OWASP
Open Web Application
Security Project

FOR DEVELOPERS

2018 v 3.0

10 Critical Security Areas That Software Developers Must Be Aware Of

PROJECT LEADERS

KATY ANTON
JIM MANICO
JIM BIRD

حول OWASP

المشروع المفتوح لأمان تطبيق الويب (OWASP) هو مؤسسة خيرية تعليمية غير ربحية هدفها تمكين المؤسسات من تصميم برامج آمنة وتطويرها واكتسابها وتشغيلها وصيانتها. جميع أدوات ووثائق ومنتديات وفصول OWASP مجانية ومفتوحة لأي شخص مهتم بتحسين أمان التطبيق. يمكن العثور علينا من خلال زيارة الرابط www.owasp.org .

OWASP هي نوع جديد من المنظمات. إن تحررنا من الضغوط التجارية يسمح لنا بتقديم معلومات غير متحيزة وعملية وفعالة من حيث التكلفة حول أمان التطبيق.

إن OWASP ليست تابعة لأي شركة تكنولوجيا. على غرار العديد من مشاريع البرمجيات مفتوحة المصدر ، تنتج OWASP أنواعًا عديدة من المواد بطريقة تعاونية ومنفتحة. إن مؤسسة OWASP هي كيان غير هادف للربح ويضمن نجاح المشروع على المدى الطويل.

تمهيد

تعمل البرامج غير الآمنة على تقويض (تقييد undermining) مواردنا المالية ، والرعاية الصحية ، والدفاع ، والطاقة ، والبنى التحتية الحيوية الأخرى في جميع أنحاء العالم. نظرًا لأن بنيتنا التحتية الرقمية والعالمية تزداد تعقيدًا وترابطًا ، تزداد صعوبة تحقيق أمان التطبيقات بشكل كبير. لم نعد قادرين على التساهل مع المشاكل الأمنية البسيطة نسبيًا.

الأهداف

الهدف من مشروع OWASP لأفضل عشر ضوابط استباقية (OPC) OWASP Top 10 Proactive Controls هو زيادة الوعي بأمان التطبيقات من خلال وصف أهم مجالات الاهتمام التي يجب على مطوري البرمجيات إدراكها. نحن نشجعك على استخدام ضوابط OWASP الاستباقية لمساعدة المطورين لديك على البدء في تأمين التطبيقات. يمكن للمطورين التعلم من أخطاء المنظمات الأخرى. نأمل أن تكون ضوابط OWASP الاستباقية مفيدة لجهودك في بناء برمجيات آمنة.

دعوة للعمل واتخاذ إجراء

من فضلك لا تتردد في الاتصال بمشروع OWASP للضوابط الاستباقية ومشاركة أسئلتك وتعليقاتك وأفكارك ، إما علنًا في قائمة البريد الإلكتروني لدينا أو بشكل خاص إلى jim@owasp.org.

حقوق التأليف والنشر والترخيص

قادة المشروع

Katy Anton | Jim Bird | Jim Manico

المساهمون

Chris Romeo | Dan Anderson | David Cybuck

Dave Ferguson | Josh Grossman | Osama Elnaggar

Colin Watson | Rick Mitchell | والكثير من المساهمين

فريق الترجمة للغة العربية

Aref Shaheed

Mhd Ghassan Al-Habash

بنية المستند

تم صياغة هذا المستند كقائمة من ضوابط الأمان security controls. كل ضابط أمان يتم وصفه وفق ما يلي:

التوصيف

وصف تفصيلي لضابط الأمان بما في ذلك بعض أفضل الممارسات التي يجب مراعاتها.

التنفيذ

تنفيذ أفضل الممارسات والأمثلة لتوضيح كيفية تنفيذ كل ضابط أمان.

نقاط الضعف الي تم منعها

-
-

قائمة نقاط الضعف التي تم منعها أو المخاطر التي تمت معالجتها (OWASP TOP 10 Risk، CWE، إلخ.)

المراجع

-
-

قائمة المراجع لمزيد من الدراسة (ملخصات OWASP ، إرشادات تعزيز الأمان ، إلخ.)

الأدوات

-
-

مجموعة من الأدوات / المشاريع لتقديم / دمج ضوابط الأمان بسهولة في برنامجك.

مقدمة

إن أفضل عشر ضوابط OWASP الاستباقية 2018 هي قائمة بتقنيات الأمان التي يجب أن يتم أخذها بعين الاعتبار في كل مشروع تطوير برمجي. تم كتابة هذا المستند للمطورين الجدد وذلك لمساعدتهم في تأمين عملية تطوير البرمجيات.

أحد أهم الأهداف الأساسية لهذا المستند هو توفير إرشادات عملية ملموسة تساعد المطورين على بناء برمجيات آمنة. هذه التقنيات يجب أن يتم تطبيقها بشكل استباقي في مراحل مبكرة من عملية تطوير البرمجيات لضمان أعلى درجة من الفعالية.

أفضل عشر ضوابط استباقية

تم ترتيب هذه القائمة حسب الأهمية بقائمة مرقمة العناصر وتبدأ بالرقم 1 كالضابط الأكثر أهمية:

ض1: تحديد المتطلبات الأمنية

ض2: الاستفادة من أطر العمل Frameworks والمكتبات Libraries الأمنية

ض3: الوصول الآمن لقواعد البيانات

ض4: ترميز Encode واستخدام الهروب Escape في البيانات

ض5: التحقق من كافة المدخلات

ض6: تنفيذ الهوية الرقمية

ض7: فرض التحكم بالوصول

ض8: حماية البيانات في كل مكان

ض9: التسجيل logging والمراقبة الأمنية

ض10: معالجة handle كل الأخطاء والاستثناءات exceptions

كيف تم إنشاء هذه القائمة

تم إنشاء هذه القائمة في الأصل من قبل قادة المشروع الحاليين ومساهمات من عدة متطوعين. ثم تمت مشاركة الوثيقة على الصعيد العالمي للاستفادة من الاقتراحات مجهولة المصدر وأخذها بعين الاعتبار. تم قبول مئات التغييرات من عملية المجتمع المفتوح هذه.

الجمهور المستهدف

تم كتابة هذا المستند بشكل أساسي للمطورين. ومع ذلك ، يمكن أيضًا لمديري التطوير development managers ومالكي المنتجات product owners ومختصي ضمان الجودة Q/A professionals ومديري البرامج program managers وأي شخص مشارك في بناء البرامج الاستفادة من هذا المستند.

كيفية استخدام هذا المستند

يهدف هذا المستند إلى توفير الوعي الأولي حول بناء برمجيات آمنة. سيوفر هذا المستند أيضًا أساسًا جيدًا للموضوعات للمساعدة في توجيه التدريب التمهيدي لمطوري أمان البرمجيات software security developer. يجب استخدام هذه الضوابط بشكل متسق وشامل في جميع التطبيقات. ومع ذلك ، ينبغي النظر إلى هذه الوثيقة كنقطة انطلاق وليس كمجموعة شاملة من التقنيات والممارسات. يجب أن تتضمن عملية التطوير الآمن الكاملة متطلبات شاملة من معيار مثل OWASP ASVS بالإضافة إلى تضمين مجموعة من أنشطة تطوير البرامج الموضحة في نماذج النضج maturity models مثل OWASP SAMM و BSIMM .

الربط مع مشروع OWASP للعشر الأوائل OWASP Top 10 Project

إن مشروع OWASP لأفضل عشر ضوابط استباقية يشبه مشروع OWASP للعشر الأوائل ولكنه يركز على تقنيات وضوابط دفاعية بدلاً من المخاطر. سيتم ربط كل تقنية أو ضابط في هذا المستند إلى عنصر أو أكثر لمشروع OWASP للعشر الأوائل القائم على المخاطر. تم تضمين معلومات الربط هذه في نهاية الوصف لكل ضابط.

ضوابط **OWASP** الاستباقية النسخة **3.0**

ض1: تحديد المتطلبات الأمنية

التوصيف

متطلب الأمان عبارة عن بيان أو إيضاح بوظيفة الأمان المطلوبة التي تضمن تلبية واحدة من العديد من خصائص الأمان المختلفة للبرنامج. يتم اشتقاق متطلبات الأمان من معايير الصناعة والقوانين المعمول بها وتاريخ الثغرات الأمنية السابقة. تحدد متطلبات الأمان ميزات جديدة أو إضافات للميزات الحالية لحل مشكلة أمنية معينة أو التخلص من ثغرة أمنية محتملة.

توفر متطلبات الأمان أساسًا لوظيفة الأمان التي تم فحصها لتطبيق ما. بدلاً من إنشاء نهج مخصص للأمان لكل تطبيق ، تسمح متطلبات الأمان القياسية للمطورين بإعادة استخدام تعريف ضوابط الأمان وأفضل الممارسات. توفر نفس متطلبات الأمان التي تم فحصها حلولًا لمشكلات الأمان التي حدثت في الماضي. توجد متطلبات لمنع تكرار حالات فشل الأمان السابقة.

مشروع *OWASP ASVS*

إن معيار OWASP للتحقق من أمان التطبيقات (ASVS) OWASP Application Security Verification Standard (ASVS) هو قائمة بمتطلبات الأمان ومعايير التحقق المتاحة. يمكن أن يكون OWASP ASVS مصدرًا لمتطلبات الأمان التفصيلية لفرق التطوير.

يتم تصنيف متطلبات الأمان ضمن فئات (buckets) مختلفة بناءً على وظيفة أمان مشتركة ذات ترتيب أعلى. على سبيل المثال ، يحتوي ASVS على فئات مثل المصادقة والتحكم في الوصول ومعالجة الأخطاء / التسجيل وخدمات الويب. تحتوي كل فئة على مجموعة من المتطلبات التي تمثل أفضل الممارسات لتلك الفئة والتي تمت صياغتها كبيانات أو إيضاحات (جمع بيان statements وسيتم استخدام كلمة إيضاحات بدلا من بيانات لمنع الالتباس) يمكن التحقق منها.

توسيع المتطلبات بقصص المستخدم *User Stories* وحالات إساءة الاستخدام *Misuse Cases*

متطلبات ASVS هي إيضاحات أساسية يمكن التحقق منها والتي يمكن توسيعها بقصص المستخدم وحالات إساءة الاستخدام. تمثل ميزة قصة المستخدم أو حالة إساءة الاستخدام في أنها تربط التطبيق بما يفعله المستخدم أو المهاجم بالضبط في النظام ، مقابل وصف ما يقدمه النظام للمستخدم.

