האוניברסיטה הפתוחה
המחלקה למתמטיקה ולמדעי המחשב

תהליך הפיתוח כקו הגנה מרכזי מפני תקיפות סייבר

עבודה מסכמת זו הוגשה כחלק מהדרישות לקבלת תואר
"מוסמך למדעים" .M.Sc במדעי המחשב
באוניברסיטה הפתוחה
המחלקה למתמטיקה למדעי המחשב

על-ידי
אראלי פלאח

העבודה הוכנה בהדרכתם של **פרופ' אהוד גודס ומר יצחק בייז**

דצמבר 2018

תודות

ראשית, תודה לכל חבריי ומכריי שהיו מוכנים להקשיב לתשובה לשאלתם – "במה עוסקת עבודת המסטר שלך"? מבלי להראות משועממים או מתחרטים על השאלה.

תודה רבה מסורה לפרופסור אהוד גודס ולמר יצחק בייז, על ההדרכה וההכוונה, התיקונים והדגשים במהלך העבודה, ועל כך שחרף העובדה שהעבודה נכתבה במשך תקופה ארוכה - לא אמרו נואש ותמיד היו נכונים לעזור.

תודה למכללת See security ובראשה מר אבי וייסמן על ההזדמנות להיחשף לעולם ה"האקינג" בצורה סדורה כחלק מקורס HDE, ובמיוחד למר ג'קי אלטל על הלימוד והשיתוף מניסיונו הרב.

תודה לחבריי בחברת סיסקו, שהיו שותפים לצעדיי הראשונים בעולם אבטחת התוכנה ולמשימות ראשונות בתחום, שענו על השאיפה להיכנס בעובי הקורה של הנושא.
תודה גדולה לכריס רומנו על בניית הפרויקט המושקע Security Ninja ועל האפשרות לקחת בו חלק פעיל.

תודה לאבי ז"ל, אימי ואחיותיי על החינוך להשכלה והטמעת ההכרה על חשיבותה הגדולה ועל החינוך לאמונה בעצמי וביכולות שלי.

תודה אחרונה וענקית לנגה, אשתי המדהימה – לא אוכל לתאר את הערכתי הגדולה אליה - וילדינו רננה, שהם, יאיר ברקאי וכפיר על השלמתם עם השעות הארוכות שבילית בכתיבת העבודה - בלעדיכם זה לא היה קורה!

תוכן עניינים

רשימת איורים

תקציר

תקיפת מערכות מחשבים והמרחב המקוון (סייבר) הינו עולם רחב, במורצת השנים הוצעו מספר דרכים וכיוונים להתמודד עם תקיפות אלו.
מטרת עבודה זו הינה להוכיח שמקורן של רוב הפגיעויות הוא בשגיאה שהייתה יכולה להימנע על ידי מפתחי המערכת ולהציע את ריכוז ההגנה מפני תקיפות סייבר בתהליך פיתוח מאובטח.
בעבודה זו סקרתי טכנולוגית 5 פגיעויות ותקיפות מתחומים שונים שנעשו בשנים האחרונות עד מציאת שורש השגיאה בכל פגיעות וכיצד היה אפשר להימנע משגיאה זו.
הראיתי כיצד אפשר לתחם את הפגיעויות השונות ל 3 אזורים עיקריים – מערכת הפעלה, רשת ותקשורת וקוד האפליקציה – והדגמה של תקיפות שניצלו פגיעויות אלו.
מטרת קטלוג זה הינה להוות סמן עבור חיפוש אחר פגיעויות בשלבים השונים בתהליך הפיתוח המאובטח.
תיארתי בפירוט מהו תהליך פיתוח מאובטח ואת שלבי התהליך הכלולים בשלבי הפיתוח השונים – תכנון, פיתוח ובדיקה, תוך שילוב סוגי בדיקות האבטחה ברמות השונות.
תיאור התהליך נעשה בד בבד עם מתן דגש על העובדה שתהליך פיתוח מאובטח נכון אינו נחלתו של המפתח בלבד, אלא של גורמים רבים המעורבים במעגל חייה של התוכנה – מנהלים, ארכיטקטורים, מפתחים, בודקים ומתקיני המערכת.
תהליך הפיתוח המאובטח שעסקתי בו כלל את השלבים הבאים:

- הגדרת המערכת
- איפיון איומים אפשריים
- תוכנות צד שלישי
- אבטחת קוד סטטי
- בדיקת פגיעויות

הצגתי את חסרונותיו של תהליך זה ואת האפשרות להתגבר על חסרונות אלו באמצעות שילוב התהליך כחלק מתהליך ה Agile [1] של פיתוח התוכנה. כתוצאה מהאפקטיביות של תהליך זה ובעקבות פגיעויות חדשות בעולם החומרה העליתי את האפשרות שנדרש תהליך פיתוח מאובטח אף בפיתוח רכיבי החומרה.
עבור הוכחת הרעיון פיתחתי מערכת פגיעה, מימשתי את התקיפות הנפוצות הקיימות על מערכת מסוג זה והצעתי דרך כללית למימוש קוד מאובטח כחלק מתהליך הפיתוח המאובטח.
מסקנתי מהעבודה שנעשתה היא שתהליך הפיתוח המאובטח הינו תהליך הדורש למידה והכנה והן השקעת משאבים אולם יש לו השפעה ישירה וקריטית לחוסנה של המערכת ובהיעדר תהליך מבוסס מעין זה, המערכת תכלול פגיעויות ממשיות אשר יתורגמו בשלב כלשהו בעתיד ע"י גורם זדוני לתקיפה ממשית העשויה לגרום נזק חמור למערכת, עקב כך מימוש תהליך זה בצורה נכונה אינו רשות אלא מחויב המציאות.
אדגיש כי אין מדובר במערכות או אפליקציות מתחום האבטחה דווקא, אלא לפיתוח מערכת מבוססת תוכנה.

[1] יפורט בהרחבה כחלק מפרק 4 העוסק בתהליך פיתוח מאובטח

1. מבוא

עולם המחשבים פרץ לחיינו בצורה משמעותית במאה ה 20, אך בראשית דרכו היה עולם זה נחלתם של מתי-מעט וככל שהשנים מתקדמות אנו רואים את השימוש במוצרי תוכנה וחומרה רק הולך ומתרחב, הן ברמת האינבידואל והן ברמת שירותי האינטרנט הרבים המוצעים לקהל הרחב.
אולם, עולם המחשבים טומן בחובו גם תלות רבה וסכנות גדולות– כסף שהיה מוחבא בעבר מתחת לרצפת הבית, כיום מתאפשר להגיע אליו באמצעות הקשת מספר מקשים במקלדת המחשב או הזזת מסכים בטלפון החכם..
האם רק אני – כבעל החשבון - יכול להגיע לכסף הזה? האם אני יכול להיות סמוך ובטוח שהכסף שלי מוגן?

בשנים האחרונות אנו שומעים הרבה מאוד על התקפות סייבר ועל ההגנה מפני תקיפות אלו, או בכלל את המושג סייבר נזרק לאוויר לעתים תכופות.
דוגמאות להתקפות מהסוג הנ"ל אפשר למצוא מפוזרות על גביי קשת רחבה של תחומים – בטחון, תעשייה, רפואה ואפילו בחיי הפרט. עובדה זו משפיעה על רמת הסיכון ובהתאמה על האחריות והיחס של הגורמים האמונים על תפקודה של המערכת ביום שהיא תצטרך להתמודד עם תקיפה מסוג כזה.
מפליאה העובדה שאפילו בקרב האנשים הטכניים בעולם המחשבים ישנו יחס מגוון ואף מנוגד לרמת חשיבות התייחסות לאבטחת מערכות ופרט לאבטחת תוכנה.
משיחות שניהלתי בשנים האחרונות עם אנשי תוכנה בכל הדרגות ומחברות שונות ומובילות בתחום ניכרת העובדה שפעמים רבות תינתן עדיפות להבטחת פונקציונליות מסוימת או בדיקות ביצועי מערכת על פני חוסנה של המערכת במדדי אבטחת מחשבים - עובדה מדאיגה כשלעצמה. אולם אם היינו מספיק רוצים, האם היינו יכולים לייצר מערכת חסינה לגמרי?

אחת הגישות בקשר לאבטחת מוצר תוכנה היא שזו אחריות גדולה שצריכה להיות מטופלת ע"י קבוצת אבטחה במנותק בקבוצת הפיתוח – הפרדת רשויות על סמך מיומנויות שונות.
מטרת העבודה זו הינה לספק הוכחה לכך שישנה חשיבות גדולה לעסוקים בתחום הפיתוח להיות אמונים על אבטחת המערכת ולספק כלים וכיווני חשיבה, לחשוף ולהוכיח את חשיבותו של פיתוח מאובטח ומיקומו כקו הגנה עיקרי מפני תקיפות סייבר. את החינוך לכך יש להנחיל במסגרת הכשרתו הבסיסית של המפתח – בין בלימודיו האקדמיים/מקצועיים טרם העסקתו בחברה, והמשך בהכשרות מקצועיות במקום עבודתו.

2. פריצות אבטחה – סקירה טכנולוגית

בעמודים הבאים אסקור פריצות אבטחה ותקיפות בולטות מהשנים האחרונות.
חלק מפריצות אלו היו בולטות בהיקפן הרחב בקרב רכיבי התוכנה והחומרה וחלקן בולטות ברמת חומרתן – האפשרות לנצל פריצות אלו לתקיפה משמעותית.
במהלך הסקירה אנסה להתמקד בניתוח הטכנולוגי של הפריצה תוך מתן דגש על שכבות המערכת השונות והיכן בדיוק הייתה נקודת התורפה בקוד, במערכת או במחשב שאליו פרצו.

2.1 2014 – שנה קשה בהחלט

שנת 2014 הייתה שנה עמוסה בהיבטים של אבטחת תוכנה.
בשנה זו נתגלו 3 פגיעויות תוכנה רציניות, שייחודן היא העובדה שפגיעויות אלו לא היו קשורות למערכת מסוימת, דפדפן מסוים או תוכנה ספציפית – אלא ששתיים מהן היו קשורות לליבת תשתית האינטרנט והשלישית נתגלתה במכשירים מבוססי לינוקס.
ההשלכה של העובדה המוזכרת לעיל הינה שכמעט כל המכשירים שמאפשרים חיבור לאינטרנט חושפים את המשתמש בהם לתקיפת מרחב מקוון.

2.1.1 Heartbleed (CVE-2014-0160)[1][2]

בחודש אפריל של שנת 2014 פורסמה על ידי חברת google[2] פגיעות המשפיעה על כ 78% מאתרי האינטרנט.
הפגיעות נתגלתה בספריית OpenSSL, ספרייה נפוצה למימוש פרוטוקול SSL/TLS בשירותי web המאפשר לבצע קישור מאובטח בין רכיבי אינטרנט שונים.

[2] חילוקי דעות בין חברת גוגל לחברת קודנומיקון מי חשף לראשונה פגיעות זו

OpenSSL היא ספריית תוכנה חופשית שממממשת את פרוטוקול TLS (אבטחת שכבת התעבורה) וקודמו SSL . הספרייה כתובה בשפת C , מממשת פרימיטיבים קריפטוגרפיים רבים ומספקת מגוון רחב של פונקציות והגדרות. כמו כן קיימים ממשקי מעטפת לשפות תכנות אחרות. קיימות גרסאות לרוב מערכות הפעלה.
TLSהוא פרוטוקול קריפטוגרפי שפותח לאבטחת תעבורת רשת האינטרנט ומיישם את תקן X.509 לתשתית מפתח פומבי – PKI (Public Key infrastructure), המשתמש בסרטיפיקטים וחתימה דיגיטלית להעברה ואימות של מפתח סימטרי.

Heartbeat[3]

ה heartbleed הינה חולשה שהתגלתה במנגנון ה heartbeat, לכן אסביר תחילה בקצרה על מנגנון זה.

הקמת חיבור TLS (כאמור לעיל, פרוטוקול המאפשר הצפנת נתונים העוברים בין הצדדים ; נקרא במקור SSL) ובכלליות, הקמה של חיבורים מוצפנים דורשים לא מעט משאבים ולוקחים לא מעט זמן ביחס להקמה של חיבורים שאינם מאובטחים, מפני שבהרבה מקרים הם דורשים לבצע מספר חישובים מתמטיים.
הרעיון של Heartbeat נועד לחסוך את הצורך בהקמה של חיבור כזה כל מספר שניות - הרעיון הוא שלאחר יצירת חיבור ראשוני תוכנת הלקוח תוכל להחזיק את החיבור "חי" גם אם כרגע אין לו צורך לשלוח בו לשליחת מידע.

אם נסתכל ב RFC6520[3] - בעמוד 4 נראה שהמבנה של חבילת Heartbeat כזאת נראה כך :

```
struct {
HeartbeatMessageType type;
uint16 payload_length;
opaque payload[HeartbeatMessage.payload_length];
opaque padding[padding_length];
} HeartbeatMessage;
```

[3] מזכר המתאר את פרוטוקול ה TLS

type - מגדיר את סוג חבילת ה- Heartbeat - יכולה להיות REQUEST_HB_TLS1
או RESPONSE_HB_TLS1 ובשני המקרים, היא לא יכולה להיות גדולה יותר מ-2^14 בתים או כפי שהוגדר ב length_fragment_max- בזמן הקמת התקשורת.

payload_length - מגדיר את גודל ה Payload שנשלח עם החבילה.

payload - מכיל את תוכן החבילה עצמה.

padding - מכיל מידע אקראי לצורך ריפוד החבילה.

במקרים ששרת מזהה כי חבילת ה Heartbeat-הינה REQUEST_HB_TLS1 ע"י כך ששדה ה Type-שווה ל-1, עליו להגיב עם חבילת RESPONSE_HB_TLS1 מתאימה על מנת לשמור על קשר עם הלקוח.

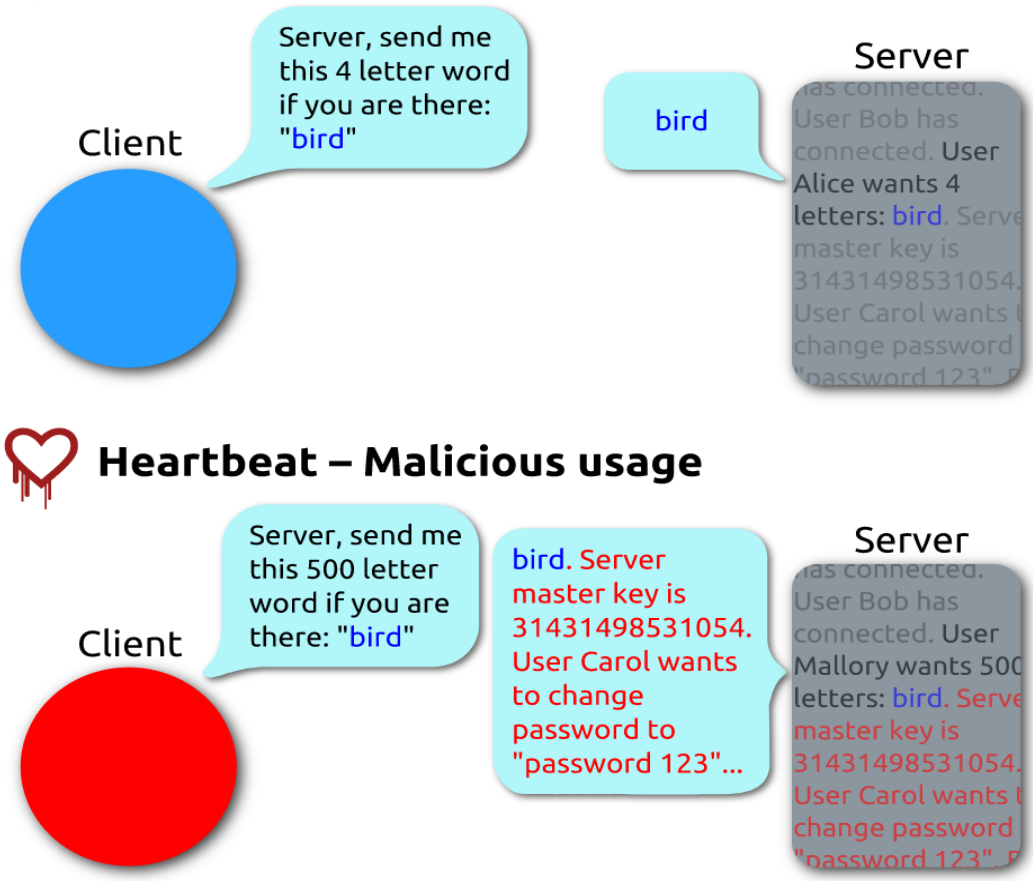
שרת המריץ את הקוד הנמצא בספריית OpenSSL, יבנה את חבילת המענה בצורה הבאה:

```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);

bp = buffer;
bp++ = TLS1_HB_RESPONSE; s2n(payload, bp);
memcpy(bp, pl, payload);
```

כשנתבונן בקוד, בשלב בו memcpy() מעתיקה את המידע מ pl ל pb, נראה כי השימוש ב memcpy() - מתבצע על מנת להעביר payload בתים מ pl-אל, pb , אולם את הערך הקיים ב payload-המשתמש קובע!
מה המשמעות של עובדה זו?
זאת אומרת שלא מתבצעת שום בדיקה בצד השרת ואף אחד לא מבטיח לנו שגודל ה payload שהמשתמש שלח בפועל, אכן תואם את הערך הקיים בשדה length_payload שגם הוא נשלח מהמשתמש ועל פיו אנו קובעים את גודל החבילה שתוחזר למשתמש.
מכאן שהמשתמש יכול לדווח כי הוא שלח משהו בגודל אחד, אך בפועל לשלוח מידע בגודל שונה.

איך אפשר לנצל זאת לרעה?
תמונה אחת שווה יותר מאלף מילים.



Heartbeat session – **איור 1**

נסביר את מה שאנחנו רואים באיור 1[4]:

אם נעבור על התרחיש הרגיל, התהליך אמור להתבצע בצורה הבאה – הלקוח שולח לשרת מחרוזת בעלת X תווים, ומספר המסמל את אורך המחרוזת. שרת מקצה מקום בזיכרון הערימה (heap) של התהליך, שומר מצביע למקום ומכניס לשם את המחרוזת.

כעת על השרת להחזיר את אותה המחרוזת על מנת להגיב לחבילת ה Heartbeat בצורה תקינה.

הוא ניגש לאותו אזור בזיכרון, ושולף מהתא אליו מצביע המצביע למחרוזת X תווים - ומחזיר אותם למשתמש. המשתמש מקבל את המחרוזת - משווה למחרוזת שהוא שלח - ומבין כי השיחה (Session) עדיין חיה.

אולם בתרחיש זדוני יקרה הדבר הבא - תוקף שולח לשרת מחרוזת בעלת X תווים, ומספר Y הגדול בהרבה מאורך המחרוזת. ושוב, שרת מקצה מקום בזיכרון הערימה (heap) של התהליך, שומר מצביע למקום ומכניס

לשם את המחרוזת. כעת, על השרת להחזיר את אותה המחרוזת. הוא ניגש לאותו אזור בזיכרון שאליו מצביע המצביע ושולף Y תווים, Y התווים הנ"ל כוללים את המחרוזת שהתוקף שלח ועוד X-Y תווים אקראיים הנמצאים בהמשך הזיכרון בערימה של התהליך. המשמעות היא לעיתים קריטית - ייתכן שהתווים האקראיים הללו מכילים סיסמאות ונתונים רגישים הנמצאים בזכרון. (על נושא גלישת חוצץ ארחיב בסעיף 2.4)

חשוב לציין:

א. כשל האבטחה במקרה של Heartbleed הוא אינו בפרוטוקול עצמו אלא במימוש שלו.

ב. הבאג כשלעצמו לא מאפשר להשתלט על השרת שמריץ את גרסת ה OpenSSL הפגיעה, אלא אפשרות לקבל מידע חסוי.

