Checklist Keamanan *Web*

By Restia Moegiono {ISO 27001 LI|QRMO|ECSA|CHFI|CEH}

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
Pencegahan terhadap Pengumpulan Informasi (*Information Gathering*)					
1.	Melakukan pencarian di *search engine* untuk memastikan tidak ada kebocoran informasi	▪ Pertimbangkan dengan cermat sensitivitas informasi desain dan konfigurasi sebelum di-*posting* secara *online* ▪ Meninjau secara berkala sensitivitas desain yang ada dan informasi konfigurasi yang di-*posting* secara *online*			
2.	Pastikan tidak terdapat *fingerprint* dari *web server*	▪ Mengaburkan informasi *server web* di *header*, seperti dengan *module* mod_headers di Apache ▪ Menggunakan *server reverse proxy* yang diperkuat untuk menciptakan lapisan keamanan tambahan antara *server web* dan internet ▪ Memastikan bahwa *server web* selalu diperbarui dengan perangkat			
3.	Meninjau metafile dari *web server* untuk memastikan tidak ada kebocoran informasi	▪ Menghapus *file metadata* (mis. .git) ▪ Tag meta yang berkaitan dengan IDE atau teknologi yang digunakan untuk mengembangkan aplikasi harus dihapus ▪ Pada *server* HTTP Apache, pastikan direktori seperti WEB-INF dan META-INF terlindungi. Jika izin untuk direktori dan subdirektori ditentukan dalam *file* .htaccess, pastikan *file* tersebut dilindungi menggunakan aturan "*deny all*"			
4.	Pastikan tidak terdapat informasi aplikasi yang berjalan di *web server*	▪ Pada *server web* Apache, tambahkan pada *file* konfigurasi berupa "ServerSignature Off" dan "ServerTokens Prod" ▪ Menggunakan *port* yang tidak standar ▪ Melakukan pemblokiran melalui *firewall* terhadap upaya *enumeration*			
5.	Meninjau konten halaman *web* untuk memastikan tidak ada kebocoran informasi	▪ Pertimbangkan dengan cermat konten di halaman *web* sebelum di-*posting* secara *online* ▪ Meninjau secara berkala konten di halaman *web* yang di-*posting* secara *online*			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
6.	Meninjau titik masuk dan memastikan tidak terdapat injeksi ke aplikasi	▪ Melakukan *hardening* pada aplikasi untuk mencegah adanya *entry point* dan *injection point* ▪ Meninjau aplikasi dengan *tools* yang dapat mendeteksi *attack surface,* seperti *tools Dynamic Application Security Testing (DAST)*			
7.	Pastikan tidak terdapat *fingerprint* dari *web application framework*	▪ *Header* HTTP *response* yang tidak wajib harus dihapus ▪ *Comment* dan *metadata* yang memberikan informasi aplikasi *web* harus dihapus ▪ *Error message* harus ditangani dengan benar, dan respons umum diberikan ▪ Informasi versi pada aplikasi *web* harus dihapus ▪ Halaman *server*/*framework* yang *default* harus dihapus jika tidak diperlukan lagi			
Keamanan pada Manajemen Konfigurasi dan Implementasi (*Configuration and Deployment Management*)					
8.	Meninjau konfigurasi infrastruktur jaringan untuk mendeteksi adanya kerentanan	▪ Membuat konfigurasi perangkat standar untuk setiap perangkat jaringan (seperti *router*, *switch* LAN, *switch* WAN) yang memenuhi kaidah *hardening* ▪ Melakukan pemeliharaan konfigurasi operasional saat ini untuk semua perangkat dan sejumlah versi yang berjalan sebelumnya ▪ Melakukan pencatatan kapan dan siapa yang melakukan perubahan konfigurasi ▪ Memantau pembaruan konfigurasi jaringan di seluruh elemen jaringan, dan menjaga kepatuhan dan standar			
9.	Meninjau konfigurasi *platform* aplikasi untuk memastikan tidak ada kerentanan	▪ Menghapus *file* yang *default* dan *file* yang dikenal pada *platform* aplikasi ▪ Menghapus informasi yang tersisa di lingkungan *production* ▪ Menerapkan mekanisme *logging* pada aplikasi secara memadai ▪ Memberikan prinsip *least privilege* pada pemberian hak akses pada aplikasi			
10.	Meninjau *file extension handling* untuk	▪ Melakukan validasi ekstensi dengan menerapkan regex ▪ Menerapkan *allow list* pada ekstensi yang dibolehkan ▪ Menerapkan pemblokiran pada ekstensi jenis *file* yang berpotensi berbahaya pada sistem			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
	memastikan tidak ada informasi sensitif	▪ Menerapkan validasi Content-Type dan *file signature* untuk mencegah pengguna mengunggah *file* dengan jenis yang salah secara tidak sengaja			
11.	Meninjau *backup* lama dan *file* tanpa referensi untuk memastikan tidak ada informasi sensitif	▪ Jangan mengedit *file* di *server web* atau sistem *file* pada *server* aplikasi. ▪ Memeriksa dengan cermat aktivitas lain yang dilakukan pada sistem *file* yang diekspos oleh *server web* ▪ *File* data, *file* log, *file* konfigurasi, dan lain-lain harus disimpan dalam direktori yang tidak dapat diakses oleh *server web* ▪ *Snapshot* dari sistem *file* tidak boleh diakses melalui *web* jika *root* dokumen ada pada sistem *file*			
12.	Meninjau antarmuka *administrator* aplikasi untuk memastikan tidak ada kerentanan	▪ Memisahkan *interface* manajemen pada VLAN manajemen yang khusus ▪ Membatasi alamat IP yang diizinkan masuk ke jaringan manajemen hanya pada perangkat manajemen khusus, seperti *server jump* atau *bastion host* ▪ Menggunakan *service route* untuk mengakses layanan eksternal ▪ Menggunakan *password* yang kuat pada akun administrator ▪ Men-*scanning* semua lalu lintas yang dituju untuk *interface* manajemen			
13.	Meninjau HTTP *Methods* untuk memastikan tidak ada kerentanan	▪ Gunakan HTTP *method* yang aman dan sesuai kebutuhan ▪ Hanya *header* yang diperlukan yang diizinkan, dan *header* yang diizinkan dikonfigurasi dengan benar ▪ Tidak ada solusi yang diterapkan untuk mem-*bypass* keamanan yang diterapkan oleh *user-agents, frameworks,* atau *server web*			
14.	Memastikan penerapan HTTP *Strict Transport Security*	▪ Agar situs ingin domainnya disertakan dalam *preload list* HSTS yang dikelola oleh Chrome (dan digunakan oleh Firefox dan Safari), gunakan *header* berikut: ▪ Mengirimkan *preload directive* dari situs untuk dapat menimbulkan konsekuensi permanen dan mencegah pengguna mengakses situs dan subdomainnya kembali ke HTTP:			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		`Strict-Transport-Security: max-age=31536000; includeSubDomains; preload`			
15.	Memastikan tidak ada kerentanan pada RIA *Cross Domain Policy*	Menentukan hanya domain tepercaya yang dimasukkan ke dalam *file cross-domain policy*			
