Application Security

Application Secure Design Foundations

يعني إيه Application Security؟

Application Security هو المجال اللي بيهتم إزاي نأمّن الـ applications بتاعتنا من أول مرحلة الـ development لحد ما تطلع live في الـ production.

هو مش بس Penetration Testing زي ما ناس كتير فاكرة، لأن الـ Penetration Testing بييجي في آخر مرحلة من الـ development، وده بيبقى متأخر جدًا لو اكتشفنا vulnerability. ساعات بنضطر نرجع نعيد الشغل من الأول، وده بيضيع وقت وفلوس.

الفكرة إننا نكون proactive مش reactive، يعني نفكر في الأمان من البداية، نستخدم secure coding، تصميم كويس، وندمج الأمان في كل خطوة من دورة تطوير البرمجيات.

Reactive vs. Proactive Approaches

- الـ Penetration Testing هو reactive approach، بيبص على الثغرات بعد ما الـ development يخلّص.
- الـ Application Security هو proactive approach، بيركز على منع الثغرات من أول ما نبدأ الـ development.

DevSecOps: دمج الأمان في DevOps

DevSecOps هو إننا نجمع بين الـ development، الأمان، والعمليات عشان نضمن إن الأمان يكون جزء من كل خطوة في تطوير البرمجيات.

النقط المهمة هنا:

- **Shifting Security Left**: نبدأ بالأمان من أول ما نكتب الـ code، في الـ commit، الـ build، وصولًا للـ release.
- **Automation**: نستخدم tools بتعمل الفحوصات الأمنية المتكررة أوتوماتيك، وده بيسرّع الشغل ويقلل الغلط اليدوي.

أهداف إضافية

- نكتشف الثغرات قبل ما الـ application تطلع للـ production.
- ندير الـ configurations والـ third-party dependencies بطريقة آمنة.
- نقلل الثغرات في مرحلة الـ production.
- نساعد على إصدارات سريعة من غير ما نهمل الأمان.
- نسرّع تصليح أي مشاكل نكتشفها.

مراحل DevSecOps في التطوير

مرحلة Pre-Commit

- بتحصل قبل ما نعمل commit للـ code.
- نستخدم pre-commit hooks عشان نكتشف أي secrets أو أنماط غلط.
- نستخدم IDE plugins بتعمل code scanning في الوقت الحقيقي.
- نتبع إرشادات secure coding.

مرحلة Commit (Integration)

- بتحصل لما نعمل commit للـ code في الـ repository.
- الأدوات والممارسات:
  - **SAST (Static Analysis)**: بيسكان الـ code عشان يكتشف الثغرات المعروفة.
  - **SCA (Software Composition Analysis)**: بيسكان الـ dependencies عشان يكتشف أي مخاطر معروفة (زي OWASP Dependency-Check، Trivy).
  - **IaC Security**: بيراجع الـ infrastructure-as-code عشان يكتشف أي misconfigurations.
  - **Container Security**: بيسكان الـ Docker images (زي Trivy، Anchor، Docker Scan).

مرحلة Acceptance/Test

- بتحصل في بيئات pre-production.
- الأدوات والطرق:
  - **DAST (Dynamic Analysis)**: بيسكان الـ application وهي شغالة runtime (زي OWASP ZAP، Burp Suite) عشان يكتشف ثغرات.
  - **IAST (Interactive Analysis)**: بيدمج بين DAST وSAST عشان دقة أعلى باستخدام agents جوا الـ app.

مرحلة Production

- البيئة النهائية اللي الـ application بتشتغل فيها live.
- التركيز على:
  - فحوصات الـ regulatory والـ compliance.
  - **RASP (Runtime Application Self-Protection)**: بيراقب ويدافع عن الـ application من الهجمات في الوقت الحقيقي.

مرحلة Operation

- المراقبة المستمرة والاستجابة.
- الممارسات تشمل:
  - استخدام SIEM tools والـ centralized logging.
  - وضع سياسات retention monitoringg.
  - الاستعداد لكشف ومواجهة أي تهديدات بعد الـ deployment.

