בס"ד

פרויקט יסודות אבטחת תוכנה

קבוצה 4

PE infector - cleaner

**מוגש ע"י**

**מס' קבוצה 4**

**יונתן קרביץ 313772741**

**אלעד אליהו 305317877**

מנחה פרויקט:

אריה הנל

המרכז האקדמי לב

תשע"ז סמסטר א'

תוכן עניניים:

[Abstract 3](#_Toc473891952)

[תיאור הפרויקט: 4](#_Toc473891953)

[רקע תיאורטי 4](#_Toc473891954)

[רקע על קבצי PE: 4](#_Toc473891955)

[מבנה הקובץ: 4](#_Toc473891956)

[PE injection- 6](#_Toc473891957)

[Code cave 6](#_Toc473891958)

[Add section 6](#_Toc473891959)

[בנית אזור חדש 7](#_Toc473891960)

[אופן פעולת ההדבקה: 8](#_Toc473891961)

[Cleaner 8](#_Toc473891962)

[בעיות והתמודדויות: 9](#_Toc473891963)

[תובנות ומסקנות: 9](#_Toc473891964)

[הרצת הקוד: 10](#_Toc473891965)

[ביבליוגרפיה: 11](#_Toc473891966)

[נספחים 12](#_Toc473891967)

1. Abstract

* **Introduction**: Our project is about Portable Executable file manipulation. At first we thought it would be more about how evil payload are done, but like with any power you can use it for good or evil. Learning the PE structure is a huge tool to have for anyone who wants to get a deeper knowledge about how executable files work. Anyone wanting to dive deeper into the way malware analysis is done and how attackers can insert harmful programs into a system must have this knowledge in their tool kit. It will make anyone think twice before running their next executable they download from the net. The program we developed here is just the tip of the iceberg of the capabilities a programmer can do to PE files.
* **Our Goal**: our goal was to take an executable file (or PE- portable executable) and insert into it a program that we would run before the original file would. Our program would ask for the user a password to run the program. If the user doesn’t have it the program will exit. If the user entered the correct password the window would close and return the control back to the original file. We also created a program to clean the license requirement thus returning the program to its original state.
* **Procedures**: We investigated and researched a lot about the structure of the executable since we had no previous knowledge of the PE. It was a must since the structure is very precise and any change in the structure can destroy the entire program from running. We had to also choose the way we were going to implement the program to run within the new PE.
* **The end result**: we learnt a lot about the portable executable and different techniques people use to manipulate the structure to work in different ways. We now have an independent program we can run on a PE and make it add the license check, as well as clean it from the license request.

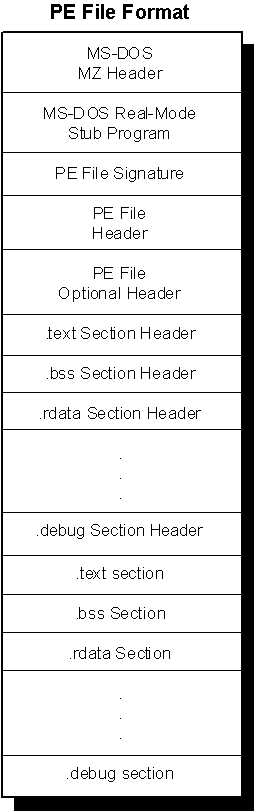
1. תיאור הפרויקט:

הנושא עליו עבדנו בפרויקט הינו הדבקת קובץ הרצה (portable executable- להלן PE) בתוכנית אחרת שתרוץ. התוכנית צריכה לבקש מהמשתמש את הסיסמא הנכונה, במידה והסיסמא שהוכנסה נכונה התוכנית תמשיך ותריץ את התוכנית המקורית. במידה והמשתמש הכניס סיסמא לא נכונה התוכנית תודיע שסיסמא לא נכונה ותפסיק את התוכנית. במסגרת הפרויקט היה לנו המון ידע קודם שהיינו צריכים ללמוד על מנת לבצע את התוכנית. נביא כאן בדו"ח את עיקרי העניינים התיאורטיים וכמה קטעי קוד נבחרים להראות איך הקוד עובד. כמו כן נביא את מבנה הקובץ שהוכנס ואיך מריצים אותו.