فيما يلي مثال على التوسع في متطلبات ASVS 3.0.1 من قسم "متطلبات التحقق من المصادقة" . في ASVS 3.0.1 ، يركز المتطلب 2.19 على كلمات المرور الافتراضية.

المتطلب 2.19 يتحقق من عدم وجود كلمات مرور افتراضية قيد الاستخدام لإطار عمل التطبيق أو أي مكونات يستخدمها التطبيق (مثل "admin /password").

يحتوي هذا المتطلب على إجراء للتحقق من عدم وجود كلمات مرور افتراضية ، ويتضمن أيضًا إرشادات تفيد بعدم استخدام كلمات مرور افتراضية داخل التطبيق.

تركز قصة المستخدم user story على منظور المستخدم أو المسؤول أو المهاجم للنظام ،وتصف الوظائف بناءً على ما يريد المستخدم أن يفعله النظام له. تأخذ قصة المستخدم شكل " As a user, I can do x, y, and z". بصفتي مستخدمًا، يمكنني تنفيذ x و y و z".

As a user, I can enter my username and password to gain access to the application.

بصفتي مستخدماً، أستطيع إدخال اسم المستخدم وكلمة المرور للوصول إلى التطبيق

As a user, I can enter a long password that has a maximum of 1023 characters.

بصفتي مستخدماً، أستطيع إدخال كلمة مرور طويلة على ألا تتجاوز 1023 محرف.

عندما تركز القصة على المهاجم وأفعاله ، يشار إليها على أنها حالة إساءة استخدام Misuse case.

As an attacker, I can enter in a default username and password to gain access.
بصفتي مهاجماً، أستطيع إدخال اسم المستخدم الافتراضي وكلمة المرور الافتراضية للوصول للتطبيق.

تحتوي هذه القصة على نفس الرسالة مثل المتطلبات التقليدية من ASVS ، مع تفاصيل إضافية للمستخدم أو المهاجم للمساعدة في جعل المتطلبات أكثر قابلية للاختبار.

التنفيذ

يتضمن الاستخدام الناجح لمتطلبات الأمان أربع خطوات. تتضمن العملية اكتشاف / اختيار ، وتوثيق ، وتنفيذ ، ثم تأكيد التنفيذ الصحيح لميزات ووظائف الأمان الجديدة داخل التطبيق.

الاكتشاف والاختيار

تبدأ العملية باكتشاف واختيار متطلبات الأمان. في هذه المرحلة ، يفهم المطور متطلبات الأمان من مصدر معياري مثل ASVS ويختار المتطلبات التي يجب تضمينها لإصدار معين من التطبيق. تتمثل نقطة الاكتشاف والانتقاء في اختيار عدد يمكن التحكم فيه من متطلبات الأمان لهذا الإصدار release أو السباق sprint، ثم الاستمرار في التكرار لكل سباق sprint، لإضافة المزيد من وظائف الأمان بمرور الوقت.

التحقيق والتوثيق

أثناء التحقيق والتوثيق ، يراجع المطور التطبيق الحالي من أجل المجموعة الجديدة من متطلبات الأمان لتحديد ما إذا كان التطبيق يفي حاليًا بالمتطلبات أو إذا كان يتطلب بعض التطوير. يُتوج هذا التحقيق بتوثيق نتائج المراجعة.

التنفيذ والاختبار

بعد تحديد الحاجة إلى التطوير ، يجب على المطور الآن تعديل التطبيق بطريقة ما لإضافة وظيفة جديدة أو إزالة خيار غير آمن. في هذه المرحلة ، يحدد المطور أولاً التصميم المطلوب لتلبية المتطلبات، ثم يكمل تغييرات الشيفرة المصدرية لتلبية المتطلبات. يجب بناء حالات الاختبار لتأكيد وجود الوظيفة أو الميزة (functionality) الجديدة أو دحض وجود خيار غير آمن سابقًا.

نقاط الضعف التي تم منعها

تقوم متطلبات الأمان بتحديد وظائف الأمان للتطبيق. يؤدي تحسين الأمان المدمج built-in منذ بداية دورة حياة التطبيقات إلى منع العديد من أنواع الثغرات الأمنية.

المراجع

- OWASP Application Security Verification Standard (ASVS)
- OWASP Mobile Application Security Verification Standard (MASVS)
- OWASP Top Ten

ضوابط **OWASP** الاستباقية النسخة **3.0**

ض2: الاستفادة من أطر العمل **Frameworks** والمكتبات **Libraries** الأمنية

التوصيف

تساعد مكتبات كتابة الشيفرة المصدرية الآمنة Secure coding libraries وأطر عمل البرامج المزودة بأمان مضمّن software frameworks with embedded security مطوري البرامج على الحماية من عيوب التصميم والتنفيذ المتعلقة بالأمان. قد لا يمتلك المطور الذي يكتب تطبيقًا من البداية المعرفة الكافية أو الوقت أو الميزانية الكافية لتنفيذ ميزات الأمان أو الحفاظ عليها بشكل صحيح. تساعد الاستفادة من أطر العمل الأمنية في تحقيق أهداف الأمان بشكل أكثر كفاءة ودقة.

أفضل ممارسات التنفيذ

عند دمج مكتبات أو أطر عمل تابعة لأطراف ثالثة third party في برنامجك ، من المهم مراعاة أفضل الممارسات التالية:

1. استخدم مكتبات وأطر عمل من مصادر موثوقة يتم صيانتها واستخدامها على نطاق واسع بواسطة العديد من التطبيقات.
2. إنشاء وصيانة قائمة جرد inventory catalog لجميع مكتبات الطرف الثالث.
3. تحديث جميع المكتبات وأطر العمل بشكل استباقي، استخدم أداة كـ OWASP Dependency Check و Retire.JS لتحديد تبعيات المشروع project dependencies والتحقق من وجود نقاط ضعف معروفة ومعلن عنها publicly disclosed في الشيفرة المصدرية الخاصة بجميع مكونات الأطراف الثالثة.
4. تقليص سطح الهجوم attack surface (المقصود بهذا المصطلح هو عدد النقاط التي يمكن من خلالها الهجوم على التطبيق) وذلك بتغليف encapsulation المكتبات وكشف السلوك المطلوب فقط في البرنامج.

نقاط الضعف التي تم منعها

يمكن أن تساعد المكتبات والأطر الآمنة في منع مجموعة كبيرة من الثغرات الأمنية في تطبيقات الويب. من المهم جدا الحفاظ على هذه الأطر والمكتبات محدثة كما هو موضح في استخدام المكونات ذات الثغرات الأمنية المعروفة في مشروع العشر الأوائل 2017 using components with known vulnerabilities Top Ten 2017 risk .

الأدوات

- OWASP Dependency Check – تقوم بتحديد تبعيات المشروع والتحقق من نقاط الضعف المعلن عنها في هذه التبعيات.
- Retire.JS – أداة مسح لمكتبات JavaScript.

ضوابط **OWASP** الاستباقية النسخة **3.0**

ض3: الوصول الآمن لقواعد البيانات

التوصيف

يصف هذا القسم الوصول الآمن إلى جميع مخازن البيانات data stores، بما في ذلك قواعد البيانات العلائقية relational databases وقواعد بيانات NoSQL. بعض المجالات التي يجب مراعاتها:

1. الاستعلامات آمنة Secure queries
2. التكوين آمن Secure configuration
3. المصادقة الآمنة Secure authentication
4. الاتصال الآمن Secure communication

الاستعلامات الآمنة

يحدث حقن SQL عندما تتم إضافة مدخلات مستخدم غير موثوق بها ديناميكيًا إلى استعلام SQL بطريقة غير آمنة ، غالبًا ما يحدث ذلك من خلال ربط السلاسل concatenation. يعد حقن SQL واحدة من أكبر مخاطر أمان التطبيقات. من السهل استغلال حقن SQL ويمكن أن يؤدي إلى سرقة قاعدة البيانات بأكملها أو مسحها أو تعديلها. يمكن أيضًا استخدام التطبيق لتشغيل أوامر خطيرة ضد نظام التشغيل الذي يستضيف قاعدة البيانات الخاصة بك ، مما يمنح المهاجم موطئ قدم على شبكتك.

من أجل التخفيف من حقن SQL ، يجب منع تفسير interpret مع المدخلات غير الموثوق بها كجزء من استعلام SQL.أفضل طريقة للقيام بذلك هي باستخدام تقنية البرمجة المعروفة باسم "Query Parameterization". يجب تطبيق هذا الإجراء الدفاعي على SQL، OQL، بالإضافة إلى استخدامه عند بناء الإجراءات المخزنة stored procedure construction.

يمكن العثور على قائمة جيدة تتضمن أمثلة تقنية query parameterization في ASP و ColdFusion و C # و Delphi و .NET و Go و Java و Perl و PHP و PL / SQL و PostgreSQL و Python و R و Ruby و Scheme في الروابط التالية: http://bobby-tables.com و OWASP Cheat Sheet on Query Parameterization.

تنبيه بخصوص تقنية *Query Parameterization*

بعض المواقع locations في استعلام قاعدة البيانات غير قابلة للتحويل إلى بارامترات parameterizable . تختلف هذه المواقع حسب نوع قاعدة بيانات. تأكد من إجراء تحقق دقيق للغاية من صحة المطابقة التامة أو الهروب اليدوي manual escaping عند وجود بارامترات لا يمكن ربطها باستعلام ذو بارامترات parameterized query. بالإضافة إلى أن استخدام الاستعلامات ذات البارامترات له تأثير إيجابي إلى حد كبير على الأداء ، فإن بعض الاستعلامات ذات البارامترات في تطبيقات قاعدة بيانات معينة ستؤثر سلبًا على الأداء. تأكد من اختبار استعلامات الأداء وخاصة الاستعلامات المعقدة مع إمكانات بحث شاملة مثل الجمل أو النص.

التكوين الآمن

لسوء الحظ ، لا يتم تشغيل أنظمة إدارة قواعد البيانات دائمًا في تكوين "آمن افتراضيًا" عند بداية وضع الخدمة. يجب توخي الحذر لضمان تمكين الضوابط الأمنية المتاحة من نظام إدارة قاعدة البيانات (Database Management System DBMS) والنظام الأساسي للاستضافة وتكوينها بشكل صحيح. هناك معايير وأدلة ومعايير متاحة لمعظم نظم إدارة قواعد البيانات الشائعة.

