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.

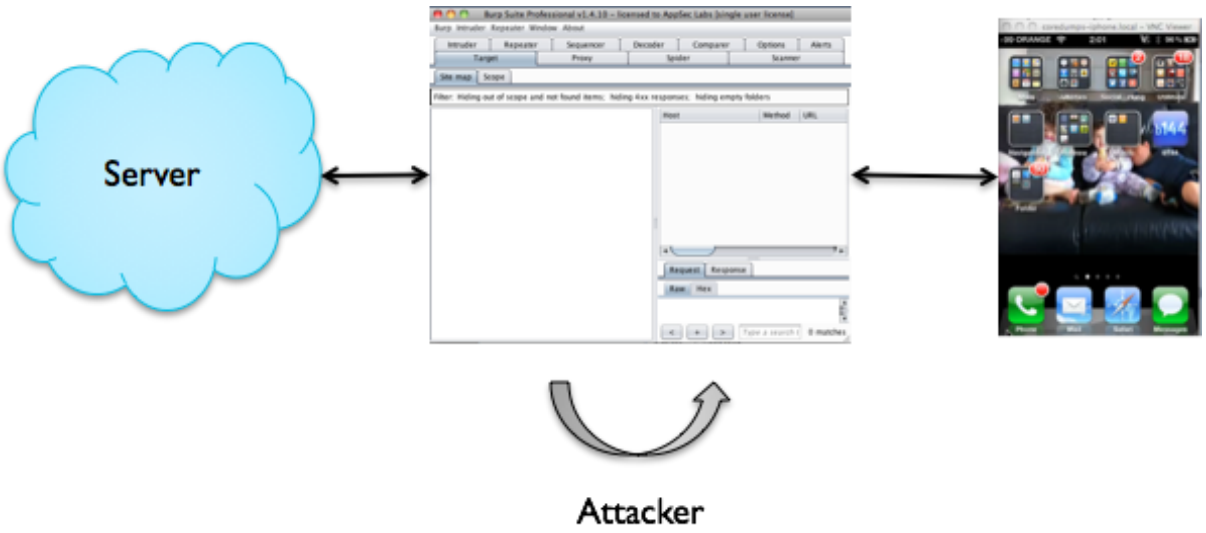
לא תענה - לשיחות טלפון במהלך הבדיקות, זה הורס את הבדיקה ומכניס אותה ל-Delay, עבור למצב טיסה או הוצא את כרטיס הסים.

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 בינאריים 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!

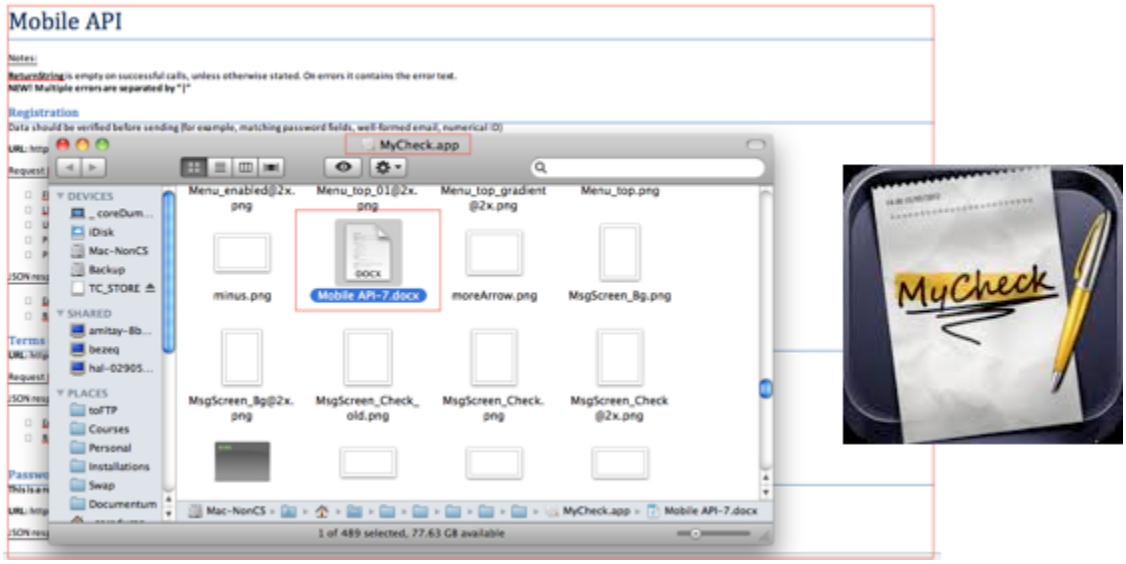
"תחתום לי פה" - מה קורה אם אנחנו רוצים לבצע בדיקות על אפליקציה, כאשר הקליינט שולח בקשות חתומות, או שהוא שולח בקשות דרך 3G, ואין לנו יכולת טובה ליירט אותן ולשחק איתן. יש לנו סיכוי גבוה לפספס כיסוי ובעיות, מאחר וכל הבקשות שלנו ייפלו על ולידציות או על התאמות של צד שרת.

← 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!

