Digital Whisper
Technologic Papers

זיהוי באפלה

ניצול חולשת XSS בשרתי C&C לטובת זיהוי נוזקות

מאת שי נחום

הקדמה

מאמר זה מבוסס על מחקר שהתבצע בטכניון. המחקר בוצע ע"י שי נחום, אסף שוסטר ועופר עציון ופורסם ביוני 2018 בכנס CSCML 2018. קישור למאמר המקורי ניתן למצוא בסוף המאמר. במאמר זה נסביר על פיתוח שיטת הזיהוי, בניית המתודולוגיה, ביצוע הניסויים והרצת הכלי בפועל על פוגענים. הכותב מניח כי הקורא מכיר את הנושאים הבאים הדרושים לקריאת המאמר:

- XSS Injection
- Inline Hooking
- DLL Injection
- COM Objects
- Application Verifier

למעוניינים - נושאים אלו הוסברו בפירוט במאמרים אחרים במגזין זה.

תקציר

קיימות אינספור טכניקות הגנה לשם מניעה וזיהוי פוגענים בתחנות קצה ובשרתים. למרות שטכניקות אלה מוטמעות באופן נרחב ברשתות ארגונים, סוגים רבים של פוגענים מצליחים להישאר מתחת לראדר ולבצע פעולות זדוניות שוב ושוב. לפיכך, נחוץ פתרון נוסף יצירתי ויעיל יותר, במיוחד מאחר שטכניקות הזיהוי הקלאסיות לא משתמשות בכל השלבים הקיימים בשרשרת התקיפה בנסיונן לזהות התנהגות זדונית בעמדות הקצה.

במאמר זה, אנו מציעים גישה חדשה לזיהוי פוגענים. גישתנו משתמשת בטכניקות התקפה והגנה לשם זיהוי תקיפות פעילות. זאת, על-ידי ניצול פגיעות פאנלי השליטה של הפוגענים, ושימוש במניפולציה על ערכים משמעותיים במערכת ההפעלה של עמדות הקצה - כדי לתקוף פאנלים אלה ולהשתמש בתקשורת המהימנה בין המכונה המודבקת לבין פאנל השליטה של הפוגען. הכלי שפיתחנו מבצע הזרקה של תגית תמונה בפונקציות API שפוגענים משתמשים בהם ומנצל חולשה בפאנל השליטה של פוגענים כדי לזהות את הפוגען.

מבוא

במהלך העשור האחרון, אינספור התקפות סייבר עלו לכותרות, כל התקפה חמורה מהקודמות לה: גדולה יותר בהיקפה, מתוחכמת יותר ומתקדמת יותר בהיקף גניבת הנתונים. מתקפות אלה היו אפשריות הודות לתוכניות זדוניות (לדוגמה: Bot, Trojan, ו-POS), שכוללות טכניקות תקיפה חשאיות, חדשות ומתקדמות אשר מונעות מלזהותן. למרות שבמשך השנים פותחו מגוון טכניקות זיהוי לשם מאבק בפוגענים כאלה, התוקפים ממשיכים למצוא שיטות חדשות ויצירתיות לעקיפת טכניקות אלה. במילים אחרות, טכניקות הגנה קלאסיות למניעה וזיהוי, כגון חומת אש, תכניות מבוססות חתימות וחוקים, אנטי-וירוס ומערכות זיהוי חדירה (IDS-Intrusion Detection Systems), אינן מספקות במאבק כנגד מתקפות סייבר. אמנם ייתכן שהן מסוגלות להגן כנגד מתקפות ידועות, אבל מערכות הגנה קלאסיות אינן מועילות בעת הגנה כנגד מתקפות חדשות. לפיכך יש צורך אמיתי לשיטה יצירתית וסתגלנית יותר, שתספק שכבת הגנה נוספת נגד פוגענים.

מאמר זה מציע טכניקה חדשה לזיהוי אוטומטי של הדבקות פוגענים בעמדות קצה, תוך שימוש הן בטכניקות התקפה והן בטכניקות הגנה. טכניקה זו ממנפת את הרעיון שכל פיסת קוד מכילה חורי אבטחה שניתן לנצל. בניסויי מעבדה, חקרנו וניתחנו מספר סוגים של פוגענים, פותחה מתודולוגיה וטכניקת זיהוי חדשה אשר מספקות שכבת הגנה נוספת במאבק נגד פוגענים - טכניקה המסוגלת אף לזהות נוזקות בלתי ידועות.

הרעיון העיקרי

שרשרת התקיפה - שיטת מודל לפלישה לרשת מחשבים שפותחה על-ידי לוקהיד מרטין (2011), מורכבת משבעה צעדים:

- Reconnaissance - איסוף המידע
- weaponization - חימוש
- delivery - שלב החדירה
- exploitation - ניצול
- installation - התקנה
- command and control - תקשורת עם שרת השליטה
- action on objectives - פעולות על יעדים.

בעוד שרוב טכניקות זיהוי קלאסיות מתמקדות בזיהוי הפלישה בשלבי החדירה, הניצול וההתקנה, הרעיון שלנו הוא לזהות את הפלישה בשלב התקשורת עם שרת השליטה. שלב ההתקנה הוא השלב בו הפוגען מותקן על עמדת הקצה, וזה מספק לו גישה מתמדת. לאחר מכן, הפוגען מתקשר עם שרת השליטה, ושולח לו אינפורמציה דרך עמדת הקצה שהודבקה על-ידי הפוגען (אינפורמציה כגון: שם המחשב, שם המעבד, גודל הזיכרון וכדומה). טבלה 1 מציגה רשימה חלקית של המידע שנאסף על-ידי פוגענים. מידע

זה (בדר"כ) נשמר בבסיס נתונים של שרת השליטה ומוצג בפאנל הניהול של התוקף שהוא ווב או אפליקטיבי.

בעבודתנו, אנו מנצלים את התקשורת שבין עמדת-הקצה הנגועה לבין פאנל השליטה כדי לגרום לפאנל השליטה לבצע פעולה בלתי רצונית שתשלח התרעה על הדבקה של פוגען בעמדת הקצה הנגועה. כדי לבצע זאת, אנו משתמשים בטכניקת XSS, ומזריקים מחרוזת XSS לתוך הנתונים שהפוגען שולף מעמדת הקצה הנגועה. במחקר זה, המיקוד היה בתוכניות פוגענים בעלי כלי אדמיניסטרציה שהוא פאנל ווב, כי דפי אינטרנט הם גורם יסודי לשם ניצול פגיעות ה-XSS. שינוי הנתונים נעשה רק על נתונים שאיסופם מעמדות הקצה (ללא ידיעת המשתמש) והצגתו נחשבים זדוניים.