16.	Memastikan keamanan pada *file permission*	Mengatur izin pada *file* dan direktori dengan benar sehingga pengguna yang tidak berwenang tidak dapat mengakses sumber daya penting jika tidak diperlukan			
17.	Memastikan pengambilalihan subdomain tidak bisa dilakukan	▪ Untuk mengurangi risiko pengambilalihan subdomain, sumber daya dari DNS *record* yang rentan harus dihapus dari DNS *zone* ▪ Melakukan *monitoring* secara terus-menerus dan pemeriksaan secara berkala			
18.	Memastikan keamanan pada *Cloud Storage*	▪ Mereviu *shared responsibility model* pada penyedia *cloud storage* ▪ Mengaktifkan *two-factor authentication* (2FA) ▪ Memberikan akses ke pengguna dengan prinsip *least privilege* ▪ Melindungi akun super admin ▪ Meninjau dan menghapus secara berkala: pengguna, izin, peran, kredensial, dan kebijakan yang tidak digunakan ▪ Memantau akses dan aktivitas pengguna untuk mencari anomali			
Keamanan pada Manajemen Identitas (*Identity Management*)					
19.	Memastikan keamanan *role* pada aplikasi	▪ Menentukan data dan *resource* yang aksesnya harus dibatasi ▪ Membuat *role* dengan kebutuhan akses yang sama ▪ Menghindari membuat terlalu banyak *role* karena berisiko membuat kontrol akses berbasis pengguna ▪ Menyejajarkan *role* dengan pegawai dalam organisasi ▪ Menganalisis bagaimana *role* dapat diubah dan bagaimana pegawai baru dapat didaftarkan dan akun lama dapat dihentikan ▪ Memastikan RBAC di seluruh unit kerja organisasi yang terintegrasi di seluruh sistem ▪ Menyelenggarakan pelatihan pegawai, sehingga pegawai dapat mengetahui prinsip-prinsip RBAC			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		▪ Melakukan audit untuk memastikan implementasi berjalan sesuai rencana			
20.	Memastikan keamanan pada proses registrasi pengguna	Menerapkan persyaratan identifikasi dan verifikasi yang sesuai dengan persyaratan keamanan informasi untuk pelindungan kredensial			
21.	Memastikan keamanan pada proses penyediaan akun	▪ Mengimplementasikan *Identify and Access Management* (IAM) secara terpusat ▪ Menerapkan prinsip *least privilege* pada pemberian hak akses ▪ Menggunakan *software* penyediaan akun otomatis untuk menyederhanakan pembuatan akun dan mengatur hak akses ▪ Melakukan pencabutan hak akses pada karyawan atau vendor yang selesai masa kerja/kontrak ▪ Membuat panduan yang jelas untuk berbagi atau membatasi akses ▪ Mendukung sumber daya tim TI dengan menyediakan *tools* dan aplikasi yang tepat ▪ Menambahkan lapisan keamanan ekstra dengan *multi-factor authentication* (MFA) ▪ Menerapkan *risk-based authentication* (RBA) ▪ Mempertimbangkan audit dan kepatuhan dengan cermat			
22.	Memastikan keamanan pada mekanisme identifikasi akun dan tidak ada akun pengguna yang mudah ditebak	▪ Memastikan aplikasi mengembalikan *error message* umum yang konsisten sebagai *response* terhadap nama akun, *password*, atau kredensial pengguna lain yang tidak valid yang dimasukkan selama proses *login* ▪ Memastikan akun sistem *default* dan akun pengujian dihapus sebelum sistem dirilis ke *production* (atau memaparkannya ke jaringan yang tidak tepercaya)			
23.	Memastikan tidak terdapat kebijakan *username* yang lemah atau bahkan kebijakan *username* tidak diberlakukan sama sekali	Memastikan aplikasi mengembalikan *error message* umum yang konsisten sebagai respons terhadap nama akun, *password*, atau kredensial pengguna lain yang tidak valid yang dimasukkan selama proses *login*			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
Keamanan pada Autentikasi (*Authentication*)					
24.	Memastikan keamanan kredensial yang dikirimkan melalui saluran terenkripsi	Menggunakan HTTPS untuk seluruh situs *web* dan menerapkan HSTS dengan mengalihkan HTTP apa pun ke HTTPS agar dapat: ▪ Mencegah penyerang mengubah interaksi dengan server *web* (termasuk menempatkan *malware* JavaScript melalui *router* yang disusupi) ▪ Menghindari kehilangan pelanggan karena peringatan situs yang tidak aman. *Browser* akan menandai situs *web* berbasis HTTP sebagai situs yang tidak aman			
25.	Memastikan tidak ada kredensial yang *default*	▪ Memastikan *password* yang digunakan untuk akses administratif pada aplikasi dikelola dengan aman ▪ Memastikan *password default* telah diubah dan akun pengguna *default* telah dinonaktifkan ▪ Memeriksa *database* pengguna untuk menncegah adanya kredensial *default* dan *field password* yang kosong ▪ Memeriksa *code* untuk *username* dan *password* yang di-*hard code* ▪ Memeriksa *file* konfigurasi yang berisi *username* dan *password* ▪ Memeriksa kebijakan *password* dan implementasinya			
26.	Memastikan tidak ada mekanisme penguncian (*lock out*) yang lemah	Menerapkan mekanisme *unlock* akun tergantung pada tingkat risiko. Urutan jaminan terendah hingga tertinggi: ▪ *Lockout* dan *unlock* kunci berbasis waktu ▪ Layanan *unlock* secara mandiri (mengirimkan *email unlock* ke alamat *email* yang terdaftar) ▪ *Unlock* administrator manual ▪ *Unlock* administrator manual dengan identifikasi pengguna positif			
27.	Memastikan tidak ada mekanisme *bypass* pada skema autentikasi	▪ Memperbarui semua sistem, aplikasi, perangkat lunak, dan OS ▪ Mem-*patch* semua kerentanan dan menginstal program antivirus yang tepercaya ▪ Menerapkan kebijakan autentikasi yang aman dan kuat ▪ Memastikan semua sistem, *folder*, aplikasi dilindungi *password* ▪ Me-*reset password* yang *default* dengan *password* unik yang kuat dan merotasi *password* secara berkala			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		▪ Tidak mengekspos protokol autentikasi di *script browser web* sisi *client* ▪ Memastikan bahwa *session* ID pengguna dan *cookie* dienkripsi ▪ Memvalidasi semua *input* pengguna di sisi *server* ▪ Pengiriman semua *cookie* dan *session* data melalui saluran terenkripsi			
28.	Memastikan tidak ada kerentanan dari *remember password*	▪ Mengikuti *best practice* manajemen sesi ▪ Memastikan tidak ada kredensial yang disimpan dalam *plaintext* atau dapat diambil dengan mudah dalam bentuk yang *encoded* atau dienkripsi dalam mekanisme penyimpanan *browser*, penyimpanan kredensial di sisi *server* dan mengikuti praktik penyimpanan *password* yang baik			