2.2 ShellShock (CVE-2014-6271)

__מעטפת__ (באנגלית: shell) הינה רכיב תוכנה המספק ממשק למשתמשים. המונח מתייחס בדרך כלל למעטפת של מערכת הפעלה, המספקת גישה לשירותי הליבה. עם זאת, המונח משמש גם לציון יישומים ה"בנויים מסביב" לרכיב מסוים, לדוגמה דפדפנים ותוכניות לדואר אלקטרוני המשמשות כ"מעטפת" למנוע תצוגה מבוסס HTML. למערכות הפעלה יש בדרך כלל שני סוגים של מעטפות: שורת פקודה וגרפית. מעטפת שורת הפקודה מספקת ממשק שורת פקודה (CLI) למערכת ההפעלה, ואילו המעטפת הגרפית מספקת ממשק משתמש גרפי (GUI). בשתי הקטגוריות, היעוד העיקרי של המעטפת הוא לשפעל או "לשגר" תוכניות אחרות, אך עם זאת, יש למעטפות לרוב גם יכולות נוספות כגון הצגת ספריות וקבצים.

__באש__ (באנגלית: Bash) משמעות השם היא ראשי תיבות של Bourne-again shell . היא מעטפת פקודה למערכות UNIX ומערכות דמויות יוניקס.

Shellshock (נקרא גם Bashdoor) היא משפחה של באגים הקשורים לאבטחת תוכנה במחשבים הקיימות ב-Unix bash shell הראשון שבהם נחשף ופורסם בספטמבר 2014.

שירותי אינטרנט רבים, עושים שימוש ב Bash כדי לבצע פעולות, אשר עשויות להיות מנוצלות על ידי מתקיף כדי לגרום ל Shell לבצע פקודות ערוכות שלו.

על ידי כך ניתנת אפשרות לכאורה למתקיף להשיג גישה למערכות ממוחשבות מוגבלות גישה.
סטפן שאזלאס (Stéphane Chazelas) יצר קשר עם מתחזק ה Bash, צ'ט רמיי (Chet Ramey) ב-12 בספטמבר כדי לדווח לו על פרצת האבטחה. הוא קרא לפרצה זו "Bashdoor". בעבודה משותפת עם מומחי אבטחת מידע, הם יצרו גרסה אשר מטפלת בפרצה האמורה.
ב-24 בספטמבר יצאה הודעה לציבור על הבעיה כאשר כבר יצאה לאור הפצה עם תיקון.[5]

אז מהו בעצם Shellshock ?

במערכת מבוססת לינוקס ישנה האפשרות באמצעות ה Bash ניתן ליצור סקריפטים המבצעים פעולות שונות וכן להגדיר משתני סביבה על מנת להעביר ערכים ספציפיים עבור סקריפט Bash וכן להשתמש במשתנים אלו באופן חוזר במהלך הריצה של המערכת.
למעשה ניתן להעביר לא רק משתנים, אלא אף להריץ רצף פקודות ותרחישים לביצוע.

דוגמא להגדרת משתנה סביבה:

```
func
{
   echo 'let's define our environment variable'
   env a = 'this is environment variable value'
  return 0
}
func
echo $a
```

מכיוון שהמשתנה הוגדר כמשתנה סביבה הקוד ידפיס את ערכו של משתנה הסביבה a, למרות שהוא הוגדר ב scope של הפונקציה.

עתה ניראה דוגמא להגדרת משתנה סביבה המכיל פונקציה.

```
env f = '() { echo "This is a cool function";}'
```

כחלק מהיכולת הזו נתגלתה הפגיעות המאפשרת לבצע הגדרת משתנה סביבה, אשר יגרום למערכת להריץ קוד.

ניצול הפגיעות[6]

הגדרת משתנה סביבה:

```
env x='() { :;}; echo Areli' bash -c "echo Fallach"
```

הפקודה תתבצע על ידי ה Bash בצורה הבאה:

1. תחילת ההגדרה env = x בעצם מתחילה להגדיר משתנה סביבה בשם x.
2. לאחר מכן המשתנה x מתחיל ונגמר בפונקציה ריקה שאינה מבצעת כלום.
3. בסוף הפונקציה מופיע הקוד: "echo Areli"
4. ולאחר סיום הגדרת משתנה הסביבה מופיעה הקריאה לפקודת bash עם הדגל c- שמבצעת הרצת הפקודה שבאה לאחר מכן: " echo Fallach"

הקוד אשר אינו אמור להתבצע הוא הקוד המתואר בסעיף 3.

לאחר הגדרת המשתנה נוכל לפנות ל Bash על מנת להריץ פקודות מסויימת, פקודות אלו יעברו במנגנון הרגיל וה Bash יוכל להגביל אותן כמו פקודות שרצות משורת הפקודה, אולם ה Bash לא מזהה את הפקודה המשורשרת להגדרת הפונקציה ובשלב טעינת משתנה הסביבה הוא מריץ אותה כאשר הוא חושב שהפקודה הינה חלק מהגדרת המשתנה.

לאחר התיקון הקונסול יראה כך: (ראה איור 2)

```
root ~ > env x='() { :;}; echo Areli' bash -c "echo Fallach"
bash: warning: x: ignoring function definition attempt
bash: error importing function definition for`x`
Fallach
root ~ >
```

איור 2 – תגובת קונסול תקינה עבור ניסיון ניצול פגיעות ShellShock

2.3 Poodle (CVE-2014-3566)

בחודש אוקטובר של שנת 2014 נתגלתה מתקפה אפשרית חדשה שקיבלה את השם Poodle.
מתקפה זו מאפשרת לתוקף לגלות סוד הנמצא בבקשת לקוח העוברת בטווח תקשורת המוצפן באמצעות SSL.
בכך, למשל,יכול תוקף לגנוב ממשתמש את הCookie - שלו גם אם העוגייה הוגדרה עם מאפייני אבטחה כמו Secure ו HTTP Only .

מאפייני אבטחה Secure ו Http Only בעוגיות (Cookies)

Secure – כאשר הדגל secure מסומן, העוגייה לא תשלח על גביי תקשורת HTTP אלא רק על גביי תקשורת HTTPS.
HttpOnly – כאשר הדגל HttpOnly מסומן, העוגייה תמנע גישה ע"י תרחיש (Script) צד לקוח.

החולשה[7][8]

השם POODLE הינו ראשי תיבות של Padding Oracle On Downgraded Legacy Encryption
וכמו שהשם מרמז, המתקפה מבוססת על מניפולציות שהתוקף מבצע על ה - Padding בפרוטוקול.
החולשה נמצאת למעשה באופן שבו הפרוטוקול SSL 3.0 מבצע את התהליך ה Padding ואינו חלק מה-MAC, לפיכך לא מתבצעת בדיקה על ה Padding האם הוא זהה לערך שנשלח ע״י המשתמש.

קוד אימות מסרים MAC(Message Authentication Code)

שם כולל לפונקציות עם מפתח סודי המשמשות לאימות מסרים .פונקציית MAC מקבלת מפתח סודי ומסר באורך שרירותי ומפיקה פיסת מידע קצרה הנקראת תג אימות, והוא נשלח לצד המקבל יחד עם המידע המאומת או בנפרד. המקבל יכול בעזרת אלגוריתם מתאים לוודא באמצעות התג שקיבל שהמסמך אותנטי.
אלגוריתם קוד אימות מסרים הוא סימטרי, במובן שהשולח והמקבל חייבים לשתף ביניהם מראש מפתח סודי, באמצעותו יכול המקבל לוודא שהמסמך הגיע מהמקור שהוצהר וכי לא נעשה כל שינוי בתוכנו במהלך ההעברה. היות שלא ניתן להכין תג אימות מתאים ללא ידיעת מפתח האימות הסודי, אם נעשה שינוי כלשהו בתוכן המסר, לא יצליח היריב לשנות גם את התג בצורה מתאימה, ולכן המקבל יבחין בשינוי בסבירות גבוהה מאוד, ידחה את המסר המזויף על הסף, ויעביר הודעה מתאימה לשולח.

משום שה – Padding אינו חלק מחלק המחושב ב – MAC, ניתן לערוך אותו וע"י כך לגלות מידע סודי הנמצא בבקשת ה – Request.

הצפנת SSL(3.0) מתבצעת במפורט להלן.

הלקוח לוקח את ההודעה הגלויה ומחשב עבורה את ערך ה MAC. לאחר מכן מוסיפים את ערך ה Padding ומשלימים למספר הביטים הנדרש במסגרת ההצפנה, כאשר הביט האחרון ב Padding משמש לספירת גודל ה Padding.
בשלב האחרון מצפין הלקוח את ההודעה באחת מהשיטות השונות של ההצפנה ב SSL כך שכל חבילה נראית כך:

[((Message in plain text) $_{\text{HMAC-SHA-1}}$) + padding]AES128

דוגמא

נניח והבקשה שלנו היא:

```
My secret password is So8dYAd14V. I want you to log me in.
```

לפני ההצפנה ההודעה שלנו נראית כך (בהקסדצימלי כאשר כל שורה היא 16 ביטים):

```
4d79 2073 6563 7265 7420 7061 7373 776f  My secret passwo
7264 2069 7320 536f 3864 5941 6431 3456  rd is So8dYAd14V
2e20 4920 7761 6e74 2079 6f75 2074 6f20  . I want you to
6c6f 6720 6d65 2069 6e2e 0a bc 3a1f 0cf3  log me in.
02c5 80dd c869 66c0 1b2c 2a53 3c9d 5b00
```

כאשר הקטע המודגש מ bc עד 5b הינו התג של HMAC_SHA1 שהוסף אל ההודעה ו 00 מסמל את הגודל ה – Padding.

ולאחר ההצפנה, ההודעה תיראה כך:

```
33c5 3aa2 08a4 0ecf c408 c6df 0c7d ac47
c9fb a2e4 54c8 a316 78de 1ec2 cc6e 9600
0572 dcfd 25fc c941 e9fd 6ea9 9ca2 9e50
e4c3 5bc9 f4c6 d973 2412 03fb 1615 be93
7faf 2119 c740 becc 9095 c595 034a 6a61
```

נניח שהחלטנו לקחת את ההודעה המוצפנת הזו ולשנות בה את הביט האחרון בשורה הראשונה מ 47 ל 46,
לאחר שנבצע את הפענוח נקבל את ההודעה הבאה:

```
860b 238a 9fc2 b909 6dd6 7642 05a8 85fe
7264 2069 7320 536f 3864 5941 6431 3457  rd is So8dYAd14W
2e20 4920 7761 6e74 2079 6f75 2074 6f20  . I want you to
6c6f 6720 6d65 2069 6e2e 0abc 3a1f 0cf3  log me in.
02c5 80dd c869 66c0 1b2c 2a53 3c9d 5b00
```

השורה הראשונה נהרסה משום שערכנו את המידע, אך אפשרי לראות עכשיו כי האות האחרונה בסיסמא השתנתה מ V ל w.

הדבר נובע מכך שרוב שיטות ההצפנה של SSL3.0 עובדת בשיטת CBC.

מצבי הפעלה עבור הצפנת בלוקים

ECB (Electronic CodeBook)

במצב פעולה זה נחלק את המסר לבלוקים בגודל n סיביות ונצפין כל בלוק בפני עצמו. במקרה שאורך המסר לא מתחלק ב n, הבלוק האחרון ירופד.
הצפנה:
עבור כל בלוק J במסר x בצע:

$$Cj = Ek\ (xi)$$

פענוח:
עבור כל בלוק מוצפן cj בצע:

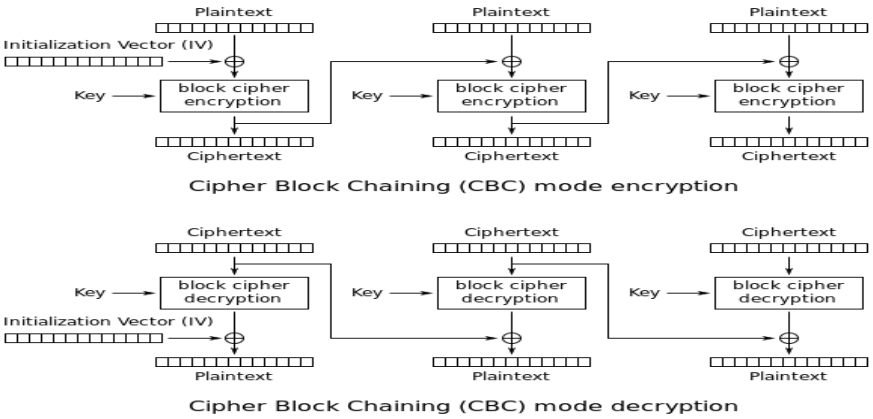
$$Xj = E\text{-}1k\ (cj)$$

תכונות:

ביטחון: בלוקים של טקסט-גלוי זהים מפיקים תמיד בלוקים של טקסט-מוצפן זהים - כל עוד מפתח ההצפנה זהה. לכן קל לזהות שהוצפנו בלוקים זהים. (חסרון)
כשל מצטבר: שגיאה בסיבית אחת או יותר של בלוק טקסט-מוצפן משבשת את פענוח הבלוק הגלוי המתאים בלבד. יתר הבלוקים אינם משתבשים. פענוח של בלוק המכיל סיבית שגויה אחת יהיה אקראי, דהיינו בממוצע 50 אחוז מסיביות הבלוק יהיו שגויות. (יתרון)
תלות הדדית; הבלוקים מוצפנים בנפרד ללא תלות זה בזה. שינוי סדר הבלוקים המוצפנים גורר אחריו לאחר פענוח שינוי בסדר הבלוקים הגלויים בהתאם. סידור מחדש או הסרה זדונית של בלוקים שלמים עלולים שלא להתגלות. (חסרון)

CBC (Cipher-Block Chaining)

מצב הפעלה המוסיף ווקטור אתחול IV באורך n סיביות להצפנה. במצב זה כל בלוק טקסט-קריא מחובר לפני שהוא מוצפן בחיבור XOR עם תוצאת הצפנת הבלוק הקודם.
וקטור האתחול הוא ערך אקראי חד-פעמי. הוא אינו סודי ואינו נחשב למפתח הצפנה. כדי שהמקבל יצליח לפענח את הטקסט המוצפן הוא צריך לקבל את וקטור האתחול במצב גלוי, יחד עם הטקסט המוצפן.



איור 3 – הצפנת בלוק בשיטת CBC[9]

נחזור לדוגמא, ניקח את ההודעה ונוסיף לה בלוק חדש של Padding.
מכיון שבפרוטוקול SSL3.0 ה Padding הם ערכים רנדומליים ואין עליהם בדיקה, נוכל להשתמש באילו ערכים שנרצה.
נצפין את ההודעה, ניקח את המידע המוצפן ונעתיק את השורה בה מופיעה הסיסמא שלנו לסוף ההודעה במקום השורה האחרונה הנוכחית בתור ה Padding שלנו:

```
33c5 3aa2 08a4 0ecf c408 c6df 0c7d ac47
c9fb a2e4 54c8 a316 78de 1ec2 cc6e 9600
0572 dcfd 25fc c941 e9fd 6ea9 9ca2 9e50
e4c3 5bc9 f4c6 d973 2412 03fb 1615 be93
c9fb a2e4 54c8 a316 78de 1ec2 cc6e 9600
```

כשנבצע פענוח להודעה כעת נקבל את התוצאה הבאה:

```
4d79 2073 6563 7265 7420 7061 7373 776f  My secret passwo
7264 2069 7320 536f 3864 5941 6431 3456  rd is So8dYAd14V
2e20 4920 7761 6e74 2079 6f75 2074 6f20  . I want you to
6c6f 6720 6d65 2069 6e2e 0abc 3a1f 0cf3  log me in.
a562 4102 8f42 84d3 d87e 9c65 7e59 2682
```

אנו רואים שהערך שמייצג את אורך ה Padding הוא 82. איך הגענו ל 82?
בתהליך ההצפנה אנו עושים XOR עם הערך 93 שהוא בבלוק שמעל האחרון. בהצפנה המקורית ה-XOR נעשה עם הערך 47 (ניתן להסתכל באיור הראשון המציג את המצב לאחר ההצפנה, בערך שבסוף השורה הראשונה).
עכשיו באמצעות חישוב מתמטי פשוט נוכל לגלות מה הערך: במידה ו - X האות האחרונה בסיסמא אותה אנחנו רוצים לגלות, נבצע את החישוב הבא:

$$X \text{ XOR } 47 \text{ XOR } 93 = 82 \Leftrightarrow X \text{ XOR } D4 = 82$$

על פי החוק הבא:

$$X \text{ XOR } A = Y \Leftrightarrow X = Y \text{ XOR } A$$

נקבל:

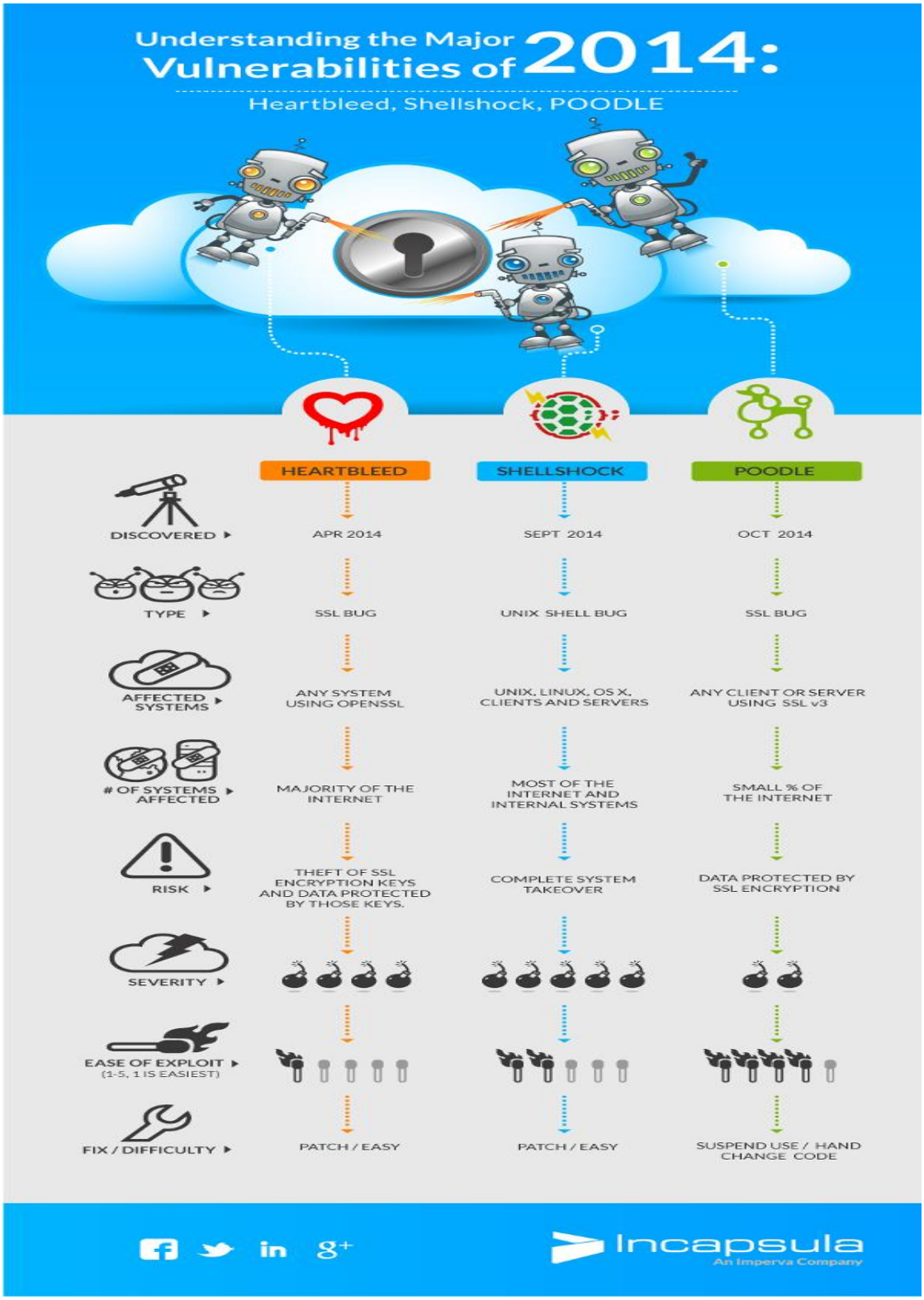
$$X = D4 \text{ XOR } 82 \Leftrightarrow X = 56.$$

56 ב Hex משמעותו V, כפי שניראה זוהי באמת האות האחרונה מהסיסמא המקורית שלנו.[4]
נמשיך בשיטה זו ונוכל לגלות אות אחר אות את הסיסמא כולה.

