נוהל פיתוח מערכות מאובטחות

משרד הבריאות

ישראל

משרד הבריאות

משרד הבריאות

נוהל פיתוח מערכות מאובטחות

03/06/2012	חברת אבנת	1.1	כתיבת הנוהל
20/08/2012	תמיר פלדמן	1.2	התאמה לתקן ISO 27799
30/09/2012	שי אמיר	1.3	אישור הנוהל

תוכן עניינים

1. כללי 7
מבוא 7
מטרת המסמך 7
הגדרות 7
אחריות 7
שיטה 8
2. מחזור החיים של מערכת מידע 9
כללי 9
עקרונות בסיסים אליהם יש להתייחס בשלבי הגדרת הדרישות, איפיון ועיצוב מערכת 9
הגדרת דרישות למערכת – Requirements 11
אפיון ועיצוב המערכת – Design 12
תכנון מנגנוני הגנה בשלב איפיון ועיצוב המערכת 12
ניתוח עלות/תועלת וקביעת קדימויות 19
יישום המערכת – Implementation 19
עקרונות מונחים למימוש מנגנוני הגנה 20
בחינת מימוש לשגיאות נפוצות 20
בדיקת המערכת והטמעתה 21
העברת המערכת לסביבת הייצור 22
עדכון ו/או תחזוקה של המערכת 22
הסרת / החלפת המערכת במערכת חדשה 23
3. עקרונות הפיתוח המאובטח 24
הקדמה 24
א"מ ברמת האפליקציה לעומת א"מ ברמת התשתית והתקשורת 24
הפרדה לשכבות 25
בדיקת תקינות קלטים 27
3.1.1. כללי 27
3.1.2. הנחיות פרטניות לסוגי קלט נפוצים 33
3.1.3. תווים מיוחדים 33
3.1.4. תו ה-NULL 34
3.1.5 שימוש בתשתית קיימת ב .NET. 34
קטע הקוד המבצע זאת בצד השרת 35
קריסה מבוקרת 35
פשטות היישום 36
זיהוי משתמשים 36
3.1.5. מנגנוני זיהוי 36
3.1.6. מדיניות משתמשים וסיסמאות 36

3.1.7. מנגנון שינוי סיסמה 37
3.1.8. מניעת User Enumeration 38
3.1.9. שימוש במנגנון יציאה מהמערכת 38
3.1.6 קיים מנגנון לביטול חשבונות במערכת 38
3.1.7 אחסון מאובטח של סיסמאות המשתמשים 38
הרשאות, מידור ובקרת גישה 39
3.1.10. עקרון ההרשאה המינימאלית 39
3.1.11. הפרדת תפקידים 39
3.1.12. סוגי הרשאות 39
3.1.13. ניהול הרשאות 40
3.1.14. חלוקת האפליקציה לאזורי סיווג 40
3.1.15. הרשאות ריצת המערכת 40
מנגנון התיעוד 41
3.1.16. כללי 41
3.1.17. מה לתעד 41
3.1.18. מה לא לתעד 42
3.1.19. לוגים המיוצרים ע"י האפליקציה 42
מנגנון טיפול בשגיאות בלתי צפויות 45
3.1.20. כללי 45
3.1.21. תיעוד 45
3.1.22. הודעות שגיאה 46
עבודה עם בסיס נתונים 47
3.1.23. כללי 47
3.1.24. גישה לבסיס הנתונים 47
3.1.25. הרשאות משתמש 47
3.1.26. שמירת נתונים מזהים בבסיס הנתונים 47
3.1.27. שימוש בשאילתה פרמטרית כתחליף לשאילתות דינאמיות 48
3.1.28. ADO Commands 48
3.1.29. Stored Procedures 48
4. דגשים לפיתוח מאובטח בסביבת **WEB** 49
סניטציה של קלט 49
הצפנה 50
4.1.1. הצפנת תקשורת לשרתי ה- Web 56
מידע שאין להסתמך עליו 57
4.1.2. שדות מסוג Hidden בטפסים 57
4.1.3. שדה ה-Referrer 57
4.1.4. שדות מסוג Password בטפסים 57
4.1.5. כמות וסוג המידע שנקבל ממשתמשים 57
4.1.6. השימוש ב-Size בשדות מסוג Input 57
4.1.7. בדיקות בצד הדפדפן 57

מניעת התקפות DoS 58
Password Auto Complete 58
SeSSion & Cookies 58
ניהול Session 58
4.1.8. הגדרת HTTP meta tags 59
4.1.9. סוגי Cookies 60
4.1.10. אי שמירת מידע רגיש ב Persistent Cookies 60
4.1.11. אי ביצוע Login אוטומטי בהתבסס על Cookie 61
4.1.1. הנחיות נוספות לגבי שימוש מאובטח ב cookie 61
שימוש בתשתיות מערכת ההפעלה 62
4.1.2. זיהוי והרשאות 62
4.1.3. תיעוד האפליקציה 62
4.1.4. מערכת הקבצים בשרת 62
ממשקים עם מערכות חיצוניות 63
הנחיות אבטחה ל- Web Services 64
שימוש ב- XML 66
4.1.5. סיכונים ב-XML: 66
4.1.6. כתיבה מאובטחת בעת שימוש ב-XML 68
מניעת התקפות DoS 69
4.1.7. שימוש יתר ב I/O 69
4.1.8. שימוש יתר במעבד 69
4.1.9. נעילת משתמשים 69
שמירה על Flow 70
5. יישום העקרונות בטכנולוגיות אשר בשימוש במשרד הבריאות 71
הקדמה 71
5.1.1. Strong Name Assemblies 71
5.1.2. הרשאות ב- Assembly 71
5.1.3. Partial Trust Assemblies 74
5.1.4. Unmanaged Code 75
5.1.5. הורשה 75
5.1.6. הגבלה לפי חתימה 75
5.1.7. Serialization 76
5.1.8. Deserialization ממקור לא בטוח 76
5.1.9. Isolated Storage 76
5.1.10. Custom Errors 76
5.1.11. התאמת פירוט הודעות השגיאה 77
5.1.12. ספריות הקריפטוגרפיה של Net. 77
5.1.13. FXCop.exe 77
דגשים ל-ASP.NET 77
5.1.14. אחסון מידע רגיש 77
5.1.15. אימות (Authentication) 79

5.1.16. הרשאות (Authorization) 80
5.1.17. קבלת מידע אודות משתמשים מאומתים 80
5.1.18. אבטחה מבוססת הרשאות 81

6. איומי אבטחת מידע..... 84
הקדמה..... 84
גניבת זהות בעקבות מדיניות סיסמאות לקויה 84
הזרקת שאילתות זדוניות (SQL Injection) 84
Parameter Tampering 84
מניעת שירות - Denial of Service 85
חריגת הרשאות..... 85
טעויות קונפיגורציה 85
"דלתות אחוריות" ואופציות Debug 85
Buffer Overflow 86
עקיפה לוגית - Flow Bypass 86
נפילה לא מאובטחת של אפליקציות 86
יירוט התעבורה (Man in the Middle) 87
ניתוח פרוטוקולים 87
פרצות במוצרי צד שלישי..... 87
נעילת מוות (Dead Lock) 87
מרוצים - Race Conditions 87
איומים ייחודיים לאפליקציות WEB 88
6.1.1. מניפולציות שדות Hidden 88
6.1.2. הרעלתCookies 88
6.1.3. Forceful Browsing..... 88
6.1.4. Cross Site Scripting 89
6.1.5. CSRF Cross-Site Request Forgery 89

7. שינויים באפליקציה ותחזוקת הקוד 92
ניהול הקוד והאפליקציה 92

8. נספח – כלי מתאר (Check List) 93

1. כללי

מבוא

מסמך זה מפרט את הנחיות משרד הבריאות לפיתוח מערכות מאובטחות. המסמך מחולק להנחיות כלליות ולאחר מכן לדגשים עבור טכנולוגיות שונות בהן המערכות מפותחות.

מסמך זה הינו כלי בידי המפתח ואינו יכול להקיף את כלל הנושאים אשר בהם הוא יכול להיתקל.

מטרת המסמך

מטרת הנוהל הינה להנחות מתכנתים, ראשי צוותים ומנהלי פרויקטים, על עקרונות הפיתוח המאובטח וכיצד לכתוב קוד אשר הינו בנוי על פי עקרונות אלו. המסמך כולל דגשים לגבי סביבות תכנות שונות כגון: .NET, סביבות בסיס נתונים (Oracle, SQL Server) אשר נמצאות בשימוש במשרד הבריאות.

הגדרות

#	מושג	תיאור
.1	א.המידע, א"מ	אבטחת מידע
.2	RBAC	Role Base Access Control – בקרת הרשאות מבוססת תפקידים
.3	DAC	Discretionary Access Control – בקרת הרשאות ברמה בדידה לאובייקט הרשאות (למשל ב File System של Windows).
.4	התקפות DoS	התקפות מניעת שירות (Denial of Service).
.5	WAS	Web Application Server
.6	WS	Web Service

אחריות

- אחריות אכיפת הנוהל הינה של מפתחי המערכת, ראשי הצוותים ומנהלי הפיתוח.
- אחריות בקרה על אכיפת הנוהל הינה של מחלקת אבטחת מידע של משרד הבריאות.

נוהל פיתוח מערכות מאובטחות

שיטה

- יש להתייחס לסעיפי הנחיות אלו כמחייבים.
- על המתכנת לקרוא נוהל זה, להבינו ולהטמיע את מנגנוני א.המידע המופיעים בו.
- במידה ולא ניתן לתת מענה לאחד הסעיפים (הרלוונטיים), יש לקבל על כך אישור ממחלקת אבטחת מידע ומחשוב ולתעד את מספר הסעיף והסיבה אשר בגינה הוא לא מתקיים.

2. מחזור החיים של מערכת מידע

כללי

על מנת להטמיע במערכת מנגנוני א.מידע בצורה נכונה ויעילה, יש להתייחס להיבטי אבטחת המידע בכל אחד מהשלבים במחזור החיים של המערכת.

מחזור חיים של מערכת, בדרך כלל מורכב מהשלבים הבאים:

- הגדרת דרישות למערכת.
- אפיון ועיצוב המערכת.
- יישום המערכת (תכנותה).
- בדיקת המערכת והטמעתה.
- העברת המערכת לסביבת היצור.
- עדכון ו/או תחזוקה של המערכת.
- הסרת / החלפת המערכת במערכת חדשה.

כל אחד מהשלבים הנ"ל חייב להתבצע בצורה מאובטחת. נוהל זה מתמקד בכל אחד מהשלבים.

עקרונות בסיסים אליהם יש להתייחס בשלבי הגדרת הדרישות, איפיון ועיצוב מערכת

בתכנון מערך הגנה למערכת קיימות סוגיות תכנון שמופיעות וחוזרות במערכות שונות. בדומה לתכנון אמצעי ההגנה לא ניתן לבנות "ספר מתכונים" לשימוש בהם, אלא עדיף להציגם כעקרונות האוריסטיים בעיצוב אמצעי הגנה:

א. **<u>עקרון החוליה החלשה</u>**, מערכת הגנה לעולם מושפעת באופן החד ביותר מעקרון זה. הרכיב החלש ביותר אבטחתית במערכת קובע את חוזק המערכת כולה. לפיכך בקביעת קדימויות ניתן בהחלט לוותר על פיתוח מנגנון הגנה, גם אם פשוט למימוש, כל עוד האיום אינו בנקודה החלשה של המערכת. צריך לזכור שנקודת החולשה מורכבת גם ממידת החשיפה לאיום, לא רק מהאיום עצמו.

ב. **<u>מינימום הרשאה</u>**, פעולה במערכת צריכה להתבצע בהרשאה המינימלית הדרושה לביצועה, ולזמן הקצר היותר. מערכות רבות נחשפות כיון שפעולות טריוויאליות מתבצעות בהרשאה גבוהה מהנדרש.

ג. **<u>פשטות</u>**, מערכות פשוטות הן צפויות יותר וקלות יותר לבדיקה. על מנת לסייע באיתור כשלים במערך האבטחה צריכה המערכת להיות פשוטה. בניגוד לתכונות פונקציונליות, קשה בהרבה לבדוק מערכת

מבחינה אבטחתית, כיון שאבטחה עוסקת באירועים צפויים פחות. במערכות מורכבות, צריכה מערכת האבטחה להיות פשוטה, וכל פעילויות המערכת, שלהן גוון אבטחתי צריכות להתנקז אליה.

ד. **מצב ברירת מחדל בטוח**, מצב ברירת המחדל של מערכת צריך להיות מצב בטוח. המערכת צריכה להגדיר מה מותר לבצע ולשלול כל פעולה שאינה מתאימה לפרופיל, או שאינה צפויה. כאשר מתבצע מצב כשל במערכת, עדיפה בדרך כלל מערכת נעולה עם מידע אמין על מערכת נגישה עם מידע חשוד.

ה. **נקודות גישה מוגבלות**, מערכת מאובטחת צריכה לאפשר נקודות גישה מעטות, מוגדרות היטב, ושמורות. כיון שלא ניתן לפרוס מערך ההגנה יעיל על שטח נרחב, צריכות נקודות הגישה להיות מעטות, ואמצעי ההגנה צריכים להגן עליהן.

ו. **הגנת עומק**, מערכת צריכה לבנות מספר קווי הגנה, ולהגן על אמצעי ההגנה עצמם. זאת על מנת למנוע אפקט דומינו, התפשטות הנזק ותלות בנקודה קריטית. למשל: מערכת הגישה יכולה לזהות משתמש על ידי מגנון ה- Challenge-Response, לפתוח עבורו את ה- Session, לבנות סביבת ה- Sand-Box ביישום שתתאים לעולם האפשרויות של המשתמש, להכריח את המשתמש להחליף סיסמא על בסיס קבוע (החלפת סוד), לבנות עבורו סיסמא אקראית, ולנקות את הסיסמא מהזיכרון עם סיום ה- Session. ערכת שקולטת מידע ממערכת משיקה צריכה לוודא זיהוי של מקור המידע (למשל על ידי חתימה דיגיטלית), זיהוי שלמות נתונים פנימית באמצעות ה-Message Authenticating Code, לבדוק על ידי מספר סידורי ו- Checksum שהמידע התקבל בסדר הנכון ובשלמותו, לבצע בדיקת תחביר/תוכן כדי לוודא שהמידע מתאים להנחות פנימיות של המערכת, לקלוט את המידע עם מנגנון Rollback, ולבצע רישום על גבי ה- Audit Trail. מערכת ה- Audit עצמה צריכה לוודא על ידי מנגנונים דומים (בקרת גישה, Checksum) שנתוני ה- Audit נמצאים במצב תקין.

ז. **שימוש ברכיבים קיימים**, מנגנון אבטחה אמין הנו קשה לתכנון ולמימוש. על פי רוב עוברות שנים של התקפות חוזרות ונשנות עד שרכיב אבטחה נעשה אמין מספיק. שימוש ברכיבים קיימים, למילוי פונקציות במערכת האבטחה הוא הכרח. "המצאת גלגלים" הנה במרבית במקרים מתכון למערכת חלשה.

ח. **החלטות של משתמשים**, אבטחת מערכת צריכה להתקיים ללא תלות בהחלטות משתמש. משתמש בהיכרך תופס את מערכת האבטחה כנטל, או במקרה הטוב, לא מכיר את עולם האיומים. לפיכך קבלת החלטות אבטחה אסור שתהיה בתחום ההחלטה של המשתמש.

ט. **<u>השארת ראיות</u>**, במקרים רבים מניעת ההתקפה או אפילו גילוי ההתקפה בזמן אמת אינו אפשרי. מערכת שדואגת להשאיר ראיות, מאפשרת לפחות לשקם את הנזק, ולמצוא דרכים להתמודד עם ההתקפה הבאה. בנוסף ,השארת ראיות מאפשרת איתור מקור הפגיעה (Accountability) שגם נחוץ למניעה עתידית.

י. **<u>שימוש בעיקרון הקרקהופס</u>** , עקרון קרקהופס (Kerckhoffs) מתייחס בעיקר למערכות הצפנה, בהן אבטחת המערכת לא יכולה להיות תלויה בסודיות אלגוריתם ההצפנה אלא רק במפתח. יש להניח שבמוקדם או במאוחר ייחשפו אמצעי ההגנה. המערכת צריכה להתמודד עם חשיפה כזו בהצלחה.

יא. **<u>פרטיות</u>**, למרות העיקרון הקודם, אין צורך לפרסם את מנגנוני האבטחה הקיימים, על מנת להכריח את התוקף לנוע באזור לא מוכר. דליפת מידע על מערכת ההגנה מאפשרת לתוקף זמן לתכנון מוקדם, ומעבירה את היוזמה אליו.

יב. **<u>הפרדת רשויות</u>**, מערכת בקרה צריכה להיות מחוץ לעולם המבוקר. מערכת שנמצאת בטווח ידו של הגורם/אירוע מבוקר היא על פי רוב חסרת משמעות. עקרון זה תקף
למנגנוני Audit ואמצעי גילוי שונים, כמו גם למערכות מניעה. רכיב האבטחה שמהווה חלק ממערכת ההפעלה נמצא מחוץ להישג ידם של המשתמשים.

הגדרת דרישות למערכת – REQUIREMENTS

בשלב זה של יש להגדיר גם את דרישות אבטחת המידע הרלוונטיות למערכת בנוסף לדרישות האחרות.

דרישות אלו צריכות לכלול בין היתר את הנושאים הבאים:

- שמירה על פרטיות המשתמשים.
- שמירה על חיסיון הנתונים.
- שמירה על שלמות המידע.
- שמירה על אמינות המידע.
- שמירה על זמינות המערכת.
- שמירה על שלמות תהליכים עסקיים.
- יכולת לשחזר אירועי א.מידע.
- מניעת התכחשות משתמשים לביצוע פעולות.

- זיהוי חזק של משתמשים.
- זיהוי חזק בין תת מערכות ומערכות חיצוניות (בסיס נתונים, mainframe,שירותי רשת וכדומה).

אפיון ועיצוב המערכת – DESIGN

בשלב זה מתכננים **איך** המערכת תיתן מענה לכל אחת מדרישות א.המידע. יש לפרט את המנגנוני א.המידע אשר ימומשו, איומים עיקריים על המערכת ודרכי התגוננות. לאחר ששלב האפיון מסתיים יש לעשות בדיקת א.מידע (Design Review) בשיתוף מחלקת אבטחת מידע. במסגרת הבדיקה יש לבצע Attack on Paper למערכת.

בין היתר, מסמך האפיון אמור לתת מענה למימוש המנגנונים הבאים:

- הצפנת נתונים רגישים.
- בחירת מנגנון זיהוי משתמשים.
- תיעוד אירועי אבטחת מידע.
- תיעוד פעולות משתמשים.
- תיעוד פעולות תחזוקת מערכת.
- שיטת ההרשאות ומימושה.
- מנגנון נעילת משתמשים.
- מדיניות ניהול משתמשים.
- ממשקים מאובטחים למערכות נוספות.
- פירוט של רכיבי צד שלישי אשר יהיו בשימוש.

תכנון מנגנוני הגנה בשלב איפיון ועיצוב המערכת

תכנון מנגנוני הגנה נועד לספק רכיבי בקרה (Controls) המיועדים להתמודד עם האיומים שזוהו בשלב זה. בדומה לשלב תכנון הפתרון הפונקציונאלי במערכת מידע, לא ניתן לספק "ספר מתכונים" המכיל פתרונות מן המוכן. עם זאת, ניתן למפות את סוגי הפתרונות הידועים, כדי להקל על איתור או פיתוח של מנגנוני הגנה.
ניתן לחלק את כל מנגנוני ההגנה לשלושה טיפוסים עיקריים:

- **מניעה**
- **גילוי**
- **תגובה**

ובתוכם לטיפוסי משנה. הטקסונומיה להלן מנסה לפרט מנגנוני הגנה ברמה מופשטת, אשר אינם אופייניים למערכות מידע בלבד. יש לזכור שמנגנוני ההגנה המפורטים להלן מיועדים למגוון טיפוסי התקפות על מידע:

- **סודיות**
- **שלמות**
- **נגישות**

מנגנוני מניעה מנסים למנוע את התרחשות הנזק למערכת **באופן פסיבי.** אמצעי מניעה לעיתים קרובות משולבים באמצעים מטיפוסים אחרים כדי לאפשר הגנה אקטיבית. ניתן לחלקם בין אמצעים שתפקידם צמצום סבירות הנזק וכאלה שמיועדים לצמצום היקף הנזק.

דוגמאות לאמצעי הגנה:

הטבלה הבאה מדגימה מגוון אמצעי הגנה נפוצים במערכות מידע. אמצעי ההגנה מחולקים לפי שלושה חתכי העל שהוגדרו עבור סוגי הפגיעה במידע: סודיות, אמינות (שלמות) ונגישות (זמינות). חלק מאמצעי ההגנה ניתנים לשימוש כפתרונות ליותר מבעיה אחת ולכן מופיעים במספר קטגוריות. עבור כל אמצעי הגנה מפורטים האיומים מולם הוא מנסה להתמודד ושיטת ההגנה. כמו כן מובאות מספר דוגמאות לשימוש במנגנון במערכות קיימות.

א. אמצעי הגנה על סודיות המידע:

אמצעי	אופי ההגנה	דוגמאות
הצפנה	אמצעי מנע, הפועל על ידי הורדת הסבירות להתרחשות הנזק באמצעות הרמת עלות התקיפה הנדרשת כדי לשבור את מנגנון ההצפנה.	הצפנת תקשורת בין רכיבים בתוך המערכת, הצפנת תקשורת עם רכיבים משיקיים, הצפנת מידע על גבי מדיית אחסון (למשל: קובץ סיסמאות, רשומות בבסיס הנתונים), הצפנת מידע על מערכת הגיבוי, מנגנון Challenge-Response להזדהות.
Watermarking	אמצעי מנע המשתמש בהסוואה ופועל כ-"דיו סתרים" דיגיטלי כדי להסתיר את עצם קיומו של המידע (בשונה מהצפנה שמנסה להסתיר את תוכנו) פועל על ידי צמצום חשיפה והרמת עלות התקיפה	אמצעי Watermarking משמשים להחבאת מידע רגיש בתוך מידע נושא (Cover) באופן שקשה לזהות את קיום המידע או להסירו: העברת מידע סודי בתוך תמונות/אודיו, החבאת מנגנוני הגנה בתוך מורכבות פונקציונאלית של מערכת התוכנה, וכדומה.

אמצעי	אופי ההגנה	דוגמאות
פיתיון, Fingerprinting	אמצעי גילוי לחשיפת מידע שפועל על ידי הוספת מידע ייחודי אך שגוי למאגר הרגיל. מידע זה, כאשר נחשף בתווך גלוי משמש אות שבוצעה חדירה, ומאפשר לזהות את מקורה.	הכנסת ידיעות כוזבות למערכת מודיעינית פנימית, הכנסת תיק רפואי פיקטיבי של אישיות בעלת חשיפה תקשורתית גבוהה למאגר הרפואי כדי לגלות אם מתבצעת הדלפה, הכנסת מידע ייחודי לתמונות/אודיו כדי לטפל בבעיית זכויות יוצרים (לזהות איזה משתמש חוקי מאפשר העתקת המידע).
החלפת סוד	אמצעי שפועל לשמירת "טריות" המערכת. יכול לפעול בשני אופנים: כאמצעי מנע לצמצום חלון החשיפה, או כמנגנון תגובה על ידי יצירת תחליף בתגובה לחשיפה.	החלפת מפתחות הצפנה (Revocation), החלפת סיסמאות באופן קבוע, החלפת מנגנוני הגנה/נתיבי גישה למשאבים, החלפת כתובות רשת, הרשאות לזמן מוגבל.
מחיקת מידע	אמצעי מנע שפועל לצמצום החשיפה על ידי מחיקת המידע הרגיש ממדית האחסון ברגע שאינו נחוץ יותר.	מחיקת מפתחות הצפנה מהזיכרון/קבצי מערכת. ניקוי רכיבי סנכרון/קבצים זמניים. מחיקת זיכרון שמכיל סיסמאות, מחיקת מידע מקורי לאחר הצפנתו.
אקראיות	אמצעי מנע שמנסה למנוע אפשרות לחזות תכולת מידע רגיש במערכת, ובכל להעלות את עלות התקיפה.	סיסמאות אקראיות שמוגרלות על ידי המערכת, סיסמאות מעורבות (תווים + ספרות) מפתחות הצפנה אקראיים, מספרים סידוריים עם התחלה אקראית וכן הלאה.
הורדת רזולוציה/ הוספת רעש	אמצעי מנע לצמצום חשיפה במערכות בהן נדרשת חשיפה חלקית. מנגנון ההגנה מונע גישה למידע שלם, אבל מאפשר גישה למידע חלקי.	מערכת רפואית שמנסה להגן על פרטיות החולה, ועדיין לאפשר שאילתות סטטיסטיות ההגנה תתבצע על ידי: הסתרת פרטים מזהים (שם, גיל), מניעת שאילתות סטטיסטיות על קבוצות קטנות, הוספת רעש אקראי למדגם, שינוי פרטים מזהים (למשל תווי פנים בתמונה) וכדומה.
תגובה מושהית	אמצעי מנע שפועל לצמצום החשיפה והאטת התפשטות הנזק. מנגנוני תגובה מושהית משתמשים להסתרת אמצעי ההגנה עצמם על ידי ביצוע התגובה רחוק מזמן איתור התקיפה.	הצגת הודעת שגיאה רק לאחר הקלדת שם+סיסמא, גם אם השם אינו קיים. הפעלת מנגנוני תגובה (למשל חסימת פעולות) בהשהיה ביחס לביצוע העברה, הפניית התוקף לסביבה מוגנת תוך ניסיון לאתר את מקור התקיפה.

אמצעי	אופי ההגנה	דוגמאות
השמדה עצמית	אמצעי תגובה שמבצע השמדת מידע רגיש כתגובה לזיהוי חדירה. על פי רוב משתמש להגנה על מנגנוני ההגנה עצמם.	מנגנון Tamper resistance בכרטיסים חכמים, שמבצע מחיקה של מפתחות ההצפנה בתגובה לניסיונות חדירה פיזיים (באמצעות מעגל חשמלי מיניאטורי שנסגר עם חיבור probe למעגל הראשי, או איתור ניסיונות לניתוק מקור הכוח), סיסמאות משתמש שנמחקות לאחר מספר ניסיונות שגויים עד שמנהל המערכת יוצר חשבון חדש עבורו.