באופן כללי ופשטני, בתקיפת סייבר מעורבות שתי ישויות מרכזיות: עמדת הקצה שהודבקה על ידי הפוגען, ושרת השליטה של התוקף. הפוגען מבצע את פעילויותיו הזדוניות על עמדת הקצה על-ידי קבלת פקודות מפאנל השליטה בתדירות מסויימת. הפאנל מסוגל לקבל אינפורמציה ולשלוח פקודות לעמדת הקצה הנגועה. כדי לבנות את מתודולוגיית הזיהוי החדשה שלנו, הוספנו ישות חדשה בתהליך התקיפה - שרת ניטור ייעודי (שרת C), המכיל קבצי תמונה הניתנים לפי דרישה (איזו דרישה? זוכרים את השימוש ב-XSS בפסקה למעלה?!, עכשיו רק צריך לראות איך זה מתבצע בפועל). אנו נגדיר ששרת זה לא אמור לתקשר עם אף ישות אינטרנטית. לפיכך, אם ישות מסוימת פנתה אליו זוהי אינדיקציה לפעילות זדונית. הרעיון ממומש בעמדות קצה בכלי הוכחת יכולת (POC) שפיתחנו, הנקרא PhoeniXSS, כפי שמתואר במפורט בסעיף "מתודולוגיית הזיהוי".

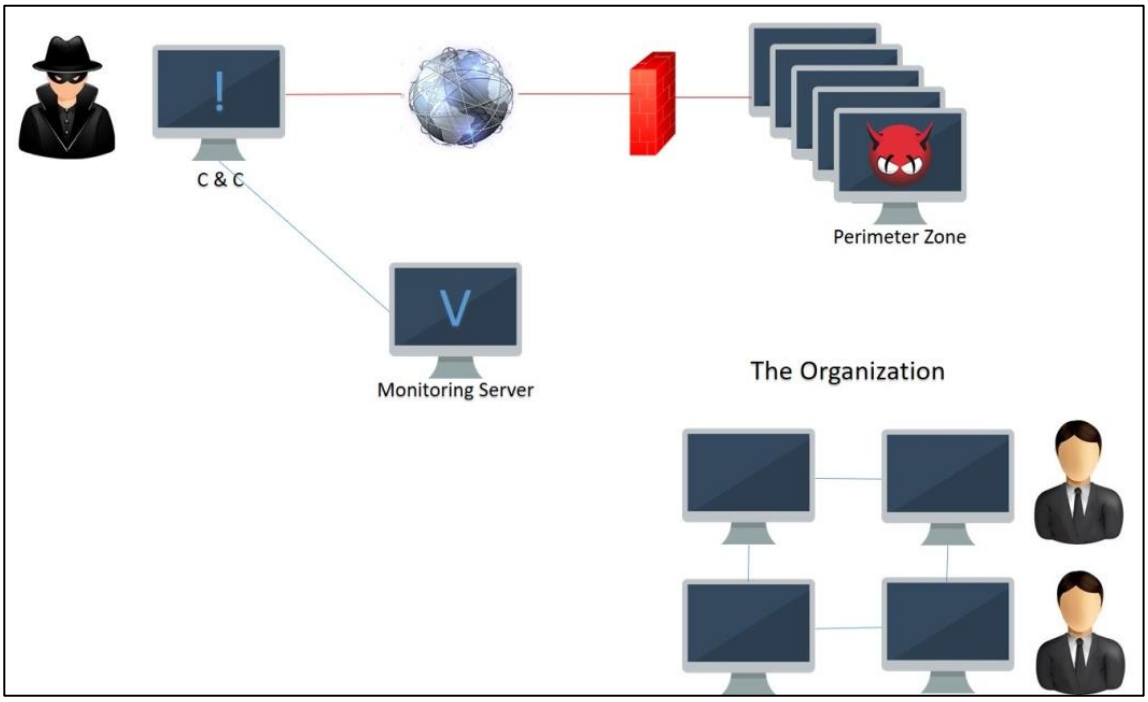
טבלה 1. נתונים ומידע שנאספים על ידי פוגענים:

Information Type	Details
System Info	OS version, install date, keyboard language, time-zone, hostname…
Hardware Info	Connected storage devices, Bios/CPU/RAM/Motherboard/ Network adapter info, installed video/audio devices, connected printers/monitors
Local Users Info	Description, username, password expiry date
Running Processes Info	Creation-date, process ID, path
Installed Services Info	Description, path, status
Environment Variables	
File Associations	File extension, associated program name, associated command
Network info	Internal IP, configured DNS servers
Directory listing	Root C drive, Desktop, Documents, Temp folder, …
Print screen	