المصادقة الآمنة

يجب مصادقة كل عمليات الوصول إلى قاعدة البيانات بشكل صحيح. يجب أن تتم المصادقة على نظام إدارة قواعد البيانات بطريقة آمنة. كما يجب أن تتم المصادقة عبر قناة آمنة فقط ويجب إتاحة بيانات المصادقة Credentials وتأمينها بشكل صحيح.

الاتصال الآمن

تدعم معظم نظم إدارة قواعد البيانات (DBMS) مجموعة متنوعة من طرق الاتصال (الخدمات services، واجهات برمجة التطبيقات APIs، إلخ) - بطرق آمنة (باستخدام المصادقة والتشفير) وغير آمنة (بدون استخدام مصادقة أو التشفير). من الممارسات الجيدة استخدام خيارات الاتصالات الآمنة فقط وفقًا لضابط الأمان "قم بحماية البيانات في أي مكانProtect Data Everywhere".

نقاط الضعف التي تم منعها

- OWASP Top 10 2017- A1: Injection
- OWASP Mobile Top 10 2014-M1 Weak Server Side Controls

المراجع

- OWASP Cheat Sheet: Query Parameterization
- Bobby Tables: A guide to preventing SQL injection
- CIS Database Hardening Standards

ضوابط **OWASP** الاستباقية النسخة **3.0**

ض4: ترميز **Encode** واستخدام الهروب **Escape** في البيانات

التوصيف

الترميز Encoding والهروب Escaping هي تقنيات دفاعية تهدف إلى منع هجمات الحقن. **الترميز** (يسمى عادةً "ترميز المخرجات Output Encoding") يتضمن تفسير المحارف الخاصة إلى شكل مختلف ولكنه مكافئ آمن لمحتواها ولا يشكل خطر في المفسر الهدف ، على سبيل المثال تفسير المحرف ">" إلى السلسلة **<** عند الكتابة في صفحة HTML. يتضمن **الهروب** إضافة محرف خاص قبل المحرف أو السلسلة لتجنب تفسيره بشكل خاطئ، على سبيل المثال ، إضافة محرف " \ " قبل محرف " **"** " (اقتباس مزدوج) بحيث يتم تفسيره كنص وليس كمحرف إغلاق سلسلة.

من الأفضل تطبيق ترميز المخرجات **مباشرة قبل** أن يتم تمرير المحتوى إلى المفسر الهدف. إذا تم تنفيذ هذا الأسلوب الدفاعي مبكرًا جدًا في معالجة الطلب ، فقد يتداخل الترميز أو الهروب مع استخدام المحتوى في أجزاء أخرى من البرنامج. على سبيل المثال ، إذا قمت باستخدام هروب HTML في المحتوى (HTML escape) قبل تخزين تلك البيانات في قاعدة البيانات وقامت واجهة المستخدم UI باستخدام الهروب أيضاً وبشكل تلقائي في هذه البيانات مرة ثانية ، فلن يتم عرض المحتوى بشكل صحيح بسبب استخدام الهروب المزدوج.

ترميز المخرجات السياقي Contextual Output Encoding

يعد ترميز المخرجات السياقي تقنية مهمة في برمجة الأمان اللازمة لإيقافXSS. يتم تنفيذ هذا الدفاع عند المخرجات، عند إنشاء واجهة مستخدم ، في اللحظة الأخيرة قبل إضافة البيانات غير الموثوق بها ديناميكيًا إلى HTML. يعتمد نوع الترميز على الموقع (أو السياق) في المستند حيث يتم عرض البيانات أو تخزينها. تشمل الأنواع المختلفة من الترميز التي يمكن استخدامها لبناء واجهات مستخدم آمنة ترميز كيان HTML (HTML Entity Encoding)، وترميز سمات HTML (HTML Attribute Encoding)، وترميز JavaScript (JavaScript Encoding)، وترميز الرابط (URL Encoding).

أمثلة الترميز في *Java*

على سبيل المثال، OWASP Java Encoder يدعم ترميز المخرجات السياقي، قم بزيارة OWASP Java Encoder Project Examples :

أمثلة الترميز في .NET

بدءًا من .NET 4.5 ، تعد مكتبة Anti-Cross Site Scripting جزءًا من إطار العمل ، ولكن لا يتم تفعيلها افتراضيًا. يمكنك تحديد استخدام AntiXssEncoder من هذه المكتبة باعتباره المرمز الافتراضي للتطبيق بأكمله باستخدام إعدادات web.conf عند تطبيقه ، من المهم ترميز مخرجاتك حسب السياق - وهذا يعني استخدام الوظيفة الصحيحة من مكتبة AntiXSSEncoder للموقع المناسب للبيانات في المستند.

أمثلة الترميز في PHP

Zend Framework 2

يمكن استخدام Zend\Escaper في Zend Framework 2 (ZF2) لترميز المخرجات. للاطلاع على أمثلة لترميز المخرجات السياقي قم بزيارة Context-specific escaping with zend-escaper.

أنواع أخرى من الترميز والحماية من الحقن

يمكن استخدام الترميز / الهروب لتحييد أو إبطال مفعول neutralize أشكال الحقن المتواجدة في المحتوى. على سبيل المثال ، من الممكن تحييد بعض المحارف الوصفية meta-characters الخاصة عند إضافة مدخلات إلى أمر نظام التشغيل. وهذا ما يسمى " OS command escaping" أو "shell escaping". يمكن استخدام هذا الأسلوب الدفاعي لإيقاف ثغرات "حقن الأوامر Command Injection ".

هناك أشكال أخرى من الهروب يمكن استخدامها لإيقاف الحقن ، مثل الهروب في سمة XML (XML attribute escaping) الذي يستخدم لإيقاف الأشكال المختلفة لحقن XML ومسار XML (XML and XML path injection)، بالإضافة إلى الهروب في الاسم المميز لـ LDAP (LDAP distinguished name escaping) الذي يمكن استخدامه لإيقاف الأشكال المختلفة من حقن LDAP (LDAP Injection).

ترميز المخرجات والتحويل القياسي Canonicalization

ترميز Unicode هو طريقة لتخزين محارف متعددة البايت multiple bytes. يمكن إدخال البيانات باستخدام Unicode لإخفاء التعليمات البرمجية الضارة والسماح بمجموعة متنوعة من الهجمات. يشير RFC 2279 إلى العديد من الطرق التي يمكن بها ترميز النص.

التحويل القياسي Canonicalization هو طريقة تقوم فيها الأنظمة بتحويل البيانات إلى نموذج بسيط أو قياسي. تستخدم تطبيقات الويب بشكل شائع التحويل القياسي للمحارف لضمان أن يكون كل المحتوى من نفس نوع المحرف عند تخزينه أو عرضه.

لكي تكون آمنًا ضد الهجمات المتعلقة بالتحويل القياسي، يجب أن يكون التطبيق آمنًا عند إدخال Unicode المشوه (malformed Unicode) وتمثيلات المحارف المشوهة (malformed character representations) الأخرى.

نقاط الضعف التي تم منعها

- OWASP Top 10 2017 - A1: Injection
- OWASP Top 10 2017 - A7: Cross Site Scripting (XSS)
- OWASP Mobile_Top_10_2014-M7 Client Side Injection

المراجع

- XSS – معلومات عامة
- OWASP Cheat Sheet: XSS Prevention – إيقاف XSS في تطبيق الويب الخاص بك
- OWASP Cheat Sheet: DOM based XSS Prevention
- OWASP Cheat Sheet: Injection Prevention

الأدوات

- OWASP Java Encoder Project
- AntiXSSEncoder
- Zend\Escaper – أمثلة على الترميز السياقي

ضوابط **OWASP** الاستباقية النسخة **3.0**

ض5: التحقق من كافة المدخلات

التوصيف

التحقق من صحة المدخلات هو أسلوب برمجي يضمن فقط إدخال البيانات المنسقة بشكل صحيح إلى مكون نظام برمجي software system component.

صحة بناء الجملة (Syntax) والدلالات (Semantics)

يجب أن يتحقق التطبيق من أن البيانات صحيحة نحويًا syntactically ودلالياً semantically (بهذا الترتيب) قبل استخدامها بأي طريقة (بما في ذلك عرضها مرة أخرى للمستخدم).

صحة بناء الجملة Syntax validity تعني أن البيانات بالشكل المتوقع. على سبيل المثال، قد يسمح التطبيق للمستخدم بتحديد " account ID" المكون من أربعة أرقام لإجراء نوع من العمليات. يجب أن يفترض التطبيق أن المستخدم يدخل حمولة حقن SQL، ويجب أن يتحقق من أن البيانات التي أدخلها المستخدم هي بالضبط أربعة أرقام (الطول)، وتتكون فقط من أرقام (بالإضافة إلى استخدام بارامترات الاستعلام المناسبة utilizing proper query parameterization).

صحة الدلالية Semantic validity تتضمن قبول المدخلات التي تقع ضمن مجال مقبول لوظيفة وسياق التطبيق المحدد. على سبيل المثال، يجب أن يكون تاريخ البدء قبل تاريخ الانتهاء عند اختيار مجالات التاريخ.

وضع القائمة البيضاء Whitelisting ووضع القائمة السوداء Blacklisting

هناك طريقتان عامتان لإجراء التحقق من صحة بناء المدخلات، تعرف هذه الطريقتان وضع القائمة السوداء blacklisting ووضع القائمة البيضاء whitelisting:

- التحقق في وضع القائمة السوداء (أو القائمة السوداء blacklist) وفيها يتم التحقق من أن البيانات لا تتضمن محتوى ضار معروف "known bad". على سبيل المثال: قد يقوم تطبيق الويب بمنع المدخلات التي تتضمن بالضبط المحتوى "<SCRIPT>" لمنع هجومات XSS. ومع ذلك يمكن تجاوز هذا الإجراء الدفاعي بالتلاعب بالمحتوى الضار كاستخدام أحرف صغيرة أو خليط من الأحرف الصغيرة والكبيرة فيه.
- التحقق في وضع القائمة البيضاء (أو القائمة البيضاء whitelist) وفيها يتم التحقق من أن البيانات تطابق مجموعة من النماذج والقواعد المعروفة أنها جيدة وآمنة "known good". على سبيل المثال، قاعدة التحقق لإدخال اسم ولاية أمريكية US state في القائمة البيضاء يجب أن تتكون من حرفين وهي فقط النموذج المسموح فيه لإدخال هذا النوع من البيانات.

