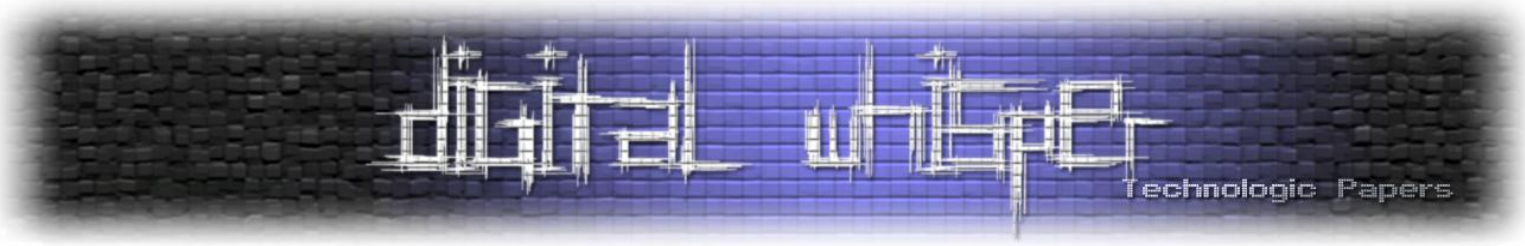


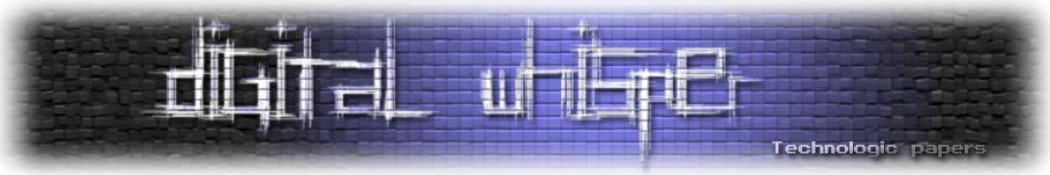
Digital Whisper

גליון 3, דצמבר 2009

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורך:	ניר אדר
כתבים:	אפיק קסטיאל, הלל חימוביץ', ניר אדר, סולימני יגאל

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



דבר העורכים

ברוכים הבאים לגליון השלישי 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.

מאז יציאתו של הגליון השני היו מספר אירועים בסצינה העולמית שרצינו לציין, החודש הסתובבה שמועה ש-Str0ke, הבחור שהגיש לנו את הפרוייקט Milw0rm מת עקב-שקר-כלשהו, מה שהתברר כטעות, אך הבחור הקפיא את הפרוייקט שלו, ונכון לרגעים אלה החבר'ה של Muts עובדים קשה כדי לאחד את הכל-אפשר לראות תוצאות ב: Exploit-db.com. בנוסף, HD Moore שעומד בראש הפרוייקט Metasploit התחיל לעבוד החודש כ-CSO בחברת אבטחה בשם Rapid7, חברה שמתעסקת ב-Penetration Testing וב-Vulnerability Management, נראה איך הדבר ישפיע על הפרוייקט. (שכבר הספיק לצאת בגירסאתו ה-3.3)

לגליון הנוכחי תרמו HLL שזה כבר המאמר השני שלו במגזין, הפעם מאמר מצוין על עולם ה-PKI, ויגאל סולימני שכתב מאמר מעניין מאוד על ארכיטקטורת ה-IPSEC, שני מאמרים מצויינים שאנחנו מקווים שתאהבו. תודה רבה לשניהם!

אנו מחפשים אנשים נוספים שרוצים לעזור לנו ביצירת הגליונות. אם ברצונכם לנסות ולגרום לשינוי בקהילה שלנו ולהפיץ מידע איכותי בעברית - פנו אלינו! כתיבת מאמרים ל-Digital Whisper כוללת עבודה בתנאיי לחץ מחפירים שלי והתמודדות עם עריכות אין-סופיות של ניר אך הכי חשוב- עבודה עם אנשים שרוצים להפוך את האינטרנט הישראלי למעניין יותר. אם אתם חושבים שאתם מתאימים- תרגישו חופשי לשלוח אלינו דוא"ל.

קריאה מהנה!

אפיק קסטיאל

ניר אדר

תוכן עניינים

PKI, תעודות, כרטיסים חכמים ומה שבינהם

מאת הלל חימוביץ' (HLL)

הקדמה

אחד הנושאים המעניינים ביותר שקיימים, לדעתי, הוא נושא אימות והצפנת הנתונים, והפתרונות המוצגים לכל סוגיה או פרצה בתהליך זה. כבר שנים רבות שקיימים פתרונות מקיפים לזיהוי, אימות, חתימה והצפנה של נתונים ושל גורמים, בהם ניתן להשתמש ללא חשש שהמקור זויף או שונה.

במאמר זה אציג לכם מספר רעיונות מרכזיים הקשורים ב'ארכיטקטורת מפתח ציבורי' או - Public Key Infrastructure.

בעיה מספר 1 - חיסיון המידע - Data Confidentiality

תיאור הבעיה

בעולם ההצפנה ישנם שני חברים טובים מאוד ומוכרים מאוד, אליס ובוב. במקרה שלנו, אליס ובוב רוצים לדבר אחד עם השני במסנג'ר (אודיגו, IRC, מה שתבחרו) ולתכנן לחברם דני מסיבת הפתעה. לרוע מזלם של בוב ואליס, דני עובד בספקית השירות של בוב והוא יכול לראות את כל הנתונים היוצאים ונכנסים בשיחה בין בוב ואליס. בוב ואליס מחפשים פתרון איך למנוע מדני להבין מה נאמר בשיחה שלהם.

לבעיה בסיסית זו וריאציות רבות:

- מצב בו בוב מחובר ברשת אלחוטית ציבורית.
- בחברה שבה עובדת אליס, יש רכיב שמצותת לתקשורת בין המחשב שלה לאינטרנט (במקור תפקידו נועד למנוע ממידע רגיש לצאת, או ממזיקים - להכנס).
- אליס חוששת מרשת ה-Ethernet כי היא יודעת שדני שעובד בעמדה לידה, יודע לבצע מתקפת ARP Poisoning \ Man In The-Middle.

בעית חיסיון המידע: כיצד נמנע מגורם זר שמצותת לקו או לערוץ הפומבי, לצפות במידע שעובר באותו הקו/הערוץ.

פתרונות לבעית חסיון המידע

באופן בסיסי הפתרון לבעית חסיון המידע הוא הצפנת המידע או הסתרתו בצורה כזו שדני לא ידע מה המפתח על מנת שיוכל לקרוא את הנתונים מאליס או מבוב.

פתרון ראשון הוא שימוש בצופן סימטרי. **צופן סימטרי** הוא אלגוריתם הצפנה המשתמש במפתח אחד הן להצפנה והן לפענוח הטקסט. הבעיה: כל הצפנה סימטרית תחייב הסכמה מוקדמת של אליס ובוב על המפתח - ולא, יהיו חייבים בוב/אליס לציינו בתקשורת, ובהנחה שדני מאזין לתקשורת, לדני יהיה את המפתח הנ"ל והוא יוכל לקרוא ללא כל בעיה את השיחה ביניהם באמצעות המפתח שאיתר מהתקשורת.

אליס ובוב צריכים להחליט ביניהם בצורה מסוימת על מפתח משותף מבלי שדני יוכל לדעת מהו. שני חוקרים, Whitfield Diffie ו-Martin Hellman, הם הראשונים שהציעו פתרון לבעיה זו.

לא נכנס לעומק למתמטיקה מאחורי הפתרון, אך נציג את העקרון של הפתרון:

1. שני הצדדים בוחרים בסיס משותף g
2. כל צד בוחר מעריך משלו לבסיס זה, a ו-b.
   a. אליס מגרילה את a ושולחת את g^a לבוב
   b. בוב מגריל את b ושולח את g^b לאליס.
3. כל צד מעלה בחזקה את הנתון שנשלח אליו במספר שהוא הגריל, ועל כן אליס מקבלת את הערך (g^b)^a, ובוב מקבל את הערך (g^a)^b. כאשר לבסוף כמובן שני הערכים האלה זהים לפי חוקי חזקות. כעת בוב ואליס יכולים לנהל ערוץ תקשורת מאובטח באמצעות מפתח **סימטרי** זה.

הסיבה שדני לא יכול לעלות על מפתח זה מכיוון שהוא רואה רק את התוצאות הסופיות של החישובים שבוב שולח לאליס, ואליס לבוב. דני יכול לעלות על המעריכים אולם לצורך כך הוא יזדקק לכוח מחשוב עצום לבצע פעולת log (ההופכית לחזקה) עבור מעריך ראשוני גדול מאוד.

לבעית חסיון המידע, ובפרט, למציאת מפתח משותף על גבי ערוץ פומבי, קיימות שיטות נוספות בהן ניתן להשתמש. המתעניינים יכולים להעמיק בשיטה בשם Quantum cryptography. המתעניינים בהרחבה נוספת בנושא שיטת Diffie-Hellman יכולים לקרוא עליה כאן: Diffie-Hellman_key_exchange

ומה קורה במקרה של תקשורת לא ישירה?

נניח ואליס ובוב לא מדברים במסנג'ר, אלא במייל, כאשר כל אחד בודק את המייל שלו פעם ביום. אם הם ישתמשו באלגוריתם Diffie-Hellman, היה צורך בשלושה ימים רק כדי להקים את ערוץ התקשורת. במקרים בהם אין תקשורת ישירה, ובמצבים נוספים אחרים, ישנו פתרון אחר לשליחת הודעות סודיות על גבי ערוץ פומבי, והוא נקרא הצפנה א-סימטרית. דוגמה להצפנה כזו היא RSA.

הצפנה א-סימטרית משתמשת בכלים מתמטיים הלקוחים מתורת המספרים לייצור של זוג מפתחות, כאשר האחד משתמש להצפנת הנתונים, והשני לקריאתם, כאשר:

$$C = E(K_{pub}, P)$$
$$P = D(K_{pri}, C)$$

- C - טקסט מוצפן
- P - טקסט קריא
- Kpub - מפתח ציבורי של נמען ההודעה
- Kpri - מפתח פרטי של נמען ההודעה

ובעברית פשוטה, נאמר שההצפנה מתבצעת עם מפתח אחד - המפתח הציבורי, והפענוח של הנתונים מתבצע עם מפתח אחר - המפתח הפרטי.

בוב ואליס כל אחד מייצר להם זוג מפתחות כזה, וכל אחד שולח את המפתח הציבורי שלו לצד השני. כאשר אליס רוצה להגיד לבוב מה היא מתכננת במסיבת ההפתעה, היא מצפינה את ההודעה **במיוחד עבור בוב** עם המפתח הציבורי שבוב שלח לה, ושולחת אותה לדרכה.

כעת, האדם היחיד שיכול לקרוא את ההודעה הוא בוב מכיוון שבוב הוא היחידי שמחזיק את המפתח שמסוגל לפתוח את ההצפנה. כאשר בוב רוצה לשלוח הודעה חזרה לאליס הוא מצפין את ההודעה **במיוחד עבור אליס** עם המפתח הציבורי שאליס שלחה לו, ושולח אותה לדרכה.

בעיות מספר 2 ו-3 - מהימנות המידע ושלמות המידע - Data Authenticity and Integrity

בפתרון שהצגנו מעלה, בין אם באלגוריתם Diffie-Hellman ובין עם הצפנה א-סימטרית, יש בעיה נוספת. כאשר בוב מדבר עם אליס הוא אינו יכול לוודא שדני אינו שינה את תוכן ההודעה ששלחה אליס. לדוגמא, אם אליס שלחה הודעה שהמסיבה תתקיים בשעה 4, אך דני ישנה את תוכן ההודעה ויציין שזו שעה 8, בוב לא ידע שההודעה שונתה.

בעיה זו נקראת בעית **שלמות המידע** - כיצד ניתן לוודא שהמידע שנשלח אכן לא שונה בדרך (או נפגם)?

בנוסף, בוב הוא בחור מאוד ספקן. כאשר הוא מקבל הודעה - איך הוא יכול לדעת שזו אליס שבאמת שלחה את ההודעה? אולי זה בעצם דני שמתחזה לאליס, ובמצב כזה בוב יחשוב שהוא מקים ערוץ תקשורת מאובטח/לא מאובטח ישיר מול אליס, אבל בפועל הוא משוחח עם דני המתחזה לאליס. דני ישלח לבוב את כל מה שהוא צריך בשביל שבוב יאמין שדני הוא אליס, ויקים עם בוב תקשורת באותה דרך שאליס הייתה עושה זו.

בעיה זו נקראת בעית **מהימנות המידע** - כיצד ניתן לוודא שהמידע שנשלח אכן נשלח מהמקור שהוא מתיימר להיות?

לצורך הבהרת העניין נעזוב לרגע בצד את ההצפנה של ההודעה. נניח שקיימת הודעה T שבוב רוצה לשלוח לאליס. פתרון נאיבי יעבוד כך: בוב כותב את ההודעה במלואה, ובסופה מציב נתון בן 10 בתים שהוא הסכום הכולל של כל התווים בהודעה. כאשר אליס מקבלת את ההודעה, היא יכולה לוודא שאינה שונתה באמצעות ביצוע חישוב זהה לבתים הנמצאים בהודעה והשוואה של התוצאה שיצאה לה, למה ששלח בוב.

האומנם? למרות שבוב מנסה לעזור לאליס לוודא את שלמות ההודעה, דני הוא בחור חכם. דני רוצה לשנות את ההודעה. הוא רואה את הערך הנקוב ב-10 הבתים ששלח בוב, ומשנה את ההודעה (או כותב הודעה חדשה) בצורה כזו שסכום הבתים בהודעה יהיה זהה לערך שנקב בו בוב. כאשר אליס תנסה לוודא את שלמות ההודעה (המפוברקת מדני) של בוב, היא לא תגלה שום פגם. דני מסוגל לעשות את זה, מכיוון שחיבור היא פעולה הפיכה בצורה פשוטה, וכוח המחשוב הנדרש לחשב אותה הוא נמוך למדי.

אם כך, הוספת הסכום הכולל של תווי ההודעה לגוף ההודעה אינו פתרון לבעיה שלנו. יש צורך בטכניקה מתוחכמת יותר.

Hash או Message Digest הוא אלגוריתם מתמטי שבעזרתו ניתן לייצג הודעה שלמה במספר מועט של בתים, בצורה כזו ש:

- קל יחסית לחשב את ה-Hash עבור הודעה מסוימת.
- בהינתן Hash מסוים, בלתי אפשרי לחשב את ההודעה המקורית בזמן סביר.
- לא ניתן לשנות את ההודעה בלי לשנות את ה-Hash שלה, אולם, בהינתן זמן ארוך מאוד, ניתן לשנות את ההודעה בצורה כזו שתניב Hash זהה.
- לא ניתן למצוא שתי הודעות שיש להן אותו Hash, בזמן סביר (דומה לסעיף הקודם).

כל אחד מהסעיפים הנ"ל הוא **<u>אפשרי לביצוע</u>**, אך לא במסגרת זמן סבירה. מכאן אנחנו יוצרים פעולת Hash שמכוונת להיות פעולה חד כיוונית שלא ניתן להפוך אותה. עבור אלגוריתם Hash אידיאלי - תמיד נצטרך לבצע מתקפת Brute-Force על כל הקומבינציות האפשריות כדי למצוא הודעה עם Hash זהה, או כדי למצוא הודעה מתוך Hash נתון.

> הרחבה: כאשר אומרים שאלגוריתם Hash נפרץ, לא הכוונה היא שאפשר להפוך אותו. לדוגמה כאשר MD5 נפרץ, הכוונה היתה לא להשיג את ההודעה המקורית מתוך ה-Hash , אלא - קיצור זמן משמעותי של מציאת הודעה שונה בעלת אותו Hash כאשר ההודעה המקורית זמינה (נקרא גם Hash Collision).

מנגנוני ה-Hash חיסלו את האפשרות שדני יוכל ליצור הודעה שונה, אך בעלת 'סכום תווים' זהה. כפי שבטח הספקתם לשים לב, 'סכום התווים בהודעה' הוא גם סוג של Message Digest, אך איננו עומד בקריטריונים שכתובים מעלה.

אבל מה הבעיה פשוט לשנות את ה-Hash המצוין בהודעה?

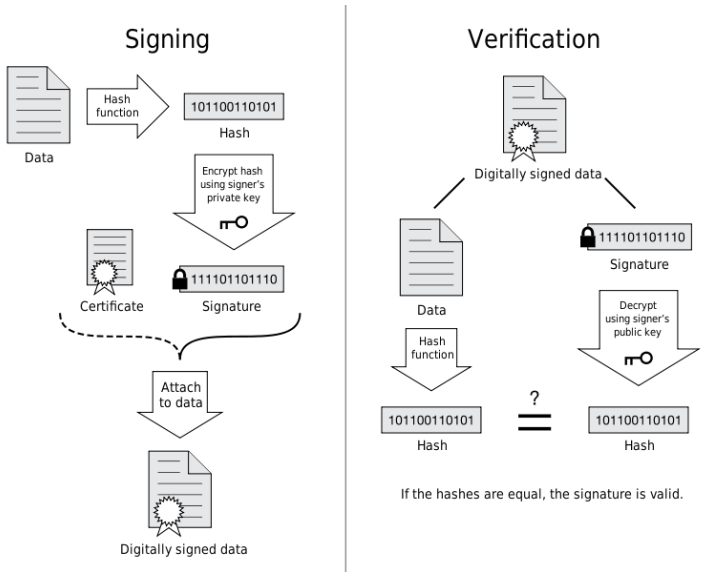
גם לאחר שימוש ב-Hash יכול דני פשוט **להמציא הודעה חדשה, וגם לשנות לה את ה-Hash**. כך אליס לעולם לא תוכל לשים לב שההודעה שונתה, למרות שאלגוריתם ה-Hash הוא בטוח לחלוטין.

איך יכולה אליס לוודא שה-Hash שנשלח לה אכן הגיע מבוב ולא מדני? - כאן נכנסת בעית **מהימנות המידע**. כדי לפתור בעיה זו ניצלו את מנגנוני ההצפנה **הא-סימטרית** כפי שהוסברה קודם לביצוע של פעולה חדשה נוצצת בתחום הקריפטוגרפיה ואבטחת המידע, והיא נקראת **חתימה דיגיטלית.**

חתימה דיגיטלית - Digital Signature

חתימה דיגיטלית היא סוג של 'הצפנה הפוכה' שעובדת רק על ה-Hash של ההודעה, משמע - המידע עובר פעולת Hashing ואותו ה-Hash **<u>נחתם</u>** עם המפתח הפרטי של שולח ההודעה, מקבל או מקבלי ההודעה באשר הם יוכלו לאמת את ההודעה, את מקוריותה ואת שלמותה ע"י בדיקת החתימה - ביצוע Hash בעצמם להודעה, ולהשתמש במפתח הציבורי של שולח ההודעה יחד עם החתימה הדיגיטלית ע"מ לאמת את ה-Hash הנ"ל.

פעולת החתימה הינה בעצם פעולת ההצפנה של ה-Hash באמצעות המפתח הפרטי של המוען, היא אינה נקראת הצפנה מכיוון שהיא לא נועדה להסתיר את הנתונים מאף אחד, אלא רק לאמת את אותנטיות הנתונים, וכי כל אחד יכול לפתוח את הצפנה זו כי המפתח הציבורי הוא פומבי לכל.



[נלקח מ'ויקיפדיה']

סיכום ביניים

אז מה היה לנו עד כה? אליס ובוב רוצים לנהל שיחה פרטית. לצורך כך, בתור התחלה, הם רוצים להיות בטוחים עם מי הם מדברים. לפני השיחה הם מתחילים באימות אחד את השני באמצעות חתימה דיגיטלית. לאחר מכן, על בסיס האמון הזה אליס ובוב מבצעים את החלפת המפתחות (ניתן לראות זו בצורה כזו שתוצאות החישוב של בוב ותוצאת החישוב של אליס בתהליך ה-DH כאשר הם שולחים אחד לשני - חתומות דיגיטלית באמצעות המפתח הפרטי של הצד ששלח את תוצאת החישוב).

אחרי שהבנו מה מהות החתימה הדיגיטלית, מה הרעיון מאחורי אלגוריתם Diffie-Hellman ומהי הצפנה א-סימטרית נמשיך בהצגת בעיות נוספות.

הפתרונות לבעיות שהוצגו קודם יפות ואסטטיות, אולם אינן מושלמות, הפתרונות האלו עדיין לא פותרים לנו את הבעיה הראשונית - בעיית חלוקת המפתחות.

