Coding Review Guide

Security and review guideline and practice, life cycle view

إعداد: أنيس حكمت أبوحميد

المقدمة

بسم الله الرحمن الرحيم

الحمد لله ربّ العالمين، يُحب من دعاه خفياً، ويُجيب من ناداه نجيّاً، ويزيدُ من كان منه حييّاً، ويكرم من كان له وفيّاً، ويهدي من كان صادق الوعد رضيّاً، الحمد لله ربّ العالمين.

العلوم التقنية علوم متنوعة كثيرة، تتعدد في وظائفها ومهامها، ومتابعة هذا التطور والإهتمام به من الأمور المهمة لكل من يرغب في تطوير قدراته العلمية والعملية، وهي من الأهمية بمكان لتجعل صاحب العلوم التقنية على قيد الحياة مع أقرانه مهما تعددت التقنيات وتجددت في عصره أو ما بعد عصره.

لذلك، وبناءا على ما سبق، سنتطرق في هذه الشرائح لأمر مهم في أي شيفرة برمجية، وهو **كيف يمكن مراجعة أي شيفرة برمجية وكيف يمكن اعتبارها شيفرة برمجية آمنة؟**، ومن هنا نبدأ بإذن الله.

قبل أن نبدأ

- هذه الشرائح تركز على مفاهيم ال coding review وال secure coding review، وقد تمت كتابة هذه الشرائح لتتناسب مع ال development plan الخاصة بفريق العمل ومهاراتهم.
- قد يتم الإشارة للعديد من المصطلحات التقنية هنا دون التطرق إليها بشكل مفصل، ويمكن البحث عنها بسهولة.
- تم عنونة بعض الشرائح لتتناسب مع طريقة السرد، لذلك قد تجد عناوين ترتبط بما قبلها مع أنها شريحة مستقلة وذلك للتفصيل وسهولة الشرح والسرد.

القاعدة الذهبية

القاعدة الذهبية قبل أن نبدأ، هي أننا نسعى لأن نحمي أنفسنا من الإختراق ومن المشاكل التقنية والأمنية قدر الإمكان، لكننا مهما كنا حريصين على ذلك، فإن المخترقين لديهم القدرة والوقت اللازم لتفحص الثغرات الموجودة والقيام بعمليات الاختراق حين العثور عليها، هي معركة غير عادلة، لكننا يمكن أن ننال شرف الصمود على أرض المعركة، وأن لا نكتفي بالاستسلام، بل يجب أن نبقى صامدون ونقاوم.

ماذا يقصد بال Secure Code Review

هي العملية التي تهدف لمراجعة الشيفرة البرمجية والتحقق من العيوب الأمنية الموجودة في التطبيق -البرنامج، الموقع..الخ- والمتعلقة بالمزايا الخاصة بهذا التطبيق، وطريقة التصميم الخاص به وبناء الشيفرة البرمجية بطريقة آمنة ومناسبة وبأقل عدد ممكن من المشاكل الأمنية، وبحيث تضمن أن التطبيق يمكنه الصمود أمام أكثر الهجمات شيوعا ولديه القدرة على حماية نفسه قدر الإمكان "self-defending".

يمكن القيام بعملية المراجعة الأمنية من خلال الجهد البشري أو من خلال استخدام بعض التطبيقات الموجودة، لكن على كل الأحوال لا يمكن الإستغناء عن الجهد البشري، ويمكن الإستفادة من التطبيقات خصوصا في التطبيقات ذات الشيفرة البرمجية الكبيرة لتحديد أماكن المشاكل الأمنية المتوقعة، ومن ثم قيام الخبير الأمني بمراجعة هذه الأماكن والتحقق من كل النتائج لكل العمليات ودراستها وسير العمل الخاص بها...

ما هو الفرق بين Code Review and Secure Code Review

يمكن القول أن أي شركة تقوم بتطوير تطبيقها تقوم بعمل مراجعة للشيفرة البرمجية، يمكن تقسيم هذه المراجعة إلى 5 درجات، وهذا المقياس (CMM)، فيكون أول مستوى هو المستوى الخاص بالشيفرة البرمجية التي تحتوي تكرار والنتائج غير مستقرة، وصولا إلى المستوى الخامس وهو بيئة تطوير منظمة ومنسقة وبأقل عدد من المشاكل، والشيفرة البرمجية موثقة، وهناك توثيق خاص بالمشروع، ويمر المشروع بمراحل للتحقق من أن العمل يسير بالشكل الصحيح، هذا كله يقبع ضمن ال Code Review.

وهنا يأتي دور ال Secure Code Review، وهي عبارة عن تحسين للنموذج السابق بما فيه وإعطاء أولوية وقدرة على اتخاذ القرار حول ما يخص المشاكل الأمنية المتوقعة وبناء نموذج وقواعد لسير العمل تراعي المشاكل الأمنية المتوقعة قبل حصولها، ويتعامل معها المطورين ضمن مجموعة من المعايير التي يجب أن يلتزموا بها أثناء كتابة الشيفرة البرمجية.

التوثيق وال Coding Review

إن من أكبر التحديات في كتابة أي شيفرة برمجية هي آلية وطريقة توثيقها، ويترواح التوثيق في الشركات بين الصفر إلى مستوى NASA -التوثيق في NASA يفوق حجم ال module-، واحدة من نقاط التوثيق التي قد تغيب عنا ويمكن الإستفادة منها بشكل فعال جدا هو كتابة شرح **لماذا استخدم المبرمج هذه الخوارزمية، ولمذا قام بوضع التسلسل للعمليات بشكل معين...**، هذا سيساعد أي مبرمج جديد أو قديم، بعد سنوات من الرجوع للشيفرة البرمجية وفهم سير العمل، بشكل سهل، ويكون دور المراجعة هنا في التحقق من أن **ما كتب هو الأفضل لحل المشكلة، وهل التوثيق صحيح، والتنفيذ صحيح...**، بالإضافة إلى هذا فإن هذه العملية تساعد في تقليل كمية ال bugs الناتجة لأي شيفرة برمجية، وتقليل ال bugs عند أي تعديل...

ال unit test وال Coding Review

واحدة من المعايير الجميلة التي يمكن أن تتخذها الشركة هي استخدام أو اجبار المبرمج على كتابة unit test، هذه ال unit test تحوي في طياتها النتائج المتوقعة لعملية ما بناءا على شيفرة برمجية موجودة، مثلا لو فرضنا وجود function sum(var1,var2) وكانت الأرقام في 1 و 2، فإن الناتج يجب أن يكون 3، فإن كان الناتج غير ذلك، سيكون هذا دليلا على وجود خطأ، ويمكن للمطور التحقق منه، فإن لم يقم بذلك، يأتي دور ال reviewer وعادة هنا يكون automated test، فينظر هل ال unit test موجودة، وهل نتيجة تنفيذها صحيحة، إن حصل أي خطأ فلن يتم رفع الشيفرة البرمجية الجديدة...

ما يتعلمه المبرمجين المبتدئين من ال code review

إن مراجعة الشيفرة البرمجية واحدة من أهم الطرق التي يتعلم منها المبرمجين المبتدئين ال (junior) -بعد تعلمهم البرمجة و قراءتهم لبعض الكتب-، هذه المراجعة تتحقق من خلال نقل المعرفة الضمنية للمبرمجين الأكثر الخبرة للآخرين، عملية نقل الخبرة هذه تقدم قفزات كبيرة في مستوى ال junior لما تقدمه له من قدرة على الإطلاع على حلول أخرى لحل مشكلة ما، وتقنيات ومكتبات تساعده على العمل، بالإضافة إلى أن المبرمجين ذوي الخبرة يتعلمون من هذا ال junior أساليب تقنية جديدة لحل المشكلة ربما قرأ عنها وتعلمها ولم يعلمها من هو أقدم منه، بالإضافة إلى الحلول التي قد تكون ذكية وابداعية لحل مشكلة ما، فتنتقل الفكرة من المبرمجين المبتدئين للمبرمجين الأكثر خبرة، وهذا يقودنا لقاعدة مهمة جدا، وهي أن ما صح من العلم وثبت هو فوق رأي الخبير أو المبتدئ، والخبير إن كان لا يأخذ إلا برأيه سينقرض أو سيسبقه الآخرون، والمبتدأ الذي يرفض التعلم، ويرفض مشورة واقتراح من ألقى إليه معلومة فلن يتطور أبدا...

اللهم إني أسألك الثبات في الأمر, والعزيمة على الرشد, وأسألك شكر نعمتك, وأسألك حسن عبادتك, وأسألك قلبا سليما, وأسألك لسانا صادقا, وأسألك من خير ما تعلم وأعوذ بك من شر ما تعلم, وأستغفرك لما تعلم, إنك أنت علام الغيوب

Familiarization with code base

واحدة من أهم النقاط التي تنبثق من مراجعة الشيفرة البرمجية بين أعضاء الفريق هي جعل الفريق متآلفا مع أجزاء الشيفرة البرمجية المختلفة والموجودة ضمن المشروع، هذا التآلف يعطي قوة لأي مطور متواجد في فريق العمل بأن يبدأ بتطوير أو تحسين هذه الإضافة بكل سهولة ودون الحاجة لتثبيت مجموعة من المهام عند شخص واحد، وبهذا فإن المعرفة تنتقل للجميع، والجميع تتشكل له نظرة شاملة لمجموع الشيفرة البرمجية، ومنها لممانعة أقل في الإنتقال من جزئية لجزئية والعمل عليها بشكل أسهل وأقل عبئا وإضاعة للوقت...

Pre-warning of integration clashes

عملية مراجعة الشيفرة البرمجية يمكن أن تقدم تحذيرا مبكرا عن أي خطأ غير متوقع لأي تعديل برمجي على الشيفرة البرمجية للمشروع في أي جزئية أو في مجموع أجزائه، خصوصا في الأجزاء والأقسام التي يعمل عليها أكثر من مطور في ذات الوقت، أو أن التعديل الخاص بها مرتبط بجزئية تقع تحت مسئولية مبرمج آخر، والسبب في ذلك يعود في تقليل نسبة الخطأ من خلال مراجعة الشيفرة البرمجية وتعديلاتها من قبل المطور الذي قد يتأثر بالتعديل، أو من خلال آخرين يدركون خطورة هذا التعديل أو المتطلبات اللازمة لتفعيل هذا التعديل...

الجوانب الفنية لمراجعة الشيفرة البرمجية الآمنة

هناك مجموعة من الجوانب الفنية لمراجعة أي شيفرة برمجية بنظرة الباحث الأمني، أو المراجع الأمني الذي يبحث عن المشاكل الأمنية الموجودة في الشيفرة البرمجية ويرغب بالتحقق منها، هذه النظرة يجب أن لا تكتفي فقط بالنظر للشيفرة البرمجية وطريقة كتابتها، بل يجب أن تكون هذه النظرة أوسع من ذلك لتفهم تسلسل العمل منذ البداية وحتى النهاية لكل وظيفة برمجية، وحصر جميع المدخلات التي سيتم استقبالها والتي سيتم معالجتها في الشيفرة البرمجية، وهذا يعني

1. التحقق من جميع ال input field وال validation الذي تم استخدامه معها، بالإضافة لدراسة جميع ما يمكن كتابته في هذا الحقل ومكان استخدامه وهل ال validation الموجود يغطي جميع هذه الحالات أم لا

الجوانب الفنية لمراجعة الشيفرة البرمجية الآمنة

2. أي sql query تعتمد على قيم (dynamic query) او أي log writer أو response تتم معالجته من قبل الشيفرة البرمجية يجب فحصه بدقة ودراسة جمع المشاكل الأمنية المحتملة...
3. المراجعة الأمنية تشمل أيضا دراسة جميع ال classes أو ال component التي ستمر من خلالها البيانات وصولا لمكان معالجة البيانات لإدخالها لقواعد البيانات أو إرجاعها على شكل response للمستخدم، وهذا يضمن التحقق من التسلسل الأمني لسير العمل في كل أجزاء المشروع وصولا لآخر مرحلة...
4. بالإضافة لذلك، فإن المراجعة الأمنية تشمل النظر إلى الحالات والأماكن المتوقعة لحصول ثغرة أمنية من خلالها (توقع أماكن حصولها) مثل ثغرات ال XSS أو توقع ثغرات خاصة بالتقنيات مثل ال css injection عند استخدام ال css in js.

