נוהל פיתוח מערכות מאובטחות

OPEN EMR

משרד הבריאות

ניהול שינויים

תאריך	מחבר	גרסה	שינוי
23/03/2016	חברת אבנת	1.1	כתיבת הנוהל
24/03/2016	תמיר פלדמן	1.2	התאמה לתקן ISO 27799
27/03/2016	שי אמיר	1.3	אישור הנוהל

כללי

1.1. מבוא

מסמך זה מפרט הנחיות פיתוח מאובטח עבור מערכת קלינקל בה עושים שימוש ב - Open EMR, שמבוססת PHP.

1.2. מטרת הנוהל

מטרת המסמך הינה להנחות מתכנתים, ראשי צוותים ומנהלי פרויקטים, על עקרונות הפיתוח המאובטח ועל אופן כתיבת קוד אשר בנוי על פי עקרונות אלו.

1.3. אחריות

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

2. הקדמה

2.1. מערכת Open EMR

מערכת Open EMR הינה מערכת Open Source. המערכת משמשת עבור ניהול רשומות רפואיות, מרפאות וכו. המערכת מאפשרת לנהל רשומות רפואיות, ניהול זמנים, ניהול חשבונות ועוד.

2.2. רקע טכנולוגי

מערכת Open EMR עושה שימוש ב - PHP, אשר רץ על-גבי שרת Web מסוג Apache. לרוב נעשה שימוש בבסיס נתונים מסוג mySQL עבור שמירה ואגירת המידע העסקי של המערכת (אנשי הפיתוח של מערכת קלינקל שוקלים לעשות שימוש בבסיס נתונים מסוג Maria DB).

מערכת Open EMR מספקת מספר מנגנוני אבטחת מידע, כמו: הזדהות, הרשאות, הצפנה ועוד. יחד עם זאת, מנגנוני האבטחה שמסופקים ע"י Open EMR אינם מלאים, וחשוב לשים לב למספר נקודות חשובות בעת פיתוח האפליקציה אשר מפורטות בפרק מספר 4.

כאמור, Open EMR נכתבה בשפת PHP ולכן תוספות של מנגנוני אבטחת מידע למערכת אשר מובאים בהמשך מסמך זה, נכתבו והודגמו בשפת PHP.

2.3. מבנה המסמך

פרק מספר 3 - מפרט מספר איומי אבטחת מידע באפליקציות Web.

פרק מספר 4 - מפרט מספר עקרונות פיתוח מאובטח אשר מומלץ ליישם עבור מערכת קלינקל.

כמו כן, ישנם מספר נספחים אשר מספקים מידע נוסף אודות קלטים מסוכנים אשר יש לסננם והגדרות אבטחה שמומלץ ליישם בקובץ php.ini.

3. איומי אבטחת מידע

3.1. 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.

3.2. 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.

3.3. 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.

3.4. 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 שמשתמשת 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). לכן מומלץ לבדוק את כלל הקלטים, באופן מלא.

3.5. 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 זו מתבצעת ע"י גרימת ניצול קריטי של משאבים בצד השרת. המשאבים יכולים להיות משאבי זיכרון, משאבי דיסק קשיח, משאבי מעבד ומשאבי רשת.
ההתקפה יכולה לנבוע ממספר גורמים, ביניהן – הורדה והעלאה של קבצים גדולים, או הפקת דו"חות ארוכים הגורמים לניצול רב של משאבי רשת.
להלן תיאור הבעיות אשר יכולות לגרום למניעת שירות במערכת ואמצעי ההתגוננות שיש לנקוט על מנת להתמודד איתן:

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.

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.

3.6. 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 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.

כמו כן, עצם העובדה שכל אחד מהרכיבים שתוארו מפותח בדרך כלל על ידי גורם אחר, לא מקל 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.

3.7. 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.

3.8. Backdoors ו - 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.

3.9. 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.

3.10. 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) ועל ידי כך לגנוב פרטי הזדהות או מידע רגיש העובר בדרך או אל השרת המתחזה.

