Digital Whisper

גליון 2, נובמבר 2009

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורך:	ניר אדר
כתבים:	אפיק קסטיאל, אורי (Zerith)

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

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

דבר העורכים

ברוכים הבאים לגליון השני של Digital Whisper - מגזין אלקטרוני בנושאי טכנולוגיה. את הגליון מביאים לכם **ניר אדר**, מהנדס תוכנה, מנהל פרוייקט UnderWarrior (www.underwar.co.il) **ואפיק קסטיאל** (aka cp77fk4r), אחד מהבעלים של www.Trythis0ne.com, Penetration Tester בחברת BugSec, איש אבטחת מידע וגבר-גבר באופן כללי (ופרטי).

הרעיון מאחורי Digital Whisper הוא ליצור נקודה ישראלית איכותית שתרכז נושאים הקשורים למחשבים בכלל ובאבטחת מידע בפרט, והכל - בעברית. הגיליון אינו מכיל רק כתבות בנושא אבטחת מידע, אבל הדגש העיקרי שלנו הוא על אבטחת מידע.

הגליון הראשון קיבל מספר רב של תגובות. מאוד נהננו לקבל מכם את כל המחמאות, השבחים וגם את הערות הבונות. הסקרנים שבינכם יכולים לראות מעט מהתגובות שקיבלנו בדף הגליון הראשון, בכתובת http://www.digitalwhisper.co.il/issue1. במייל, בשיחות בצ'אט ובמקומות נוספים קיבלנו עוד הרבה פידבקים נוספים. תודה!

בגליון זה 7 מאמרים חדשים, וכרגיל ניסינו להקיף מגוון רחב של נושאים ולהציג את העולם המרתק של אבטחת המידע. בגליון זה לא יפורסמו מאמרים שלי (ניר), אך אני אחזור בחודש הבא.

דיון מיוחד התעורר מסביב לכתבה בנושא Privilege Escalation שפורסמה בגליון הקודם, בפורומים באתר www.underwar.co.il. אלכס קרפמן מביא את סיכום דעתו לגבי הנושא:

> ה-"תקיפה" באמצעות Windows Defender היא איננה PE בשום פנים ואופן, כי כדי לכתוב אל %SYSTEMDRIVE% או אל %PROGRAMFILES% מלכתחילה צריך הרשאות אדמין, וזה שאדמין יכול להריץ פרוססים בהרשאות אדמין זה לא בדיוק PE. בניגוד לטענות שעלו בדיון בנושא, הדבר איננו קשור ל-UAC בשום צורה: רק אדמין יכול לכתוב לתיקיות האלה, ואף "limited user" לא יכול לעשות זאת באף גרסה של Windows NT; בלי קשר, UAC הוא "טריק" שהחל מווינדוס ויסטה הגורם לכך שגם פרוססים שמריץ משתמש שעשה לוגין כאדמין ירוצו לפעמים ללא ההרשאות המלאות של אדמין.

חשוב לציין כי הבעיה הזו לא קיימת בשימוש נכון (לא "עצלן") ב-CreateProcess. היא כן קיימת בשימוש ב-ShellExecute (וכנראה בכל פונקציות ה-shell). אם מייקרוסופט היו מממשים את HKLM\Software\Microsoft\Windows\CurrentVersion\Run כמו בני-אדם - דורשים שם קובץ מלא ומשתמשים ב-CreateProcess כמו שצריך - לא הייתה בעיה.

כמו כן, גם השימוש ב-at לא נחשב כ-PE. אמנם שימוש בו אכן מאפשר לך לעלות להרשאות של LOCAL_SYSTEM שקצת יותר גבוהות משל אדמין רגיל (יש privileges שיש רק ל-LOCAL_SYSTEM לאחר התקנה נקייה), אבל זה שאדמין יכול להריץ תהליכים בהרשאות גבוהות זו לא חולשה, אלא ההגדרה של "להיות אדמין": לאדמין בווינדוס יש יכולת לשלוט על ה-"TCB" וזה תקין. באופן דומה אדמין יכול "לגנוב" ownership על קובץ שאין לו הרשאות אליו ולתת לעצמו הרשאות לקובץ הזה. גם זו לא חולשה, אלא היכולת הבסיסית שמגדירה אדמין. רק אדמין יכול להשתמש ב-at כדי לתזמן משימה בהרשאות LOCAL_SYSTEM וזה תקף לכל גרסה של Windows NT.

"לרוץ בהרשאות SYSTEM" במובן של לרוץ תחת היוזר LOCAL_SYSTEM זה לחלוטין לא לרוץ ב-ring 0. מה שמקבלים זו ריצה רגילה לחלוטין ב-usermode, תחת יוזר עם הרבה הרשאות. גם LOCAL_SYSTEM וגם אדמין "רגיל" יכולים לטעון דרייברים, ודרייברים הם רכיבי התוכנה היחידים בווינדוס שבאמת רצים ב-ring 0 ומקבלים את כל היכולות המיוחדות שלו: גישה רציפה לכל הזיכרון הפיזי, גישה לכל החומרה (כתובות ו-I/O ports) וכו'.

כיוון שאין בשימוש ב-at שום באג, שום באג לא תוקן בויסטה. גם בויסטה at יכול להריץ פרוססים תחת SYSTEM, אם הוא מורץ על-ידי אדמין, אבל לא באופן אינטראקטיבי. אני מעריך שהדבר נועד כדי למנוע תקיפות מבוססות-UI כמו shatter attacks, בדומה ל-UIPI.

דיון נוסף שהתעורר במספר מקומות ברשת, ובמיוחד בקבוצת Yahoo groups בשם hackers-il הוא לגבי פורמט הגליון. האם לדעתכם הגליון האלקטרוני צריך להיות מופץ בפורמט PDF? בפורמט HTML? בפורמט טקסט? הבחירה שלנו בפורמט PDF הגיעה אחרי מחשבה רבה. מצד אחד רצינו למצוא פורמט שיתאים לאנשים רבים ככל האפשר. מצד שני הדגש היה לאפשר לכם, הקוראים, גם להדפיס ביתר קלות את המגזין בפורמט שיראה טוב. נשמח לשמוע את דעתכם בתגובות על המגזין.

בנוסף רצינו להגיד תודה לאורי (Zerith) על שכתב לנו מאמר מעניין ומקצועי לגליון.

קריאה מהנה!

אפיק קסטיאל		ניר אדר

תוכן עניינים

דבר העורכים 2

תוכן עניינים 4

SSL & Trasport Layer Security Protocol 5

Manual Unpacking 16

וירוסים - שיטות טעינה 27

RFID Hacking 40

Port Knocking 48

הפרוטוקול Kerberos v5 57

DNS Cache Poisoning 63

דברי סיום 75

SSL & Trasport Layer Security Protocol

מאת אפיק קסטיאל (cp77fk4r)

הקדמה (Secure Sockets Layer & Trasport Layer Security Protocol)

פרוטוקול ה-SSL (קיצור של Secure Sockets Layer) פותח בתחילת שנת 1994 ע"י חברת NETSCAPE בכדי להוסיף נדבך לאבטחת רשת האינטרנט. במקור הפרוטוקול פותח בכדי להוסיף אבטחה לאפליקציות WEB, אבל כיום הוא יכול לשמש גם כמאפיין אבטחה עיקרי בחיבורים רבים אחרים, כגון NNTP, SMTP, או FTP (כמו למשל חיבור ה-JSCAPE לשרתי AIX) ועוד.

כמו שאפשר להבין משמו - הפרוטוקול מהווה שכבה נוספת מתחת לפרוטוקול הקיים, כדוגמת השימוש בו על גבי פרוטוקול ה-HTTP, או מה שאנחנו מכירים כיום יותר כ-"HTTPS", פרוטוקול ה-SSL "מתיישב" על גבי פרוטוקול ה-HTTP ודואג להצפנת הנתונים העוברים דרך הפרוטוקול התחתון.

ההצפנה בפרוטוקול ה-SSL עוצבה באופן כזה שהיא פועלת ברמת ה-Socket, ולכן השימוש הנכון בו הוא שימוש על גבי פרוטוקול אמין כגון TCP, בשל העיצוב שלו הוא אינו תלוי מערכת, מה שנותן לו את העדיפות כשמדובר במקרים שבהם צריכים להעביר מידע על גבי מספר פלטפורמות (כגון שימוש באינטרנט). פרוטוקול ה-TLS (קיצור של Transport Layer Security) מבוסס ברובו על הגירסה השלישית של פרוטוקול ה-SSL, שיצאה כשנתיים לאחר הגירסה הראשונה של הפרוטוקול המקורי - בשנת 1996, שלוש שנים לאחר שיצאה הגירסה השלישית הושלם הפיתוח של ה-TLS ע"י החברה' מ-IETF (קיצור של Internet Engineering Task Force) והוא הובא לשימוש ציבורי (בתחילה של חברות אשראי בעיקר) לקראת סוף שנת 1999.

הרעיון הכללי

הרעיון מאחורי שיטת העברת המפתחות של הפרוטוקול מאוד מזכירה את ה-PGP ואת השימוש במפתח ציבורי של אחד מהגורמים (הצד המארח), אך החידוש המרכזי כאן הוא שימוש בגורם חיצוני - צד שלישי אמין - בכדי לאמת את נתוני ההצפנה לפני השימוש הראשוני בהם.

השימוש בהצפנה אסימטרית חזק בהרבה משימוש בהצפנה סימטרית, אך שימוש בה בפרוטוקול ה-FTP או ה-HTTP, למשל, פחות יעיל בהשימוש בהצפנה סימטרית, מפני שהוא יפגום מאוד ב"חווית הגלישה" - המידע יעבור באופן איטי יותר, ולכן השימוש בו לאורך כל התקשורת - נכון לימים אלה - לא מתקבל על הדעת.

לחיצת היד של הפרוטוקול משתמשת בשתי השיטות - היא לוקחת את חוסנה של ההצפנה הסימטרית מצד אחד, ומצד שני היא לוקחת את יעילותה וגמישותה של ההצפנה האסימטרית. הרעיון הוא שאם התקשורת נפלה קורבן למתקפת Men In The Middle מוצלחת ואנחנו נבצע את החלפת מפתחות ההצפנה באופן פשוט כל המנגנון הלך, הצד השלישי מחזיק במפתח גם הוא, הוא יוכל לקבל את המידע המוצפן, לפענח אותו, לערוך אותו או לדלות ממנו את המידע הרגיש (סיסמאות, מספרי כרטיסי אשראי וכו') ולשחרר אותו ליעדו (במקרה כזה גם לא תהיה שום אפקטיביות למנגנוני Time-Stamp למניהם, מפני שלא תהיה בעיה לתוקף לזייף אותם ב-Request/Response הנוכחי). אך במקרה והתוקף מקבל מידע מוצפן בעזרת מפתח שאין לו - הוא לא יוכל לבצע בעזרתו שום דבר- אם הוא ינסה לפגום במידע, להחליפו באחר או לבצע כל מניפולציה, ולו הקטנה ביותר, בזמן שהלקוח ינסה לבצע פיענוח על החבילה המתקבלת ויתקל בבעיות, הוא יבין שגורם שלישי "נגע" בחבילה, יזרוק אותה מיד וידע לא לסמוך על התקשורת הנוכחית.

פרוטוקול ה-SSL, תומך במספר אלגוריתמים לשני שלבי ההצפנה שבהם הוא משתמש.

- **להצפנה ה-אסימטרית (שלב שני - שלב החלפת המפתחות):**
  הפרוטוקול תומך ב-RSA, Diffie-Hellman , DSA ובגירסתו השלישית של הפרוטוקול נוספה היכולת להשתמש אף ב-FORTEZZA.

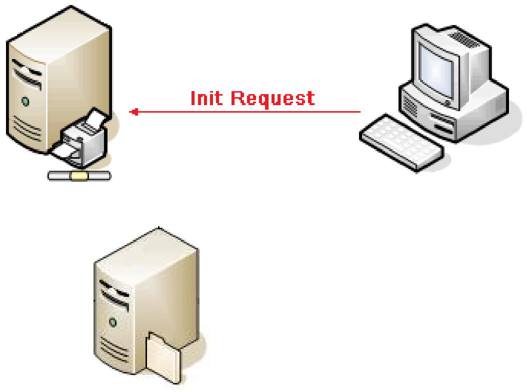
- **להצפנה הסימטרית (שלב שישי - העברת המידע הרגיש):**
  הפרוטוקול תומך ב-AES, משפחת ה-RC (הנפוצים ביותר לשימוש הם ה-RC2 וה-RC4), IDEA, וכמובן ב-DES וב-TripleDES.

- במקרים רבים אפשר למצוא נדבך אבטחה נוסף והוא העברת המידע באופן מגובב. ברוב המקומות הסטנדרטיים מבוצע שימוש באלגוריתמי גיבוב מוכרים כגון MD5 (בגירסאות ישנות של הפרוטוקול ישנו שימוש אף ב-MD2 ו-MD4) ומשפחת SHA.

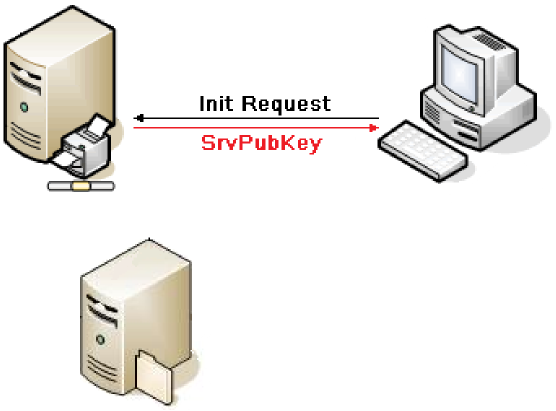
מנגנון ה-Handshake

מהלך לחיצת היד מתנהלת לפי האופן הבא:

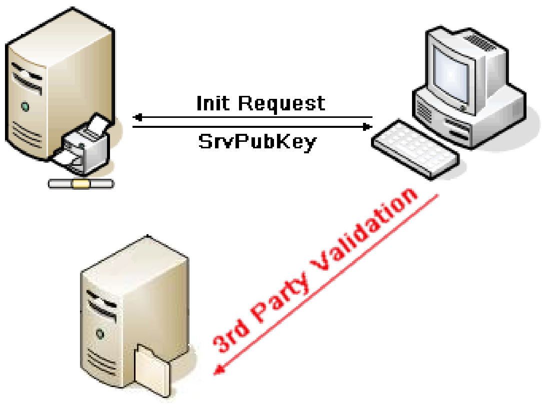
- **שלב ראשון** - הלקוח שולח בקשה ליצירת קשר עם השרת.



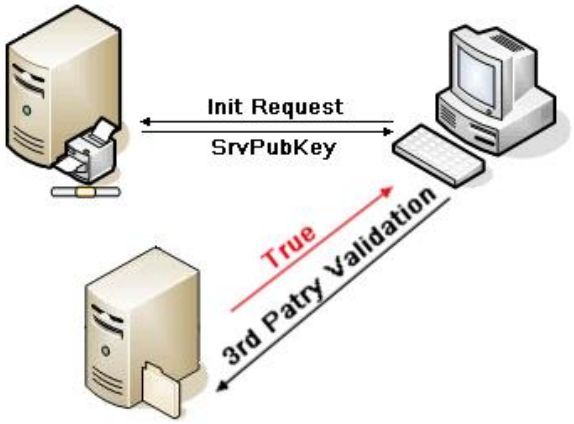
- **שלב שני** - השרת שולח Certificate ייחודי לו, אשר מכיל בין היתר את המפתח הציבורי שלו:



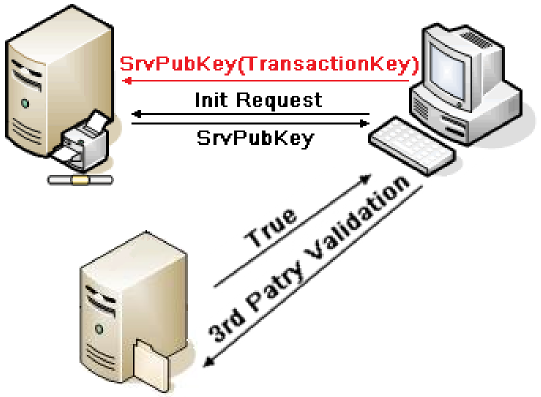
- **שלב שלישי** - הלקוח מתשאל צד שלישי (החברה המנפיקה של אותו ה-Certificate) ומוודא כי הוא אכן אותנטי.



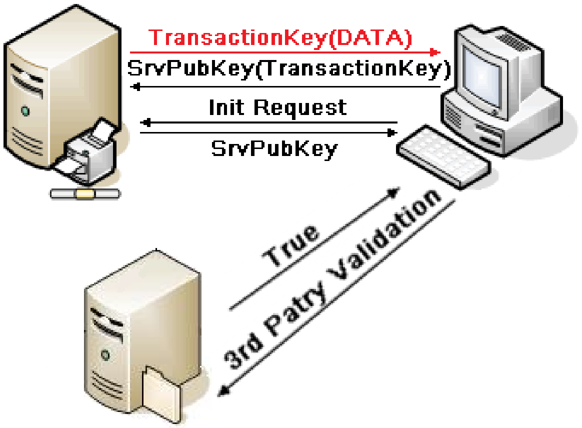
- **שלב רביעי** - הלקוח מקבל תשובה חיובית (במידה ואכן ה-Certificate אמיתי) מהגורם השלישי.



- **שלב חמישי** - הלקוח מצפין מספר רנדומאלי (Transaction Key) בעזרת המפתח הציבורי שאותו הוא שלף מתוך ה-Certificate המאושר ושולח אותו לשרת.



- **שלב שישי** - השרת מפענח את ההודעה שהגיע מהלקוח בעזרת המפתח הפרטי שברשותו, שולף מתוכה את המספר הרנדומאלי שאותו יצר הלקוח, ומצפין בעזרתו את המידע הרגיש שאותו הלקוח ביקש.



המפתח הרנדומאלי הזה מהווה את מפתח ההצפנה של השיחה, ומכאן- שכבת האבטחה תצפין בעזרתו כל חבילת מידע שתעבור דרכה ותדאג לפיענוחה (בעזרת אותו מספר) כאשר החבילה הגיעה ליעדה. שימו לב שגם אם תוקף מאזין לתקשורת כבר מהשלב הראשון אין לו אפשרות לגלות מה מכילה חבילת מידע האחרונה שנשלחה בשלב השישי- המידע הרגיש שאותו ביקש הלקוח.

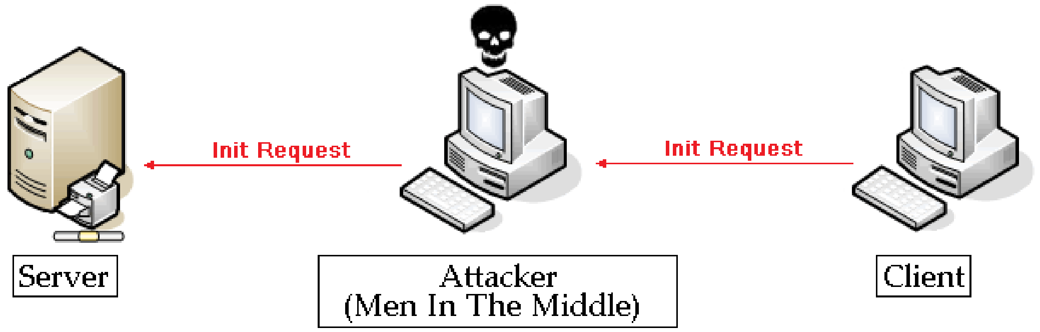
למה? כי בכדי להשיג את ה-Transaction Key על התוקף לפענח את המידע שנשלח ע"י הלקוח - מידע זה הוצפן באמצעות המפתח הציבורי של השרת ורק בעזרת המפתח הפרטי שלו יהיה ניתן לפענח את חבילת המידע הזו. כמובן שהמנגנון מכיל רכיבי Time-Stamp הנשלחים מהלקוח לשרת המוצפנים בעזרת המפתח הציבורי של השרת בכדי למנוע מתוקף שהצליח ליירט את התקשורת בין שני המחשבים לבצע מתקפות Replay Attack או לבצע סילוף של חבילות המידע.

מנגנון ה-Certificate Authority

כמו שראינו בפרק הקודם, בשלב השני והשלישי ישנו שימוש ב-Certificate של ה-SSL, בשלב השני- השרת שולח ללקוח את ה-SSL Certificate שלו - שכולל את מפתחו הציבורי שבו הלקוח ישתמש בכדי להצפין את מפתח השיחה העתידית, ובשלב השלישי הלקוח שולח לגורם שלישי **מוגדר מראש** את פרטי השרת בכדי לאמת כי אכן השרת הוא מי שהוא טוען לזהותו, או בשפה המקצועית, כאן מתבצעת פעולת ה-CA (קיצור של "Certificate Authority").

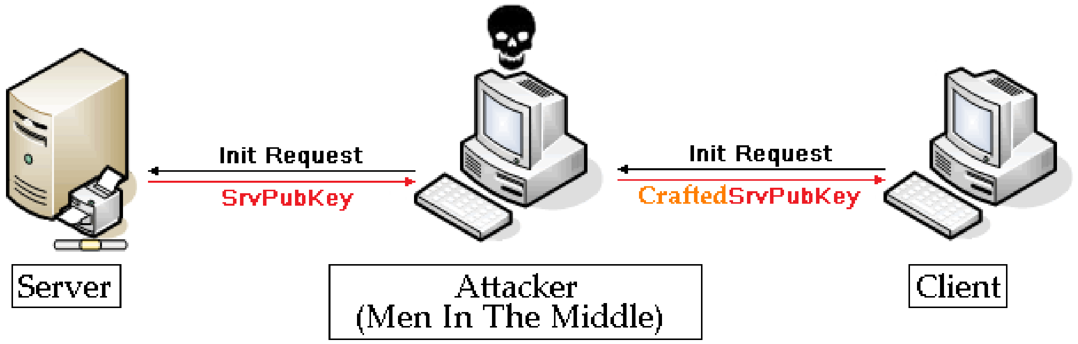
במקרים ואין שימוש ב-CA ובוצעה האזנה לאותו קו תקשורת שבו מתבצעת לחיצת היד שלנו, תוקף יוכל לנסות לבצע מניפולציה של המפתח הציבורי של השרת בכדי לגרום ללקוח להצפין את המידע הרגיש בעזרת מפתח ציבורי השייך לתוקף- וכך בעת השליחה לגנוב את המידע ולפענחו בעזרת המפתח הפרטי שלו עצמו, כמו בדוגמא הבאה:

- **שלב ראשון -** הלקוח שולח בקשת יצירת קשר עם השרת [על-גבי תקשורת תחת ציתות]:



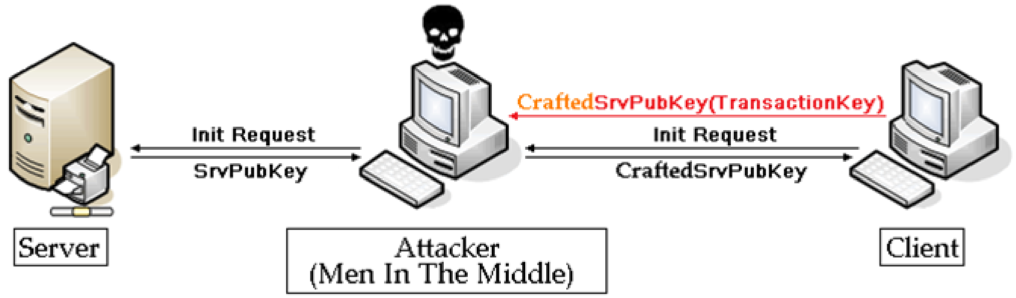
(הלקוח שולח את הבקשה לשרת > הבקשה מגיעה לתוקף > התוקף מעביר אותה לשרת מבלי לגעת בה)

- **שלב שני -** השרת שולח Certificate ייחודי לו אשר מכיל בין היתר את המפתח הציבורי שלו:



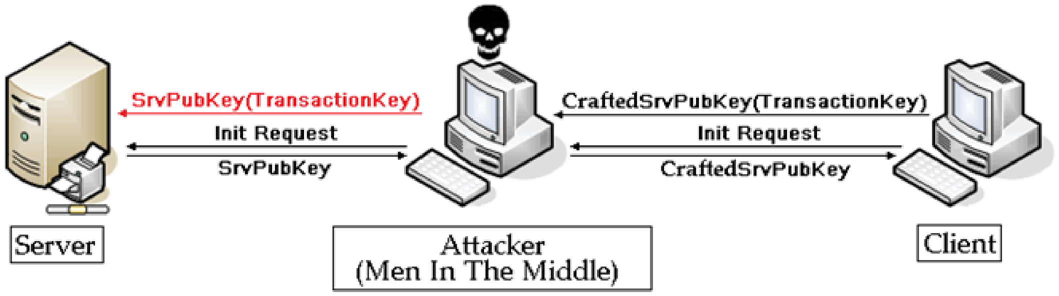
(השרת שולח תגובה המכילה את ה-Certificate, המכיל את המפתח הציבורי שלו ללקוח > התגובה מגיעה לתוקף > **התוקף עורך את פרטי המפתח הציבורי של השרת ומחליף אותו בפרטי המפתח הציבורי שלו (של התוקף)** > התוקף מעביר את תגובת השרת ללקוח).

- **שלב שלישי-** הלקוח מצפין מספר רנדומאלי (Transaction Key) בעזרת המפתח הציבורי שאותו הוא שלף מתוך ה-Certificate שקיבל מהשרת ושולח אותו לשרת.