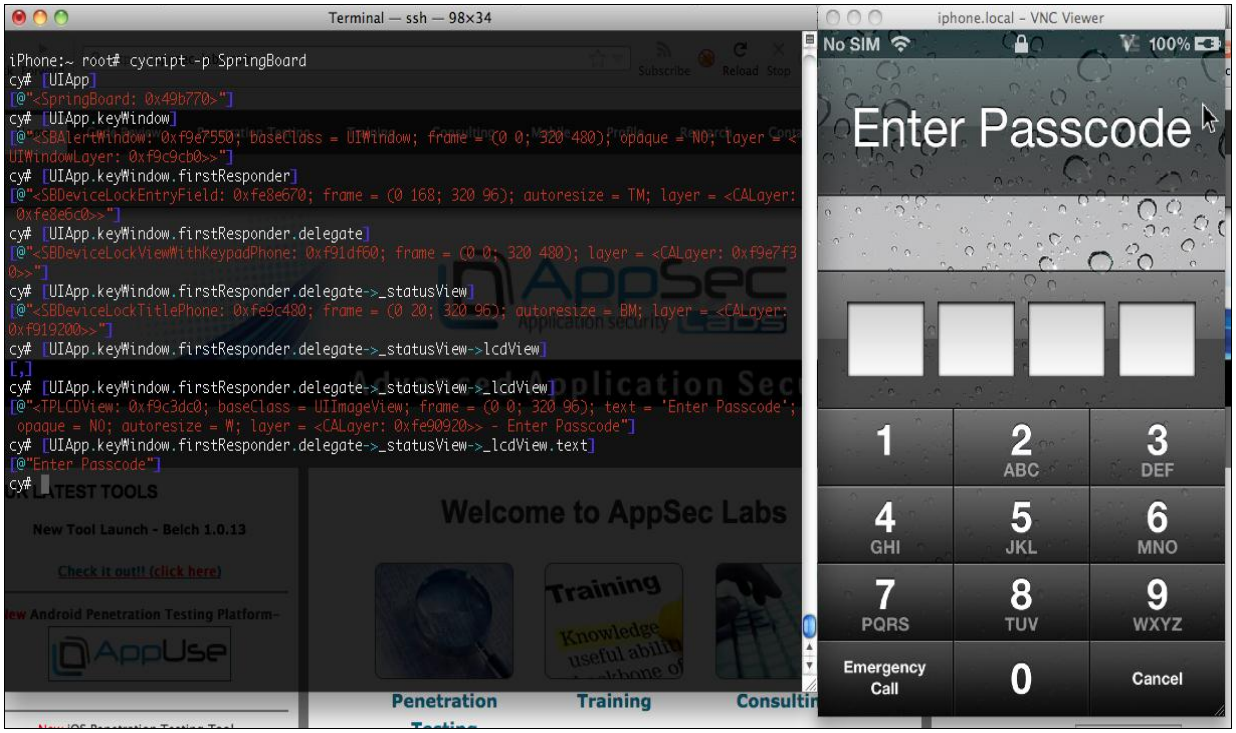
הבעיה בכל השאלות הללו היא שמתוך הפרוקסי אנחנו לא יכולים לענות עליהן, אנחנו לא יודעים אם כל הבקשות הפוטנציאליות עברו דרכינו או אם כל המסכים הרלוונטיים נטענו על המכשיר. במקרים רבים התווך מוצפן כך שגם בדיקות רגילות של טימפור הם עסק בעייתי ואנחנו נקבל המון false-positive.

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.

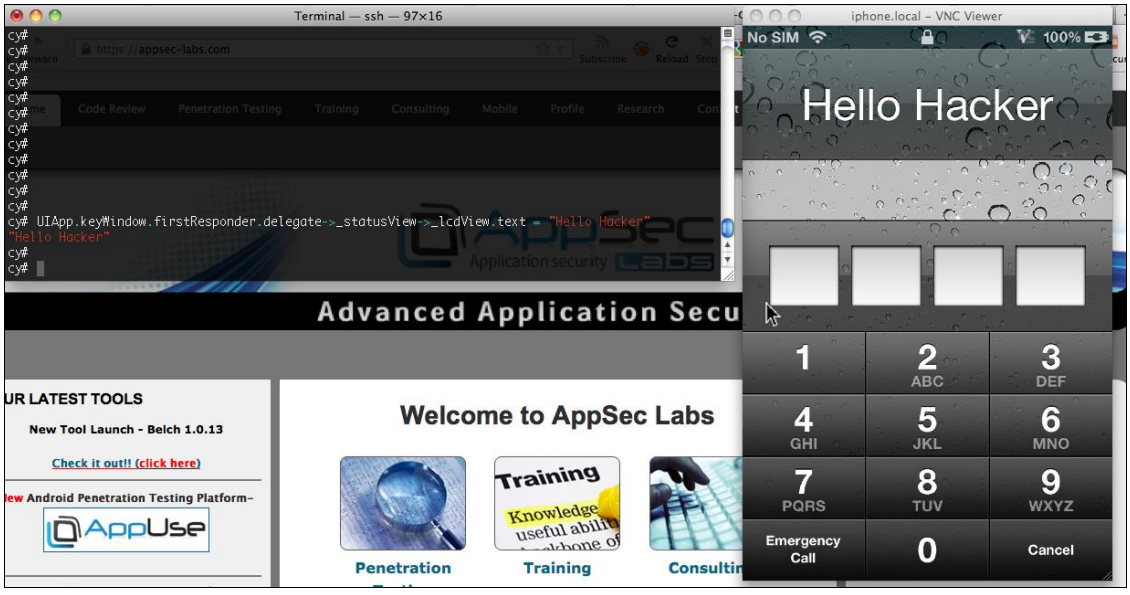
Cycript הינו אינטרפטר אשר משלב בתוכו סינטקסט של ObjC כמו גם JavaScript ומאפשר לנו להתחבר לתהליך קיים ולשחק איתו ישירות. בואו נראה דוגמא בה נתחבר להתליך של SpringBoard ונתחיל לשחק:



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?