4. עקרונות הפיתוח המאובטח

4.1 הקדמה

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

4.2 בדיקת תקינות קלטים - Input Validations

4.2.1 כללי

בדיקות קלט אשר מתקבל ממשתמשים הוא אחד מהיסודות החשובים ביותר של אבטחת מידע בתחום האפליקטיבי והמקור העיקרי לבעיות רבות 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 ואין 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 ל-Cross Site Scripting או ל- Script Injection (document,/,(,),<.> וכו').

4.2.2. 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.

4.2.3. בדיקות קלטים ב - PHP

בדיקות קלטים ניתן לעשות באמצעות הפונקציה המובנית: filter_var.

באמצעות פונקציה זו ניתן לבצע:

1) בדיקות קלטים - בדיקת הקלט לפי פורמט מסויים.
2) Sanitize - הסרת תווים בלתי חוקיים מ - String.

דוגמא 1 - בדיקות קלטים באמצעות Regular Expression:

```
$nameRegex = '/[a-zA-Z\s]/';

$value = 'john';

if (!filter_var($value,FILTER_VALIDATE_REGEXP,$ nameRegex)){
    $message = 'wrong';
    echo $message;
}else{
    $message = 'correct';
    echo $message;
}
```

דוגמא 2 - ביצוע Sanitize:

```
<?php
    if (isset($_POST[emailAddress])) {
        echo filter_var($_POST['emailAddress'], FILTER_SANITIZE_EMAIL);
    }

    if (isset($_POST['homepage'])) {
        echo filter_var($_POST['url'], FILTER_SANITIZE_URL);
    }
?>
<form name="form" method="post" action="sanitize.php">
    Email:
     <input type="text" name="email" value="<?php echo
         $_POST['emailAddress']; ?>”/>
    Home Page:
    <input type="text" name="url" value="<?php echo $_POST[url]; ?>"/>
    <input type="submit" />
</form>
```

הערה: מידע נוסף על שימוש בפונקציה filter_var ניתנה למצוא בקישור הבא:

http://php.net/manual/en/filter.filters.validate.php

4.3. Authentication

4.3.1. כללי

זיהוי המשתמשים במערכת הינו אחד ממנגנוני אבטחת המידע החשובים ביותר. בשלב זה המשתמש "מזדהה" בפני המערכת ואימות זיהוי זה ייקבע 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.

4.3.2. Authentication Mechanisms in Open EMR

Open EMR מאפשר לעשות שימוש בשני מנגנוני הזדהות:

1) מנגנוני הזדהות תשתיתיים, למשל מול שרת ה - Active Directory.
2) לעיתים כאשר הזיהוי ברמת התשתית אינו מספק, נוצר צורך לממש מנגנון זיהוי ייחודי לאפליקציה. מנגנון זה ממומש על ידי המתכנתים ברמת הקוד של האפליקציה. Open EMR מציע הזדהות אפליקטיבית מובנת במערכת, ע"י שמירת פרטי ההזדהות של משתמשי המערכת בבסיס הנתונים.

4.3.3. דגשים

במידה ועושים שימוש במנגנוני הזדהות אפליקטיביים מובנים ב - Open EMR, חשוב לשים לב לנושאים הבאים:

- שמירת סיסמאות בבסיס הנתונים: מומלץ לשמור את פרטי ההזדהות בצורה מגובבת. מומלץ לעשות זאת ע"י שימוש בפונקציות Hash חזקות, כדוגמת: SHA256bits.
  - ניתן לעשות זאת ע"י שימוש בפונקציה המובנית *password_hash()*.
  - דוגמא לשימוש:

```
$hash = password_hash($password, PASSWORD_DEFAULT);
```

הערה: מידע נוסף ניתן למצוא בקישור הבא -

http://php.net/manual/en/faq.passwords.php

- הגנה על סיסמאות בתווך התקשורת: מומלץ להצפין את תווך התקשורת בו עוברים פרטי ההזדהות. מומלץ לעשות שימוש ב - TLS v1.2.