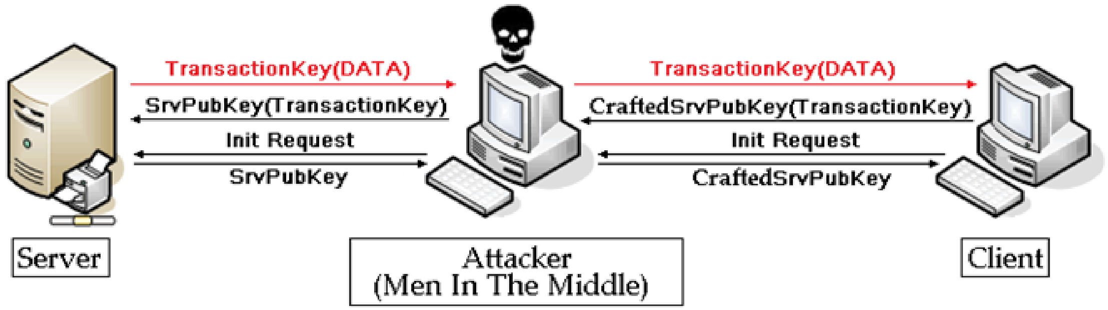


(הלקוח שולח לשרת מספר רנדומאלי בעזרת מפתח ההצפנה שקיבל מהשרת, לאחר שהתוקף ערך אותו > המידע מגיע לתוקף > התוקף מפענח את ההודעה ורואה מה המספר שנבחר ע"י הלקוח)

- **שלב רביעי -** התוקף מצפין את המספר הרנדומלי שקיבל מהלקוח בעזרת מפתחו הציבורי של השרת ושולח את החבילה לשרת:



- **שלב חמישי -** השרת מפענח את ההודעה שהגיע מהלקוח בעזרת המפתח הפרטי שברשותו, שולף מתוכה את המספר הרנדומאלי שאותו יצר הלקוח, ומצפין בעזרתו את המידע הרגיש שאותו הלקוח ביקש- ושולח ללקוח.



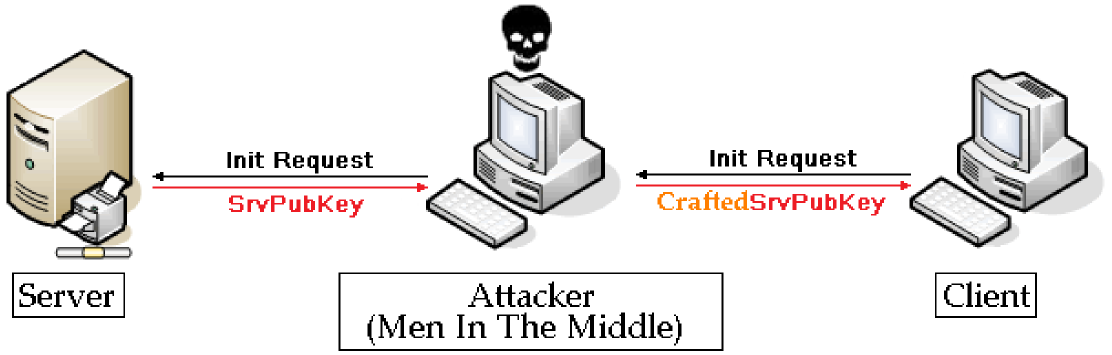
(השרת מצפין את המידע הרגיש שאותו ביקש הלקוח בעזרת מפתח ההצפנה שיצר הלקוח > השרת שולח את המידע המוצפן ללקוח > המידע מגיע לתוקף, התוקף מפענח את המידע, קורא אותו/ עורך אותו > התוקף מעביר את המידע ללקוח > GAME-OVER).

כמו שראינו, גם במקרה שיש שימוש בפרוטוקולי הצפנה והמידע נשלח באופן מוצפן, כל עוד אין לנו מנגנון שמאמת כי אכן אנחנו מדברים עם השרת האמיתי ולא עם מתחזה- המידע שלנו נתון לסכנה- גם במקרים כמו קניה באתרי אינטרנט המצב דומה:

- המידע המוצפן הגיע ללקוח (נניח- טופס תשלום הקנייה).
- הלקוח מכניס את כרטיס האשראי שלו ושולח לשרת.
- הפרוטוקול מצפין את המידע בעזרת אותו TransactionKey.
- המידע מגיע לתוקף- התוקף יודע כבר יודע מהו ה-TransactionKey, מפענח את החבילה- שולף את המידע הרגיש (לדוגמא- מספר האשראי), מצפין מחדש ושולח לשרת.

בדיוק בגלל התרחיש הנ"ל פותח מנגנון ה-Certificate Authority שנועד לאמת כי אכן השלב השני (שליחת המפתח הציבורי) בלחיצת היד מתבצע כמו שצריך, מבלי גורמים נוספים. השימוש הכללי של ה-CA הוא לאמת כי אותו מפתח ציבורי שקיבלנו מהשרת הוא אכן שייך לאותו שרת, וכי אין שום חשש להצפין בעזרתו את המידע הרגיש שאותו נרצה לשלוח לשרת, שימו לב- נחזור לשלב השני בתקשורת הנמצאת תחת האזנה:

- **שלב שני -** השרת שולח Certificate ייחודי לו אשר מכיל בין היתר את המפתח הציבורי שלו:



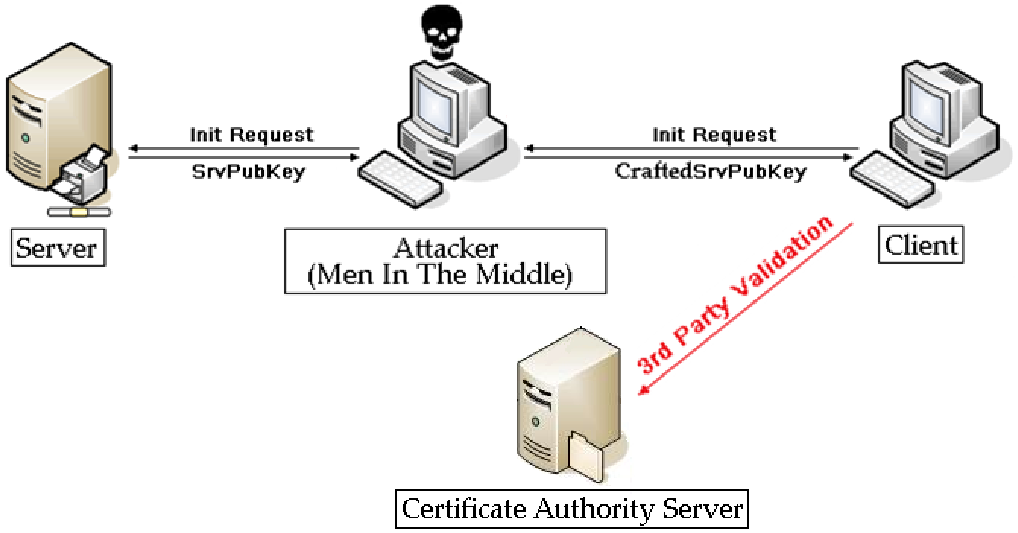
(השרת שולח תגובה המכילה את ה-Certificate המכיל את המפתח הציבורי שלו ללקוח > התגובה מגיעה לתוקף > **התוקף עורך את פרטי המפתח הציבורי של השרת ומחליף אותו בפרטי המפתח הציבורי שלו (של התוקף)** > התוקף מעביר את תגובת השרת ללקוח).

[כאשר מדובר בפרוטוקול התומך ב-"Certificate Authority" השלב הבא לא יהיה הצפנת מפתח השיחה ע"י הלקוח בעזרת המפתח הציבורי של השרת אלא בדיקה אל מול גורם שלישי (לרב חיצוני אך הדבר לא מחייב!) כי אותה חתימה אכן שייכת לאותה הכתובת.]

אופן השימוש בדרך זו מתבצע לרוב אל מול גורם שלישי בתקשורת, לאחר שקיבלנו את ה-Certificate מהשרת, אנו מבצעים בדיקה כי אכן ה-Fingerprint שלה זהה ל-Fingerprint שלה אצל הגורם השלישי (אצל הספק של אותה התעודה) ואם כן- אנחנו יכולים להיות בטוחים כי אכן מדובר ב-Certificate אמיתי ואין שום בעיה להשתמש במפתח הציבורי שהיא כוללת ולהצפין בעזרתו את המידע הרגיש שלנו שאותו נשלח בעתיד לשרת.

ה-Fingerprint של ה-Certificate הוא מעין ה-Hash של התעודה, והוא ייחודי לכל תעודה ותעודה, במקרה ותוקף או כל גורם אחר ינסה לבצע שינוי, ולו הקל ביותר, בשלב השני (שלב החלפת המפתחות) -ה Fingerprint של ה-Certificate ישתנה ולא יתאים לאותו Fingerprint שקיים אצל ספק התעודות (הגורם השלישי), מה שיגרום לכשל במהלך ה-Certificate Authority.

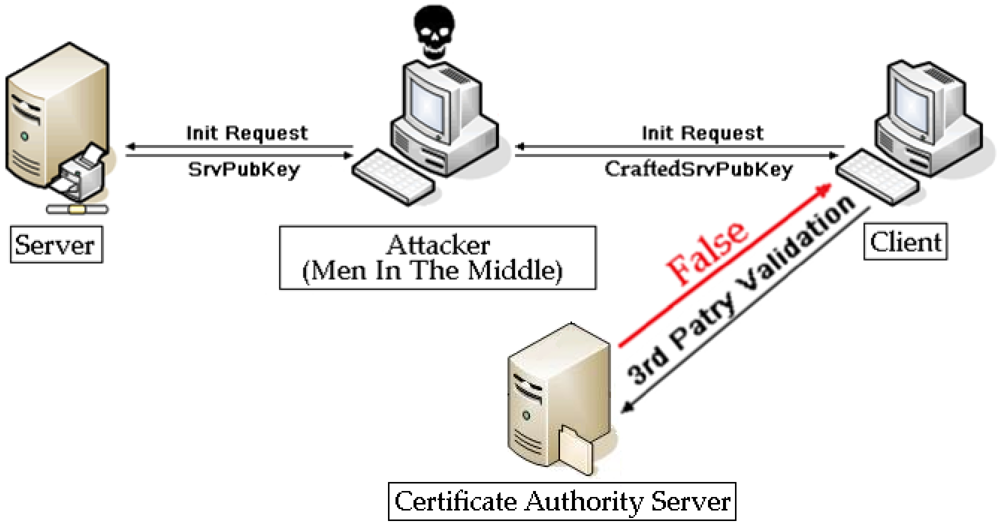
- **שלב שלישי** - הלקוח מתשאל צד שלישי (החברה המנפיקה של אותו Certificate) ומוודא כי הוא אכן אותנטי.



(שימו לב שבשלב הנוכחי הלקוח מתשאל את הצד השלישי בעזרת ה-Fingerprint של ה-CraftedSrvPubKey ולא של המפתח המקורי של השרת!)

הצד השלישי ישווה את ה-Fingerprint שהוא קיבל מהלקוח מול מסד הנתונים של שאר החתימות שלו ויבין שמישהו ערך את התעודה ושינה בה נתונים, במקרה כזה, השרת יחזיר תשובה שלילית ללקוח.

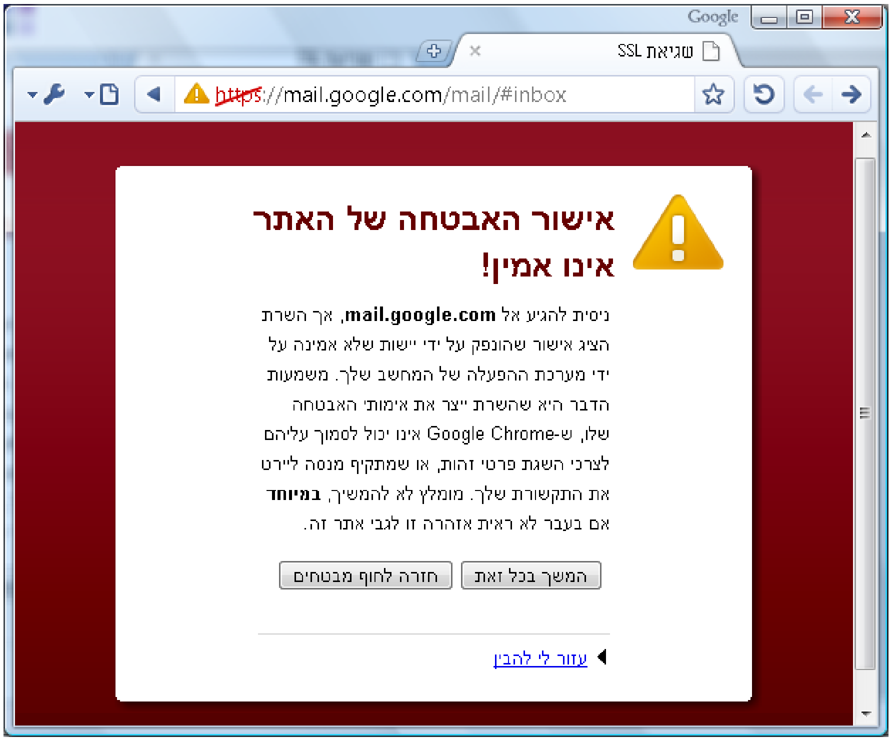
- **שלב רביעי** - הלקוח מקבל תשובה שלילית מהגורם השלישי, עקב כך שהחתימה של התעודה אינה תקינה.



במקרה וקיבלנו תשובה שלילית משרת ה-CA שלנו התקשורת נסגרת. שימו לב שעד כאן לא עבר שום מידע חסוי, המפתח הציבורי של השרת אליו רצינו לשלוח את המידע החסוי הוא לא מידע חסוי, התעודה והשרת שמולו ביצענו את הבדיקה גם הם לא מידע חסוי, הצלחנו לעצור את התקשורת לפני שהתוקף הצליח להשיג מידע חסוי.

נוכל לראות דוגמה נחמדה למקרה כזה אם למשל נגלוש באתר אינטרנט התומך במנגנון SSL כשאנחנו משתמשים בתוכנה Burp - שהיא מעין פרוקסי בין תוכנת הדפדפן שלנו לבין האינטרנט, התוכנה מאפשרת לנו לבצע מניפולציות על המידע העובר בין תוכנת הדפדפן לבין האינטרנט, ובמקרה שמדובר בתקשורת מול אתר התומך ב-SSL, התוכנה מפענחת את המידע, מציגה אותו למשתמש, וכאשר הוא מבקש לבצע את שליחת המידע- היא חותמת עליו בעזרת מפתח שלה (אין לה יכולת לחתום עליו בעזרת התעודה של השרת מפני שאין לה את המפתח הפרטי של השרת) ושולחת אותו לדפדפן- לאחר שהמידע הגיע לדפדפן, הדפדפן מזהה כי המידע נערך ומודיע על כך למשתמש.

דוגמא לניסיון התחברות לשרת הדוא"ל של גוגל, הפועל תחת פרוטוקול SSL בעזרת חיבור הנתון תחת ההזנה:



כיום כמעט (או אולי אפילו כולם?) כל הדפדפנים תומכים בהודעה מסוג זה, ההודעה באה להודיע למשתמש כי מישהו, היכנשהו, במהלך התקשורת שינה/חתם מחדש על התעודה שהונפקה ע"י גוגל.

סיכום + קישורים

אני מקווה שהבנתם איך הולך כל הרעיון של ה-SSL, למה הוא נועד, ואיך הוא פותר לנו את בעיית ההזנה / כיום קיימות מספר מתקפות-Men In The Middle אשר מסוגלות לעקוף מנגנון זה, אך הדבר לא פשוט כל כך והוא דורש תחכום רב.

קישורים רלוונטיים:

- http://www.openssl.org - האתר של OpenSSL - כיום הספריה הגדולה והמתקדמת ביותר (בקוד פתוח תחת GPL) למימוש הפרוטוקול.
- http://www.ietf.org/rfc/rfc2818.txt - ה-RFC של השימוש ב-"HTTP Over TLS".
- http://www.webstart.com/jed/papers/HRM/references/ssl.html - מאמר ישן אבל טוב על הפרוטוקול SSL.
- http://www.kegel.com/ssl/api.html - מאמר על ה-APIs של ה-OpenSSL.

Manual Unpacking

מאת Zerith (אורי / Uri Farkas)

הקדמה

בגיליון שעבר קראתם על Manual Packing או "אריזה ידנית" במאמר המושקע מאת הלל חימוביץ'(HLL) . מאמר זה ידבר על הפעולה ההפוכה בדיוק - Unpacking, החזרת התוכנה למצבה המקורי לפני שארזו אותה.

למה בעצם משתמשים ב-Packers? אריזת התוכנה מטרתה למנוע מעיניים לא רצויות לחקור את התוכנה ולגלות איך היא עובדת - לגלות פרצות אבטחה, לעקוף מנגנוני אבטחה וכדו'. לדוגמא, הרבה ווירוסים ו-Malwares משתמשים ב-Packers כדי למנוע מחוקרי ווירוסים לחקור את הווירוס ולמצוא דרך להשמיד אותו/למצוא את מפעילו. ללא אריזה כל חוקר ווירוס ממוצע יוכל לפתוח את הווירוס ב-debugger ולחקור את קוד האסמבלי שלו ללא בעיה בכלל - זאת תהיה משימה פשוטה למדי. משחקים שונים משתמשים באריזה כדי למנוע מהאקרים לעשות "צ'יטים" או למצוא פרצות אבטחה במשחק ולנצל אותן.

Packers מוכרים ומסחריים בהם משתמשים Malware ומשחקים רבים הם Themida, Winlicense. קיימים לא מעט כלים מסחריים למשימה זאת. לפעמים אפילו משתמשים בכמה פאקרים שונים על אותו הקובץ על מנת להגיע לאבטחה גדולה אפילו עוד יותר.

במאמר זה נדבר על כמה טכניקות נפוצות של Anti-Debugging וכן על שיטות להקשות על ביצוע פעולת Unpacking.

תוכנת המטרה שלנו היום היא UnpackMe ברמה בינונית שכתבה Lena151. ניתן להוריד את התוכנה בכתובת http://tuts4you.com/download.php?view.1114

הכלים שנצטרך:

- OllyDbg - כל גרסה מתחת ל-2.0 תתאים.
- ImpRec - תוכנה לתיקון IAT שהושחתו.
- LordPE - כלי לעריכת כותרות PE Headers של קבצי Object של מיקרוסופט.
- PEiD - כלי חינמי המיועד לזיהוי חתימות של Packers, הצפנות וכו'. ניתן להורדה בכתובת http://www.peid.info

קצת רקע

תהליך ביטול האריזה:

1. **שחזור הקוד המקורי**: כדי לארוז קובץ עלינו להצפין חלקים מהתוכנה ולשבש את הקוד. משום שה-Packer הוא גם ה-Unpacker של התוכנה (הוא משחזר את הקוד המקורי של התוכנה כדי להריץ אותה) - כל מה שעלינו לעשות על מנת להפוך את תהליך האריזה הוא לעקוב אחרי התוכנה שלב-שלב עד שכל ההצפנות בוטלו וכל קוד התוכנה המקורית שוחזר, ולשמור את התוצאה. כותב ה-Packer עשוי לשים בשלב זה מלכודות שונות על מנת להקשות על התהליך.
2. **שחזור נקודת הכניסה**: מכיוון שנקודת הכניסה של ה-Packer שונה מנקודת הכניסה המקורית של התוכנה, עלינו להגיע לנקודת הכניסה המקורית של התוכנה (Original Entry Point - OEP). אפשר להשיג זאת ע"י מעקב אחרי הקוד הרץ, משום שה-Packer **חייב** לקפוץ לנקודת הכניסה המקורית של התוכנה ולהריץ אותה ברגע כלשהו!
3. **תיקון Import Address Table**: הדבר האחרון שיש לעשות בתהליך ביטול האריזה הוא לתקן את ה-Import Address Table (בקיצור IAT), במקרה ששובשה על ידי ה-Packer.

נרחיב מעט על IAT - **Import Address Table** (בעברית: **טבלת פונקציות מיובאות**).

ספריות מיובאות הן DLL-ים (Dynamic Link Libraries) שקובץ ה-EXE מקושר אליהן. כתובות של פונקציות בתוך הספריות הללו אינן קבועות, מכיוון שיוצאות כל הזמן גרסאות חדשות של הספריות והכתובות משתנות בין הגרסאות. לפיכך התוכנה אינה יכולה להשתמש בכתובות קבועות מראש, וחייבת דרך כלשהי לגשת לפונקציות מבלי לקמפל מחדש את התוכנה או לבדוק את כתובתן בזמן ריצה.

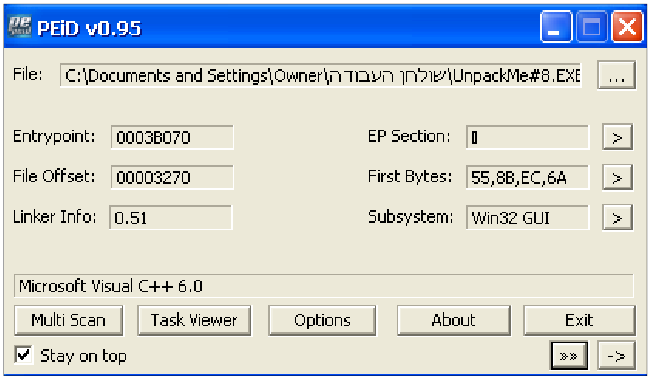
כל זה נעשה על ידי ה-IAT, ה-IAT הוא טבלה של מצביעים לפונקציות המתמלא ע"י ה-Windows Loader בעת טעינת הקובץ והספריות המקושרות אליו לזיכרון, במקום לשנות כל קריאה לפונקציה שתקרא לכתובת המתאימה, פשוט נקרא לכתובת השמורה ב-IAT.

ה-Packer לעיתים משנה את ה-IAT ומפנה את הכתובות לפונקציות משלו אשר קוראות בעקיפין לכתובות האמיתיות של ה-Imports, על מנת להקשות עלינו עוד יותר. עלינו לתקן את ה-IAT במקרה שהיא שונתה על ידי ה-Packer.

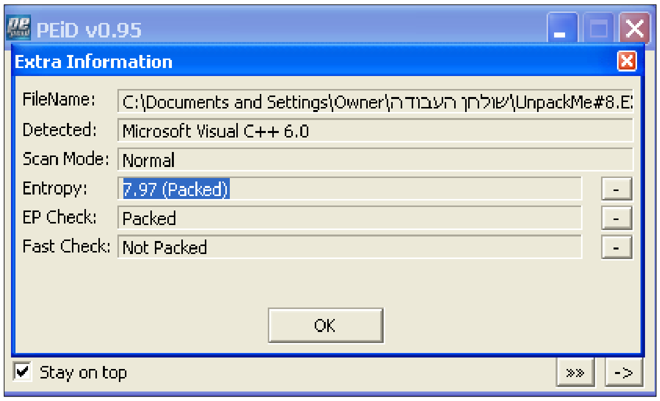
מתחילים

בהרצה ראשונה של המטרה מחוץ ל-Debugger, ניתן להבחין כי נפתח חלון פשוט הנראה כמו פנקס הרשימות. נפתח את התוכנה עם PEiD ונראה האם הוא מזהה משהו.

בבדיקה ראשונית, PEiD לא מוצא Packer וכותב לנו שהתוכנה נכתבה ב-Microsoft Visual C++ 6.0.



אך אם נלחץ על Extra Information ([>>]), נוכל לראות כי הקובץ אכן ארוז!

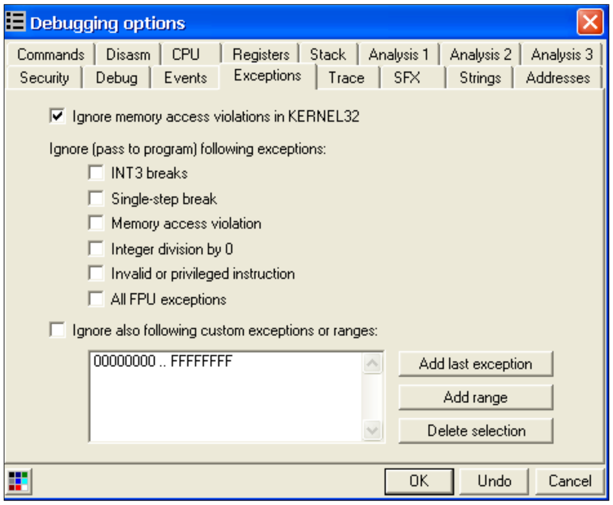


אחרי שראינו כי הקובץ ארוז, ננסה להריצו ב-Olly. בניסיון הראשון להריץ את התוכנה ב-Olly, התוכנה מגיעה לחריגה שבה אינה מסוגלת לטפל, וקורסת.

מה זה בעצם אומר? אנו שמים לב כי זרימת התוכנה מתחת ל-Debugger שונה מזרימת התוכנה מחוצה לו. זה סימן מובהק שהקובץ מוגן בהגנת Anti-Debugging. Anti-Debugging הן טכניקות שנועדו לזהות את הדיבאגר, ולשבש את זרימת התוכנה כתוצאה מזיהויו. נסביר לכם על כמה מטכניקות אלו בהמשך.

בניסיון לראות את מקור החריגה, נבטל את העברת כל החריגות לתוכנה ב-Olly, ונראה אם יש חריגות עוד לפני החריגה שלא ניתן לטפל בה.

ניתן לעשות זאת על ידי לחיצה על Options -> Debugging Options -> Exceptions -> והורדת סימון ה-V מכל האפשרויות חוץ מ-Ignore Memory Access violations in Kernel32. (משום שלא ממש מעניין אותנו אם יש חריגות שלא קשורות לקוד שלנו).



אחרי שביטלנו את העברת כל החריגות לטיפול התוכנה, נריץ מחדש את התוכנה....וניתקל בחריגה!