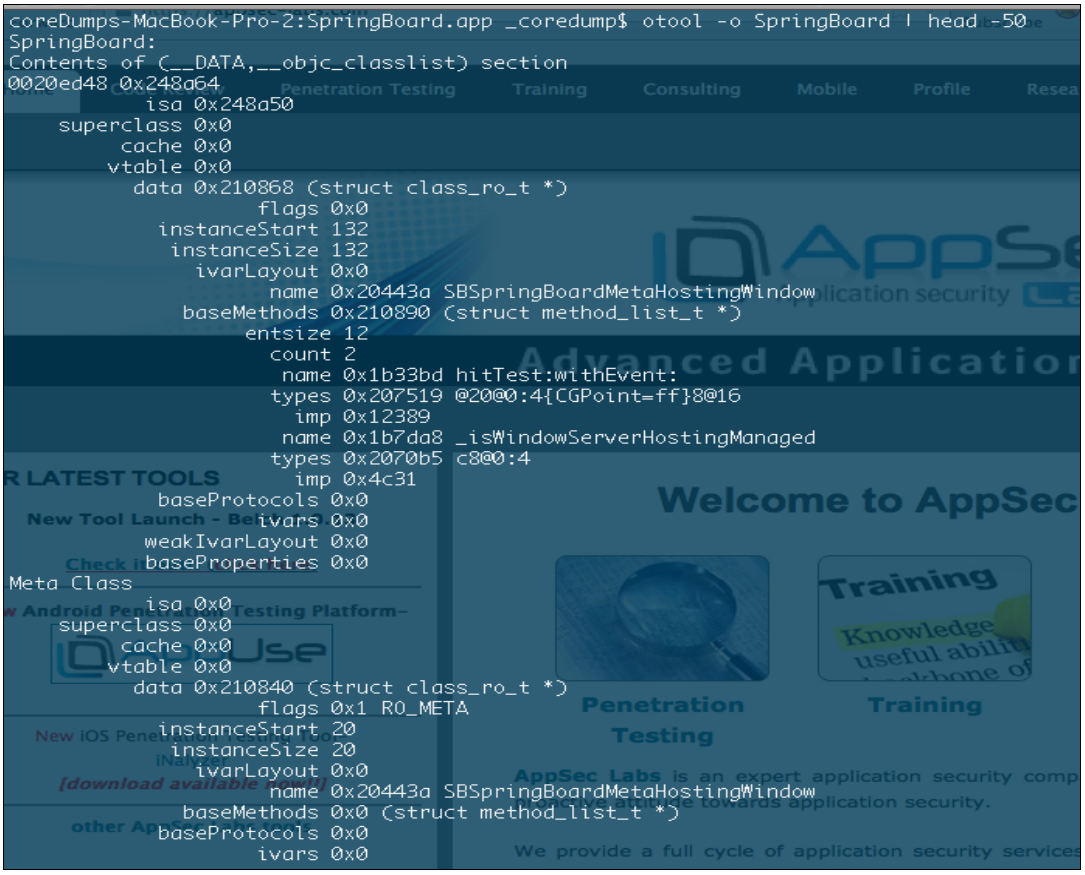
כן, אנחנו הולכים לנצל את המבנה הייחודי של קובץ ה"מאקו" (mach-o) לטובת חליבת אינפורמציה על המערכת שלנו.

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:

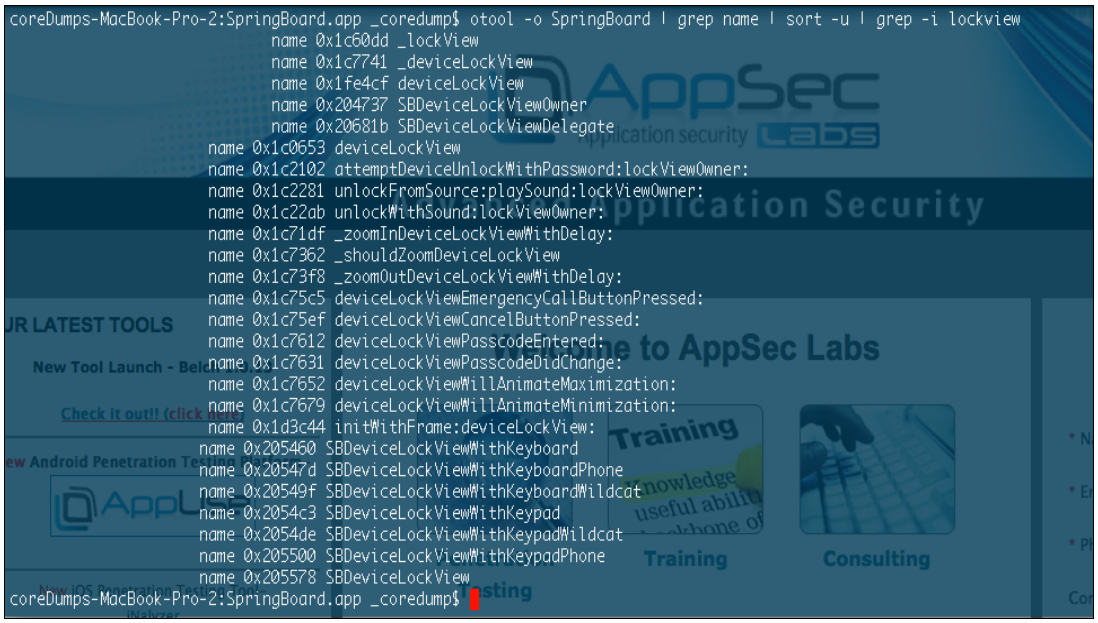
```
coreDumps-MacBook-Pro-2:SpringBoard.app _coredump$ otool
Usage: otool [-fahlLDtdorSTMRIHvVcXm] <object file> ...
        -f print the fat headers
        -a print the archive header
        -h print the mach header
        -l print the load commands
        -L print shared libraries used
        -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'.