ב. אמצעי הגנה על שלמות המידע:

אמצעי	אופי ההגנה	דוגמאות
שער (Gate)	אמצעי מנע נפוץ שפועל באמצעות ניתוב כל הגישות למידע למספר מוגבל של נקודות מוגנות היטב. באופן זה מושגת הקטנת חשיפה יחד עם הרמת עלות התקיפה.	בניית בקרת גישה במערכת ברוטינה אחת, מערכות Firewall, הזדהות מול מערכת הפעלה או Database ככלי בקרת גישה מרכזי למערכת, מערכות Single Sign On.
אישור	אמצעי מנע הפועל להורדת סבירות הפגיעה באמצעות בקשת אישור לביצוע הפעולה, או הספקת אישור שהפעולה בוצעה.	אזהרה לפני ביצוע פעולה במערכת (יציאה, מחיקה, ביצוע), אישור בפרוטוקול תקשורת (אישור קבלה)
אישור כפול	אמצעי מנע הפועל להורדת סבירות הפגיעה באמצעות חלוקת אחריות ודרישת אישור נוסף (למשל על ידי גורם מאשר בלתי תלוי) לביצוע פעולה.	כפל חתימות בפעולה מורכבת ,כפל מפתחות במערכת נשק גרעינית, זיהוי מחודש לפני בצוע פעולה רגישה במיוחד, הזדהות כפולה בכניסה למערכת (למשל סיסמא + טביעת אצבע, כרטיס מגנטי + קוד)

אמצעי	אופי ההגנה	דוגמאות
Checksum, Hashing	אמצעי זיהוי נזק למידע על ידי הוספת נתון שמאפיין את המידע באופן חד ערכי ביחס מידע ששודר מנקודה מסוימת.	סיפרת ביקורת בשדה ת.ז., בדיקה שרכיבי מערכת לא השתנו (נוסח Tripwire), חתימות התקפה על מערכת (חתימות באפליקציות אנטי-וירוס או IDS), חתימת יחידת מידע משודרת (TCP/IP), הוספת CRC לבדיקת שלמות בקובץ שמועבר בין מערכות שונות.
Error correction codes	אמצעי זיהוי ותגובה להתאוששות ותיקון נזק. קודי תיקון שגיאות משתמשים ביתירות מידע על מנת לתקן שגיאות אקראיות בזמן העברת יחידות מידע	רכיבי תקשורת, רכיבי חומרה וזיכרון.
בדיקת תחביר/תוכן	אמצעי לזיהוי מידע לא תקין, על פי רוב בזמן קלט ממערכות חיצוניות לדוגמא: משתמש, מערכת ההפעלה, מערכות משיקות רכיבי צד ג'. המערכת מבצעת בדיקות באם המידע הנקלט מתאים להנחות התחביריות והתוכניות שהיא מניחה לגביו, ולא מאפשרת כניסת מידע שאינו עומד בתנאי הכניסה	בדיקות פורמט שדות (ערך אלפא-נומרי בשדה תאריך), בדיקות ולידציה (ערכים בטווח נכון או בתנאים לוגיים מתאימים, יחס בין ערכים ברשומה נקלטת), בדיקת קודי שגיאה מרוטינות של מערכת ההפעלה.
בדיקת אנומליות	אמצעי לזיהוי פעולות מותרות ואסורות על ידי איתור התנהגות חריגה במערכת, או מידע בעל משמעות שגויה באמצעות.	מערכות בנקאיות (סכומים לא שגרתיים, בהיקף לא שגרתי ללקוח, סדר פעולות תמוה), מערכות רפואיות (סדר טיפולים לא שגרתי, תרופות לא מתאימות, מינון לא תקין)
Double Entry	אמצעי זיהוי שנועד לאתר בעיות שלמות מידע באמצעות תכונה שחייבת להישמר במערכת לאחר כל פעולה מותרת. אם המערכת אינה מכילה את אותה תכונה. פירושו של דבר שנפגעה שלמות הנתונים.	במערכות הנהלת חשבונות (למשל בנקאות), כל התנועות חייבות להתאזן. במערכת מלאי, ירידת כמות מלאי חייבת להתבטא דרך פעולת מכירה של כמויות בסכומים זהים, בכניסה פיזית של אנשים לאתרים מאובטחים, מספר הכניסות חייב להתאזן עם מספר היציאות כדי לוודא שאיש לא נותר בפנים בסוף יום.
חתימה דיגיטלית	אמצעי זיהוי משתמש. על פי רוב פועל בשיתוף עם מנגנון Hashing (לעיל) כדי להבטיח שלמות מקור המידע יחד עם שלמות פנימית של המידע עצמו.	זיהוי שרת בפרוטוקול SSL, מערכות PKI המאפשרות חתימה על מסמכים. Message Authenticating Code (MAC): זיהוי חד ערכי של שולח ומשלוח
הזדהות הדדית (Handshake)	מנגנוני הזדהות הדדית פועלים לזיהוי של כל הצדדים המעורבים בפעולה. אמצעים אלה פועלים למניעת	מערכת Kerberos, פרוטוקולי Key Exchange בהצפנה. מנגנון הזדהות בין כרטיס חכם ל-

אמצעי	אופי ההגנה	דוגמאות
	התקפות Man-in-middle (מצב של התחזות לאחד הצדדים)	Reader. מנגנון הזדהות של המערכת למשתמש ולהפך.
Callback	אמצעים לזיהוי מקור המידע (למשל משתמש) על ידי ניתוק הקשר, והתקשרות מחודשת של ספק השירות ללקוח. (תוך בדיקה אם הלקוח זכאי לשרות כזה)	התקשרות יזומה למערכות משיקיות, לאחר שאלה מבקשות מידע, חיוג חוזר של מערכות תחזוקה אל המפעיל.
Audit Trail	אמצעי השארת ראיות, הפועל על ידי תיעוד כל פעולה, בעלת השלכה אבטחתית, על גבי מדיום נפרד מהמערכת הראשית, באופן שמאפשר ניתוח שלאחר מעשה	קבצי Log של מערכות הפעלה או אפליקציות, דוחות שינויים/הבדלים תקופתיים, מערכות היסטוריה
גישה אקסקלוסיבית	אמצעי מנע שמוודא גישה אטומית ליחידת מידע כך שלא נוצר מצב בו שני גורמים עצמאיים מנסים בו בעת לשנות את המידע, כך שבסוף התהליך מתקבל מידע שגוי.	נעילת רשומות/שדות/טבלאות במערכת, נעילת משאבי מערכת הפעלה משותפים שנמצאים בשימוש (זיכרון, קבצים, מנגנוני סנכרון).
Rollback	אמצעי תגובה שמוודא ביצוע אטומי של פעולה (או סדרת פעולות) על ידי שיחזור מצב קודם שידוע שהינו תקין, אם פעולת עדכון הנתונים לא הסתיימה בשלמותה.	מגנוני Transaction Processing בבסיסי נתונים, מערכות גיבוי/העברה להיסטוריה. קליטת אוסף רשומות ממערכות משיקות כיחידה שלמה.
מספר סידורי רציף	אמצעי גילוי שנועד לגלות בעיות שלמות ברצף של פעולות (שלמות הטרנזקציה) על ידי שמירה של רצף, בו חייבים כל הערכים להופיע, ובסדר הנכון.	מספר חשבונית רציף, זמן ביצוע פעולה, מספר רשומה בתוך Batch, מספר Packet בפרוטוקול תקשורת, מערכות Time stamping.
Session	Session הוא אמצעי נפוץ שמנסה לצמצם חשיפה שכרוכה בהזדהות משתמש עבור ביצוע כל פעולה. מנגנון Session מלווה את המשתמש משלב ביצוע הזיהוי הראשוני (login) ועד סיומו, ומזדהה עבורו לביצוע פעולות. מערכות אלה כוללות בדרך	מערכות הפעלה, מערכות Single sign on, פרוטוקולי תקשורת (למשל TCP)

אמצעי	אופי ההגנה	דוגמאות
	כלל Sign-Off אוטומטי בעקבות חוסר פעילות כדי לצמצם עוד יותר את חלון החשיפה	
Fingerprinting	מנגנוני Fingerprinting פועלים להרמת עלות התקיפה על ידי יצירת גרסאות יחידניות של מערכת ההגנה. שיטה כזו מקשה על בניית מנגנון התקפה אוטומטי.	הוספת salt לפני הצפנת סיסמאות בקובץ סיסמאות, שידור בתדר מדלג (spread spectrum) שמונע חסימת שידור, מנגנון מוטציה שמשמש וירוסים להגנה עצמית מפני גילוי.
פיתיון (Decoy)	מערכות שמטרתן לצמצם את סבירות התקיפה על ידי הכנת אוסף מערכות דמי מקבילות שניתנות להקרבה, כך שהתוקף מבזבז משאבי תקיפה על מערכות חסרות חשיבות	לא ידוע עדיין על אמצעי הגנה מסוג זה במערכות מידע.
בלימה הדרגתית	בלימה הדרגתית היא יותר ארכיטקטורה מאשר אמצעי הגנה בודד. מערכות הגנה הדרגתיות מנסות למנוע אזעקות שווא על ידי בניית ההגנה במעגלים פנימיים כאשר כל מעגל הוא בעל סף חדירה גבוה יותר, ובעל רגישות גילוי גבוהה יותר. באופן כזה נחסמות ההתקפות הפשוטות בקלות על ידי המעגל החיצוני, ומערכת ההגנה יכולה להתייחס ברצינות רבה יותר לתקיפות מורכבות.	בדיקת אנומאליות של מידע רק לאחר שהסתיימה בדיקת תחביר, מערכת Intrusion detection מאחורי מערכת Firewall, מערכת שחוסמת גישה לאחר שלושה ניסיונות כושלים אבל מתריעה למנהל המערכת לאחר עשרה ניסיונות.

ג. אמצעי הגנה על נגישות המידע:

אמצעי	אופי ההגנה	דוגמאות
הוספת אמצעי גישה למשאבים	אמצעי מנע הפועל בדרך של יתירות על ידי הכפלת נתיבי גישה למשאבים, כדי לוודא שגם אם נחסם נתיב גישה	מערת RAID, מערכות Clustering, הכפלת רכיבי תקשורת, מערכות אל-פסק

אמצעי	אופי ההגנה	דוגמאות
	יחיד, יכולה המערכת לאפשר גישה (לפחות חלקית).	
מעקף	מעקף הוא מנגנון Fallback אקטיבי למקרה שגישה למשאבים קריטיים נחסמת. מעקף מעביר את המידע (או את נתיבי הגישה אליו) למערכת חליפית על מנת לצמצם את הנזק.	מערכות אל-פסק, שרת נתונים משני עם נתונים חלקיים, בסיס נתונים משני עם יכולות קריאה בלבד.
סינון / ויסות	אמצעי תגובה שנועד לנהל גישה למשאבים באופן דינמי על ידי בקרה על תווך הגישה למשאבים ובדיקת בקשות הגישה אליהם.	תיעדוף משתמשים בגישה למשאבים על בסיס עומס, Load balancing של אמצעי גישה למשאבים, ניתוק אפיק תקשורת ממנו מתבצעת תקיפה, שינוי הקצאת מקום בשרת על בסיס צורך, override גישה למשתמשים בפרופיל גבוה בזמן חרום.

ניתוח עלות/תועלת וקביעת קדימויות

לאחר תכנון אמצעי ההגנה למערכת מתבצע ניתוח עלות/תועלת עבור האמצעים השונים. על פי רוב קיימים מספר אמצעי הגנה שנותנים מענה לאיומים שאותרו קודם לכן, ומדיניות האבטחה משמשת לקביעת קדימויות לגבי בחירת אמצעי מסוים. שיקולים בהעדפת אמצעי אבטחה נובעים מעלות כספית שחל מימוש הפתרון, עלות ונגישות כוח האדם הנדרש לתחזוקתו, זמן המימוש, עלות הפתרון כנגד עלות האיום וכדומה.

יישום המערכת – IMPLEMENTATION

- בשלב זה מפותחת המערכת ומנגנוני א.המידע אשר תוארו קודם לכן.
- כתיבת קוד המקור צריכה להתבצע בהתאם לנוהל "נוהל פיתוח מאובטח - כתיבת קוד מאובטח" (מסמך זה)
- לאחר סיום מימוש המערכת, מומלץ לבצע סקר קוד (Code Review) על מנת לוודא את יישום דרישות אבטחת המידע. רצוי שסקר זה יתבצע ע"י מומחה אבטחת מידע אפליקטיבי חיצוני ולא ע"י מתכנתי המערכת, על מנת שהבקרה תתבצע ע"י גורם מומחה בלתי תלוי.

נוהל פיתוח מערכות מאובטחות

עקרונות מונחים למימוש מנגנוני הגנה

במימוש מנגנוני ההגנה מתבצע מאמץ ליישם את הרכיבים שתוכננו, מבלי לייצר מפגעי אבטחה נוספים. בדומה ליישום הפתרון הפונקציונלי, איכות היישום מושפעת משיטות עבודה מובנות, ומאוסף של "טעויות קלאסיות" מהן מנסה המיישם להימנע. במקרים רבים, מגבלות סביבת היישום (שפה, מערכת הפעלה, רכיבי צד ג') מחייבים תכנון מחודש של חלק ממנגנוני האבטחה.
שיטות העבודה ביישום הפתרון, משותפות לפיתוח הפתרון הפונקציונלי ולפיתוח אמצעי האבטחה. ככלל, יש לבצע:

- ✓ תיעוד מערכת מלא הכולל: רכיבים, מבנה נתונים, נתיבי תקשורות,תיאורים דינמיים ותלויות ברכיבי צג שלישי(מערכות הפעלה,רכיבים מוכנים וכולי).
- ✓ התאמה לתכנון, יש לבדוק שמימוש המערכת נעשה לפי קווים המנחים.
- ✓ בקרת איכות.
- ✓ Code Review, יש לעשות בחינת קוד על ידי מומחה אבטחה.

בחינת מימוש לשגיאות נפוצות

יישום מערכות אבטחה סובל משגיאות טיפוסיות שחוזרות במערכות רבות. קיימים מקורות רבים הסוקרים בעיות בסביבות ספציפיות (בשפות כגון: C, C++,C#, Java, או במערכות הפעלה כגון: WinNT, Unix, OS2 וכדומה) אין כוונה במסמך זה לסקור את הסביבות הספציפיות. הרשימה המובאת להלן סוקרת מספר בעיות נפוצות מבלי להתייחס לסביבת יישום מסוימת:

1. **שגיאות תנאי:**

שגיאות הנוצרות בעקבות בדיקה חסרה או לא מספקת של תנאי זרימה. על פי רוב כתוצאה של בדיקה חסרה של פרמטרים, נתונים שהוקלדו על ידי המשתמש, קודי סטטוס שמוחזרים ממערכת ההפעלה וכן הלאה. שגיאות תנאי גורמות כיום ליותר מ-90% של בעיות האבטחה ביישום מערכות. כל בעיות ה- Buffer Overflow נובעות מבדיקה לא מספקת, או חסרה לחלוטין של פרמטרים למערכת.

2. **מצבי אטומיות:**

שגיאות שנוצרות כאשר לא הובאה בחשבון הפרעה בזרימת התוכנית. במקרים רבים קיימת הנחה, שזרימת התוכנית של תופרע. הנחה זו מוטעית לחלוטין במערכות מרובות משתמשים, בעלי משימות מרובות או מספר מעבדים. מצבי Race Conditions, מהווים אחד הגורמים המובילים לבעיות

אבטחה. לדוגמא: מערכת שבודקת תאימות של שדה להנחות תחביר, אבל לא מביאה בחשבון שהמשתמש יכול לשנות את השדה בין זמן הבדיקה וזמן הכנסת המידע לבסיס הנתונים.

3. **<u>דלתות אחוריות</u>:**

שגיאות מכוונות שנוצרות כאשר מפתחי המערכת השאירו רכיבי בדיקה או תחזוקה שעוקפים את מערכת האבטחה, או לחלופין חושפים מידע חסוי על המערכת (מפתחות, סיסמאות וכדומה). ברוב במקרים מושארים דלתות אחוריות ללא כוונת זדון מצד צוות הפיתוח, אבל אין להוציא את האפשרות מכלל חשבון.

4. **<u>מצבי קצה</u>:**

סוג של שגיאת תנאי שנוצר כאשר המערכת אינה בודקת תנאי קצה: משאבים מצומצמים (זיכרון, מקום פנוי בדיסק, גישה בעומס גבוה) וכתוצאה פועלת באופן לא צפוי.

5. **<u>גבולות בין רכיבים</u> :**

כאשר מידע זולג מרכיבים מאובטחים לרכיבים ציבוריים, או כאשר קיימת אפשרות גישה בנתיב ההפוך, בעיית גבולות קורה פעמים רבות דרך השימוש במשאב משותף לדוגמא: כתיבת מידע חסוי לזיכרון משותף, מחייבת מחיקתו לפני שהזיכרון יעבור לשימוש רכיב אחר, או ייכתב ל-Swap File.

6. **<u>שגיאות אינטגרציה</u>:**

שגיאות שנובעות מבדיקה חסרה של סביבת היישום. כאשר שילוב המערכת ברכיבי סביבה יוצר מצב לא בטוח (למשל מערכת הפעלה ללא בקרת גישה, רכיבי צד ג' לא בטוחים, רכיבי תקשורת לא בטוחים וכד').

7. **<u>זליגת מידע</u>:**

כאשר המערכת "מנדבת" מידע פנימי ללא צורך שעשוי לסייע לתוקף למקד את ההתקפה לדוגמא: גרסה, משתמש נוכחי.

בדיקת המערכת והטמעתה

בשלב זה המערכת מתפקדת באופן מלא ונעשות בדיקות איכות על מנת למצוא תקלות אפשריות באופן פעולתה השוטף. בדיקות איכות אלו נעשות באמצעות צוות ה- QA, אשר מתעד כל תקלה, מעביר אותה למתכנתים ובודקת אם תוקנה. <u>במקביל לשלב זה או אחריו</u> יש לבצע בדיקות א.מידע למערכת כולה. האחריות לבדיקות אלו היא של אגף א.המידע ומבלי אישורה, המערכת לא תעלה לסביבת הייצור.

ניתן לבצע את בדיקות א.המידע במספר אופנים:

- **White-Box** – מעבר על קוד המקור של המערכת בהביטי 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 delta של השינוי בלבד ואין צורך לבדוק את כל המערכת מחדש.
- **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.
- **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 ביעילות שלה הן מבחינת זמן והן מבחינת היסודיות.

יש להדגיש:

- בכל אחת מהבדיקות יש לקבל דו״ח המתאר את ממצאי א.המידע והמלצות לתיקונם.
- יש לקיים ישיבה טכנית עם נציג מצוות הפיתוח, לאחר שקרא את הדו״ח, עם יועץ א. המידע שבדק את המערכת. בישיבה זאת יוסברו הממצאים ויוחלט על זמן לתיקונם.
- לאחר תיקון הממצאים יש לערוך בדיקת א. מידע חוזרת.
- יש לקבל את אישורו של מנהל אבטחת מידע לצורך העלאת המערכת לסביבת הייצור לאחר ביצוע תיקון של כל הליקויים שנמצאו.

העברת המערכת לסביבת הייצור

את המערכת אשר אושרה על ידי מנהל א.מידע ניתן להעביר לסביבת הייצור תוך כדי הקפדה על מספר כללי יסוד הבאים:

- יש להחליף את כל הסיסמאות של המערכת למזהים קשים לניחוש על פי מדיניות הסיסמאות של משרד הבריאות.
- יש לדאוג מחיקת כל קוד מערכת מיותר שנועד לצורכי Debug, כגון מחלקות עזר (utility classes) העוזרים לפתח את האפליקציה ולבצע בדיקות של המערכת
- יש למחוק את כל חשבונות המשתמשים אשר היו קיימים במערכת.
- יש לדאוג להפרדה מלאה בין סביבת היצור לסביבות נוספות (פיתוח, בדיקות וכדומה) ולהשתמש במזהים שונים עבור כל סביבה.

עדכון ו/או תחזוקה של המערכת

לאחר שהמערכת פועלת בסביבת הייצור, יש לדאוג לבצע עדכוני א.מידע שוטפים לתשתית המערכת (Patches).

נוהל פיתוח מערכות מאובטחות

כל עדכון אפליקטיבי במערכת מחייב בדיקה ואישור של מחלקת אבטחת מידע. במידה והתבצע Code Review על המערכת, ניתן לבדוק רק את החלק החדש שהשתנה. במידה ולא התבצע Code Review , יש לבדוק את כל המערכת מחדש על מנת לוודא כי רמת א. המידע של המערכת לא נפגעה.

מעבר לכך, יש לבצע בדיקות Black-Box תקופתיות על מנת לוודא שטכניקות פריצה חדשות או ששינויים במערכת לא עלולים לפגוע במערכת ובמשתמשיה.

הסרת / החלפת המערכת במערכת חדשה

כאשר מעדכנים גרסת מערכת או מחליפים אותה במערכת חדשה, יש לבצע גיבוי מלא למערכת הישנה ולנתוניה על מנת שנוכל לבצע שיחזור במידת הצורך.

יש לתת דגש על שמירת דיוק הנתונים בעת העברתם למערכת החדשה.

יש לשים לב שבשלב המעבר, לא נחשפים נתונים רגישים של המערכת לגורמים פנימיים ו/או גורמים חיצוניים.

את המערכת הישנה ואת נתוניה יש לשמור בגיבוי במקום אשר מאובטח פיסית (כספת / חדר נעול) על מנת למנוע זליגת הנתונים. כמו כן, במידה והמערכת החדשה מותקנת על גבי חומרה חדשה, יש לוודא שהכונן הקשיח עליו הותקנה המערכת הישנה גובה ואחר כך הושמד פיסית.

3. עקרונות הפיתוח המאובטח

הקדמה

פרק זה כולל את עקרונות הפיתוח המאובטח אשר כל מתכנת צריך להכיר על מנת לבנות מערכת אשר הינה מאובטחת ברמת קוד התוכנה.

א"מ ברמת האפליקציה לעומת א"מ ברמת התשתית והתקשורת

המידע אשר מנוהל באמצעות אפליקציות ומערכות המידע הוא המשאב העיקרי של כל ארגון. כל אפליקציה/מערכת מידע בדרך כלל כוללת קהל יעד שונה ומציגה פונקציונאליות שונה. קהל היעד יכול להתחלק ללקוחות משרד הבריאות, לקבוצות ממוקדות של לקוחות, לעובדי המשרד, לקבוצות ממוקדות של המשרד כגון: רופאים אחיות ועוד. הפונקציונאליות יכולה להשתנות החל ממערכות שמציגות מידע שיווקי בלבד למערכות אשר מאפשרות לבצע פעולות מורכבות ורגישות.

בשל היותן משאב רגיש וחיוני, יש לאבטח אפליקציות מפני גישה בלתי מורשית ומפני ביצוע פעולות בלתי חוקיות. עיקר העשייה בתחום אבטחת המידע בארגונים כיום מתמקד באופן מסורתי דווקא בשכבת התשתיות ולא במרחב האפליקציה. התשתיות הכוללות את ציוד התקשורת, החומרה, מערכות ההפעלה ותוכנות הבסיס והיו באופן מסורתי מוקד להתקפות של גורמים עוינים וזאת בשל פגיעותן הרבה והיכולת הקלה של תוקפים לבצע פעולות בלתי מורשות באמצעות סוג זה של התקפה. אולם, בשנים האחרונות חל מפנה מסוים בכיוון זה עקב מעבר לשימוש באפליקציות WEB אשר חשפו את הארגונים להתקפות גם ברמת האפליקציה.

מוצרי אבטחה לתשתיות כדוגמת Firewalls ותהליכי אבטחה ארגוניים הגיעו לבשלות מסוימת אשר מאפשרת לארגונים להגן בקלות יחסית על תשתיות המערכת ובשנים האחרונות גם על האפליקציה. לעומת זאת, הארגונים הוסיפו יותר אפליקציות והרחיבו את האפליקציות לקהלי יעד גדולים יותר. האפליקציה הפכה לעקב אכילס של מרבית הארגונים ולמטרה העיקרית של התוקפים.

הפרדה לשכבות

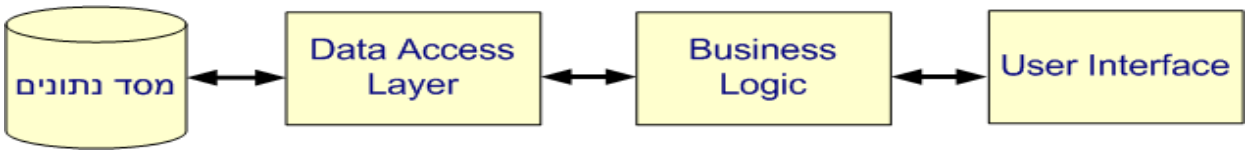
בעת פיתוח אפליקציות יש לחשוב בצורה גנרית ובצורה של ביצוע Reuse לקוד. הקוד צריך להיות כתוב בצורה כזו המאפשר הרחבתו מבלי לגרום למערכת ליפול ו/או שכתוב הקוד מחדש. לכן עוד לפני תחילת הפיתוח, צריך לכתוב מסמכי אפיון מפורטים הן בהיבט הפונקציונאלי של המערכת והן בהיבט אבטחתי. בשלב זה יש לתכנן את ארכיטקטורת המערכת. תכנון נכון של ארכיטקטורת המערכת מהווה את אחד הגורמים היותר חשובים להצלחת הפרויקט כולו.

