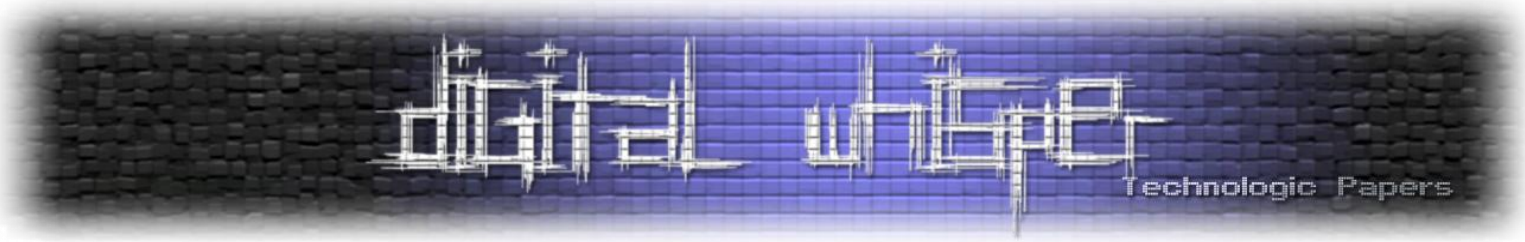


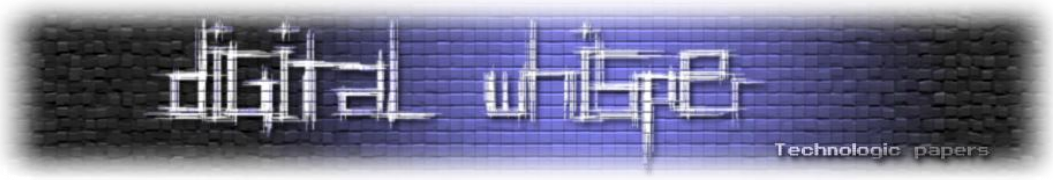
Digital Whisper

גליון 6, מרץ 2010

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורכים:	ניר אדר, סילאן דלאל, Ratinho
כתבים:	אורי (Zerith), אלכס רויכמן, אפיק קסטיאל, בנימין כהן, הרצל לוי, יגאל סולימאני

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



דבר העורכים

ברוכים הבאים לגליון השישי של Digital Whisper – מגזין אלקטרוני בנושאי טכנולוגיה. את הגליון מביאים לכם **ניר אדר**, מהנדס תוכנה, מנהל פרוייקט UnderWarrior (www.underwar.co.il) ו**אפיק 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.

סוף פברואר הגיע והגליון השישי של Digital Whisper סוף סוף בחוץ, רק עוד שלושה גליונות וניר חייב לי ארוחה! :)

בזמן האחרון אנחנו שומעים עוד ועוד על רשתות Botnets חדשות שחוקרי אבטחת מידע מגלים, אם זה גירסאות שונות ומשונות של Zeus, ואם זה שרידים של ה-Storm, אם זה חשבונות חשודים ב-Twitter או ב-Tumblr ואם זה דרכי שליטה בעזרת Emule. בגליון הנוכחי שילבנו מספר מאמרים המתקשרים לנושא, מקווים שתאהבו.

וכמו בכל פעם- לפני שנעבור לחלק החשוב, אנו רוצים להגיד תודה למי לכל שעזרו לנו בהגשת הגליון:

- אלכס רויכמן שהגיש לנו מאמר מבריק על המתקפה "Cross Site History Manipulation".
- אורי (שזה כבר המאמר השלישי שלו במגזין) שכתב לנו מאמר מרתק על עולם ה-Rookits.
- הרצל לוי (מופע שני במגזין שלנו) שכתב לנו ניתוח על ה-Conficker.
- יגאל סולימאני (מופע שני גם שלו במגזין!) שכתב לנו מאמר על טכנולוגיות ה-Firewalling השונות.
- בנימין כהן שכתב לנו סיקור על טכנולוגיות ההצפנה של מיקרוסופט- EFS ו-BitLocker.
- תודה רבה ל-Ratinho על שעזר בעריכת הגליון.

קריאה מהנה!

אפיק קסטיאל ניר אדר

תוכן עניינים

BotNet- מה זאת החיה הזאת?

מאת אפיק קסטיאל (cp77fk4r)

ביום שישי, 25 לאוגוסט שנת 2006, קיבל כריסטופר מקסוול, בחור בן 21, את גזר דינו בבית המשפט הפדרלי בסיאטל על ידי השופטת מרשה פצ'מן- למעלה משלוש שנות מאסר ולאחריהם עוד שלוש שנים של שחרור תחת פיקוח, בנוסף לפיצויים בסך 252,000$ שהוא נאלץ לשלם. מקסוול, בין השאר, היה אחראי על כתיבת והפצת תולעת Botnet שפעלה במשך למעלה משנתיים. בין שאר המחשבים שהצליחה תולעת זאת להדביק, נכללו גם למעלה מ-400 מחשבים של רשויות הבטחון של ארצות הברית.

המקרה של כריסטופר מקסוול הוא אחד מני מקרים רבים שבהם אנחנו שומעים על החיה הזאת, לאחרונה, אנו שומעים על מקרים אלו יותר ויותר (ויותר ויותר ויותר ויותר).

חוקרי אבטחת מידע ואינטרנט מעריכים כי קיימות עשרות (אם לא מאות) של רשתות כאלה והן רק גדלות ומתחזקות מיום ליום.

מה זה בכלל Botnet?

תוכנת Botnet היא שילוב מתבקש וקטלני במיוחד בין מספר תוכנות זדוניות בעלות אופי ותכונות שונות שהולחמו לתוכנה אחת. הרעיון הוא לקחת תכונות שונות של מזיקים שונים, לשלבן יחד וכך ליצור תוכנה זדונית אחת ש"נהנית" מתכונות אלו. לדוגמא:

- התכונה העיקרית של **תולעת** היא יכולת ההתרבות שלה, אם זה בעזרת מנגנוני Mass-Mailing (כמו ה-Mydoom) או ע"י הדבקה של קבצים (כמו ה-Melissa), אם זה ניצול חולשה באחד מרכיבי המערכת (כמו ה-SQL Slammer) או אם זה בעזרת שימוש ברשתות Peer to Peer (כמו ה-Tibick.f)
- התכונה העיקרית של **Rootkit** היא יכולת ההסוואה שלו, אם זה בעזרת התחפשות לכלי מערכת קיימים או השתלטות עליהם (כמו למשל ה-SHV4 או ה-SHV5), אם זה בעזרת הזרקת הקוד שלו לתוך תהליך קיים (Vanquish), אם זה בעזרת ביצוע Hooking לפונקציות מערכת או אם זה בעזרת שינויי ערכים חשובים בתהליכי מערכת.
- התכונה העיקרית של **Trojan Horse** היא יכולת השליטה בו מרחוק, כמו למשל ה-Back Orifice שכתב Sir Dystic מ-Cult o the Dead Cow (cDc).

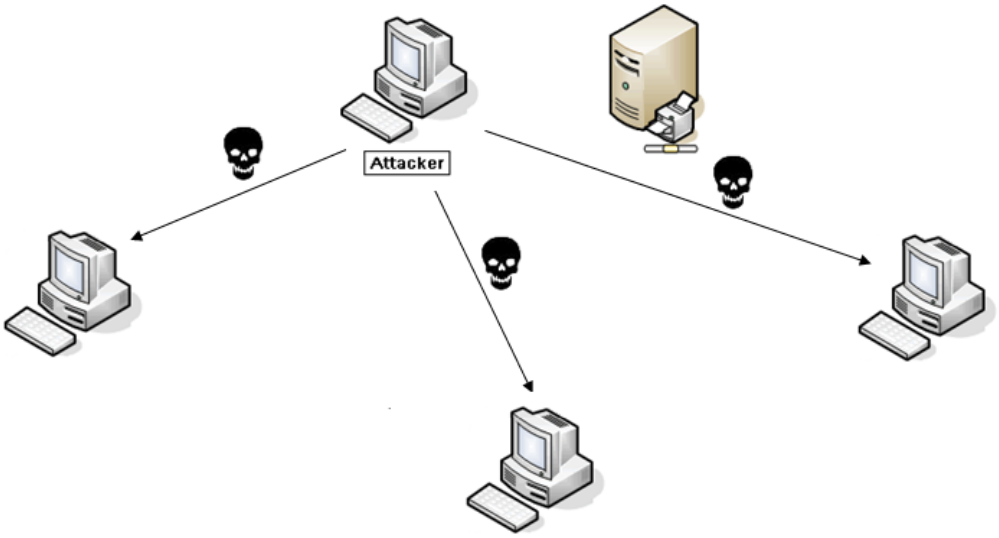
שילוב של שלושת מנגנונים אלה יוכלו ליצור תוכנה זדונית שגם תפיץ את עצמה כמו תולעת, בעזרת מנגנוני Mass-Mailing או ניצול חולשה במערכת ההפעלה, גם תסתיר את הפעולות שלה בעזרת מנגנונים המאופיינים בעיקר ל-Rootkits כך שהן יהיו כמעט שקופות, וגם תאפשר ליוצר שלה לשלוט על המחשב מרחוק.

אם כלי בעל יכולות כאלה או דומות נכתב בצורה נכונה, תוך פרק זמן קצר מאוד יהיו בידי היוצר שלו מספר רב מאוד של מחשבים הנמצאים בשליטתו ורק מחכים לקבל ממנו פקודות. מחשב אחד כזה נקרא "זומבי" (או Bot) וכלל המחשבים הנמצאים תחת רשת (Net) כזאת נחשבים "צבא", או פשוט- Botnet.

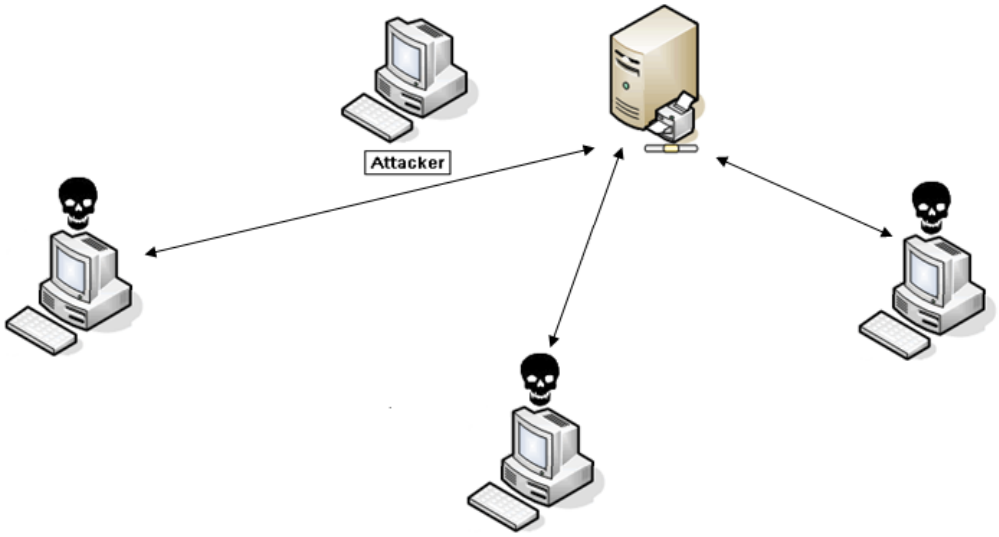
בשל הפוטנציאל הענקי שיש לכלי שכזה, לא מנהלים את צבא הזומבים בעזרת אפליקציה ממחשב יחיד, אלה בעזרת שרתי IRC או שרתי HTTP.

מעגל החיים של Botnet

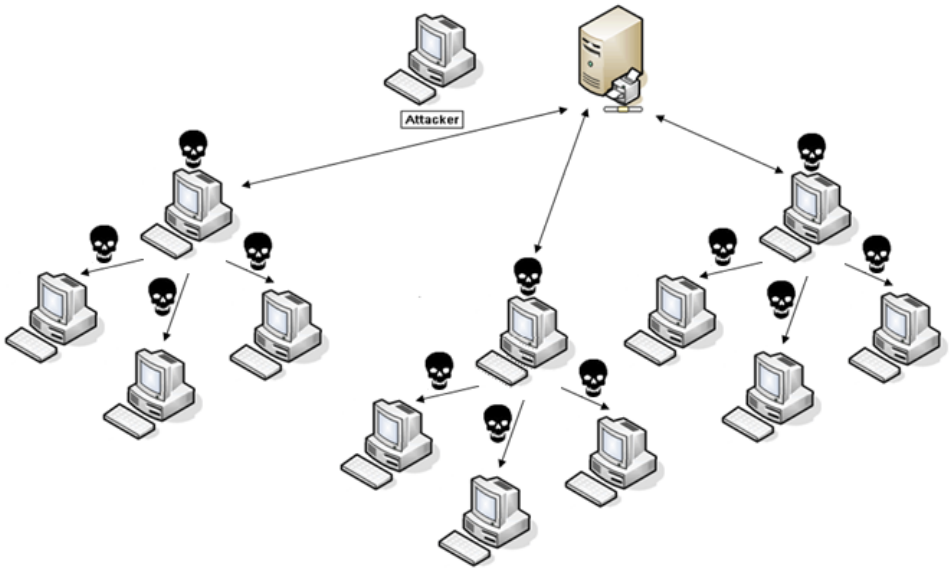
- **שלב ראשון-** התוקף מפיץ את ה-Botnet למספר קורבנות:



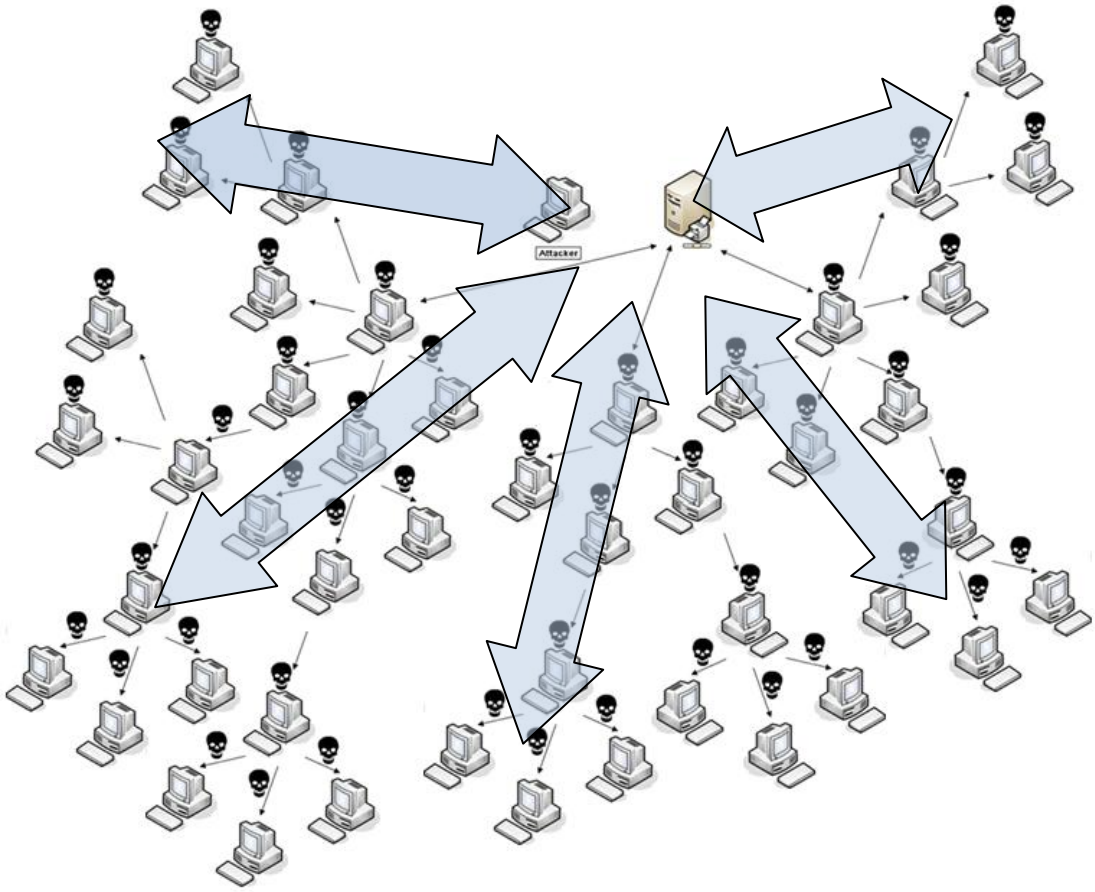
- **שלב שני-** ה-Botnet מדביק את הקורבנות, מחבר באופן שקוף את המחשבים לשרת ה-IRC (שרת ה-"Command & Control") ומחכה לפקודות מהתוקף:



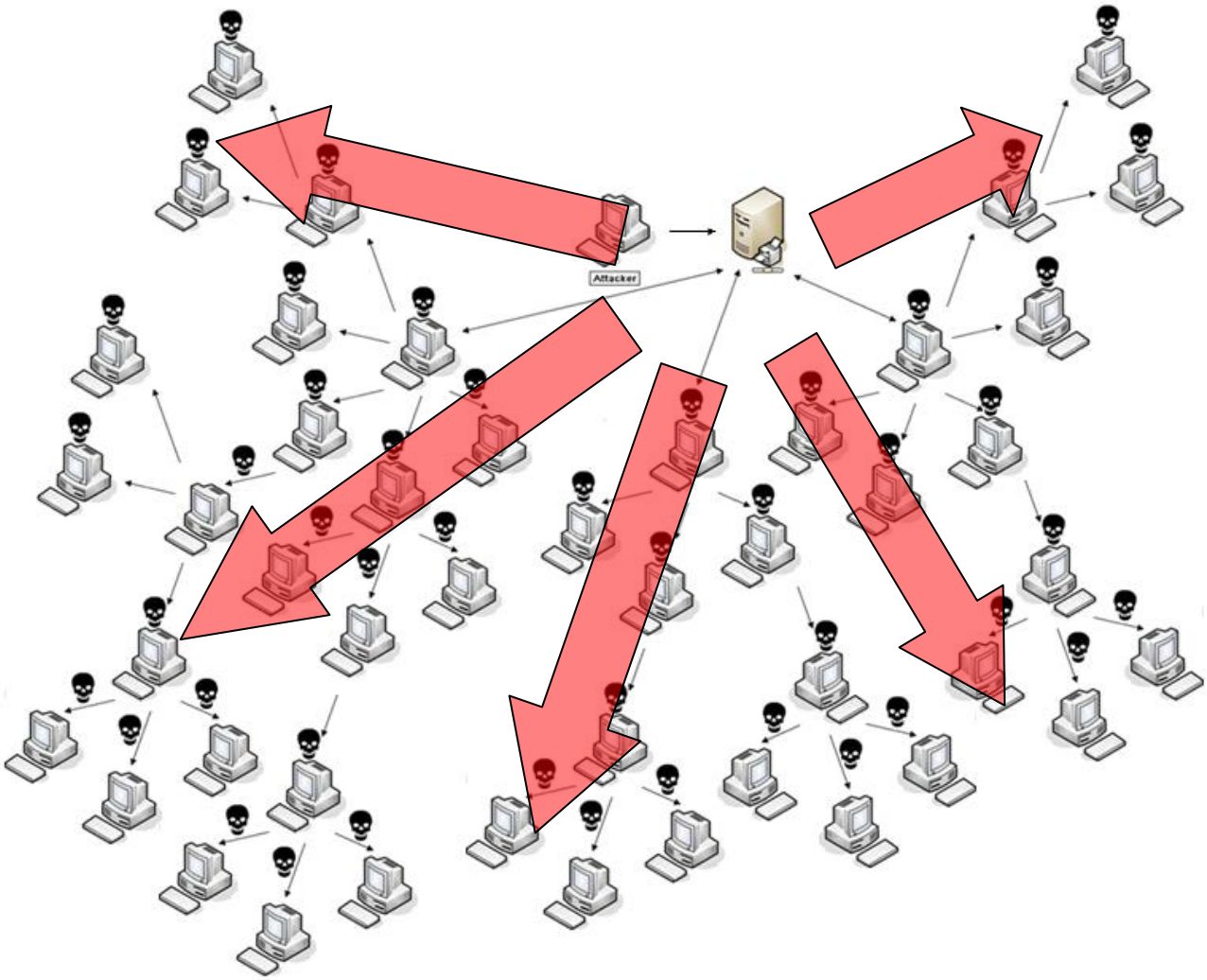
- **שלב שלישי-** במקביל להתחברות לשרת הבקרה של התוקף, מתחילים המחשבים הנגועים לסרוק את הרשת או לשלוח מיילים עם קוד זדוני במטרה להדביק עוד ועוד מחשבים:



- **שלב רביעי-** התחברות המחשבים הנגועים לשרת הבקרה של התוקף ונסיון הדבקה של מחשבים נוספים:



- **שלב חמישי –** לתוקף שליטה מלאה על כלל המחשבים ברשת ה-Botnet שלו. בכדי לשלוח פקודה, הוא מתחבר לשרת ה-Command & Control, משגר פקודה וכלל המחשבים מבצעים אותה. מה הוא יעשה בהם? זה תלוי מי ישלם יותר. האפשרויות שעומדות לרשותו רבות:
  - **ביצוע DDoS על מאסיבי אתרים ברשת**
  - **שימוש במחשבים להפצת ספאם**
  - **גניבת סיסמאות ופרטים אישיים של בעלי המחשבים**
  - **ניצול כח המיחשוב של הרשת ליישום Rainbow Cracking למפתחות RSA/MD/SHA וכו'**
  - **מכירת הרשת לאירגוני Cyber-Mafia (כמו למשל ה-RBN)**



כח עיבוד שכזה מאפשר לתוקפים לבצע התקפות על אתרים של חברות אנטי וירוס או אתרים שמספקים תמיכה וכלים להסרת ה-Botnet וכמובן - ככל שיעבור הזמן כך רק תגדל אותה הרשת ותהפוך חזקה יותר.

טכניקות בשימוש ה-Botnets

לא פשוט לצוד רשת כזאת, וקיימות מספר טכניקות בהן משתמשים כותבי ה-Botnets בכדי לעשות את התהליך קשה אף יותר.

שימוש ברשתות Fast-Flux

במידה ותצורת ההתקשרות של המחשבים הנגועים עם שרת ה-Command & Control הייתה ישירה, לא הייתה כל בעיה להאזין לתעבורה ולראות מאיפה המחשבים הנגועים מקבלים את הפקודות ואז להוריד את שרת השליטה. אך כותבי ה-Botnets משתמשים במנגנון DNS המכונה "Fast-Flux".

העקרון הוא שברשתות IP-Flux כתובת DNS אחת תחזיר בכל פעם את המידע מכתובת IP שונה המשוייכת כל פעם לשרת Proxy שונה המגשר בין המחשב הנתקף לבין שרת ה-Command & Control. בידי התוקפים נמצאים אלפי מחשבים כאלה המרכיבים את רשתות ה-Fast-Flux. קיימים מספר סוגי רשתות Fast-Flux. רשת Fast-Flux המבוססת על IP-Flux יכולה להיות ממומשת באופן של:

- Single-Flux
- Double-Flux

בנוסף קיים מנגנון בשם Domain-Flux, כאשר הרעיון הוא בדיוק הפוך מ-IP-Flux, במקום ש-DNS אחד יפנה למספר רב של כתובות IP, מדובר במספר רב של כתובות DNS שמפנות לכתובת IP אחת. מימוש אחד לדוגמא הוא השימוש ב-Domain Wildcarding, פשוט מאוד:

```
sdfasdsffds.digitalwhisper.co.il
vzcvvcas34.digitalwhisper.co.il
efasaa32ds.digitalwhisper.co.il
vs1caddbbs.digitalwhisper.co.il
xxf555fdg.digitalwhisper.co.il
```

כולם מפנים לאותה כתובת IP, אך כך שרת ה-Command & Control יכול לזהות פניה ספציפית. העניין דומה מאוד לרעיון שעומד מאחורי טכנולוגיות ה-Balancing הממומשים באתרים בעלי תעבורה נרחבת.

ישנן רשתות המוסיפות עוד שכבת אנונימיות בעזרת שימוש במנגנון הנקרא "Blind-Proxy Redirection".
למידע נוסף אודות רשתות ה-Fast-Flux אפשר לקרוא בסדרת המאמרים KYE של Honeynet בנושא:

http://www.honeynet.org/papers/ff/

מנגנוני עדכונים

כותבי ה-Botnets מיישמים מנגנונים לעדכון וקטורי התקיפה של התולעים שלהם, ממש כמו שיצרנית תוכנה בשוק מיישמת מנגנון המאפשר לה לעדכן את התוכנות שאנו רוכשים (כדוגמאת LiveUpdate של Microsoft או מנגנוני עדכון החתימות של תוכנות האנטי-וירוסים למניהם).

נניח וכותב ה-Botnet יישם תולעת המפיצה את עצמה בעזרת ניצול החשיפה MS08-067, מה שצריך לעשות בכדי לעצור את התולעת מלהמשיך ולהפיץ את עצמה זה פשוט לעדכן את המערכת בעדכון הרלוונטי. בכדי להתגבר על מקרים כאלה, כותבי התולעים מיישמים מנגנוני עדכון שמאפשרות להן לעדכן את וקטורי התקיפה של התולעים ב-0day חדש, וכך, במידה ויצא עדכון לחשיפה שאותה הוא ניצל- הוא מעדכן את התולעת להמשיך לתקוף בעזרת וקטור חשיפה חדש, וכך להמשיך להפיץ את עצמה.

שיתוף פעולה בין רשתות Botnets

לטכניקה הזאת אין עדיין הוכחות ממשיות, אבל לאחרונה חוקרים סבורים כי אכן יש שימוש בטכניקה כזאת, כדוגמאת Kneber. מצד שני, חוקרים אחרים סוברים כי מדובר בעצם ב-Botnet המוכר Zbot/ZeuS.

הרעיון הוא שבמידה ותולעת אחת הצליחה לחדור לתוך מחשב, היא מורידה אליו ומדביקה אותו בתולעים אחרות, וכך גם הן עושות, הדבר מקשה מאוד על הסרתן ובמידה והצליחו להסיר את אחת התולעים - התולעים האחרות דואגות להוריד גירסא מחודשת שלה. טכניקה זו מגדילה בהרבה את יכולת ההשרדות של התולעת על המחשב בנוסף על כך גם מוכפלת מהירות ההתרבות שלה.

שימוש במנגנוני ניהול מתוחכמים

מנגנונים כאלה לא נצפו במספר רב של Botnets, אך דוגמא טובה אפשר למצוא ב-Botnet המכונה Storm. כיום, דרכי השליטה המוכרות ביותר ב-Botnets הן:

- **בעזרת פרוטוקול ה-HTTP:** מנהל הרשת מעדכן קובץ מוגדר מראש, והמחשבים הנגועים יודעים לבדוק כל פרק זמן קצר האם יש שינויים בקובץ.

- **בעזרת פרוטוקול ה-IRC:** המחשבים הנגועים מחוברים 24 שעות לשרת ה-IRC ומקבלים פקודות און ליין ממנהל הרשת.

בשני המקרים, נתוני ההתחברות לשרת רשומים Hard-Coded בתוך קוד התולעת. בעזרת ביצוע Sniffing למידע הנשלח מהתולעת או בעזרת Reverse Engineering חוקרי תולעים מסוגלים לשלוף את נתוני השרת, בין אם מדובר בסיסמא לערוץ ה-IRC שאליו מתחברים כלל התולעים, או אם מדובר ב-Credentials שדורש שרת ה-HTTP לזיהוי.

צורת הניהול של ה-Storm מתבצעת בעזרת רשתות Peer to Peer, באופן הבא:

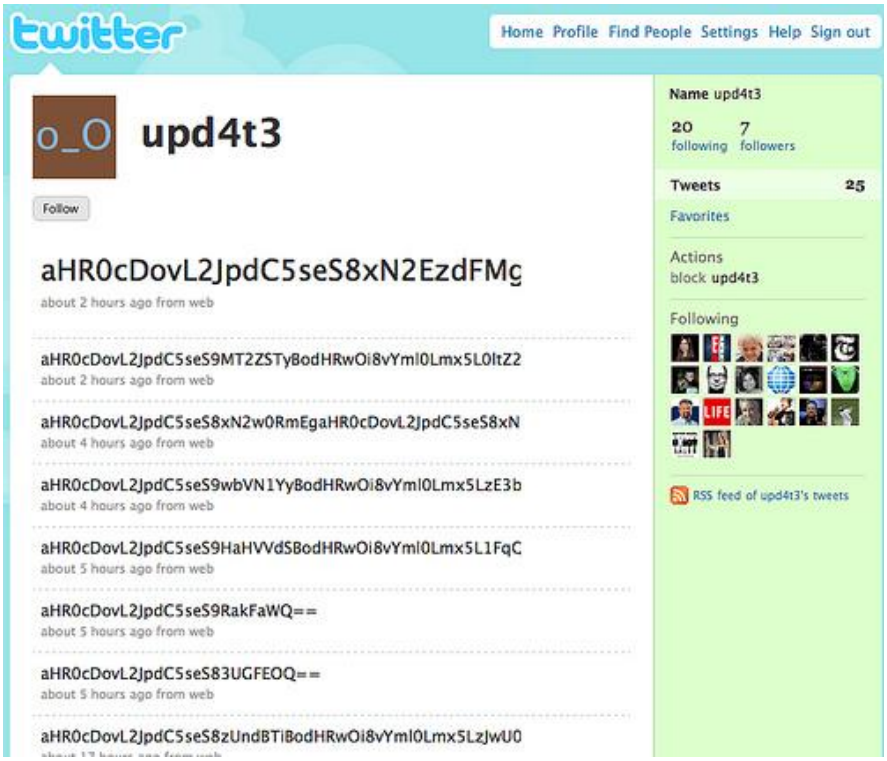
- מנהל הרשת מפרסם ברשת ה-p2p מחרוזת מסויימת שהוגדרה מראש ובנוסף- צמוד לאותה המחרוזת, מפרסם המנהל ערך מוצפן, לדוגמא:
  a@432&dAB=/a3sfaSFwFA5f35aFfW3462sAfASdefraUpO
- המחרוזת שאותה מפרסם מנהל הרשת הוגדרה קודם לכן (Hard-Coded) בתולעים, והן יודעות לחפש אותה (a@432&dAB=) באותה הרשת (לפעמים מדובר במספר מחרוזות שונות שעל התולעים לחפש וכל פעם מחרוזת אחת נמצאת בשימוש) ולשלוף ממנה את הערך המוצפן (a3sfaSFwFA5f35aFfW3462sAfASdefraUpO).
- בתוך התולעים מיושם מנגנון אשר יודע לפענח את המחרוזת ולקבל ממנה כתובת IP או מיקום של קובץ על שרת אינטרנט.
- מנהל הרשת העלה מבעוד מועד קבצים לאותו שרת המכילים פקודות/Payloads לתולעים.
- התולעים מפענחות את המחרוזת המוצפנת, ניגשות לכתובת ה-IP, מורידות את הקבצים ומבצעות את הפקודות שהשאיר מנהל הרשת.

מדובר במנגנון ניהול קצת פחות אמין, אך בצורה כזאת, מנהל הרשת מרוויח מספר דברים:

- **קושי חיזוי-** קשה לעקוב אחרי ניהול התולעת ומי שלא יודע מה הן המחרוזות שהתולעים מחפשות ברשתות השיתוף לא יוכל לדעת לאיזה שרת יש לפנות. גם אם הוא ביצע האזנה לתעבורת התולעת בעת התכתבותה עם השרת לאחר שפיענחה את המחרוזת המוצפנת, הוא יוכל לדעת רק באיזה שרתים מאוחסנות פקודות ישנות של התולעת, אבל לא שרתים עתידיים.
- **אנונימיות-** מנהל הרשת יכול לקבוע בכל פעם מחדש היכן לאחסן את הפקודות למתקפה הבאה וכן לקבוע כל פעם מהיכן לפרסם את המחרוזות הבאות ברשת שיתוף הקבצים – דבר המקשה באופן ניכר על זיהוי מיקום מנהל הרשת.

ה-Storm הייתה תולעת חכמה שחידשה מספר דברים בכל עולם ה-Botnets, אך פתאום, לקראת סוף שנת 2008, בלי שום סיבה נראת לעין- הפסיקה התולעת את הפעילות שלה. ספקולציות רבות צצו בנושא זה, חלק מהבלוגרים אמרו שחוקרי תולעים הצליחו להשתלט על הרשת ולשתק אותה, חלק אמרו שמדובר בפעילות של ה-FBI ויש אפילו שאמרו שיוצר התולעת פשוט מת.

תולעת נוספת שנצפתה עושה שימוש במנגנון ניהול מיוחד היא תולעת אנונימית שמנהל הרשת שלה ניהל אותה בעזרת חשבונות באתרי ה-Micro-blogging המוכרים Twitter, Tumblr ו-Jaiku.com. בשני האתרים נפתחו חשבונות משתמש תחת השם "upd4t3" ובעזרת "Status Massages" מקודדות שלח מנהל הרשת פקודות לתולעים שידעו להאזין לאותם החשבונות דרך שירותי RSS:



(במקור: http://asert.arbornetworks.com/2009/08/twitter-based-botnet-command-channel/)

תולעת זאת התגלתה במקרה ע"י Jose Nazario. במקרה זה יוצר התולעת פחות הקפיד על שימוש במנגנוני הצפנה ורק קידד את המידע שהוכנס ל-"Status Massages" ב-Base64, כך שלא הייתה בעיה להתחקות אחר השרתים. לדוגמא, ה-"Status Massage" הבא:

```
aHR0cDovL2JpdC5seS9MT2ZSTyBodHRwOi8vYml0Lmx5L0ltZ2
```

תורגם ע"י התולעת (Base64):

```
http://bit.ly/LOfRO http://bit.ly/ImZ2
```

כתובות אלו יכולות להיות או מטרות לתקיפה, או עדכונים שעל התולעת להוריד וכו'. שיטה נוספת היא השיטה שנצפתה בגרסאות שונות של ה-ZeuS משתמשות לניהול ובקרה- שירותי ה-EC2 של Amazon.

שימוש בהצפנה, מנגנוני אימות ו-Obfuscation

במספר לא קטן של תולעים נצפה השימוש בהצפנת הפקודות והמידע בתקשורת הנתונים המשמשת את ה-Botnets, נצפה שימוש בחתימות דיגיטליות מבוססות אלגוריתמים מתקדמים כגון MD6, שימוש ב-PKE, יישום מנגנוני RSA בעלי מפתחות גבוהים ועוד. בנוסף, נצפה שימוש רב בשיטות Obfuscation שונות, בין אם מדובר בביצוע Code Obfuscation או בזיהוי נסיונות התחקות אחר התולעת ואז שינוי דפוס ההתנהגות שלה.

בגליון זה מפורסם מאמר על ה-Conficker שנכתב על ידי הרצל לוי ועל-ידי, בו פירטנו באופן מובן ומפורט יותר את השימוש בשיטות אלו הנמצאות בגרסאות השונות של ה-Conficker.

היקף הנזק

קשה מאוד לאמוד את כמות הנזק הנגרמת מרשתות ה-Botnets, גם מפני גודל רשת האינטרנט, גם מפני שחברות וגופים אשר נפגעו מהן לא ממהרות לפרט כמה נזק נגרם להם (פגיעה בתדמית) ומעדיפות לנסות להתמודד עם התופעה לבד, וגם מפני שקשה מאוד לצפות מה יהיה הצעד הבא של מנהלי הרשתות הללו. בעזרת כח מיחשוב עצום שכזה ניתן לבצע כמעט כל דבר וכשגופים כגון ה- Russian Business Network שולחים יד ומקדמים תופעות דומות, כסף לא חסר והכיוון הוא רע מאוד. אם בעבר היקף נזק של תולעי האינטרנט הראשונות הגיעה ל-6000 מחשבים (תולעת ה- Morris) וזה היה נחשב לנזק רציני ביותר, הרי שהיום מדובר בבדיחה.

עד לפני כמעט שנתיים, תולעת ה-SQL Slammer עמדה בראש רשימת התולעים שהדביקו את הכמות המירבית של המחשבים (שרתי MS SQL) – המספר כמעט בלתי נתפס, מדובר ב-75,000 מחשבים בפחות מעשר דקות! יש עדויות לכך שהתולעת אף **גרמה להאטה בקצב רשת האינטרנט העולמית**.

ככל שעובר הזמן כך מתפתחות טכנולוגיות חדשות ולא רק בשוק התוכנה ה-"רגיל", אלא גם בעולם הוירוסים. בתחילת שנת 2009, היקף ההדבקה של ה-Conficker עמד על כמעט 9 מיליון מחשבים! קשה להבין איזה כח עיבוד יש בכזאת כמות של מחשבים וקשה עוד יותר לחשב את כמות הנזק שיכולה להגרם ממנו, שלא נדבר על כמות המידע הרגיש שמאוחסן על מחשבים אלו.

סיכום

אם בעבר שמענו על סיפורים כמו הסיפור של GRC.Com, כיום אנחנו שומעים על מקרים כאלה בתדירות יומיומית. במידה ותתרחש Cyber-Attack רצינית, רשתות ה-Botnets יהיו לכלי הנשק המרכזי והמשפיע ביותר במלחמה זו. חשוב להיות מעודכנים תמיד ובמידה והתגלו פרצות באפליקציות כמו דפדפנים או יישומי אינטרנט אחרים - יש עדיפות עליונה לא להשתמש בהן עד שלא יפתרו בעיות אלו. כמו במקרה של תולעת ה-Conficker, חברת Microsoft שיחררה טלאי אבטחה לכשל שאותו ניצלה התולעת מספר חודשים לפני שהתולעת שוחררה. במקרים אחרים אין יותר מדי מה לעשות, כמו במקרה של Operation Aurora- שחברת Microsoft ידעה על כשל האבטחה כמעט ארבעה חודשים לפני המתקפה ולא שחררה שום טלאי.

