Fuzzing untuk Menemukan Kerentanan Aplikasi Web

Makalah Tugas Kuliah EL5215
Keamanan Piranti Lunak

Oleh
YOGI KRISTIAWAN
NIM : 23215097

Dosen :
Ir. BUDI RAHARDJO, MSc., PhD.

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2016

ABSTRAK

Fuzzing merupakan suatu metode mencari kesalahan piranti lunak dengan menyediakan input yang tidak diduga lalu memonitoring hasilnya. *Fuzzing* pada umumnya merupakan proses otomatisasi atau semi automatisasi yang melibatkan manipulasi dan menyediakan data secara berulang-ulang untuk selanjutnya akan diproses oleh target piranti lunak. *Website* merupakan salah satu media piranti lunak yang berkembang dengan pesat saat ini. Kemudahan serta murahnya biaya untuk membuatnya menjadi salah satu faktor mendukungnya. Namun kemudahan tersebut tidaklah diimbangi dengan keamanan yang memadai. Salah satu penyebab rendahnya keamanannya ialah tidak dilakukannya pengujian piranti lunak sebelum suatu *website* ditampilkan. *Fuzzing* dapat menjadi salah satu solusi pengujian suatu *website* untuk melihat kerentanan apa saja yang terdapat pada suatu *website*.

Kata kunci: *fuzzing*; piranti lunak; *website*; pengujian; kerentanan

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

Bab I Pendahuluan

Piranti lunak menjadi bagian yang penting dalam menunjang sebagian besar aktifitas manusia saat ini. Kemudahan dalam membangun sebuah piranti lunak juga seakan menjadi penunjang semakin banyaknya piranti lunak yang beredar. Namun dibalik kemudahan yang ditawarkan, keamanan piranti lunak tersebut menjadi masalah tersendiri. Sumber masalah keamanan piranti lunak yang umum terjadi dapat dibedakan menjadi 3 yakni: konektivitas, kemampuan update, dan kompleksitas [1].

Aplikasi web merupakan salah satu jenis piranti lunak yang umum digunakan saat ini. Potensi resiko keamanan aplikasi web yang dapat dimanfaatkan penyerang tidaklah sedikit. menurut data dari Imperva pada tahun 2014 terdapat 44% peningkatan serangan pada website dibanding tahun 2013. Dengan persentasi 24,1% website menggunakan CMS wordpress diserang lebih banyak dibanding CMS lain. Sementara OWASP pada tahun 2013 [2] merilis sepuluh potensi resiko keamanan aplikasi web (tabel 1.1)

Tabel I.1 Sepuluh Besar Potensi Resiko Aplikasi Web Tahun 2013

No	Resiko	Deskripsi
1.	*Injection*	*Injection flaws* bisa berupa injeksi SQL, OS, dan LDAP. Terjadi karena *untrusted* data dikirim ke sebuah penterjemah yang akan memberikan perintah atau akses data tanpa otorisasi yang tepat.
2.	*Broken Authentication and Session Management*	Fungsi aplikasi yang terkait dengan otentifikasi dan manajemen sesi seringkali tidak benar implementasinya, sehingga penyerang dapat mengambil password, key, *session tokens*, atau exploit kelemahan lainnya untuk mengambil data identitas user.
3.	*Cross – Site Scripting (XSS)*	XSS flaws terjadi karena sebuah aplikasi mengambil *untrusted* data dan mengirim data tersebut ke browser tanpa melalui validasi. XSS memungkinkan penyerang mengeksekusi scripts pada browser yang