הצגנו פתרון לבעיית המהימנות, אך אליה וקוץ בה, חדי השכל ישימו לב, שע"י פתרון החתימה הדיגיטלית הבעיה לא נפתרה, אלא רק קיבלה פנים אחרות. השאלה המתבקשת היא **כיצד נחליף מפתחות פומביים, ושהצד השני ידע שאנחנו, זה אנחנו?** או אם להשליך את הסיטואציה על אליס ובוב: כיצד בוב יכול לוודא שהמפתח הציבורי שנשלח לו כביכול ע"י אליס, הוא באמת של אליס. כיצד תוודא אליס שהמפתח הציבורי שבוב שולח לה הוא באמת ובתמים של בוב, ולא של גורם זדוני המיירט את התקשורת?

אמון - Trust

במדינת ישראל הדרך שבה מזוהים אנשים לראשונה הוא לפי תעודות הזהות המנוילנות שלהם (לפחות נכון לאוקטובר 09). נניח לרגע **שלא היה ניתן לזייף** את תעודת הזהות, ושהיא יכולה להיות **מיוצרת רק ע"י משרד הפנים**. האם כאשר מישהו יגיע אליכם, ויטען שהוא האדם הנקוב בתעודת הזהות שהוא מביא עימו, האם תאמינו לו? ואם כן - מדוע?

התשובה לשאלה מדוע אתם מאמינים לו היא פשוט מאוד - **אתם סומכים על הגורם שהנפיק לו את התעודה** - אתם סומכים על משרד הפנים.

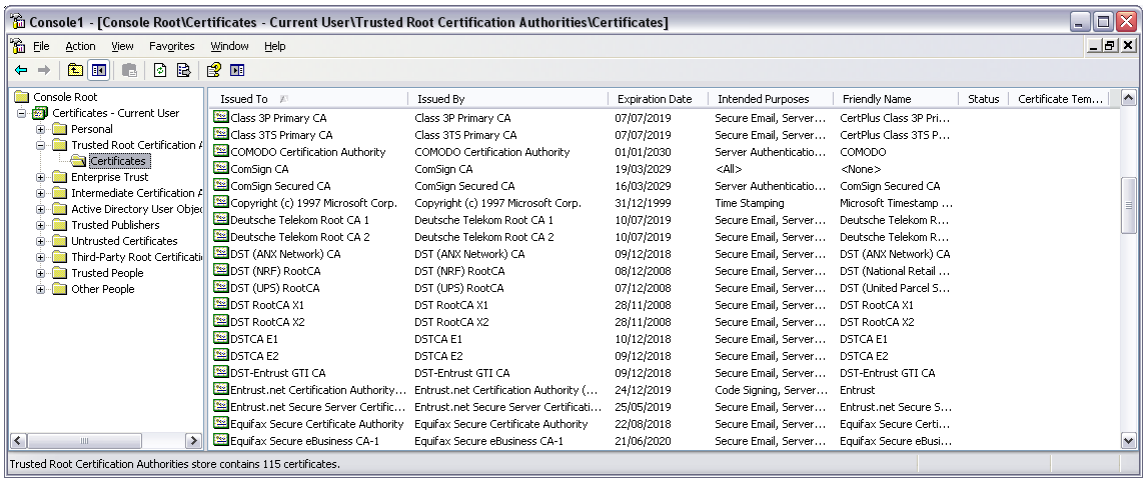
אם רק משרד הפנים יכול להנפיק תעודות זהות, ואם אתם סומכים על משרד הפנים שיעשה את העבודה הנדרשת בשביל לזהות מי זה מי, אתם יכולים **<u>להיות בטוחים ב-100% שהאדם העומד מולכם הוא מי שהוא מצהיר שהוא</u>**.

בעולם ה-PKI, קיים אותו רעיון, קיים גורם צד שלישי, Trusted Third-Party (להלן 'משרד הפנים') שהוא מפיק תעודת זהות דיגיטלית לגורם המגיע אליו ומבקש להנפיק לו תעודה, אותו גורם אחראי לוודא שהאדם שהגיע אליו הוא אכן הוא ומפיק לו תעודה.

תעודה דיגיטלית - Digital Certificate

תעודה דיגיטלית אינה ניתנת לזיוף בזמן סביר בגלל השימוש במתמטיקה שהוצגה בפרק הראשון. הגורם שהנפיק את התעודה, גם לו חייבת להיות תעודה כזו, וגם לזה שהנפיק לו את התעודה יש תעודה משל עצמו, וכך הלאה עד אשר מדובר בשורש העץ, באבא של כל התעודות -Root Certificate Authority.

תעודת האבא **לא ניתנת לאימות** ללא מידע מוקדם, על כן, תעודות Root CA צריכות להגיע **בדרך כלשהי** למחשבים השותפים לאותו האמון, בין אם באמצעות Disk-On-Key, Floppy, Group Policy, הורדה והתקנה ידנית (אך מסוכנת מאוד) מהאינטרנט. הדרך שמקובלת היום באינטרנט היא **שהתעודות של ה-Root CAs העולמיים מגיעות מובנות עם מערכת ההפעלה של המשתמשים.**



[האוסף הדיפולטיבי של Trusted Root CA המגיע עם Windows XP 64bit.]

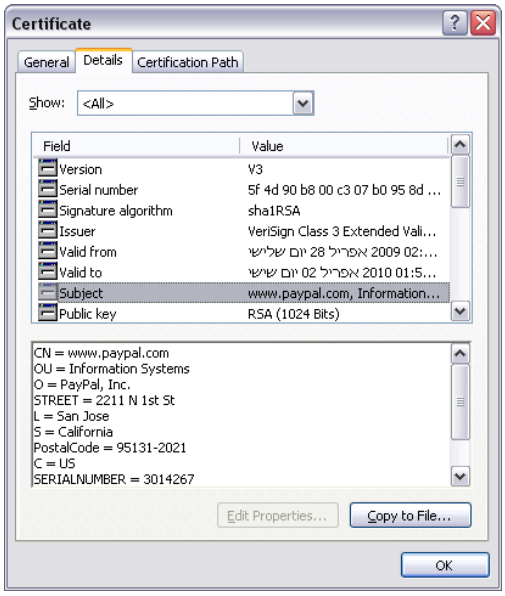
אז מהי תעודה דיגיטלית בעצם?

תעודה דיגיטלית הינה אוסף של פרמטרים ונתונים החתומים דיגיטלית ע"י מנפיק התעודה. בין הפרמטרים האלה חובה להיות קיימים **שם בעל התעודה** וכן **המפתח הציבורי שלו**.

אם שם בעל התעודה מופיע יחד עם המפתח הציבורי של בעל התעודה, והנתונים האלה **יחדיו** חתומים דיגיטלית ע"י גורם אחר שעליו אנו סומכים, ויש לנו את המפתח הציבורי שלו לאימות החתימה. **בהכרח** שהמפתח הציבורי המצוין שייך לאדם המצוין (ראו בעיית שלמות מידע ומהימנות מידע).

אם אליס תקבל תעודה כזו מבוב, היא יכולה להיות בטוחה שהמפתח הציבורי של בוב הוא הנקוב בתעודה. גם אם לא בוב, אלא מישהו אחר ישלח את התעודה, המפתח הציבורי הוא עדיין של בוב ורק בוב יכול לקרוא הודעות שהוצפנו עם המפתח הציבורי המצוין. רק בוב יכול להפיק חתימות דיגיטליות שיאומתו באמצעות המפתח המצוין.

אם גורם אחר ינסה לשנות את אחד הפרמטרים - הוא לא יוכל לעשות זאת בגלל שאז החתימה לא תהייה תקפה. **החתימה הדיגיטלית בתעודה 'שומרת' על שלמות ואמינות הנתונים המופיעים בה ומצליבה את שם בעל התעודה עם המפתח הפומבי שלו**.



[חלק מפרטי תעודה דיגיטלית לדוגמא]

Certificate Revocation List

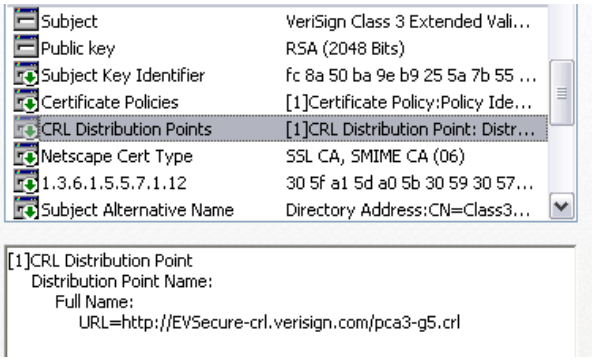
נאמר לרגע שאתם עובדים בארגון שבו יש 1000 עובדים וכולם עובדים מהבית עם תעודות דיגיטליות שהונפקו ע"י החברה, ומותקנות על המחשבים הביתיים של העובדים. בתעודה הזו משתמשים גם בכדי לזהות את העובדים מול השרת וגם את העובדים מול העובדים האחרים (השותפים לאותו אמון - Trust). יום אחד נפרץ המחשב האישי של אחד העובדים ונגנבה התעודה, בעל התעודה דיווח לחברה, והנפיקה לו תעודה חדשה, אך נשאלת השאלה - כיצד תבטל החברה את האפשרות לשימוש בתעודה שנגנבה? יש צורך במנגנון תשתיתי שיאפשר ל-CA לבטל תעודות דיגיטליות ברגע שנגנבו, פגו או במקרים אחרים.

על כן, כחלק מהפתרון הכללי, נהגה מנגנון ה-CRL. מנגנון ה-CRL הוא חלק מתשתית ה-PKI, והוא מאפשר ביטול תעודה ע"י המנפיק בצורה כזו שברגע הביטול - או בזמן הכי קצר האפשרי, ידעו משתמשים ושרתים אחרים (או כל גורם המשתתף באמון) שלא לאשר תעודה זו.

בתעודה דיגיטלית של Certificate Authority המנפיק תעודות למשתמשים/שרתים קיים פרמטר נוסף אופציונאלי, הפרמטר הזה מכיל כתובת, URL, או כתובת שיתוף (UNC Path) לקובץ CRL. שימו לב שגם הפרמטר הזה חתום בתעודה אזי לא ניתן לשנותו מבלי לבטל את החתימה הדיגיטלית. קובץ ה-CRL הינו רשימה של מספרי תעודות שבוטלו ע"י מנפיק התעודה, להחלטתו. לווידוא שלמות ומהימנות הקובץ, גם קובץ זה חתום דיגיטלית ע"י ה-CA.

שרת שעליו מאוחסנים CRL-ים , ותעודות של שרת CA, נקרא שרת Publish.

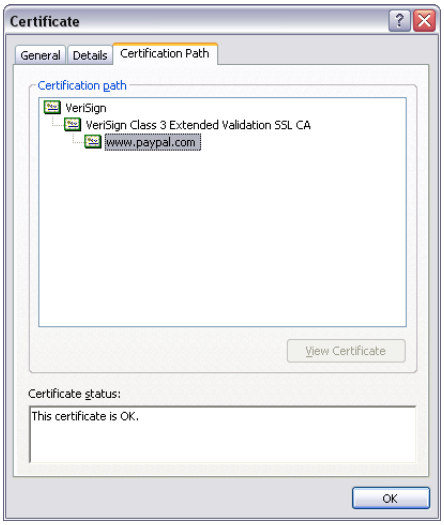
בנוסף, בתעודה של ה-CA, ניתן לציין מרווח זמן מקסימלי לעדכון של CRL-ים. לדוגמא, שרת CA יכול לציין בתעודה שלו שעדכון CRL חייב לצאת כל 48 שעות. אם משתמש בודק תעודה של משתמש אחר החתומה ע"י ה-CRL הנ"ל, חייבת להיות למשתמש שמוודא את התעודה גישה לקבצי CRL-ים "טריים" שנוצרו ב-48 שעות האחרונות, אם לא, המשתמש **אינו יכול לוודא מהימנות של התעודה** ללא גישה לקובצי ה-CRL.



[כתובת ל-CRL של VeriSign ICA מצוינת בתעודתו.]

Certificate Path/Chain

בארגונים גדולים (וגם באינטרנט), מקובל לא לרכז את השליטה ב-CA אחד, מפאת חלוקת הרשאות, הפרדת רשויות, חלוקת עומס, אזורי מיקום שונים ויחידות ארגוניות שונות. אולם עדיין חייב להיות רשות אחת שמפעילה את שאר הרשויות. הרשות העליונה ביותר נקראת Root Certificate Authority, ויש לה הרשאה להנפיק תעודות ל-CA אחרים.



[דוגמה לשרשרת תעודות עבור Paypal]

בארגונים גדולים מאוד, מקובלת החלוקה ל-לא יותר מ-3 רמות של CAs:

- Root Certificate Authority - השרת הראשי, וקיים אחד כזה בכל ארגון, התעודה שלו חייבת להגיע באופן ידני למחשבים המשתתפים באמון.
- Intermediate Certificate Authority - שרת CA 'ביניים', משמש להורדת עומס מה-Root CA.
- Operational Certificate Authority - השרת האחרון בשרשרת שלרוב הוא שמנפיק תעודות למשתמשים, מחשבים או שרתים. לרוב, השרת הזה מחובר לרשת הארגונית או לאינטרנט.

להלן רשימה אפשרית לפרמטרים מנהלתיים שמאומתים באמצעות ה-PKI.

פרמטר	ROOT CA	ICA	OCA	תעודת משתמש
יכול להנפיק תעודות ל-CA אחרים	V	V	X	X
זמן חלוקת CRL-ים מקסימלי	שבועיים - חודש	שבוע	יומיים	אין
קישוריות לרשת	X	X	V (גם לא תמיד)	תלוי אם Client או Certificate Token
סיכון במידה שנפרץ	מוחלט, כל הרשת בסכנה	הסכנה הינה עד לחודש, לאחר מכן ימנע החוסר ב-CRL-ים אימות נוסף	הסכנה הינה עד לשבוע, לאחר מכן ימנע החוסר ב-CRL-ים אימות נוסף	תוקף יכול להתחזות למשתמש עד לטווח של יומיים
אבטחה	מאוישת גבוהה מאוד, 24/7	גבוהה	בינונית	ברמת המשתמש, או ברמת החומרה (ראו המשך)

למעשה, מעבר לזה שה-RCA וה-ICA אינם מחוברים לרשת, הם לרוב בכלל כבויים - כי פשוט אין להם סיבה להיות פועלים.

כיצד לא נאפשר את שכפולו של המפתח הפרטי?

כאשר שרתי ה- CA מאובטחים עדיין לא פתרנו את נושא האבטחה הפיזית. הגנו על מספר מצומצם של שרתי CA באבטחה פיזית, אבל מה קורה אם מאות ואלפי המשתמשים שקיימים בארגון? אי אפשר לצרף לכל משתמש 2 שומרי ראש, נכון?

אין לנו שליטה מספקת על האבטחה הפיזית כאשר מדובר במשתמשים, אולם פותחה שיטה טכנולוגית בכדי להגן על המפתח הפרטי של המשתמשים גם במקרה ונגנב. שיטה זו הינה שיטת 'כרטיס חכם'.

התקן נייד 'חכם'

כדי להגן פיזית על המפתח הפרטי חייבים סוג של התקן פיזי שניתן לחברו למחשב ובעזרתו לעשות שימוש במפתח הפרטי. בנוסף, המכונה שאנחנו מחברים אליה את ההתקן יכולה לגנוב לנו את המפתח, מה שמחייב אותנו לעשות את כל פעולות החישוב עם המפתח הפרטי אך ורק **על ההתקן**, כך שרק את תוצאת החישוב נשלח למחשב. המפתח הפרטי לעולם לא ייצא מגבולות ההתקן.

אז הלכו וייצרו התקן כזה, לקחו מעבד קטן, שמו מאחוריו זיכרון (מאחוריו כלומר שהוא מגן על הזיכרון ואי אפשר לקרוא את הזיכרון המוגן ישירות מההתקן) ובזכרון איכסנו את המפתח הפרטי. את כל הפעולות שדורשות מפתח פרטי (בין אם מדובר בהחתמת מידע ובין אם מדובר בפענוח מידע שהוצפן בעזרת המפתח הציבורי) יעשו על המעבד של ההתקן. הוא יכול להגיע בצורה של כרטיס אשראי שנקרא באמצעות קורא, או בצורה של USB Token שבכביכול הוא גם הקורא וגם הכרטיס יחד.

Factors of Authentication

אם ההתקן נגנב אזי מי שגונב את ה-Token לא יכול להוציא את המפתח הפרטי, אבל יש לו את הכרטיס, והוא עדיין יכול להשתמש בו כאוות נפשו.

כדי לפתור את הבעיה הזו הגנו על השימוש במפתח הפרטי באימות באמצעות סיסמה. לפני ביצוע פעולה **על הכרטיס שמשתמשת במפתח הפרטי** יש צורך להזדהות מולו עם סיסמה.

כלומר, כדי שתוקף יוכל להתחזות למישהו אחר, עליו **לגנוב** את ה-Token ולדאוג להשיג את הסיסמה בדרך כלשהי. על כן שיטת הזדהות המשתמשת ב-Token ובסיסמה, קרויה 2-Factor Authentication, מכיוון שההזדהות משתמשת בשתי שיטות:

1. What You Have (Token \ SmartCard)
2. What you know (Pin Code \ Password)

קיים פקטור הזדהות נוסף אחר והוא:

3. What you are.

כלומר, מה אתה - הכוונה הוא לזיהוי ביומטרי, בין אם באמצעות רשתית, דגימת DNA, טביעת אצבע או שבלונת רגל (☺). הזדהות עם 'PIN ביומטרי' מתבצעת בצורה דומה ל-Pin רגיל, המשתמש מזין את טביעת האצבע שלו לקורא, הקורא מפענח את נתוני טביעת האצבע וגוזר ממנה נתונים מספריים. את הנתונים הללו הוא שולח לכרטיס החכם עצמו, אם הכרטיס החכם מאשר, הוא נותן למשתמש להשתמש בפונקציונליות שהוא מממש (חתימה דיגיטלית, פענוח נתונים שהוצפנו באמצעות המפתח הציבורי המקושר לתעודת הכרטיס וכד').

כמה מילים לסיום

נכון לזמן כתיבת מאמר זה הכנסת מנסה להעביר חוק שיחייב את כולנו להכניס את טביעת האצבע שלנו למאגר מרוכז. כדי לנסות להמחיש כמה זה לא הגיוני בעליל תחשבו על זה, שהסיסמאות של כוווולכם ל-Gmail היו מאוכסנות אצל משרד הפנים, נשמע הגיוני? לא ממש, בכלל, אז מה היה קורה אם היה לא מדובר בסיסמה שלכם לג'ימייל, אלא בזהות שלכם, בכסף שלכם, ברישיון שלכם, והדרכון שלכם?

מאגרים זה דבר רגיש שעדיף שבכלל לא יהיו קיימים. כולנו יודעים שמרשם האוכלוסין החל משנת 2000 או קצת אחרי דלף כל שנה לאינטרנט וניתן להורדה היום מאימיול. אני חושש שאם אכן יוקם המאגר, לא רחוק היום שגם הוא יגיע לידים הלא נכונות.

היום ניתן בקלות ליצור שבלונת סיליקון בכל צורה של טביעת אצבע רצויה, יותר מזה, באחד מהפרקים ב MythBusters הראו איך בקלות עוקפים את מנגנון ההגנה הביומטרי (**ביומטרי בלבד**) יחסית מתוחכם שקיים בשוק - http://www.metacafe.com/watch/252534/myth_busters_finger_print_lock

כמו שהראתי במאמר זה, קיימים פתרונות טכנים רבים, בין עם שימוש ובין בלי שימוש בטביעת האצבע, לפתרון תעודות זהות דיגיטליות (שנראה שלשם פני הממשל מכוונות). פתרונות אלו קשיחים ברמה הפיזית (של הכרטיס החכם) ומאוד קשה לפצחן. אף אחד מהפתרונות לתעודות זהות דיגיטליות לא מצריך את קיומו של מאגר נתונים ראשי שכזה.

קריאה נוספת

- http://en.wikipedia.org/wiki/One_time_password - One Time Password הינו פתרון אחר להזדהות שהיא מעט יותר חזקה מסיסמה רגילה.
- http://blogs.zdnet.com/security/?p=2339 - השתמשו ב-300 PlayStation3, והפיקו סט מפתחות ותעודה דיגיטלית מפוברקת של CA ע"י שימוש ב MD5 Collision Attack. - הם כעת יכולים לחתום בצורה מפוברקת על כל אתר שיחפצו.
- http://en.wikipedia.org/wiki/Transport_Layer_Security - SSL\TLS הפרוטוקול מספר 1 בשימוש באינטרנט להצפנה של ערוץ קריא בצורה מאובטחת.
- http://docs.sun.com/source/816-6156-10/contents.htm - עוד על SSL\TLS.
- http://en.wikipedia.org/wiki/PKCS - PKCS הינו סט של סטנדרטים לייצוג של תעודות מפתחות תהליכים, בקשות להנפקות, Software API, ועוד לשימוש בעולם הקריפטוגרפיה.