29.	Memastikan tidak ada kerentanan dari *cache browser*	Menyimpan *cache* dalam keadaan terenkripsi			
30.	Memastikan penerapan kebijakan *weak password*	Untuk mengurangi risiko *password* yang mudah ditebak dan memfasilitasi akses tidak sah, ada 2 (dua) solusi: ▪ Menerapkan kontrol autentikasi tambahan (*two factor authentication*) ▪ Menerapkan kebijakan *password* yang kuat denngan memastikan panjang, kompleksitas, penggunaan kembali, dan umur *password*			
31.	Memastikan tidak ada kerentanan dari jawaban pertanyaan keamanan	Menggunakan pertanyaan keamanan dengan ketentuan berikut: ▪ Mudah diingat: pengguna harus dapat mengingat jawaban atas pertanyaannya, bahkan setelah bertahun-tahun setelah membuat akun ▪ Konsisten: jawaban atas pertanyaan tidak boleh berubah seiring berjalannya waktu ▪ Dapat diterapkan: pengguna harus dapat menjawab pertanyaan ▪ Rahasia: jawaban atas pertanyaan tersebut pasti sulit diperoleh oleh penyerang ▪ Spesifik: jawabannya harus jelas bagi pengguna			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
32.	Memastikan tidak ada kerentanan pada fungsionalitas *change/reset password*	Terdapat pelindungan pada fungsi perubahan atau *reset password*, seperti mengharuskan pengguna untuk melakukan autentikasi ulang atau menampilkan layar konfirmasi kepada pengguna selama proses berlangsung			
33.	Memastikan tidak ada autentikasi yang lebih lemah di saluran alternatif	Pastikan kebijakan autentikasi yang konsisten diterapkan di semua saluran (saluran utama dan alternatif) sehingga autentikasi dilakukan secara aman			
Keamanan pada Otorisasi (*Authorization*)					
34.	Memastikan tidak ada kerentanan *directory traversal* yang melibatkan *file*	▪ Aplikasi harus memvalidasi *input* pengguna sebelum memprosesnya. Idealnya, validasi harus dibandingkan dengan *whitelist* nilai yang diizinkan. Jika fungsi yang diperlukan tidak memungkinkan, maka validasi harus memverifikasi bahwa *input* hanya berisi konten yang diizinkan, seperti karakter alfanumerik. ▪ Setelah memvalidasi *input* yang diberikan, aplikasi harus menambahkan *input* ke direktori dasar dan menggunakan API sistem *file platform* untuk meng-*canonicalize* *path*. Ini harus memverifikasi bahwa *path* yang di-*canonicalize* dimulai dengan direktori dasar yang diharapkan.			
35.	Memastikan tidak ada mekanisme *bypass* pada skema otorisasi	▪ Selalu perbarui sistem, perangkat lunak, aplikasi, jaringan, dan sistem operasi ▪ Menginstal program antivirus yang tepercaya dan mem-*patch* semua kerentanan ▪ Menerapkan kebijakan autentikasi yang kuat dan aman ▪ Memastikan semua sistem, aplikasi, dan *folder* dilindungi *password* ▪ Menerapkan *password* yang unik dan kuat daripada *password* yang *default* ▪ Tidak mengekspos protokol autentikasi di *script browser web* sisi *client*, dan validasi *input* pengguna di sisi *server* ▪ Tidak menggunakan *interpreter* SQL eksternal ▪ Mengenkripsi *session* ID dan *cookie* pengguna			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
36.	Memastikan tidak ada kerentanan eskalasi hak istimewa (*privilege escalation*)	▪ Mengelola *identity life cycle*, termasuk penyediaan dan pencabutan identitas dan akun untuk memastikan tidak ada akun *orphan* yang dapat dibajak ▪ Menerapkan manajemen kredensial yang kuat ▪ Menerapkan *least privilege* ▪ Menerapkan kontrol dan perlindungan aplikasi tingkat lanjut untuk menerapkan kontrol granular atas semua akses aplikasi, komunikasi, dan upaya peningkatan hak istimewa ▪ Memantau dan kelola semua *session* yang memiliki hak istimewa untuk mendeteksi dan dengan cepat mengatasi aktivitas mencurigakan apa pun yang mungkin mengindikasikan akun yang dibajak atau upaya ilegal untuk meningkatkan hak istimewa atau *lateral movement* ▪ Melakukan *hardening* sistem dan aplikasi ▪ Menerapkan manajemen kerentanan ▪ Memantau dan mengelola akses jarak jauh yang aman			
37.	Memastikan tidak ada kerentanan *Insecure Direct Object References* (IDOR)	▪ Menerapkan pemeriksaan kontrol akses untuk setiap objek yang coba diakses pengguna dan gunakan *identifier* yang kompleks sebagai tindakan pertahanan mendalam ▪ Hindari mengekspos *identifier* di URL dan POST *body* jika memungkinkan ▪ Verifikasi izin pengguna setiap kali upaya akses dilakukan ▪ Gantikan *identifier* numerik dengan *identifier* acak yang lebih kompleks			
Keamanan pada Manajemen Sesi (*Session Management*)					
38.	Memastikan keamanan pada skema manajemen sesi	▪ Untuk menjaga keadaan terautentikasi dan melacak *progress* pengguna dalam aplikasi *web*, aplikasi menyediakan *identifier session* kepada pengguna (*session* ID atau token) yang ditetapkan pada waktu pembuatan *session*, dan dibagikan serta dipertukarkan oleh pengguna dan aplikasi *web* ▪ Pada pertukaran *session* ID berdasarkan *cookie* menggunakan fitur keamanan berupa atribut *cookie*:			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		○ Atribut *secure*: memerintahkan *browser web* untuk mengirim *cookie* hanya melalui koneksi HTTPS (SSL/TLS) terenkripsi, sehingga mencegah dari serangan *Man-in-the-Middle* (MitM) ○ Atribut HttpOnly: memerintahkan *browser web* untuk tidak mengizinkan *script* dapat mengakses *cookie* melalui objek DOM document.cookie untuk mencegah pencurian *session* ID melalui serangan XSS ○ Atribut SameSite: mencegah *browser* mengirimkan *cookie* bertanda SameSite dengan *request cross-site* untuk memitigasi risiko kebocoran informasi *cross-origin* dan memberikan perlindungan terhadap serangan *cross-site request forgery*. ○ Atribut Domain: memerintahkan *browser web* untuk hanya mengirim *cookie* ke domain tertentu dan semua subdomain. ○ Atribut path: memerintahkan *browser web* untuk hanya mengirimkan *cookie* ke direktori atau subdirektori tertentu (atau jalur atau sumber daya) dalam aplikasi web ○ Atribut Max-Age atau Expire: *cookie* tersebut akan dianggap sebagai *cookie* persisten dan akan disimpan pada *disk* oleh *browser web* hingga waktu kedaluwarsa			
39.	Memastikan keamanan pada atribut *cookies*	Konfigurasi atribut *cookie* paling aman yang disarankan, yaitu: `Set-Cookie: __Host-SID=<session token>; path=/; Secure; HttpOnly; SameSite=Strict`			