مبادئ Secure by Design

Least Privilege

- المستخدمين والـ services لازم يكون عندهم بس الصلاحيات اللي محتاجينها لدورهم.

Defense in Depth

- نستخدم طبقات أمان كتير عشان لو طبقة فشلت، الـ system ميتأثرش كله.

Secure Defaults / Fail Secure

- الـ systems لازم تكون افتراضيًا في حالة آمنة، ولو فشلت، تفشل بطريقة آمنة (يعني تقفل الوصول مثلًا).

Open Design

- نستخدم بروتوكولات أمان وتشفير معروفة ومفتوحة للتدقيق (زي AES، RSA، OpenPGP).

Minimize Attack Surface

- نقلل نقاط الدخول عن طريق تعطيل الـ services أو الـ ports أو الـ features اللي مش ضرورية.

Validate Input

- لازم نعمل input validation من ناحية الـ client والـ server عشان نمنع هجمات زي SQL Injection.
- نستخدم parameterized queries بدل دمج الـ input بشكل ديناميكي.

Threat Modeling

- بيساعدنا نحدد ونقيّم ونقلل التهديدات المحتملة قبل ما تحصل.

العملية

1. نفهم الـ architecture بتاعة الـ application والـ trust boundaries.
2. نحدد التهديدات بناءً على الـ components وتدفق البيانات.
3. نقيّم الثغرات المحتملة.
4. نحدد الدفاعات والحلول.

STRIDE Framework

- **Spoofing**: حد ينتحل شخصية مستخدم - بنحلها بـ MFA، CAPTCHA.
- **Tampering**: تعديل غير مصرح للبيانات - بنحلها بـ validation.
- **Repudiation**: إنكار عمل حاجة - بنحلها بـ auditingg logging.
- **Information Disclosure**: تسريب بيانات حساسة - بنحلها بـ access control secure API designg.
- **Denial of Service**: تعطيل الخدمة - بنحلها بـ rate limiting، timeouts، lockoutsg.
- **Elevation of Privilege**: الحصول على صلاحيات أعلى - بنحلها بـ strict access controls.

الأدوار الرئيسية في Application Security

- **DevSecOps Engineer**: بيأتمت الأمان في الـ CI/CD pipelines وبيأمن بيئات الـ cloud.
- **Application Security Engineer**: بيعمل code reviews وبيستخدم أدوات SAST/DAST/IAST.
- **Cloud Security Engineer**: متخصص في تأمين الـ cloud infrastructure والـ services.
- **Security Architect**: بيصمم الـ security architecture والسياسات للمشاريع.

المهارات المطلوبة

- إتقان لغة برمجة واحدة على الأقل.
- فهم قوي لمفاهيم الأمان والثغرات الشائعة (زي OWASP Top 10).
- معرفة ببيئات الـ web، الـ mobile، أو الـ cloud.
- خبرة في الـ CI/CD pipelines.
- معرفة بـ secure codingg code review.
- خبرة عملية مع أدوات الأمان.
- خبرة في الـ penetration testing بتكون ميزة إضافية.

Threat Modeling - Key Concepts & Notes

يعني إيه Threat Modeling؟

Threat Modeling هو عملية proactive بنحاول فيها نكتشف التهديدات المحتملة للـ application أو الـ system قبل ما المهاجمين يكتشفوها.

بيساعدنا نفهم:

- الـ exposure (يعني الـ app مفتوح على الإنترنت ولا داخلي؟).
- الأصول الحساسة اللي ممكن المهاجمين يستهدفوها.

مين بيعمل Threat Modeling؟

- ممكن يتعمل:
  - داخل الشركة.
  - كخدمة أمان مقدمة من شركات خارجية للعملاء.