		akan diserang dengan mengambil *user sessions*, *deface web sites*, atau *redirect* pengguna ke situs lainnya.
4.	*Insecure Direct Object References*	Akses ke *object reference* terjadi bila pengembang exposes sebuah referensi dari sebuah internal implementation seperti: file, direktori, atau database key tanpa adanya pengecekan *access control* atau proteksi, sehingga penyerang dapat melakukan manipulasi referensi tersebut yang dapat digunakan untuk melakukan akses unauthorized data.
5.	*Security Misconfiguration*	Kesalahan konfigurasi kemanan terjadi karena adanya kesalahan konfigurasi pada aplikasi, *framework*, server aplikasi, web server, server database, dan platform.
6.	*Sensitive Data Exposure*	Banyak aplikasi web tidak benar melindungi data sensitif, seperti kartu kredit, nomor pajak, dan autentifikasi. Penyerang dapat mencuri atau memodifikasi data yang dilindungi lemah seperti untuk melakukan penipuan kartu kredit, pencurian identitas, atau kejahatan lainnya. data sensitif layak perlindungan ekstra seperti enkripsi saat istirahat atau dalam perjalanan, serta tindakan pencegahan khusus ketika ditukar dengan browser.
7.	*Missing Function Level Access Control*	Aplikasi web yang paling memverifikasi hak akses tingkat fungsi sebelum membuat fungsi yang terlihat di UI. Namun, aplikasi perlu melakukan pemeriksaan kendali akses yang sama pada server ketika setiap fungsi diakses. Jika permintaan tidak diverifikasi, penyerang akan dapat menempa permintaan untuk mengakses fungsi tanpa otorisasi yang tepat.
8.	*Cross-Site Request Forgery*	Serangan CSRF terjadi karena seorang

		pengguna aplikasi yang melakukan login akan dialihkan ke aplikasi web yang sudah dipalsukan, kemudian data dari user tersebut akan diambil.
9.	*Using Known Vulnerable Components*	Komponen, seperti perpustakaan, kerangka kerja, dan modul perangkat lunak lain, hampir selalu dijalankan dengan hak ull. Jika komponen rentan dieksploitasi, seperti serangan dapat memfasilitasi data yang seriusoss atau server pengambilalihan. Aplikasi menggunakan komponen dengan kerentanan dikenal mungkin melemahkan pertahanan aplikasi dan memungkinkan berbagai serangan dan dampak yang mungkin.
10.	*Unvalidated Redirects and Forwards*	Terjadi pengalihan/redirect dan meneruskan/ forward ke web site lain karena tidak adanya validasi terhadap halaman yang dituju tersebut. Hal ini memungkinkan terjadinya pengalihan ke situs phishing atau malware.

Karena besarnya potensi resiko keamanan pada aplikasi web tersebut, maka perlu dilakukan *vurnerability assessment* yang bertujuan untuk mengidentifikasi dan mengukur security kerentanan dari aplikasi web tersebut. Ada berbagai macam metode yang dapat digunakan untuk melakukan *vurnerability assessment*, salah satu metode VA yang bisa digunakan adalah metode *fuzzing*.

Fuzzing adalah suatu metode dengan cara memasukan nilai tak terduga ke sistem target dan memonitor respon abnormal untuk menemukan kerentanan aplikasi web. [3]. *Fuzzing* pada umumnya merupakan proses otomatisasi atau semi automatisasi yang melibatkan manipulasi dan menyediakan data secara berulang-ulang untuk selanjutnya akan diproses oleh target piranti lunak.

Bab II Tinjauan Pustaka

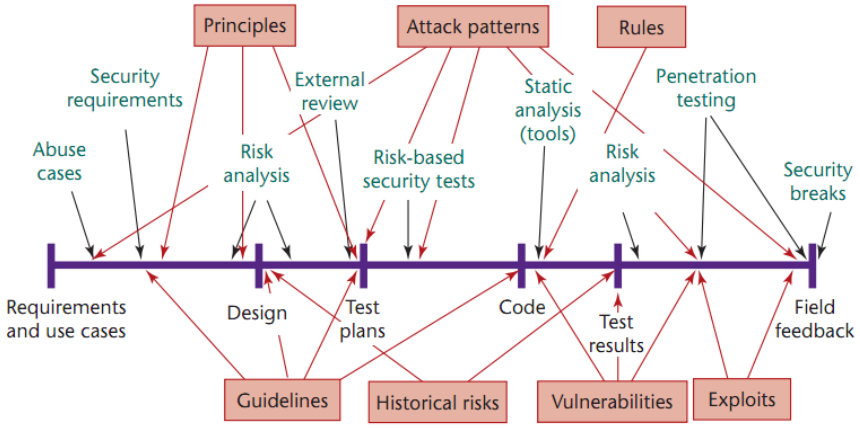
Pada bab ini berisi tinjauan pustaka mengenai penelitian yang berhubungan dengan topik makalah antara lain:

II.I Keamanan Piranti Lunak

Keamanan piranti lunak merupakan proses mendesain, membangun dan melakukan uji coba sebuah piranti lunak dengan memperhatikan keamanan di tiap bagiannya agar piranti lunak tersebut dapat menahan serangan [1]. Untuk membangun piranti lunak yang aman pada umumnya bergantung pada proses saat rekayasa perangkat lunak, bahasa pemrograman yang digunakan dan teknik keamanan yang diterapkan.

Membangun sebuah piranti lunak yang tanpa celah keamanan tidaklah mungkin. Hanya menunggu waktu, cara atau kesempatan bagi penyerang untuk dapat menemukan celah keamanan tersebut. Untuk itu pentingnya kesadaran akan keamanan sejak piranti lunak akan dibangun. Terdapat tiga pillar keamanan piranti lunak yakni manajemen resiko, *touchpoint*, dan pengetahuan. Dengan menerapkan ketiganya secara bertahap dan ukuran yang sesuai maka program keamanan piranti lunak yang standar dan hemat dapat terwujud [1].

Adapun tahapan pembangunan piranti lunak yang aman dengan dasar tiga pilar tersebut menurut Gary McGraw adalah sebagai berikut



Gambar II.1 Tahapan *Secure Software Development Life Cycle* (SDLC) [1]

II.2 Metode Pencarian Kerentanan

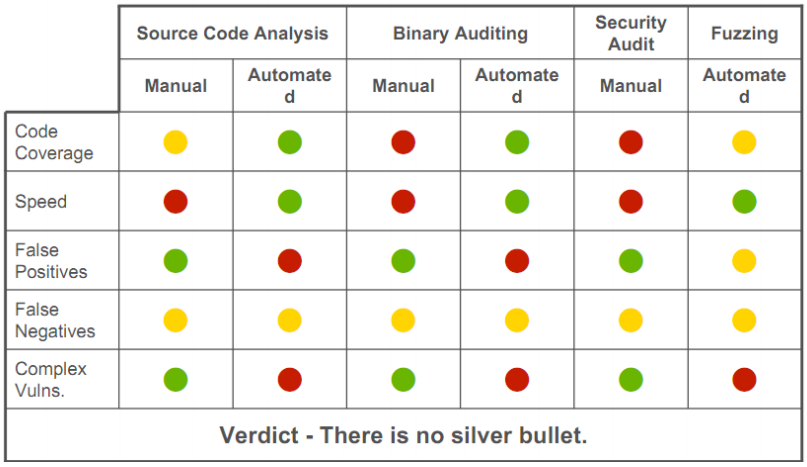
Kerentanan adalah cacat dan kekurangan pada sistem komputer yang secara spesifik diimplementasikan pada piranti lunak, piranti keras, protocol, atau pada kebijakan keamanan [4]. Kerentanan tersebut memungkinkan adanya serangan yang dapat menganggu sistem baik dari sisi kerahasiaan, integritas maupun ketersediaan.

Untuk mengatasinya diperlukan suatu metode untuk dapat mengetahui kerentanan yang ada sebelum digunakan penyerang. Secara umum dikenal tiga jenis pendekatan untuk mencari kerentanan tersebut yakni *black box, white box,* dan *gray box* [3].Perbedaan mendasar dari ketiga pendekatan tersebut ditentukan dari *resources* apa yang dimiliki. Pendekatan *white box* memerlukan full akses ke *resources* seperti *source code* dan spesisifiikasi desain. Pendekatan *black box* memiliki sedikit tentang target. Sementara pendekatan *gray box* memerlukan *resource* tertentu seperti akses untuk *compiled binaries* dan dokumentasi dasar.