4.4. Password Policy

4.4.1. כללי

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: אורך מינמאלי של 6 תווים ואורך מקסימלי של 20 תווים.

ב. אכיפת סיבוכיות הסיסמא: מקובל שסיסמת משתמש תכיל לפחות שלוש מתוך ארבע קבוצות התווים הבאות: אותיות קטנות [a-z], אותיות גדולות [A-Z], ספרות [0-9] ותווים מיוחדים [!,@,#,$,%,^].

ג. החלפת סיסמה: יש לאכוף החלפת סיסמה אחת לתקופה מוגדרת.

ד. היסטוריית סיסמאות: יש לאכוף אי שימוש בחזרה על סיסמאות שהיו בשימוש בעבר. מומלץ למנוע החלפה לשבע 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.

ו. נעילת חשבון המשתמש: מומלץ לנעול את חשבון המשתמש לאחר מספר ניסיונות כושלים בהקשת פרטי ההזדהות לפרק זמן מוגבל. מומלץ להגדיר נעילה לאחר כחמישה ניסיונות כושלים. יש לשחרר את הנעילה לאחר פרק זמן שהוגדר (למשל לאחר חצי שעה) על מנת לא לגרום למניעת שירות.

4.4.2. מנגנון שינוי סיסמה

להלן מספר דגשים לגבי יישום של מנגנון שינוי סיסמה:

- תהליך שינוי הסיסמא צריך תמיד לכלול את השדות הבאים:
  - סיסמא ישנה.
  - הסיסמא חדשה.
  - אימות הסיסמא החדשה.
- יש לאמת בצד השרת את הסיסמא הישנה שהזין המשתמש מול הסיסמא של המשתמש בבסיס הנתונים.

4.5. Session Managment

4.5.1. כללי

מיד לאחר סיום תהליך ה - Authentication, מתחיל תהליך ה - Session של המשתמש. ה - Session משמש כדי לעקוב אחר פעולות המשתמשים ולאכוף את ההרשאות של המשתמש, החל מהבקשה הראשונה של המשתמש.

4.5.2. SessionID

בכדי לעקוב אחר המצב של המשתמש המזוהה, שרת ה - Web מייצר Session ID אשר ייחודי עבורו וילווה אותו עד ליציאתו מהמערכת. המלצות ליצירת SessionID חזק:

- אורך: ה - Session ID צריך להיות ארוך מספר כדי להתגונן בפני מתקפות Brute Force.
- בלתי צפוי: ה - Session ID צריך להיות מורכב ממספר נתונים משתנים כך שלא יהיה ניתן לנחש אותו.
- שם: ה - Session ID לרוב מועבר באמצעות ערך ב - Cookie. לכל ערך ב - Cookie יש שם. מומלץ שהשם של ה - Cookie לא יזהה אותו כ - Session ID. למשל ב - PHP, השם ברירת מחדל הינו PHPSESSID - מומלץ לשנות את שם ברירת המחדל.

4.5.3. שימוש במנגנון יציאה מהמערכת

סביר כי משתמש באפליקציה ייצא ממנה באחת משתי הדרכים הבאות:

- סגירה לא מבוקרת - סגירת הדפדפן, סגירת המחשב עצמו וכדומה.במקרה זה לשרת אין דרך לדעת כי המשתמש סיים את עבודתו עם האפליקציה. מסיבה זו מקובל להגדיר 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 מהמערכת. 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.

4.5.4. המלצות ל - PHP:

מומלץ לעשות שימוש בפונקציות המובנות של PHP:

1) session_start() - ליצירת ה - Session.
2) setcookie() - ליצירת ה - Cookie.
3) session_destroy() - מחיקת ה - Session.

דוגמא:

```
session_start()
$cookie_name = "session";
$id = session_id();
setcookie($cookie_name, $id, time()+3600, “path”, "domain",
true, true);
```

4.6. Authorization

4.6.1. כללי

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 שני עקרונות חשובים בנושא הרשאות, מידור ובקרת גישה אשר יש לפעול לפיהן:

4.6.1.1. עקרון ההרשאה המינימאלית

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

4.6.1.2. 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.