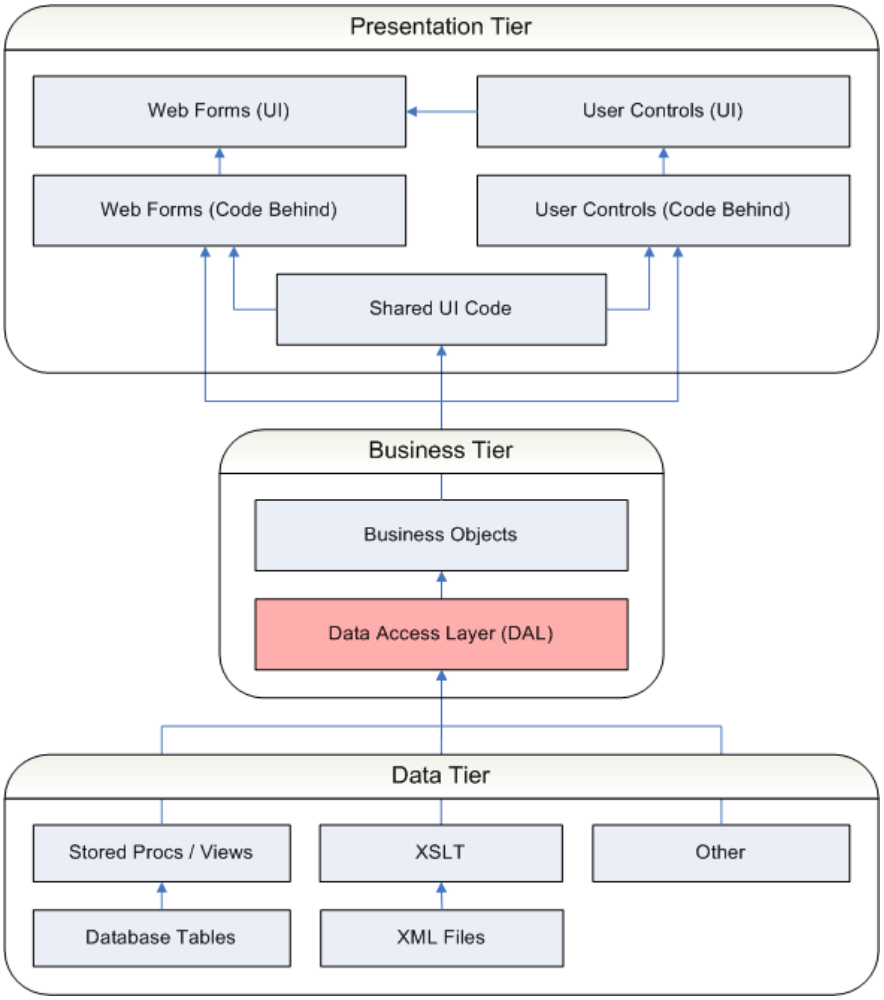
הארכיטקטורה שהוכחה כארכיטקטורה המאובטחת והמומלצת הינה ארכיטקטורה הנקראת ארכיטקטורת שלושת השכבות (3- Tier Architecture) (כמובן הכוונה ללפחות שלוש שכבות עיקריות, כלומר ניתן לבנות גם ארכיטקטורה מסוג N- Tiers, כאשר N גדול מ – 3):

- **שכבת הפרזנטציה** – כוללת את הקוד שמציג את המידע למשתמש. בדרך כלל ממומשת על שרתי Web.מורכבת גם מ- Windows Forms וגם מ- Web Forms.שכבת התצוגה תטפל רק בקלט והפלט מהמשתמש ולא תכיל קטעי קוד ארוכים שאינם מטפלים בממשק המשתמש. בעולם ה-ASP.NET אפשר להשוות רכיבים אלה לקונטרולים (ascx) או דפים (aspx).
- **שכבת הלוגיקה העסקית** – כוללת רכיבים המבטאים את הלוגיקה העסקית של הארגון, שכבה זו ממושת בדרך כלל על שרתי אפליקציה. זוהי השכבה המנהלת את התהליכים העסקיים והלוגיקה, כמו למשל: מנגנוני אבטחה, יצירת אובייקטים, חישובים וכולי...
- **שכבת הנתונים** – כוללת את הנתונים עצמם, שכבה זו היא בדרך כלל הרגישה ביותר ומכילה את כל הנתונים הרגישים של הארגון. שכבה זו ממומשת על שרתי מסד נתונים (Database Servers).
  שכבה זו בעצם מנהלת את ההתקשרות מול בסיסי הנתונים. מבצעת פעולות כמו קריאה, עדכון, מחיקה והוספה לבסיסי הנתונים, יצירה של אובייקטים לוגיים המאופיינים לפי טבלאות נתונים וכו'..

חלוקת היישום לשכבות מאפשרת להגדיר תפקיד לכל חלק ביישום בצורה ברורה יותר.

שינויים המתבצעים בשכבה אינן מחייבים שינויים בשכבות אחרות.





התקפות ואיומים על מרחב האפליקציה יכולות להיות מכוונות לעבר כל אחת מהשכבות המוזכרות:

1. שכבת הפרזנטציה (באמצעות התקפות על שרתי ה-WEB).
2. שכבת הלוגיקה העסקית (באמצעות התקפות המנסות לעקוף הגבלות על ביצוע פעולות ועוד)
3. שכבת הנתונים (על ידי התקפות שמטרתן להוציא מידע בצורה בלתי מורשית מבסיסי הנתונים או לבצע שינויים בנתונים). לכן, יש ליישם את מנגנוני האבטחה בכל השכבות ולשים דגש על פיתוח ותכנון מאובטח של המערכת על מנת לצמצם את הסיכונים הפוטנציאליים על המערכות.

בדיקת תקינות קלטים

3.1.1. כללי

בדיקות על קלט שמתקבל ממשתמשים הוא אחד מהיסודות החשובים ביותר של אבטחת מידע בתחום האפליקטיבי והמקור העיקרי לרבות מבעיות בתחום זה. תוכניתנים רבים מסתמכים על ההנחה כי המשתמש יזין לאפליקציה קלט חוקי בכל עת. משתמש זדוני ינסה להזין קלט לא חוקי – קלט מסוג לא הגיוני אשר לרוב אינו מטופל על ידי מפתחי המערכת. על כן חובה לוודא שהמערכת מסוגלת להתמודד עם כל סוגי הקלט האפשריים.

חובה לבצע בקלט בדיקה חיובית (White list check) – כלומר להרשות רק תווים ומחרוזות שמותר (בניגוד למניעת מעבר של תווים אסורים). בכל מקרה בעת בדיקת הקלט יש לשים לב לדברים הבאים:

- יש לבצע בדיקת תקינות של כל הקלטים לפי קיום, סוג , אורך, טווח ומבנה.
- את כל בדיקות הקלט יש לבצע בצד של השרת אין להסתמך על בדיקות תקינות המבוצעות בצד הלקוח.
- יש לקבל רק תווים מסוג שרלוונטי לאותו שדה.
- יש לבדוק שהמספר המתקבל בקלט הוא בגבול המותר (לדוגמא עבור גיל, הגבלה יכולה להיות מספר גדול מ-0 וקטן מ-200).
- יש לקבל רק מחרוזות המתאימות לתבנית מסוימת (לדוגמא תאריך dd/mm/yyyy).
- יש לקבל רק קלט באורך מוגדר – לא ארוך מדי ולא קצר מדי.
- בכל מקרה יש לוודא שהקלט לא מכיל פקודות SQL (select,update,insert,or,and,;,union וכו) או פקודות HTML המשמשות לCross Site Scripting או ל- Script Injection (document,/,(,),<.> וכו').
    - דוגמאות:

SQL Injection	TABLE
SQL Injection	ANY
SQL Injection	AND
SQL Injection	CMD
SQL Injection	Set-cookie
SQL Injection	INJECTE
SQL Injection	Select
SQL Injection	FROM

SQL Injection	WHERE
SQL Injection	IS NULL
SQL Injection	1=(
SQL Injection	1=1
SQL Injection	LIKE %
SQL Injection	DROP
SQL Injection	INSERT INTO
SQL Injection	VALUES
SQL Injection	UPDATE
SQL Injection	SET
SQL Injection	OR 1=1
SQL Injection	OR '1'='1'
SQL Injection	CHAR(
SQL Injection	LOAD_FILE
SQL Injection	ASCII(
SQL Injection	UNION ALL
SQL Injection	GROUP BY
SQL Injection	HAVING
SQL Injection	CONVERT(
SQL Injection	EXEC
SQL Injection	XP_
SQL Injection	NOT in
SQL Injection	NOT EXIST
SQL Injection	DECLARE
SQL Injection	WAIT FOR DELETE
SQL Injection	COMPRESS(

SQL Injection	ENCODE(
SQL Injection	SCHEMA(

XSS	SRC=
XSS	</TITLE>
XSS	http://
XSS	fromCharCode
XSS	String
XSS	SRC=
XSS	IMG
XSS	javascript
XSS	JaVaScRiPt
XSS	BODY
XSS	onload
XSS	iframe
XSS	INPUT
XSS event handler	FSCommand
XSS event handler	onAbort
XSS event handler	onActivate
XSS event handler	onAfterPrint
XSS event handler	onAfterUpdate
XSS event handler	onBeforeActivate
XSS event handler	onBeforeCopy
XSS event handler	onBeforeCut
XSS event handler	onBeforeDeactivate

XSS event handler	onBeforeEditFocus
XSS event handler	onBeforePaste
XSS event handler	onBeforePrint
XSS event handler	onBeforeUnload
XSS event handler	onBegin
XSS event handler	onBlur
XSS event handler	onBounce
XSS event handler	onCellChange
XSS event handler	onChange
XSS event handler	onClick
XSS event handler	onControlSelect
XSS event handler	onCopy
XSS event handler	onCut
XSS event handler	onDataAvailable
XSS event handler	onDataSetChanged
XSS event handler	onDataSetComplete
XSS event handler	onDblClick
XSS event handler	onDeactivate
XSS event handler	onDrag
XSS event handler	onDragEnd
XSS event handler	onDragLeave
XSS event handler	onDragEnter
XSS event handler	DYNSRC
XSS event handler	LOWSRC
XSS event handler	BGSOUND
XSS event handler	LAYER

XSS event handler	LINK REL
XSS event handler	STYLE
XSS event handler	REL=
XSS event handler	Content>"=
XSS event handler	.htc
XSS event handler	Vbscript
US-ASCII encoding	¼script¾
XSS	META
XSS	FRAMESET
XSS	BACKGROUND
XSS	DIV
XSS	background-image
XSS	expression
XSS	text/javascript
XSS	background:url
XSS	and // BASE HREF
XSS	text/x-scriptlet
XSS	OBJECT
XSS	EMBED
XSS	AllowScriptAcces
XSS	Eval
XSS	.htc
XSS	CDATA
XSS	and # DATASRC

- יש לתעד ניסיונות מובהקים של הכנסת תווים בעייתיים המשמשים בהתקפות SQL Injection או Cross Site Scripting

שם מתקפה	סימנים
SQL Injection	'
SQL Injection	'\\
SQL Injection	;
XSS	//
XSS	';
XSS	;''
XSS	`
XSS	'''''
UTF-8 Unicode XSS	and ; #&
Long UTF-8 Unicode XSS	and numbers #&
XSS Hex encoding	and x #&
Embedded tab to break up string script XSS	\t TAB
Embedded tab to break up string script encoded XSS	#&x09
Embeded newline to break up XSS	#&x0A
Embedded carriage return XSS	#&x0D
XSS	Null
XSS	%00
XSS	\0
meta chars XSS	;14#&
Non-alpha-non-digit XSS	
XSS	>>
XSS with no single quotes or double	/=

quotes or semicolons	
XSS	;''\
US-ASCII encoding	¾
unicoded XSS	and DIV 00\
	newline
	*/
	/*

3.1.2. הנחיות פרטניות לסוגי קלט נפוצים

- **שמות** – ככלל, בעת קליטת שמות, ניתן להגביל את המשתמש לסט התווים הנדרש, הכולל אותיות, רווחים, גרש בודד ומקף. הגרש הבודד נדרש לצורך הזנת שמות הכוללים ג' או צ', ומהווה בעיה בעת עבודה מול שרת SQL. לאור זאת – יש לוודא כי בעת הטיפול בשדות מסוג זה, ננקטים צעדים למניעת SQL Injection כמפורטים לעיל ובנוהל הפיתוח המאובטח לשכבת ה-Database.
- **מספרי חשבון** - מספרי חשבון יכולים לכלול ספרות, מקפים ולוכסן.
- **תאריכים** - השימוש המומלץ בתאריכים הוא על ידי ייצוג התאריך בשלושה שדות נפרדים: יום, חודש ושנה. באופן זה, ניתן לוודא כי כל אחד מהשדות הללו הוא משתנה מספרי פשוט (ובעל טווח חוקי).
- **סכומים** - סכומים יכולים להכיל רק ספרות ונקודה.
- **מנועי חיפוש** - בד"כ ניתן להגביל את התווים המותרים במנוע חיפוש לתווים אלפא נומריים. במרבית המקרים ניתן לבצע את כל החיפושים גם ללא שימוש בתווים מיוחדים.

3.1.3. תווים מיוחדים

תווים מסוימים כדוגמת *$ & ! >* הינם בעלי משמעות מיוחדת במערכות הפעלה מסוג Windows ו-Unix. כך למשל התו > (קטן מ-) מסמן "קרא קלט מתוך קובץ". תווים אלו נקראים במקרים רבים Meta Characters. כאשר אפליקציית ה-Web משתמשת בקלט שהגיע ממשתמשים על מנת לפתוח קבצים, לשלוח דואר אלקטרוני או לפנות למערכת ההפעלה, יכול התוקף לשתול Meta Characters בתוך הקלט ובכך להשפיע על הפעולה שבסופו של דבר תבצע האפליקציה. בצורה זו יכול התוקף לנסות ולצרף פקודות כגון קריאת קבצים או הרצת תוכנות. לכן, יש להימנע משימוש בתווים אלה באם לא נדרשים ולבצע בדיקות תקינות הדוקות במידה ויש צורך לבצע בהם שימוש.

3.1.4. תו ה-NULL

דגש מיוחד יש לשים לתו ה-NULL. הערך ה-ASCII של תו זה הוא 0 והוא משמש בשפות מסוימות כגון C לציין את מיקום סוף המחרוזת. בשפות אחרות כגון Perl אין ל-NULL משמעות שכזו. כאשר התוקף מחדיר NULL בתוך פרמטרים בטופס, הדבר עלול לסיים מחרוזות מוקדם מהמצופה או לעקוף מסננים מסוימים.

3.1.5 שימוש בתשתית קיימת ב NET.

תשתית .NET כוללת מספר אוביקטים המיועדים לבצע בדיקות תקינות קלט בסביבת , אוביקטים אלה מבצעים פעולה כפולה:

1. בדיקת תקינות באמצעות Javascript – פעולה המתבצעת בצד הדפדפן ומטרתה לציג למשתמש הודעת שגיאה במידה והוקש קלט שגוי.
2. בדיקות תקינות המתבצעות בצד השרת – לבדיקות בצד זה יזנ חשיבות רבה בהיבטי אבטחת מידע.

התשתית מספקת פקדים רבים על מנת לתמוך בבדיקות קלטים כגון :

1. RegularExpressionValidator – המאפשרת להצמיד לקלט בדיקת תקינות אל מול ביטוי רגולאר.
2. RangeValidator – מאפשר להגידר לערכים נומריים, טווח מספרים.
3. בנוסף קיים CustomValidator המאפשר להצתמיד לכל שדה קלט את הבדיקות הנדרשות.

דוגמא:

הגדרת הבדיקות עבור שדה הקלט

```
Custom text:<br />

<asp:TextBox runat="server"
id="txtCustom" />

<asp:CustomValidator runat="server"
id="cusCustom"
controltovalidate="txtCustom"
onservervalidate="cusCustom_ServerVa
lidate" errormessage="The text must
be exactly 8 characters long!" />

<br /><br />
```

קטע הקוד המבצע זאת בצד השרת

```
protected void
cusCustom_ServerValidate(object
sender, ServerValidateEventArgs e)

{

    if(e.Value.Length == 8)

        e.IsValid = true;

    else

        e.IsValid = false;

}
```

קריסה מבוקרת

עקרון קריסה מבוקרת (Fail Closed) מתייחס למצב בו לאחר קריסה של מערכת, (עקב סוגים שונים של התקפות כגון גרימת שגיאות מערכת על ידי הזנת קלט זדוני) מנגנוני אבטחת המידע ממשיכים לתפקד ואינם יוצרים מצב של "דלתות פתוחות".

דוגמא לתהליך: אם תהליך ב-Firewall קרס, המכונה צריכה לעבור למצב של חסימת העברת מידע מסוכן בין ממשקים שונים.

דוגמא נוספת – תהליך מערכת, אשר מנסה לבצע התחזות (impersonation) לעבודה בזהות של מבקש השירות ונכשל בתהליך ההתחזות, צריך להפסיק את התהליך ולא להשתמש בזהותו החזקה.

מערכת ששכיחות מספר הקריסות בה גבוה ,תותקף על ידי משתמשים זדונים ובכך יתאפשר להם מצב של פרצות של אבטחת מידע, בזמן שמנגנוני האבטחה אינם פועלים (עקב הקריסה).

חובה לבצע הכנות בקוד לתהליכים של נפילת המערכת ולבחון את השפעת הקוד שנכתב על אבטחת המערכת בעת הנפילה. תכנון נכון של קוד יגרום לכך שהמערכת לא תבצע פעולות שדורשות רמת אבטחה מסוימת במידה ורמת אבטחה זו איננה קיימת במערכת ברגע נתון. יש לפתח את המערכות בצורה מאובטחת ולטפל במקרי קצה כך שבמצב של קריסה המערכת תעבור למצב בטוח/סגור בברירת מחדל.

הדרך הנכונה לטפל בנפילה לא מתוכננת של האפליקציה היא לעטוף כל קטע קוד ב try -ו catch -או מקבליהם בשפה הרלוונטית. הדף המצביע על כשל במערכת, צריך להיות כללי ולא לחשוף למשתמש את פרטי השגיאה. הפרטים המלאים על הנפילה צריכים להירשם במנגנון התיעוד והמעקב של המערכת בצד השרת כפי שמפורט בסעיף 0.

פשטות היישום

כתיבת מנגנוני אבטחת מידע צריכה להיות פשוטה להבנה וליישום.

כתיבת API, מורכב ומסובך לשימוש והבנה ,לצורך אבטחת מידע, היא דוגמא טובה למצב בו כותב ה API מאלץ את המפתח להשתמש בפרוצדורות ארוכות וקשות לצורך כתיבה מאובטחת. דבר זה יגרום למפתח לנסות ליישם מנגנון אבטחתי פרטי, דבר שיגרום לחוסר אחידות כלפי מערכות אחרות בארגון.

פשטות עוזרת להבנת אופן השימוש, איתור בעיות פוטנציאליות, קלות בתחזוקה, מפחיתה מקומות פוטנציאלים לטעות בהם (כל טעות במנגנון אבטחת מידע מובילה לחשיפה פוטנציאלית) וכדומה.

זיהוי משתמשים

זיהוי המשתמשים במערכת הינו אחד ממנגנוני אבטחת המידע החשובים ביותר. בשלב זה המשתמש "מזדהה" בפני המערכת ואימות זיהוי זה יקבע אילו הרשאות פעולה יוענקו למשתמש זה.

חשיפה במנגנון הזיהוי עלולה להוביל לחשיפת מידע פרטי של משתמשים, התחזות למשתמשים על מנת לבצע פעולות בשמם ולבצע התקפות שונות על המערכת בשם משתמש אחר.

3.1.5. <u>מנגנוני זיהוי</u>

מומלץ להשתמש במנגנוני זיהוי תשתיתיים כגון Kerberos. מנגנונים אלו נחשבים מנגנונים חזקים , אשר נבדקו מאות פעמים ונמצאים בשימוש נרחב במערכות שונות בארץ ובעולם.

לעיתים כאשר הזיהוי ברמת התשתית אינו מספק, או אינו גמיש מספיק ונוצר צורך לממש מנגנון זיהוי ייחודי לאפליקציה. מנגנון זה ממומש על ידי המתכנתים ברמת הקוד של האפליקציה. במידה וקיים צורך להצפין את המידע (את המזהים או כל מידע אחר בהודעה), יש להשתמש באלגוריתם הצפנה סטנדרטי וחזק.

3.1.6. <u>מדיניות משתמשים וסיסמאות</u>

במנגנון זיהוי המבוסס על סיסמאות יש ליישם מנגנון המאפשר הגדרה ואכיפה של מדיניות משתמשים וסיסמאות חזקה בהתאם לסוג המערכת. להלן מספר עקרונות בנושא ניהול סיסמאות אותם יש לאכוף בהתאם למדיניות שאופיינה למערכת ע"י גורמי אבטחת המידע:

א. אילוץ אורך סיסמה מינימאלי ומקסימאלי.

בדרך כלל אורך מינמאלי של 6 תווים.

אורך מקסימלי של 20 תווים, במידה ואין צורך עיסקי מעבר לכך.

ב. אכיפת מבנה סיסמה – לדוגמה: מקובל שסיסמת משתמש תכיל לפחות משתיים מתוך 3 קבוצות התווים הבאות:

- אותיות קטנות וגדולות [a-z, A-Z]
- ספרות [0-9]

- תווים מיוחדים [~,!,@,#,$,%,^,...].

ג. החלפת סיסמה - יש לאלץ החלפת סיסמה פעם בתקופה (למשל כל 3 חודשים).

ד. היסטוריית סיסמאות - יש לאכוף אי שימוש בחזרה על סיסמאות שהיו בשימוש בעבר.

בדרך כלל יש למנוע החלפה ל 7 סיסמאות אחרונות

ה. מניעת דימיון סיסמאות - על המערכת לזהות בעת הכנסת הסיסמא החדשה דמיון לסיסמא הישנה. כך ימנע מצב בו משתמש ייצר באופן סדרתי סיסמאות בעלות אותו בסיס, לדוגמא: pass1,pass2,pass3... כך ימנע המצב בו גילוי הסיסמא יעזור לתוקף לחזות את הסיסמאות העתידיות.

ו. נעילת חשבון משתמש –

- יש לנעול את חשבון המשתמש לאחר מספר ניסיונות כושלים בהקשת פרטי ההזדהות לפרק זמן מוגבל. בדרך כלל מדובר על נעילה לאחר כחמישה ניסיונות כושלים.
- יש לשחרר את הנעילה לאחר פרק זמן שהוגדר (למשל לאחר חצי שעה) על מנת לא לגרום למניעת שירות.
- ראה הנחיות נוספות בסעיף 4.1.9

3.1.7. מנגנון שינוי סיסמה

להלן מספר דגשים לגבי יישום של מנגנון שינוי סיסמה:

א. תהליך שינוי הסיסמא צריך תמיד לכלול את השדות הבאים:

- הסיסמה הישנה
- הסיסמה החדשה
- אימות הסיסמה החדשה

ב. יש לאמת בצד השרת את הסיסמה הישנה מול הסיסמה של ה- USER שזה ה- Session שלו.

ג. אין לשלוח את פרטי המשתמש כחלק מהבקשה ולהסתמך עליהם.

ד. יש לדאוג להצפנת תווך התקשורת בו עוברים המזהים (סיסמאות).

ה. יש ליישם את מנגנון הנעילה ושחרור מנעילה גם כאשר המשתמש משנה סיסמה.

3.1.8. מניעת User Enumeration

ברוב האפליקציות יש רק מקום אחד בו מזינים שמות משתמשים – מסך ההזדהות. אך לעיתים הזנת שם משתמש יכולה להיות גם במסכים של שליחת דואר פנימי, מימוש לקוי של מנגנון שינוי סיסמא ועוד. יש לוודא שאין מקום באפליקציה שמגיב בצורה שונה עבור הזנת משתמש קיים, ועבור הזנת משתמש לא קיים.

3.1.9. שימוש במנגנון יציאה מהמערכת

סביר כי משתמש באפליקציה ייצא ממנה באחת משתי הדרכים הבאות:

- סגירה לא מבוקרת - סגירת הדפדפן, סגירת המחשב עצמו וכדומה. מכיוון שאין קשר "חי" בין הלקוח לשרת, לשרת אין דרך לדעת כי המשתמש סיים את עבודתו עם האפליקציה. מסיבה זו מקובל להגדיר, אם יש שימוש באובייקט Session, זמן Timeout. כאשר השרת לא קיבל פנייה בפרק הזמן המוגדר – אובייקט ה-Session יימחק, ותידרש הזדהות חוזרת על מנת להיכנס למערכת.
- יציאה מבוקרת – שימוש במנגנון יציאה מערכת. בזמן יציאה מהמערכת יש לבצע את הפעולות הבאות:
  - תיעוד פרטי הפעולה: מבצע, זמן ופרמטרים נוספים הדרושים במערכת התיעוד (Auditing System).
  - מחיקת אובייקט ה Session בשרת שהוקצה למשתמש.
    לדוגמה במערכת WEB:

```
<% Session.Abandon () %>
```

3.1.6 קיים מנגנון לביטול חשבונות במערכת

במידה ובוצעה תקיפה מוצלחת של המערכת ומנגנוני ההגנה נפרצו, יש לאפשר ביטול/הקפאת חשבונות במערכת אשר חשודים במעורבות במתקפה.קיום מנגנון שכזה יעזור במניעת התקפות נוספות.

3.1.7 אחסון מאובטח של סיסמאות המשתמשים

על סיסמאות המשתמשים להישמר בצורה מוסתרת במערכת, כלומר אין לשמור את המידע בצורת clear text. יש לשמור hash של המידע בלבד, ובעת הבדיקה יש להפעיל על הקלט את אותה פונקצית hash ולהשוות עם מה שנשמר. מומלץ להשתמש בפונקצית hash חזקה כגון SHA256

הרשאות, מידור ובקרת גישה

לאחר תהליך זיהוי המשתמש על ידי המערכת, יש לשייך לו הרשאות לפעולות אשר הוא מורשה לבצע (במערכות מסוימות ישנה אפשרות ואף רצוי להגדיר אילו פעולות אסורות על המשתמש). להלן מספר עקרונות בנשוא הרשאות, מידור ובקרת גישה אשר יש לפעול לפיהן.

3.1.10. עקרון ההרשאה המינימאלית

במתן הרשאות, יש להגדיר את ההרשאה המינימאלית הדרושה על מנת לבצע את הפעולה בחלון זמן נתון.