Improving Images Steganography

מאת אפיק קסטיאל (cp77fk4r)

פתיחה

"סטגנוגרפיה היא האמנות והמדע של הסתרת מסרים, באופן שאף אחד זולת המקבל לא יוכל לראותם או לדעת על קיומם. בניגוד לקריפטוגרפיה שבה קיום המידע עצמו אינו מוסתר, אלא רק תוכנו. המילה סטגנוגרפיה מקורה בלטינית, סטגנו פירושה מכוסה או חבוי . זוהי אמנות עתיקה למדי, כבר לפני הספירה הנוצרית יש תיעוד על שימוש בסטגנוגרפיה, כמו כיסוי מסר שנכתב על לוח עץ בעזרת שעווה. בדרך כלל מסר סטגנוגרפי נראה על פניו כמשהו תמים אחר, כגון תמונה, קטע עיתונות, רשימת קניות או כל דבר אחר שאינו מעורר חשד, המשמש ככיסוי למסר האמיתי."

(מתוך ויקידפיה העברית)

במאמר זה אני מתכוון להציג מספר הבטים או "שיטות" בהן הסטגנוגראפים משתמשים בכדי ליעל את האלגוריתמים שלהם. נתחיל בהצגת מספר הנחות בסיס ונגדיר מספר ביטויים חשובים, לאחר מכן נציג שיטות שונות ליעול האלגוריתמים.

מונחים כללים

כאשר אנו מדברים על סטגנוגרפיה ויזואלית, המושג "רעש" אינו מתייחס לרעשים ווקאלים אלא מצביע על סטיה בצבעי התמונה המקורית. הרעיון העומד מאחורי מדידת רעש הוא לקבוע את כמות הרעש שיכול להופיע בתמונה מבלי שעין הצופה הממוצע תוכל לקבוע בוודאות כי אכן בוצעו שינויים בתמונה. מספר מונחי יסוד הקיימים בנושא יוכלו לעזור לנו להגדיר מה הקריטריונים שעל-פיהם נעבוד:

- **SNR**- [Signal-to-Noise Ratio] - **היחס בין עוצמת האות לעוצמת הרעש הכולל בהעברת אות ממקור מסוים ליעד**. הרעש עשוי להיות רעש נלווה לאות המקורי (למשל רעש שוט), רעש שקיים בתווך שבו מועבר השידור ,או רעש בגלאי.
- **PSNR**- [Peak Signal-to-Noise Ratio] - **היחס המרבי האפשרי בין הספק האות לבין הספק הרעש** שמשפיע על אמינות האות המוצג.
- **MSE**- [Mean Squared Error] - הוא **ממוצע השגיאות (השינוי היחסי) בריבוע**. (מסתכלים על ממוצע בריבוע בגלל שככל שהממוצע יגדל כך נוכל לעקוב אחריו בקלות יותר)

מתחילים

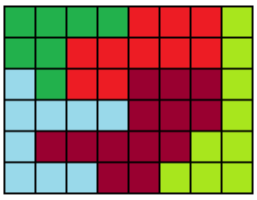
אנחנו מחפשים עקרונות ונקודות בהם אנחנו יכולים להשתמש בכדי להשפיע על איכות ה-Embed (הטמעה) של כל אלגוריתם סטגנוגראפי שהוא. נתחיל בהצגת משפט חשוב:

- אלגוריתם סטגנוגראפי "קלאסי" ישאף לחלק את המידע אותו הוא מעוניין להטמיע (Payload) למספר רב של חלקים ככל הניתן, ביחס לגודל האיזור בו הוא מחביא את המידע (Container), בכדי לשנות את התוצר כמה שפחות (שאיפה ל-MSE כמה שיותר קטן).

זה נשמע הגיוני- נניח ואנחנו רוצים להחביא מרשם סודי להעשרת אורניום בתוך תמונה, כך שנוכל להעביר אותה בשדה-התעופה מבלי שיגלו אותנו. ההנחה הראשונה אומרת שככל ש"נפזר" את המרשם ביותר מקומות בתמונה כך הצופה ירגיש בשינוי כמה שפחות- כל שנפזר את ה-Payload כך חלקיו יהיו קטנים יותר וכך ירגישו בהם פחות.

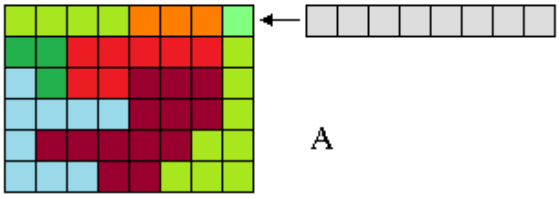
דוגמא

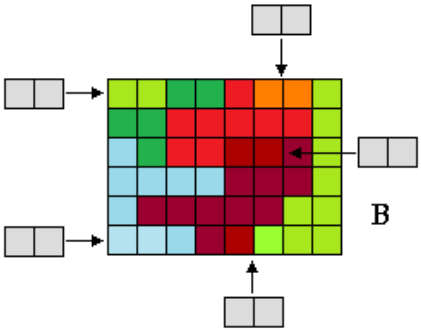
נניח שיש לנו את התמונה הבאה (48 בתים):



ואנחנו מעוניינים להחביא בה את המידע הבא (8 בתים):

שימו לב להבדל בתוצאה (פחות או יותר- MSE) בין שתי האפשרויות שלנו:





- A - הכנסנו את ה-Payload בחתיכה אחת, פשוט כמו שהוא.
- B - חתכנו את ה-Payload לארבעה חתיכות בנות 2 בתים כל אחת.

הדוגמאות עצמן אינן מדגימות במדויק את שינוי הגוונים שהיו אמורים להתרחש אלא גוונים "קרובים" פחות או יותר בכדי להמחיש את הרעיון.

הכוונה ב- "ירגישו בהם כמה שפחות" היא שאם מישהו באמת יקח את שתי התמונות, יציב אותן אחד ליד השניה וינסה למצוא הבדלים הוא ישים לב לכמה שפחות שינויים, זאת בדיוק ההגדרה של אלגוריתם סטגנוגרפי מוצלח לעומת אלגוריתם סטגנוגרפי פחות מוצלח.

אם על ידי שימוש ביחס של ה-Container וה-Payload הצלחנו לגלות, את מספר החלקים המירבי בו נוכל לחלק את ה-Payload, מה עוד נשאר לשפר באלגוריתם שלנו?

כמובן שמבחינת שיפור ה-MSE, בעזרת מציאת כמות החלוקה האפקטיבית ביותר אין יותר מה לשפר, יחס זה יחס, אבל אנחנו יכולים לשפר נתון אחר- נוכל לשפר את האלגוריתם שלנו ע"י מציאת המיקום האפקטיבי ביותר להחבאת ה-Payload!

כאן נשאלת השאלה- האם המיקום של כל חתיכה משנה? ואם כן, איפה הכי כדאי למקם אותן?

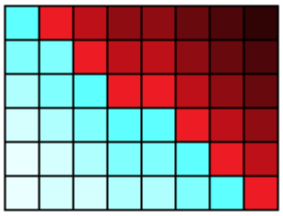
בכדי לענות על שאלות אלה אנחנו צריכים לזכור דבר אחד- **מדובר בהשוואה שלא מתבצעת בעזרת מחשב, אלא בעזרת העין האנושית**. אתם שואלים מה הקשר? הרעיון הוא לנצל חולשות בעין האנושית (או יותר נכון במח האנושי שמפרש את מה שהעין שלנו רואה) ולפיכך למקם את החתיכות שלנו.

מספר עובדות חשובות על העין האנושית

- **לעין האנושית קשה יותר לזהות סטיות בין גוונים כהים מאשר סטיות בין גוונים בהירים.** משמעות הדבר שאותו PSNR בגוונים כהים יתן לנו MSE נמוך הרבה יותר מבגוונים בהירים.
- **לעין האנושית קשה יותר לזהות סטיות בשולי התמונה מאשר סטיות במרכזה , ניתן להבין מכאן** שאותו PSNR בשולי התמונה יתן לנו MSE נמוך מבמרכזה.
- **לעין האנושית קשה יותר לזהות סטיות כשמדובר במצב של חפיפה בין ניגוד גוונים.** לפיכך, אם נמצא אזור בתמונה שיש לו SNR גבוה יחסית לשאר התמונה, נוכל ליצור שם PSNR שיתן לנו MSE נמוך יחסית לPSNR שממוקם באיזור בעל SNR נמוך ביחס לשאר התמונה.

מיפוי ביחס לגוונים:

נבחן את העובדה הראשונה, לפיה לעין האנושית קשה יותר לזהות סטיות בין גוונים כהים. נראה איך זה מתקיים. נניח וקיימת לנו התמונה הבאה:



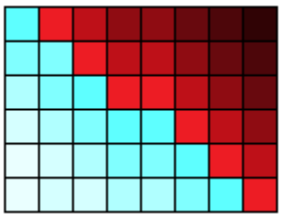
אנחנו צריכים לבצע בדיקת גוון (Shade Analysis), לבדוק מה ממוצע הגוונים בתמונה ולמפות את התמונה לאיזורים ולכל איזור לתת ניקוד ביחד לממוצע. נניח ולאחר שליפת הממוצע מניתוח הגוונים קיבלנו שמרכז התמונה הוא הממוצע. המפוי שלנו יראה כך:

0	0	1	2	2	3	4	5
-1	0	0	1	1	2	3	4
-2	-1	0	0	0	1	2	3
-3	-2	-1	0	0	0	1	2
-4	-3	-2	-1	-1	0	0	1
-5	-4	-3	-2	-2	-1	0	0

בדוגמה זו נוכל לראות שככל שהגוון כהה יותר, כך הרלוונטיות שלו בלהכיל PSNR שייתן לנו MSE-קטן יחסית- גדלה.

העובדה השנייה מבהירה כי לעין האנושית קשה יותר לזהות סטיות בשולי התמונה מאשר במרכזה. לעין האנושית (או שוב, למוח האנושי) **קשה יותר לקלוט סטיות בגוונים המקוריים ככל שהם מרוחקים מהמרכז**. למה? בגלל "הפוקוס" האוטומטי שהעין מבצעת. אם תחזיקו עיפרון מולכם ותסתכלו רק עליו, העפרון יראה בבירור והאובייקטים בצדדיו (לא מאחוריו) יהיו חסרי פרטים, העין שלנו רגישה פחות ופחות לפרטים במסגרות התמונה.

לכן, כמעט תמיד נשאף ליצור את הסטיות שלנו רחוק ככל שניתן מהמרכז. בואו נראה איך זה בא לידי ביטוי, אם קיימת לנו התמונה הבאה:



אנחנו (בהתייחסות רק לעובדה השניה) ננקד אותה באופן הבא:

3	2	2	2	2	2	2	3
3	2	1	1	1	1	2	3
3	2	1	0	0	1	2	3
3	2	1	0	0	1	2	3
3	2	1	1	1	1	2	3
3	2	2	2	2	2	2	3

כמובן שבמציאות אנחנו לא מתחשבים רק במיקום הבתים, אך לכל דבר הזמן שלו. אם נממש אלגוריתם שיבצע מיפוי רלוונטיות על פי שתי עובדות אלו, התוצאה תראה כך:

0	0	1	2	2	3	4	5
-1	0	0	1	1	2	3	4
-2	-1	0	0	0	1	2	3
-3	-2	-1	0	0	0	1	2
-4	-3	-2	-1	-1	0	0	1
-5	-4	-3	-2	-2	-1	0	0

+

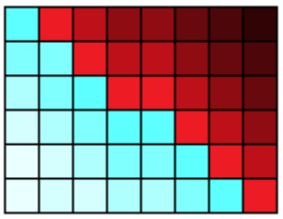
3	2	2	2	2	2	2	3
3	2	1	1	1	1	2	3
3	2	1	0	0	1	2	3
3	2	1	0	0	1	2	3
3	2	1	1	1	1	2	3
3	2	2	2	2	2	2	3

=

3	2	1	3	4	3	5	8
2	2	2	2	2	3	5	7
1	1	1	0	0	2	4	6
0	0	0	0	0	1	3	5
-1	-1	-1	0	0	1	2	4
-2	-2	-1	0	0	1	2	3

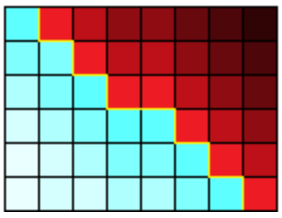
שימו לב לתוצאה כאשר מתחשבים בשתי העובדות ולא רק באחת מהן, ננסה להבין ולשלב גם את העובדה השלישית שלנו.

לפי העובדה השלישית, לעין האנושית קשה יותר לזהות סטיות כשמדובר במצב של חפיפה בין ניגוד גוונים. עובדה זו מעט מסובכת יותר היות ומדובר בהבנה של מצב ניגודי בין גוונים:



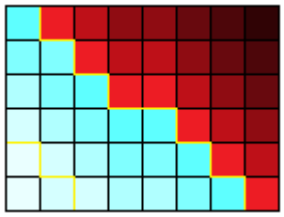
(כאן הזיהוי עוד פשוט יחסית- תחשבו שמדובר בפיקסלים הרבה הרבה יותר קטנים)

העין האנושית כמובן תמשך ישירות למרכז התמונה היות ושם מעבר הגוונים הוא החד ביותר:



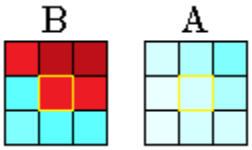
הגבול הצהוב מסמל את הניגוד המירבי ביותר בתמונה, או בשפה מקצועית יותר - כאן ה-SNR הוא הגבוה ביותר. העובדה השלישית מבהירה כי אם ניצור PSNR- סטייה עם גוון לא קשור- דווקא איפה שיש SNR גבוה, העין האנושית פחות תשים לב לסטייה.

אם נרצה להסביר את זה בצורה פשוטה, קשה יותר לעין שלנו למצוא פרח בצבע מסויים בתוך שדה עם מלא פרחים בצבעים שונים, מאשר למצוא ענן אפור בשמיים כחולים לגמרי. שימו לב לדוגמא:



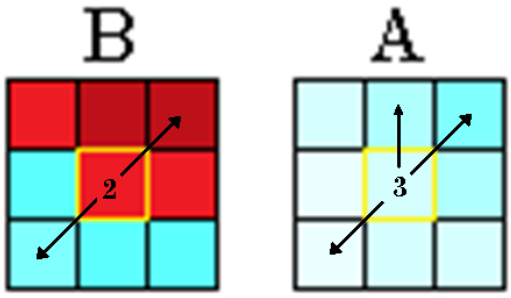
איזה קו יותר קל לזהות? זה שבמרכז או זה שבפינה השמאלית התחתונה?

צורת הניקוד כאן טיפה יותר מסובכת, אנחנו מחלקים כל פיקסל על ידי שיכלול של כל שמונת הפיקסלים סביבו, מבצעים SNR מקומי לכל פיקסל, ולפי ה-MSE (המקומי) נותנים ציון לפיקסל. כאן זה הפוך- ככל שה-MSE המקומי גדל כך הרלוונטיות להחביא שם מידע גדלה גם היא. הסתכלו על המצבים הבאים:



אם ננסה ליצור PSNR בפיקסל המסומן ב-A נקבל MSE גבוה בהרבה מיצירת PSNR בפיקסל המסומן ב-B. למה? כי ה-SNR המקומי ביחס לפיקסל המסומן ב-A הרבה יותר נמוך מה-SNR המקומי ביחס לפיקסל המסומן ב-B. מהלך המיפוי מתנהל כך: בוחרים פיקסל, בודקים בכמה פיקסלים שונים ממנו הוא "נוגע" ולפיכך נותנים לו ציון (אם היינו רוצים להיות ממש יעילים, היינו יוצרים פונקציה שבודקת מה רמת הניגוד בכל פיקסל ביחס לפיקסלים שלידו).

לדוגמא:

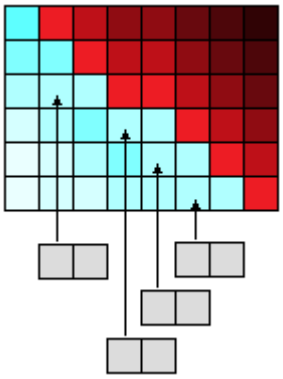


הפיקסל המסומן ב-A נוגע בשלושה גוונים השונים ממנו, הפיקסל המסומן ב-B נוגע רק בשניים. לכן, המיפוי של התמונה שלנו יראה כך:

1	2	3	2	2	2	2	1
2	3	3	2	2	2	3	2
2	3	3	2	3	3	2	2
3	2	3	3	2	3	2	2
2	3	2	2	2	3	3	3
1	2	3	2	2	2	2	2

הבה נמחיש זאת, במידה ונרצה להכניס את התמונה הקודמת את המידע הבא:

לאור הבנת העובדה השלישית, נעדיף למקם את המידע איפה שה-SNR הכי גבוהה, כך נקבל מינימום של MSE. לכן, נעדיף לבצע את ההטמעה באופן הבא:



(שימו לב שגם בפיקסלים כאלה מדובר בהטמעה כמעט מושלמת.)

כאן ננסה לראות איך המיפוי יראה כשנתחשב בשלושת העובדות שלמדנו:

0	0	1	2	2	3	4	5
-1	0	0	1	1	2	3	4
-2	-1	0	0	0	1	2	3
-3	-2	-1	0	0	0	1	2
-4	-3	-2	-1	-1	0	0	1
-5	-4	-3	-2	-2	-1	0	0

+

3	2	2	2	2	2	2	3
3	2	1	1	1	1	2	3
3	2	1	0	0	1	2	3
3	2	1	0	0	1	2	3
3	2	1	1	1	1	2	3
3	2	2	2	2	2	2	3

+

1	2	3	2	2	2	2	1
2	3	3	2	2	2	3	2
2	3	3	2	3	3	2	2
3	2	3	3	2	3	2	2
2	3	2	2	2	3	3	3
1	2	3	2	2	2	2	2

=

4	4	4	5	6	5	7	9
4	5	5	4	4	5	8	9
3	4	4	2	2	4	7	8
3	2	3	3	2	4	5	7
1	2	1	2	2	4	5	7
-1	0	2	2	2	3	4	5

לאחר שקיבלנו את מפת הרלוונטיות המשכללת בתוכה את שלושת העובדות, נוכל בקלות לדעת היכן הכי כדאי לנו למקם את המידע שלנו.

סיכום

יש עוד המון "עובדות" או לוגיקות מהן אפשר להגיע לעוד דרכים ליעול הטמעה וכך להגיע לרמות סטגנו מאוד גבוהות. הרעיון הכללי הוא להבין ממי אנחנו רוצים להסתיר את המידע (במקרה שלנו- העין האנושית, אך יש המון "גורמים עויינים" כגון אוזן, תוכנת מחשב וכו'). מימוש אלגוריתם שכזה איננו קשה במיוחד וניתן לביצוע גם אם נתבסס רק על שלוש עובדות אלו. כמו כן, נוכל ליעל אותו יותר אם נתייחס לנקודות הבאות:

- התייחסות לגוונים נוגדים המבלבלים את העין.
- התייחסות לגוונים משלימים.
- התייחסות לרעשים נוספים ולאו דווקא לממוצע ה-SNR.
- שינוי הניקוד על פי איזורים קטנים יותר בתמונה.
- שינוי הניקוד על פי שטחים בעלי SNR ממוצע לשמונת גזרות SNR קרובות (ולא רק ברמת הפיקסל).

משפחת פרוטוקולי IPSec

מאת סולימני יגאל

הקדמה

משפחת הפרוטוקולים IPSEC אחראית על אבטחת התקשורת העוברת בין ציוד ברשת, מאמר זה יתמקד במכלול השירותים המסופקים בה. נבין מהו פרוטוקול להחלפת וניהול מפתחות מוצפנים, נכיר את מבנה הפרוטוקולים המרכיבים אותה ונסביר על אופן הגנתה מפני המתקפות השונות.

מדוע עלינו לאבטח את הרשת?

