SINCRITY

Secure Coding -

Office of the Basic Education Commission

สารบัญ

SINQRITY

ความสำคัญของการพัฒนาแอปพลิเคชันอย่างปลอดภัย

การพัฒนาแอปพลิเคชันอย่างปลอดภัยเป็นสิ่งสำคัญอย่างยิ่งในยุคดิจิทัลปัจจุบัน โดยมีหลายเหตุผลที่เน้นความสำคัญของการมีแนวปฏิบัติที่ปลอดภัยในการพัฒนาแอปพลิเคชัน ดังนี้

ป้องกันการโจมตีทางไซเบอร์

เพิ่มความน่าเชื่อถือและความปลอดภัยขององค์กร

ปกป้องข้อมูลส่วนบุคคล

การปฏิบัติตามมาตรฐานและข้อกำหนดทางกฎหมาย

ลดค่าใช้จ่ายในระยะยาว

การส่งเสริมแนวปฏิบัติที่ดีในองค์กร

SINQRITY

ภัยเงียบของการใช้ AI ช่วยในการเขียนโค๊ด

ChatGPT

Gemini

Copilot

Claude

SINQRITY

ภัยที่อาจเกิดจากการใช้ AI

การรั่วไหลของข้อมูลที่เป็นความลับ (Leaking Sensitive Data)

นักพัฒนาอาจเผลอใส่ API Keys, รหัสผ่าน หรือข้อมูลโครงสร้างพื้นฐานขององค์กรลงใน Prompt เพื่อให้ AI ช่วยแก้ปัญหา ซึ่งข้อมูลเหล่านี้อาจถูกนำไปใช้เทรนโมเดลต่อและอาจหลุดไปสู่สาธารณะได้

SINQRITY

ภัยที่อาจเกิดจากการใช้ AI

ช่องโหว่ความปลอดภัยในโค้ด (Security Vulnerabilities)

AI มักจะแนะนำโค้ดที่ "ทำงานได้" แต่ไม่ได้ "ปลอดภัยที่สุด" เช่น โค้ดที่เสี่ยงต่อ SQL Injection, Cross-Site Scripting (XSS) หรือการไม่ทำ Input Validation

ภัยที่อาจเกิดจากการใช้ AI

การละเมิดลิขสิทธิ์ซอฟต์แวร์ (Intellectual Property & Licensing)

AI อาจนำโค้ดที่มีสัญญาอนุญาตแบบ Open Source ที่เข้มงวด เช่นห้ามนำมาใช้ในเชิงพาณิชย์ มาเป็นคำแนะนำ หากเรานำไปใช้ในโปรเจกต์เชิงพาณิชย์โดยไม่ตรวจสอบ อาจถูกฟ้องร้องภายหลังได้

SINORITY

ภัยที่อาจเกิดจากการใช้ AI

การเข้าใจผิดของ AI เรื่องแพ็กเกจ (AI Package Hallucination)

AI อาจแนะนำให้ใช้ Library หรือ Package ที่ไม่มีอยู่จริง แฮกเกอร์อาจสร้าง Package ปลอมในชื่อนั้นๆ ทิ้งไว้ (Supply Chain Attack) เมื่อนักพัฒนาสั่งติดตั้งตาม AI ก็จะได้รับมัลแวร์ทันที

ภัยที่อาจเกิดจากการใช้ AI

การขาดความเข้าใจในสิ่งที่เขียน (Skill Erosion)

หากนักพัฒนา Copy-Paste โค้ดโดยไม่พยายามทำความเข้าใจ เมื่อเกิด Bug ร้ายแรงในระบบ ทำให้หาจุดผิดเกิด bug และแก้ไขได้ยาก

SINQRITY

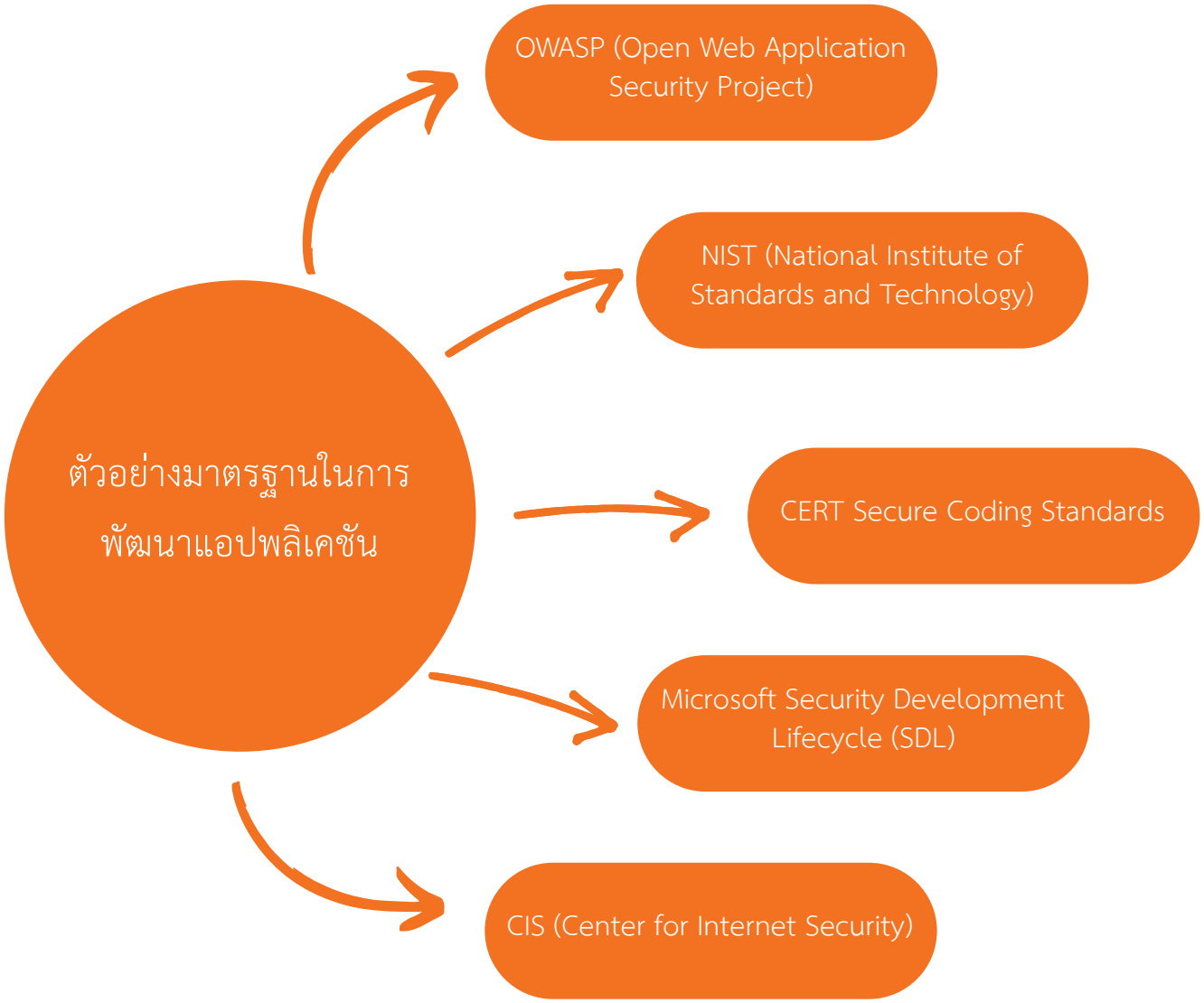
แนวทางการแก้ไข

- ทุกบรรทัดที่ AI เขียน ต้องถูก Review โดยมนุษย์ที่มีความเชี่ยวชาญเสมอ
- ใช้เครื่องมืออย่าง SAST (Static Application Security Testing) เพื่อสแกนโค้ดทุกครั้งก่อน Merge
- เวลาเราสั่ง AI เราควรระบุไปเลยว่า 'ให้เขียนโค้ดโดยคำนึงถึงความปลอดภัยระดับสูงสุด' หรือ 'ห้ามใช้ Library ภายนอกที่ไม่ได้มาตรฐาน’
- องค์กรต้องมี Guideline ที่ชัดเจนว่าข้อมูลไหนใส่ใน AI ได้ (Non-sensitive) และข้อมูลไหนห้ามใส่เด็ดขาด“

SINQRITY

มาตรฐานที่ช่วยในการพัฒนาแอปพลิเคชันอย่างปลอดภัย

การเขียนโค้ดอย่างปลอดภัยเป็นกระบวนการที่จำเป็นต้องปฏิบัติตามมาตรฐานและแนวทางที่ถูกต้องเพื่อป้องกันช่องโหว่และการโจมตีทางไซเบอร์ มีหลายมาตรฐานและแนวทางที่ช่วยให้การเขียนโค้ดมีความปลอดภัยยิ่งขึ้น



SINQRITY

OWASP TOP 10 (2025) คืออะไร

รายการช่องโหว่ด้านความปลอดภัยที่พบบ่อยและมีความสำคัญมากที่สุดในแอปพลิเคชันเว็บ จัดทำโดย Open Web Application Security Project (OWASP) ซึ่งเป็นองค์กรไม่แสวงหาผลกำไรที่มุ่งเน้นการปรับปรุงความปลอดภัยของซอฟต์แวร์ รายการนี้จะถูกอัปเดตเป็นระยะเพื่อสะท้อนถึงภัยคุกคามและช่องโหว่ที่พบบ่อยที่สุดในขณะนั้น

01
A1 - Broken Access Control

02
A2 - Security Misconfiguration

03
A3 - Software Supply Chain Failures

04
A4 - Cryptographic Failures

05
A5 - Injection

06
A6 - Insecure Design

07
A7 - Authentication Failures

08
A8 - Software or Data Integrity Failures

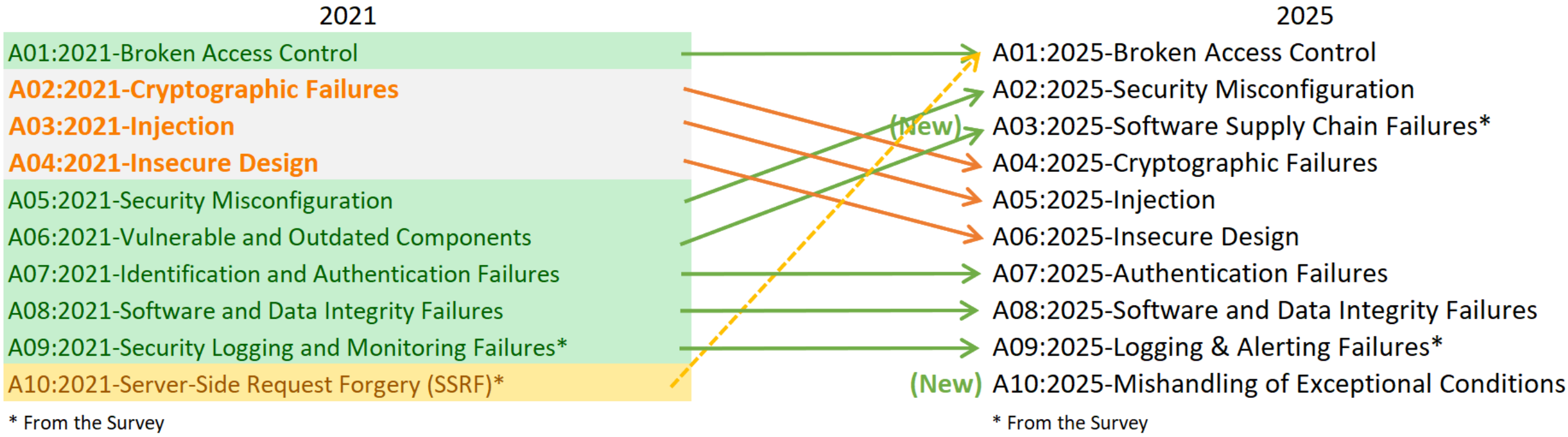
09
A9 - Security Logging & Alerting Failures

10
A10 - Mishandling of Exceptional Conditions

อ้างอิงตาม : https://owasp.org/Top10/2025/

SINQRITY

OWASP TOP 10 (2025) คืออะไร

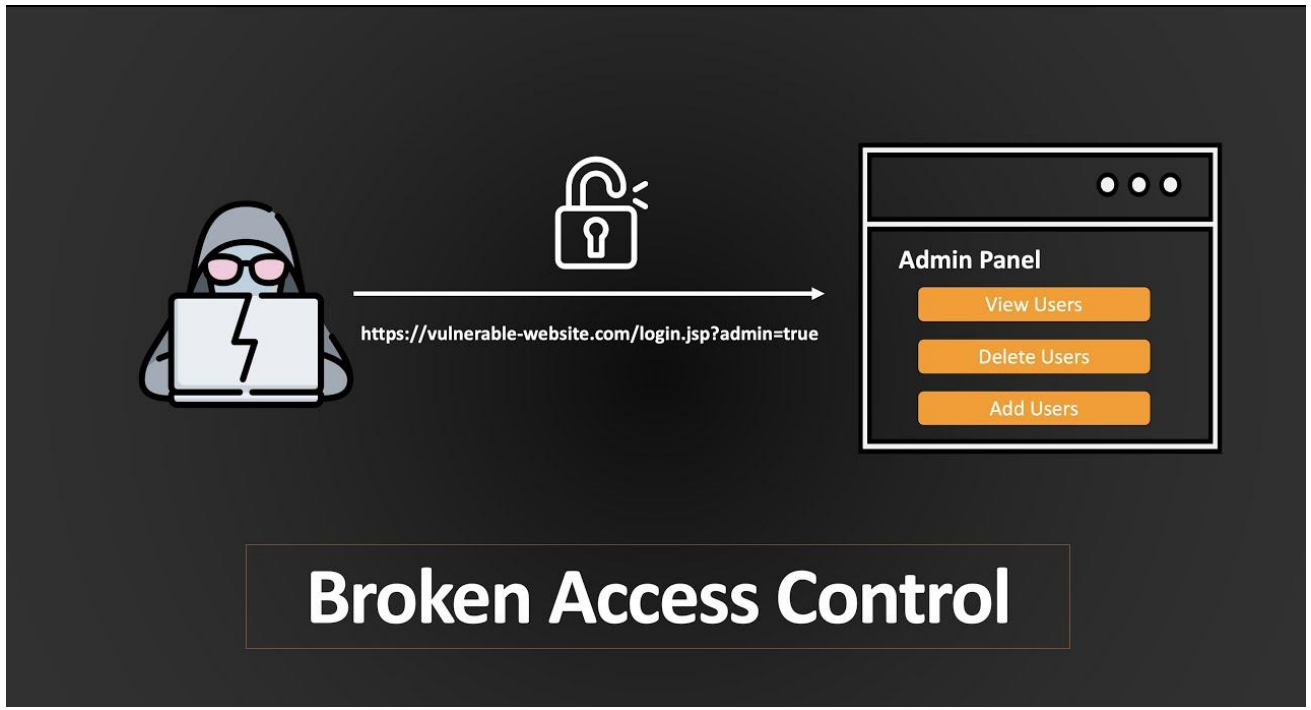


อ้างอิงตาม : https://owasp.org/Top10/2025/

SINQRITY

A1 - Broken Access Control (การควบคุมการเข้าถึงที่ไม่ถูกต้อง)

การที่ผู้ใช้สามารถเข้าถึงข้อมูลหรือฟังก์ชันที่ตนเองไม่มีสิทธิ์ ระบบควรจะมีการจำกัดสิทธิ์ (Authorization) เพื่อให้ User ทั่วไปเข้าได้เฉพาะข้อมูลของตนเอง และ Admin เท่านั้นที่เข้าหลังบ้านได้ แต่หากระบบมีช่องโหว่นี้ Hacker จะสามารถข้ามผ่านข้อจำกัดเหล่านั้นได้



อ้างอิงตาม : https://owasp.org/Top10/2025/A01_2025-Broken_Access_Control/

ตัวอย่างการโจมตี Scenario #1:

แอปพลิเคชันใช้ข้อมูลที่ยังไม่ได้รับการตรวจสอบในการเรียกใช้ SQL เพื่อเข้าถึงข้อมูลของบัญชี

```
// นำค่าที่รับมาจากผู้ใช้ไปใส่ในคำสั่ง SQL
statement.setString(1, requestedAccountId);
// ประมวลผลและดึงข้อมูลออกมา
ResultSet accountData = statement.executeQuery();
```

ผู้โจมตีเพียงแค่แก้ไขพารามิเตอร์ 'requestedAccountId' ในเบราว์เซอร์เพื่อส่งหมายเลขบัญชีที่ต้องการ หากไม่มีการตรวจสอบอย่างถูกต้อง ผู้โจมตีสามารถเข้าถึงข้อมูลบัญชีผู้ใช้อื่นๆ ได้

https://example.com/app/accountInfo?acct=2

SINQRITY

ตัวอย่างการโจมตี Scenario #1:

ผู้โจมตีเพียงแค่ทราบ URL ของแอดมิน จากนั้นใช้เบราว์เซอร์เข้าไปยัง URL เป้าหมาย ที่ปกติแล้วต้องใช้สิทธิ์ของแอดมินสำหรับการเข้า URL นั้นโดยตรง

https://example.com/app/getappInfo

https://example.com/app/admin_getappInfo

หากผู้ใช้ที่ไม่ได้รับการยืนยันสามารถเข้าถึงหน้าใดหน้าหนึ่งได้ นั่นถือว่าเป็นช่องโหว่
หากผู้ใช้ที่ไม่ใช่ผู้ดูแลระบบสามารถเข้าถึงหน้าผู้ดูแลระบบได้ นี่ก็เป็นช่องโหว่เช่นกัน

SINQRITY

https://0a7e00a9048c305b805d2b71008000dd.web-security-academy.net

WE LIKE TO
SHOP

Babbage Web Spray

$25.94

View details

WTF? - The adult party game

$86.08

View details

The Alternative Christmas Tree

$79.37

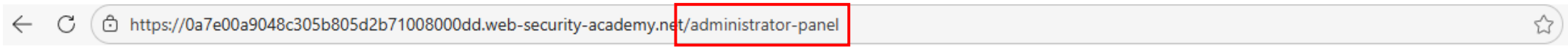
View details

Six Pack Beer Belt

$18.45

View details

SINORITY



Users

wiener - Delete
carlos - Delete

วิธีการแก้ไข และป้องกัน

ใช้หลักการ "**Deny by Default**" (ปฏิเสธไว้ก่อน)
ระบบควรจะ **ไม่อนุญาต** ให้ใครเข้าถึงอะไรเลย จนกว่าจะมีการระบุสิทธิ์ที่ชัดเจน

ตรวจสอบค่า parameter ที่ฝั่ง **Client** และ **Sever** ไม่ควรตรวจสอบแค่ฝั่งใดฝั่งเดียว

การออกแบบควรคำนึงถึงสิทธิ์ของผู้ใช้งานในแต่ละสิทธิ์ ให้ตรงตามการ
ทำงานของแอปพลิเคชันนั้นๆ

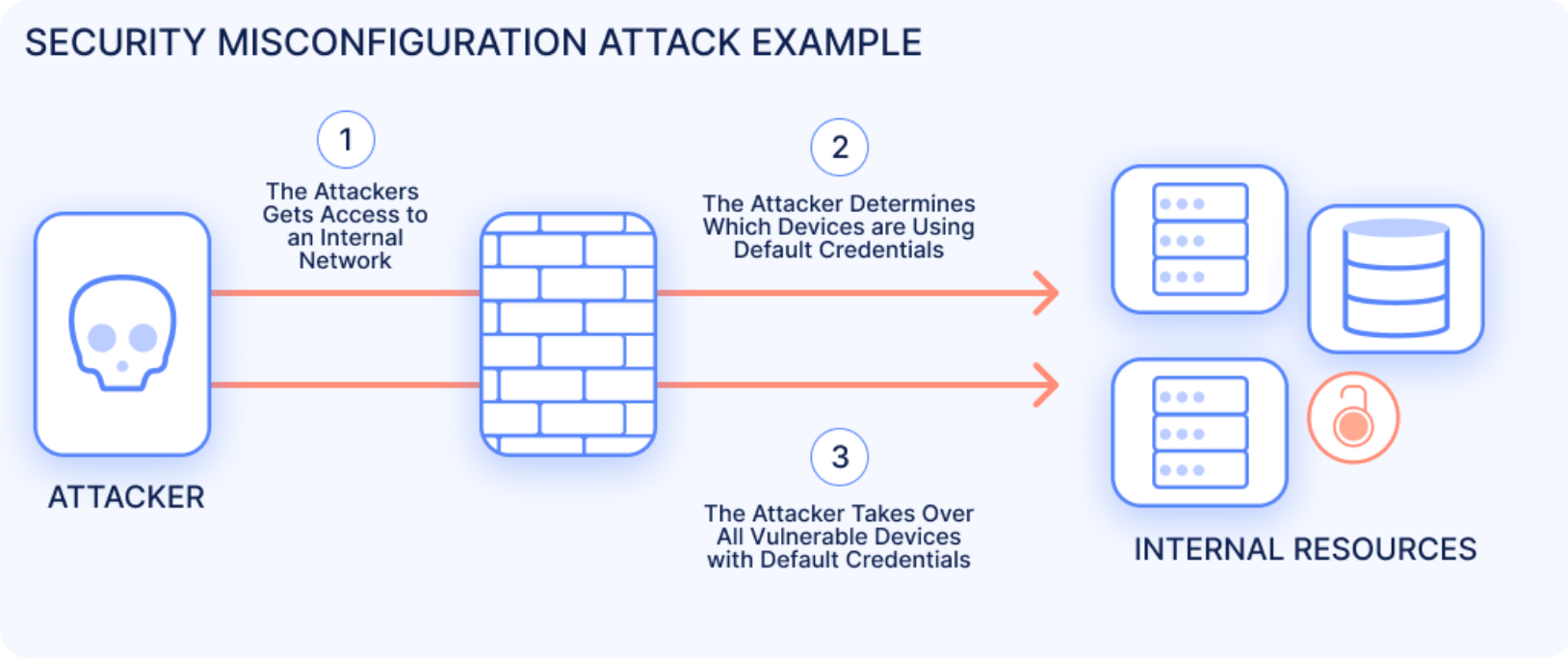
SINQRITY

A2 - Security Misconfiguration

(การตั้งค่าความปลอดภัยที่ผิดพลาด)

02

ช่องโหว่ที่เกิดจากการกำหนดค่าระบบหรือแอปพลิเคชันอย่างไม่ปลอดภัย
เช่น การใช้ค่าดั้งเดิมที่ไม่ปลอดภัย หรือการเปิดใช้งานฟังก์ชันที่ไม่จำเป็น



อ้างอิงตาม : https://owasp.org/Top10/2025/A02_2025-Security_Misconfiguration/

SINQRITY

ตัวอย่างการโจมตี

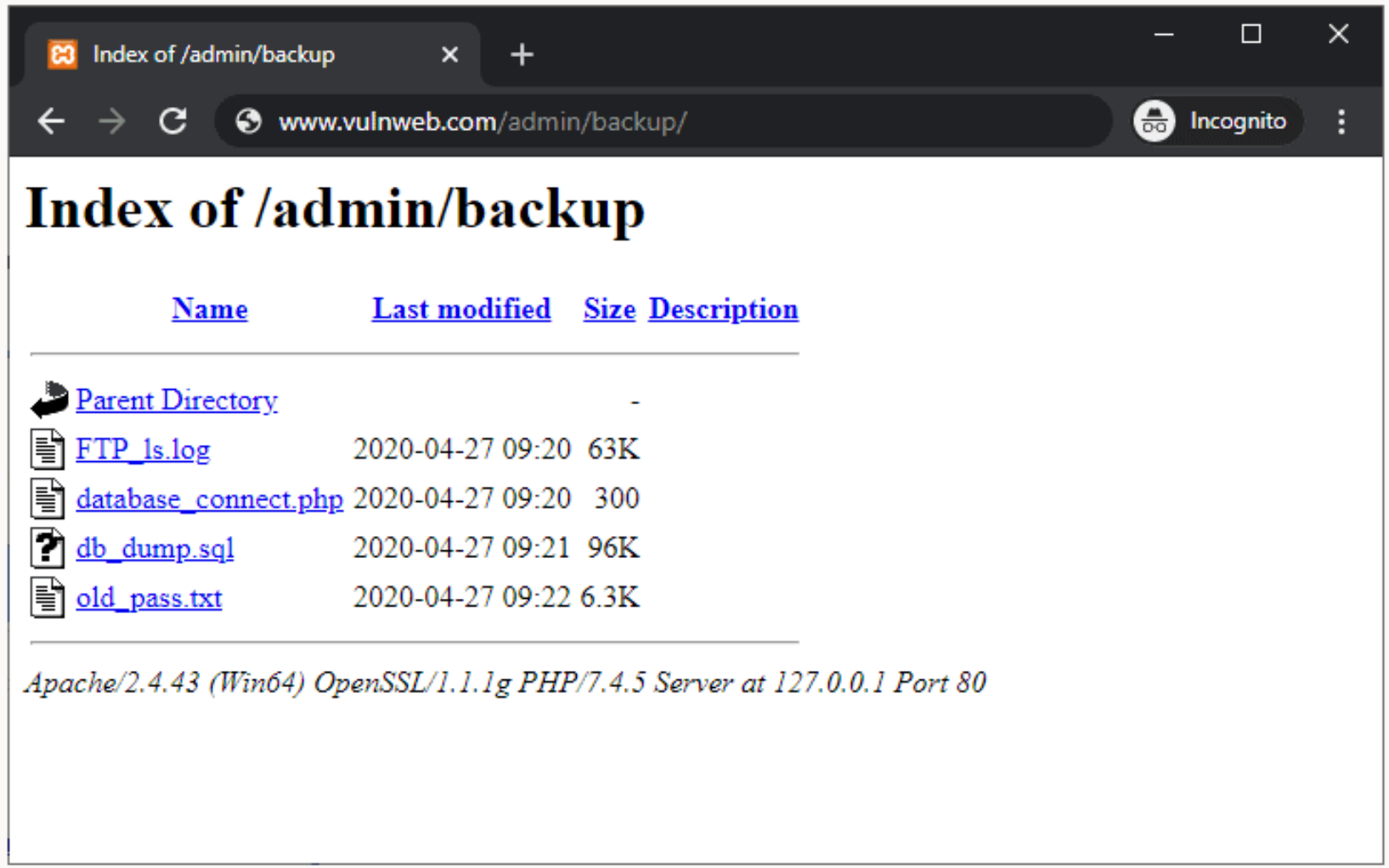
Scenario #1:

เซิร์ฟเวอร์มาพร้อมกับแอปพลิเคชันตั้งต้นโดยไม่ได้มีการลบแอปพลิเคชันที่ไม่จำเป็นออกจากเซิร์ฟเวอร์ แอปพลิเคชันที่ติดมาด้วยนั้น อาจช่องโหว่ด้านความปลอดภัยที่มักถูกผู้โจมตีใช้เพื่อโจมตีเซิร์ฟเวอร์ เช่น บัญชีเริ่มต้นไม่ได้ถูกเปลี่ยนแปลง ผู้โจมตีจะเข้าสู่ระบบด้วยรหัสผ่านเริ่มต้นและยึดครองความควบคุมในระบบไป

Scenario #2:

Directory Listing ถูกเปิดใช้งานบนเซิร์ฟเวอร์ ผู้โจมตีพบว่าสามารถเรียกดูรายการ Directory ได้ ทำให้ผู้โจมตีสามารถทำการดูไฟล์ต่างๆที่อยู่บนเซิร์ฟเวอร์ได้ ซึ่งอาจพบไฟล์ Source code หรือ Config ต่างๆ จากนั้นผู้โจมตีใช้ช่องโหว่ที่พบเพื่อในการโจมตีระบบต่อไป

SINQRITY


Index of /admin/backup
www.vulnweb.com/admin/backup/
Incognito
Index of /admin/backup
Name
Last modified
Size
Description
Parent Directory -
FTP_ls.log 2020-04-27 09:20 63K
database_connect.php 2020-04-27 09:20 300
db_dump.sql 2020-04-27 09:21 96K
old_pass.txt 2020-04-27 09:22 6.3K
Apache/2.4.43 (Win64) OpenSSL/1.1.1g PHP/7.4.5 Server at 127.0.0.1 Port 80

SINQRITY

วิธีการแก้ไข และป้องกัน

- สร้างมาตรฐานการตั้งค่าระบบที่ปลอดภัย และนำมาใช้กับทุกสภาพแวดล้อม (Dev, Test(Uat), Prod) ให้เหมือนกัน
- ลบซอฟต์แวร์, ตัวอย่างแอปพลิเคชัน, หน้าเว็บทดสอบ, และบัญชีผู้ใช้ที่ไม่จำเป็นออกให้หมด
- ใช้เครื่องมือสแกนช่องโหว่อัตโนมัติเพื่อตรวจสอบการตั้งค่าที่ผิดพลาดเป็นระยะ

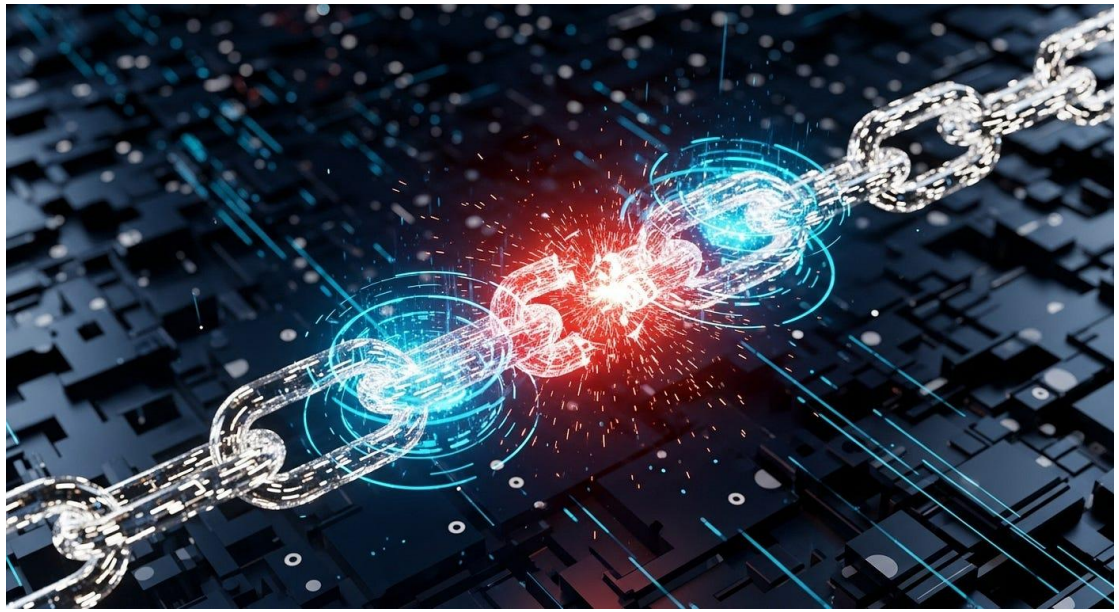
SINQRITY

03

A3 - Software Supply Chain Failures

(ความล้มเหลวของซอฟต์แวร์นำมาใช้)

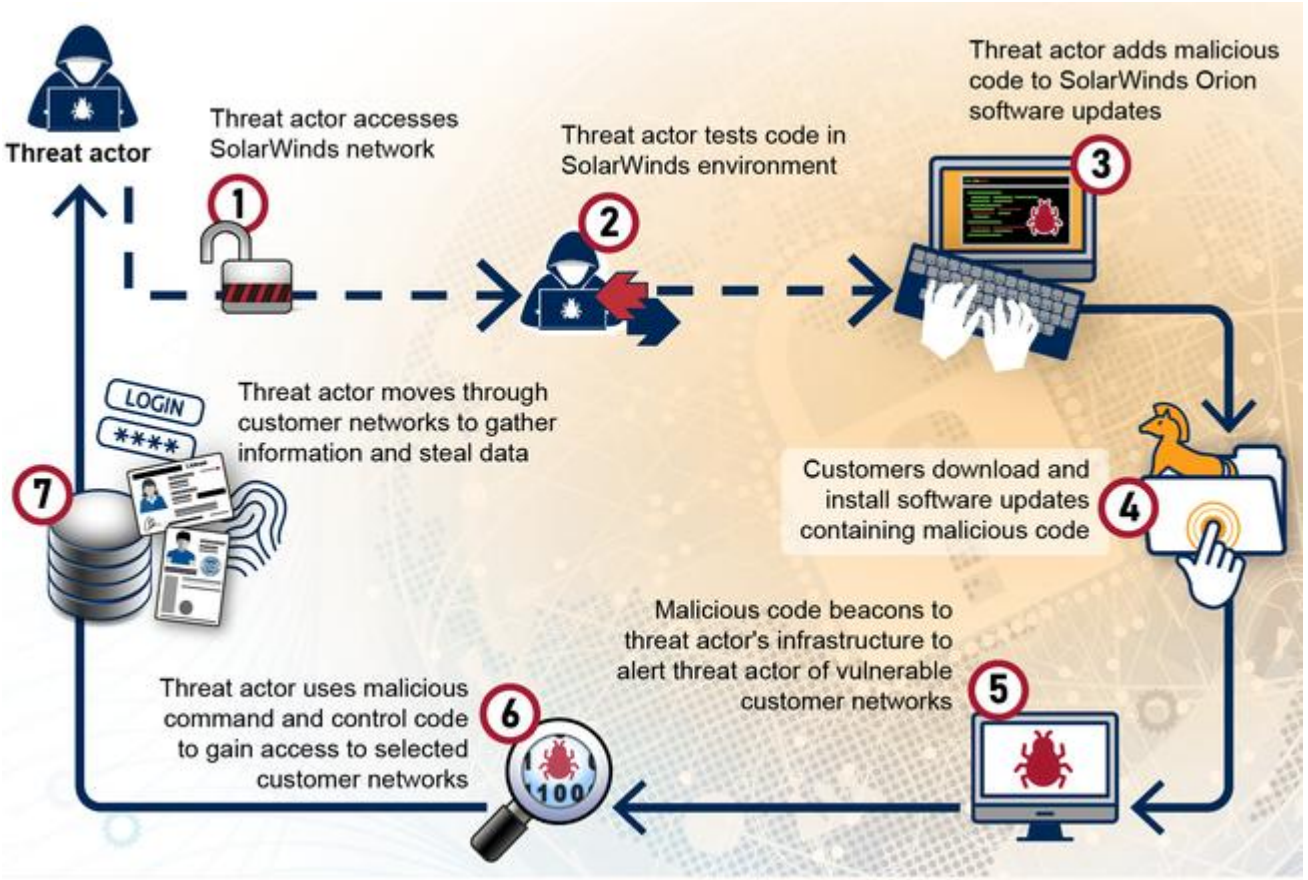
เป็นการโจมตีที่ไม่ได้มุ่งเป้าไปที่ซอฟต์แวร์ตัวหลักโดยตรง แต่โจมตีจาก "แหล่งที่มา" หรือ "กระบวนการสร้าง" แทน ทำให้ซอฟต์แวร์ที่ผู้ใช้ได้รับไปนั้นไม่ปลอดภัยตั้งแต่แรก เช่น การใช้ฟังก์ชันที่ล้าสมัย, การใช้ซอฟต์แวร์ third- party ที่มีช่องโหว่