XSHM - Cross-Site History Manipulation

מאמר מאת: אלכס רויכמן, ארכיטקט ראשי ומנהל מעבדת המחקר בצ'קמרקס

אפליקציות ווביות טיפוסיות מורכבות משני רכיבים – רכיב השרת (server-side) ורכיב הלקוח (client-side). היסטורית, רכיב השרת תמיד משך את תשומת לבם של החבר'ה הרעים, אך בשנים אחרונות המגמה הזאת התהפכה ואנו רואים יותר ויותר התקפות על רכיבי הלקוח (כגון XSS, CSRF, JSON Hijacking Clickhacking). ההסבר להיפוך המגמה הוא כנראה ש"יותר קל" להגן על רכיב השרת – יש רק **אחד** כזה ואם כותבים קוד בטוח, מקנפגים את האפליקציה כמו שצריך, משתמשים ב- WAF וכו' אז מקבלים שרת יחסית מוגן. לא כך עם רכיב הלקוח – יש **הרבה** כאלה ולהגן על כמות גדולה קשה יותר מאשר על אחד: תמיד הפורץ יכול למצוא מישהו שמריץ דפדפן בגרסא לא מעודכנת ותמיד יש משתמשים תמימים שלוחצים על לינקים מאתרים מפוקפקים.

אחד המנגנונים הכי חשובים בהגנה על רכיבי הלקוח הוא מנגנון המכונה Same Origin Policy (SOP). במשפט אחד, המנגנון הזה מונע מאפליקציות ממקורות (origins) שונים לשתף ביניהן את התוכן שלהן. כך, למשל, אם משתמש גולש לאתר הבנק שלו ובו זמנית פותח אתר זדוני – האתר הזדוני אינו מסוגל לגנוב מידע מאתר הבנק. לכן SOP מונע מאתרים זדוניים להתייחס לתכנים מאתרים אחרים ובכך מגן על האתרים "טובים" מהתקפות של האתרים ה"רעים", כאשר ההגנה הזאת ממומשת בתוך הדפדפנים של המשתמשים, משמע – ברכיב הלקוח.

כיום, קיימות מספר פרצות ידועות במימוש של מנגנון SOP בדפדפנים הקיימים, פרצות אלה משתמשות באלמנטים הקיימים בצד הלקוח המשותפים לאפליקציות שונות – כמו מנגנון זיכרון המטמון של דפדפנים, או בערוצים נסתרים אחרים כמו timing. במאמר זה נציג פרצה נוספת במנגנון SOP שמנצלת את נקודת התורפה שהתגלתה בניהול אובייקט ההיסטוריה בדפדפים השונים.

ישנם שני סוגים שונים של רכיבי היסטוריה: הסוג הראשון מתייחס לשמירה מקומית של עותקים מדפים שמשתמש גלש אליהם על מנת לזרז גישות עתידיות לדפים האלה – סוג של רכיב cache. זו ההיסטוריה הקבועה (persistent) של הדפדפנים. הפרצה הידועה במנגנון ניהול של היסטוריה הקבועה נקראת CSS history hacking והתגלתה לפני כשלוש שנים על ידי חוקר אבטחה בשם ג'רמי גרוסמן, שמצא כי על ידי הצגת קישורים לאתרים שונים מתוך אתר הפורץ ודגימת צבע של קישורים אלה, ניתן לדעת מה הם האתרים שהמשתמש גלש אליהם בעבר.

הסוג השני של היסטוריה מתייחס לרשימת כל אותם הדפים שמשתמש גלש אליהם מתוך החלון הנוכחי של הדפדפן. לא כמו עבור ההיסטוריה הקבועה, ההיסטוריה הזאת נמחקת כל פעם שחלון הדפדפן נסגר, לכן זוהי היסטוריה זמנית. למעשה, ההיסטוריה הזאת מיוצגת על ידי אובייקט תכנותי שזמין מסקריפטים (JS/VBS) ומנוהל כמערך של הלינקים שהמשתמש פתח מהחלון הנוכחי.

אובייקט ההיסטוריה הינו אובייקט גלובאלי, יצרני דפדפנים הבינו זאת ומנעו גישה לתוכן האיברים שבתוך המערך של היסטוריה, על מנת לא להפר את מנגנון SOP. עם זאת, במחקר שערכנו, התגלה כי מאפיין האורך של מערך ההיסטוריה הוא גלוי וזמין לכל אפליקציה ועל ידי ניצול תכונה זו, ניתן להפר את מנגנון SOP ולפגוע בפרטיות של המשתמשים. אנו חושדים כי התוקפים ידעו וניצלו את הפרצה הזאת ולכן רצינו לחשוף אותה לקהילת המפתחים ומומחי אבטחת מידע.

נמחיש את השימוש באובייקט היסטוריה על ידי דוגמא פשוטה זאת: נניח כי משתמש פותח דפדפן וגולש לחמישה דפים שונים. כעת המשתמש ניגש לדף שישי אשר שנפתח מהאתר הפורץ. הפורץ קודם כל יכול לדגום את אורך אובייקט היסטוריה ולגלות שערכו 6, משמע המשתמש גלש לחמישה דפים/אתרים לפני שהגיע לאתר הפורץ. כפי שנראה מייד, פגיעה בפרטיות המשתמש עלולה להיות הרבה יותר גדולה במקרים מסוימים ואפילו לגרום לדליפת מידע מאוד רגיש. להתקפה שמתבססת על ניצול הגישה לאורך ההיסטוריה קראנו Cross-Site History Manipulation או XSHM. כל אפליקציה שמכילה תבנית הבאה של הקוד, חשופה במידה כזאת או אחרת להתקפה הזאת:

Page A: If (CONDITION)

Redirect(Page B)

הווקטור של ההתקפה הוא די פשוט:

1. Create IFRAME with src=Page B
2. Remember the current value of history.length
3. Change src of IFRAME to Page A
4. If the value of history.length is the same– then the CONDITION is TRUE

ננסה להבין כיצד ההתקפה עובדת: כצעד ראשון, הפורץ אמור לאתר אפליקציה שמכילה תבנית מותנית של redirect. נניח שהוא מצא כזאת אפליקציה שמעבירה מדף א' לדף ב' במידה ותנאי מסוים מתקיים. כעת, הפורץ יכול לפתח דף משלו שיכיל בתוך IFRAME דף ב', מהאפליקציה החשופה אותה הוא מעוניין לתקוף. נשים לב שבשלב הזה האיבר האחרון במערך רכיב ההיסטוריה הוא דף ב'. הפורץ מודד את אורך רכיב ההיסטוריה אחרי שדף ב' נפתח בתוך IFRAME ומייד אחר כך פותח דף א' מתוך אותו IFRAME. במידה ודף א' יעביר לדף ב' ודף ב' כבר למעלה במערך רכיב ההיסטוריה, אז לא יתווסף להיסטוריה עוד איבר משום שההיסטוריה מנוהלת כך שאין שני איברים זהים ברצף. אם כך, אורך ההיסטוריה יישאר כמו שהוא היה לפני פתיחת דף א', מה שמעיד על כך שהתנאי של redirect מתקיים.

במידה ודף א' לא יעביר לדף ב', אז אורך רכיב ההיסטוריה יגדל באחד- וזה מה יעיד על כך שהתנאי לא מתקיים. אם כן, התוקף יכול, מתוך הדף שלו, לקבל אינדיקציה על ערך של התנאי באפליקציה שרצה ממקום אחר- זליגה של מידע בין דומיינים שונים. כאן נציג שתי דוגמאות כיצד זליגה כזאת יכולה להביא לפגיעה בפרטיות וסודיות של משתמשים.

אחד הדברים שחשובים לפורץ הוא לדעת האם המשתמשים אותם הוא רוצה לתקוף מחוברים ברגע נתון לאפליקציה מסוימת ועברו כבר את התהליך של זיהוי משתמש. למשל, אם משתמש הזדהה לאפליקציה בנקאית, וידוע שהאפליקציה הזו חשופה ל-CSRF, אז התוקף יכול לשלוח payload של CSRF ולהיות בטוח שההתקפה שלו תצליח. השיטות הקיימות לגילוי סטאטוס של זיהוי המשתמש בתוך האפליקציה מתבססות על פתיחה של תמונה מ-"האזור המוגן של האפליקציה" ודגימה של אירוע onerror. שיטה זו לא תמיד עובדת כי לא תמיד קיימות תמונות באזור המוגן שנפתחות אך ורק כאשר המשתמש מזוהה.

אנו מציעים שיטה אחרת של גילוי סטאטוס של זיהוי משתמש בעזרת XSHM. האפליקציה תהיה חשופה להתקפה כזאת במידה והיא מממשת תבנית הבאה בקוד:

```
If (!isAuthenticated())
      Response.Redirect("Login.aspx")
```

או בקונפיגורציה:

```
      <authentication mode="Forms">
            <forms loginUrl="Login.aspx”/>
</authentication>http://he.wikipedia.org/wiki/Advanced_Encryption_Stand
ard
```

כעת התוקף יכול לבצע את הצעדים הבאים:

1. Create IFRAME with src='Login.aspx'
2. Remember the current value of history.length
3. Change src of IFRAME to 'Protected.aspx'
4. If the value of history.length remains the same– then a user is not authenticated

אם כן, בהנחה כי האפליקציה מגדירה אזור מוגן וגישות לאזור הזה ללא זיהוי משתמש מעבירות לדף של זיהוי משתמש, התוקף יכול בקלות לדעת מהו סטאטוס הזהוי של המשתמש בתוך האפליקציה. משמעות הדבר שמשתמש יכול לגלוש לאתר כלשהו שמציג לדוגמא מכונית יפה, אך מאחורי הקלעים האתר יכול לקבל אינדיקציה אמינה מה הן האפליקציות שאתם כרגע מזוהים בהן. דרך אגב, אפליקציות רבות וביניהן Twitter ו-Facebook חשופות לסוג התקפה זאת. אתם יכולים לראות הדגמה עבור Facebook בלינק הזה.

דבר נוסף שאפשרי לעשות בעזרת מתקפת ה-XSHM, הוא לקבל מידע רגיש מתוך אפליקציה חשופה. תתארו מערכת לניהול משאבי אנוש שמאפשרת לחפש עובדים לפי פרמטרים שונים כמו שם העובד, גיל ומשכורות. כאשר מבצעים חיפוש, נשלח הלינק הבא:

http://Intranet/SearchEmployee.aspx?name=Jon&SalaryFrom=3000&SalaryTo=3500

במידה ואין אף עובד שעונה לקריטריונים של החיפוש, אז האפליקציה מעבירה לדף של Not found (או לחיפוש מתקדם). בנסיבות אלה תוקף יכול לבצע את הצעדים הבאים:

1. Create IFRAME with src='NotFound.aspx'
2. Remember the current value of history.length
3. Change src of IFRAME to 'SearchEmployee.aspx?name=Jon&SalaryFrom=3000&SalaryTo=3500'
4. If the value of history.length remains the same– then your search has no results

כך שלמעשה, התוקף יכול לדעת כמה מרוויח כל עובד כאשר אין לתוקף גישה ישירה לאפליקציה והוא נעזר אך ורק בטכניקה של XSHM. ישנם עוד תסריטים רבים של XSHM שמאפשרים לדלות מידע רגיש מאפליקציה חשופה לתוך אפליקציה של תוקף, על כך אפשר לקרוא במאמר הזה.

יצרני הדפדפנים, כעקרון, יכולים למנוע את ההתקפה של XSHM על ידי הכנסת מגבלות גישה לאובייקט ההיסטוריה. למשל, אם משתמש גלש לדפים הבאים:

- http://Site1.com
- http://Site2.com
- http://Site3.com/Home.aspx
- http://Site3.com/Report.aspx

וכעת הדף Report.aspx מתוך Site3 ניגש לאובייקט ההיסטוריה, אז אורך רכיש ההיסטוריה המוחזר אמור להיות 2 עבור Site3 ולא 4, כך אפשר לעשות הפרדה בין הדומיינים השונים גם בתוך האובייקט של ההיסטוריה ולא להפר מנגנון SOP. עם זאת, יצרני דפדפנים אינם ממהרים לבצע את השינוי הזה כי הוא יפגע בפונקציונאליות של אפליקציות קיימות שמשתמשות באובייקט של היסטוריה. לכן, הדרך למנוע את ההתקפה בשלב זה היא דרך אפליקטיבית – קודם כל המפתחים חייבים להיות מודעים להתקפה.

מאוד חשוב לדעת לזהות את תבניות הקוד שמובילות להתקפה הזאת. הכלי האוטומטי היחיד לניתוח קוד שקיים בשוק ומסוגל לזהות תבניות שחשופות ל- XSHM הוא הכלי של חברה ישראלית בשם "צ'קמרקס" וניתן להוריד גרסת דמו שסורקת קוד ומתריעה על XSHM.

על מנת להתגונן כנגד התקפת XSHM יש לשלב פרמטר עם ערך שרירותי בתוך URL. למשל, על מנת למנוע גילוי סטאטוס של זיהוי משתמש, אפשר להיעזר בקוד הזה:

```
If ( !isAuthenticated)
      Redirect('Login.aspx?r=' + Random())
```

חשוב להבין שלא כמו עבור Anti-CSRF, אין לבדוק את הפרמטר השרירותי באפליקציה – המטרה של הפרמטר היא להוסיף אנטרופיה ל-URL ובכך למנוע מהפורצים לבצע את XSHM. החידוש של התקפת XSHM מול הפרצות האחרות הידועות במנגנון ה-SOP ובפרט, מול פרצת CSS history hacking הוא בכך שהתקפת XSHM משתמשת רכיב ההיסטוריה הזמנית ולא ברכיש ההיסטוריה הקבועה. ההתקפה של CSS לא מסוגלת לגלות סטאטוס של זיהוי משתמש ולא לדלות מידע על משכורות העובדים משתי סיבות עיקריות:

1. היסטוריה הקבועה נשארת בין ה-sessions השונים, ולכן הלינק שמתוך האזור המוגן יישאר visited להרבה זמן גם אם המשתמש מזמן כבר לא בתוך האפליקציה, אבל ב-XSHM התוקף כן מקבל אינדיקציה לגבי ה-session הנוכחי וזה מאוד חשוב להתקפות על רכיב הלקוח כמו CSRF ו-Clickjacking.

2. אם פותחים קישור מתוך IFRAME, אז הצבע שלו לא ישתנה ל-visited, לכן אי אפשר לדלות מידע על משכורות עובדים בעזרת CSS, אלא רק בעזרת XSHM.

שורה תחתונה: התקפה של XSHM היא עוד התקפה על מנגנון SOP ויכולה להביא לזליגה של מידע מאתר חשוף לאתר של פורץ. חשוב להכיר את ההתקפה הזאת וגם לדעת להתגונן כנגדה.

על המחבר:

אלכס רויכמן הוא הארכיטקט הראשי ומנהל מעבדת המחקר של צ'קמרקס (www.Checkmarx.com). החברה מתעסקת בפיתוח כלים אוטומטיים לניתוח סטאטי של קוד המאתרים בעיות באבטחת המידע. החברה נוסדה ב-2006 ואלכס היה בין הראשוניים שעבד על מחקר ופיתוח של אלגוריתמים שמזהים בעיות אבטחה בקוד. לאלכס מעל 10 שנות נסיון בבנית מערכות מתוחכמות ומורכבות. הוא גילה פרצות אבטחה רבות ובינהן, למשל, פרצת האבטחה בשם ReDoS שתפסה תשומת לב רבה בשנה אחרונה. אלכס פרסם מאמרים במגזינים מובילים כמו ACM ו-Springlink. אלכס רויכמן סיים בהצטיינות את לימודי התואר השני במדעי המחשב עם התמחות באבטחת אפליקציות ובסיס נתונים, נושא התזה שלו הוא "מניעה וגילוי חדירות לבסיסי נתונים ווביים". אלכס נגיש לשאלותיכם ב: Alexr@Checkmarx.com

ניתוח תולעת ה-Conficker

מאת הרצל לוי ואפיק קסטיאל (cp77fk4r)

(התמונה נלקחה מ- slate.com)

תולעת הקונפיקר (Conficker), הידועה גם בשמות Downup, Downadup ו-Kido, היא תולעת שהתגלתה לראשונה בנובמבר 2008 והתפשטה מאז במידה רחבה כל כך, שרבים הכתירו אותה בתור התולעת המסוכנת ביותר עד היום. נכון להיום, מעריכים שיותר מ-12 מליון מחשבים נדבקו בתולעת. מאז החשיפה הראשונה של התולעת נמצאו 5 וריאציות שלה (E,D,C,B,A). התולעת פוגעת רק במחשבים שעליהם מותקנות מערכות ההפעלה Windows למיניהן, כולל את Windows 7.

התולעת לא פסחה אף על אתרי ממשלה, רשתות צבאיות, בתי חולים, משטרות ברחבי העולם ואפילו רשת צהלנט נפגעה. התולעת הצליחה לעצבן את מיקרוסופט עד כדי כך שגרמה להם להציע סכום של $250,000 למי שיצליח לאתר את יוצריה (דרך אגב, הצעה זו עדיין בתוקף למי שתוהה). התולעת קונפיקר משתמשת בהרבה שיטות מתקדמות ומעניינות להדבקה, הפצה ופגיעה במשתמשי Windows. במאמר זה נתאר וננתח חלק מהן.

אז מה הופך את התולעת הזאת לכל כך נפוצה ומסוכנת?

חדירה

1. קונפיקר מדביקה מחשבים על ידי ניצול חולשה (MS08-067) ב-Windows Server service (SVCHOST.EXE) של מערכת ההפעלה, שמאפשרת הפעלת פקודות מרחוק (RPC) כאשר שיתוף קבצים מאופשר. כותבי קונפיקר השתמשו במודול של Metasploit לאקספלויט שנקרא ms08_067_netapi שנכתב ע"י HD Moore ושמנצל חולשה זו.
2. גילוי כל המשתמשים (למשל על ידי שימוש בפקודה: net user) והתקפת Brute Force עם שימוש במילון סיסמאות על כל משתמש (בעצם התקפה על סיסמאות חלשות ומוכרות).
3. דרך התקנים ניידים כמו דיסק און קי – על ידי יצירת קבצי autorun.inf שיגרמו להפעלה אוטומטית של הזרקת קובץ ה-DLL המכיל את הקוד הזדוני.