العوامل التي يجب مراعاتها عند وضع خطة ل Security Review Code

عند التخطيط للعمل بنظام يحتوي في ثناياه على ال Securty Coding Review يجب الإنتباه إلى مجموعة من العوامل التي ستؤثر على المراجعة، ولكل عملية مراجعة سياقها الخاص، وحالتها المستقلة والتي تستلزم المراجعة بطريقة مختلفة عن الأخرى، باختلاف معدل التأثير....

هذه العوامل هي:

- الخطورة (risk): من المستحيل أن تكون أي شيفرة برمجية آمنة 100%، لكن يمكن قياس معدل الخطورة لأي شيفرة برمجية يتم تنفيذها حتى يتم وضع معايير قاسية لمراجعة أي تعديل فيها، وكلما زاد تصنيف الخطورة للخاصية المراد إضافتها أو تعديلها تزداد الإحتياطات الأمنية اللازمة لهذه الجزئية، ويجب منع أي شيفرة برمجية من الإنتقال للمرحلة التي تليها قبل التأكد من سلامتها -حتى لو كان عندك deadline-

العوامل التي يجب مراعاتها عند وضع خطة ل Security Review Code

- Purpose & Context: يجب تحديد السلوك الخاص بكل جزئية يتم مراجعتها على حدا
- Lines of Code: يفضل أن يتم تقسيم الشيفرة البرمجية بطريقة صحيحة تضمن أقل عدد ممكن من الأسطر البرمجية في داخل كل block of code، لإمكانية تحديد مكان الأخطاء المتوقعة بشكل أسهل وأسرع.
- Programming language: تختلف لغات البرمجة في مستوى أو معدل الخطورة للمشاكل الأمنية التي قد تتحقق من استخدامها لتنفيذ ميزة تقنية معينة، وهذا من الأمور التي يجب أن يتم أخذها بعين الاعتبار خصوصا اذا كان فريق العمل ليس له خبرة حقيقة في اللغة المقصودة، فمثلا مشاكل ال buffer overflows موجودة بال C/C++ بشكل أكبر بكثير من ال Java على سبيل المثال...
- Resources, Time & Deadlines: يجب الأخذ بعين الاعتبار المدة اللازمة لتسليم المشروع ككل، أو تسليم كل مزية أو اضافة بشكل مستقل يضمن وجود هذه الحلقة من المراجعة الأمنية ومن ضمن فترة الحياة لهذا المشروع...، مع الأخذ بعين الإعتبار مقدار الخطورة...

Code Review Reports

عند إدخال نظام المراجعة الأمنية للمشروع، فإن هناك صيغة يتم إنشائها على شكل تقارير لكل Module على حدا، هذا التقرير يحتوي على تاريخ التقرير، إسم المشروع، ال Module أو الجزئية التي تم فحصها، المبرمج الذي قام بتصميم هذا ال Module، وال reviewer الذي قام بمراجعة هذا العمل، ال task المطلوبة، وصف مختصر عن المشكلة مع إعطاء الأولوية المناسبة لذلك، وربط هذا التقرير بال Ticket، وعادة ما يتم هذا بشكل تلقائي من خلال بعض ال tools مثل FxCop, BinScope Binary Analyzer

When to Code Review؟

متى انسب وقت لعمل ال code review؟

الشركات تنقسم عادة لثلاثة أقسام

1. تقوم هذا الشركات بمراجعة الشيفرة البرمجية قبل رفعها على ال branch الأساسية -pre-commit-، هذه الطريقة -شخصيا- تعد أفضل الطرق لمراجعة أي شيفرة برمجية، فهي تضمن حل المشاكل قبل صدورها أو التقليل منها، كما تحافظ على الشيفرة البرمجية نظيفة، لكن مشكلتها الأساسية هي الوقت الذي تحتاجه...
2. تقوم الشركات هنا بسحب التعديلات بعد رفعها -post-commit-، وأهم مميزات هذا الأسلوب السرعة في رفع الأعمال، وتتم المراجعة هنا بعد عملية الرفع، وعند وجود أي مشكلة يتم الرجوع فيها للمطور لتعديلها مباشرة، لكن أسوأ ما فيها أن تجعل الشيفرة البرمجية سيئة، كما تحتمل وجود أخطاء أكثر خصوصا بأجزاء التوثيق، بالإضافة إلى إمكانية وقوع أي تعديل من قبل أي مبرمج آخر أثناء هذه الفترة في نفس المكان...

When to Code Review؟

3. الأسلوب الثالث هو وضع المراجعة في وقت محدد أو عدد مرات معين سنويا، أو عند وجود مشكلة أمنية فحينها يتم مراجعة الشيفرة ككل، هذا الأسلوب لا يستخدم عادة إلا للمرور على جميع أجزاء الشيفرة البرمجية ومراجعتها لأنماط معينة فقط، لكن إن اعتمدت الشركة فقط على هذا الأسلوب، فسيكون هذا أكبر خطأ وقعت به -وجهة نظر شخصية-

اللَّهُمَّ لكَ الحَمْدُ أنْتَ نُورُ السَّمَوَاتِ والأرْضِ، ولَكَ الحَمْدُ أنْتَ قَيِّمُ السَّمَوَاتِ والأرْضِ، ولَكَ الحَمْدُ أنْتَ رَبُّ السَّمَوَاتِ والأرْضِ ومَن فِيهِنَّ، أنْتَ الحَقُّ، ووَعْدُكَ الحَقُّ، وقَوْلُكَ الحَقُّ، ولِقَاؤُكَ الحَقُّ، والجَنَّةُ حَقٌّ، والنَّارُ حَقٌّ، والنَّبِيُّونَ حَقٌّ، والسَّاعَةُ حَقٌّ، اللَّهُمَّ لكَ أسْلَمْتُ، وبِكَ آمَنْتُ، وعَلَيْكَ تَوَكَّلْتُ، وإلَيْكَ أنَبْتُ، وبِكَ خَاصَمْتُ، وإلَيْكَ حَاكَمْتُ، فَاغْفِرْ لي ما قَدَّمْتُ وما أخَّرْتُ، وما أسْرَرْتُ وما أعْلَنْتُ، أنْتَ إلَهِي لا إلَهَ إلَّا أنْتَ.

:reviewer should develop familiarity with the following aspects

على كل مطور يرغب بمراجعة الأعمال البرمجية والإهتمام بالجوانب الأمنية أن يقوم بتطوير مهاراته في عدد من المواضيع المهمة وأن يهتم بها، وهي:

- Application features and Business Rules: يجب على كل مطور أن يفهم المزايا الخاصة المشروع، وكل القواعد والمحددات الخاصة بالمشروع، والتقنيات، والمحددات الخاصة بال Business، وسبب أهمية هذه النقطة تكمن في تحديد التأثير الخاص بالمراجعة وتأثيرها على نجح مزية معينة أو فشلها والتأكد من هذه الخاصية ستعمل على النحو الصحيح والمطلوب بناءا على ال Business
- Context: يجب أن يتم كتابة جميع المعايير التي تتم من خلالها المراجعة، ويجب أن يتم توثيق جميع المعايير الأمنية للشيفرة البرمجية كذلك، ويشمل ذلك جميع أنواع البيانات التي يتم التعامل معها ومعالجتها، ومقدار الضرر المتوقع من هذه البيانات -بالأخذ بعين الاعتبار مكان معالجة هذه البيانات وطريقة الحصول عليها وطريقة إرجاعها- .

:reviewer should develop familiarity with the following aspects

- Sensitive Data: وتشمل هذه الجزئية الإهتمام بجميع البيانات الحساسة في النظام، والتي قد تأثر بشكل خطير على خصوصية الأفراد المنتسبين للمنتج، أو على إمكانية وصولهم للمنتج بالشكل الصحيح، أو معالجة معلوماتهم بنائا على ما تم فقده، لذلك تكون معرفة هذه البيانات بشكل جيد، والإهتمام بها والتأكد من حفظها بالشكل الصحيح بالطريقة الصحيحة من أهم النقاط عند مراجعة أي شيفرة برمجية، ومن الأمثلة على ذلك كلمة المرور لأي عضو...

- User roles and access rights: هذه النقطة مهمة جدا في أي نظام، وعادة ما يتم تحديد جميع الصلاحيات للأعضاء وإمكانية الوصول للمزايا الموجودة داخل المنتج مسبقا، لذلك من المهم جدا أن يعرف المراجع نظام الصلاحيات المتبع في العمل، ويمكن تصنيف هذه الجزئية إلى قسمين، منتج يمكن الوصول إليه من خلال الإنترنت، وهذا يجب أن يكون الإهتمام فيه في أعلى الدرجات، ومستوى يمكن الوصول إليه فقط من خلال الموظفين في المؤسسة والتي يمكن أن تخضع لمعايير أقل صرامة وأكثر وضوحا...

:reviewer should develop familiarity with the following aspects

- Application type: يجب على المراجع أن يهتم بنوع التطبيق الذي يعمل عليه، فإن المشاكل الأمنية التي تظهر على المواقع الإلكترونية قد تختلف عن المشاكل الأمنية التي تظهر على تطبيقات ال desktop أو الهواتف المحمولة، لذلك العلم بطبيعة ونوع التطبيق مهمة للمراجع الذي سيهتم بالجانب الأمني خصوصا...
- Code: معرفة اللغة التي تم كتابة الشيفرة البرمجية فيها أمر مهم جدا جدا، لأن لكل لغة مزاياها وخبايها التي قد تخفى على من هو خارج هذه اللغة، لذلك يجب على المراجع أن يكون ملماً في هذه اللغة حتى يتم تحسين الشيفرة البرمجية ليكون بأفضل شكل له، مع أفضل حماية ممكنة بسبب العلم بأماكن الضعف الممكنة...

:reviewer should develop familiarity with the following aspects

- Design: وهذه النقطة مهمة جدا، فطريقة كتابة الشيفرة البرمجية تختلف من Design Pattern لآخر، ومن Code Layout إلى آخر، فمثلا من يستخدم ال MVC يختلف عمن قام بكتابة Pattern خاص به، في أماكن ال Configuration وحفظها وحمايتها ستختلف من شكل إلى آخر، وكذلك ال design تشمل شكل ال Url وتسلسل هذه الروابط، كما تشمل طريقة ال rendering وشكله لأي user...الخ
- Guideline: في حال وجود أي مرجعية للشركة لكتابة الشيفرة البرمجية فيجب مشاركتها بين الأقسام، ويجب أن تتم مشاركتها بين أعضاء فريق التطوير، ويجب على الجميع فهم هذه المعايير...

Questions During Secure Code Review

Table 4: Example Design Questions During Secure Code Review

Design Area	Questions to consider
Data Flow	• Are user inputs used to directly reference business logic? • Is there potential for data binding flaws? • Is the execution flow correct in failure cases?
Authentication and access control	• Does the design implement access control for all resources? • Are sessions handled correctly? • What functionality can be accessed without authentication?
Existing security controls	• Are there any known weaknesses in third-part security controls • Is the placements of security controls correct?
Architecture	• Are connections to external servers secure? • Are inputs from external sources validated?
Configuration files and data stores	• Is there any sensitive data in configuration files? • Who has access to configuration or data files?