Source code review merupakan salah satu cara mencari kerentanan sistem berdasarkan pendekatan *white box* dengan cara melihat *source code* tiap baris lalu menentukan jenis kerentanan yang ada. Secara umum hal ini dapat dilakukan secara manual ataupun automatisasi. Kelebihan metode ini adalah dapat mengakses *complete coverage* dari sistem. Sementara kekurangannya ialah waktu yang dibutuhkan sebanding dengan kompleksitas sistem.

Selanjutnya *Binary auditing* merupakan salah satu cara mencari kerentanan sistem berdasarkan pendekatan *gray box* dengan menggunakan *reverse code engineering* (*RCE*) untuk menentukan fungsionalitas *compiled binary* dari aplikasi. Kelebihan metode ini adalah dari sisi *coverage* walau tidak seperti *source code review* namun hasilnya dapat membantu memperbesar *coverage* metode *black box.* Sementara kekurangannya dari sisi kompleksitas membutuhkan kemampuan yang baik dalam menggunakan *RCE*.

Untuk pendekatan metode *black box* cara yang dapat digunakan ialah *fuzzing.* Pembahasan lengkap tentang *fuzzing* terdapat pada pembahasan bab II.3. Berikut merupakan gambar terkait kelebihan dan kekurangan masing-masing metode pencarian kerentanan sistem menurut Michael Sutton [5].



Gambar II.2 Kelebihan dan kekurangan metode-metode mencari kerentanan sistem

II.3 *Fuzzing*

Penggunaan kata *fuzzing* pertama kali datang dari proyek riset di Universitas Wisconsin-Madison. *Fuzzing* merupakan suatu metode mencari kesalahan piranti lunak dengan menyediakan input yang tidak diduga lalu memonitoring hasilnya [3]. *Fuzzing* pada umumnya merupakan proses otomatisasi atau semi automatisasi yang melibatkan manipulasi dan menyediakan data secara berulang-ulang untuk selanjutnya akan diproses oleh target piranti lunak.

Tahapan dalam *fuzzing* secara umum dapat dibedakan menjadi 6 tahapan yakni: identifikasi target, identifikasi input, Membuat data *fuzzing*, input data *fuzzing*, monitoring respon abnormal, dan evaluasi [4] seperti tertera pada tabel II.1.

Tabel II.1 Tahapan dalam *Fuzzing*

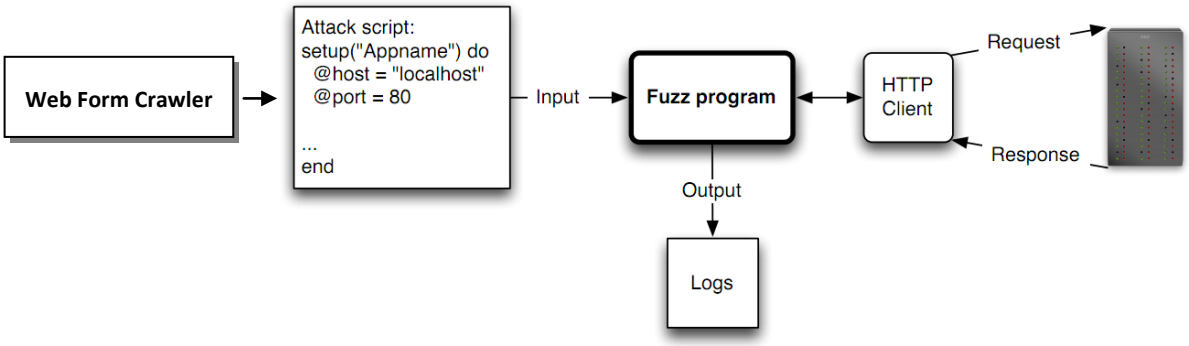
No	Tahapan	Keterangan
1.	Identifikasi target	Melakukan identifikasi aplikasi web yang akan diberikan *fuzzing*, misal: jumlah form
2.	Identifikasi input	Melakukan identifikasi metode pngirimam data pada aplikasi web, misal: Get, Post, Ajax, dan lain-lain.
3.	Membuat data *fuzzing*	Setelah dilakukan identifikasi metode input yang digunakan, langkah selanjutnya yaitu membuat Fuzzer yang