أهداف Threat Modeling

- نعمل threat model موثق نقدر نتواصل بيه مع:
  - الـ developers.
  - أصحاب الـ application.
  - مديرين المشروع.
- نتابع التهديدات وحالة الحلول بتاعتها (يعني تم حلها، ولا لسه، ولا اتقبلت كـ risk).

ليه نعمل Threat Modeling؟

- بيقلل تكلفة تصليح الثغرات لأننا بنكتشفها بدري.
- بيدينا رؤية شاملة:
  - التفاعلات، الـ trust boundaries، الـ components، تدفق البيانات.
  - حالة حل التهديدات.
- بيستخدم كـ input لأدوات زي SAST، DAST، SCA.

إمتى بنعمل Threat Modeling؟

- عادةً في مرحلة الـ Design من الـ Secure Development Life Cycle (SDLC).
- بيحصل قبل ما نبدأ نكتب الـ code.
- بيحتاج تعاون بين:
  - الـ Application Security Engineers.
  - الـ Developers.
  - الـ Architects.

أطر العمل لـ Threat Modeling

- STRIDE (الأساسي هنا).
- PASTA.
- OCTAVE.
- VAST.

الأدوات الشائعة

- OWASP Threat Canvas.
- Threat Dragon.
- Microsoft Threat Modeling Tool.

مكونات Threat Model

Entity (Threat Actor)

- مستخدم أو system بيتعامل مع الـ app (زي admin، customer).
- بيترسم بشكل برتقالي في Threat Canvas.

Data Store

- المكان اللي البيانات موجودة فيه (زي DB، S3 bucket، FTP).
- ليه أيقونة خاصة.

Process

- أي عملية أو خدمة (زي web app، API).
- بيمثل الـ business logic الأساسي.

Trust Boundary

- بيحدد مكان الـ component: داخلي، على الإنترنت، cloud، إلخ.
- بيساعدنا نحدد الـ attack surfaces.

Data Flow Diagram (DFD)

- الناتج النهائي اللي بيورّي العلاقات بين الـ components.
- الأسهم بتورّي اتجاه التفاعل (request/response).
- ممكن يكون one-way (زي sensor → DB) أو two-way (زي app ↔️ user).

سيناريو مثال - Book Store Web App

المكونات

- **Entity**: مستخدم مسجّل (على الإنترنت).
- **Process**: Web Application (بتواجه الإنترنت).
- **Data Stores**:
  - AWS S3 Bucket → صور المستخدمين (في cloud network).
  - Internal Database → بيانات العملاء (في internal network).

Trust Boundaries

- الإنترنت، الشبكة الداخلية، الـ cloud network.

التهديدات المكتشفة

- مشاكل Auth/Authz (زي privilege escalation).
- Clickjacking.
- XSS.
- DoS.
- Insecure file upload.
- SQL Injection (لو بنستخدم SQL DB).
- التهديدات اللي مش منطقية (زي NoSQLi لو بنستخدم MySQL) بنستبعدها.

الحلول

- لكل تهديد، بنحدد controls:
  - الحالة: تم تنفيذه / تحت التنفيذ / لسه مفقود.

Risk Acceptance

- الشركة ممكن تقبل بعض المخاطر لأسباب استراتيجية.

Threat Model Summary (PDF)

- DFD.
- ملخص المخاطر حسب فئات STRIDE.
- حالة الحلول.
- حدود المخاطر.
- بيستخدم للتقارير لأصحاب المصلحة.

🛠️ OWASP Top 10 CI/CD & DevSecOps Risks

يعني إيه DevSecOps؟

DevSecOps هو امتداد لـ DevOps، بيدمج الـ development، الأمان، والعمليات.

الهدف إننا نبني برمجيات آمنة وسريعة في نفس الوقت، مع الحفاظ على التوازن بين السرعة، الأمان، والجودة.