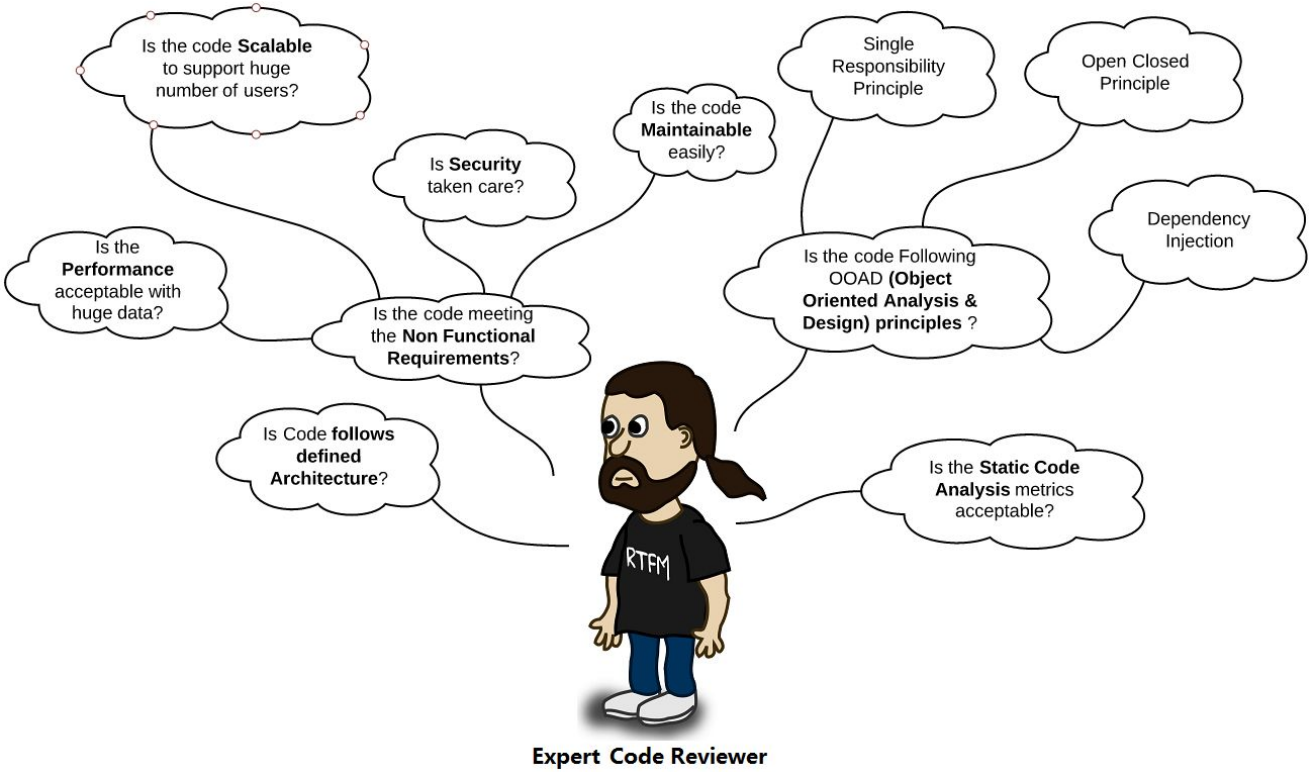
Code Review Checklist

هناك مجموعة من النماذج المتنوعة التي صممت لتحديد المبادئ الخاصة بأي مراجعة لأي شيفرة برمجية، هذه المبادئ إن تم العمل بها ستجعل المراجع يرتقي لدرجة المحترفين والخبراء، بل إنه يعد من الخبراء في حال عمله بهذه المبادئ بشكلها الصحيح، هذه القائمة يجب أن تحتوي على مجموعة من النقاط الأساسية أو العناوين العريضة التي تهتم فيها وتهدف لمراقبتها ومعالجة المشاكل المتوقعة منها، وهي:

- Security & Architecture: Data Validation, Authentication, Session Management, Authorization, Cryptography, Error Handling, Logging, Security Configuration, Network Architecture.
- Application Design: Font uniformity, Color Accessibility, Uniformity of color and design scheme, Ease of use for users, Minimum number of screens to achieve a use-case, Performance, Presentation of information, Responsiveness, Accessible content, Accuracy of information

طبعا تتعدد النماذج وأهدافها، وهناك نماذج تفصيلية ونماذج عامة، ويتم اختيار النموذج أو بنائه بناءا على احتياجات الشركة...، في النقاط التي في الأعلى قد تكون في بعض النماذج هي العناوين الأساسية، وفي بعضها الآخر جزئية من مجموعة نقاط مرتبة تحت عنوان اكبر، أو تتوزع بشكل مختلف...

Code Review Checklist



Code Review Checklist

- Code formatting: أثناء تصفح الشيفرة البرمجية، تحقق من أن تنسيق الشيفرة البرمجية!، وذلك لتحسين قابلية القراءة والتأكد من عدم وجود أي إزعاج عند النظر إليه أو قرائته، ويشمل هذا الكلام ال tabbing, spacing, naming conventions, line length, remove any commented code... الخ
- Architecture: يجب أن تكون معمارية المشروع منظمة ومبنية بشكل واضح، وهذا يشمل تقسيم المشروع إلى عدة طبقات مثل ال data layer, presentation، وتقسيم الملفات بشكل منظم مثلا html, css, js في مسارات محددة وواضحة، وعلى شكل ملفات مستقلة، بالإضافة إلى التأكد من أن الشيفرة البرمجية المكتوبة تتبع وتسير بشكل صحيح مع ال framework المستخدم، بالإضافة إلى اعتماد واستخدام ال design pattern المناسب قبل البدء ببناء المشروع برمجيا...

Code Review Checklist

- Coding best practices: أثناء كتابة الشيفرة البرمجية يجب الإلتزام بأهم المعايير لكتابة الشيفرة البرمجة كما نلتزم بتنسيقها -كما ذكرنا في أول نقطة-، هذه المعايير تشمل Don't write hard coded، بتقدر دايما تستخدم أو تعرف const أو تضيف configuration variable، وعليك بكتابة ال comments مع الأخذ بعين الإعتبار أن أهمية ال comments تكمن في شرح وتوضيح لماذا تم اضافة أو كتابة هذا ال block of code بهذا الشكل، وإن كان هناك مشاكل محتملة يجب النظر إليها قبل التعديل، والكتابة عند المتغيرات وظيفة هذا المتغير أو ما يحتويه، بالإضافة بناء مجموعات من ال enums الصغيرة داخل الشيفرة البرمجة مثل {Gender: {Male, Female...، بالإضافة إلى تجنب ال nested loop, nested if لأكثر من 3 levels، والتفكير بطرق أخرى لحل مثل هذه المشاكل في حال ظهورها، وتحقيق أكبر استفادة ممكنة من خلال ال framework او المكاتب المستخدمة قبل كتابة أي custom code...

Code Review Checklist

- Non Functional requirements: وتشمل جميع المفاهيم والقواعد المهمة لأي مبرمج ويمكن اختصارها فيما يلي:
  - Maintainability (Supportability): يجب أن يكون المشروع المراد بنائه قابل للصيانة والتحديث بأقل جهد ممكن، وهذا يعني الإهتمام في 4 مواضيع رئيسية وهي: Configurability، Debuggability، Testability، Readability، فالشيفرة البرمجية يجب أن تكون واضحة تفسر نفسها بنفسها، فإن لم يكن ذلك سهلا فيجب أن يتم ذلك من خلال ال comments، وهذه الشيفرة يجب أن تكون سهلة عند القيام بأي عملية Test، وهذا يتم من خلال تقسيم الشيفرة البرمجية small blocks، تحتوي الوظائف المراد تنفيذها في كل block، والتحقق يتم ضمن هذا ال block، بالإضافة إلى إمكانية تتبع المشاكل الممكنة من خلال عمل log واضح للعمليات التي حدثت، أو التي تحديث أثناء عملية الفحص، وأن يكون ال configuration في مكان يتم التعامل معه بشكل dynamically وأن لا يكتب على شكل hard code...
  - Reusability: يجب أن تكون الشيفرة البرمجية التي تكتبها قابلة للإستخدام في أكثر من مكان، وأن تكون الشيفرة البرمجية لل classes & function مصممة بشكل يمكن أن يخدم أكثر من مكان بسهولة، ومن أهم المبادئ هنا هو DRY، فلا تقم أبدا بكتابة شيفرة برمجية متطابقة مكررة في أكثر من مكان.

Code Review Checklist

- Reliability: وهي تشمل معالجة الأخطاء غير المتوقعة، بالإضافة إلى تنظيف وحذف أي من المصادر غير المستخدمة في المشروع.
- Extensibility: وهذه تعني أن الشيفرة البرمجية يجب أن تكون قابلة للتحديث والاستبدال في شيفرة برمجية أخرى بسهولة.
- Security: وهي تشمل كل Authentication, authorization, input data validation والتهديدات الأمنية المتوقعة من وراء ذلك كال XSS, SQL injection...ألخ، كما تشتمل على الاهتمام بالحفاظ على البيانات الحساسة وحمايتها ككلمات المرور، ومعلومات البطاقات الائتمانية وغيرها...
- Performance: وهنا تشمل كل الوسائل الممكنة للحفاظ على أداء عالي وممتاز للنظام، وأهم النقاط العريضة لهذا الموضوع هو استخدام ال Lazy loading, asynchronous and parallel processing والامتناع عن استخدام ال synchronous إلا للضرورة، بالإضافة إلى استخدام ال Caching وال session data.
- Scalability: يجب أن يتم التفكير مسبقا وأنثاء تنفيذ المشروع، هل ما أقوم فيه وما أبنيه قادر على تحمل الزيادة المفاجئة أو المتوقعة في حال زيادة الأعضاء المسجلين مثلا؟، وهل النظام يسمح بتطوير هذه الجزئية لتتناسب مع الزيادات في وقت لاحق؟
- Usability: ويجب التأكد أن أي API تقوم ببنائها وأي وظيفة تظهر في النظام يجب أن تكون قابلة للإستخدام بشكل سهل ومفهوم، فإن لم تكن مقتنعا بالتصميم فعليك مراجعة ال product ومناقشة الأفكار الخاصة بك معهم...

Code Review Checklist

- Object-Oriented Analysis and Design (OOAD) Principles:
  - Single Responsibility Principle (SRS): قم بوضع مبرمج أو مطور واحد على ال function أو ال class، وليكن نظام التوزيع للعمل على أساس OOP مقسما على شكل قطع من ال classes وال functions، وكل class أو function يوجد مصدر واحد فقط يعمل عليه لبنائه، وطبعا هذا لا يتعارض مع دمج الآخرين في تطوير هذا ال class او ال function، لكن أثناء البناء يختلف الكلام، ويفضل أن تكون التوزيع والاعتمادية بين المصادر المختلفة واضحة قبل شروع كل من المطورين بتطوير أجزائه، لمعرفة نقاط التقاطع فيما بينهم.
  - Open Closed Principle: عند القيام بتنفيذ وتطوير خاصية جديدة -New functionality- فيجب التوقف عن تعديل أي شيفرة برمجية مرتبطة بهذه ال functionality لمنع حدوث أي مشاكل غير متوقعة أو أي تحديث غير متناسق فيتسبب ذلك بحدوث مشاكل لا حصر لها...
  - Liskov substitutability principle: في حال وجود super class وال sub class، فإن ال sub class يجب أن يكتب بطريقة تمكنه من إستبدال ال object الخاص بال super class بال sub class دون حصول أي مشكلة، لهذا لا يمكن مثلا وضع محددات أو حذف param من method انعمل عليها override دون أن يسمح ال super class بذلك!، فهنا يجب أن تكون ال override method هي نسخة معدلة لل sub class لكن بنفس القواعد...