40.	Memastikan keamanan pada *session fixation*	▪ Menerapkan pembaruan token *session* setelah pengguna berhasil melakukan autentikasi ▪ Aplikasi harus selalu membatalkan *session* ID yang ada terlebih dahulu sebelum mengautentikasi pengguna, dan jika autentikasi berhasil, berikan *session* ID yang lain			
41.	Memastikan tidak ada variabel session yang terekspos	▪ *Session* ID harus dilindungi pada kondisi *in transit* dengan cara enkripsi dan diterapkan untuk setiap *request* atau *response* ketika *session* ID diteruskan ▪ *Session* ID harus dikirim secara terenkripsi dan tidak boleh di-*cache*, baik oleh *cache* perantara dan *cache* lokal			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		▪ Atribut `Expires: 0` dan `Cache-Control: max-age=0` digunakan untuk memastikan *cache* tidak mengekspos data			
42.	Memastikan tidak ada kerentanan *Cross Site Request Forgery* (CSRF)	▪ Memeriksa apakah *framework* memiliki pelindungan CSRF *built-in,* jika ada maka gunakanlah. Jika *framework* tidak memiliki pelindungan CSRF *built-in*, tambahkan token CSRF ke semua *request* perubahan status dan validasi permintaan tersebut di *backend* ▪ Untuk perangkat lunak *stateful* gunakan *synchronizer token pattern* ▪ Untuk perangkat lunak *stateless*, gunakan *double submit cookies* ▪ Untuk situs berbasis API yang tidak menggunakan tag <form>, pertimbangkan untuk menggunakan *header* yang *custom request* ▪ Melaksanakan setidaknya 1 (satu) mitigasi berikut: o Menggunakan atribut *cookie* SameSite untuk *session cookie* sesi tetapi jangan menetapkan *cookie* khusus untuk domain o Menerapkan pelindungan berbasis interaksi pengguna untuk operasi yang sangat sensitif o Memverifikasi *origin* dengan *header* standar ▪ Ingatlah bahwa Cross-Site Scripting (XSS) apa pun dapat digunakan untuk mengalahkan semua teknik mitigasi CSRF! ▪ Jangan gunakan *request* GET untuk operasi perubahan status			
43.	Memastikan keamanan pada fungsionalitas *logout*	▪ Tersedianya UI yang memungkinkan pengguna untuk melakukan *logout* secara manual ▪ Penghentian *session* setelah jangka waktu tertentu pada kondisi *idle* (batas waktu *session*) ▪ Pembatalan yang tepat dari status *session* sisi *server*			
44.	Memastikan keamanan pada *session timeout*	▪ Fungsi *logout* secara efektif menghancurkan semua token *session*, atau setidaknya menjadikannya tidak dapat digunakan kembali ▪ *Server* melakukan pemeriksaan yang tepat pada status *session*, sehingga melarang penyerang melakukan *replay* pada *session identifier* yang sebelumnya dihancurkan ▪ Batas waktu diberlakukan dan diterapkan dengan benar pada *server*, namun jika *server* menggunakan waktu kedaluwarsa yang dibaca			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		dari token *session* yang dikirim oleh *client*, maka token tersebut harus dilindungi secara kriptografis			
45.	Memastikan tidak ada kerentanan *session puzzling*	Variabel *session* hanya boleh digunakan untuk 1 (satu) tujuan yang konsisten			
46.	Memastikan tidak ada kerentanan *session hijacking*	▪ Menggunakan HTTPS untuk memastikan *session* diamankan di setiap tahap interaksi ▪ Menggunakan pertahanan sisi klien yang kuat untuk melindungi browser *client* dan *session cookie* dari serangan XXS ▪ Menginstal *framework* manajemen *cookie session web* karena *framework web* menyederhanakan manajemen *session* dengan menghasilkan *session cookie* yang lebih lama dan *random* ▪ Merotasi *session key* setelah autentikasi, sehingga mempersulit pembajak *session* untuk mencari tahu *session* pengguna meskipun mengetahui *key*-nya ▪ Menggunakan *Intrusion Detection System* (IDS) dan *Intrusion Prevention System* (IPS)			
Keamanan pada Validasi Input (*Input Validation*)					
47.	Memastikan tidak ada kerentanan *Reflected Cross Site Scripting*	▪ Mem-*filter input* pada saat *entry* seketat mungkin ▪ Melakukan *encoding* data pada *output* data yang dapat dikontrol pengguna dikeluarkan dalam *response* HTTP, hal ini dapat mencegah *output* ditafsirkan sebagai konten aktif. Bergantung pada konteks *output*, hal ini mungkin memerlukan penerapan kombinasi *encoding* HTML, URL, JavaScript, dan CSS ▪ Menggunakan *header response* yang sesuai untuk mencegah XSS dalam *response* HTTP yang tidak dimaksudkan untuk berisi HTML atau JavaScript apa pun ▪ Menggunakan *Content Security Policy* (CSP) untuk mengurangi tingkat keparahan kerentanan XSS yang masih terjadi			
48.	Memastikan tidak ada kerentanan *Stored Cross Site Scripting*				

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
50.	Memastikan tidak ada kerentanan HTTP *Parameter Pollution*	▪ Hanya menerima parameter yang diketahui/diharapkan, parameter yang tidak dikenali harus diperlakukan sebagai anomali yang mengakibatkan *error* atau permintaan diabaikan. ▪ Memvalidasi parameter secara ketat untuk jenis, format, dan rentang. Jika suatu parameter diharapkan berupa angka, semua input non-numerik harus ditolak ▪ *Instance* parameter tunggal yaitu aplikasi dirancang untuk hanya menerima *instance* pertama dari parameter karena menggunakan parameter yang sama beberapa kali dapat mengeksploitasi HTTP *parameter pollution* ▪ Melakukan sanitasi pada *input* untuk menghilangkan *string* yang berpotensi membahayakan dari *input* pengguna ▪ Menggunakan *header security*, seperti *Content Security Policy* untuk mengurangi risiko serangan *code injection* ▪ Memperbarui mekanisme keamanan, seperti *Web Application Firewall*, *Intrusion Detection System* (IDS), dan pertahanan keamanan lainnya secara berkala untuk lebih mengenali dan memblokir bentuk-bentuk serangan baru ▪ Meninjau *code* secara berkala untuk mengidentifikasi dan mem-*patch* potensi kerentanan sebelum dapat dieksploitasi			
51.	Memastikan tidak ada kerentanan SQL *injection*	▪ Penggunaan *statement* yang telah disiapkan dengan variabel *binding* (*parameterized query*), sehingga memungkinkan *database* membedakan antara *code* dan data, apapun *input* pengguna yang diberikan ▪ Menggunakan *stored procedure* sesuai bahasa pemrograman ▪ Memvalidasi *input* dengan *allow-list* untuk memastikan pertahanan yang paling tepat ▪ Tidak menggunakan semua *input* yang diberikan pengguna. Teknik ini hanya boleh digunakan sebagai upaya terakhir, jika tidak ada satu pun cara di atas yang dapat dilakukan ▪ Menerapkan *least privilege* untuk meminimalkan potensi kerusakan akibat serangan SQL *injection* yang berhasil			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