הדבקה

לאחר שלב החדירה, הקונפיקר מנסה להעתיק את עצמה לתיקיית המערכת (%SysDir%) של מחשבים אחרים על ידי אפשרות שיתוף הקבצים. הניסיון הראשון הוא לחדור דרך השיתוף של מנהל המערכת (ADMIN$). אם ניסיון ההעתקה לשיתוף זה נכשל, התולעת תשתמש בהתקפת ה-Brute Force שתוארה לפני כן ואז שוב תנסה להעתיק את עצמה לתיקיית המערכת או לתיקיות של תוכנות שמגיעות עם מערכת ההפעלה כמו IE או Windows Movie Maker. הקונפיקר מעתיקה את עצמה בצורת קובץ DLL חבוי בעל שם רנדומאלי (אי שמירה על תבנית אחידה מוסיפה על הקושי באיתור התולעת) ומכיוון שקובץ DLL הוא לא קובץ הרצה, אופן ההטמעה שלו היא על ידי הזרקה של ה-DLL לתהליך אחר שכבר רץ במערכת (DLL Injection). כדי שקונפיקר תשרוד חסימות של Firewall והרשאות היא מזריקה עצמה לתהליכי מערכת כמו rundll32.exe. ההזרקה נעשית בצורה הבאה:

```
seg000:00B3CCB4          pop     ecx
seg000:00B3CCB5          push    [ebp+ProcessID]
seg000:00B3CCB8          mov     esi, eax
seg000:00B3CCBA          push    edi
seg000:00B3CCBB          push    2Ah
seg000:00B3CCBD          inc     esi
seg000:00B3CCBE          call    ds:OpenProcess          פתיחת התהליך הנבחר
seg000:00B3CCC4          cmp     eax, edi
seg000:00B3CCC6          mov     [ebp+hProcess], eax     שמירת המצביע שלו
seg000:00B3CCC9          jz      loc_B3CE34
seg000:00B3CCCF          push    40h
seg000:00B3CCD1          push    3000h
seg000:00B3CCD6          lea     ecx, [esi+20h]
seg000:00B3CCD9          push    ecx
seg000:00B3CCDA          push    edi
seg000:00B3CCDB          push    eax
seg000:00B3CCDC          call    ds:VirtualAllocEx       הקצאת הזכרון בתהליך הנבחר
seg000:00B3CCE2          cmp     eax, edi
seg000:00B3CCE4          mov     [ebp+AllocatedBuffer], eax
seg000:00B3CCE7          jz      close_quit
seg000:00B3CCED          mov     edi, ds:GetModuleHandleA
seg000:00B3CCF3          push    ebx
seg000:00B3CCF4          push    offset aLoadlibrarya ; "LoadLibraryA"
seg000:00B3CCF9          push    offset aKernel32_dll ; "kernel32.dll"
seg000:00B3CCFE          call    edi ; GetModuleHandleA
seg000:00B3CD00          mov     ebx, ds:GetProcAddress
seg000:00B3CD06          push    eax
seg000:00B3CD07          call    ebx ; GetProcAddress
seg000:00B3CD09          mov     [ebp+lpLoadLibraryA], eax
seg000:00B3CD0C          lea     eax, [ebp+NumberOfBytesWritten]
seg000:00B3CD0F          push    eax
seg000:00B3CD10          inc     esi
seg000:00B3CD11          push    esi
seg000:00B3CD12          push    [ebp+ConfickerDllFilename]   העתקה של הנתיב והשם של ה-
seg000:00B3CD15          push    [ebp+AllocatedBuffer]        Conficker DLL
seg000:00B3CD18          push    [ebp+hProcess]
seg000:00B3CD1B          call    ds:WriteProcessMemory
seg000:00B3CD21          test    eax, eax
seg000:00B3CD23          jz      loc_B3CE19
seg000:00B3CD29          lea     eax, [ebp+var_20]
seg000:00B3CD2C          push    eax                          העתקה של תוכן ה-
seg000:00B3CD2D          xor     esi, esi                     DLL
seg000:00B3CD2F          push    esi                          על ידי הפונקציה
seg000:00B3CD30          push    [ebp+AllocatedBuffer]        LoadLibrary
seg000:00B3CD33          push    [ebp+lpLoadLibraryA]         והרצה על ידי הפונקציה
seg000:00B3CD36          push    esi                          CreateRemoteThread
seg000:00B3CD37          push    esi
seg000:00B3CD38          push    [ebp+hProcess]
seg000:00B3CD3B          call    ds:CreateRemoteThread
```

(התמונה המקורית נלקחה מ- "Threat Experts Blog")

הפונקציות המתוארות בפיסקה זו הן פונקציות API של מערכת ההפעלה. בתור התחלה התולעת תפתח את תהליך היעד בעזרת OpenProcess, תקצה זיכרון במרחב הכתובות שלו בעזרת VirtualAllocEX ותרשום לתוכו את הנתיב המלא ל-Conficker DLL. לאחר מכן היא תשיג את הכתובת של הפונקציה LoadLibraryA שבעזרתה נתן לטעון DLL לתוך מרחב כתובות של תהליך מסוים על ידי שימוש ב-kernel32.dll (קובץ שמגיע עם מערכת ההפעלה Windows) ותטען את תוכן קובץ ה-DLL על ידי השימוש בפונקציה זו. בעזרת CreateRemoteThread נוצר Theard שרץ במרחב הכתובות של תהליך המערכת ומריץ את ה-Conficker DLL.

הישרדות

כדי לשרוד אתחול, גילוי והסרה, הקונפיקר חייבת לבצע שינויים במערכת ההפעלה:

1. כדי לשרוד אתחול, היא רושמת את עצמה כשירות של המערכת (service) המופעל אוטומטית בעליית המערכת, על ידי הוספת ערכים בעורך הרישום תחת המפתח הבא:

```
HKLM\SYSTEM\CurrentControlSet\Services
```

2. ביטול ה-Firewall של Windows.
3. על מנת לשרוד עדכוני אבטחה של Windows התולעת מבטלת עדכונים אוטומטיים, את Windows Defender ואת האפשרות להרצת המערכת במצב בטוח.
4. דרך הישרדות נוספת היא מחיקה של נקודות השחזור (restore points) של המערכת.
5. יצירת משימות מתוזמנות להזרקת ה-DLL מחדש.
6. חסימת תהליכים שיכולים לגרום להסרתה כגון תוכנות אנטי-וירוס.
7. חסימת גישה לאתרי חברות אנטי-וירוס.

הפצה

הקונפיקר היא תולעת, כלומר, היא יודעת להתרבות ולהפיץ את עצמה. לאחר שתולעת חודרת למערכת ומבצעת את כל הפעולות בכדי להבטיח את ההישרדות שלה, היא תפיץ את עצמה בדרכים הבאות:

1. סריקת כל הרשת הפנימית לחיפוש מחשבים עם פורטים פתוחים ושימוש בטכניקות חדירה והדבקה שתוארו קודם לכן.
מחשב שנגוע בקונפיקר משמש גם כלקוח וגם כשרת. כדי לאפשר זאת, הקונפיקר פותחת 4 פורטים לשימוש השרת שהם: 2 פורטים של UDP ו-2 של TCP שלהם היא מאזינה ופורט נוסף שישמש כלקוח.

2. לאחר שלב החדירה, הקונפיקר מזהה חיבורים של התקנים ניידים וכוננים ממופים ואז מעתיקה עצמה לשם כפי שתואר בשלב ההדבקה.

שיטות מתקדמות בהן משתמשת התולעת

הפצת ה-Payload

כפי שתואר לפני כן, קונפיקר פותחת 4 פורטים, מאזינה להם ובין היתר מנתחת את תעבורת הנתונים בפורטים אלו. כאשר קונפיקר מפיצה עצמה ברשת הפנימית שבה היא נמצאת, היא משתמשת בתקשורת P2P וחודרת לקורבן באמצעות שליחת האקספלויט שמנצל את החולשה שתוארה קודם לכן וגורמת לכך שלתוקף תהיה אפשרות להפעלת פרוצדורות מרחוק (RPC) על הקורבן.

כאשר מחשב נגוע אחד מנסה לחדור למחשב נגוע אחר (זאת אומרת שהוא שולח לו את האקספלויט), או שמחשב נגוע מזהה ששלחו לו את אותו אקספלויט שבו גם הוא השתמש, הוא מבין שאותו מחשב שניסה לחדור אליו גם הוא נגוע ולכן באותו הרגע הוא שולח למחשב שניסה לחדור אליו את ה-Payload שיש אצלו, כך שה-Payload מופץ בין כל מחשבי הרשת הנגועים.

Anti-Analyzing

על מנת להגיע למצב בו חברות האנטי-וירוסים יוכלו לכלול כלים להסרת התולעת מהמחשב בתוכנות שלהן, על חוקרי הוירוסים להצליח לנתח את התולעת. ניתוח התולעת יכול לכלול אלמנטים ממספר תחומים, אם זה ביצוע Debugging על התהליך שהתולעת הוזרקה אליו או על קובץ ה-Deploy שלה, אם זה ביצוע Sniffing וניתוח תעבורת המידע היוצאת והנכנסת מעמדה הנגועה באחת מגרסאות התולעת או אם זה ניתוח השינויים בסביבת התולעת.

ברוב המקרים שלב הניתוח מבוצע במעבדות המכונות "Security-lab" - רשתות מחשבים המדמות את רשת האינטרנט הנמצאות תחת מעקב של אין ספור כלים, החל מכלים המבצעים Sniffing לכלל תעבורת הרשת וכלה בכלים המבצעים האזנה וניתוח השינויים בתהליכים פנימיים במחשב בזמן ריצת התולעת.

יוצרי התולעת יישמו מספר מנגנונים בכדי להקשות על חוקרי הוירוסים מלבצע את אותן הפעולות שצויינו כאן. שלב ניתוח התולעת הוא מאוד קריטי, חומרת וכמות הנזק שהתולעת תעשה ברשת האינטרנט הן כמעט תמיד נגזרת של המהירות בה ניתוח התולעת יעשה וברמת הדיוק שאליה יגיעו חוקרי התולעת.

המנגנונים העיקריים בהם השתמש יוצר הווירוס להקשות על נסיונות המחקר הם:

- שינוי הרשאות כתיבה/צפייה של ערכים שונים בעורך הרישום של מערכת ההפעלה.
- שינוי הרשאות קריאה ומחיקה של הקבצים הנמצאים בשימוש של התולעת.
- ביצוע Hooking לפונקציות מערכת ותהליכים שונים במערכת ההפעלה.
- ביצוע Hooking ל-Dll הספציפי שבו קיימת החולשה אותה מנצלת התולעת.
- ביטול שירותי אבטחה עדכון גיבוי ושחזור של מערכת ההפעלה.

- שימוש במידה רבה של Obfuscation לקוד התולעת.
- זיהוי האם התולעת רצה בסביבה וירטואלית או לא ויישום שני מנגנוני ריצה שונים בהתאם.
- שימוש בערוץ תקשורת מוצפן תחת אלגוריתמי הצפנה "כבדים" (כגון ה-RSA) ושימוש במפתחות הצפנה גבוהים.
- ניטור התהליכים הרצים במערכת ההפעלה ובדיקה האם הינם "כלי ניתוח".
- קינפוג מחדש של כלים כגון תוכנת ה-Firewall של מערכת ההפעלה.

שינוי ערכים קריטיים בעורך הרישום

כחלק מהשינויים שהתולעת הייתה מבצעת בעורך הרישום של מערכת ההפעלה היא הייתה מוחקת/משנה את הערכים תחת המפתחות הבאים:

- מחיקת המפתח:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SafeBoot
```

בכדי למנוע מהמשתמש לבצע אתחול מחדש של המחשב במצב "Safemode". (השימוש הנצפה הראשון בטכניקה הזאת היה של הווירוס "W32/Bagle.fb@MM" בשנת 2006)

- שינוי הערכים במפתח:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\explorer\A
dvanced\Folder\Hidden\
```

(זהו הערך האחראי על אפשרות ה-show hidden files and folders) כך שגם אם המשתמש יבחר להציג את הקבצים המוסתרים (ATTRIB +H) במערכת- המערכת לא תציג אותם.

- שינוי הערך: TcpNumConnections

תחת המפתח:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\
```

ל-00FFFFFE (הערך המקסימאלי) בכדי לאפשר את הכמות המירבית של חיבורי TCP/IP בתחנה.

- הוספת ערכים למפתח:

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

בכדי להיכנס לרשימת ה-Services של המערכת ולהטען מיד עם הפעלתה

- הוספת ערכים למפתח:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\WindowsNT\CurrentVersion\Svchost
```

בכדי להטען בהפעלת המערכת דרך תהליך ה-Svchost.exe. (ריצה של מספר תהליכי ה-Svchost.exe במערכת ההפעלה היא דרך נורמלית) התהליך אחראי על "אירוח" של מספר תהליכים במערכת ההפעלה והוא טוען אותם בזמן טעינת מערכת ההפעלה.

בנוסף, ע"י השימוש בפונקציה "RegSetKeySecurity" (תחת ADVAPI32.dll) התולעת הייתה מורידה את ההרשאות לכלל המשתמשים משינוי ערכים ספציפיים בעורך הרישום (ערכים כגון רשימת ה-Services) ומותירה הרשאה מיוחדת רק למשתמש System – כך שגם למשתמשים מקבוצת "Administrators" לא הייתה גישה לשנות/לצפות בהם.

גרסאות שונות של תולעת ה-Conficker מבצעות שינויים שונים בעורך הרישום, אלה השינויים שלרוב יתרחשו כחלק ממנגנון פריסת התולעת.

Hooking לפונקציות מערכת/ תהליכי מערכת

כפי שנכתב בעמודים הקודמים, כחלק מהפעולות שהתולעת עושה היא גם מבצעת Hooking לכל מיני פונקציות מערכת ותהליכי מערכת שונים, לדוגמא:

- פעולת ה-Hook המרכזית שהתולעת מבצעת היא לפונקציה NetpwPathCanonicalize

הפונקציה NetpwPathCanonicalize מאוחסנת תחת הקובץ netapi32.dll, והיא ה-API האחראי על ה-RPC שבו קיימת החולשה שאותה מנצלת התולעת (MS08-067).

התולעת מחליפה את חמשת הבייטים הראשונים של הפונקציה בפקודת JMP המעבירה את המעבד לקוד האחראי על הרצת התולעת.

תחילת הקוד של NetpwPathCanonicalize לאחר עריכת התולעת

```
5B86A259 E9 A0B028A6   JMP 01AF52FE
5B86A25E 53            PUSH EBX
5B86A25F 8B5D 14       MOV EBX,DWORD PTR SS:[EBP+14]
5B86A262 56            PUSH ESI
5B86A263 57            PUSH EDI
5B86A264 33FF          XOR EDI,EDI
5B86A266 3BDF          CMP EBX,EDI
5B86A268 0F85 8EDE0000 JNZ NETAPI32.5B8780FC
```

תחילת הקוד המקורי של NetpwPathCanonicalize

```
5B86A259   8BFF          MOV EDI,EDI
5B86A25B   55            PUSH EBP
5B86A25C   8BEC          MOV EBP,ESP
5B86A25E   53            PUSH EBX
5B86A25F   8B5D 14       MOV EBX,DWORD PTR SS:[EBP+14]
5B86A262   56            PUSH ESI
5B86A263   57            PUSH EDI
5B86A264   33FF          XOR EDI,EDI
5B86A266   3BDF          CMP EBX,EDI
5B86A268   0F85 8EDE0000 JNZ NETAPI32.5B8780FC
```

(התמונה המקורית נלקחה מהמאמר המצויין "Know Your Enemy: Containing Conficker" של הפרויקט Honeynet,
נכתב במקור ע"י Felix Leder ו-Tillmann Werner)

בנוסף, התולעת מבצעת פעולות Hooking לארבעה (בגרסאות מתקדמות אף יותר) קבצי DLL נוספים:

- פעולת Hooking לפונקציות הבאות:

- DnsQuery_A
- DnsQuery_UTF8
- DnsQuery_W
- Query_Main

המאוחסנות תחת הקובץ dnsapi.dll. מטרת הפעולה הנ"ל היא ברובה בכדי למנוע מהתחנה המקומית לגשת לשרתי חברות האנטי-וירוסים המשמשים את תוכנות האנטי-וירוס לעדכן את מערך החתימות הקיימות באפליקציות שלהן וכך למנוע מהם לזהותה.

- ועוד פעולות Hooking לפונקציות "קלאסיות", כגון:

- NtQueryInformationProcess
- InetnetGetConnectedState
- Sendto

המאוחסנות בקבצים ntdll.dll, wininet.dll ו- ws2_32.dll (בהתאמה) אשר משמשות את מערכת ההפעלה להשיג מידע על תהליכים ואירועים במערכת:

- NtQueryInformationProcess - אחראית על השגת מידע אודות תהליכים במערכת. (שם, נתוני PEB וכו')
- InetnetGetConnectedState - מאפשרת בדיקה האם בוצעה "Internet dialup".
- Sendto - אחראית על שליחת מידע ליעד ספציפי מהתחנה.

השימושים המשמעותים ביותר ב-Obfuscation נצפו בגרסתה השנייה של התולעת ומעליה, גרסתה השנייה התגלתה ע"י החוקרים של חברת Sophos באמצעות Honeypots. אחד הסעיפים המעניינים שעלה בדו"ח של החוקרים מאותה החברה הוא שהתולעת משתמשת במנגנון המכונה "Red Pill" (כן, בהשראת הסצנה ממטריקס) – מנגנון פשוט להפליא אשר נועד לבדוק האם התהליך רץ תחת מכונה וירטואלית. הנתון המעניין הוא שבמידה והתולעת "הבינה" שהיא מורצת בתוך סביבה וירטואלית – כנראה במסגרת ניסיונות לנתח אותה – היא הייתה מתנהגת באופן שונה ממה שהיא הייתה מתנהגת בסביבה טבעית. רעיון ה-Red Pill הוא אלגנטי ופשוט, במקור הוא הוצג בשנת 2004 ע"י Joanna Rutkowska, במאמר:

"Red Pill... or how to detect VMM using (almost) one CPU instruction"

ופותח לאחר מכן ע"י שני חברי קהילת Offensive Computing בשם Danny Quist ו-Val Smith, במאמר:

"Detecting the Presence of Virtual Machines Using the Local Data Table"

על ידי השוואה פשוטה בין ערכים שונים (הידועים מראש) של ה-LDT, התולעת יכלה לדעת היכן היא רצה. באופן כזה אופי התולעת משתנה בזמן ריצתה, במידה וחוקר יפספס את המנגנון הנ"ל- הוא יוכל לבזבז זמן יקר בניתוח הקוד שלא מבצע שום דבר. לא מדובר פה על Code Obfuscation רגיל, אך עדיין מדובר במהלך אשר יקשה על מנתח התולעת. בנוסף, ביצעו יוצרי התולעת פעולות Obfuscation לקריאות ה-API שבהן השתמשה התולעת בכדי להסוות קריאות אלו ולהקשות על חוקרי הוירוסים למצוא את כתובותיהן המקוריות.

מנגנוני הצפנה ומנגנוני אימות

תולעת הקונפיקר מבצעת שימוש בשליחת וקבלת נתונים על גבי רשת האינטרנט למספר מטרות, בין אם מדובר בעדכון Payload התקיפה של התולעת, בפקודות לביצוע (כמו התקפת שרת מסוים וכו') או בסריקת הרשת למציאת קורבנות נוספים.