Code Review Checklist

- Interface segregation: وهذه تقوم على مبدأ فصل ال interface وتقليل حجمها قد المستطاع ليشمل كل interface الوظائف التي بني لأجلها فقط، بالإضافة إلى الإمتناع عن إضافة أي params غير ضرورية -required- داخل ال interface.
- Dependency Inversion principle: بشكل عام يمكن القول بأنها الطريقة أو المبدأ التي تمنع من كتابة أي dependency داخل جزئية وسيتم استخدام هذه ال dependency في أكثر من مكان، ومن الأمثلة على ذلك ما قمنا بتصميمه لعمل fetch للبيانات من ال API عن طريق ال axios.
  ملاحظة: في لغات البرمجة التي تحتوي في ثناياها ال abstractions فإن High level لا يجب أن يعتمد على ال Low level، بل ال low level & high level لازم يعتمدو على ال abstractions

Code Review Checklist

- Application Design:
  - Font uniformity: يجب أن تكون الخطوط المستخدمة داخل التطبيق متناسقة الحجم، وذات حجم واحد في جميع الأجزاء والأقسام المتشابهة، وكذلك الألوان، ويجب أن تكون أحجام الخطوط قابلة للقرائة على جميع الأجهزة...
  - Color Accessibility: الألوان المستخدمة في النظام يجب أن تكون مستخدمة ويمكن الوصول إليها من جميع فئات المستخدمين في الموقع
  - Uniformity of color and design scheme: يجب أن تكون الألوان والتصميم متناسقين معا، ويتسمان بالوضوح.
  - Ease of use for users: يجب أن يكون الوصول أو معرفة طريقة الوصول سهلة وسلسلة للمستخدمين.
  - Minimum number of screens to achieve a use-case: يحب أن تكون الوصول للمطلوب من قبل المستخدمين بأقل عدد من الواجهات والتنقلات
  - Performance: وتشمل ال caching وأقل عدد ممكن ال requests على السيرفر، واسترجاع البيانات المطلوبة فقط للعرض.

Code Review Checklist

- Presentation of information: المعلومات التي يتم عرضها يجب أن تكون متاحة بأفضل format أو طريقة متاحة، ويجب أن يعرض ال layout هذه المعلومات بطريقة واضحة، ويجب أن يتم الوصول لهذه المعلومات من خلال جميع أنواع الأجهزة والهواتف والمواقع وغيرها بنفس الطريقة...
- Responsiveness: يجب أن يدعم تطبيق الويب جميع الواجهات الأساسية لأحجام الشاشات المختلفة.
- Accessible content: يجب أن تهتم بأن المحتوى الذي تقوم بتصميمه سيظهر وستعمل على جميع الأجهزة او المتصفحات حتى لو اختلفت الشركات، كما يجب الأخذ بعين الإعتبار بعض الخصائص المهمة كال touch screen في حال وجود mobile، والامتناع عن أي redirects لا تلزم.
- Accuracy of information: المعلومات التي يقدمها ال API يجب أن تكون كفيلة بأن يصل المستخدم لما يحتاج دون أن يتوقف وبأقل قدر ممكن من أي قطع لتسلسل ال flow الذي يقوم فيه.

قال تعالى في سور الزمر:

"أَمَّنْ هُوَ قَانِتٌ آنَاءَ اللَّيْلِ سَاجِدًا وَقَائِمًا يَحْذَرُ الْآخِرَةَ وَيَرْجُو رَحْمَةَ رَبِّهِۗ قُلْ هَلْ يَسْتَوِي الَّذِينَ يَعْلَمُونَ وَالَّذِينَ لَا يَعْلَمُونَۗ إِنَّمَا يَتَذَكَّرُ أُولُو الْأَلْبَابِ"

Application Threat Modeling

تعد ال Application Threat Modeling إحدى الطرق التي تدخل في صميم تحليل النظام الأمني في أي تطبيق، ومع أن هذه الجزئية ليست من صميم هذه الشرائح، إلا أنها تقدم نموذج للفتة مهمة للمطورين عند النظر في الشيفرة البرمجية، وسأذكر هنا الأجزاء الأمنية التي يهتم بها هذا ال modeling لكن بصيغة أو تعداد قد يكون أقرب لموضوع هذه الشرائح، و سأعتمد نفس الخطوات المعتمدة في الكتاب لسرد التسلسل، وهي:

الخطوة الأولى:

Decompose the Application: هذه الخطوة أول وأهم خطوة والتي تفيد بوجوب الاهتمام بفهم التطبيق وسير عمله وكيفية تعامل التطبيق مع **أي جهة أو طرف ثالث**

وهذه هي أهم النقاط التي يجب الإهتمام بها:

Application Threat Modeling

- External Dependencies: عند إضافة أي عنصر ك dependencies خارجية يجب التأكد من إضافتها للتوثيق بشكل يشرح التأثير طريقة الإستخدام والتأثير على ال production env تحديدا، فمثلا إذا اعتمدنا على AWS فيجب أن يتم توثيق آلية بناء وتطبيق المراد، بالإضافة إلى ال production environment، وهذا التوثيق بجب أن يتم بشكل منظم بناءا على طريقة التوثيق المعتمدة بالمؤسسة وعدم الإكتفاء بتوثيقها على الأنظمة التي لا تتعلق بها مثل الإكتفاء بكتابة التعليق على task ticket، مع الفائدة حين كتابتها على التكت قبل إغلاقها أو تحريكها...
- Entry Points: هذه النقطة تشمل سرد جميع المنافذ التي يمكن من خلالها المهاجم أو المخترق أو المخرب من استغلالها للوصول إلى هدفه، وتسمى أيضا ب attack vector، وتشمل هذه الجزئية في المواقع الإلكترونية مثلا form pages، وتشمل مثلا ال emails، وتشمل ال chat، ال popup window، ال socket messages...، لذلك يجب أن تكون المراجعة في هذه الجزئيات دقيقة جدا، والتعديل في جزئياتها يتطلب مراجعة شاملة لهذا المنفذ.

Application Threat Modeling

- Assets: إن أي مخترق عادة ما يقوم بذلك لهدف من إثنين، إما رغبة في ال Physical Assets، وإما abstract asset، في الأولى فهو مهتم في ما قد يحصل عليه من معلومات وبيانات قد تقدم له عونا في عمليات اختراق أخرى، منفعة مادية...الخ، والثانية تكون للإضرار بسمعة الشركة المراد اختراقها، لذلك، إن استطعت تحديد أهم البيانات التي لديك والتي يجب حمايتها قللت من مقدار الخطورة أو تنبأت بوجودها، لهذا فإن أي عملية مراجعة يجب أن تأخذ بعين الإعتبار ما ترغب بحمايته وما لا ترغب بخسارته...
- Determining the Attack Surface: يعد تحديد طرق الهجوم وتوقعها من أكثر الطرق التي يستخدمها المخترقون للدخول إلى أي نظام، وكذلك من هو مسؤول عن حماية هذا النظام، وأول جزئية هي حصر جميع ال input paths الممكنة والتحقق منها ومراجعة أي تعديل يطرأ عليها أثناء العمل، ويشمل هذا Browser input, Cookies, Property files, External processes, Data feeds, Service responses, Flat files, Command line parameters, Environment variables

Application Threat Modeling

- Trust Levels: تحديد مقدار الصلاحية اللازمة عند العمل على أي جزئية أمر مهم في التعامل مع أي شيفرة برمجية قابلة للتنفيذ خصوصا من ال external resource، وهذا يعني التأكيد على هذا المبدأ أثناء مراجعة الشيفرة البرمجية، فلا داعي لإعطاء صلاحية أكبر من المطلوب، وعند تغيير القواعد إما زيادة أو نقصان فيجب التعديل على الصلاحية أيضا بما يتناسب مع التعديل الجديد...

- Data flow analysis: معرفة ال dynamic data & static data وما هي آلية معالجتها وكيفية التعامل مع ال params الخاصة بها ومعالجة القيم التي تحملها...

Application Threat Modeling

- Transaction analysis: هذه النقطة من النقاط المهمة جدا جدا، فيجب على المراجع أن يعرف عن جميع ال Transaction الحاصلة أمامه على الشيفرة البرمجية التي يتم التعامل معها، وتكمن أهمية هذا الموضوع بأن النظام هو قائم على مجموعة من الحركات "من وإلى"، وهذه الحركات يمكن جمعها ب:

Data/Input Validation of data from all untrusted sources, Authentication, Session Management, Authorization, Cryptography (data at rest and in transit), Error Handling /Information Leakage, Logging /Auditing

Application Threat Modeling

الخطوة الثانية: Determine and rank threats: في هذه الخطوة يبرز الجانب الأمني من المراجعة، فمن خلالها يتم تحديد التهديدات المتوقعة وترتيبها من حيث الخطورة، في هذه المرحلة يتم بناء تصنيفات لهذه التهديدات المحتملة، وهناك عدة نماذج لذلك نذكر منها ال STRIDE.

ال STRIDE هي اختصار ل Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service (DoS), and Elevation of privilege، وهي إحدى ال models المستخدمة لبناء تصنيف للتهديدات الأمنية بناءا على الاختصارات التي تم بناء الإسم من خلالها، والآن سنقوم بتوضيح بسيط لهذه المصطلحات:

Application Threat Modeling

- Spoofing: ويقصد بها ال "Identity spoofing"، وهي واحدة من أهم المخاطر التي يمكن أن تحدث في أي تطبيق، انتحال الهوية هذا قد يكون على مستوى قواعد البيانات أو على مستوى التطبيق، وهذا أمر لا يجب أن يكون في النظام وهو خطر جدا، ودور المراجع هنا من التحقق من أن الشيفرة البرمجية فيها ما يمنع مثلا من أن يقوم مستخدم x بتعديل معلومات مستخدم y -بكل تأكيد إذا كانت موجودة فهذا يعني أنها مسموحة ويجب أن تتم من خلال role واضحة تسمح بذلك-.
- Tampering: العبث بالبيانات واحدة من أشهر الطرق للتلاعب بالبيانات واكتشاف الثغرات، هذا التلاعب يحصل على جانب ال client side مستغلا البيانات القادمة من السيرفر ك response أو من خلال سرقتها من ال memory أو ال network..الخ، ثم يقوم هذا المهاجم بتعديلها لتحقق له هدفه!، ودور المراجع هنا من التحقق من أن ال server قادر على التحقق من أي عبث موجود قبل عمل أي معالجة أو حفظها داخل قواعد البيانات، وأهم شرط للتحقق هنا التأكد من هذه ال request قادم من auth user.

Application Threat Modeling

- Repudiation: عادة ما يقوم المخترقين بإخفاء وجودهم حتى لا يتم كشفهم، وإحدى هذه الطرق التحايل على نظام ال log/track من خلال منع التتبع الصحيح لهذا اليوزر أو من خلال تعديل ال log.
- Information Disclosure: سرقة أو كشف المعلومات الحساسة والمهمة للأعضاء المسجلين يعد من أكبر المخاطر التي قد تتحقق بشركة ما، لذلك هناك دور مهم جدا للمراجع هنا في التأكد بأن الشيفرة البرمجية تضمن حماية هذه البيانات من خلال التأكد من خلوها من الثغرات المتوقعة كال SQL Injection، وذلك يشمل أيضا التحقق من صلاحيات قواعد البيانات أنها وضعت بشكل صحيح، وال query...الخ

Application Threat Modeling

- Denial of Service (DoS): مراجع الشيفرة البرمجية ومصمم الشيفرة البرمجية يجب أن يكون حذرا أشد الحذر من هجمات ال Dos، وأهم القواعد لتجنب أو تقليل حدة هذه الهجمات هي منع أي request لأي non-auth user على أي api/function يحتاج لوقت و resource عالية للتنفيذ...