52.	Memastikan tidak ada kerentanan LDAP *injection*	▪ Membatasi semua variabel menggunakan fungsi *encoding* LDAP yang tepat: o Membatasi *Distinguished Name* (DN) agar menggunakan nama yang unik o Membatasi *filter* pencarian ▪ Menggunakan *framework* yang secara otomatis melindungi dari LDAP *injection* ▪ Menerapkan *least privilege* untuk meminimalkan potensi kerusakan akibat serangan LDAP *injection* yang berhasil ▪ Mengaktifkan *bind authentication* agar penyerang tidak akan dapat melakukan serangan LDAP *injection* karena verifikasi dan pemeriksaan otorisasi yang dilakukan terhadap kredensial valid yang diberikan oleh pengguna ▪ Memvalidasi *input* dengan *allow-list* untuk mendeteksi *input* yang tidak sah sebelum diteruskan ke *query*LDAP			
53.	Memastikan tidak ada kerentanan XML *injection*	▪ Melakukan sanitasi pada *input* pengguna untuk menyaring karakter yang tidak dapat diterima ▪ Menentukan *input* yang diperbolehkan ▪ Mengawasi XML *parser* untuk mengidentifikasi kerentanan apa pun Pastikan juga untuk melarang *Document Type Definition* (DTD) ▪ Menerapkan *Content Security Policy* (CSP) untuk membatasi jenis *resource* yang dapat dimuat pengguna saat menggunakan situs menggunakan daftar *predetermined resources*			
54.	Memastikan tidak ada kerentanan SSI *injection*	▪ Jangan gunakan SSI saat men-*develop* situs *web* dimana konten dinamis dapat dimuat ke halaman *web* menggunakan cara lain, seperti JavaScript dan AJAX ▪ Jangan mencampur *input* pengguna dan halaman SSI untuk menurunkan kemungkinan keberhasilan serangan SSI ▪ Hindari menggunakan halaman .stm, .shtm, dan .shtml, disarankan menggunakan halaman .htm, .html ▪ Memvalidasi *input* pengguna			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
55.	Memastikan tidak ada kerentanan XPath *injection*	▪ Memvalidasi *input* untuk memastikan bahwa aplikasi hanya menerima *input* yang sah ▪ *Parameterization*, yaitu *query* dikompilasi sebelumnya dan dengan demikian meneruskan *input* pengguna sebagai parameter, bukan *expression*			
56.	Memastikan tidak ada kerentanan IMAP SMTP *injection*	▪ Melakukan sanitasi pada *input* pengguna ▪ Menggunakan *Web Application Firewall* (WAF) untuk pelindungan yang lebih baik ▪ Memeriksa HTTP *method* dan menerapkan *filter* pada *request* GET dan POST sehingga penyerang tidak dapat mengubahnya ▪ Parameter yang dikirimkan pengguna ke *server* harus dibatasi dengan *whitelist*			
57.	Memastikan tidak ada kerentanan *code injection*	▪ Memanfaatkan *whitelist* untuk memvalidasi *input*, sehingga membantu mengurangi risiko penyerang mengeksekusi *code* berbahaya ▪ *Encoding output* HTML, sehingga data pengguna dapat ditampilkan tetapi tidak dieksekusi sebagai *code* ▪ Menggunakan sistem tipe statis untuk menerapkan *language separation* untuk memeriksa kontrol deklaratif tanpa tambahan *runtime overhead* ▪ Memanfaatkan *query* berparameter dan API berbasis kriteria untuk menginterpretasikan *string* data pengguna, sehingga dapat memastikan bahwa API tidak menerima nilai *string* apa pun selain yang ditentukan ▪ Menghindari penggunaan fungsi yang tidak aman dalam *source code* dengan menggunakan fitur khusus yang aman dan berdedikasi untuk memproses *input* yang diberikan pengguna ▪ Menggunakan atribut HttpOnly pada *cookie* untuk menonaktifkan interaksi *script* sisi *client* ▪ Menggunakan *tools code static analyzer* otomatis untuk menemukan dan menghilangkan vektor *injection* dalam *source code*			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		▪ Hindari *serialization* pada *code* karena tidak dapat membersihkan karakter tidak tepercaya dalam *reguler expression* dengan benar			
58.	Memastikan tidak ada kerentanan *command injection*	▪ Menghindari *call* ke sistem dan *input* pengguna untuk mencegah penyerang memasukkan karakter ke *command* OS ▪ Menerapkan validasi *input* untuk mencegah serangan seperti XSS dan SQL *injection* ▪ Membuat *whitelist* dari kemungkinan *input* untuk memastikan sistem hanya menerima *input* yang telah disetujui sebelumnya ▪ Hanya menggunakan API yang aman saat menjalankan *command* sistem seperti execFile() ▪ Menggunakan execFile() secara aman untuk mencegah pengguna mendapatkan kendali melalui program			
59.	Memastikan tidak ada kerentanan *format string injection*	▪ Menentukan format string sebagai bagian dari program, bukan sebagai *input* ▪ Jika memungkinkan, jadikan format string sebagai konstanta. Ekstrak semua bagian variabel sebagai argumen lain untuk *call* tersebut ▪ Jika dua praktik di atas tidak memungkinkan, gunakan pertahanan seperti FormatGuard			
61.	Memastikan tidak ada kerentanan HTTP *splitting smuggling*	▪ Mencegah HTTP *Request Smuggling* (HRS), antara lain: ○ Memprioritaskan *header Transfer-Encoding* (TE) daripada *header Content-Length* (CL). Untuk mencegah serangan TE:CL dan CL:TE ○ Melarang *request* dengan TE dan CL serta *header* CL ganda untuk mencegah serangan CL:CL, serta serangan TE:CL dan CL:TE, gunakan alternatif yang lebih efektif ini dengan memprioritaskan TE dibandingkan CL ○ Melarang *header* TE yang rusak dan memproses beberapa nilai TE dengan benar ▪ Mencegah HTTP *Response Splitting,* antara lain: ○ Melakukan sanitasi nilai dengan benar di *header* `*location*` dan mem-*filter* karakter seperti \r dan \n			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		○ Jangan biarkan pengguna mengontrol seluruh nilai di *header* `*location*` ○ Saat menggunakan *Content-Security-Policy*, kebijakan tersebut juga harus ditentukan dalam HTML			
62.	Memastikan keamanan pada HTTP *incoming request*	▪ Melakukan *monitoring* pada *incoming* atau *outgoing* HTTP *request* ▪ Melakukan *blocking* pada HTTP *request* yang mencurigakan atau tidak diperlukan			
63.	Memastikan tidak ada kerentanan *host header injection*	▪ Memvalidasi *input* pengguna untuk memastikan bahwa *input* hanya berisi karakter dan *pattern* yang diharapkan ▪ Menggunakan *whitelist* untuk memvalidasi *input* ▪ Menetapkan nilai *header Host* secara eksplisit untuk membantu mencegah penyerang memasukkan nilai berbahaya ▪ Menghindari penggabungan string untuk membuat *header* HTTP, disarankan gunakan *library* atau fungsi yang menangani format nilai *header* dengan benar ▪ Menggunakan HTTPS untuk mencegah serangan MITM dan memastikan bahwa semua *request* dan *response* HTTP sudah dienkripsi			