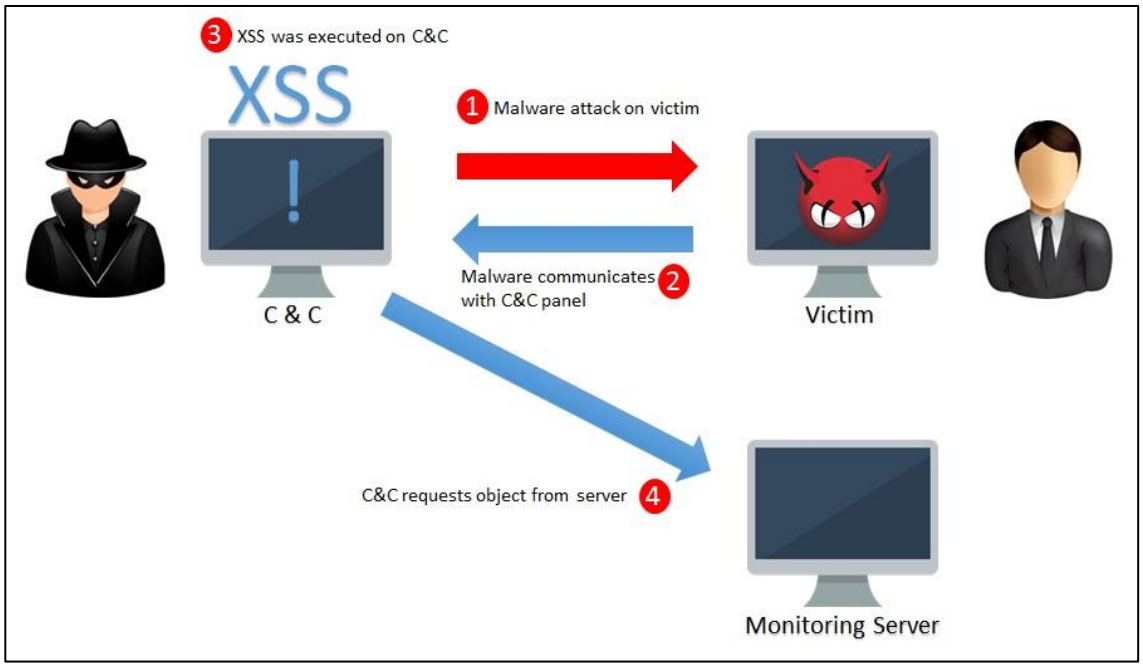
מתודולוגיית הזיהוי

בואו נניח שטכניקת הזיהוי שלנו כבר עובדת. כדי לפתח טכניקה אפקטיבית לזיהוי פוגענים, מבלי לפגוע בביצועים של אפליקציות בעמדות הקצה בארגון, נקים סביבה חדשה שתהווה שכבה נוספת בפרימטר של הארגון ותרכז את כל קבצי ההרצה הנכנסים לארגון. כל קובץ שיכנס לארגון יורץ בנפרד בשכבה חדשה זו עם הכלי שלנו (המכיל את טכניקת הזיהוי, שכבר עובדת ;). כיום, ארגונים רבים מבצעים בדיקות לזיהוי פעולות זדוניות בקבצים לפני שאלו נכנסים לסביבת העבודה, מערכות ייעודיות, כלי חקירה אוטומטיים ושימוש בארגזי חול. ניתן לצרף את טכניקת הזיהוי שלנו לכל אחד מיכולות אלו הקיימות בארגון. שרת הניטור שלנו ישב באינטרנט וימתין לבקשות. הסביבה החדשה מתוארת באיור 1.

עמדות הקצה בסביבת הפרימטר החדשה שלנו משמשות להרצת קבצים בינאריים שהגיעו אל הארגון. כל קובץ חייב היה להיות מופעל על-ידי כלי PhoeniXSS למשך זמן X מסוים שהארגון הגדיר מראש לפני שהוא נכנס לארגון. במהלך הפעלת הקובץ, PhoeniXSS מבצע הוקינג לפונקציות API שהוגדרו מראש, ומוסיף מחרוזת XSS מוגדרת מראש לנתונים שהקובץ הבינארי שולף מהעמדה. במחקר זה, נבדקו פוגענים עם מחרוזת XSS ספציפית באמצעות שימוש בכלי ה-PhoeniXSS. במהלך ההדבקה (בהנחה שהקובץ הוא זדוני), הנתונים שנשלפו נשלחים לפאנל הניהול המרוחק של התוקף. פאנלי ניהול רבים מציגים נתונים חופפים/זהים על המכונות שהודבקו (לדוגמה, שם המחשב, שם המשתמש, ושם המעבד). אולם מספיק שפעם אחת הנתונים האלו לא עברו ולידציה ו/או טופלו באופן לא מתאים, פאנל הניהול של התוקף הופך להיות פגיע למתקפת XSS. אנו השתמשנו בפגיעות זו כדי ליצור התרעה בשרת הניטור שלנו. לבסוף, שרת הניטור מתוכנת לחכות לבקשת HTTP/S. אמנם כל רכיב המחובר לאינטרנט יכול לגשת לשרת הניטור, אך אף רכיב לא אמור לשלוח אליו בקשה לקבלת קבצי התמונה שהוא מארח. לפיכך, בקשה שמתקבלת שקולה לפלישה לארגון.



[איור 1. מתודולוגיית הזיהוי]



[איור **2**. טכניקת הזיהוי]

מימוש

בסעיף זה, ניתנים פרטי המימוש עבור כלי ה-PhoeniXSS.

PhoeniXSS

אנו מימשנו את הגישה שלנו בכלי הוכחת-יכולת הנקרא PhoeniXSS, המורכב משני פרוייקטים נפרדים:

1. Application Verifier - הוא כלי המגיע עם Windows ומאפשר לנו להוציא מידע דיאגנוסטי רב על תוכנות הרצות במחשב כגון הקצאות זיכרון שמתרחשות, Handles שנפתחים/נסגרים, אירועי רשת וכו'.
2. MinHook - API hooking library, שמספקת פונקציונליות הוקינג בסיסית עבור סביבות x64 ו-x86. ספרייה זו מספקת מימוש של inline, IAT הוקינג. ה-PhoeniXSS מורכב מקובץ DLL יחיד ומרכזי, שמצרף את כל הפרוייקטים הנ"ל לתוך קובץ בינארי המופעל בתחילת הביצוע, תוך שימוש בכלי Microsoft Application Verifier.

הכלי PhoeniXSS מריץ קבצים בינאריים בהרשאות של Administrator, ומבצע inline hooking על פונקציות API שהגדרנו מראש. מאחר שתוכניות פוגענים רבות משתמשות ב-COM objects לשם ביצוע שאילתא על נתונים בעמדות הקצה, הכלי שלנו יכול גם לבצע הוק על שאילתות COM מוגדרות מראש. קובץ ה-DLL של הכלי שלנו נטען לזיכרון לפני טעינת ה-DLLs המיובאים של קובץ ההרצה. אולם הוקינג של פונקציות API שהוגדרו מראש מבוצע רק לאחר שה-DLLs המיובאים נטענו לזיכרון (כלומר, ה-DLL שלנו נטען לזיכרון ומבצע הוקינג לפונקציית ה-API "GetWindowTextA", ששייכת ל-User32.dll, רק לאחר שה-DLL הזה נטען לזיכרון). לאחר שה-DLL המרכזי שלנו נטען לזיכרון, הוא מיירט את פונקציית ה-API המקורית, דורס את הפרולוג של הפונקציה, ומדלג מקטע הקוד החדש שלנו. קטע קוד זה שולף את הנתונים של פונקציית ה-API המקורית ומשנה אותם. שינוי זה מוסיף מחרוזת XSS ספציפית מוגדרת מראש לנתונים שנשלפו מהפונקציה המקורית.

מחרוזת ה-XSS המוגדרת מראש מורכבת מתגית תמונה, תגית זו מכילה את קובץ התמונה שנמצאת על שרת C שלנו. שרת זה מארח אפליקציית ווב שמפרסמת מספר תמונות gif. כל תמונה מוכלת בתוך מחרוזת XSS שונה. אורך מחרוזת ה-XSS צריך להיות מינימלי כדי להימנע ממגבלות אורך בבסיס הנתונים של פאנל השליטה. בנוסף, מחרוזת ה-XSS צריכה להיות מורכבת מתווים שונים כדי להימנע ממגבלות תווים בפאנל השליטה. התבנית של מחרוזת ה-XSS שלנו הוא:

```
<img/src=//j.en/X.gif>
```

כאשר: 'X' - הוא מספר והשם של תמונת ה-GIF. תמונת ה-GIF היא תמונה שקופה, על מנת לשמור על חשאיות לאחר הרצת ה-XSS בדפדפן של התוקף. יתרה מזאת, גודל התמונה הוא 1×1 pixel, כדי להימנע מבקשות עם תקורה משמעותית בפאנל השליטה וכמובן להמשיך לשמור על חשאיות. לאחר שינויים