عند بناء برنامج آمن، فإن وضع القائمة البيضاء هو الأسلوب الدفاعي الموصى به كحد أدنى من طرق الحماية. إن وضع القائمة السوداء عرضة للخطأ ويمكن تجاوزها بأساليب تجاوز bypassed مختلفة ويمكن أن تكون خطيرة عندما تعتمد على نفسها. على الرغم من إمكانية التجاوز في وضع القائمة السوداء في كثير من الأحيان، إلا أنه غالبًا ما يكون مفيدًا للمساعدة في اكتشاف الهجمات الواضحة. لذلك، بينما يساعد وضع القائمة البيضاء في تقليص سطح الهجوم من خلال التأكد من أن البيانات ذات صحة نحوية ودلالية، فإن وضع القائمة السوداء يساعد في اكتشاف الهجمات الواضحة وإيقافها.

التحقق من جهة العميل client-side ومن جهة المخدم server-side

لتحقيق أمان أعلى، يجب أن يتم التحقق من صحة الإدخال دائمًا من جهة المخدم. في حين أن التحقق من جهة العميل يمكن أن يكون مفيدًا لكل من الأغراض الوظيفية وبعض الأغراض الأمنية، فإنه غالبًا ما يمكن تجاوزه بسهولة. هذا يجعل التحقق من جهة المخدم أكثر أهمية للأمان. على سبيل المثال، قد ينبه التحقق من صحة JavaScript المستخدم إلى أن حقلًا معينًا يجب أن يتكون من أرقام ولكن يجب أن يتحقق التطبيق من جهة المخدم من أن البيانات المقدمة تتكون فقط من أرقام في المجال العددي المناسب لتلك الميزة.

التعابير النمطية Regular Expressions

توفر التعبيرات النمطية طريقة للتحقق مما إذا كانت البيانات تتطابق مع نمط معين. لنبدأ بمثال أساسي.

يستخدم التعبير النمطي التالي لتعريف قائمة بيضاء للتحقق من أسماء المستخدمين usernames.

```
^[a-z0-9_]{3,16}$
```

يسمح هذا التعبير النمطي فقط بالأحرف الصغيرة والأرقام وعلامة الشرطة السفلية. اسم المستخدم مقيد أيضًا بطول بين 3 و 16 حرفًا.

تحذير: احتمالية تعطيل الخدمة *DoS*

يجب توخي الحذر عند بناء التعبيرات النمطية. قد تؤدي التعبيرات المصممة بشكل سيء إلى تعطيل محتمل للخدمة denial of service (المعروف أيضًا باسم ReDoS). يمكن اختبار الأدوات المختلفة للتحقق من أن التعبيرات النمطية ليست عرضة لـ ReDoS .

تحذير: التعقيد *Complexity*

التعبيرات النمطية هي طريقة واحدة فقط لإنجاز التحقق من الصحة. قد يكون من الصعب الحفاظ على التعبيرات النمطية أو فهمها لبعض المطورين. تتضمن بدائل التحقق الأخرى كتابة طرق التحقق برمجيًا والتي قد يكون من الأسهل صيانتها لبعض المطورين.

قيود التحقق من المدخلات

لا يؤدي التحقق من صحة المدخلات دائمًا إلى جعل البيانات "آمنة" نظرًا لأن بعض أشكال المدخلات المعقدة قد تكون "صالحة" ولكنها لا تزال خطرة. على سبيل المثال ، قد يحتوي عنوان البريد الإلكتروني الصالح على هجوم حقن SQL أو قد يحتوي عنوان URL الصالح على هجوم البرمجة النصية عبر الموقع XSS. يجب دائمًا تطبيق الدفاعات الإضافية إلى جانب التحقق من صحة المدخلات على البيانات مثل بارامترات الاستعلام أو الهروب.

تحديات التحقق من صحة البيانات المتسلسلة serialized data

بعض أشكال المدخلات معقدة للغاية لدرجة أن التحقق من الصحة لا يمكن إلا أن يحمي التطبيق بالحد الأدنى. على سبيل المثال، من الخطر فك تسلسل deserialize البيانات أو البيانات غير الموثوق بها التي يمكن للمهاجم التلاعب بها. النمط المعماري الوحيد الآمن هو عدم قبول كائنات متسلسلة serialized objects من مصادر غير موثوق بها أو فك التسلسل بسعة محدودة فقط لأنواع البيانات البسيطة فقط. يجب تجنب معالجة تنسيقات البيانات المتسلسلة واستخدام تنسيقات أسهل للدفاع مثل JSON عندما يكون ذلك ممكنًا.

إذا لم يكن ذلك ممكنًا، ففكر في سلسلة من دفاعات التحقق عند معالجة البيانات المتسلسلة.

تنفيذ عمليات التحقق من سلامة العناصر المتسلسلة أو تشفيرها لمنع إنشاء كائن معاد أو التلاعب بالبيانات.

- تنفيذ عمليات التحقق من سلامة العناصر المتسلسلة أو تشفيرها لمنع إنشاء كائن معادي hostile object أو التلاعب بالبيانات data tampering .
- فرض قيود صارمة على النوع أثناء فك التسلسل قبل إنشاء الكائن؛ عادةً ما يتوقع الكود مجموعة محددة من الفئات. تم إثبات تجاوز هذه التقنية.
- قم بعزل التعليمات البرمجية التي تقوم بفك التسلسل، بحيث يتم تشغيلها في بيئات ذات امتيازات منخفضة للغاية، مثل الحاويات المؤقتة temporary containers.
- تسجيل استثناءات exceptions وحالات فشل failure أمان فك التسلسل، مثال: عندما لا يكون النوع الوارد هو النوع المتوقع، أو عند حدوث استثناءات أثناء فك التسلسل.
- تقييد أو مراقبة اتصالات الشبكة الواردة والصادرة من الحاويات أو المخدمات التي تقوم بفك التسلسل.
- مراقبة فك التسلسل، والتنبيه في حال قام المستخدم بفك التسلسل بشكل مستمر.

مدخلات غير متوقعة من المستخدم (إسناد ضخم Mass Assignment)

تدعم بعض أطر العمل الربط التلقائي automatic binding لبارامترات طلبات HTTP إلى كائنات من جهة المخدم التي يستخدمها التطبيق. يمكن أن تسمح ميزة الربط التلقائي هذه للمهاجم بتحديث كائنات من جهة المخدم لم يكن من المفترض تعديلها. يمكن للمهاجم تعديل مستوى التحكم في الوصول أو التحايل على منطق العمل المقصود للتطبيق باستخدام هذه الميزة.

لهذا الهجوم عدد من الأسماء كـ : الإسناد الضخم mass assignment والربط التلقائي autobinding وحقن الكائنات object injection.

كمثال بسيط ، إذا كان كائن المستخدم user object لديه حقل privileges يحدد مستوى امتياز المستخدم في التطبيق ، يمكن للمستخدم الضار البحث عن الصفحات التي يتم فيها تعديل بيانات المستخدم ووضع الامتياز كمسؤول وذلك بإضافة privilege=admin إلى بارامترات HTTP المرسلة. إذا تم تمكين الربط التلقائي بطريقة غير آمنة ، فسيتم تعديل كائن من جهة المخدم الذي يمثل المستخدم وفقًا لذلك.

هناك طريقتين لمعالجة هذه المشكلة:

- تجنب ربط المدخلات مباشرة واستخدام Data Transfer Objects (DTOs).
- تفعيل خاصية الربط التلقائي auto-binding مع ضبط قواعد للقائمة البيضاء لكل صفحة أو ميزة لتحديد أي حقل مسموح له ليتم ربطه تلقائياً.

يمكنك الاطلاع على المزيد من الأمثلة هنا OWASP Mass Assignment Cheat Sheet

التحقق من صحة وتعقيم HTML (Sanitizing)

لنفترض أنه لدينا تطبيقًا يحتاج إلى قبول HTML من المستخدمين (عبر محرر WYSIWYG الذي يمثل المحتوى بتنسيق HTML أو الميزات التي تقبل HTML كمدخلات مباشرةً) . في هذه الحالة ، لن يساعد التحقق من الصحة أو الهروب.

- التعبيرات النمطية ليست مفيدة بما فيه الكفاية لفهم تعقيد HTML5.
- لن يساعد الترميز أو الهروب في HTML لأنه سيؤدي إلى عرض HTML (Rendering) بطريقة غير مناسبة.

لذلك فأنت بحاجة لمكتبة يمكنها تحليل parse وتنظيف النص بتنسيق HTML. يرجى الاطلاع على الأوراق الخاصة بمنع XSS حول تعقيم HTML XSS Prevention Cheat Sheet on HTML Sanitization للحصول على مزيد من المعلومات حول تعقيم HTML

وظيفة التحقق من الصحة في المكتبات وأطر العمل

توفر جميع اللغات ومعظم أطر العمل مكتبات أو وظائف للتحقق من صحة البيانات. تغطي مكتبات التحقق من الصحة عادةً أنواع البيانات الشائعة ومتطلبات الطول ونطاقات الأعداد الصحيحة وفحوصات "is null" والمزيد. تسمح لك العديد من مكتبات وأطر عمل التحقق من الصحة بتعريف التعبير النمطي أو المنطق الخاص بك للتحقق المخصص بطريقة تسمح للمبرمج بالاستفادة من هذه الوظيفة في جميع مكونات التطبيق. تتضمن أمثلة وظائف التحقق من الصحة وظائف مرشح (filter functions) PHP أو Hibernate Validator for Java. تتضمن أمثلة معقمات HTML (HTML Sanitizers) طريقة تعقيم Ruby on Rails sanitize method أو OWASP Java HTML Sanitizer أو DOMPurify.

نقاط الضعف التي تم منعها

- يخفض التحقق من صحة المدخلات من سطح الهجوم على التطبيقات ويمكن في بعض الأحيان أن يجعل الهجمات ضد أحد التطبيقات أكثر صعوبة.
- التحقق من صحة المدخلات هو تقنية توفر الأمان لأنواع معينة من البيانات ، خاصة بهجمات معينة ولا يمكن تطبيقها بشكل موثوق كقاعدة أمان عامة.
- لا ينبغي استخدام التحقق من صحة المدخلات كطريقة أساسية لمنع XSS و SQL Injection والهجمات الأخرى.