- Elevation of privilege: تقييم نظام الصلاحيات والتحقق بأن الذي بالأسفل لا يمكن أن يقوم بتعديل على من هو أعلى منه صلاحية، ويجب تقسيم الصلاحيات بحذر وضمن تصنيفات واضحة.

Application Threat Modeling

- Microsoft DREAD threat-risk ranking model:

Table 9: Explanation Of The Dread Attributes

DREAD	Questions
Damage	How big would the damage be if the attack succeeded? Can an attacker completely take over and manipulate the system? Can an attacker crash the system?
Reproducibility	How easy is it to reproduce an attack to work? Can the exploit be automated?
Exploitability	How much time, effort, and expertise is needed to exploit the threat? Does the attacker need to be authenticated?
Affected Users	If a threat were exploited, what percentage of users would be affected? Can an attacker gain administrative access to the system?
Discoverability	How easy is it for an attacker to discover this threat?

يستخدم هذا النموذج
لتحديد أو حساب
معامل التأثير
والخطورة بناءا على
سهولة استغلال
الثغرات أو اكتشافها
وكمية الضرر
المترتبة عليها...

Application Threat Modeling

الخطوة الثالثة: Determine countermeasures and mitigation

يعد تحديد المخاطر الأمنية أو التقنية المتوقعة من خلال اتخاذ التدابير اللازمة لمنع حدوث مثل هذه المشاكل من الأمور المهمة والتي ستؤثر بكل تأكيد على المشروع، ويمكن التعامل مع نقاط الضعف هذه من خلال 3 طرق أساسية:

1. بناء خريطة تحدد نقاط الضعف والقوة وترتيبها من الأخطر إلى الأقل خطورة.
2. تتعامل بعض المؤسسات على قبول الضعف الكائن في مكان ما باعتبار نسبة الخطورة هذه يمكن تحملها والتعامل معها بناءا على طبيعة المشروع مع وضع ضوابط تعلم المستخدمين بهذا وطرق التعامل معه في حال حدوثه.
3. تتعامل بعض المؤسسات على تجاهل نقاط الضعف تماما وبدون أي احتراز، وفي هذه الحالة إذا كانت الخطورة والضرر أكبر من المنفعة المرجوة من التطبيق فسيتم إغلاق التطبيق

قال تعالى في سورة الفتح:

"إِنَّا أَرْسَلْنَاكَ شَاهِدًا وَمُبَشِّرًا وَنَذِيرًا (8) لِّتُؤْمِنُوا بِاللَّهِ وَرَسُولِهِ وَتُعَزِّرُوهُ وَتُوَقِّرُوهُ وَتُسَبِّحُوهُ بُكْرَةً وَأَصِيلًا"

فلتعلم يا أخي أن نصرة الرسول صلى الله عليه وسلم واجبة على كل مسلم وفيما يقدر عليه, وأبسط ما نقدر عليه الآن أمام نذالة فرنسا وغطرستها وحربها على الإسلام والمسلمين هو مقاطعتها بكل السبل، والعمل للخروج من عبودية هذا القرن بكل ما أوتينا من قوة, ولا حول ولا قوة إلا بالله...

قال تعالى في سورة التوبة:

" إِلَّا تَنْصُرُوهُ فَقَدْ نَصَرَهُ

اللَّهُ إِذْ أَخْرَجَهُ الَّذِينَ كَفَرُوا ثَانِيَ اثْنَيْنِ إِذْ هُمَا فِي الْغَارِ"

Metrics and Code Review

بناءا على ما ذكرناه سابقا فيجب أن تكون هناك طريقة واضحة يمكن من خلالها حساب معدل التعقيد في الشيفرة البرمجية ومستوى الجودة والقدرة على تعديل هذه الشيفرة البرمجية ونقلها وإعادة إستخدامها والعديد من الخصائص الأخرى، لذلك سنذكر هنا أهم الخصائص التي يمكن الانطلاق منها عند مراجعة الشيفرة البرمجية وهي:

- LOC: عدد الأسطر البرمجية (الأسطر الفارغة وأسطر ال comments لاتحسب)
- Function Point: وهي مجموعة الأسطر البرمجية التي تقوم بتنفيذ مهمة محددة في المشروع، وتختلف هذه بناءا على اختلاف لغة البرمجة، فمثلا بال OOP يمثل ال Class ال Function Point.

Metrics and Code Review

- Defect Density: يشير هذا المصطلح إلى عدد العيوب المؤكدة التي تم اكتشافها في البرنامج أو أحد المكونات خلال فترة محددة من التطوير أو التشغيل، مقسومًا على حجم البرنامج (LOC)، ولحساب مستوى كثافة العيوب(الخلل والمشاكل) في التطبيق يمكن تنفيذ هذه المعادلة (سنذكر مثالا بعد هذه الشريحة):

```
Defect Density = Defect count/size of the release
```

- Risk Density: وهو مصطلح يشير إلى مقدار كثافة المخاطر المتحققة ضمن النقاط السابقة وتقسم إلى Low, Medium and High ويمكن تمثيلها بالآتي:

```
Risk Level / LOC OR Risk Level / Function Point
```

```
Example:
4 High Risk Defects per 1000 (Lines of Code)
2 Medium Risk Defects per 3 Function Points
```

Metrics and Code Review

مثال:

Module	LOC	Number of Detected Bugs
A	1000	5
B	3000	25
C	2000	10
Totals	6000	40

بناءا على الجدول السابق يمكننا حساب كثافة الأخطاء لهذه ال release بالآتي:

Defect Density = 40 / 6000= 0.00666666667=> هذه مستوى الكثافة لل release

وهذا يعني أن كثافة الأخطاء بالنسبة لمجموع الأسطر لهذه ال release 0.00666 وهذا رقم ممتاز، وهذا فقط مؤشر يعطي انطباعا عاما عن جودة الشيفرة البرمجية.

ملاحظة: لا يوجد رقم محدد لتحديد الأفضل أو الأسوأ، لكن كلما كبر الرقم كلما كان أسوأ، وكلما اقترب الرقم من الصفر كلما كان أفضل.

Metrics and Code Review

كما يمكن حساب ال avg الخاص بمستوى كثافة الأخطاء وذلك من خلال ال KLOC، وال KLOC يعني:

Short for thousands (kilo) of lines of code. KLOC is a measure of the size of a computer program

وبالنسبة للمثال السابق فالنتيجة هي:

KLOC = 40 / 6 = 6.6666 for Every 1KLOC

Metrics and Code Review

Cyclomatic complexity (CC):

تم تصميم مقياس التعقيد السيكلومي (CC) من McCabe للإشارة إلى قابلية البرنامج للاختبار والفهم وقابليته للصيانة، وهو من المقاييس السهلة والبسيطة لحساب مستوى التعقيد، ويمكن حساب مستوى التعقيد السيكلومي من خلال هذه المعادلة:

CC = Number of decisions +1

بحيث ال Number of decisions تمثل if/else, switch, case, catch, while, do, templated class calls...إلى آخره

نتيجة المعادلة السابقة يمكن تصنيفها بناءا على هذا الجدول:

Value Range	Description
0-10	Stable code, acceptable complexity
11-15	Medium Risk, more complex
16-20	High Risk code, too many decisions for a unit of code.

Metrics and Code Review

بناءا على مستوى التعقيد السابق يتم اتخاذ القرار بفصل أجزاء الشيفرة البرمجية وإعادة كتابتها بشكل أفضل وفصلها إلى method مستقلة للتقليل من مستوى التعقيد!

مثال 1 لحساب العملية السابقة:

```
IF A = 10 THEN
 IF B > C THEN
   A = B
 ELSE
   A = C
 ENDIF
ENDIF
Print A
Print B
 Print C
```

في هذا المثال وبناءا على المعادلة السابقة يكون الناتج هو:

CC = 2 + 1 = 3

ال 3 أقل من 10، إذا مستوى التعقيد جيد و الشيفرة البرمجية غير معقدة.

Metrics and Code Review

```
Function doSomething ()
{
    if (condition1){
     // statements
    } else if (condition2){
     // statements
    } else {
     // statements
    }
}
```

مثال 2:

CC = 2 + 1 = 3

ال 3 أقل من 10، إذا مستوى التعقيد جيد
و الشيفرة البرمجية غير معقدة.

ملاحظة: كل else if تعتبر شرط مستقل يزيد من مستوى التعقيد لذلك

if/else = 1 + else if = 1 then = 2

ملاحظة: قد تجد أكثر من تمثيل لحساب التعقيد مثال هذا التمثيل CC = E - N + 2 لكننا اعتمدنا ما ذكرناه...ولك الحرية في البحث عن أي method مناسبة لك...

Metrics and Code Review

بناءا على الجدول الخاص بمستوى التعقيد، فإن احتمالية حدوث خطأ عند تصحيح أحد الأخطاء تكون كالآتي:

Value Range	Ratio
0-10	5%
20-30	20%
> 50	40%
Approaching 100	60%

نقاط غير متوقعة يجب الإنتباه لها أثناء المراجعة

لقد ذكرنا العديد من النقاط المهمة التي تشكل فارقا مهما أثناء العمل على الشيفرة البرمجية وإن لم يكن تأثيرها مباشرا على الشيفرة البرمجية بذاتها! ومن الأمثلة على ذلك أن الشيفرة البرمجية يمكن رفضها لوظيفة معينة وإن كانت تعمل إن خالفت سياسة ال Terms & condition أو ال Privacy & Policy الخاصة بالشركة!
كما يجب الإنتباه للغات التي يمكن أن يتم إستخدامها أثناء التعامل مع النظام، وأسماء الملفات التي يمكن أن ترفع من خلال التطبيق واللغة المستخدمة لتسمية هذه الملفات!

كما يجب الإنتباه للغات البرمجة أو المكاتب المستخدمة في التطبيق والتأكد من أن هذه المزية لا تقبع في إحدى نقاط الضعف الخاصة بلغة البرمجة! وإن كانت كذلك ونحتاج إلى استخدمها كيف يمكن سد هذا الضعف والوصول إلى أكبر قدر من الموثوقية لها...فإذا رأيت من خلال الشيفرة البرمجية أي نتيجة غير متوقعة ويمكن أن تخالف أمرا متوقعا فقم بالتوقف قليلا قبل الموافقة أو الرفض...

ال Injections

تسمح هجمات ال injection لمستخدم ما من إضافة أو حقن محتوى بمجموعة من الأوامر لتعديل سلوك التطبيق الأصلي. هذا النوع من الهجمات شائع وواسع الانتشار، ويسهل على أي Attacker من اختبار إذا ما كان أي تطبيق ويب ضعيف ويسهل للمهاجم للاستفادة منه أم لا، ولتعلم أن كمية التطبيقات التي لم يتم تحديثها وتحتوي في ثناياها مثل هذه الثغرات كثيرة...، هناك العديد من أنواع ال injections واحدة من أشهر هذه الإختراقات ال SQL injections

أنواع ال injection:

SQL, LDAP, Xpath, OS commands, XML parsers, Code injection, Email header injection...إلى آخره، سنذكر بعضها في الشريحة القادمة والتي تليها...