ישנן מספר רב של מתקפות המאיימות על הרשת הפנים אירגונית שלנו. ישנן מתקפות המגיעות מתוך הרשת (תוקף פנימי- יכול להיות עובד אירגון המנסה לקבל גישה למחשבו של עובד אחר לדוגמא) וישנן מתקפות המגיעות מחוץ לרשת. מספר רב של מתקפות נובע מחוסרים במנגנוני האבטחה הקיימים בפרוטוקולים אשר נמצאים בשימוש ברשת הפנימית. לדוגמא, מתקפת Arp Poisoning הופכת לכמעט לא רלוונטית כאשר המידע העובר ברשת מוצפן, ומתקפת Replay Attack אינה יכולה להתבצע על מערכות אשר עושות שימוש במנגנוני Timestamp. אלו רק דוגמאות אחדות למתקפות המאיימות על הרשת שלנו שאנו יכולים למנוע בעזרת השימוש בתקשורת מאובטחת. ולכך נועדה משפחת ה-IPSEC.

Internet Protocol Security (IPSec)

משפחת IPSEC מספקת שרותי אבטחה בשכבת ה-IP, היא מגדירה את האלגוריתם לשימוש המערכת, ומשתמשת ב-"מפתח" מוצפן כדי לספק תקשורת מאובטחת בין מחשבים או רשתות. הפרוטוקול יכול להתנהל בין זוג hosts ,זוג secure gateways או בין host ו-secure gateway.

מכלול שרותי האבטחה ש-IPSEC מספק כוללים:

- בקרת גישה, הגבלת גישה לקבצים או מערכת.
- אימות נתונים במוצא.
- דחייה של מנות.
- שליחה חוזרת של מנות פגומות או לא רציפות שהתקבלו עקב בעיה או רעשים בקו.
- הצפנה.

כיוון ששרותים אלו ניתנים בשכבת ה-IP, פרוטוקולים בשכבות גבוהות יותר יכולים להיעזר בשרותים הללו. פרוטוקולים לדוגמא: TCP, UDP, ICMP, BGP וכו'.

יכולות אלו מאופשרות על ידי 2 פרוטוקולי אבטחת תעבורה הקיימים במשפחה זו: Authentication Header (AH) ו-Encapsulation Security Payload (ESP), ובאמצעות מפתחות מוצפנים ופרוצדורות ניהול.

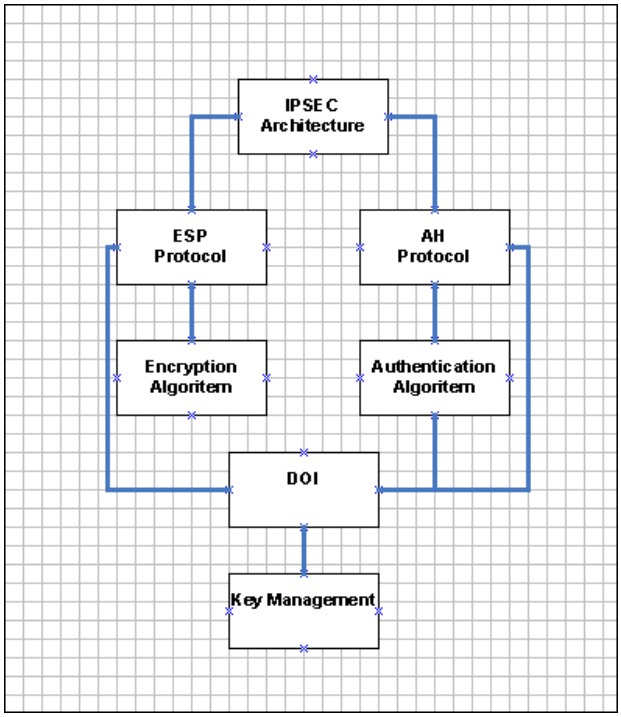
מכלול פרוטוקולי IPSec ואופן הגדרתם מתבצע על פי דרישת המערכת, המשתמשים או מנהלי האתר. מכלול הפרוטוקולים הסטנדרטי מתאים לכלל המשתמשים ברשת, אולם השימוש בפרוטוקולים אלו בתוספת IPSEC מאפשר את אבטחת התעבורה ברמה גבוהה ביותר.

פרוטוקולים להחלפת מפתחות

המטרה של פרוטוקולים להחלפת מתפחות היא הסכמה על מפתח משותף לשני המשתתפים. פרוטוקול כזה צריך לקיים את התכונות הבאות:

- **Authenticity**- אימות הזהות של המשתתף השני (מניעת התקפות Man in the Middle).
- **Secrecy**- רק שני המשתתפים המקוריים יודעים את המפתח הסודי המשותף.

מבנה הפרוטוקול:



IP Authentication Header (AH)

ה-AH הוא פרוטוקול מרכזי ב-IPSEC, אחראי על רציפות התקשורת, אימות הנתונים בתעבורת הרשת ומספק הגנה מתעבורה חוזרת (שידורים חוזרים שלא זקוקים להם), ניתן להגדיר את הפרוטוקול כפרוטוקול הראשי בשילוב עם IP Encapsulating Security Payload (ESP) שעליו נרחיב בהמשך.

פרוטוקול AH ניתן לשימוש בין תחנות ברשת, בין secure gateways. בגדול, הפרוטוקול AH מעניק את אותן יכולות האבטחה כמו ESP מלבד ההצפנה. כאשר משתמשים ב-IPv6 ה-"Authentication Header" מופיע בין "IPv6 Hop-by-Hop Header" לבין "IPv6 Destination Options", אך כשמשתמשים ב-IPv4 ה-"Authentication Head" מופיע ישירות אחרי ה-"IPv4 header" הראשי.

מבנה חבילה בפרוטוקול:

8 bits	16 bits	32 bits
Next Header	Payload Length	Reserved
Security parameters index (SPI)		
Sequence Number Field		
Authentication data (variable)		

- "Next header"- מגדיר את סוג השדה שלאחר ה-"Authentication Header".
- "Payload Length"- מציין את אורך ה-AH ב-32Bit.
- "SPI"- שדה מספרי בגודל- 32 Bit שבשקלול עם כתובת ה-IP של היעד וה-AH מגדיר את קוד ההתקשרות, ליתר דיוק את קוד השיחה. יכולות להיות מספר שיחות מאובטחות, השקלול הזה מגדיר מעין מספר סידורי של ההתקשרות.
- "Sequence Number"- מונה מסגרות- שדה שמכיל מספר רץ.
- "Authentication Data"- שדה ביקורת Integrity Check Value (ICV) הוא קוד לזיהוי שגיאות שמעניק את האפשרות לזהות את השגיאות ואפילו לתקנן, שדה זה הוא חשוב ביותר ומורכב מאוד לזיהוי שגיאות במידע, הוא מופעל כפונקציית בדיקה על המסגרת, 2 הצדדים מחליטים על פונקציה מסוימת מראש, ומצרפים חלק נוסף להודעה, שהיא בעצם התוצאה של הפונקציה שהופעלה על המסגרת , הצד המקבל צריך להפעיל את הפונקציה שהוסכמה מראש על ההודעה ולוודא שמתקבלת תוצאה זהה לחלק שצורף להודעה, אם היא לא זהה, המקבל מבין שנקלטה הודעה עם שגיאות ומבקשת שליחה מחדש.

IP Encapsulating Security Payload (ESP)

ה-ESP הוא פרוטוקול מרכזי בארכיטקטורת IPSec. הפרוטוקול מעניק שרותי אבטחה ב-IPv4 וב-IPv6, שירותי האבטחה כוללים הצפנה ושליחה רציפה ללא שגיאות (במקרה של שגיאה תתבצע שליחה חוזרת). המידע המוצפן מועבר בשדה Payload data.

פרוטוקול ESP מזוהה ע"י המספר 50, מספר זה ניתן לפרוטוקול על ידי IANA (Internet Assigned Numbers Authority) שהיא הרשות שהיתה אחראית על המדיניות של הקצאות מספרים לפרוטוקולים, לפורטים או למספרי IP, הרשות שאחראית על הנושא כיום היא ICANN (Internet Corporation of Assigned Names and Numbers)

מבנה חבילה בפרוטוקול:

16 bits	24 bits	32 bits
Security association identifier (SAID)		
Sequence Number		
Payload data (variable length)		
Padding (0-255 bytes)		
	Pad Length	Next Header
Authentication Data (variable)		

- "Security association identifier"- מספר הזיהוי של הפרוטוקול, כדי שהתחנה הקולטת תדע כיצד לפעול עם הקלט של המסגרת.
- "Payload Data"- שדה באורך משתנה המכיל את המידע.
- "Padding C"- שדה המכיל ריפוד עבור ההצפנה, 2 הצדדים מסכימים על גודל מסויים של סך ה-bytes במידע, את השאר מרפדים באפסים ואז מצפינים, לדוגמא: הסכמה על שדה בגודל 10bytes, אם נכנסים למסגרת רק 97bytes, אזי מרפדים ב-3 אפסים ואז מצפינים. מנגנון זה מגביר את האבטחה על תעבורת המידע.
- "Pad length"- מציין את מספר ה-bytes בשדה הבא.
- "Next header"- מגדיר את סוג המידע שמכיל שדה ה-Payload Data.

פרוטוקול מרכזי בארכיטקטורת IPSec, נוצר על ידי ה-NSA, מגדיר את התהליכים ואת עיצוב המסגרות, מעניק ערוץ מאומת ומאובטח המשמש להסכמה על מפתחות ומנהל את החלפת המפתחות המוצפנים.

פרוטוקול זה מכיל שני שלבים עיקריים:
שני הצדדים יוצרים ערוץ מוגן ומסכימים על נהלי השליחה, בשלב זה נוצר ה-SA בינהם. יש קישור מאובטח בין שני הצדדים ומידע נשלח בהתאם לנהלים שנקבעו בשלב א.

מבנה חבילה בפרוטוקול:

8 bits	12 bits	16 bits	24 bits	32 bits
Initiator Cookie				
Responder Cookie				
Next Payload	MjVer	MnVer	Exchange Type	Flags
Message ID				
Length				

- "Initiator Cookie"- המחרוזת שעל השולח לשלוח כדי להזדהות בפני הצד המקבל.
- "Responder Cookie"- המחרוזת שנשלחת כדי לאשר הקמת או מחיקת SA.
- "Next Payload"- סוג השדה הבא בהודעה.
- "MjVer"- הגירסה הראשית של פרוטוקול ISAKMP שבשימוש כרגע.
- "MnVer"- הגירסה המשנית של פרוטוקול ISAKMP שבשימוש כרגע.
- "Exchange Type"- סוג החילוף שמתבצע כרגע.
- "Flags"- אפשרויות שונות שנקבעות לחילופי ISAKMP.
- "Message ID"- קוד הזיהוי למצב הפרוטוקול במהלך המעבר בין השלב הראשון לשני .
- "Length"- אורך ההודעה.

Internet Key Exchange (IKE)

ה-IKE הוא פרוטוקול לבניית וניהול IPSEC SA האחראי על החלפת המפתחות בין שני מחשבים שמשתמשים ב-IPSEC, פרוטוקול זה ממומש כאפליקציה העובדת מעל UDP בפורט 500.

ה-IKE משלב בתוכו שני פרוטוקולים ISAKMP ו-OAKLEY (פרוטוקול להסכמה על מפתחות ועל תכונות ה-IPSEC SA).

כאשר מחשב ברשת מבקש חיבור מאובטח, IKE מופעל על חבילות המידע רק בשלב הראשוני, כלומר לפני שהוסכם על SA ועד שיוסכם על SA.

ה-IKE מורכב משתי פאזות, הפאזה הראשונה מייצרת ערוץ מאובטח כדי להגן על הפאזה השנייה, במהלך הפאזה הראשונה, הצדדים מסכימים על ה-ISAKMP שיגן על הפאזה השנייה, על שיטות האימות והמפתחות, הכנת keying material סודי משותף לשני הצדדים-ורק להם שממנו יגזרו מפתחות ל- ISAKMP SA. הפאזה השנייה בונה את ה-SA עבור IPSEC.

יתרונות ה-IKE:

- סודיות ואימות.
- חסינות לשידור חוזר.
- הגנה על זהות המשתתפים בשיחה.
- פרוטוקול פשוט לאנליזה ושימוש.
- עמיד בהתקפות שונות (התחזות, Man in the middle, Denial of service).

חסרון ה-IKE:

- מבצע חישובים יקרים (מפתח DH למשל, הוא חישוב יקר שדורש העלאות רבות בחזקה מודולו מספר ראשוני, דבר האורך זמן רב).

הגנה מפני מתקפות שונות

השימוש במשפחת ה-IPSec מונע מהרשת שלנו להיות חשופה למספר גדול של מתקפות, בחלק זה של המאמר נסקור אותן ונסביר כיצד הדבר מתבצע.

הגנה מפני Denial of service Attack

התקפה זו מתבצעת על ידי תוקף שמבצע בדרך כלל IP spoofing, התוקף בעצם מציף את הקורבן בבקשות IKE ומכריח אותו לבצע חישובים יקרים, כדי למנוע את החישובים היקרים האלו, בזמן קבלת המידע יש צורך לוודא שהצד השולח נמצא בכתובת ה-IP שמופיעה ב-IP header .

הבעיה: התוקף יוזם הפעלות רבות של ה-IKE מול הנתקף בפרק זמן קצר, שמכריחות את הנתקף לבצע חישובים כבדים ויקרים, לאחר זמן מסוים הנתקף לא יוכל להפעיל IKE עם משתמשים אחרים והתוקף לא מבצע כלל את חישובי ה-DH.

בדרך כלל בהתקפות DoS בכל הפעלה של IKE התוקף ישתמש בכתובת מקור אחרת (IP Spoofing)

בדרך זו התוקף אינו חושף את עצמו, כיוון שהוא לא משתמש בכתובת ה IP שלו, לכן הנתקף אינו יכול להגן על עצמו על ידי הגבלת מספר ההתקשרויות המותרות בו זמנית מכתובת IP מסוימת.

הפתרון: שימוש מחרוזות Cookies.

כל משתמש בפרוטוקול שולח מחרוזת אקראית (Cookie) לצד השני, שנדרש להשיב את המחרוזת שקיבל כדי להוכיח שהוא אכן מקיים מהלך שיחה רציף, החישובים הכבדים והיקרים יתבצעו רק לאחר קבלת המחרוזת המקורית.

כך, נוכל לדעת **לפני שנבצע את החישובים היקרים** האם בקשות ה-IKE אכן רלוונטיות או לא.

הגנה מפני מתקפות MITM Attack כדוגמת Arp Poisoning

התקפה זו מתבצעת ע"י תוקף שהצליח לנווט את המידע היוצא מתחנה אחת לתחנה שניה- דרכו (לדוגמא- למידת טבלאות ה- Arp ועידכון ה-Physical Address של ה-Destination ל-Physical Address שלו עצמו), וכך יוכל לדלות מידע (כגון שמות משתמשים וסיסמאות) או לערוך מידע וכד'.

הבעיה: התוקף לומד את טבלאות ה-Arp ברשת, מעדכן אותן בפרטיו לתחנות ספציפיות וגורם למידע שנשלח מהן לעבור דרכו וכך יוכל לראות את תוכן חבילות המידע העוברות בתקשורת.

הפתרון: שימוש בהצפנה.

המידע העובר ברשת מוצפן בעזרת מנגנוני הצפנה מבוססי מפתח, במידה ותוקף יבצע מתקפת MITM בכדי לעיין במידע- התוקף לא יוכל לעשות עם המידע כלום מפני שהוא מוצפן.

הגנה מפני Replay Attack

התקפה זו מתבצעת ע"י תוקף המאזין לתקשורת ברשת, ומתחקה לאחד המשתתפים בשיחה על-ידי שידור חוזר של המסגרות שלו, בכוונה שהצד השני יעביר את השידור אליו.

הבעיה: התוקף מבצע מתקפת MITM לתחנות ברשת ומאזין לתקשורת ביניהן, במטרה לשלוף מסגרת מידע רלוונטית לזהותו של אחד הצדדים בשיחה במטרה לשלוח אותה לצד השני עם פרטיו בכוונה שהצד השני יעביר את השידור אליו.

הפתרון: שימוש במנגנון Timestamp.

כאשר כל צד משתמש במנגנון Timestamp בכדי לאמת את "טריות" המסגרת. יש לציין כי חתימת הזמן מוצפנת ביחד חבילת המידע כך שלא יהיה ניתן לזייפה. כך כל מסגרת שמתקבלת נבדקת ואם היא לא "טרייה" , אזי יש נסיון פריצה לרשת . חשוב לסנכרן בין כל הציוד ברשת , כדי שהשעון יהיה זהה.

סיכום

במאמר זה הצגנו את מבנה פרוטוקולי האבטחה והפרוטוקולים לניהול והחלפת מפתחות היוצרים את משפחת ה-IPSec, הבנו את חשיבות השימוש בתעבורה מוצפנת ברשת שלנו וסקרנו מספר מתקפות ואת אופן מניעתן בעזרת שימוש בפרוטוקולים אלו.

IP Tables

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

IPTables מהווה כלי רב עוצמה המשתמש בעיקר כתוכנית Firewall מבוססת אירועים ומאופיינת כ-Stateful Firewalling (טכנולוגיה המבוססת על בדיקת תוכן/כותרות ה-Packets), במערכות הלינוקס למינהן. הרעיון מאחורי ה-IPTables מאוד פשוט, בעזרתה נוכל לקבוע מה יעלה בגורל כל Packet שיגיע למחשבינו, בשל גמישותה נוכל להבדיל כמעט בין כל Packet ו-Packet, בין אם מדובר בזיהוי מקורו/יעדו, סוגו/תפקידו, או כמעט כל מאפיין שאפשר לחשוב עליו.

הסבר כללי

הכוונה ב-"לקבוע מה יעלה בגורל כל Packet" היא בעצם אישור כניסה ל-Packet, התעלמות מה-Packet, התעלמות ושליחת הודעת שגיאה לשולח ה-Packet וכו', העברת ה-Packet ליעד אחר וכו'.

הרעיון המרכזי הוא שבעזרת מכלול חוקים, כלליים או נקודתיים, נוכל לקבוע איך תצורת תעבורת הרשת מהמחשב שלנו ואל המחשב שלנו תתנהג, וכך למנוע מחבילות מידע "זדוניות" או מכל מני פורענויות לחדור למחשב שלנו.

כיום, בקרנלים (עוד מ-2.4.X) מובנית טבלה שנקראת Filtering Table, דרך הטבלה הזאת המערכת קובעת מה יעלה בגורל Packet המגיע למערכת על ידי מערכת חוקים הכתובים בה.

מבנה המערכת

אפליקצית ה-IPTables הגיעה לעולם כחלק מפרוייקט NetFilter של Rusty Russell מ-Core Team המופץ תחת GPL, היא כלי אשר נועד לנהל בצורה נוחה את ה- Filtering Table. אפליקציה זו מבוססת על מספר טבלאות/שרשראות-חוקים עקריות אשר בעזרתן נוכל לנהל את תצורת תעבורת ה-Packets במערכת שלנו.

הטבלאות הן:

- **FILTER** - טבלת ברירת המחדל, הטבלה הכי בסיסית אם לא תקבע שום טבלה שתוגדר כאחראית לטיפול באירוע, ה-Packet יגיע לכאן.
  בטבלה הזאת קיימות שלוש שרשראות:
  - **INPUT**- שרשרת המוגדרת לטפל ב-Packets אשר נכנסים למערכת.
  - **OUTPUT** - שרשרת המוגדרת לטפל ב-Packets אשר יוצאים מהמערכת.
  - **FORWARD** - שרשרת המוגדרת לטיפול ב-Packets המיועדים לניתוב.

- **NAT** - טבלת הניתוב, הטבלה האחראית לניתוב ה-Packets וקיימות בה שלוש שרשראות:
  - **PREROUTING** - שרשרת המוגדרת לטפל ב-Packets לפני הניתוב.
  - **POSTROUTING** - שרשרת המוגדרת לטפל ב-Packets לאחר הניתוב.
  - **OUTPUT** - שרשרת המוגדרת לטפל ב-Packets היוצאים.