מטרתנו העיקרית בבחירת הנושא היתה לדעת כיצד ניתן להזריק קוד אל תוך קובץ מסוג PE מתוך הבנה כי זהו כלי מאוד חשוב ושימושי בכנות בכלל , וכי זהו כלי שמביא הבנה עמוקה יותר בתחום אבטחת התוכנה.

הרעיון הכללי שקידמנו בפרויקט הוא הזרקה של קטע קוד חדש של License אל תוך קובץ exe קיים. לצורך כך היה עלינו לחקור את מבנה קובץ הPE. הנושא הוא ממש גדול ורחב וברור לנו שלא הקפנו לחלוטין את כל הנושאים, אך הבנו את עיקרי הענינים המספיקים לנו כדי ליישם את הפרויקט והבאנו אותם כאן במסמך זה. לאחר יצירת הLicencer, עסקנו ביצירת cleaner על מנת שבסוף התהליך נוכל לנקות את מה שיצרנו בקובץ והחזרת הקובץ למצבו המקורי.

1. רקע תיאורטי

* רקע על קבצי PE:

PE הוא פורמט של קבצי הרצה עבור קבצי כגון executable, object code ו dll. הפורמט מכיל מבנה היררכי ששומר את כל הנתונים הנצרכים עבור מערכת ההפעלה כדי להריץ את הקוד. הפורמט הוצג לראשונה בגירסת windows 3.1 ומאז נמצא בכל הגרסאות של windows. מבנה הPE הוא שכלול של מבנה שהיה קיים בUnix ה-Common object file, לכן יש המשתמשים במינוח- PE/COFF.

* מבנה הקובץ:

כעת נעבור על מבנה הקובץ ע"פ סדר הופעתו בקובץ. הקובץ מורכב מסדרה של מיכלים שכל אחד מכיל מידע נצרך עבור מערכת ההפעלה להריץ את הקוד.

* **Dos MZ Header:** חלק זה כולל תמיכה אחורה עבור הרצה במצב Dos. הוא מכיל גם את הDos Stub שתפקידו במידה והתוכנית מיועדת לרוץ בסביבת win32ביט והמשתמש הריץ במצב win16 ביט הוא ידפיס שגיאה: "This program cannot be run in MS-DOS mode.". 2 הדברים המרכזיים שהוא מכיל. e\_magic נמצא בתחילת כל קובץ PE והוא מכיל את האותיות "MZ" שזה ראשי תיבות של Mark Zbikowski שהיה מעורב ביצירת ה Dos executables. הוא נמצא שם כדי להודיע למחשב שהוא מריץ קובץ executable. החלק האחרון בdos header נקרא e\_lfanew והוא מכיל את ה- offset לחלק ה- PE file header.
* **PE File Header :** חלק זה מכיל בתוכו את הFile Header והOptional header. בתוכו הוא מכיל את המידע הנדרש בכדי להריץ את התוכנית. הוא מכיל נתונים כגון גדלים ומקומות של האיזורים השונים בקובץ שלו, עבור איזה מכונה הוא מיועד, מאפינים של הקובץ וכו'. כמו כן הוא מכיל את המקום בו הקוד מתחיל לרוץ (AddressOfEntryPoint). בנוסף יש בו את ה- data directory שהוא מראה לנו איפה נמצאים הקטעים שהתוכנית צריכה על מנת לרוץ.
* **Section Table**: לאחר מכן נמצא איזור הsection table. הוא מכיל מידע הנצרך עבור כל הsections. Section הינו איזור של מידע שהקובץ משתמשת בו לצורך ההרצה. הsection table מכיל את המידע בצורת מערך של איברים שמכילים את המידע על הsections. הם מסודרים ע"פ המקום שלהם בזכרון. כל section כזה מיוצג ע"י section header. הsection header מכיל את השם שלו, הכתובת הפיזית והוירטואלית, והגודל הפיזי והוירטואלי והמאפיינים שלו (כתיבה קריאה וכו').