כפי שניתן היה להבין עד כה- תולעת הקונפיקר פותחת על העמדה הנגועה ערוצי תקשורת, אשר דרכם היא בודקת כל פרק זמן מסוים האם יוצרי התולעת שחררו עדכונים, כמו למשל- עדכון וקטור התקיפה שבה התולעת תשתמש בכדי לתקוף מחשבים חדשים, וכך, גם במידה ומיקרוסופט ישחררו טלאי אבטחה לאותו RPC חשוף- על יוצרי התולעת לשלוח עדכון חדש שכולל בתוכו וקטור תקיפה לחולשה חדשה במערכת, וכך להדביק גם עמדות מעודכנות.

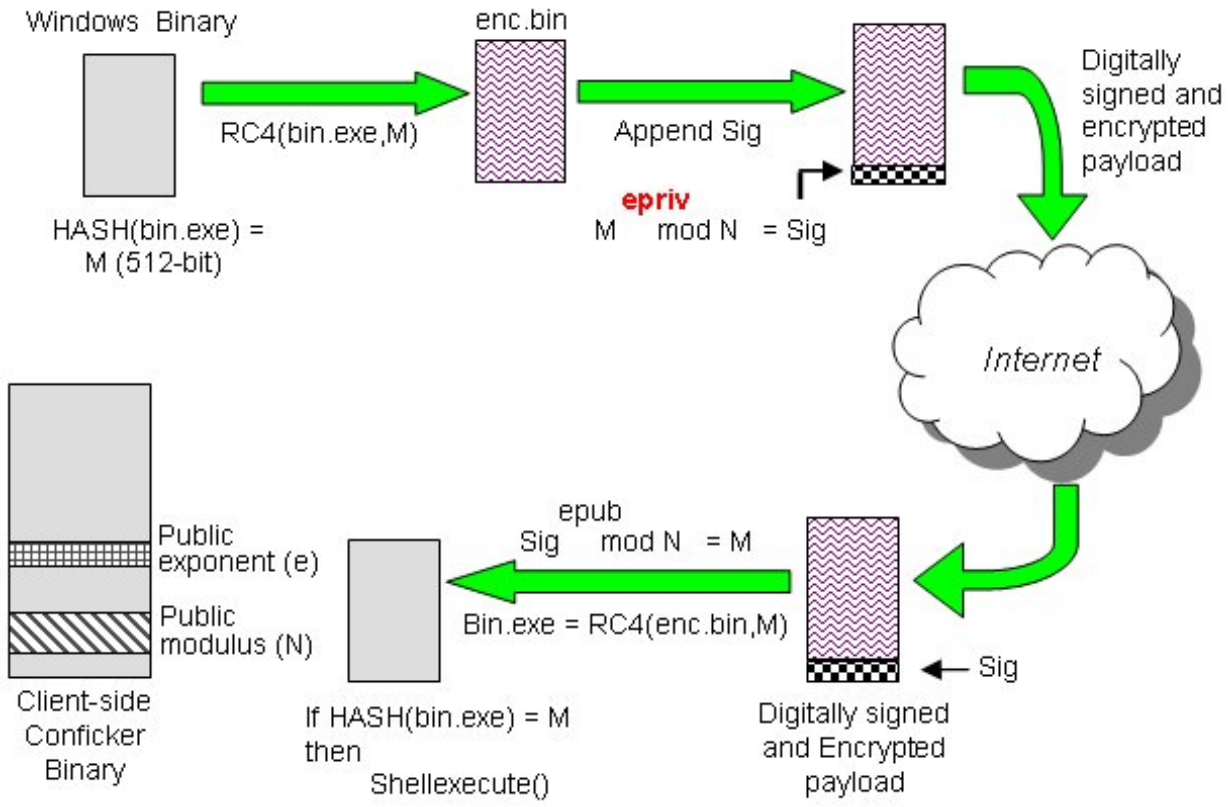
יוצרי הקונפיקר מימשו מנגנון אימות Hashing המבוסס על האלגוריתם MD6 (גם כיום, יותר משנה לאחר התקפת התולעת, אלגוריתם ה-Hashing ממשפחת MD הנפוץ ביותר הוא: 5 – כך שהדבר נחשב אז לפריצת דרך). המידע היה עובר תחת אלגוריתם ה-Stream Cipher המוכר-RC4 בשימוש של מפתח

הצפנה בגודל 512-bit ומימוש של אלגוריתם ה-RSA בכדי לייצר חתימה דיגיטלית בעזרת מפתח ציבורי המשותף לכלל תולעי הקונפיקר. החתימה הדיגיטלית הייתה נשלחת ביחד עם קובץ העדכון המוצפן.

כאשר תולעת הקונפיקר הייתה רואה כי פורסם עדכון חדש- היא הייתה נגשת לאחד מהשרתים שהוקצו לפעולה זו, מורידה אותו למחשב ומוודאת כי המידע לא שונה בדרך ושום גורם ביניים לא החליף או ערך את תוכנו. במידה והקובץ נבדק ונמצא כי שום גורם לא ערך את המידע בדרך והקובץ אכן אותנטי - התולעת הייתה מריצה אותו ומעדכנת את עצמה בעזרתו.

הסבר ויזואלי של כלל התהליך פורסם בדיוק לפני שנה, בפברואר 2009, במסגרת מאמר של הארגון SRI INTERNATIONAL המפעיל מערכות / רשתות Honeypots ומפרסם Infection Logs יומיים בנוגע להתקפות/ניסיונות להתקפות אשר בוצעו עליהם:

"AN ANALYSIS OF CONFICKER'S LOGIC AND RENDEZVOUS POINTS"



בעזרת מימוש של מנגנוני אימות אלו הצליחו יוצרי התולעת למנוע מחוקרי הוירוסים להרוס אותה על ידי הזרקת "עדכונים" פיקטיביים שנועדו לעצור אותה. עקב שימוש בדרכים אלו ואחרות, ניסו יוצרי תולעת הקונפיקר, בין היתר, למנוע מחוקרי הוירוסים לעצור את התולעת.

אז מה עושים עם 12 מליון בוטים?

התשובה היא כמובן- הרבה מאוד כסף.
המטרה העיקרית של יוצרי קונפיקר היא כנראה הקמת רשת בוטים שעליהם יוכלו לשלוט בעיקר לצרכים כספיים כמו הפצת ספאם, סחיטות להתקפות DDOS (מפחיד לחשוב על DDOS עם כמות כזאת של בוטים), סחיטות למחיקות קבצים על שרתים וכן הלאה.

פשעי רשת או Cyber Crime הוא מושג המתאר את כל סוגי הפשעים המבוצעים באמצעות או דרך המחשב. בעשור האחרון הפך סוג פשיעה זה למקצועי ומאורגן באופן מדאיג. ארגוני פרסום ספאם או ארגונים שעוסקים בפשעים אלו (נקראים גם Cyber-Mafia) מוכנים לשלם הרבה מאוד כסף תמורת רשתות בוטים, כתיבת תולעים או אפילו מידע מסוים הרלוונטי לצרכיהם. קחו לדוגמה את ה-RBN (Russian Business Network), אחד מארגוני ה-Cyber-mafia הגדולים בעולם. ההתמחות שלהם היא גניבת זהויות ובין היתר מעריכים כי הם אלו ששולטים ברשת הבוטים העצומה Storm. ארגון זה עודד את הוגי התולעת Storm ליצור את אותה תולעת שהכניסה להם הרבה מאוד כסף. ולנו? הרבה מאוד נזק.

אנו מצרפים מחירון לדוגמה ליחידה:

Rank	Item	Percentage	Range of prices
1	Credit cards	22	$0.50 - $5
2	Bank accounts	21	$30 - $400
3	email passwords	8	$1 - $350
4	Mailers	8	$8 - $10
5	Email addresses	8	$2/MB - $4/MB
6	Proxies	6	$0.50 - $3
7	Full identity	6	$10 - $150
8	Scams	6	$10/week
9	Social Security Numbers	3	$5 - $7
10	Compromised Unix shells	2	$2 - $10

Breakdown of goods available for sale on underground economy servers.
(Source: Symantec Corporation, 2007)

(נלקח מ- newcriminologist.com)

לסיכום

אי אפשר שלא להתרשם מהדרך המתוחכמת בה פועלת תולעת הקונפיקר. כותבי קונפיקר השקיעו המון ידע ותכננו תולעת ברמה מאוד גבוהה, השתמשו בשיטות מתקדמות, מימשו אלגוריתמים חדשניים כמו ה-MD6 של רון ריווסט (למי שלא מכיר, ה-R ב-RSA מייצג את רון ריווסט) ואפילו דאגו להתעדכן ולשפר את מנגנוני התולעת שאותם הצליחו לעצור.

התולעת קונפיקר הופעלה בראשון באפריל 2008 והייתה אמורה לטרוף את דרכה ברשת. אבל לא קיימת הוכחה שיוצריה הפעילו אותה בצורה כלשהי. אולי עצם העובדה שהיא הופעלה ב-1 באפריל (April Fools' day) נותנת את התשובה שזאת סתם הייתה מתיחה של כמה גאונים, למרות שקשה להאמין כי השקעה בתולעת מתוחכמת כל כך נעשתה לשם מתיחה.

Rootkits- דרכי פעולה וטכניקות בשימוש

מאת Zerith (אורי)

במהלך השנים האחרונות, כל הנושא של Rootkits נעשה מאוד פופולארי בתחום הרוורסינג, Rootkits נפוצים בעיקר בתחום ה-Malware והגנות. מצד ה-Malware למטרות זדוניות – השגה ומסירה של מידע בצורה מאוד מסוות, או פשוט להשחתה. מצד ההגנות – יש שימוש מאוד רחב של ה-Rootkits בתחום ההגנות על משחקי הרשת, כגון GameGuard שטוען דרייבר (או Rootkit) על מנת לזהות פעולות כמו כתיבה לא רצויה לתהליך המשחק ומניעת שימוש בבוטים. במאמר זה אסביר לכם על: Rootkit – מהו? מהן השימושים שלו? איך ניתן לממש את כל הטכניקות האלו וכיצד ניתן להתגונן.

מהו Rootkit? הגדרה מקובלת של Rootkit, היא ערכה (kit) המכילה אפליקציות קטנות ושימושיות שמאפשרת לתוקף להשיג גישת "Root" למחשב הנתקף – הגישה הכי גבוהה שיכולה להינתן למשתמש. במילים אחרות, Rootkit הוא אוסף של אפליקציות ופונקציות אשר מאפשרים נוכחות קבועה ובלתי ניתנת לזיהוי של התוקף על המחשב של הנתקף, לרוב אלו נמצאים ברמת הקרנל (ה-Rootkit הוא דרייבר).

מונח המפתח פה הוא "בלתי ניתנת לזיהוי"- הסיבה העיקרית לרדת את כל הדרך אל רמת הקרנל, היא בשביל **הסוואה**. רוב הטכניקות בהן ה-Rootkit משתמשת הן בעצם כדי להסוות קוד או מידע באיזושהי צורה, למשל, החבאה של תיקיות, קבצים ותהליכים. טכניקות אחרות כוללות ציתות, הסנפה של פאקטים הנשלחים מכרטיס הרשת, ושליחת מידע בצורה **בלתי ניתנת לזיהוי**.

לפני תחילת המאמר, יש להזכיר כי כל המידע הניתן הוא משתנה מפלטפורמה לפלטפורמה ומגרסאות שונות של מערכת ההפעלה. רוב המידע פה מתייחס למערכת ההפעלה Windows NT עד Windows XP.

מבנה הדרייבר

נקודת הכניסה של דרייברים מוגדרת כ-"DriverEntry" – ממש כמו הפונקציה ()main או ()WinMain, גם לדרייברים יש נקודת הכניסה משלהם. ההגדרה שלה היא כזאת:

```
NTSTATUS
  DriverEntry(struct _DRIVER_OBJECT  *DriverObject,
PUNICODE_STRING  RegistryPath )
```

הפונקציה מקבלת שני ארגומנטים, ארגומט ראשון הוא הפוינטר למבנה הנתונים- DRIVER_OBJECT – המייצג את הדרייבר בקרנל, המבנה מכיל בין השאר את הטבלה של הפוינטרים לפונקציות המשמשות לתקשורת עם הדרייבר. התפקיד המרכזי של ה-DriverEntry בדרייברים רגילים הוא להקצות פוינטרים לפונקציות החשובות שקשורות בעיקר בתקשורת מקומית עם אותו הדרייבר (MAJOR_FUNCTIONS שמוגדרות ב-DRIVER_OBJECT) פונקציות אלו נקראות בשליחה של כל IRP מתאים (IRP- או " I/O Request Packet" הן חבילות מידע הנשלחות לדרייבר לביצוע משימות כאלה ואחרות), בין הפונקציות המוכרות יש את: IRP_MJ_READ IRP_MJ_WRITE וכו'. ברב המקרים, כאשר מדובר בדרייברים "רגילים", ה-DriverEntry פשוט מקצה פונקציות ל-MAJOR_FUNCTIONS שבשימושה וחוזרת.

הארגומנט השני, הוא פשוט מחרוזת שמייצגת את ה-Path למפתח הרג'יסטרי של הדרייבר. דוגמא:

```
#include "ntddk.h" //מכיל הגדרות לשימוש בדרייבר
VOID OnUnload( IN PDRIVER_OBJECT pDriverObject )
{
 DbgPrint(" OnUnload called.");
}

NTSTATUS DriverEntry( IN PDRIVER_OBJECT pDriverObject, IN
PUNICODE_STRING theRegistryPath )
{

// Allow the driver to be unloaded
 pDriverObject->DriverUnload = OnUnload;
 return STATUS_SUCCESS;
}
```

לשרוד את ה- Reboot

כנראה שהמחשב יבצע פעולת Reboot בשלב מסוים – על ידי המשתמש או על ידי תוכנה, וכדאי מאוד שה-Rootkit שלנו יעמוד בזה, אחרת הוא פשוט לא ירוץ. קיימות טכניקות רבות לשרידת ה-Reboot, אציג פה מספר מצומצם שלהן:

שימוש בדרייבר קיים

שיטה נפוצה לשרידת ה-Reboot היא "אינפקציה" של דרייבר קיים, דרייבר שאמור להטען בכל מקרה ב-Reboot. הכוונה היא השתלטות על דרייבר קיים או על קובץ אחד מהקבצים שהוא טוען. Rootkits רבים משתמשים (בצורה אירונית למדי) בדרייברים של אנטי-וירוסים על מנת לטעון את עצמם. למשל Rootkit ה-TDL3 המפורסם הלך עמוק ונמוך יותר, במקום להכניס את הקוד שלו בקבצים ב-File System כמו הגרסאות הקודמות שלו, החליט המפתח לכתוב את הקוד **ישירות לסקטורים של ה-Hard Disk,** והוא נמצא בסקטורים האחרונים של הדיסק הקשיח, ולא כחלק ממערכת הקבצים.

הדרייבר של ה-Rootkit השתמש בדרייבר קיים של ה-Miniport, כתב ב-rsrc section את ה-824 בתים שהולכים לטעון את הקוד מהדיסק הקשיח שהחליפו 824 בתים אחרים שהיו שם (כך גודל הדרייבר לא השתנה ולא היה ניתן לראות את השינוי, הבתים שהוחלפו נשמרים בדיסק הקשיח ויטענו בהמשך כדי שהדרייבר יתפקד כראוי) ושינה את נקודת הכניסה של הדרייבר לקוד שלו.

כך, בפעם הבאה שהמערכת עושה Reboot, הדרייבר נטען, מחכה לסיום טעינת מערכת הקבצים על ידי כך שהוא רושם את עצמו כ-File System Notification Routine, ואז טוען את הקוד עצמו מהסקטורים האחרונים של הדיסק.

רישום ה- Rootkit כשירות מערכת

ה-Rootkit יכול לרשום את עצמו כשירות מערכת אשר יטען עם הפעלתה של מערכת ההפעלה באמצעות הפונקציה OpenSCManager, CreateService. טכניקה זו דורשת רישום של מפתח רג'יסטרי, שיכול להיות מזוהה בקלות, אך ה-Rootkit יוכל להחביא את המפתח הנתון מהמערכת, כפי שאסביר בהמשך המאמר.

שינוי הקרנל או ה-Boot-Loader

ניתן לשנות את הקוד של הקרנל עצמו הנמצא על הדיסק על מנת לגרום לטעינתו של הדרייבר בהתחלת המערכת, כמו כן, ניתן כמובן לשנות את הקוד של ה-Boot Loader שטוען את הקרנל עצמו, אך יש להזהר במיוחד כשמשתמשים בטכניקות אלו משום שהן יכולות לגרום נזק תמידי למערכת ההפעלה או מניעת הפעלתה.

Kernel Hooks

שיטה מאוד נפוצה של Rootkits היא ביצוע Hooks בפונקציות הנמצאות ב-SSDT (System Service Dispatch Table). השימוש העיקרי של Rootkits בתחום ה-Hooks הוא הסוואה של כל מיני פעילויות שיכולות להראות חשודות ליישומי אנטי-וירוס וכלים שונים, יש להסוות את הקבצים של ה-Rootkit, את הקוד, את התקשורת וכו'. כמובן שנוכל פשוט לבצע Hooks על פונקציות שנקראות ב-User Mode כמו ReadFile/CreateFile כדי לשבש תוצאות אך מהיות האנטי-וירוס דרייבר, יש לו יתרון **מאוד** גדול עלינו, וזיהוי Hook כזה הוא כמעט ברור מאליו. כל מה שעליו לעשות הוא להשוות את הכתובת ב-IAT (במקרה

של Hook IAT) או את ה-Prolog (חמשת הבתים הראשונים של הפונקציה) לערכים המקוריים. מפני שהרבה יותר קשה להערים על האנטי-וירוס ב-User Mode, נצטרך לבצע Kernel-Mode Hooks!