ال Injections

نوع ال Injection Attack	الوصف	التأثير المحتمل
Code injection	يتم حقن شيفرة برمجية متوافقة مع اللغة التي تم كتابة التطبيق من خلالها للوصول إلى صلاحية معينة تمكنه في اسوأ الظروف على التحكم في السيرفر كامل	على كامل النظام أو التطبيق.. مثلا: `eval("\$_GET[....];");`
Cross-site Scripting (XSS)	يتم حقن شيفرة برمجية من خلال ال client side للوصول إلى معلومات المستخدمين من خلال استخدام تطبيق الويب، وأشهر عمليات الحقن تتم على مستوى ال JS لانتشارها في أكثر المواقع...	أخطر تأثير قد يكون هو انتحال شخصية العميل من خلال سرقة ال token الخاص به مثالا أو استغلال هذا ال injection في القيام بعمليات أو تنفيذ شيفرة برمجية عشوائية مثلا reload إلى ما لا نهاية...
Email Header Injection	يقوم المخترق بحقن ال header الخاص بال email	إرسال spam وجمع معلومات من خلال انتحال شخصية مرسل الإيميل

ال Injections

نوع ال Injection Attack	الوصف	التأثير المحتمل
LDAP Injection	حقن ال LDAP بروتوكول ببعض ال command التي يمكن تنفيذها لتعديل المحتوى في ال tree أو إعطاء صلاحية...	الوصول إلى معلومات الأعضاء واستغلالها أو وتجاوز الصلاحيات على التطبيق من خلال تعديل الصلاحية لنفسه أو تسجيل الدخول بحسابات الأعضاء الآخرين...
OS Command Injection	يتم حقن command لتنفيذها على ال OS، وتختلف هذه عن ال code injection أنها تنفذ عن طريق ال OS وليس عن طريق ال application...(كتنفيذ لل command)	يمكن للمهاجم التحكم في النظام كاملا بأسوء الأحوال...مثال PHP: `system("rm $file");`
SQL Injection	يقوم هنا المهاجم بحقن SQL Statement	التأثير كبير جدا بنائا على مدى احتراف المهاجم وصولا للتحكم بالنظام كاملا

Client Side - Security Guide (Hints)

- لا تستخدم eval نهائيا، فإن استخدمتها فاعلم أن هناك مشكلة معينة في طريقة تصميم المشروع أو تسلسل الإجراءات التي جعلتك تصل لهذه المرحلة...
- استخدم ال `innerText` أو `createTextNode` بدلا من `innerHtml`
- استخدم ال JSON.parse مع ال JSON response دوما قبل اتخاذ أي إجراء
- يجب عمل Html Encode قبل إضافة أي Untrusted Data إلى المحتوى الخاص بأي عنصر مثل ال div, p, h1...الخ، مثال:

& ==> & **<** ==> < **>** ==> > **"** ==> " **'** ==> '

- يجب عمل encode لأي قيمة غير موثوقة سيتم وضعها داخل متغير جافا سكربت أو css أو داخل ال common attribute مثل ال alert, href, width...إلى آخره.
- قم بإضافة HTTPOnly لأي cookie يجب منع الوصول إليها من خلال الجافا سكربت، هذا سيقلل من خطر ال XSS Attack.

Client Side - Security Guide (Hints)

- قم باستخدام ال wss بدلا من ال ws للويب سوكيت وذلك لمنع محاولة الاختراق من Man-In-The-Middle
- يجب التحقق من جميع البيانات التي تصل إلى المتصفح من خلال ال web socket وذلك لإمكانية اختراقها.
- لا تقم بحفظ معلومات حساسة وذات أهمية على ال local storage.
- لا تقم بحفظ معلومات سيتم استخدامها مرة واحدة وستنتهي بعد اغلاق ال window أو ال tap داخل ال local storage!، استخدم ال session storage بدلا من ذلك.
- لا تقم بالتعامل مع ال localstorage على أنها بيانات موثوقة بسبب إمكانية تعديلها، لذلك عند معالجتها قم بمعالجتها بطريقة آمنة.
- Normalization validate وهي التحقق من أن ال encode للنصوص يمنع أي invalid character

Client Side - Security Guide (Hints)

- عند ال file upload يجب التحقق من ال file type and ext، كما يجب التحقق من ال file size
- يجب التحقق من البريد الإلكتروني بطريقة صحيحة، هناك عدة صيغ ممكنة يمكن أن تعطي بريدا إلكترونيا صحيحا مثل email+subaddress@email.com (بعض مزودين الخدمة يسمحون بها مثل جوجل وبعضهم يمنعها مثل مايكروسوفت)، من القواعد التي يمكن استخدامها للتحقق من البريد الإلكتروني هي أن يكون البريد الإلكتروني من مقطعين يفصل بينهم @، ويجب أن لا يحتوي على أي dangers char مثل ال single quote، واسم الدومين في البريد الإلكتروني يجب أن يكون أحرف فقط، ويمكن وجود - أو . أو أرقام، ويجب أن لا يزيد المقطع الأول عن 63 حرف، والمقطع الثاني 254.
- لا تقم بحفظ ال session id داخل ال local storage!

Client Side - Security Guide (Hints)

Data Type	Context	Code Sample	Defense
String	HTML Body	`<span>UNTRUSTED DATA </span>`	HTML Entity Encoding (rule #1).
String	Safe HTML Attributes	`<input type="text" name="fname" value="UNTRUSTED DATA ">`	Aggressive HTML Entity Encoding (rule #2), Only place untrusted data into a whitelist of safe attributes (listed below), Strictly validate unsafe attributes such as background, ID and name.
String	GET Parameter	`<a href="/site/search?value=UNTRUSTED DATA ">clickme</a>`	URL Encoding (rule #5).
String	Untrusted URL in a SRC or HREF attribute	`<a href="UNTRUSTED URL ">clickme</a> <iframe src="UNTRUSTED URL " />`	Canonicalize input, URL Validation, Safe URL verification, Whitelist http and HTTPS URLs only (Avoid the JavaScript Protocol to Open a new Window), Attribute encoder.
String	CSS Value	`html <div style="width: UNTRUSTED DATA ;">Selection</div>`	Strict structural validation (rule #4), CSS Hex encoding, Good design of CSS Features.
String	JavaScript Variable	`<script>var currentValue='UNTRUSTED DATA ';</script> <script>someFunction('UNTRUSTED DATA ');</script>`	Ensure JavaScript variables are quoted, JavaScript Hex Encoding, JavaScript Unicode Encoding, Avoid backslash encoding (`\"` or `\'` or `\\`).
HTML	HTML Body	`<div>UNTRUSTED HTML</div>`	HTML Validation (JSoup, AntiSamy, HTML Sanitizer...).
String	DOM XSS	`<script>document.write("UNTRUSTED INPUT: " + document.location.hash );<script/>`	DOM based XSS Prevention Cheat Sheet

Ref

فلْتعلمْ أن التراجع عند الخطأ نعمة يُغبط عليها صاحبها، ولْتعلمْ أن الإصرار على الخطأ ثم محاولة تبريره ونشره والقول بصحته أعظم من الخطأ ذاته، فإن أخطأت فاستغفر وتب إلى اللّٰه، ولا تصر ولا تكابر ولا تجاهر.

فمن أدرك هذه، أفلح وتعلم وعلا مقامه! وهذه القاعدة تشمل أمور الدين والدنيا!

HTTP Strict Transport Security (HSTS)

إن آلية الوصول إلى المواقع الإلكترونية وطريقة الإتصال بها وطريقة التحكم بال connection الخاصة بال users على مواقع معينة آلية مهمة جدا للحفاظ على أمن البيانات والإتصال بين كل المستخدمين وهذه المواقع، هذا الأمر قاد لإنشاء مصطلح ال HSTS وهو يشير إلى ضرورة اتخاذ إجراءات صارمة أثناء نقل أي معلومات ضمن سياسة واضحة تطبق على متصفحات الإنترنت ويتم إعدادها من خلال ال web server configuration، العديد من هذه السياسات هي الخيار الإفتراضي للمتصفحات ولل web server configuration، وسنتطرق هنا لبعض ال Security Headers التي قد نراها ونتعامل معها دائما والتي تكون معظم الأسئلة التي نراها بسببها "كيف يمكن تعطيل ميزة الأمان هذه بدلا من كيف يمكن العمل بطريقة آمنة معها"!...(سنذكر 3 فقط)

HTTP Strict Transport Security (HSTS)

- Strict-Transport-Security (STS header field): من خلال هذا ال header يتم تعطيل إمكانية الإتصال من دون ال HTTPS، بالإضافة إلى كونه نظاما مهما لتقليل الأخطاء التي يمكن أن تقع من تطبيقات الويب والتي بسببها قد تتسرب بعض المعلومات المهمة للمهاجمين مثل ال cookies ضمن session معين.. مثال:

```
Strict-Transport-Security: max-age=15768000 ; includeSubDomains
```

- X-Frame-Options: يعد هذا ال header واحد من أهم الطرق لتقديم حماية أفضل لتطبيقات الويب من خلال إنشاء policy تضبط الإتصال بين ال host وال web browser لمنع ال Clickjacking، تقوم هذه ال policy بتحديد ما إذا كان هذا ال frame يسمح له بالعرض في صفحات (مواقع أخرى) أم لا...مثال:

```
X-Frame-Options: deny
```

ملاحظة: Clickjacking ويسمى أيضا ب "UI redress attack" وهو أحد التكتيكات المستخدمة من قبل المهاجمين لخداع المستخدمين من خلال إنشاء مجموعة من الطبقات (layers) بدرجات شفافية ووصول مختلفة (transparent) تجعل من المستخدم يقوم بالضغط على شيء لا يريده من خلال التحكم بهذه الطبقات، وعادة ما تكون هذه النقرات وظيفتها تحويلك إلى التطبيق الخاص بالمهاجم أو صفحاته الخاصة، ومن أشهر الأمثلة على ذلك الإعلانات في المواقع التي تخبرك "إضغط هنا لتربح فيزا خضراء" وعند الضغط يقوم بعمل معين...