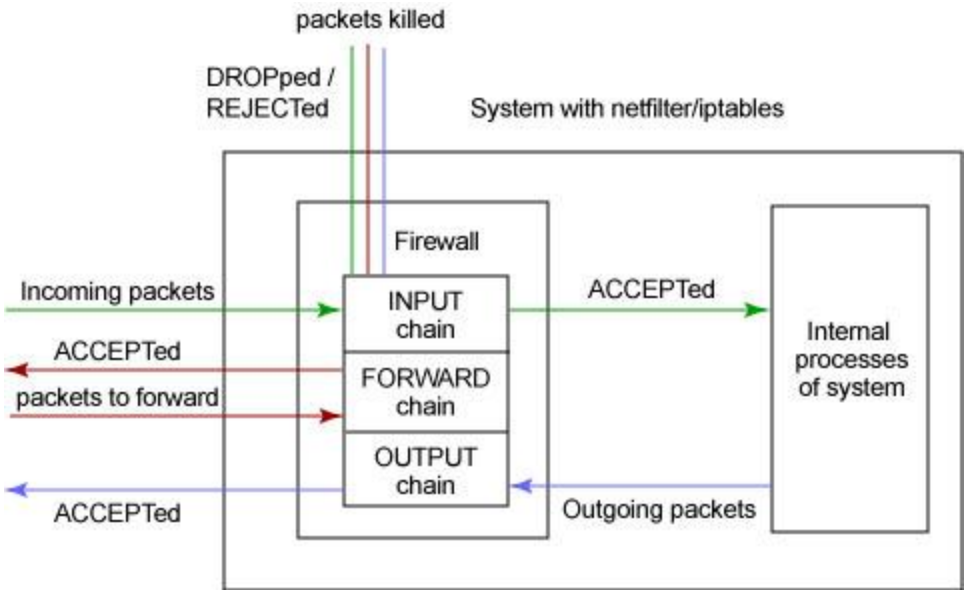
- **MANGLE** - טבלה לטיפול מתקדם ב-Packets. בקרנלים מ-2.4.18 קיימות חמש שרשראות:
  - **PREROUTING** - שרשרת המוגדרת לטפל ב-Packets לפני הניתוב.
  - **POSTROUTING** - שרשרת המוגדרת לטפל ב-Packets לאחר הניתוב.
  - **INPUT** - שרשרת המוגדרת לטפל ב-Packets אשר נכנסים למערכת.
  - **OUTPUT** - שרשרת המוגדרת לטפל ב-Packets היוצאים.
  - **FORWARD** - שרשרת המוגדרת לטיפול ב-Packets המיועדים לניתוב.

כמו שראינו, כל טבלה מכילה מספר שרשראות, וכל שרשרת יכולה להכיל מספר חוקים.

למרות שהחוקים דומים בכל הטבלאות, והעקרונות אף הם דומים למדי, במאמר זה אגע רק בשרשרת בעלת השימוש הנפוץ ביותר בטבלת ה-**FILTER** - שרשרת ה-**INPUT.**

תפקוד

כאשר מגיע Packet למערכת ה-IPTables (מבפנים או מבחוץ), מערכת ה-IPTables מזהה לאיזה אירוע מתאים ה-Packet, Packet יוצא ישלח לשרשרת ה-OUTPUT, Packet נכנס ישלח לשרשרת ה-INPUT.



[התמונה נלקחה מ-DeveloperWorks של IBM]

לאחר שזוהתה השרשרת בטבלה אליה ה-Packet מתאים, המערכת עוברת חוק חוק (בסדר מהעליון לתחתון) בשרשרת חוקים ובודקת האם קיים חוק העונה ל-Packet הספציפי, לאחר שהיא מוצאת חוק שמאפייני ה-Packet עונים לו, תפסיק המערכת בחיפוש אחר החוקים ותבצע את מה שהחוק אומר. לפיכך חשוב לשים לב טוב טוב לסדר בו כותבים את החוקים.

אם חוק אחד אומר **לקבל כל Packet** המשתמש בפרוטוקול TCP, ואחריו יש חוק האומר **שאין לקבל שום Packet**, ה-Packet **יכנס למערכת ללא בעיה**. לעומת זאת, אם קיים מצב ובו יש חוק האומר למערכת **לא לקבל שום Packet** ואחריו קיים חוק האומר **לקבל כל Packet מכתובת IP מסויימת**- שום Packet **לא יתקבל**. חשוב מאוד לשים לב לסדר כתיבת החוקים ולזכור כי לאחר שיימצא החוק **הראשון** המתאים ל-Packet, רק הוא יתבצע!

עבור כל Packet שמגיע נוכל לבחור כיצד המערכת תגיב מתוך חמשת התגובות הבאות:

- **ACCEPT**- המערכת תאפשר ל-Packet שהתקבל לעבור.
- **DROP**- המערכת לא תתייחס ל-Packet שהתקבל.
- **REJECT**- המערכת לא תתייחס ל-Packet שהתקבל אך תשלח שגיאה למקור ה-Packet.
- **QUEUE**- המערכת תעביר את חבילת המידע לשימוש ב-Userspace.
- **RETURN**- המערכת תחזור אחורה לחוק שממנו הופעל החוק הנוכחי.

מתחילים

בכדי להבין כיצד לכתוב או לקרוא את החוקים הנמצאים בטבלאות, נשתמש במקרים נפוצים (יותר ופחות) ונראה איך ניתן להגיב אליהם וכיצד ליישם את התגובה בעזרת ה-IPTables. טבלת ברירת המחדל שלנו היא filter ולכן, אם לא נציין שום טבלה, המידע יכנס או ישלף מהטבלה הזאת.

תוכן הטבלאות

דבר ראשון, כתבו בקונסול:

```
iptables -t filter -L INPUT
```

הפקודה הנ"ל אומרת ל-IPTable להציג את כל החוקים שקיימים בשרשרת INPUT בטבלת filter.

שימו לב שאם תכתבו:

```
iptables -L INPUT
```

תקבלו את אותה התוצאה, היות ו-filter היא ברירת המחדל.

אם לא נגעתם ב-iptables עד עכשיו, מרבית הסיכויים הם שתראו את הפלט הבא:

```
Chain INPUT (policy ACCEPT)
target           port opt source   destination
```

הטבלה ריקה, ואלה ראשי העמודות:

- **TARGET** - מה יעלה בגורל ה-Packet (לאן הוא ישלח)
- **PORT** - באיזה פורט משתמש הפקט.
- **OPT** - אפשרויות שונות בהן נגע בהמשך.
- **SOURCE** - מאיפה נשלח ה-Packet.
- **DESTINATION** - לאן הוא נשלח.

במרבית המקרים המערכת תפלוט לכם כי אין לכם גישה ל-IPTables, במקרים כאלה תאלצו להתחבר לחשבון בעל הרשאות מערכת, או לבצע את הכל תחת sudo.

כתיבת החוקים

כל חוק שנכתוב, אם לא נאמר בפירוש לאיזו שורה אנחנו רוצים לכתוב אותו, יכתב בסוף השרשרת.

חוק ראשון- אנחנו רוצים לנתק את כל תעבורת האינטרנט הנכנסת למחשב, כיתבו:

```
Iptables -A INPUT -j DROP
```

והנה פירוט של מה שעשינו, הצבענו על השרשרת INPUT בעזרת:

```
-A INPUT
```

וקבענו את גורל ה-Packets שמגיעים לשם, בעזרת:

```
-j DROP
```

*לא קבענו שום סינון ולכן החוק יחול על כל ה-Packets.

שימו לב, בעזרת המתג **-j** נקובעים מה יעלה בגורל ה-Packets שמקיימים את אותו החוק, במקרה שלנו קבענו DROP, יכולנו לקבוע כל גורל אחר.

כדי לבחון איך החוק נשמר, נכתוב שוב:

```
iptables -L INPUT
```

ונקבל את הפלט הבא:

```
Chain INPUT (policy ACCEPT)
target            port  opt   source            destination
DROP              0     --    anywhere          anywhere
```

שימו לב:

קבענו רק מה יעלה בגורל ה-Packet ולא קבענו שום מסננים, לכן תחת TARGET הוכנס ה-DROP ותחת כל השאר- הוכנס anywhere (פורט 0 אומר "כל הפורטים").

איך נבדוק שהחוק באמת פועל? שלחו פינג לגוגל ותראו מה קורה, כתבו:

```
 ping www.google.com
```

לאחר שיעבור ה-timed_out, נקבל:

```
ping: unknown host www.google.com
```

הבה נמחק חוק זה ונמשיך הלאה, כיתבו:

```
Iptables -D INPUT 1
```

ושוב, כיתבו:

```
iptables -L INPUT
```

ונראה שוב- טבלה ריקה. שימו לב שבעזרת:

-D INPUT 1

אמרנו ל-IPTables שאנחנו רוצים למחוק את חוק מספר 1 משרשרת INPUT. פשוט ביותר. עכשיו נשלח שוב פינג לגוגל רק בשביל לבדוק שהכל פועל כשורה:

```
ping www.google.com
```

ונוכל לראות לפי הפלט שאנחנו אכן מקבלים echo מגוגל:

```
PING www.l.google.com (74.125.39.106) 56(84) bytes of data.
64 bytes from fx-in-f106.google.com (74.125.39.106): icmp_seq=1 ttl=241 time=73.4 ms
64 bytes from fx-in-f106.google.com (74.125.39.106): icmp_seq=2 ttl=241 time=74.6 ms
64 bytes from fx-in-f106.google.com (74.125.39.106): icmp_seq=3 ttl=241 time=72.3 ms
64 bytes from fx-in-f106.google.com (74.125.39.106): icmp_seq=3 ttl=241 time=74.5 ms

64 bytes from fx-in-f106.google.com (74.125.39.106): icmp_seq=1 ttl=241 time=76.0 ms
--- www.l.google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4003ms
Rtt min/avg/max/mdev = 72.324/74.211/76.076/1.298 ms
```

אגב, מומלץ לעבוד עם Packet Sniffer על מנת להבין טוב יותר בדוגמאות הבאות.

כעת נראה כיצד ניתן לאפשר רק ל-Packet בעל אופי מסויים להכנס למחשב, נניח, רק Packets מפורט 80 יוכלו להכנס למערכת שלנו, וכל השאר ילכו לפח. כך נאפשר רק לגלוש באינטרנט, אבל לא להשתמש בשאר התוכנות, כמו ICQ, IRC, FTP וכו', כיתבו:

```
iptables -A INPUT -p tcp --sport 80 -j ACCEPT
```

החוק די מובן: "Packets על-גבי פרוטוקול TCP מ-80 יכנסו למערכת". בעזרת:

```
-p tcp --sport 80
```

קבענו שהמאפיין שיבדק על ידי החוק שלנו הוא source-port של ה-Packet, ואם הוא יהיה שווה ל-80 על גבי פרוטקול ה-TCP נקבל את ה-Packet.

נסו להכנס עם הדפדפן לאיזה אתר, האם הכל עובד? הפעילו את קליינט ה-IRC האהוב עליכם, ו-גם עובד! מוזר, לא? בואו נראה מה עשינו, אמרנו למערכת שתאפשר ל-Packets מפורט 80 להכנס, אך מה בדבר שאר ה-Packets? הם לא מקיימים שום חוק בשום פילטר ולכן הם עוברים. מה שאנחנו צריכים לעשות זה להוסיף את השורה הבאה:

```
Iptables -A INPUT -j DROP
```

את השורה הזאת אנחנו מכירים, היא חוסמת את כל ה-Packets הנכנסים למערכת שלנו.

רק רגע.. זה לא מתנגש עם החוק הקודם? זה מתנגש בהחלט, אבל כמו שאמרתי בתחילת המאמר- המערכת תחפש את החוק **הראשון** שמאפייני הפאקים מקיימים ואחרי שהיא תמצא אחד כזה, היא תפסיק לחפש ותבצע את הפעולה, מה שאומר שאם מגיעים למערכת שלנו Packets מפורט 80, הם יענו על החוק הראשון ולכן הם יגיעו אלינו. לעומת זאת, Packets שלא מגיעים מפורט 80 לא יענו על החוק

הראשון, ולכן המערכת תבדוק האם החוק השני חל עליהם- אנחנו כבר יודעים שהחוק השני חל על **כל ה-Packets הנכנסים** ולכן מאפייני ה-Packet אכן יענו עליו- מה שיוביל לכך שהמערכת תבצע את מה שהחוק אומר- להתעלם ממנו. כיתבו:

```
iptables -L INPUT
```

אתם אמורים לקבל:

```
Chain INPUT (policy ACCEPT)
target      port        opt         source      destination
ACCEPT      tcp         anywhere    anywhere    tcp spt:www
DROP        0           --          anywhere    anywhere
```

נניח ואנחנו רוצים להוסיף עוד חוק שיקבע שמהמחשב שלנו יהיה אפשר להתחבר גם לשרתי IRC, רובם כיום משתמשים בפורט 6667. אנחנו מכירים את הפקודה לאפשר ל-Packet מפורט מסויים להכנס למערכת, אנחנו רק צריכים לשנות אותה ל-6667:

```
iptables -A INPUT -p tcp --sport 6667 -j ACCEPT
```

אבל אנחנו גם יודעים שכל פעם שאנחנו מכניסים חוק חדש לטבלה, אותו החוק הוא יכנס לסוף הטבלה, כך שאם טבלת ה-INPUT שלנו נראת ככה:

```
target      port        opt         source              destination
ACCEPT      tcp         anywhere    anywhere            tcp spt:www
DROP        0           --          anywhere            anywhere
```

אנחנו יודעים שהחוק שנקבע לאפשור Packets בפורט 6667 לא יהיה בר תוקף היות והוא יכנס לאחר החוק שאומר לזרוק את כל ה-Packets. לפיכך, אנחנו צריכים להכניס אותו לפני אותו החוק, בשורה הראשונה או השניה, אך בשום אופן לא בשורה השלישית.

כיצד כותבים חוק לשורה ספציפית? פשוט מאוד:

```
iptable -I INPUT 1 -p tcp --sport 667 -j ACCEPT
```

החלפנו את: **A INPUT-** שאומר "Append to chain" ב: **I INPUT 1-** שאומר "Insert in chain as rulenum" - וקבענו ש-rulenum יהיה שווה ל-1, משמעות הדבר שהשורה תכנס כשורה ראשונה ותוריד את שאר השורות הקיימות למטה. כעת, אם נסתכל בטבלה שלנו, נראה שהיא בנוייה באופן הבא:

```
target      port  opt   source      destination
ACCEPT      tcp         anywhere    anywhere    tcp spt:ircd
ACCEPT      tcp         anywhere    anywhere    tcp spt:www
DROP        0     --    anywhere    anywhere
```

מה שאומר שרק Packets שיכנסו מה-ircd (6667) וה-www (פורט 80) יכנסו למערכת, ולכל השאר המערכת תבצע DROP. כעת, אם נרצה לשכתב שורה קיימת בטבלה, נחליף את המתג I במתג R, והרעיון אותו רעיון- כותבים את מספר השורה שאותה רוצים לשכתב.

נניח ואנחנו לא רוצים לאפשר 6667 אלא לאפשר גישת SSH למחשב שלנו. SSH רץ לרוב על פורט 22, כך שעלינו רק לשנות בשורה את 6667 ל-22, השורה של 6667 (ה-ircd) היא השורה הראשונה, אז נכתוב את הפקודה באופן הבא:

```
iptables -R INPUT 1 -p tcp --sport 22 -j ACCEPT
```

כעת, אם נסתכל על הטבלה, נראה שהיא עודכנה:

```
target            port  opt   source            destination
ACCEPT      tcp         anywhere    anywhere    tcp spt:ssh
ACCEPT      tcp         anywhere    anywhere    tcp spt:www
DROP        0     --    anywhere    anywhere
```

ב-spt, שונה הערך מ-ircd, ל-ssh, מה שיאפשר לכל משתמש שירצה לגשת לפורט 22.

נעבור לעוד אפשרות- מי אמר שאנחנו רוצים לאפשר לכל אחד לגשת לפורט 22? אולי נרצה להגדיר רק לכתובת IP מסויימת או לטווח כתובות IP? (מאוד לא בריא לאפשר לכל אחד לגשת ל-ssh אפילו אם נדרשת סיסמה בכדי להכנס). כשמשתמשים באפשרות הזאת, יש לזכור כי כתובות IP הן לא סטטיות ומשתנות בכל פעם שמתבצע חיוג ל-ISP. נכון שכיום גובר השימוש ב-Static IP או בחיבורי DHCP, אבל עדיין, צריך לזכור את זה כאשר משתמשים באימות מבוסס IP.

מחקו את כל החוקים שכתבנו עד עכשיו בשרשרת, על ידי:

```
Iptables -F INPUT
```

(אם לא נכתוב את שם השרשרת באופן שמצויין כאן, פקודה זו תמחק את כל החוקים הקיימים בכל השרשראות, אז שימו לב!)

כיתבו את החוק הבא:

```
iptables -A INPUT -s XXX.XXX.XXX.XXX -p tcp --sport 22 -j ACCEPT
```

אנחנו כבר מכירים כמעט את כל המתגים המרכיבים את החוק הזה, הדבר היחיד שהוספנו הוא:

```
-s XXX.XXX.XXX.XXX
```

כפי שבוודאי הבנתם, מדובר במאפיין ה-Source IP - כתובת ה-IP (או הדומיין) שממנה נשלח אלינו ה-Packet. כמובן שחוק זה לבדו לא יעזור לנו היות ובעצם לא נמנעה שום גישה לשאר האנשים, לכן יש להוסיף גם את החוק הבא:

```
iptables -A INPUT -p tcp --sport 22 -j DROP
```

אם נרצה לאפשר ל-SUBNET ספציפי, נוכל לכתוב את החוק באופן זה:

```
iptables -A INPUT -s XXX.XXX.XXX.XXX/YYY -p tcp --sport 22 -j ACCEPT
```

אפשרות זו כבר מוכרת לנו - זרוק לפח כל נסיון גישה לפורט 22 על גבי פרוטוקול ה-tcp. חשוב לציין כי ניתן לבצע את הפעולה הזאת גם על ידי חוק אחד בלבד:

```
iptables -A INPUT -s ! XXX.XXX.XXX.XXX -p tcp --sport 22 -j DROP
```

נראה שהוא לבד יכול לבצע את הסינון, דבר ראשון, הוספנו סימן קריאה לפני כתובת ה-IP, מה שאומר כאן כמו כמעט בכל שפות התיכנות- "NOT", ודבר שני, שינינו את גורל ה-Packet ל-"DROP". חוק זה אומר "זרוק לפח כל Packet שמאפיין ה-Source-IP שלו **לא שווה** ל:XXX.XXX.XXX.XXX שמנסה לגשת לפורט 22 על גבי הפרוטוקול TCP." לפיכך, רק ה-IP שקבענו יוכל לגשת לפורט 22 (כי הוא לא מתאים למאפיין שבפילטר) ושאר האנשים- לא.

באופן כללי אין נכון או לא נכון בכתיבת חוקים, אם החוק מבצע את תפקידו כהלכה הוא נכון, אך כל משתמש אוהב ורואה לנכון לבצע מספר פעולות בדרכים שונות.

מאפייני ה-Packet שנגענו בהם עד כה הם:

- **p** - ה-**Protocol** בו משתמש ה-Packet.
- **sport** - ה-**Source Port** ממנו הגיע ה-Packet.
- **s** - ה-**Source IP/Domain** ממנו הגיע ה-Packet.

קיימים מספר מאפיינים נוספים הנוגעים לטבלת ה-**INPUT** שבהם אפשר להשתמש:

- **dport** - מאפיין המצביע על ה-**Destination Port** - הפורט אליו נשלח ה-Packet.
- **i** - ממשק (**In interface**), כרטיס הרשת ממנו הגיע ה-Packet (ppp, wlan, eth וכו').
- **syn** - מאפיין הבודק האם ה-Packet מאופיין כ-**syn-packet**.

בכדי להמחיש דוגמה לשימוש ב-**i** וב-**syn** נשתמש במקרה בו אנחנו לא רוצים שמחשבים חיצוניים יכלו לפתוח חיבור איתנו דרך eth1, אך בכל זאת רוצים לאפשר לעצמנו להתחבר אליהם. משמעות הדבר היא שאם אנחנו נבקש ליצור חיבור- הוא יתאפשר, אך אם מישהו חיצוני יבקש ליצור חיבור הוא ידחה על ידי המערכת. נכתוב את השורה הבאה:

```
iptables -A INPUT -p tcp -i eth1 --syn -j DROP
```

חוק זה הוא פשוט להבנה: כל syn שיגיע מרכיב ה-eth1 יזרק לפח. שימו לב שהחוק שונה מהחוק הראשון שלמדנו:

```
iptables -A INPUT -j DROP
```

החוק הראשון מונע מכל Packet לעבור ולכן תעבורת הרשת הנכנסת למערכת שלנו תחסם. אם נרצה, נכון להוציא מידע מהמחשב נוכל, אך לא נוכל לקבל את התוצאות מפני שחסמנו את כל תעבורת הרשת

פנימה. לעומת זאת, בחוק הנוכחי תעבורת האינטרנט פנימה לא תחסם, אלא רק Packets אשר נועדו להגיד למערכת שהם מעוניינים לפתוח חיבור TCP.