4.6.2. ניהול הרשאות

כל פניה צריכה להיבדק תחילה ברמת הרשאות- האם המשתמש רשאי לבצע את הפעולה הספציפית. לדוגמא האם מנהל מחלקה 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?

4.6.3. Authorization ב - Open EMR

מערכת Open EMR מספקת מנגנון הרשאות מלא למשתמשי המערכת. מערכת ההרשאות שנבנתה במערכת Open EMR הינה מבוססת User Roles. ניתן למצוא מידע אודות התפקידים השונים בקישור הבא:

http://www.open-emr.org/wiki/index.php/Access_Controls_Listing

מומלץ ליישם את הרשאות המשתמשים עפ"י שני העקרונות שהוצגו.

מידע נוסף על ניהול המשתמשים וההרשאות במערכת Open EMR ניתן למצוא בקישור הבא:

http://www.open-emr.org/wiki/index.php/ACL_Fine_Granular_Control

4.7. Auditing

4.7.1. 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).

4.7.2. 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".

4.7.3. מה לא לתעד

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.

4.7.4. תיעוד מערכת Open EMR

מערכת Open EMR מגיעה עם מודול מובנה שמספק רישום ללוג של משתמשי המערכת. ניתן למצוא מידע בקישור הבא:

http://www.open-emr.org/wiki/index.php/3.1_Auditing_in_OpenEMR

4.7.5. <u>המלצות נוספות:</u>

מומלץ לרשום ללוג את הנתונים הבאים:

- מבצע הפעולה - Session ID + Username.
- מהות הפעולה.
- סטטוס הצלחת הפעולה.
- זמן ביצוע הפעולה.

4.8. Error Handling Mechanism

4.8.1. 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.

4.8.2. 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.

4.8.3. Error Handling in PHP

```
php.ini configuration file:
error_reporting = E_ALL
log_errors = On
display_errors = Off
```

4.9. עבודה עם בסיס נתונים

4.9.1. כללי

כמעט בכל מערכת נעשה שימוש זה או אחר בבסיסי נתונים. לרוב בסיסי הנתונים יש מנגנוני אבטחת מידע פנימיים אשר ניתן להשתמש בהם.

4.9.2. הרשאות משתמש

הרשאות המשתמש של בסיס הנתונים צריכות להיות מוגבלות. יש לזכור כי משתמש בעל הרשאות חזקות 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, db_owner וכו'.
- 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).

4.9.3. שמירת נתונים מזהים בבסיס הנתונים

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.

4.9.4. 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.

4.9.5. 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.

4.9.6. דוגמא לשימוש בשאילתות פרמטריות PHP & mySQL

```
<?php
      $database = "localhost";
      $username = "username";
      $password = "password";
      $database_name = "db_name";

      // Creating the connection to the db
      $connection = new mysqli($database, $username,
      $password, $database_name);

      // Preparing the statement
      $statement = $connection->prepare("INSERT INTO
      SomeTable (first, second) VALUES (?, ?, ?)");
      $statement->bind_param("sss", $first, $second);

      // Set the parameters and execute
      $first = "First";
      $second = "Second";
      $statement->execute();

      $first = "First_2";
      $second = "Second_2";
      $statement->execute();
?>
```

4.10. Encryption

4.10.1. כללי

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

4.10.2. 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.

4.10.3. 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 איזו 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 - AES256bit או במקרה הצורך הצפנה א-סימטרית כגון RSA.

4.10.4. 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.