לצורך הגנה מפני פגיעות זו, ראשית אנו נוודא שאנחנו לא משתמשים ב SSL (3.0) אלא בחלופות מאובטחות יותר כגון TLS.
במקרה הזה חשוב לבדוק בנוסף שמערכות קיימות לא מאפשרות התחברות בפרוטוקול זה על מנת לשמור על תאימות לאחור (backward compatibility).

[4] כשאין לנו מידע על החלק המפוענח, נצטרך לבצע עוד מספר חישובי Xor על מנת להגיע לתוצאה

ולסיכום[10]**:**



איור 4 – סיכום פגיעויות שנת 2014

2.4 GHOST (CVE-2015-0235)

בחודש ינואר בשנת 2015 חוקרי האבטחה של חברת Qualys גילו שהפונקציה _nss_hostname digits_dots שבספריית glibc פגיעה ל גלישת חוצץ (buffer overflow) הניתן לניצול גם מרחוק.
הפגיעות נמצאת ב glibc-2.2 שיצאה בנובמבר 2000 הפגיעות תוקנה במאי 2013 אך לא זוהתה באותו זמן כבעיית אבטחה, לכן הפצות מערכות ההפעלה המוכרות לא הטמיעו את העדכון בגרסאות ששוחררו מאז ועד אשר התפרסמה פגיעות זו. [11]

ספריית gilbc

The GNU C Library, commonly known as glibc, is the GNU Project's implementation of the C standard library. Despite its name, it now also directly supports C++ (and, indirectly, other programming languages). It was started in the early 1990s by the Free Software Foundation (FSF) for their GNU operating system.
Released under the GNU Lesser General Public License.
glibc is free software.
glibc was adopted by popular Linux distributions – Centos, Ubuntu, Debian, Red Hat.

על מנת שתיאור הפגיעות יהיה יותר קל להבנה, אסביר מהי גלישת חוצץ (Buffer overflow) באופן כללי.

גלישת חוצץ (Buffer overflow)[12]

כל תוכנית מחשב תקרא ותכתוב מאזור זיכרון כלשהו.
שגיאת תכנות המתבטאת בכך שתוכנית מחשב כותבת לאזור בזיכרון המחשב (החוצץ) יותר מידע מאשר אותו אזור מסוגל להכיל. כתוצאה מכך "גולש" חלק מהמידע אל מחוץ לגבולות החוצץ, ומשנה נתונים שלא היו אמורים להשתנות. המידע שנמחק לעתים קרובות הכרחי להמשך ריצתה התקינה של התוכנית, ובשל כך גלישת חוצץ עלולה לגרום לתוכנית להחזיר תוצאות לא נכונות, לקרוס לחלוטין, או אף לאפשר הרצה של "קוד זדוני" הגורם לתוכנית לפעול באופן שלא תוכנן מראש. בשל כך, גלישות חוצץ מהוות בסיס לפרצות אבטחה רבות. ברוב המקרים, גלישת חוצץ תגרום לקריסה או התנהגות לא רצויה אחרת.

מרגע שהתרחשה גלישת חוצץ, מהלך התוכנית "איננו מוגדר", כלומר לא ניתן להבטיח דבר לגבי המשך ריצת התוכנית: היא עשויה לקרוס, לרוץ באופן תקין, להמשיך לרוץ ולתת תוצאות שגויות, או לגרום נזק.

לעתים קרובות מאוחסן המידע הדרוש להבטחת ריצתה התקינה של תוכנית בסמוך לחוצצים בה היא משתמשת. גלישת חוצץ עשויה לגרום למחיקת מידע זה והחלפתו במידע אחר. פרט לפגיעה בריצתה התקינה של התוכנית, הדבר עלול לסייע לוירוס מחשב "להשתלט" על ריצת התוכנית ולהריץ קטעי קוד זדוניים וזאת על ידי כתיבה על אזור בזיכרון המכיל קוד תוכנה שמתבצע. גלישת חוצץ הגורמת לקריסה של התוכנית, ללא הרצת קוד זדוני, תגרום למצב של "מניעת שירות" (Denial of Service).

במקרה שהחוצץ נמצא במחסנית קריאות, הזיכרון שמוקצה לחוצץ נמצא בסמיכות לתא הזיכרון שמכיל את כתובת הזיכרון שאליה יש לחזור לאחר סיום הפונקציה, לצורך המשכה התקין של התוכנית. אם ידוע כי בתוכנית קיים כשל של גלישת חוצץ, ניתן לעתים לגרום לכך שתוכנו של תא זיכרון זה ישתנה, והוא יצביע על כתובתו של קטע קוד זדוני. סוג זה של גלישת חוצץ נקרא גם "גלישת חוצץ במחסנית" (Stack buffer overflow).

כאשר החוצץ נמצא בערימה, הוא לרוב אינו נמצא בסמיכות לכתובות חזרה כל שהן, ולכן באופן כללי ניתן לומר כי קשה יותר לנצל גלישת חוצץ בערימה לטובת הרצת קוד זדוני מאשר במחסנית. סוג זה של גלישת חוצץ נקרא גלישת ערימה (Heap overflow).

גלישת החוצץ ב GHOST[13]

גלישת החוצץ בספריית glibc היתה גלישת ערימה (heap overflow).

כחלק מפונקציית nss_hostname_digits_dots(), אשר מטרתה להימנע מקריאות מיותרת לגילוי ה host name באמצעות DNS lookups במקרה שהפרמטר hostname כבר מיוצג ככתובת IP (IPv4 or IPv6), ישנה הכנה של אזור כתיבה על פי הפרמטרים המתקבלים.

```
__nss_hostname_digits_dots (const char *name, struct hostent *resbuf, char **buffer, size_t
*buffer_size, size_t buflen, struct hostent **result  enum nss_status *status, int af, int *h_errnop)
.
.
.
 85 size_needed = (sizeof (*host_addr)  + sizeof (*h_addr_ptrs) + strlen (name) + 1);
 ..
 88   if (buffer_size == NULL)
 89         {
 90           if (buflen < size_needed)
 91             {
 ..
 95               goto done;
```

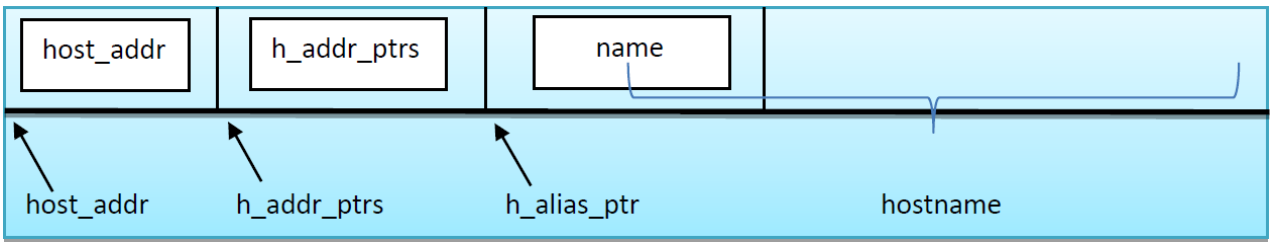
```
 96              }
 97          }
 98        else if (buffer_size != NULL && *buffer_size < size_needed)
 99          {
100            char *new_buf;
101            *buffer_size = size_needed;
102            new_buf = (char *) realloc (*buffer, *buffer_size);
103
104            if (new_buf == NULL)
105              {
...
114                goto done;
115              }
116            *buffer = new_buf;
117          }
...
```

בשורות הבאות אנו רואים שכחלק מהקוד מכינים מצביעים לאחסון 4 ערכים שונים –

Host_addr, h_addr_ptrs, h_alias_ptr, and hostname.

```
121        host_addr = (host_addr_t *) *buffer;
122        h_addr_ptrs = (host_addr_list_t *)
123          ((char *) host_addr + sizeof (*host_addr));
124        h_alias_ptr = (char **) ((char *) h_addr_ptrs + sizeof
(*h_addr_ptrs));
125        hostname = (char *) h_alias_ptr + sizeof (*h_alias_ptr);
```

אולם אפשר לראות שגודלו של h_alias_ptr* – גודלו של מצביע char – לא נמצא בחישוב של size_needed המובא למעלה בשורה 85, ומכיון שמיקום המצביעים נקבע על פי הגדלים של שאר המצביעים, יוצא שכאשר ממקמים את המצביעים ב buffer שהוקצה בשורה 102 נקבל הצבעה משובשת, הנראית כך:



אנו רואים שמכיון שגודלו של h_alias_ptr* לא נכלל בחישוב גודל ה buffer, אבל כן הוצב מצביע עבור ערך זה לכאורה – ישנה גלישה של ה buffer לאזור שלא מוקצה עבורו על ידי הערך name.

בשימוש נכון, באמצעות פגיעות זו אנו יכולים לגרום לתוכנית להריץ קוד שאנחנו נשתול בה.[5]

[5] ניצול הפגיעות מתואר באופן מלא בפרק 5 במסמך תיאור הפגיעות [13]

בפגיעות זו אנו רואים שגיאה בכתיבת קוד העוסק בנושא רגיש – בניית חוצצים והקצאות זיכרון בספריית קוד פתוח שאומצה על ידי גרסאות רבות של מערכת הפעלה Linux.

2017 – הלו?

2.5 WhatsApp & Telegram vulnerability

בחודש אפריל של שנת 2016, חברת פייסבוק פירסמה שמעתה תוכנת WhatsApp המשמשת לשיחת וידאו והעברת הודעות טקסט על גביי האינטרנט, תשתמש בהצפנה מקצה לקצה.

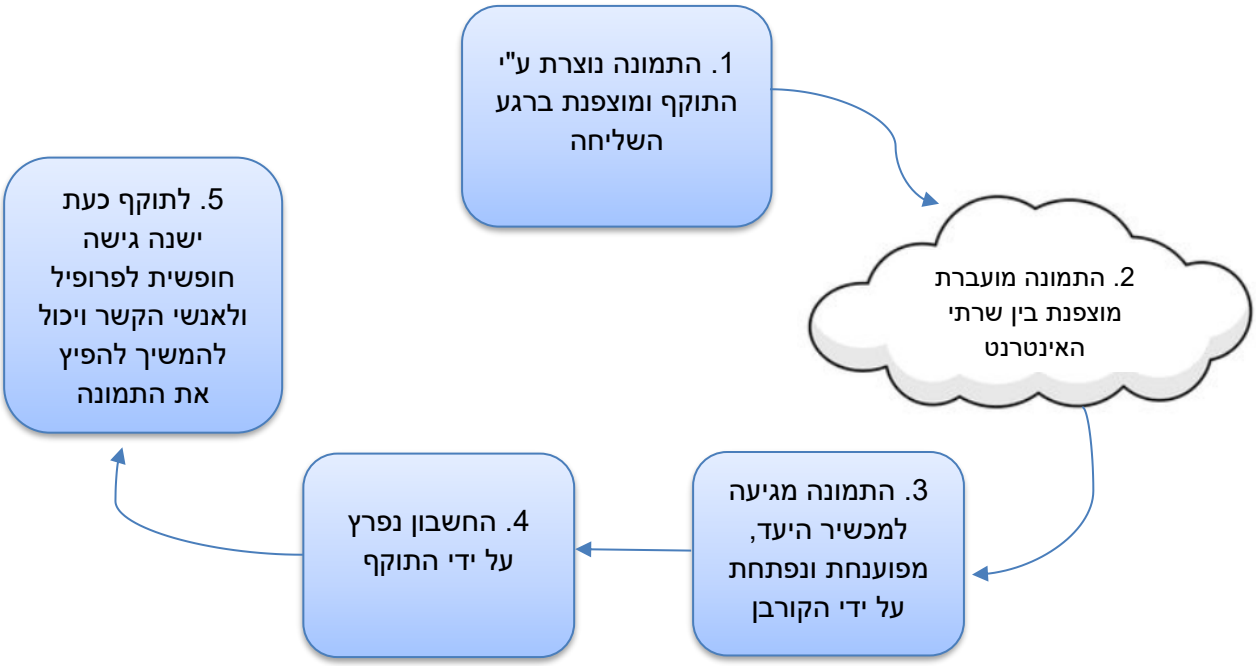
__הצפנה מקצה-לקצה__ - מערכת תקשורת שבה רק הצדדים המשתתפים בשיחה יכולים לקרוא את ההודעות. למעשה, נמנעת האזנה של צד שלישי וגורמים שאינם מהווים חלק מן השיחה - כגון: ספקי אינטרנט, ספקי תקשורת סלולרית, ואפילו מספק התקשורת עצמו, מלגשת למפתחות ההצפנה הדרושים כדי לפענח את השיחה. מערכות אלו נועדו למנוע כל ניסיון של מעקב או שינוי של המידע המועבר משום שאף צד שלישי לא יכול לפענח את הנתונים שמועברים או מאוחסנים. למשל, חברות שמיישמות הצפנה מקצה לקצה אינן יכולות למסור טקסטים של לקוחותיהם לרשויות.

אולם במרץ 2017 התברר שזו "אליה וקוץ בה".

חברת Checkpoint פירסמה שנתגלתה פירצה במנגנון ההצפנה של תוכנות whatsapp ו Telegram, פירצה זו מאפשרת לתוקף לקבל שליטה מלאה על חשבון המשתמש, לשלוח הודעות בשמו ולגשת למידע האישי של המשתמש הכולל תמונות, סרטונים והתכתבויות בקבוצות שונות.

הפגיעות באה לידי ביטוי בצורה הבאה:

התוקף שולח קובץ המכיל קוד זדוני, ברגע השליחה - הקובץ הנגוע יעבור הצפנה ויישלח ליעד, במכשיר הקורבן המכשיר יפענח את הצפנת הקובץ ויאפשר למשתמש לפתוח את הקובץ. ברגע שהמשתמש יעשה זאת יורץ הקוד הזדוני המוכל בקובץ ויאפשר לתוקף גישה לתוכן השמור במכשיר הקשור לאפליקציה ואפשרות להמשיך להפיץ את הקובץ הנגוע בשם המשתמש. הסכימה מתוארת באיור 5.



איור 5 – סכמת התקיפה

ניתוח טכנולוגי[14]

התוקף ייצור קובץ כדוגמת הקובץ הזה:

```
<html>
          <header>
                    <title>WhatsApp</title>
        <script>
        function GetStorage()
        {
            var values = {};
            var keys = Object.keys(localStorage);
            var i = keys.length;
            while ( i-- )
            {
                        values[keys[i].replace(/ /g, '+')] = localStorage.getItem(keys[i]).replace(/
/g, '+');
            }
            return values;
        }

        //send data to attacker server
        function sendacct(data) {
            var xhttp = new XMLHttpRequest();
            xhttp.open("POST", "https://www.AttackerWebsite.com/whatsapp.php", true);
            xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
            xhttp.send("account_data=" + data);

        }//end of sendacct

        var result = GetStorage();
        var json = JSON.stringify(result);
        sendacct(json);

        </script>
    </header>
    <body>
        <img src="https://s-media-cache-
ak0.pinimg.com/736x/f4/49/bc/f449bc4db763ba65378ba659fe7fc865.jpg" />
    </body>
</html>
```

אפשר לראות בקובץ HTML הנ״ל מספר אלמנטים חשובים.
פונקציית GetStorage ניגשת לאחסון של האפליקציה ואוספת את כל התוכן הקיים שם.
פונקציית sendaact שולחת מידע ל web service כלשהו שנוצר על ידי התוקף.

אולם, מה שיוצג למשתמש זה הקוד המוכל בחלק Body - התמונה המוחזרת בכתובת URL:

https://s-media-cache-ak0.pinimg.com/736x/f4/49/bc/f449bc4db763ba65378ba659fe7fc865.jpg

בשליחת הקובץ ע״י התוקף, הפונקציה encryptE2Media תצפין את הקובץ ותשלח את ה Blob לשרת WhatsApp ומשם לחשבון ה whatsapp של הקורבן. כשהמשתמש יפתח את קובץ התמונה ה״תמים״, הקוד הזדוני המוכל בקובץ ה HTML יבוצע ומעתה יש לתוקף גישה לכל המידע הנמצא תחת web.whatsapp.com. כאשר שבידו של התוקף כל רכיבי חשבון הקורבן יכול התוקף לטעון אותם ולהריץ על המחשב שלו את חשבון הקורבן.

התיקון לפגיעות זו הינו פשוט כמובן – הצפנת קצה לקצה הינה חשובה, אולם חשיבות בדיקת הקובץ לפני ההצפנה הינה חשובה אף היא.

3. אפיון וקטלוג פגיעויות ותקיפות

למרות ששטח התקיפה במערכות מחשבים הוא רחב, על מנת לנסות ולהתגבר מפני ניצול פגיעויות אלו ותקיפת המערכת, אנו נדרשים ליצור מטריה כלשהי המכילה את הסוגים השונים.
המטרה היא שבאמצעות סיווג נכון של תקיפות ופגיעויות מוכרות, נוכל לקטלג ולרכז את אזורי הפגיעות למספר קטן ככל שניתן על מנת שתוקל עלינו בניית ההגנה למערכת, תוך סבירות גבוהה לכיסוי מירבי.
כותבי המאמר "A survey on Taxonomies of Attacks and Vulnerabilities in Computer Systems"[15] משתמשים בקטלוג מעין זה על מנת ליצור מן רשימת בדיקה המשמשת להערכת אבטחת המערכת.
המאמר מציע סקירה מפורטת של בניית סיווג של התקפות ופגיעויות במערכות מחשב. פרט לסיווגם של פגיעויות ותקיפות, המאמר בוחן את היעילות השימוש בהן בזמן הערכת אבטחה של מערכת.
לסיכום, המאמר מצביע על המאפיינים המשמעותיים של צורות חלוקה שונות כדי לספק מבנה לארגון מידע על התקפות ופגיעות מוכרות על מנת לבחון בצורה נכונה ולסייע להליך הערכת האבטחה.

כותבי המאמר הגדירו בצורה יפה את המושג "פגיעות".

הפגיעות היא חולשה - נקודה, המאפשרת לתוקף להפחית את בטחון המערכת ולהשפיע על פעילותה.
פגיעות הינה שילוב של שלושה יסודות:

1. שגיאה או התנהגות בלתי מתוכננת במערכת
2. גישת התוקף לליקוי
3. ופוטנציאל התוקף לנצל את החולשה

תוקף צריך לפחות מכשיר אחד או שיטות מתאימות שיכולים לצל נקודת תורפה במערכת. במסגרת זו, הפגיעות ידועה גם כמשטח ההתקפה.
ניהול פגיעות הוא ביצוע מחזורי של הכרה, ארגון, תיקון, ומיתון.
חלון הזמן של פגיעות הינו מהרגע שנוצר באג אבטחה עד אשר הגישה אליו הוסרה, תיקון האבטחה מומש או שהתוקף נוטרל.