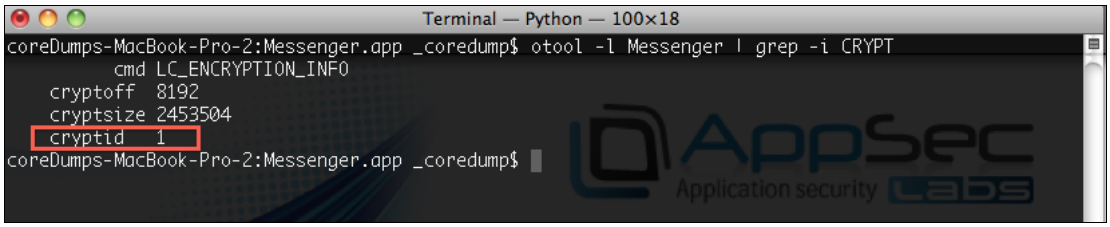
אם כך אנחנו יכולים להשתמש ב-Cycript בכדי לשחק עם האפליקציה מתוך הכרות יותר מעמיקה של המתודות והאובייקטים שלה. זאת אומרת שאנחנו נתעסק עם האפליקציה במקום עם התקשורת - ולסמוך עליה שהתעבורה תצא באופן שאנחנו רוצים.

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.

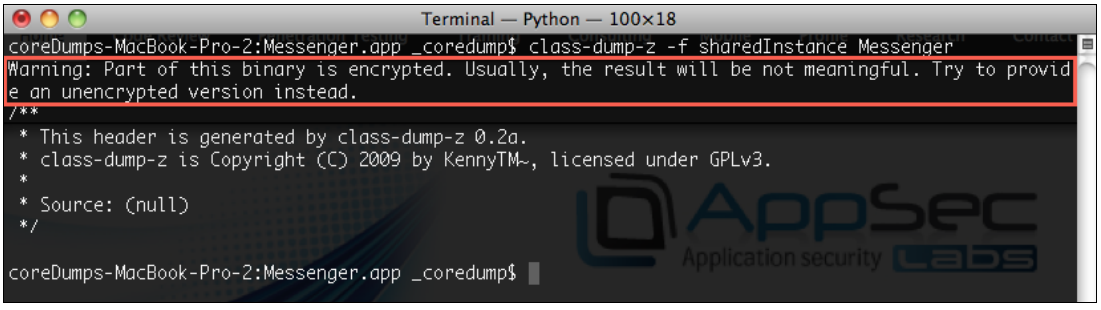
עצור! סיסמא!

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,

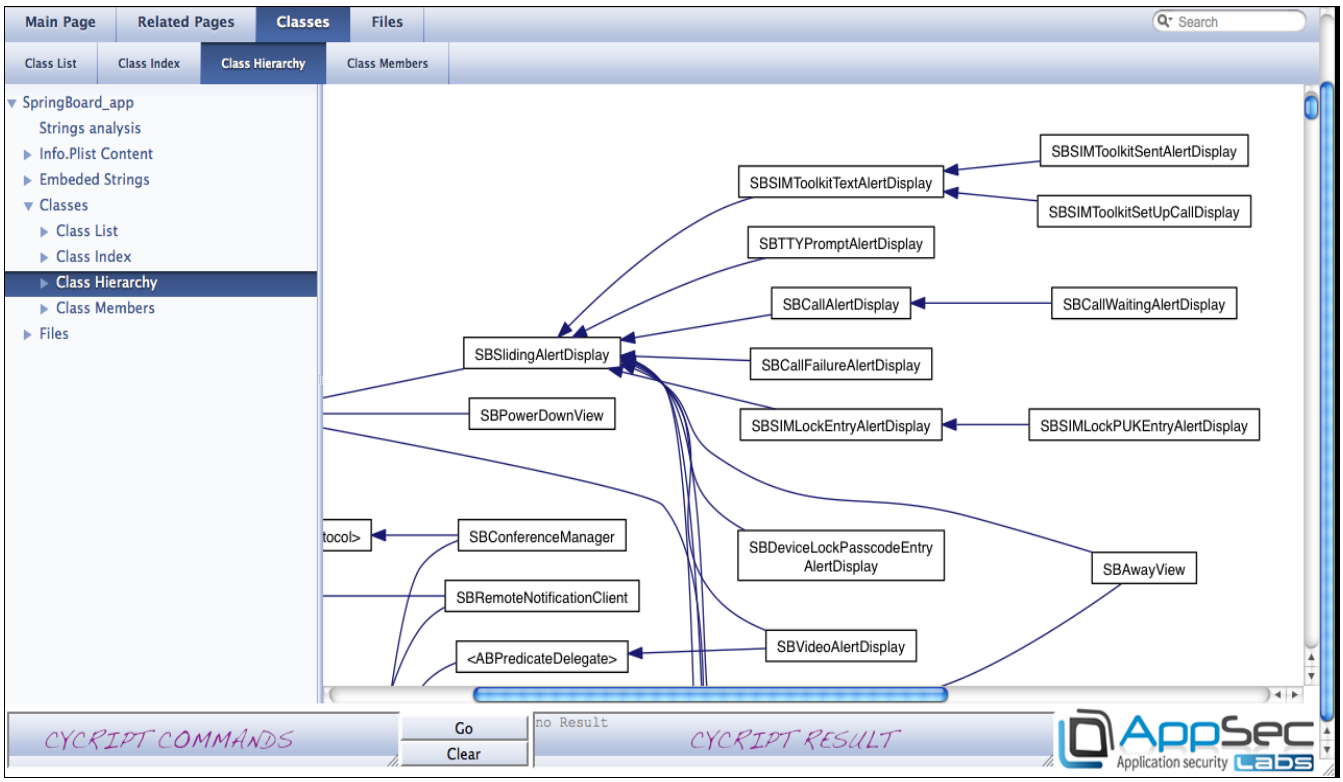
המערכת טוענת את המפתחות ומבצעת פענוח של הסגמנט המוצפן בתוך קובץ המאקו (mach-o) שהורד מה-AppStore. לאחר שהפענוח הושלם האפליקציה מתחילה לרוץ.

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!