		akan membuat random input yang akan digunakan untuk *fuzzing*.
4.	Input data *fuzzing*	Melakukan request ke aplikasi web dengan menggunakan data random yang telah digenerate (fuzzed data).
5.	Monitoring response abnormal	Response dari aplikasi web akan dimasukkan ke database dan dilakukan monitoring, data yang dimonitoring meliputi: response time, maximum dan minimum response time, dan lain-lain.
6.	Evaluasi	Melakukan evaluasi terhadap response abnormal dan jenis bug yang belum tercover untuk *fuzzing* selanjutnya.

BAB III Pembahasan

III.1 Model dan cara kerja Rfuzz fuzzer

Proses dan komponen yang digunakan dalam *fuzzing* pada aplikasi web dapat secara umum dapat digambarkan sebagai berikut:



Gambar III.1 Model Rfuzz fuzzer

Proses dan Komponen tersebut dapat dijelaskan sebagai berikut:

1. Web Form Crawler, berfungsi untuk mencari form input dari web yang akan menjadi target *fuzzing*. Sebagai contoh *crawler* mencari "search box" pada halaman web yang dituju dengan nilai *default* "Search…". Setelah itu akan keluar hasilnya kurang lebih sebagai berikut:

   ```
   attack("/Welcome_to_myblog") do
   many :post, "/search", {"q" => "Search ..."}
   end
   ```

   Dari output dapat kita lihat bahwa di halaman dengan URI / Welcome_to_mbblog, *crawler* menemukan *form input* ke URI /search dan yang memiliki satu *field* dengan nama q dan nilai *default* dari "Search ...". Selanjutnya parameter inilah yang akan menjadi input *attack script*

2. *Attack script*, yaitu *script* yang digunakan sebagai input pada fuzzing.

   Setelah mendapat hasil output dari *crawler* lalu hasilnya akan diubah baik dilakukan secara otomatis atau semi otomatis menjadi seperti berikut:

```
attack("Search box") do
many :post, "/search", {:q => str(100)}
many :post, "/search", {:q => byte(100)}
many :post, "/search", {:q => big}
end
```

Saat nilai tersebut dimasukkan ke fuzzer akan memiliki nilai:

a) Kirim banyak nilai dengan parameter q set dengan nilai *random string*, dan panjang 100 menggunakan metode POST

b) Kirim banyak nilai dengan parameter q set dengan nilai *random byte sequence*, dan panjang 100 menggunakan metode POST

c) Kirim banyak nilai dengan parameter q set dengan nilai *random big* menggunakan metode POST

3. Fuzz program yaitu bagian yang digunakan untuk menggenerate *Attack script* untuk diubah menjadi bahasa HTTP yang akan digunakan sebagai input ke aplikasi web. Fuzz program juga menerima hasil output dari respon aplikasi web untuk selanjutnya dilakukan *monitoring* dan pencatatan ke *log.*

4. *Web Application (HTTP Client*)

   HTTP Client merupakan sasaran tempat kita menginput nilai-nilai acak yang digenerate sebelumnya. *HTTP client* akan melanjutkan input tersebut ke *host.* Respon dari *host* akan ditampilan *HTTP client* untuk selanjutnya akan dicatat dan disimpan pada file *log*.

5. Logs merupakan file hasil dari response *host* terhadap nila-nilai input yang diberikan. Nilai yang dicatat seperti jenis error, waktu input, jumlah input, dll. Nilai inilah yang nantinya akan dianalisis untuk diketahui kerentanan pada aplikasi web yang diuji.

III.2 Pengujian dan hasil

Penelitian terkait pengujian *fuzzing* pada aplikasi web yang sudah dilakukan oleh Rune Hammersland and Einar Snekkenes [6]. Pengujian dilakukan menggunakan Rfuzz fuzzer terhadap 8 web aplikasi dan hasilnya seperti pada gambar 3.2.

