.com/.

php.net, (2013). "PHP: htmlspecialchars," https://secure.php.net/manual/en/function.htmlspecialchars.php.

php.net, "PHP: foreach,", (2013), https://secure.php.net/manual/en/control-structures.foreach.php.