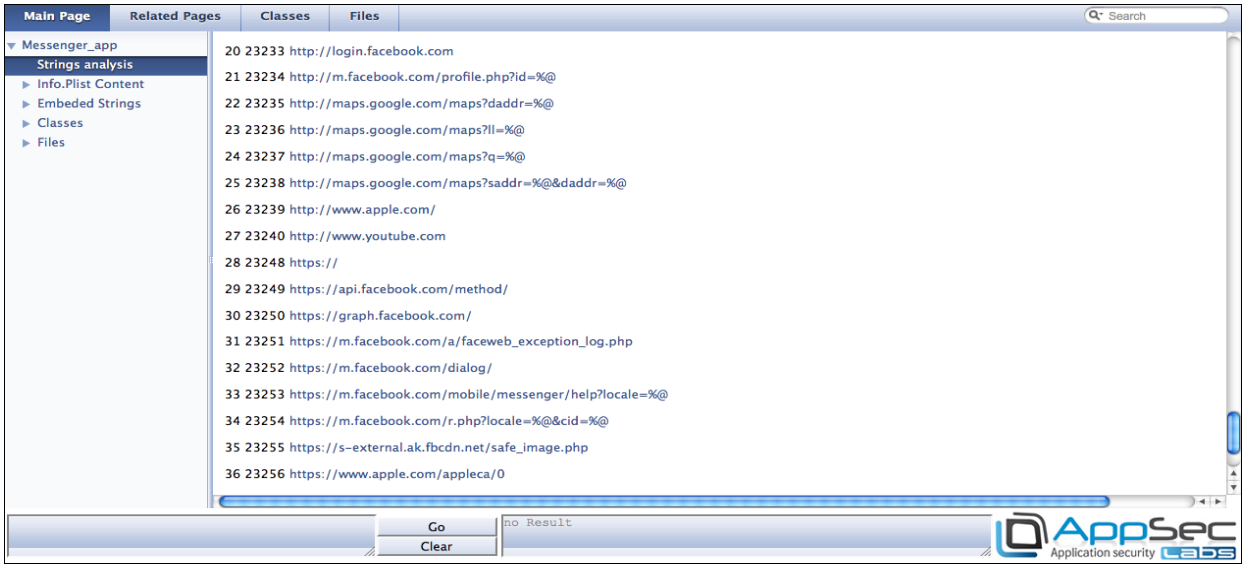
ה-iNalyzer מהווה סביבה מיוחדת לבדיקות של מערכות מבוססות iOS. הוא אוסף את כל הנתונים מתוך קובץ המערכת ומתוך class-dump-z ואז מחולל ממשק Command&Control (מבוסס דוקסיג'ן) עבור Cycript, כך שקיבלנו ממשק של סביבת בדיקות מלאה.

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:

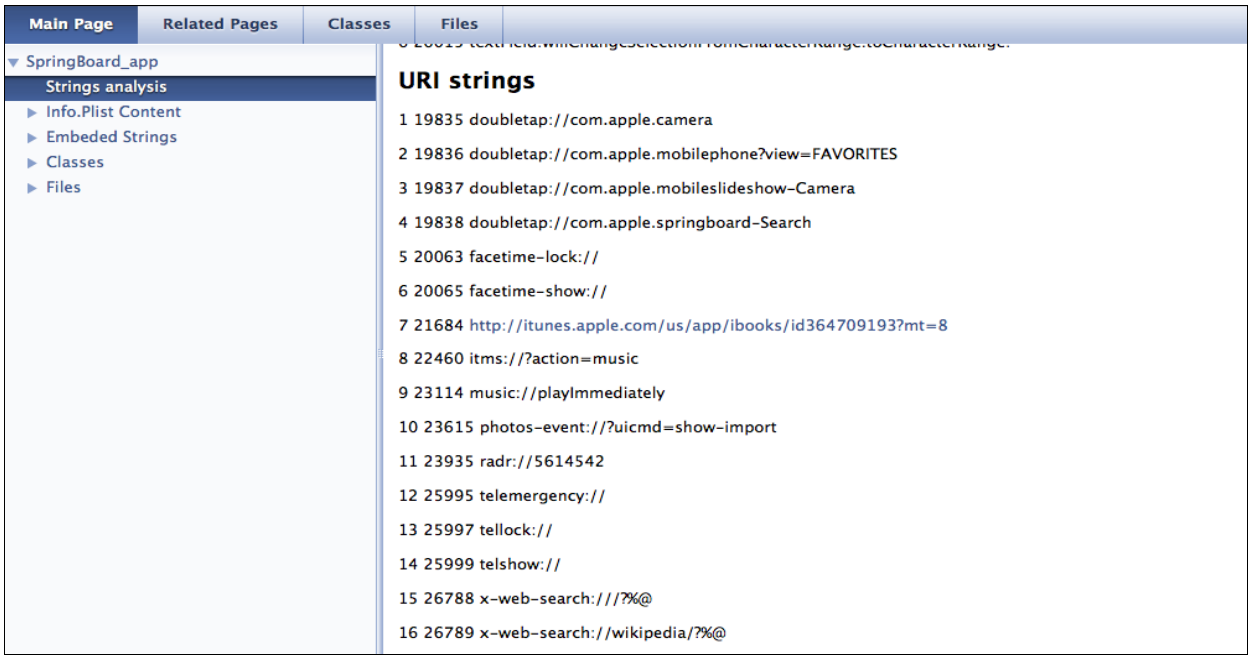