64.	Memastikan tidak ada kerentanan *server-side template injection*	▪ Jangan pernah izinkan pengguna memodifikasi atau membuat *template* ▪ Melakukan sanitasi pada *input* untuk mendeteksi dan menghapus konten yang berpotensi berbahaya sebelum digunakan pada *template* ▪ Menggunakan *sandbox* untuk menyediakan lingkungan tertutup, dimana modul dan fitur berisiko dapat dinonaktifkan ▪ Menggunakan *logic less template* yang sebisa mungkin memisahkan *rendering* visual dan interpretasi *code*. Mustache adalah salah satu yang paling populer			
65.	Memastikan tidak ada kerentanan *server-side request forgery*	▪ *Whitelist* domain atau alamat apa pun yang diakses aplikasi pada DNS ▪ Jangan kirim *raws response* dari *server* ke *client* dan *response* yang diterima *client* perlu ditentukan			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		▪ Menerapkan skema URL dan jika terdapat skema lain, maka pastikan skema tersebut hanya dapat diakses dari bagian yang perlu mengaksesnya ▪ Mengaktifkan autentikasi di semua layanan apa pun yang berjalan di dalam jaringan meskipun layanan tersebut tidak memerlukannya untuk mencegah layanan tersebut dapat dieksploitasi ▪ Menerapkan sanitasi dan validasi *input* untuk memastikan tidak ada *input* berbahaya yang masuk			
Keamanan pada Penanganan Kesalahan (*Error Handling*)					
66.	Memastikan *error handling* sudah tepat	▪ Menerapkan kebijakan khusus tentang *error handling* dan mendokumentasikan, termasuk jenis *error* yang akan ditangani, informasi apa yang akan dilaporkan kembali ke pengguna, dan informasi apa yang akan dicatat. Semua tim pengembang perlu memahami kebijakan tersebut dan memastikan bahwa *code* sudah memenuhi ketentuan pada kebijakan ▪ Memastikan bahwa situs dibangun untuk menangani semua kemungkinan *error* dengan baik ▪ Ketika *error* terjadi, situs harus merespons dengan hasil yang dirancang khusus yang bermanfaat bagi pengguna tanpa mengungkapkan detail internal yang tidak perlu ▪ *Class* pada *error* tertentu harus dicatat untuk membantu mendeteksi kelemahan implementasi di situs dan adanya upaya peretasan			
Pencegahan pada Kriptografi yang Lemah (*Weak Cryptography*)					
68.	Memastikan tidak ada *Transport Layer Security* (TLS) yang lemah	▪ Menggunakan konfigurasi *server* dengan kategori kompabilitas modern ▪ Menggunakan sertifikat digital dengan kekuatan dan valididas yang kuat ▪ Memastikan TLS *security* tidak bisa di-*bypass* dan telah diimplementasikan dengan baik			
69.	Memastikan keamanan pada *padding* Oracle	▪ Menggunakan mode operasi enkripsi yang lebih kuat seperti *Galois/Counter Mode* (GCM) atau *Offset Codebook Mode* (OCB)			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		yang menggunakan penghitung untuk setiap blok data, kemudian menggunakan nomor tersebut untuk membuat *ciphertext*, sehingga tidak rentan terhadap serangan *padding oracle* ▪ Menerapkan kontrol *error handling* yang baik untuk mengurangi peluang keberhasilan penyerang karena serangan *padding oracle* bergantung pada kebocoran informasi ketika mengembalikan *error message* yang umum ▪ Menerapkan *Message Authentication Code* (MAC) untuk melindungi integritas data serta keasliannya, dengan memungkinkan verifikator mendeteksi perubahan apa pun pada konten pesan menggunakan kunci rahasia ▪ Membatasi jumlah *request* yang masuk dari sumber yang sama dapat menghentikan serangan *padding Oracle* karena menolak akses penyerang			
70.	Memastikan tidak ada informasi sensitif yang dikirim melalui saluran yang tidak terenkripsi	Menggunakan saluran aman (terenkripsi) untuk mentransmisikan data			
71.	Memastikan tidak ada enkripsi yang lemah	Menggunakan algoritma kriptografi yang disarankan, misalnya: ▪ Saat menggunakan AES128 atau AES256, IV (*Initialization Vector*) harus acak dan tidak dapat diprediksi ▪ Untuk enkripsi asimetris, gunakan *Elliptic Curve Cryptography* (ECC) dengan kurva aman seperti enkripsi Curve25519 atau RSA dengan kunci minimum 2048bit ▪ Saat menggunakan RSA dalam *signature*, PSS *padding* direkomendasikan ▪ Algoritma hash/enkripsi yang lemah tidak boleh digunakan seperti MD5, RC4, DES, Blowfish, SHA1. RSA atau DSA 1024-bit, ECDSA 160-bit (kurva elips), 2TDEA 80/112-bit (dua kunci triple DES) ▪ Persyaratan panjang kunci minimum harus dipenuhi ▪ Penggunaan SSH, *mode* CBC tidak boleh digunakan			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		▪ Jika algoritma enkripsi simetris digunakan, *mode* ECB (*Electronic Code Boo*k) tidak boleh digunakan ▪ Ketika PBKDF2 digunakan untuk *password hash*, parameter iterasi disarankan lebih dari 10.000			
Keamanan pada Logika Bisnis (*Business Logic*)					
72.	Memastikan keamanan pada validasi data logika bisnis	▪ Memelihara dokumen desain dan *data flow* yang jelas untuk semua transaksi dan alur kerja, dengan mencatat setiap asumsi yang dibuat pada setiap tahap ▪ Menulis *code* sejelas mungkin. Jika sulit memahami apa yang seharusnya terjadi, maka akan sulit menemukan kesalahan logika. Idealnya, *code* yang ditulis dengan baik tidak memerlukan dokumentasi untuk memahaminya. Dalam kasus-kasus rumit yang tidak dapat dihindari, pembuatan dokumentasi yang jelas sangat penting untuk memastikan bahwa pengembang dan penguji lain mengetahui asumsi apa yang dibuat dan perilaku apa yang diharapkan ▪ Mencatat setiap referensi ke *code* lain yang digunakan pada setiap komponen. Pikirkan tentang dampak dari ketergantungan jika pihak jahat memanipulasinya dengan cara yang tidak biasa			
73.	Memastikan pemalsuan *request* tidak bisa dilakukan	Aplikasi harus cukup cerdas dan dirancang dengan *business logic* yang dapat mencegah penyerang memprediksi dan memanipulasi parameter untuk mem-*bypass business logic* terprogram atau bisnis, atau mengeksploitasi fungsionalitas tersembunyi/tidak terdokumentasi seperti *debugging*			