נגענו מספיק בטבלת ה-INPUT, הבה נראה קצת דוגמאות מטבלת ה-OUTPUT. הרעיון הכללי הוא אותו רעיון וברוב הדגלים משתמשים בלא שום שינוי. לדוגמא, חסימת כל תעבורת הרשת החוצה, תתבצע ע"י הפקודה הבאה:

```
iptables -A OUTPUT -j DROP
```

חסימת כל תעבורת הרשת היוצאת, חוץ מפורט 25 (SMTP לשליחת דואר בין מערכות) תתבצע כך:

```
iptables -A OUTPUT -p tcp --dport ! 25 -j DROP
```

שימו לב, שכשמדובר בתעבורה יוצאת, משתמשים ב-dport, שמצביע על Destination-Port, ולא ב-sport שמצביע על Source Port.

חסימת ה-Packets היוצאים מממשק eth1 שלנו, תתבצע עם שינוי דומה:

```
iptables -A OUTPUT -p tcp -o eth1 -j DENY
```

שוב, שימו לב שהשימוש בממשק הרשת בוצע על ידי המתג: **o-** שאומר "Out Interface", ולא על ידי: **i-** שאומר "In Interface".

כמה דברים שחשוב לדעת

אם נרצה לעקוב אחרי Packets מסויימים, נוכל להגדיר ל-IPTables לשמור ב-LOG Packets ספציפים. לדוגמא, אם נרצה לבצע מעקב אחרי כל לקוחות שירות ה-SSH שהתקנו על השרת, נדע שלשם כך אנחנו צריכים לשמור את המידע שמגיע אלינו בפורט 22 על גבי ה-TCP ונבצע זאת על ידי הפקודה הבאה:

```
Iptables -A INPUT -tcp --sport 22 -j LOG --log-prefix "SSH CONNECTION"
```

אין הרבה חדש בפקודה זו, השורה לא קובעת אם ה-Packet יתקבל או לא יתקבל, אלא פשוט אומרת למערכת לשמור את המידע הזה בקובץ בכדי שבשלב מאוחר יותר נוכל לראות מה בדיוק קרה במהלך ההתחברות.

קבענו ש"גורל ה-Packet" ילך ל-LOG, החלק של ה-"log-prefix--" ומה שמגיע אחריו בגרשיים יכנס כהערה בשורה בכדי שנדע בזמן ניתוח הלוג מה שייך למה.

כברירת המחדל של ה-IPTables, קבצי הלוגים נשמרים במיקום:

```
/var/log/messages
```

אך אם תרצו לשנות את המיקום בו ישמרו הקבצים, תוכלו לגשת ל:

```
kate etc/syslog.conf
```

ותוסיפו את השורה הבאה:

```
kern.warning /var/log/your_file
```

כמובן שתאלצו להפעיל מחדש את מנגנון ה-syslog. אני משתמש ב-Ubuntu, הפקודה היא:

```
/etc/init.d/sysklogd restart
```

ב-Fedora, Redhat ודומיהן, הפקודה קצת שונה:

```
/etc/init.d/syslog restart
```

חשוב לזכור כי אם נכבה את המחשב, כל החוקים שכתבנו ימחקו . כדי לשמור את החוקים שכתבנו נפעל כך: נשמור את ההגדרות שכתבנו בקובץ חיצוני:

```
iptables-save > your-rules-file
```

לאחר מכן נקבע למערכת להפעיל את הפקודה:

```
 ptables-restore < your-rules-file
```

בכל פעם שהמערכת עולה, על ידי הוספת הפקודה הזאת, לקובץ: **etc/init.d/networking** בסוף הסקטור **start**. דרך זו אמנם אינה אידיאלית או הכי נוחה, אך היא ללא ספק היעילה ביותר.

סיכום

עד כאן הפרק הראשון בסדרת מאמרים זו. כפי שראיתם, נגענו רק בקצה המערכת ועדיין השגנו ידע המאפשר לנו לקנפג את מערכת ה-Filtering Table בצורה שתוסיף די אבטחה לשרת/מחשב שלנו, בהמשך הסידרה נגע בטבלאות מתקדמות יותר ומתוחכמות מ-INPUT.

נקודה אחרונה: שימו לב שיש מחרוזות שנכתבו באותיות גדולות, כמו למשל: ACCEPT, DENY, INPUT, ויש מחרוזות שנכתבו באותיות קטנות, כמו למשל: sport, tcp. הרעיון הוא נובע מכך, כפי שאתם כנראה יודעים, שמספר מערכות, וביניהן לינוקס, רגישות לאותיות קטנות ולאותיות גדולות (Case Sensitive), כך ש-ACCEPT לא שווה ל-Accept או ל-AccepT. לפיכך יש תמיד לזכור מה כותבים באותיות קטנות או גדולות, זכרון זה מגיע עם תרגול ועם הזמן.

לקריאה נוספת

- האתר של הפרוייקט Netfilter, כולל עידכונים, כלים ועוד הרבה: http://www.netfilter.org/
- מדריך ענקי של Oskar Andreasson על IPTables: http://iptables-tutorial.frozentux.net/iptables-tutorial.html
- ממשק ויז'ואלי ל-IPTables התומך בהרבה מאוד רכיבים: http://www.fwbuilder.org/

תכנות גנרי בשפת C#

מאת ניר אדר (UnderWarrior)

דוגמת פתיחה ומוטיבציה

תכנות גנרי הופיע לאורך השנים בשפות מונחות עצמים רבות. הרעיון הוא מימוש אלגוריתמים שאינם תלויים בטיפוס הנתונים עליהם הם פועלים.

באופן רגיל, כאשר אנחנו כותבים קוד בשפה Strongly Typed (כדוגמת C#, Java), אנחנו מגדירים בצורה מפורשת את הטיפוס עליו אנו עובדים, int למשל, ויוצרים קוד שמתאים לסוג טיפוס זה. לעומת זאת, כשאנחנו כותבים אלגוריתמים כלליים, אנחנו מסוגלים לתאר את הפעולות מבלי להזדקק לציון הסוג.

דוגמה קלאסית למקרה זה היא הפונקציה swap. פונקציה זו מקבלת שני משתנים מסוג int ומחליפה בין ערכיהם. נדגים את הפונקציה ופונקצית main המשתמשת בה:

```
static void swap(ref int i, ref int j)
{
      int temp = i;
      i = j;
      j = temp;
}

static void Main(string[] args)
{
      int i = 5, j = 7;
      Console.WriteLine("i = " + i.ToString() + ", j = " +
j.ToString());
      swap(ref i, ref j);
      Console.WriteLine("i = " + i.ToString() + ", j = " +
j.ToString());
}
```

הפלט הוא:

```
i = 5, j = 7
i = 7, j = 5
```

swap מחליפה בין ערכי המשתנים שהיא מקבלת על ידי שמירת ערכו של אחד מהם במשתנה זמני ולאחר מכן מעבירה את הערך שהיה במשתנה אחד אל המשתנה השני, והערך שהיה בשני אל המשתנה הראשון.

איך תראה הפונקציה אם נרצה להתאימה למשתנים מסוג double? זהה לחלוטין! רק שנצטרך לכתוב double בכל מקום שבו כרגע כתוב int:

```
static void swap(ref double i, ref double j)
{
      double temp = i;
      i = j;
      j = temp;
}
```

דבר זה נכון למעשה עבור כל סוג של משתנים שנרצה לכתוב עבורם פונקצית swap - הגוף יראה זהה ורק סוג המשתנה ישתנה. אנחנו רוצים למצוא דרך יעילה לכתוב זאת בלי לשכפל את הקוד שלנו פעמים רבות. (משפט ידוע המלווה שפות רבות אומר כי קוד המופיע פעמיים - מופיע פעם אחת יותר מדי).

הפתרון הינו תכנות גנרי, תכנות שבו איננו מגדירים את סוג המשתנה אלא מציינים "סוג כלשהו", הקוד בשפת C# יראה כך:

```
static void swap<T>(ref T i, ref T j)
{
      T temp = i;
      i = j;
      j = temp;
}
```

אנחנו בעצם יוצרים תבנית, וכשנקרא ל-swap השפה תזהה את טיפוס המשתנים שלנו ותיצור פונקציית swap המתאימה לטיפול בערכים אלה. הגנריות מבחינת תחביר מסומנת על ידי <T> - אנחנו מציינים ש-T הוא סוג כללי כלשהו.

אם נחליף את פונקצית ה-swap בדוגמת הפתיחה בפונקציה זו התוכנית תעבוד ללא שינוי. בנוסף, אם פתאום נצטרך פונקצית swap בין double, לא נצטרך לכתוב פונקציה מיוחדת באופן מפורש - C# תדאג לתבנית מתאימה גם עבור מקרה זה.

מקום קלאסי נוסף בו משתמשים בתכנות גנרי הוא Collections - רשימות, עצים ושאר מבני הנתונים - הרי מבנה הנתונים (והפעולות עליו) נשארים זהים תמיד. סוגי הנתונים הנשמרים בכל פעם משתנים ולכן מימוש של מבני נתונים הוא דוגמה שימושית וחשובה לצורך שלנו בתכנות גנרי.

קצת היסטוריה

תכנות גנרי קיים בצורות שונות עוד מ-1970. בהמשך, שפת C++ הציגה תכנות גנרי בצורת תבניות (templates) - מבנה שמבחינת תחביר זהה מאוד לפתרון לתכנות גנרי של C# המכונה generics.

בשפת C# במקור לא היתה תמיכה בתכנות גנרי, מכיוון שכל המחלקות נגזרות מהאובייקט object, היה ניתן ליצור לצורך העניין רשימה כללית שתכיל אובייקטים מכל סוג שהוא ללא צורך בתבניות. עם זאת, ל-generics יש מספר יתרונות שגרמו למעצבי השפה להכניס אותם לשפה החל מ-Net 2005.

- **Type Safety -** במקרים רבים נרצה ליצור רשימה שיש לה type checking, שנקבל הודעת שגיאה בזמן קומפילציה במידה ומנסים להכניס אליה אובייקט מסוג לא נכון, ושנדע שאנחנו יכולים להשוות בין כל האובייקטים השונים הנמצאים באותה הרשימה. לדוגמא, ללא תכנות גנרי הקוד הבא היה עשוי לגרום לשגיאה בזמן ריצה:

```
ArrayList intList = new ArrayList();
// some ops on the list here
// ...
foreach (int x in intList)
{
      // some comments here
}
```

במקרה ובו לא כל האיברים ברשימה יהיו int, התוכנית תקרוס. אנחנו נהיה מעוניינים למנוע מצב כזה ולדאוג שאם ננסה להכניס איבר שאינו int נקבל שגיאה עוד בזמן הקומפילציה.

- **ביצועים -** חיסכון בפעולות Boxing/Unboxing. כל פעם שאנחנו שומרים מחלקה/ערך בתוך מבנה נתונים כללי - הוא עובר פעולת boxing, וכשמוציאים אותו הוא עובר פעולת unboxing. אלו פעולות יקרות שפוגעות משמעותית ביעילות מבנה הנתונים. שימוש ברשימה המיועדת ל-int בלבד, למשל, יכול למנוע את ביצוע פעולות אלו ולשפר משמעותית את ביצועי התוכנית.

- **Code reuse -** כדי להשיג פונקציונליות של type safety ללא Generics אנחנו צריכים מבנה נתונים נפרד לכל סוג, למשל רשימה מקושרת של int, רשימה מקושרת של char וכדומה. ראינו למשל בדוגמאת swap שללא generics היינו צריכים לכתוב פונקציה נפרדת עבור int, פונקציה אחרת ל-double, וכו'. Generics מאפשר לנו למנוע את שכפול הקוד ולכתוב את הפונקציה פעם אחת. נושא זה חשוב במיוחד כשאנחנו כותבים פונקציה מעט מורכבת יותר. אם הפונקציה משוכפלת, לעדכן את כל העותקים שלה יכול להיות סיוט ופתח משמעותי לשגיאות.

השם generics בא לציין שאנחנו מגדירים מחלקות ומתודות גנריות. (בשפות אחרות השם היה templates, תבניות, ובא להגיד שיוצרים תבנית, ולא פונקציה "אמיתית". שפת C# בחרה להדגיש בשם את הרעיון שאנחנו באים לתכנת משהו כללי שהוא לא ספציפי לסוג מסוים.

כמו שכבר ציינתי מקודם, השימוש העיקרי שעושים ב-generics בשפה הוא ב-collections. C# 2.0 הציגה ספריית מחלקות חדשה, תחת המסלול System.Collections.Generic הממומשות באמצעות generics להשגת היתרונות שמנינו. **מומלץ להשתמש במחלקה זו בכל מקום אפשרי במקום הספריות הישנות מתוך System.Collections.**

מתודות גנריות

ב-C# ניתן להגדיר מתודות גנריות. דוגמה למתודה גנרית ולשימוש בה נראית כך:

```
using System;

class GenericMethod
{
	static void Main(string[] args)
	{
		// create arrays of int and double
		int[] intArray = { 1, 2, 3, 4, 5, 6 };
		double[] doubleArray =
			{ 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };

		// pass an int array argument
		PrintArray(intArray);

		// pass a double array argument
		PrintArray(doubleArray);
	}

	// output array of all types
	static void PrintArray<E>( E[] inputArray )
	{
		foreach (E element in inputArray)
			Console.Write( element + " " );
		Console.WriteLine( "\n" );
	}
}
```

הפונקציה PrintArray<E> היא פונקציה גנרית (סוג המשתנה מסומן כ-E). בכל פעם הפונקציה מדפיסה מערך מסוג אחר.

נביט בדוגמה ונציג מספר נקודות חשובות בה:

1. **הקריאה לפונקציה הגנרית:** בניגוד ל-C++, הקריאה לפונקציה נעשית על ידי שמה, בלי שצריך לציין באופן מפורש מהו E בקריאה לפונקציה, כלומר נכתוב PrintArray(intArray) והשפה תנחש לבד את הסוג בו משתמשים. אם נרצה, ניתן גם לציין את הסוג במפורש, למשל לכתוב PrintArray<int>(intArray) על מנת להגיד לשפה שאנחנו מעוניינים להשתמש בגרסת הפונקציה עם הסוג int. יש צורך לציין במפורש את הסוג גם במקרים בהם C# לא תצליח לנחש אותו לבד, ותשלח הודעת שגיאה

2. **לא כל המחלקה חייבת להיות גנרית**: הבדל נוסף משפת C++ הוא שלא כל המחלקה חייבת להיות גנרית כדי שנוכל להשתמש בה במתודות גנריות - מתודה ספציפית במחלקה יכולה להיות גנרית בזמן שהמחלקה הינה מחלקה רגילה.

3. **הפרמטר יכול להיות מיוצג על ידי כל מזהה**, ולאו דווקא T או E. מקובל שהפרמטר הוא באותיות גדולות ולעתים מקובל להשתמש באותיות מסויימות (T עבור Type, K עבור Key וכדו')

מחלקה גנרית

מחלקה שלמה יכולה להיות גנרית, כלומר לקבל סוג גנרי שעובר לאורך כל המחלקה. הדוגמה הקלאסית היא דוגמת ה-Collections, למשל יצירת רשימה כללית ולאחר מכן יצירת רשימה של int. באמצעותה.
מבחינת התחביר יש לחלק את התחביר לשני חלקים- התחביר הנדרש לצורך כתיבת מחלקה גנרית, והתחביר הנדרש לצורך שימוש במחלקה הגנרית שהגדרנו.

מבחינת התחביר ליצירת מחלקה גנרית:

- הפרמטר T מופיע בסוגריים משולשות שם המחלקה. בדומה לשפת C++, T הוא אותו T בכל חלקי המחלקה.
- בשאר המתודות של המחלקה לא צריך לשים סוגריים משולשות לציין שמדובר בתבנית - כל הפונקציות הן גנריות בהינתן שהן חלק ממחלקה גנרית.

מבחינת התחביר לשימוש במחלקה הגנרית:

- כאשר יוצרים מופע של המחלקה, עלינו לציין כחלק משם המחלקה מהו הסוג שאנחנו יוצרים.

הקוד הבא מציג רשימה גנרית בסיסית ומדגים את התחביר. הסוג של הנתונים הינו T, סוג כלשהו:

```
// type parameter T in angle brackets
public class GenericList<T>
{
      // The nested class is also generic on T.
      private class Node
      {
            // T used in non-generic constructor.
            public Node(T t)
            {
                  next = null;
                  data = t;
            }

            private Node next;
            public Node Next
            {
                  get { return next; }
                  set { next = value; }
            }

            // T as private member data type.
            private T data;

            // T as return type of property.
            public T Data
            {
                  get { return data; }
                  set { data = value; }
            }
      }

      private Node head;

      // constructor
      public GenericList()
      {
            head = null;
      }

      // T as method parameter type:
      public void AddHead(T t)
      {
            Node n = new Node(t);
            n.Next = head;
            head = n;
      }
```

```
	public IEnumerator<T> GetEnumerator()
	{
		Node current = head;

		while (current != null)
		{
			yield return current.Data;
			current = current.Next;
		}
	}
}
```

שימוש לדוגמה במחלקה:

```
class TestGenericList
{
	static void Main()
	{
		// int is the type argument
		GenericList<int> list = new GenericList<int>();

		for (int x = 0; x < 10; x++)
		{
			list.AddHead(x);
		}

		foreach (int i in list)
		{
			System.Console.Write(i + " ");
		}
		System.Console.WriteLine("\nDone");
	}
}
```

הגדרנו כי GenericList היא מחלקה גנרית עם פרמטר T, שהוא סוג הנתונים הנשמר במחלקה. הגדרנו מחלקה פנימית Node לשמירת צומת ברשימה, בתור חלק ממחלקה גנרית, המחלקה Node גנרית באופן אוטומטי.

מגבלות על הפרמטרים

שפת C# מאפשרת לנו להגדיר מגבלות על הפרמטרים הגנרים אותם נקבל (T לא יהיה מסוג כלשהו, אלא יהיה חייב לעמוד במגבלות שנדרוש). המגבלות ש-C# מאפשרת לנו להגדיר (לקוח מהגדרת השפה ב-MSDN):

Constraint	Description
where T: struct	The type argument must be a value type. Any value type except Nullable can be specified. See Using Nullable Types (C# Programming Guide) for more information.
where T : class	The type argument must be a reference type, including any class, interface, delegate, or array type.
where T : new()	The type argument must have a public parameterless constructor. When used in conjunction with other constraints, the new() constraint must be specified last.
where T : <base class name>	The type argument must be or derive from the specified base class.
where T : <interface name>	The type argument must be or implement the specified interface. Multiple interface constraints can be specified. The constraining interface can also be generic.
where T : U	The type argument supplied for T must be or derive from the argument supplied for U. This is called a naked type constraint.

לדוגמא, נגדיר מחלקה המייצגת עובד ורשימה המטפלת בעובדים:

```
public class Employee
{
	private string name;
	private int id;

	public Employee(string s, int i)
	{
		name = s;
		id = i;
	}

	public string Name
	{
		get { return name; }
		set { name = value; }
	}

	public int ID
	{
		get { return id; }
		set { id = value; }
	}
}

public class GenericList<T> where T : Employee
{
	private class Node
	{
		private Node next;
		private T data;

		public Node(T t)
		{
			next = null;
			data = t;
		}

		public Node Next
		{
			get { return next; }
			set { next = value; }
		}

		public T Data
		{
			get { return data; }
			set { data = value; }
		}
	}
```

```
        private Node head;

        public GenericList() //constructor
        {
                head = null;
        }

        public void AddHead(T t)
        {
                Node n = new Node(t);
                n.Next = head;
                head = n;
        }

        public IEnumerator<T> GetEnumerator()
        {
                Node current = head;

                while (current != null)
                {
                        yield return current.Data;
                        current = current.Next;
                }
        }

        public T FindFirstOccurrence(string s)
        {
                Node current = head;
                T t = null;

                while (current != null)
                {
                        //The constraint enables access to the Name property.
                        if (current.Data.Name == s)
                        {
                                t = current.Data;
                                break;
                        }
                        else
                        {
                                current = current.Next;
                        }
                }
                return t;
        }
}
```

הגדרנו מחלקה בשם Employee, ואז הגדרנו מחלקה שהיא רשימה שמקבלת רק אובייקטים מסוג Employee או מחלקות הנורשות ממנה (את זאת דרשנו באמצעות ההגבלה על הסוג). נשים לב שברגע שהגדרנו הגבלה זו, אנחנו מסוגלים לגשת לשדות של Employee בתוך המתודות של המחלקה. ההגבלה מאפשרת לנו כוח ביטוי נוסף, מכיוון שיש לנו ידע נוסף על הסוג.