הנה מספר דוגמאות למידע שה-iNalyzer מספק:

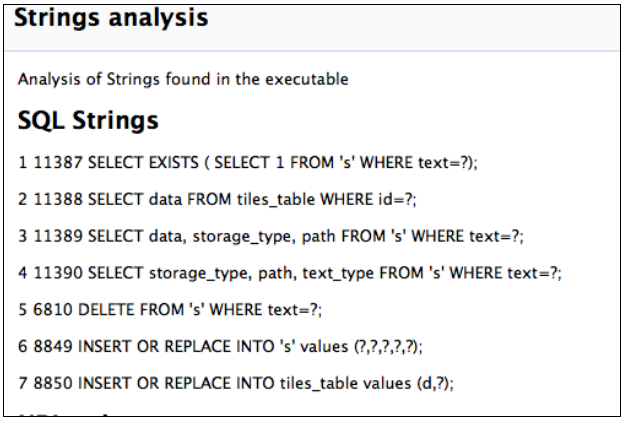
הצגת קישורים חיצוניים, לטובת מיפוי נקודות התממשקות מול שרתים חיצוניים:



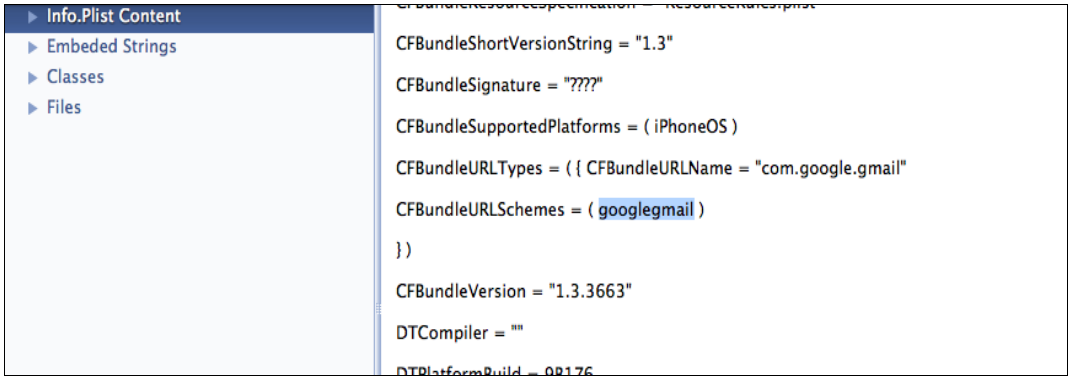
הצגת ממשקי URI שבשימוש, לטובת מיפוי הפעלות חיצוניות של אפליקציות אחרות והזרקות:



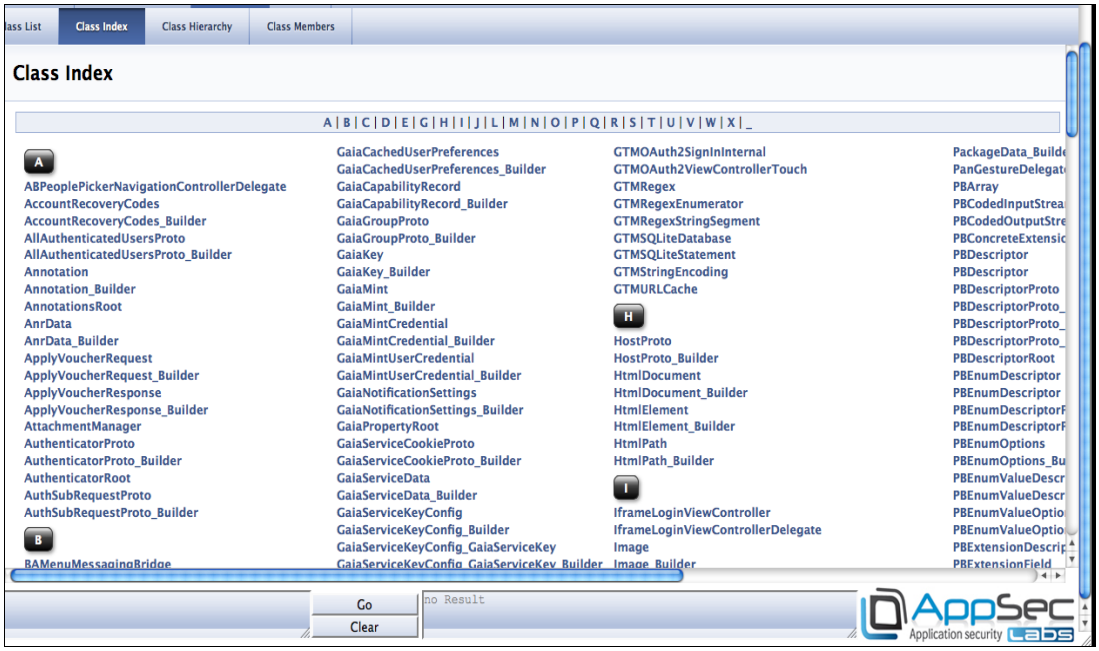