המאמר מציג את החלוקה הבאה:

א. פגיעויות חומרה
ב. פגיעויות תוכנה
ג. פגיעויות מידע
ד. פגיעויות רשתות
ה. פגיעויות גישה
ו. פגיעויות אנוש

כפי שאפשר ללמוד מהמאמר, הכותבים שילבו פגיעויות וצורות תקיפה, כמו כן חלקן של הקטגוריות נובעות משגיאות זהות.
דבר נוסף, הרשימה מכילה אלמנטים שאינם טכנולוגיים.
רשימה זו אכן מועילה בצורה נושאית על מנת לחשוב על דברים המסכנים את המערכת, אולם בעקבות סקירה מדגמית וניתוח טכנולוגי של פגיעויות מהשנים האחרונות שנעשתה בפרק הקודם, אתקדם עוד צעד מהקטלוג המוזכר במאמר ואצמצם את קטלוג תחומי פגיעויות המחשבים ל 3 קטגוריות עיקריות - חלוקה על פי אזור הפגיעות.
הקטלוג ישמש על מנת להישמר מפני יצירת פגיעויות בזמן בניית המערכת, יעזור במיקוד ויהווה חלק בלתי נפרד משלבי תהליך הפיתוח המאובטח.
לעניות דעתי שלושת קטגוריות אלו מכסות את חלקן הארי של הפגיעויות והתקיפות.[6]

3.1 שכבת מערכת ההפעלה [7]

מבלי להיכנס להבדלים בין מערכות הפעלה השונות, נשים דגש של הסתכלות על מערכת ההפעלה כאזור שבו מורצת המערכת שלנו.

פגיעויות אי הקשחת המערכת

הפצתן של מערכות ההפעלה הינה רחבה והשימוש בהן בא לידי ביטוי בסוגי מערכות שונות אשר אין קשר בייעוד מערכת אחת לחברתה.
נובע מכך שכל התקנת מערכת הפעלה תדרוש את תשומת הלב הראויה והדרישות המתאימות (ולא מעבר) עבור המערכת שאותה היא עתידה לשרת.

קביעת תצורה נכונה, חלוקת הרשאות מתאימה, הורדת שירותים לא נצרכים, הוספת אמצעי הגנה נדרשים ובדיקת של אבטחת המערכת – כל אלו אלמנטים של הקשחה בהתקנת מערכת ההפעלה, אולם הקשחת מערכת ההפעלה נדרשת גם במהלך החיים של המערכת בעדכוני מערכת ההפעלה ונתינת זכויות מוגבלות בהתאם לצורך.

פגיעויות ניהול זיכרון

זיכרון הוא משאב חשוב מאוד הדורש הגנה. הדוגמה הפשוטה ביותר היא הגנה על אזור של משתמש אחד בזיכרון מגישה של משתמש אחר (לקריאה או כתיבה). דוגמה אחרת היא הגנה על קוד תכנית מפני גישה שגויה של המשתמש עצמו, ודוגמה שלישית היא הגנה על זיכרון מערכת ההפעלה שהוא אזור רגיש מאוד.
בכדי שמערכת ההפעלה תוכל לספק הגנה על זיכרון, נדרש שיתוף פעולה בין מערכת ההפעלה לבין המעבד, אשר אחראי על הגישה בפועל לזיכרון.

[6] העבודה מתרכזת בפגיעויות תוכנה, להוציא פגיעויות בשכבת החומרה
[7] בפרק זה נעשו שימושים בחומרי הקורס אבטחת מערכות תוכנה (22923)

פגיעויות ניהול תהליכים

בהגנה על תהליכים נדרש מציאת האיזון המתאים בין גישה של התהליך לכל המשאבים שהוא צריך ומאידך מניעת הגישה עבור שאר המשאבים, נצטרך להגן על תהליך מערכת ההפעלה מפני תהליכי משתמש – מוכרים ושאינם מוכרים ונצטרך להגן על תהליך המשתמש מפני תהליכי הגנה שונים.

פגיעויות חשיפת נתונים וקבצים

הגנה על קבצים היא חלק מההגנה על כל סוגי העצמים והמשאבים.

האחריות על הגנת הקבצים מתחלקת בין מנגנוני ההגנה של מערכת הפעלה ובין השתמש, מכיוון שחלק מההרשאות עבור הקובץ ניתנות על ידי הבעלים.

לדוגמא: הגנה באמצעות סיסמא, אי אפשרות מחיקת או העברת קבצים, הרצות מבוקרות של קבצים הניתנים להרצה.

3.2 שכבת האפליקציה

בקטגוריה זו ישנן אפשרויות לפגיעויות רבות, בהתאם לאפיון האפליקציה ולתפקידה, אתייחס לתחומים הנפוצים ביותר.

שכבה זו חולקה לשתי קטגוריות:

א. רשת ותקשורת

ב. קוד פעולת היישום

3.2.1 רשת ותקשורת [16]

פגיעויות פרוטוקולים

מעצם היות רשת האינטרנט בנויה משכבות על גבי שכבות המכילות פרוטוקולים שונים עם פרמטרים רבים, ניתן לנצל לרעה אזורים שנכתבו מבלי לשים דגש על אפשרויות תקיפה בעתיד.

הדבר נכון גם עבור פיתוח ייעודי של פרוטוקול תקשורת עברו מערכת ספציפית, ככל שהפרוטוקול יותר מורכב ופרטיו רבים, ישנו סיכוי גדול יותר לפגיעות – ייתכן בשלב העיצוב וייתכן בשלב המימוש.

ARP (Address resolution protocol) הוא פרוטוקול תקשורת המשמש ברשת מחשבים לאיתור כתובת ה־ MAC (הכתובת הפיזית) של תחנה ברשת על פי כתובת ה־ IP שלה.

הפרוטוקול מופעל בעת שליבת מערכת הפעלה נקראת לייצר מסגרת (frame) להעברה בין חוליות ברשת המחוברים דרך קו פיזי ואין בזיכרון המטמון שלה (ARP cache) רשומה לכתובת ה IP המבוקשת. משפט לא ברור. לנסח מחדש או לוותר.

פגיעות ה Heartbleed שתוארה בפרק סקירת הפגיעויות היא תוצאה של שגיאה במימוש הפרוטוקול.

ARP poisoning

הרעלת ARP מתבצעת כאשר תוקף מאזין לתקשורת וממתין לשליחת בקשת ARP מהמחשב אותו הוא מעוניין לתקוף, וברגע שהוא מזהה בקשה כזו הוא ישלח מענה המצמיד את כתובת ה IP המוכרת למחשב הנתקף המופיעה בבקשה, לכתובת ה MAC של מחשב התוקף או שרת מטעמו.
כעת התקשורת שתישלח עבור כתובת ה IP תגיע באמצעות כתובת ה MAC למחשב מתחזה.
תקיפה זו נובעת מחולשה בפרוטוקול ARP המאפשרת את "גניבת הזהות" המתוארת.

פגיעויות פורטים גלויים

פורטים פתוחים עשויים להוות דרך כניסה למערכת. הרבה מהתקיפות מתחילות בסריקת המערכת בחיפוש אחר פורטים פתוחים.
התשובה מפורט מסוים יכולה לתת מידע על סוג המערכת, מערכת ההפעלה או רכיבי פגיעות נוספים שיכולים לשמש עבור תקיפה עתידית.

תקיפת DNS

שרתי DNS מתחזקים רשימת דומיינים ותרגומים לכתובת IP.
תקיפת DNS תתרחש באמצעות הכנסת מידע שגוי בזיכרון המידע של שרת ה DNS, שינוי מידע זה יגרום לשרת ה DNS להחזיר כתובת IP שגויה עבור דומיין מסויים ובכך לגרום למידע לעבור למחשב אחר ולשימוש התוקף או להפנות את המשתמש לאתר אחר.
שאילתת DNS בדרך כלל מתנהלת בפורט 53, מה שגורם להשארת פורט זה פתוח.

פגיעויות תעבורת נתונים

רכיבים שונים במערכת מושפעים מכמת הפניות המגיעות, ניהול לא נכון של התעבורה יכול לגרום לשיתוק המערכת או לחילופין אי יכולת לספק את המענה שהיא נדרשת.
תקיפות מסוג זה נקראות התקפת מניעת שירות.

התקפת מניעת שירות (DoS and DDoS)

תקיפות מסוג זה הן בעצם ניצול של מתן שירות מוגבל ע"י המערכת המותקפת, אשר תנוצל ע"י שליחה של בקשות רבות – לעיתים אף ממספר מחשבים המסונכרנים ביניהם – שיגרמו למערכת לא להצליח לתת מענה לבקשות אמיתיות ולפעמים אפילו לקרוס.

הסנפה (Sniffing)

יכולת לקרוא תעבורת נתונים ברשת ולהגיע למידע רגיש.
תעבורת נתונים שאינה מוצפנת או מוצפנת לא כראוי תהווה קורבן קל עבור תוקף שמעוניין בתוכן שמועבר או שמעוניין בשינוי התוכן.

<u>התקפת "אדם בתווך" (MITM)</u>
לעתים מסומנת בקיצור MITM היא סוג של ציתות אקטיבי שבו התוקף המאזין לשיחה המתקיימת בין שני מחשבים או ישויות ברשתו מצליח להתחזות לכל אחד מהם בנפרד, להעביר את התמסורת ביניהם ולגרום להם להאמין שהם מתקשרים ביניהם ישירות בערוץ פרטי, בעוד שלמעשה ההתקשרות נשלטת לגמרי בידיו.

פגיעויות דפדפן
עבור מערכות web אשר גישה אליהן היא דרך דפדפן, לעיתים המערכת מושפעת מפעולות אשר מתרחשות ע"י הדפדפן כגון הרצת קוד Javascript או Flash.

<u>XSS</u>

תקיפות מסוג זה יכולות גם להיות כחלק מפגיעויות נתוני קלט המופיעים בהמשך, אולם מכיוון שייחודם נובע מריצה דווקא על ידי הדפדפן המהווה שער למערכות web, החלטתי לכלול אותן כאן.

XSS היא התקפה נגד גולש אינטרנט המנוצלת באמצעות פגיעות ביישומי אינטרנט ומאפשרת לתוקף להזריק סקריפט זדוני שמטרתו לרוץ בדפדפנים של משתמשי מערכת אחרים.

<u>ישנם 3 סוגים עיקריים</u>:

1. Reflected XSS

כאשר דף אינטרנט משתמש בקלט מהמשתמש בייצור עמוד HTML דינמי, ולא מוודא שאין קוד זדוני בקלט זה, האתר עלול להיות פגיע לפרצת XSS זו.
תוקף יוכל לשכנע משתמש תמים לגשת לכתובת של אתר מוכר לו, כאשר הכתובת תזריק קוד זדוני לעמוד התוצאות, ובכך תיתן לתוקף שליטה מלאה על התוכן של עמוד זה.

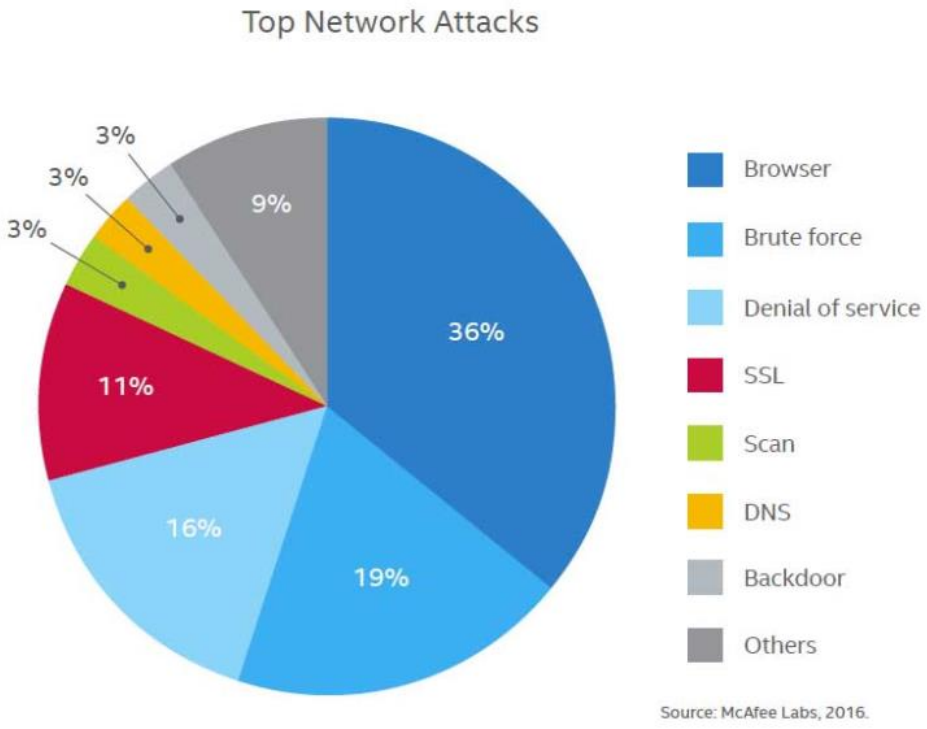
2. Stored XSS

פרצה זו לרוב תימצא באתר המאחסן באופן תמידי קלט ממשתמש מסוים, בלי לוודא שהקלט לא מכיל קוד זדוני, ולאחר מכן מציג תוכן זה למשתמשים אחרים.

3. DOM XSS

קיימת לרוב בסקריפט צד לקוח אשר משתמש בקלט כדי לייצר עמוד HTML, בלי לוודא כי קלט זה אינו מכיל קוד זדוני. לדוגמה, קוד JavaScript אשר מקבל כתובת אינטרנט כקלט, ומשתמש בה לייצור עמוד HTML, בלי לוודא שהכתובת עצמה אינה מכילה קוד, יכיל פרצת XSS מקומית.

באיור מספר 6 אנו יכולים לראות את התפלגות תקיפות הרשת בשנת 2016.



איור 6 – התפלגות תקיפות הרשת בשנת 2016

3.2.2 קוד[8]

פגיעויות נתוני קלט[9]

מערכות מחשבים בנויות להתנהג בצורה מסוימת ולהפיק תוצר מסוים על פי נתוני הקלט שנכנסים למערכת, נהיה מעוניינים להשיג שליטה מירבית על נתוני הקלט במערכת שלנו.

נתונים חשובים: גודל, אורך, סוג, טווח, קידוד ולאן הנתון אמור להגיע במערכת.

[8] בפרק "הוכחת הרעיון" יוצגו מימושים של פגיעויות המופיעות בפרק זה

[9] נושא זה הינו נושא כללי, אשר מתפצל לתתי נושאים אשר חלקם מתוארים תחת סוג פגיעות ספציפית

מערכת מאובטחת תדאג לסמן ולהיות מודעת לנתוני הקלט התקינים ("קלט טוב") ולא רק לנתוני הקלט האסורים.

אי בדיקה ראויה של נתוני הקלט יכולה להוות וקטור תקיפה עבור אזורים שונים במערכת.

פגיעויות מסד נתונים

בעבודה עם מסד נתונים נרצה להגן על אפשרויות שליפת המידע, השחתת מסד הנתונים או הכנסת מידע שגוי. נצטרך לאכוף בקרת גישה על מנת למנוע גישה מאנשים שאינם מאושרים לכך ונצטרך לשמור על שלמות הנתונים.

הזרקת קוד SQL (SQL injection)

הזרקת קוד SQL הינה שיטה לניצול פרצת אבטחה בתוכנית מחשב בעזרת פניה אל מסד הנתונים. השם נובע מכך שהמשתמש מכניס קוד SQL לשדה קלט אליו אמורים היו להיכנס נתונים תמימים. באופן זה יכול משתמש זדוני לחרוג לחלוטין מן התבנית המקורית של השאילתא, ולגרום לה לבצע פעולה שונה מזו שיועדה לה במקור.

גלישת חוצץ[10]

פגיעויות באי שמירת משאבים יקרי ערך

ראשית, אנחנו צריכים לזהות את המשאבים היקרים לנו ביותר - משאב יכול להיות נתונים אישיים, סיסמאות, מפתחות, קבצי מערכת וכדומה.
פתרון ההצפנה הוא אפשרות טובה, אך חשוב להקפיד על שימוש ברכיבים קריפטוגרפים נכונים ואמינים ושימוש נכון.

פגיעויות באי טיפול בשגיאות

אפשרויות תקיפה רבות מתגלות על ידי תגובות בלתי שגרתיות או לא מצופות של המערכת.
תוקף ינסה לגרום למערכת להיכנס למצבים לא חוקיים ולמסור הודעות שגיאה המכילות מידע רגיש.

פגיעויות בחשיפת נתוני פלט

בניגוד לפיסקה הקודמת, אנו עוסקים בנתוני פלט חוקיים של המערכת.
אולם, נתוני הפלט של המערכת יכולים לספק הרבה מידע, לעיתים הרבה יותר מידע ממה שנדרש, מידע כזה יכול לספק חומר עבור התוקף הן מבחינת תיאור המערכת והן מבחינת המידע אותו הוא מבקש להשיג.

[10] תוארה כחלק מפגיעות Ghost

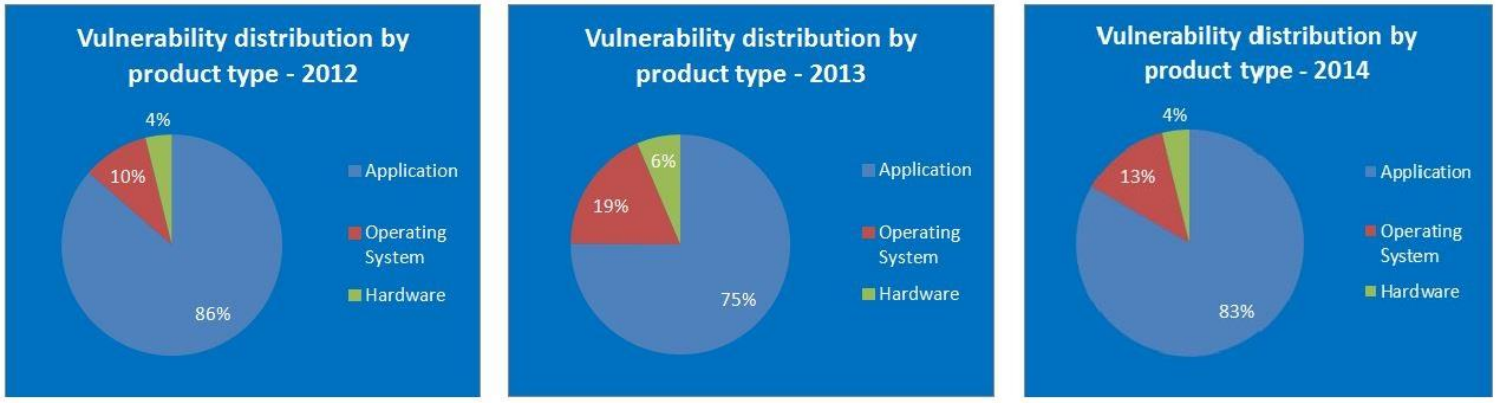
תגובות המערכת צריכות להיות מבוקרות ומנוסחות כראוי.

בכתיבת מידע לקבצי Log או לקבצי ארכיון צריך לקחת בחשבון האם נהיה פגיעים במקרה שמישהו החפץ לתקוף את המערכת ולהשיג מידע יוכל להשיג את הקבצים האלו.

פגיעויות ספריות צד שלישי (Third-party libraries)

נושא שהוא רוחבי ושייך לכל השכבות המוזכרות הוא פגיעויות ספריות צד שלישי.
ספריות מסוג זה נמצאות בשימוש נרחב בכל מערכת מחשבים הן יכולות להימצא במערכת ההפעלה, בפרוטוקולי התקשורת ובמימוש פונקציונאליות ביישום שלנו.
מכיוון שאין מנוס משימוש בספריות אלו, נצטרך לעצב פתרון המאפשר להשתמש בספריות אלו תוך כדי בקרה על מצב האבטחה שלהן.

לסיכום, באיור מספר 7 אנו רואים את חלוקת הפגיעויות לפי מוצרים – חומרה, מערכות הפעלה או יישום בשנים 2012 -2014.
ניתן לראות בבירור שהאחוז הגבוה של הפגיעויות (80% ~) הוא ברמת היישום, שזהו בעצם האזור היותר נגיש לנו בתור מפתחי מערכת והאחריות על אזור זה הינה בלעדית של צוות הפיתוח.[11]



איור 7 – התפלגות פגיעויות על פי סוג מוצר

[11] אין הכוונה לאנשי הפיתוח בלבד, אלא לגורמים המעורבים בזמן פיתוח המוצר

4. תהליך פיתוח מאובטח כמענה עיקרי להגנה

בהרבה מן המקורות העוסקים בפיתוח מאובטח, עיקר הדיון סובב סביב פעולות מסוימות או best practice שעל המפתח לעשות על מנת שהקוד שלו יהיה מאובטח כראוי.
אכן, זהו חלק גדול בתחום זה שנקרא פיתוח מאובטח, אך כתוצאה מאפיוני הפגיעויות בפרק הקודם, ארצה בפרק זה להרחיב מעט את היריעה לסקור את עולם בדיקות אבטחת התוכנה ולתת מקום גם לשלבים נוספים בחיי בניית המערכת כגון – ניהול צוות, בקרת האיכות (QA) ואינטגרציה ו"לצוות" אותם למאמץ המשותף של יצירת SDL (Secure Development Life cycle)[12] פעיל ומערכת חסינה מפני תקיפות.