אנחנו עדיין לא יודעים שחריגה זו היא הגורמת לחריגה שראינו קודם ולכן נמשיך להריץ את התוכנה. אחרי הרצה שנייה נתקל בעוד חריגה. אם נמשיך להריץ את התוכנה נגיע לחריגה השלישית - ונקרוס. בבדיקה ניתן לראות כי אחרי החריגה השנייה מסלול הקוד תמיד מגיע לחריגה השלישית. אנו לומדים כי ככל הנראה עלינו למנוע את התרחשות החריגה השנייה כדי לפתור את הקריסה.

אנחנו בחריגה השנייה, אם נלך קצת לאחור בקוד (Backtrace) ונרצה לראות את מקור החריגה - נתקל בקריאה מוזרה, CALL EAX. ואחריה בדיקה של הערך החוזר וקפיצה בהתאם.

מדוע קריאה זו חשודה? משום שאנחנו לא יכולים לדעת מראש איזה פונקציה תיקרא פה. התוכן של EAX נקבע רק בזמן ריצה. אנחנו לא יכולים לדעת מה היה התוכן של EAX ברגע שנקרא לפני ריצת התוכנה. טריק זה הינו דרך של ה-Packer להסוות קריאות שהוא לא רוצה שנראה.

נשים Breakpoint על הקריאה כדי לחקור אותה. נפעיל מחדש את התוכנה, ונריץ עד הקריאה.

```
00291228    F3:A4           REP MOVS BYTE PTR ES:[EDI],BYTE PTR DS:
0029122A    8BF3            MOV ESI,EBX
0029122C    8DBD 491F0010   LEA EDI,DWORD PTR SS:[EBP+10001F49]
00291232    012F            ADD DWORD PTR DS:[EDI],EBP
00291234    016F 04         ADD DWORD PTR DS:[EDI+4],EBP
00291237    016F 08         ADD DWORD PTR DS:[EDI+8],EBP
0029123A    8D8D 2D1F0010   LEA ECX,DWORD PTR SS:[EBP+10001F2D]
00291240    51              PUSH ECX
00291241    E8 57010000     CALL 0029139D
00291246    6A 00           PUSH 0
00291248    56              PUSH ESI
00291249    E8 74050000     CALL 002917C2
0029124E    8B85 291F0010   MOV EAX,DWORD PTR SS:[EBP+10001F29]
00291254    85C0            TEST EAX,EAX
00291256  v 74 07           JE SHORT 0029125F
00291258    FFD0            CALL EAX                                   kernel32.IsDebuggerPresent
0029125A    85C0            TEST EAX,EAX
0029125C  v 74 01           JE SHORT 0029125F
0029125E    4B              DEC EBX
0029125F    8B4E 2C         MOV ECX,DWORD PTR DS:[ESI+2C]
00291262    898D 591F0010   MOV DWORD PTR SS:[EBP+10001F59],ECX
00291268    6A 40           PUSH 40
0029126A    68 00100000     PUSH 1000
0029126F    51              PUSH ECX
00291270    6A 00           PUSH 0
00291272    FF95 651F0010   CALL DWORD PTR SS:[EBP+10001F65]
00291278    8985 551F0010   MOV DWORD PTR SS:[EBP+10001F55],EAX
0029127E    56              PUSH ESI
0029127F    E8 F6030000     CALL 0029167A
00291284    8D8D D11D0010   LEA ECX,DWORD PTR SS:[EBP+10001DD1]
0029128A    85C0            TEST EAX,EAX
0029128C  v 0F85 94000000   JNZ 00291326
00291292    56              PUSH ESI
00291293    E8 40030000     CALL 002915D8
00291298    56              PUSH ESI
00291299    E8 55020000     CALL 002914F3
0029129E    90              NOP
0029129F    90              NOP
002912A0    90              NOP
002912A1    90              NOP
002912A2    90              NOP
002912A3    90              NOP
002912A4    6A 01           PUSH 1
002912A6    56              PUSH ESI
```

אנו רואים כי אכן צדקנו והקריאה באמת הייתה חשודה. בכתובת זו ישנה קריאה לפונקציה IsDebuggerPresent (הבודקת אם התוכנה רצה מתחת לדיבאגר), בדיקה של הערך החוזר - וקפיצה בהתאם, טכניקת Anti-Debugging נפוצה.

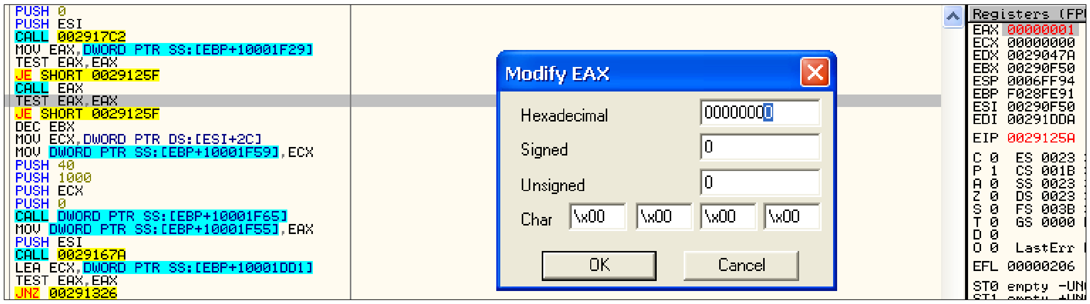
הסבר קצר על IsDebuggerPresent:

הפונקציה IsDebuggerPresent משתמשת במבנה נתונים הנקרא Process Environment Block (מבנה נתונים המכיל פרטים על התהליך הרץ), המכיל בין השאר שדה הנקרא "IsDebugged" שמייצג האם התהליך רץ מתחת לדיבאגר - או לא.

```
7C8130A1     90              NOP
7C8130A2     90              NOP
7C8130A3  r$ 64:A1 180000( MOV EAX,DWORD PTR FS:[18]
7C8130A9  .  8B40 30         MOV EAX,DWORD PTR DS:[EAX+30]
7C8130AC  .  0FB640 02       MOVZX EAX,BYTE PTR DS:[EAX+2]
7C8130B0  L. C3              RETN
7C8130B1     90              NOP
7C8130B2     90              NOP
7C8130B3     90              NOP
7C8130B4     90              NOP
```

- FS - הוא הסגמנט המייצג את כתובות הזיכרון המתחילות מ-0x7FFDF000 (כברירת מחדל).
- FS[18] הוא מצביע ל-TEB (Thread Environment Block, בדומה ל-PEB מכיל פרטים על החוט הרץ).
- TEB+0x30 מצביע ל-PEB.
- [PEB+0x02] הוא השדה IsDebugged ב-PEB.

אנו רואים כי IsDebuggerPresent מחזיר את הערך של שדה IsDebugged. שינוי של ערך זה ב- Process Environment Block (PEB) לא משנה דבר מבחינת פעולת התוכנה ולא ימנע מאיתנו לעשות Debugging לתוכנה. כל מה שעלינו לעשות על מנת לעבור את הקריאה ל-IsDebuggerPresent הוא לשנות את הערך החוזר מהפונקציה מ-1 ל-0.



נריץ את הפונקציה עד הסוף (עד ה-RETN), ונראה שאין חריגה!

שיטה נוספת של anti-debugging: שילוב INT3

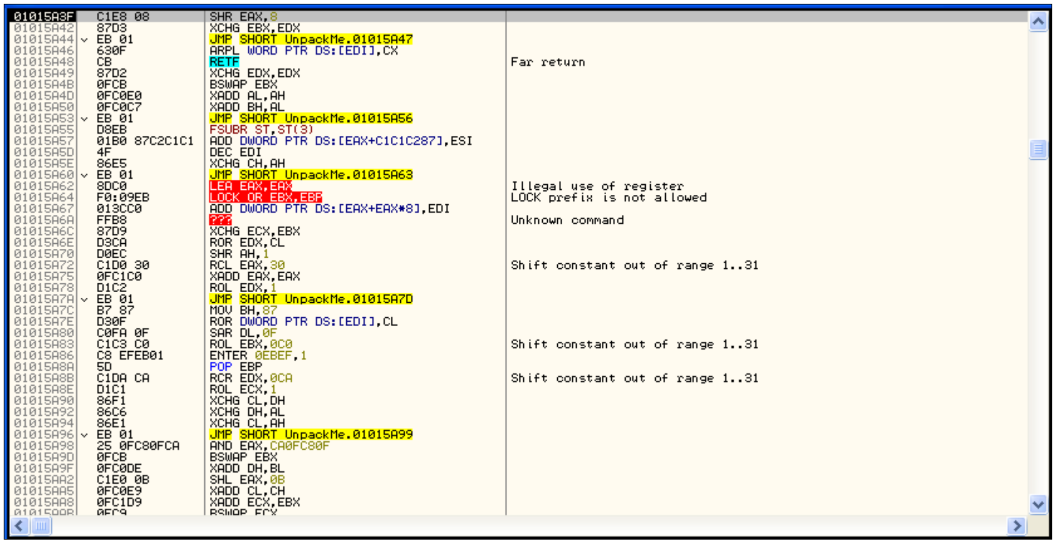
לפני שנמשיך בניתוח התוכנה הנתונה נציג טכניקה נפוצה נוספת של Anti-Debugging, שהיא הכנסת ההוראה INT3 באמצע קוד רגיל.

INT3 הינו Software Breakpoint. במקרה של Olly, ושל Debuggers אחרים - כאשר אנחנו שמים Breakpoint בתוכנה (Software BP) ה-Debugger מכניס את ההוראה INT3 כבית הראשון של ההוראה, ובזמן ריצה כאשר המחשב נתקל בהוראה INT3 הריצה נעצרת והשליטה חוזרת למשתמש ול-Debugger.

מה קורה כשהקוד עצמו מכיל את ההוראה INT3? מחוץ לדיבאגר ההוראה תיצור חריגה וזרימת התוכנה תעבור ל-Exception Handler. כאשר התוכנה בשליטת הדיבאגר, הוא חושב שה-INT3 הוא בעצם אחד מה-Software Breakpoints שלו - השליטה לא מועברת ל-Exception Handler והקוד ממשיך מאותו המקום.

למתעניינים בטכניקות Anti-Debugging נוספות, נמליץ על קריאת מאמרים בנושא. מאמר מומלץ לדוגמא הוא http://www.veracode.com/images/pdf/whitepaper_antidebugging.pdf.

נמשיך לחקור את התוכנה. "נצעד" בקוד וננסה להבין אותו. אחרי הפונקציה שנתקלנו בה קודם, אנחנו מגיעים אל הקוד הבא:



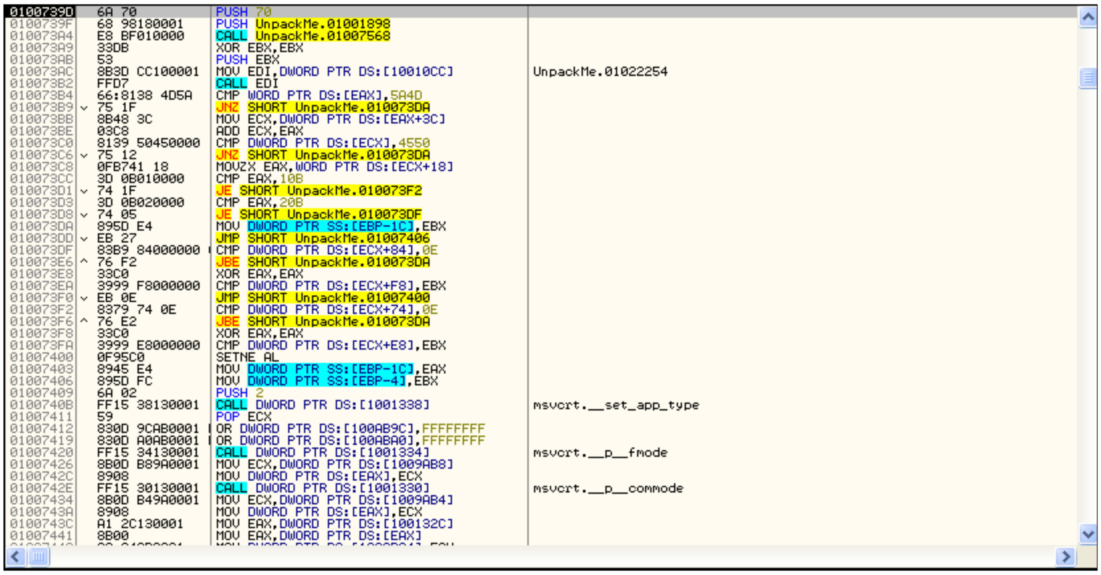
```
01015A3F    C1E8 08          SHR EAX,8
01015A42    87D3             XCHG EBX,EDX
01015A44  v EB 01            JMP SHORT UnpackMe.01015A47
01015A46    630F             ARPL WORD PTR DS:[EDI],CX
01015A48    CB               RETF                                 Far return
01015A49    87D2             XCHG EDX,EDX
01015A4B    0FCB             BSWAP EBX
01015A4D    0FC0E0           XADD AL,AH
01015A50    0FC0C7           XADD BH,AL
01015A53  v EB 01            JMP SHORT UnpackMe.01015A56
01015A55    D8EB             FSUBR ST,ST(3)
01015A57    01B0 87C2C1C1    ADD DWORD PTR DS:[EAX+C1C1C287],ESI
01015A5D    4F               DEC EDI
01015A5E    86E5             XCHG CH,AH
01015A60  v EB 01            JMP SHORT UnpackMe.01015A63
01015A62    8DC0             LEA EAX,EAX                          Illegal use of register
01015A64    F0:09EB          LOCK OR EBX,EBP                      LOCK prefix is not allowed
01015A67    013CC0           ADD DWORD PTR DS:[EAX+EAX*8],EDI
01015A6A    FFB8             ???                                  Unknown command
01015A6C    87D9             XCHG ECX,EBX
01015A6E    D3CA             ROR EDX,CL
01015A70    D0EC             SHR AH,1
01015A72    C1D0 30          RCL EAX,30                           Shift constant out of range 1..31
01015A75    0FC1C0           XADD EAX,EAX
01015A78    D1C2             ROL EDX,1
01015A7A  v EB 01            JMP SHORT UnpackMe.01015A7D
01015A7C    B7 87            MOV BH,87
01015A7E    D30F             ROR DWORD PTR DS:[EDI],CL
01015A80    C0FA 0F          SAR DL,0F
01015A83    C1C3 C0          ROL EBX,0C0                          Shift constant out of range 1..31
01015A86    C8 EFEB01        ENTER 0EBEF,1
01015A8A    5D               POP EBP
01015A8B    C1DA CA          RCR EDX,0CA                          Shift constant out of range 1..31
01015A8E    D1C1             ROL ECX,1
01015A90    86F1             XCHG CL,DH
01015A92    86C6             XCHG DH,AL
01015A94    86E1             XCHG CL,AH
01015A96  v EB 01            JMP SHORT UnpackMe.01015A99
01015A98    25 0FC80FCA      AND EAX,CA0FC80F
01015A9D    0FCB             BSWAP EBX
01015A9F    0FC0DE           XADD DH,BL
01015AA2    C1E0 0B          SHL EAX,0B
01015AA5    0FC0E9           XADD CL,CH
01015AA8    0FC1D9           XADD ECX,EBX
```

קוד זה נראה מאוד מוזר, זהו Obfuscation Code (קוד הטעייה) - קוד שנועד לבלבל את הריברסר. לרוב קטע קוד זה לא משפיע כלל על התוכנה ואין צורך בו.

כשאתם נתקלים בקוד מסוג זה, היזהרו מאוד לא לאבד שליטה על התוכנה. (להיכנס בתוך הקריאות ולא מעליהן, וכו'). אין הרבה דרכים לדעת מהו קוד הטעייה ולזהות אותו, אך עם הניסיון תדעו כבר להבחין כשתראו אחד.

דרך לדעת אם קיים קוד הטעייה בקובץ הוא להסתכל ב-PEiD, איפה שראינו שכתוב Entropy: 7.97 (Packed). Entropy בתרגום חופשי הוא "רמת המבולגנות" של הקובץ, כאשר 10 היא הדרגה הגבוהה ביותר.

אחרי הרבה קוד הטעייה, אנו מגיעים לפונקציה שכבר נראית בטוחה (ללא קוד הטעייה). חקירה של הפונקציה עד סופה, תביא אותנו לקריאה חשודה נוספת - CALL EAX. עם זאת, אנחנו רואים כי קריאה זאת אינה קריאה ל-API, וכאן עולה השאלה - למה ירצו להסוות קריאה זו?



אנו רואים כי זו נקודת הכניסה המקורית של התוכנה! איך אנו יודעים זאת? אם פעם פתחתם קובץ שקומפל ב-++Microsoft Visual C בדיבאגר, ראיתם בדיוק את נקודת הכניסה הזו. (ניתן להבחין בכך לפי הקריאות ל-msvcrt). כאמור, נקודה זו היא ה-Original Entry Point (OEP).

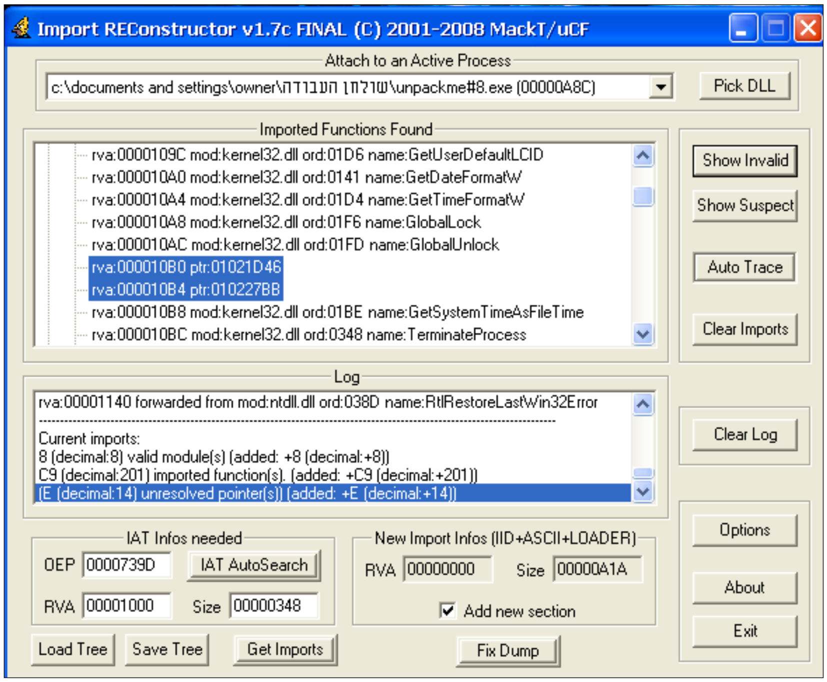
נפתח את LordPE, ונעשה לתהליך Dump, משום שהקוד האמיתי של התוכנה כבר לא ארוז. (Dump הוא העתקה של זיכרון התהליך הרץ ברגע נתון, לתוך קובץ חדש, למשל EXE).

על מנת לעשות Dumping מה שצריך לעשות זה לפתוח את LordPE, לבחור את התהליך הרצוי, לחיצה ימנית -> ולבחור Full Dump.

תיקון ה-IAT (Import Address Table)

נפתח את התוכנה ImpRec, תוכנה אשר מיועדת לתיקון Imports. נכניס את כל הערכים הרצויים (זכרו להחליף את ה-EP של ה-Packer ב-OEP!) ונלחץ על Get Imports. במקרה שלנו, הערך היחיד שיש להכניס ב-ImpRec הוא ה-RVA של ה-OEP שמצאנו. RVA הוא כתובת וירטואלית יחסית לכתובת הבסיס (ImageBase). מושג זה הוסבר במאמר של HLL על Manual Packing בגיליון שעבר.

נוודא שכל ה-Imports תקינים על ידי לחיצה על Show Invalid. במקרה שלנו יש Imports מושחתים ☹



נצטרך לתקן את הערכים המושחתים.

כמו שאנחנו רואים ב-ImpRec, ה-IAT שוכן ב-RVA 0x00001000. נפתח את ה-Debugger בכתובת זו (יש לזכור כי זאת כתובת יחסית ל-ImageBase, ובמקרה הזה ה-ImageBase הוא 0x01000000).

אנו רואים כי 0x010010B0 היא הכתובת של ה-Import הראשון ששונה. מכיוון שבתוכנה המקורית ה-IAT לא היה מושחת כנראה שה-Packer שינה בזמן ריצה את ה-IAT, אז בשביל לראות מתי הכתובת שונתה, נשים Hardware Breakpoint on write בכתובת 0x010010B0 ונריץ מחדש את התוכנה. (לא לשכוח לעקוף מחדש את IsDebuggerPresent).

אחרי ההרצה אנחנו מגיעים לנקודה בה 0x010010B0 שונתה. ניתן לראות בקוד באזור כי ה-Packer בודק אילו Imports לשנות או לא. ננתח קוד זה.

```
0101CEB0    MOV EAX,DWORD PTR SS:[EBP+8]
0101CEB3    MOV ECX,DWORD PTR DS:[EAX]
```

פה כתובתה של הפונקציה המיובאת נשמרת ב-ECX וניתנת כפרמטר לפונקציה:

```
0101CEB5    PUSH ECX
0101CEB6    MOV ECX,DWORD PTR DS:[102D034]
0101CEBC    CALL UnpackMe.01022B4C
```

קריאה לפונקציה שבודקת אם עליה לשנות ב-IAT את הכתובת של הפונקציה הניתנה כפרמטר

```
0101CEC1    MOV DWORD PTR SS:[EBP-8],EAX
0101CEC4    CMP DWORD PTR SS:[EBP-8],0
```

בדיקה של הערך החוזר מהפונקציה - הפונקציה מחזירה את כתובתה של הפונקציה המחליפה במקרה ויש להחליף את הכתובת ב-IAT, ו-0 במקרה שלא.

```
0101CEC8    JE SHORT UnpackMe.0101CF0F
```

קפיצה בהתאם לתוצאה:

```
0101CECA    LEA EDX,DWORD PTR SS:[EBP-10]
0101CECD    PUSH EDX
0101CECE    PUSH 4
0101CED0    PUSH 4
0101CED2    MOV EAX,DWORD PTR SS:[EBP+8]
0101CED5    PUSH EAX
0101CED6    CALL DWORD PTR DS:[102872C]
```

קריאה לפונקציה kernel32.VirtualProtect על מנת לשנות את הגנת הדף על ארבעת בתי הכתובת ב-IAT. הפונקציה VirtualProtect משנה את הגנת הזיכרון הוירטואלי של התהליך בכתובת הניתנה כפרמטר במספר הבתים (שגם כן ניתנו כפרמטר). כברירת מחדל ההגנה היא PAGE_READEXECUTE (לא ניתן לכתוב לאזור הזיכרון), בקריאה זו ההגנה משתנה ל-PAGE_READWRITE:

```
0101CEDC    TEST EAX,EAX
0101CEDE    JNZ SHORT UnpackMe.0101CEEA
0101CEE0    MOV ECX,EF00000B
0101CEE5    CALL UnpackMe.0101FA32
```

קריאה זאת לא מורצת במקרה והקריאה ל-VirtualProtect הצליחה:

```
0101CEEA    MOV ECX,DWORD PTR SS:[EBP+8]
```

השגת מצביע לכתובת ב-IAT שיש לשנות:

```
0101CEED    MOV EDX,DWORD PTR SS:[EBP-8]
```

מצביע לכתובת המחליפה:

```
0101CEF0    MOV EAX,DWORD PTR DS:[EDX]
0101CEF2    MOV DWORD PTR DS:[ECX],EAX
```

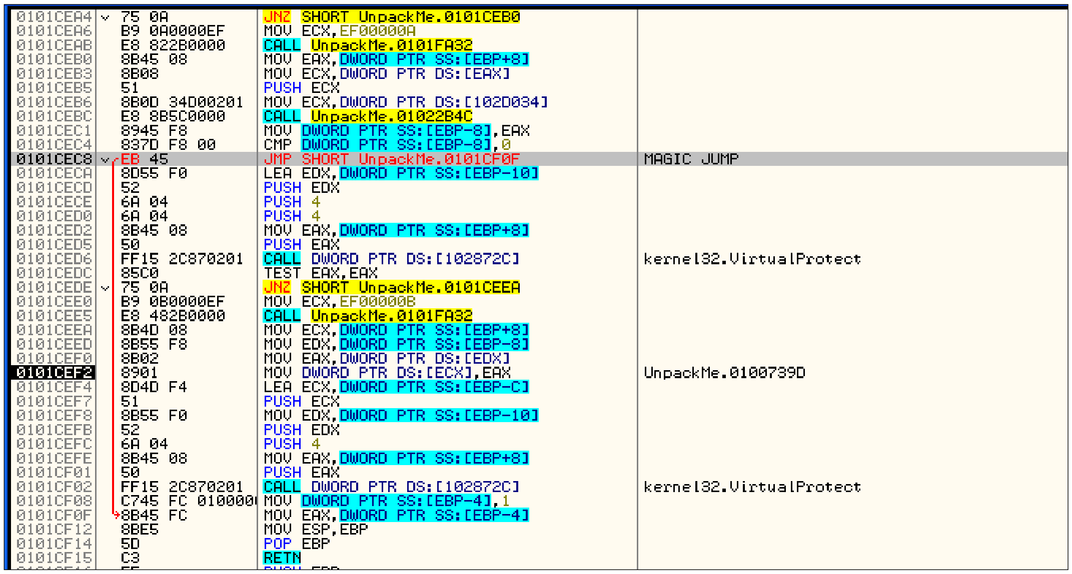
כאן מתבצעת ההחלפה:

```
0101CEF4    LEA ECX,DWORD PTR SS:[EBP-C]
0101CEF7    PUSH ECX
0101CEF8    MOV EDX,DWORD PTR SS:[EBP-10]
0101CEFB    PUSH EDX
0101CEFC    PUSH 4
0101CEFE    MOV EAX,DWORD PTR SS:[EBP+8]
0101CF01    PUSH EAX
0101CF02    CALL DWORD PTR DS:[102872C]
```

kernel32.VirtualProtect - כאן משחזרים את ההגנה הקודמת של הדף על מנת למנוע בעיות בעתיד:

```
0101CF08     MOV DWORD PTR SS:[EBP-4],1
0101CF0F     MOV EAX,DWORD PTR SS:[EBP-4]
0101CF12     MOV ESP,EBP
0101CF14     POP EBP
0101CF15     RETN ; Exit
```

כל מה שעלינו לעשות על מנת למנוע מה-Packer לשנות את ה-Imports הוא לשנות את הקפיצה כדי שתמיד תקפוץ.