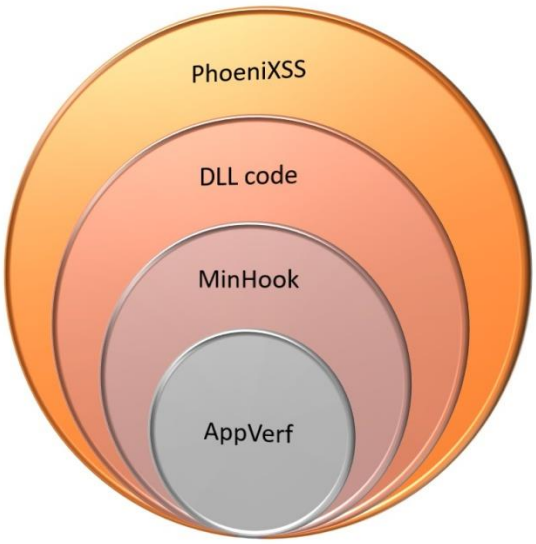
שביצע ה-PhoeniXSS, הנתונים המוחזרים של פונקציות ה-API מופיעים בפורמט הבא: ערך מקורי + התג של התמונה. למשל, אם הערך המקורי של פונקציות ה-GetWindowTextA" API" היה "explorer.exe", אז אחרי השינויים של ה-PhoeniXSS, הערך החדש יהיה:

```
explorer.exe<img/src=//j.en/1.gif>
```

הפונקציה "GetWindowTextA" היא דוגמה לפונקציה אחת מתוך הרשימה של פונקציות מוגדרות מראש בכלי שלנו. פונקציות ה-API שהוגדרו מראש לביצוע הוקינג בכלי ה-PhoeniXSS הן:
"GetWindowTextW","GetComputerNameA", "GetComputerNameW", ו-"GetWindowTextA".
הפונקציה "GetComputerName" מאחזרת את שם ה-NetBIOS של המחשב המקומי, בעוד שהפונקציה GetWindowText מאחזרת את הטקסט של שורת הכותרת של החלון שצויין.

כפי שנזכר קודם לכן, הכלי PhoeniXSS גם מיירט קריאות לממשקי COM. הוא מממש את שיטת ההוקינג "vtable patching" עבור הפונקציה "ExecQuery" של הממשק "IWbemServices". לאחר שה-DLL המרכזי שלנו נטען לזיכרון, הוא מחכה שהקובץ "wbemsvc.dll" ייטען, כדי ליצור אובייקט של הממשק לעיל ולבצע הוקינג לפונקציה הנ"ל. ה-PhoeniXSS משנה את טבלת ה-methods הוירטואלית של האובייקט, שמכילה מצביעים לכל ה-methods הציבוריות של אובייקט COM, כך שניתן יהיה להחליפם במצביעים לפונקציות ההוק החדשות שלנו. כל פונקציית הוק כזו קוראת לפונקציה המקורית, מקבלת את הנתונים המוחזרים ממנה, ומוסיפה אליהם את מחרוזת ה-XSS. התנהגות הכלי מתוארת באמצעות הפסאודו-קוד הבא:

1. עבור כל קובץ בינארי שנכנס לארגון:
   1.1 קינפוג ברגיסטרי כדי שירוץ עם Application Verifier
   1.2 הרצה של הקובץ עם הרשאות אדמין
   1.3 הזרקת הDLL לתהליך
   1.4 ביצוע הוקינג לפונקציות API מרכזיות
   1.5 עריכה של הנתונים והוספת המחרוזת שלנו
   1.6 קפיצה חזרה לקטע הקוד המקורי
   1.7 דילוג לסעיף 1.1



[איור 3: המבנה של הכלי PhoeniXSS]

ניסויים על פוגענים

הקמנו סביבה מבודדת לשם אירוח שרת השליטה, המכונה של הקורבן, ושרת הניטור שלנו, כדי לערוך ניסויים עם טכניקת הזיהוי על פוגענים. השגנו את ארגז הכלים המלא עבור מספר פוגענים שדלפו. ארגזי הכלים כללו תוכנת בניית בוט ואת אפליקציית פאנל השליטה. ארגזי הכלים היו עבור הפוגענים הבאים: MegalodonHTTP v1.0, Dexter POS v1.0, ו-DiamondFox v4.2.650. הנחנו כי ארגזי הכלים המודלפים הינם זהים במהותם למקורות שנמצאו במרחב הפרוע. לא ביצענו מבדק על פוגענים "חיים" מפני שמתקפות-נגד (כלומר, ביצוע האקינג-נגדי) אינן חוקיות. חקרנו את הפונקציונאליות של הפוגענים ואת קוד המקור שלהם, כדי לקבוע האם טכניקת הזיהוי שלנו יכולה להיות ישימה כנגדם. כדי לבדוק אם ה-PhoeniXSS מסוגל לזהות את הפוגענים לעיל, הקמנו פאנלי שליטה על השרת שלנו וקינפגנו את כלי ה-PhoeniXSS שלנו.

ניסויי הפוגענים שלנו נוהלו כדלקמן: הפוגען הראשון שנבדק (Dexter) תקשר עם פאנל השליטה המרוחק שלו בטקסט ברור וללא תיקוף קלט על מכונת הקורבן. הפוגען השני (MegalodonHTTP) שנבדק תקשר עם פאנל האדמיניסטרציה המרוחק שלו תוך שימוש בהצפנה וללא תיקוף קלט. הפוגען האחרון (DiamondFox) שנבדק תקשר עם פאנל האדמיניסטרציה המרוחק שלו תוך שימוש בהצפנה ועם תיקוף קלט, אבל הוא עדיין נתגלה כפגיע לטכניקת זיהוי ה-XSS שלנו. השלבים השונים של הניסוי מתוארים בסעיף הבא.

הקמת הסביבה

סביבת הבדיקה שלנו כללה שלוש מכונות וירטואליות (VM) שרצו על ה-Intel Core i7-4720HQ CPU עם 16GB של RAM ו-230GB של מקום בדיסק. המכונה הראשונה היא מכונת הקורבן, ורצה על Windows 7VM, 32-bit, Service Pack 1. מכונה זו היתה אחראית להפעלת ה-payload file תוך שימוש בכלי ה-PhoeniXSS. הקמנו מכונה נוספת שתארח פאנלי שליטה של הפוגענים. מכונה זו רצה על: Windows 7 VM 32-bit, Service Pack 1. שלושה דפדפנים שונים הותקנו על שרת השליטה עם קונפיגורציית ברירת המחדל. תוך שימוש בכל אחד מהדפדפנים - דפדפן Firefox browser version 53.0.3, דפדפן Chrome 51.0.2704.103, ודפדפן Microsoft Edge 38.14393.1066.0 - טכניקת הזיהוי שלנו נבדקה על פאנלי השליטה של הפוגענים שנאספו. פאנלי השליטה של תוכניות הפוגענים שנאספו התארחו על מכונת Windows עם שרת Apache web server ובסיס נתונים MYSQL DB. המכונה האחרונה, שרת הניטור שלנו, רצה על Windows Server 2008 VM, 32-bit Service Pack 1.