4.1 מתודולוגיות בדיקות אבטחת תוכנה

המאמר "A Survey on Software Security Testing Techniques" [19] עוסק בשיטות הנפוצות לבדיקות אבטחת תוכנה.
ראשית, לפני שנסקור את השיטות השונות, נחדד את המושג המובא בכותרת – "בדיקות אבטחת תוכנה".
בדיקות איכות תוכנה (QA) כוללות את מהימנות, גמישות וחוסנה של התוכנה, במספר רמות שונות – בדיקות יחידה, בדיקות רגרסיה ובדיקות אינטגרציה.
בדיקות אבטחת התוכנה ובדיקות איכות התוכנה הן מאותה משפחה – פעולתה של התוכנה בשונה מהדרך שהיא תוכננה, יכולה לגרום לפגיעות של המערכת, ניראה לומר שבדיקות אבטחת תוכנה משלימות את הכיסוי בהיבט אבטחת המערכת שנעשה ע"י בדיקות האיכות של התוכנה תוך התחשבות באפשרויות תקיפה רחבות וע"י גורם עוין.
בהמשך הפרק אציג את הימצאותם של בדיקות אבטחת התוכנה בשלבים שונים של תכנון ובניית המערכת.

1. ביקורת הקוד (Code review)

ביקורת הקוד אינה קשורה בהכרח לאבטחת תוכנה, זוהי מתודולוגיה הדורשת מעבר על הקוד ע"י מפתח שלא כתב אותו, אך שיטה זו מקבלת יתר תוקף במשמעות של בדיקות אבטחת תוכנה.
לעיתים בדיקת קוד מקור מתבצעת בעקבות תוצאות ניתוח הסטטי שניראה בהמשך, בעצם זהו תהליך של בדיקה ידנית של קוד המקור עבור פגיעות אבטחה.

הרבה חולשות אבטחה רציניות לא ניתן לזהות עם כל הליך אחר של ניתוח או בדיקה, לדברי הקהילה האבטחה אין חלופה טובה עבור מעבר אנושי על הקוד לאיתור פגיעויות עדינות.
בעיות אבטחה רבות לא מכוונות, אך משמעותיות, הן גם קשות מאוד לגילוי בצורות אחרות של ניתוח או בדיקה, כגון בדיקות חדירה, ביצוע ניתוח קוד המקור הטכניקה של בחירה לבדיקות טכניות.
עם קוד המקור, בודק יכול לקבוע במדויק את מה שקורה ולהסיר את הספקולציות שנבעו מבדיקות

[12] יוסבר בהרחבה בהמשך

"קופסה שחורה". הנושאים כוללים בעיות מקבילות, פצצות זמן, פצצות לוגיקה, לוגיקה עסקית פגומה, בעיות בקרת גישה וחולשות קריפטוגרפיות, כמו גם דלתות אחוריות, סוסים טרויאניים וצורות אחרות של קוד זדוני, עלולים להיחשף על ידי ביקורות קוד המקור.
ניתוח קוד המקור יכול גם להיות יעיל ביותר כדי למצוא בעיות יישום כגון קטעים של הקוד שבו אימות קלט לא בוצע כראוי.
ביקורות קוד שימושיות גם עבור גילוי נוכחותו של קוד זדוני. לדוגמה, אם הקוד כתוב ב- C, ייתכן שהבודק ימצא הפניות המתאימות לחלקי קוד המבצעים פעולות חריגות ו / או חלקים מקודדים מורכבים וקשים למעקב או המכילים קוד אסמבלר מוטבע.

<u>יתרונות</u> - שלמות, יעילות ודיוק.
<u>חסרונות</u> - לא מעשי עבור בסיסי קוד גדול, דורש בודקים מיומנים, עבודה אינטנסיבית, בלתי אפשרי לזהות שגיאות זמן ריצה.

2. בדיקת קוד סטטית אוטומטית (Static code analysis) [20]

ניתוח סטטי אוטומטי הוא כל ניתוח שבוחן את התוכנה מבלי לבצע אותה, בדיקה זו כוללת את השימוש בכלי ניתוח סטטי.
ברוב המקרים, בדיקה זו אומרה לנתח את קוד המקור של התוכנית, אם כי ישנם מספר כלים לניתוח סטטי של קובץ הרצה בינארי.
ניתוח סטטי אינו דורש גרסה משולבת במלואה או מותקנת של התוכנה, לכן ניתוח זה יכול להתבצע באופן חוזר על עצמו לאורך כל יישום של התוכנה. ניתוח סטטי אוטומטי אינו דורש כל מקרה בדיקה ואינו יודע מה הקוד נועד לעשות.
המטרה העיקרית של ניתוח סטטי היא לברר פגמי אבטחה ולזהות את התיקונים הפוטנציאליים שלהם.
הפלט של כלי הניתוח הסטטי אמור לספק מידע מפורט מספיק על נקודות הכשל האפשריות של התוכנה כדי לאפשר למפתח שלה לסווג ולעדכן את נקודות התורפה של התוכנה על בסיס רמת הסיכון שהן מציבות למערכת.
בדיקות ניתוח סטטי צריך להתבצע מוקדם ככל האפשר במחזור החיים ככל האפשר.

<u>יתרונות</u> – סריקה אוטומטית, רמת מומחיות נדרשת הינה בינונית, דו"ח מפורט על ידי הכלי.
<u>חסרונות</u> – False positive הנובע מבדיקה שאינה קשורה לתחום הפעילות של המערכת.

כלים לדוגמא: Coverity, Checkmarx, Veracode.

3. בדיקת טשטוש/ערפול (Fuzzing testing)

בדיקת טשטוש הינה בעצם בדיקת המערכת באמצעות הזרקת נתונים אקראיים ל API החוקי של המערכת או דרך התממשקות של המערכת עם מערכת אחרת.
הרעיון הוא לחפש התנהגות מעניינת הנובעת מהזרקת "הרעש" העשויה להצביע על נוכחות של פגיעות או

תקלות תוכנה.
בדרך כלל בדיקת הטשטוש תהיה ייעודית לפרוטוקול מסוים ולערכי קלט מסוימים אולם טווח הערכים נדרש להיות גדול.
בדיקות טשטוש יכולות להיות מוגדרות כחיפוש מחט בערימת שחת מכיוון שאנו מחפשים בעיית תוכנה מסוימת, מבלי שיהיה לנו מושג מספיק רחב על מה אנחנו מחפשים בדיוק, אולם מכיוון שאופן ביצוען של בדיקות טשטוש הוא אוטומטי הן יכולות להתרחש בשלבים רבים של מחזור החיים של התוכנה.

יתרונות – בדיקה יסודית עבור תוצאות לערכי הכניסה למערכת.
חסרונות – חיפוש שאינו מוגדר.

כלים לדוגמא: Fuddly, Radamsa, Peach, Honggfuzz, Codenomicon Defensics.

4. ניהול סיכונים (Risk analysis)

כדי לבחון את דרישות האבטחה ולזהות סיכונים ביטחוניים, ניתוח וניהול סיכונים מתבצע בשני שלבים עיקריים בחיי המערכת - במהלך שלב התכנון של הפיתוח וכחלק מהחלטת שחרור המערכת.
תהליך זה עוזר למעצבי המערכת לנתח ולחשוב על איומי האבטחה שהמערכת שלהם עשויה להתמודד איתם.
בבדיקה זו קשורה לשלב "אפיון איומים אפשריים" (Threat modeling) שיתואר בהמשך.
לעיתים יהיו איומים מסוימים שנחליט לקבלם כחלק מהמערכת.

יתרונות – תהליך יסודי.
חסרונות – פוטנציאל לתהליך ארוך. דורש מיומנות והיכרות עמוקה של המערכת.

5. סריקת פגיעויות (Vulnerability scanning)

שימוש בסורקי פגיעויות הוא טכניקה מאוד חשובה בבדיקות אבטחת תוכנה.
השימוש בכלים אלו מאפשר לסרוק את התנהגות התוכנה עבור קלטים ופלטים ועבור תבניות מוכרות הקשורות לפגיעויות ברמת היישום, דפוסי פגיעויות אלה, "חתימות" הניתנות להשוואה לחתימות המחפשות על-ידי סורקי וירוסים, או ל"מבני קידוד מסוכנים" שנבדקו על-ידי סורק קוד מקור אוטומטי, מה שהופך את סורק הפגיעות, למעשה, לכלי אוטומטי של התאמת תבניות.
למרות שכלים אלו יכולים למצוא דפוסים פשוטים המשויכים לפגיעויות, סורקי הפגיעות האוטומטית אינם מסוגלים לאתר במדויק את רמת הסיכון של מערכת בצירוף פגיעויות אלו.
רוב סורקי הפגיעות מנסים לספק מנגנון לצבירה של דפוסי פגיעות ובמקרים רבים, מספקים גם מידע והדרכה כיצד לצמצם את הפגיעויות שהם מזהים.

יתרונות – מתבצע באמצעות כלי אוטומטי.
חסרונות – רוב הכלים אינם מסוגלים לתאר את רמת הסיכון של המערכת.

כלים לדוגמא: Nessus

6. בדיקות חדירה (Penetration testing)

השם החלופי של בדיקות חדירה הוא "האקריות אתית" (Ethical Haker).

זוהי טכניקה נפוצה מאוד לבדיקת אבטחת הרשת. אולם, בעוד בדיקות חדירה הוכיחה להיות יעילה בבדיקת אבטחת הרשת, הטכניקה לא מתרגמת באופן טבעי ליישומים, ביחס ליישום, בדיקות חדירה יבדקו את התנהגות היישום בסביבה שבה הוא אמור "לחיות".
בדיקות החדירה מציינות אם המערכת מתנגדת להתקפות בהצלחה, וכיצד היא מתנהגת כאשר היא אינה מסוגלת לעמוד בפני התקפה. בודקי החדירה מנסים גם לנצל פגיעויות שאותרו בביקורת קודמת.
סוגי בדיקות חדירה כוללים "קופסה שחורה", "קופסה לבנה" ו"קופסה אפורה".
בבדיקת חדירה של קופסאות שחורות, הבודקים אינם מקבלים מידע על היישום.
בדיקות קופסה לבנה הן ההפך של בדיקות קופסה שחורה, מידע מלא על היישום עשוי להינתן לבודקים.
מבחני חדירה בגוון אפור, הנפוץ ביותר, הוא המקום שבו לבוחן ניתנות אותן הרשאות כמו משתמש רגיל כדי לדמות זיהוי פנים זדוני.
בדיקת החדירה צריכה להתמקד באותם היבטים של התנהגות המערכת, האינטראקציה והפגיעות שלא ניתן לזהות באמצעות בדיקות אחרות שבוצעו מחוץ לסביבה התפעולית החיה. בודקי החדירה צריכים להכפיף את המערכת למתקפות מתוחכמות שנועדו ליצור סדרה מורכבת של התנהגויות על פני רכיבי המערכת, כולל רכיבים שאינם רציפים. אלו הם סוגי התנהגויות שלא ניתן לכפות על ידי כל טכניקת בדיקה אחרת.

בדיקות חדירה משמשים כדי למצוא בעיות אבטחה העלולות להיווצר בארכיטקטורה ובתכנון של התוכנה, שכן זהו סוג של פגיעות הנוטה להתעלם מטכניקות בדיקה אחרות.

יתרונות – מדמה התנהגות של תוקף אמיתי.
חסרונות – דורש מיומנות גבוהה ואנשי מקצוע ייעודיים.

4.2 תהליך פיתוח מאובטח

מהו SSDL/SDLC?

על מנת להצליח ליישם את יסודות הפיתוח המאובטח והבדיקות הנדרשות במהלך חיי הפיתוח של מערכת, אנו נדרשים לתהליך מוגדר. תהליך ה SSDL (Secure Software Development Lifecycle) הוגדר לראשונה על ידי חברת Microsoft והוטמע בחברה כמדיניות מחייבת בשנת 2004, במרוצת השנים חברות רבות הלכו בעקבותיה והטמיעו תהליך זה או דומה לזה בפיתוח המוצרים שלהן.
בהמשך אשתמש בחלקים משלבי תהליך זה על מנת לשים דגש על הפעולות או השיטות הנדרשות לביצוע על מנת להימנע מפגיעויות שונות המשמשות לביצוע תקיפות.

המאמר "A Survey on Secure Software Development Lifecycles"[21] עוסק בהשוואה בין שיטות שונות לתהליך פיתוח מאובטח, תוך הבדלה בין סוגי תהליכים המיושמים מתחילת פיתוח המערכת לבין סוגי תהליכים המיושמים על מנת לשפר את חוסנה של מערכת קיימת.

המאמר מציין מספר נקודות חשובות לאזכור:

- תהליכי הפיתוח המסורתיים אינם כוללים במידה מספקת התייחסות ספציפית לבעיות אבטחת המערכת.
- מודעות חברות הפיתוח בנושא האבטחה הולכת ועולה, גם במקרה שזה לא נדרש ע"י הלקוחות מתוך הידיעה שבמקרה של פריצה, הלקוחות יפנו לחיפוש פתרון אחר.
- בטווח הרחוק שימוש ב SSDL יוצא משתלם מבחינת עלות המערכת.
- חברת Microsoft פרסמה את השפעת מימוש SSDL על מוצרי החברה – הפחתה של כ 66% במספר הבאגים הקריטיים.

התהליכים המוצגים במאמר:

	Name	Year	Highlights
SSDL	CLASP	2006	24 formalized security best practices that cover the entire SDL.
	Microsoft SDL	2004	Complete integrated SDL heavily based on risk analysis and based on a set of activities for each phase. Integrated in the Microsoft development suite. Suited for Operating System developers and large software houses.
	Touchpoints	2004	7 best practices that can be applied to the existing SDL.
Security Initiative	OpenSAMM	2009	Measurable maturity model with 3 maturity levels and 12 security practices. Can be used as a benchmark.
	BSIMM	2009	12 actual best practices used by the industry with 3 maturity levels. Can be used as a benchmark.
	SAFECode	2008	Overview of industry best practices with an emphasis on leadership.
	Securosis	2009	SSDL targeted specifically for web applications and focusing on automation by using tools.

איור 8 - טבלת השוואה בין תהליכי פיתוח מאובטח

בתיאור תהליך ה SSDL אתבסס על תהליך המוצע והממומש ע"י חברת Cisco הנקרא CSDL (Cisco Secure Development Lifecycle)[22] תוך שילוב של סוגי הבדיקות שתוארו בפרק הקודם.

1. **שלב ראשון: הגדרת המערכת**

בשלב זה נגדיר את המערכת מהיבט האבטחה שלה.

דרישות האבטחה הבסיסית של המוצר מגדירות את הפונקציונליות של האבטחה, את תהליך הפיתוח ואת צורת השימוש המצופים מהמערכת. שלב זה מתמקד בהיבטי אבטחה חשובים כמו ניהול אישורים וניהול מפתח, תקני הצפנה וכדומה.
בשלב זה נאפיין גם את הדרישות המינימליות של עמידות וחוסן, סילוק נתונים רגישים, רישום ותיעוד של שירותים ושימוש פרוטוקולים.

שלב זה דורש היכרות רבה עם התחום של המערכת ועם הפונקציונאליות שהיא נדרשת לספק, כחלק משלב זה ייתכנו שינויים בהגדרת מערכת בעקבות קשיי אבטחה שיתעוררו.

נכון ששלב זה יושם על ידי ה PO (product owner)[13] של המוצר או מי שאחראי על משימות השייכות לתפקיד זה.

2. איפיון האיומים האפשריים

איפיון האיומים האפשריים מבוצע על ידי מעקב אחר זרימת הנתונים באמצעות המערכת, וכן זיהוי גבולות אמון שבו הנתונים עלולים להיות בסכנה ובשלב זה אנו נבחן את כל אפשרויות הגישה והכנסת הנתונים לתוך המערכת ע״י שימוש בתרשימי העיצוב של המערכת.
למעשה, שלב זה מאפשר למעצב לפתח אסטרטגיות הפחתת פגיעות פוטנציאלית ומסייע להם למקד את המשאבים המוגבלים שלהם ותשומת הלב שלהם על חלקי המערכת בסיכון הגבוה ביותר. מומלץ כי לכל היישומים יהיה מודל איום שפותח ותועד.

Threat modeling
Threat modeling הינה גישה העוזרת לזהות את האיומים השונים על המערכת.
גישה זו כוללת: פירוק היישום - להבין, באמצעות תהליך של בדיקה ידנית, איך היישום עובד, הנכסים שלו, הפונקציונליות, ואת הקישוריות. הגדרת וסיווג הנכסים – סיווג הנכסים לנכסים מוחשיים ובלתי מוחשיים ולדרג אותם בהתאם לחשיבות העסקית.
חקירת פגיעויות אפשריות - בין אם טכניים, תפעוליים או ניהוליים. חקירת איומים פוטנציאליים - לפתח ראייה מציאותית של כיווני תקיפה פוטנציאליים מנקודת מבט של התוקף, באמצעות תרחישי איום. יצירת אסטרטגיות להפחתת הסיכון - לפתח בקרות מקלות על כל אחד מהאיומים הנחשבים מציאותיים.

חשיבות שלב זה היא גדולה מכיוון שהיא מספקת הסתכלות במבט כללי על המערכת ויכולת לזהות את נקודות החולשה מבעוד מועד ומספק תוצר שאמור להוות פנס וסמן עבור מפתחי המערכת למקומות הרגישים יותר.
חלק זה בתהליך צריך להיות מבוצע על ידי ארכיטקט המערכת או לחילופין על ידי מפתח בכיר בעל

[13] תיאור תפקיד ה PO יוסבר בהמשך

היכרות טכנולוגית עם תחום ואזור הפעילות של המערכת.
בשלב זה יבוצע בפעם הראשונה ניהול סיכונים המתואר בפרק הקודם.

3. בדיקת תוכנות צד שלישי שבשימוש המערכת

מוצרים רבים משלבים תוכנות צד שלישי, הן מסחריות והן קוד פתוח. כתוצאה מכך, מוצרים (ולקוחות) מושפעים כאשר פגיעויות נמצאות בתוכנה מסוג זה.
אם נדרשנו להשתמש בתוכנת צד שלישי אין זה אומר שאנחנו פטורים מלוודא את מצבה האבטחתי של תוכנה זו.
הדבר הנכון בשלב זה הוא יצירת מאגר - המתמלא באופן דינמי ככל שהמוצר מתקדם - של כל תוכנת צד השלישי שהמערכת שלנו מכילה ואת מספרי הגרסאות ולהתעדכן על פרסומי פגיעויות עבור תוכנות צד השלישי שברשימה.

שלב זה יבוצע על ידי נציג קבוצת הפיתוח המכיר את קוד המערכת ואת הספריות הנמצאות בו, ככל שתהליך זה יבוצע בצורה אוטומטית יותר, התערבותו של המפתח תהיה פחותה.

4. אבטחת קוד סטטי

שלב זה מתחיל מרגע תחילת כתיבת הקוד.
מפתחי המערכת יהיו אחראים על ביצוע ההמלצות עבור כתיבת קוד מאובטח[14] ותוך כדי הפיתוח יבצעו את פעולות ביקורת הקוד (Code review) ו בדיקת קוד סטטית אוטומטית (Static code analysist) המתוארות בתחילת הפרק.

אתן מספר דוגמאות לקוד פגיע וכיצד ניתן לתקן קוד זה.[15] ראה איור 9.