לקפיצה מסוג זה קוראים Magic Jump והיא נפוצה בקרב Packers שונים. נריץ את התוכנה כדי שה-Packer יכתוב את כל כתובות ה-Imports המקוריים. נפתח מחדש את ImpRec ונכניס את אותם הערכים.

ו...הכול תקין! ☺ כל מה שנשאר לעשות הוא ללחוץ על Fix Dump ולבחור את ה-Dump שיצרנו קודם לכן, והתוכנה תרוץ ללא אריזה.

- המהדרין יכולים להכנס ל-LordPE ולחתוך את איזור הזיכרון שבו שכן ה-Packer כדי להקטין את גודל הקובץ, משום שאנחנו לא צריכים אותו יותר.

להמשך ההעמקה בתחום...

אם ברצונכם ללמוד עוד על התחום, הרשו לי להפנות אתכם למדריכים של Lena151, שלדעתי הם חובה לכל מתחיל בתחום: http://tuts4you.com/download.php?list.17

וירוסים - שיטות טעינה

מאת אפיק קסטיאל (cp77fk4r)

לאחר שוירוס או תולעת מצליחים להריץ את עצמם על מחשב-קורבן, אחד הנושאים הקריטיים מצד כותב הוירוס היא לדאוג לפעם הבאה בה הוירוס ירוץ. כאן קיים tradeoff מצד כותב הוירוס. מצד אחד מטרת כותב הוירוס היא להשתמש בדרכים אפקטיביות ורבות ככל שניתן לדאוג שהוירוס ירוץ שוב ושוב על המחשב, כך שלמשתמש יהיה קשה יותר להפטר מהוירוס, והוירוס יצליח לשמור על אורך חיים ארוך יותר. מצד שני, ככל שהוירוס משנה יותר דברים במערכת ההפעלה - כך יגדל הסיכוי שהוא יתגלה.

במאמר זה נסקור מספר דרכים, או "טכנולוגיות", שבהן הוירוסים והתולעים משתמשים בכדי לגרום למערכת/למשתמש לטעון את עצמם מבלי ידיעת המשתמש. בנוסף נסקור דרכים ופעולות אותן ניתן לבצע בכדי להתמודד נגד וירוסים המשתמשים בדרכים אלו, בכדי לגרום למחשבכם להיות מאובטח יותר מפני האיומים.

Startup

הדרך הפשוטה וגם המוכרת ביותר להפעלת תוכנה עם עליית המחשב היא שימוש בתיקיה Startup. מערכת ההפעלה Windows מאפשרת למשתמש לקבוע תוכנות שירוצו בעת טעינתה ע"י הוספת קובץ ההפעלה של התוכנה (או קיצור דרך אליו) בתיקיה Startup.

ב-Windows XP מיקומה של התיקיה הוא:

```
C:\Documents and Settings\[User]\Start Menu\Programs
```

ובמערכת Vista מיקומה של התיקיה הוא:

```
C:\Users\[User]\AppData\Roaming\Microsoft\Windows\Start Menu\Programs
```

כל קובץ הממוקם בתיקיה הנ"ל בזמן טעינתה של מערכת ההפעלה יורץ לאחר שהמערכת תעלה.כיום השימוש בתיקיה זו אינו רב, וכמעט ולא ניתן למצוא וירוסים או תולעים אשר משתמשים בה לצורך טעינתם - קל מאוד לאתר שימוש בה, תוכן התיקיה מופיע בתפריט ה-"Program Files" הממוקם תחת תפריט ה-"Start" ונגיש מאוד אף למשתמש הממוצע.

System Configuration Loading Files

השיטה הבאה שנציין בה וירוסים וטרויאנים משתמשים היא שימוש במספר קבצי האצווה וקבצי ה-ini בהם Windows משתמשת. קבצים אלה כוללים קבצים או פקודות שעל מערכת ההפעלה לבצע בעת עלייתה. שמות הקבצים הנקראים באופן אוטומטי:

```
%homedrive%\Autoexec.bat
%homedrive%\Config.sys
%windir%\Win.ini
%windir%\Wininit.ini
%windir%\System.ini
```

- %homedrive% זהו הכונן בו מותקנת מערכת ההפעלה.
- %windir% זו הספריה בה מצוייה מערכת ההפעלה (בד"כ C:\Windows).

שיטה זו מעט קשה יותר לאיתור למשתמשי המחשב הממוצעים מאשר שימוש בספריה Startup, ואפשר למצוא שימוש בה בוירוסים ישנים, אך כיום לא ניתן למצוא וירוסים המשתמשים בה.

<u>שימוש בקבצים Autoexec.bat ו-Config.sys:</u>

שני הקבצים Autoexec.bat ו-Config.sys הם קבצי אצווה רגילים. כדי לגרום להם להריץ אפליקציות אפשר להשתמש בפקודה הקריאה Call. דוגמה לשימוש:

```
Call %temp%\virus.exe
```

בשאר הקבצים ניתן להשתמש רק במערכת הישנות מ-XP, כגון 98 ודומותיה. על מנת להתגונן מוירוסים הנמצאים בקבצים אלו ניתן לסרוק תקופתית קבצים אלה לשינויים או תוספות חשודות.

<u>שימוש בקבצים Win.ini ו-Wininit.ini.</u>

הקבצים win.ini, Wininit.ini הם קבצי ini. קבצים אלו מחולקים על ידי תוויות (Labels). תחת [Windows] נמצאות תכונות שנקראות על ידי מערכת ההפעלה עם עלייתה, באופן הבא:

```
[windows]
LOAD=%temp%\virus.exe
```

או באופן הבא:

```
[windows]
RUN=%temp%\virus.exe
```

מערכת הקבצים מבצעת שימוש בקובץ Wininit.ini כדי לטעון הגדרות ושירותים לאחר התקנת מערכת ההפעלה - קבצים והגדרות שהמערכת לא יכולה לטעון בעת ההתקנה.

לאחר שמערכת ההפעלה תטען את המידע הקיים ב-Wininit.ini היא תשנה אותו מיד ל-Wininit.BAK, כך שבפעם הבאה המערכת לא תמצא שום קובץ "Wininit.ini" ולכן לא תבצע שום הרצה של התוכן הקיים בו.

<u>שימוש בקובץ-System.ini.</u>

מערכת הקבצים משתמשת בקובץ הנ"ל בכדי לטעון דרייברים נחוצים להפעלה תקינה של מערכת ההפעלה, הקובץ עצמו הוא גם קובץ ini, וה-label שממנו נטענים הדרייברים הוא: "[386enh]", והשימוש בו הוא בדיוק כמו השימוש בקבצי ה-ini הקודמים.

<u>מעקב והתגוננות</u>

בכדי לעבוד באופן נח ומסודר עם הקבצים הנ"ל מייקרוסופט הוסיפו למערכת ההפעלה עורך קטן בשם "System Configuration Editor", והוא ממוקם ב:

```
%windir%\System32\sysedit.exe
```

אפשר להשתמש בכלי כדי לבדוק במהירות את כל הקבצים שהצגנו ולאתר בהם שינויים חשודים.

Startup Regedit Values

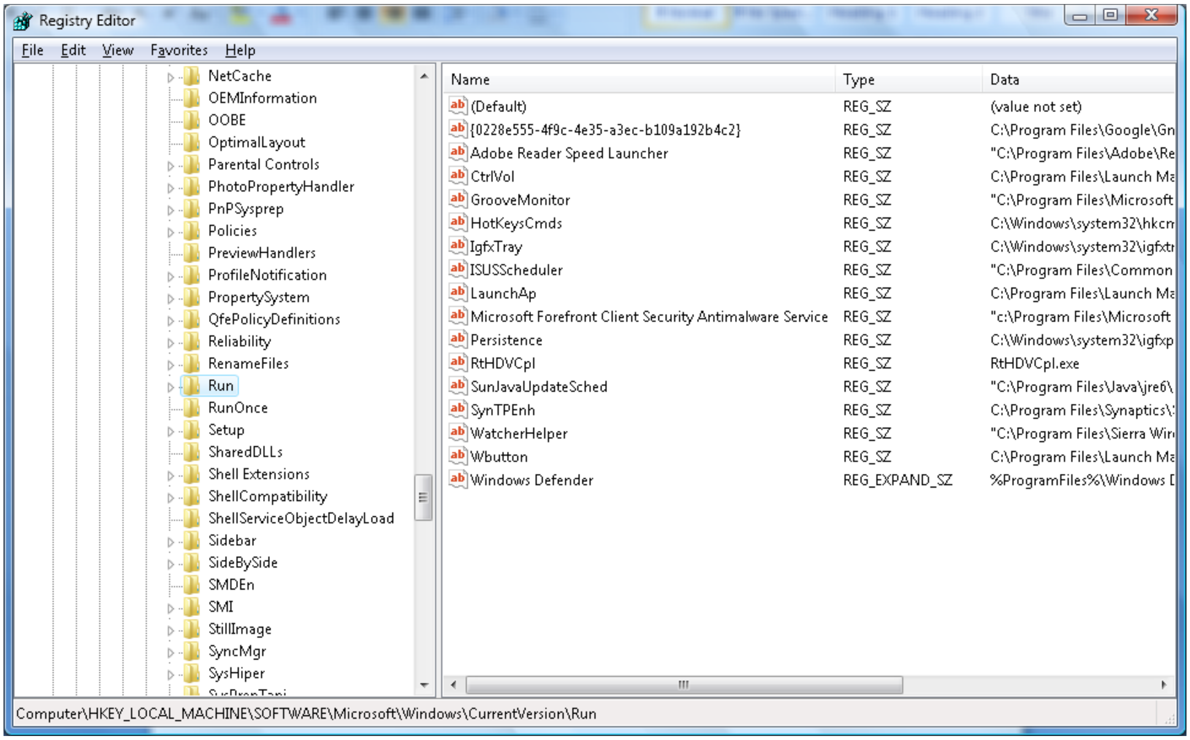
עורך הרישום של מערכת ההפעלה (registry editor) טומן בחובו הרבה מאוד נסתרות. כחלק מאפיונו הוא גם אחראי על טעינת קבצים בעת מספר אירועים, אירועים כגון טעינת מערכת ההפעלה, התחברות משתמש מסויים, כניסה לכונן מסויים, ואף אירועים חיצוניים כגון - חיבור התקן USB ליציאת ה-USB במחשב, שימוש בפרוטוקולים ועוד נושאים רבים. נציג מעט ערכים הנוגעים להפעלה אוטומטית של תוכנות.

<u>טעינת מערכת ההפעלה:</u>

זהו אולי המפתח שוירוסים משתמשים בו לעיתים הקרובות ביותר:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
```

כל ערך שיופיע תחת המפתח הנ"ל יטען בעת טעינת מערכת ההפעלה.



וירוס המעוניין להוסיף את עצמו למפתח זה משתמש בפקודה "Reg" עם המתג-"add", למשל:

```
Reg add HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /v
Virus /d %temp%\virus.exe
```

הפקודה תוסיף עוד ערך בשם "Virus" המכיל את מיקום הקובץ שיש להריץ-"%temp\virus.exe" אשר יטען כל פעם בעת טעינת מערכת ההפעלה.

כמובן שוירוסים ושאר מזיקים ישתמשו בשמות פחות חשודים כגון-"svchosts.exe" או "explorer.exe", ולכן חשוב מאוד לבדוק מה הערך המוכנס ל-Data (מיקום הקובץ אותו המערכת תריץ) ולברר האם הקובץ הנ"ל אכן שייך למערכת ההפעלה או לא.

קיימים עוד מפתחות כאלה בעורך הרישום, השימוש בהם נפוץ פחות, אך פעולתן זהה (ברב המקרים), המפתחות הם:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce\Setup
```

במערכת ההפעלה-XP, קיים גם המפתח הבא:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnceEx
```

כל הערכים אשר ימוקמו תחת המפתחות הנ"ל יטענו בעת טעינת מערכת ההפעלה, הערכים הבאים יטענו רק בעת טעינת משתמש ספציפי (המשתמש תחתיו הריצו את הפקודה):

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServices
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\RunServicesOnce
```

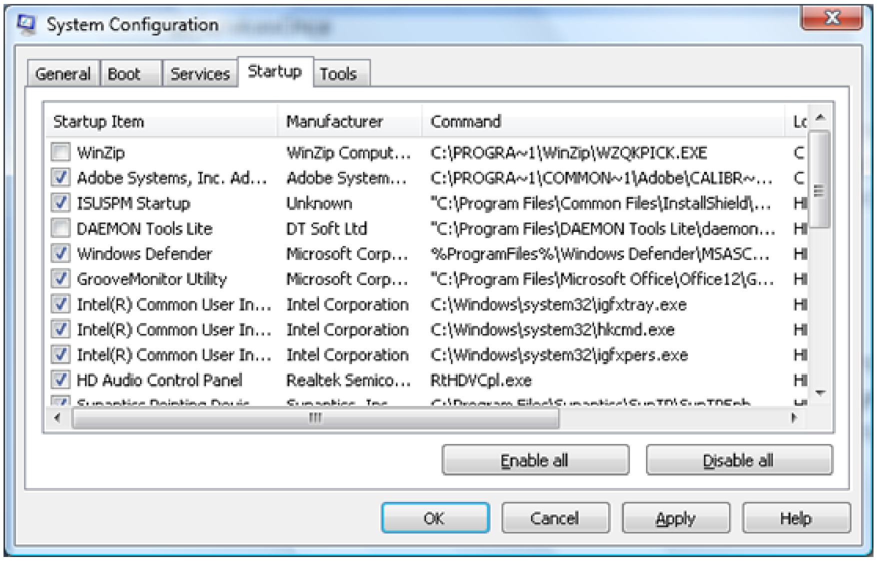
בעורך הרישום תחת מערכות ההפעלה-XP, NT ו-Server2003, ישנו מפתח המתנהג באופן זהה, אשר תפקידו לטעון את הקובץ userinit.exe המקושר לקביעת תצורת המשתמש, המפתח הוא:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
```

ובכדי לטעון בעזרתו אפליקציות שונות, פשוט מאוד מוסיפים "," (פסיק) לאחר המיקום של הקובץ userinit.exe ומוסיפים בו את המיקום של האפליקציה שברצוננו לטעון, לדוגמא:

```
%windir%\system32\userinit.exe,%temp%virus.exe
```

לאחר טעינת כל הערכים מהמפתחות שציויינו עד כה מערכת הקבצים תרכז את כולם לתוך רשימה מסודרת בכדי להקל על ניהול המערכת, את הרשימה הנ"ל אפשר למצוא תחת החוצץ "Startup" באפליקציה Msconfig.exe:



עורך הרישום מנהל עוד מספר מפתחות וערכים אשר תולעים ווירוסים "מתקדמים" מנצלים בכדי לשפר את אורך חייהם, ערכים אלה לא נטענים בעת עליית מערכת ההפעלה או התחברות משתמש מסויים, אך ערכים אלה מנהלים אירועים המתקיימים מספר רב של פעמים המספיק בכדי לשמור על פעילות "תקינה" של אותה התולעת, לדוגמא:

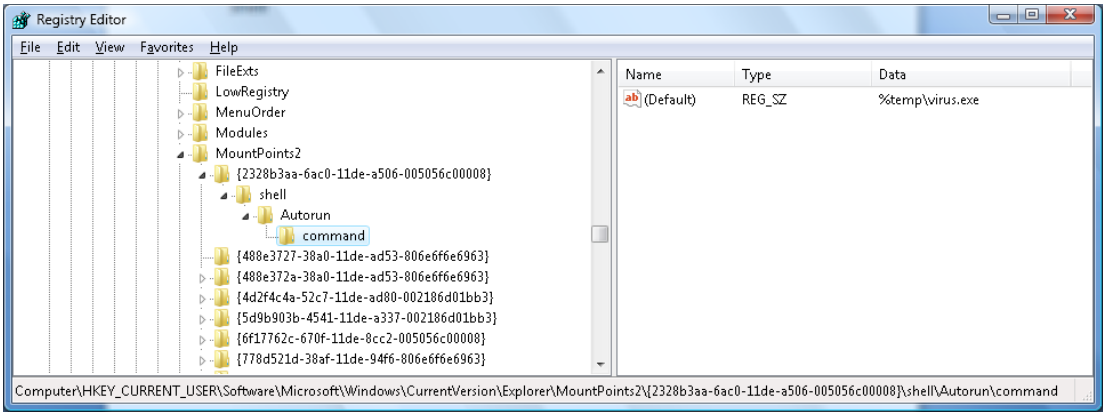
בעת הכנסת התקן USB או כל כונן (אפילו קשיח) עורך הרישום "זוכר" באיזה תצורת טעינה המשתמש בחר בכדי לפתוח אותו (טעינת המידע לנגן ה-Media, פתיחת הכונן בעזרת סייר החלונות, ביצוע סינכרון בעזרת ה-WinSync וכו'), וכך, בפעם הבאה שהמשתמש יחבר את אותו ההתקן - עורך הרישום יידע להגיד למערכת איזו פעולה לבצע וכך להקל על המשתמש ולהגדיל את "חווית השימוש" במערכת ההפעלה, פעולה זאת נקראת-"Autorun". המפתח האחראי על "זכירת" תצורת הטעינה הוא:

```
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\MountPoint
s2
```

תחת מפתח זה, עורך הרישום מנהל תת-מפתחות עם שמות זיהוי ייחודיים לאותו התקן, כגון:

```
{2328b3aa-6ac0-11de-a506-005056c00008}
{5d9b903b-4541-11de-a337-002186d01bb3}
{9916dd74-653d-11de-b8be-002186d01bb3}
{ae9a98c3-3f2b-11de-ae4c-002186d01bb3}
```

כל המפתח אחראי על התקן שונה, וירוסים משנים ערכים של מפתחות אשר אחראים על התקנים דומיננטים, כגון כונן מערכת ההפעלה, או מחיצות שונות על הכוננים הקשיחים. על מנת להשתמש בפונקציה זו, יש להוסיף למפתח תת-מפתח בשם: Shellעם ה-data: "Autorun", ובו תת-מפתח בשם: Autorun או Autoplay עם ה-data: "Auto&Play", ובו עוד תת-מפתח בשם: Commandובו, בערך ה- (Default) יש להוסיף את מיקום האפליקציה אותה יש לטעון בעת כניסה לאותו הכונן. התוצר הסופי אמור להראות באופן הבא:

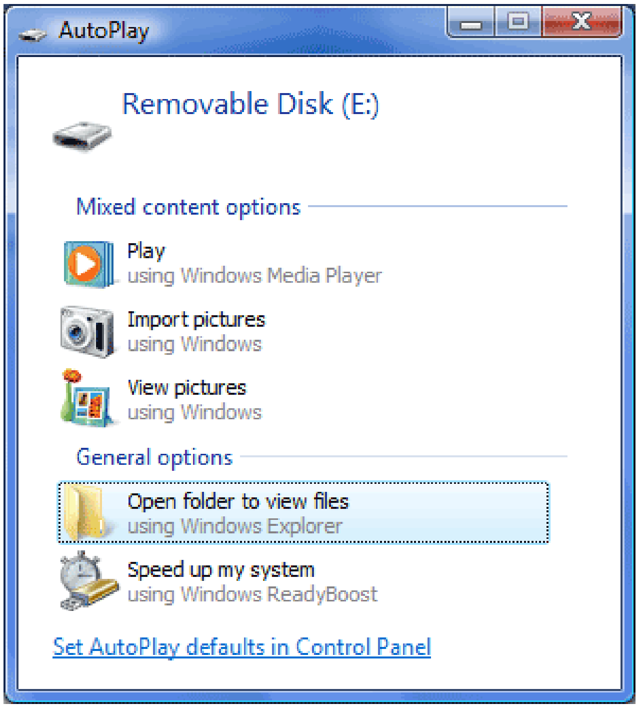


כך, בכל כניסה של משתמש לכונן C ירוץ הוירוס. חשוב לציין כי כניסה משמעה לחיצה פעמים על הכונן "C" במחשב שלי, כל כניסה בדרך אחרת לא תגרום להרצה של הוירוס, לדוגמא ע"י לחיצה כפתור ימני ו-"explore" או כניסה ל-Start משם ל-Run ושם ל-"\:C" לא תגרום להרצה של הוירוס.

אין דרך לבטל את הפיצ'ר הזה באופן גורף, ולכן וירוסים ותולעים נוהגים להשתמש בטכנולוגיה זאת באופן תדיר, מה שאפשר לעשות זה ליצור קובץ אצווה שמוחק את כל המפתחות הקיימים ב-Mountpoint2 ולבקש מהמערכת לטעון אותו בכל פעם שהמערכת עולה, בכל אופן, בכל פעם שיש חשד שהמחשב ננגע באיזה מזיק- מומלץ ללכת למפתחות הקיימות בנקודה זאת, וכך לאתר את היישום הסורר.

AUTORUN.INF

וירוסים ותולעים מנצלים עוד טכנולוגיה שנתמכת ע"י מערכת הקבצים של חלונות, והוא עוד פיצ'ר שמתפעל את ה-Auto&Play, הרעיון הוא שברגע שמחברים התקן USB למחשב, מערכת הקבצים מחפשת בתיקיית השורש שלו קובץ בשם Autorun.inf (לרב הוא יהיה עם מאפייני R+ S+ H+, אך זה לא מחייב) ובקובץ הנ"ל נשמרת הדרך שבה מערכת הקבצים תתייחס לאותו התקן. הקובץ אחראי על תצורתו ותפקודו של תפריט ה-AutoPlay:



מבנה הקובץ בנוי באופן המזכיר קבצי ini, דוגמה לקובץ Autorun.inf:

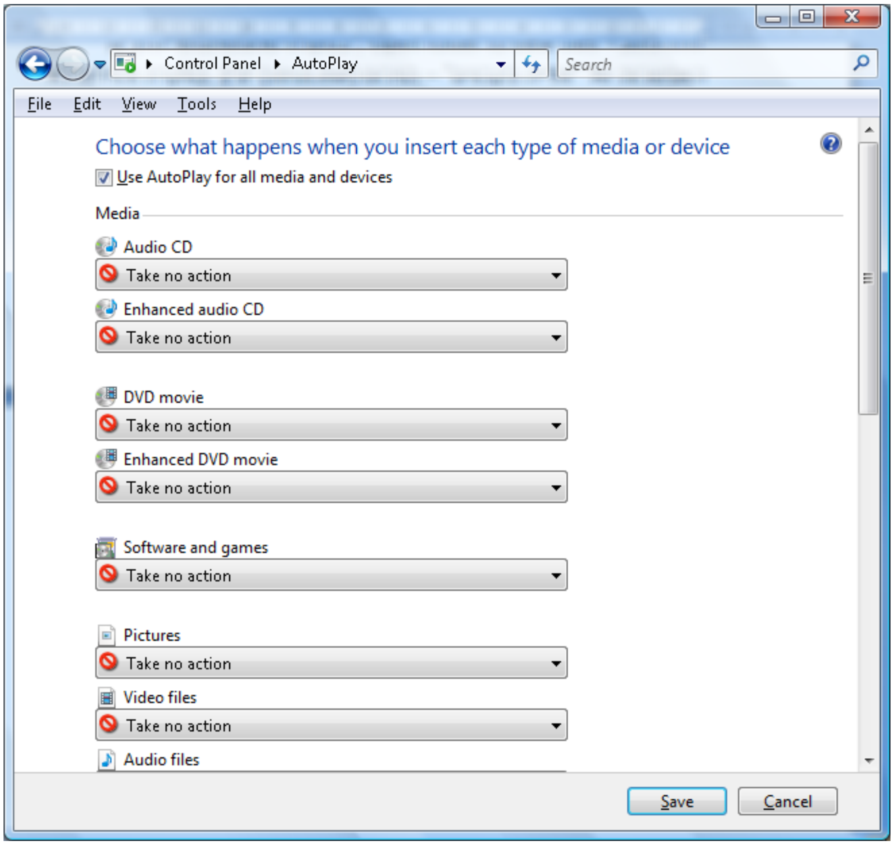
```
[autorun]
open=virus.exe
icon=folder.ico
label="Open folder to view files"
```

התוכן הבא יגיד למערכת הקבצים לטעון את תפריט ה-Autoplay, ולהכניס בו אפשרות לצפייה בקבצים, יקבע איזה אייקון יופיע, וכמובן גם מה יהיה כתוב לידו-"Open folder to view files", במקרה שהמשתמש ילחץ על האפשרות של "צפייה בקבצים" - הוירוס (virus.exe) יורץ. במקרה וכותב הוירוס לא רוצה ליצור חשד הוא יגיד לוירוס גם לפתוח את כונן ה-USB לצפייה בקבצים בכדי שהפעולה תהיה חלקה והמשתמש לא יוכל לשים לב לשינויים.

בכדי לבטל אפשרות זאת יש למנוע ממערכת ההפעלה להשתמש בתפריט ה-Autoplay, לכל התקני הקבצים (כולל CD, FLOOPY, USB, SMARTCARD וכו'). אפשר לבצע זאת ע"י:

<u>ב-Vista:</u>

כניסה ל-"Control Panel" ושם כניסה ל-"Autoplay", ושם בחירת "Take no action" לכל ההתקנים.



<u>ב-XP:</u>

כניסה ל-"My computer", ושם כפתור ימני על כונן ושם בחירה ב-"Properties", בתפריט שהופיע יש לבחור בחוצץ "AutoPlay", תחת התווים "Actions" יש לסמן את "Select an Action to perform" ואז לבחור את "Take no action". לחיצה על "Apply" ואז "OK" יקבעו את התצורה הנוכחית מעכשיו והלך.

שינויים אלה לא מספיקים! הפעולה שהצגנו אומנם תמנע ממערכת הקבצים להקפיץ לנו את תפריט ה-AutoPlay, שזה טוב ויפה, אבל היא לא תמנע ממנה לאתר ולהריץ את קובץ ה-Autorun.inf, כך שהתפריט לא יופיע - אבל הוירוס עדיין ירוץ.

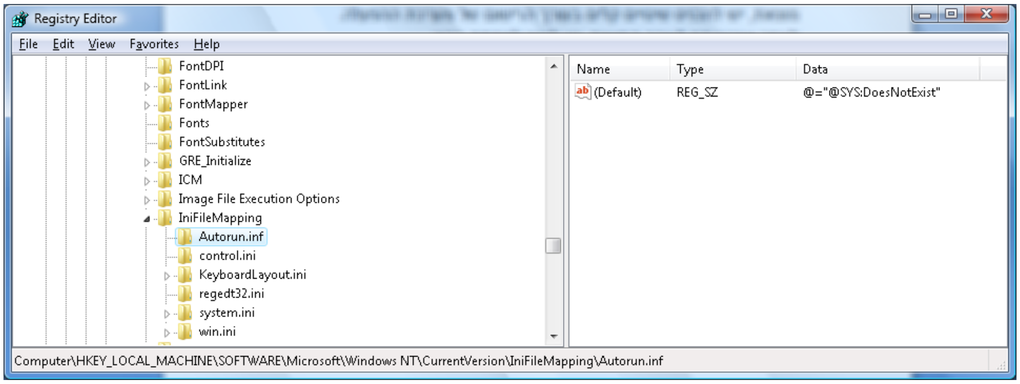
בכדי למנוע ממערכת הקבצים להריץ את כל קבצי ה-Autorun.inf שהיא מוצאת, יש להכניס שינויים קלים בעורך הרישום של מערכת ההפעלה. לאחר שנכנסתם לעורך הרישום, יש לנווט למפתח הבא:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\IniFileMapping\Autorun.inf
```

אם תת-המפתח "Autorun.inf" לא קיים- יש ליצור אותו. לאחר מכן יש להכניס אליו את הערך הבא:

```
@="@SYS:DoesNotExist"
```

מהפעם הבאה שמערכת ההפעלה תעלה, מערכת הקבצים לא תנסה לאתר את קבצי ה-Autorun.inf בכל ההתקנים אשר יחוברו למחשב.