4.10.5. הצפנות ב - PHP:

```
$iv_size = mcrypt_get_iv_size(MCRYPT_RIJNDAEL_128,
      MCRYPT_MODE_CBC);
$iv = mcrypt_create_iv($iv_size, MCRYPT_DEV_URANDOM);
$key = "somekey";
$text = "Some string to encrypt";
$encryptedText = mcrypt_encrypt(MCRYPT_RIJNDAEL_128, $key,
      $text, MCRYPT_MODE_CBC, $iv);
```

הערה: מידע נוסף על הצפנה מובנית ב - Open EMR ניתן למצוא בקישור הבא:

http://www.open-emr.org/wiki/index.php/Encryption_and_Decryption_of_Documents

5. נספחים

5.1. נספח א' - הגדרות אבטחה בקובץ php.ini:

#	Name & value	הסבר
1.	`display_errors = Off`	מניעת שליחת הודעות שגיאה למשתמשי הקצה.
2.	`log_errors = On` `error_log = /file.log`	הגדרת שמירת הודעות השגיאה לקובץ מסויים.
3.	`file_uploads = Off`	מניעת העלאת קבצים (לפי הצורך העסקי).
4.	`upload_max_filesize = 1M`	הגדרת גודל מקסימלי להעלאת קובץ (לפי הצורך העסקי).
5.	`allow_url_fopen = Off`	מניעת Code Injection.
6.	`allow_url_include = Off`	מניעת Code Injection.
7.	`sql.safe_mode = On`	מניעת SQL Injection.
8.	`magic_quotes_gpc = Off`	מניעת SQL Injection.
9.	`post_max_size = 1K`	הגבלת גודל ה - body של ה - POST (לפי צורך עסקי).
10.	`max_execution_time = 30`	מניעת DoS.
11.	`max_input_time = 30`	מניעת DoS.
12.	`memory_limit = 40M`	מניעת DoS.
13.	`disable_functions = exec, shell_exec, system, proc_open, popen, curl_exec ,parse_ini_file, show_source`	חסימת פונציות מסוכנות.

5.2. נספח ב' - בדיקות קלטים - SQL Injection:

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

נוהל פיתוח מערכות מאובטחות

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(

5.3. נספח ג' - בדיקות קלטים XSS

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

5.4. נספח ד' - תווים נוספים

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#&
XSS	>>
XSS with no single quotes or double quotes or semicolons	/=
XSS	;''\
US-ASCII encoding	¾
unicoded XSS	and DIV ȣȣ\

Checklist עבור פיתוח מאובטח של מערכת קלינקל

מס״ד	נושא	בוצע / לא בוצע	הערות
	מדיניות סיסמאות		
1.	אורך סיסמא מינימלי - לפחות 7 תווים.		
2.	מורכבות סיסמא - שימוש בשלושה סוגי תווים לפחות - אותיות גדולות, אותיות קטנות ומספרים.		
3.	תאריך תפוגה לכל הסיסמאות - 30 עד 90 יום לפי רמת האבטחה.		
4.	היסטוריה של סיסמאות - שמירת עשר סיסמאות אחרונות, ואכיפת אי-שימוש בסיסמאות קודמות \ דומות.		
5.	נעילת משתמש לאחר שלושה עד חמישה ניסיונות גישה כושלים.		
6.	הסתרת הסיסמא בממשק המשתמש, למשל ע״י כוכביות.		
	ניהול הרשאות		
7.	על המשתמש לראות ״עולם״ שמתאים לפרופיל ההרשאות שלו. אין לחשוף בפני המשתמש יכולות של המערכת שאינן פתוחות לגביו. מומלץ להגדיר ולאפיין את תפקידי המשתמשים השונים במערכת, ואת רמת ההרשאה הנדרשת לכל תפקיד.		
8.	מערכת ההרשאות צריכה להיות מרכזית במערכת וצריכה לנהל את כל הפניות למשאבים. אין לאפשר גישה למשאבים משום נקודה אחרת במערכת, אלא דרך מערכת ההרשאות.		
9.	מומלץ לבצע הרצת השירותים תוך מתן מינימום הרשאות.		
10.	מומלץ להגביל הרשאות גישה לקבצי המערכת בצד השרת.		
	תווך התקשורת בין משתמשי המערכת לשרת ה - Web		