סוג החולשה	שפת הדוגמא	קוד רגיש	הסבר	קוד מאובטח
הגבלת זמן החיים של מידע רגיש	Java	Consle cl = System.console(); String user = cl.readLine("Enter your ID:"); String pass = cl.readLine("Enter your pass:");	מכיוון ש String בשפת Java הינו אובייקט, תוכנו ינוקה רק על ידי תזמון ה GC. ייתכן שהסיסמא תשהה בזיכרון זמן רב יותר מן הנדרש.	Consle cl = System.console(); String user = cl.readLine("Enter your ID:"); char[] pass = cl.readPassword("Enter your pass:");
שימוש בפונקציות	C/C++	#include <stdio.h> int main () { char username[8];	ישנן מספר פונקציות אשר לא מוודאות את האורך של המידע שהן מעתיקות	char* username, *nlptr; int allow = 0;

[14] מצורף כנספח Secure coding practices checklist (OWSAP)
[15] בפרק "הוכחת הרעיון" יוצגו דוגמאות נוספות ושיטה כללית לפיתוח מאובטח

קבלת / העתקת מידע שגויות		int allow = 0;
		printf external link("Enter your username, please: ");
		gets(username); // user inputs "malicious"
		if (grantAccess(username)) {
		allow = 1;
		}
		if (allow != 0) { // has been overwritten by the overflow of the username.
		privilegedAction();
		}
		return 0;
		}
			ביחס לאזור אליו הוא מועתק.
			
			במקום gets צריך להשתמש בפונקציית fgets המקבלת אורך buffer לנתון הקלט.
			ובמקום פונקציות strcpy,strcat,strcmp להשתמש ב strncpy, strncat,strncmp המצבעות בדיקת אורך.
				username = malloc(LENGTH * sizeof(*username));
				
				if (!username)
				return EXIT_FAILURE;
				printf external link("Enter your username, please: ");
				
				fgets(username,LENGTH, stdin);

העלאת קבצים שרירותית	Java	public String execute() {
		try {
		File fileToCreate = new File("filepath", "filename");
		FileUtils.copyFile(uploadedFile, fileToCreate);
		return "SUCCESS";
		} catch (Throwable e) {
		addActionError(e.getMessage());
		return "ERROR"; }
			הקובץ uploadedFile הועלה ע"י המשתמש, ומבלי לבצע שום בדיקה הוא מועתק למיקום filePath.
			בדיקה בסיסית להעלאת קבצים היא בדיקת סוג הקובץ והאם הוא מתאים למה ציפינו לקבל והאם סיומת הקובץ מתאימה לתוכנה. (mime type)
				public String execute() { try { File fileToCreate = new File("filepath", "filename"); boolean PlainText = checkMetaData(uploadedFile, "text/plain"); boolean image = checkMetaData(uploadedFile, "image/JPEG"); boolean HtmlTxt = checkMetaData(uploadedFile, "text/html"); if (!textPlain || !img || !textHtml) { return "ERROR";

הקצאת זיכרון מסוכנת	C++	int *piBuffer = NULL;
		int n = 10;
		piBuffer = malloc(n * sizeof(int));
			הקצאת זיכרון הינה פעולה רגישה.
			ישנה עדיפות להקצות ולשחרר זיכרון באמצעות new & delete
			ולא באמצעות malloc & free.
			
			הבדליים עיקריים בין השיטות:
			
			1. החזרת מצביע בעל טיפוס לעומת מצביע void.
			2. הגודל הנדרש מחושב ע"י ה compiler ולא ע"י המשתמש.
			3. בשגיאה יזרוק exception ולא יחזיר null
				int * piBuffer;
				piBuffer = new int [5];

איור 9 - טבלת תיאור שגיאות אבטחה בקוד

הדבר הנכון בשלב זה הוא להוציא לפועל את הפעולות הנדרשות על חלקי קוד קטנים על מנת להפחית מאחוז הזמן הנדרש למשימות אלו.
רוב המערכות בנויות ממספר רכיבים (Components), בשלב זה אנחנו עדיין לא מתעסקים עם צורת העבודה של המערכת והתנהגותה באופן דינמי, לכן יחידת הבדיקה שלנו הינה רכיב בודד.

שלב זה יבוצע על ידי צוות הפיתוח – תוך חלוקת המשימות בין חברי הקבוצה.

החשיבות הגדולה של שלב זה היא שבמובן מסוים זהו האזור הקל לזהות שגיאות שנוצרו על ידינו ושבמקרה של שגיאה כזו, הזמינות לתקן אותה היא גבוהה מכיוון שהיא תתוקן על ידי מפתח מהצוות.

5. **בדיקת פגיעויות**

בשלב זה אנו כבר מסתכלים על המערכת כמערכת שלמה וחיה, המקבלת פרמטרים ומגיבה בהתאם.
את שלב בדיקת הפגיעויות ניתן ורצוי לחלק ל 2 חלקים:

a. **אבטחת איכות (QA) ואינטגרציה**

הבדיקות שיבוצעו על ידי אנשי אבטחת האיכות והאינטגרציה כתוספת לבדיקות הפונקציונאליות שייעשו, תהיינה הרצת הכלים האוטומטים המבצעים בדיקת ערפול (Fuzzing testing) ו סריקת פגיעויות (Vulnerability scanning) המתוארות בתחילת הפרק.
בשלב זה הבדיקות עדיין תהיינה בסביבת מעבדה.

b. **האקינג**

בדיקות החדירה (Penetration testing) יבוצעו על ידי צוות המומחה בתחום התקיפה, רמת הבידול של קבוצה זו תוחלט ע"י חברי ההנהלה.
השאיפה עבור בדיקות אלו היא שהמערכת תוקם בצורה שהיא אמורה לפעול בשטח ועם התצורה הרלוונטית.

באופן אופטימלי - שלבי התהליך יחזרו על עצמם עבור כל פיתוח במערכת.
ניהול סיכונים (Risk analysis) יבוצע בשיתוף חברי ההנהלה עבור כל שחרור גירסא של המערכת.
למעשה פגיעויות ניתנות לגילוי בכל אחד מהשלבים של תהליך הפיתוח המאובטח, אולם סוגים מסוימים של פגיעויות בדרך כלל יתגלו בשלבים מתאימים.
כשנבחן את הפגיעויות המתוארות בפרק 2 נוכל לשים לב שתחת תהליך פיתוח מאובטח הן היו מתגלות בשלב תהליך הפיתוח.

מספר דוגמאות:
פגיעות ה Heartbleed הינה פגיעות במימוש הפרוטוקול אשר הייתה מתגלה באמצעות בדיקות ערפול.

פגיעות ההצפנה קצה אל קצה של אפליקציית WhatsApp הייתה מתגלה בשלב איפיון האיומים האפשריים בזמן תכנון הפיתוח.

4.2 פיתוח מאובטח במתודולוגיית Agile - ASDP (Agile Secure Development Process)

תהליך ה SDLC הינו ככל תהליך ונרמז בשמו - הוא דורש משאבים רבים.
עובדה זו עלולה לגרום לכך שארגונים רבים יימנעו מליישם תהליך זה או במקרה היותר גרוע יישמו אותו בצורה שגויה.

אמנה מספר טענות עיקריות כנגד התהליך.

1. נדרשת מעורבות חוזרת ונשנית ולעיתים לא מתוכננת של גורמים בכירים.
2. תהליך מאסיבי מקביל לתהליך פיתוח המערכת.
3. בשונה מפעילותה הפונקציונאלית של המערכת זהו תהליך שתוצאותיו קשות לכימות.
4. דורש מיומנות.

אולם אם נצליח לשלב תהליך זה כחלק מתהליך הפיתוח עצמו, כאשר הבקרה על התהליך תהיה חלק אינטגרלי מהבקרה על התקדמות הפיתוח של המערכת ייתכן שנצליח לרכז את המאמצים ולמזער את ההשפעות של הטענות המוזכרות.

בתחילת דרכה של תעשיית ההייטק, הועתקו היסודות של תהליך ניהול ופיתוח המוצר מהתעשיות שהיו מוכרות באותה תקופה.
התהליך שהיה נהוג לממש בתעשיות אלו הוא תהליך ה Waterfall (מפל מים), אלמנטים עיקריים בתהליך זה הם שדרישות המוצר היו ניתנות מושלמות בשלב מקדים לתהליך הייצור, התוצר או תוצר זמני היו מוכנים בפרקי זמן גדולים (מספר חודשים ולעיתים מספר שנים), יכולת השינוי בתוך התהליך הייתה מזערית עד בכלל לא.
בשנות ה 90 החלו לזהות ששיטה זו אינה מתאימה לעולם מוצרי התוכנה והוצעו מספר שינויים ושיטות חדשות לתהליך פיתוח מוצרי התוכנה.
בתחילת שנות ה 2000, התכנסו מספר אנשי תוכנה על מנת לחשוב על האלמנטים החשובים בתהליך פיתוח התוכנה ולהציע מתודולוגיה חדשה שתענה על הצרכים של עולם התוכנה, וכך נולדה מתודולוגיית ה Agile הרווחת בעולם הפיתוח כיום.

למתודולוגיה זו יש מספר מימושים שונים, אולם הדגשים החשובים בה הם:

1. איטרציות (סבבים) קצרות, כאשר בכל איטרציה ישנו מוצר שאפשר להגדיר את רמת המוכנות שלו, שחרור גירסא ללקוחות יוחלט על פי הסכמה.
2. התקשרויות בין אנשי הצוות בפגישות פנים אל פנים.
   a. פגישת קדם תכנון איטרציה – פגישה בין PO ל Scrum master על תכולת ה backlog.

b. פגישת תכנון איטרציה – הצוות מחליט אילו משימות ייכנסו לאיטרציה הקרובה.
c. פגישה יומית – פגישה זו צריכה להיות קצרה ככל האפשר כאשר כל אחד מחברי הצוות מעדכן על סטטוס ההתקדמות שלו והאם יש משהו שמעכב אותו.
d. סיכום איטרציה – הדגמת הפיתוחים שנעשו באיטרציה ומסקנות על האיטרציה.

3. אינטגרציה מתמשכת בין חלקי המערכת.
4. משימות קטנות מוגדרות לכל סבב הנלקחות מ backlog צרכים של המוצר.

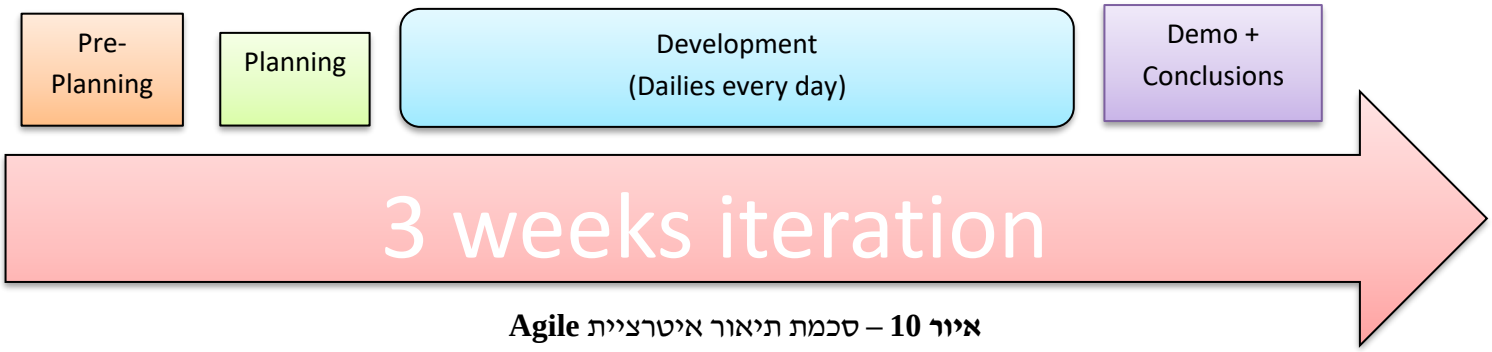
שחקנים ראשיים בעולם ה Agile[16]:

1. PO (Product owner)

ה PO אחראי למקסם את הערך של המוצר ואת עבודתו של צוות הפיתוח.
ה PO הוא האחראי הבלעדי על ניהול ה Product backlog , שזה כולל לבטא בבהירות את פריטי ה-backlog וסידור הפריטים ב backlog על מנת להשיג את המטרות ולמלא את המשימות בצורה הטובה ביותר.

2. Scrum Master

תפקידו של ה Scrum master הינו לדאוג שצוות מתפקד ב "אג'יליות" – תוך ניהול הפגישות היומיות ופתרון אתגרים ובעיות שצצות לחברי הצוות במהלך האיטרציה.
תפקיד נוסף של ה Scrum master הוא להוות צינור בין ה PO לבין חברי הצוות בהבנת המשימות הנמצאות ב backlog. ראה איור 10.



איור 10 – סכמת תיאור איטרציית **Agile**

בהסתמך על מתודולוגיית ה Agile, אני רוצה להציע את התהליך ASDP (Agile Secure Development Process) אשר ישולב כחלק מתהליך הפיתוח במתודולוגיית ה Agile.

[16] בתיאור התפקידים העיקריים השתמשתי בשיטת Scrum שהיא אחת השיטות למימוש מתודולוגיית Agile

אבני הבניין של תהליך ה ASDP

- ניהול התכולה – אבטחת מוצר דורשות המון פעולות ושלבים שונים בדיסציפלינות שונות.
  אנחנו צריכים להיות מודעים לעובדה שמוצר מסוים לא יהיה 100% מאובטח, אבל הוא יהיה מבחינתנו בשלב שחרור כלשהו כמאובטח מספיק (Secure enough).
  משימות מסוימות – הבהולות יותר – ייכנסו לטיפול באיטרציה הקרובה ואחרות יידחו.

- אנשים – משימות האבטחה יחולקו בין אנשי צוות בתפקידים שונים:
  - ארכיטקט/מפתח בכיר – אחראי על יצירת ה Threat model עבור ה feature שהוא אפיין.
  - מפתח – מימוש הנחיות קוד מאובטח והרצת בדיקת קוד סטטי וחקירת תוצאותיו.
  - מפתח/בודק – בדיקה מצבן של גרסאות ספריות צד שלישי.
  - בודק – בדיקת פרוטוקולים, בדיקות API.

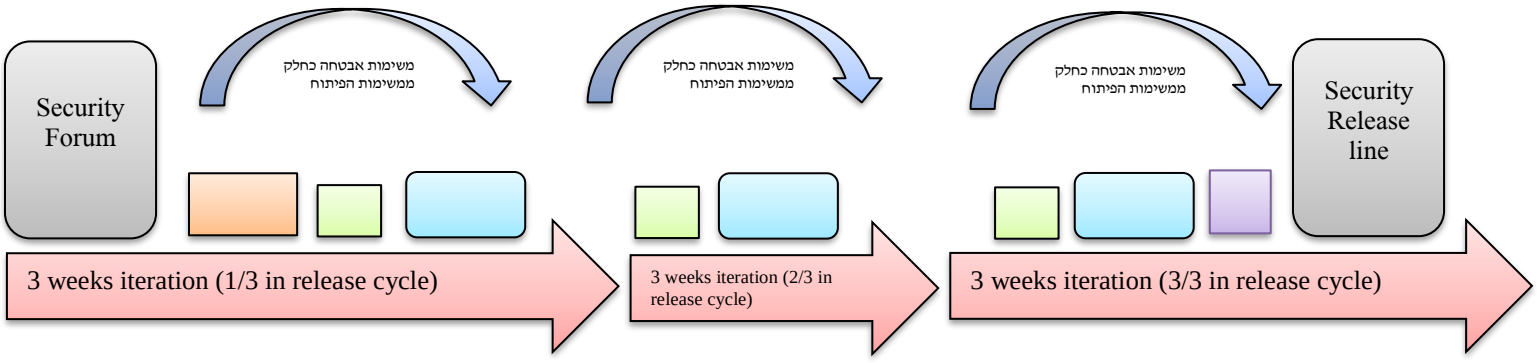
  הכשרות יינתנו בצורה ייעודית עבור האנשים עבורם זה נדרש.

- ניהול סיכונים – מעורבות מוגדרת מראש כחלק מהאיטרציה של מקבלי ההחלטות על מנת לאשר את מצב האבטחה של גירסא נוכחית.

- פורום אבטחת תוכנה – בדרך כלל יהיו מספר צוותי Agile,לכן יידרש פורום הבנוי מנציג מכל צוות (Security advocate) שיהיה אחראי על קבלת משימות האבטחה ושילובן ב backlog משימות הפיתוח.
  אחת למספר איטרציות, יתכנס הפורום על מנת לאמוד את מצב האבטחה של המוצר ועל אילו נושאים צריך להתמקד באיטרציות הקרובות.

- Security Release line – באיטרציה האחרונה לפני שחרור גירסא רשמית יבוצעו Penetration testing עבור המערכת בצורת הקמתה בשטח.

על פי שיטה זו אנו יוצרים "שיגרת אבטחה" (Security routine) כחלק מתהליך הפיתוח במתודולוגיית ה Agile כמתואר באיור 11.



איור 11 - תהליך ASDP

5. לא רק בעולם התוכנה – פגיעויות ברכיבי חומרה

עד כה העבודה התרכזה בעולם התוכנה - הפגיעויות שתוארו ותהליך הפיתוח המאובטח עסק בטעויות שבסופו של דבר הינן שגיאות קוד תוכנתי בשכבה כזו או אחרת וכן רובם של התיקונים או הימנעויות משגיאות אלו היו קשורים לפתרון ברמת התוכנה.

בתחילת שנת 2018 נתגלו מספר פגיעויות חומרה אשר יכולות להשפיע בשנים הקרובות על בטיחותן של מערכות שונות, לפני שאסקור בקצרה נציגה (Spectre)[17] של תקיפות אלו, אני מעוניין לציין מספר הבדלים מהותיים רלוונטיים בין רכיבי חומרה לרכיבי תוכנה.

זמינות

בזמן שאת התוכנה המותקנת במערכת בדרך כלל ניתן לעדכן בצורה פשוטה יחסית, לעיתים אפילו מרחוק, עדכון החומרה ידרוש מאמץ גדול יותר וזמן רב יותר – הן בצד של הכנת הרכיבים והן בצד של החלפתם במערכת.

עלות

עלות החלפת רכיבי חומרה יכולה להיות גבוהה במיוחד המבוססת על עלות ראשונית של החומרים המרכיבים את הרכיבים.

אנו רואים אפוא, שפגיעות ברכיב חומרה בעלת פוטנציאל לנזק גדול יותר מפגיעות תוכנה.

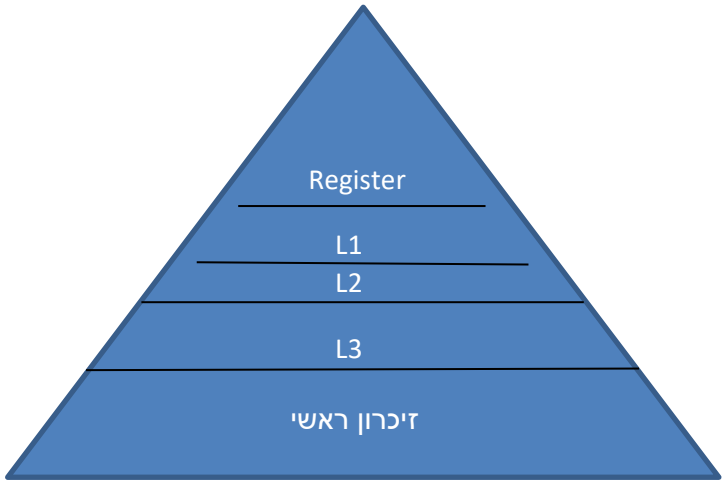
Spectre

את ההסבר על Spectre אתחיל בביאור מושגים שחושבים להבנת הפגיעות ודרכם נבין שלב אחר שלב את פעולת התקיפה.