System Services

מערכת ההפעלה בנויה באופן מודולרי, ומספר רב מאפשרויותיה מבוססות על "שירותים" אשר היא מריצה. שירותים אלו ("Services") הם יישומים האחראים לנהל או לתת שירות בנוגע לרכיבים או איפיונים ספציפים, כמו למשל רכיבי רשת, רכיבי שמע, רכיבי בקרה וניהול וכו', רכיבים אלו רצים ברקע המערכת וכמעט ולא נראים לעין. וירוסים ותולעים מנצלים לפעמים אופי זה בכדי לטעון את עצמם ביחד עם שירותי המערכת, וכך לדאוג שהם יטענו בכל פעם שהמערכת עולה.

בכדי לראות אילו שירותים נטענים ביחד עם מערכת ההפעלה, מיקרוסופט הוסיפו לנו את היישום Services.msc הממוקם ב: C:\Windows\System32.

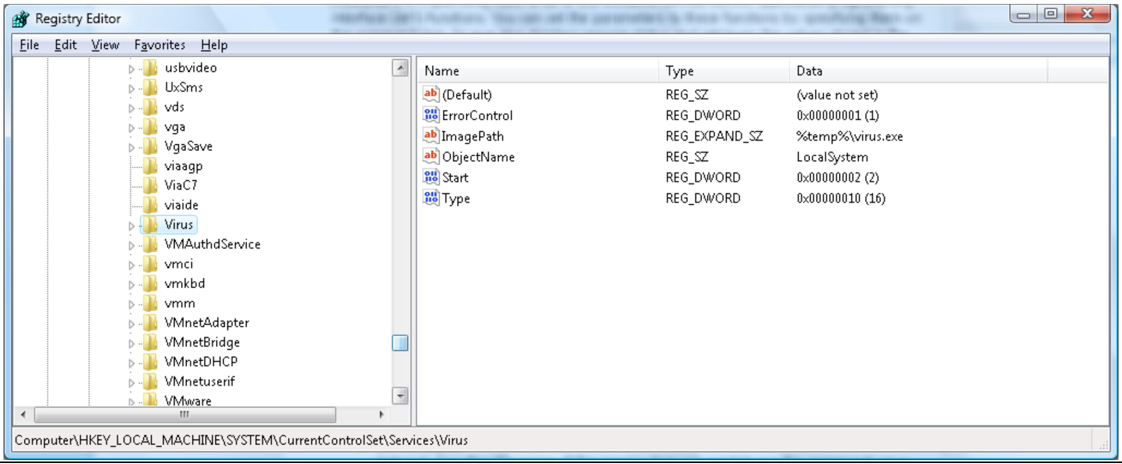
היישום לוקח את הרשימה הנ"ל מעורך הרישום, תחת המפתח:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\
```

מבנה המפתח בנוי באופן הבא:

```
ErrorControl [REG_SZ]
ImagePath    [REG_DWORD]
ObjectName   [REG_EXPAND_SZ]
Start        [REG_DWORD]
Type         [REG_DWORD]
```

- ImagePath - שומר את המיקום של היישום אותו יש להריץ.
- ObjectName - שומר את שמו של השירות אשר יופיע במנהל השירותים.
- Start - סוג הריצה (ידנית, אוטומטית, מבוטל, בעת טעינת המערכת וכו')
- Error - רמת השגיאות (רגיל, בינוני, קריטי).



יש אפשרות להוסיף את השירות באופן ידני, אך מיקרוסופט הוסיפו עוד כלי קטן בשם "SC.exe" המאפשר לבצע הוספה ומחיקה של שירות באופן זריז ויעיל. בכדי להוסיף שירות למערכת ההפעלה יש להשתמש בכלי באופן הבא (דרך שורת הפקודה):

```
sc.exe create "Virus" binPath= "%temp%\virus.exe" start= "auto"
```

מעכשיו, בכל טעינה של מערכת ההפעלה יורץ גם הקובץ %temp%\virus.exe. בכדי למחוק שירות, יש להשתמש בכלי באופן הבא:

```
sc.exe delete "Virus"
```

ושוב- יש לזכור כי לרוב וירוסים יקראו לעצמם בשמות קצת פחות מסגירים, שמות של כלי מערכת או בקרים/שירותים אמיתיים עם שינויים קלים.

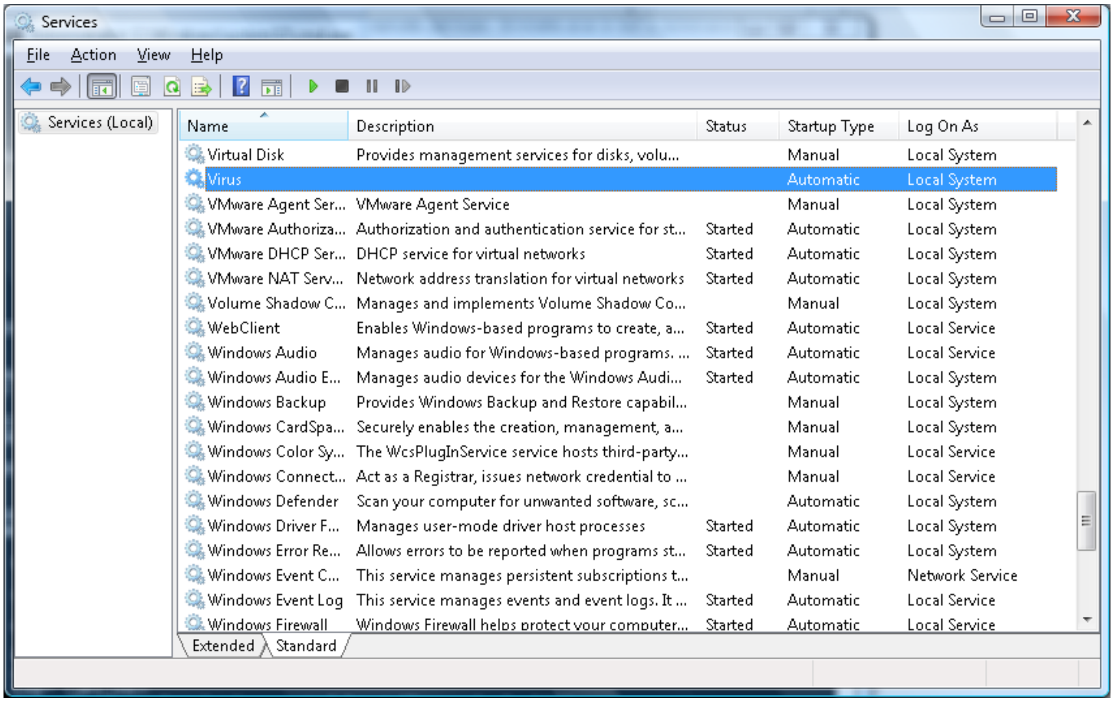


Image File Execution Options

ישנו עוד ערך אשר מנוצל טוב טוב ע"י כותבי התולעים, שימוש נכון בערך הנ"ל מאפשר לכותבי התולעים לבצע השתלטות מלאה על הרצת קובץ מסויים וזאת מבלי להשתמש בשיטות כגון API Hooking או Binary Code Injection אלה ע"י שינוי הערכים ב-Image File Execution Options אותם מנהל עורך הרישום.

בעורך הרישום, קיים המפתח הבא:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image
File Execution Options
```

אם נרצה למשל לבצע "השתלטות" על הקובץ Msconfig.exe, כך שכל פעם שמישהו ירצה להריץ אותו- המערכת, במקום להריץ את ה-Msconfig, תריץ את התולעת שלנו - נניח Calc.exe, נוסיף בעורך הרישום מפתח בשם "msconfig.exe", ובתוכו ניצור מחרוזת (String Type) בשם "Debugger", ערכה יהיה המיקום של התולעת שלנו:

```
%windir%\system32\calc.exe
```

מעכשיו, בכל פעם שמישהו יפעיל את ה-Msconfig (כל אופן שהוא) במקומו תרוץ התולעת שלנו-Calc.exe.

יותר מכך, אם למשל "נדביק" תוכניות שמוגדרות להריץ סוג מסויים של קובץ- למשל Notepad מוגדר להריץ קבצים בעלי סיומת ".txt" - בכל פעם שיריצו את אותם קבצים, הקבצים האלה לא ירוצו ובמקומם תרוץ התולעת שלנו.

וירוסים משתמשים בתכונה זו בכדי "להדביק" בעיקר את ה-Taskmgr, Msconfig, Notepad, Cmd ותוכנות דומות.

ישנן תולעים אשר נכתבו באופן כזה שבכל הרצה שלהן הן גם מריצות את הערך שקיים ב-1% וב-2% בדיוק בשביל מקרים כאלה - כך שאם "נדביק" את Notepad.exe, ונרצה להריץ את 1.txt, גם התולעת שלנו תרוץ וגם אותו הקובץ, וכך הדבר מוסיף "לשקיפות" שלה ומאפשר לה להיות פחות מורגשת.

סיכום

שיטות לטעינת הוירוס רק גדלות וגדלות במהלך הזמן החולף, ולכן חשוב להכיר דרכים אלו, כך במקרים ובהם לא ניתן להריץ אנטי-וירוס מעודכן - חיפוש טוב במנגנונים שהצגתי היום יוכל להוביל לאיתור הוירוס או התולעת, וכך להקל בהסרתה.

בטקסט זה הצגתי מספר דרכים וטכנולוגיות אשר קיימות במערכת ההפעלה Windows אשר מנוצלות ע"י וירוסים ותולעים בכדי לשפר את אורך חייהם וכך להגביר את יעילותם. חשוב להזכיר כי "שיטות" אלה נוצרו בכדי להקל על המשתמש ו-"לשפר את חווית השימוש" שלו במערכת ההפעלה, אך כמו שראינו- מספר רב של גורמים "מזיקים" מבצעים שימוש נוסף במרכיבים אלו. ניתן לחשוב שככל שהוירוס ישתמש ביותר מנגנוני גיבוי כך סביר להניח כי אורך חייו יגדל, אך חשוב מאוד לזכור - שככל שאותו הוירוס ישנה את סביבתו, כך יגדלו הסיכויים שהוא יתגלה.

RFID Hacking

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

היום נדבר על טכנולוגיה "חדשה" יחסית שנמצאת כרגע עוד בתחילת דרכה: RFID. ‏RFID זהו קיצור של Radio Frequency Identification, שזה אומר "זיהוי אלקטרוני על גבי רדיו". בפועל מדובר על מעין התקן קטן (כמו מדבקה קטנה) שמצמידים לאותו חפץ שאותו רוצים לזהות. הרעיון המעניין מאחורי מדבקה זו הוא עניין הרדיו - המדבקה משדרת את הזיהוי שלה באופן אלחוטי על גבי גלי הרדיו לקורא הרלוונטי.

למה כתבתי את המילה "חדשה" במרכאות? כי הטכנולוגיה קיימת כבר הרבה זמן - היא הומצאה לפני כמעט מאה שנה, אך עקב העלויות, הגודל והתיפקוד- הנושא לא היה רלוונטי לשוק הכללי, ורק לאחרונה, כמה שנים בודדות אחרי שנת 2000, החל הרעיון לצבור תאוצה, עלות הפיתוח ירדה, וכמו כל דבר- המדע הצליח למזער אותה עוד ועוד עד שלאט לאט התחילו לפתח את זה לשוק הכללי.

קצת מידע כללי

בערך לפני שנתיים שלוש, (קצת לפני תחילת 2007) משרד התקשורת אישר שימוש אזרחי של RFID בארץ (אושרו התדרים 915-917MHz) והשוק התחיל לפרוח.

כל המערכת מחולקת לשני חלקים עיקריים:

- **Transponders** - ("תגי קרבה" - המשדר) - מדובר בתג קטן אשר מסוגל לשדר את המידע המאוחסן עליו באופן אלחוטי, כל תג כזה מורכב משני חלקים עיקריים, משבב קטן המאחסן את המידע, ומאנטנה פנימית/חיצונית שתפקידה לשדר את המידע המאוחסן על השבב, לרוב התג לא יהיה מחובר לשום מקור אנרגיה, והוא ישתמש באנרגיה שהוא מקבל מהשדה האלקטרו-מגנטי שמפיץ קורא הכרטיסים- אך הדבר אינו מחייב.

  בכל תג מותקנים אחד משני סוגי צ'יפים, צ'יפים לקריאה בלבד, וצ'יפים גם לקריאה וגם לכתיבה, שטח של כ-2MB, אך לכל צ'יפ הניתן לכתיבה יש סקטור שבו מאוחסן הקוד היחודי של השבב שעליו אפשרות הכתיבה מוגבלת.

- **Interrogator** - ("קוראי קרבה" - הקולט) - מקלט, מחובר לאנטנה פנימית או חיצונית (לפעמים המקלט יהיה מחובר למספר מרובה של אנטנות בכדי לשפר את יעילותו), המקלט גם מפיץ שדה אלקטרו-מגנטי בתדר ספציפי קטן יחסית אך בסדר גודל המספיק בכדי לשמש כמקור האנרגיה לתגי הקרבה הנמצאים בסביבתו. הקורא יהיה עצמו מחובר בדרך כלל למחשב, בצורה חוטית, או בצורה אל-חוטית על גבי תשדורת Wi-Fi או Blue-Tooth וכד' אשר מספקת את השרת שמנהל את הנתונים.

קיימים מספר סוגים של תגי RFID:

- Passive TAG - תגית RFID ללא שום מקור אנרגיה, זאת אומרת שבמצב רגיל היא לא משדרת שום דבר, ורק לאחר שהוא נכנס לשדה האלקטרו-מגנטי של קורא הכרטיסים הוא נטען על-ידו ומשדר את המידע.
- Active TAG - תגית RFID המחוברת למקור אנרגיה, וכך היא יכולה לשדר את המידע שלה בכל זמן נתון גם במצב שבו היא ממוקמת רחוק מכל שדה מגנטי.
- "Semi Active TAG" - תגית RFID אשר משתמשת בסוללה קטנה יחסית אבל את שידור המידע היא מבצעת באמצעות אנרגיה חיצונית (כגון השדה המגנטי של קורא הכרטיסים).

כיום תגי ה-RFID משתמשים בארבעה אורכי-גל שונים, כל אורך-גל משדר למרחק שונה, ומאופיין ביכולות שונות ולכן משמש לצרכים שונים.

- **125 KHz** - מיוחס כ-Low Frequency, התדר הנמוך ביותר בשימוש כיום (גם בארץ), משדר למרחק של עד שלושים סנטימטר בקירוב, ואינו מסוגל להתמודד עם יותר מקריאה של תגית אחת בכל פעם, משמש בעיקר למערכת בקרת כניסה שונות - כמו למשל כמפתח כניסה לדלת, שעוני נוכחות, כרטיסי אשראי וכו'.

- **13.56 MHz** - מיוחס כ-High Frequency, משדר למרחקים של קצת יותר ממטר, משתמשים בו בעיקר לניהול לוגיסטי של מכולות, מעקב אחרי מוצרים בזמן פיתוח המוצר וכו' - קיים בארץ.

- **UHF -** מיוחס כ-Ultra High Frequency, משדר למרחקים של קצת פחות משבעה מטרים. משתמשים בו בעיקר לניהול מחסנים שלמים. השימוש בו אושר בארץ אך עם הגבלות מרובות ולכן כמעט ולא תמצאו אותו בשימוש.

- **2.45 GHz - מיוחס כ-**Microwave**, הוא התדר הגבוה ביותר שנמצא בשימוש כיום והוא משדר למרחקים עצומים, השימוש העיקרי שנעשה בו הוא למעקב אחרי מכוניות. בעיקר נגד גנבות ומעקב אחרי פס ייצור. לא קיים בארץ.**

- **Ultrasound** - כמעט ולא נמצא בשימוש, אך עדיין כדאי להזכיר אותו - משדר למרחק של עד 5 מטר, משתמש לזיהוי מרחבי, נקרא גם RTLS, קיצור של "Real Time Location System". לא קיים בארץ.

המתקפה

הרעיון הכללי

איך כל מתודת השימוש ב-RFID עובדת:

- שלב ראשון - תג RFID מוצמד לאובייקט כלשהו. התג מכיל קוד ייחודי לו אשר מאפשר לקורא התגים לזהותו.
- אם מדובר ב-Active-Tag, התג משדר את הקוד ללא הפסקה. אם מדובר בPassive/Semi-Tag, התג אינו משדר את הקוד אך בהגיעו לשדה אלקטרו-מגנטי (שלרב מופץ על-ידי קורא התגים) משתמש התג באנרגיה שבשדה ומשדר את הקוד הייחודי לו בתדר קבוע מראש.
- קורא התגים מוגדר להאזין באותו התדר עליו משדר תג ה-RFID וקולט את המידע ששידר אותו התג.
- קורא הכרטיסים מעביר את הנתונים שקיבל מתג ה-RFID אל מערכת הבקרה הכללית- מחשב אשר תפקידו לנהל/להשתמש במידע (אם מדובר במערכת בנק או מעקב כניסה/יציאה בעבודה, מוצרים בעגלת מכולת וכו').

חולשות מתודת השימוש ב-RFID

קיימות מספר חולשות בסכמה זו, נגע בשתיים מהן:

- **Unencrypted Storage** - כיום כמעט ולא נמצא בשימוש שום תקן אשר תפקידו לקבוע הצפנה או שימוש בהצפנה כל-שהיא בעת אחסון הנתונים על גבי תג ה-RFID עקב עלויות השימוש בתג זה. הנתונים המאוכסנים על גבי זכרון הצ'יפ נשמרים כמעט תמיד כמו שהם (Clear Text) ולא בשום צורת הצפנה/עירבול/גיבוב, לא בעת השידור לקורא התגים, ולא בעת העברת המידע למערכת המיחשוב האחראית לניהול המידע. **התקנים אשר כן נמצאים בשימוש חשופים לחולשות רבות וניתן לפרוץ אותם באופן פשוט עד כדי מגוחך** (מדובר בשני מנגנוני הצפנה, הראשון הוא DST אשר פותח בידי חברת Texas Instruments, אשר תומך במפתחות עד 40-bit, המנגנון נפרץ ע"י JHU-RSA כשנה לאחר תחילת השימושו בו בשנת 2005. השני הוא NXP אשר נפרץ ב-2008 ע"י קבוצת האקרים הגרמנית הידועה "The Chaos Computer Club".
- **None Authorization System** - תג ה-RFID מוגדר לשדר את המידע שהוא מאחסן כל עוד הוא מחובר למקור אנרגיה, אם פנימי ואם חיצוני ללא שום מערכת אשר קובעת כי אכן מדובר בקורא תגים מהימן. בנוסף, למשדר ה-RFID אין שום בקרה כי גם אם אכן מדובר בקורא מהימן אין שום יישות אשר מאזינה לאותו שדר.

בעזרת שילוב של שתי החולשות שראינו, נוכל לבצע התקפה אשר בעזרתה נוכל לגנוב את המידע הקיים על הכרטיס.

קורבנות פוטנציאלים

בארץ הנושא עוד לא הגיע לשיאו, אך בחו"ל (ארה"ב/אירופה) נושא פריצת ה-RFID מפותח מאוד ונמצא בשימוש נרחב. בין הקורבנות פוטנציאלים אפשר למצוא:

- כרטיסי אשראי (בינהם Visa Card, Master Card, American Express ועוד).
- כרטיסי גישה למקומות מוגבלים (נקראים גם "כרטיסי גישה חכמים").
- כרטיסי חניה (גם בארץ ניתן למצוא כאלה).
- כרטיסי תחבורה ציבורית המופעלים בעזרת RFID.
- מפתחות רכבים המופעלים בעזרת מפתחות RFID.

- כל מערכת המבוססת על RFID בלבד כמנגנון זיהוי.

מימוש המתקפה

כדי לממש את המתקפה יש צורך במספר כלים. הכלים אינם כלים ביתיים, אך הם עדיין זולים (באופן מפחיד) וחוקיים לחלוטין.

- RFID Reader/Writer - כגון: Point-RX, S300, XR-400, J168. (עולים כ-95 דולר ב-EBay).

- RFID Cable - כגון: SM8836, SM8837, SM8838 (יש מ-PVC ויש מ-Nylon-66, שניהם מספקים עבודה טובה) עולה כ-25 דולר ב-EBay.

יציאת SM8836 - כניסת USB למחשב.

- Driver מתאים - הנפוץ ביותר (ל-Win32) הוא ISC.MR101, אך התקנים שונים צורכים דרייברים שונים, ולכן אם מזמינים חובה לבדוק האם מגיע דרייבר מתאים.

- כלי לפיצוח ההצפנה (במקרה וקיימת), מדובר ב-Key-Search מבוסס Brute-Force פשוט.

בכדי לפצח את מפתח ההצפנה (40bit) לוקח (במקרה הגרוע ביותר) קצת יותר משבועיים, ולכן במקרים כאלה יש אפשרות ליעל את המנגנון (עד לתזמון של 10 שעות!) ע"י שילוב התוכנה על גבי לוח עיבוד הניתן לתיכנות (FPGA - ראשי תיבות של Field Programmable Gate Array). הנפוץ ביותר לשימוש כיום הוא "Cyclone II", והוא מגיע עם ערכת פיתוח מוכנה מראש.

מהלך המתקפה:

שלב ראשון - השגת המידע:

החלק הקשה במהלך הפריצה הוא להגיע לקרבה פיזית לתג, אך בגלל שמדובר בגלי רדיו אין בעיה לזהות את הזליגה גם מעבר לבדים או ארנקי עור לדוגמא (במצב שה-RFID הוצמד לכרטיס חכם אשר נמצא בארנק בתוך כיס אחורי או תיק של מישהו). מפני שהתג המשדר את המידע על גבי הצ'יפ משדר אותו ללא הבחנה, אין בעיה להשתמש בכל קורא ואין חובה להשתמש דווקא בקורא הייעודי לאותו כרטיס.

שלב שני - פיענוח המידע:

לאחר שהשגנו את המידע הנמצא על הכרטיס המצב הוא- או שהמידע מפוענח, כך שאין בעיה, או שהמידע הוצפן בעזרת אחד מהאלגוריתמים, בכדי לפענח את המידע יש צורך בהרצת ה-Brute-Force, בעזרת הכלים הנכונים, גם במצב הגרוע ביותר ייקח לא יותר מעשר שעות.

כמו שכבר ציינו, אחד האלגוריתמים השמישים ביותר להצפנת המידע הנמצא בכרטיסים אלו נקרא "Crypto-1", והוא הומצא ע"י חברה בשם NXP Semiconductors. החבר'ה שהצליחו לפרוץ את האלגוריתם, יצרו ספריה ב-C, המשמשת לבצע מספר התקפות על המידע המוצפן בעזרת האלגוריתם, שמה הוא Crapto1 (בדיחה על השם המקורי של האלגוריתם).