דוגמה לתחביר - מחלקה עם מספר פרמטרים גנריים ועם הגבלות שונות על כל פרמטר:

```
class SuperKeyType<K, V, U>
    where U : System.IComparable<U>
    where V : new()
{ }
```

ממשקים גנריים

גם ממשקים (interface) מסוגלים להיות גנריים. צורת הכתיבה שלהם דומה לצורת הכתיבה של מחלקות גנריות:

```
interface IDictionary<K, V>
{
}
```

ממשק גנרי המקבל שני טיפוסים כלשהם. הממשקים Icomparable, Ienumerable קיבלו גרסאות גנריות גם הם, כדי למנוע פעולות של Boxing ו-Unboxing. כמו ב-Collections, גם בהם מומלץ להשתמש בכל מקום בו ניתן.

דוגמה נוספת למקרה בו כותבים איסורים מרובים ומשתמשים גם בממשקים הגנריים:

```
class Stack<T> where T : System.IComparable<T>, IEnumerable<T>
{
}
```

ערך ברירת מחדל

בעיה שעולה כשאנחנו רוצים לאתחל משתנה גנרי היא אתחול ערך "ברירת מחדל" לערך הגנרי. ערכי ברירת מחדל למשתנים רגילים מוגדרים ב-C# כך- אם המשתנה מסוג int, הערך הוא 0, אם המשתנה הוא מספר עשרוני הערך הוא 0.0, ואם המשתנה הוא מחלקה אזי ערך ברירת המחדל הוא null. (ערכים אלו מושמים באופן אוטומטי למשתנים אם אנחנו מגדירים אותם ללא קביעת ערך).

הבעיה בעת שימוש במשתנה generic נובעת מכך שאנחנו אפילו לא יודעים אם נקבל ערך שהוא by value או reference.by.הפתרון של שפת C# הוא שימוש במילת המפתח default במשמעות חדשה של החזרת ערך ברירת המחדל עבור הסוג המועבר:

```
// The following declaration initializes temp to
// the appropriate  default value for type T.
T temp = default(T);
```

סיכום

במאמר זה נגעתי על קצה המזלג בתכנות גנרי בשפת C#. הנקודה החשובה ביותר שרציתי להעביר במסמך זה היא קיומן של אפשרויות תכנות אלו והשימושיות שלהן. אם תרצו להשאר עם דגש אחד ממאמר זה, הדגש הוא להשתמש רק ב-collections המומשים בעזרת תכנות גנרי - יעילות התוכניות שלכם תשתפר בצורה משמעותית ובאגים יחסכו מזמן הריצה.

תכנות גנרי יכול לפתור בצורה אלגנטית בעיות רבות של שכפול קוד ואני מקווה שאחרי מאמר זה תמצאו את ההזדמנות לשלב אותו בקוד שלכם.

Bootable Back|Track from USB - Persistent Changes

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

פרוייקט Back|Track הוא מערכת הפעלה אשר נועדה לשימוש של חוקרי אבטחה, הפצת לינוקס (בגירסא הרביעית היא מבוססת על Ubuntu) המיועדת ל-Penetration Testers והאקרים. הפצה זאת מגיעה עם כמות נכבדת מאוד של כלים לביצוע שלל דברים הקשורים בנושא, כגון כלים למיפוי/פריצת רשתות מקומיות ואלחוטיות, כלים לפיענוח סיסמאות, כלים לביצוע מספר רב של מתקפות שונות, כלים לביצוע האזנות לחיבורי רשת, כלים לביצוע Reverse Engineering, כלים לסריקות חורי אבטחה במערכות שונות, כלים להרצת אקספלויטים ועוד. במאמר זה אסביר איך להתקין את המערכת על התקן Disk On Key "וקינפוגה" כך שהיא "תזכור" (Persistent Changes) את השינויים שביצענו על כל מחשב. כך בעצם ניתן יהיה לטעון אותה מכל מקום, על כל מחשב, מבלי צורך להתקין אותה כל פעם בחדש. שימוש באפשרות זאת מגביר את גמישותה של המערכת בכך שבעזרתה אפשר להפוך כל עמדת-קצה לסביבת עבודה נוחה שלא תבייש שום האקר.

קצת רקע על הפרוייקט:

פרוייקט Back|Track כמו שאנחנו מכירים אותו כיום הוא שילוב של שני הפרוייקטים הבאים: הפרוייקט הראשון-Whoppix, הפצת Penetration מבוססות על Knoppix שבגרסתה השלישית התבססה על Slax ועקב כל שמה שונה ל-Whax (אגב, ה-W מגיע מהביטוי "White-hat"). והפרוייקט השני- Auditor Security Collection (הפצת LiveCD שמבוססת על Knoppix). את הפרוייקט יצרו מתי אהרוני (Muts) - חוקר אבטחה ישראלי, ומקס מוסר (Max Moser). הגירסא הראשונה של Back|Track יצאה בשנת 2006, ונכון להיום (אוקטובר 2009) הגירסה הרביעית של ההפצה נמצאת בשלבי סיום ("Pre-final/Pre-Released").

הכנות לפני ההתקנה

במאמר אני אשתמש ב-Back|Tack 4 Beta, אך הפעולה זהה גם בגירסאות אחרות של ההפצה. במאמר גם אסביר כיצד אפשר לגרום למערכת לזכור את השינויים שביצענו גם לאחר ניתוק התקן ה-USB, ולכך נאלץ ליצור שתי מחיצות על ההתקן שלנו.

המחיצה הראשונה שניצור היא מסוג FAT32 תהיה בגודל של לפחות 2GB והמחיצה השניה שניצור תשמש לשמירת השינויים שלנו - והיא תהיה מסוג Ext2 (עבור מחיצה זו מספיק אפילו חצי GB).

- רב התקני ה-USB מגיעים עם מערכת הקבצים NTFS, ולכן עלינו לפרמט אותם ל-FAT32 בשלב הראשון. הדבר לא הכי פשוט מכיוון ש-Windows אינה תומכת בפיצול התקני Removable למחיצות. נרחיב על כך בהמשך.
- בכדי ליצור את המחיצה השניה (Ext2) נשתמש בכלי Fdisk שמגיע עם (כמעט?) כל הפצת לינוקס.
- בכדי לפרוס את קובץ ה-ISO של המערכת על התקן ה-USB ולהגדירו כ-"Bootable" השתמשתי בכלי: "UNetbootin".

חלוקת ההתקן למחיצות:

הכניסו את התקן ה-USB לאחת היציאות המתאימות במחשב. כנסו ל-My Computer ותראו שהמחשב שלנו מזהה את ההתקן כ-Removable Media, אם ננסה לגשת ל:

```
Control Panel=>Administrative Tools=>Computer Management=>Storage=>Disk Management
```

שימו לב שההתקן אינו מופיע ככונן שאפשר לחלקו למחיצות. למה Windows לא תומכת בזה? רק השטן יודע. בכל זאת, ישנה דרך מאוד מעניינת לגרום למערכת להתייחס להתקן ה-USB שלנו כ-Local Disk. על מנת לגרום ל-Windows להתייחס להתקן ה-USB כ-Local Disk אנחנו צריכים להוריד את הקובץ הבא, (הוא ישמש לנו כ-"Driver" תואם):

http://www.lancelhoff.com/downloads/USB_LocalDisk.zip

לאחר הורדת הקובץ עלינו להגדיר מספר דברים במנהל ההתקנים (Device Manager) של מערכת ההפעלה. ב-"My Computer" יש ללחוץ כפתור ימני על התקן ה-USB, ולבחור ב-"Properties". שם ניגש ללחוצץ "Hardware", תחת "All Disk Drives" נסמן את התקן ה-USB ונבחר שוב ב-"Properties". ניגש

לחוצץ האחרון-"Details" ואיפה שכתוב "Property" נבחר ב-"Device Instance Path" (למשתמשי XP יהיה כתוב "Device Instance Id") ונעתיק את התוכן שקיים ב-"Value". תוכן זה משתנה מהתקן להתקן. אצלי הוא:

```
USBSTOR\DISK&VEN_SANDISK&PROD_CRUZER&REV_8.01\0876011D7201D615&0
```

[לכל התקן יש מחרוזת "Device Instance" שונה, המהווה את ה"מיקום" של ההתקן לגבי הקרנל.]

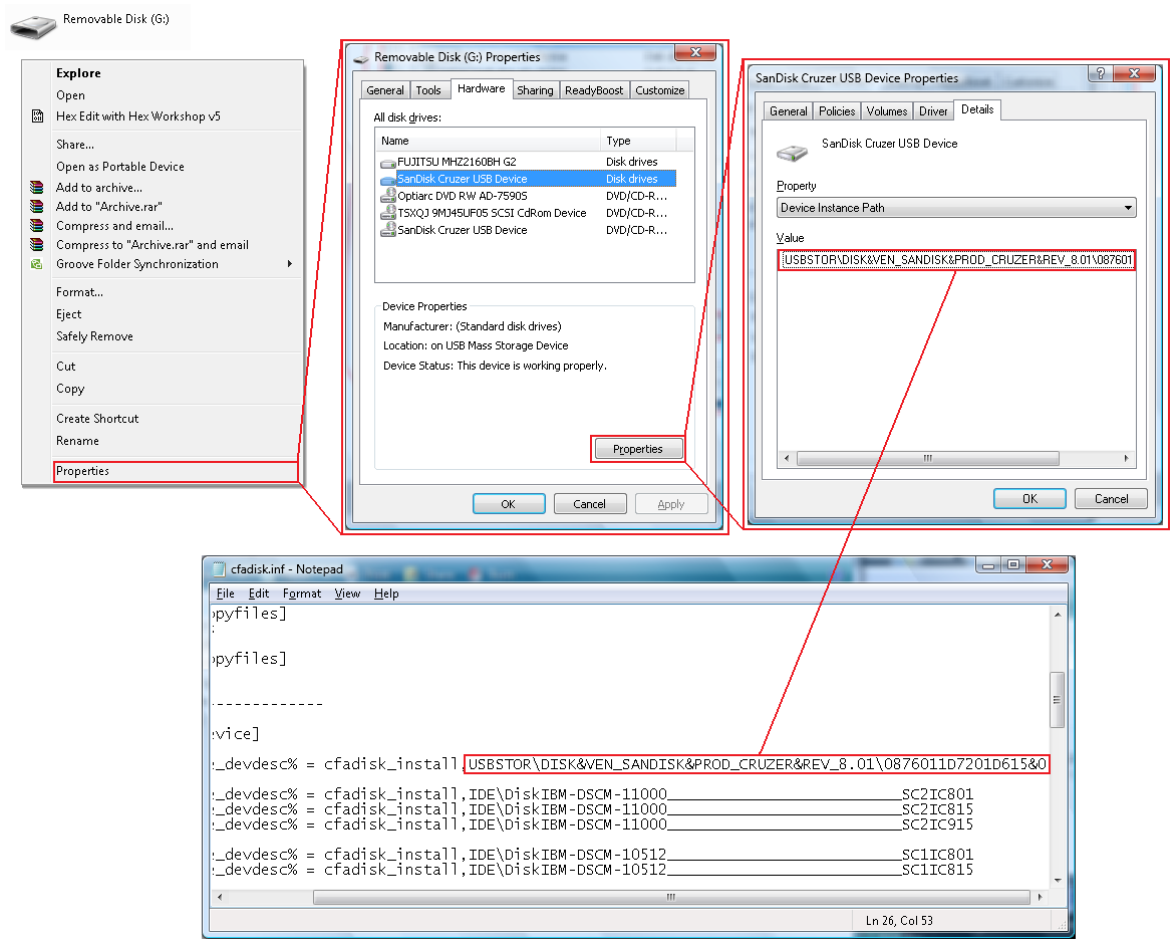
פיתחו את הקובץ שהורדנו בתחילת הפרק (USB_LockDisk) ופירסו אותו לתיקיה על מחשבכם. חפשו בתיקיה את הקובץ שאחראי על הקונפיגורציה של הדרייבר (cfadisk.inf), פיתחו אותו עם כתבן ובשורה 26 אמורה להופיע לכם המחרוזת הבאה:

```
%Microdrive_devdesc% = cfadisk_install,device_instance_id_goes_here
```

את איפה שכתוב "device_instance_id_goes_here" שנו למחרוזת ששמרנו קודם. התוצאה הסופית:

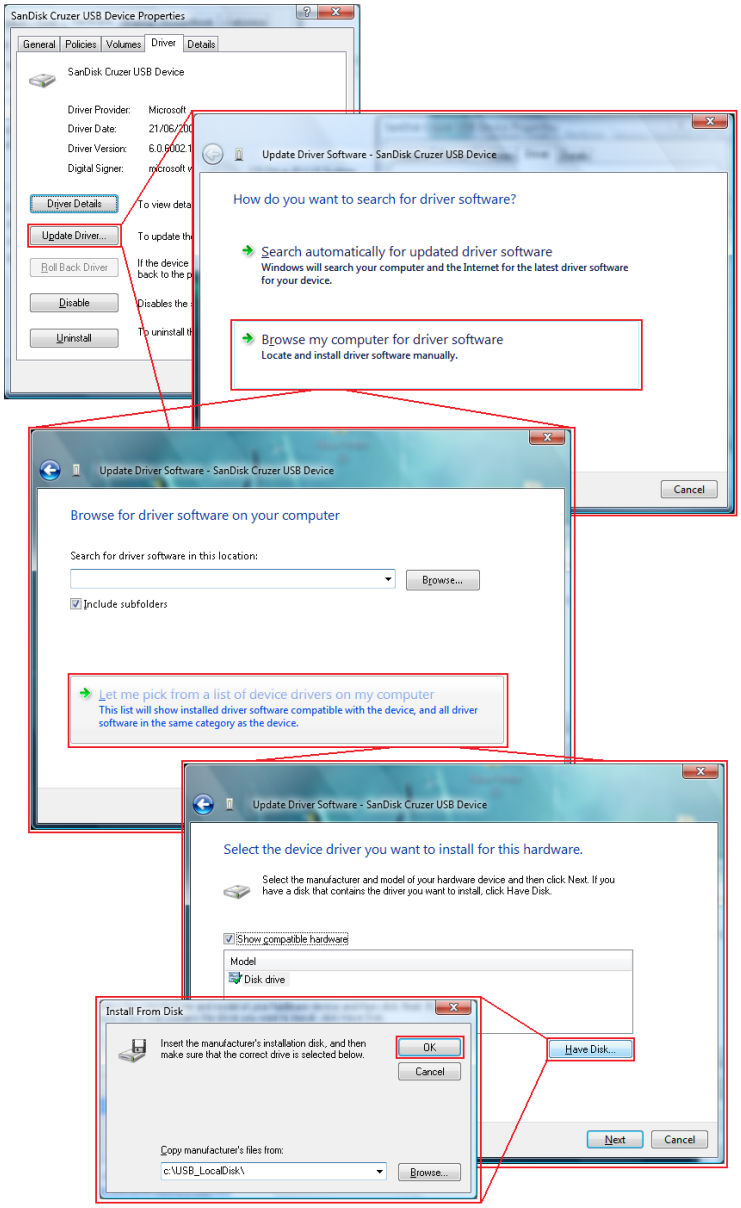
```
%Microdrive_devdesc% =
cfadisk_install,USBSTOR\DISK&VEN_SANDISK&PROD_CRUZER&REV_8.01\0876011D7
201D615&0
```

שימרו את הקובץ וסיגרו אותו. התרשים הבא ממחיש את התהליך:



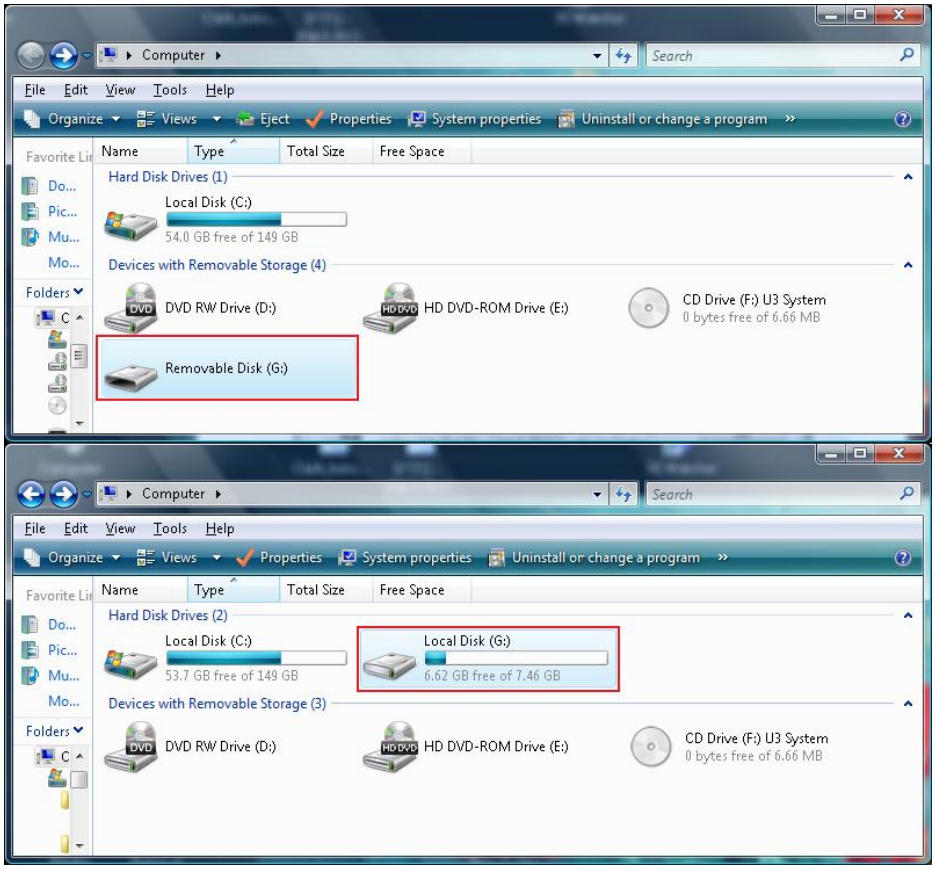
לאחר שהתאמנו את הדרייבר שלנו- אנחנו צריכים להתקין אותו להתקן הספציפי שלנו, בכדי לעשות זאת נשתמש במנהל ההתקנים של המערכת.

גשו שנית לתפריט המאפיינים של התקן ה-USB שלכם (אחרי שבחרתם תחת "All Disk Drives" ולחצתם על "Properties"), ושם במקום לבחור ב-"Hardware" ביחרו ב-"Driver". בתפריט שנפתח לכם, ביחרו ב-"Update Driver..." ושם ביחרו באפשרות השניה "Browse my computer for driver software" (האפשרות הידנית), לאחר מכן ביחרו באפשרות של "Let me pick from a lift of device drivers on my computer" ושם ליחצו על "Have Disk..", יפתח לכם חלון שם "Browse" ושם נווטו לתיקית ה-"USB_LockDisk" שהורדתם וליחצו על אישור.



לאחר שתלחצו על "OK" - מנהל ההתקנים לבד יזהה את קובץ ה-inf וישאל אתכם האם אתם בטוחים שאתם רוצים לבצע את הפעולה. לאחר שתאשרו תקפוץ לכם הודעה אשר תזהיר אתכם כי הדרייבר שאתם מנסים להתקין אינו נחתם באופן תקני והדבר יכול להוות סיכון. אשרו וחכו לסוף התהליך.