	Complexity			Issues			
Application	#forms	#inputs	time	E1	E2	E3	E4
Chyrp	4	11	≈15m	–	–	–	–
eZ	6	20	≈60m	–	–	–	–
Junebug	5	8	≈15m	–	–	3	–
Mephisto	10	49	≈60m	–	2	1	–
ozimodo	5	26	≈30m	–	2	–	–
RT	4	64	≈20m	1	–	–	–
Sciret	6	24	≈20m	–	–	–	–
Wordpress	4	10	≈20m	–	–	–	2
Sum	44	212	≈240m	1	4	4	2

Gambar III.2 Hasil *fuzzing* berdasarkan tingkat kompleksitas dan jenis bug

Penjelasan lebih lanjut dari gambar 3.2 ialah sebagai berikut:

1. *Forms* merupakan banyaknya halaman yang menjadi tujuan pengujian
2. *Inputs* merupakan banyaknya *form input* (*text fields*, *dropdown box*, dll) yang terdapat pada halaman yang di *crawl*.
3. *Time* merupakan waktu yang dibutuhkan untuk melakukan fuzzing pada aplikasi.
4. *E1 Resource exhaustion:* merupakan jenis *bug* yang disebabkan oleh increased response time dan mungkin tidak akan memberikan response.
5. *E2 Failure to check return values:* merupakan jenis *bug* yang terjadi disebabkan oleh not catching exceptions
6. *E3 No server side validation of input:* jenis bug yang disebabkan tidak adanya validasi atau penyaringan data di sisi pengguna (validating or sanitizing incoming data).
7. *E4 Incorrect use of HTTP status codes:* Kesalahan keamanan yang bukan disebabkan karena adanya *bug* tetapi pelanggaran semantik dijelaskan dalam protokol HTTP (RFC 2616)

BAB IV Kesimpulan dan saran

Fuzzing merupakan salah satu cara menguji aplikasi web yang murah dan teruji dapat membantu menemukan kerentanan yang ada. Walau hasil dari pengujian tidak dapat menjadi jaminan suatu aplikasi web tidak memiliki kerentanan. Dari hasil test yang tidak terlalu besar pada bab III.2 dapat menjelaskan potensi kerentanan aplikasi web yang ada dan mungkin masih banyak yang belum diketahui.

Banyak faktor yang masih dapat dikembangkan terkait dengan penelitian *fuzzing* seperti; menmbuat *input* yang efektif, mengetahui *code coverage* aplikasi yang diuji, menganalisis hasil *fuzzing* dalam skala besar, dll. Hal ini memungkinkan untuk lebih jauh lagi kedepannya untuk mengungkap kerentanan dari suatu aplikasi web.

Saran

1. Penelitian untuk melakukan atomatisasi terhadap tahapan yang masih manual seperti analisis *log file*.
2. Penelitian terkait menggunakan *log file* untuk menmbuat *input* baru ke aplikasi web.
3. Penelitian *monitoring* terhadap *code coverage* saat *fuzzing* berjalan akan membantu untuk menentukan waktu yang tepat berhenti melakukan *fuzzing*.
4. Menambah "*blacklisting*" pada form *input logout*.

DAFTAR PUSTAKA

[1] G. McGraw, *Software Security : Building Security In*, Addison-Wesley, 2006.

[2] OWASP. OWASP Top 10 – 2013: The Ten Most Critical Web Application security Risk. www.owasp.org, 2014. Diakses Maret 2016

[3] M. Sutton, A. Greene, P. Amini. *Fuzzing Brute Force Kerentanan Discovery*, United State of Amerika: Pearson Education. Inc , 2007.

[4] Li Li, Et Al., "The Appilication of Fuzzing in Web Software Security Vulnerabilities Test" in *International Conference on Information Technology and Applications*, Chengdu, 2013.

[5] *Smashing Web Apps: Applying Fuzzing to Web Applications and Web Services*, SPI Dynamics, 2007

[6] H. Rune, and E. Snekkenes, "Fuzz testing of web applications", Faculty of Computer Science and Media Technology - Gjøvik University College, Norway