להלן כמה דוגמאות:

- יש להימנע מהפעלה תחת הרשאות אדמיניסטרטיביות administrator (כללי או מקומי).
- רצוי למנוע גישה לכתיבה ושינוי לאובייקטים – אם אין צורך. לדוגמה: להעניק הרשאות קריאה בלבד למשתמש אשר רק קורא נתונים מקובץ.
- כדאי למנוע גישה גם ברמת הספרייה ולאפשר גישה ישירה רק לספריה המכילה את האובייקטים הדרושים למשתמש.
- אין לאפשר גישה לבסיס הנתונים תחת משתמש sa – במידה ויש צורך ניתן להגדיר משתמש (login) אשר נגיש רק לאובייקטים (טבלאות, פרוצדורות וכדומה) הדרושים לו.
- מומלץ להסיר את ההרשאות עבור Everyone ו-Guest (או Anonymous) מאובייקטים שאינם אומרים להיות נגישים לכל משתמש.

3.1.11. הפרדת תפקידים

בכל אפליקציה יש פעולות ותהליכים רבים. לא כל המשתמשים צריכים לבצע את כל התהליכים. יש לחלק את כל המשתמשים לסוגים ולהתאים לכל סוג רק את ההרשאות שהוא צריך ולא מעבר לכך. לאחר יישום המנגנון יש לוודא שמשתמשים בעלי רמת הרשאות נמוכה אינם יכולים לבצע פעולות הדורשות רמת הרשאות גבוהה יותר.

3.1.12. סוגי הרשאות

3.1.12.1. הרשאות הצהרתיות ברמת התצורה (Declarative Authorization)

מנגנון הרשאות דקלרטיבי מאפשר להגדיר גישה למשאבים רק עבור משתמשים אשר נמצאים ב-Role מסוים, או בקבוצה (Group) שנמצאת ב-Role מסוים. ניהול המשתמשים נעשה במקום אחד מרכזי במערכת (כל סוג של LDAP, קובצי התצורה וכדומה) והינו מתאים להרשאות בין מערכות בעיקר. לדוגמא שני שרתי Web אשר מתקשרים אחד עם השני וכדומה.

3.1.12.2. הרשאות ברמת הקוד (Programmatic Authorization)

ניתן לקבוע מתן הרשאות לצרכנים ברמת הקוד של האפליקציה. גישה זאת בעלת גמישות רבה בזמן כתיבת הקוד, אך מאבדת את יתרון הגמישות בזמן העבודה עם המערכת, משום שלתחזק מנגנון זה, משמעותו לשנות את קוד האפליקציה.

מומלץ להשתמש בגישה זאת רק כאשר מנגנון ההרשאות הדקלרטיבי אינו גמיש מספיק במענה על צרכי האפליקציה, על מנת ליישם מנגנון הרשאות משלים או על מנת למנוע שינויים זדוניים בקבצי התצורה.

3.1.13. ניהול הרשאות

את ניהול ההרשאות באפליקציה ניתן לחלק לשתי רמות עיקריות:

א. הרשאות לפי פעולות.

ב. הרשאות לפי משאבים.

על כל פניה צריכה להיבדק תחילה רמת הרשאות או מעשית,האם המשתמש רשאי לבצע את הפעולה הספציפית. לדוגמא האם מנהל מחלקה במשרד הבריאות רשאי לראות פרטים אישיים של לקוחות. הרמה השנייה היא הרשאות לפי משאבים. איזה משאבים בדיוק רשאי לראות הלקוח? האם מנהל מחלקה רשאי לראות את פרטי כל הלקוחות או רק את פרטי הלקוחות השייכים למחלקה שלו?

3.1.14. חלוקת האפליקציה לאזורי סיווג

אחת הדרכים לצמצם את סיכוני אבטחת מידע באפליקציה היא לחלק את האפליקציה לאזורי סיווג שונים. במבנה זה משתמשים לא מזוהים יכולים לגשת רק לחלק חיצוני של האפליקציה שאינו מכיל לוגיקה של אפליקציה אלא רק פריטי מידע. לאחר זיהוי משתמשים מורשים יכולים לגשת למהות העסקית של האפליקציה. בכך ניתן להקטין את הסיכון שכן מי שניגש ללוגיקה של האפליקציה הם משתמשים מוכרים ומזוהים.

3.1.15. הרשאות ריצת המערכת

האפליקציה בדרך כלל מופעלת כ process או כ service על השרת. התהליך שמייצג את האפליקציה ברמת מערכת הפעלה עם הרשאות ריצה של משתמש מסוים. טעות נפוצה היא להריץ את התהליך עם משתמש בעל הרשאות נרחבות כגון Administrator או System . במצב כזה עם וכאשר ישתלט פורץ על האפליקציה תהיה לו שליטה על כל השרת וגם על אפליקציות אחרות שקיימות באותו שרת. לכן יש להגדיר משתמש מיוחד עבור התהליך ויש לתת למשתמש זה את מינימום ההרשאות הנדרשות. חשוב שעקרון זה יחול על האפליקציה עוד בשלבי הפיתוח והבדיקות, כך יכול להיות מובטח שהאפליקציה תעבוד בצורה טובה.

עקרון דומה חל גם על משתמש בסיס הנתונים בו עושה שימוש האפליקציה. אסור שהאפליקציה תעבוד עם משתמש בעל הרשאות גבוהות בבסיס הנתונים (כגון SA). המערכת צריכה לעבוד עם משתמש ייעודי בעל הרשאות המינימאליות הנדרשות לפעולות מסוימות ולטבלאות מסוימת.

מנגנון התיעוד

3.1.16. כללי

תיעוד ומעקב (Logging) הינו אלמנט חשוב באבטחה של מערכת מידע ויכול לספק מידע חשוב לגבי תהליכים שבוצעו במערכת, שיוך לגורם האחראי על התהליך וטיפול בתוצאות תהליכים אלו (בין אם גרמו לבעיית אבטחה או ניסיון לגרימת נזק ובין אם תהליך בלתי מזיק אשר דורש מעקב) . התיעוד יכול להיעשות בצורה אוטומטית או בצורה ידנית (מופעלת על ידי משתמש ומאפשרת הגדרת תיעוד לפי הגדרתו).

3.1.17. מה לתעד

פעולות שצריך לתעד הן בין השאר אירועי אבטחת מידע שהצליחו וכשלו. לדוגמא: ניסיון כניסה מוצלח, ניסיון לבצע פעולה בלתי חוקית וכדומה. בנוסף,יש לתעד כל פעילות שכשלה.

רצוי להוסיף לכל רשומת תיעוד את זמן הפעולה, מי הגורם שיזם את הפעולה ו/או תחת איזה משתמש/מערכת הפעולה בוצעה (לדוגמה: אפליקציה פועלת תחת הרשאות משתמש מערכת אולם מנהלת באופן עצמאי רשימת משתמשים) וכמו כן תיאור מפורט של האירוע.

פן נוסף הוא השמת ערך "חשיבות" ברשומת האירוע שתועדה. הרשימה הבאה כוללת רשימת האירועים בעלי "חשיבות קריטית".

להן רשימה של אירועים חשובים אשר יש לתעד:

- קריאה וכתיבה של מידע.
- תיעוד של פונקציות, פרמטרים ומשתנים חשובים והערכים שפרמטרים אלו צפויים לקבל.
- כל שינוי מאפיין של אובייקט (הרשאות גישה, מיקום, שם וכדומה).
- מחיקה של אובייקט.
- פעילות תקשורת (פתיחת ערוצים תקשורת, אתחול קשר וכדומה).
- אירועי זיהוי גורמים במערכת (כניסה, יציאה, כישלונות בכניסה).
- אירועי הרשאות (פעולה נכשלה/הצליחה, אובייקט, פונקציה וכדומה).
- אירועים ברמת ניהול מערכת (גישה לחשבונות משתמשים, צפייה בנתוני משתמש וכדומה).

- שגיאות אפליקטיביות במערכת.
- פעילות Debug.
- פעילות "חריגה" אשר חשובה לפעולת המערכת – בהתאם לאופייה.

3.1.18. מה לא לתעד

רשומות לתיעוד, באופן טבעי, מכילות מידע רב ולכן חשוב לוודא כי המידע אינו יכול לגרום לבעיות אבטחה. מומלץ לוודא כי לא מתועדים מזהי גישה וסיסמאות לשרתים השונים במערכת, מזהי מערכות אחרות או כל מידע אחר אשר עלול לחשוף חלקים מהמערכת לגורמים מזיקים.

3.1.19. לוגים המיוצרים ע"י האפליקציה

לכתיבת לוגים יש מספר מטרות. בתפעול רציף של האפליקציה חשוב לקבל חיווי על מצב ריצת הקוד (Debug) התאוששות האפליקציה לאחר נפילה ובחינת אירועים הקשורים לאבטחת מידע. חובה לבצע לוגים במערכת מאחר והם מאפשרים לקבל תמונת מצב על מה התרחש במערכת בסמוך לנקודות קריטיות או לפני שהמערכת נפלה.

חשוב לאבטח את הלוגים של המערכת מפני שינויים לא מבוקרים. משתמש זדוני ינסה לבצע לשנות או למחוק את הלוגים בכדי ל"העלים" את הפשע שביצע, או בכדי למנוע התחקות אחר הנזקים שביצע. ישנן מספר שיטות להגן על הלוגים המפורטת להלן:

3.1.19.1. כתיבת ה- Log למערכת חיצונית

ניתן לכתוב את את נתוני התיעוד (Log הפעילות) למערכת חיצונית באמצעות API. ל API זה יש לאפשר שליחה של הודעות לרישום בלבד ולהסיר את ההרשאות למחוק או לשנות הודעות.

3.1.19.2. כתיבת ה - Log למסד נתונים מוגבל

ניתן לכתוב את נתוני התיעוד (Log הפעילות) לבסיס נתונים שהגישה אליו מותרת למשתמש מסוים שלו הרשאות להוסיף רשומות בלבד ולא לשנות את למחוק רשומות.

3.1.19.3. חתימה על ה- Log

מומלץ לחתום על נתוני התיעוד (Log הפעילות) על מנת למנוע ולאתר שינויים בהם אשר עלולים להתבצע ע"י גורמים זדוניים. כמו כן, מומלץ כי המפתח לפתיחת Log הפעילות לא יהיה מצוי במערכת שכותבת את נתוני התיעוד. הדבר איננו מבטיח שנתוני התיעוד לא ישונו או ימחקו, אך מבטיח שהם נכתבו על ידי המערכת ולא ידי גורם אחר.

3.1.19.4. הנחיות נוספות לשימוש נכון בלוג

נוהל פיתוח מערכות מאובטחות

- ✓ **יש לתכנן את רמת פירוט התיעוד**

יש לבצע תכנון מקדים לגבי הפרטים אותם יש לשמור. רצוי לשמור פרטים חשובים כגון כניסה ויציאה מהמערכת, שם המשתמש, שעה, (עבור מערכות WEB יש לתעד גם את כתובת ה – IP) , שם המשאב אליו ניגש וכדומה.כמו כן, יש לדאוג לאבטח קובץ זה, בשל העובדה שהוא מכיל נתונים רגישים.

- ✓ **יש לאפיין את הפרמטרים והערכים אותם יש לתעד**

לאחר הגדרת רמת התיעוד, יש לבצע מיפוי ברמה אפליקטיבית, כלומר למפות את המשתנים והפרמטרים בקוד המכילים את הפרטים הרצויים.

- ✓ **רצוי לשמור את הקבצים על שרת נפרד**

על מנת להתמודד עם אפשרות בה השרת המריץ את האפליקציה נפרץ, וקבצי הלוג גלויים וניתנים לשינוי, יש לכתוב את המידע לשרת נפרד. במצב זה, הפורץ לא יוכל למחוק רשומות אשר נכתבו לשרת המחזיק את קובץ ה- Log אשר ימסרו מידע על התנהגות המערכת בזמן הפריצה, מי היה המשתמש וכו׳.

- ✓ **יש לשמור חיווי על ביצוע כניסות ויציאות של משתמשים**

מידע שכזה ייתן אינפורמציה לגבי מי היה המשתמש החשוד כמעורב, במידה והמערכת נפרצה. ניתן יהיה להצליב מידע זה עם נתונים נוספים לקבלת תמונה

ברורה יותר. כמו כן, התבוננות בכניסות ויציאות המשתמשים תעזור בחשיפת פעילות חשודה במערכת.

- ✓ **יש לשמור חיווי על ביצוע פעולות מערכת רגישות**

בדומה לסעיף קודם. מידע שכזה יעזור בחשיפת ליקויים קיימים במערכת, כגון ליקויים ברמת מערכת ההפעלה לדוגמא.

- ✓ **יש לקיים מנגנון נוח לניתוח והצגת המידע**

בשל העובדה כי המערכת שומרת אלפי רשומות, קובץ ה-Log עלול לגדול להכיל רשומות פחות רלוונטיות מאחרות. לצורך צפייה בנתונים, קיים הצורך במנגנון נוח לצפייה בנתונים, ביצוע חתכים ופילטרים, התאמות למידע, וכדומה. בהעדר מנגנון שכזה, מידע חשוב עלול להיות קבור בין אלפי

רשומות אחרות, בלתי ניתן לזיהוי. בעזרת מנגנון שכזה, ניתן יהיה לבצע צמצום של כמות המידע הנדרש לבדיקה, דבר אשר יאפשר להתבונן במידע החשוב יותר בקלות יתר ולא ייתן למידע חשוב "ללכת לאיבוד".

- ✓ **יש להקפיד כי מידע רגיש אינו נרשם ב-Log המערכת**

ניהול log הינו צעד חשוב ביותר, אך יש לבצע כתיבה של נתונים שאינם רגישים בלבד. כתיבת נתונים רגישים ל-Log המערכת, כגון סיסמאות משתמשים וכדומה תוריד את רמת אבטחת המערכת.

- ✓ **יש להקפיד כי קובץ ה-Log מאובטח ואינו מכיל מידע רגיש**

יש לוודא כי הקובץ מוגן מפני גישה בלתי מורשית, מהסיבה שהוא מכיל פרטים רגישים על פרטי המערכת כמתואר מלעיל. כמו כן, אסור שהקובץ יכיל פרטים רגישים כגון שמות או סיסמאות משתמשים וכדומה. יש לאבטח את הקובץ ברמת מערכת ההפעלה, ואף להצפינו.

- ✓ **יש להקפיד כי קובץ ה-Log מגובה אחת לתקופה מוסכמת מראש**

יש לוודא כי מתבצע גיבוי לקובצי ה-Log. מידע היושב בקבצי ה-Log הינו בעל חשיבות גבוהה ויש להתייחס אליו כמו אל כל מידע אחר של האפליקציה ולגבותו

בהתאם. כמו כן, גיבוי ושמירת קבצי ה-Log ייתן מענה לחקירת אירועים אשר קרו בעבר.

- ✓ **כל גישה לקובץ ה- log צריכה אף היא להופיע ב- log עצמו.**

3.1.19.5. <u>שימוש בתשתית קיימת לרישום ללוג</u>

תשתית . Net מכילה ספריה בשם Log4Net, המספקת כלי יעיל למתכנתים לבצע רישום ללוג למגון מקומות. Log4Net מכילה שלושה חלקים עיקריים :

1. קונפיגורציה
2. התקנה
3. הפעלה / קריאה.

1. OFF - nothing gets logged (cannot be called)
2. FATAL
3. ERROR
4. WARN
5. INFO
6. DEBUG
7. ALL - everything gets logged (cannot be called)

http://logging.apache.org/log4net/

מנגנון טיפול בשגיאות בלתי צפויות

3.1.20. כללי

אחד השלבים הראשונים של פורץ בהתקפת מערכת, הוא שלב איסוף המידע. אחת הדרכים של הפורץ לאסוף מידע פנימי על המערכת, חורי א. מידע פוטנציאלים וללמוד על יציבותה, היא על ידי גרימת שגיאות בלתי צפויות למערכת.

כל מערכת צפויה להתמודד עם בעיות ושגיאות בלתי צפויות. חלק מהבעיות הינן בעיה באפליקציה עצמה (bugs) וחלקן יכולות להיות כחלק מניסיון לפרוץ את מנגנוני האבטחה של האפליקציה. בכל מקרה, לא מומלץ להשאיר שגיאה בלתי מטופלת ויש לתקן ו/או "להגיב" לה במהירות רבה ככל האפשר, כמו כן, במידה וזאת שגיאת אבטחת מידע יש לסגור את ה Session של המשתמש.

3.1.21. תיעוד

בפרק הקודם דנו בתיעוד אירועים במערכת ובין היתר צוין כי רצוי לתעד שגיאות בלתי צפויות. במקרים בהם נזרקת שגיאה במערכת יש לתעד, בנוסף לתיעוד הרגיל, את השדות הבאים:

- קוד שגיאה-מספר בעל ערך חד-חד ערכי המתאר את סוג השגיאה שארעה. לדוגמא:
  - 1- המשתמש אינו קיים במערכת
  - 2 - סיסמא לא נכונה
  - 3 – חשבון המשתמש נעול
  - 4 – משתנה מסוג לא מתאים
  - 5 – לא התקבל ערך (או – התקבל NULL)
  - 6 – התקבל ערך החורג מהטווח המוגדר

- Session – 7 המשתמש אינו תקף

- פירוט נתיב מלא במחסנית (stack trace) של השגיאה.
- במידה וזוהתה בעיית אבטחה רצוי גם להוסיף נתוני משתמש (מספר מזהה, הרשאות וכדומה).

3.1.22. הודעות שגיאה

מומלץ לוודא כי כל שגיאה או אפשרות לשגיאה מטופלת בקוד האפליקציה. אולם במידה ושגיאה הצליחה "לחמוק" ויש להציג הודעה למשתמש יש להציג תמיד הודעת שגיאה כללית ("אירעה שגיאת מערכת"). במקרים רבים מפתחים מציגים את כל הודעה השגיאה הכוללת שורות בקוד, מחסנית פונקציות (stack trace) ועוד – מידע שיכול לשמש גורמים זדוניים בהתקפות עתידיות על המערכת.

דוגמא לשימוש לא מאובטח:

```
try{
:
}catch(Exception e) { }
```

דוגמא לשימוש מאובטח:

```
try{
:
}catch(Exception e){
   log.Fatal(e.ToString());
    Console.WriteLine("{0}Fatal Error", e);
  Session.abandon ();
  }
```

עבודה עם בסיס נתונים

3.1.23. כללי

כמעט בכל מערכת נעשה שימוש זה או אחר בבסיסי נתונים. לרוב בסיסי הנתונים יש מנגנוני אבטחת מידע פנימיים אשר ניתן להשתמש בהם, אך גם הגישה עצמה מקוד המקור חשובה.

3.1.24. גישה לבסיס הנתונים

בהתאם להנחיות ולעקרונות של "הרשאות מינימאליות" יש להעניק למשתמש המתחבר לבסיס הנתונים את ההרשאות המינימאליות הדרושות לו לביצוע הפעולות שלו. מפתחים רבים מבצעים לפחות אחת מהטעויות הבאות:

- מאפשרים גישה לבסיס הנתונים תחת משתמש חזק - sa (מנהל בסיס הנתונים).
- סיסמת המשתמש sa נותרת ריקה.
- הרשאות המשתמש זהות למשתמש sa.
- מחרוזת החיבור לבסיס הנתונים (Connection String) נשמרת במקום לא מאובטח ובצורה לא מוצפנת

3.1.25. הרשאות משתמש

הרשאות המשתמש של בסיס הנתונים צריכה להיות מוגבלת. יש לזכור כי משתמש sa ו/או בעל הרשאות חזקות דומות יכול למחוק טבלאות, מידע, משתמשים אחרים ולגרום נזק רב מאוד. בזמן הגדרת משתמש של בסיס הנתונים יש להקפיד על הכללים הבאים (שוב, עקרון ההרשאות המינימאליות הדרושות):

- אין להשתמש בהרשאות החזקות ביותר כגון sa.
- יש להגביל את המשתמש לטבלאות האפליקטיביות בלבד ועדיף לאפשר גישה אליהם רק באמצעות פרוצדורות שמורות.
- יש להגביל את הרשאות המשתמש לקריאה וכתיבה בלבד בטבלאות האפליקציה.
- יש להגביל את המשתמש בגישה לטבלאות system (בברירת מחדל).

3.1.26. שמירת נתונים מזהים בבסיס הנתונים

בסיס הנתונים מאחסן בתוכו את רוב או אף את כל המידע של המערכת. בין הנתונים המאוחסנים נמצאים גם נתונים "רגישים" כגון נתוני משתמש (שם, סיסמה, דוא"ל, כרטיסי אשראי , פרטים רפואים וכדומה), גישה לשרתים אחרים, הרשאות אפליקטיביות ועוד.

בעוד שרוב הנתונים נשמרים בטבלאות ללא הצפנה, יש להצפין את הנתונים הרגישים באמצעות שיטות הצפנה סטנדרטיות ומקובלות בתעשייה על מנת לוודא כי חשיפת בסיס הנתונים לא תחשוף את נתוני המשתמשים.

3.1.27. שימוש בשאילתה פרמטרית כתחליף לשאילתות דינאמיות

שימוש בשאילתה דינאמית (שאילתה המורכבת ממחרוזות בקוד האפליקציה) הינו מסוכן וחושף את האפליקציה להתקפות אבטחה רבות ומגוונות. מומלץ מאוד להשתמש בפרוצדורות שמורות (Stored Procedures) שהם קוד SQL שנכתב מראש ומקבל פרמטרים מובנים שפחות חשופים למניפולציות על מחרוזות.

בשאילתה פרמטרית בונים תחילה את השאילתה כמחרוזת ומצהירים על הפרמטרים שלה. השאילתה עוברת תהליך Parsing. תהליך זה קובע את מבנה השאילתה. לאחר מכן מציבים את הקלט מן המשתמש בפרמטרים ומפעילים את השאילתה (Execute). בשלב זה הפרמטרים יכולים להשפיע רק על התוצאה ואינם יכולים להשפיע על המבנה.

דוגמא שלילית לשאילתה דינאמית:

```
:
strQuery = "SELECT user, pass FROM t_users WHERE user ='" + userName + "'
AND pass ='" + userPassword + "';";
:
```

דוגמא לשימוש ב-Stored Procedure:

```
DECLARE @userID int,
SELECT * FROM DynUser WHERE ID = @userID
```

3.1.28. ADO Commands

באופן דומה לאמור לעיל, ניתן לממש שאילתות פרמטריות תוך שימוש ב- ADO Commands (ולא ADO Recordsets עבור הרצת String Queries, שכן זה פגיע כמו כל String Query אחר).

שימוש בשאליתות פרמטריות מאפשר להעביר פרמטרים באופן מובנה לשאילתות המורצות וכתוצאה מכך נימנעת האפשרות לשנות את מבנה השאילתא מול בסיס הנתונים.

3.1.29. Stored Procedures

שימוש ב-Stored Procedures, בדומה לשאילתות פרמטריות, איננו מאפשר לתוקף לשנות את מבנה השאילתא אלא רק אזורים מוגדרים מראש לפי קביעת כותב האפליקציה וזו הדרך העדיפה לבצע גישה והרצה של שאילתות מול בסיס הנתונים.

4.דגשים לפיתוח מאובטח בסביבת WEB

סניטציה של קלט

סניטציה על הקלט הינה הפעולה בה הופכים קלט אשר עלול להיות מזיק לבטוח.

ניתן לבצע זאת תוך שימוש באובייקטים הבאים:

- HtmlEncode – עוטף את המידע לצורך התייחסות כמידע לתצוגה בלבד. לא ניתן להריץ מידע זה.

דוגמא:

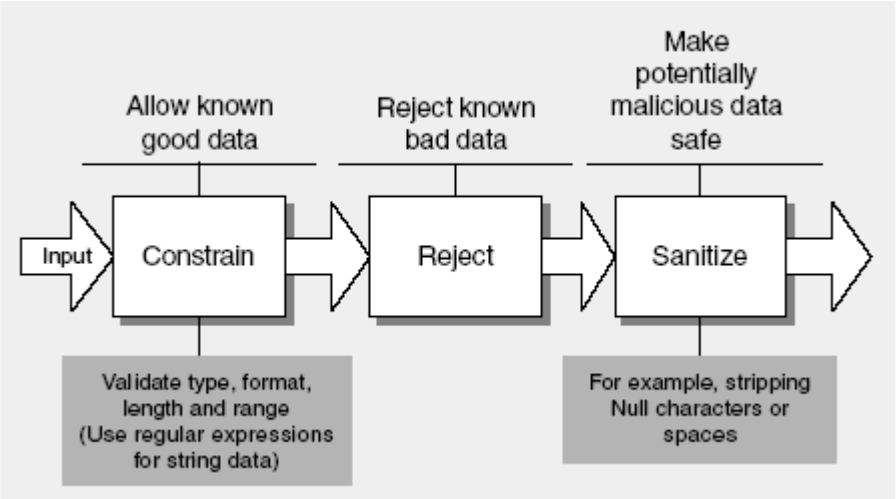
```
String TestString = "This is a <Test String>.";
String EncodedString = Server.HtmlEncode(TestString);
```

- UrlEncode – קידוד ה-URL כך שהוא ייהפך לבקשת URL תקינה מבחינה אבטחתית ולא תהווה סיכון.

דוגמא:

```
String MyURL;
MyURL = "http://www.contoso.com/articles.aspx?title=" + Server.UrlEncode("ASP.NET
Examples");
Response.Write("<a href=" + MyURL + "> ASP.NET Examples </a>");
```

להלן תרשים התהליך:



הצפנה

יש להצפין כל מידע רגיש במערכת (סיסמאות, מספרי כרטיס אשראי וכדומה) בשמירתו ובשליחתו. אנו מצפינים מידע רגיש במאגרי נתונים על מנת שבמידה ומאגר הנתונים נחשף לא ידלוף מידע פרטי רגיש של משתמשים, ומצפינים את המידע בתעבורת הרשת על מנת שלא ייפול לידי פורץ המאזין לתווך התקשורת.

- ✓ **יש להקפיד כי נעשה שימוש במנגנון הצפנה מוכר:**

על המפתח לעשות שימוש במנגנוני הצפנה מוכרים ובדוקים ולא לנסות להמציא את הגלגל ע"י יצירת מנגנונים חדשים. על מנת לפתח מנגנוני הצפנה על המפתח להיות בקיא ביותר בתחום ומאחר ורוב המפתחים אינם כאלה עלול להיווצר חור אבטחה דרך מנגנון אבטחה שלא מומש כראוי. לכן יש להשתמש באלגוריתמים ידועים.