SSDT Hooks

ה-SSDT (או System Service Dispatch Table), למרות שמו המפחיד, הוא פשוט טבלה של פוינטרים המצביעים לפונקציות, לרוב טבלה זאת משומשת בUser Mode בצורה עקיפה. נניח שאנו קוראים לפונקציה CreateFileA בתוכניתנו. המימוש שלה ממש פשוט, בהתחלה הכנה של הפרמטרים – ואז קריאה ל- ZwCreateFile שנמצאת ב- ntdll. ומה המימוש של ZwCreateFile?

```
7C95D0AE   B8 25000000     MOV EAX,25
7C95D0B3   BA 0003FE7F     MOV EDX,7FFE0300
7C95D0B8   FF12            CALL DWORD PTR DS:[EDX]
7C95D0BA   C2 2C00         RETN 2C
```

הערך (או האינדקס) 25 הושם בEAX וישנה קריאה ל- KiFastSystemCall ([0x7FFE0300]):

```
7C95E510   8BD4            MOV EDX,ESP
7C95E512   0F34            SYSENTER
7C95E514   C3              RETN
```

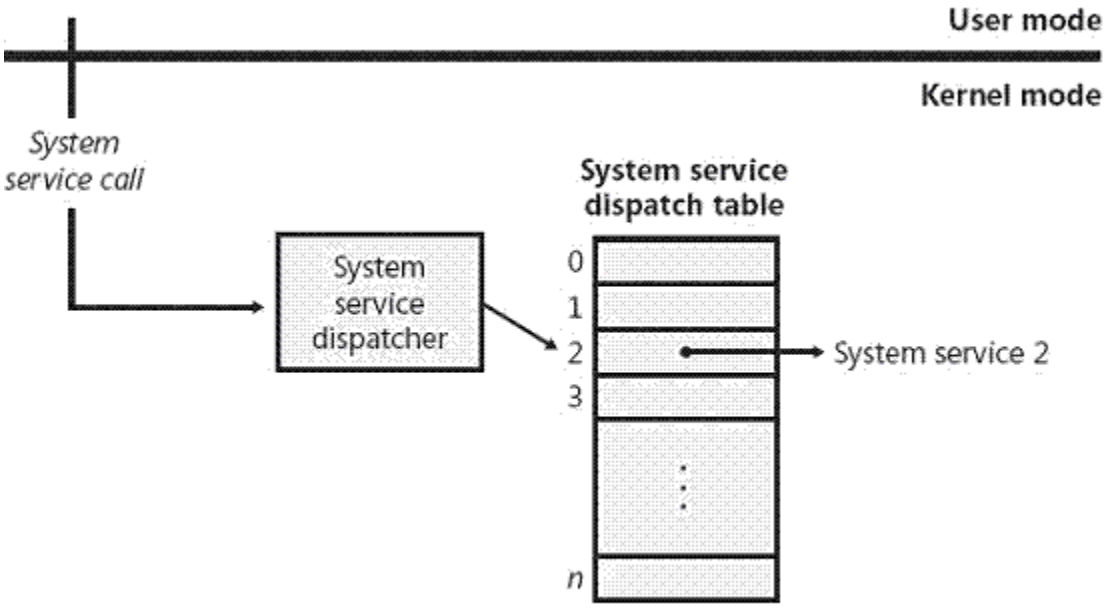
שמבצע:

1. שמירת כתובת המחסנית (לשימוש הפרמטרים).
2. שימוש בהוראה SYSENTER. ההוראה SYSENTER נכנסת לקרנל בצורה הבאה: קוראת באוגר ה- MSR (שהוא ייחודי לכל מעבד) את השדות הבאים:

- IA32_SYSENTER_CS – שדה המכיל את ה- Code Segment Selector של הקרנל שיוחלף בנוכחי (של User Mode), כאן אנחנו כבר עוברים לקרנל.
- IA32_SYSENTER_ESP – שדה המכיל את ה-Stack Segment Selector שיחליף את ה-User Mode Stack.
- IA32_SYSENTER_EIP – שדה המכיל את הכתובת של הפונקציה KiSystemService שהוא ה- Dispatch Handler שלנו. ומכאן ממשיכה הריצה ואנחנו בקרנל.

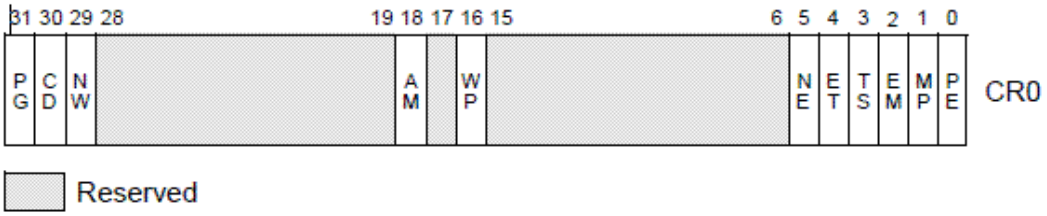
הפונקציה KiSystemService לוקחת את האינדקס (ששמנו ב-EAX ב-ZwCreateFile) של הפונקציה (CreateFile במקרה שלנו) ומוצאת את הפוינטר לפונקציה בטבלת ה-SSDT, ומשם ממשיכה הריצה עד שחוזרים לUser Mode, ומהפונקציה CreateFile.

ניתן לתאר את כל מה שהסברתי בתרשים הבא:



(מתוך
http://cfs4.tistory.com/upload_control/download.blog?fhandle=YmxvZzUxOTQ0QGZzNC50aXN0b3J5LmNvbTovYXR0YWNoLzAvMDEwMDAwMDAwMDAwLnBuZw%3D%3D)

נמשיך לחלק המעניין, ה-Hook עצמו. מבחינת הדרייבר, ביצוע ה-Hook הוא ממש פשוט – כל מה שעלינו לעשות הוא לשנות את הפוינטר ב-SSDT באינדקס הנכון להצביע לפונקציה שלנו. אך ה-SSDT הוא אזור מוגן בזיכרון, לא ניתן לכתוב עליו! למזלנו, יש טריק פשוט שבזכותו ניתן לבצע SSDT Hooks בפשטות רבה - משנים את תוכן ה-0 Control Register (CR0). ה-CR0 נראה כך:



מה שמעניין אותנו הוא ביט מספר 16, WP - ביט זה קובע האם המעבד יוכל לכתוב לדפים אשר מוגדרים כ-Read-Only במערכת ההפעלה, לכן, אם נכבה אותו - נוכל לכתוב לכל דף בזיכרון. לכן נוכל לכתוב Code Snippet קטן שיכבה זמנית את הביט , יכתוב לאזור ה SSDT -ויחזיר את המצב לקדמותו, על מנת שלא ניצור בעיות בעתיד:

```
#pragma pack(1)
typedef struct ServiceDescriptorEntry
{
 unsigned int *ServiceTableBase;
 unsigned int *ServiceCounterTableBase;
 unsigned int NumberOfServices;
 unsigned char *ParamTableBase;
} ServiceDescriptorTableEntry_t, *PServiceDescriptorTableEntry_t;
#pragma pack()

__declspec(dllimport) ServiceDescriptorTableEntry_t
KeServiceDescriptorTable;

NTSTATUS HookSSDTEntry(WORD INDEX, DWORD Handler, DWORD* Original)
{
      __asm
      {
            cli
            push eax
            mov  eax, CR0
            and  eax, 0FFFEFFFFh
            mov  CR0, eax
            pop eax
        }
        *Original =
InterlockedExchange((PLONG)&(KeServiceDescriptorTable.ServiceTableBase[
INDEX]), (LONG)Handler);
        __asm
        {
              push eax
              mov eax, CR0
              or eax, NOT 0FFFEFFFFh
              mov CR0, eax
              pop eax
              sti
        }
}
```

יש לזכור שיש עוד טכניקות נוספות לכתיבה ל-SSDT ולהתחמק מההגנה, בחרתי פה לתת את הפשוטה ביותר. אסביר בקצרה את הקוד. בתחילתו ישנה הגדרה של המבנה ServiceDescriptorEntry, למה אנו צריכים להגדיר אותו? כי הדרך היחידה להגיע ל-SSDT הוא דרך ה- System Service **Descriptor** table, שהוא ServiceDescriptorEntry של ה-SSDT שמיוצא על ידי המערכת. אחר כך ייבוא והגדרת הסמל KeServiceDescriptorTable שדיברתי עליו קודם.

אז הגדרת הפונקציה HookSSDTEntry שמקבלת כארגומנט את האינדקס, הכתובת להחלפה והפוינטר למשתנה שיכיל את הכתובת המקורית של הפונקציה. הפונקציה מכבה את ה-Write-Protect bit ב-cr0, מחליפה את הערכים הרצויים ומשחזרת את אוגר ה-cr0 למצבו הקודם. הסיבה שבגללה ישנם ההוראות cli וsti (שמכבות ומדליקות את האפשרות ל-Interrupts בהתאמה) לפני ואחרי כיבוי והדלקת הגנות הדף, היא שאין ברצוננו שיפריעו לנו באמצע הקוד כשהגנות הדף לא פועלות. דבר זה יכול לגרום לשיבושים רבים במערכת.

ניתן להשתמש ב-SSDT Hooks לכל מיני מטרות כמו החבאת תהליכים, החבאת מפתחות ב-Registry (שלפעמים דרוש לאחסן מידע או לשרוד Reboot) באמצעות Hook ל-ZwOpenKey וכיוצא בזה. גילוי SSDT Hooks מאוד פשוט, כלים ואנטי וירוסים רבים פשוט עוברים על ה-Entries ב-SSDT לבדוק אם כל כתובת לא נמצאת במרחב הזיכרון השייך לדרייברים.

המאמר הבא מציג שתי טכניקות להחבאת ה-SSDT Hooks:

http://rootkit.com/newsread.php?newsid=922

ניתן לראות מימוש של Rootkit המשתמשת בSSDT Hooks להחבאת תהליכים בכתובת הבאה:

https://www.rootkit.com/vault/fuzen_op/HideProcessHookMDL.zip

IDT Hooks

ה-IDT (קיצור של Interrupt Descriptor Table) הוא טבלה, כמו ה-SSDT, המכילה פוינטרים ל-Handlers שיקראו בעת Interrupt. Interrupt (או בעברית: 'פסיקה') זאת פונקציה של מערכת ההפעלה שמאפשרת יעילות רבה בתקשורת עם חומרה. במקום שהתוכנה המקושרת לחומרה מסוימת, כמו למשל מקלדת, תעמוד בלופ אינסופי שתבדוק כל פרק זמן מסוים את האוגרים של המקלדת, סתם תבזבז לנו זמן יקר ומשאבים יקרים לא פחות, הומצא ה- PIC (Programmable Interrupt Controller) שמאפשר ל**חומרה עצמה** להגיד למערכת מתי **היא מוכנה**! נקח לדוגמא את המקלדת, כאשר המשתמש מקליד הצ'יפ של המקלדת מעדכן את האוגרים שלו ואז שולח אות ל-PIC שהמידע מוכן. ברגע זה ה-PIC קוטע את פעולת המעבד (אם ה-Interrupt Flag פועל) ושולח אותו להריץ מיד את ה-Interrupt Handler המתאים, לפי הכתובת שממופה ב-PIC.

כך Rootkit מסוים יכול לבצע IDT Hook בדומה ל-SSDT Hook ולשבש את ריצת הקוד לטובתו. בדרך זו גם ניתן לשנות או להקליט מידע המתקבל או נשלח מחומרה כמו מקלדת, מדפסת, כרטיס רשת, וכו'. שיטה זאת נפוצה בקרב Keyloggers (תוכנות המקליטות את הקלדות המשתמש ומתעדות אותם), הם רושמים Interrupt Handler משלהם במקום IRQ1 (שהיא של המקלדת, ניתן לראות רשימה מלאה של ההקצאות פה: http://www.simulationexams.com/SampleQuestions/a+_q4.htm) ומתעדים כל לחיצה על מקש.

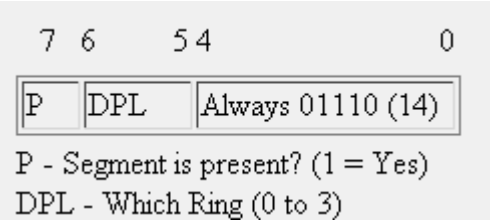
לשם שינוי ה-IDT עלינו להשיג את הכתובת שלה איכשהו וניתן להשתמש בהוראה SIDT כדי לאחסן את תוכן אוגר ה-IDTR (שמכיל פוינטר ל-IDT) בכתובת הרצויה. ה-IDTR מורכב מ-6 בתים: 2 הבתים התחתונים הם ה-Limit Field של האוגר – הגודל המירבי של ה-idt, שהוא בדרך כלל 2048 (או 256 Entries כשיודעים שכל Entry היא 8 בתים). 4 הבתים העליונים הם כתובת פיזית (32 ביט) אל ה- IDT Base Address.

```
struct idtr
{
unsigned short limit;
unsigned short LoBase;
unsigned short HiBase;
} __attribute__((packed));
```

בנוסף, יש לזכור את מבנה ה-Interrupt Descriptor Entries:

```
struct idt_entry
{
unsigned short base_lo;
unsigned short sel; //Kernel Segment Selector
unsigned char always0; // reserved
אסביר בהמשךunsigned char flags; //
unsigned short base_hi;
} __attribute__((packed));
```

שדה ה-Flags מייצג את מפת הביטים הבאה:



כש-DPL מייצג את הטבעת בה ה-Interrupt יכול להקרא, ring0 או ring3 (לפסיקות תוכנה). כך, ניתן לממש IDT Hook למקלדת בצורה הבאה:

```
#define KEYBOARD_ENTRY 1

struct idtr IDTR;
struct idt_entry KeyboardEntry;
struct idt_entry *IDT;

unsigned int HookKeyboard(unsigned int Handler)
{

      __asm sidt IDTR;  //Load IDTR with the idtr
      IDT = (idt_entry *)((IDTR.HiBase << 16) | IDTR.LoBase);
      unsigned int Original = ((IDT[KEYBOARD_ENTRY].base_hi << 16) |
IDT[2].base_loh); //store original entry.
      KeyboardEntry = &(IDT[KEYBOARD_ENTRY]);
      __asm cli; //disable interrupts so we won't crash.
      *(short *)(KeyboardEntry) = (short)Handler;
      *(short *)(KeyboardEntry+6) = (short)Handler >> 16;
      __asm sti; //re-enable interrupts
      return Original;
}
```

ניתן לראות מימוש מלא של Rootkit המשתמש ב-IDT Hook בשביל Keylogging בכתובת הבאה:

https://www.rootkit.com/vault/chpie/idt_src.zip

יש לזכור כי ה-IDT הוא ייחודי לכל מעבד, זאת אומרת שאם יש לנו מערכת עם 2 מעבדים, נצטרך לבצע את ה-IDT hook על כל מעבד בנפרד. (כמובן שניתן לעשות זאת באמצעות לולאה וקריאה לפונקציה KeSetTargetProcessorDpc)

לסיכום

זהו המאמר הראשון מתוך סדרת מאמרים על Rootkits, במאמר זה הסברנו מהו Rootkit, השימוש שלו ומטרתו. הצגנו את מבנה הדרייבר, Kernel Hooks – מה מטרתם, ולמה הם כדאיים. הצגנו נושאים נוספים כגון SSDT ו-IDT. במאמר הבא נסביר עוד מספר טכניקות המשמשות Rootkits למיניהם.

ביבליוגרפיה מומלצת: Programming the Windows Driver Model,Rootkit.com.

סקירת טכנולוגיות Firewalling שונות

מאת יגאל סולימאני ואפיק קסטיאל (cp77fk4r)

הקדמה

כידוע לכל, Firewall הוא כלי הנועד למדר את הסקטורים השונים ברשת שלנו, בין אם מדובר במידור חלקים שונים ברשת הפנים-אירגונית שלנו, ובין אם מדובר במידור שבין רשת האינטרנט לבין הרשת הפרטית בבית שלנו.

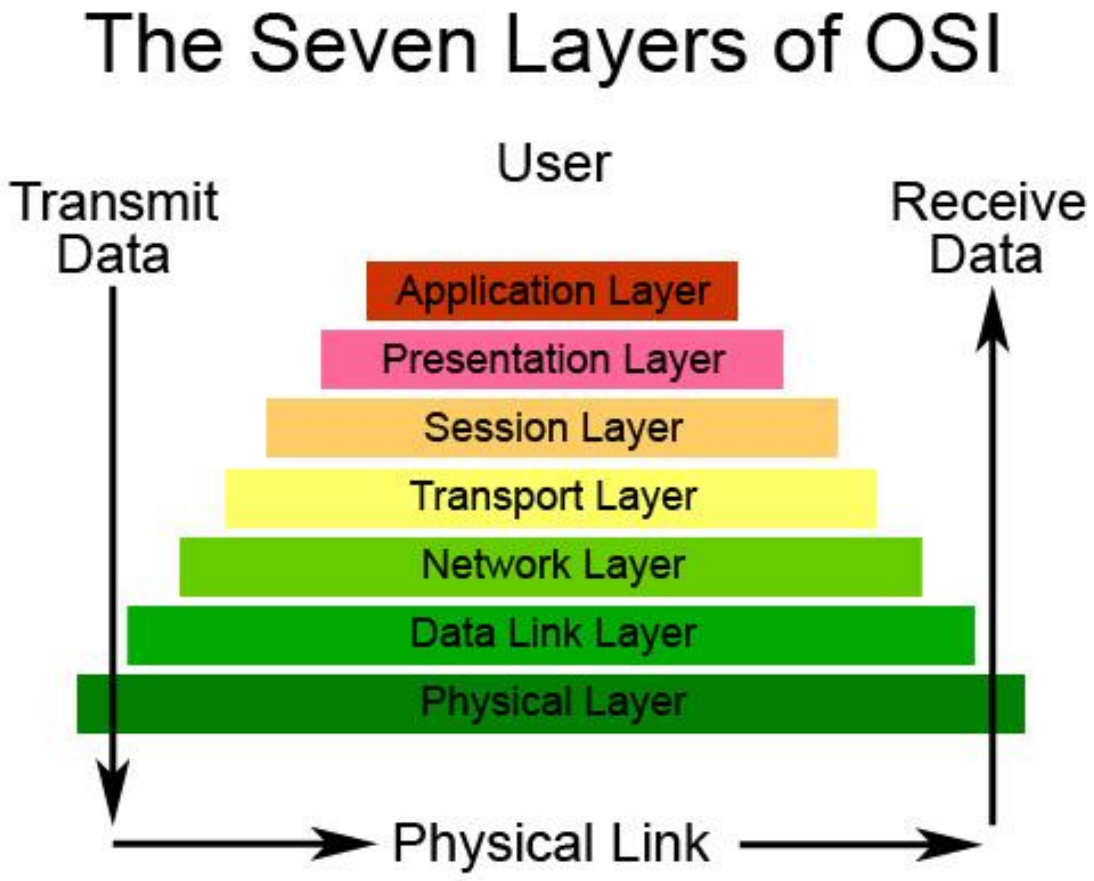
למרות שהכלי עצמו מוכר, לא כולם מכירים לעומק את השיטות השונות הקיימות לביצוע פעולה זאת ובכך עוסק מאמר זה. טכנולוגיית ה-Firewall אמורה להוות מעין פילטר שתפקידו להעביר אך ורק את חבילות המידע העומדות בסטנדרטים שקבענו מראש ולסנן את שאר חבילות המידע. המשימה יחסית פשוטה, אך כאשר מדובר במשימות שדורשות יותר מחסימת גישה לערוצים ספציפיים, או חסימת גישה לכתובת מסוימת, המשימה נעשית קצת יותר מורכבת. ישנו מספר לא קטן של דרכים לענות על המשימות השונות - ולכן ישנם סוגים רבים של טכנולוגיות Firewalling. במסגרת מאמר זה נציג את הטכנולוגיות הנפוצות כיום הנמצאות בשימוש, נסביר את ייעודן, את יתרונותיהן ואת חסרונותיהן.