לאחר מכן נמצא כל התוכן של הsections השונים.

איזורים שראוים לציון שיש בד"כ בקוד (אם כי לא מוכרח שיהיו במקומות שכתובים להלן, אלה רק שמות נפוצים).

.text section: איזור זה מיועד בד"כ לכתיבת הקוד. כאן יהיו כל הפקודות והקריאות לפיהן התוכנית רצה.

.data section: איזור זה מכיל מידע שהקוד צריך להשתמש בו כגון משתנים גלובאלים וסטטים כמו Strings שמאותחלים בזמן קומפילציה.

Section .rdata: מכיל פונקציות וimports של dll שהתוכנית צריכה בכדי לרוץ. חלק זה נקרא בד"כ גם הimport table. הimport table. הוא למעשה מערך שכל מבנה מכיל מידע על הdll שקובץ הPE מייבא ממנו פונקציות. הוא מכיל שמות dll ושמות פונקציות שהכרחים כשהתוכנית רצה לבקש אותם ע"פ מיקומם הוירטואלי במחשב.

לאחר שלמדנו את המידע הזה הנחוץ לנו על הPE ניתן לגשת לרעיון למימוש התוכנית.

1. PE injection-

ראינו שהoptional header בעצם מצביע למקום שממנו התוכנית מתחילה לרוץ. אם כן ניתן לשנות את הaddressOfEntryPoint למקום שאנחנו נחליט ולהריץ את הקוד שנמצא במקום הזה. אז איך נעשה את זה? נראה שתי דרכים בהם ניתן להכניס קוד לתוך קובץ Code cave, וAdd section.

* Code cave

Code cave הוא בעצם שיטה המבצעת סריקה על הקובץ PE שרוצים להדביק ומחפשת מקום שאין בו מידע בשימוש. אם הוא מוצא מקום כזה בגדול של הקובץ שאני רוצים להדביק אז הוא מכניס לשם את הקוד, שומר את ה entry pointהמקורי ומשנה את addressOfEntryPoint שיתחיל מהמקום של הcode cave. לאחר סיום הקוד שהוכנס הוא יבצע קפיצה לentry point המקורי וימשיך את הקוד כרגיל.

היתרון בצורה הזאת היא שלא שינינו שום דבר בקוד מלבד הentry point, כמו כן לא שינינו את גודל הקובץ. החסרון לעומת זאת הוא שאנחנו מוגבלים מבחינת מקום. אם אין code cave בגודל מתאים לא ניתן יהיה לבצע את מה שרצינו בצורה הזאת. ניתן אולי לחפש כמה code cave ולקפוץ מאחד לשני אבל זו כבר עבודה מוסבכת יותר. ניתן גם להרחיב גודל של section אבל זה כבר ידרוש לשנות שדות.

* Add section

במקום לחפש מקום פנוי ניתן להוסיף עוד sections לקובץ. נוכל להחליט על איזה גודל שנרצה ובכך לא נוגבל מבחינת מקום. נצטרך רק לדעת איזה שדות צריך לשנות על מנת שהקוד ימשיך לרוץ באופן תקין. השדות הרלונטים שצריך לשנות הם: PointerToRawData, VirtualAddress, SizeOfRawData, VirtualSize, Characteristics, NumberOfSections. כמו כן צריך לוודא שהsection alignment הוא תקין. לאחר מכן אפשר להכניס לשם את הקוד.

אך אי אפשר רק להכניס קוד ולהצביע עליו בentry point. מכיון שהקוד שהוכנס היה חיצוני לתוכנית המודבקת - אין לו מידע לגבי משתנים קריטים שלו לשימוש. למשל אי אפשר להשתמש בstring כלשהו מכיון שstring נמצא באיזור של המידע ולא באיזור של הקוד. אנו צריכים לקשר בין הstrings שנשתמש בהם והמקום שהם הם נמצאים. מאותה סיבה לא ניתן להשתמש בפונקציות מכיון שאין לנו את הכתובת שלהם. לכן השתמשנו בשיטה שמייבאת את כל המידע לתוך הsection החדש והוא גם מקשר בין הקוד שרץ לבין המידע שאנו צריכים להשתמש בו.