הצגת משפטי sql שבשימוש, לטובת ניתוח פגיעויות של הזרקות מקומיות ומרוחקות:



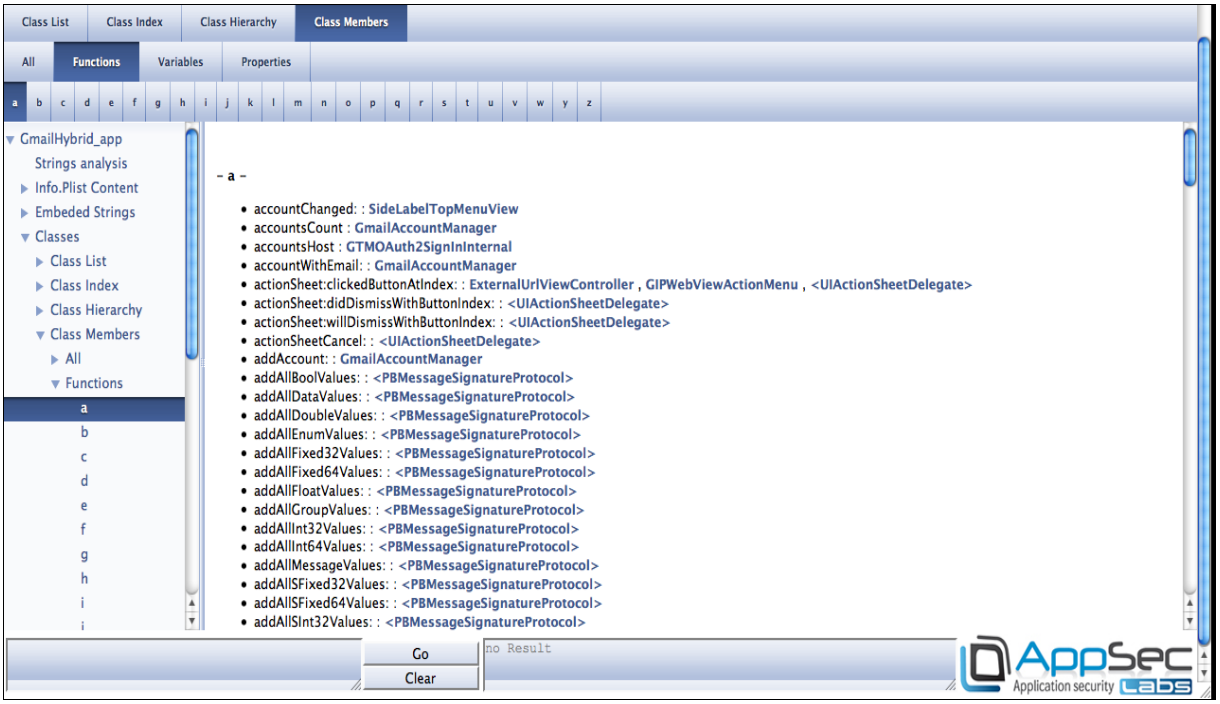
ממשקי CFURL אשר נרשמים לטובת המערכת ומהווים נקודות הפעלה נוספות, ופתח נוח למתקפות:



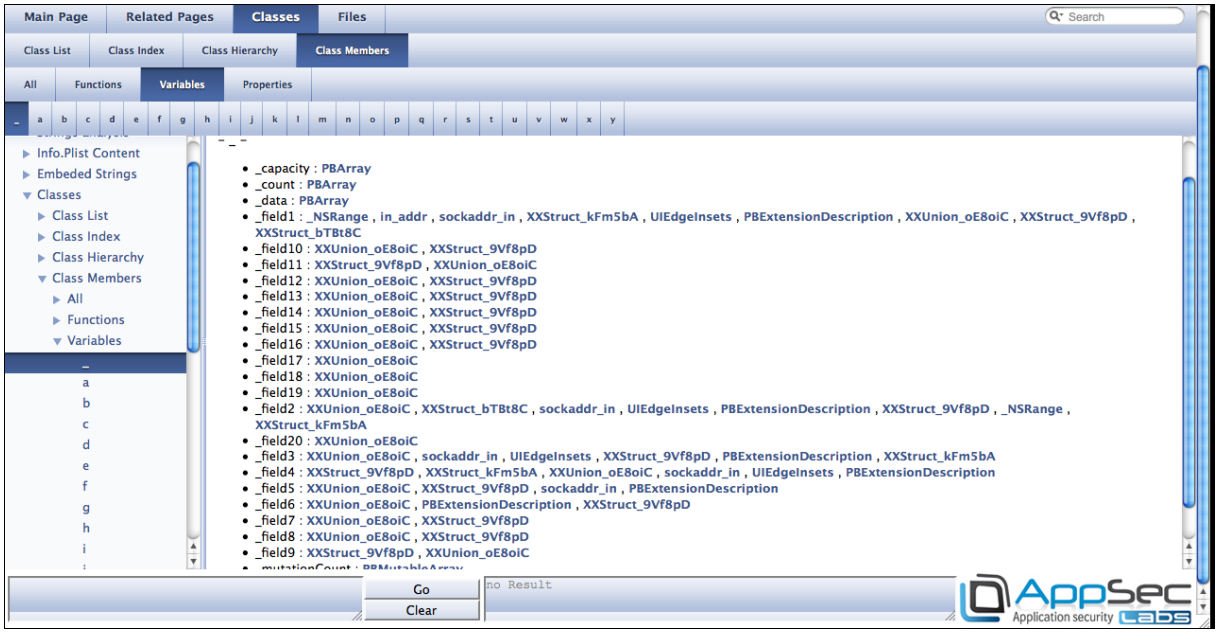
הצגת כל האובייקטים של המערכת, לטובת חשיפת פונקציונאלית בעייתית או עודפת:



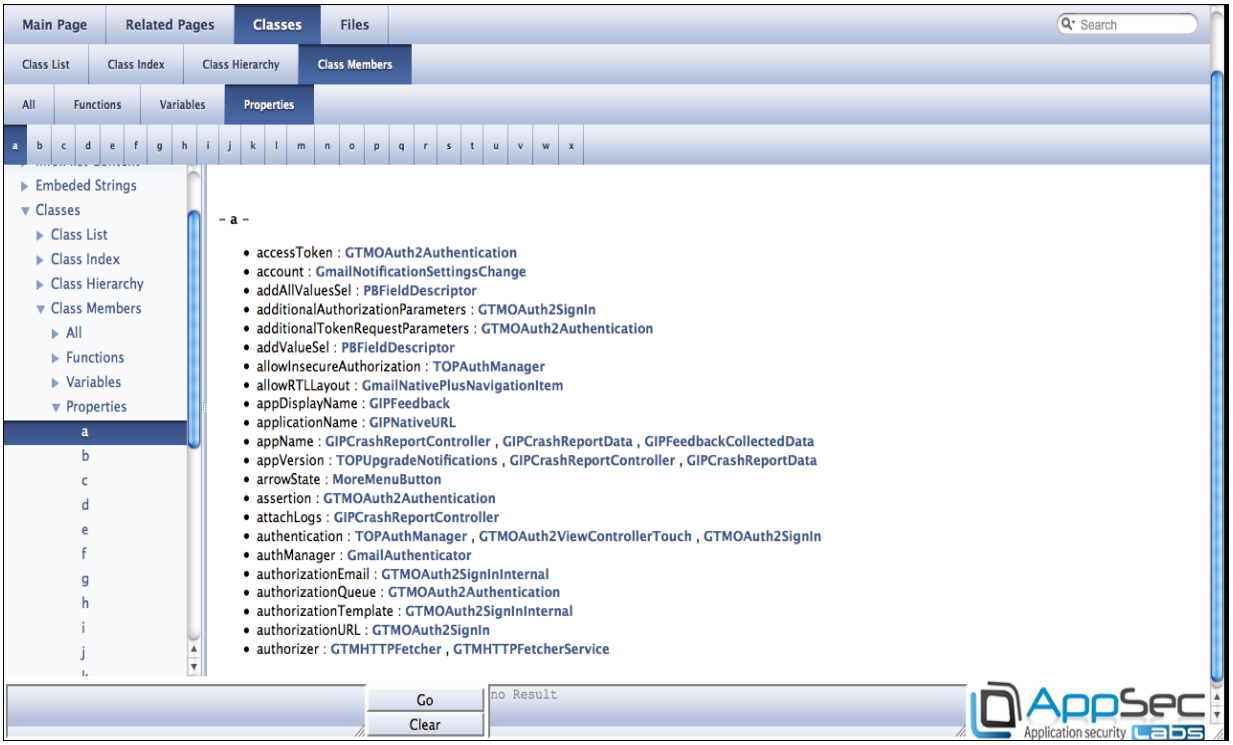