כידוע, בכדי לממש העברת נתונים על גבי הרשת אנו משתמשים במודל ה-OSI או "מודל שבע השכבות":

> מודל ה-OSI (קיצור של: Open Systems Interconnection) הוא מודל שכבתי אשר נוצר על ידי ארגון התקינה הבינלאומי. מטרת המודל היא להציג את הפעולות השונות הנדרשות על-מנת להעביר נתונים ברשת תקשורת ,ואת הסדר בין הפעולות השונות. המודל מתייחס לחומרה , לתוכנה ולשידור וקליטת הנתונים, ובין השאר, מספק הסבר כללי על מרכיביה השונים של הרשת. במודל שבע שכבות, והוא מכונה לעתים "מודל שבע השכבות."
>
> (צוטט מוויקיפדיה תחת הערך: מודל ה-OSI)

לא נכנס ונסביר את תפקיד כל שכבה (את זה אפשר לקרוא בויקיפדיה), אך להמשך הבנת המאמר חשוב להבין דבר אחד - ככל שנעלה בשכבות (נתרחק מהשכבה הפיזית ונתקרב לשכבה האפליקטיבית) כך נקבל יותר מידע לגבי תעבורת הרשת, וככל שיהיה בידינו יותר מידע לגבי תעבורת הרשת - כך נוכל להיות יעילים יותר בעת מימוש טכנולוגיית Firewalling.

כך נראה מודל 7 השכבות:



(במקור: http://www.washington.edu/lst/help/computing_fundamentals/networking/img/osi_model.jpg)

השכבה הנמוכה ביותר במודל, שבה טכנולוגיות ה-Firewalling פועלות (להוציא מקרים מיוחדים) היא השכבה השלישית (שכבת הרשת, Network). בשכבה זו ניתן לנתח את נתוני חבילות המידע לפי נתוני התקשורת הבסיסיים ביותר - מקורה וייעדה של חבילת המידע. במידה ואנו עובדים בשכבה השלישית, לא נוכל לבצע בדיקות על תוכן חבילת המידע (בכדי למנוע מתקפות ספציפיות), בנוסף, לא נוכל לקשור חבילת מידע אחת לחברתה (Indexing) - וכך למנוע מתקפות הנחשבות למתקדמות יותר. בשכבה זו פועלות טכנולוגיות Firewalling בסיסיות ביותר.

Packet Filtering

הטכנולוגיה הראשונה שבה ניגע במאמר זה היא טכנולוגיית ה- Packet Filtering. טכנולוגיה זו סורקת כל חבילה שיוצאת או נכנסת, על פי טבלת החוקים המוגדרת לה, ומחליטה האם החבילה הזו מורשת לעבור הלאה או שהחבילה נחסמת ונשמטת.

את טבלת החוקים ניתן לקטלג רק על פי כתובת היעד או כתובת המקור, פורט המקור והיעד וסוג הפרוטוקול שבו מועבר המידע (TCP /UDP). טכניקה זאת הוצגה לראשונה בשנת 1988, לאחר מחקר שביצעו מספר מפתחים מחברת DEC ואז טכנולוגיה זו נחשבה כפריצת דרך בתחום אבטחת המידע.

טכניקה זו היא הנפוצה ביותר (והזולה ביותר) מכיוון שאין צורך בתוכנה נוספת שתעמיס על הציוד, ולכן רוב הנתבים תומכים בטכניקה זו. לדוגמא, אם נרצה לחסום כתובות IP מסוימות או סגמנטים מסוימים לשימוש FTP, בטבלת החוקים נחסום את כתובות היעד (Any) והמקור לשימוש בפורט 21 . לרב, טכנולוגיות Firewalling אלו, מאופיינות בממשק ניהול הכולל "Access List" – רשימה המאפשרת למנהל היישום לקבוע לאילו כתובות תאופשר גישה ולאלו כתובות לא.

<u>יתרונות ב Packet Filtering:</u>

- **שקטה** - לא דורשת משאבים רבים מפני שהיא לא נכנסת לעומקה של חבילת המידע.
- **זריזה** - היא אינה מתעכבת על ניתוח נתוני החבילה על פי חוקים ורגולציות.

<u>חסרונות:</u>

- **לא חכמה -** הטכנולוגיה אינה לומדת את כלל מאפייני התקשורת (State) ולכן היא פגיעה למתקפות כגון IP Spoofing, Data Driven Attack, Source Route Attack, SYN Flood וכו'.
- **לא פשוטה לניהול -** במערכות גדולות, קשה מאוד לנהל את טבלת החוקים ויכולות להיות סתירות, מכיוון שחבילה עוברת/נחסמת על פי הכלל הראשון שמתאים לחבילה זו. למשל, כתובת IP מסויימת יכולה להשתייך לקבוצה A וגם לקבוצה B, לשתי הקבוצות הרשאות שונות בטבלת החוקים. כשחבילה מכתובת ה- IP הזו תגיע ל-Firewall, היא תגיע לשורה של קבוצה A שחסומה ליציאה ולכן החבילה תחסם ותישמט למרות שבקבוצה B היא מוגדרת כאחת שכן יכולה לצאת החוצה.
החבילה מועברת הלאה או נזרקת על פי השורה הראשונה בטבלת החוקים שמתאימה לחבילה, אם אף שורה לא מתאימה החבילה נשמטת.

השכבה הבאה במודל, שבה מופעלות טכנולוגיות ה- Firewalling, היא השכבה החמישית (שכבת ה-SESSION), שכבה זו עדיין נחשבת שכבה "נמוכה" ולא נדרשים משאבים רבים בכדי לבצע את החישובים בה.

Circuit-Level Gateway

טכנולוגיית ה- Firewalling המוכרת ביותר הפועלת בשכבת הרשת החמישית (שכבת ה-SESSION), הינה טכנולוגיית ה-Circuit Gateway הידועה גם כ-Relay firewall Circuit. משום שהטכנולוגיה פועלת בשכבת ה-SESSION, היא אינה נתונה למגבלות הקיימות בטכנולוגיות הממוקמות מתחתיה ושימוש בה מאופיין בניתוח מאפיינים שונים בתקשורת.

שלא כמו בשכבה השלישית במודל, בשכבה החמישית ניתן ללמוד את כל הנתונים בחבילת התקשורת ולא רק את נתוני מקור החבילה או את יעדה. בעזרת ניתוח של כלל נתוני חבילת התקשורת ניתן לייעל את סינון המידע ולאפשר למנהל היישום לממש חוקים מורכבים יותר. מתקפות המבוססות על אי-יכולת ה- Firewall לזהות קשרים בין חבילות התקשורת יפלו כנגד הטכנולוגייה הזאת, כמו כן גם מתקפות כגון DoS המבוססות על מנגנון הרכבת חבילת המידע השלמה וניתוחה ע"י ה-Firewall. לעומת זאת מתקפות Stateless, כגון העברת נתונים זדוניים בחבילת נתונים אחת מתוך State שלם של חבילות נתונים לא תזוהה על ידי טכנולוגיה זו מפני שאין כאן בדיקה של חבילת הנתונים הבודדת, אלא של כלל ה-State.

<u>היתרונות הקיימים בטכנולוגיה זו:</u>

- **שקטה** - הטכנולוגיה אינה דורשת משאבים רבים מפני שאינה נכנסת לעומקה של חבילת מידע אחת אלא לעומקו של כלל ה-State.
- **זריזה** - הטכנולוגיה אינה מתעכבת על ניתוח נתוני החבילה הבודדת.
- **חכמה-** לומדת ומסיקה מסקנות על ידי כלל State התקשורת ולא על פי חבילות מידע בודדות.

<u>והחסרונות הם:</u>

- **פזיזה -** הטכנולוגיה אינה מסוגלת לבצע בדיקת חבילת נתונים יחידה ועל כן נופלת במתקפות כגון מתקפות המבצעות שימוש Tunneling דרך פרוטוקול תקשורת מאושר אחר.

Filtering Gateway

טכנולוגיית ה-Stateful Packet Filtering הראשונה שנכיר היא מסוג Application Filtering. ישנן שתי דרכים לממש טכנולוגיה זו:

- שימוש מקומי של המערכת.
- שימוש בשרת חיצוני המהווה שרת Proxy לתקשורת הרשת.

שימוש בשרת פרוקסי לסינון וניטור תעבורת הרשת נקרא- Filtering Gateway. הרעיון הוא להקצות שרת ייעודי אשר ימוקם בקצה (או במרכז, תלוי בתפקידו) של רשת התקשורת - בנקודה בה הרשת מתחברת לרשתות השונות בארגון המסכנות אותה (הרשת הכללית של הארגון, רשת האינטרנט, DMZ וכו') וכך לבצע ניטור של המידע הנכנס או היוצא מהרשת.

במידה והתוקף ירצה לבצע מתקפה על משאב רשת הממוקם ברשת הפנימית (המוגנת) של הארגון, הוא יחוייב לעבור דרך ה-Filtering Gateway. לפיכך אנו יכולים להבטיח שכל הנתונים היוצאים והנכנסים לאותה הרשת יעברו דרך שרת הפרוקסי שלנו. כך, למשל, אפשר להימנע ממתקפות כגון Source Route Attack.

<u>היתרונות בטכנולוגיה זו:</u>

- **יציבה -** הטכנולוגיה רצה בדרך כלל על שרתים ייעודיים נפרדים שלא משתמשים לעבודה רגילה ולכן אין בעיה שתוקצה לה כמות גבוהה של משאבי מחשוב.
- **אבטחה גבוהה -** שימוש בשרת Filtering Gateway עבור גישור בין רשתות הארגון השונות, מחייבות את מנהלי הרשתות בארגון לעבוד באופן מאובטח ויכולות למנוע הרבה "טעויות אנוש" המהוות חלק נכבד מבעיות האבטחה הנפוצות ביותר כיום.
- **גמישה -** Application Filtering איכותי מגיע כיום עם מגוון אפשרויות לניהול הרשת, ביסוסו על טכנולוגיית ה-Stateful Packet Filtering מאפשרת למנהל הרשת לקבוע חוקים המשלבים מאפייני תקשורת רבים.
- **חכמה -** ביסוסה על טכנולוגיית ה-Stateful Packet Filtering מונעת ממנה ליפול למתקפות Spoofing בסיסיות בשל למידתה את כלל ה-State של התקשורת.

<u>החסרונות בטכנולוגיה:</u>

- **לא פשוטה לניהול -** בהרבה מהמקרים ישנם קורסים שלמים והסמכות שצריך לעבור בכדי לדעת לתפעל שרת שכזה באופן איכותי.
- **כבדה -** דורשת משאבי חישוב רבים (לרוב משאב רשת ייעודי).
- **זמינות -** השרת אמנם יציב אך בשל תצורתו, הוא מהווה את החוליה היחידה המקשרת בין הרשת הפנימית לכלל הרשת של הארגון או רשת האינטרנט. כך, במידה והוא ייפול- **כלל הרשת הפנימית** לא תהיה זמינה עד שיקימו את השרת בחזרה.

IPTables

טכנולוגיית ה-Stateful Packet Filtering שנציג היא טכנולוגיית ה-IPTables, טכנולוגיה זו מיושמת ברוב הפצות הלינוקס כיום ומהווה אחד מגורמי האבטחה המרכזיים בהפצה. טכנולוגיה זו מבוססת על קבוצות חוקים, המשורשרים במספר סוגי טבלאות, כדוגמת:

- **FILTER** – טבלת ברירת המחדל, הטבלה הכי בסיסית, במידה ולא תקבע שום טבלה שתוגדר כאחראית לטיפול באירוע, ה-Packet יגיע לכאן.
  בטבלה זו קיימות שלוש שרשראות:
  1. **INPUT**- שרשרת המוגדרת לטפל ב-Packets אשר נכנסים למערכת.
  2. **OUTPUT**- שרשרת המוגדרת לטפל ב-Packets אשר יוצאים מהמערכת.
  3. **FORWARD**- שרשרת המוגדרת לטיפול ב-Packets המיועדים לניתוב.
- **NAT** - הטבלה האחראית לניתוב ה-Packets, בטבלה קיימות שלוש שרשראות:
  1. **PREROUTING**- שרשרת המוגדרת לטפל ב-Packets לפני הניתוב.
  2. **POSTROUTING**- שרשרת המוגדרת לטפל ב-Packets לאחר הניתוב.
  3. **OUTPUT**- שרשרת המוגדרת לטפל ב-Packets היוצאים.
- **MANGLE**- טבלה לטיפול מתקדם ב-Packets. בקרנלים החדשים (מ-2.4.18) קיימות חמש שרשראות:
  1. **PREROUTING**- שרשרת המוגדרת לטפל ב-Packets לפני הניתוב.
  2. **POSTROUTING**- שרשרת המוגדרת לטפל ב-Packets לאחר הניתוב.
  3. **INPUT**- שרשרת המוגדרת לטפל ב-Packets אשר נכנסים למערכת.
  4. **OUTPUT**- שרשרת המוגדרת לטפל ב-Packets היוצאים.
  5. **FORWARD**- שרשרת המוגדרת לטיפול ב-Packets המיועדים לניתוב.

(נלקח מהמאמר "שימוש ב-IPTables" ע"י אפיק קסטיאל הפורסם בגליון השלישי של Digital Whisper)

לכל שרשרת היכולה להכיל מספר חוקים שני מאפיינים:

- הגדרות לזיהוי חבילת המידע.
- גורל חבילת המידע.

אירועים

במידה ויווצר אירוע רלוונטי חבילת המידע תגיע לטבלה הרלוונטית (חבילות מידע יוצאות- OUTPUT, חבילות מידע נכנסות-INPUT וכו') ותעבור מול שרשראות החוקים הקיימות באותה הטבלה. במידה ותמצא התאמה בין חוק הקיים בשרשרת לבין חבילת המידע: גורל חבילת המידע יקבע לפי המצויין בטבלת החוקים. במידה ולא תמצא התאמה תתבצע בדיקה מול החוק הבא ברשימה.

<u>היתרונות של טכנולוגיה זו:</u>

- **גמישה -** ביסוסה על טכנולוגיית ה-Stateful Packet Filtering מאפשרת למנהל הרשת לקבוע חוקים המשלבים מאפייני תקשורת רבים. בנוסף, הטכנולוגיה תוכל לרוץ גם על עמדת הקצה.

- **פשוטה (יחסית)** - השימוש בה פשוט יחסית ובכדי ליצור בה חוקים בסיסיים אין צורך בהבנה עמוקה של המערכת.

- **חכמה-** ביסוסה על טכנולוגיית ה-Stateful Packet Filtering מונעת ממנה ליפול למתקפות Spoofing בסיסיות מאחר והטכנולוגיה לומדת את כלל ה-State של התקשורת.

<u>החסרונות:</u>

- **לא פשוטה לניהול -** במידה ומדובר באירגונים גדולים המכילים משאבי מערכת ותצורות רשתות הדורשות אילוצים שונים, יהיה קשה לנהל טבלאות ניתוב. שינוי של טבלה או חוק אחד בטבלה עלול לגרום לפגיעה בשרשרת חוקים שונים.

- **כבדה-** במידה ומדובר ביישום הטכנולוגיה כ-Filtering Gateway ולא ניהול של עמדת קצה- יש להקצות כוח מחשוב רב.

Application level gateways

כפי שראינו בתחילת המאמר, מודל ה-OSI מחולק לשכבות, השכבה העליונה ביותר היא שכבת האפליקציה ("Application Layer"), טכנולוגיות Firewalling שיושבות על השכבה הזאת נקראים בדרך כלל "Application level gateways" או בקיצור- ALG. לרוב מדובר באפליקציות הצורכות משאבים רבים (כמובן שהרבה תלוי באופן מימושה של הטכנולוגיה) - אך האפשרויות שהן מציעות רבות.

טכנולוגיה כזאת יכולה להיות ממומשת באופן של Session flow או כ-Packet flow. מפני שהטכנולוגיה יושבת באופן הקרוב ביותר למשתמש (מבחינת שכבות המודל) היא מסוגלת לקחת בחשבון את כלל נתוני התקשורת, במקומות שטכנולוגיות אחרות יכולות רק למנוע או לאפשר למשתמש לגלוש באתר אינטרנט. למשל, טכנולוגיה זו מאפשרת למנהל הרשת גם לקבוע מאילו כתובות יהיה ניתן לקבל מידע ואילו כתובות ספציפיות יהיו חסומות לגלישה.

<u>היתרונות בטכנולוגיה זו:</u>

- **חכמה -** בשל מיקומה הגבוהה במודל היא מסוגלת להייחס לכלל נתוני התקשורת ובאפשרותה להתייחס לכלל ה-State של התקשורת.
- **גמישה-** מפני שהטכנולוגיה מסוגלת להתחשב בכלל נתוני התקשורת, היא מאפשרת למנהל הרשת לקבוע חוקים המשלבים מאפייני תקשורת רבים.

<u>החסרונות הם:</u>

- **כבדה-** רוב צורות המימוש של הטכנולוגיה דורשים משאבי מיחשוב רבים.
- **איטית –** טכנולוגיה זאת מתחשבת בכלל נתוני התקשורת ובודקת את כלל שדות חבילת המידע, שימוש בהרבה חוקים יורגש בשל האטה משמעותית של התקשורת.

סיכום

במאמר זה הצגנו את הדרכים הנפוצות בהן ניתן לממש טכנולוגיות Firewalling להגנה על עמדת הקצה או כלל הרשת. כיום ישנם לא מעט יישומי Firewalling המבצעים שימוש במספר טכנולוגיות Firewalling בכדי לשאוב מכל טכנולוגיה את יתרונותיה וכך להגביר את האבטחה שהם מספקים. כאשר רוכשים יישום Firewalling לאירגון או למחשב האישי יש להתחשב במספר גורמים כגון תצורת כלל הרשת, סוג השימוש בחיבור האינטרנט (הורדת/העלאת קבצים, גלישה, שליחת מיילים וכו') והיקף השימוש.

סקירת טכנולוגיות ההצפנה EFS ו- BitLocker

מאת בנימין כהן

הקדמה

הצפנה הינה רצף פעולות מתמטיות הפועלות על מידע נתון והופכות אותו להיות בלתי קריא או בלתי מובן עבור מי שלא ידע איך לפענח את ההצפנה. ישנם שני סוגי הצפנות :

1. הצפנה סימטרית.
2. הצפנה אסימטרית.

הצפנה סימטרית הינה הצפנה הכוללת מפתח (או cipher – צופן). אלגוריתם ההצפנה, הידוע לשני הצדדים (השולח והמקבל) עושה שימוש במפתח לשם הצפנה ופענוח.

אלגוריתמי ההצפנה הסימטרית מתחלקים לשני סוגים :

- Stream Cipher: אלגוריתם המצפין את כל המידע, ביט (Bit) אחר ביט עם מידע פסאודו-רנדומלי (מידע הנראה אקראי, אך בעצם אינו כזה), בדרך כלל ע"י פעולת XOR.
- Block Cipher: אלגוריתם המצפין בלוקים של מידע (בלוק אחד - 128Bit). לאלגוריתם זה מספר תצורות עבודה.
  אלגוריתמים נפוצים לסוג הצפנה זה : RC4, RC5, DES, 3DES, AES

הצפנה אסימטרית (או Public Key) תלויה בשני מפתחות אשר יש ביניהם קשר מתמטי: מפתח ציבורי ומפתח פרטי. המפתח הפרטי נשמר במקום סודי (כדוגמת SmartCard) והמפתח הציבורי ניתן לפרסום. בשונה מהצפנה סימטרית, כאן אי אפשר להצפין ולפענח באמצעות מפתח אחד. הצפנה אסימטרית נחשבת כהצפנה חזקה יותר מהצפנה סימטרית מכיוון שיש שימוש במפתחות הצפנה גדולים יותר.