11.	שליחת סיסמאות על גבי התווך (במקרה של הזדהות רשת) צריך להתבצע באופן מוצפן, ע"י שימוש בפרוטוקול HTTPS, עם אלגוריתם TLS v1.2.		
ניהול Session			
12.	הגדרת Session Timeout של 30 דקות		
13.	יצירת SessionID באמצעות פונקציית session_start(),		
14.	שימוש בפונקציית session_destroy() בעת סגירת Session של משתמש.		
15.	שימוש ב - Cookies עבור שליחת ה - SessionID למשתמש.		
16.	שינוי שם ברירת המחדל של ה - Cookie, PHPSESSID.		
17.	הגדרת תכונות ה - Cookie: http_only, path & secure.		
שמירת מידע רגיש			
18.	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.		
19.	רצוי להשתמש בספריות הצפנה קיימות, למשל: *mcrypt_encrypt*.		

20.	אין לשמור מידע רגיש בתחנות הקצה. במידה ומבוצעת שמירה של מידע רגיש עפ"י הצורך העסקי, יש לשמור אותו באופן מוצפן.		
21.	מידע רגיש בבסיס הנתונים צריך להיות מוצפן באלגוריתם הצפנה חזק.		
22.	יש להימנע מהצגה ושליחת מידע רגיש בבקשות GET.		
	חיווים ולוגים		
23.	קיום מנגנון התראה על אירועים חריגים בזמן אמת.		
24.	רישום זמן שימוש אחרון במערכת והצגתו למשתמש בעת כניסה למערכת.		
25.	רישום מלא של כלל פעולות המשתמש. עבור כל פעולה יש לתעד את מבצע הפעולה באופן חד ערכי, זמן הפעולה, כתובת IP ממנה בוצעה הפעולה וסטאטוס סיום.		
26.	יש להימנע משמירת מידע רגיש בלוגים.		
27.	כל גישה ל – log צריכה אף היא להופיע ב – log עצמו.		
	עבודה מול בסיס הנתונים		
28.	**שאילתות מאובטחות:** שימוש ב - Stored Procedures בעת תשאול בסיס הנתונים.		
29.	**הזדהות לבסיס הנתונים:** שימוש ב - Windows Authentication.		
30.	**הרשאות המשתמש האפליקטיבי בבסיס הנתונים:** מתן הרשאות מינימום למשתמש המתשאל את בסיס הנתונים.		
	בדיקות קלטים		

.31	**<u>ממשקים חיצוניים:</u>** 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.		
.32	**<u>משתמשי קצה:</u>** בדיקות קלטים צריכות להתבצע על כלל הקלטים. במתודולוגית White List, מתודולוגיה המגדירה עבור כל קלט מהו הקלט התקין עבורו. בדיקות קלטים צריכות להיעשות בכל שכבה ושכבה. מומלץ לעשות שימוש בפונקציית filer_var؟		
	<u>בדיקות פלטים</u>		
.33	ביצוע Santize לתווים בעת שליחתם למשתמש הקצה. מומלץ לעשות שימוש בפונקציית filter_var		
	<u>טיפול בשגיאות ריצה</u>		
.34	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).		
.35	במקרה שיש שגיאה יש להציג הודעה גנרית אחידה ומוכנה מראש.		
	הגדרות PHP.ini		
.36	`error_reporting = E_ALL`		
.37	`log_errors = On`		

		display_errors = Off	.38
		display_errors = Off	.39
		log_errors = On *error_log = /file.log*	.40
		file_uploads = Off	.41
		upload_max_filesize = 1M	.42
		allow_url_fopen = Off	.43
		allow_url_include = Off	.44
		sql.safe_mode = On	.45
		magic_quotes_gpc = Off	.46
		post_max_size = 1K	.47
		max_execution_time = 30	.48
		max_input_time = 30	.49
		memory_limit = 40M	.50
		disable_functions = exec, shell_exec, system, proc_open, popen, curl_exec ,parse_ini_file, show_source	.51