[17] על בסיס המחקר האקדמי שפרסם אותה [23]

זיכרון מחשב

זיכרון מחשב מנוהל בהיררכיה, כאשר ככל ששכבת זיכרון קרובה יותר לראש הפירמידה היא אזור זיכרון קטן יותר והגישה אליו היא מהירה יותר.



איור 8 - היררכית זיכרון

עם השנים ככל ששופרו המעבדים וכח העיבוד נהיה מהיר יותר, הגישה לזיכרון היוותה את צוואר הבקבוק בתפקוד המערכת, על מנת להתמודד עם האיטיות הזו קיים במעבדים המודרניים זכרון מטמון (cache) אליו המעבד טוען נתונים במהלך התוכנית לזיכרון המטמון על מנת שיהיו זמינים לו בהמשך ותיחסך הפנייה לאזורם המקורי בזיכרון.

Speculative execution[18]

על מנת לשפר את ביצועי המערכת, המעבד מבצע מספר אופטימיזציות, אחת מהן נקראת Speculative execution – ניחוש הפעולה הבאה אותה יידרש לבצע וחישובה מראש.
זאת אומרת, כאשר המעבד מזהה שנעשית עכשיו פעולת קריאה/כתיבה (I/O) שתדרוש זמן ארוך יחסית, הוא ינצל את הזמן על מנת לחשב את הפעולות האפשריות הבאות שהוא יידרש לבצע.

[18] Branch predication - אתמקד בהסבר זה על החלק הרלוונטי למימוש התקיפה שאציג

דוגמא:

```
data = [1,2,3,4,5,6]
input =1000

if (input < data.size)
{
   data[input]
}
```

בהינתן הקוד הנ"ל, תחת תנאים מסוימים, המעבד יתנהג בצורה הבאה – בזמן שתתבצע גישה לזיכרון על מנת להבין מה הוא האורך של מערך data, הוא ייחשב את data[input] למקרה שהתנאי input < data.size יחזיר "אמת" הוא כבר יהיה מוכן עם הנתון data[input].

בהיבט אבטחת המערכת, ישנה פגיעות מסוימת מכיוון שהמעבד מרשה לעצמו לגלוש בזיכרון לאזור שאינו מורשה – או לפחות ישנה אפשרות שהוא יהיה כזה.
פגיעות זו לא גרמה לדאגה גדולה, מכיוון שלאחר שהמעבד הבין שהתנאי לא נכון והקוד התלוי בו לא יתבצע – הוא "זורק" את החישוב וחוזר למצב תקין.

Side-channel attack

התקפת ערוץ צדדי הינה התקפה המנצלת מידע שהושג מאופן היישום הפיזי או צורת ניצולים המשאבים של המערכת מבלי לפעול כנגד האלגוריתם או מימוש הקוד של המערכת.

תקיפת ערוץ צדדי הרלוונטית לנו היא מידע תזמון – על פי זמן החישוב אנו יכולים לקבל מידע על הערך הנסתר מאיתנו.

דוגמא:

נניח שישנה סיסמא לגישה לאתר:

```
Password = "strongPa$$word";
```

כמו כן נניח שבדיקת הסיסמה שסיפק המשתמש נעשית ע"י השוואה של האות הראשונה בשתי הסיסמאות, ואם היא זהה, עוברים לאות הבאה וכן הלאה.

כעת, בתור תוקף, אני מתחיל לנסות את האפשרויות לנחש את הסיסמא, כאשר אני מקליד את האות a – אני מקבל שגיאה שהסיסמא אינה נכונה תוך 1 מאית השניה וכן עבור b,c וכו'.
אולם כשאגיע לאות s, אגלה שאני מקבל הודעת שגיאה תוך 2 מאיות השנייה - מכיוון שההשוואה עבור

האות הראשונה עברה בהצלחה.
מעתה אני יודע שהאות הראשונה בסיסמא היא s ואפשר להמשיך כך הלאה.

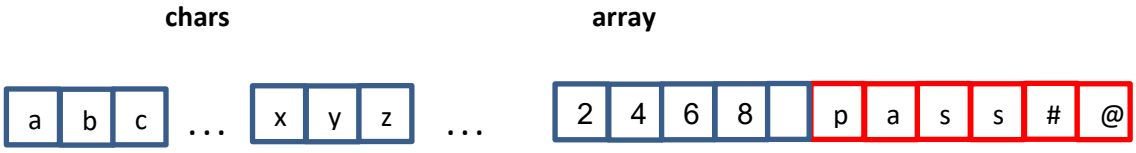
תקיפת Spectre משתמשת באלמנטים המוזכרים מעלה בכך שראשית התוקף כותב תוכנית שתורץ במצב משתמש כחלק מהמערכת ובונה מערך שישב בזיכרון ויהיה בלתי ניתן לשמירה בזיכרון מטמון בריצת התוכנית.[19]

```
chars = [a,b,c,d,e,f,g,h,I,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]
```

ומערך נוסף:

```
array = [2,4,6,8]
```

תיאור הזיכרון הראשי:



איור 9 - תיאור זיכרון ראשי בתקיפת Spectre

כאשר האזור האדום הוא אזור שאינו נגיש לתוכנה ונשמרים שם נתונים רגישים.
התוקף ייצור תנאי ארוך שתוצאתו היא false , אשר מצריך את המעבד לבצע Speculative execution ולחשב את אפשרויות לביצוע בהתאם לתנאי.

```
If (. . .)
{
  return chars[array[5]];
}
```

בזמן שהמעבד מבצע את ה speculative execution הוא לא מודע לכך שהאזור אותו הוא קורה הוא מוחץ לתחום - זהו תפקידו של ה kernel בזמן ריצת התוכנית – ויטען לזיכרון המטמון את הערך p, כשתוצאת התנאי תהיה false, המעבד ימחק את הערך מה state הנוכחי של ריצת התוכנה, אולם הערך יישאר בזיכרון המטמון.

כל מה שנותר לתוקף הוא לנסות לפנות לכל אחד מתאי המערך chars ולתזמן את התשובה.

[19] בשפת Java לדוגמא זה נעשה באמצעות המילה השמורה volatile

ובאמצעות תקיפת ערוץ צדדי מבוססת זמן - התא היחיד שהחזיר את התשובה המהירה ביותר הוא מכיל את התו מהאזור הזיכרון הבלתי נגיש.

Spectre על צורותיו השונות הוכח בפועל על קוד native ואף על JavaScript המורץ על ידי הדפדפנים.

תקיפה נוספת המשתמשת באלמנט זהה של פגיעות היא Meltdown, כאשר ההבדלים העיקריים הם ש Meltdown עובדת על מעבדי אינטל ואפל לעומת Spectre שעובדת על רוב המעבדים הקיימים בשוק וש Meltdown מאפשרת גישה לזיכרון ה Kernel לעומת Spectre המאפשרת קריאה של אזורים אחרים בתוכנית רצות של משתמשים שונים. [24] [25]

האם תהליך פיתוח חומרה מאובטח היה יכול למנוע מפגיעויות מסוג זה להיות מופצות?
קשה לומר, אך בוודאי שישנו מקום לתהליך מוגדר עובר פיתוח וייצור רכיבי חומרה אשר לכל הפחות היה מעלה תהיות על צורת פעולתם של הרכיבים השונים.

6. הוכחת הרעיון – POC

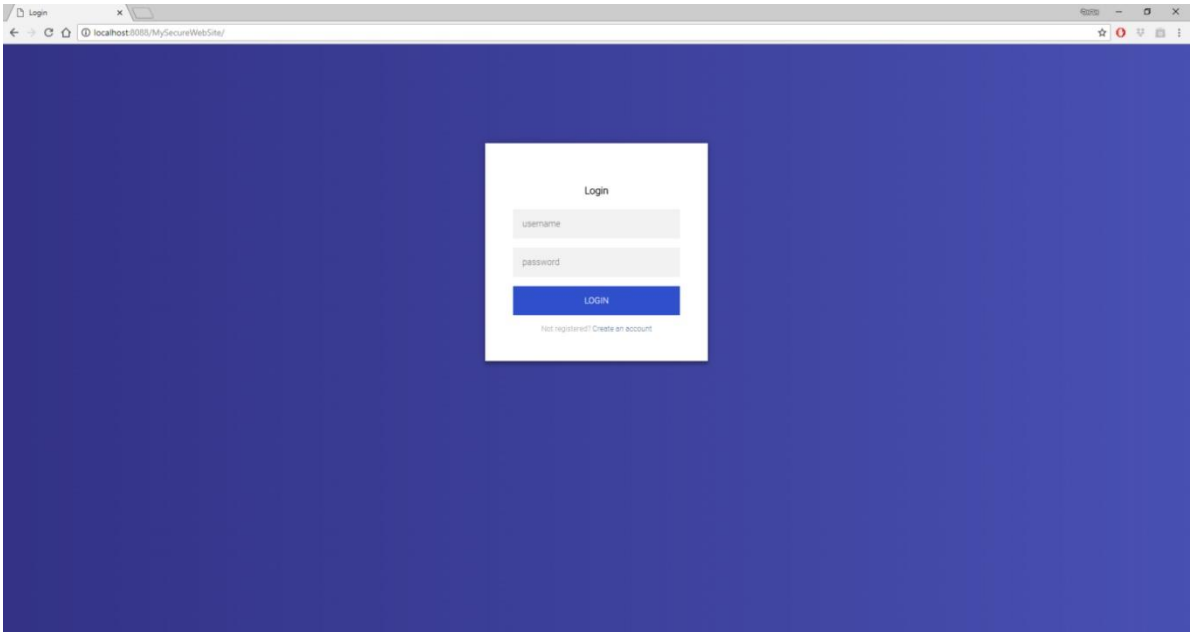
כחלק מהוכחת הרעיון של חשיבות תהליך פיתוח מאובטח בהגנה מפני תקיפות סייבר, פיתחתי מערכת המכילה פגיעויות באזור שכבת הקוד ומימשתי תקיפות המנצלות חולשות אלו ומשפיעות לרעה על תפקוד המערכת ועל המשתמשים בה. המערכת פותחה כחלק מפרויקט הגמר.
כחלק מן הפרויקט פותחה ספריית code-security-layer על מנת לספק קונספציה למימוש קוד מאובטח ולתת מענה עבור תקיפות המנצלת חולשות הקשורות לקוד המערכת.
הקונספציה שמומשה בספריה בעצם מציעה יצירת "קיר" (wall) עבור כל סוג של תקיפה או תחום חשיפה שיבצע את הפעולה הספציפית השייכת לקיר זה באזור הרלוונטי בקוד המערכת.
ייחודיות נוספת של ספריה זו הינה בעצם יכולת הפרדת מומחיות פיתוח קוד המערכת לבין מומחיות, ידע והבנה בנושא אבטחת הקוד, בכך שאת חוקי ונתוני הספרייה יוכל לפתח Security engineer או כל גורם רלוונטי אחר.

תיאור המערכת[20]

המערכת הינה אתר אינטרנט חברתי המאפשר למשתמשים רשומים להעלות תמונות פרי יצירתם ולקבל תגובות וחוות דעת משאר משתמשי האתר.

<u>מסך 1</u>

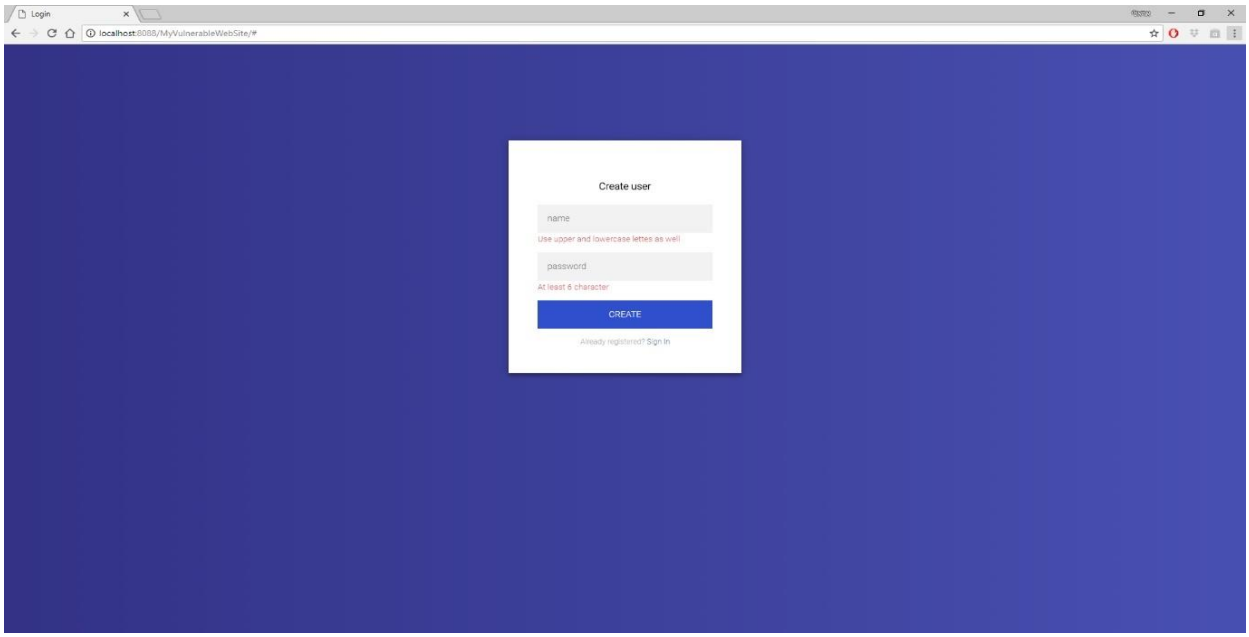
מסך הכניסה הינו מסך Login המאפשר למשתמשים רשומים להזין שם משתמש וסיסמא ולהיכנס לאתר. משתמש שאינו רשום, יכול לעבור ממסך זה למסך יצירת משתמש חדש.



[20] פרטים מלאים על מימוש המערכת ניתן למצוא בדו"ח הפרויקט

מסך 2

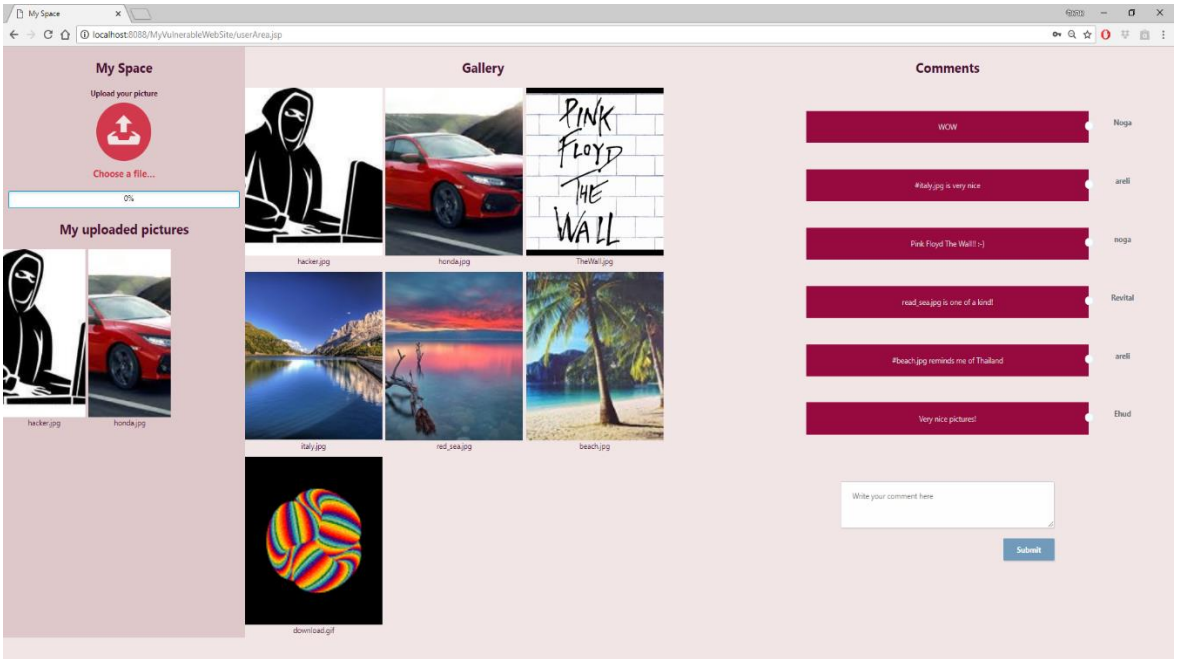
מסך יצירת משתמש חדש, משתמש חדש נדרש להזין שם משתמש וסיסמא.



מסך 3

מסך זה מחולק לשני חלקים:

1. האזור האישי של המשתמש בו ניתנת למשתמש האפשרות להעלות תמונות ובאזור זה הוא יכול לראות את התמונות שהוא העלה בעבר.
2. גלריית התמונות המכילה את כלל התמונות שהועלו ע"י משתמשים, כאשר שם התמונה מופיעה בצמוד לתמונה.



סביבת המערכת
המערכת מתקשרת עם מסד נתונים המכיל את הטבלאות הבאות:

- Users – טבלה המכילה את המשתמשים הרשומים.
  שדות הטבלה – ID, UserName, Password
- GalleryPictures – תמונה המכילה את התמונות שהועלו לאתר.
  שדות הטבלה – img_id, img_title, img_data, user
- Comments – טבלה המכילה את התגובות שהועלו ע"י המשתמשים.
  שדות הטבלה – ID,UserName,Comment

משתמשי המערכת
המערכת היא מערכת web המספקת אתר אינטרנט המאפשר גישה והרשמה לכל מי שמעוניין.
צפייה בתמונות, העלאת תמונה והוספת תגובה תתאפשר אך ורק למשתמש רשום.

פגיעויות וחולשות
המערכת פותחה בצורה הנאיבית ומכילה פגיעויות וחולשות הקשורות לאלמנטים בפיתוח קוד מאובטח,
אתמקד באלו הרלוונטיות לפגיעויות אשר נוצלו על מנת לבצע את התקיפות ופגיעה במערכת.

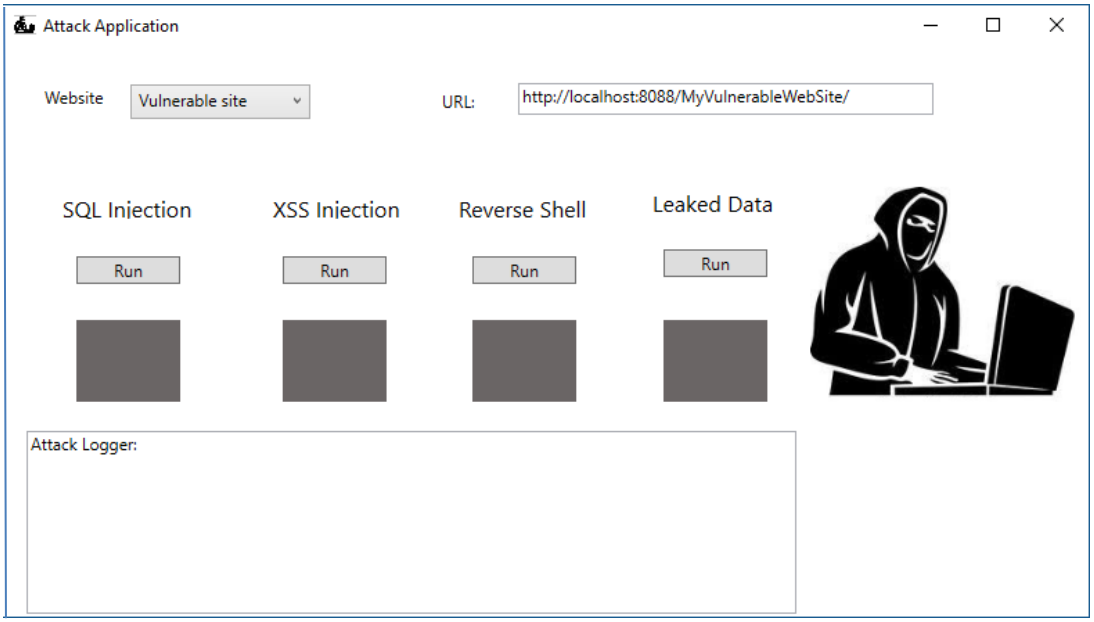
בדיקת הקלט (Input validation)
אי בדיקת תקינות נתוני הקלט, אפשרו לתוקף לבצע SQL injection ו XSS attack (יוסבר בהמשך) ולפגוע בתפעול התקין של האתר.