אנו נבנה מערך שמורכב מ2 חלקים. מערך הראשון – newEPdata- הוא חלק המידע שהוא מכיל את כל הstrings והכתובות שאנו נצטרך בתוכנית. הstrings כולל שמות הdll והdll API functions והמילים שאיתם נשתמש בתוכנית. בתוכו גם יהיה האסמבלי של הקוד המקשר בין הstrings לקוד המבנה החלונות ויציג אותו. המערך השני –newEPcode- יהיה המידע הנצרך עבור הרצת הקוד (הוא גם ידע את המידע שיש במערך הראשון), והוא יגיב לדברים שהתרחשו בחלון שרץ (ז"א יבדוק מה לעשות עבור כפתור שנלחץ וכו').

על מנת שנוכל להריץ פונקציות API של מערכת ההפעלה אנו צריכים שיהיה לנו את הכתובות המדויקות של הפונקציות במערכת ההפעלה על מנת להשתמש בהם. לכן בתחילת טעינת הקובץ הדבר הראשון שנבדוק זה שהוא מכיל את LoadLibrary ו getProccAddress. בלי זה לא ניתן יהיה לממש את הקוד. אנו נחפש בkernel32.dll האם יש בו את הפונקציות האלה. תפקיד LoadLibrary הוא לטעון קובץ dll. אנו נחפש בתוכו את הפונקציות שאנו רוצים להשתמש בהם. נמצא אותם ע"י שימוש בgetProccAddress. הוא מביא לנו את הכתובת שבה נמצאת הפונקציה של הDll שאנו רוצים.

נפרט את מבנה הקובץ בצורה יותר מפורטת:

* בנית אזור חדש

מערך newEPdata

|  |  |  |
| --- | --- | --- |
| **Address- offset** | **Value** | **Data** |
| 0-31 | מידע בסיסי לצורך הכנסת הנתונים | OEP, LoadLibrary, GetproccAddress, Epsize, password |
| 32-199 | כל המילים שמשתמשים בהם | Strings |
| 200-249 | שמות הDll שנתשמש בהם | Dll imports |
| 250-599 | שמות הפונקציות | Function names |
| 600-1024 | הכתובות המדויקות של הפונקציות | Function address |

לאחר שיש לנו את זה אפשר להתחיל להעתיק את המידע שאנו רוצים להשתמש. מבנה המידע יראה בצורה כזאת:

ה32 ביט הראשונים מכילים את המידע בסיסי לצורך המשך התוכנית. הוא יכיל את הכתובת המקורית של הקובץ המודבק, הכתובות של LoadLibrary וgetProccAddress. כמו כן יש לו את הגודל של של הקוד שמריץ את הטענת הנתונים, ואת הסיסמא שהמשתמש מכניס לקובץ.

מoffset 32 שומרים את כל המילים שבהם התוכנית משתמשת כגון מילים שיודפסו למסך וסוגי כפתורים שהם נרצה להשתמש.

בOffset 200 יש את שמות כל הdll שאותם צריכים בשביל להשתמש בפונקציות.

החל ממקום 250 יש את שמות api function שאיתם אנחנו משתמשים בתוכנית.

מקום 600 מכיל את הכתובות האמיתיים של הfunction names שלהם משתמשים בתוכנית.