הספריה מכילה מספר פונקציות שמנצלות חולשות במנגנון האימות של ה-Crypto-1.
את הספריה אפשר להוריד מכאן:

http://crapto1.googlecode.com/files/crapto1-v2.4.tar.gz

מאמר המנתח לעומק את החולשות הקיימות באלגוריתם של NXP, ודרך ניצולם כמתקפה אפשר למצוא פה:

http://www.sos.cs.ru.nl/applications/rfid/2008-esorics.pdf

שלב שלישי - הנפקת הכרטיס

- אם מדובר בכרטיסי אשראי, המתקפה נגמרת פה - כל המידע על הכרטיס נמצא אצלנו, מספר הכרטיס ופרטיו האישיים של בעליו (שם/תעודת זהות וכו').
- אם מדובר בכרטיס חכם המאפשר גישה למקום מסויים - או כל כרטיס הצורך שימוש פיזי במידע שיש עליו, יש צורך גם בהנפקת כרטיס חדש עם הפרטים שהשגנו, ולכן אנחנו צריכים להשיג גם RFID Writer. כיום קיימים לא מעט התקנים אשר קוראים מידע וגם מסוגלים לכתוב מידע חדש על הכרטיס. הפעולה פשוטה, ובעזרת התוכנה המגיעה עם הכותב אין שום בעיה לבצעה.

בכדי לבצע פעולה זאת ישנה אפשרות להשתמש בכלים כגון "RFDump" שמסוגל להציג באופן מסודר את כל המידע "יבש" (Meta Information) כגון Tag ID, Tag Type, יצרן וכו', בנוסף הכלי מסוגל גם להציג ולערוך את המידע הקיים על הכרטיס (במידה וההתקן מאפשר זאת). התוכנה הנ"ל מאפשרת לייצא/לייבא את המידע מפורמט XML.

מפני שישנם הרבה סוגי כרטיסים וישנם מספר דרכי יישום לשמירת המידע, התוכנות משתנות בין חומרה לחומרה, התוכנה RFDump, למשל, מסוגלת לבצע את הפעולות רק מהתקנים שתואמים לרכיבי "ACG Multi-Tag" (תגים התומכים בתדרים 125 kHz ו- 134.2 kHz).

אפשר להשיג אותה בכתובת:

http://www.rfdump.org/dl/rfdump-1.3.tar.gz

במצבים בהם המידע המאוחסן על הכרטיס מוצפן יש להצפין את המידע לפני האחסון (בכדי לא ליצור אי-תאימות קריאה בזמן פיענוח המידע ע"י קורא הכרטיסים), אך זה לא מהווה בעיה, אלגוריתם ההצפנה מוכר לנו ואת מפתח ההצפנה השגנו כבר בשלב השני.

שלב רביעי - שימוש

מפני שרב המערכות משתמשות בתג ה-RFID כאמצעי היחיד לזיהוי המשתמש, אין בעיה לבצע את השימוש, למשל- כאשר מעבירים כרטיס עובד בדלת-חכמה אין צורך בהצגת תעודה מזהה, התעודה המזהה היא התעודה בעלת תג ה-RFID.

התגוננות

מה אפשר לעשות בכדי להתגונן? נכון לכתיבת שורות אלה, ההמלצה הטובה ביותר היא להמנע משימוש ב-RFID. מספר פתרונות אחרים:

- שימוש בתגי RFID מבוססי מפתחות הצפנה של 128-bit ויותר. (כגון BUSlink).
- שימוש בתגי RFID מבוססי מנגנון הזדהות חכם אשר ממוקם לפני שידור המידע כגון התגים של CryptoRF.
- שימוש בתגי RFID אשר מממשים את עקרון ה-RSA וכך בעצם מונעים מגורמים זרים להגיע למידע המוצפן עליהם.
- תגים התומכים בהעברת מידע על-גבי SSL או TLS.
- שימוש בתגי RFID מבוססי Token.

כאן תוכלו לקרוא מאמר מעניין מאוד של VeriSign על עקרונות אבטחת ה-RFID:

http://www.verisign.com/static/028573.pdf

סיכום + קישורים

נושא האבטחה בענף ה-RFID לא נמצא במודעות החברות או האירגונים הגדולים - לא בעולם ובמיוחד לא בארץ. המודעות לאבטחה בנושא נמצאת בעליה בשנה-שנתיים האחרונות, אך עדיין יש הרבה מאוד מה לשפר. אני מקווה שמאמר זה העשיר את הידע שלכם על הנושא ועל סכנות השימוש ב-RFID.

<u>מספר קישורים רלוונטיים:</u>

- ההסבר של JHU-RSA על פריצת ה-DST: http://www.rfidjournal.com/article/print/1415
- קישור (וידאו) להרצאה של CCC על פיצוח ה-XNP:
  http://www.videogold.de/iw/chaos-communication-camp-2007-24c3-mifare-security/
- מידע על ה-FPGA:
  http://www.altera.com/products/devkits/altera/kit-cyc2-2C20N.html
- קישור (וידאו) להסבר על מימוש הפריצה ב-Boing-Boing TV:
  http://tv.boingboing.net/2008/03/19/how-to-hack-an-rfide.html
- מקור מידע מעניין עם כלים, תוכנות ומאמרים על הנושא: http://rfidiot.org/

Port Knocking

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

השלב הבא בפריצה לאחר שהתוקף הגדיר לו מטרה הוא להשיג עליה כמה שיותר נתונים. אחד החלקים העיקריים בשלב הזה הוא למפות את המטרה, מבחינה תשתיתית. מיפוי המטרה יכול לכלול מספר נסיונות:

- נסיונות לגלות אילו **פורטים** פתוחים על השרת.
- נסיונות לגלות אילו **פרוטוקולים** משתמשים בפורטים האלו.
- נסיונות לגלות לאילו **שירותים** משמשים אותם פרוטוקולים.
- נסיונות לגלות את **גירסאותיהם** של השירותים.
- נסיונות לגלות **חולשות** באותן גירסאות.

ברור שקיימים עוד מהלכים שבהם תוקף יכול להשתמש בכדי למפות מבחינה תשתיתית את מטרתו, נתמקד במסמך זה באלה.

למה אנחנו צריכים לדעת את זה?

נניח תוקף קבע מטרה, שרת מרכזי של איזה אירגון, הוא סורק את השרת בעזרת NMAP או משהו בסיגנון, מקבל רשימת פורטים פתוחים, מגלה שהפורט 22 פתוח על השרת, מה שאומר שאם מנהל הרשת לא היה חכם מדי, כניראה מדובר בשירות SSH, התוקף יודע היטב שבכדי להתחבר לשרת ה-SSH הוא צריך שם משתמש וסיסמא, אבל מה, אחרי נסיון ההתחברות, הוא פתאום מגלה מהבאנר, שמדובר בשרת OpenSSH v3.3.

זהו, Game-Over, כולנו מכירים את ה-Buffer Overflow שקיים בגירסאות 2.9.9-3.3 של OpenSSH. המרחק מכאן עד להשתלטות מלאה של התוקף על השרת- קצר מאוד.

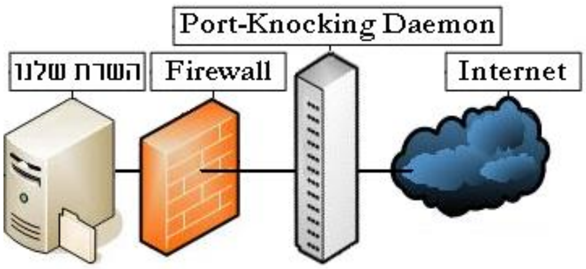
איך בכל זאת אפשר להתגונן מפני מקרים כאלה? חוץ מלעדכן את כל הגירסאות ולהיות רשום במליון ואחת רשימות-מיילים לעדכוני אבטחה- יש אפשרות פשוט להשתמש ב-Port -Knocking.

הרעיון הכללי

הרעיון הוא כזה- תשאיר כמה פורטים פתוחים על השרת שאתה רוצה, תשתמש באיזה גירסאות שאתה רוצה - שום תוקף לא יוכל לדעת איזה פורטים פתוחים על השרת שלך, שלא נדבר על לגשת אליהם.

איך בדיוק זה עובד? כל מי שירצה להתחבר לפורט 22 על השרת שלנו- בכדי לתקשר עם שירות ה-SSH, יהיה חייב קודם לכן להתחבר למספר פורטים קבועים מראש בסדר מסויים ורק אז פורט 22 יהיה פתוח לסייבר-ספייס ויהיה אפשר להתחבר אליו ולתקשר איתו.

היישום הוא כזה- לפני שכבת ה-Firewall שעל השרת, מתקינים Port Knocking Daemon שמנתר את כל התקשורת המגיעה לשרת, באופן הבא:



ציירתי את ה-Port-Knocking Daemon כחומרה נפרדת, אך כמו ה-Firewall, יש אפשרות שהוא יהיה אפליקטיבי ולאו דווקא כחומרה חיצונית.

אז כמו שאמרנו, כל מידע שמגיע למחשב שלנו עובר דרך שירות ה-Port Knocking [מעכשיו: PK] - נניח והפורט 22 אכן פתוח על השרת שלנו, וב-PK הגדרנו שה-Sequence לפורט 22 הוא:

50222<-102<-3302<-56432

ב-knockd.conf, זה נראה ככה פחות או יותר:

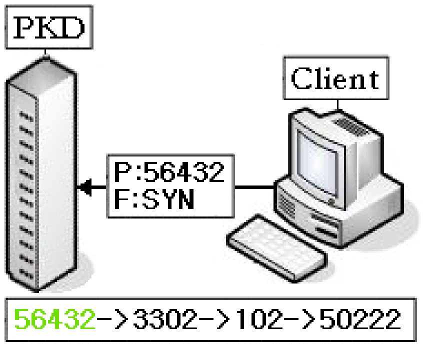
```
[SSH]
sequence    = 56432,3302,102,50222
seq_timeout = <timed out>
command     = <command to Bind the SSH to this connection>
ACCEPT
tcpflags    = syn
```

אם נשלח SYN ל-22 בכדי לנסות להתחיל את ה-Handshake של ה-TCP/IP אנחנו לא נקבל שום תשובה, למה? כי שירות ה-PK לא יעביר את המידע ששלחנו ל-22, פורט 22 על השרת שלנו אף פעם לא קיבל שום מידע!

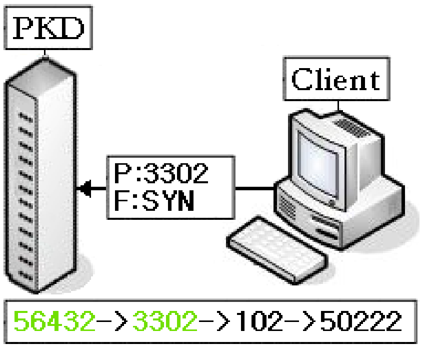
אופן ההתחברות

בכדי כן להצליח להתחבר אליו, אנחנו נאלץ לשלוח SYN לכל פורט שהוגדר ב-Sequence של ה-PK לפי סדר ה-Sequence, בצורה הבאה:

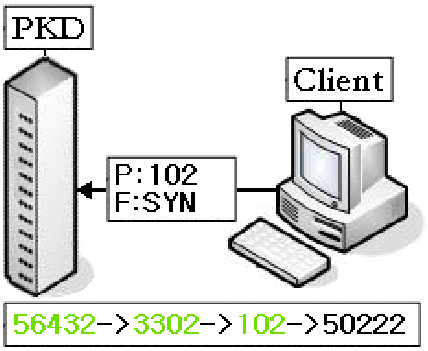
- שלב ראשון- שליחת SYN לפורט 56432:



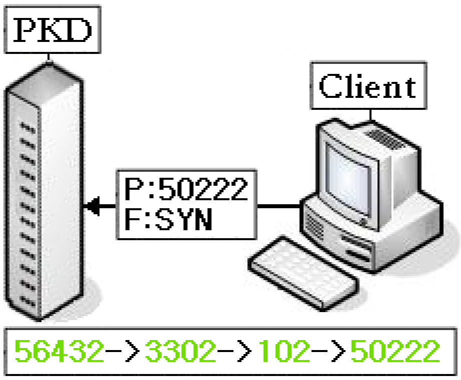
- שלב שני- שליחת SYN לפורט 3302:



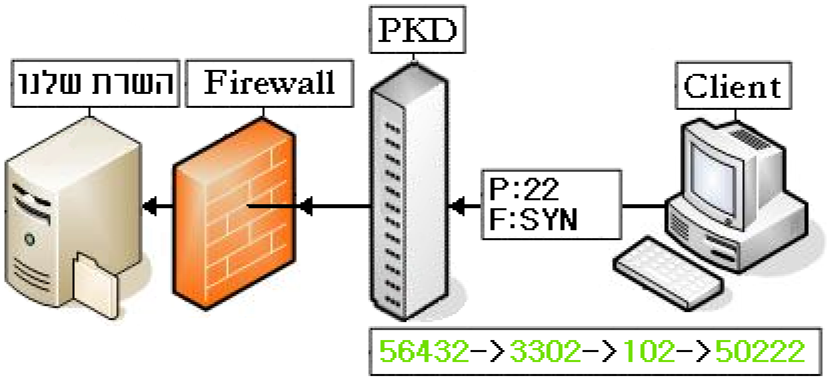
- שלב שלישי- שליחת SYN לפורט 102:



- שלב רביעי- שליחת SYN לפורט 55022:



- שלב חמישי- ה-PKD מאפשר את פתיחת הפורט ב-Firewall וככה ה-Client יכול לתקשר עם שירות ה-SSH:



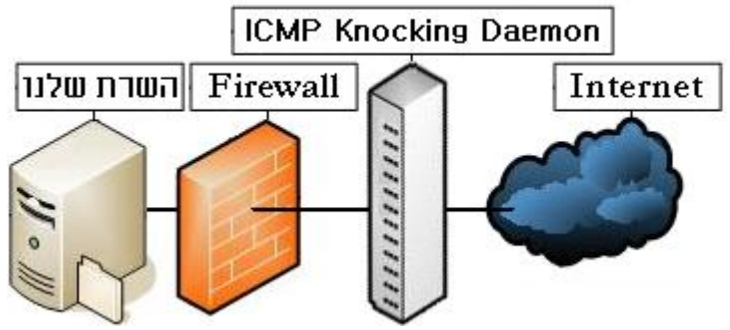
ורק כך בעצם מתחיל כל ה-TCP/IP Handshake לרוץ.

כמו שראינו- בלי לדעת את ה-Sequence הנכון, אין לנו סיכוי לדעת בכלל שפורט 22 בכלל פתוח!

עוד טכניקות למימוש ה-Knocking

- **:ICMP Knocking**

ICMP Knocking נקראת כך מפני שהיא מממשת את עקרון ה-Knocking כל הפורטים סגורים עד שמתבצעת הקשה כלשהי - רק בהתבסס על בקשות "פינג" ולא שליחת דגלים לפורטים ב-Sequence ידוע מראש. בהתחלה, המצב בדיוק אותו דבר כמו ב-PK:



כל הפורטים סגורים ושרת ה-ICMP מאזין לכל התקשורת הנשלחת למחשב. לקוח ה-ICMP Knocking שולח מספר "בקשות פינג" (ICMP) לפני כל בקשת חיבור לפורט מסויים בשרת ICMP, כל בקשת פינג שנשלחה בעלת Payload-length שונה. הבקשות נבדקות על ידי שירות ה-ICMP Knocking ואם ה-Payload-length מתאים לגדלים שנקבעו מראש והפורט שנתבקש לפתוח אכן מתאימים ב-Sequenc שירות ה-ICMP Knocking מאשר ל-Firewall לפתוח את הפורט המבוקש, אם לא-הפורט נשאר סגור. הרעיון הוא שמגדירים את ה-Payload-length Sequence כמו שמגדירים את ה-Port Sequence.

- **:Single Packet Authorization**

מתבצע באופן דומה מאוד ל-Port Knocking רק מתבצע ע"י שליחת חבילת מידע (Packet) אחת שנקראת "SPA Packet" שנכון לעכשיו נתמכת רק ע"י FWKnop.

תצורת ה-Packet נראת כך: [נלקח מהקונפיג של -FWKnop]:

```
16 bytes of random data
local username
local timestamp
fwknop version
mode (access or command)
desired access (or command string)
MD5 sum
```

- ה-16bytes הראשונים מכילים תווים רנדומאלים, המקושרים ל-Local TimeStamp, שניהם ביחד נועדו בכדי למנוע Replay Attack.
- ה-Local username קיים בכדי לאפשר מידור הרשאות.
- ה-Mode אומר לשרת האם ה-Packet מבקש להריץ פקודה או לקבל גישה למשאב מסויים.
- ה-Desired Access זאת בעצם הפקודה או הבקשת גישה עצמה.
- ה-MD5sum הוא כמובן ה-sum של כל ה-Packet עצמו ונועד להבטיח את שלמות ה-Packet.

כל המידע הזה נכתב בשורה אחת והערכים מובדלים בעזרת ":" (נקודותיים), את השורה מקודדים ב-Base64 - ואת המחרוזת שקיבלנו מצפינים ב-Rijndael בעזרת מפתח שידוע מראש גם ללקוח וגם לשרת.

לאחר שה-Packet נשלח השרת מפענח את המידע בעזרת המפתח, בודק את ה-MD5sum, בודק אם ה-TimeStamp וה-Ramdon-16bytes לא קיימים כבר ב-Cache שלו, ואם הכל תקין- הוא בודק את הרשאות המשתמש, ואם זה תקין- הוא מבצע את הפעולה ושומר את ה-TimeStamp וה-16bytes-Random ב-Cache למקרה שישלח אותו Packet ע"י תוקף שיבקש לבצע Replay Attack.

כמובן שאם המפתח של ה-Rijndael מתגלה לתוקף הוא יוכל להשתלט על השרת בקלות רבה.

גמישות

ישנן מספר דרכים בכדי לשלוח לאנשים מורשי גישה את ה-Sequence של הפורט הספציפי אליו הם מעוניינים להתחבר, אני אציין אחת מהן:

- **שלב ראשון:** לוקחים את ה-(56432,3302,102,50222) Sequence וממירים כל מספר לבינארית ומוסיפים אפסים לפני התוצאה כך שיהיה לנו 16 ביטים בכל ייצוג, נניח שהפורט הראשון שלנו הוא: 56432. בבינארית זה יוצא: 1101110001110000.יש לנו כאן 16 ביטים ולכן לא צריך להוסיף שום 0. הפורט הבא שלנו הוא: 3302. בבינארית זה יוצא: 110011100110. יש לנו כאן 12 ביטים, ולכן אנחנו צריכים להוסיף עוד **4 אפסים**: **0000**110011100110.

- **שלב שני:** מחברים את הבינארית של כל הפורטים שלנו, ונקבל את המספר הבא: 110111000111000000001100111001100000000001100110110001000010111‏0. זה המפתח שלנו לפורט 22, מבצעים למפתח הזה XOR בעזרת מחרוזת (שנקבעה מראש עם הלקוח) כמובן שאפשר להשתמש ב-AES או כל אלגוריתם הצפנה מבוסס מפתח שהוא.

- **שלב שלישי:** הלקוח מבצע שאילתת בקשת מפתח מוגדרת מראש לפורט 22 ומקבל את הסייפר שיצרנו מה-Sequence.

- **שלב רביעי:** הקליינט מפענח את הסייפר בעזרת המפתח של ה-XOR או של האלגוריתם שקבענו מראש ומקבל את הצירוף המקורי.

- **שלב חמישי:** הקליינט מפרק את התוצאה למחרוזות בנות 16 ביטים ומקנפג בעזרתו את פורט 22 על ה-PK קליינט שלו וכך יוכל להתחבר לשרת.

אם תוקף מבצע שאילתת בקשת מפתח הוא יקבל את המחרוזת של המפתח הכניסה שלנו באופן מוצפן, ולא יוכל לעשות בה שימוש בלי המפתח והאלגוריתם שבהם השתמשנו בכדי להצפין את ה-Sequence לפורט הספציפי.

הרעיון בשימוש במנגנון כזה הוא שאם נרצה בעתיד לשנות את ה-Sequence של אחד הפורטים שלנו, לא תהיה לנו שום בעיה, כי בפעם הבאה שהלקוח ינסה להתחבר לשרת ויראה שהוא לא מצליח להתחבר בעזרת הצירוף הישן- הוא יבין שהצירוף כניראה שונה, והוא פשוט יבצע שאילתת בקשת מפתח חדשה.

חולשות

כיום לא ידועה שום מתקפה חיצונית המאפשרת לדעת אילו פורטים פתוחים ברשת מאחורי ה-PKD, אך המנגנון הזה לא מוגן מפני גורמים פנימיים - כגון מקרים שבהם תוקף הצליח לבצע מתקפת MITM באופן מוצלח (למשל ע"י ARP Poisoning), אין לשירות ה-PKD אפשרות לדעת כי אכן מדובר בתוקף, גם אם חבילות המידע הנשלחות יהיו מוצפנות במפתח הציבורי של השרת ולתוקף לא תהיה אף אפשרות לדעת

מה תכולתן- הוא עדיין יוכל לדעת לאילו פורטים מיועדות החבילות, וכך לעלות את ה-Sequence המקורי, וגם אם מדובר ב-ICMP Knocking.

ישנן מספר דרכים להתגונן מפני התקפות כאלה. נציג אחת מהן: כל חבילת מידע שנשלחת מהקליינט ל-PKD תכלול בתוכה גם TimeStamp ומפתח סודי שנקבע מראש (אם רוצים אפשר שלכל פורט ה-Sequence יהיה מפתח שונה), את חבילת המידע מצפינים בעזרת המפתח הציבורי של אותו השרת, כאשר החבילה תעבור אצל התוקף (MITM), הוא לא יוכל לקרוא את מה שכתוב בפנים מפני שאין לו את המפתח הפרטי של השרת, הדבר היחידי שהוא יוכל לעשות- זה להמשיך להעביר את החבילה הלאה.

השרת מקבל את החבילה, מפענח אותה, ובודק את ה-TimeStamp. אם היא אותנטית - הוא בודק את המפתח הסודי שנכלל בחבילת המידע, אם כן- הוא מחכה להמשך ה-Sequence.

במקרה כזה, לא מנענו מהתוקף לקבל את ה-Sequence הסודי, אך ידיעת הנתון הזה בלבד- לא תעזור לו להתחבר לשום פורט הפתוח בשרת שלנו, מפני שהוא צריך את המפתח לכל פורט! זה נכון שהתוקף יוכל לדעת את ה-Sequence, אבל סתם לשלוח SYN ברצף הנכון כבר לא מספיק.

מה בדבר לבצע Replay Attack? הרעיון הוא כזה- התוקף לא יודע מה המפתח, אבל יש לו את חבילות המידע שנשלחו ובתוכן יש את המפתח לכל פורט- הוא יוכל לשלוח אותן וכך ליצור חיבור! אז זהו, שלא. כאן בדיוק נכנס הרעיון של ה-TimeStamp שהתפקיד שלה בדרך כלל הוא למנוע Replay Attack, החתימה נוצרת בעזרת השעון, ובשרת נקבע Time-out שלאחר זמן קצוב (וקצר מאוד) חבילת המידע הופכת ללא רלוונטית מבחינת השרת, גם אם כל המידע בפנים נכון ותקף.

סיכום + קישורים

הרעיון לא מסובך, לא להבנה ולא לביצוע, אבל אין ויכוח שאכן מדובר בפיתרון אלגנטי ויפה ביותר להסתרת הפורטים הפתוחים על השרת שלנו, מה שמוסיף ללא ספק עוד שכבה באבטחתו. בתיאוריה אפשר לשלוח SYN לפורטים שעל השרת באופן שיטתי (Brute-Force) וכך לנסות לגלות איזה פורטים אכן פתוחים, אבל גם אפשר להגביל את מספר נסיונות ההתחברות בשרת למספר קטן של פעמים וכך למנוע את זה, וגם בלי זה- קיימים 65535 פורטים שונים, התחברות שיטתית לכל הפורטים (בהתחשב ש-Sequence אחד יכול להיות גם עשרים ,חמישים פורטים ויותר) היא מלאכה אשר תגזול יותר שנים ממה שהתוקף יחיה.

חשוב לציין ולהדגיש כי ה-Port Knocking **לא בא להחליף שום מנגנון אבטחה בשום פרוטוקול**. אם משתמשים ב-Port Knocking בשום פנים ואופן אין שום סיבה להסיר את מנגנוני ההזדהות הקיימים בפרוטוקולים!

קישורים:

- האתר של Martin Krzywinski, אתר שלם העוסק בנושא:

http://www.portknocking.org

- עוד אתר מומלץ עם המון חומר בנושא, הוא Aldabaknocking:

http://www.aldabaknocking.com

- מאמר ב-Linuxjournal המסביר בין השאר איך להשתמש ב-PK בשילוב עם IPTables:

www.linuxjournal.com/article/6811

- fwknop - הסבר ומימוש על Single Packet Authorization ב-FWKnop:

http://www.cipherdyne.org/fwknop

- Knockd - עמוד הבית של אחד משרתי ה-Port Knocking המוכרים ביותר:

http://www.zeroflux.org/projects/knock

- מימוש נחמד ב-Python ל-ICMP Knock Server:

http://homework.nwsnet.de/products/0d09_icmp-knock-server

הפרוטוקול Kerberos v5

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

Kerberos הוא פרוטוקול שרת-לקוח מבוסס הצפנה אשר תפקידו למנוע דליפת מידע מהרשת, או בשפה המקצועית - "פרוטוקול אימות" (Authentication Protocol). כחלק מתפקידו הוא מוגדר גם כפרוטוקול לניהול מפתחות (Key Infrastructure). צורת מימוש תפקידיו שונה משאר הפרוטוקולים הדומים לו (כגון CHAP, PAP, RADIUS , TACACS ועוד) באופן שבו הוא פותר את בעיות האבטחה הידועות במנגנונים כאלה - בהמשך הטקסט ניגע גם בהם.