ليه DevSecOps مهم؟

- أساليب الأمان التقليدية بطيئة وبتيجي متأخر.
- DevSecOps بيخلّي الأمان يبدأ بدري (shifting left) في الـ SDLC.
- بيمنع حوادث زي اللي حصلت في Cloudflare في يوليو 2024، لما patch غير آمن ومش مراقب تسبب في انقطاع كبير.
- بيضمن التوافق مع معايير زي اللي بتطلبها Visa Mastercardg.

مبادئ DevSecOps الأساسية

Security as Code

- دمج فحوصات الأمان في الـ code ومراحل الـ pipeline.
- بيستخدم policy as code، code scanning، وفحص infrastructure as code.

Automation and Scalability

- الـ automation بتضمن تسليم أسرع مع فحوصات أمان ثابتة.
- الـ scalability بتتعامل مع الـ codebases الكبيرة وتعقيد الـ deployment.

Continuous Security

- مراقبة أمان مستمرة وإدارة الثغرات.
- اختبارات أمان أوتوماتيك ويدوية.
- مراجعات code مركزة على الأمان.

أدوات DevSecOps

SAST (Static Application Security Testing)

- بيسكان الـ code من غير ما ينفذه عشان يكتشف ثغرات زي SQL Injectiong XSS.
- ممكن يطلّع false positives.

DAST (Dynamic Application Security Testing)

- بيختبر الـ application وهي شغالة.
- أدوات زي Burp Suite بتحاكي هجمات على APIsg web interfaces.

SCA (Software Composition Analysis)

- بيسكان الـ third-party libraries عشان يكتشف ثغرات معروفة.
- أدوات زي Snyk، Black Duck، Trivy، OWASP Dependency-Check.

Mobile Security Tools

- بتستخدم لـ static analysisg dynamic للـ mobile apps.
- أمثلة: MobSF، NowSecure.

IAST (Interactive Application Security Testing)

- بيدمج تقنيات SAST وDAST.

IaC and Container Scanning

- أدوات زي TFSec Checkovg بتحلل الـ infrastructure code.
- الـ container image scanning بتكتشف الثغرات قبل الـ deployment.

شرح CI/CD Pipeline

Continuous Integration (CI)

- الـ code بتاع الـ developers بيتحوّل لـ build files (زي WAR، JAR).

Continuous Deployment/Delivery (CD)

- الـ build files دي بترفع على الـ servers للاستخدام.

مراحل DevSecOps Pipeline

1. **Planning**: Threat Modeling، تحديد المتطلبات، كتابة user stories.
2. **Development**: Secure coding، code reviews، static analysis.
3. **Build**: تحويل الـ code لـ executable build files.
4. **Release**: إعداد النسخة النهائية الموقّعة للـ deployment.
5. **Deployment**: رفع الـ code على الـ servers الحية.
6. **Monitoring**: مراقبة الـ application من ناحية الهجمات والأداء.

OWASP Top 10 CI/CD Risks

1. Insufficient Flow Control Mechanisms

   - نقص في التحكم في العمليات زي الترقيات، الموافقات، أو الرجوع للنسخة السابقة.

2. Insufficient Identity and Access Management

   - حسابات بصلاحيات زيادة، أو نقص في MFA أو RBAC.

3. Insecure System Configuration

   - استخدام إعدادات افتراضية، ports مكشوفة، أو CI servers غير محمية.

4. Insecure Third-Party Dependencies

   - استخدام packages فيها ثغرات. لازم نعمل SBOM ونسكان باستمرار.

5. Poisoned Pipeline Execution

   - scripts خبيثة أو CI agents متخترقة ممكن تخرّب الـ builds.

6. Insufficient Pipeline Access Control

- لازم نقلل صلاحيات الـ pipelines عشان ما يأثروش على الـ hosts أو يوصلوا للـ secrets.

7. Improper Credential Management

- secrets مخزنة بشكل مكشوف، نقص في rotation، أو حسابات مشتركة.

8. Insufficient Logging and Monitoring

- نقص في أدوات المراقبة أو تنبيهات للـ anomalies أو التسللات.