- ✓ **יש להקפיד כי קיימת התאמה בין מנגנון ההצפנה לדרישות אבטחת המידע:**

יש לשים לב לדרישות ההצפנה של הארגון ולרמת הרגישות של המידע המוצפן. בשלב הראשון יש לבדוק איזה שיטת הצפנה דרושה, סימטרית או א-סימטרית. לאחר מכן, יש לוודא כי אורך המפתח תואם את רגישות המידע. לדוגמא, עבור מידע רגיש יש להשתמש ב-Triple Des שהוא בעל אורך מפתח של 168 ביט (ולא ב-DES שאורך מפתח שלו הוא רק 56 ביט), AES, או במקרה הצורך הצפנה א-סימטרית כגון RSA.

- ✓ **יש להקפיד על תכנון אופן שמירת המפתחות והגנת הגישה אליהם:**

על מנת לפענח מידע אשר הוצפן בעבר, יש לספק למנגנון ההצפנה מפתח לפתיחה. על מפתח זה להיות מוגן, ובשום מקרה לא להיות HardCoded בקוד המקור, בשל העובדה כי מתן נגישות למפתח שקולה למחסור בהצפנה ותאפשר פענוח של המידע המוצפן. לכן, יש לתכנן את מיקום שמירת המפתח. יש להסתיר מפתח זה מפני משתמשים שאינם מורשים ואולי אף להגן עליו ברמת מערכת ההפעלה. פתרון אפשרי הוא להסתירו ב-Registry של המערכת ולהגביל את הגישה אליו.

4.1.0 הצפנה בתשתית . Net שימוש בספריה System.Security.Cryptography

להלן דוגמא לביצוע הצפנה ופיענול באלגוריתם Rijndaelב c#

```
SAMPLE: Symmetric key encryption and decryption using Rijndael algorithm.
//
// To run this sample, create a new Visual C# project using the Console
// Application template and replace the contents of the Class1.cs file with
// the code below.
//
```

```
// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND,
// EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED
// WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE.
//
// Copyright (C) 2002 Obviex(TM). All rights reserved.
//
using System;
using System.IO;
using System.Text;
using System.Security.Cryptography;

/// <summary>
/// This class uses a symmetric key algorithm (Rijndael/AES) to encrypt and
/// decrypt data. As long as encryption and decryption routines use the same
/// parameters to generate the keys, the keys are guaranteed to be the same.
/// The class uses static functions with duplicate code to make it easier to
/// demonstrate encryption and decryption logic. In a real-life application,
/// this may not be the most efficient way of handling encryption, so - as
/// soon as you feel comfortable with it - you may want to redesign this class.
/// </summary>
public class RijndaelSimple
{
    /// <summary>
    /// Encrypts specified plaintext using Rijndael symmetric key algorithm
    /// and returns a base64-encoded result.
    /// </summary>
    /// <param name="plainText">
    /// Plaintext value to be encrypted.
    /// </param>
    /// <param name="passPhrase">
    /// Passphrase from which a pseudo-random password will be derived. The
    /// derived password will be used to generate the encryption key.
    /// Passphrase can be any string. In this example we assume that this
    /// passphrase is an ASCII string.
    /// </param>
    /// <param name="saltValue">
    /// Salt value used along with passphrase to generate password. Salt can
    /// be any string. In this example we assume that salt is an ASCII string.
    /// </param>
    /// <param name="hashAlgorithm">
    /// Hash algorithm used to generate password. Allowed values are: "MD5" and
    /// "SHA1". SHA1 hashes are a bit slower, but more secure than MD5 hashes.
    /// </param>
    /// <param name="passwordIterations">
    /// Number of iterations used to generate password. One or two iterations
    /// should be enough.
    /// </param>
    /// <param name="initVector">
    /// Initialization vector (or IV). This value is required to encrypt the
    /// first block of plaintext data. For RijndaelManaged class IV must be
    /// exactly 16 ASCII characters long.
    /// </param>
    /// <param name="keySize">
    /// Size of encryption key in bits. Allowed values are: 128, 192, and 256.
    /// Longer keys are more secure than shorter keys.
    /// </param>
```

```
    /// <returns>
    /// Encrypted value formatted as a base64-encoded string.
    /// </returns>
    public static string Encrypt(string   plainText,
                                 string   passPhrase,
                                 string   saltValue,
                                 string   hashAlgorithm,
                                 int      passwordIterations,
                                 string   initVector,
                                 int      keySize)
    {
        // Convert strings into byte arrays.
        // Let us assume that strings only contain ASCII codes.
        // If strings include Unicode characters, use Unicode, UTF7, or UTF8
        // encoding.
        byte[] initVectorBytes = Encoding.ASCII.GetBytes(initVector);
        byte[] saltValueBytes  = Encoding.ASCII.GetBytes(saltValue);

        // Convert our plaintext into a byte array.
        // Let us assume that plaintext contains UTF8-encoded characters.
        byte[] plainTextBytes  = Encoding.UTF8.GetBytes(plainText);

        // First, we must create a password, from which the key will be derived.
        // This password will be generated from the specified passphrase and
        // salt value. The password will be created using the specified hash
        // algorithm. Password creation can be done in several iterations.
        PasswordDeriveBytes password = new PasswordDeriveBytes(
                                                        passPhrase,
                                                        saltValueBytes,
                                                        hashAlgorithm,
                                                        passwordIterations);

        // Use the password to generate pseudo-random bytes for the encryption
        // key. Specify the size of the key in bytes (instead of bits).
        byte[] keyBytes = password.GetBytes(keySize / 8);

        // Create uninitialized Rijndael encryption object.
        RijndaelManaged symmetricKey = new RijndaelManaged();

        // It is reasonable to set encryption mode to Cipher Block Chaining
        // (CBC). Use default options for other symmetric key parameters.
        symmetricKey.Mode = CipherMode.CBC;

        // Generate encryptor from the existing key bytes and initialization
        // vector. Key size will be defined based on the number of the key
        // bytes.
        ICryptoTransform encryptor = symmetricKey.CreateEncryptor(
                                                         keyBytes,
                                                         initVectorBytes);

        // Define memory stream which will be used to hold encrypted data.
        MemoryStream memoryStream = new MemoryStream();

        // Define cryptographic stream (always use Write mode for encryption).
        CryptoStream cryptoStream = new CryptoStream(memoryStream,
                                                     encryptor,
```

```
                                                    CryptoStreamMode.Write);
        // Start encrypting.
        cryptoStream.Write(plainTextBytes, 0, plainTextBytes.Length);

        // Finish encrypting.
        cryptoStream.FlushFinalBlock();

        // Convert our encrypted data from a memory stream into a byte array.
        byte[] cipherTextBytes = memoryStream.ToArray();

        // Close both streams.
        memoryStream.Close();
        cryptoStream.Close();

        // Convert encrypted data into a base64-encoded string.
        string cipherText = Convert.ToBase64String(cipherTextBytes);

        // Return encrypted string.
        return cipherText;
    }

    /// <summary>
    /// Decrypts specified ciphertext using Rijndael symmetric key algorithm.
    /// </summary>
    /// <param name="cipherText">
    /// Base64-formatted ciphertext value.
    /// </param>
    /// <param name="passPhrase">
    /// Passphrase from which a pseudo-random password will be derived. The
    /// derived password will be used to generate the encryption key.
    /// Passphrase can be any string. In this example we assume that this
    /// passphrase is an ASCII string.
    /// </param>
    /// <param name="saltValue">
    /// Salt value used along with passphrase to generate password. Salt can
    /// be any string. In this example we assume that salt is an ASCII string.
    /// </param>
    /// <param name="hashAlgorithm">
    /// Hash algorithm used to generate password. Allowed values are: "MD5" and
    /// "SHA1". SHA1 hashes are a bit slower, but more secure than MD5 hashes.
    /// </param>
    /// <param name="passwordIterations">
    /// Number of iterations used to generate password. One or two iterations
    /// should be enough.
    /// </param>
    /// <param name="initVector">
    /// Initialization vector (or IV). This value is required to encrypt the
    /// first block of plaintext data. For RijndaelManaged class IV must be
    /// exactly 16 ASCII characters long.
    /// </param>
    /// <param name="keySize">
    /// Size of encryption key in bits. Allowed values are: 128, 192, and 256.
    /// Longer keys are more secure than shorter keys.
    /// </param>
    /// <returns>
    /// Decrypted string value.
```

```
    /// </returns>
    /// <remarks>
    /// Most of the logic in this function is similar to the Encrypt
    /// logic. In order for decryption to work, all parameters of this function
    /// - except cipherText value - must match the corresponding parameters of
    /// the Encrypt function which was called to generate the
    /// ciphertext.
    /// </remarks>
    public static string Decrypt(string   cipherText,
                                 string   passPhrase,
                                 string   saltValue,
                                 string   hashAlgorithm,
                                 int      passwordIterations,
                                 string   initVector,
                                 int      keySize)
    {
        // Convert strings defining encryption key characteristics into byte
        // arrays. Let us assume that strings only contain ASCII codes.
        // If strings include Unicode characters, use Unicode, UTF7, or UTF8
        // encoding.
        byte[] initVectorBytes = Encoding.ASCII.GetBytes(initVector);
        byte[] saltValueBytes  = Encoding.ASCII.GetBytes(saltValue);

        // Convert our ciphertext into a byte array.
        byte[] cipherTextBytes = Convert.FromBase64String(cipherText);

        // First, we must create a password, from which the key will be
        // derived. This password will be generated from the specified
        // passphrase and salt value. The password will be created using
        // the specified hash algorithm. Password creation can be done in
        // several iterations.
        PasswordDeriveBytes password = new PasswordDeriveBytes(
                                                        passPhrase,
                                                        saltValueBytes,
                                                        hashAlgorithm,
                                                        passwordIterations);

        // Use the password to generate pseudo-random bytes for the encryption
        // key. Specify the size of the key in bytes (instead of bits).
        byte[] keyBytes = password.GetBytes(keySize / 8);

        // Create uninitialized Rijndael encryption object.
        RijndaelManaged    symmetricKey = new RijndaelManaged();

        // It is reasonable to set encryption mode to Cipher Block Chaining
        // (CBC). Use default options for other symmetric key parameters.
        symmetricKey.Mode = CipherMode.CBC;

        // Generate decryptor from the existing key bytes and initialization
        // vector. Key size will be defined based on the number of the key
        // bytes.
        ICryptoTransform decryptor = symmetricKey.CreateDecryptor(
                                                         keyBytes,
                                                         initVectorBytes);

        // Define memory stream which will be used to hold encrypted data.
```

```
        MemoryStream  memoryStream = new MemoryStream(cipherTextBytes);

        // Define cryptographic stream (always use Read mode for encryption).
        CryptoStream  cryptoStream = new CryptoStream(memoryStream,
                                                      decryptor,
                                                      CryptoStreamMode.Read);

        // Since at this point we don't know what the size of decrypted data
        // will be, allocate the buffer long enough to hold ciphertext;
        // plaintext is never longer than ciphertext.
        byte[] plainTextBytes = new byte[cipherTextBytes.Length];

        // Start decrypting.
        int decryptedByteCount = cryptoStream.Read(plainTextBytes,
                                                   0,
                                                   plainTextBytes.Length);

        // Close both streams.
        memoryStream.Close();
        cryptoStream.Close();

        // Convert decrypted data into a string.
        // Let us assume that the original plaintext string was UTF8-encoded.
        string plainText = Encoding.UTF8.GetString(plainTextBytes,
                                                   0,
                                                   decryptedByteCount);

        // Return decrypted string.
        return plainText;
    }
}

/// <summary>
/// Illustrates the use of RijndaelSimple class to encrypt and decrypt data.
/// </summary>
public class RijndaelSimpleTest
{
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
    {
        string   plainText          = "Hello, World!";    // original plaintext

        string   passPhrase         = "Pas5pr@se";        // can be any string
        string   saltValue          = "s@1tValue";        // can be any string
        string   hashAlgorithm      = "SHA1";             // can be "MD5"
        int      passwordIterations = 2;                  // can be any number
        string   initVector         = "@1B2c3D4e5F6g7H8"; // must be 16 bytes
        int      keySize            = 256;                // can be 192 or 128

        Console.WriteLine(String.Format("Plaintext : {0}", plainText));

        string  cipherText = RijndaelSimple.Encrypt(plainText,
                                                    passPhrase,
```

```
                                                saltValue,
                                                hashAlgorithm,
                                                passwordIterations,
                                                initVector,
                                                keySize);

        Console.WriteLine(String.Format("Encrypted : {0}", cipherText));

        plainText           = RijndaelSimple.Decrypt(cipherText,
                                                passPhrase,
                                                saltValue,
                                                hashAlgorithm,
                                                passwordIterations,
                                                initVector,
                                                keySize);

        Console.WriteLine(String.Format("Decrypted : {0}", plainText));
    }
}
//
// END OF FILE
```

4.1.1. הצפנת תקשורת לשרתי ה- Web

חשוב לדעת כי השימוש בפרוטוקול SSL איננו מבטיח כי האפליקציה בטוחה. טעות נפוצה היא הצהרת אתרי Web שונים כי המערכת מאובטחת כי יש בה SSL ו-Firewall. אלו הם מרכיבים חשובים באבטחת התקשורת, אך לא מספקת בהיבט הכולל של אבטחת האפליקציה.

פרוטוקול ה-SSL מאפשר למנוע האזנה או שינוי של התשדורת בין המחשב של המשתמש לשרת ה-Web. הפרוטוקול מבטיח למשתמש כי הוא יוצר תשדורת עם השרת הנכון ולא עם שרת מתחזה ומבטיח כי המידע הרגיש אשר הוא מעביר לאתר עובר בצורה מאובטחת ממנו לאתר ולהיפך. הפרוטוקול איננו מבטיח כי המערכת תטפל בנתונים בצורה מאובטחת או לחלופין כי לא תתאפשר לתוקפים להוציא ו/או לשנות נתונים ללא הרשאה מן המערכת.

קיימות שתי אפשרויות למימוש SSL:

- אפשרות ראשונה היא שהשרת מציג למשתמש תעודה דיגיטאלית אמיתית (ומאומתת על ידי CA מוכר), כך שלמשתמש ביטחון כי הוא עובד מול אתר מאומת. בנוסף לכך קיימת הצפנה של התווך בין המשתמש לשרת ולהיפך.
- אפשרות שנייה היא כי הזיהוי של המשתמש, כמו גם של השרת נעשה על ידי תשתית תעודות דיגיטאליות (PKI).

בחינת הביצועים ורמת ההצפנה

- לשימוש ב-SSL יש השפעות ביצועים על האפליקציה והתקשורת וחשוב לבחון זאת לפני השימוש. בכל מקרה חשוב להשתמש ב-SSL בכל פעם שמתבצעת העברה של מידע פרטי רגיש בין המשתמש למערכת ולהיפך.
- יש לבדוק כי התעודה בה משתמש שרת ה-Web היא חוקית, תקינה ובתוקף.
- מומלץ להשתמש במפתח לפחות בגודל 256 ביט.
- יש לבדוק כי אורך המפתח בו משתמשים הוא המקסימאלי בזמן כתיבת המערכת.
- יש להקפיד להשתמש בפרוטוקולי SSLv3 או TLS לצורך הצפנה יעילה,

מידע שאין להסתמך עליו

4.1.2. שדות מסוג Hidden בטפסים

אין להסתמך על שדות Hidden בשום מקרה ואין לאחסן שם מידע שמשתמשים לא אמורים לראות או לשנות. מידע על מצב המערכת צריך להישמר בשרת בקבצים או בבסיס הנתונים כאשר הקשר בין המידע לבין המשתמש מתבצע על ידי Session Cookie שמגיע מהמשתמש.

4.1.3. שדה ה-Referrer

אחד מהשדות המצויים בפרוטוקול ה-HTTP הוא שדה ה-Referrer. שדה זה נשתל על ידי הדפדפן ונשלח לשרת ה-Web עם כל בקשה. השדה מציין את הדף (ה-URL) האחרון בו ביקר המשתמש לפני שהוא שלח את הבקשה. אין להשתמש בשדה ה referrer לצרכי הזדהות או הרשאות מפני שהוא ניתן לשינוי וזיוף בקלות.

4.1.4. שדות מסוג Password בטפסים

כאשר המשתמש מקיש את הסיסמא, הסיסמא תוצג על המסך ככוכביות(masking). כל תקשורת בה מועברים מזהים וסיסמאות צריכה להתבצע בתווך מוצפן (שימוש ב SSL).

4.1.5. כמות וסוג המידע שנקבל ממשתמשים

מהרגע שהמשתמש קיבל לידיו את הטופס הוא יכול לבצע בו ככל העולה על רוחו – להוריד ולהוסיף שדות ולשנות את סוגם של השדות. על כן, אין להסתמך אף פעם על מידע המגיע מצד המשתמש.

4.1.6. השימוש ב-Size בשדות מסוג Input

אין להסתמך שיטה זו (size = x) בלבד לצורך הגבלת הקלט.

4.1.7. בדיקות בצד הדפדפן

נהוג לכתוב בצד הדפדפן קוד JavaScript או VBScript אשר מבצע בדיקות על הקלט שהמשתמש הכניס לפני שליחתו לשרת. הדבר שכיח מאחר והוא מוריד את העומס מהשרת ונוח ללקוח מפני שהוא חוסך זמן יקר בו הטופס נשלח לשרת, מתבצעות הבדיקות והתשובה השלילית מגיעה.

כפי שכבר ציינו קל מאוד להסיר כל הגבלה או בדיקה שמתבצעת ברמת הדפדפן ואין לייחס לבדיקה זו שום חשיבות ואין להסתמך בשום אופן על כך שהקלט שמגיע מהדפדפן כבר עבר בדיקת תקינות. האפליקציה חייבת לבצע מחדש בדיקת תקינות בצד השרת על כל הקלט שמגיע מהדפדפן.

מניעת התקפות DOS

התקפת DoS (מניעת שירות) יכולה לנבוע ממספר גורמים, ביניהן – הורדה והעלאה של קבצים גדולים, או הפקת דו״חות ארוכים הגורמים לניצול רב של משאבי רשת.

כדי למנוע מצב זה יש לבצע את הפעולות הבאות:

- יש לאפשר רק למשתמשים מזוהים במערכת לקבל גישה למשאבי הרשת.
- יש להקטין במידת האפשר את נפח הדפים והמשאבים הנגישים למשתמשים לא מזוהים.

PASSWORD AUTO COMPLETE

שימוש במנגנון השלמת סיסמה אוטומטית הנתמכת ע״י דפדפנים, הינה אופציה נוחה מאד עבור המשתמש והשימוש בה גובר והולך, אך היא אינה בטוחה. שימוש ב- Password Auto Complete גורם לסיסמה ושם המשתמש להישמר באופן אוטומטי בדפדפן המשתמש הגולש לאתר ה WEB (זוהי ברירת המחדל המוגדרת בדפדפנים).

גורם זדוני אשר יקבל גישה למחשב המשתמש עשוי לגנוב את הסיסמה השמורה בדפדפן המשתמש ולנצל אותה על מנת להתחזות למשתמש חוקי במערכת. הדבר בולט בעיקר במקרים בהם הגלישה לאתר ה WEB נעשית ממחשבים ציבוריים או משותפים, דבר שחושף את מזהי המשתמש לכל גורם בעל גישה מחשב בו התבצעה הגלישה. לכן, יש לבטל אופציה זו בקוד ה HTML של דף ההזדהות של המערכת ע״י שימוש במאפיין autocomplete=false הנתמך ע״י הדפדפנים המובילים.

SESSION & COOKIES

ניהול SESSION

היות ופרוטוקול ה HTTP הינו חסר מצב (stateless), אנו משתמשים ב Cookies על מנת לשמור על רצף בקשות אפליקטיבי. ה Cookie יכולה לשמור מידע על המשתמש, כגון ה session ID שלו וכדומה.

ישנם שני סוגים של Cookies:

- ✓ **יש להגביל את זמן תקפות ה- Session**

יש לנהל מנגנון שבודק את זמן התקפות של הSession הפעיל. לאחר זמן מסוים של חוסר פעילות אשר הוגדר מראש, יש לנתקו. כלל זה יצמצם את חלון הפגיעות של משתמש שאינו פעיל, ויועיל בצמצום משאבי המערכת.

- ✓ **יש להשמיד את ה-Session בעת Logout**

בעת התנתקות המשתמש, יש לשחרר את כל משאבי המערכת, כך ימנע מצב של Denial Of Service.

- ✓ **יש להקפיד כי ה-Session value לא ממוחזר**

יש לבצע הגרלה אקראית של מספר ה-Session Id. שימוש במנגנון אשר לא מבטיח מספר רנדומאלי חזק קריפטוגרפית עלול לגרום לניחוש ה-Session Id ולהוביל למתקפה מסוג Session Hijacking. ניתן לבצע הגרלה רנדומאלית איכותית על ליצרת מספרים רנדומאליים חזקים קריפטוגרפית, לדוגמא (יצירת מספר רנדומאלי תוך שימוש במנגנון sha-2).

4.1.8. הגדרת HTTP meta tags

יש להגדיר ב-HTTP headers את ה-META tags הבאים:

תאור	Value	Tag
פג תוקף הדף – מייד עם טעינתו. על הדפדפן לפנות לשרת לקבלת הדף המעודכן לפני הצגתו.	"now"	Expires
אין לשמור את הנתונים שיועברו ב-cache.	"no-cache"	Pragma
אין לעשות שימוש ב-cache לנתונים אלו.	"no-cache"	cache-control

ניתן לשלוח את ה-tags ישירות ב-HTTP headers. כמו כן, ניתן להוסיף אותם בראש דף ה-HTML:

```
<HTML>
<HEAD>
<TITLE>…</TITLE>
```

```
<META HTTP-Equiv = “Expires” Content = “now”>
<META HTTP-Equiv = “Pragma” Content = “no-cache”>
<META HTTP-Equiv = “Pragma” Content = “no-cache”>
</HEAD>
<BODY>
```

ב-IIS ניתן להוסיף את השורות הבאות לתחילת ה-ASP Script:

```
<% Response.CacheControl = "no-cache" %>
<% Response.AddHeader "Pragma", "no-cache" %>
<% Response.Expires = -1 %>
```

4.1.9. סוגי Cookies

- **Non Persistent Cookies (Session Cookies)** – נשמרים בצד השרת ובדרך כלל לזמן קצר (לתקופת העבודה של המשתמש מול האתר או עד שהמשתמש סוגר את הדפדפן). Session Cookies נשמרים בדרך כלל בזיכרון השרת ואינן נכתבים למערכת הקבצים.
- **Persistent Cookies** – נשמרים לתמיד או על למועד הפקיעה שהוגדר ע"י האפליקציה מראש.

4.1.10. אי שמירת מידע רגיש ב Persistent Cookies

היות ופרוטוקול ה HTTP הינו חסר מצב (stateless), אנו משתמשים ב Cookies על מנת לשמור על רצף באופן כללי יש להימנע משימוש ב Persistent Cookies. סופן של אלו להגיע לידי התוקף במוקדם או במאוחר ואז הוא יוכל להשתמש בהן על מנת להיכנס לחשבון של המשתמש. במקרים רבים מכילות Cookies באופן לקוי בלתי מאובטח מידע המשמש לזיהוי המשתמש מול שרת ה WEB דבר המאפשר לגורם זדוני לנצל זאת על מנת להתחזות למשתמש.

יש להימנע משמירת מידע רגיש בתוך Cookies. בכלל זה אין לשמור סיסמאות או מידע מזהה אחר אשר אם מגיע לתוקף עלול לפגוע בביטחון המידע של המשתמשים. במידה וה Cookies מכילות מידע רגיש כלשהו, יש לקחת בחשבון כי את הקשר עם השרת יש לאבטח באמצעות SSL, אחרת הן חשופות לציתות בכל בקשה שמבצע הדפדפן מול השרת.

מידע רגיש יש לשמור בשרת. אל המשתמש יש להעביר רק Session ID שהוא מחרוזת תווים ארוכה וקשה לניחוש. מספר זה צריך להיות מקושר באפליקציה לנתונים הרגישים של המשתמש. בצורה זו אנו מצמצמים את החשיפה של הנתונים לתוקפים. ה-ID כמובן מועבר באמצעות Session Cookie ולא יוכל לשמש מעבר ל-Session הספציפי.

4.1.11. אי ביצוע Login אוטומטי בהתבסס על Cookie

אין להשתמש ב-login אוטומטי של משתמשים על בסיס Persistent Cookies או על בסיס כל שיטה אחרת. משתמשים חייבים לבצע login בכל פעם שהם ניגשים מחדש לאפליקציה והם חייבים להעביר את הנתונים המזהים שלהם.

4.1.1. הנחיות נוספות לגבי שימוש מאובטח ב cookie

- ✓ **יש להקפיד כי נעשה שימוש ב-SSL להגנה על Authentication Cookies**

יש להשתמש במנגנון הצפנת תווך על מנת לאפשר הגנת המידע אשר נמצא בתוך Cookie, ראה סעיפים קודמים.

- ✓ **יש להקפיד על הצפנת תוכן ה-Authentication Cookies**

לעיתים, נרצה להצפין לא רק את התווך אלא גם את המידע שקיים בתוך ה- Cookie על מנת למנוע שינויים מצד המשתמש, באופן עקרוני לא מומלץ להתבסס על המידע אשר נמצא ב- Cookie.

- ✓ **קינפוג ה cookie**

יש להגדיר את האפשרויות הבאות:

1. תוקף חייב להיות קצר במידת האפשר
2. דומיין- במידת האפשר צריך להכיל את הדומיין הנדרש בלבד
3. פרמטר HTTP ONLY צריך להיות בעל ערך True
4. פרמטר Secure צריך להיות בעל ערך True
5. Path יש להגדירו ב path אשר אינו ה root

```
HttpCookie myHttpCookie = new HttpCookie("LastVisit", DateTime.Now.ToString());
myHttpOnlyCookie.HttpOnly = true;
myHttpOnlyCookie.path = /root/app/web;
myHttpOnlyCookie.secure = true;
myHttpOnlyCookie.Expires = DateTime.Now.AddDays(1);
myHttpOnlyCookie.Domain = "contoso.com";
```

שימוש בתשתיות מערכת ההפעלה

באופן כללי ניתן לומר כי עדיף להשתמש בתשתית אשר ניתנת על ידי מערכת ההפעלה לצורך זיהוי ומתן הרשאות. מנגנונים אלה אשר נכתבו ונבדקו (ובהם הושקעו אלפי שעות אדם) בדר״כ סביר שיהיו מאובטחים יותר מכל מערכת ומנגנון אשר יפותח באופן פרטי.