อ้างอิงตาม : https://owasp.org/Top10/2025/A03_2025-Software_Supply_Chain_Failures/

SINQRITY

ตัวอย่างการโจมตี



Vender ที่เชื่อถือถูกโจมตีด้วยมัลแวร์ เมื่อใช้ซอฟต์แวร์ของ Vender ทำให้เครื่องของคุณถูกโจมตีไปด้วย

ตัวอย่าง

- SolarWinds ในปี 2019
  - ผู้โจมตีได้เจาะเข้าสู่ระบบภายในของ SolarWinds และฝังโค้ดมัลแวร์ เข้าไปในไฟล์ไบนารีของซอฟต์แวร์
  - เมื่อ SolarWinds ปล่อยอัปเดตซอฟต์แวร์ ที่ถูกฝังโค้ต เมื่อลูกค้าทำการอัปเดตซอฟต์แวร์ที่ถูกฝังโค้ดมัลแวร์ ทำให้เครื่องของลูกค้าจะถูกติดตั้งมัลแวร์โดยไม่รู้ตัวSolarWinds ประมาณ 18,000 องค์กรได้รับผลกระทบ

SINQRITY

ตัวอย่างการโจมตี

- Bybit (เดือนกุมภาพันธ์ 2025)
  - ผู้โจมตีได้มุ่งเป้าโจมตีไปที่ Safe{Wallet} ซึ่งเป็นโซลูชัน Third-party ที่ Bybit ใช้
  - โดยผู้โจมตีสามารถ bypass การตรวจสอบทำให้สามารถทำการโอนย้ายเงินจากระบบออกไปได้
  - เหตุการณ์นี้ทำให้สูญเสียเงินไปประมาณ 1.5 พันล้านดอลลาร์ส

วิธีการแก้ไข และป้องกัน

- ลบ dependencies ที่ไม่ได้ใช้, องค์ประกอบ, ไฟล์, และเอกสารที่ไม่จำเป็นออก
- ทำการตรวจสอบเวอร์ชั่นที่ใช้ ทั้งฝั่งไคลเอนต์และเซิร์ฟเวอร์ (เช่น เฟรมเวิร์ก, ไลบรารี) โดยใช้เครื่องมือ เช่น versions, OWASP Dependency Check, retire.js ฯลฯ
- เฝ้าระวังแหล่งข้อมูลอย่างต่อเนื่อง เช่น Common Vulnerability and Exposures (CVE) และ National Vulnerability Database (NVD) สำหรับช่องโหว่ในส่วนประกอบที่คุณใช้ ใช้เครื่องมือวิเคราะห์ส่วนประกอบซอฟต์แวร์ (Software Composition Analysis)

SINQRITY

04

A4 - Cryptographic Failures (ความผิดพลาดของการตั้งค่าการเข้ารหัส)

เป็นช่องโหว่ที่เกิดจากการใช้การเข้ารหัสที่ไม่เหมาะสม เช่น การใช้การเข้ารหัสที่ไม่ปลอดภัยหรือการจัดการกุญแจเข้ารหัส(Encryption key) ที่ไม่ดี

อ้างอิงตาม : https://owasp.org/Top10/2025/A04_2025-Cryptographic_Failures/

ตัวอย่างการโจมตี

Scenario #1:

แอปพลิเคชันไม่ได้บังคับใช้ HTTPS ทำให้เมื่อผู้ใช้ล็อกอินผ่าน Wi-Fi สาธารณะ แฮ็กเกอร์สามารถดักจับ Cookie และสวมรอยเป็นผู้ใช้คนนั้นได้ทันที

Scenario #2:

ฐานข้อมูลเก็บเลขบัตรเครดิตโดยใช้การเข้ารหัสแบบ AES-128 แต่ตัว Encryption Key กลับถูกเก็บไว้ในไฟล์ config.js บนเซิร์ฟเวอร์เดียวกัน หากแฮ็กเกอร์เข้าถึงไฟล์ได้ ข้อมูลบัตรทั้งหมดก็จะไม่เป็นความลับอีกต่อไป

SINQRITY

วิธีการแก้ไข และป้องกัน

- ระบุว่าข้อมูลไหน "อ่อนไหว" (เช่น เลขบัตรประชาชน, ข้อมูลสุขภาพ) เพื่อรับรองว่าข้อมูลเหล่านั้นจะได้รับการป้องกันสูงสุด
- บังคับใช้ HSTS (HTTP Strict Transport Security) เพื่อให้มั่นใจว่าการเชื่อมต่อทั้งหมดถูกเข้ารหัสเสมอ
- จัดเก็บรหัสผ่านโดยใช้ฟังก์ชันแฮชที่แข็งแกร่งและมีการเพิ่มค่าซอลต์ พร้อมด้วย delay factor เช่น Argon2, scrypt, bcrypt หรือ PBKDF2
- ปิดการใช้งาน TLS 1.0/1.1 และหันมาใช้ TLS 1.2 หรือ 1.3 เท่านั้น
- อย่าใช้โปรโตคอลเก่า เช่น FTP และ SMTP สำหรับการขนส่งข้อมูลที่มีความสำคัญ

SINQRITY

A5 – Injection (การแทรกคำสั่ง)

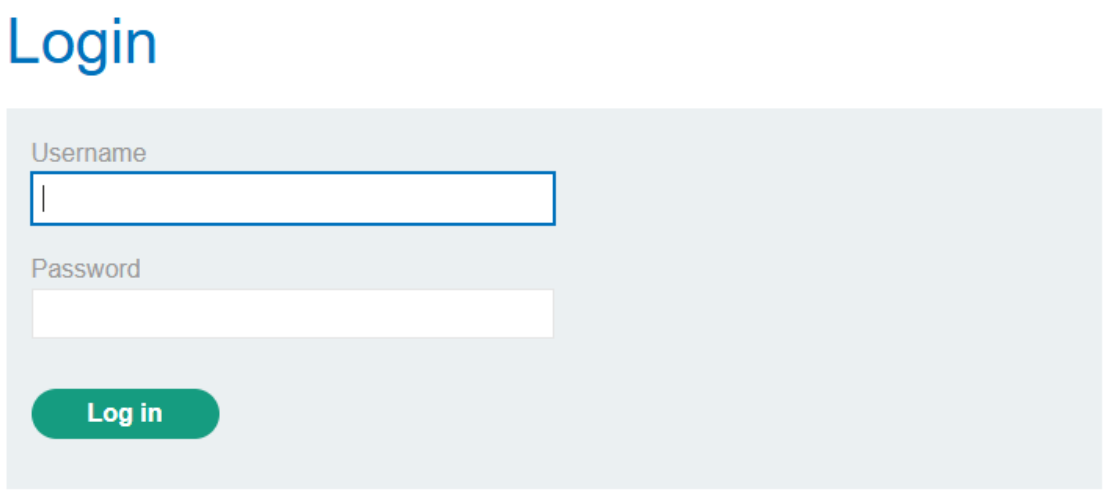
05

ช่องโหว่ที่เกิดจากการนำข้อมูลที่ไม่น่าเชื่อถือมาใช้ในคำสั่งเช่น SQL Injection, Command Injection ทำให้เซิร์ฟเวอร์มีการประมวลผลคำสั่งอันตรายที่ไม่ได้เป็นจุดมุ่งหมายของแอปพลิเคชัน

อ้างอิงตาม : https://owasp.org/Top10/2025/A05_2025-Injection/

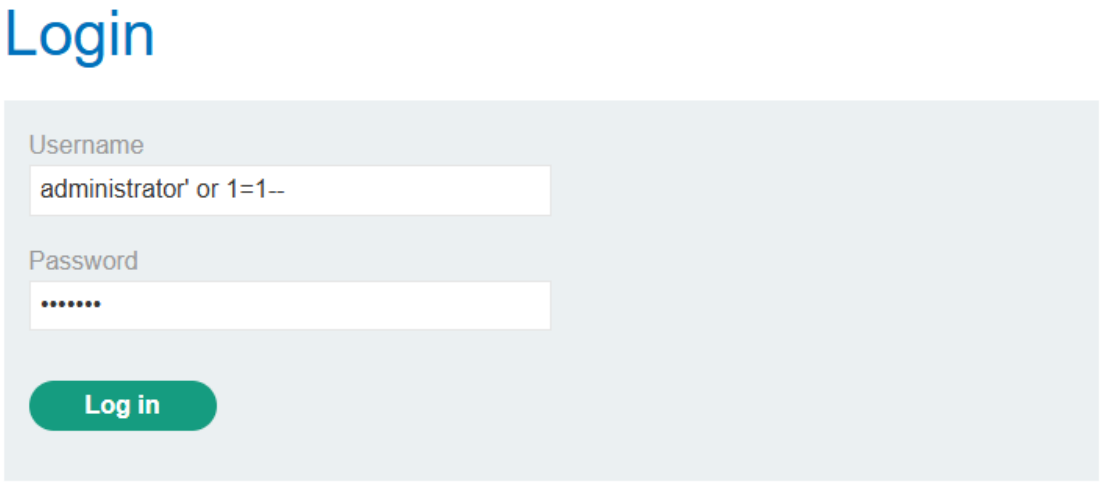
SINCRITY

ตัวอย่างการโจมตี



แอปพลิเคชันใช้ข้อมูลที่ไม่น่าเชื่อถือในการสร้างคำสั่ง SQL ที่มีช่องโหว่ดังต่อไปนี้:

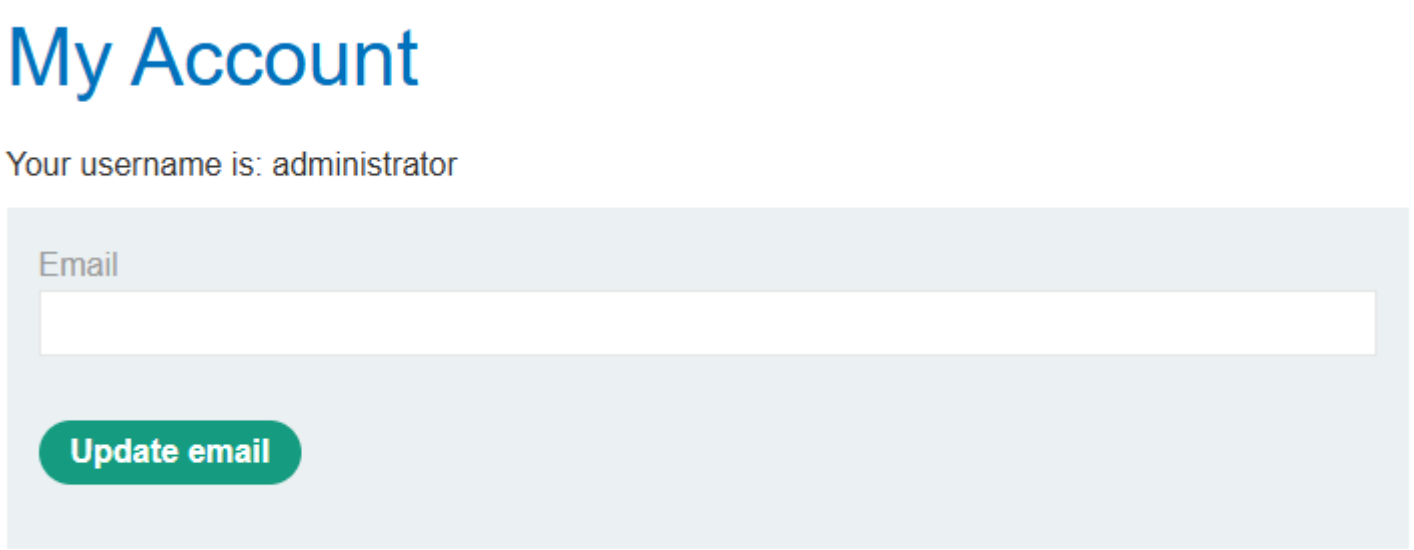
SELECT name FROM user WHERE name='admin' AND password='xDK9&GoP1'



SELECT name FROM user WHERE name='administrator' or 1=1 --' AND passwd='xDK9&GoP1'

SINORITY

ตัวอย่างการโจมตี



ตัวอย่างการโจมตี

Scenario #2:

การโจมตีแบบ SQL Injection โดยการปรับเปลี่ยนค่า id ในเบราว์เซอร์เพื่อส่งค่า ' UNION SLEEP(10);-- นั้นสามารถทำให้เกิดการโจมตีและส่งผลกระทบต่อฐานข้อมูลและแอปพลิเคชันได้อย่างมาก ดังตัวอย่าง:

```
http://example.com/app/accountView?id=' UNION SELECT SLEEP(10);--
```

การโจมตีแบบ SQL Injection โดยการปรับเปลี่ยนค่า id สามารถเปลี่ยนความหมายของคำสั่ง SQL ให้ดึงข้อมูลทั้งหมดจากตารางหรือแม้กระทั่งแก้ไขหรือลบข้อมูลได้ นอกจากนี้ยังสามารถเรียกใช้ Stored Procedures ที่เป็นอันตรายได้

SINQRITY

วิธีการแก้ไข และป้องกัน

- วิธีที่แนะนำในการป้องกัน SQL Injection คือการใช้ API แทนการใช้ Parameter โดยตรง จัดหาการเชื่อมต่อแบบพารามิเตอร์ เพื่อให้มั่นใจว่าการจัดการข้อมูลเป็นไปอย่างปลอดภัยและมีประสิทธิภาพ
- ใช้การตรวจสอบข้อมูล input ฝั่งเซิร์ฟเวอร์ แทนที่การตรวจสอบที่ client (web-browser)
- สำหรับคำสั่งแบบไดนามิกที่เหลืออยู่ ให้ทำการสร้าง whitelist อักขระพิเศษโดยใช้การสร้างชุด whitelist ที่เฉพาะเจาะจงสำหรับตัวแปลนั้นๆ
- ใช้ LIMIT และการควบคุม SQL อื่น ๆ ภายในคำสั่งเพื่อป้องกันการเปิดเผยข้อมูลจำนวนมากในกรณีที่เกิด SQL Injection

SINQRITY

06

A6 - Insecure Design (การออกแบบที่ไม่ปลอดภัย)

ช่องโหว่ที่เกิดจากการออกแบบแอปพลิเคชันที่ไม่คำนึงถึงความปลอดภัย เช่น ขาดการพิจารณาด้านความปลอดภัยตั้งแต่ขั้นตอนการออกแบบ ไม่ได้มีการป้องกันรูปแบบการโจมตีต่างๆ ที่อาจจะเกิดขึ้นได้กับตัวแอปพลิเคชัน

อ้างอิงตาม : https://owasp.org/Top10/2025/A06_2025-Insecure_Design/

SINQRITY

ตัวอย่างการโจมตี

Scenario #1:

การใช้ “questions and answers” ในการยืนยันตัวตน เป็นการยืนยันตัวตนที่ไม่ปลอดภัย และไม่สามารถยืนยันความเป็นเจ้าของของบัญชีนั้นได้จริง