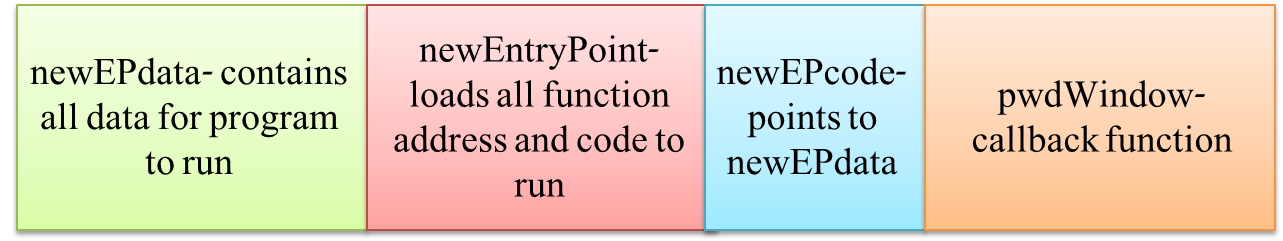
כמו כן הוא ישמור את הכתובת של המערך ההראשון על ויכניס למערך השני בכדי שיוכל להשתמש במידע של המערך הראשון.

כעת נכתוב את הקוד שירוץ. בזמן ריצה הפונקציה newEntryPoint- יחפש את המידע ששמרנו במערך לעיל ויטעין את כל הכתובות הרלונטיים. ע"י GetProcAddress נטען למקום 600 ואילך את הכתובות של הפונקציות api. כשיצטרך string מסוים ילך לoffset 32 + הoffset של המקום בו נמצא הstring עצמו. עשינו מיפוי של המערך על מנת לדעת איפה כל string נמצא ולהצביע לoffset הנכון (ראה נספחים). כמו כן נכתוב את הcallback function. הפונקציה הזאת תרוץ כאשר נלחץ על אחד החלונות, והוא ישתמש במידע שהטענו במערך הקודם ויריץ את הקוד.

1. אופן פעולת ההדבקה:

בתחילה התוכנית פותחת את הקובץ לקריאה ובודקת שהוא קובץ PE תקין. היא טוענת את כל המידע שלה לbuffer שאיתו נוכל לעבוד. כעת היא בודקת בimport table שלו אם נמצא בkernel32.dll LoadLibrary וGetProcAddress. אם הפונקציות האלה לא נמצאות התוכנית תדפיס שזה שלא נמצא ותסגור את התוכנית. אם הפונקציות נמצאות התוכנית תטעין את כל המידע שאנו צריכים למערך כמו שהסברנו לעיל.

לאחר מכן התוכנית לוקחת את הקוד שלנו שהולך לרוץ ומוסיפה אותו לסוף של המערך מידע. אחר כך היא מעתיקה את המידע של המערך השני (שהוא מכיל מצביע למערך הראשון) לסוף של הקוד הקודם ולבסוף שומרת את הקוד של הcallback function. למעשה הsection החדש יראה כך:



אחרי שביצענו את ההוראות דלעיל מבצעים הוספת section חדש בשם .proj ומשנים את השדות שהזכרנו לעיל, בנוסף צריך לשנות את הAddressOfEntryPoint ואת הsizeOfImage ולבסוף שומרים את המבנה החדש.

על מנת שהקוד יופיע אחד אחרי השני בצורה לעיל היה צריך לשנות את הגדרת הקומפיילר למהירות מקסימלית. בדרך כלל על פי שיקולים שלו, הקומפיילר יכול לשנות מיקום של התוכנית על מנת לייעל את הקוד. היה צריך לשנות את מהירות הקומפילציה למהירות מקסימלית בכדי שזה יקרה.

* Cleaner

מכיון שאנו יודעים איפה נמצא המידע, הcleaner רק צריך למשוך את המידע של הOriginal entry point. זה ממש קל לביצוע מאחר ששמרנו אותו בnewEPdata ב4 בתים הראשונים. התוכנית צריכה להצביע רק על הsection האחרון מכיון שהוספנו אותו בסוף, לשמור את הentry point, להקטין את הגודל ע"י שנצביע לתחילת הsection האחרון פחות אחד ולהגדיר שהוא יהיה סוף הקובץ. כמו כן צריך לשנות את sizeOfImage וnumberOfSections. (יש לציין שגודל הקובץ לא קטן אלא רק גודל קובץ הPE.)

1. בעיות והתמודדויות:

במהלך הפרויקט שלנו נתקלנו בהרבה קשיים ובעיות שהקשו עלינו בכתיבתו, להלן פירוט העיקרים שבהם.