	Memastikan integritas pada proses bisnis	▪ Aplikasi harus mengikuti kontrol akses yang ketat tentang bagaimana data dan artefak dapat dimodifikasi dan dibaca, dan melalui saluran tepercaya yang menjamin integritas data ▪ *Logging* yang tepat dan aman harus dilakukan untuk meninjau dan memastikan bahwa tidak ada akses atau modifikasi yang tidak sah yang terjadi ▪ Jika aplikasi memperlihatkan nilai yang terkait dengan aturan bisnis (seperti kuantitas) sebagai *field* yang tidak dapat diedit, maka			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		aplikasi harus menyimpan salinan di sisi *server* dan menggunakannya untuk pemrosesan *business logic*			
74.	Memastikan keamanan pada waktu proses	▪ Melakukan pengembangan aplikasi dengan mempertimbangkan waktu pemrosesan. Jika penyerang dapat memperoleh keuntungan dengan mengetahui waktu pemrosesan dan hasil yang berbeda, tambahkan langkah atau pemrosesan tambahan sehingga apa pun hasilnya, hasilnya akan diberikan dalam jangka waktu yang sama ▪ Selain itu, aplikasi/sistem harus memiliki mekanisme yang tidak mengizinkan penyerang memperpanjang transaksi dalam jangka waktu yang dapat diterima. Hal ini dapat dilakukan dengan membatalkan atau mengatur ulang transaksi setelah jangka waktu tertentu berlalu			
75.	Memastikan terdapat batasan dari jumlah pemanggilan fungsi	Aplikasi harus menetapkan kontrol yang ketat untuk mencegah penyalahgunaan batas. Hal ini dapat dicapai dengan menetapkan kupon agar tidak lagi valid di tingkat *database*, untuk menetapkan batas penghitung per pengguna di tingkat *back-end* atau *database* karena semua pengguna harus diidentifikasi melalui *session*, mana saja yang lebih baik sesuai dengan kebutuhan bisnis			
76.	Memastikan tidak ada kerentanan pada alur kerja	▪ Aplikasi harus memiliki pemeriksaan untuk memastikan bahwa pengguna menyelesaikan setiap langkah dalam proses alur kerja dalam urutan yang benar dan mencegah penyerang menghindari/melewatkan/atau mengulangi langkah/proses apa pun dalam alur kerja ▪ Pengujian kerentanan alur kerja melibatkan pengembangan kasus penyalahgunaan/penyalahgunaan *business logic* dengan tujuan berhasil menyelesaikan proses bisnis namun tidak menyelesaikan langkah-langkah yang benar dalam urutan yang benar			
77.	Memastikan pertahanan yang baik terhadap penyalahgunaan aplikasi	▪ Aplikasi harus menerapkan pertahanan aktif untuk menangkis penyerang dan pelaku penyalahgunaan ▪ Pengujian harus dilakukan untuk menentukan apakah terdapat mekanisme pertahanan lapisan aplikasi untuk melindungi aplikasi			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
78.	Memastikan tidak bisa dilakukan *upload file* terbahaya	▪ Aplikasi harus dikembangkan dengan mekanisme untuk hanya menerima dan memanipulasi *file* yang dapat diterima yang siap ditangani dan diharapkan oleh fungsionalitas aplikasi lainnya ▪ Beberapa contoh spesifik meliputi: *blacklist* atau izinkan daftar ekstensi *file*, menggunakan *header* "Content-Type", atau menggunakan pengenal jenis *file*, semuanya hanya mengizinkan jenis *file* tertentu ke dalam sistem			
Keamanan pada Sisi Client (*Client Side*)					
79.	Memastikan tidak ada kerentanan DOM-*based Cross Site Scripting*	▪ Melakukan sanitasi pada semua data yang tidak tepercaya, meskipun hanya digunakan dalam *script* sisi *client*. Jika menggunakan *input* pengguna pada halaman *web*, selalu gunakan dalam konteks teks, jangan pernah sebagai tag HTML atau kode potensial lainnya. ▪ Gunakan hanya fungsi aman seperti document.innerText dan document.textContent.			
80.	Memastikan tidak ada kerentanan pada eksekusi JavaScript	▪ Menghindari penggunaan karakter dan fungsi yang tidak aman termasuk pernyataan eval(), setInterval(), setTimeout() dan konstruktor fungsi pada kolom input pengguna ▪ Menghindari eksekusi *code* dinamis bila diperlukan ▪ Membuat *whitelist* nilai *input* untuk dipilih pengguna ▪ Menggunakan *Firewall Aplikasi Web* (WAF) untuk mendeteksi string *code* dari pengguna jahat ▪ Mengunci *interpreter* aplikasi *browser* dan membatasi kapasitasnya hingga batas minimum yang diperlukan oleh konfigurasi server *web* ▪ Menggunakan *security linter* untuk menerapkan *secure code*			
81.	Memastikan tidak ada kerentanan HTML *injection*	▪ Memvalidasi *output* dan *input* karena serangan hanya menargetkan *input*/*output* yang tidak terverifikasi ▪ Memeriksa setiap *input* dan mencari tahu apakah ada *code* HTML atau *script* yang disebutkan dalam *input* ▪ Menggunakan *tools* pengujian otomatis sehingga tidak ada satu pun komponen situs *web* yang terlewat dalam pengujian			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		▪ Menggunakan *Web Application Firewall* (WAF) untuk menghentikan penyerang yang akan mengubah kode *input* Dengan cara ini, *code* HTML tidak dapat menjadi bagian dari *input* situs *web* ▪ Penggunaan *Content Security Policy* (CSP) juga berguna untuk memitigasi serangan jenis ini			
82.	Memastikan tidak ada kerentanan *client-side* URL *redirect*	▪ Hindari data yang dapat dikontrol pengguna dalam URL jika memungkinkan dan lakukan sanitasi secara hati-hati saat harus digunakan ▪ Masukkan semua lokasi target yang diizinkan ke *whitelist* (jika memungkinkan) dan alihkan semua nilai lainnya ke lokasi *default* ▪ Men-*generate* ID unik untuk setiap target *redirection*, sehingga menghilangkan nama yang dapat dikontrol pengguna dari URL ▪ Dengan mengatur *header* Referrer-Policy yang sesuai, dapat membatasi paparan URL *referrer* untuk lebih mengurangi risiko kebocoran token ▪ *Scanning* situs *web* dan aplikasi secara teratur untuk mengidentifikasi kerentanan			
83.	Memastikan tidak ada kerentanan CSS *injection*	▪ Selalu jaga agar CSS tetap terisolasi berdasarkan tingkat kontrol akses ▪ Menghapus informasi identitas apa pun pada *file* CSS ▪ Situs *web* memiliki *style* yang konsisten antar halaman, dan yang terbaik adalah menulis aturan CSS umum sedemikian rupa sehingga berlaku di beberapa halaman ▪ Jangan izinkan pengguna membuat konten melalui *input* HTML			
84.	Memastikan tidak ada kerentanan *client-side resource manipulation*	▪ Mengidentifikasi apakah aplikasi menggunakan *input* tanpa memvalidasinya dengan benar, sehingga *input* berada di bawah kendali pengguna dan dapat digunakan untuk menentukan *resource* eksternal ▪ *Client-side scripts* yang menangani URL terkait harus diselidiki untuk kemungkinan masalah karena terdapat banyak *resource* yang dapat disertakan dalam aplikasi (seperti gambar, video, Object, CSS, dan iFrame)			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