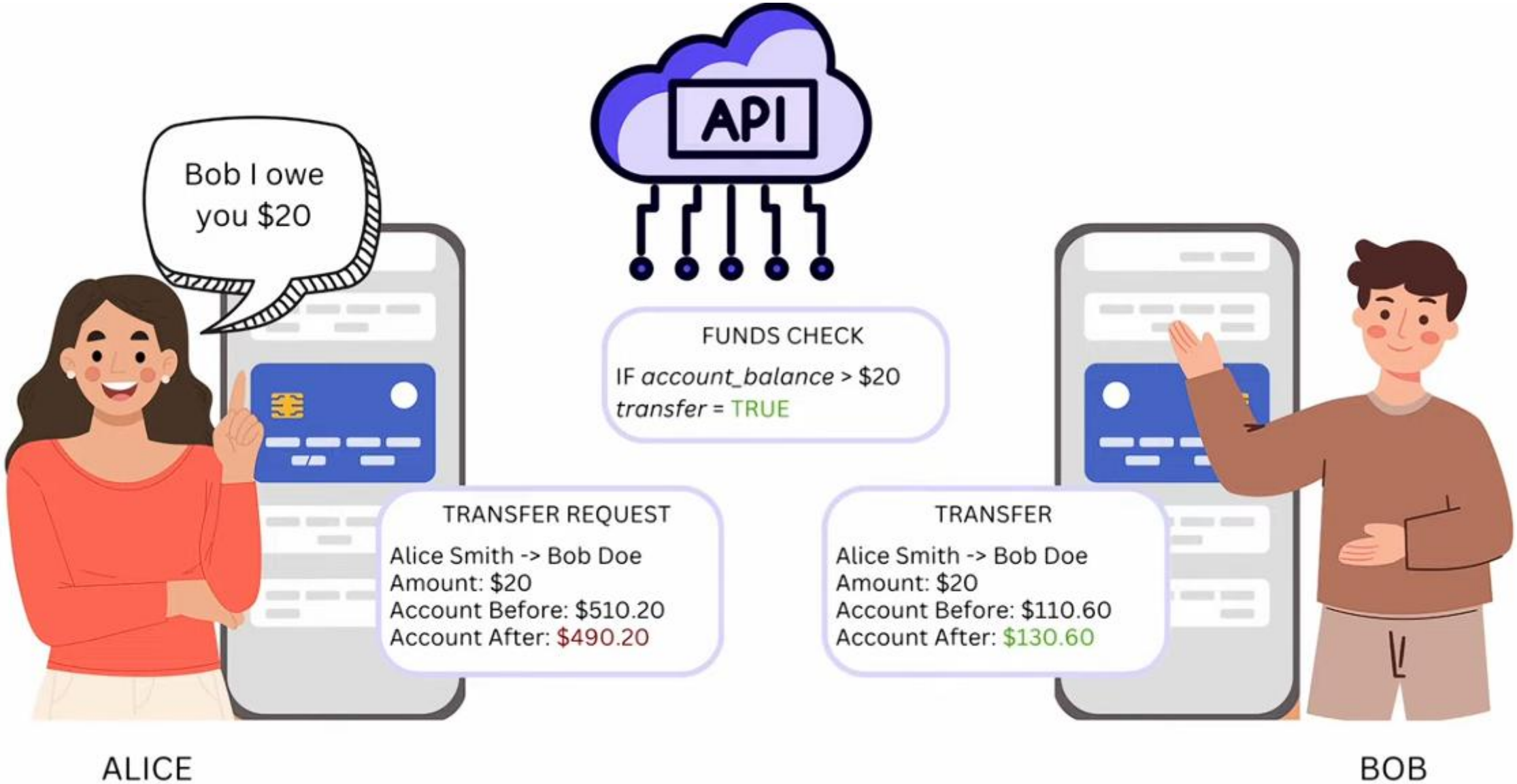
Scenario #2:

โรงภาพยนตร์โปรโมชั่นมีส่วนลดสำหรับการจองแบบกลุ่มและกำหนดให้มีผู้เข้าชมสูงสุดสิบห้าคนโดยไม่ต้องมัดจำ ผู้โจมตีสามารถหาช่องโหว่จากออกแบบที่ผิดพลาดและพบว่าสามารถจองที่นั่งหกร้อยที่นั่งและทุกโรงภาพยนตร์ในครั้งเดียวด้วยการร้องขอเพียงไม่กี่ครั้ง ซึ่งทำให้เกิดการสูญเสียรายได้อย่างมหาศาล

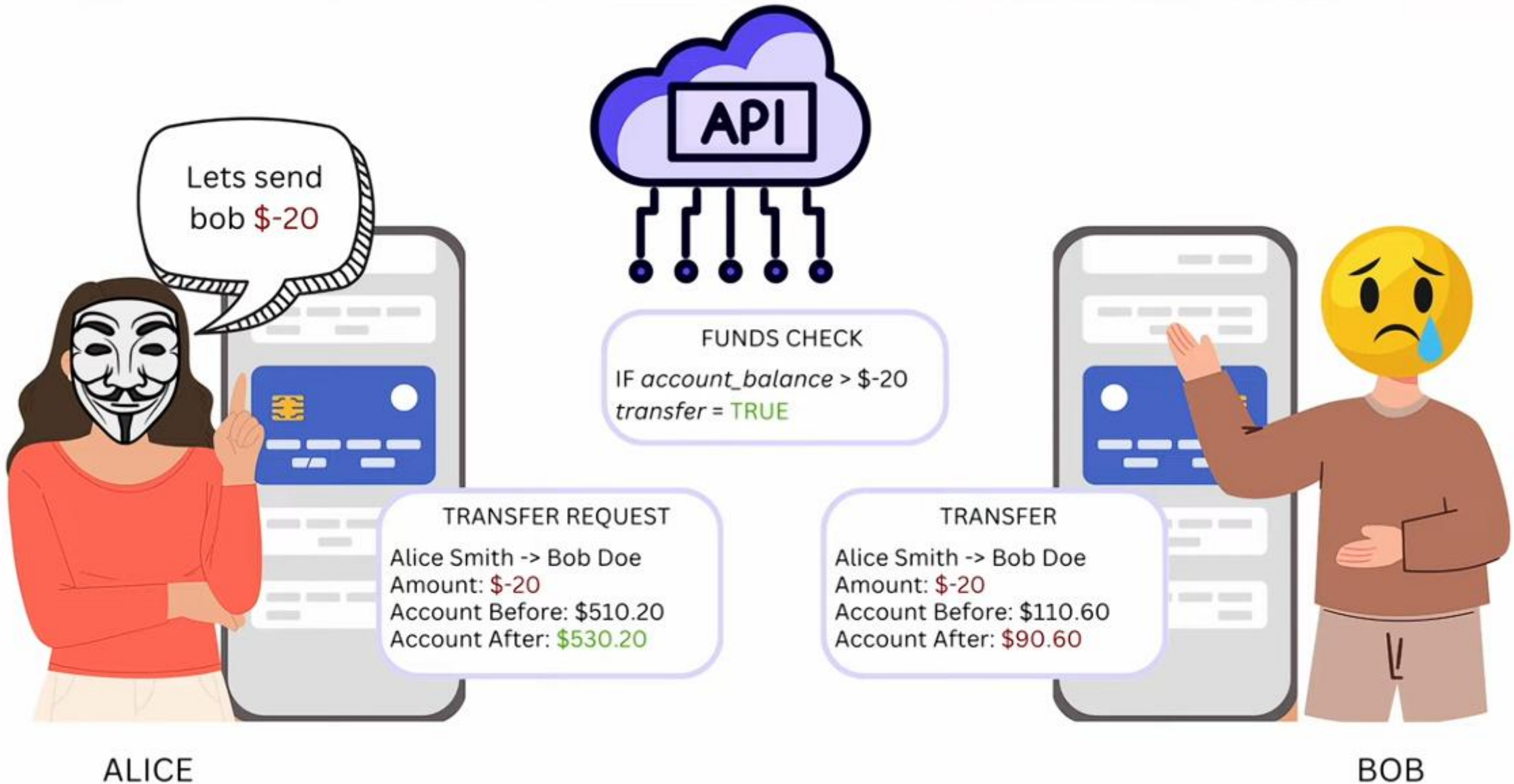
Scenario #3:

แอปธนาคารยอมให้ผู้ใช้ใส่จำนวนเงินที่มีค่า ติดลบ เข้าไปได้ทำให้จากจะโอนเงินไปหาเป้าหมาย กลายเป็นไปดึงเงินจากเป้าหมายมาแทน

SINORITY


API
Bob I owe you $20
FUNDS CHECK
IF account_balance > $20
transfer = TRUE
TRANSFER REQUEST
Alice Smith -> Bob Doe
Amount: $20
Account Before: $510.20
Account After: $490.20
TRANSFER
Alice Smith -> Bob Doe
Amount: $20
Account Before: $110.60
Account After: $130.60
ALICE
BOB

SINQRITY


API
Lets send bob $-20
FUNDS CHECK
IF account_balance > $-20
transfer = TRUE
TRANSFER REQUEST
Alice Smith -> Bob Doe
Amount: $-20
Account Before: $510.20
Account After: $530.20
TRANSFER
Alice Smith -> Bob Doe
Amount: $-20
Account Before: $110.60
Account After: $90.60
ALICE
BOB

SINQRITY

วิธีการแก้ไข และป้องกัน

- การเชื่อมต่อด้านความปลอดภัยและการควบคุมความปลอดภัยเข้ากับ User Stories เป็นสิ่งสำคัญในการพัฒนาซอฟต์แวร์ที่ปลอดภัย
- การตรวจสอบความน่าเชื่อถือในแต่ละชั้นของแอปพลิเคชัน (จาก frontend ถึง backend)
- จำกัดการใช้งาน Resource โดยผู้ใช้งานหรือเซอร์วิสของแอปพลิเคชัน
- ดำเนินการจัดตั้งการทำ secure development lifecycle กับผู้เชี่ยวชาญด้าน ความปลอดภัยด้านแอปพลิเคชันในการประเมินความปลอดภัยในการออกแบบแอปพลิเคชัน

SINQRITY

07

A7 - Authentication Failures

(การระบุตัวตนและการตรวจสอบสิทธิ์ที่ล้มเหลว)

ช่องโหว่ที่เกิดจากการจัดการการระบุตัวตนและการตรวจสอบสิทธิ์ที่ไม่ปลอดภัย เช่น การใช้รหัสผ่านท่าคาดเดาง่าย หรือการจัดการเซสชันที่ไม่ปลอดภัย

อ้างอิงตาม : https://owasp.org/Top10/2025/A07_2025-Authentication_Failures/

SINORITY

ตัวอย่างการโจมตี

Scenario #1:

การใช้ password ที่คาดเดาง่าย ไม่มีการเปลี่ยน password เป็นประจำ หรือไม่มี policy ในการตั้ง password มักจะถูกโจมตีได้ง่าย เนื่องจากสามารถคาดเดา password ได้ง่าย

Scenario #2:

การตั้งค่าเวลาหมดอายุของเซสชั่นในแอปพลิเคชั่นไม่ถูกต้อง ผู้ที่ใช้คอมพิวเตอร์สาธารณะเพื่อเข้าถึงแอปพลิเคชั่น แทนที่จะเลือก "ออกจากระบบ" ผู้ใช้จะปิดแท็บบราวเซอร์และเดินออกไป หากมีผู้ไม่ประสงค์ดีเข้ามาใช้คอมพิวเตอร์สาธารณะเครื่องนั้นต่อ สามารถใช้แอปพลิเคชันที่เซสชั่นยังไม่หมดอายุของผู้ใช้งานได้

SINQRITY

วิธีการแก้ไข และป้องกัน

- ควรใช้การตรวจสอบบัญชีแบบหลายปัจจัย (multi-factor authentication - MFA) เพื่อป้องกันการโจมตีประเภท Brute force และการนำข้อมูลเข้ารหัสที่ถูกขโมยไปมาใช้ในการเข้าระบบ
- ห้ามแชร์ Password กับผู้อื่น โดยเฉพาะสำหรับผู้ใช้แอดมิน
- จำกัดหรือเพิ่มระยะเวลาในการพยายามเข้าสู่ระบบ ในกรณีที่เข้าระบบล้มเหลว

SINQRITY

08

A8 - Software and Data Integrity Failures

(การล้มเหลวของความสมบูรณ์ของซอฟต์แวร์และข้อมูล)

ช่องโหว่ที่เกิดจากการขาดการตรวจสอบความสมบูรณ์ของซอฟต์แวร์และข้อมูล เช่น การอัปเดตซอฟต์แวร์จากแหล่งที่ไม่เชื่อถือ ทำให้ในโค๊ดที่อัปเดตนั้น มีการฝังโค๊ดอันตรายที่ทำให้ผู้ไม่ประสงค์ดีสามารถก่อความเสียหายต่อระบบได้

อ้างอิงตาม : https://owasp.org/Top10/2025/A08_2025-Software_or_Data_Integrity_Failures/

SINQRITY

ตัวอย่างการโจมตี

Scenario #1:

อุปกรณ์อย่างเราเตอร์หรือกล่องทีวี มักขาดระบบตรวจสอบความปลอดภัยขณะอัปเดต (Unsigned Firmware) ทำให้ผู้บุกรุกสามารถส่งไฟล์ปลอมเข้าไปควบคุมเครื่องได้ง่าย ซึ่งปัญหานี้แก้ไขยาก เพราะต้องรออัปเดตใหญ่ในอนาคตหรือรอจนกว่าคนจะเลิกใช้เครื่องรุ่นนั้นไปเอง

Scenario #2:

ผู้โจมตีเข้าถึงระบบ Build System (เช่น Jenkins หรือ GitHub Actions) และแอบแก้ไขสคริปต์ที่ใช้ในการ Compile ซอฟต์แวร์ แม้โค้ดต้นฉบับจะปลอดภัย แต่ซอฟต์แวร์ตัวสุดท้าย (Artifact) ที่ถูกส่งออกไปหาลูกค้ากลับมี Backdoor ฝังอยู่ เพราะไม่มีการตรวจสอบความถูกต้องของไฟล์ในทุกขั้นตอนของ Pipeline

SINORITY

วิธีการแก้ไข และป้องกัน

- ใช้ digital signatures ในการยืนยันว่า ซอฟต์แวร์หรือข้อมูลมาจากแหล่งที่ถูกต้อง และไม่ได้ถูกแก้ไข
- ตรวจสอบให้แน่ใจว่า libraries และ dependencies เช่น npm หรือ Maven ที่ใช้งานมาจากแหล่งที่เชื่อถือได้
- ตรวจสอบให้แน่ใจว่ามีกระบวนการตรวจสอบ (Review Process) สำหรับการเปลี่ยนแปลงโค้ดและการตั้งค่า เพื่อลดโอกาสที่โค้ดหรือการตั้งค่าที่เป็นอันตรายจะถูกนำเข้าสู่ซอฟต์แวร์ของคุณ

SINQRITY

A09 - Security Logging & Alerting Failures

(การบันทึกและการตรวจสอบความปลอดภัยที่ล้มเหลว)

09

ช่องโหว่ที่เกิดจากการขาดการบันทึกและการตรวจสอบกิจกรรมที่สำคัญ เช่น
การไม่มีการบันทึกเหตุการณ์ที่สำคัญ (Logging)
หรือการไม่ตรวจสอบการแจ้งเตือนความปลอดภัย (Alert)

อ้างอิงตาม : https://owasp.org/Top10/2025/A09_2025-Security_Logging_and_Alerting_Failures/

SINQRITY

ตัวอย่างการโจมตี

Scenario #1:

ระบบบันทึกเฉพาะการเข้าสู่ระบบที่สำเร็จ แต่ ไม่บันทึก การเข้าสู่ระบบที่ล้มเหลว (Failed Logins ทำให้ผู้โจมตีสามารถใช้วิธี Brute Force หรือ Credential Stuffing ลองรหัสผ่านไปเรื่อยๆ ได้หลายล้านครั้งโดยที่ทีม Security ไม่เห็นความผิดปกติใน Log เลย

Scenario #2:

สิทธิ์ในการเข้าถึงไฟล์ Log ไม่ได้รับการป้องกันที่ดีพอ เมื่อแฮ็กเกอร์เจาะระบบได้สำเร็จ สิ่งแรกที่เขาจะทำคือ "ลบประวัติ" เพื่อปกปิดร่องรอย หากเราไม่มีการทำ Log Integrity หรือส่ง Log ไปยัง Storage ที่เขียนได้อย่างเดียว เราจะไม่สามารถทำ Digital Forensics เพื่อหาต้นตอได้เลย

SINORITY

วิธีการแก้ไข และป้องกัน

- ตรวจสอบให้แน่ใจว่าการเข้าสู่ระบบ การเข้าถึงข้อมูล และรวมถึง Error ที่เกิดขึ้น สามารถตรวจสอบข้อมูลอินพุตฝั่งเซิร์ฟเวอร์ทั้งหมดสามารถบันทึกได้พร้อมกับบริบทของผู้ใช้ที่เพียงพอเพื่อระบุบัญชีที่น่าสงสัยหรือเป็นอันตราย และเก็บรักษาข้อมูลนี้ไว้นานพอเพื่อให้สามารถวิเคราะห์ทางนิติเวชในภายหลังได้