1. לימוד מבנה הPE - מבנה ה PE הוא מבנה היררכי מורכב ומדויק מאוד שכל שינוי קטן שנעשה באחד הערכים יכול לשבש לחלוטין את הרצה התוכנית. היינו צריכים להשקיע זמן רב בלמידה איך הוא בנוי ומה תפקידו של כל חלק. היינו צריכים ללמוד מה צריך לשנות בשביל להוסיף section ואיך הקוד מקבל את הנתונים שלו. כמו כן בזמן כתיבת הקוד היה צורך בלימוד איך ניגשים לחלקי המבנה של הPE. היתרון בכתיבת הקוד בשפת C/C++ ב- Visual Studio הוא שיש לו כבר קובץ winnt.h ובו נמצא כבר המבנה של של PE הקושי היה שהכל עבד בצורה של מצביעים דבר שהקשה עלינו בזמן הכתיבה.
2. יש מידע נפוץ מאוד ברחבי הרשת שעוסק על הדבקת קובץ, אך הבעיה העיקרית היא שרובם דורשים יכולות גבוהות מאוד באסמבלי ורובם לא מתארים איך לעשות את התוכנית שאנחנו רצינו לעשות. לרוב הם רק מביאים מקרה פשוט של messegeBox ולא יותר מזה. השיטה הזאת שמצאנו היא טובה בכך שנותנת אפשרות לכל אחד ללמוד וליישם הדבקת תוכנית, ובעצם הגבול לתוכנית שתרוץ תלויה רק ביכולת של המתכנת בשימוש בפונקציות API. כמו כן מכיוון שהחומר הוא נושא רגיש לגבי אבטחת תוכנה הרבה אנשים נמנעים מלשתף קוד באינטרנט דבר שהקשה עלינו במציאת פתרון קשיים.
3. תובנות ומסקנות:

למעשה מצאנו דרך להדבקת קוד כל שיעבוד בצורה אוטונומית שבו הקשר היחיד שיש לו עם הקובץ המודבק הם פונקציות GetProcAddress ו LoadLibrary. למדנו על מבנה הPE בצורה מעמיקה וצברנו כלי מאוד חשוב בתכנות לגבי מבנה תוכניות הרצה. כמו כן למדנו על דיוק בפרטים הקטנים כמה שbyte אחד יכול לשבור או לתקן קובץ שלם, ולמדנו הרבה דרכים בהם מתכנתים משתמשים בכלים האלה לטוב ולרע.

1. הרצת הקוד:

הוראות הרצת התוכנית:

Infector:

C:\...\project\executable>Infector.exe <file path> <password>

Cleaner:

C:\...\project\executable>Infector.exe <file path>

1. ביבליוגרפיה:

# 

1. Corporation, Microsoft. (n.d.). *Microsoft Portable Executable and Common Object File Format Specification*. Retrieved from Microsoft: <https://www.microsoft.com/en-us/download/details.aspx?id=19509>
2. Pietrek, M. (1994, March ). Peering Inside the PE: A Tour of the Win32 Portable Executable File Format. Retrieved from msdn: <https://msdn.microsoft.com/en-us/library/ms809762.aspx>
3. Pereira, C. S. (2008, March 16). Portable Executable (P.E.) Code Injection: Injecting an Entire C Compiled Application. Retrieved from Code Project: <https://www.codeproject.com/Articles/24417/Portable-Executable-P-E-Code-Injection-Injecting-a>
4. The Beginners Guide to Codecaves. (n.d.). Retrieved from Code project: <https://www.codeproject.com/articles/20240/the-beginners-guide-to-codecaves>
5. PE File Code Injection. (2014, march 4). Retrieved from Weikipedia: <https://wshen0123.wordpress.com/2014/03/04/pe-file-code-injection/>
6. Randy Kath Microsoft Developer Network Technology Group. (n.d.). The Portable Executable File Format from Top to Bottom. Retrieved from <http://www.csn.ul.ie/~caolan/publink/winresdump/winresdump/doc/pefile2.html>
7. (2005, December). Retrieved from ntcore: <http://www.ntcore.com/files/inject2exe.htm#Figure3>
8. n.d.). Retrieved from rohitab: <http://www.rohitab.com/discuss/topic/40187-q-how-to-add-section-to-pe-exe/>
9. נספחים

