digital whisper
Technologic Papers

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.

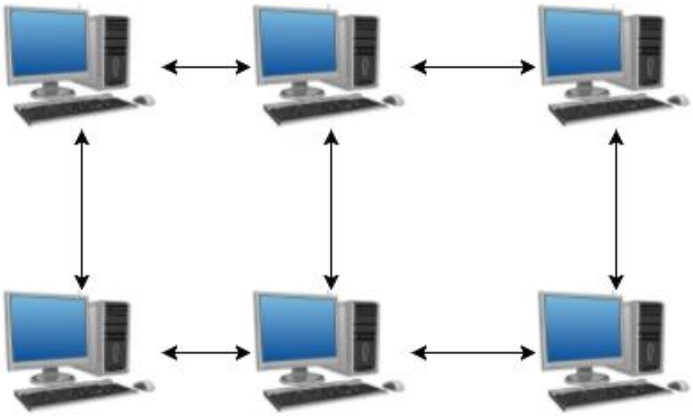
בעקבות כך פותחו שיטות התחמקות מבעיות שיתוק הדומיינים והשרתים בעזרת אלגוריתמי DGA (Domain Generation Algorithm), הרעיון מאחורי האלגוריתם הוא בעצם לגרום לבוטים לפנות אל דומיין חדש/מספר דומיינים חדשים כל יום.

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.

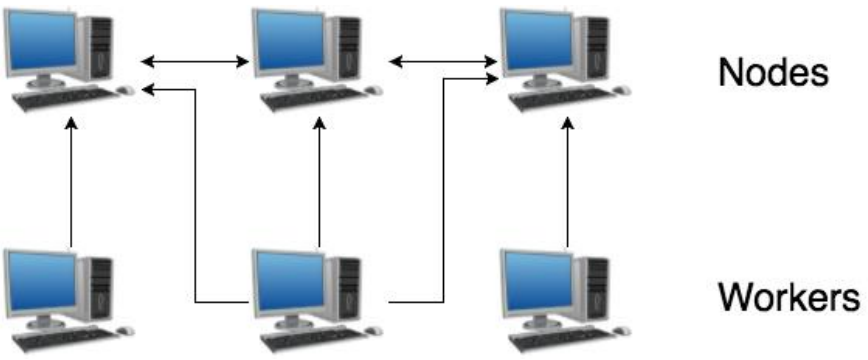
כשמדברים לרוב על רשתות בוטנט מבוססי P2P, התמונה שמצויירת בראשנו לרוב היא רשת אשר כל בוט מתקשר עם שכניו ויכול לשמש כ-"שרת" להעברת נתונים וגם כ-"לקוח" לקבלת נתונים. משהו כזה פחות או יותר:



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.

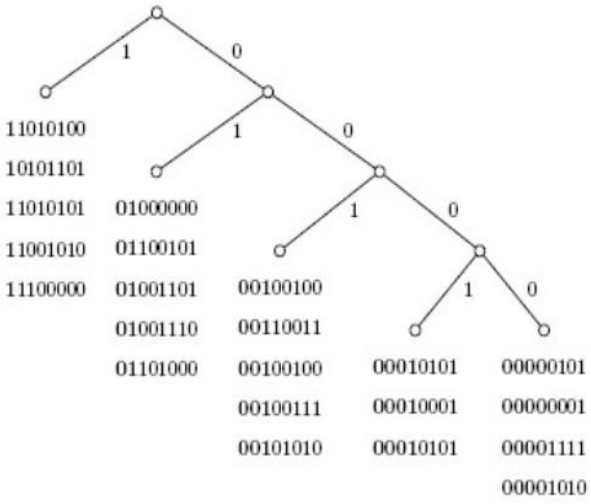
קדמילה (Kadmelia) - הקדמה

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

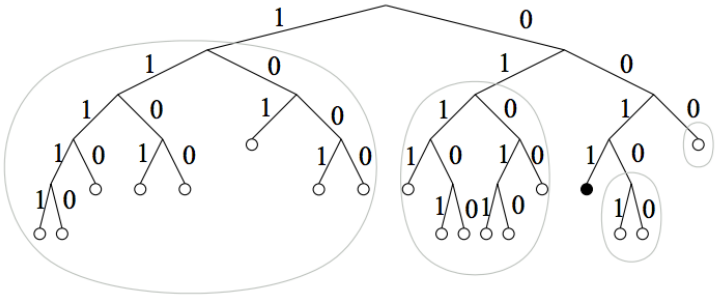
K-Bucket: כל דלי מיוצג בעזרת עץ בינארי, המכונה טבלת ניתוב. כל דלי בגודל K מכיל מקסימום K רשומות. במקרה שלנו, כל Node מחזיק רשימת Nodes תקינים אחרים, במרחק בין 2^i ל-2^(i+1) עבור i=1…N. כל K-Bucket מזוהה ע"י קידומת ה-ID שהוא מכיל. למשל, העץ הבא מייצג דוגמא של טבלת ניתוב עבור Node 00000000 ו-K-Bucket באורך 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_FROMכלומר, קבלת כל הפקודות החדשות החל מהפקודה ה-3. הסיבה היחידה שהבוט מבקש את הפקודה החל מפקודה מסויימת 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.