- ตรวจสอบให้แน่ใจว่าการบันทึกข้อมูลถูกสร้างในรูปแบบที่โซลูชันการจัดการ Log สามารถนำไปใช้ได้อย่างง่ายสำหรับอุปกรณ์ที่ใช้ในการวิเคราะห์ข้อมูล

- ตรวจสอบให้แน่ใจว่าข้อมูล Log ได้มีการเข้ารหัสอย่างถูกต้อง เพื่อป้องกันไม่ให้มีการโจมตีระบบเก็บข้อมูลหรือระบบที่ใช้ในการวิเคราะห์ข้อมูล

SINQRITY

10

A10 - Mishandling of Exceptional Conditions (การรับมือความผิดปกติที่ผิดพลาด)

ข้อบกพร่องในการป้องกัน, ตรวจจับ, และตอบสนองต่อสถานการณ์ที่ไม่ปกติหรือไม่คาดคิดในระบบ ซึ่งอาจนำไปสู่ช่องโหว่ด้านความปลอดภัย, ระบบล่ม, หรือพฤติกรรมที่ไม่พึงประสงค์

อ้างอิงตาม : https://owasp.org/Top10/2025/A10_2025-Mishandling_of_Exceptional_Conditions/

SINQRITY

ตัวอย่างการโจมตี

Scenario #1:

เมื่อระบบเกิด Error (เช่น Database เชื่อมต่อไม่ได้ หรือ Query ผิดพลาด) หน้าจอแสดงผลกลับ Stack Trace, ชื่อ Table, เวอร์ชันของซอฟต์แวร์ หรือแม้แต่ File Path ในเซิร์ฟเวอร์ออกมาให้ผู้ใช้เห็น ผู้โจมตสามารถใช้ข้อมูลดังกล่าวในการโจมตีได้

Scenario #2:

ระบบมีการเช็คสิทธิ์ (Authorization) ก่อนเข้าถึงข้อมูล แต่เกิด Exception ในระหว่างกระบวนการเช็คสิทธิ์นั้นพอดี หากโค้ดเขียนไว้ไม่รัดกุม ระบบอาจจะข้ามขั้นตอนการเช็คไปและอนุญาตให้ผู้ใช้เข้าถึงข้อมูลได้เลย

Scenario #3:

แอปพลิเคชันมีการตั้งให้ดักการอัปโหลดไฟล์ แต่ไม่ปล่อยไฟล์ที่ถูกดักไว้มีมากเกินไปทำให้ทรัพยากรถูกใช้ไปจนหมด ทำให้เกิด Denial of Service - DoS

SINQRITY

วิธีการแก้ไข และป้องกัน

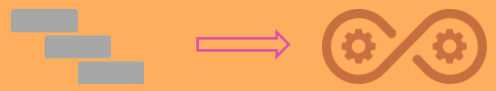
- ใช้ Rate Limiting, Resource Quotas เพื่อป้องกันสภาวะผิดปกติ ตั้งแต่ต้นทาง
- ตรวจสอบอินพุตอย่างเข้มงวด (Strict Input Validation) รวมถึงการกรอง หรือการหลีกเลี่ยงอักขระที่เป็นอันตราย
- แสดงข้อความ Error แบบทั่วไปให้ผู้ใช้เห็น เช่น "เกิดข้อผิดพลาด กรุณาลองใหม่ภายหลัง" ส่วนรายละเอียดเชิงเทคนิคให้เก็บไว้ใน Log ภายในเท่านั้น
- ตรวจสอบให้แน่ใจว่าเมื่อเกิด Exception ระบบได้ทำการปิด Connection หรือคืน Memory อย่างถูกต้อง เพื่อป้องกันการโจมตีแบบ Denial of Service (DoS) จากการที่ทรัพยากรถูกใช้จนหมด

SINQRITY

SINQRITY

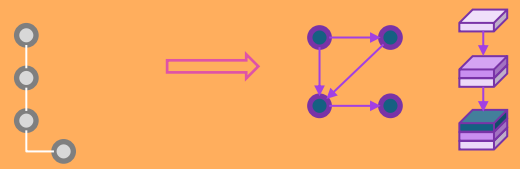
รูปแบบการพัฒนาแอปพลิเคชันได้เปลี่ยนไปแล้ว

จำนวนแอปพลิเคชันมากขึ้น จำนวนทีมผู้พัฒนามากขึ้น ต้องการแอปพลิเคชันเร็วขึ้น



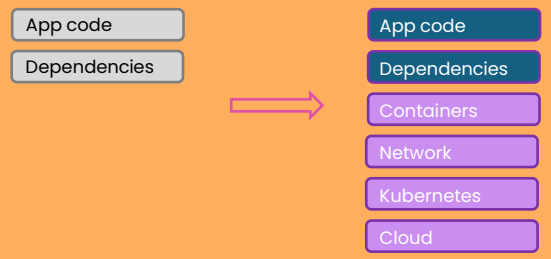
ต้องการผู้พัฒนาที่ความความเข้าใจในด้านความปลอดภัยในการพัฒนาแอปพลิเคชัน

องค์ประกอบของ Software ที่นำมาใช้ในการพัฒนามีความซับซ้อนมากขึ้น



ต้องการการตรวจสอบองค์ประกอบว่าองค์ประกอบที่นำมาใช้ นำมาใช้อย่างถูกต้องหรือมีช่องโหว่หรือไม่

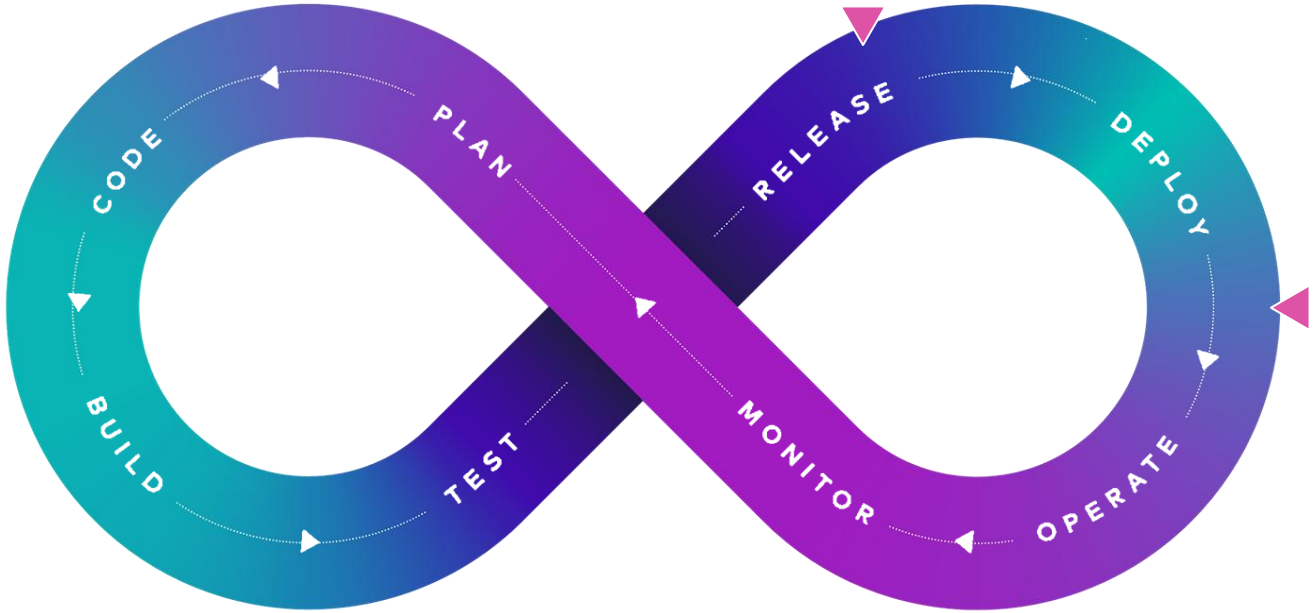
คลาวด์เป็นส่วนหนึ่งการการพัฒนาแอปพลิเคชัน



ต้องการเข้าใจในส่วนของทั้งการพัฒนาแอปพลิเคชันและการตั้งค่าของคลาวด์

แนวทางที่ควรดำเนินการสำหรับการพัฒนาแอปพลิเคชันอย่างปลอดภัย

ตรวจสอบการพัฒนาแอปพลิเคชัน เช่น ทำ Pentest



ตรวจสอบ Operating System เช่น ทำ Vulnerability Assessment

ข้อเสีย

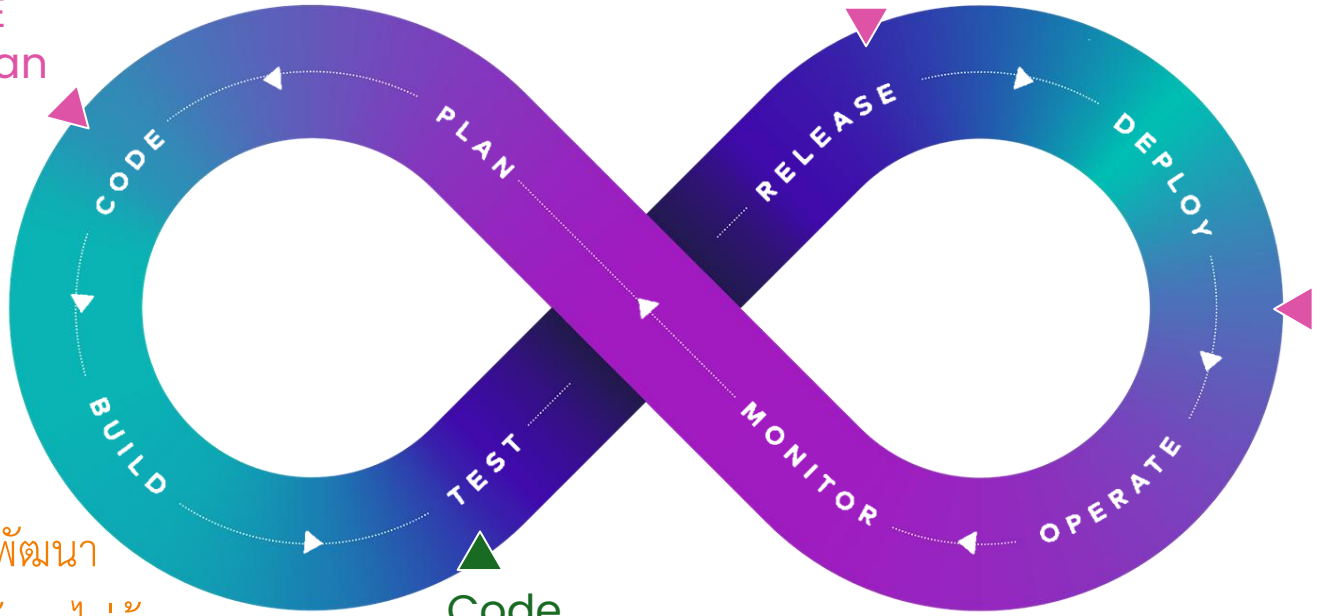
- การตรวจสอบทำช้าเกินไป
- จำเป็นต้องมีผู้เชี่ยวชาญเข้ามาเกี่ยวข้อง
- หากพบปัญหา อาจจะทำให้การส่งงานล่าช้า

SINQRITY

แนวทางที่ควรดำเนินการสำหรับการพัฒนาแอปพลิเคชันอย่างปลอดภัย

ตรวจสอบการพัฒนาแอปพลิเคชัน เช่น ทำ Pentest

IDE scan



ตรวจสอบ Operating System เช่น ทำ Vulnerability Assessment

Code audit “shifted left”

ข้อดี

- พบช่องโหว่ในขณะที่ดำเนินการพัฒนา
- ผู้พัฒนาสามารถแก้ไขช่องโหว่ได้เอง ไม่ต้องรอให้ผู้เชี่ยวชาญมาตรวจสอบ
- ช่องโหว่ที่ตรวจสอบจากการทำ Pentest, VA ลดลง ทำให้แผนการส่งมอบมีโอกาสเลื่อนได้น้อย

SINQRITY

แนวทางที่ควรดำเนินการสำหรับการพัฒนาแอปพลิเคชันอย่างปลอดภัย

A modern approach to security is required

Traditional App Sec

Testing after development

Audit Based

Code and Infrastructure Secured Independently

Dev-First

Continuous Testing

Fix Based

Holistic Cloud Native Application Context

SINQRITY

แนวทางที่ควรดำเนินการสำหรับการพัฒนาแอปพลิเคชันอย่างปลอดภัย

https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Web_Application_Penetration_Checklist_v1_1.pdf

Checklist

Category	Ref Number	Name	Objective	Notes
AppDOS	OWASP-AD-001	Application Flooding	Ensure that the application functions correctly when presented with large volumes of requests, transactions and / or network traffic.	Use various fuzzing tools to perform this test (e.g. SPIKE)
	OWASP-AD-002	Application Lockout	Ensure that the application does not allow an attacker to reset or lockout user's accounts.	

Category	Ref Number	Name	Objective	Notes
InputValidation	OWASP-IV-001	Script Injection	Ensure that any part of the application that allows input does not process scripts as part of the input.	Classic case of Cross Site Scripting but includes other scripting as well.
InputValidation.SQL	OWASP-IV-002	SQL Injection	Ensure the application will not process SQL commands from the user.	
InputValidation.OS	OWASP-IV-003	OS Command Injection	Ensure the applications will not process operating system commands from the user.	This typically includes issues such as path traversal, spawning command shells and OS functions.
InputValidation.LDAP	OWASP-IV-004	LDAP Injection	Ensure the application will not process LDAP commands form the user.	
InputValidation.XSS	OWASP-IV-005	Cross Site Scripting	Ensure that the application will not store or reflect malicious script code.	

SINQRITY

OWASP ZAP (ZAP Proxy)

OWASP Zed Attack Proxy (ZAP) เป็นเครื่องมือ Web Proxy อีกหนึ่งตัวที่นิยมใช้กันทั่วไปสำหรับการทดสอบเจาะระบบเว็บ (Web Penetration Testing)

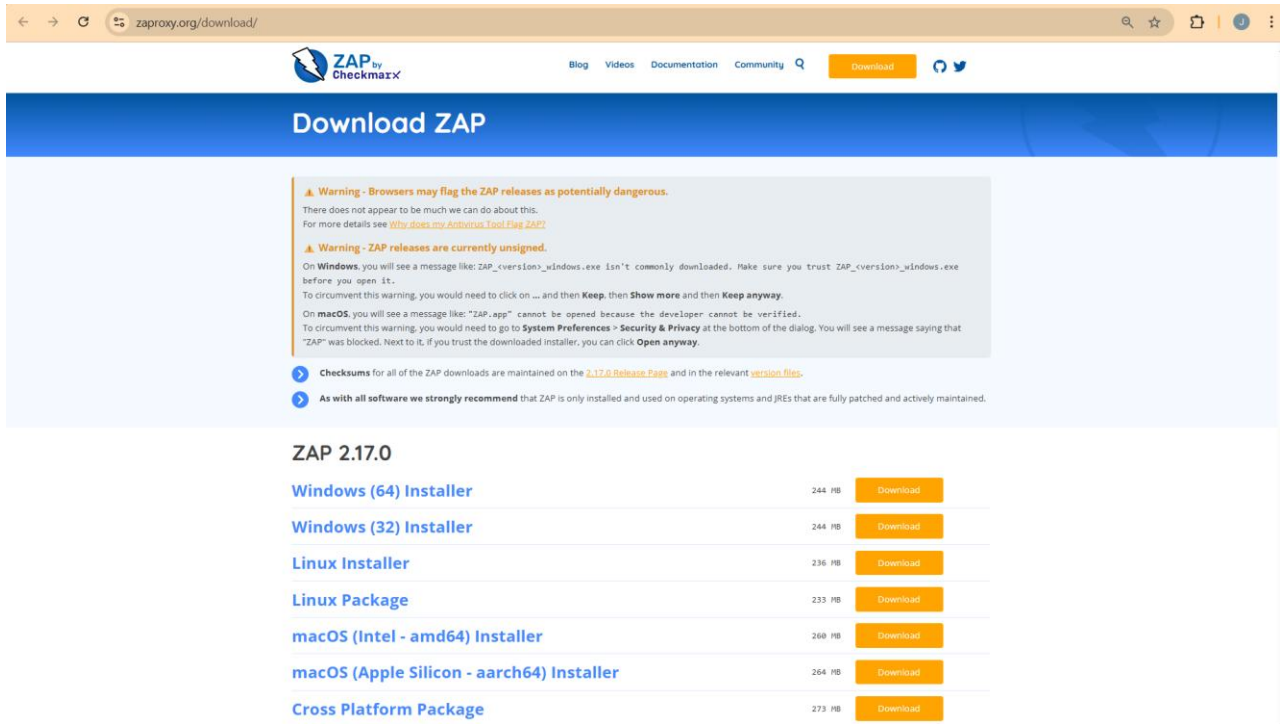
ZAP เป็นโปรเจกต์ Open-source ที่เปิดให้ใช้งานฟรี ริเริ่มโดย Open Web Application Security Project (OWASP) และได้รับการดูแลรักษาโดยชุมชนนักพัฒนา ซึ่งในช่วงไม่กี่ปีที่ผ่านมาเครื่องมือนี้มีการเติบโตอย่างมาก และกำลังได้รับการยอมรับในตลาดอย่างรวดเร็วในฐานะผู้นำด้านเครื่องมือ Web Proxy แบบ Open-source ในปัจจุบัน ถูกบริษัทที่ชื่อว่า Checkmarx ดำเนินการซื้อไป แต่ยังคงปล่อยให้มีการใช้ฟรีอยู่