קצת רקע כללי: Kerberos פותח כחלק מפרוייקט חופשי בשם Athena. את הפרוייקט התחילה לפתח קבוצה קטנה יחסית ב-MIT בשנת 1979. הפרוייקט נועד לנהל את מידור ההרשאות במערכת הקבצים של רשת האוניברסיטה. לאחר תיקונים רבים של מספר כשלי-אבטחה בפרוטוקולי החבילה, שוחררה החבילה כ"מוצר מוגמר" רק בגירסתה החמישית. פיתוח כלל החבילה (שכללה בין השאר את Zephyr, Thin client, OLHואת-Discuss) הסתיים קצת יותר מעשר שנים לאחר מכן ב-1993 ונועדה לשימוש הן למערכות PC והן למערכות MAC. זמן קצר לאחר מכן גם שוחררה חבילת ה-"Generic Security Services" שכללה ממשקי API רבים עבור Kerberos בכדי לעזור למפתחים ליצור תוכניות שיכלו "לדבר" עם הפרוטוקול וכך להשתמש בחוסנו.

בכדי לממש מספר הצפנות במהלך ה-Hand-Shake של הפרוטוקול, החבר'ה ב-MIT השתמשו בהצפנות כגון RC ו-DES (שלאחר מכן שונו ל-AES ו-SHA-1 במערכות החלונאיות), ועקב כך ארצות הברית הגדירה אותו כ-"כלי נשק" ולכן היה אסור לייצא אותו אל מעבר ליבשה המערבית.

בנוסף, הפרוייקט מהווה בסיס גדול מאוד לפרוטוקולי ניהול הרשאות אחרים, אשר משמשים אותנו כיום כגון LDAP ו-Active Directory.

הצורך בפרוטוקול

למה אנחנו צריכים אבטחה ברשת הפנימית של האירגון?

נניח ויש לנו באירגון מספר מחלקות/מדורים. לכל מדור יש את הנתונים המיוחסים לאותו מדור. למשל, למחלקת ניהול הכספים יהיה מידע על ההכנסות וההוצאות של הארגון. בנוסף יתכן ויהיה לה את מספרי חשבונות הבנק של האירגון וכדו'. אם מדובר במחלקת משאבי אנוש - סביר להניח שהם מאכסנים במחשביהם את הנתונים האישיים על כל עובד, מצב משפחתי, תפקיד, אולי גם תיק מעקבים וכד'. באירגון מסודר, אסור שלעובד ממחלקה אחת תהיה גישה לקבצים של מחלקה אחרת - המידע אמור להיות ממודר. נכון שכל עובדי הארגון שייכים לאותו הגוף, אבל כל אחד צריך לדעת רק את מה שהוא עוסק בו ואת דרכי הממשק שלנו עם הגופים השונים.

אנחנו יודעים שאנחנו לא רוצים שלכל משתמש תהיה גישה לכלל הרשת, אבל איך עושים את זה? על ידי מידור חשבונות בעזרת ניהול ההרשאות ע"י הגבלת המשתמשים לפי קבוצות או לפי תפקידים.

נניח ויש לנו שלוש מחלקות: מחלקת כספים, משאבי אנוש ומחלקה של עובדי הניקיון, בכל מחלקה יש לנו ראש מחלקה ועשרה עובדים.

לכל עובד יש מחשב. כל עובד הוא משתמש מוגבל ברמת המחשב שלו. בנוסף, כל ראש מחלקה הוא מנהל ברמת הרשת הפנימית של המחלקה שלו, ואנחנו - ראשי האירגון הזה - מנהלים ברמת הרשת כולה. זאת אומרת שאנחנו יכולים ליצור שלוש קבוצות:

- **מנהלים ראשיים** - מנהלים ראשיים מוגדרים כמנהלים ראשיים בכלל האירגון, להם יש גישה לכל מחשב ולכל משאבי הרשת.
- **מנהלי מחלקות** - מנהלי מחלקות מוגדרים כמנהלים מוגבלים בכלל הרשת אך כמנהלים ראשיים ברשת הפנימית של המחלקה שלהם.
- **משתמש מקומי** - משתמש מקומי מוגדר כמשתמש מוגבל או כמנהל ראשי במחשב הפרטי שלו אך מוגבל הן לרשת המחלקתית והן לכלל הרשת האירגונית.

בצורה כזאת, אנחנו יכולים להבטיח שאף משתמש לא יוכל לגשת או לקרוא מידע ממחשבים שהוא לא אמור לגשת אליהם.

בדיוק בכדי לממש את המודל שהצגתי לכם עכשיו, פותחו הפרוטוקול Kerberos ודומיו. הפרוטוקול מקבל בקשת גישה למשאב מסויים ברשת האירגונית ותפקידו של Kerberos היא להבין מי בעצם נמצא מולו והאם הוא אכן רשאי לגשת לאותו משאב. אם אכן מדובר במשתמש מורשה - ניתנת הגישה. אם מדובר במשתמש לא מורשה Kerberos יזרוק אותו מכל המדרגות. או לפחות ככה הוא אמור לעשות.

נבחן מה קורה כאשר Kerberos מקבל בקשת גישה למשאב מגורם חיצוני.

מימוש

הרעיון הכללי

הרעיון הכללי שעומד מאחורי Kerberos הוא ניצול היתרונות של העבודה מול גורם שלישי כ-KDC (שזה קיצור של Key Distribution Center). במימוש של Kerberos מדובר בעצם בגורם שלישי שמחולק לשני גורמים נפרדים אחד מהשני, הראשון ישמש בעת ביצוע ה-Hand-Shake כ-AS - Authentication Server בכדי שנוכל לדעת במי מדובר, והשני ישתמש כ-TGS - Ticket Granting Server בכדי ליצור את החתימה הרלוונטית לאותו חיבור Ticket.

Kerberos מגיב בגדול כמו שרת proxy ונכנס כמתווך מקדים לפני תקשורת בין Client ל-Server ברשת, אך תפקידו כ-proxy יגמר ברגע שהחיבור אומת.

ניתוח ה-Hand-Shake

בכדי להגדיר Client מסויים כפעיל ברשת, עליו לקבוע מפתח אימות ולתאם אותה עם Kerberos מראש (סומנה כ-CK - Client Key). לאחר מכן הוא יוכל לתפקד עם שאר רכיבי הרשת. נניח וה-Client מעוניין להתחבר ל-Server מסויים ברשת (שקבע עם Kerberos מפתח אימות מראש- סומן כ-SK על אותו משקל), לפני שיווצר החיבור בינהם, על ה-Client להזדהות מול Kerberos והמהלך יתבצע באופן הבא:

1. ה-**Client** מחולל מפתח (N) זמני-חד-פעמי (נקרא גם Nonce) ושולח אותו לשרת ה-Kerberos ביחד עם Credential לחיבור עם ה-**Server**.
2. בזמן זה, KDC מחולל מפתח (K) ומגדיר אותו כ-"פעיל" לזמן קצר אשר נקבע מראש (KT). בעזרת CK (מפתח האימות של **Client**) הוא מצפין את N+K+KT+פרטי ה-Credential של ה-**Server** כיחידה אחת - P1, ולאחר מכן ה-TGS מחולל "Ticket" אשר כולל את K+KT+פרטיו של **Client**, ומצפין אותו בעזרת ה-SK (**מפתח האימות של Server**) כיחידה שניה-P2. את P1+P2 הוא שולח ל-**Client**.
3. ה-**Client** מפענח את-P1 בעזרת CK ומאמת כי:
    1. הפרטים אכן נכונים.
    2. KT - אכן בתוקף.

אם תוצאות הבדיקה יצאו חיוביות, אזי אכן מדובר בשרת Kerberos אותנטי והחבילה אכן בתוקף, ועל כן ה-**Client** מכין Authenticator, אשר מכיל את:

1. Time Stamp ע"פ שעונו המקומי.
2. מאפייני זהותו.
3. מאפייני הבקשה ל-**Server**.
4. נתוני הבקשה.

את ה-Authenticator הוא מצפין בעזרת K ושולח את הבקשה הזאת ל-**Server** ביחד עם P2. **שימו לב ש-Client שולח את P2 למרות שאין לו מושג מה יש בפנים- כי כמו שראינו, P2 הוצפנה בעזרת SK אשר ידוע רק ל-Server ול-KDC! - ההנחה היא שאם P1 אומתה כאותנטית אז אפשר להניח בלב שקט שגם P2 אותנטית.**

4. ה-**Server** מפענח את P2 שנשלח מה-**Client** ע"י ה-SK. לאחר הפענוח ה-**Server** יכול לחלץ את K. בעזרת K הוא מפענח את Authenticator ומאמת כי:
   1. ה-Time Stamp - אכן רלוונטי.
   2. נתוני הזהות.

אם הנתונים אכן תקפים, אזי אין ספק שמדובר ב-**Client** שאכן מאושר אצל ה-KDC ולכן **Server** מאפשר ל-**Client** גישה.

את השלבים הבאים לא חובה לממש, והם לא חלק מהפרוטוקול הבסיסי, אך כדאי ואף מומלץ לבצעם, השלבים 5 ו-6 דומים מאוד ל-3 ול-4 רק בכיוון ההפוך והם עוזרים ל-Client לוודא כי הוא אכן מדבר עם Server אותנטי:

5. ה-**Server** שולח ל-**Client** את ה-Time Stamp מוצפן בעזרת K.
6. ה-**Client** מפענח את החבילה שהגיע מה-**Server**, מפענח אותה בעזרת K ואם הTime-Stamp הוא אכן ה-Time Stamp המקורי (שהוא עצמו שלח ל-**Server** בשלב 3) אזי אפשר להניח שמדובר ב-**Server** אותנטי ו-**Client** מאפשר את החיבור.

שימו לב שבכל שלבי השיחה, גם אם נשלחו פרטים חסויים, כגון K, או ה-Time Stamp, הם נשלחו מוצפנים, ולכן, גם אם נניח אליס הצליחה להאזין לתשדורת של בוב, היא לא תקבל שום מידע חיוני שיעזור לה להבין על מה בעצם בוב מדבר. הרעיון מאחורי Kerberos הוא פשוט אימות המשתתפים בשיחה ע"י ערך אשר נקבע מבעוד מועד עם גורם חיצוני (שרת ה-KDC).

רק לאחר שה-Hand-Shake עבר בשלום, מתחיל ה-Triple Hand-Shake בין ה-Client וה-Server, וכך, ה-Server יוכל לדעת שאכן מדובר במשתמש אמיתי, וכך יוכל המשתמש לקבל גישה לאותו משאב רשת בצורה מאובטחת.

עוד נקודה, כשמדובר על ה-KDC ועל ה-TGS, לאו דווקא מדובר על שני שרתים שונים פיזית, וכיום מקובל להריץ את שני השירותים הללו על אותו שרת.

יתרונות וחסרונות

יתרונות:

- **גמישות** - הפרוטוקול מוגדר כפרוטוקול קריפטוגרפי מודולרי, זאת אומרת שאפשר להלביש עליו כל אלגוריתם הצפנה סימטרי (דו-כיווני) מבוסס מפתח אשר ישמש כגורם ההצפנה במהלך ה--Hand Shake. בתחילה השימוש הנפוץ ביותר היה ב-DES, אך כיום, רוב השימוש שלו הוא ב-RC4 (בדומה לרשתות האלחוט המשתמשות ב-WEP) ובכדי לאמת את שלמות ה-Packets, הוסיפו לו את SHA-1.
- **תוצר / מאמץ**- בעזרת הפרוטוקול אנחנו מקבלים תוצר (אימות המשתמשים) במעט מאוד שלבים ובמעט מאוד מידע שעובר ברשת.

חסרונות:

- **שימוש בחותמת זמן** - שימוש בחותמת זמן נועד בכדי למנוע זיוף של בקשות מקוריות ע"י התוקף בעזרת שליחתן שנית (Replay Attack), אך פתרון זה גם יוצר את אחד החסרונות הגדולים של Kerberos. בגלל שבמהלך ההזדהות, שרת ה-KDC ושרת ה-GTS משתמשים בחותמות-זמן (ה--Time Stamp), חובה על ה-User וה-Client להיות מסונכרני שעון עם שרת ה-Kerberos. אך בכדי לסנכרן את השעונים אי אפשר להשתמש בפרוטוקול עצמו- כי בכדי להשתמש בו על המשתתפים להיות מסונכרני שעון (וחוזר חלילה...)
- **אחסון ה-CK** - אם לתוקף יש גישה פיזית לאחד המחשבים המדברים עם הפרוטוקול- הוא יוכל לגלות בקלות את ה-CK, כי עליו להיות מאוחסן באותו אופן ששרת ה-KDC מבין אותו, מה שאומר שאי אפשר להצפין את ה-CK בעת איכסונו על המחשב המקומי.
- **שימוש בגורם חיצוני** - כמו שראינו, שימוש בפרוטוקול מחייב את המשתמשים לדבר תחילה דרך שרתי ה-Kerberos, ולכן אם קורה מצב והשרתים קרסו- תעבורת הרשת תחסם ומשאבי הרשת לא יוכלו לדבר אחד עם השני.

סיכום

זהו סופו של המאמר, אני מקווה שלמדתם ממנו. תמיד טוב לדעת מול מה אנחנו עומדים ועובדים, ותמיד טוב להכיר את הטכנולוגיות שנמצאות מאחורי הקלעים, מי שרוצה ללמוד עוד על הנושא, אפשר למצוא המון מידע בקישורים הבאים:

- הפרוייקט Athena באתר של MIT - http://web.mit.edu/acs/athena.html
- Kerberos באתר של MIT - http://web.mit.edu/kerberos
- איך להקים שרת Kerberos בבית - http://www.ornl.gov/~jar/HowToKerb.html

DNS Cache Poisoning

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

זהו המאמר השני בסדרת המאמרים שהחלה מנושא ה-HTTP Attacks. המאמר הראשון בסידרה דיבר על מתקפת Response-Splitting, וכמו שנכתב במאמר הראשון, בסידרה זו נסקור מספר התקפות אשר מנצלות את אופי ואופן פעולותיהן של מספר פרוטוקולים. במאמר הזה נתמקד במתקפה המתבצעת על שירות ה-DNS. למתקפה קוראים "DNS Cache Poisoning", ובאמצעותה מתבצעות רוב מתקפות ה- Pharming למינהן, שאנו שומעים עליהן לאחרונה.

הרעיון ב-Pharming או ב-Phishing (מדובר בשתי דרכי פעולה שונות) הוא לגרום למשתמש לחשוב שהוא גולש או מתקשר עם גורם אמין ורלוונטי מסויים, אך במציאות, הגולש נמצא באתר אחר- לרוב גורם זדוני.

כמו שאתם יודעים, האינטרנט מורכב מכתובות IP, כתובת IP יכולה להכיל רק ספרות מ-0 עד ל-255, מופרדות בנקודות (למרות שכיום חלק נכבד מהשירותים תומכים גם ב-IPv6 שמצוין כערכים הקסדצימאלים ויכולים להכיל גם את התווים A-F, אבל הרעיון הוא אותו רעיון). כדי להתחבר לרשת האינטרנט חובה על האובייקט (כל גוף המבקש להתחבר לאינטרנט), לקבל כתובת IP שתוקצה אך ורק לו, ובאמצעותה יהיה ניתן לזהותו (במקרה של נתבים ומתגים הם מקבלים כתובת אינטרנט חיצונית ומאפשרים למספר עמדות קצה להשתמש בה ע"י תיוגן בכתובות IP פנימיות לנתב). למה? בכדי ששאר "משתתפי" האינטרנט יוכלו לדעת לאיפה לשלוח את המידע שאמור להגיע לאותו אובייקט, וכך תוכל להתנהל תקשורת תקינה. לפני שהחל השימוש בשרתי ה-DNS, היתה חובה לדעת את כתובת ה-IP שלהם, ע"מ ליצור איתם קשר. אם היינו רוצים להתחבר לשרת מסויים דרך ה-BBS שלנו, היינו חייבים להכניס את כתובת ה-IP של השרת אליו רצינו להתחבר, זאת אומרת, שהיינו צריכים לזכור או לרשום הרבה מאוד מספרים, שלא אומרים יותר מדי, וזה דבר קשה יחסית ומעצבן. לכן, בשנת 1983, פותח פרוטוקול ה-DNS (קיצור של Domain Name Server) ע"י שני סטודנטים.

הרעיון שלהם היה ליצור מאגר מידע אשר יכיל רשימה ובה שמות מתחמים (באותיות **ולא** במספרים), ולצידם - כתובות ה-IP של המחשבים המארחים. כך, אם היינו רוצים לגשת לשרת שכתובת ה-IP שלו היא 212.123.33.65, לדוגמא, במקום לזכור את כל הספרות האלו, היינו פשוט צריכים לזכור את שמות האתרים (כמו שאנו זוכרים היום כתובות). הרעיון נחל הצלחה ענקית.

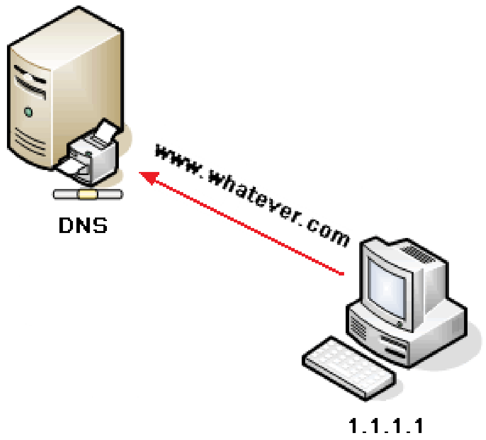
איך זה עובד?

אם אני רוצה להגיע לאתר בשם: www.whatever.com, אז אני כותב בדפדפן שלי, בשורת הכתובת, את כתובת האתר. המחשב לא יודע להתחבר לכתובת כזאת, כי לא מדובר פה בכתובת IP נומרית שהוא מסוגל להבין, אז הוא ניגש לשרת ה-DNS, ומתשאל אותו לגבי הכתובת הזאת. שרת ה-DNS בודק ברשימה שלו לאיזה IP שייכת הכתובת שהוא נשאל לגביה, ומחזיר את כתובת ה-IP למחשב. אז הדפדפן שלי ניגש לכתובת ה-IP ושולח בקשת HTTP בכדי להשיג את המידע.

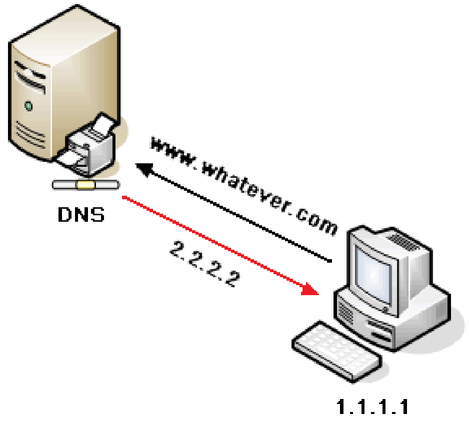
לפני ההסבר - חשוב לציין שלושה עובדות אשר משפיעות מאוד על נושא האבטחה בפרוטוקול:

- בכדי שהכל ירוץ במהירות, פרוטוקול ה-DNS רץ על-גבי UDP כברירת מחדל, דבר שמצד אחד מזכה אותו במהירות, אך גם גורם לצרות כשמדובר באבטחה ואמינות.
- בכדי לסדר יותר את עניין האמינות, שרת ה-DNS בוחר בפורט ראנדומאלי שממנו הוא שולח את המידע, ורק מהפורט הזה יהיה אפשר להחזיר לו את המידע, אך את המידע השולח יקבל, כברירת מחדל, בפורט 53.
- בכל תשאול, שרת ה-DNS מחולל ID מיוחד, שרק בעזרת צירופו לתשובה, הוא יוכל לדעת כי אכן מדובר בשרת שאותו הוא תשאל.

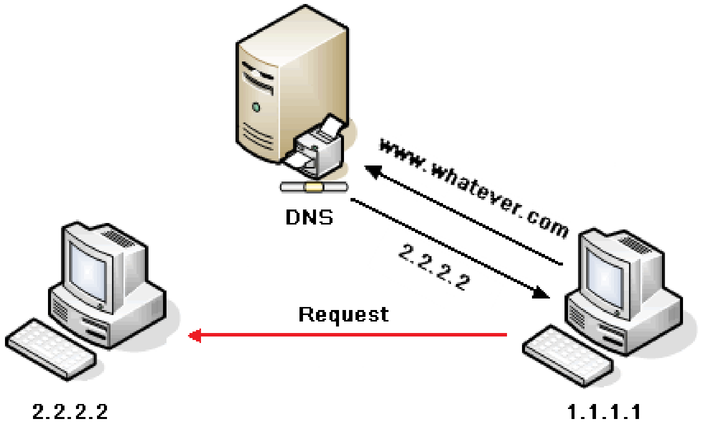
שלב ראשון - המחשב (1.1.1.1) מקבל כתובת מילולית (www.whatever.com) וניגש לשרת ה-DNS לתרגם אותה לכתובת נומרית בכדי לדעת לאיפה לשלוח את הבקשה:



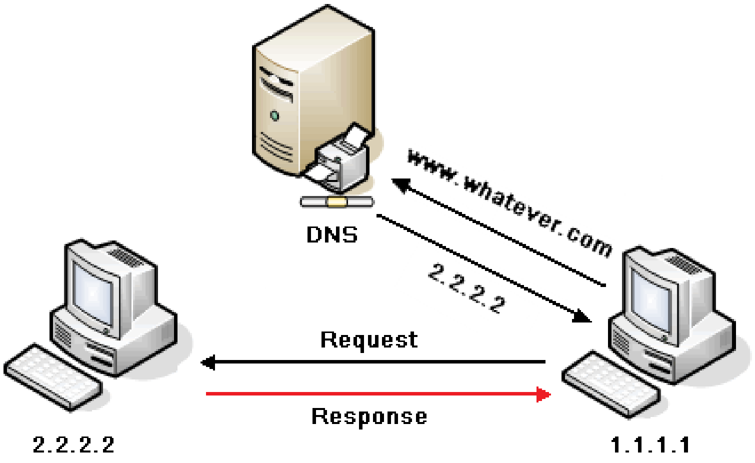
שלב שני - שרת ה-DNS בודק ברשימה שלו לאיזה כתובת נומרית (IP) שייכת הכתובת שהוא קיבל, ומחזיר את התוצאה (2.2.2.2) לשואל (1.1.1.1) :



שלב שלישי - השואל ניגש לכתובת שהוא קיבל משרת ה-DNS ומבקש ממנו תוכן:



שלב רביעי - שרת היעד (2.2.2.2) מחזיר את המידע שביקש הלקוח (1.1.1.1):



כך המחשב המבקש (1.1.1.1) מקבל את המידע שהוא ביקש, מבלי לדעת את כתובתו האמיתית של 2.2.2.2. בפרוטוקול ה-DNS ישנם סוגים רבים של רשומות, ובעזרתן אפשר לתשאל את שרת ה-DNS לגבי שירות מסויים, כגון שרת הדוא"ל, שמו של השרת וכו'.

רשומת לדוגמא:

- A- כתובת ה-IP של אותו שרת.
- MX- השרת האחראי על שליחת וקבלת הדוא"ל.
- NS- מידע לגבי אותו דומיין.
- AAAA - כתובת ה-IPv6 של אותו שרת.

ישנן עוד רשומות אך הבנתם את הרעיון הכללי.

במצבים ובהן שרת ה-DNS מבין כי אין ברשותו את המידע שביקשנו (כתובת הדומיין אינה נמצאת ברשימותיו), הוא מתשאל את שרת ה-DNS שמעליו, המכיל כתובות גלובליות יותר, ומבצע תשאול רקורסיבי עד שהוא מגיע לכתובת המדוייקת.

התשאול הרקורסיבי

נניח ותשאלנו את השרת לגבי הכתובת הבאה: **subdomain.whatever.edu.org.il**

לצורך הדוגמא - שרת ה-DNS מגלה שאין לו מושג מה כתובת ה-IP של אותו דומיין, אז הוא מבצע תשאול (בדיוק כמו שאנחנו תשאלנו אותו) לשרתי DNS שמעליו. לדוגמא, הוא יגיע לשרת הראשי (root) שמצויין ע"י נקודה: ".", וישאל אותו האם הוא מכיר את הכתובת המדוברת.

- השרת יגיד לו: "שמע אחי, אין לי מושג, אבל אני מכיר את מי שאחראי על: **.il**"
- המחשב יפנה אל מי שאחראי על ".il", וישאל אותו את אותה השאלה, התשובה שהוא יקבל תהיה דומה- "שמע אחי, אין לי מושג, אבל אני מכיר את מי שאחראי על: **.org.il**"
- כך הלאה, עד שהוא יגיע ל: **www.whatever.edu.org.il** וישאל אותו מה כתובת ה-IP של הסאב-דומיין: **subdomain.whatever.edu.org.il**
- כשהוא יקבל את התשובה הוא ישלח אותה אלינו בכדי שנוכל לבקש/לשלוח מידע מאותו מחשב.

בסופו של דבר, כל שרתי ה-DNS מתנקזים ל-13 שרתים הנקראים- Root Servers, ודרכם עוברות כל חבילות המידע בצורה זו או אחרת.

מנגנון ה- Caching

בכדי לייעל את הרעיון, ישנו מנגנון Caching, כך שלאחר שהשרת השיג כתובת IP של דומיין מסויים שקיים, הוא מוסיף אותו לרשימותיו (ל-Cache), וכך, אם נתשאל אותו עוד יומיים על אותה הכתובת- הוא יוכל לשלוף לנו אותה במהירות מבלי לרוץ שוב את כל הדרך שהוא עשה.