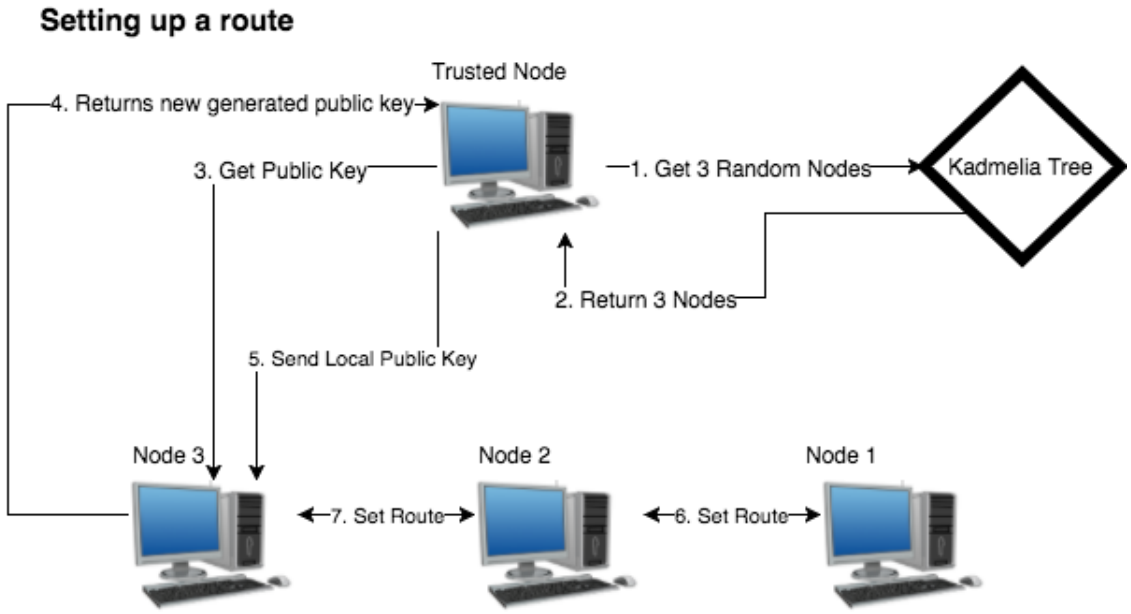
שימוש בבוטנט כפרוקסי מתקדם עם הצפנת תעבורה