SINQRITY

OWASP ZAP

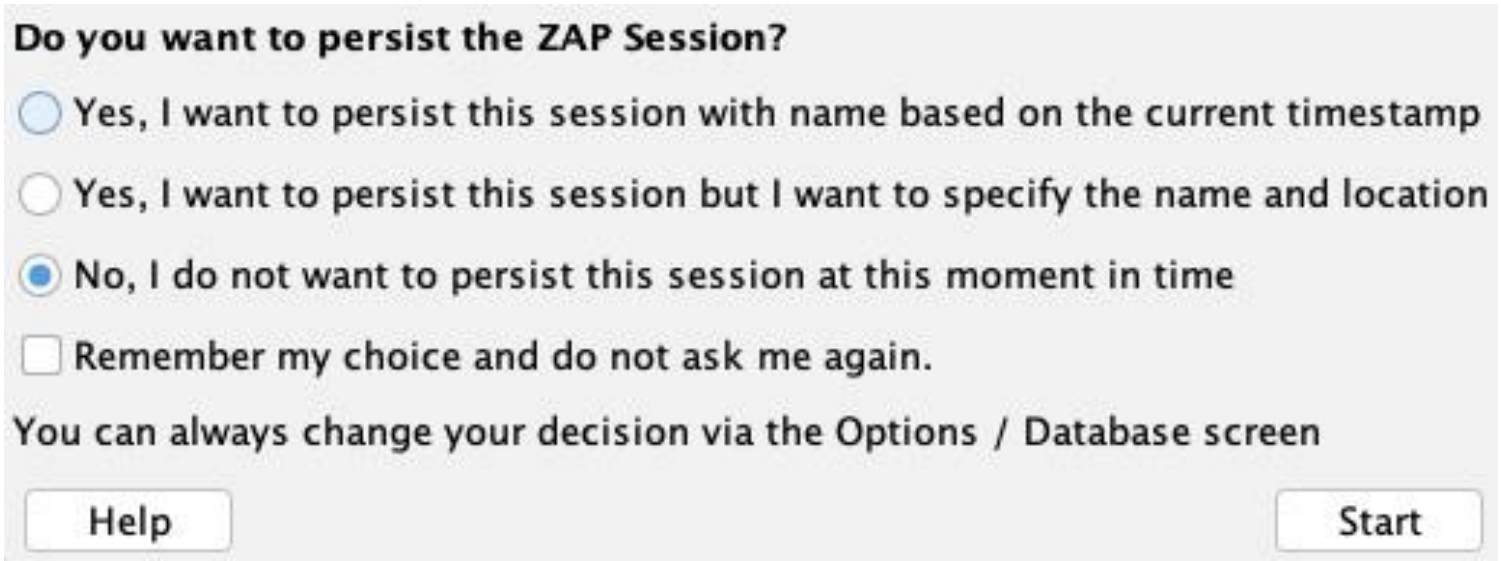
เราสามารถดาวน์โหลด ZAP ได้จากหน้าดาวน์โหลด (https://www.zaproxy.org/download/) โดยเลือกตัวติดตั้ง (Installer) ที่ตรงกับระบบปฏิบัติการที่เราใช้ และปฏิบัติตามคำแนะนำพื้นฐานเพื่อทำการติดตั้งนอกจากนี้

ZAP ยังมีให้ดาวน์โหลดในรูปแบบไฟล์ JAR ที่รองรับการทำงานข้ามแพลตฟอร์ม (Cross-platform) ซึ่งสามารถเปิดใช้งานได้ด้วยคำสั่ง java -jar หรือโดยการดับเบิลคลิกที่ไฟล์ดังกล่าว



OWASP ZAP

ในการเริ่มต้นใช้งาน ZAP เราสามารถเปิดโปรแกรมผ่าน Terminal ด้วยคำสั่ง zaproxy หรือเรียกใช้จากเมนูแอปพลิเคชันได้
เมื่อ ZAP เริ่มทำงาน จะมีหน้าต่างสอบถามว่าเราต้องการสร้างโปรเจกต์ใหม่หรือใช้เป็นโปรเจกต์ชั่วคราว ในที่นี้ให้เราเลือกใช้โปรเจกต์ชั่วคราวโดยเลือก No เนื่องจากเราไม่ได้กำลังทำงานกับโปรเจกต์ขนาดใหญ่ที่จำเป็นต้องบันทึกข้อมูลเก็บไว้ใช้งานต่อเนื่องหลายวัน



OWASP ZAP

การตั้งค่า Proxy

ใน ZAP เราสามารถคลิกที่ไอคอนเบราว์เซอร์ Firefox ที่อยู่ตรงด้านขวาสุดของแถบด้านบน เพื่อเปิดเบราว์เซอร์ที่ถูกตั้งค่าไว้ล่วงหน้าขึ้นมาได้ทันที:

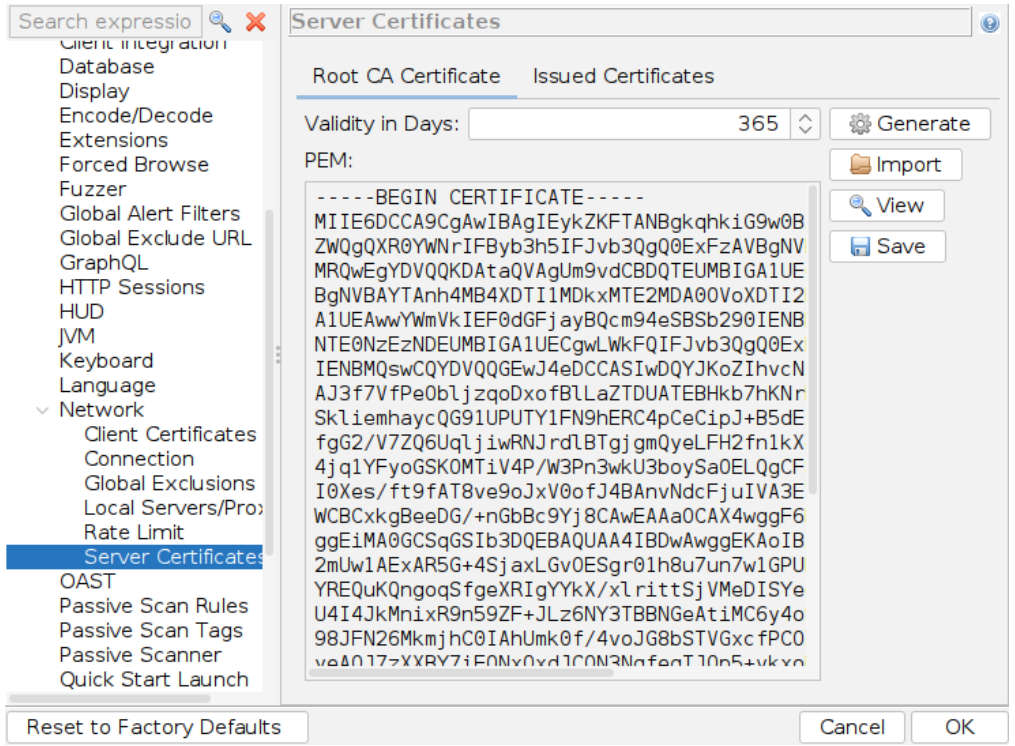
ในหลายๆ กรณี เราอาจต้องการใช้เว็บเบราว์เซอร์จริงๆ เช่น Firefox ในการทำ Pentest ซึ่งการจะใช้งาน Firefox ร่วมกับเครื่องมือ Web Proxy นั้น เราจำเป็นต้องเข้าไปตั้งค่าให้เบราว์เซอร์วิ่งผ่าน Proxy เหล่านั้นเสียก่อน เราสามารถเข้าไปกำหนดค่าเองได้ที่เมนูการตั้งค่า (Preferences) ของ Firefox เพื่อระบุ Proxy ให้ตรงกับ Listening Port ของ Web Proxy นั้นๆโดยปกติแล้ว ZAP จะใช้พอร์ต 8080 เป็นค่าเริ่มต้น แต่เราสามารถเลือกใช้พอร์ตอื่นๆ ที่ว่างอยู่ได้เช่นกัน แต่หากเราเลือกพอร์ตที่กำลังถูกใช้งานอยู่ Proxy จะไม่สามารถเริ่มทำงานได้และจะมีข้อความแจ้งเตือนข้อผิดพลาด (Error message) ปรากฏขึ้นหมายเหตุ: หากเราต้องการเปลี่ยนพอร์ตของ Web Proxy ในโปรแกรม ZAP ไปเป็นพอร์ตอื่น เราสามารถเข้าไปตั้งค่าได้ที่เมนู (Tools > Options > Network > Local Servers/Proxies) ซึ่งไม่ว่าจะใช้พอร์ตไหน เราต้องตรวจสอบให้แน่ใจว่าการตั้งค่า Proxy ใน Firefox นั้นระบุหมายเลขพอร์ตตรงกันกับที่ตั้งไว้

SINORITY

OWASP ZAP

การติดตั้งใบรับรองความปลอดภัย (CA Certificate)

ใน ZAP เราสามารถคลิกที่ไอคอนเบราว์เซอร์ Firefox ที่อยู่ตรงด้านขวาสุดของแถบด้านบน เพื่อเปิดเบราว์เซอร์ที่ถูกตั้งค่าไว้ล่วงหน้าขึ้นมาใช้งานได้เลย:



OWASP ZAP

ZAP Scanner

ZAP ยังมาพร้อมกับ Web Scanner ในตัว โดย ZAP Scanner นั้นมีความสามารถในการสร้างแผนผังเว็บไซต์ (Site Maps) ด้วยการใช้ ZAP Spider และสามารถทำการสแกนได้ทั้งแบบ Passive และ Active เพื่อค้นหาช่องโหว่ประเภทต่างๆ

SINQRITY

OWASP ZAP

Spider

มาเริ่มกันที่ ZAP การจะเริ่มใช้งาน Spider สแกนเว็บไซต์นั้น เราสามารถทำได้โดยไปที่แท็บ History เลือก Request ที่ต้องการ แล้วคลิกขวาเลือกเมนู (Attack > Spider) อีกวิธีหนึ่งคือการใช้งานผ่าน HUD บนเบราว์เซอร์ที่ ZAP ตั้งค่ามาให้ เมื่อเราเข้าไปยังหน้าเว็บหรือเว็บไซต์ที่ต้องการสแกนแล้ว ให้คลิกที่ปุ่มที่สองบนแถบด้านขวา (Spider Start) ระบบจะแสดงหน้าต่างขึ้นมาเพื่อให้เรายืนยันการเริ่มสแกน:

หมายเหตุ: เมื่อเราคลิกที่ปุ่ม Spider โปรแกรม ZAP อาจแจ้งเตือนว่าเว็บไซต์ปัจจุบันไม่ได้อยู่ในขอบเขต (Scope) ของเรา และจะถามว่าเราต้องการเพิ่มเว็บไซต์นี้เข้าไปใน Scope โดยอัตโนมัติก่อนเริ่มสแกนหรือไม่ ซึ่งในกรณีนี้เราสามารถตอบว่า 'Yes' ได้เลย Scope คือชุดของ URL ที่ ZAP จะทำการทดสอบหากเราเริ่มการสแกนแบบทั่วไป ซึ่งเราสามารถปรับแต่ง Scope ได้เองเพื่อกำหนดให้สแกนหลายเว็บไซต์หรือหลาย URL พร้อมกัน ลองฝึกเพิ่มเป้าหมาย (Targets) หลายๆ แห่งเข้าไปใน Scope ดู เพื่อสังเกตความแตกต่างในการทำงานของการสแกน

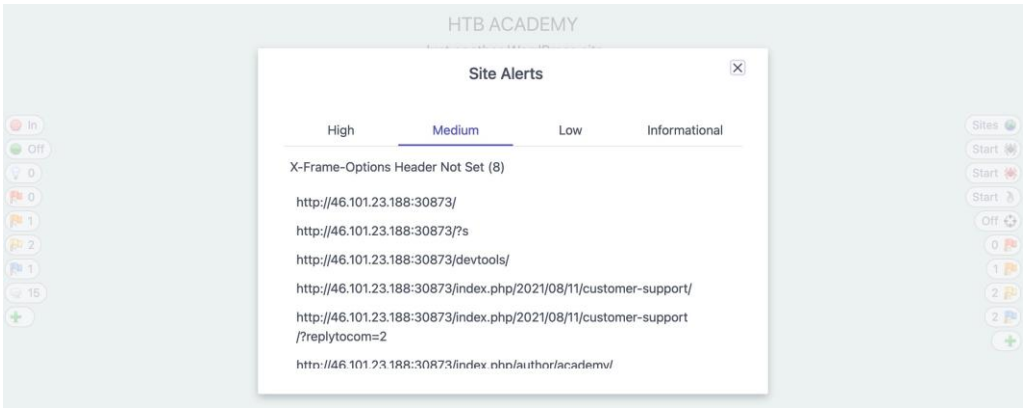
OWASP ZAP

ทันทีที่เราคลิกปุ่ม Start บนหน้าต่าง Pop-up ระบบจะเริ่มทำการ Spider เว็บไซต์ โดยจะค้นหาลิงก์ต่างๆ และตรวจสอบความถูกต้อง ซึ่งกระบวนการนี้จะคล้ายคลึงกับการทำงานของ Burp Crawlerเราสามารถติดตามความคืบหน้าของการสแกนได้ทั้งจาก HUD บริเวณปุ่ม Spider หรือผ่านหน้าจอหลัก (ZAP UI) ซึ่งปกติแล้วโปรแกรมจะสลับไปยังแท็บ Spider ให้โดยอัตโนมัติ เพื่อแสดงสถานะความคืบหน้าและ Request ต่างๆ ที่ถูกส่งออกไปเมื่อการสแกนเสร็จสิ้น เราสามารถตรวจสอบผลลัพธ์ได้ที่แท็บ Sites บนหน้าจอหลักของ ZAP หรือคลิกที่ปุ่มแรกบนแถบด้านขวา (Sites Tree) บน HUD ซึ่งจะแสดงรายการเว็บไซต์และไดเรกทอรีย่อยทั้งหมดที่ตรวจพบในรูปแบบโครงสร้างต้นไม้ (Tree-list) ที่สามารถกดขยายดูรายละเอียดได้

OWASP ZAP

Passive Scanner

ในขณะที่ ZAP Spider กำลังทำงานและส่ง Request ไปยัง Endpoint ต่างๆ ระบบจะทำการรัน Passive Scanner ไปพร้อมกันโดยอัตโนมัติในทุกๆ Response ที่ได้รับ เพื่อตรวจสอบหาปัญหาที่อาจซ่อนอยู่ใน Source Code เช่น การขาด Security Headers หรือช่องโหว่ประเภท DOM-based XSSด้วยเหตุนี้ แม้ว่าเราจะยังไม่ได้เริ่มรัน Active Scanner แต่เราอาจเห็นว่าปุ่มแจ้งเตือน (Alerts) เริ่มแสดงรายการปัญหาที่ตรวจพบขึ้นมาบ้างแล้ว โดยการแจ้งเตือนบนแถบด้านซ้ายจะแสดงปัญหาที่พบในหน้าเว็บปัจจุบันที่เรากำลังเปิดอยู่ ส่วนแถบด้านขวาจะแสดงภาพรวมของการแจ้งเตือนทั้งหมดในเว็บแอปพลิเคชันนั้น ซึ่งรวมถึงปัญหาที่ตรวจพบบนหน้าอื่นๆ ด้วย:



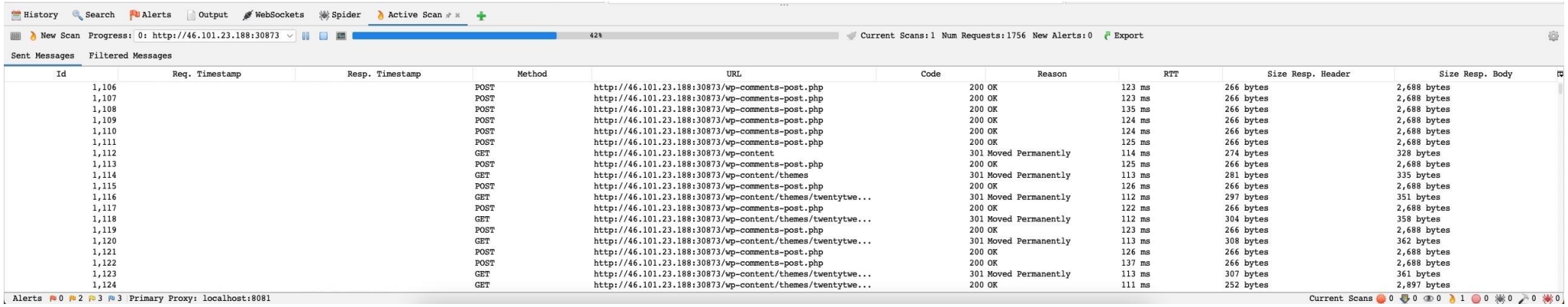
OWASP ZAP

Active Scanner

เมื่อข้อมูลใน Site Tree ของเราแสดงขึ้นมาแล้ว เราสามารถคลิกที่ปุ่ม Active Scan บนแถบด้านขวาเพื่อเริ่มการสแกนแบบ Active บนหน้าเว็บทั้งหมดที่ตรวจพบ หากเรายังไม่ได้ทำการรัน Spider Scan บนเว็บแอปพลิเคชันมาก่อน ZAP จะทำการรัน Spider ให้โดยอัตโนมัติเพื่อสร้าง Site Tree สำหรับใช้เป็นเป้าหมายในการสแกน และเมื่อ Active Scan เริ่มทำงาน เราสามารถติดตามความคืบหน้าได้ในลักษณะเดียวกับตอนที่เราทำ Spider Scan

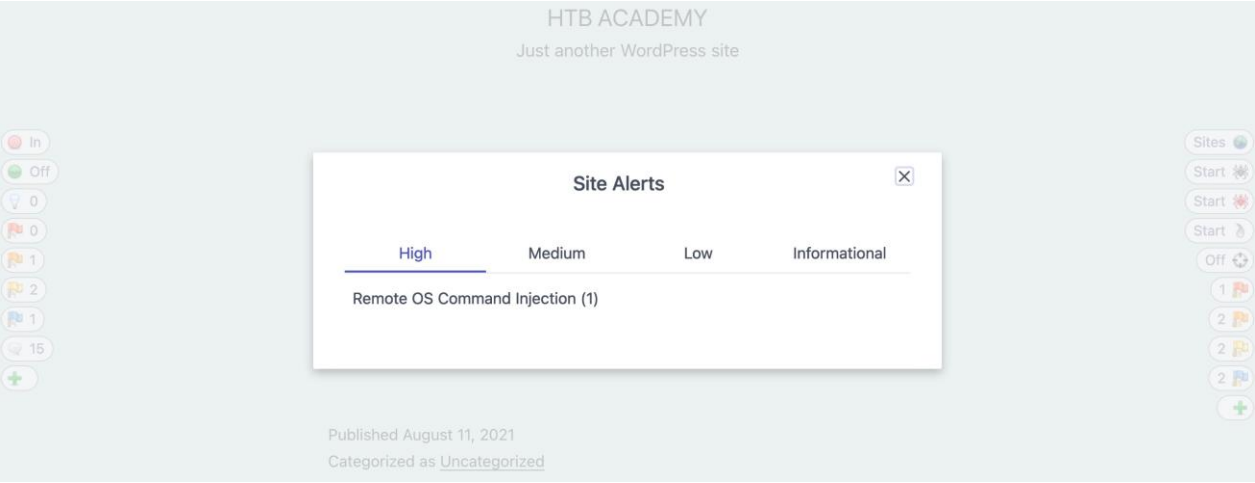
OWASP ZAP

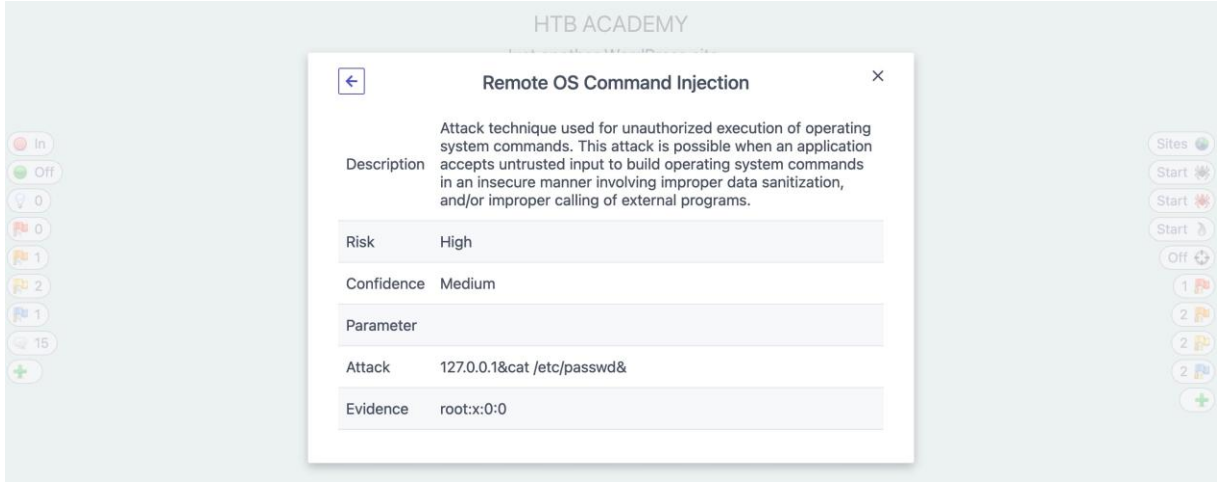
Active Scanner จะพยายามใช้รูปแบบการโจมตีที่หลากหลายกับหน้าเว็บและ HTTP Parameters ทั้งหมดที่ระบุได้ เพื่อค้นหาช่องโหว่ให้ได้มากที่สุดเท่าที่จะทำได้ ด้วยเหตุนี้การทำงานของ Active Scanner จึงใช้เวลานานกว่าจะเสร็จสิ้นในขณะที่ Active Scan กำลังทำงาน เราจะเห็นว่าปุ่มแจ้งเตือน (Alerts) เริ่มมีรายการแจ้งเตือนเพิ่มมากขึ้นเรื่อยๆ ตามจำนวนปัญหาที่ ZAP ตรวจพบ ยิ่งไปกว่านั้น เราสามารถตรวจสอบรายละเอียดเพิ่มเติมเกี่ยวกับการสแกนที่กำลังทำงานอยู่ได้ที่หน้าจอหลัก (ZAP UI) และสามารถดู Request ต่างๆ ที่ ZAP ส่งออกไปได้:



OWASP ZAP

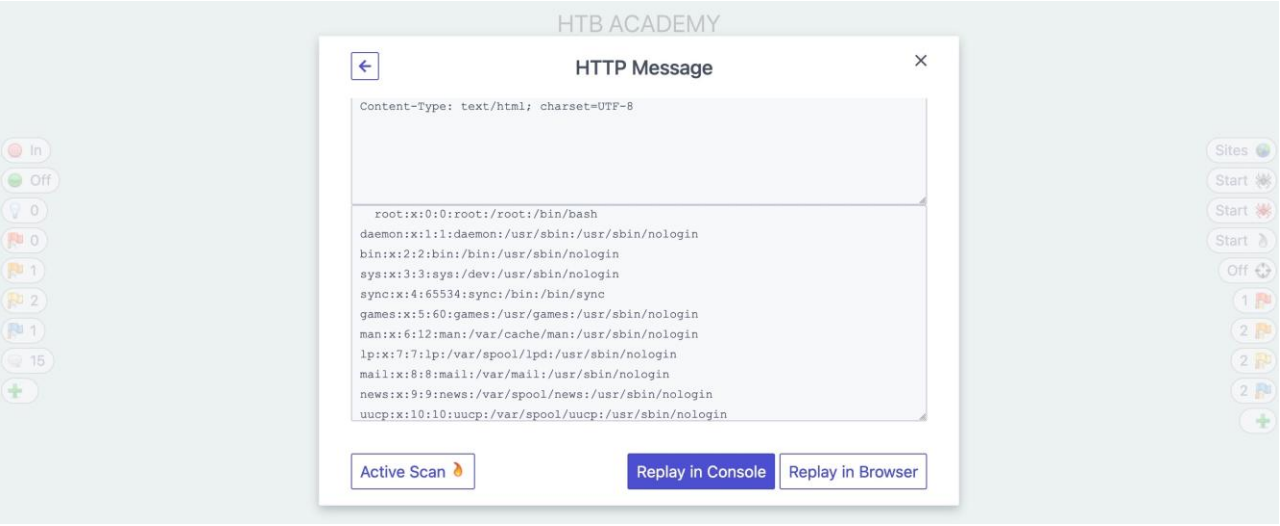
เมื่อการทำงานของ Active Scan เสร็จสิ้น เราสามารถเข้าไปดูรายการแจ้งเตือน (Alerts) เพื่อพิจารณาว่ารายการใดบ้างที่ควรต้องติดตามผลต่อแม้ว่าการแจ้งเตือนทั้งหมดควรถูกรายงานและนำมาพิจารณา แต่การแจ้งเตือนระดับสูง (High Alerts) นั้นมักจะเป็นช่องโหว่ที่นำไปสู่การถูกยึดครอง (Compromise) เว็บแอปพลิเคชันหรือ Back-end Server ได้โดยตรงหากเราคลิกที่ปุ่ม High Alerts ระบบจะแสดงรายการแจ้งเตือนระดับความรุนแรงสูงที่ตรวจพบขึ้นมา:





OWASP ZAP

บนหน้าต่างรายละเอียดการแจ้งเตือน (Alert details window) เรายังสามารถคลิกที่ URL เพื่อดูรายละเอียดของ Request และ Response ที่ ZAP ใช้ในการตรวจสอบพบช่องโหว่นี้ และเรายังสามารถทำการส่ง Request นี้ซ้ำ (Repeat) ผ่านทาง ZAP HUD หรือ ZAP Request Editor ได้อีกด้วย:

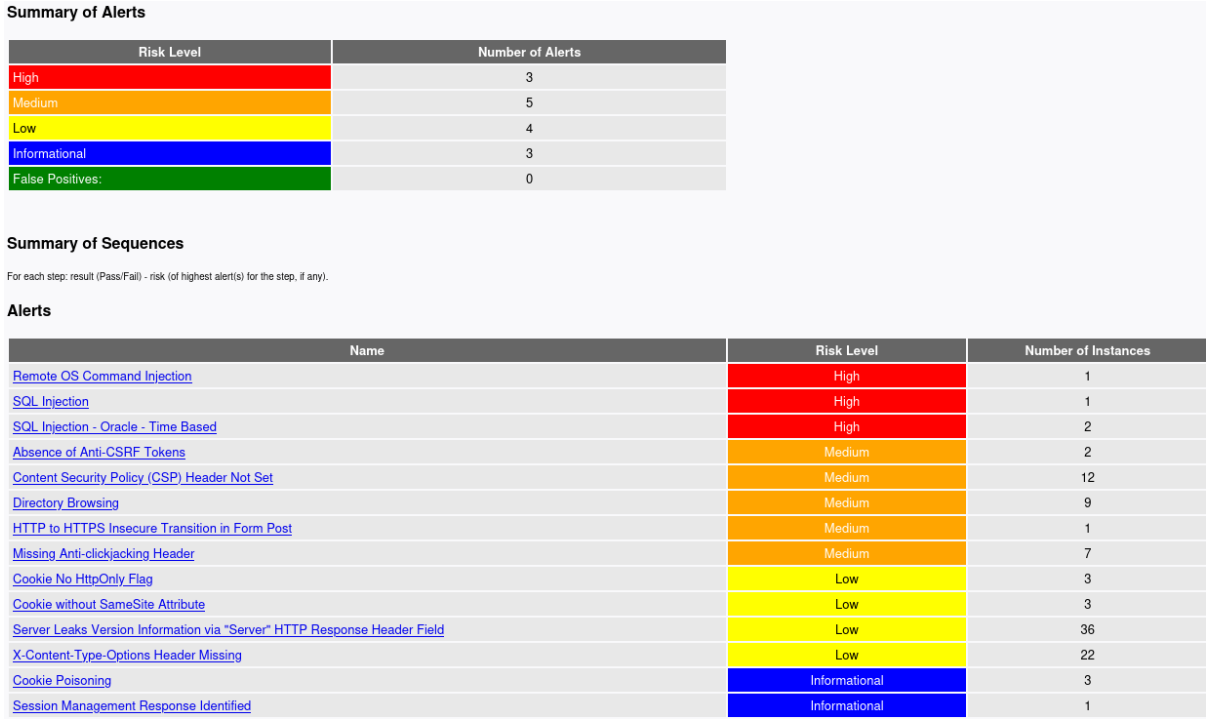


OWASP ZAP

SINQRITY

การออกรายงาน (Reporting)

ในขั้นตอนสุดท้าย เราสามารถสร้างรายงาน (Report) ที่รวบรวมสิ่งที่ตรวจพบ (Findings) ทั้งหมดจากการสแกนรูปแบบต่างๆ ของ ZAP ได้ โดยไปที่แถบเมนูด้านบน แล้วเลือก (Report > Generate HTML Report) จากนั้นระบบจะให้เราระบุตำแหน่งที่ต้องการบันทึกไฟล์รายงานนอกจากนี้ เรายังสามารถส่งออก (Export) รายงานในรูปแบบอื่นๆ ได้อีก เช่น XML หรือ Markdown เมื่อสร้างรายงานเสร็จเรียบร้อยแล้ว เราสามารถเปิดดูไฟล์ดังกล่าวผ่านเว็บเบราว์เซอร์ใดก็ได้:



Summary of Alerts

Risk Level	Number of Alerts
High	3
Medium	5
Low	4
Informational	3
False Positives:	0

Summary of Sequences

For each step: result (Pass/Fail) - risk (of highest alert(s) for the step, if any).

Alerts

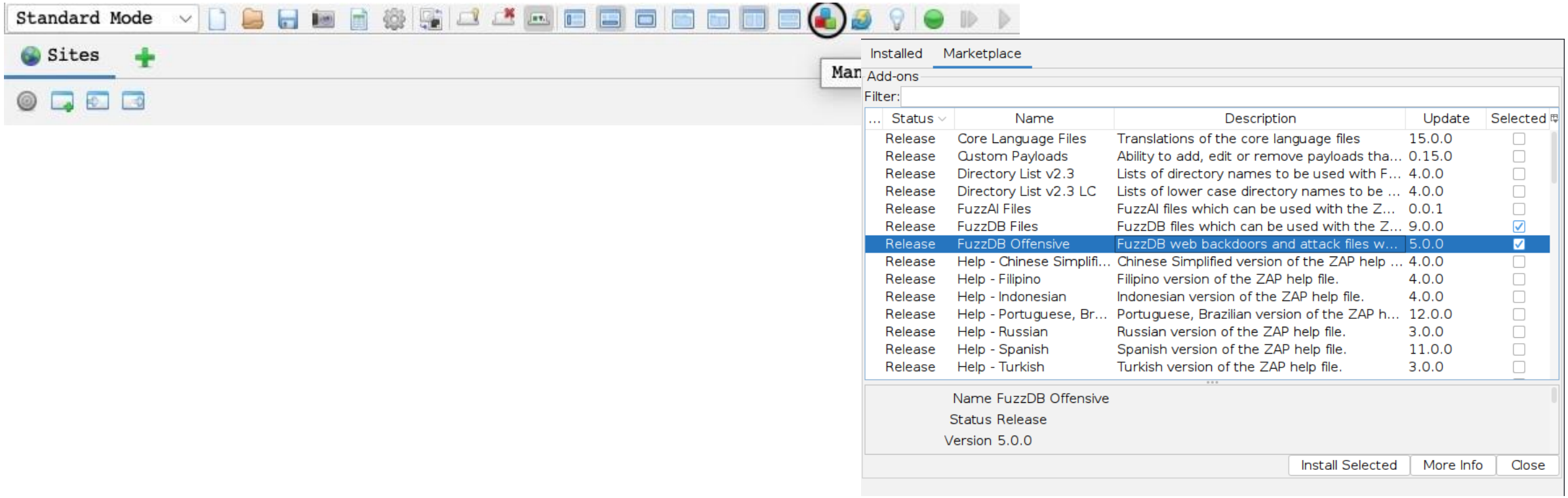
Name	Risk Level	Number of Instances
Remote OS Command Injection	High	1
SQL Injection	High	1
SQL Injection - Oracle - Time Based	High	2
Absence of Anti-CSRF Tokens	Medium	2
Content Security Policy (CSP) Header Not Set	Medium	12
Directory Browsing	Medium	9
HTTP to HTTPS Insecure Transition in Form Post	Medium	1
Missing Anti-clickjacking Header	Medium	7
Cookie No HttpOnly Flag	Low	3
Cookie without SameSite Attribute	Low	3
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	36
X-Content-Type-Options Header Missing	Low	22
Cookie Poisoning	Informational	3
Session Management Response Identified	Informational	1