9. Insufficient Supply Chain Integrity

- نقص في فحوصات التحقق زي hashes عشان نضمن سلامة الـ build.

10. Insecure Integration of External Tools

- الـ plugins أو الأدوات المستخدمة في الـ pipeline لازم تكون موثوقة.

حوادث شهيرة

- Cloudflare (2024): Patch غير آمن تسبب في فشل واسع، كلّف مليارات.
- SolarWinds Attack: كود خبيث اترفع في تحديث برمجي موثوق.
- Codecov Breach: اختراق في pipeline أداة أمان، أثر على المستخدمين.

🔍 Threat Modeling with STRIDE & OWASP Threat Dragon

نظرة عامة

Threat Modeling هو جزء أساسي من Application Security.

STRIDE هو الـ framework الأكثر استخدامًا للـ Threat Modeling.

OWASP Threat Dragon هي أداة مجانية ومفتوحة المصدر بتدعم STRIDE.

مبادئ STRIDE Model

STRIDE هو اختصار لست فئات تهديد:

1. Spoofing
   - لما حد ينتحل شخصية مستخدم.
   - أمثلة: سرقة credentials، brute-forcing logins، token theft.
2. Tampering
   - تعديل غير مصرح للبيانات.
   - أمثلة: اعتراض وتعديل HTTP requests، تعديل ملفات.
3. Repudiation
   - لما حد ينكر إنه عمل حاجة.
   - بنحلها بـ logging وauditing عشان نضمن التعقب.

4. Information Disclosure

- وصول غير مصرح للبيانات الحساسة.
- أمثلة: تسريب passwords، بيانات المستخدمين، أو سجلات داخلية.

5. Denial of Service (DoS)

- تعطيل توفر الخدمة.
- أمثلة: DoS/DDoS attacks، استنفاد الموارد، إسقاط الـ application.

6. Elevation of Privilege

- الحصول على صلاحيات أعلى من المسموح.
- أمثلة: privilege escalation، تخطي access controls، الحصول على admin rights.

OWASP Threat Dragon

نظرة على الأداة

- مجانية ومفتوحة المصدر.
- من تطوير OWASP.
- بتستخدم Data Flow Diagrams (DFD).
- بتدعم STRIDE لتحديد التهديدات وتخطيط الحلول.

إزاي تشتغل؟

- نعمل موديل باستخدام مكونات DFD:
  - Process.
  - Data Store.
  - Actor.
- نحدد الـ trust boundaries وتدفقات البيانات بين الـ components.
- نستخدم STRIDE عشان يقترح التهديدات المناسبة.

خطوات استخدام Threat Dragon

1. ندخل على threatdragon.com.
2. نسجّل باستخدام GitHub أو Local Session.
3. نختار:
   - نشوف موديل مثال.
   - نكمّل موديل موجود.
   - نعمل موديل جديد.

إنشاء موديل جديد

- نسمّي الموديل (مثلًا "Our New Web Application").
- نضيف owners، reviewers، system contactsg (اختياري).
- نختار STRIDE كـ threat modeling framework.
- نضيف مكونات:
  - Web Server (Process).
  - Web App Server (Process).
  - Database (Data Store).
  - Normal User / Admin (Actor).
- نربط الـ components بـ Data Flows.
- نحدد Trust Boundaries (مثلًا DMZ ↔️ internet).

إضافة التهديدات

- نسمّي التهديد (مثلًا "Impersonation of user via credentials theft").
- نوضح إزاي ممكن يحصل (مثلًا brute-force attacks).
- نضيف حلول:
  - تفعيل MFA.
  - استخدام strong password policies.
  - تطبيق rate-limiting.
- نحدد أولوية التهديد (مثلًا High).
- نضيف scores (اختياري).

أمثلة على التهديدات

- Data Tampering in Transit
  - الهجمات المحتملة: MITM.
  - الحلول: تفعيل TLS/SSL، إدارة الـ certificates، استخدام checksums.