4.1.2. זיהוי והרשאות

יש לעקוב אחר הוראות היצרן (Microsoft .NET או J2EE או כל פלטפורמה אחרת) לצורך שימוש נכון במנגנוני הזיהוי וההרשאות. ניהול הזהויות וההרשאות של כל זהות צריך להיעשות במבנה נתונים מתאים (מערכת LDAP מסוג כלשהו) וההזדהות למנגנון זה, או בקשה של הרשאות עבור משתמש, נעשית בפרוטוקולים סטנדרטיים. לכל אחד מהמוצרים בתחום זה קיים מסמך המתאר את הדרך המאובטחת ליישום..

4.1.3. תיעוד האפליקציה

על המתכנתים לשים לב למספר נתונים הקשורים למערכת ההפעלה שאותם ראוי לתעד במהלך פיתוח המערכת. תיעוד נתונים אלו יכול לאפשר לגורמים המאבטחים את מערכת ההפעלה לבצע את מלאכתם ביעילות וללא הפרעה לאפליקציה עצמה.

להלן הדגשים העיקריים:

א. קראו את נוהל ההקשחה והאבטחה של מערכת ההפעלה. היו מודעים לפעולות שתתבצענה במערכות הייצור.

ב. תעדו שירותי מערכת שבהם אתם עושים שימוש על מנת ששירותים אלו לא יוסרו בעתיד ועל מנת שמי שמאבטח את מערכת ההפעלה ידע אלו שירותים הוא יכול להסיר.

ג. תעדו קבצים וספריות שהאפליקציה יוצרת ואת סוג השימוש שאתם עושים בספריות וקבצים אלו על מנת שיהיה ניתן לתת להם את ההרשאות המתאימות.

ד. תעדו ערכי Registry שבהם אתם עושים שימוש או שאותם יצרתם.

ה. תעדו הרשאות מיוחדות להן אתם נזקקים.

ו. תעדו Users ו-Groups שבהם אתם עושים שימוש ואת סוג השימוש וההרשאות.

4.1.4. מערכת הקבצים בשרת

להלן הדגשים בנושא טיפול ושימוש במערכת הקבצים במערכת ההפעלה בשרת באפליקציות WEB:

4.1.4.1. היזהרות משמירת קבצים בתיקיות ייעודיות וייחודיות

כל אפליקציה אשר יוצרת קבצים במערכת הקבצים נדרשת לקבל אישור מאבטחת מידע לכתיבה זו ולהיכן האפליקציה כותבת את הקבצים. ישנן ספריות אשר כברירת מחדל ניתן להריץ אותן מהאינטרנט (למשל

cgi-bin/) או לקרוא את תוכנן (למשל htdocs/). חשוב להימנע מלכתוב קבצים לספריות אלו וספריות דומות. יש ליצור ספריה חדשה שלא תהיה תחת ספריות אלו (רצוי תחת ה-home directory) ושם לאחסן את הקבצים שאותם האפליקציה כותבת. יש לתעד ספריות אלו ולהעביר את ההרשאות המתאימות לגורמים האחראיים על אבטחת המידע.

4.1.4.2. אי הסתמכות על קלט מהמשתמש ביצירת קבצים

אין ליצור קבצים או ספריות במערכת הקבצים בהסתמך על קלט שמגיע מהמשתמש. במקרה זה קיימת סבירות רבה שהמשתמש יוכל לקרוא או לכתוב קבצים בצורה בלתי מבוקרת או להריץ פקודות במערכת ההפעלה. על האפליקציה לקבוע את שם הקובץ ומיקומו מבלי להסתמך כלל על מידע מהמשתמש. יש לבנות טבלת חיפוש אשר מתאימה את שם הקובץ או הספרייה שייצרה האפליקציה לשם הקובץ או הספרייה שבחר המשתמש.

4.1.4.3. אי שימוש בנתיבים יחסיים

אין להסתמך גישה לקבצים באופן יחסי (לדוגמא ..\..\..\data\mydata.txt). יש להשתמש במיקומים מוחלטים (לדוגמא sites\internet\data\mydata.txt\).

4.1.4.4. בדיקת מצב הקובץ

כאשר פותחים קובץ לקריאה או כתיבה חשוב לציין את המצב (mode) בו פותחים את הקובץ. במיוחד אם מדובר בקבצים שנפתחים לקריאה בלבד, יש לדאוג לכך שהמצב יהיה Read Only. מערכות ההפעלה מאפשרות לנו לקבוע באיזה מצב אנו רוצים לפתוח את הקובץ: read-only, write, append, read/write וחשוב להשתמש באופציה זו. אין להסתמך על ברירת המחדל של מערכת ההפעלה מאחר וברירות מחדל עלולות להשתנות.

4.1.4.5. בדיקת קיום הקובץ בשרת

לפני שהאפליקציה פותחת קובץ לקריאה עליה לוודא שהקובץ קיים ושהוא ניתן לקריאה, בייחוד אם מדובר במידע שנשען על קלט ממשתמשים. גישה לקבצים לא קיימים או קבצים שאינם ניתנים לקריאה עלולה להעיד על התקפה.

ממשקים עם מערכות חיצוניות

מרבית המערכות אינן פועלות בחלל ריק. המערכת מושכת מידע ומעבירה מידע למערכות משיקיות. בבניית ממשק עם מערכות חיצוניות יש צורך לוודא שהמידע נשלח ומתקבל כמצופה, ושתוכן המידע תואם את ההנחות לגביו. עקרונות בניית ממשקים מובאים להלן:

נוהל פיתוח מערכות מאובטחות

- ✓ **זיהוי של מערכות**

מערכות משיקיות צריכות להזדהות זו בפני זו. בפרט, המערכת השולחת צריכה להזדהות בפני המערכת המקבלת. הזדהות מערכת יכולה להתבצע באמצעות חתימה דיגיטלית, תהליך hand shake המערב החלפת סיסמאות (סוד משותף) בין המערכות, וכדומה.

- ✓ **אימות מידע מועבר**

למידע המועבר צריך להתלוות זיהוי אימות תוכן המידע בצורת Hash, checksum, CRC, או שיטה מקבילה. ברוב פרוטוקולי התקשורת מובנה בקרת שגיאות מעין זו,

אבל היא אינה מתאימה למצב בו מועברים הנתונים דרך קבצים משותפים (כפי שבדרך כלל קורה).

- ✓ **בדיקת קלטים**

קליטת נתונים ממערכות חיצוניות צריכה לכלול בדיקות לוגיות, כדי לוודא שהקלט מתאים להנחות המערכת הקולטת. מבחינה זו יש להתייחס לקלט ממערכת משיקה באותה רמת של חשד כאל קלט ממשתמש. צריך להניח שלא בוצעה שום "בדיקת

שפיות" על הקלט ולכן יש לוודא בדיקות של שדות, רשומות, וכדומה, לפי ההנחות הלוגיות של המערכת המקבלת. כמו כן אין להסתמך על בדיקת קלטים בצד הלקוח.

- ✓ **קליטת נתונים**

קליטת נתונים צריכה להתבצע על ידי מנגנון ה- Roll-Back, על מנת לאפשר חזרה למצב ידוע באם הנתונים מתגלים כשגויים. נתונים שגויים צריכים להיות מדווחים לתוכנה השולחת (אם קיים ממשק להעברת שגיאות), ולמנהל המערכת, על מנת לברר את הסיבה.

הנחיות אבטחה ל- WEB SERVICES

בנוסף לעקרונות אבטחת הקונבנציונאליים שתקפים גם באבטחת ה- WS יש לקחת בחשבון מספר עקרונות נוספים:

- ✓ **יש להשתמש ב- WS Security**

מאשר זיהוי משתמשים על סמך תעודות דיגיטאליות. בכל קריאה לפונקציה של ה-Web Service נשתל בגוף הודעת ה- SOAP ערך מזהה אשר מבוסס על התעודה הדיגיטאלית של השולח. קיימים מנגנונים בשפות שונות על מנת לאפשר את השימוש ב- WS Security, כגון WCF עבור .NET.

- ✓ **יש להשתמש ב - XML Encryption**

הודעות SOAP עוברות ברשת בצורה גלויה (CLEAR TEXT) ללא כל הצפנה. מצב שכזה חושף את האינפורמציה בפני צפייה שאינה מורשית,ולכן יש לקחת זאת

בחשבון כאשר מדובר בפונקציה בעלת פרמטרים רגישים כגון שמות משתמשים, סיסמאות, ערכים מספריים בעלי חשיבות וכדומה.

פרוטוקול ה- XML Encryption דואג להצפין את המידע של תוכן הודעות ה-SOAP ובאמצעותו ניתן לקבוע כי חלקים מסוימים מההודעה דורשים הצפנה.

- ✓ **מידע גלוי אודות ה- WS**
  - רישום בשרת ה- UDDI, תפקיד שרת UDDI הינו לבצע רישום של ה- WSבמקום מרכזי, על מנת לאפשר לאפליקציות הצורכות שירותים אלו למצוא את השרות ואת כתובת ה Web Service- (Endpoint).

    במקרה ואופי האפליקציה הינו רגיש, אין לבצע רישום בשרת UDDI.

  - פרסום קובץ ה- WSDL, פרסום קובץ ה- WSDL חושף את ממשק ה-API של שירותי ה- Web Services בפני כל העולם, ומתאר בפני תוקף פוטנציאלי את מבנה הפונקציה וסוג הקלט הנדרש. פרסום שכזה עלול לגרום למשיכת אלפי פורצים מרחבי העולם אל כיוון המערכת.

    יש להגביל את הרשאות הגישה לקובץ זה ברמת מערכת הקבצים או לחלופין להוריד קובץ זה מהשרת, ולמסרו למפתחים באופן ידני.

- ✓ **אמינות נתונים**

בכדי לוודא כי מידע אשר עבר ברשת לא שונה בזמן התעבורה, יש לבצע חתימה דיגיטלית של ההודעה. ניתן להשתמש בפרוטוקול XML Signature על מנת לחתום תוכן הודעות SOAP אשר

נשלחות אל web service ובחזרה. ניתן להשתמש בפרוטוקול זה על מנת לחתום חלקים מסוימים מתוכן הודעות ה-SOAP על מנת לוודא כי ההודעות לא השתנו במהלך התעבורה ברשת.

- ✓ **ביטול שירותים מיותרים**

שרת האפליקציה אשר מפרסם את שרותי ה-web services מוגדר לקבל את הפניות ל-WS באמצעות 3 פרוטוקולים: HTTP GET , HTTP POST ו ה- SOAP.

יש לבטל את תמיכת השרת ב- HTTP-GET ו - HTTP-POST עבור Web Services בשרתי Production, על מנת להימנע ממתקפות פוטנציאליות אשר מגיעים דרך פרוטוקולים אלו.

להלן דוגמה לביטול שירותים:

```
<webServices>
  <protocols>
    <remove name="HttpPost" />
    <remove name="HttpGet" />
    <remove name="Documentation" />
  </protocols>
</webServices>
```

שימוש ב- XML

השימוש ב-XML צובר תאוצה ככלי לתקשורת בין רכיבים שונים במערכת. השימוש ב-XML מאפשר כתיבה קלה יותר של קוד ותקשורת בין רכיבים, תוך הסתמכות על פורמט קריא ופשוט. יחד עם זאת, השימוש ב-XML צופן בחובו מספר בעיות בעלות השלכות על אבטחת המידע של המערכת.

4.1.5. סיכונים ב-XML:

4.1.5.1 תווים Non Printable

חלק גדול מהמנגנוני לטיפול ב-XML DOM מתקשים לטפל במידע המכיל תווים non-printable, ועשויים לגרום לנפילה או לגרום להתנהגות חריגה במקרה שתווים אלו קיימים ב-XML. תוקף יכול לנצל זאת על ידי הכנסת תווים מסוג זה לקלט, ובכך לגרום למערכת להפסיק לפעול או לפעול באופן בלתי צפוי.

4.1.5.2. תווים מיוחדים

באופן דומה, הכנסת תווים מיוחדים מעולם ה-XML (כגון < או ״) עשויים לגרום לבעיות בעת ה-Parsing. התווים '>' ו-'<' בעייתיים בכל נקודה בקלט, אולם בעת השימוש ב-Attributes, גם תווים אחרים, כגון רווח, =, ״ וכו' עשויים לפגוע ב-Parsing ולגרום להתנהגויות בלתי צפויות.

4.1.5.3. XML Injection

שימוש בקלט המכיל תווים מיוחדים של XML יכול להוביל במקרים מסוימים להתקפה חמורה הרבה יותר של הזרקת XML. בהתקפה מסוג זה, התוקף בונה קלט מיוחד המכיל את התווים הדרושים לבניית TAG-ים ו-Attribute-ים נוספים בתוך הודעת ה-XML הנבנית על ידי התוכנית. שתי הדוגמאות הבאות מציגות אפשרויות אלו:

הזרקת XML Tag - נתבונן בבקשה הבאה:

```
<XFer-Req>
        <DestAccount>111111</DestAccount>
        <SrcAccount>222222</SrcAccount>
</XFer-Req>
```

הבקשה מכילה את מספר החשבון אליו להעביר כסף (מוזן על ידי המשתמש), ואת מספר החשבון ממנו מעבירים את הכסף (מוזן על ידי המערכת, על פי זיהוי המשתמש). הבקשה נבנית על ידי שרשור המחרוזות המרכיבות את ה-XML עם זו המגיעה מהמשתמש.

כעת תוקף יכול לשלוח קלט אשר ייצור את הבקשה הבאה, תוך שימוש בקלט המכיל סימנים של XML:

```
<XFer-Req>
        <DestAccount>111111</DestAccount>
        <SrcAccount>333333</SrcAccount>
        <DestAccount>111111</DestAccount>
        <SrcAccount>222222</SrcAccount>
</XFer-Req>
```

כעת הבקשה מכילה פעמיים את שני השדות המכילים את מספר החשבון, אלא שהמופע הראשון של שדה ה-SrcAccount נשלט על ידי התוקף ויכול להיות שונה מהחשבון האמיתי. האפליקציה, בסבירות גבוהה, תקבל מה-DOM את המופע הראשון של שני המשתנים, ותתבסס עליהם.

דוגמא נוספת, פשוטה אף יותר, מתרחשת כאשר הערכים מוזנים כ-Attributes ולא כ-Values, כמו בבקשה הבאה:

```
<XFer-Req DestAccount=111111 SrcAccount=222222 />
```

במקרה הנ״ל, התוקף איננו צריך להכיר את כל מבנה ה-XML ולהזריק תווים כגון < או >, ומספיק לו להכניס את הקלט אשר ייצר את הבקשה הבאה:

```
<XFer-Req DestAccount=111111 SrcAccount=333333
SrcAccount=222222 />
```

ושוב, במקרה זה, התוקף יכול לשנות את חשבון המקור, והפעם באופן קל בהרבה, ללא צורך בהכרות מלאה של מבנה ה-XML Request.

4.1.6. כתיבה מאובטחת בעת שימוש ב-XML

על מנת להתגונן מהתקפות מסוגים אלו, יש להקפיד על מספר עקרונות בעת עבודה עם XML:

4.1.6.1. בדיקות קלט

יש לבצע תמיד בדיקות של הקלט המגיע מהמשתמשים על מנת לוודא כי הקלט הוא אכן הקלט המצופה (למשל, בדיקה שמספר חשבון מכיל רק מספרים בטווח מתאים, בדיקה המוודאת כי כל התווים הינם printable, בדיקה המוודאת כי לא קיימים תווים שאינם מספרים/אותיות וכדומה).

4.1.6.2. שימוש ב-Value ולא ב-Attribute

קלט מהמשתמש יש תמיד לשמור כ-Value ב-XML ולא כ-Attribute. דבר זה מאפשר שליטה טובה יותר על האופן שבו קלט זה נכתב ונקרא אל תוך ה-XML. שמירת הקלט בתור Attributes מקילה על האפשרות של XML Injection.

4.1.6.3. ביצוע Encoding לתווים Non Printable

בכל מקרה בו יש שימוש בתווים שאינם printable יש לבצע Encoding של הערכים. הדרך הפשוטה ביותר לביצוע ה-Encoding היא באמצעות Base64 Encoding. Attribute של ה-Tag ישמור את סוג ה-Encoding לצורך פענוח על ידי הצד הקורא.

4.1.6.4. ביצוע Encoding על קלט משתמש

בעת קליטת קלט מהמשתמש, יש לבצע Encoding של המידע, על מנת למנוע מצב של XML Injection כדוגמת הדוגמא הראשונה. כלל – שימוש ב-Base64 Encoding מבטיח מניעה של הזרקת XML, אולם עשוי לפגוע בקריאות השדרים. לכן, במקרים בהם קיימת אפשרות להבטיח (באמצעות בדיקות מתאימות), שהקלט איננו מכיל תווים שהם non-printable, ניתן להשתמש ב-Encoding דמוי HTML (<, >) עבור תווים אלו.

מניעת התקפות DOS

התקפות מניעת שירות (DoS – Denial of Service) הן התקפות נפוצות באפליקציות WEB וקשה מאוד להתגונן בפניהם. עם זאת, ניתן לנקוט במספר אמצעים על מנת לצמצם אפשרות התקפה מעין זה.

להלן תיאור הבעיות אשר יכולות לגרום למניעת שירות במערכת ואמצעי ההתגוננות שיש לנקוט על מנת להתמודד איתן:

4.1.7. שימוש יתר ב I/O

חיפושי DB, תמונות גדולות או כוננים קשיחים לא איכותיים יכולים לגרום לשימוש יתר ב- I/O ובכך לפגוע באופן משמעותי בביצועים. יש לטפל בשימוש יתר ב- I/O באופן הבא:

- יש לאפשר רק למשתמשים מזוהים לצרוך כמות רבה של משאבי I/O.
- יש לנתח השפעה על פעילות I/O של פעולות רגילות, ולוודא שפעולות רגילות לא גורמות להעמסת יתר של כוננים קשיחים.

4.1.8. שימוש יתר במעבד

לוגיקה עסקית מורכבת כמו לדוגמא הפקת נתונים סטטיסטים יכולים לגרום להשבתה זמנית של מערכת כלפי משתמשים אחרים. הפעולות הבאות עשויות לצמצם את הנזק האפשרי:

- יש לאפשר רק למשתמשים מורשים שימוש בתהליכים הדורשים ניצול רב של המעבד.
- יש לבדוק נפח שימוש במעבד ע"י פעולות סטנדרטיות.

4.1.9. נעילת משתמשים

מנגנון נעילת המשתמשים הוא מנגנון חשוב לאבטחת מידע, אך כאשר מנגנון זה אינו מיושם כראוי הוא עלול לשמש גורמים זדוניים להתקפת DoS על המערכת. כדי למנוע ניצול לרעה של המנגנון יש לשמור על קיום הכללים הבאים:

- מדיניות משתמשים וסיסמאות- יש לאכוף שימוש בשמות משתמשים הקשים לניחוש (או לתת למשתמשים לבחור את שמותיהם בעצמם) דבר שיקשה על גורם זדוני לבצע enumeration של שמות המשתמשים.
- יש לדאוג לשחרור אוטומטי של נעילות חשבונות המשתמשים לאחר זמן מוגדר (למשל לאחר כ 15 – 30 דקות).
- יש לחסום גישה לשרת לכתובות IP שנעלו יותר ממשתמש אחד במהלך פרק זמן קצר (למשל דקה) ולהתריע על כך באופן מיידי למנהלי המערכת.

שמירה על FLOW

בכל מערכת שהיא מעבר לדפי מידע יש חשיבות רבה ל- Flow של המערכת. Flow הוא הדרך ליישם תהליכים עסקיים באפליקציה Web- ית. המצב שבו נמצא המשתמש צריך להישמר בצד השרת ב- Session. בשום פנים ואופן אין לשמור את מצב המשתמש רק בצד המשתמש שכן יש להניח שמשתמש זדוני ינסה לעדכן את המצב ולבצע פעולות שנוגדות את הלוגיקה העסקית. בתהליך, מצב המשתמש חייב להישמר ב- session, וכל ניסיון לעקוף את ה- Session חייב להיכשל.

5. יישום העקרונות בטכנולוגיות אשר בשימוש במשרד הבריאות

הקדמה

להלן דגשים ליישום עקרונות אבטחת המידע בסביבות הפיתוח העיקריות בהם נעשה שימוש במשרד הבריאות NET.:

5.1.1. Strong Name Assemblies

בזמן הקומפילציה של ה- Assemblies יש לחתום אותם כדי למנוע זיופים:

יוצרים את המפתח בעזרת הכלי SN:

```
SN –k keypair.snk
```

לאחר יצירת הקובץ יש לדאוג שהגישה אליו תהיה מוגבלת.

יוצרים מפתח ציבורי:

```
SN –p keypair.snk public.snk
```

שילוב המפתח בפרוייקט מתבצע:

```
Imports System.Reflection
<Assembly: AssemblyDelaysSignAttributes(true)>
<Assembly: AssemblyKeyFileAttribute("C:\Keys\keypair.snk")>
```

החתימה הסופית:

```
SN –R assemblyname.dll keypair.snk
```

יש אפשרות לבצע חתימה וירטואלית לצורכי הדמיית סביבה אמיתית:

```
SN –Vr assemblyname.dll
```

5.1.2. הרשאות ב- Assembly

סביבת ה- Common Language Runtime מאפשר להגדיר הרשאות בעת ריצת הקוד. כך ניתן לשלב את מנגנון האבטחה של מערכות הפעלה (בעיקר DAC), הכולל מערכת הרשאות מוגדרת דינאמית ופרטנית עבור כל

Assembly וליצור מנגנון אבטחה משופר. ניתן להשתמש גם במנגנון RBAC עבור הקוד. מנגנון ההרשאות מאפשר לבצע את ההגדרות הבאות בפרויקט:

Request – בקשת הרשאות עבור פעולות שמבצע ה Assembly

כאשר הקוד מבקש לבצע פעולה מול מערכת ההפעלה הוא צריך לבקש אותה. מומלץ לבקש הרשאות מינימאליות הנדרשות לפעולה כדי למנוע ניצול לרעה.

לדוגמא:

```
<Assembly: FileIOPermission(SecurityAction.RequestMinimum, Read =
@"C:\files\log.txt)>
```

Refuse – חסימת הרשאות באופן עצמי

יש להגדיר הרשאות שאין בהם צורך בפרויקט. כך במקרה שקוד מזיק ינסה לשתף את הפרויקט בפעולות בלתי חוקיות הקוד יכשל.

דוגמא:

```
<Assembly: FileIOPermission(SecurityAction.RequestRefuse,
Unrestricted=true>
```

Optional – הגדרת הרשאות לא הכרחיות

ניתן לבקש הרשאות עבור הקוד שאינן הכרחיות (כלומר הקוד יכול לרוץ גם בלעדיהם)

דוגמא:

```
<Assembly: FileIOPermission(SecurityAction.RequestOptional, Unrestricted =
true>
```

Demand – דרישת הרשאות ממפעיל הקוד

כאשר הקוד (Assembly) מופעל ע״י קוד אחר הוא רשאי לדרוש הרשאות מסוימות, ורק במידה ומי שהפעיל וכן כל השרשרת מעליו בעלת הרשאות כאלה אז הקוד ירוץ.

דוגמא:

```
<Assembly: FileIOPermission (SecurityAction.Demand, Read =
@"C:\files\log.txt)>
```

דוגמא בזמן ריצה:

```
New FileIOPermission (FileIOPermissionAccess.Read,
@"c:\files\Log.txt").Demand()
```

Link Demand – בדיקת הרשאות של המפעיל הישיר בלבד

דוגמא:

```
<PasswordPermission(SecurityAction.LinkDemand, Unrestricted=true)>
```

Remoting Demands – במערכות המשתמשות בפריסה על מספר מחשבים (Remoting) בדיקות רגילות (Demand,LinkDemand) לא נאכפות. אבל בדיקת הרשאות ברמת הקוד נאכפת.

Assert – לאפשר ביצוע פעולות שחסומות בדרך כלל למפעיל הקוד

כאשר אין למפעיל הקוד הרשאות מתאימות, יכול ה Assembly לערוב לו ולאפשר לו לבצע פעולות שלקוד יש הרשאה אליהן.

דוגמא:

```
New FileIOPermission (FileIOPermissionAccess.AllAccess,
@"c:\files\Log.txt").Assert()
```

ביטול ה- Assert

יש לצמצם למינימום את הזמן שהמשתמש "נמצא תחת" Assert, לכן יש לבטל אותו מיד בתום השימוש.

דוגמא:

```
CodeAccessPermission.RevertAssert()
```

שימוש נכון ב- Assert

בעת שימוש ב- Assert סביבת הריצה מפסיקה לבצע בדיקת הרשאות על כל שרשרת המפעילים של הקוד, וכך גורמת לפרצת אבטחה אפשרית על כן יש להיזהר בשימוש באפשרות זו ולהשתמש בה בצורה מושכלת ורק היכן שחייבים.