المراجع

- OWASP Cheat Sheet: Input Validation
- OWASP Cheat Sheet: iOS - Security Decisions via Untrusted Inputs
- OWASP Testing Guide: Testing for Input Validation

الأدوات

- OWASP Java HTML Sanitizer Project
- Java JSR-303/JSR-349 Bean Validation
- Java Hibernate Validator
- JEP-290 Filter Incoming Serialization Data
- Apache Commons Validator
- PHP's filter functions

ضوابط **OWASP** الاستباقية النسخة **3.0**

ض6: تنفيذ الهوية الرقمية

التوصيف

الهوية الرقمية هي التمثيل الفريد للمستخدم (أو أي موضوع آخر) أثناء تفاعله مع معاملة transaction عبر الإنترنت. المصادقة Authentication هي عملية التحقق من أن الفرد أو الكيان هو ما يدعونه. إدارة الجلسة Session Management هي العملية التي من خلالها يحافظ المخدم على حالة مصادقة المستخدمين بحيث يمكن للمستخدم الاستمرار في استخدام النظام دون إعادة المصادقة. توفر NIST Special Publication 800-63B: Digital Identity Guidelines (Authentication and Lifecycle Management إرشادات متينة حول تنفيذ ضوابط الأمان في الهوية الرقمية والمصادقة وإدارة الجلسة.

تجد أدناه بعض التوصيات للتنفيذ الآمن.

مستويات المصادقة

تصف NIST 800-63b ثلاثة مستويات لضمان المصادقة تسمى مستوى ضمان المصادقة authentication assurance level (AAL). مستوى AAL 1 محجوز للتطبيقات منخفضة المخاطر التي لا تحتوي على معلومات PII أو بيانات خاصة أخرى. في المستوى الأول من AAL ، لا يلزم سوى مصادقة أحادية العامل single-factor authentication، عادةً تتم من خلال استخدام كلمة مرور.

المستوى 1 : كلمة المرور

كلمات المرور مهمة حقًا. نحتاج إلى سياسة policy، نحتاج إلى تخزينها بشكل آمن ، نحتاج أحيانًا إلى السماح للمستخدمين بإعادة تعيينها.

متطلبات كلمة المرور

يجب أن تمتثل كلمات المرور للمتطلبات التالية على الأقل:

- يكون طولها 8 أحرف على الأقل إذا تم أيضًا استخدام المصادقة متعددة العوامل multi-factor authentication (MFA) والضوابط الأخرى. إذا لم يكن MFA ممكنًا ، فيجب زيادتها إلى 10 أحرف على الأقل.
- يجب أن تكون جميع أحرف طباعة ASCII بالإضافة إلى محرف المسافة مقبولة في الأسرار المحفوظة memorized secrets.
- تشجيع استخدام كلمات المرور وعبارات المرور passphrases الطويلة.

- إزالة متطلبات التعقيد complexity حيث ثبت أن هذه محدودة الفعالية. بدلاً من ذلك ، يوصى باعتماد MFA أو كلمات المرور الأطول.
- تأكد من أن كلمات المرور المستخدمة ليست كلمات مرور شائعة الاستخدام تم تسريبها بالفعل في اختراق سابق. يمكنك حظر أهم 1000 أو 10000 كلمة مرور شائعة والتي تفي بمتطلبات الطول المذكورة أعلاه والموجودة في قوائم كلمات المرور المخترقة. يحتوي الرابط التالي على أكثر كلمات المرور شيوعًا:

https://github.com/danielmiessler/SecLists/tree/master/Passwords

تنفيذ آلية استعادة كلمة المرور الآمنة

من الشائع أن يكون للتطبيق آلية تمكن المستخدم من الوصول إلى حسابه في حالة نسيان كلمة المرور الخاصة به. سيستخدم التصميم الجيد لسير عمل ميزة استعادة كلمة المرور عناصر مصادقة متعددة العوامل. على سبيل المثال ، قد يطرح سؤال أمان - شيئًا يعرفونه something they know ، ثم يرسل رمزًا تم إنشاؤه إلى جهاز - شيئًا يمتلكونه something they own .

لمزيد من المعلومات يرجى الاطلاع على Forgot_Password_Cheat_Sheet و

Choosing_and_Using_Security_Questions_Cheat_Sheet .

التنفيذ الآمن لتخزين كلمات المرور

من أجل توفير ضوابط مصادقة قوية ، يجب أن يقوم التطبيق بتخزين بيانات اعتماد المستخدم بشكل آمن. إضافة لذلك ، يجب أن تكون ضوابط التشفير في مكانها الصحيح بحيث إذا تم اختراق بيانات الاعتماد (على سبيل المثال ، كلمة المرور) ، فلن يتمكن المهاجم على الفور من الوصول إلى هذه المعلومات.

مثال **PHP** لتخزين كلمات المرور

يوجد أدناه مثال لتجزئة hashing آمنة لكلمة المرور في PHP باستخدام وظيفة password_hash() (المتوفرة منذ 5.5.0) والتي تستخدم خوارزمية bcrypt بشكل افتراضي. يستخدم المثال عامل العمل work factor بقيمة 15.

```
<?php
  $cost = 15;
  $password_hash = password_hash("secret_password", PASSWORD_DEFAULT, ["cost" =>
$cost] );
?>
```

لمزيد من المعلومات يرجى الاطلاع على OWASP Password Storage Cheat Sheet

المستوى 2 : المصادقة متعددة العوامل

AAL 2 من NIST 800-63b مخصص للتطبيقات عالية الخطورة التي تحتوي على "معلومات تحديد الهوية الشخصية المؤكدة ذاتيًا أو المعلومات الشخصية الأخرى المتاحة عبر الإنترنت." في AAL المستوى 2 ، يلزم وجود مصادقة متعددة العوامل بما في ذلك OTP أو غيرها من أشكال التنفيذ متعدد العوامل.

تضمن المصادقة متعددة العوامل (MFA) أن يكون المستخدمون كما يدعون عن طريق مطالبتهم بتعريف أنفسهم بمزيج من:

- شيئاً تعرفه Something you know – كلمة المرور أو PIN.
- شيئاً تملكه Something you own– الرمز المميز token أو رقم الهاتف.
- شيئاً أنت عليه Something you are– القياسات الحيوية ، مثل بصمة الإصبع.

استخدام كلمات المرور كعامل وحيد يوفر أمانًا ضعيفًا. توفر الحلول متعددة العوامل حلاً أكثر قوة من خلال مطالبة المهاجم بالحصول على أكثر من عنصر واحد للمصادقة مع الخدمة.

تجدر الإشارة إلى أن القياسات الحيوية ، عند استخدامها كعامل وحيد للمصادقة، لا تعتبر أسرارًا مقبولة للمصادقة الرقمية. يمكن الحصول عليها عبر الإنترنت أو عن طريق التقاط صورة لشخص به هاتف مزود بكاميرا (على سبيل المثال ، صور الوجه) بعلمه أو بدون علمه ، أو رفعها عن الأشياء التي يلمسها شخص ما (على سبيل المثال ، بصمات الأصابع الكامنة) ، أو التقاطها بصور عالية الدقة (على سبيل المثال ، أنماط قزحية العين). يجب استخدام المقاييس الحيوية فقط كجزء من مصادقة متعددة العوامل باستخدام مصدق authenticator فعلي (شيء تملكه). على سبيل المثال ، الوصول إلى جهاز متعدد العوامل يستخدم كلمة مرور لمرة واحدة one-time password (OTP) والذي سينشئ كلمة مرور لمرة واحدة يقوم المستخدم بإدخالها يدويًا للمدقق verifier.

المستوى 3 : المصادقة القائمة على التشفير

مستوى ضمان المصادقة 3 (AAL3) في NIST 800-63b مطلوب عندما يمكن أن يؤدي تأثير الأنظمة المخترقة إلى ضرر شخصي أو خسارة مالية كبيرة أو الإضرار بالمصلحة العامة أو ينطوي على انتهاكات مدنية أو جنائية. يتطلب AAL3 مصادقة "تستند إلى إثبات حيازة مفتاح من خلال بروتوكول تشفير." يستخدم هذا النوع من المصادقة لتحقيق أقوى مستوى من ضمان المصادقة. يتم ذلك عادةً من خلال وحدات تشفير الأجهزة hardware cryptographic modules .

إدارة الجلسة

بمجرد نجاح مصادقة المستخدم الأولية ، قد يختار التطبيق تتبع حالة المصادقة هذه والحفاظ عليها لفترة محدودة من الوقت. سيسمح هذا للمستخدم بمواصلة استخدام التطبيق دون الحاجة إلى إعادة المصادقة مع كل طلب. يسمى تتبع حالة المستخدم هذه إدارة الجلسة.

إنشاء الجلسة وانتهاء صلاحيتها

يتم تعقب حالة المستخدم في الجلسة. يتم تخزين هذه الجلسة عادةً على المخدم لإدارة الجلسة التقليدية المستندة إلى الويب. ثم يتم إعطاء معرف الجلسة session identifier للمستخدم حتى يتمكن المستخدم من تحديد جلسة من جهة المخدم تحتوي على بيانات المستخدم الصحيحة. يحتاج العميل فقط إلى الاحتفاظ بمعرف الجلسة هذا ، والذي يحافظ أيضًا على بيانات الجلسة الحساسة من جهة المخدم بعيدًا عن العميل.

فيما يلي بعض الضوابط التي يجب أخذها بعين الاعتبار عند بناء أو تنفيذ حلول إدارة الجلسة:

- تأكد من أن معرف الجلسة طويل وفريد وعشوائي.
- يجب على التطبيق إنشاء جلسة جديدة أو على الأقل تدوير rotate معرف الجلسة أثناء المصادقة وإعادة المصادقة.
- يجب أن يقوم التطبيق بتنفيذ مهلة الخمول idle بعد فترة من عدم النشاط وأقصى عمر مطلق لكل جلسة ، وبعد ذلك يجب على المستخدمين إعادة المصادقة. يجب أن يتناسب طول المهلات عكسياً مع قيمة البيانات المحمية.

يرجى الاطلاع على مزيد من التفاصيل من خلال زيارة الرابط Session Management Cheat Sheet. يغطي القسم 3 من ASVS متطلبات إدارة الجلسة الإضافية.

ملفات تعريف الارتباط للمتصفح *Browser Cookies*

ملفات تعريف الارتباط بالمتصفح هي طريقة شائعة لتطبيق الويب لتخزين معرفات الجلسة لتطبيقات الويب التي تطبق تقنيات إدارة الجلسة القياسية. فيما يلي بعض الإجرائيات الدفاعية التي يجب مراعاتها عند استخدام ملفات تعريف الارتباط بالمتصفح.

- عند استخدام ملفات تعريف الارتباط في المتصفح كآلية لتتبع جلسة المستخدم المصادق عليه ، يجب أن تكون متاحة لأقل مجموعة من النطاقات domains والمسارات paths ويجب وضع علامات عليها tagged لتنتهي صلاحيتها في فترة صلاحية الجلسة أو بعدها بقليل.
- يجب تعيين العلم "secure" لضمان إجراء النقل عبر القناة الآمنة فقط (TLS).