בהצפנה סימטרית משתמשים במפתח באורך 128-256 BIT בעוד שבהצפנה אסימטרית משתמשים במפתח באורך 1024-2048 BIT (ויש גם מפתחות ארוכים יותר). ככל שמפתח ארוך יותר, כך קשה יותר לפרוץ את המידע המוצפן.

שימושים להצפנה אסימטרית :

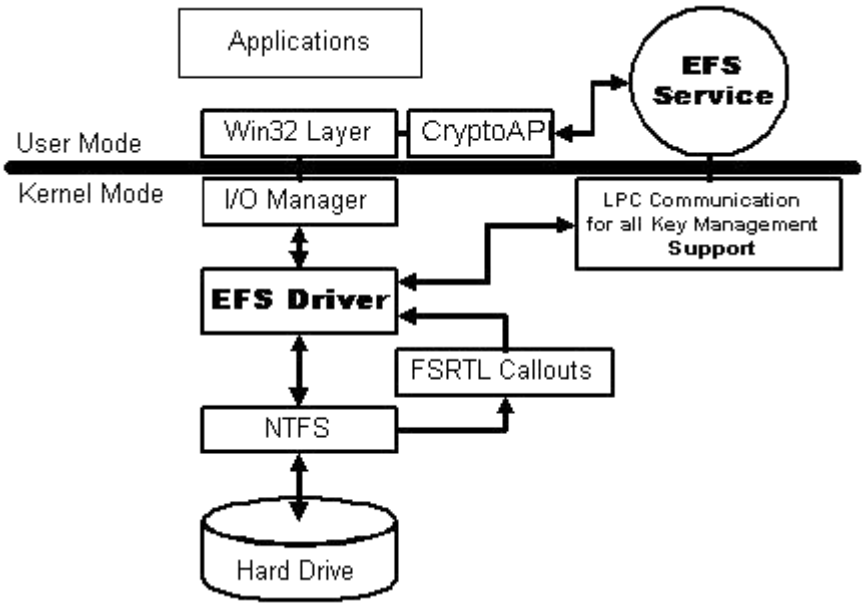
1. החלפת מפתחות סימטריים בין הצד השולח לצד המקבל.
2. חתימה דיגיטלית.

הצפנה סימטרית נחשבת להצפנה מהירה יותר מאסימטרית.

איזו הצפנה חזקה יותר? נכון להיום, חוזק הצפנה של מפתח סימטרי 128Bit שווה ערך לחוזק הצפנה של מפתח אסימטרי 1024Bit. הצפנות אלו חלשות היום ומומלץ להשתמש בהצפנות מפתח סימטרי בגודל 256Bit או אסימטרי בגודל 2048Bit.

Encrypting File System - EFS

EFS הינו רכיב של מערכת הקבצים (Ntfs). הוא מוכל בכל הגרסאות של Windows, ושימושו התחיל מ-Windows 2000 והמשיך לגרסאות הבאות. לא נדרש ידע מתקדם על מנת להשתמש ב-EFS , אך עם זאת, שימוש ב-EFS ללא הכרת השיטות המומלצות לשימוש בו עלול לתת תחושה מוטעית בכך שהקבצים אינם מוצפנים בצורה שנראה שהם. הצפנת EFS אינה מתרחשת ברמת היישום, אלא על תיקיה ברמת המערכת. אם תיקיה מסומנת על הצפנה, הקבצים אשר נוצרו בתיקיה, או שיעברו לתיקיה יהיו מוצפנים. אם משתמש מנסה לפתוח תיקיה זו, והוא בעל המפתח – היא תפתח ללא כל בעיה. במידה והוא לא בעל המפתח, הוא יקבל הודעת שגיאה "הגישה נדחתה". הצפנת קבצים זו משתמשת במפתח סימטרי.



(מקור: http://www.securityfocus.com/unix/linux/images/dicf-efs-arch.jpg)

מפתחות ה-EFS מוגנים על ידי סיסמתו של המשתמש, ובמידה והתבצעה התחברות למערכת באמצעות שם משתמש והסיסמה, המפתח נמצא אצל המשתמש, והוא יוכל להיכנס לתיקיות המוצפנות ללא כל בעיה.

EFS משתמש במפתח הצפנה סימטרי בשיתוף עם טכנולוגיית המפתח הציבורי על מנת להגן על הקבצים. קובץ הנתונים מוצפן עם אלגוריתם סימטרי (DESX). כברירת מחדל, EFS משתמש באלגוריתם DESX עם אורך מפתח של 128Bit. יש אפשרות במערכת להגדיר את השימוש באלגוריתם 3DES חזק יותר ומשתמש עם אורך מפתח של 168Bit. בעורך ה-Registry מבצעים את השינויים האם להשתמש באלגוריתם DESX או באלגוריתם 3DES.

המפתח משתמש בהצפנה סימטרית שנקראת FEK (File Encryption Key), FEK מאוחסן בקובץ, יחד עם המפתח הציבורי שמשתמש באלגוריתם RSA. הסיבה לכך שיש שימוש בשני האלגוריתמים היא מהירות ההצפנה.

תהליך ההצפנה

- צעד ראשון אחרי ההצפנה, NTFS יוצר קובץ LOG שנקרא Efs0.log כקובץ מוצפן. ה-EFS דורש גישה ל-CryptoAPI context והוא עושה זאת בעזרת Microsoft Base Cryptographic.
- ברגע שנפתח ה-Crypto context הוא יוצר FEK. אחרי שיצרנו את ה-FEK אנחנו צריכים גם מפתח ציבורי. במידה ולא קיים מפתח ציבורי (בד"כ מדובר בהפעלה ראשונה של המערכת), EFS מייצר מפתח ציבורי חדש. הוא משתמש במפתח באורך 1024Bit עם אלגוריתם RSA ומצפין אותו ביחד עם ה-FEK. לאחר שיש לנו מפתח פרטי (FEK) ומפתח ציבורי- EFS יוצר DDF (Data Decryption Field) למשתמש הנוכחי, ומאחסן בו את ה-FEK שלו ואת המפתח הציבורי.
- בנוסף, EFS יוצר DRF ומאחסן שם את ה-FEK ואת המפתח הציבורי של ה-RECOVER. DRA נוצר בנפרד לכל סוכן Recovery. לאחר שהקובץ הוצפן, רק למשתמשים התואמים למפתחות (הנמצאים ב-DDF או DRF) יש את האפשרות לגשת לקובץ. רק משתמש שיחזיק ב-FEK ובמפתח הציבורי יוכל להיכנס לקובץ.

תהליך השחזור

- תהליך השחזור דומה לתהליך הפענוח. נעשה שימוש ב-DRF (ולא ב- DDF) ובמפתח השחזור של הסוכן על מנת לפענח את ה-FEK.

אבטחה – פגיעות במערכת ופתרונות

שתי בעיות אבטחה משמעותיות קיימות ב-Windows 2000:

- <u>פענוח קבצים באמצעות חשבון מנהל מקומי</u>
ב-Windows 2000 המנהל המקומי (LOCAL) הינו ברירת המחדל של DRA (Data Recovery Agent). הוא מסוגל לפענח את כל הקבצים המוצפנים עם – EFS על ידי כל משתמש מקומי.

Windows 2000 לא יכול לתפקד ללא סוכן שחזור, ולכן תמיד יהיה מישהו שיוכל לפענח קבצים מוצפנים של המשתמשים. כל מי שאינו מנהל (Admin) ומצטרף למערכת, יהיה פגיע בכך שיהיה ניתן לפענח את הקבצים שלו דרך המנהל המקומי.

פתרון לבעיה: במערכת XP, ובמערכות הבאות אחריה, לא הוגדרה ברירת מחדל עבור Data Recovery Agent.

- <u>גישה למפתח הפרטי דרך איפוס סיסמא</u>
ב-Windows 2000, מפתח ה-RSA הפרטי של המשתמש אינו מאוחסן רק בצורה מוצפנת, יש גם גיבוי למפתח הפרטי RSA שמוגן בצורה חלשה. אם לתוקף יש גישה פיזית למערכת (Windows 2000), הוא יכול לאפס את הסיסמא של המשתמש, ובכך להיכנס ולהשיג גישה למפתח הפרטי בעזרתו ניתן לפענח את הקבצים. הסיבה לכך היא שהמפתח הפרטי נשמר כגיבוי במערכת, מוצפן עם LSA שאליו יכול להגיע כל מי שהתחבר למערכת בצורת – LocalSystem.

פתרון לבעיה: במערכת XP והבאות אחריה המפתח של המשתמש הפרטי RSA מגובה באמצעות מפתח ציבורי שמבצע התאמה למפתח פרטי אשר ממוקם ב-Active Directory.

סיכום EFS

- רכיב של מערכת הקבצים NTFS.
- ההצפנה מתרחשת ברמת התיקייה ולא ברמת היישום.
- ההצפנה מופעלת החל מגרסת Windows 2000 והבאות אחריה.
- שימוש בהצפנה סימטרית על מנת להצפין את המידע.
- האלגוריתם שבו נעשה שימוש כברירת מחדל הוא – DESX.
- אורך מפתח של אלגוריתם זה הינו 128Bit.
- יש אפשרות דרך ה- Registry לשנות את האלגוריתם ל-3DES אשר מצפין את המידע באורך מפתח של 168Bit.

BitLocker

BitLocker הינה תוכנה להצפנת דיסק מלאה, הכלולה במהדורות ה-Ultimate ו-Enterprise של Vista, Win7 וב-Windows Server 2008. בעזרת תוכנה זו ניתן להגן על נתונים בעזרת הצפנה. הצפנה באמצעות תוכנה זו היא אחת הדרכים הטובות להגן על מחשבים ניידים מפני אובדן נתונים כאשר המחשב נגנב או אובד.

BitLocker דורשת אבטחה על ההצפנה שהתוכנה משתמשת אך קיימת בעיה: האלגוריתמים שקיימים כיום, העונים על דרישות האבטחה איטיים מדי, ולכן אינם מתאימים. ואילו, לא ניתן להשתמש באלגוריתם חדש, לפני שנחקר במשך מספר שנים, ושנכתבה עליו ביקורת ציבורית.

הבעיה נפתרה באמצעות שילוב של אלגוריתם AES בשילוב עם CBC עם מרכיב חדש שמכונה – Diffuser. שכבת ה-Diffuser מוסיפה מאפייני אבטחה נוספים הרצויים בהגדרות ההצפנה, אך לא ניתנים ע"י שיטות ההצפנה של AES-CBC.

על ידי שילוב של AES-CBC ו-Diffuser אנו נהנים משילוב של שני עולמות לאבטחת הנתונים: מחד - אנו יכולים להשתמש בכל מאפייני האבטחה המסופקים לנו ע"י אלגוריתמי ההצפנה AES-CBC, ומאידך - אנו יכולים לעשות שימוש במאפייני אבטחה נוספים שלא יסופקו ע"י שימוש ב-AES-CBC בלבד, במהירות העולה על האלטרנטיבות המצויות כיום.

BitLocker תומכת במפתחות הצפנה באורך של 128Bit ו – 256Bit עם או בלי Diffuser. הגדרת ברירת המחדל הינה שימוש באלגוריתם AES-CBC , במפתח הצפנה באורך של 128Bit עם Diffuser. בכל מקרה, הגדרות אלו ניתן לשנות בעזרת ה-Local Group Policy Editor.

קיימות 3 אופציות (מנגנוני אימות) לשימוש עם ה-BitLocker:

1. מפתח USB– על המשתמש להוסיף מכשיר USB שבו נמצא המפתח להפעלת המערכת המוגנת. שימוש ב-USB דורש שה-BIOS יכיר בקריאה מהתקן USB.
2. שימוש ב-TPM 1.2(Trusted Platform Module (TPM – במצב זה יש להשתמש בחומרת ה-TPM על מנת לשמור את המפתח של ההצפנה, כאשר מפתח ההצפנה מוצפן על ידי שבב ה-TPM. רק כאשר מחברים את ה-TPM והוא מזהה את המפתח ניתן להיכנס למערכת.
3. אימות פרטי – מצב זה מחייב את המשתמש לספק אימות לפני האתחול בצורת קוד PIN.

הצפנת קבצים זו משתמשת במפתח סימטרי.

*TPM- Trusted Platform Module הוא שבב אלקטרוני התומך בתכונות אבטחה מתקדמות כדי להצפין את כונן מערכת ההפעלה.זה המקום שבו ה-BitLocker מאחסן את מפתח ההצפנה.

אבטחה – אפשרויות שונות לפגיעה במערכת.

בתאריך 10.12.09 התפרסמה ידיעה בנושא אבטחת המערכת של BitLocker. חוקרי המכון Fraunhofer SIT הגרמני הודיעו כי הצליחו למצוא מספר שיטות לפריצת מנגנון האבטחה של BitLocker. כל הפריצות שתוארו היו קשורות בהתערבות אנושית על המחשב ובטעויות של המשתמש אשר הוכיחו שוב, כי ה"חוליה החלשה" בתחום אבטחת המידע נעוצה בנו - המשתמשים.

במסמך המפרט את המחקר הובאו מספר שיטות לשבירת מנגנון ההצפנה. בכל אחת מהשיטות קיימת התערבות של בעל המחשב, או מישהו בעל הרשאות Admin אשר יקיש את קוד ה-PIN על מנת לפתוח את מנגנון ההצפנה. שיטה אחת מתארת את הצורך להתחבר פיזית למחשב, ולשתול בו קובץ (מנגנון הזדהות מזוייף) ל-BitLocker. שיטה נוספת מדברת על האפשרות להתעסק עם רכיב החומרה TPM יחד עם השתלת רכיב האזנה (Sniffing) על המערכת. באמצעות פעולה זו מנגנון ההצפנה ייחסם ויצטרכו לעשות Recovery אשר במהלכו יוכנס הקוד מנהל וכך ה-Sniffing יקלוט את הקוד ויעביר אותו ברשת בצורה מרוחקת לפורץ.

סרטון המתאר הדגמה חלקית של שיטות הפריצה:

http://testlab.sit.fraunhofer.de/content/output/project_results/bitlocker_skimming/bitlockervideo.php?s=2

הנקודה החשובה בעניין, היא שכל הפריצות המתוארות כאן דורשות התערבות פיזית של מנהל המחשב שיצטרך להכניס את קוד ה-PIN או קוד ה-RECOVERY על מנת שהפורץ יוכל לקלוט את הקוד - ולהשתמש בו לאחר מכן להשגת המידע הרצוי.

פתרון לבעיה: הדרך היחידה להתמודד עם "החוליה החלשה" (המשתמשים), היא הסבר לכלל המשתמשים על חשיבות השימוש בתוכנה, ועל חשיבות המידע הרגיש הנמצא, ועל תהליכי עבודה תקינים ומסודרים - אשר יגרמו בסופו של דבר לעליה מסיימת ברמת האבטחה ברמת המשתמש, ובכך, למנוע פגיעות אשר יכולות להיגרם כתוצאה משימוש לא "חכם" במערכת.

בנוסף, ישנה חברה בשם Passware, המאפשרת לכל המעוניין את פריצת ה- BitLocker. מה הכוונה? חברת Passware הינה חברה המייצרת תוכנות לפענוח הצפנות ושחזור סיסמאות של תוכנות נפוצות להצפנה. לאחרונה השיקה החברה את מוצר הדגל שלה- Passware Kit Forencsic 9.5. Passware Kit Forencsic 9.5 היא התוכנה המסחרית הראשונה אשר יכולה לזהות מפתחות הצפנה בכוננים שהוצפנו באמצעות ה-BitLocker, ויכולה לפענח את המידע הנמצא בתוכם. עלותה נאמדת ב- 800$. תוכנה זו מסוגלת לסרוק כל כונן קשיח, לזהות את סוגי הקבצים המוצפנים הנמצאים עליו, ולפענח את ההגנות שלהם, על ידי שימוש באלגוריתמים מתקדמים לפענוח ושחזור מידע. ישנה גרסה ניידת של התוכנה הפועלת מכונן USB , אשר סורקת ומשחזרת סיסמאות של קבצים מוצפנים, ללא פגיעה במחשב עליו היא מופעלת. באמצעות תוכנה זו ניתן להוציא את המידע ללא כל בעיה, ואף מבלי שבעל המחשב ידע לזהות

האם מישהו נגע והוציא מידע (בדיקת LOGS או כל תוכנה אחרת). הסיבה לכך היא בגלל שהכול מבוצע בסביבה וירטואלית.

סיכום BitLocker

- BitLocker הינה תוכנה להצפנת דיסק מלאה.
- בעזרת תוכנה זו יכול משתמש להגן על נתונים בעזרת הצפנה.
- כברירת מחדל התוכנה משתמשת באלגוריתם AES-CBC במפתח הצפנה באורך של 128Bit עם Diffuser.
- BitLocker תומכת במפתחות הצפנה באורך של 128Bit ו – 256Bit עם או בלי שכבת Diffuser.
- הגדרות אלו ניתנות לשינוי בעזרת ה - Local Group Policy Editor.
- הצפנת קבצים זו משתמשת במפתח סימטרי.

ההבדלים השונים בין הצפנת EFS להצפנת-BitLocker

BitLocker מצפין את כל הקבצים האישיים וקבצי המערכת הנמצאים בכונן של מערכת ההפעלה, בכוננים קבועים או כוננים נשלפים (USB), להבדיל מ-EFS - המצפינה קבצים ותיקיות בנפרד ואינה מצפינה את הכונן בצורה מלאה.

BitLocker אינו תלוי בחשבונות המשתמשים: BitLocker פועל או מבוטל, עבור כל המשתמשים או הקבוצות, להבדיל מ-EFS אשר מצפין קבצים בהתבסס על חשבון המשתמש המשויך אליו. כל אחד מהמשתמשים יכול להצפין את הקבצים שלו באופן עצמאי.

EFS אינו עושה שימוש ברכיב חומרה מסוים, להבדיל מ-BitLocker המשתמש ב- Trusted Platform Module (TPM) שבב מיוחד הקיים במחשבים רבים שתומך בתכונות אבטחה מתקדמות, כדי להצפין את כונן מערכת ההפעלה.

ב-BitLocker עליך להיות מנהל מערכת על מנת לגשת ולהפעיל או לבטל את ההצפנה, בעוד ב-EFS כל משתמש יכול להצפין את המידע שהוא חפץ בו.

הערה: אין מניעה להשתמש בשני סוגי ההצפנה. הצפנת EFS שומרת את מפתחות ההצפנה במחשב, כאשר הצפנת BitLocker יכולה לעזור בשמירה על מפתחות אלו באמצעות מניעת אתחול של המערכת (והפעלת המערכת בעזרת אחד מהמנגנונים שבהם BitLocker משתמש כגון TPM, USB או אימות בעזרת קוד PIN).

דברי סיום

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

אנחנו מחפשים כתבים, מאיירים, עורכים (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי) ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין Digital Whisper – צרו קשר!

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

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

www.DigitalWhisper.co.il

הגליון הבא ייצא ביום האחרון של מרץ 2010.

אפיק קסטיאל,

ניר אדר,

28/2/2010