HTTP Strict Transport Security (HSTS)

- Content-Security-Policy: يعطي هذا ال header صلاحية للآدمن للتحكم في معلومات ال user agent القادمة إليه ووضع القواعد المناسبة للوصول إلى الصفحات المطلوبة من قبل المستخدمين، عند إضافة ال server origins and script endpoint سيعزز هذا الأمر الحماية من هجمات ال XSS، عند القيام بهذه الأمور من قبل الآدمن فإن المتصفحات ستتأثر بشكل كبير عند عرض معلومات هذه الصفحات مثل طريقة جلب وعرض ال inline js، فهذه الخاصية ستكون غير مفعلة في الشكل الافتراضي مثال:

```
<!-- bad.html -->
<script>
  function doAmazingThings() {
    alert('YOU AM AMAZING!');
  }
</script>
<button onclick='doAmazingThings();'>Am I amazing?</button>
```

من خلال هذه الطريقة فأنت تحمي نفسك من تنفيذ أ
JS تم تضمينه للـ

```
<!-- amazing.html -->
<script src='amazing.js'></script>
<button id='amazing'>Am I amazing?</button>
```

فهذا هو الأسلوب الأمثل لكتابة الشيفرة البرمجية لأنك تستطيع عمل ache

قواعد مراجعة الشيفرة البرمجية المتعلقة بال authentication

- تأكد بأن ال form المسؤول عن عملية تسجيل الدخول خاضع ل TLS، فإن كان في سياستكم في السماح لل HTTP من الوصول إلى الصفحة واستخدامها، فعليك جعل ال action الخاص بهذا الريكوست HTTPS، وذلك لتحمي هذا المستخدم من الهجمات المحتملة Man-in-middle، ومع أن هذا الأمر لوحده غير كافي لأن الصفحة نفسها HTTP إلا أن هذا يخفف أو يصعب الأمر فقط...، لأن المخترق يمكنه تغيير ال form submission url.
- تأكد دوما من أن اسم المستخدم case-insensitive، فمثلا ال email جيب أن لا يختلف إذا كتب a@b.com عن A@b.com
- تأكد من أن رسائل الأخطاء التي تتعلق بمحاولات الدخول الفاشلة لا تعبر عن مكان الخطأ ولا تقدم معلومات تساعد المخترق على ذلك، مثلا: إذا تم طباعة رسالة تفيد بأن اسم المستخدم صحيح، فسيعلم المخترق أن كلمة المرور هي الخاطئة وبهذا سيركز جهده على كلمة المرور، والصحيح هنا: (اسم المستخدم أو كلمة المرور خاطئة)

قواعد مراجعة الشيفرة البرمجية المتعلقة بال authentication

- يجب الاهتمام بالقواعد الخاصة بكلمات المرور، فمثلا يجب الحرص على أن لا تقل كلمة المرور عن 10 خانات، لأنها ستعد كلمة سهلة للتخمين والاختراق، كما أنه يفضل وضع مجموعة من القواعد لكتابة كلمة المرور مثل وجود أحرف Capital وأحرف small، أرقام ورموز خاصة...إلى آخره
- لنحمي أنفسنا من هجمات "brute force attacks" فعلينا وضع مجموعة من القواعد التي تمنع مستخدم ما من تسجيل الدخول لحسابه لمدة زمنية معينة أو بفواصل زمنية معينة بعد القيام بعدة محاولات فاشلة، مثلا اذا قام المستخدم بكتابة بمحاولة تسجيل الدخول لخمس مرات متتالية وكانت فاشلة بمنعه من المحاولة لمدة 5 دقائق أو مثلا بزيادة طردية للمحاولة مجددا بعد كل محاولة فاشلة، مثلا 10 ثواني، بعدها 20 ثانية...وهكذا

قواعد مراجعة الشيفرة البرمجية المتعلقة بال authentication

- يجب التأكد من استخدام نظام التشفير الآمن والمناسب لحفظ كلمات المرور، فمثلا ال md5 لا يعد آمنا للاستخدام!
- يفضل تقديم طريقة وآلية حماية إضافية للتعامل مع المعلومات الحساسة والمالية للمستخدمين مثل تطبيقات البنوك من خلال إرسال رسالة نصية للهاتف قبل إرسال الأموال...
- للأنظمة الداخلية يفضل وضع نظام يجبر الموظفين أو مستخدمي هذه الأنظمة على تغيير كلمات مرورهم بشكل دوري من خلال النظام ومنعهم من استخدام كلمات مرور قديمة
- انتبه أثناء مراجعتك الشيفرة البرمجية من ال backdoor

INSECURE DIRECT OBJECT REFERENCE

تعتبر ال Insecure Direct Object Reference واحدة من الثغرات الشائعة والتي يقع فيها العديد من المبرمجين أثناء العمل على تطبيقات الويب، هذه الثغرة باختصار تتيح للمهاجم من الوصول إلى معلومات لا يحق له الوصول إليه أو تعديلها أو حذفها عن طريق الوصول إلى URL وال database records والملفات...إلى آخره، ومن أشهر الأمثلة على ذلك ال SQL Injection الذي يتم من خلال ال URL param، ومن خلال ال HTTP POST requests والتي يمكن من خلالها المهاجم من تغيير ال post request، ومن الأمثلة الشهيرة على ذلك ما اكتشفه إبراهيم رأفت في موقع ياهو

حيث تمكن من تعديل ال fid, cid والذي مكنه من حذف تعليقات لأناس آخرين وتوصل بالنهاية لإمكانية حذف 15 مليون record من المعلومات ل users قاموا بتعبئتها! لذلك، تأكد عند كتابتك لأي شيفرة برمجية أو مراجعتها أن المستخدم الذي يملك الصلاحية المناسبة هو فقط من يستطيع الوصول، وأن اليوزر رقم 5 لو قام بتغير الرقم ل 6 فيجب منعه من ذلك

prop=addressbook&fid=367443&crumb=Q4.P5LBfBe.&cid=12365478910&cmd=delete_comment

INSECURE DIRECT OBJECT REFERENCE

كما يجب التأكد من ال business flow بأنه لا يسمح بهذا الوصول والتأكد من البيانات التي يتم إرسالها قبل معالجتها من قبل أي object في تطبيق الويب مثل ال database query، ويجب أن يكون هذا التحقق عن طريق ال server side لأن ال client side يمكن تجاوزه من قبل المخترقين كما هو معلوم.

مبادئ مهمة للتعامل مع ال Authorization

ال Authorization مهمة جدا ولا تقل أهمية عن ال Authentication!، فبعد تسجيل الدخول مثلا أنت ملزم بالتحقق من أن هذا المستخدم يحق له الوصول لهذه الصفحة / المعلومات وهل لديه الصلاحية الخاصة مثلا بالحذف أو الإضافة أو العرض...إلى آخره

بالإضافة إلى ما ذكر يجب التأكد بأن التحقق من ال authorization يجب أن يكون في المكان الصحيح، ومن الأخطاء المشهورة مثلا في ال MVC design pattern وضع شرط التحقق داخل ال view! وهذا يعني أن ال attackers إذا قام بالوصول إلى action الصحيح فسيقوم بتنفيذ ما بداخله وسيتم منعه من صفحة العرض فقط! لذلك احذر من هذا النوع من الأخطاء!

مبادئ مهمة للتعامل مع ال Authorization

لذلك، عند مراجعتك لشيفرة برمجية تحتوي في ثناياها Authorization يجب عليك التأكد مما يلي:

- كل action/endpoint/function يحتاج إلى صلاحية للوصول يجب التحقق من أن ذلك يتم فيه بالشكل الصحيح، وأن التحقق موجود
- عملية التحقق يجب أن تتم بشكل مركزي وفعال، لا أن يتم كتابة شيفرة التحقق مرارا وتكرارا في كل function وفي كل method، ويمكن الاستعاضة عن ذلك من خلال عدة طرق...
- في حالة الوصول الغير مصرح به، يجب أن يتم إرجاع الخطأ 403، والذي يعني Not Authorize
- إياك أن تسمح بوجود أي شيفرة برمجية تتعلق بالصلاحية وتؤثر عليها من خلال ال client side، مثلا إرسال الصلاحية من خلال hidden field.

مبادئ مهمة للتعامل مع ال Authorization

- في حالة وجودة صفحات متعددة أو وظائف متعددة لإتمام غرض محدد، فيجب التحقق من ال authorization في كل خطوة من هذه الخطوات، ومن هذا مثلا صفحات الشراء من خلال الموقع الإلكتروني مثلا confirmPament.php، payment.php, fillCard.php, profile.php...
- عادة تكون القاعدة العامة هي منع الوصول لكل الصفحات والاستثناء يتم على صفحات محددة، إن تم هذا الأسلوب فهذا سيخفف من كمية الأخطاء الممكنة، بدلا من عكس القاعدة وهي السماح للكل وتخصيص البعض...
- يجب أن تتأكد بأن أي حسابات تستخدم لأغراض ال test و موجودة داخل ال configuration يجب أن يتم حذفها حتى لا يتم استغلالها.

External Resource VULNERABILITIES

العديد من المطورين والعديد من الشركات تقوم باستخدام العديد من السكربتات Third-part لتقليل الوقت المستخدم أثناء كتابة المشروع والحصول على موثوقية أكبر وسكربت tested بشكل أكبر وأفضل، لكن، هذا الأمر يزيد من إحتمالية وجود مشاكل في الوظائف التي تقوم بها هذه المكتبة أو المشاكل الأمنية التي يمكن أن يلحق بها... لذلك، على ال code reviews أن يراجع ويهتم بهذه التفاصيل...، وطبعا كلما زاد عدد المشاريع في الشركة وزاد حجم المشروع زادت صعوبة ذلك، لذلك يجب أن تضع الشركة بناءا على حجمها السياسة الخاصة بها والتي تناسبها، والآن نذكر بعض أهم النقاط التي يمكن أخذها بعين الاعتبار: (ملاحظة: يقصد في المكتبة أثناء السياق أي external resource مثل framework, library, script...إلخ)

- استخدم فقط ما تحتاجه من مكاتب خارجية، وقم بحذف ما لم تحتاجه أو لو تعد تستخدمه!

External Resource VULNERABILITIES

- تأكد من موثوقية المكتبة المستخدمة وكمية المشاكل التي تتعلق بها وعدد مستخدميها ـ وعدد المشاكل التي ما زالت open ولم يتم إغلاقها ونوعيتها.
- تأكد من تضمين الوظائف التي تحتاجها من المكتبة فقط، لأن ذلك سيقلل من حجم الشيفرة البرمجية النهائية، ويقلل عدد الأخطاء الممكنة، ويقلل من المشاكل الأمنية في الأجزاء التي لن نستخدمها، فمثلا بالجافا سكربت يمكنك تضمين ملف جافا سكربت واحد من أصل مكتبة كاملة أو تحميل هذا الملف فقط.
- يجب التأكد من الإصدار الخاص بالمكتبة التي ترغب باستخدامها، وهل ستسمح السياسة الخاصة بشركة بمتابعة التحديثات المستقبلية أم الوقوف على هذا الإصدار.
- يجب على الشركة تحديد ما يسمح باستخدامه كمكتبة خارجية وما لا يسمح بذلك.
- يمكن للشركة سحب نسخة من المكتبة الخارجية ومن ثم فتح branch خاصة بها لمعالجة وسحب الجزئيات التي تهتم بها فقط دون غيرها، ودون أن تتأثر بالمكتبة الأصلية.

إن نعم اللّٰه -سبحانه وتعالى- علينا عظيمة, فلتجعل قلبك دومًا يعترف بفضل اللّٰه -سبحانه وتعالى- عليك, ولتجعل لسانك دائم الثناء والشكر لله -سبحانه وتعالى- على ما وهب وأعطى، ولتجعل جوارحك تُظهر ما أخفاه قلبك, ونطق به لسانك في استعمال ما رزقك اللّٰه مما أحله اللّٰه, وتجنب كل ما نهاك اللّٰه عنه.

التحقق من ال REDIRECTS

واحدة من الطرق السهلة والتي يستخدمها ال attackers هي محاولة استغلال نقطة الضعف التي لديك في ال redirects flow والتي ستمكنه من عمل phishing لمستخدمي الموقع الخاص بك، تكمن خطورة هذا الأمر أن المستخدم قد يقع ضحية هذا التصيد من دون أن ينتبه انه ذهب لموقع ال attackers

مثال:

الرابط الأصلي:

http://www.mywebsite.com/redirect?URL=http://mywebsite.com/myLogin

الرابط بعد تعديل المهاجم للرابط:

http://www.mywebsite.com/redirect?URL=http://attackers.com/fakeLogin

كيف تتم هذه الهجمة؟ بكل بساطة إذا استطاع المهاجم تعديل الرابط في ال url بأي طريقة ممكنة أو من خلال إستغلال جهل المستخدم وضعف نظم الحماية في الموقع سيتم له ما أراد!، فمثلا لو فرضنا ال js file كان فيه: window.location.href=getParam(url) ومن دون تحقق فستكون هذه الهجمة متاحة بسهولة...