85.	Memastikan tidak ada kerentanan *cross origin resource sharing*	▪ Menguji langkah-langkah mitigasi CORS untuk mengidentifikasi dan memulihkan kerentanan keamanan apa pun. Cari beberapa *error* umum. Uji apakah *request cross-domain* diizinkan dari origin mana pun atau tidak, sehingga membuka peluang terhadap serangan penyelundupan konten ▪ Pastikan origin ditentukan dengan benar di Access-Control-Allow-Origin jika *resource web* berisi informasi sensitif ▪ Memastikan bahwa *input* origin yang sewenang-wenang tidak diperbolehkan, karena origin yang direfleksikan secara dinamis dapat dengan mudah dieksploitasi ▪ Hindari memasukkan *null* dan *wildcard* ke dalam *whitelist* di jaringan internal. *Request cross-domain* dan *request sandbox* dapat menentukan *null origin*			
86.	Memastikan tidak ada kerentanan *Cross Site Flashing* (XSF)	▪ Menerapkan validasi *input* dan pengkodean *output* yang tepat di aplikasi *web* untuk mencegah injeksi konten Flash berbahaya ▪ Gunakan *Content Security Policy* (CSP) untuk membatasi jenis konten yang dapat dimuat oleh halaman *web*, termasuk konten Flash ▪ Nonaktifkan *plugin* Flash di *browser* web kecuali benar-benar diperlukan untuk situs *web* atau aplikasi tertentu. ▪ Gunakan *browser* yang memiliki perlindungan bawaan terhadap serangan XSF, seperti Google Chrome atau Mozilla Firefox ▪ Instal ekstensi *browser* yang memblokir konten Flash, seperti Flashblock ▪ Selalu perbarui browser *web* dan Flash dengan *patch player* dengan keamanan terbaru ▪ *Scanning* aplikasi *web* secara teratur untuk mencari kerentanan ▪ Melatih personel untuk mengenali dan menghindari serangan *social engineering* yang mungkin digunakan untuk mengeksploitasi kerentanan XSF ▪ Gunakan *Web Application Firewall* (WAF) yang dapat mendeteksi dan memblokir serangan XSF			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		▪ Menerapkan *Secure Software Development Lifecycle* (SDLC) untuk meminimalkan risiko kerentanan XSF di aplikasi *web*			
87.	Memastikan tidak ada kerentanan *clickjacking*	▪ Mencegah *browser* memuat halaman dalam *frame* menggunakan *header* HTTP X-Frame-Options atau Content Security Policy (frame-ancestors) ▪ Mencegah *session cookies* disertakan saat halaman dimuat dalam *frame* menggunakan atribut *cookie* SameSite ▪ Menerapkan *code* JavaScript di halaman untuk mencegahnya dimuat dalam *frame* (dikenal sebagai "frame-buster")			
88.	Memastikan keamanan pada WebSockets	▪ WSS (WebSocket *over* SSL/TLS) aman, sehingga mencegah hal-hal seperti serangan *man-in-the-middle* ▪ Melakukan validasi *input* yang berasal *client* sebelum diproses ▪ Melakukan validasi data yang diterima dari *server* ▪ Melakukan autentikasi berbasis *ticket* ▪ Menghindari penggunaan *tunelling* ▪ Menerapkan *rate limiting* pada WebSocket ▪ Menggunakan *header* Origin dan mengkombinasikannya dengan *cookies* untuk keperluan autentikasi			
89.	Memastikan keamanan pada *web messaging*	▪ Saat mem-*posting* pesan, nyatakan secara eksplisit origin yang diharapkan sebagai argumen kedua pada postMessage daripada (*) untuk mencegah pengiriman pesan ke origin yang tidak diketahui setelah *redirection* atau cara lain untuk mengubah target pada window origin ▪ Halaman penerima harus selalu: o Periksa atribut asal *sender* untuk memverifikasi bahwa data berasal dari lokasi yang diharapkan o Melakukan validasi *input* pada atribut *data event* untuk memastikan formatnya sesuai yang diinginkan ▪ Jangan berasumsi memiliki kendali atas atribut data. Satu kelemahan *Cross Site Scripting* di halaman pengiriman memungkinkan penyerang mengirim pesan dalam format apa pun			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		▪ Kedua halaman seharusnya hanya menafsirkan pesan yang dipertukarkan sebagai data. Jangan pernah mengevaluasi pesan yang diteruskan sebagai *code* atau menyisipkannya ke halaman DOM, karena hal itu akan menciptakan kerentanan XSS berbasis DOM ▪ Untuk menetapkan nilai data ke suatu elemen, daripada menggunakan metode yang tidak aman ▪ Periksa origin dengan benar agar sesuai dengan FQDN yang diharapkan ▪ Jika perlu menyematkan konten eksternal yang tidak tepercaya dan mengizinkan *script* yang dikontrol pengguna, silakan periksa informasi pada *sandbox*			
90.	Memastikan keamanan pada *browser storage*	▪ Pola penyimpanan *origin-isolated* menawarkan cara untuk menjaga data dari jangkauan *code* berbahaya. *Attack surface* dikurangi dari akses *raw data* hingga penyalahgunaan API yang terekspos ▪ WebCrypto API memungkinkan klien untuk mengenkripsi dan mendekripsi data sebelum menyimpannya. Melakukan hal ini adalah satu-satunya cara untuk mencegah pencurian data melalui sistem *file* perangkat			
91.	Memastikan tidak ada kerentanan *Cross Site Script Inclusion* (XSSI)	Jangan interpolasi data sensitif dalam *file* JavaScript, sebaiknya menggunakan URL JSON sebagai gantinya, atau *encoding* data dalam HTML halaman itu sendiri. Tipe konten JSON dan HTML tunduk pada kebijakan *same-origin* pada browser, sehingga tidak dapat digunakan dalam serangan XSSI			
Keamanan pada API (*Application Programmable Interface*)					

92.	Memastikan keamanan pada GraphQL	▪ Menambahkan validasi *input* yang ketat dapat membantu mencegah *injection* dan DoS
		▪ Rekomendasi untuk membatasi potensi DoS:
		  ○ Menambahkan *depth limiting* pada *query* yang masuk
		  ○ Menambahkan *rate limiting* pada *query* yang masuk
		  ○ Menambahkan *pagination* untuk membatasi jumlah data yang dapat dikembalikan dalam satu *response*			

No.	Daftar Periksa	Kontrol Keamanan	Diterapkan? (Ya/Tidak)	Sesuai? (√)	Hasil Sebenarnya
		○ Menambahkan *timeout* yang wajar pada lapisan aplikasi, lapisan infrastruktur, atau keduanya ○ Mempertimbangkan untuk melakukan analisis biaya *query* dan menerapkan biaya maksimum yang diperbolehkan per *query* ○ Menerapkan *rate limiting* pada *request* masuk per IP atau pengguna (atau keduanya) untuk mencegah serangan DoS tingkat dasar ○ Menerapkan teknik *batching* dan *caching* di sisi *server*			