Deny – מניעת הרשאות ממפעיל הפרויקט

ניתן להגדיר את הקוד שימנע הרשאות ממפעיליו

דוגמא:

```
New FileIOPermission (FileIOPermissionAccess.AllAccess,
“c:\files\log.txt).Deny()
```

הרשאות לפי תפקידים

ניתן לממש מנגנון הרשאות מבוסס תפקידים – RBAC. ניתן לעשות את זה כהגדרת attribute או לבצע זאת בזמן ריצה:

```
[PrincipalPermission(SecurityAction.Demand, Role="Manager")]
                    או
permCheck = new PrincipalPermission(null, "Manager")
permCheck.Demand()
```

ניתן לבדוק האם מפעיל הקוד משויך לתפקיד מסוים:

```
IPrincipal.IsInRole("Manager")
```

או כהגדרת attribute:

```
<authorization>
  <allow roles="Manager"/>
  <deny roles="?"/>
</authorization>
```

5.1.3. Partial Trust Assemblies

קוד בטוח חלקית הוא קוד שלא קיבל אישור מלא, בדרך כלל קוד מהאינטרנט. יש להימנע ההפעלה של הפרויקט ע״י קוד מנוהל חלקית זה, ע״י שימוש באופציה זו. כדי לאפשר את האופציה יש להוסיף את ה- Attribute הבא:

```
<Assembly: AllowPartiallyTrustedCallers>
```

5.1.4. Unmanaged Code

בניגוד לפרויקטי קוד (Assemblies) המשתמשים ב Net Platform. הנחשבים קוד מנוהל, פרויקטים שמשתמשים בפלטפורמות אחרות נחשבים לא מנוהלים. קוד כזה יכול להגיע לכל המשאבים במערכת כאשר מבחינת פונקציונאליות הוא מתנהג כקוד בטוח. רצוי לא להשתמש בקוד לא מנוהל בכלל, כאשר אין ברירה יש לעטוף את הקוד הלא מנוהל באובייקטים מנוהלים. ניתן לאפשר קוד מנוהל ע"י הגדרת Attribute:

```
<SecurityPermission (SecurityAction.Assert, UnmanagedCode=true)>
```

5.1.5. הורשה

כאשר אין שליטה על הגורמים שישתמשו בקוד יש להגביל את ההורשה ממנו כדי למנוע אפשרות של בעיות אבטחה כגון חשיפת מידע שהוגדר כ- Protected או טיפול לא מלא בשגיאות. ניתן להגביל את ההורשה באחת הדרכים הבאות:

- הכרזה על האובייקט כבלתי ניתן להורשה – (לדוגמא ע"י שימוש ב- NotInheritable)
- הגבלה לפי מפעילים – בדוגמא הבאה כדי לרשת מ- TheClass הקבוצה חייבת להיות בעלת הרשאות מסוג Environment:

```
<EnvironmentPermission (SecurityAction.InhertianceDemand,
UnmanagedCode=true)>
Public Class TheClass...
```

- ניתן להגדיר הרשאות נדרשות לפונקציות וירטואליות – בדוגמא זו מפעיל שיש לו הרשאות PrivateKeyPermissions יכול לממש את הפונקציה SetKey:

```
<PrivateKeyPermissions (SecurityAction.InhertianceDemand,
UnmanagedCode=true)>
Public virtual void SetKey (byte[] key)...
```

5.1.6. הגבלה לפי חתימה

כאשר הפרויקטים חתומים (Strong Name Assemblies) אפשר להגביל את הגישה לקוד לפי החתימה עצמה:

```
<StringNameIdentityPermission (SecurityAction.LinkDemand, PublikKey="……")>
```

5.1.7. Serialization

חלקי קוד שמשתמשים ב- Serialization חושפים את המידע בהם (לדוגמא כשעושים dump לאובייקט) ניתן להיעזר ב-Attribute כדי לדרוש הרשאות מתאימות מהקוד המפעיל:

```
<SecurityPermission (SecurityAction.Demand, SerializationFormatter=true>
```

5.1.8. Deserialization ממקור לא בטוח

אין לבצע Deserialization ממקור שאינו נחשב כמקור "בטוח".

5.1.9. Isolated Storage

לעיתים ניתן להשתמש במנגנון אחסון מבודד (Isolated Storage) במקום קבצים (ואמצעי קלט פלט אחרים). מנגנון Isolated Storage מאפשר הגדרת הרשאות לפי משתמש Domain, פרויקט ועוד.

להלן דוגמא ליצירת- Isolated Storage שהגישה אליו יכולה להיות מכמה אפליקציות שונות:

```
Dim isoFile As IsolatedStorageFile = IsolatedStorageFile.GetStore
(IsolatedStorageScope.User Or IsolatedStorageScope.Assdembly, Nothing,
Nothing)
```

5.1.10. Custom Errors

כברירת מחדל ניתן לראות את הודעת השגיאה המלאה כשמריצים את האפליקציה מקומית. כך שעבור Clients מרוחקים תוצג הודעת שגיאה כללית. אך ניתן לשנות את זה ע"י שינוי של השדה customeErrors:

```
<configuration>
  <system.web>
    <customErrors>
      :
      mode="RemoteOnly"
      :
    </customErrors>
```

```
  </system.web>
</configuration>
```

במקרים רבים, במהלך העברת הקוד מסביבת הפיתוח לייצור שוכחים לשנות פרמטר זה, וכך עלולות להיחשף למשתמש הקצה הודעות שגיאה מפורטות. על כן, יש לוודא ששדה זה ב- Remote Only או ב- On על מנת למנוע מצב זה.

5.1.11. התאמת פירוט הודעות השגיאה

יש לוודא שהודעות השגיאות המפורטות מודפסות רק כשצריך. לדוגמא, כאשר הפרוייקט בתצורת הפעלה debug ניתן להדפיס הודעות מפורטות אך כשהוא בשלב הRelease יש להדפיס הודעה כללית בלבד.

5.1.12. ספריות הקריפטוגרפיה של Net.

יש להימנע מפיתוח עצמאי של תהליכי הצפנה. יש להשתדל להשתמש בספריה סטנדרטית של Net. לדוגמא System.Security.Cryptography.

5.1.13. FXCop.exe

מומלץ להשתמש בתוכנית זו, (ניתן להשיג חינם ב- www.gotdotnet.com) אשר מאפשרת לסרוק את קוד הפרויקט ולאתר חריגות ממדיניות אבטחת המידע של Microsoft או בעיות אבטחה הקיימות בקוד.

דגשים ל-ASP.NET

5.1.14. אחסון מידע רגיש

ב ASP.NET ישנם מספר מקומות בהם מתכנתים בדרך כלל מאחסנים נתוני תצורה רגישים כגון: מחרוזת ההתחברות לבסיס הנתונים. עם זאת, שמירה בלתי מאובטחת של נתוני תצורה רגישים אלה עלולה לגרום לבעיית אבטחה חמורה במידה וגורם זדוני יקבל גישה למקומות אלה.

מקומות שכיחים בהם בדרך כלל מתכנתים שומרים נתוני תצורה:

- קובץ Web.Config.
- משתני מערכת.
- קבצי טקסט.
- מקודד באופן קשיח בתוך קוד האפליקציה.

לדוגמה: באפליקציות ASP די נפוץ למצוא נתוני תצורה רגישים כגון: מחרוזת התחברות הכוללת את הסיסמה לבסיס הנתונים הנשמרים בקובץ global.asa בצורה בלתי מאובטחת ובצורה גלוייה (Clear Text). בדרך כלל סביר למצוא קטע קוד הנראה כך:

```
Sub Application_OnStart
        Application("Conn1_ConnectionString") = _
"Provider=SQLOLEDB.1;UID=sa;PWD=pass;Initial Catalog=src;Data
Source=localhost;"
End Sub
```

צורת שמירה זו חושפת את הנתון הרגיש לכל גורם אשר יקבל גישה לקובץ global.asa ובכלל זה פורץ או מנהל מערכת אשר לאו דווקא אמור לדעת מהי הסיסמה לבסיס הנתונים (בדר"כ אמורה להיות הפרדת תפקידים בין מנהל בסיס הנתונים לבין הגורם האחראי על פריסת האפליקציה לשרת הייצור).

אין מקום אחד אשר מומלץ לאחסן בו את המידע באופן קבוע יותר מכל מקום אחר אך ניתן בהחלט לומר כי מומלץ שלא לקודד משתנים אלו באופן קשיח בתוך האפליקציה.

הסיבה נעוצה בין השאר במאפיין שנוסף ב ASP.NET והוא התחקות (Tracing). מאפיין זה מאפשר למפתחים להתחקות אחר המידע ששמור ברמת האפליקציה עבור כל דף וכל תוקף בעל ידע מינימאלי ב ASP.NET יכול לנסות ולשלוף מידע זה. מומלץ לחסום אפשרות זו ע"י הגדרה בקובץ Web.Config אך לא ניתן למנוע אפשרות זו באופן מוחלט מכיוון שישנה אפשרות להגדיר דפים מסוימים באופן ספציפית לתמוך במאפיין זה. ברגע שדף בעל מידע אפליקטיבי ידלוף החוצה המידע על ההתחברות לבסיס הנתונים יגיע לתוקף ולכן מומלץ לא לקודד מידע זה באפליקציה. כמו כן, בשרתי ייצור אין להשתמש באפשרות trace ויש לנטרלה ע"י הגדרות מתאימות בקבצי התצורה של המערכת (Web.Config).

להלן הנחיות "עשה ואל תעשה" בנושא שמירה על מידע רגיש:

- יש להימנע מלאחסן מידע רגיש במשתני מערכת (Application Variables).
- אין לשמור מידע רגיש בצורה קשיחה (Hard Coded) בקוד האפליקציה ומומלץ לאחסנו במאגר אשר מאפשר הגדרת הרשאות על מקום האחסון (כגון: Registry, Database, Active Directory) ורק באופן מוצפן.
- שמור נתוני תצורה רגישים (כגון סיסמאות) בצורה מוצפנת בלבד תוך שימוש בסטנדרטים מוכרים ומוכחים של אבטחה ותוך שימוש באלגוריתמים חזקים של הצפנה.
- על מנת לאבטח את נתוני התצורה הרגישים מומלץ להשתמש באחד מהפתרונות הבאים:

- שמירה של הנתונים הרגישים בקבצי התצורה או ב Registry של השרת תוך שימוש במנגנון DPAPI של Microsoft לצורך ביצוע ההצפנה ללא צורך במפתח הצפנה.
- שמירה של מפתח ההצפנה ב Registry באמצעות DPAPI ושימוש בו על מנת להצפין את נתוני התצורה הרגישים בקבצי התצורה (למשל ב Web.Config) תוך שימוש בספריות הצפנה סטנדרטיות מסוג CryptoAPI ובאלגוריתמי הצפנה סטנדרטיים (כגון: Tripple-DES או AES).

הערה: יש להדגיש כי בשני המקרים מומלץ להגדיר הרשאות גישה למקום בו נשמרים הנתונים הרגישים רק למשתמש האפליקציה המריץ את היישום.

לדוגמה: ניתן לשמור את מחרוזת ההתחברות לבסיס הנתונים בקובץ התצורה ה Web.Config באופן הבא כאשר מחרוזת ההתחברות מוצפנת באמצעות אחת מהשיטות שהוזכרו לעיל:

```
<appSettings>
        <add key="ConnectionString" value="(your encrypted connection string)"
/>
</appSettings>
```

ואז הגישה למידע זה מתוך האפליקציה תראה כך:

```
string connstring = ConfigurationSettings.AppSettings["ConnectionString"];
```

יש לפענח באמצעות אמצעי ואלגוריתם connstringכאשר את הערך המוצפן של המשתנה וכדומה).DPAPI, TrippleDESההצפנה אשר נעשה בו שימוש (

5.1.15. אימות (Authentication)

ב ASP.NET ניתן להבחין בשלושה מנגנוני אימות עיקריים:

- אימות מבוסס חלונות (Windows Authentication)
- אימות מבוסס שירות פספורט (Passport).
- אימות מבוסס טפסים (Forms Authentication)

מטרת אימות מבוסס Windows היא לאפשר מיפוי של בקשות המופנות לשרת האינטרנט לחשבונות של משתמשים בסביבת Windows. יש לבחור מנגנון זה כאשר למשתמשי המערכת יש חשבונות ברשת ולרוב הכוונה היא למערכות אינטרנט או למערכות עבור עובדי הארגון.

מנגנון האימות של Passport הוא בעצם מחסן מרכזי של שמות משתמשים וסיסמאותיהם אשר מאוחסנים בשרתי מיקרוסופט. מפתחי אפליקציות אשר בוחרים במנגנון זה נדרשים להשתמש בערכת הפיתוח של

Passport SDK. בשיטה זו המשתמש נדרש להציג "כרטיס" אותו הוא מקבל משירות Passport ע"י אימות מול שירות זה. יש לציין כי השימוש במאגר מרכזי זה יוצא תלות בזמינות של המאגר וכי כבר קרה בעבר שמאגר זה הותקף וכתוצאה מכך לא סיפק שירותים ולכן עבור הבנק לא מומלץ להשתמש במנגנון זה.

אימות מבוסס טפסים הוא מנגנון אשר קיים מזה זמן רב בסביבת האינטרנט והשימוש בו ב ASP.NET נעשה בצורה קלה ונוחה מאד והוא נחשב למאובטח יותר. יש להגדיר בקובץ ה Web.Config את סוג האימות בצורה הבאה:

```
<configuration>
    <system.web>
        <authentication mode="Forms" />
</system.web>
</configuration>
```

יצוין כי כאשר מגדירים את אופן האימות הגדרה זו היא לרוחב כל האפליקציה ולא ניתן להשתמש במנגנונים שונים של אימות באותה האפליקציה.

5.1.16. הרשאות (Authorization)

לאחר שנעשה אימות לגבי מבקש הבקשה יש לבדוק הרשאות של המבקש: "האם המבקש רשאי לגשת למשאב אותו הוא מבקש?". ב ASP.NET ניתן להבחין בין 2 מנגנונים לניהול הרשאות: ACL/File Authorization ו URL Authorization.

ACL/File Authorization- (Access List Authorization) ידוע גם בשם הרשאות קבצים והוא בעצם מנוהל ברמת מערכת החלונות ע"י מתן הרשאות מתאימות לקבצים. נוח להשתמש במנגנון זה כאשר משתמשים במנגנון האימות של חלונות.

URL Authorization- בשונה מהמנגנון הקודם, מנגנון זה מנוהל ברמה הפונקציונאלית ע"י ASP.NET ומוגדר ע"י מיפוי הרשאות בקובץ ה Web.Config. נוח להשתמש במנגנון זה כאשר משתמשים במנגנון האימות מבוסס הטפסים.

5.1.17. קבלת מידע אודות משתמשים מאומתים

ASP.NET מאפשר, ע"י שימוש ב API, לקבל מידע אודות יוזם הבקשה. לכל בקשת HTTP אשר מגיעה לשרת מוצמד אובייקט מסוג HttpContext. ע"י שימוש באובייקט זה ניתן לשלוף את המאפיינים של המשתמש, לדוגמא מתוך דף (Page) ניתן לפנות אל: Page.Context.User. כאשר משתמשים באימות חלונות אז אובייקט זה הוא מסוג WindowsPrincipal ואחרת הוא מסוג GenericPrincipal. לאובייקט זה יש פונקציה הנקראת IsInRole אשר בודקת האם המשתמש המאומת שייך לקבוצת המשתמשים הרשאים לבצע את הפעולה. לדוגמא באימות חלונות הקבוצות הם למעשה קבוצת משתמשים (User Groups). אם רוצים לבדוק האם המשתמש מאומת מתוך הקוד ב ASP.NET יש להשתמש בקוד הנ"ל:

```
If (User.Identity.IsAuthenticated) {
        המשתמש מאומת //
{
```

5.1.18. אבטחה מבוססת הרשאות

מערכת חלונות נבנתה לפי ארכיטקטורת אבטחה מובנית הרשאות כאשר ניתן לראות בקבוצות משתמשים שונות כדוגמא לקבוצת הרשאות. קל מאד לכתוב קוד ב ASP.NET אשר בודק הרשאה:

```
void StartTheMachinery() {
  IPrincipal p = Thread.CurrentPrincipal;
  if (!p.IsInRole("Supervisors")) {
    string msg = "Only supervisors may " +
                 "start the machinery";
    throw new SecurityException(msg);
  }
  // really start the machinery...
{
```

בדיקה זו נקראת בדיקה הכרחית (Imperative) מכיוון שיש הכרח בכתיבת קוד הבדיקה. שימוש בסוג זה של בדיקה יש לבצע רק במקרים של תרחישים מסובכים. דרך שונה לביצוע בדיקה זו נקראת בדיקה הצהרתית (Declarative). שיטה זו היא פחות גמישה והכוונה היא לא לקודד את הבדיקה באופן מפורש בקוד, אלא להצהיר עליה. לדוגמא:

```
[PrincipalPermission(SecurityAction.Demand,
                     Role="Supervisors")]
void StartTheMachinery() {
  // if we make it here,
  // really start the machinery...
```

בצורה כזו ה CLR ידאג ליצור חריגת אבטחה כנדרש. ניתן להגן על קוד בצורה הצהרתית גם ברמת המחלקה ולא רק ברמת הפונקציה לדוגמא:

```
[PrincipalPermission(SecurityAction.Demand,
                    Authenticated=true)]
public class Foo {
  public void Bar() {  // caller MUST be authenticated
  }
  public static void Quux() {
    // caller MUST be authenticated
  }
```

ב ASP.NET ניתן גם לממש את בדיקת ההרשאות על הדפים עצמם לדוגמא:

```
<%@page language='c#'
        inherits='Foo' src='foo.cs'%>
<h1>If you see this you must be a Supervisor</h1>
```

כאשר הקוד מאחורי המחלקה Foo הוא המעניין מכיוון שבו נעשה שימוש בבדיקה הצהרתית:

```
[PrincipalPermission(SecurityAction.Demand,
                    Role="Supervisors")]
public class Foo : System.Web.UI.Page {  // ... }
```

ניתן גם להגדיר בקובץ ה Web.Config את בדיקות ההרשאה לסט שלם של דפים ע"י הגדרתם בספרייה מבודדת והגדרת קובץ ה Web.Config לספרייה זו בצורה הבאה:

```
<configuration>
  <system.web>
    <authorization>
      <allow roles='Supervisors'/>
      <deny users='*'/>
    </authorization>
```

```
  </system.web>
</configuration>
```

6. איומי אבטחת מידע

הקדמה

התקפות על רמת האפליקציה יכולות להיות מכוונות כלפי כל אחת משכבות המערכת: בשכבת הפרזנטציה (באמצעות התקפות על שרתי ה-Web או בממשקי המשתמש של המערכת), בשכבת האפליקציה (באמצעות התקפות המנסות לעקוף הגבלות על ביצוע פעולות ועוד) ועל שכבת הנתונים (על ידי התקפות שמטרתן להוציא מידע בצורה בלתי מורשית מבסיסי הנתונים או לבצע שינויים בנתונים).

בסעיפים הבאים מפורטים מספר סוגים של איומים אשר המערכות עלולות להיות חשופות אליהם במידה ולא יינקטו אמצעי האבטחה המתאימים ופיתוח הקוד לא ייעשה באופן מאובטח:

גניבת זהות בעקבות מדיניות סיסמאות לקויה

במידה ומדיניות הסיסמאות בארגון או באפליקציה מסוימת חלשה, כלומר – סיסמא קלה לניחוש, גם האפליקציה המאובטחת ביותר, תהיה פריצה בקלות יחסית, וזאת בשל ההזדהות חלשה. תוקף מיומן יכול להריץ כלים אשר נועדו לשבור סיסמאות, או לנסות ולנחש צירופים הגיוניים, ובכך לפרוץ למערכת ולהתחזות למשתמשים אחרים.

הזרקת שאילתות זדוניות (SQL INJECTION)

לצורך הצגת מידע דינאמי פונה האפליקציה אל בסיס נתונים, בין אם היא מבצעת זאת ישירות ובין אם היא מבצעת זאת דרך שרתי אפליקציה מתווכים. הקשר מול בסיס הנתונים מתבצע באמצעות שפת SQL . בתוך שאילתות ה- SQL מוטמעים בדרך כלל גם פרמטרים אשר מגיעים מהמשתמש (כגון, מחרוזת לחיפוש, שם משתמש וסיסמא...). על ידי בנייה מתוחכמת של הפרמטרים הללו, יכולים התוקפים במקרים מסוימים לבצע שאילתות בלתי חוקיות בבסיס הנתונים.

PARAMETER TAMPERING

בעיות אבטחת מידע רבות נגרמות כתוצאה מאי-וידוא קלט שמגיע ממשתמשים או מאובייקטים שאליהם אנו מתממשקים דבר המאפשר לתוקף לבצע שינוי בפרמטרים (Parameter Tampering) אשר יאפשר לו לבצע התקפות שונות על המערכת. תוקף יכול לשנות את הקלט כך שהוא לא יתאים למה שאנו מצפים לקבל. בהתאם למערכת והפונקציונאליות הספציפית שמשתמש בקלט זה, יכול התוקף לגרום לנזקים שונים החל מהשבתת המערכת, דרך ביצוע פעולות בלתי מורשות וכלה בהשתלטות מלאה על המערכת. במקרים מסוימים אנו מבצעים בדיקות על הקלט ברמת ה- Client מבלי לקחת בחשבון שתוקף יכול בקלות לעקוף בדיקות שמתבצעות ברמת הלקוח (Client).

נוהל פיתוח מערכות מאובטחות

מניעת שירות - DENIAL OF SERVICE

התקפת מניעת שירות (DoS – Denial of Service) היא התקפה שיכולה להשבית מערכת שלמה ע"י מניעת שירות מכל משתמשי המערכת או מחלק מהם. התקפה זה מתבצעת ע"י גרימת ניצול קריטי של משאבים בצד השרת. המשאבים יכולים להיות משאבי זיכרון, משאבי דיסק קשיח, משאבי מעבד ומשאבי רשת .

חריגת הרשאות

היכולת של משתמשים לחרוג מההרשאות המותרות להם היא אחת מבעיות האבטחה הקשות באפליקציות. מערכת ההרשאות של כל אפליקציה היא מורכבת ומשתתפים בה מספר רכיבים. כך למשל קיימות הרשאות ברמת מערכת ההפעלה אשר אחראיות על הגישה לקבצים ומשאבים של מערכת ההפעלה. ישנן הרשאות ברמת בסיס הנתונים אשר אחראיות על הגישה לטבלאות ואובייקטים בבסיס הנתונים וישנן הרשאות של משתמשים ברמת האפליקציה אשר מגדירות באלו תהליכים לוגיים יכול המשתמש לעשות שימוש.

כל מערכות ההרשאה הללו צריכות להתחבר למערכת אחת אשר דואגת לכך שמשתמשים לא יוכלו לחרוג מההרשאות שלהם בכל שלב של עבודה מול המערכת. הבעיה העיקרית היא שלא קל לשלב בין מערכות הרשאה אלו. כך למשל האפליקציה ניגשת למערכת הקבצים ולבסיס הנתונים ובדרך כלל בהרשאות נרחבות ביותר. במידה והמשתמש מצליח לנצל פרצה מסוימת באפליקציה הוא יכול לגשת למערכת הקבצים או לבסיס הנתונים ולבצע שם פעולות שונות של שליפה ושינוי מידע.

מנגנון ההרשאות צריך להיות מיושם בכל השכבות (שכבת הפרזנטציה, שכבת הלוגיקה העסקית, שכבת הנתונים) יש אפליקציות שמיישמות מנגנון זה בשכבת הפרזנטציה בלבד. באפליקציות מסוג זה מסתמכים שהמשתמש רק ילחץ על הקישורים והכפתורים שקיבל בתצוגה אך בפועל משתמשים זדוניים יכולים להשיג קישורים ישירים אל לב ליבה של המערכת וכך לעקוף בקלות את מנגנון ההגנה זה.

כמו כן, עצם העובדה שכל אחד מהרכיבים שתוארו מפותח בדרך כלל על ידי גורם אחר, לא מקלה על הבעיה. האפליקציה מפותחת על ידי מספר מפתחים ובסיס הנתונים מפותח על אנשי בסיס נתונים ובמרבית המקרים הקשר הרופף בין כל הגורמים יכול ליצור פרצות במעבר בין שכבה לשכבה.

טעויות קונפיגורציה

טעויות קונפיגורציה יכולות לגרור בעיות אבטחה קשות. טעות קונפיגורציה יכולות להופיע הן במוצרי התשתית (לדוגמה: שרת WEB) והן באפליקציה עצמה במידה והמתכנתים הכניסו אופציה לכך. במהלך התקפה זו מנסים התוקפים לנצל קונפיגורציות ברירת מחדל או ליקויים בהגדרות הקונפיגורציה של אותו רכיב על מנת לבצע פעולות זדוניות שונות במערכת.

"דלתות אחוריות" ואופציות DEBUG

בעת תהליך כתיבת הקוד של מערכות גדולות ומסובכות, נוטים לפעמים המתכנתים להשאיר "דלתות אחוריות" (Backdoors) אשר יכולות לאפשר להם גישה לנקודות שונות במערכת ללא ביצוע הזדהות מסודרת, או לאפשר להם לבצע פונקציונאליות מסוימת שהמערכת לא אמורה לאפשר למשתמשים. במקרים מסוימים שוכחים המתכנתים להסיר את הדלתות האחוריות הללו בסוף התהליך וגורם זדוני אשר גילה אותם יכול לנצלם לרעה. במקרים אחרים, מושארות הדלתות האחוריות במתכוון מתוך כוונות זדוניות ולכן יש לבצע בדיקות של כל קוד

לפני העברתו לייצור ולבחון שאין בו "דלתות אחוריות" או אפשרויות DEBUG אשר עלולות לאפשר פגיעה במערכת או השתלטות עליה.

BUFFER OVERFLOW

חוצץ (Buffer) הוא אזור בזיכרון המחשב המוקצה לאחסון רצף של נתונים בכלל, ולאחסון זמני של נתונים בפרט. סוג הנתונים הנשמר בחוצץ הוא בד"כ מחרוזות תווים, שורות קלט מקבצים, הודעות תקשורת וכו'. בד"כ משמש החוצץ לשמירת הנתונים בדרך מנקודת הקלט (מקלדת, דיסק, תקשורת) אל נקודת העיבוד. במקרים רבים עובר רצף תווי הקלט דרך מספר חוצצים עד לנקודת העיבוד. הסיבה לכך היא בראש ובראשונה אופן הבניה המודולרי של מערכות תוכנה גדולות.

גלישה היא מצב בו התוכנה כותבת אל חוצץ יותר נתונים מכפי גודלו. מכיוון שהחוצץ הנו בעל גודל מוגדר וסופי, הרי שכתוצאה מפעולה זו גולשים הנתונים העודפים אל מחוץ לחוצץ ודורסים את התאים הנמצאים מייד אחרי החוצץ בזיכרון.

גורם המעוניין לתקוף את המערכת נדרש לגלות אילו מנתוני הקלט אינם נבדקים כנגד גלישה. לאחר שהתוקף זיהה את הנקודה הבעייתית כל שנותר לעשות הוא לנחש כמה ארוך צריך להיות הקלט כדי ליצור גלישה ואז להשתמש בטכניקות ידועות כדי לנצל אותה. בכל אחד מרכיבי התוכנה, ללא קשר למקורם, עלולות להימצא בעיות גלישה.