שימוש בתוכנות צד שלישי (Third party security)
שימוש בתוכנות צד שלישי זהו דבר נפוץ ההולך ומתפתח בעולם הפיתוח.
אולם לצד השימוש בתוכנות מסוג זה, נדרשת בקרה מתמדת על התפתחות הגרסאות של תוכנות אלו ופירצות האבטחה המתגלות בהן.

דליפת נתונים (Data leakage)
חשיפת נתונים המכילים כתובות URL ומידע רגיש של המשתמשים.

תיאור אפליקציית התקיפה

כחלק מהפרויקט פותחה אפליקציית תקיפה המבצעת את ניסיונות התקיפה של התוקף על האתר הפגיע ועל האתר המאובטח ומעדכנת בצורה אינטראקטיבית על תוצאות התקיפות.



איור 11 - אפליקציית התקיפה

תהליך התקיפה

טבלת התקיפות בעמוד הבא מתארת בהסבר כללי את סוג או תחום התקיפה, תוך פירוט נקודתי בהקשר של הפגיעות וניסיון התקיפה בהתאמה למערכת שלנו, הסבר התהליך שמבצע התוקף וכיצד שני סוגי האתרים (פגיע ומאובטח) מגיבים על תקיפה זו.

סוג התקיפה	תיאור התקיפה	אתר פגיע	אתר מאובטח

SQL Injection	**רקע**: SQL injection הינה שיטה לניצול פרצת אבטחה בתוכנית מחשב בעזרת פניה אל מסד הנתונים. השם נובע מכך שהמשתמש מכניס קוד SQL לשדה קלט אליו אמורים היו להיכנס נתונים תמימים. באופן זה יכול משתמש זדוני לחרוג לחלוטין מן התבנית המקורית של השאילתא, ולגרום לה לבצע פעולה שונה מזו שיועדה לה במקור. הזרקת SQL היא מקרה פרטי של קבוצה רחבה של פרצות אבטחה הנקראות הזרקות קוד, שמתרחשות כאשר תוכנה כלשהי יוצרת קוד בזמן ריצה על־פי הקלט ובלי לבדוק את תוכן הקלט תחילה.
	
	במקרה שלנו: בתקיפה זו התוקף ינסה לעבור את מסך הכניסה מבלי ליצור משתמש חדש, אלא להשתמש בנתונים של משתמש קיים.
	התקיפה תיעשה באמצעות מניפולציות על שאילתת ה SQL שנשלחת על מנת לבצע בדיקת אימות (authentication) של שם המשתמש והסיסמא שהוזנו.
	התוקף יינצל את העובדה שהאתר בונה את שאילתת ה SQL ע"י שרשור נתוני הקלט שהוזנו ע"י המשתמש ואת העובדה שתווים מסוימים יכולים לגרום לשאילתא להתנהג בצורה שונה.
	
	פעולת התוקף:
	הזנת המחרוזות
	`Renana ' or '1'='1`
	`bla bla' or '1'='1`
	
	בשדות שם משתמש וסיסמא בהתאמה.
		התוקף מצליח להחדיר לשאילתת ה SQL תנאי שתמיד יחזיר true – 1=1, ומצליח להיכנס לאתר עם נתוני המשתמש הראשון שנמצא במסד הנתונים.	**SqlWall** שהוטמע ע"י המפתח לפני שרשור נתוני הקלט למחרוזת השאילתא ושליחתה למסד הנתונים, ביצע בדיקת תקינות מוכוונת שאילתת SQL וזיהה שישנם תווים מחשידים, בעל פוטנציאל השפעה על שאילתת ה SQL וסירב להמשיך את הטיפול בשליחת השאילתא עם נתוני הקלט הנ"ל.

XSS	**רקע**: XSS היא התקפה נגד גולש אינטרנט המנוצלת באמצעות פגיעות ביישומי אינטרנט ומאפשרת לתוקף להזריק סקריפט זדוני שמטרתו לרוץ בדפדפנים של משתמשי מערכת אחרים.
	
	ישנם 3 סוגים עיקריים:
	
	<u>Reflected XSS</u>
	
	כאשר דף אינטרנט משתמש בקלט מהמשתמש בייצור עמוד HTML דינמי ,ולא מוודא שאין קוד זדוני בקלט זה, האתר עלול להיות פגיע לפרצת XSS זו.
	
	האקר יוכל לשכנע משתמש תמים לגשת לכתובת של אתר מוכר לו, כאשר הכתובת תזריק קוד זדוני לעמוד התוצאות, ובכך תתן להאקר שליטה מלאה על התוכן של עמוד זה.
	
	<u>Stored XSS</u>
	
	פרצה זו לרוב תימצא באתר המאחסן באופן תמידי קלט ממשתמש מסוים, בלי לוודא שהקלט לא מכיל קוד זדוני, ולאחר מכן מציג תוכן זה למשתמשים אחרים.
	
	<u>DOM XSS</u>
	
	קיימת לרוב בסקריפט צד לקוח אשר משתמש בקלט כדי לייצר עמוד HTML, בלי לוודא כי קלט זה אינו מכיל קוד זדוני. לדוגמה, קוד JavaScript אשר מקבל כתובת אינטרנט כקלט, ומשתמש בה לייצור עמוד HTML, בלי לוודא שהכתובת עצמה אינה מכילה קוד, יכיל פרצת XSS מקומית.
	
	במקרה שלנו: בתקיפה זו התוקף ינסה לנצל פגיעות ל Stored XSS הקיימת במנגנון התגובות של האתר. התוקף יזין כתגובה – אפשר כמשתמש לגיטימי או כמשתמש מזויף – קוד JS אשר ירוץ בדפדפן של כל משתמשי האתר ברגע שיכנסו לאתר.
	
	פעולת התוקף:
	יצירת Java script payload:
	
	`"<img src=" +'/'.ToString() + "`
	`onerror=alert(\"You_Have_Been_Hacked!`
	`_This_is_Just_a_pop_up_window_but_it_could_have_b`
	`een_worse..\") style=display:none; />";`
	
	והוספת ה payload כתגובה באתר.
		ה payload הנ"ל ישמר כתגובה ב DB ויקפיץ הודעה ברגע שמשתמש ייכנס לגלריית התמונות. קוד ה JS יכול היה באותה מידה לבצע אסיפת נתונים מהמחשב ושליחתם לשרת מרוחק או לחילופין לגרום נזק.	**XssWall** שהוטמע ע"י המפתח לפני אחסון התגובה במסד הנתונים, ביצע בדיקת תקינות מוכוונת XSSוזיהה שישנם תווים מחשידים, הקשורים לקוד Java script ויכולים להיטען ולהיות מורצים ע"י הדפדפן וסירב לשמור את התגובה הנ"ל.

Third party vulnerability & Reverse Shell	**רקע:** Reverse shell זוהי היכולת לתוקף להגיע למחשב ולהצליח לקבל את האפשרות להריץ פקודות command line על המחשב הנתקף. **במקרה שלנו:** התוקף יחפש במערכת ספריית צד שלישי המכילה פגיעות כלשהי וינסה לנצל פגיעות זו על מנת להשתלט על השרת המריץ את האתר ולאסוף ממנו מידע נחוץ. **פעולת התוקף:** התוקף ינסה לקבל הודעת שגיאה מהאתר אשר תמסור מידע על שרת ה web שמריץ אותו, לאחר מכן התוקף יחפש פרצה הרלוונטית לגירסה המותקנת וינסה לממש את התקיפה המתאימה.	הפלת האתר הצליחה ונתקבלה הודעת שגיאה המכילה את מספר גירסה שרת ה apache tomcat. גירסה 7.0 הינה גירסה ישנה ופגיעה (CVE-2017-1566) ואפשר לבצע עליה תקיפה המאפשרת קבלת אפשרות להרצת פקודות command line. הפגיעות באה לידי ביטוי שניתנת אפשרות לכל אחד לאחסן באמצעות בקשת PUT דף Jsp אשר ירוץ על השרת. התוקף מכין דף JSP המכיל קוד המאפשר להריץ פקודות בשורת הפקודה ולהציג במסך את התוצאות של הפקודות.(מצורף כנספח בדוח הפרויקט) את הדף הנ"ל התוקף יאחסן בכתובת http://<host:port>/MyVulnerableWebSite/hack.jsp/ פניה לכתובת זו תאפשר לתוקף להריץ פקודות command line על השרת.	כשהתוקף מבצע את הפלת האתר, הוא רואה שגירסת ה apache tomcat הינה גירסה מעודכנת שאינה פגיעה. ההגנה מפני תקיפה מסוג זה נעשתה מבעוד מועד, הן בשלב הבנייה הראשוני של המערכת והן מבחינת ניטור ובקרת ספריות צד שלישי. **ThirdPartyVersionsWall** מספק לנו שתי יכולות: 1. קבלת סטטוס על כל ספריות צד שלישי שאנחנו משתמשים בהן והאם ישנן גירסאות חדשות. (מצורף כנספח בדוח הפרויקט) 2. במקרה שמזוהה ספריית צד שלישי שמוכרת לנו כפגיעה (ע"י הוספת ספריות שנתגלתה בהן פגיעות למאגר) הספרייה תתריע או לחילופין לא תאפשר למערכת לעלות.
Data leakage	**רקע**: לעיתים תוקף יכול להשיג מידע יקר ערך, בעקבות חיפוש בדברים שנכתבו בצורה מודעת על ידי מפתחי המערכת, אך לא ייחסו חשיבות לנתונים או שחשבו שהנתונים מוגנים. סיטואציה כזו מוגדרת כדליפת מידע. **במקרה שלנו**: בעקבות התקיפה הקודמת, התוקף ינצל את הנגישות למחשב השרת על מנת לבצע חיפוש על קבצים שייתכן שמכילים מידע רגיש. **פעולת התוקף**: התוקף יקליד את פקודות ה command line הבאה בחיפוש אחר log המערכת: `where /r C:\logs *.log` פקודה זו מחפשת אחר קובץ שהסיומת שלו היא log. , החיפוש יכול להתבצע בווריאציות שונות ובאזורים שונים.	התוקף אכן מצא את קובץ לוג המערכת ומצא בתוכו את שורות הלוג המעניינות: 22:43:26 2018-07-17DEBUG DbConnection:28 - login attempt to **MySQL** server with user: **sql11224641** and **password NXVhaFRSa1pMWQ==** 22:43:27 2018-07-17DEBUG LoginServcie:50 - user authentication pass successfully: areli שמות משתמשים נכתבו ללוג בצורה גלויה, סוג שרת ה SQL והפרוטוקול כולל שם משתמש וסיסמא לפניה לשרת. הסיסמא "הוצפנה" שלא כראוי ובאמצעות encoding ל Base64 אשר ניתן ל Decoding: Base64: **NXVhaFRSa1pMWQ==** Password: **5uahTRkZLY** להשלמת התמונה וניצול הזלגת מידע נוספת, התוקף הסניף את התעבורה בהתחברות לאתר (מצורף כנספח בדוח הפרוייקט) ומצא את כתובת ומיקום שרת ה SQL.	במקרה שתוקף אכן יגיע ללוג המערכת, בעזרת שימוש ב **DataEncryptionWall** ובעקבות הנחיות אבטחת המידע הנתונים הרגישים הוצפנו באמצעות AES ובכך ניתנים לשחזור במקרה הצורך רק על ידי גורם מורשה. שורות קוד לדוגמא: 11:03:46 2018-07-05 DEBUG RequestProcessor:46 - Sql input validation 11:03:46 2018-07-05 DEBUG LoginServcie:31 - Checking authentication for: **kMtiFvcXbkbBbbtJ+XHNdQ==** 11:03:46 2018-07-05 DEBUG DbConnection:30 - login attempt to MySQL server with user: **T3BItGwsmjHrURHOoW/kUQ== and password Tpbq10tZ0aZ/13Z+3hnKsQ==** 11:03:47 2018-07-05 DEBUG LoginServcie:51 - user authentication pass successfully: **kMtiFvcXbkbBbbtJ+XHNdQ==** בנוסף **DataEncryptionWall** מספק עוד שכבת הגנה ומאפשר להצפין את כל קובץ הלוג. מפתח המערכת השתמש ביכולות זו עבור לוגים ישנים.

7. סיכום

״מַה-שֶּׁהָיָה, הוּא שֶׁיִּהְיֶה, וּמַה-שֶּׁנַּעֲשָׂה, הוּא שֶׁיֵּעָשֶׂה; וְאֵין כָּל-חָדָשׁ, תַּחַת הַשָּׁמֶשׁ. יֵשׁ דָּבָר שֶׁיֹּאמַר רְאֵה-זֶה, חָדָשׁ הוּא: כְּבָר הָיָה לְעֹלָמִים, אֲשֶׁר הָיָה מִלְּפָנֵנוּ.״

(קהלת פרק א׳ ט-י)

בעולם המחשבים רווחת הדעה שבמאבק בין התוקפים לבין אלו האחראים על הגנת המערכת תמיד ידם של הקבוצה הראשונה תהיה על העליונה – כל מערכת היא פגיעה ולעולם תמצא הדרך לנצל פגיעות זו.
אולם, ככל שמסתכלים לאפיונם של תקיפות ופגיעויות אנו רואים שישנה התנקזות וחזרה לניצול אלמנטים מוכרים ופגיעויות ישנות. [26]
בעבודה זו נוצר חוט שזור היוצר חיבור בין אפיונן של פגיעויות שונות אשר נכנסות תחת קטלוג הפגיעויות המביא אותנו ליכולת מניעת פגיעויות אלו באמצעות תהליך פיתוח מאובטח, כאשר בכל אחד מפרקים שולבו מאמרים ומחקרים בנושא.
העבודה מרימה את קרנו של תהליך הפיתוח המאובטח כתהליך יסודי ומעמיק המתכתב בכל שלביו עם איומי תקיפה עתידיים ומוכיחה ששכרו של תהליך זה בצדו.
מסקנת העבודה הינה שאכן ניתן לראות שהרבה מן התקיפות מנצלות פגיעויות הנובעות משגיאות החוזרות על עצמן. שגיאות אלו היו יכולות להימנע על ידי תהליך מדוקדק בשלבי הפיתוח הכולל בדיקות ומחשבה מוכוונת לבניית מערכת מאובטחת נקייה מפגיעויות.

ביבליוגרפיה

[1] https://cve.mitre.org

[2] http://heartbleed.com

[3] www.DigitalWhisper.co.il – Heartbleed – על לבבות שבורים, (גיליון 50, 5.2014)

[4] https://en.wikipedia.org/wiki/Heartbleed

[5] https://en.wikipedia.org/wiki/Shellshock_(software_bug)

[6] https://www.symantec.com/connect/blogs/shellshock-all-you-need-know-about-bash-bug-vulnerability

[7] https://www.dfranke.us/posts/2014-10-14-how-poodle-happened.html

[8] www.DigitalWhisper.co.il - The Poodle attack , (11.2014) גיליון 55

[9] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

[10] https://www.incapsula.com/blog/2014-mega-vulnerabilities.html

[11] https://blog.qualys.com/laws-of-vulnerabilities/2015/01/27/the-ghost-vulnerability

[12] Michael Howard, David LeBlanc : **Writing Secure Code**, 2nd ed. Edition (2003)

[13] https://www.qualys.com/2015/01/27/cve-2015-0235/GHOST-CVE-2015-0235.txt?_ga=2.162238908.1653264067.1532952452-900106939.1532952452

[14] https://blog.checkpoint.com/2017/03/15/check-point-discloses-vulnerability-whatsapp-telegram/

[15] Waqas Ahmad, Zeeshan Hayat, Bilal Zafar, Fawad Ali Khan, Farid ud Din and Iqtidar Shah Gandahara University Peshawar, Pakistan, Agriculture University Peshawar, Pakistan, Ministry of Information Technology, Peshawar, Pakistan: **A survey on Taxonomies of Attacks and Vulnerabilities in Computer Systems,** International Journal of Computer Science and Telecommunications [Volume 3, Issue 5, May 2012]

[16] James A. Whittaker, Mike Andrews : **How to Break Web Software: Functional and Security Testing of Web** (2006)

[17] Gary McGraw : **Software Security: Building Security in** (2006)

[18] https://www.mcafee.com/enterprise/en-us/assets/reports/rp-threats-predictions-2016.pdf
https://www.calyptix.com/top-threats/top-7-network-attack-types-2016/

[19] Abdullah Saad AL-Malaise AL-Ghamdi, Department of Information Systems, Faculty of Computing & Information Technology. King Abdulaziz University, Kingdom of Saudi Arabia : **A Survey on**

Software Security Testing Techniques, , International Journal of Computer Science and Telecommunications [Volume 3, Issue 5, May 2012]

[20] Stuart Short, Zeineb Zhioua , SAP Labs France : **Static Code Analysis for Software Security Verification: Problems and Approaches,** 2014 IEEE 38th International Computer Software and Applications Conference Workshops.

[21] Jose Fonseca, Marco Vieira, University of Coimbra : **A survey on Secure Software Development Life cycle paper**

[22] Cisco Development LifeCycle (CSDL) - https://www.cisco.com/c/dam/en_us/.../cisco-secure-development-lifecycle.pdf

[23] University of Pennsylvania, University of Maryland, Google Project Zero : **Spectre Attacks: Exploiting Speculative Execution**

[24] https://meltdownattack.com/meltdown.pdf

[25] https://danielmiessler.com/blog/simple-explanation-difference-meltdown-spectre/

[26] Verizon 2016 Data Breach Investigations Report, Page 15.

Abstract

Computer system and cyber attacks are a wide subject. Over the years, many ways of dealing with these sorts of attacks were suggested.

The purpose of this paper is to prove that the source of most of these vulnerabilities are derived from a mistake which could have been prevented by developers and to suggest centralizing the defense of cyber-attacks in a secure development lifecycle process.

In this paper I reviewed five vulnerabilities and attacks from different areas from the last few years until I exposed the source of the mistake in each vulnerability, and how it could have been avoided. I showed how the different vulnerabilities can be defined into three main areas – operating system, network and application code – and a sample of different attacks that exploited these vulnerabilities.

The purpose of this categorization is to help us search for vulnerabilities in the different phases of the secure development lifecycle process.

I described in detail what the secure development lifecycle process is and the phases included in the different development stages – plan, develop and test – while integrating types of security tests in different levels as part of the process.

As part of the process description, an emphasis was put on the fact that a secure development lifecycle process is not only the developer's responsibility, but also of other factors involved in the software's circle of life – directors, architects, testers and system integrators.

The secure development lifecycle process includes the following phases:

- System definition
- Threat modeling
- Third party software security
- Static code security
- Vulnerability testing

I showed the disadvantages of this process and the option to overcome these disadvantages by integrating this process as part of the agile process of software development.

As a result of the effectiveness of this process and because of new vulnerabilities in the hardware world, I suggested that a secure development lifecycle process should be implemented in the hardware developing stage as well.

To prove this concept, I developed a vulnerable system, implemented the some of the most common attacks on this type of system and I suggested a general way of implementing a secure code practice as part of a secure development lifecycle process.

My conclusion from this work is that a secure development lifecycle process is a process with a learning curve and requires preparation and resource investment but has a critical and direct impact on the strength of the system. Without this type of process, the system will have real vulnerabilities that can one day be exploited by malicious elements that might attack and cause severe damage to the system. Therefore, implementing the secure development lifecycle process in the correct way is not an option but an obligation.

Table of content

The Open University of Israel

Department of Mathematics and Computer Science

Secure development process as a cyber-attack main defense strategy

Final Paper submitted as partial fulfillment of the requirements

towards an M.Sc. degree in Computer Science

The Open University of Israel

Department of Mathematics and Computer Science

By

Areli Fallach

Prepared under the supervision of Prof. Ehud Gudes And Mr. Itzik Bayaz

December 2018