إنهاء الموديل

- نحفظ الموديل كـ JSON file.
- نطلّع PDF report فيه:
  - DFD diagram.
  - قايمة الـ assets.
  - التهديدات وحالتهم.
  - الحلول المقترحة.

🗂️ Secure Source Code Control, GitHub, GitLab, Version Control

نظرة عامة على Automationg DevSecOps Pipeline

DevSecOps Pipeline بتدمج ممارسات الأمان في دورة حياة DevOps، مع التركيز على الـ version controlg automation.

الدورة بتبني على مفاهيم Application Security وبترجع لـ TryHackMe room "Intro to Pipeline Automation".

في النهاية، هتفهم:

- إيه هو DevSecOps Pipeline؟
- دور الـ automation في الأمان.
- معنى وأهمية version control.
- إزاي Git وأدوات الأمان بتشتغل في بيئات CI/CD.

مكونات DevSecOps Pipeline الأساسية

1. Source Code Storage / Version Control.
2. Dependency Management (هتتغطى في الدرس الجاي).
3. Testing، بما فيه security testing.
4. Continuous Integration.
5. Deployment Environments.

ليه محتاجين Version Control؟

- لو بتشتغل لوحدك، ممكن تخزّن الـ code على جهازك.
- لكن في فريق، لازم يكون عندك repositories مركزية.
- الـ repositories دي لازم تكون آمنة وتدعم التعاون، وده بيخلّينا نسأل:
  - إزاي ندير الـ access control والصلاحيات؟
  - إزاي نتابع التغييرات في الـ code من developers كتير؟
  - إزاي ندمج الـ code مع أدوات التطوير (IDEs)؟
  - إزاي ندير الإصدارات ونفرّق بين تحديثات المميزات؟
  - هنخزّن الـ repository فين - محليًا ولا في الـ cloud؟

إيه هو Version Control؟

Version Control بيسمح لنا:

- نحافظ على إصدارات متعددة من الـ code.
- نعمل branching لتطوير مميزات جديدة.
- نرجع لإصدارات سابقة.
- نتابع مساهمات أعضاء الفريق.
- بيدينا رؤية كاملة وتاريخ الـ codebase.

أنواع Version Control Systems

- **Git**: نظام version control موزّع، كل مستخدم عنده نسخة كاملة من الـ codebase.
- **SVN (Subversion)**: نظام مركزي، كل الـ developers بيعتمدوا على repository مركزي واحد.

منصات Git Hosting

- **GitHub**: أكبر مزود لـ Git repositories أونلاين، بيدعم الـ public والـ private repos.
- **GitLab**: حل self-hosted للشركات اللي بتفضل تخزّن الـ Git repositories داخليًا.

المخاوف الأمنية في Version Control

- **Authentication**: لو حساب developer اتخترق، ممكن يوصل لكل الـ codebase.
- **Access Control and Privilege Management**:
  - إعدادات الصلاحيات لازم تكون صح.
  - لو فشلت، ممكن يحصل مخاطر زي حذف الـ code بدون إذن.
- **Git Forensics and Git Leaks**:
  - الـ secrets (زي credentials) لو اتعملها commit في الـ repo، ممكن تبقى موجودة في تاريخ الـ repository حتى لو اتشالت.
  - أدوات زي Gitleaks بتسكان تاريخ الـ repository عشان تكتشف بيانات حساسة.

TryHackMe - "Intro to Pipeline Automation" Highlights

- استكشاف عملي لمفاهيم الـ pipeline.
- النقط الرئيسية:
  - GitHub هو أكبر مزود Git أونلاين.
  - GitLab بيستخدم لاستضافة Git server داخليًا.
  - Gitleaks بيستخدم لسكان الـ repositories عشان يكتشف secrets أو credentials.

تم بحمد الله

Contact

For questions, feedback, or support:

- X (Formly Twitter): @00xmora
- Linkedin: @00xmora