التحقق من ال REDIRECTS

لذلك، ما يجب على ال reviewer التحقق من أن الرابط المراد الذهاب إليه صحيح!، ويمكن القيام بهذا في عدة طرق:

- Whitelist ويكون ذلك من خلال التحقق من أن ال domain الخاص بال url المراد التحويل له صحيح، أو من خلال وضع keys تمثل mapping بين ال url المراد التحويل إليه وما هو موجود في ال url مثلا mywebsite.com/redirect/login تعني mywebsite.com/redirect?URL=mywebsite.com/login.php

- Encrypt: من خلال هذه العملية يتم وضع ال url المراد الذهاب إليه مشفر، ويتم فك التشفير عند التحويل، هذه العملية تصعب على المخترق العملية حتى يجد ال key المناسب لفك التشفير.

ال Error Handling

تعتبر معالجة الأخطاء ومتابعتها من المواضيع التقنية ذات الأهمية العليا!، لما تشكله هذه الأخطاء من مرجعية للمطورين لاكتشاف مواقعها ومعالجتها، وتحديد المشاكل المختلفة التي تظهر عند مستخدمين النظام، لكن إذا لم يتم معالجة الأخطاء والتحكم بها في طريقة صحيحة قد تكون هذه الأخطاء وسيلة لل attackers للوصول إلى معلومات تجعل من اختراقك أمرا سهلا، وتجعل استغلال هذه الأخطاء أمرا أسهل للمستخدمين، وعادة ما يتم معالجة الأخطاء بطريقتين هما:

1. من خلال عمل lock للنظام أو التطبيق، وإغلاق ال session وإرسال notification للمسؤولين عن هذه الأنظمة...
2. من خلال التحكم بالأخطاء عن طريق منع المستخدم من متابعة سير العمل على هذه الجزئية وإظهار الخطأ المناسب له -وعادة ما تكون في صفحة منفصلة مثلا 500 أو 404-...

ال Error Handling

بناءا على ما سبق، هناك بعض النصائح يجب أن يراعيها ال code reviewer عند مراجعة الشيفرة البرمجية والتحقق من طريقة معالجة الأخطاء المتوقعة (نذكر منها بعضها):

- الأخطاء المتوقعة لأي تطبيق يمكن أن تأتي من 4 مصادر وهي ال Hardware بشقيه ال (physical وال technical) وال business logic (نتائج التطبيق لا تتوافق مع متطلبات البيزنس أو لا تتطابق متطلبات البيزنس مع التقنية المتوفرة) وال environment (وهي الأخطاء الخاصة ب env مثل ال prod, dev, test...إلى آخره والتي تحصل نتيجة لخطأ في إعداد ملف ال env أو باستدعائه) وال dependence (والتي تمثل جميع الأخطاء التي يعتمد على وجودها التطبيق فإن حصل فيها خلل أثر على التطبيق الخاص بي بشكل مباشر أو غير مباشر مثل third part library أو أخطاء الشيفرة البرمجية للمطور أثناء العمل)، فإن علمت هذه الأنواع، علمت أن مراجعة الشيفرة البرمجية يجب أن تنظر لطبيعة الأخطاء المتوقعة لتحديد ماهيتها ومعالجتها بالطريقة الصحيحة والمناسبة...

ال Error Handling

- يمكن استخدام ال servers مثل ال apache للتحكم بال response الخاص بالأخطاء وإرسالها لل client بطريقة مناسبة وموحدة فمثلا:
  ErrorDocument 404 /errors/not_found.html
  (هناك طرق أخرى لكننا تطرقنا للأسلوب باعتباره مباشرا)
- أي خطأ يتوقع أن يصدر exception يجب أن يتم وضعه داخل try catch، ويجب عليك التحقق من ال scenario الموجود داخل ال catch!
- لا تهمل أي خطأ
- قم بعرض الأخطاء بطريقة مناسبة لا تظهر أي معلومات لما تم معالجته مثل ال sql queries
- تحقق من جميع العناصر التي يتم إدخالها وأنها ضمن مجموعة القيم المسموح إدخالها مثلا الأشهر من 1-12 إذا أرسل أحدهم 13 فيجب منعه وحفظ ال log الخاص به في مكان يجعل أمر متابعة هذا ال user والحركات التي يقوم بها أمرا سهلا لاحتمالية أن يكون attacker.

الخلاصة

بعد هذا الطرح، سنقوم بتلخيص أهم النقاط التي يجب مراجعتها عند مراجعة أي تطبيق، والتي ستمثل ال checklist الخاصة بنا، ويمكن تمثيلها ب: -ملاحظة: الإجابة على هذه الأسئلة بعد التحقق إما PASS وإما FAIL-

أولا: General Category:

- هل يوجد هناك أي backdoor محتملة؟
- هل ال external library المستخدمة في التطبيق الخاص بنا updated؟ وهل هي ضمن ال process التي تضمن تحديثها؟
- الوصول إلى أي Class يحتوي على معلومات مهمة يجب أن يتم من خلال ال protected api.
- يجب التأكد بأن معلومات الأمان يجب أن لا يتم تخزينها على شكل plan text ولمدة طويلة في ال memory.

الخلاصة

- يجب التحقق من أن جميع محتويات ال Array ضمن ال range (مثال الأشهر)
- يجب التأكد من جميع المعلومات الحساسة التي تم جلبها، وهل يلزم ذلك؟ وهل استخدامها في محله؟

ثانيا: Business Logic and Design:

- هل هناك أي configuration موجودة وغير مستخدمة؟
- إذا كانت ال request parameters ستحدد flow يتعلق بال business logic، فهل ال mapping بين صلاحيات هذا المستخدم وال actions التي سيصل إليها صحيحة؟ وتسمح له بذلك؟
- يجب التأكد من المدخلات التي تم إرسالها من قبل المستخدم تتوافق مع ال object instance وضمن القواعد المسموحة للتعديل أو التغيير، ومن هذا ال param، فإذا قام المستخدم مثلا بإرسال price على شكل param يجب أن لا يتم عمل binding لها، وإن تم فيجب معالجتها لتبقى قيمة ال object الإفتراضية لمنع أي تغيير!

الخلاصة

- يجب أن تتحقق من أن عملية التحقق منفذة بالشكل الصحيح ولا وجود لمكان أو param يمكن من خلاله تكوين backdoor.
- يجب أن تتحقق من أن جميع الملفات والمجلدات الموجودة في ال web root directory ضرورية ولا يوجد ما قد يشكل مخاطر للتطبيق لاحقا.
- يجب التحقق من عدم وجود أي configuration يعطي صلاحية Access All.
- يجب التحقق من أن النظام لا يستخدم ال flat database
- يجب التحقق من أن ال validation centralized وبإمكانه التحقق من جميع ال inputs وجميع ال requests، وإن كان هناك استثناء فلماذا وأين؟
- يجب التحقق من ال validation centralized يعالج ال special characters يمنع وصولها، وإن كان هناك استثناءات فيجب تحديدها مسبقا ولماذا وأين؟

الخلاصة

- هل يوجد أي جزئية في ال flow تمنع ال validation عند مرحلة معينة أو لسبب معين؟
- هل النظام يسمح بتنفيذ أوامر (commands) على نظام التشغيل أو يسمح بإنشاء connection للوصول إليه من الخارج؟
- هل ال privileges المعطاة للمستخدمين هي أقل ما يمكن وتلبي احتياجات المستخدم؟
- هل التطبيق أو الشيفرة البرمجية تم تصميمها للتعامل مع الأخطاء بسلاسة؟

ثالثا: Authorization:

- هل يتم التحقق من ال authentication وال authorization بشكل صحيح؟
- في حال وجود أي fail عند التحقق، هل يتوقف التطبيق عن تنفيذ ال request؟
- يجب التحقق من عدم وجود أي backdoor في ال authorization.

الخلاصة

- هل تم توزيع الصلاحيات على مستوى الفولدرات والملفات الموجودة على web root directory بالشكل الصحيح؟
- هل يتم التحقق من المشاكل الأمنية قبل التحقق من ال validation على ال user input؟
- هل تم وضع قواعد لزيادة مستوى التعقيد لكلمة المرور؟ وهل يتم التحقق منها في الشكل الصحيح؟
- هل يتم كتابة ال password الذي قام المستخدم في بكتابته في أي مكان مثل ال logs أو ال console... إلخ؟
- هل هناك password expiration؟ وهل تم التحقق من آلية عملها؟
- هل تم تطبيق ال anti-spoofing measure؟

الخلاصة

رابعا: Session Management:

- طريقة تصميم التطبيق، هل تضمن session آمن؟
- هل يتم مشاركة ال session في أكثر من مكان؟ وهل يتم التحقق من ال session في جميع الأماكن؟
- يجب التحقق من عدم وجود أي session تم إرساله من خلال parameter؟!
- يجب أن يكون وقت صلاحية ال session cookie قصير
- يجب أن يكون ال session cookie encrypted
- يجب التحقق من البيانات الموجودة بال session
- ال session id يجب أن يكون complex
- يجب تطبيق ال session in-activity timeout.

الخلاصة

خامسا: Cryptography:

- هل تم حفظ كلمة المرور على شكل encrypted format؟
- هل تم حفظ ال database credential على شكل encrypted format؟
- هل يتم إرسال البيانات من حلال encrypted channel؟ Https, SSL؟
- هل يتم إرسال جميع البيانات المهمة والحساسة من خلال encrypted form؟ (https form action)
- هل يستخدم النظام custom encryption scheme؟ أم أنه يستخدم إحدى الخوارزميات القوية والموثوقة؟
- هل يتم إستخدام آخر إصدارات نظام التشفير؟ وهل يتم ضمان تحديث الخوارزميات أولا بأول؟
- هل تم وضع ال key الخاص بال Cryptography داخل الكود؟ (تعد من أسوأ أو أسوأ طريقة)

الخلاصة

سادسا: Logging and Auditing:

- هل يتم حفظ أي sensitive data داخل ال log؟
- هل يتم تسجيل المحاولات الناجحة والفاشلة لل connection داخل ال logs؟
- هل هناك أي process تراها قيد العمل تقوم بقراءة ال logs، وهل هذا يدل على وجود سلوك unintended/malicious؟

سابعا: Input Validation:

- هل يتم التحقق من "جميع" ال input الي تصل من خلال المستخدم؟ وهل تم وضع القواعد المناسبة للتحقق وتنفيذها بالشكل الصحيح؟ مثل type, length, format, range

الخلاصة

ثامنا: User Management and Authentication:

- الصلاحيات الخاصة بكل مستخدم يجب أن تكون موثقة، مثلا Super Admin => All، Sales => orders...إلخ.
- يجب أن تكون ال cookie secure and http only
- يجب أن تكون ال cookie encrypted
- ال authentication credentials إذا تم إرسالها من خلال HTTP GET فهذه كارثة تقنية!
- يجب التحقق من أن الصلاحيات التي تعطى وال role الخاصة بكل مستخدم يتم إدراجها للمستخدم عند ال authentication بشكل صحيح وواضح
- يجب التحقق من أن ال authentication يتم بشكل صحيح ولا يوجد هناك طريقة للتحايل على ذلك!

الخلاصة

تاسعا: Data Management:

- يجب أن يتم التحقق من البيانات على مستوى ال server.
- يجب أن يتم التحقق من ال headers في كل request
- التحقق من ال output بعد معالجة البيانات قبل اتخاذ أي إجراء، هل تحتوي على untrusted tags، وهل تم التحقق من ال encoding tag؟

فلْتعلمْ أن شكر المحسن على إحسانه هو من طَيْب الأفعال، فإن أحسن إليك أحدهم، فاحمد اللّٰه، واشكر المحسن على إحسانه.

وآخر دعوانا أن الحمد لله رب العالمين

أنيس حكمت أبوحميد
Email Github Stackoverflow Slideshare