בסוף התהליך תאלצו להדליק את המחשב מחדש (המלצה של מערכת ההפעלה- לא שלי) ושימו לב שמעכשיו מערכת ההפעלה תזהה את התקן ה-USB שלכם כ-"Local Disk". (תוכלו לראות את זה אם תכנסו ל-"My Computer"..):

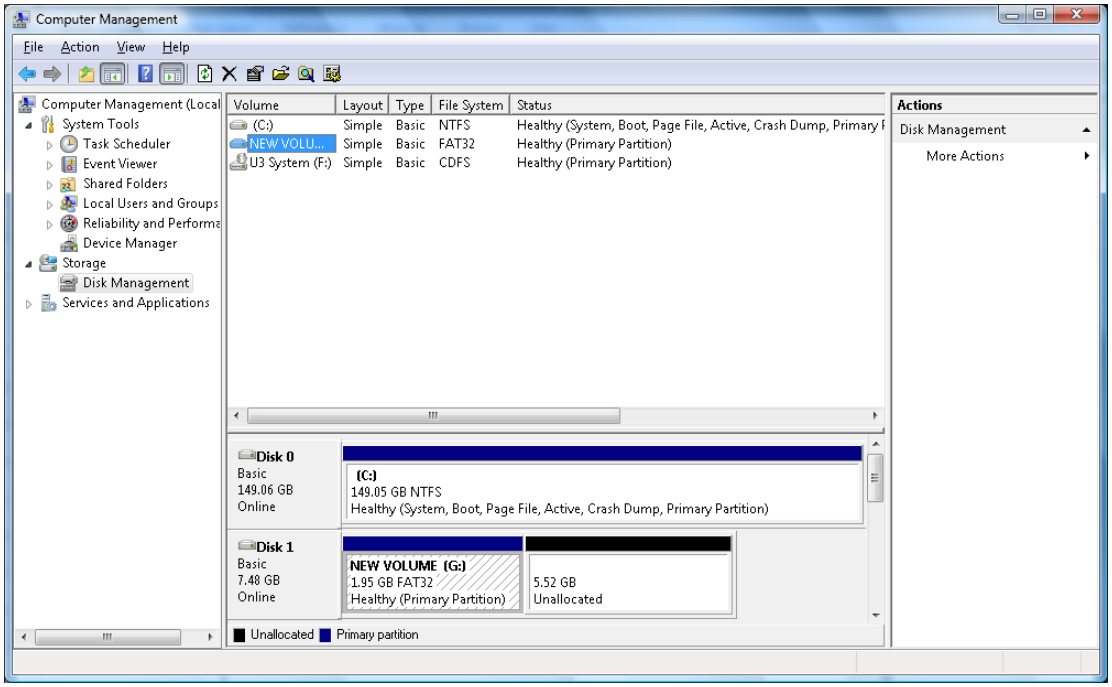


עכשיו, כל שעליכם לעשות הוא להכנס ל:

Control Panel=>Administrative Tools=>Computer Management=>Storage=>Disk Management

לבחור את התקן ה-USB שלכם שמזוהה ככונן מקומי. כפתור ימני ואז-"Delete Volume", לאחר כמה שניות תראו שכל הזיכרון בהתקן נמצא כ-"Unallocate". זה הזמן לפרמט את הכונן כ-FAT32.

לחצו על הכפתור הימני וביחרו ב-"Format" המערכת BackTrack לא זקוקה ליותר מ-2GB, ואין סיבה להשתמש ביותר, פרמטו 2GB מההתקן כ-FAT32 ואת השאר תשאירו כ-"Unallocate".



לאחר התהליך הנ"ל יש לכם ביד התקן USB מפורמט עם מערכת קבצים מסוג- 32FAT בגודל של 2GB.

התקנת המערכת:

מה שנשאר לנו בכדי להתקין את המערכת הוא להוריד את קובץ התמונה שלה (ISO), ואת התוכנה שתפרוס לנו אותו על מחיצת ה-FAT32 שלנו ותהפוך אותו ל-Bootable.

קבצי ISO (ראשי תיבות של: International Standarts Format) הוא קובץ "Image", קובצי Image או קבצי תמונה של דיסק. הם מכילים את התוכניות ואת ה-DATA בדיוק כמו שהוא נראה על הדיסק, אופן שמירתם ותצורת התיקיות. (ישנם עוד סוגים של קבצי תמונה כגון-BIN, ISO, CIF, NRG)

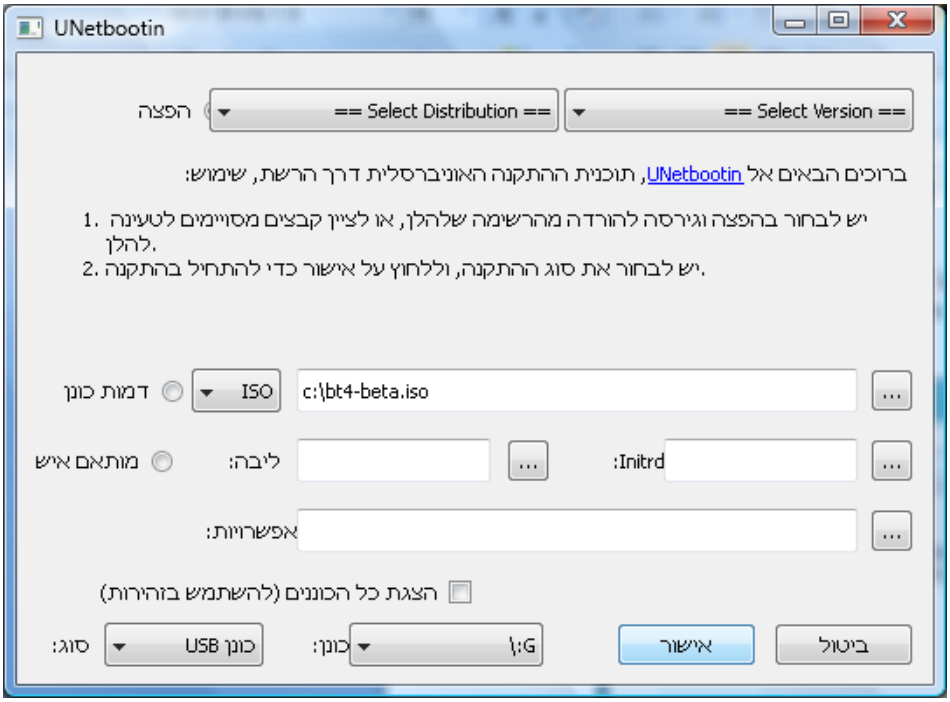
את קובץ ה-ISO המכיל את הגירסה האחרונה של Back|Track תוכלו להוריד מכאן:

http://www.remote-exploit.org/cgi-bin/fileget?version=bt4-prefinal-iso

ואת UNetbootin, תורידו מכאן:

http://unetbootin.sourceforge.net/

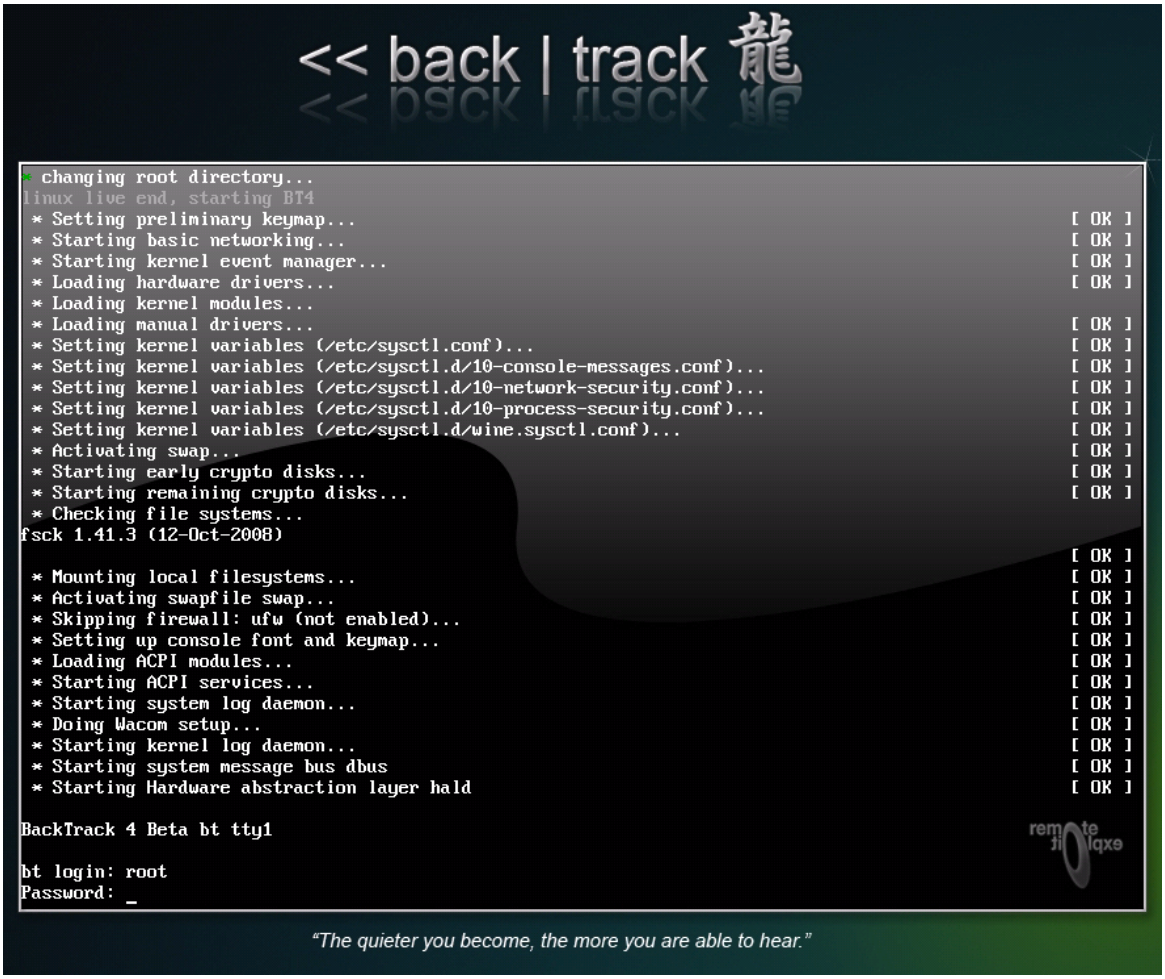
לאחר שהורדתם את שני הקבצים, פיתחו את UNetbootin וביחרו באפשרות של לבצע את הפעולה דרך קובץ ISO (וכמובן ביחרו את קובץ ה-ISO שזה הרגע סיימתם להוריד). ביחרו את התקן ה-USB שפירמטתם קודם לכן ואשרו את הפעולה:



לאחר אישור ביצוע הפעולה, התוכנה תפרוס את קובץ ה-ISO על התקן ה-USB שלכם, תדאג לאתר ולדאוג לכל מה שצריך לעדכן ב-syslinux.cfg.

אם עשיתם הכל נכון, לאחר סיום ביצוע הפעולה הנ"ל, יש לכם ביד התקן USB שעליו מותקנת מערכת Back|Track4 Pre-Released והוא מוגדר כ-Bootable. זאת אומרת שאם תכניסו את ההתקן הנ"ל לאחת מיציאות ה-USB במחשבכם ותדליקו אותו- המחשב יטען את המערכת (אם לא, תכנסו להגדרות הביוס שלכם ותדאגו שהמחשב ינסה לבצע דבר ראשון Boot מהתקן ה-USB ורק לאחר מכן מהרד-דיסק או מהרשת.

כשתפעילו את המערכת יעלו לפניכם ארבע אפשרויות, ביחרו בראשונה, לאחר שהמערכת תטען את כל ה-Live Scripts ותסיים לזהות את כל החומרה שיש לכם במחשב היא תציג בפניכם את ממשק הקונסול, הוא יבקש ממכם שם משתמש וסיסמה, שם המשתמש בברירת המחדל הוא: root, וסיסמתו היא: toor.



אחרי שתכניסו אותם תוכלו להתחיל להשתמש במערכת הפעלה. בכדי לטעון את הממשק הגראפי KDE (הפצת ה-Back|Track מגיעה עם ממשק KDE וממשק FVWM עם Crystal) יש לכתוב את הפקודה:

```
startx
```

כדי לטעון את FVWM, יש לכתוב את הפקודה:

```
bt-crystal
```

שמירת השינויים במערכת (Persistent Changes)

הצלחנו ליצור התקן USB שממנו אנחנו יכולים להריץ על כל מחשב את Back|Track, אבל מה, נסו ליצור קובץ על שולחן העבודה ותגלו שלאחר שתסגרו את המערכת ותפעילו אותה מחדש- הקובץ יעלם. למען האמת, בכל פעם שתריצו את המערכת מחדש- היא תרוץ כאילו היא רצה בפעם הראשונה, ולא משנה כמה שינויים עשיתם בה. הדבר מעצבן מאוד ואף מציק ביותר, לדוגמא- בכל פעם שנפעיל את המערכת מחדש- נאלץ לעדכן את כל הכלים בהם אנו נרצה להשתמש. לא נוכל לשמור קונפיגורציות של כלים או מידע על סריקות קודמות שעשינו.
אז איך בכל זאת נוכל להתגבר על הצרה הזאת? ממשיכים לקרוא...

בתחילת המאמר אמרתי לכם שניצור שתי מחיצות על התקן ה-USB שלנו, עד כה יצרנו רק מחיצה אחת- למערכת ההפעלה, עכשיו ניצור מחיצה שעליה נאכסן את כל המידע ה"דינאמי" שלנו. מדובר במספר פעולות די פשוטות שיקלו עלינו באופן נרחב ביותר.

הכניסו את התקן ה-USB והפעילו את הלינוקס שלכם- לא מדובר בלהפעיל את המערכת שהרגע התקנו מהכונן ה-USB, מדובר במערכת נפרדת- בכדי שנוכל לבצע שינויים בכונן ה-USB שלנו. אני מעלה את ה-Ubuntu שלי, מכניס את התקן ה-USB, ההפצה לבד מזהה את ההתקן ומצמידה לו Mount Point ("Mount Point" זאת "נקודת עגינה" - מעין הלינק לאותו התקן כדי שהמערכת תדע לזהות אותו), אני ניגש לתיקיה Media (אצלכם יש מצב שתמצאו אותו ב-mnt זה תלוי בהפצה שלכם), רואה את הנקודת עגינה- אצלי היא "/Media/NEW VOLUME", איך אני יכול לזהות אותה? פשוט מאוד- אנחנו יודעים איזה קבצים אמורים להיות על ההתקן- ואם הם נמצאים שם- אנחנו יכולים להיות בטוחים שזאת הנקודת עגינה שלנו. אחרי שזיהנו את נקודת העגינה אנחנו צריכים לנתק אותה- לעשות לה "Un-Mount" (למה? בכדי שנוכל לכתוב עליה מחיצה נוספת- לא נוכל לעשות את זה בזמן שהיא במצב "Mount"), פשוט מאוד- כפתור שמאלי ולבחור ב-"Unmount Volume".
אפשר גם לעשות את זה דרך הקונסול למי שרוצה, בעזרת הפקודה הבאה:

```
sudo umount -f [Mount-Point]
```

אצלי נקודת העגינה היא למשל /Media/NEW VOLUME אז אני אכתוב את הפקודה כך:

```
sudo umount -f /media/NEW\ VOLUME
```

כמובן שהמערכת תבקש את סיסמת האדמין.

לאחר שביטלנו את העגינה של ההתקן (אך לא ניתקנו אותו פיזית מהמחשב), כיתבו בקונסול את הפקודה הבאה:

```
fdisk /dev/sdb
```

כדי להפעיל את תוכנת ה-Fdisk ליצירת המחיצה החדשה, הקישו "p" ותקבלו את רשימת המחיצות שקיימות לכם על ההתקן- נכון לעכשיו אתם אמורים לראות שורה בסיגנון הבא:

```
Device        Boot     Start     End    Blocks      Id    System
```

ומתחתיה אמור להופיע לכם רק שורה אחת (יש לנו רק מחיצה אחת על הכונן):

```
/dev/sdb1     *        4         7155   2048000     c     W95  FAT32
```

לאחר מכן, לחצו על "n" בכדי ליצור מחיצה חדשה, התוכנה תשאל אתכם איך להתייחס למחיצה, תקישו "p" כדי לבחור "Primary Partition", ולבסוף תקישו "2" כדי לקבוע אותה כמחיצה השניה.

עכשיו התוכנה תבקש ממנו לקבוע את **גודלה** של המחיצה- אנחנו רוצים ששאר הכונן יהיה המחיצה שלנו, ולכן נלחץ אנטר פעמיים. (פעם ראשונה בכדי לקבוע מיקום ראשון פנוי לנקודת ההתחלה של המחיצה ופעם שניה בכדי לקבוע את המיקום האחרון האפשרי לנקודת סוף המחיצה).

אחרי שקבענו את הגודל- אנחנו רוצים לקבוע את **סוג** המחיצה, לכן נלחץ על p, ואז נלחץ על המספר 2 (בכדי לקבוע את סוגה של המחיצה שכרגע קבענו את גודלה), לאחר מכן נתבקש לקבוע את סוג המחיצה, אנו קובעים זאת ע"י סימון סוג המחיצה כמספר הקסדצימאלי, בכדי לראות את הרשימה של המחיצות והמספרים שמסתמנים אותן נלחץ על L. אנו רוצים לקבוע מחיצת לינוקס רגילה- אצלי המספר הוא "83 - Linux".

אם הכל עד עכשיו טוב, נקבל את השורה הבאה:

```
Changed system type of partition 2 to 83 (Linux)
```

נלחץ שוב p בכדי להגיע לתפריט שמראה לנו את מצב המחיצות על ההתקן- שימו לב שנוספה לנו מחיצה בשם sdb2 (במקרה שלי), אם הכל תואם למה שרצינו- נלחץ על w בכדי לכתוב את ההגדרות על ההתקן.

לאחר מכן מה שנותר לנו הוא לפרמט את המחיצה לסוג ext2, נבצע זאת ע"י שימוש בכלי mkfs שמאפשר לנו ליצור מחיצות ext2 ו-ext3, נשתמש בו כך:

```
mkfs.ext2 /dev/sdb2
```

אם הכל עבר חלק- סיימנו לקבוע את "התשתית" של המערכת שלנו, יש לנו כונן עם מערכת הפעלה שאפשר לבצע ממנו Boot ועוד כונן ext2 שבו אנו נשמור את השינויים שאנחנו נבצע, מה שנותר לנו לבצע הוא לשנות את הקונפיגורציה הנכונה במערכת בכדי שתדע איפה לשמור מה וסיימנו!

תפעילו מחדש את המחשב (Restart) וכנסו שוב למערכת עם הסיסמה והכל. כנסו לתיקיה:

```
/mnt
```

ושם תראו את שתי המחיצות שיצרנו, על אחת מהן יושבת מערכת ההפעלה שלנו, ועל השניה ישבו השינויים שנבצע, כנסו למחיצה sdb2 וצרו בה את התיקיה שתכיל את השינויים, קראו לה:

```
changes
```

לאחר מכן כנסו למחיצה בה יושבת מערכת ההפעלה שלנו, כנסו לתיקית BOOT, ושם כנסו לתיקיה syslinux, בתוכו כנסו לקובץ "syslinux.cfg" , זהו הקובץ שממנו המערכת טוענת את אפשרויות והגדרות מסך ה-Boot של מערכת ההפעלה שלנו, חפשו את ה-Label של האפשרות שבה אתם משתמשים (אני משתמש לרב ב-Console) ותחתיה, חפשו את השורה של ה-APPEND.

אצלי השורה נראת כך:

```
APPEND vga=0x317 initrd=/boot/initrd.gz ramdisk_size=6666
```

עלינו להוסיף את המתג "changes" ולקבוע אותו למחיצה שיצרנו באופן הבא:

```
changes=/dev/sdb2
```

השורה המקורית תראה עכשיו כך:

```
APPEND vga=0x317 changes=/dev/sdb2 initrd=/boot/initrd.gz
ramdisk_size=6666
```

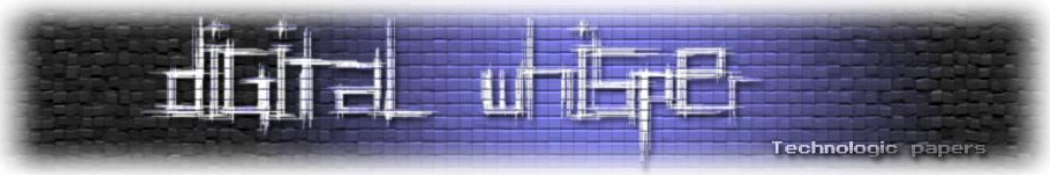
שימרו את הקובץ.

(ישנה עוד אפשרות שבמקום לשנות את השורה המקורית- תוכלו להעתיק את כל הבלוק וליצור אפשרות כניסה חדשה למערכת ואליה להוסיף את האפשרות שהמערכת תשמור את השינויים וכך לאפשר לעצמכם בעתיד לקבוע האם תירצו לשמור שינויים או לא- רק לא לשכוח לשנות את ה-Title.)

סיכום

מאוד מומלץ לעבור על סט הכלים הכלול במערכת וללמוד עליהם. המערכת מציעה מגוון רחב יחסית של כלים לביצוע מספר רב של פעולות. למי שלא התעסק עם לינוקס בחייו השימוש במערכת לא יהיה חלק ולכן מומלץ להכיר את הסביבה לפני-כן.

בנוסף, נכון לכתיבת שורות אלו עוד לא יצאה גירסת ה-Stable של הגרסא הרביעית, אבל שווה לחכות ולהתעדכן בפורומים של Remote-Exploit.



דברי סיום

בזאת אנחנו סוגרים את הגליון השלישי של Digital Whisper. אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות הוקדשו כדי להביא לכם את הגליון.

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 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 הראשון של 2010. כאילו, באמא שלכם, סילבסטר - הפעם נתעכב ביום! ☺

אפיק קסטיאל,

ניר אדר,

1/12/2009