הצגת כל המתודות של המערכת, לטובת הפעלה ישירה שלהם על ידי Cycript:



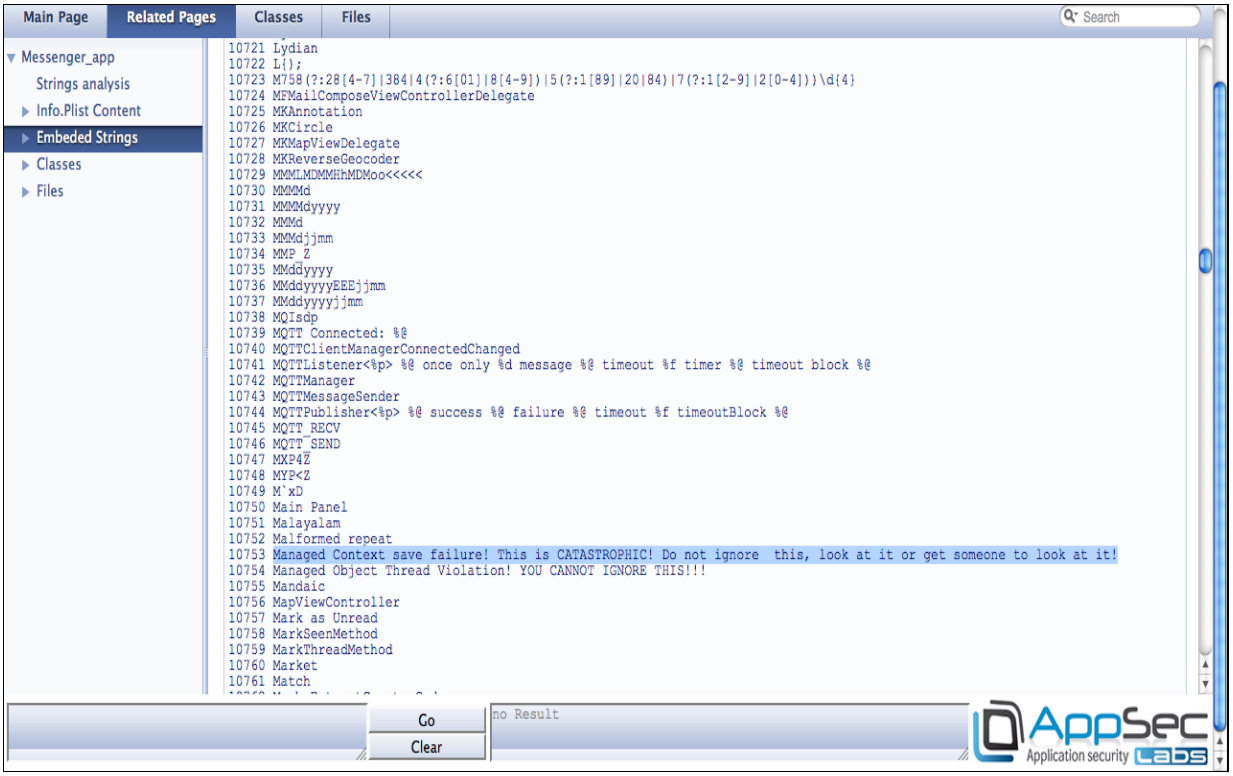
הצגת כל המשתנים של המערכת, לטובת התקפות טימפור (Tampering):



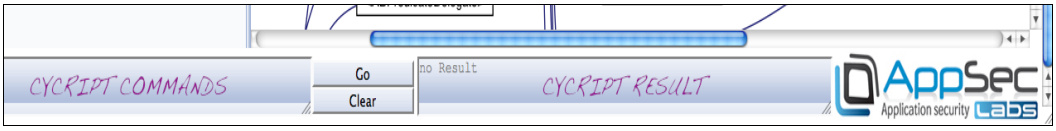
הצגת כל המאפיינים של המערכת, לטובת התקפות Injecting / Tampering :



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