בפעם הראשונה שנבקש מהשרת את ה-IP של הכתובת:

subdomain.whatever.edu.org.il

הוא אכן יתרוצץ בין כל שרתי ה-DNS בדרך לאותו מחשב בכדי להשיג את כתובת ה-IP שלו, אך מעכשיו- בכל פעם שאנו נתשאל את שרת ה-DNS שלנו לגבי אותה כתובת, הוא לא יתרוצץ וינסה לבדוק לאיזו כתובת IP אותו דומיין שייך, הוא פשוט ישלוף אותה מהרשימה שלו, וכך יקצר את הזמן שאנו נאלץ לחכות.

בכדי לראות את תוכן ה-Caching הנשמר במחשבכם כנסו ל-CMD ושם כתבו:

```
ipconfig /displaydns
```

תקבלו הרבה מאוד בלוקים של פרטי ה-DNS, הבלוקים מורכבים ממספר רשומות.

רשומה לדוגמא:

```
  stun2.l.google.com
----------------------------------------
  Record Name . . . . . : stun2.l.google.com
  Record Type . . . . . : 1
  Time To Live  . . . . : 234
  Data Length . . . . . : 4
  Section . . . . . . . : Answer
  A (Host) Record . . . : 209.85.137.126
```

כמו כן, קיים קובץ אשר בעזרתו אפשר להגדיר למערכת באופן ידני נתוני DNS, הקובץ נמצא:

```
%windir%\system32\drivers\etc
```

ושמו: Hosts

בכדי להוסיף נתונים פשוט כתבו את כתובת ה-IP, רווח (או טאב), ואז את כתובת ה-DNS שאליה אתם רוצים לשייך את כתובת ה-IP.

אם לדוגמא תוסיפו את השורה הבאה בסוף הקובץ:

```
209.85.129.147 www.microsoft.co.il
```

ותעשו PING לכתובת של מיקרוסופט ישראל, המערכת תשלח את הבקשה לכתובת ה-IP שמקבילה אליה בקובץ ה-Hosts, ואתם תגבלו תגובה מגוגל (ה-IP שייחסתם לכתובת של מיקרוסופט ישראל הוא ה-IP של אחד משרתיו של גוגל). אפשר לראות את זה באופן יותר מוחשי, ע"י הפעלת הדפדפן וכניסה לכתובת "www.microsoft.co.il", הדפדפן יציג לכם את עמוד הבית של גוגל, למרות שבשורת ה-URL תראו את הכתובת של מיקרוסופט ישראל. מה יקרה אם מישהו עם כוונות זדוניות, יצליח להגיע לקובץ הזה, ולשנות את כתובת האתר של הבנק שלכם, לכתובת של **אתר מראה** (Mirror Site) של הבנק שלכם, שנמצא בבעלותו של התוקף? כל מידע שתקישו באתר הבנק שלכם, יגיע לידי התוקף. לא חבל?

בכדי לנקות את המידע הנשמר במנגנון ה-Cache של המערכת שלכם, הכנסו שוב ל-CMD וכיתבו את הפקודה הבאה:

```
ipconfig /flushdns
```

שימו לב שאם תיכנסו לכתובת של מיקרוסופט ישראל עדיין תקבלו את האתר של גוגל. למה? כי הקובץ Hosts לא קשור למערכת ה-DNS Cache של מערכת ההפעלה, הוא נוסף אליה.

החולשה

מנגנון ה-Caching נועד להקל עלינו ולהפוך את זרימת הנתונים למהירה יותר, וזה אכן מה שהוא עושה, אך הוא גם יוצר חולשה רצינית בפרוטוקול ה-DNS. מערכת ה-Caching היא מערכת "לומדת", שתשאף להשאר מעודכנת כמה שיותר.

זאת אומרת, כשהיא נתקלת במידע חדש, היא רושמת אותו אצלה בכדי לקצר את התהליך לכשנבקש את אותו המידע בעתיד, ובמקרים ובהם היא תתקל במידע מעודכן (למשל, כתובת של שרת שהתחלפה בעקבות שינוי מיקומו), היא תבין שהיא לא מעודכנת, ותחליף את המידע הישן (הכתובת הקודמת של

השרת, אצלנו) שברשותה, במידע החדש שהיא קיבלה. הרעיון יפה מאוד, אך הוא גם החולשה של הפרוטוקול, המערכת "תלמד" את המידע החדש שהיא קיבלה מבלי לאמת שאכן המידע אותנטי (חוץ מכמובן אותו מספר ID אשר נשלח בכל חבילה), כלומר, שהכתובת החדשה שהיא קיבלה, היא באמת הכתובת המקורית של השרת. דבר המוסיף בעיה הוא השימוש בחבילות UDP.

כך, אם בדרך כלשהי, התוקף יצליח לנחש את מספר ה-ID של ה-Packet שאותו שלח שרת ה-DNS שלנו לשרת ה-DNS מעליו, והוא יהיה מספיק זריז בכדי להחזיר לשרת ה-DNS שלנו תשובה עם מידע שגוי (אך עם ה-ID הנכון), המערכת לא תנסה לאמת שהמידע שהיא קיבלה אמיתי, אלא תסתפק בכך שה-ID נכון, ותעדכן את מנגנון ה-Cache שלה במידע שהתוקף שלח. כך, התוקף יוכל "להרעיל" (ומכאן שם המתקפה) את רשימת ה-DNS שלנו בכתובות מזוייפות, אשר יובילו את הגולש התמים לשרתים אשר מכילים קודים זדוניים, או אתרים פיקטיביים- וכך לגנוב את פרטיו.

דוגמא: אם יצליח התוקף להרעיל את הכתובת: www.Bank.com שהיא במקרה גם כתובת הבנק של הגולש, ויחליף אותה בכתובת ה-IP של שרת אשר מאחסן עליו אתר מראה (Mirror) של אותו הבנק, הגולש לא יוכל לדעת כי אכן מדובר באתר מראה ולא באתר המקורי (כתובת ה-URL תצביע על כתובתו המקורית של הבנק). גם אם הוא ישלח PING לדומיין הבנק, השרת הפיקטיבי יחזיר לו תגובה, ובעצם כל מידע שהוא ישלח לכתובת: www.Bank.com תמיד תגיע לשרת הפיקטיבי, ולאתר הפיקטיבי שנמצא בידי התוקף!

אופן המתקפה

כפי שראינו, בפני התוקף ניצבים שני בעיות עיקריות:

- עליו לדעת מה ה-Source Port שבו השתמש שרת ה-DNS בכדי לתשאל את שרת ה-DNS שמעליו.
- עליו לדעת מה ה-ID שבו שרת ה-DNS השתמש בכדי לתשאל את שרת ה-DNS שמעליו.

ה-Destination Port שאליו התוקף צריך לשלוח את התשובה עם המידע המורעל אינו משתנה. הוא ראנדומלי, אבל קבוע, כך שהדבר מהווה אתגר לא קטן, אך עם זאת, אפשרי לבירור.

הבעיה העקרית היא למצוא את ה-ID שבו השתמש שרת ה-DNS בכדי לתשאל את שרת ה-DNS שמעליו. איך אפשר לעשות את זה? נסביר על ידי הדגמה.

פתחו את תוכנת ה-Sniffer המעודפת עליכם (התוכנה Wireshark מומלצת בחום). תפעילו אותה תחת הפילטר: dns

כנסו ל "CMD", רוקנו את ה-Cache של ה-DNS של המערכת שלכם ע"י:

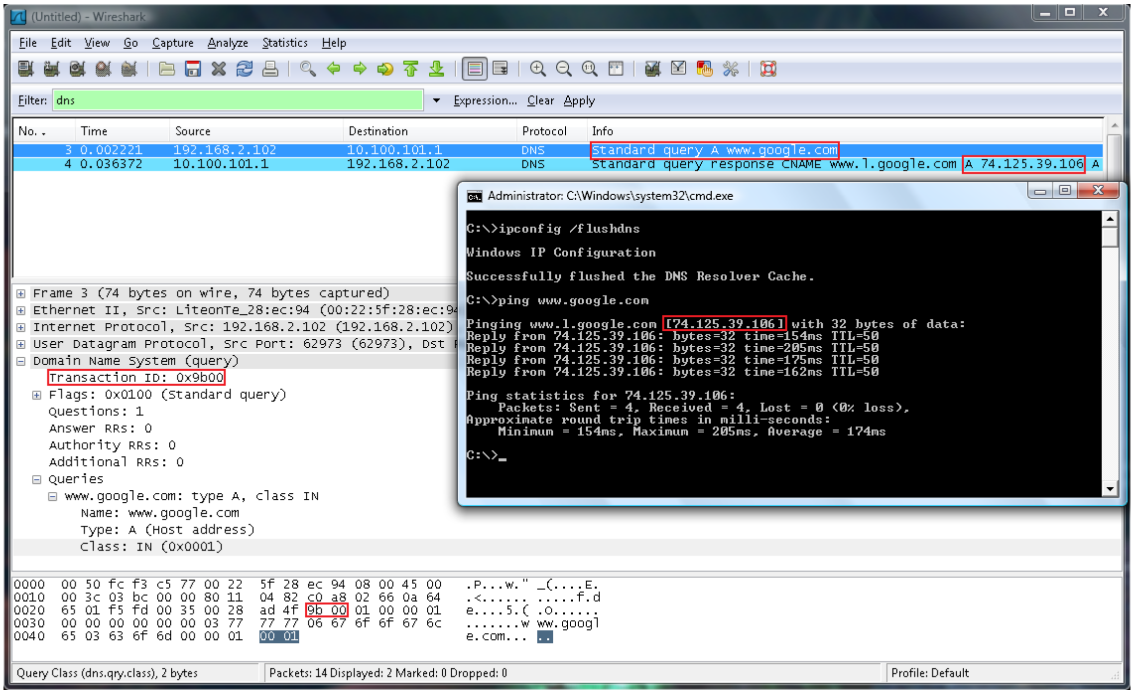
```
Ipconfig /flushdns
```

ושלחו פינג לגוגל:

```
Ping www.google.com
```

כמובן שה-Wireshark יתחיל לצעוק. בגלל שמחקנו את ה-Cache של מערכת ה-DNS שלנו, המערכת נאלצה לברר מי זה google.com, אז היא תשאלה את שרת ה-DNS שמעליכם.

ניגש ל-Wireshark, ונחפש את חבילת ה "Standard Query A" הראשונה שנשלחה, והסתכלו על הפרטים שלה. בהתחלה תראו את ה-Headers של ה-IP, אחריו את ה-Headers של ה-UDP, ואחריהם- את ה-"Transaction ID", וזה בדיוק מה שהתוקף שלנו צריך לנחש. אצלי הערך הוא: 0x9B00, שבדצימאלית זה בדיוק: 39680.



<u>שימו לב</u>: הדבר היחידי שאחראי על אימות אמינותו של ה-Packet בפרוטוקול ה-DNS, הוא מספר בגודל 2 bytes, **זאת אומרת שמספר ה-ID שמאמת את ה-Packet יכול לנוע בין 0 ל-65535!**

בעזרת מתקפת Brute-Force פשוטה (ע"י רשת של זומבים בינונית, למשל), אפשר לעלות על כל הצירופים האפשריים בפרק זמן קצר יחסית.

חלון הזמן של התוקף: על התוקף למצוא את מספר ה-ID לפני ששרת ה-NS של ה-DNS שמעליו, יענה לשרת המותקף. כי אם שרת ה-DNS שמעליו יענה לשרת ה-DNS המתשאל- הוא כבר קיבל תשובה, ואין לו שום סיבה לשאול אותה שוב (בפעם הבאה ששרת ה-DNS ישאל שוב לגבי אותו דומיין, זה יהיה רק לאחר שיפוג הזמן הקצוב ב-TTL שהוגדר ב-Packet). במצב רגיל, לשרתי ה-DNS, לוקח פחות משניה לתקשר ביניהם, וליידע אחד את השני מה כתובת ה-IP של דומיין מסויים, וזה הרבה פחות מהזמן שלוקח לתוקף לנחש את ה-ID המקורי שנשלח ב-Packet.

אז איך בכל זאת יכול התוקף לבצע מתקפה שכזאת בהצלחה? פשוט מאוד- בזמן שחצי מצבא הזומבים שברשותו מנסה לנחש נכונה את מספר ה-ID, החצי השני מבצע מתקפת DDoS / DoS על שרת ה-DNS שאמור להחזיר את התשובה לשרת ה-DNS המתשאל!

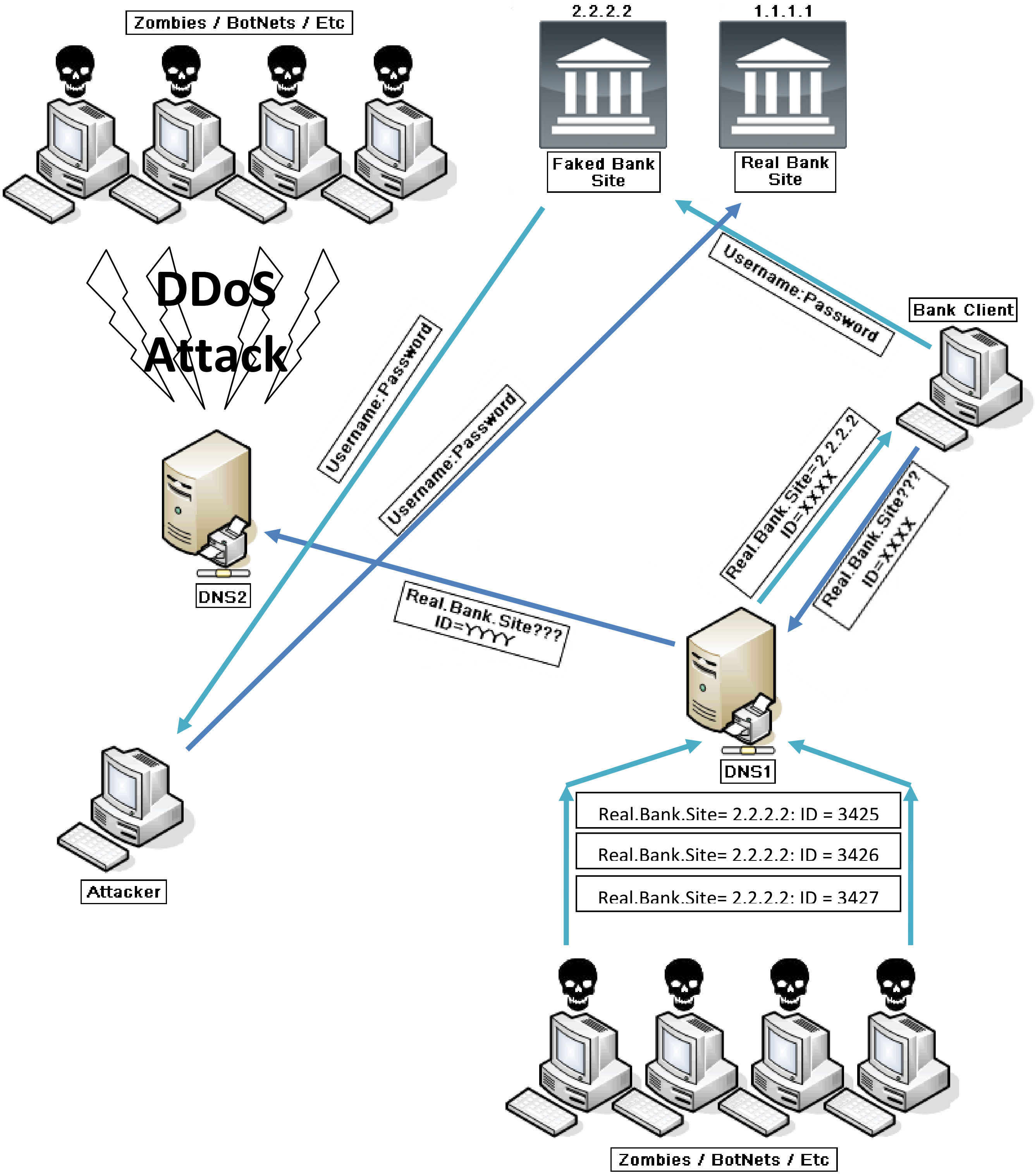
סיכום אופן המתקפה

כמו שאמרנו, הבעיה הרצינית בביצוע המתקפה על מחשבים מחוץ לרשת המקומית שלנו, על ארגונים או על אתרים ספציפיים (אתרים המתעסקים בכספים, כגון Ebay, Paypal, אחרי בנקים גדולים וכו', אתרים לעדכוני חבילות אבטחה במוצרים נפוצים, כגון שרתי עדכונים למערכות אבטחה, שרתי עדכונים לדפדפנים נפוצים, עדכוני לשרתים וכו') היא למצוא את מספר ה-ID. וכמו שאמרנו, בידיים ריקות, לתוקף אין יותר מדי סיכוי. אך אם התוקף יבצע מספר פעולות בפרק זמן קצר- סיכוייו יעלו.

על התוקף לבצע בו זמנית:

- מתקפת מניעת שירות על שרת ה-DNS הנמצא "מעל" שרת ה-DNS המותקף, וע"י מתקפה זו- להפיל, או להאט אותו, וכך להרוויח זמן יקר.
- תשאול מאסיבי של שרת ה-DNS המותקף, לכתובת אותה הוא מעוניין להרעיל, וכך לגרום לשרת ה-DNS לשלוח מספר רב של תשאולים, כך ששרת ה-DNS ישתמש במספר רב יותר של חבילות המכילות מספרי ID שונים- מה שיקל על התוקף בניחוש (ככל ששרת ה-DNS שלח יותר חבילות, כך הסיכוי לנחש ID אחד גדל).
- שליחה מאסיבית של תשובה אחת מורעלת, כך שבכל שליחה מספר ה-ID שונה (ביצוע Brute Force - ניחוש שיטתי של מספר ה-ID).

ככל שהתוקף יפעל במסיביות רבה יותר, בכל אחד מהסעיפים, כך סיכוייו לבצע את המתקפה הזאת בהצלחה- גדלים. לכן, ככל שלרשות התוקף יעמדו מספר רב יותר של מחשבים- כך יגדלו סיכוייו להשלים את המתקפה. פירוט סכמתי של המתקפה, להבהרה ויזואלית של אופן המתקפה בעמוד הבא.


Zombies / BotNets / Etc
DDoS Attack
2.2.2.2
Faked Bank Site
1.1.1.1
Real Bank Site
Username:Password
Bank Client
Username:Password
Username:Password
DNS2
Real.Bank.Site=2.2.2.2 ID=XXXX
Real.Bank.Site??? ID=XXXX
Real.Bank.Site??? ID=YYYY
DNS1
Real.Bank.Site= 2.2.2.2: ID = 3425
Real.Bank.Site= 2.2.2.2: ID = 3426
Real.Bank.Site= 2.2.2.2: ID = 3427
Attacker
Zombies / BotNets / Etc

- **שלב ראשון** - לקוח הבנק (Bank Client) שואל את שרת ה-DNS שלו (DNS1) מה כתובת ה-IP של Real.Bank.Site, הוא משתמש ב-Transaction ID שלא ידוע לתוקף.
- **שלב שני** - שרת DNS1 מגלה כי הוא אינו בעל המידע הדרוש ולכן הוא מתשאל את שרת DNS2.
- **שלב שלישי** - התוקף (Attacker) מגייס זומבים, ומבצע מתקפת מניעת שירות (DDoS) לשרת DNS2 בכדי שהוא לא יוכל לשרת את DNS1.
- **שלב רביעי** - התוקף (Attacker) מגייס עוד זומבים, ומבצע מתקפת Brute-Force על DNS1 עם תשובות (מזוייפות) מ-DNS2. התשובות מכילות מידע כוזב לגבי כתובתו האמיתית של Real.Bank.Site ומפנות לכתובת ה-IP של Faked.Bank.Site - אתר מראה של הבנק אשר נמצא ברשותו של התוקף (Attacker).

התוקף חייב לבצע Brute-Force לערך ה-ID ששרת DNS1 השתמש בו, בכדי לתשאל את DNS2, ורק אם הוא יצליח לגלות את אותו מספר סודי- הוא יצליח לבצע את המתקפה בהצלחה.

- **שלב חמישי** - שרת DNS1 מקבל אלפי תשובות מהזומבים של התוקף, אחת מהתשובות מכילה את מספר ה-ID הסודי שבו הוא השתמש בכדי לתשאל את DNS2, ולכן הוא מניח כי הבקשה אכן התקבלה ממקור אמין ומשגר את התשובה ללקוח הבנק (Bank Client).
- **שלב שישי** - לקוח הבנק (Bank Client) נכנס לאתר המראה (Faked.Bank.Site) שנמצא תחת חסותו של התוקף בהנחה כי הוא אכן נמצא באתר המקורי של הבנק שלו, מכניס את פרטי ההתחברות (Username:Password), ומקבל הודעת שגיאה כי עקב תקלות אתר הבנק לא עובד (או כל הודעה שהתוקף קובע).
- **שלב שביעי** - אתר המראה (Faked.Bank.Site) שולח את פרטי ההתחברות של לקוח הבנק (Bank Client) לתוקף (Attacker).
- **שלב שמיני** - התוקף מתחבר לאתר המקורי של הבנק (Real.Bank.Site), מכניס את פרטי החשבון שהוא קיבל מאתר המראה שלו (Faked.Bank.Site) ומעביר את כל כספו של לקוח הבנק לחשבון סודי בשוויץ.

עוד נקודה חשובה היא שהתוקף **לא חייב לחכות שמישהו ישלח בקשה לשרת ה-DNS בכדי לנסות לזייף אותה**, התוקף לא צריך לדעת מתי לקוח הבנק מתשאל את שרת ה-DNS, הוא יכול לשלוח בעצמו בקשה לשרת ה-DNS בכל זמן ואז לבצע את המתקפה, ולהמתין ללקוח מזדמן.

הנושא הרבה פחות מעניין ומורכב, אך כאשר מדובר במתקפת DNS Cache Poisoning ברשת מקומית (Intranet), הדבר הופך לפשוט ביותר. ע"י היכולת לבצע מתקפת MITM בעזרת Arp Poisoning , בין הנתקף לבין שרת ה-DNS, אפשר בקלות לקבל את כל חבילות המידע הנשלחות מהנתקף, וע"י ביצוע Packet Manipulating פשוט לחבילות ה-DNS, ושינוי התשובה החוזרת משרת ה-DNS, בקלות אפשר לגרום לנתקף לחשוב שהוא נמצא בכל מקום שרוצים. במקרה כזה, אין שום צורך בניחוש מספר ה-ID שהנתקף השתמש בו, מפני שהוא שולח אותו אלינו. וכשאין צורך לנחש את ה-ID, אין צורך במתקפות DDoS, לא על שולח ה-Request ולא על שולח ה-Response.

עוד נקודה היא, שבמצב כזה התוקף לא חייב להרעיל את מנגנון ה-Cache של שרת ה-DNS, הוא פשוט יכול לכתוב סקריפט שמבצע Match & Replace לחבילות המידע העוברות דרכו, וכך לגנוב מידע.

בעזרת כלים מצויינים כגון Cain & Able, אפשר לבצע מתקפות כאלה ע"י לחיצת כמה קליקים, הכלי אפילו מתוכנן לזהות לבד מידע "מעניין" כגון סיסמאות, Certificates וכו'.

קישורים

הגענו לסוף המאמר. להלן מספר קישורים למאמרים שיכולים לעזור לכם להבין את הנושא באופן עמוק יותר:

- ארוך, אבל שווה לעבור עליו, אפילו בריפרוף, ה-RFC של הפרוטוקול:
  - http://www.ietf.org/rfc/rfc1034.txt
  - http://www.ietf.org/rfc/rfc1035.txt
- הערך בויקיפדיה על הנושא: http://en.wikipedia.org/wiki/Domain_Name_System
- האתר של ה-Root Servers, מכיל שמות, מיקומים, ומידע על כל שרתי ה-Root של האינטרנט שלנו: http://www.root-servers.org/

דברי סיום

בזאת אנחנו סוגרים את הגליון השני של Digital Whisper. הגליון הנ"ל נכתב בתקופה מאוד לא נוחה ולחוצה מבחינתנו - ניר שקוע במיליון ואחד דברים בתחום האישי (מה שמנע ממנו לכתוב מאמרים לגליון הנוכחי) ואני כותב לכם מילים אלו תוך כדי טיול באירופה. אך שימו לב שדברים אלו לא מנעו מאיתנו להוציא את הגליון בזמן - למרות כל מה שזה דרש מאיתנו עמדנו בזמנים. יכול להיות שאם היינו מאחרים את תאריך יציאת הגליון הייתם מקבלים מוצר מוגמר יותר עם מספר קטן יותר של טעויות ושגיאות - אבל העדפנו (יותר נכון - העדפתי אני) להוציא את הגליון בזמן בכדי לעמוד בדברינו. בנוסף, הייתי רוצה לנצל במה זאת בכדי להגיד שוב תודה רבה לאורי (Zerith) על כתיבת מאמר לגליון. תודה רבה!

יותר מנשמח לקבל תגובות על הגליון, תיקונים, עיצות וכד' באתר המגזין.

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"

בהזדמנות זו כמו בכל הזדמנות נרצה להזכיר כי נשמח לקבל את הכתבות שלכם ולפרסמן בגליונות הבאים (תתחילו לכתוב, עכשיו!). ניתן לשלוח כתבות דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il

הגליון הבא ייצא בדיוק ביום האחרון של נובמבר 2009 - יש למה לחכות.

אפיק קסטיאל,

ניר אדר,

31 באוקטובבר, 2009