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

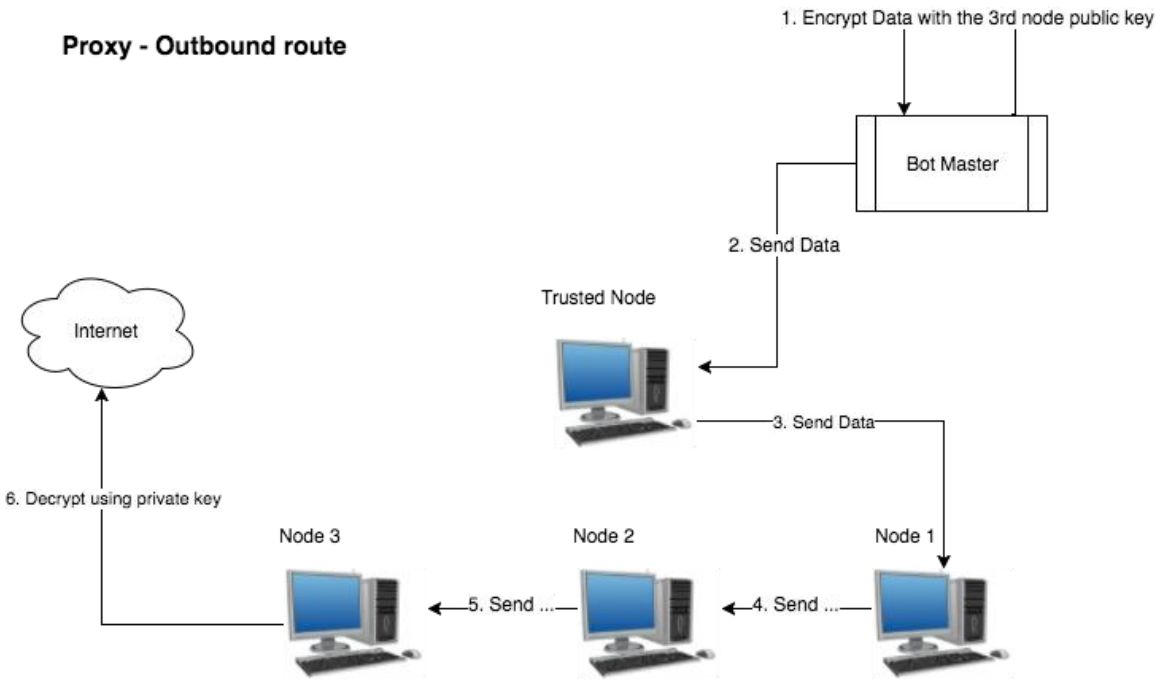


נבקש מאחד ה-Trusted Node ליצור לנו שרשרת של נתיב יציאה לאינטרנט בגודל K כלשהו, לשם הנוחות ומהירות הפרוקסי נבחר K=3 של Nodes. כלומר ה-Trusted Node יבחר ראנדומלית מהעץ של Kadmelia כשלושה Nodes ויחבר בניהם נתיב לרשת כפי שמתואר בתרשים הזרימה למעלה.

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.

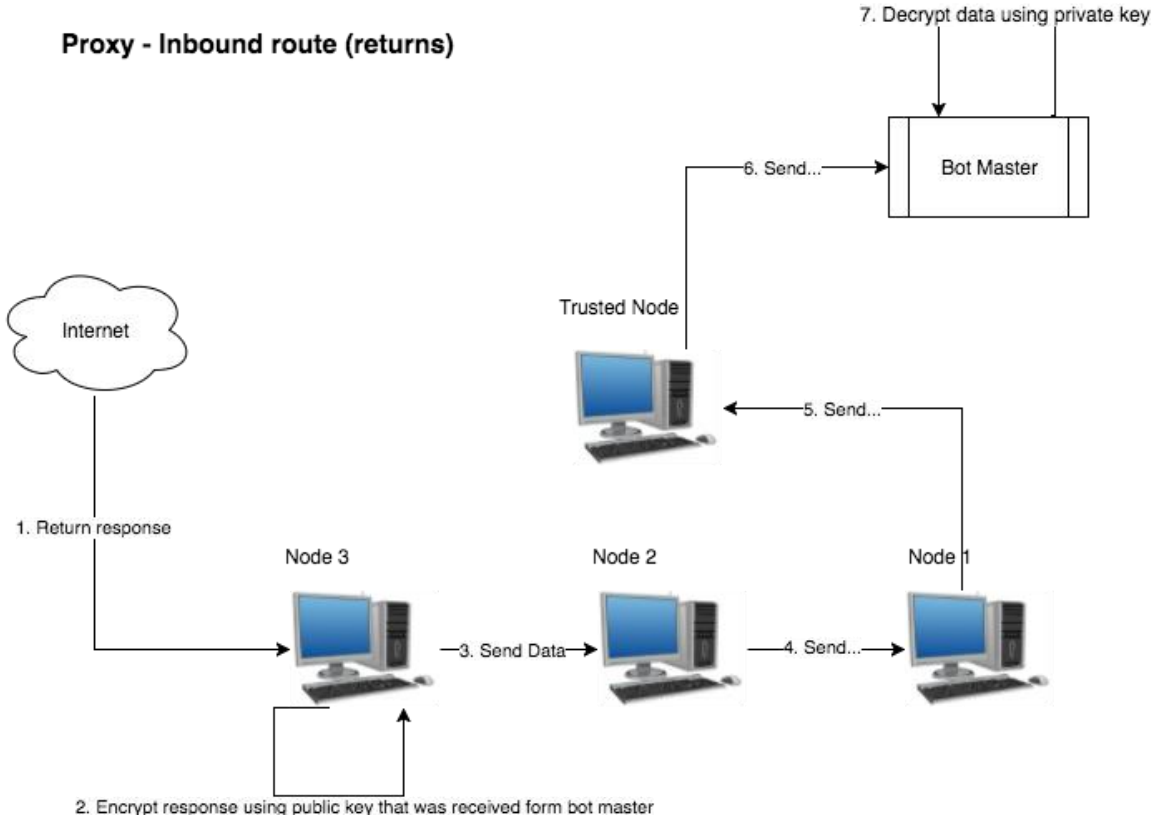
שלב ב: יציאה לרשת דרך הנתיב שהוגדר



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 מסויימת.

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