הרכיבים המועדים ביותר להמצאות גלישות הם רכיבי התוכנה המורכבים ביותר כגון שירותי מערכת ההפעלה או שירותי תקשורת מורכבים. כמו כן ישנה רגישות גבוהה לבעיות גלישה בתוכנות אשר נבנו בשיטה של טלאי על טלאי. מלבד רכיבי התשתית המוכרים בהם מתעוררות בעיות גלישה, הרי שגלישות יכולות להיווצר גם באפליקציות שפותחו בארגון עצמו או אפליקציות שפותחו עבור הארגון ע"י צד שלישי. כמו כן, בהחלט ייתכנו בעיות גלישה ברכיבים בסיסיים יותר כמו רכיב ה- BIOS של המחשב (למעשה, התגלו בעיות כאלה בעבר). גלישת חוצצים יכולה לאפשר בידי התוקף לפגוע בזמינות המערכת או להשתלט על המערכת.

עקיפה לוגית - FLOW BYPASS

בשכבת הלוגיקה העסקית, קיימים לרוב תהליכים המורכבים ממספר שלבים. שלבים אלה יוצרו על מנת לספק למשתמש תהליך עבודה. אחת מההתקפות המהותיות ביותר כנגד מנגנון מסוג זה, היא לנסות ולעקוף את הלוגיקה והתהליכים אשר להם התכוון מתכנן במערכת. דוגמא טובה לכך היא תהליך פתיחת חשבון המחייב מספר שלבים: הראשון הקלדת נתוני החשבון והשני אישור מסגרת האשראי בחשבון. משתמש זדוני, ינסה לעקוף את תהליך אישור מסגרת האשראי, ולעבור לתהליך הבא אחריו ישירות, תוך דילוג על שלב מסוים מתוך התהליך הסדור. לכן, מפתחי המערכת צריכים למנוע מצבים כאלה ולוודא זרימה נכונה של התהליך ללא יכולת לדלג על שלבים.

נפילה לא מאובטחת של אפליקציות

אפליקציות נופלות כתוצאה מבאגים שונים באפליקציה או בתקלות בתשתית עליה האפליקציה רצה. חלק מבעיות האבטחה של מערכת נחשפות רק כאשר המערכת נופלת. המערכת יכולה ליפול לגמרי או חלקית, תוך כדי כך שחלקים מסוימים במערכת לא מתפקדים וחלקים אחרים כן מתפקדים. בעת נפילת המערכת עלול להיווצר מצב שבו פונקציונאליות מסוימת האחראית על אבטחת המידע איננה עובדת וכתוצאה מכך המערכת נותרת ללא אבטחה נאותה. כמו כן נפילה יכולה לחשוף את הארכיטקטורה של המערכת ע"י הצגת הודעות שגיאה מפורטות .

יירוט התעבורה (MAN IN THE MIDDLE)

תעבורת המידע בין מודולי וממשקי המערכת השונים עוברת בדרך כלל דרך רכיבי תקשורת רבים אשר לא כולם בשליטתנו. בכל רכיבי התקשורת, במידה ולא מתקיימת הצפנה של המידע וזיהוי חד ערכי של הגורמים המורשים למידע, קיימת יכולת של גורמים עוינים המאזינים לתווך התקשורת ולנתונים העוברים בין המערכות ובין המודולים שלהם לגנוב את המידע או לשנותו. כמו כן אותם גורמים יכולים להקים שרת מתחזה הדומה לשרת המקורי (מה שנקרא Phishing Attack) וכך לגנוב פרטי הזדהות או מידע רגיש העוברים בדרך או אל השרת המתחזה.

ניתוח פרוטוקולים

מערכות המבוססות שרת-לקוח נוטות לבסס חלק מאבטחת המידע של המערכת על בדיקות שמתבצעות ברמת יישום הלקוח. במקרה זה, על ידי ניתוח הפרוטוקול שעובר ברשת בין הלקוח לשרת, ניתן במקרים רבים לעקוף את מנגנון האבטחה של המערכת ולשלוח פקודות בלתי מורשות למערכת.

פרצות במוצרי צד שלישי

במקרים רבים האפליקציה מותקנת על שרת אשר מהווה מוצר מדף או משתמשת ברכיבי תשתית שונים מתוצרת צד שלישי. מדי פעם, עלולות להתגלות באותם מוצרי מדף, רכיבי תשתית ורכיבים מצד שלישי פרצות אבטחת מידע ידועות אשר עלולות לשמש תוקפים על מנת להשיג מידע על המערכת הנתקפת ועל מנת לבצע פעולות בלתי מורשות בה. במהלך התקפה זו מנסים התוקפים לנצל את הפרצות הידועות המפורסמות בדרך כלל באתרי WEB ברחבי האינטרנט שנמצאו בתוכנות התשתית של המערכת ולפרוץ באמצעותם ליישום או במקרים מסוימים לקבל שליטה מלאה על המערכת.

נעילת מוות (DEAD LOCK)

נעילת מוות (Dead Lock) היא מצב שבו תהליך A נועל קובץ א' ואז מנסה לקרוא משהו מקובץ ב'. בו בזמן תהליך B נועל קובץ ב' ואז מנסה לקרוא משהו מקובץ א'. שני התהליכים לא ישחררו את הקבצים שהם נעלו עד שהם לא יצליחו לקרוא מהקובץ. זוהי נעילת מוות שלא תשתחרר לעולם. כאשר מצב זה אפשרי משתמש בודד יכול לגרום לנעילה של משאב מסוים ובכך להשבית פונקציונאליות עבור משתמשים אחרים.

מרוצים - RACE CONDITIONS

חשוב לזכור שאפליקציות לא רצות בתהליך אחד ולא עובדות מול משתמש אחד. השימוש במספר תהליכים במקביל עלול לגרום לבעיות אבטחת מידע שונות כגון מרוץ. מרוץ הוא מצב שבו התוצאה של פעולה מסוימת תלויה בשני תהליכים או יותר ונקבעת על ידי הסדר שבו תהליכים אלו אמורים להתבצע. מערכות הפעלה לא מבטיחות לנו ששורת קוד ב' תתבצע מיידית אחרי שורת קוד א' מבלי ששום דבר אחר יתבצע ביניהן. תזמון תהליכים יכול לגרום למצב שבו שורת קוד א' תתבצע ולאחריה יפעל תהליך אחר לגמרי לפרק זמן מסוים ואז יחזור התהליך הראשון ותתבצע שורת קוד ב'. מצב זה יכול לגרום לבעיות אבטחה שונות אם הוא לא נלקח בחשבון מלכתחילה. דוגמא פשוטה היא כאשר שורת קוד א' יוצרת קובץ במערכת ההפעלה ואז שורת קוד ב' משנה את ההרשאות של הקובץ כך שגורמים לא מורשים לא יוכלו לגשת לקובץ. מצב שבו תהליך אחר מופעל בין שורת קוד א' לשורת קוד ב' יכול לאפשר לאותו תהליך לכתוב לאותו קובץ או להתחבר אליו כך שגם לאחר הניסיון של התהליך הראשון לשנות את ההרשאות, עדיין תהיה לתהליך השני אפשרות גישה לקובץ.

איומים ייחודיים לאפליקציות WEB

להלן תיאור ההתקפות והסיכונים העיקריים הבאים לידי ביטוי בעיקר באפליקציות בסביבת WEB:

6.1.1. מניפולציות שדות Hidden

בהתקפה זו מנסה התוקף לשנות ערכים שונים המגיעים לדפדפן "מוחבאים" על ידי שימוש בשדה Hidden. במקרים מסוימים האפליקציה עלולה להסתמך על ערכים אלו ועל ההנחה כי הם נסתרים מהמשתמש ואין ביכולתו לשנותם. הנחה זו איננה נכונה מאחר ומשתמש בעל כוונות זדוניות יכול לבצע שינויים מושכלים של ערכים אלו ולשלוח אותם בחזרה לשרת. כך לדוגמא, אם האפליקציה השתמשה ב-Hidden Fields על מנת לאחסן את מספר הלקוח ועל פיו מעניקה הרשאות, התוקף יכול לשנות את מספר הלקוח ולקבל במקרים מסוימים גישה לחשבונות של משתמשים אחרים.

6.1.2. הרעלת Cookies

שימוש במנגנון ה Cookies מאפשר לאפליקציה לשמור מידע במחשב המשתמש. מערכות ואפליקציות בלתי מאובטחות ששומרות נתונים זמנים הרלוונטיים לאותו משתמש ב Cookie כגון מחיר של המוצר שהוא קנה עכשיו ועל פיו מבצעות החלטות במערכת או מתבססות על נתונים אלה ללא בדיקה לתקינותם עלולות לאפשר פרצות ביישום ולאפשר התקפה על המערכת. בהתקפה זו התוקף מנתח את ה- Cookies שהאפליקציה שולחת למשתמש ובודק האם ניתן לבצע שינויים ב – Cookie אשר יאפשרו לו גישה למידע של משתמשים אחרים או לבצע פעולות שאין לו הרשאה לבצע.

יש להבין כי נתונים הנשמרים ב Cookies הינם קלטים למערכת לכל דבר והמערכת צריכה לבדוק את תקינותם בצד השרת. יש לצאת מנקודת הנחה כי גורמים זדוניים ינסו לשנות מידע זה על מנת לתקוף את המערכת או לפרוץ אליה.

כמו כן, חלק מהאפליקציות יכולות להצפין את ה- Cookies לפני העברתם למשתמש אך אם נשמר בהם מידע הנוגע לזיהוי או להרשאות עלול להיווצר מצב בו גורם שיגנוב Cookie כזה יוכל לבצע Replay Attack ולהשתמש ב Cookie זה על מנת להתחזות למשתמש חוקי או לקבל הרשאות גבוהות יותר.

לכן, אין להסתמך על נתונים הנשמרים ב Cookies ללא בדיקה לתקינותם, ולהשתמש בהם בצורה מושכלת.

6.1.3. Forceful Browsing

לכל אפליקציה יש נקודת גישה מרכזית אחת או יותר. בדרך כלל מדובר בדף ההזדהות של האפליקציה דרכו חייב המשתמש לעבור. לאחר למידה מדוקדקת של האתר, ניתן לפעמים לגשת ישירות לדפים בשרת שהמתכנתים לא תכננו שניגש אליהם ישירות, ללא מעבר דרך דף ההזדהות. במידה ובאותם דפים שאליהם אנו ניגשים ישירות לא מתבצע תהליך מחודש של בדיקת ההזדהות של המשתמש, אזי ניתן יהיה לעקוף את מנגנון ההזדהות של המערכת ו/או לקבל גישה לקבצים ונתונים אשר איננו מורשים לראות.

6.1.4. Cross Site Scripting

התקפה זו הנפוצה כיום בעיקר ביישומי WEB מאפשרת לתוקף לנצל את אי בדיקת תקינות הקלט והפלט ברמת ממשק התצוגה של האפליקציה על מנת להריץ קוד זדוני במחשב המשתמש או במערכות משיקות המשתמשות בקלט הזדוני. הפעולות אותן ניתן לבצע כוללות בין השאר:

- **גניבת זהות** – גניבת משתנה ה SESSION המשמש לזיהוי המשתמש מול המערכת ושליחתו לתוקף אשר יוכל להשתמש בו על מנת לבצע התחזות למשתמש לגיטימי ולבצע פעולות בשמו.
- **ביצוע הונאות** – שתילה של הודעות כוזבות אשר נראים כאילו הגיעו מהמערכת אשר ישכנעו את המשתמש להזין פרטים רגישים (כגון סיסמה , פרטי כרטיס אשראי) תוך ניצול בטחונו במערכת ואשר יאפשרו שליחת המידע הרגיש לתוקף או ניצול המידע לרעה.
- **שינוי מראה אתר** – ביצוע Defacement לאתר המערכת תוך פגיעה תדמיתית בארגון.
- **שתילת סוסים טרויאניים ותוכנות זדוניות** במחשב המשתמש תוך ניצול פרצות בדפדפן המשתמש.

קיימת ספריה בתשתית . Net בשם antiXss המספקת פונקמיונאליות להגנה כנגד התקפת XSS.

6.1.5. CSRF Cross-Site Request Forgery

- **איך זה עובד?**

נפוץ מאוד באפליקציות web

כל פונקציה ללא הגנה ייעודית לנושא פגיעה

- **איך הלקוחות מותקפים?**

הלקוח פותח דף אינטרנט או מקיש על לינק באי מייל שיש בו המתקפה

- **מה תוקף יכול לעשות?**

- Tags

<img src="https://bank.co.il/fn?param=1">

<iframe src="https://bank.co.il/fn?param=1">

<script src="https://bank.co.il/fn?param=1">

- Autoposting Forms

```
<body onload="document.forms[0].submit()">
<form method="POST" action="https://bank.co.il/fn">
  <input type="hidden" name="sp" value="8109"/>
</form>
```

- XmlHttpRequest
  - Subject to same origin policy

- **מה לא לעשות כדי למנוע**

שליחת בקשות רק POST

- עוצר מתקפות מבוססות לינק פשוטות (IMG, frames,.)
- לא עוצר בקשות בהם יש FRAME וסקריפט…

בדיקת Referer

- יש לקוחות שלא משתמשים ב Referer
- לא תמיד הוא קיים

- **דוגמאות מהשטח**

- **מה כן לעשות**
  - CAPTCHA
  - הזדהות מחודשת (Re-Authentication)
    - מבוסס סיסמא
      - התוקף חייב לדעת את הסיסמא של הלקוח
    - חתימה חד פעמית(One-Time Token)
      - התוקף חייב לדעת מהו הטוקן כרגע
      - הגנה חזקה מאוד
  - חתימה יחודית (Unique Request Tokens)
    - התוקף צריך לדעת חתימה ייחודית לגולש ספציפי עבור Sessionספציפי
    - ההנחה כי החתימה מוצפנת ואינה ניתנת לפענוח
    - **/accounts?auth=687965fdfaew87agrde …**

בנוסף קיימת ספריה המספקת פונקציונאליות ומענה להתקפה זו בשם OWASP CSRFGuard Project

7. שינויים באפליקציה ותחזוקת הקוד

ניהול הקוד והאפליקציה

- יש לדאוג להפרדה מוחלטת בין סביבת הייצור לסביבת הפיתוח וסביבות הניסוי.
- אין לאפשר לתוכניתנים גישה לבסיסי נתונים בסביבת הייצור.
- מומלץ לחתום את הקוד שיצרת באמצעות אלגוריתמים קריפטוגרפים.
- בכל עלייה של המערכת יש לוודא את החתימה על הקוד.
- יש לדאוג לחסום את הגישה לממשקי הניהול של האפליקציה מפני גורמים לא מורשים.
- יש לדאוג לתעד כל כניסה לממשקי הניהול של האפליקציה וכל שינוי שהתרחש באפליקציה.
- יש לצמצם למינימום את הסיסמאות לממשקי הניהול ולתשתיות המערכת.

8. נספח – כלי מתאר (Check List)

הרשימה להלן מהווה אוסף של כללים יסודיים לתכנון מערכת מאובטחת. אין בה כדי להחליף את העקרונות עליהם דובר בסעיפים הקודמים, והיא מהווה אוסף שלעולם יהא חלקי ולא שלם. ככלל, הבנה של עקרונות האבטחה בסעיפים הקודמים נותנת בידי מתכנן המערכת את הכלים לתכנן מערכת מאובטחת בסביבות שונות ובשפות שונות, וכוחם יפה גם למערכות וטכנולוגיות עתידיות.

מס"ד	פרמטר שנבדק	בוצע/לא בוצע	הערות
1	אורך סיסמא מינימלי (6 תווים לפחות).		
2	איסור על סיסמאות קלות לניחוש: סיסמאות מעורבות: תווים +ספרות. רצוי סיסמאות Case Sensitive.		
3	תאריך תפוגה לכל הסיסמאות (של כ- 30 עד 90 יום לפי רמת האבטחה)		
4	היסטוריה של סיסמאות-כדי לבדוק שאין מחזור לסיסמאות ישנות.		
5	בדיקת דמיון לסיסמאות עבר (למשל משה1, משה2, משה3).		
6	נעילת מערכת והודעה למנהל המערכת לאחר 3/5 ניסיונות כושלים.		
7	איסור על כניסת משתמש משתי תחנות בו זמנית. מצב שכזה יגרום למשתמש שביצע log-in ראשון לצאת באופן כפוי מהמערכת.		
8	הסיסמא צריכה להיות נסתרת בממשק המשתמש (למשל ע"י כוכביות).		
9	אין לדווח על סיסמא שגויה תוך כדי ההקלדה, אלא רק בסיום הקלדת כל תווי הסיסמא + שם המשתמש.		
10	משלוח סיסמאות על גבי התווך (במקרה של הזדהות רשת) צריך להתבצע מוצפן, ע"י מנגנון Challenge-response או ע"י הקמת תווך מוצפן עם המערכת, ומשלוח הסיסמא על גבי תווך זה (בדומה לאופן העבודה ב – SSL).		

11	אחסון הסיסמאות ע״י המערכת צריך להיות מוצפן או Hashed . אין לאפשר למשתמשים יכולת קריאה למערכת הסיסמאות ובוודאי ללא יכולת כתיבה. הצפנת הסיסמאות צריכה למנוע ממנהל המערכת הזדהות כאחד המשתמשים, גם אם יש ביכולתו להגיע לסיסמאות המאוחסנות. אין לאחסן בשום אופן סיסמאות בתוך קוד המערכת.		
12	על המשתמש לראות ״ עולם״ שמתאים לפרופיל ההרשאות שלו. אין לחשוף בפני המשתמש יכולות של המערכת שאינן פתוחות לגביו. מבחינה זו מתבצעות פעולות המשתמש ב – Sand Box.		
13	מערכת ההרשאות צריכה להיות מרכזית במערכת, וצריכה לנהל את כל הפניות למשאבים. אין לאפשר גישה למשאבים משום נקודה אחרת במערכת, אלא דרך מערכת ההרשאות (פרוצדורות גישה יחידה בתוכנה באופן מקביל לפונקציית SeAccessCheck במערכת ההפעלה WinNT).		
14	מערכת ההרשאות חייבתלהתנהל מחוץ להישג ידו של המשתמש. במערכות Client-server, ניתן לממש עקרון זה במספר אופנים :(בחר אחד מהאופנים) מימוש מערכת בקרת הגישה ע״י בסיס הנתונים. באופן זה מוגדרים כל המשתמשים במערכת על בסיס הנתונים, ועבורם מוגדרים Rolls שמבצעים את בקרת הגישה. העברת רכיב הבקרה (כולל הגישה למשאבים ובדיקת ההרשאות) לרכיב שרת (service). באופן כזה נמצאת מערכת הבקרה מחוץ להישג ידו של המשתמש. שיטה זו עדיפה על פי רוב על ניהול ההרשאות בבסיס הנתונים משום שהיא מאפשרת הגדרות בקרה גמישות ברמת היישויות הלוגיות של המערכת, ולא ברמת טבלאות בבסיס הנתונים.		
15	בקרת הגישה לא נמצאת בצד ה – Client למעט בשיטות Tamper resistance בחומרה (למשל שיטות ההקשחה של כרטיסים חכמים) לפיכך מערכת שבה בקרת הגישה נמצאת ב – Client אינה בטוחה.		

16	אין לאפשר גישה פיזית למערכת בקרת הגישה (כלומר לשרת במערכות Client-server. מערכות בהן השרת מגיש פיזית אינן בטוחות.		
17	רצוי שתחנת הקצה של המשתמש תעבור הקשחה ברמת קיבוע תצורה (ע"י Group Policy למשל), כך שלא ניתן יהיה להריץ בתחנת הקצה יישומים שאינם מורשים. בקרת תצורה על תחנת הקצה לא נמצאת באופן ישיר בתחום אחריות היישום, אבל יכולה להשפיע על אבטחת המערכת לדוגמא תיתכן התקיפה הבאה: משתמש מתקין בכוונה או בשוגג תוכנה עויינת (למשל סוס טרויאני או וירוס) על תחנת הקצה. התוכנה העויינת מבצעת פעולות כגון: רישום סיסמאות משתמשים, "רכיבה" על Session קיים של משתמש מזוהה וביצוע פעולות בשמו, הדבקת השרת או צמיעת גישה אליו ע"י תקשורת מוגברת וכדומה.		
18	קיום מנגנון התרעה על חריגות בזמן אמת		
19	רישום והצגת שימוש אחרון במערכת		
20	במידת האפשר, רצוי להשתמש בספריות הצפנה קיימות: OpenSSL,CryptoAPI,++ Crypto, PGP ואחרות. ספריות הצפנה רבות מסופקות חינם בתוספת קוד מקור, והן נבחנו לאורך השנים ע"י מומחים.		
21	שי להשתמש ב- TLS/SSLv3 עם אורך של 224 ביט לפחות.		
22	יש להשתמש ב Store procedures ולא ב שאילתות Sql דינמיות.		
23	קליטת נתונים ממערכות חיצוניות צריכה לכלול בדיקות לוגיות, כדי לוודא שהקלט מתאים להנחות המערכת הקולטת. מבחינה זו יש להתייחס לקלט ממערכת משיקה באותה רמת של חשד כאל קלט ממשתמש. צריך להניח שלא בוצעה שום "בדיקת שפיות" על הקלט ולכן יש לוודא בדיקות של שדות, רשומות, וכדומה, לפי ההנחות הלוגיות של המערכת המקבלת.		
24	קליטת נתונים צריכה להתבצע ע"י מנגנון Roll-Back, על מנת לאפשר חזרה למצב ידוע באם הנתונים0 מתגלים כשגויים. נתונים שגויים צריכים להיות מדווחים לתוכנה השולחת (אם קיים ממשק להעברת שגיאות), ולמנהל המערכת, על מנת לברר את הסיבה.		

25	בהעברת נתונים אסינכרונית, יש לוודא שנתונים ישנים אינם דורסים נתונים חדשים יותר, ניתן לביצוע ע"י ספרור רץ של הנתונים, אם באמצעים מתוחכמים יותר כגון: Timestamps.		
26	אמצעי דיווח חזקים ונגישים למערכת ה – Log: מערכת Log ללא יכולת דיווח נגישה, ובחתכים שימושיים, הינה חסרת ערך לחלוטין.		
27	במערכות מסוימות, רצוי לבצע גם מעקב אוטומטי על ה – Log, במטרה לאתר אנומאליות. אירועים חריגים אלה יש לדווח ע"י מערכת ה – Log.		
28	כל גישה ל – log צריכה אף היא להופיע ב – log עצמו.		
29	ניהול ה – Log, רצוי שיתבצע על גבי שרת נפרד, שבו אפילו למנהל המערכת אין גישת כתיבה. (למשל CDR). יש להוסיף חתימה דיגיטלית לקובץ ה – Log.		
30	כל קלט מהמשתמש צריך להתאים להנחות המערכת. יש לתעד הנחות אלה בקוד ובמסמכי התיעוד, ולוודא שהנחות אלה נכפות ע"י המערכת. למשל: אם המשתמש נדרש להקליד כתובת האורך של עד 50 תוים. יש לוודא שלא יתבצע קלט למערכת, עד שנבדק בוודאות שהקלט אכן מתאים להנחה זו. אין בשום מקרב לנסות לסנן קלטים לא מתאימים על פי רשימת שגויים: ההנחה היא שקל יותר ובטוח יותר לאפיין מה מותר ולאסור כל מה שלאף מאשר המקרה ההפוך. כלומר יש לבצע בדיקת קלטים ע"י ה- White List.		
31	קריאות לשירותי מערכת ההפעלה, או למערכות תשתית אחרות, צריכות להתבצע עם בדיקת שגיאות מלאה. לעולם אין להניח שפעולה בוצעה, אם לא התקבל אישור מהמערכת המבצעת (למשל הקצאת זיכרון, פתיחת קובץ, הקצאת משאבי סנכרון וכדומה).		
32	נהלים: לא ניתן לנהל סביבת אבטחה ללא אכיפת העקרונות, ע"י מנהלי המערכת והמשתמשים. פתרונות טכנולוגיים טובים ככל שיהיו לא יכולים לתת מענה לדו"ח חסוי שהושאר על שולחן העבודה, למשתמשים שמשתפים סיסמא, למנהלי מערכת שאינם מעיינים ב – log, ולבעיות דומות. אבטחת המערכת צריכה להביא בחשבון נהלים מתאימים שיסייעו באכיפת מדיניות האבטחה.		
33	אבטחה פיזית: מרבית המערכות אינן עומדות בתקיפה פיזית של הנתונים. למרות שגם נושא זה אינו בתחום אחריות מתכנן היישום,		

	יש להתריע בפני הלקוח על בעיות פיזיות, שהופכות את מערכות ההגנה ללא רלוונטיות.		
34	מערכות הפעלה: מערכות הפעלה מסוימות אינן מהוות פלטפורמה מתאימה למימוש מערכת מאובטחת (למשל Win9x). מערכות הפעלה שלא הוקשחו כראוי, יאפשרו מעקף למערכת הבקרה של היישום ללא שליטה של המתכנן. בסיס הנתונים שמאפשר גישה ישירה, שלא דרך שכבת היישום, לא מאפשר הקמת מערכת בקרת גישה על הנתונים.		
35	אבטחה פיזית: מרבית המערכות אינן עומדות בתקיפה פיזית של הנתונים. למרות שגם נושא זה אינו בתחום אחריות מתכנן היישום, יש להתריע בפני הלקוח על בעיות פיזיות, שהופכות את מערכות ההגנה ללא רלוונטיות.		
36	מערכות הפעלה: מערכות הפעלה מסוימות אינן מהוות פלטפורמה מתאימה למימוש מערכת מאובטחת (למשל Win9x). מערכות הפעלה שלא הוקשחו כראוי, יאפשרו מעקף למערכת הבקרה של היישום ללא שליטה של המתכנן. בסיס הנתונים שמאפשר גישה ישירה, שלא דרך שכבת היישום, לא מאפשר הקמת מערכת בקרת גישה על הנתונים.		
37	במקרה שיש שגיאה יש להציג הודעה אחידה ומוכנה מראש.		
38	ניהול Session בצורה מאובטחת שכוללת מנגנון לניתוק אוטומטי לאחר זמן מסוים והשמדת ה- Session בזמן ה- LogOut.		
39	ה- WS משתמש במנגנוני ה- WS Security ו XML Encryptions		