תהליך הבדיקה

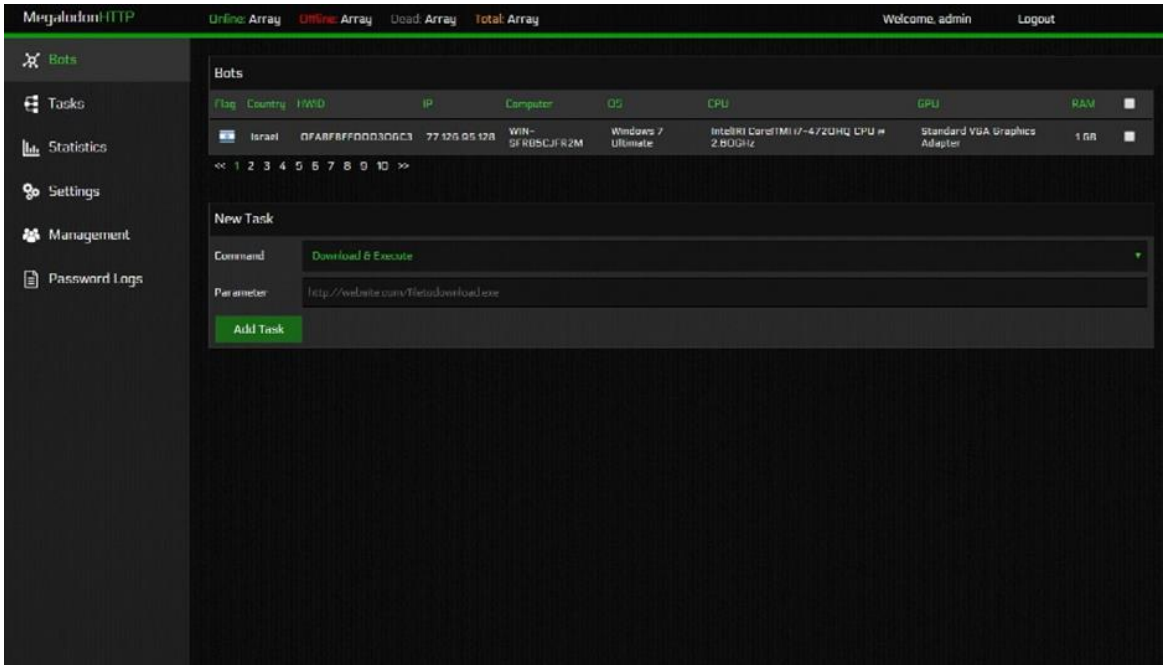
לאחר הקמת סביבת הניסויים שלנו, השתמשנו בכלי שלנו כדי להריץ את ה-payload של הפוגענים שנאספו. הבדיקה של כל אחד מהפוגענים בוצעה עם אימג' מ"ה נקי ומופרד. יתרה מזאת, כל פאנל הותקן בנפרד מהפאנלים אחרים.

הבדיקה הראשונה שלנו בוצעה על הבוטנט MegalodonHTTP. MegalodonHTTP הוא בוטנט, פוגען שפותח להפעלת בוטים של תקיפות DDOS. בוטנטים כאלה יכולים לבצע שבעה סוגים של התקפות DDoS, לפתוח shell מרוחק על המערכת המודבקת על-ידי פוגען, לבצע כרייה של מטבעות קריפטו, ולהרוג אנטי-וירוסים על העמדה המודבקת. ה-MegalodonHTTP מתקשר עם פאנל השליטה שלו מעל HTTP.

לאחר בניית בסיס הנתונים שלו, ולאחר הקמת הפאנל הניהול של הבוטנט, חקרנו את קוד-הפוגען בצד השרת כדי למצוא פגיעויות XSS. בנוסף, בדקנו את המידע ש-MegalodonHTTP חושף בפאנל שלו, למשל: שם המדינה, מספר זיהוי חומרה, IP, שם המחשב, מערכת ההפעלה, מעבד, כרטיס גרפי, זיכרון נדיף, אנטי-וירוס ועוד. מידע זה נשלח לפאנל השליטה ונשמר בבסיס הנתונים, אך ללא תיקוף קלט. לפיכך, פאנל ה-MegalodonHTTP כנראה פגיע למתקפת XSS. תמונה של פאנל ה-MegalodonHTTP מוצג באיור 2. יתרה מזאת, בדיקה נוספת נדרשת כדי לאפשר את מתקפת ה-XSS. המידע הגנוב נשלח לשרת ונשמר בטבלאות בסיס הנתונים שלו. לכל טבלה יש שדות שונים. שדות אלה הם באורכים שונים. אם קיימת מגבלת אורך על שדות אלה יתכן שלא נוכל להכניס את מחרוזת ה-XSS שלנו לבסיס הנתונים, ולכן יתכן כי מתקפת ה-XSS אינה ישימה. היו מספר שדות שאורכם לא היה מוגבל-דיו ולכן התקיפה שלנו הייתה ישימה. לדוגמה, האורך המקסימלי של שם המחשב במ"ה Windows מוגבל ל-15 תווים, והאורך של שדה זה בבסיס הנתונים של ה-MegalodonHTTP הוא 255 תווים. אנו בחרנו לשנות את ערך שדה ה-CPU של המכונה הנגועה. כדי לנצל את פגיעות ה-XSS, MegalodonHTTP מקבל את הערך המוזכר לעיל תוך שימוש בשאילתת WQL הבאה:

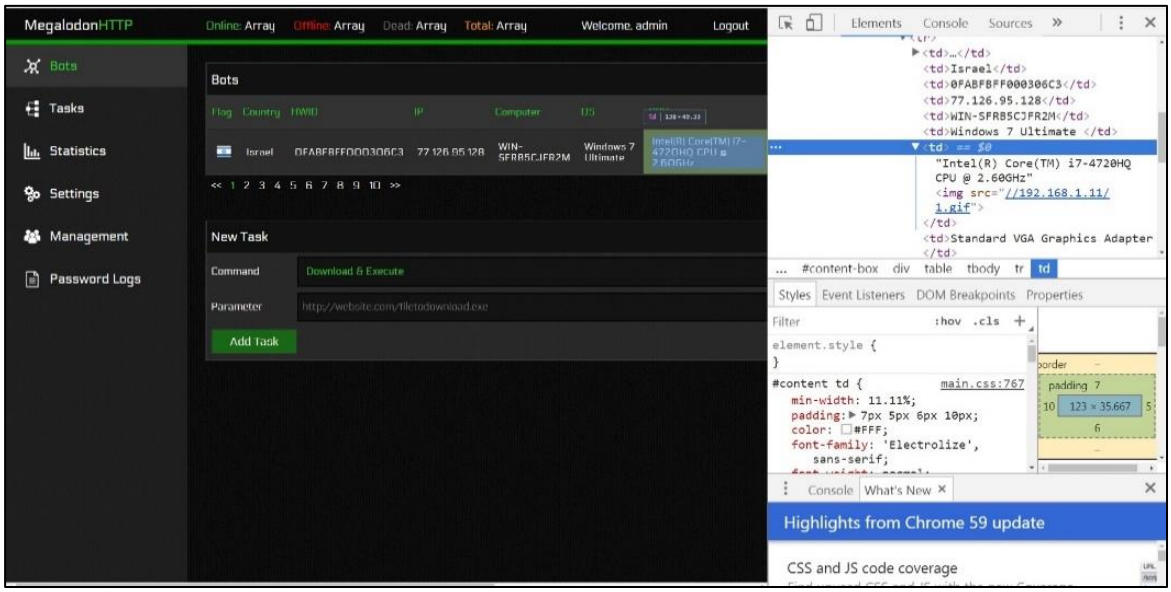
```
SELECT Name FROM Win32_Processor
```

המחלקה WMI של ה-Win32_Processor מאחזרת את הסטאטוס של המעבד וכן מידע עבור מכונות בעלות מעבד אחד וגם עבור מכונות בעלות מעבדים רבים. השאילתא לעיל מחזירה את שם המעבד של עמדת הקצה הנגועה. כלי ה-PhoeniXSS מתוכנת לשנות את שדה ה-CPU תוך שימוש ביכולתו לבצע הוקינג על ממשקי COM.



[איור 4: פאנל האדמיניסטרציה האינטרנטי של ה-MegalodonHTTP]

תוך שימוש בבונה הבוט של MegalodonHTTP, בנינו את ה-payload file עם קונפיגורציית ברירת מחדל, והרצנו אותו בעזרת הכלי ה-PhoeniXSS על מכונת הקורבן שלנו. לאחר ההפעלה, מידע ממכונת הקורבן הוצג על פאנל השליטה. מחרוזת ה-XSS של הכלי שלנו שהוזרקה לא הוצגה על הפאנל, מפני שהדפדפן התייחס למחרוזת המוזרקת כאל קוד HTML במקור של הדף. לפיכך, מנקודת הראות של התוקף, זה נראה היה כאילו המידע המקורי הוצג. אולם, בו-בזמן, המגן (כלומר, הקורבן) זיהה שהמכונה שלו הודבקה בפוגען וששרת הניטור שלו מספק מידע למכונה של התוקף. תמונה של פאנל ה-MegalodonHTTP לאחר מתקפת ה-XSS שלנו ניתנת באיור 5.



[איור 5: פאנל הניהול של MegalodonHTTP לאחר מתקפת ה-PhoeniXSS]

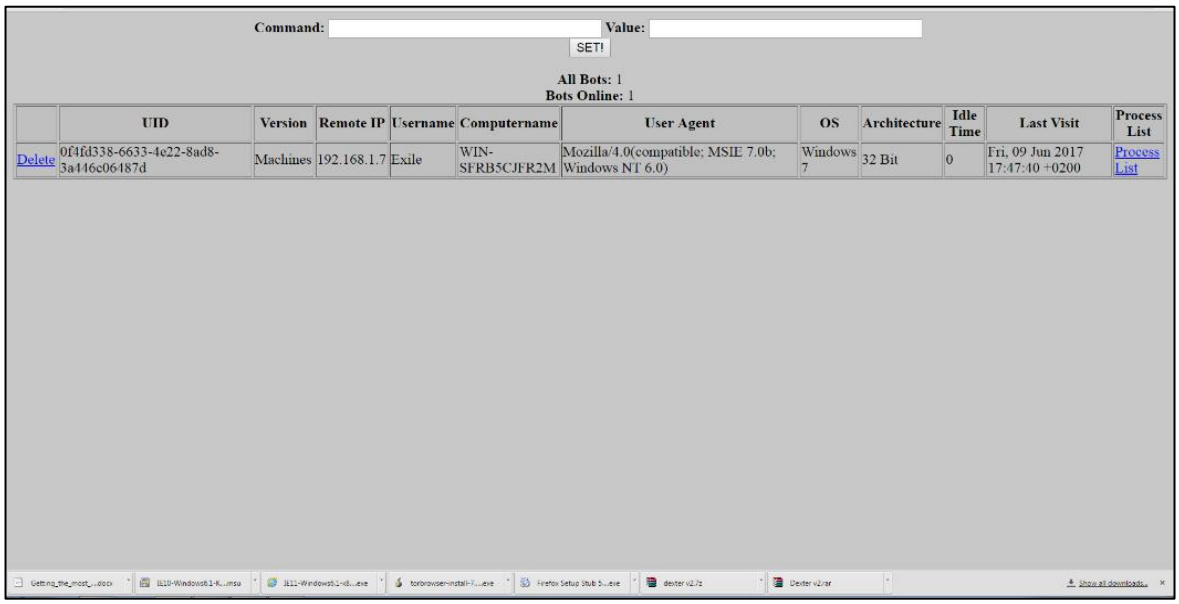
בדיקה נוספת בוצעה על הפוגען Dexter. Dexter הוא וירוס המטרגט עמדות סליקה, אשר מדביק מחשבים בעלי מערכת הפעלה של Microsoft Windows. הוא מדביק מערכות PoS ברחבי העולם, גונב מידע רגיש כגון מידע על כרטיסי אשראי וכרטיסי חיוב, ושולח את המידע לשרת השליטה. קובץ ההרצה הזדוני שמותקן על עמדת קצה ב-PoS ידוע כ-PoSGrabber. הוא מתוכנת כך שהוא יכול לרוץ כקובץ-הרצה או להיות מוזרק לתוך תהליך אחר. ה-PoSGrabber מתקשר עם פאנל השליטה מעל HTTP. התקשורת הזו מקודדת (encoded), והרבה ממנה "מוצפן".

אחד ההבדלים העיקריים שבין הפוגען MegalodonHTTP לבין הפוגען Dexter הוא התעבורה שנשלחת לפאנלי השליטה שלהם. התעבורה של הראשון נשלחת בטקסט ברור, בעוד שהתעבורה של השני נשלחת בהצפנה תוך שימוש בקידוד Base64 ובצופן XOR. תוך שימוש בכלי PhoeniXSS על הפוגען Dexter, ביססנו את העובדה שהטכניקה שלנו עובדת גם על פוגענים שמשתמשים בהצפנה כדי לתקשר עם פאנל השליטה שלהם.