- يجب تعيين العلم HttpOnly لمنع الوصول إلى ملف تعريف الارتباط عبر JavaScript.
- تؤدي إضافة سمات "samesite" إلى ملفات تعريف الارتباط إلى منع بعض المتصفحات الحديثة من إرسال ملفات تعريف الارتباط مع طلبات عبر المواقع وتوفر الحماية ضد تزوير الطلبات عبر المواقع cross-site request forgery وهجمات تسرب المعلومات information leakage attacks.

الرموز المميزة *Tokens*

يمكن أن تكون الجلسات من جهة المخدم محدودة لبعض أشكال المصادقة. تسمح "الخدمات عديمة الحالة Stateless services " بإدارة بيانات الجلسة من جهة العميل لأغراض الأداء بحيث يكون للمخدم عبء أقل للتخزين والتحقق من جلسة المستخدم. تنشئ هذه التطبيقات "عديمة الحالة stateless " رمز مميز للوصول قصير العمر يمكن استخدامه لمصادقة طلب العميل دون إرسال بيانات اعتماد المستخدم بعد المصادقة الأولية.

JWT (JSON Web Tokens)

JSON Web Token (JWT) هو معيار مفتوح (RFC 7519) يحدد طريقة مدمجة وقائمة بذاتها لنقل المعلومات بشكل آمن بين الأطراف ككائن JSON. يمكن التحقق من هذه المعلومات والوثوق بها لأنها موقعة رقمياً. يتم إنشاء رمز JWT أثناء المصادقة ويتم التحقق منه بواسطة المخدم (أو المخدمات) قبل أي معالجة.

ومع ذلك ، غالبًا لا يتم حفظ JWT بواسطة المخدم بعد الإنشاء الأولي. عادةً ما يتم إنشاء JWT's ومن ثم تسليمها إلى العميل دون حفظها بواسطة المخدم بأي شكل من الأشكال. يتم الحفاظ على سلامة الرمز المميز من خلال استخدام التوقيعات الرقمية حتى يتمكن المخدم لاحقًا من التحقق من أن JWT لا تزال صالحة ولم يتم العبث بها منذ إنشائها.

هذا النهج عديم الحالة ومحمول portable بالطريقة التي يمكن أن تختلف بها تقنيات العميل والمخدم مع استمرار التفاعل.

تحذير

تعد الهوية الرقمية والمصادقة وإدارة الجلسات من الموضوعات الكبيرة جدًا. نحن نقوم بالخرمشة على سطح موضوع الهوية الرقمية هنا. تأكد من أن المواهب الهندسية الأكثر قدرة لديك هي المسؤولة عن الحفاظ على التعقيد الذي تنطوي عليه معظم حلول الهوية.

نقاط الضعف التي تم منعها

- OWASP Top 10 2017 A2- Broken Authentication and Session Management
- OWASP Mobile Top 10 2014-M5- Poor Authorization and Authentication

المراجع

- OWASP Cheat Sheet: Authentication
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: Forgot Password
- OWASP Cheat Sheet: Choosing and Using Security Questions
- OWASP Cheat Sheet: Session Management
- OWASP Cheat Sheet: IOS Developer
- OWASP Testing Guide: Testing for Authentication
- NIST Special Publication 800-63 Revision 3 - Digital Identity Guidelines

الأدوات

- Daniel Miessler: Most commonly found passwords

ضوابط **OWASP** الاستباقية النسخة **3.0**

ض7: فرض ضوابط الوصول

التوصيف

التحكم في الوصول Access Control (أو التفويض Authorization) هو عملية منح أو رفض طلبات محددة من مستخدم أو برنامج أو عملية. يتضمن التحكم في الوصول أيضًا إجرائيات منح granting وإلغاء revoking تلك الامتيازات.

وتجدر الإشارة إلى أن التفويض (التحقق من الوصول إلى ميزات أو موارد معينة) تختلف عن المصادقة (التحقق من الهوية).

غالبًا ما تشمل وظيفة التحكم في الوصول إلى العديد من ميزات البرامج اعتمادًا على مدى تعقيد نظام التحكم في الوصول. على سبيل المثال ، غالبًا ما تكون إدارة البيانات الوصفية للتحكم في الوصول managing access control metadata أو بناء التخزين المؤقت building caching لأغراض قابلية التوسع scalability purposes هي مكونات إضافية في نظام التحكم في الوصول التي تحتاج إلى الإنشاء أو الإدارة.

هناك عدة أنواع مختلفة من تصميم التحكم في الوصول التي يجب أخذها في الاعتبار.

- التحكم في الوصول التقديري Discretionary Access Control (DAC) وهي طريقة لتقييد الوصول إلى الكائنات (على سبيل المثال ، الملفات وكيانات البيانات data entities) بناءً على الهوية identity وما هناك حاجة إلى معرفته need-to-know (مثل المستخدمين والعمليات) و / أو المجموعات التي ينتمي إليها الكائن.
- التحكم في الوصول الإلزامي Mandatory Access Control (MAC) هو وسيلة لتقييد الوصول إلى موارد النظام بناءً على الحساسية (كما تشير التسمية) للمعلومات الواردة في مورد النظام والتفويض الرسمي (أي الترخيص clearance) للمستخدمين للوصول إلى معلومات بمثل هذه الحساسية.
- التحكم في الوصول المستند إلى الصلاحية Role Based Access Control (RBAC) هو نموذج للتحكم في الوصول إلى الموارد حيث يتم تحديد الإجراءات المسموح بها على الموارد بالصلاحيات أو الأدوار بدلاً من الهويات الفردية.
- التحكم في الوصول المستند إلى السمات Attribute Based Access Control (ABAC) وهي طريقة توافق على طلبات المستخدم أو ترفضها بناءً على السمات العشوائية arbitrary attributes للمستخدم والسمات العشوائية للكائن ، وظروف البيئة التي يمكن التعرف عليها بالمجال العام globally وأكثر صلة بالسياسات المطروحة.

مبادئ تصميم التحكم في الوصول

يجب مراعاة متطلبات تصميم التحكم في الوصول "الإيجابية" التالية في المراحل الأولى من تطوير التطبيق.

1) تصميم التحكم في الوصول بدقة في المقدمة

بمجرد اختيار نمط تصميم محدد للتحكم في الوصول ، غالبًا ما يكون من الصعب ويستغرق وقتًا طويلاً لإعادة هندسة التحكم في الوصول في تطبيقك بنمط جديد. يعد التحكم في الوصول أحد المجالات الرئيسية لتصميم أمان التطبيق الذي يجب تصميمه بدقة مقدمًا ، لا سيما عند معالجة متطلبات مثل تعدد المستأجرين multi-tenancy والتحكم في الوصول الأفقي (المعتمد على البيانات) horizontal access control (data dependent).

قد يبدأ تصميم التحكم في الوصول بشكل بسيط ولكنه غالبًا ما يتطور إلى ضوابط أمنية معقدة ومليئة بالميزات. عند تقييم قدرة التحكم في الوصول لأطر عمل البرامج ، تأكد من أن وظيفة التحكم في الوصول ستسمح بالتخصيص لاحتياجاتك الخاصة بميزة التحكم في الوصول.

2) إجبار جميع الطلبات على الخضوع لفحوصات التحكم في الوصول

تأكد من أن كل الطلبات تمر عبر نوع من طبقة التحقق من التحكم في الوصول access control verification layer. تعد بعض التقنيات مثل عوامل تصفية Java (Java filters) أو غيرها من آليات معالجة الطلبات الآلية أدوات برمجة مثالية من شأنها أن تساعد في ضمان أن جميع الطلبات تمر عبر نوع من فحص التحكم في الوصول.

3) الرفض هو الحالة الافتراضية *Deny by default*

الرفض افتراضيًا (أو الرفض هو الحالة الافتراضية) هو مبدأ أنه في حالة عدم السماح للطلب على وجه التحديد ، يتم رفضه. هناك العديد من الطرق التي ستظهر بها هذه القاعدة في كود التطبيق. بعض الأمثلة على ذلك هي:

1. قد يؤدي كود التطبيق إلى ظهور خطأ أو استثناء أثناء معالجة طلبات التحكم في الوصول. في هذه الحالات ، يجب دائمًا رفض التحكم في الوصول.
2. عندما ينشئ مسؤول مستخدمًا جديدًا أو يسجل مستخدم لحساب جديد ، يجب أن يكون لهذا الحساب حد أدنى من الوصول أو لا يمتلك حق الوصول بشكل افتراضي حتى يتم تكوين هذا الوصول.
3. عند إضافة ميزة جديدة إلى تطبيق ما ، يجب منع جميع المستخدمين من استخدام هذه الميزة حتى يتم تكوينها بشكل صحيح.

4) مبدأ الامتيازات الأدنى *Principle of Least Privilege*

تأكد من أن جميع المستخدمين أو البرامج أو العمليات يتم منحهم فقط أقل قدر ممكن من الوصول الضروري. كن حذرًا من الأنظمة التي لا توفر إمكانات تكوين التحكم في الوصول بشكل مفصل granular .

5) عدم كتابة الأدوار بشكل ثابت برمجياً *Don't Hardcode Roles*

تعتمد العديد من أطر التطبيقات الافتراضية للتحكم في الوصول على الدور. من الشائع العثور على كود لتطبيق برمجي مليء بالعديد من عمليات التحقق من هذا النوع.