OWASP ZAP

SINCRITY

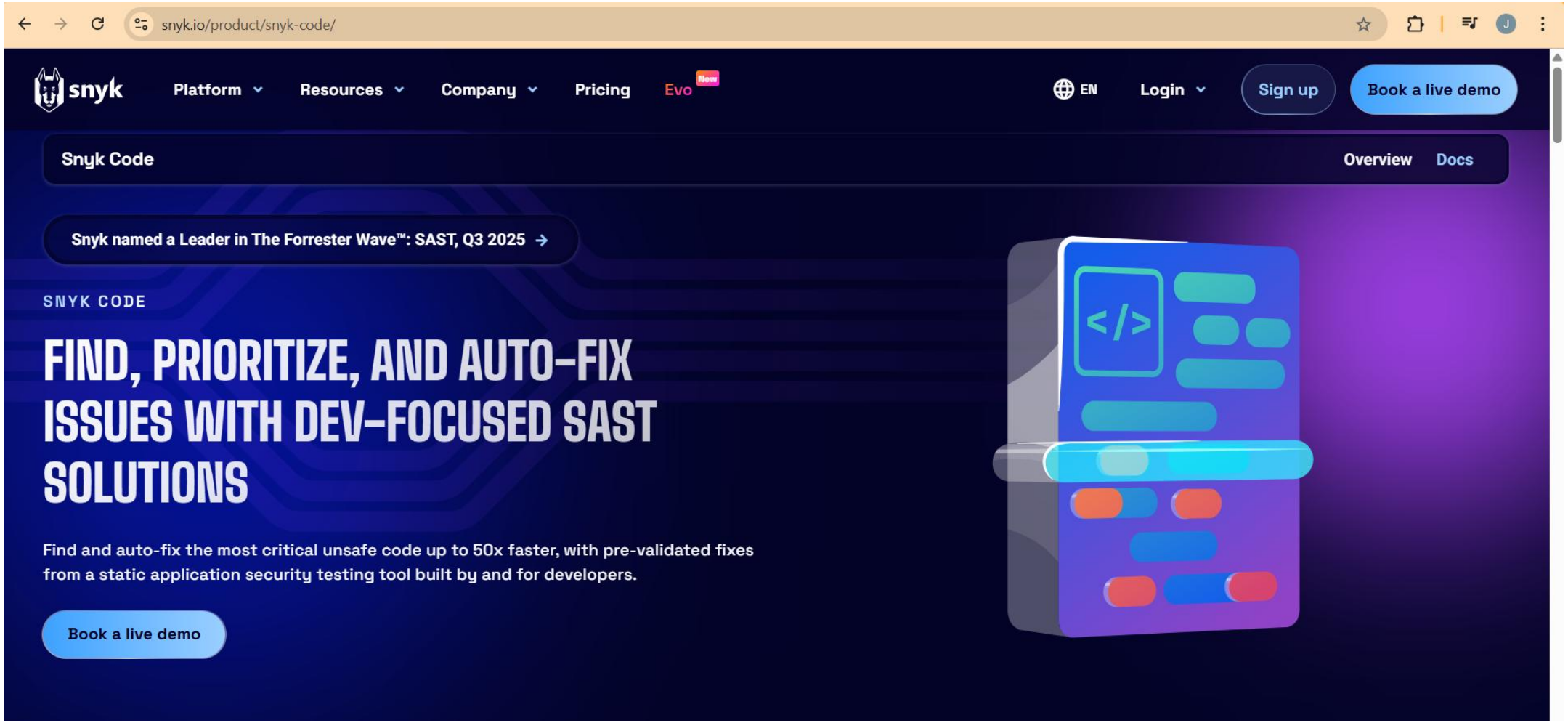
ZAP Marketplace

ZAP ยังมีฟีเจอร์ในการขยายความสามารถ (Extensibility) เป็นของตัวเองผ่านทาง Marketplace ซึ่งช่วยให้เราสามารถติดตั้ง Add-ons ประเภทต่างๆ ที่ถูกพัฒนาโดยชุมชนผู้ใช้งานได้ในการเข้าถึง Marketplace ของ ZAP ให้เราคลิกที่ปุ่ม Manage Add-ons จากนั้นเลือกไปที่แท็บ Marketplace:



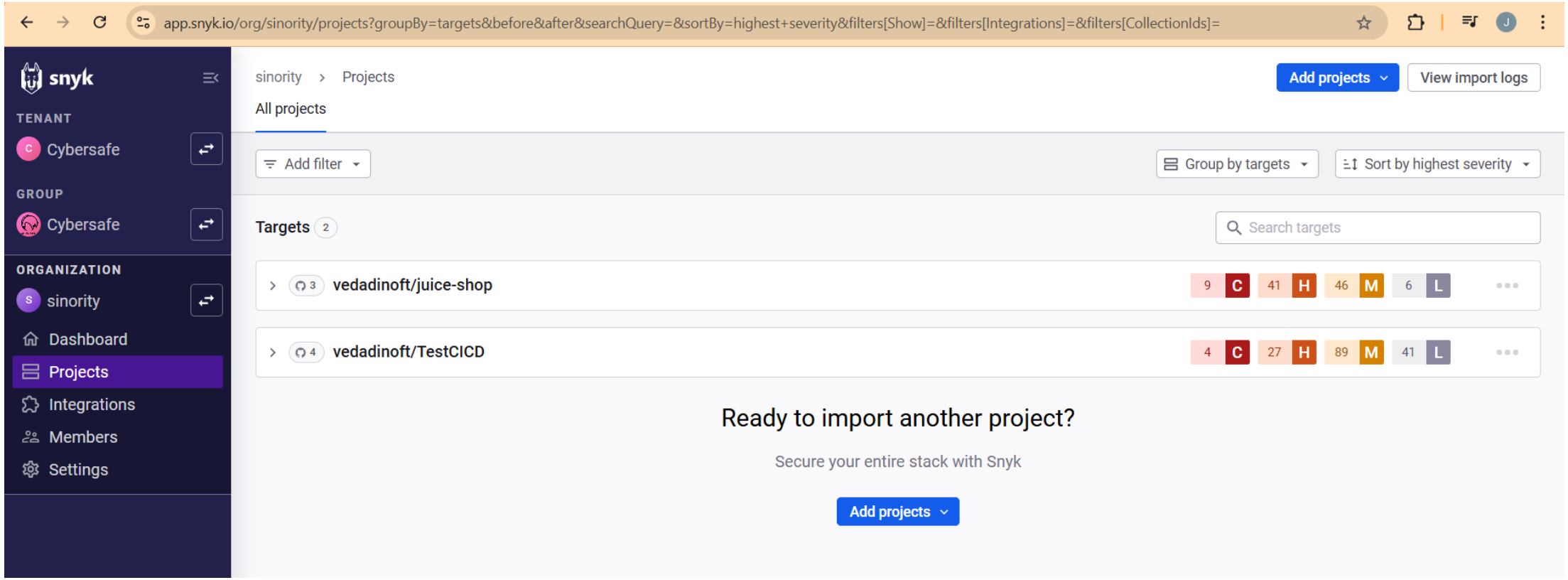
Snyk Code

SINQRITY

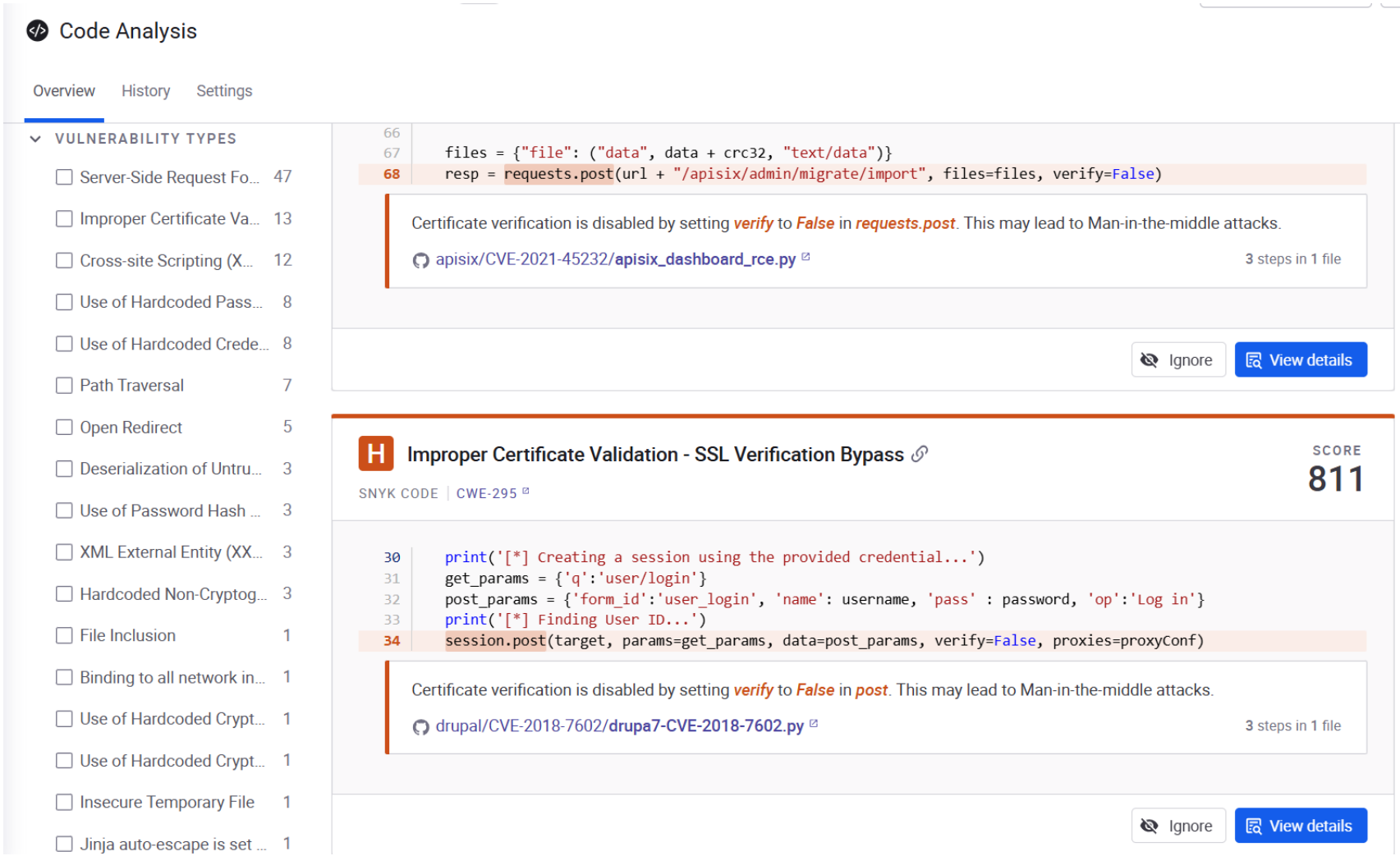


Snyk Code

SINORITY



Snyk Code


Code Analysis
Overview
History
Settings
VULNERABILITY TYPES
Server-Side Request Fo... 47
Improper Certificate Va... 13
Cross-site Scripting (X... 12
Use of Hardcoded Pass... 8
Use of Hardcoded Crede... 8
Path Traversal 7
Open Redirect 5
Deserialization of Untru... 3
Use of Password Hash ... 3
XML External Entity (XX... 3
Hardcoded Non-Cryptog... 3
File Inclusion 1
Binding to all network in... 1
Use of Hardcoded Crypt... 1
Use of Hardcoded Crypt... 1
Insecure Temporary File 1
Jinja auto-escape is set ... 1
66
67 files = {"file": ("data", data + crc32, "text/data")}
68 resp = requests.post(url + "/apisix/admin/migrate/import", files=files, verify=False)
Certificate verification is disabled by setting verify to False in requests.post. This may lead to Man-in-the-middle attacks.
apisix/CVE-2021-45232/apisix_dashboard_rce.py
3 steps in 1 file
Ignore
View details
H
Improper Certificate Validation - SSL Verification Bypass
SNYK CODE | CWE-295
SCORE
811
30 print('[*] Creating a session using the provided credential...')
31 get_params = {'q':'user/login'}
32 post_params = {'form_id':'user_login', 'name': username, 'pass' : password, 'op':'Log in'}
33 print('[*] Finding User ID...')
34 session.post(target, params=get_params, data=post_params, verify=False, proxies=proxyConf)
Certificate verification is disabled by setting verify to False in post. This may lead to Man-in-the-middle attacks.
drupal/CVE-2018-7602/drupa7-CVE-2018-7602.py
3 steps in 1 file
Ignore
View details

Snyk Code

SINQRITY

1 of 122 issues

Group by **none** Sort by **highest severity**

H SQL Injection

SNYK CODE | CWE-89

SCORE
801

```
15    ctx.response.type = 'html'
16
17    let id = ctx.request.query.id || 1
18    let sql = `SELECT * FROM "user" WHERE "id"
19    const res = await client.query(sql)
```

Unsanitized input from *an HTTP parameter flows* into
vulnerability.

node/CVE-2017-16082/www/**app.js**

Learn about this type of vulnerability and how to fix it

H SQL Injection

SNYK CODE | CWE-89

Data flow | Fix analysis

Unsanitized input from *an HTTP parameter flows* into *query*, where it is used in an SQL query. This may result in an SQL Injection vulnerability.

Data flow - 12 steps in 1 file

node/CVE-2017-16082/www/**app.js** 12 steps

17:26	query, query, ctx.request.query,...	SOURCE	1 - 7
18:52	id, `SELECT * FROM "user" WHERE "id" = $...		8 - 10
19:36	sql, query		11 - 12
:36	sql		11
:30	query	SINK	12

Find out how to remediate this issue through our Fix analysis »

node/CVE-2017-16082/www/**app.js**

```
11 })
12 client.connect()
13
14 app.use(async ctx => {
15     ctx.response.type = 'html'
16
17     let id = ctx.request.query.id || 1
18     let sql = `SELECT * FROM "user" WHERE "id" = ${id}`
19     const res = await client.query(sql)
20
21     ctx.body = `<html>
22                     <body>
23                         <table>
24                             <tr><th>id</th><td>${res.rows[0].id}</td></tr>
25                             <tr><th>name</th><td>${res.rows[0].name}</td></tr>
26                             <tr><th>score</th><td>${res.rows[0].score}</td></tr>
27                         </table>
28                     </body>
29                 </html>`
30 })
31
32 app.listen(3000)
```

Ignore

Snyk Code

SINQRITY

SQL Injection

SNYK CODE | CWE-89

Data flow | Fix analysis

Details

In an SQL injection attack, the user can submit an SQL query directly to the database, gaining access without providing appropriate credentials. Attackers can then view, export, modify, and delete confidential information; change passwords and other authentication information; and possibly gain access to other systems within the network. This is one of the most commonly exploited categories of vulnerability, but can largely be avoided through good coding practices.

Best practices for prevention

- Avoid passing user-entered parameters directly to the SQL server.
- Avoid using string concatenation to build SQL queries from user-entered parameters.
- When coding, define SQL code first, then pass in parameters. Use prepared statements with parameterized queries. Examples include `SqlCommand()` in .NET and `bindParam()` in PHP.
- Use strong typing for all parameters so unexpected user data will be rejected.
- Where direct user input cannot be avoided for performance reasons, validate input against a very strict allowlist of permitted characters, avoiding special characters such as `? & / < > ; - ' " \` and spaces. Use a vendor-supplied escaping routine if possible.
- Develop your application in an environment and/or using libraries that provide protection against SQL injection.
- Harden your entire environment around a least-privilege model, ideally with isolated accounts with privileges only for particular tasks.

Example fixes

andrei-tatar/HAP.js 1 / 3

```
7  -        var tableName = "Temp" + req.body.id;
   7+       var device = node.device(req.body.id);
8  8
9  -        var time = Math.round(Date.now()/1000);
   9+       if (device) {
10 10
11 -        var value = req.body.temperature;
   11+          if (!device.init_temperature) {
12 12
   13+             device.init_temperature = true;
13 14
   15+             util.createProperty(device, 'temperature', req.body.
14 16
15 -        db.serialize(function() {
   17+          }
16 18
17 -             db.run("begin transaction");
18 -
19 -             db.run("create table if not exists " + tableName + " (ti
20 -
21 -             db.run("insert into " + tableName + " values (?, ?)", [t
22 -
23 -             db.run("commit");
24 -
25 -         });
   19+          else
26 20
   21+             device.temperature = req.body.temperature;
27 22
```

Ignore

ตอบคำถามผู้เข้าอบรม


SINQRITY