עקבנו אחר אותם שלבי התקנה עם ה-setup של ה-MegalodonHTTP. לפאנל יש שלושה דפים עיקריים: Dump Viewer, Bot Control, ו-File Uploader. הראשון מייצג מעקבים על כרטיסי אשראי, השני מאפשר שליחת פקודות לבוטים ומראה מידע גנוב מעמדות הקצה המודבקות בפוגען, והדף השלישי מאפשר העלאה של קבצים נוספים שאז ה-PoSGrabbers יוכלו להשתמש בהם.

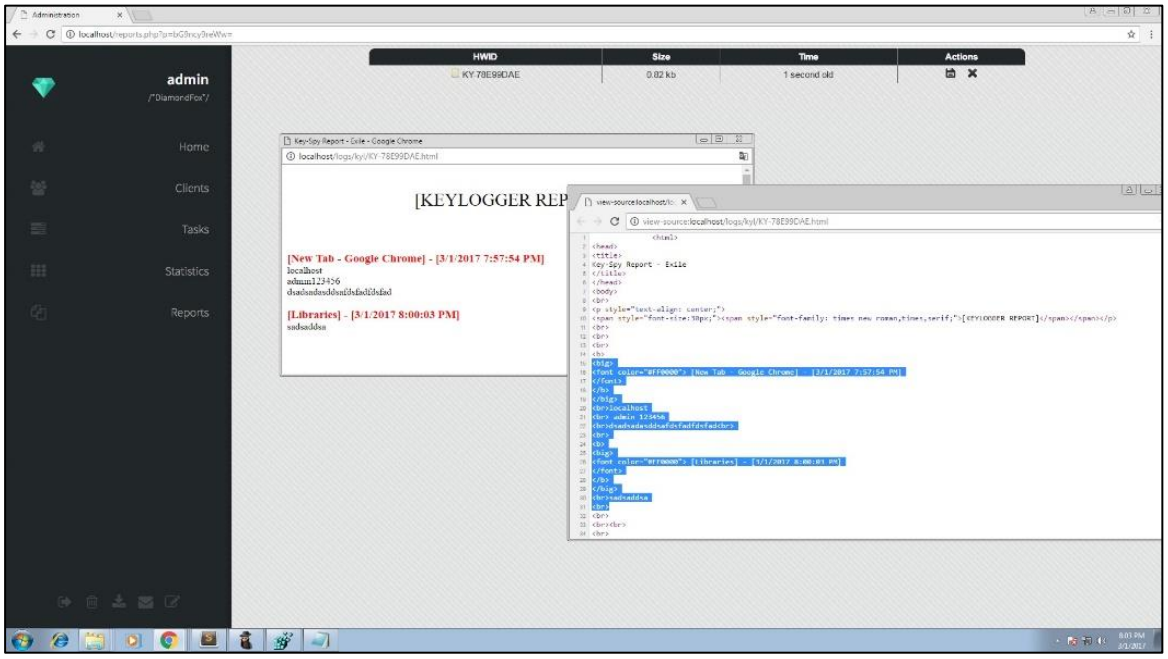
בדקנו את קוד צד השרת וצפינו במידע שהוצג על פאנל השליטה של Dexter ומצאנו את המידע הגנוב הבא: UID Version, IP, שם משתמש, שם מחשב, user-agent, מערכת הפעלה ועוד. נבדק כל שדה מידע גנוב ב-DB ונבדקו השיטות ששימשו כדי לאחזר מידע גנוב זה מהמכונה המודבקת, וכך נתגלה שלא היה שום ניקוי קלט על המידע שנשלח לפאנל השליטה של Dexter. בחרנו לשנות את הערך של שדה שם המחשב שנשלח לפאנל ה-Dexter. ה-payload file עם קונפיגורציית ברירת המחדל נבנה תוך שימוש בקוד המקור של הפוגען, ולאחר מכן הורץ בסביבה המבודדת שלנו.

לאחר הביצוע, המידע הגנוב הוצג בדף ה-Bot Control על הפאנל. קוד צד השרת פענח את המידע שנשלח לפאנל הניהול המרוחק של הפוגען, כולל את מחרוזת ה-XSS שהוזרקה על-ידי כלי ה-PhoeniXSS. הדפדפן של התוקף התייחס למחרוזת המוזרקת כאל HTML במקור של הדף, וכך התרעה על הדבקה קפצה בשרת הניטור של המגן. תמונה של פאנל ה-Dexter לאחר ההדבקה של הפוגען ניתנת באיור 6.



[איור 6: פאנל האדמיניסטרציה האינטרנטי של Dexter לאחר מתקפת PhoeniXSS]

המבדק האחרון בוצע על הפוגען DiamondFox, שהוא רב-פונקציונאליות, כולל גניבה של כרטיסי אשראי וגם גניבה של הרשאות בעמדות סליקה. הוא נגיש מאד אפילו להאקרים המוגבלים ביותר, מכיוון שהוא מופץ בפורומי האקרים רבים. DiamondFox מתקשר מעל HTTP מוצפן עם מפתח (key) שנבנה באופן סטאטי לתוך פאנלי השליטה ולתוך הבוט עצמו.

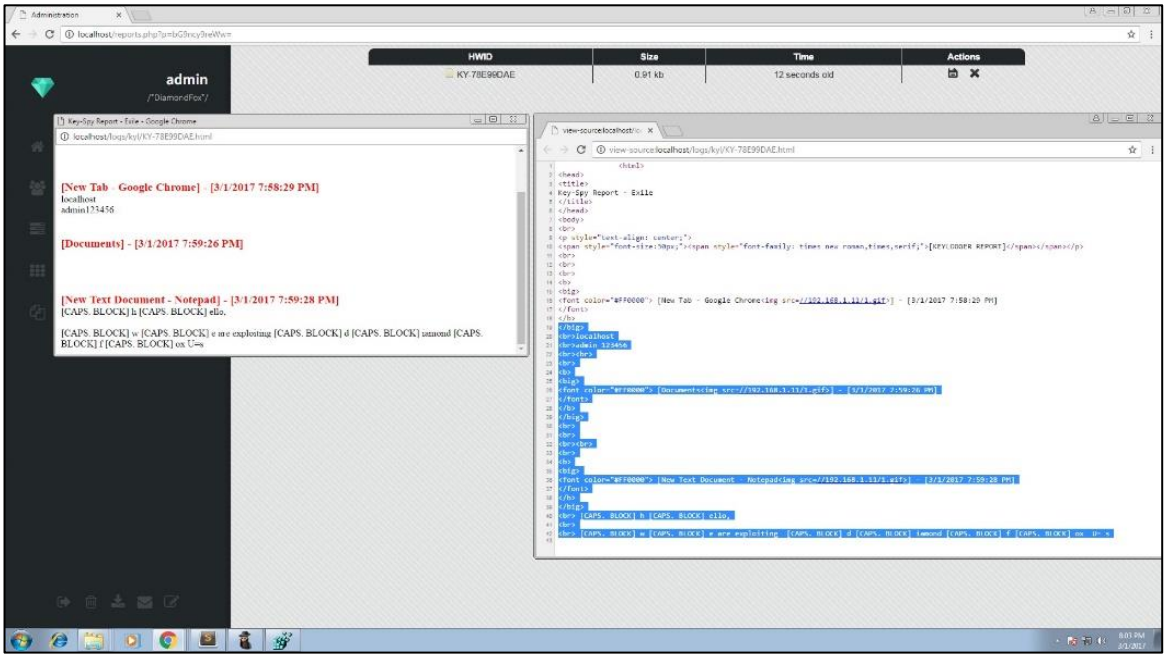


[איור 7: מודול ה-key logging של DiamondFox]