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) {
      deleteAccount();
}
```

كن حذرًا بشأن هذا النوع من البرمجة المستندة إلى الأدوار role-based programming في الكود. فهي تتضمن القيود أو المخاطر التالية.

- البرمجة القائمة على الدور هي هشة، من السهل إغفال أو إنشاء عمليات تحقق من الأدوار غير صحيحة.
- لا يسمح هذا النوع من البرمجة ب تعدد المستأجرين multi-tenancy. ستكون الإجراءات المتطرفة مثل تفريع الكود forking the code أو عمليات التحقق المضافة لكل عميل مطلوبة للسماح للأنظمة القائمة على الدور بأن يكون لها قواعد مختلفة للعملاء المختلفين.
- لا تسمح البرمجة القائمة على الدور بقواعد التحكم في الوصول الأفقية أو الخاصة بالبيانات data-specific or horizontal access control rules .
- قد يكون من الصعب تدقيق قواعد التعليمات البرمجية codebases الكبيرة التي تحتوي على العديد من فحوصات التحكم في الوصول أو التحقق من السياسة العامة للتحكم في الوصول إلى التطبيق.

بدلاً من ذلك ، يرجى مراعاة منهجية برمجة التحكم في الوصول التالية:

```
if (user.hasAccess("DELETE_ACCOUNT")) {
      deleteAccount();
}
```

تعد عمليات فحص التحكم في الوصول القائمة على السمات أو الميزات Attribute or feature-based من هذا النوع نقطة البداية لبناء أنظمة تحكم في الوصول مصممة جيدًا well-designed وغنية بالميزات feature-rich. يسمح هذا النوع من البرمجة أيضًا بإمكانية تخصيص التحكم في الوصول بشكل أكبر بمرور الوقت.

6) قم بتسجيل *log* كل أحداث التحكم في الوصول

يجب تسجيل جميع حالات فشل التحكم في الوصول لأنها قد تشير إلى مستخدم ضار يبحث في التطبيق عن ثغرات أمنية.

نقاط الضعف التي تم منعها

- OWASP Top 10 2017-A5-Broken Access Control
- OWASP Mobile Top 10 2014-M5 Poor Authorization and Authentication

المراجع

- OWASP Cheat Sheet: Access Control
- OWASP Cheat Sheet: iOS Developer - Poor Authorization and Authentication
- OWASP Testing Guide: Testing for Authorization

الأدوات

- OWASP ZAP مع الإضافة add-on الاختيارية Access Control Testing

ضوابط **OWASP** الاستباقية النسخة **3.0**

ض8: حماية البيانات في كل مكان

التوصيف

تتطلب البيانات الحساسة مثل كلمات المرور وأرقام بطاقات الائتمان والسجلات الصحية والمعلومات الشخصية وأسرار العمل حماية إضافية ، لا سيما إذا كانت تلك البيانات تندرج تحت قوانين الخصوصية (اللائحة العامة لحماية البيانات في الاتحاد الأوروبي EU's General Data Protection Regulation GDPR) ، وقواعد حماية البيانات المالية مثل PCI Data Security Standard (PCI DSS) أو لوائح أخرى.

يمكن للمهاجمين سرقة البيانات من تطبيقات الويب وخدمات الويب بعدة طرق. على سبيل المثال ، إذا تم إرسال معلومات حساسة عبر الإنترنت بدون تأمين الاتصالات ، فيمكن للمهاجم على اتصال لاسلكي مشترك shared wireless connection رؤية بيانات مستخدم آخر وسرقتها. أيضًا ، يمكن للمهاجم استخدام حقن SQL لسرقة كلمات المرور وبيانات الاعتماد الأخرى من قاعدة بيانات التطبيقات وكشف هذه المعلومات للجميع.

تصنيف البيانات

من الأهمية تصنيف البيانات في نظامك وتحديد مستوى الحساسية الذي ينتمي إليه كل جزء من البيانات. يمكن بعد ذلك ربط كل فئة من فئات البيانات بقواعد الحماية اللازمة لكل مستوى من مستويات الحساسية. على سبيل المثال ، يمكن تصنيف معلومات التسويق العامة غير الحساسة على أنها بيانات عامة لا بأس بوضعها على موقع الويب العام. قد يتم تصنيف أرقام بطاقات الائتمان على أنها بيانات مستخدم خاصة قد تحتاج إلى تشفيرها أثناء تخزينها أو نقلها.

تشفير البيانات عند النقل

عند إرسال بيانات حساسة عبر أي شبكة ، ينبغي مراعاة أمان الاتصالات من طرف إلى طرف (أو التشفير أثناء النقل encryption-in-transit) . TLS حتى الآن هو بروتوكول التشفير الأكثر شيوعًا والأكثر دعمًا على نطاق واسع لأمان الاتصالات. يتم استخدامه من قبل العديد من أنواع التطبيقات (الويب ، خدمة الويب ، الهاتف المحمول) للتواصل عبر شبكة بطريقة آمنة. يجب تكوين TLS بشكل صحيح بعدة طرق لحماية الاتصالات الآمنة بشكل صحيح.

تتمثل الفائدة الأساسية لأمان طبقة النقل transport layer security في حماية بيانات تطبيقات الويب من الكشف والتعديل غير المصرح به عند نقلها بين العملاء (متصفحات الويب) ومخدم تطبيق الويب ، وبين مخدم تطبيق الويب الجهة الخلفية back-end و المكونات الأخرى غير المعتمدة على المتصفح non-browser based enterprise components.

تشفير البيانات في حالة الراحة Encrypting Data at Rest

القاعدة الأولى لإدارة البيانات الحساسة هي تجنب تخزين البيانات الحساسة عندما يكون ذلك ممكنًا. إذا كان يجب عليك تخزين البيانات الحساسة ، فتأكد من أنها محمية بشكل مشفر بطريقة ما لتجنب الكشف والتعديل غير المصرح به.

يعد التشفير (أو crypto) أحد الموضوعات الأكثر تقدمًا في مجال أمن المعلومات ، والذي يتطلب فهمه تعليم وخبرة كبيرتين. من الصعب الحصول على معلومات صحيحة نظرًا لوجود العديد من الأساليب للتشفير ، ولكل منها مزايا وعيوب يجب أن يفهمها مهندسو ومطورو حلول الويب تمامًا. بالإضافة إلى ذلك ، تعتمد أبحاث التشفير الجادة عادةً على الرياضيات المتقدمة ونظرية الأعداد ، مما يوفر حاجزًا خطيرًا أمام الدخول.

بدلاً من بناء القدرة على التشفير cryptographic capability من البداية ، يوصى بشدة باستخدام حلول مفتوحة ومراجعة الأقران ، مثل مشروع Google Tink و Libsodium وإمكانية التخزين الآمن المضمنة في العديد من أطر عمل البرامج والخدمات السحابية cloud services.

تطبيق الهاتف المحمول: تخزين محلي آمن *Secure Local Storage*

تتعرض تطبيقات الهاتف المحمول لخطر تسرب البيانات بشكل خاص لأن الأجهزة المحمولة تُفقد أو تُسرق بانتظام وهي في النهاية تحتوي على بيانات حساسة.

كقاعدة عامة ، يجب تخزين الحد الأدنى من البيانات المطلوبة فقط على الهاتف المحمول. ولكن إذا كان لا بد من تخزين البيانات الحساسة على هاتف محمول ، فيجب تخزين البيانات الحساسة داخل دليل تخزين بيانات محدد لأنظمة تشغيل الهواتف المحمولة mobile operating systems specific data storage directory. على نظام Android هو متجر مفاتيح Android (Android keystore) وفي نظام هو سلسلة مفاتيح iOS (iOS keychain).

دورة حياة المفتاح *Key Lifecycle*

تستخدم المفاتيح السرية Secret keys في العديد من الوظائف الحساسة للتطبيقات. على سبيل المثال ، يمكن استخدام المفاتيح السرية للتوقيع على JWTs وحماية بطاقات الائتمان وتوفير أشكال مختلفة من المصادقة بالإضافة إلى تسهيل ميزات الأمان الحساسة الأخرى. في إدارة المفاتيح ، يجب اتباع عدد من القواعد بما في ذلك:

- تأكد من أن أي مفتاح سري محمي من الوصول غير المصرح به.
- قم بتخزين المفاتيح في مخزن أسرار secrets vault مناسب كما هو موضح أدناه.
- استخدم مفاتيح مستقلة عندما تكون هناك حاجة إلى مفاتيح متعددة.
- قم بوضع الدعم التقني لتغيير الخوارزميات والمفاتيح عند الحاجة.
- قم ببناء ميزات التطبيق للتعامل مع دوران المفتاح key rotation.

إدارة أسرار التطبيق *Application Secrets Management*

تحتوي التطبيقات على العديد من "الأسرار secrets" اللازمة للعمليات الأمنية. ويشمل ذلك الشهادات certificates وكلمات مرور اتصال SQL (SQL connection passwords) وبيانات اعتماد حساب خدمة الطرف الثالث (third party service account credentials) وكلمات المرور ومفاتيح SSH ومفاتيح التشفير والمزيد. قد يؤدي الكشف غير المصرح به أو تعديل هذه الأسرار إلى اختراق النظام بالكامل. في إدارة أسرار التطبيق ، ضع في اعتبارك ما يلي:

- لا تخزن الأسرار في التعليمات البرمجية أو ملفات التكوين أو تمررها عبر متغيرات البيئة environment variables. استخدم أدوات مثل GitRob أو TruffleHog لمسح المستودعات البرمجية بحثًا عن الأسرار.
- احتفظ بالمفاتيح والأسرار الأخرى على مستوى التطبيق application-level secrets في مخزن أسرار secrets vault مثل KeyWhiz أو Vault project الخاصة بـ Hashicorp أو Amazon KMS لتوفير تخزين آمن والوصول إلى الأسرار على مستوى التطبيق في وقت التشغيل run-time.

نقاط الضعف التي تم منعها

- OWASP Top 10 2017 - A3: Sensitive Data Exposure
- OWASP Mobile Top 10 2014-M2 Insecure Data Storage

المراجع

- OWASP Cheat Sheet: Transport Layer Protection
- Ivan Ristic: SSL/TLS Deployment Best Practices
- OWASP Cheat Sheet: HSTS
- OWASP Cheat Sheet: Cryptographic Storage
- OWASP Cheat Sheet: Password Storage
- OWASP Cheat Sheet: IOS Developer - Insecure Data Storage
- OWASP Testing Guide: Testing for TLS

الأدوات

- SSLyze - مكتبة فحص تكوين SSL وأداة CLI.
- SSLLabs - خدمة مجانية للمسح والتحقق من تكوين SSL / TLS.
- OWASP O-Saft TLS Tool - أداة اختبار اتصال TLS.