מבנה היררכי של המידע של הקוד:

|  |  |  |  |
| --- | --- | --- | --- |
| newEPdata | Value | Offset | Size |
| OriginalEntryPoint | Address of OEP | **0** |  |
| LoadLibraryAddress | LoadLibrary | 4 |  |
| GetProcAddress | GetProcAddress | 8 |  |
| EPSize | Size of new entry point = 0 | 12 |  |
| pass | User input password | 16 | 12 |
|  |  |  |  |
| Strings: |  | **32** |  |
| "licence" |  | 32 | offset = 0 bytes |
| "Please purchase software" |  | 41 | offset = 9 bytes |
| "BUTTON" |  | 67 | offset = 35 bytes |
| "EDIT" |  | 71 | offset = 41 bytes |
| "OK" |  | 78 | offset = 46 bytes |
| "Cancel" |  | 81 | offset = 49 bytes |
| "Sorry! Wrong password." |  | 88 | offset = 56 bytes |
| Password |  | 111 | offset = 79 bytes (not used in codeinject.exe) only in template |
| "Insert Password" |  | 120 | offset = 88 |
|  |  |  |  |
| DLL imports: |  | **200** |  |
| User32.dll |  | 200 |  |
| Kernel32.dll |  | 212 |  |
|  |  |  |  |
| Function names: |  | **250** |  |
| RegisterClassExA |  | 250 | 00 / ref offset = 0 |
| CreateWindowExA |  | 267 | 17 / ref offset = 4 |
| SetWindowTextA |  | 283 | 33 / ref offset = 8 |
| ShowWindow |  | 298 | 48 / ref offset = 12 |
| UpdateWindow |  | 309 | 59 / ref offset = 16 |
| SetFocus |  | 322 | 72 / ref offset = 20 |
| GetMessageA |  | 331 | 81 / ref offset = 24 |
| TranslateMessage |  | 343 | 93 / ref offset = 28 |
| DispatchMessageA |  | 360 | 110 / ref offset = 32 |
| GetWindowTextA |  | 377 | 127 / ref offset = 36 |
| MessageBoxA |  | 392 | 142 / ref offset = 40 |
| PostQuitMessage |  | 404 | 154 / ref offset = 44 |
| DefWindowProcA |  | 420 | 170 / ref offset = 48 |
| GetSystemMetrics |  | 435 | 185 / ref offset = 52 |
| GetDlgItem |  | 452 | 202 / ref offset = 56 |
| DestroyWindow |  | 463 | 212 / ref offset = 60 |
| GAP |  | 477 | Size of 5 |
| ExitProcess |  | 482 | 213 / ref offset = 60 |
|  |  |  |  |
|  |  |  |  |
| Function address: | Pointer to real add | 600 | Finds real address of dll API function calls |
| User32-> RegisterClassExA |  | 600 |  |
| CreateWindowExA |  | 604 |  |
| SetWindowTextA |  | 608 |  |
| ShowWindow |  | 612 |  |
| UpdateWindow |  | 616 |  |
| SetFocus |  | 620 |  |
| GetMessageA |  | 624 |  |
| TranslateMessage |  | 628 |  |
| DispatchMessageA |  | 632 |  |
| GetWindowTextA |  | 636 |  |
| MessageBoxA |  | 640 |  |
| PostQuitMessage |  | 644 |  |
| DefWindowProcA |  | 648 |  |
| GetSystemMetrics |  | 652 |  |
| GetDlgItem |  | 656 |  |
| DestroyWindow |  | 660 |  |
| GAP |  |  |  |
| Kernel32-> ExitProcess |  | 664 |  |
| <E> |  | 1021 |  |