נעשה שימוש בדף ההתקנה של DiamondFox כדי להקים את פאנל הניהול וכדי לקנפג את בונה הבוט שלו לייצא payload file הכולל את מודול הקילוגר שלו. כדי לבסס את העובדה שפאנל הDiamondFox-הוא פגיע, חקרנו את קוד צד השרת של הפאנל, וקיבלנו מידע באמצעות הבוט ממכונת הקורבן.

המידע הבא הוצג על דף ה"לקוח" על הפאנל של DiamondFox: מזהה חד ערכי, ארץ, שם מחשב, מערכת הפעלה, ומידע נוסף אודות מכונת הקורבן (גודל הדיסק הקשיח , גודל הזיכרון הנדיף ועוד). הפאנל של DiamondFox משתמש במספר פונקציות מניעת XSS כדי לטפל בנתוני קלט מהבוטים שלו לפני שהוא מכניס אותם לבסיס הנתונים שלו. פירוט נוסף על הפונקציות מופיע במאמר המלא באנגלית.

שימוש פונקציות אלו מנעה מהכלי שלנו מלנצל חלק זה של הבוטנט. ואולם, DiamondFox הינו בוטנט עם פונקציונאליות רבה ומגוון מודולים, ואחד מהם הוא יכולת הקילוגינג שלו. ואכן, גם מודול זה כולל בדיקת input escaping על הקלט המתקבל מהעמדה המודבקת, אבל לא על כל הקלט. תמונה של מודול הקילוגינג של DiamondFox ניתנת באיור 7.



[איור 8: מודול הקילוגינג של DiamondFox לאחר שימוש בכלי PhoeniXSS]

קובץ ה-payload השתמש בפונקציית ה-"GetWindowTextA API" , כדי לעקוב אחר הקלדות המשתמש על לוח המקשים בין חלונות שונים. אבל פאנל הבוטנט לא הצליח לתקף ולסנן את קלט הנתונים בפונקציה GetWindowTextA, ועל-כן דבר זה הופך את פאנל הבוטנט לפגיע למתקפת XSS ולניתן לזיהוי על-ידי הטכניקה שלנו. תמונה של מודול הקילוגינג של DiamondFox לאחר שימוש בכלי ה-PhoeniXSS ניתנת באיור 8.

סיכום ועבודות עתידיות

מחקר זה מגלה תחום מחקרי חדש בנושא זיהוי והגנה אופנסיבית. בכדי לזהות פעילות זדונית על עמדות קצה, אנו פיתחנו מתודולוגיית זיהוי חדשנית וכלי הוכחת יכולת לביצוע מתקפת-נגד מסוג XSS על פאנלי השליטה של פוגענים. על-ידי ניצול חולשות הפוגען, הראינו כיצד הטכניקה שלנו מסוגלת לזהות שלושה פוגענים שונים, באמצעות מניפולציה של המידע בעמדות הקצה ושימוש בתקשורת המהימנה בין הפוגען לפאנל השליטה שלו.

מחקר זה מקדם פתרון חדשני שמתמקד בהתקפה, פתרון שיאפשר לקורבנות פוגען לבצע האקינג-נגדי באופן לגיטימי. מעבר לכך, ניתן להרחיב את המתודולוגיה שלנו בכיוונים שונים בעתיד, כגון ניצול הזרקת SQL בפאנלי אפליקציה או שימוש במסמכים המכילים חולשה כדי להקפיץ התרעה כאשר הם נפתחים, ועוד. בעבודות עתידיות, אנו מקווים להיות מסוגלים לזהות משפחות חדשות של פוגענים על-ידי שימוש בכל הטכניקות הנזכרות לעיל in the wild. לדעתנו, למימוש של כיוונים אלו יש פוטנציאל לשנות את חוקי המלחמה הקיברנטית וליצור מערכת כוחות מאוזנת בין גופי ההגנה לגופי ההתקפה במימד הסייבר.

יריבים ימשיכו לפתח טכנולוגיות חדשות בכדי להימנע מזיהוי ולהשיג את מטרותיהם. אסור לנו לפגר מאחור, אלא עלינו להפוך את הקערה על פיה ולנצל את חולשותיהם נגדם, תוך שימוש שיטות התקפיות שכוללות טכניקות זיהוי בעלות אמינות גבוהה.

על המחבר

מאמר זה נכתב על ידי שי נחום. שי הינו מהנדס מערכות מידע ובעל תואר שני מהטכניון עם ניסיון בתחום פיתוח מאובטח, בדיקות חדירות, חקירת פוגענים וייעוץ לפרויקטים.

לפניות או שאלות ניתן לפנות אלי דרך חשבון ה-LinkedIn:

https://www.linkedin.com/in/shay-nachum-49260b61/

תודות

תודה רבה לאורית כהן, עידן ברגמן, גיא כלפון ושרה לחצר שטרחו ונתנו מזמנם לקרוא את הטיוטה והביאו הצעות לשיפורים.

קריאה נוספת

Original Paper

- https://ufile.io/zg3rv

Application Verifier

- http://www.kernelmode.info/forum/viewtopic.php?f=15&t=3418
- https://msdn.microsoft.com/en-us/library/windows/hardware/ff538115(v=vs.85).aspx
- http://blogs.msdn.com/b/reiley/archive/2012/08/17/a-debugging-approach-to-application-verifier.aspx
- https://www.digitalwhisper.co.il/files/Zines/0x46/DW70-2-ASM&AVRF.pdf

COM Hooking

- https://msdn.microsoft.com/en-us/library/cc226936.aspx
- https://msdn.microsoft.com/en-us/library/cc250762.aspx
- https://www.codeproject.com/articles/153096/intercepting-calls-to-com-interfaces
- https://he.wikipedia.org/wiki/COM
- https://msdn.microsoft.com/en-us/library/windows/desktop/ms680573(v=vs.85).aspx

XSS

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

MinHook

- https://www.codeproject.com/Articles/44326/MinHook-The-Minimalistic-x-x-API-Hooking-Libra