- GitRob - أداة تعمل في مفسر الأوامر command line للعثور على معلومات حساسة في الملفات المتاحة للعامة على GitHub.
- TruffleHog - عمليات البحث عن الأسرار التي تم حفظها committed عن طريق الخطأ.
- KeyWhiz – مدير أسرار.
- Hashicorp Vault – مدير أسرار.
- Amazon KMS - إدارة المفاتيح على Amazon AWS.

ضوابط **OWASP** الاستباقية النسخة **3.0**

ض9: التسجيل **logging** والمراقبة الأمنية

التوصيف

التسجيل Logging هو مفهوم يستخدمه معظم المطورين بالفعل لأغراض التصحيح debugging والتشخيص diagnostic. يعد تسجيل الأمان مفهومًا أساسيًا بنفس القدر: لتسجيل معلومات الأمان في وقت تشغيل أحد التطبيقات. المراقبة هي المراجعة الحية لسجلات التطبيق والأمان باستخدام أشكال مختلفة من الأتمتة. يمكن استخدام نفس الأدوات والأنماط للعمليات وتصحيح الأخطاء وأغراض الأمان.

فوائد تسجيل الأمان Security Logging

يمكن استخدام التسجيل الأمني لـ :

1) تغذية أنظمة كشف التسلل IDS
2) تحليل الأدلة الرقمية Forensic analysis والتحقيقات
3) التحقق من الامتثال للمتطلبات التنظيمية

تنفيذ تسجيل الأمان

فيما يلي قائمة بأفضل الممارسات لتطبيق تسجيل الأمان:

- اتبع منهجية وتنسيق format للتسجيل داخل النظام وعبر أنظمة المؤسسة. من الأمثلة على إطار عمل التسجيل الشائع خدمات Apache Logging التي تساعد على توفير تناسق في التسجيل بين تطبيقات Java و PHP و .NET و C++.
- لا تسجل أكثر أو أقل من اللازم. على سبيل المثال ، تأكد دائمًا من تسجيل الوقت timestamp ومعلومات التعريف بما في ذلك source IP و user-id ، واحرص على عدم تسجيل البيانات الخاصة أو السرية.
- انتبه جيدًا لمزامنة الوقت عبر العقد time syncing across nodes لضمان تناسق الأوقات timestamps.

تسجيل كشف التسلل والاستجابة

استخدم التسجيل لتحديد النشاط الذي يشير إلى أن المستخدم يتصرف بشكل ضار. يتضمن النشاط الذي يُحتمل أن يكون ضارًا للتسجيل ما يلي:

- البيانات المرسلة التي تقع خارج النطاق الرقمي المتوقع.
- البيانات المقدمة Submitted data التي تتضمن تغييرات على البيانات التي يجب ألا تكون قابلة للتعديل (select list ، checkbox، أو أي مدخل آخر).
- الطلبات التي تنتهك قواعد التحكم في الوصول من جهة المخدم.
- تتوفر قائمة أكثر شمولاً لنقاط الكشف المحتملة هنا.

عندما يواجه تطبيقك مثل هذا النشاط ، يجب على التطبيق الخاص بك على الأقل تسجيل النشاط ووضع علامة عليه كمشكلة شديدة الخطورة. من الناحية المثالية ، يجب أن يستجيب تطبيقك أيضًا للهجوم المحتمل الذي تم تحديده ، على سبيل المثال من خلال إبطال invalidating جلسة المستخدم وقفل locking حساب المستخدم. تسمح آليات الاستجابة للبرنامج بالرد في الوقت الفعلي على الهجمات المحتملة المحددة.

تصميم التسجيل الآمن

يجب بناء حلول التسجيل وإدارتها بطريقة آمنة. قد يتضمن تصميم التسجيل الآمن ما يلي:

- قم بتشفير والتحقق من صحة أي أحرف خطيرة قبل التسجيل لمنع هجمات حقن السجل log injection أو هجمات تزوير السجل log forging .
- لا تسجل المعلومات الحساسة. على سبيل المثال ، لا تقم بتسجيل كلمة المرور أو معرف الجلسة session ID , أو بطاقات الائتمان أو أرقام الضمان الاجتماعي social security numbers.
- قم بحماية سلامة السجل log integrity. قد يحاول المهاجم العبث بالسجلات. لذلك ، ينبغي النظر في إذن ملفات السجل وتدقيق تغييرات السجل.
- قم بتمرير forward السجلات من الأنظمة الموزعة إلى خدمة تسجيل مركزية وآمنة. سيضمن هذا عدم إمكانية فقد بيانات السجلات في حالة اختراق عقدة node في النظام. هذا يسمح أيضا بالمراقبة المركزية.

المراجع

- OWASP AppSensor Detection Points - تستخدم نقاط الكشف لتحديد مستخدم ضار يبحث عن نقاط الضعف أو نقاط الضعف في التطبيق.
- OWASP Log injection
- OWASP Log forging
- OWASP Cheat Sheet: Logging كيفية التسجيل بشكل صحيح في التطبيق.
- OWASP Development Guide: Logging
- OWASP Code Review Guide: Reviewing Code for Logging Issues

الأدوات

- OWASP Security Logging Project
- Apache Logging Services

ضوابط **OWASP** الاستباقية النسخة **3.0**

ض10: معالجة **handle** كل الأخطاء والاستثناءات **exceptions**

التوصيف

معالجة الاستثناءات هي مفهوم برمجي يسمح للتطبيق بالاستجابة لحالات الخطأ المختلفة (مثل تعطل الشبكة أو فشل اتصال قاعدة البيانات ، إلخ) بطرق مختلفة. تعتبر معالجة الاستثناءات والأخطاء بشكل صحيح أمرًا بالغ الأهمية لجعل التعليمات البرمجية الخاصة بك موثوقة وآمنة.

تحدث معالجة الأخطاء والاستثناءات في جميع مجالات التطبيق بما في ذلك منطق الأعمال (business logic) المهم بالإضافة إلى ميزات الأمان وكود إطار العمل framework code.

تعتبر معالجة الأخطاء مهمة أيضًا من منظور اكتشاف التطفل intrusion. قد تؤدي هجمات معينة ضد تطبيقك إلى حدوث أخطاء يمكن أن تساعد في اكتشاف الهجمات الجارية.

الأخطاء في معالجة الخطأ

وجد الباحثون في جامعة تورنتو أنه حتى الأخطاء الصغيرة في معالجة الأخطاء أو نسيان التعامل مع الأخطاء يمكن أن تؤدي إلى إخفاقات كارثية في الأنظمة الموزعة catastrophic failures in distributed systems .

يمكن أن تؤدي الأخطاء في معالجة الأخطاء إلى أنواع مختلفة من الثغرات الأمنية.

- **تسريب المعلومات Information leakage**: إن تسريب المعلومات الحساسة في رسائل الخطأ قد يساعد المهاجمين عن غير قصد. على سبيل المثال ، يمكن لخطأ يُرجع تتبع الخطأ stack trace أو تفاصيل خطأ داخلي أخرى أن يزود المهاجم بمعلومات حول بيئتك. حتى الاختلافات الصغيرة في التعامل مع حالات الخطأ المختلفة (على سبيل المثال ، إعادة "مستخدم غير صالح invalid user " أو "كلمة مرور غير صالحة invalid password " لأخطاء المصادقة) يمكن أن توفر أدلة قيمة للمهاجمين. كما هو موضح أعلاه ، تأكد من تسجيل تفاصيل الخطأ لأغراض تحليل الأدلة الرقمية وتصحيح الأخطاء ، ولكن لا تعرض هذه المعلومات ، خاصةً لعميل خارجي.
- **تجاوز TLS (TLS bypass)**: كان "خطأ فشل" Apple goto Apple goto "fail bug" عبارة عن خطأ في تدفق التحكم control-flow في كود معالجة الخطأ الذي أدى إلى اختراق كامل لاتصالات TLS على أنظمة Apple.
- **تعطيل الخدمة DOS**: قد يؤدي النقص أو الضعف في معالجة الأخطاء الأساسية إلى إيقاف تشغيل النظام. عادة ما تكون هذه ثغرة أمنية يسهل على المهاجمين استغلالها. قد تؤدي مشكلات معالجة الأخطاء الأخرى إلى زيادة استخدام وحدة المعالجة المركزية CPU أو القرص disk بطرق قد تؤدي إلى تدهور degrade النظام.

نصيحة إيجابية

- إدارة الاستثناءات بطريقة مركزية centralized manner لتجنب تكرار كتل try/catch في التعليمات البرمجية. تأكد من التعامل مع جميع السلوكيات غير المتوقعة بشكل صحيح داخل التطبيق.
- تأكد من أن رسائل الخطأ المعروضة للمستخدمين لا تسرّب البيانات الهامة ، لكنها لا تزال مطولة بما يكفي لتمكين استجابة المستخدم المناسبة.
- تأكد من تسجيل الاستثناءات بطريقة توفر معلومات كافية للدعم support أو ضمان الجودة QA أو تحليل الأدلة الرقمية forensics أو فرق الاستجابة للحوادث incident response teams لفهم المشكلة.
- اختبر بعناية وتحقق من رمز معالجة الأخطاء error handling code .

المراجع

- OWASP Code Review Guide: Error Handling
- OWASP Testing Guide: Testing for Error Handling
- OWASP Improper Error Handling
- CWE 209: Information Exposure Through an Error Message
- CWE 391: Unchecked Error Condition

الأدوات

- Error Prone - أداة تحليل ستاتيكي static analysis من Google لاكتشاف الأخطاء الشائعة في معالجة الأخطاء لمطوري Java.
- من أشهر الأدوات المؤتمتة للعثور على الأخطاء في وقت التشغيل هي Chaos Monkey من Netflix ، والتي تعطل مثيلات النظام system instances عمدًا لضمان استعادة الخدمة بشكل كامل بشكل صحيح

الخاتمة

يجب النظر لهذا المستند على أنه نقطة بداية لا أنه مجموعة شاملة من التقنيات والممارسات. نؤكد مجدداً أن الغاية من هذا المستند هو تقديم توعية أساسية ومبدئية حول بناء البرمجيات الآمنة.

الخطوات التالية المقترحة والتي تساعد على بناء برمجيات آمنة هي:

1) فهم بعض المخاطر في أمن تطبيقات الويب ، يرجى الاطلاع على OWASP Top Ten و OWASP Mobile Top Ten.
2) كما يوضح الضابط الاستباقي الأول في هذا المستند، يجب أن يتضمن تطوير التطبيقات الآمن قائمة شاملة من متطلبات الأمان مستندة إلى معيار (أي أن تكون معيارية) كـ OWASP (Web) ASVS و OWASP (Mobile) MASVS.
3) لفهم نواة بناء المكونات الآمنة للتطبيقات البرمجية من منظور تفصيلي macro أكثر ، يرجى الاطلاع على OWASP OpenSAMM project.

إذا كان لديك أي استفسارات لفريق قادة المشروع يرجى الاتصال بقائمة البريد الالكتروني الموجودة في الرابط التالي:

https://lists.owasp.org/mailman/listinfo/owasp_proactive_controls.

OWASP
Open Web Application
Security Project

FOR DEVELOPERS