Overview of Virus.Win32.Virut.ce malware pattern

In the following article, TipsMake.com will introduce you to some polymorphic viruses - here is Virus.Win32.Virut and its ce variants

In the following article, TipsMake.com will introduce you to some polymorphic viruses - here is Virus.Win32.Virut and its ce variants .

Why is it called Virut.ce?

Virut.ce is currently one of the most widespread forms of malware on personal computers. It infects all executable files (tail * .exe in the Windows environment) using the most sophisticated technologies to evade security software. The way they spread is mainly through polymorphic server models, which are significantly different from the five years ago. Another part is because the use and construction of simulation data also increased sharply. On the other hand, the 'technology' applied to Virut.ce templates reflects fairly precisely the sophistication and sophistication of the methods used to create malware. Tools to combat simulation and data analysis are widely applied, such as 'counting' formulas of formula functions that are obtained using multi-layer, and rdtsc instructions. function series 'call' GetTickCount API or the process of 'calling' other fake API functions .

Viral - is one of the most rapidly and rapidly spreading forms of virus, with an average of 1 new variant within 1 week. This once again reaffirms the ability of the people behind and directly creating virus types - getting closer to accessing and controlling the database of antivirus programs, so they have Warning and hiding actions can be easily performed whenever there is an updated virus identification version. As soon as the update and identification process is applied to antivirus programs, the 'author' of the virus samples will change the behavior and the ability to disguise them to continue to cause difficulties for experts. security. But it's interesting that these malicious programs are always guaranteed and make sure that the latest versions are always downloaded to the victim's computer through browser vulnerabilities in HTML format.

In the next paragraph, the article will describe in more detail the ways used to infect any file system, the processes of formation and development corresponding to each component of the virus tested, listed since they appeared so far. All data is collected and used technology from Kaspersky Security Network (KSN), owned by Kaspersky Lab.

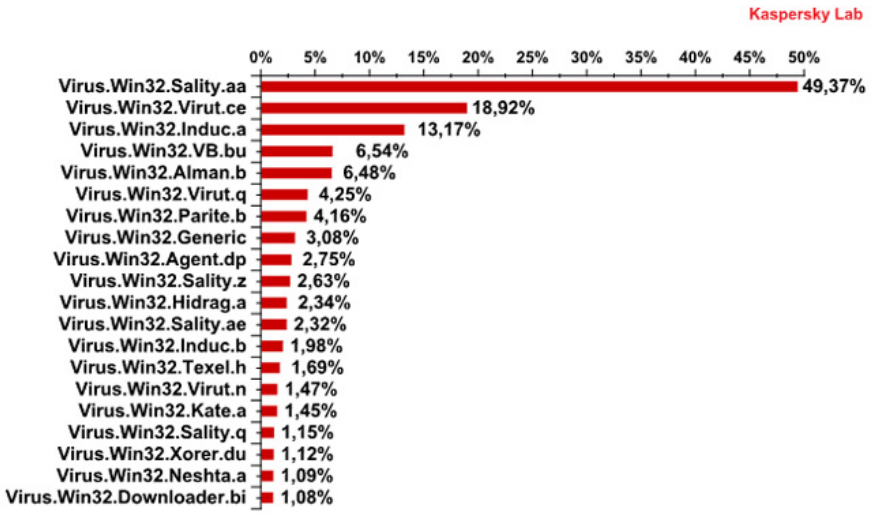
Brief description of the statistics and the extent of spread

The first variant of the virus called Virut.a has reappeared in the middle of 2006. Since that time, they have made steady strides in the development process, there have been variations. Virut.q appears several times around September 2007.

At that time, the appearance of Virut.q was quite common, but today is very rare. Simply because hackers have stopped supporting the virus since the second half of 2008, but only a short time later - specifically the first week

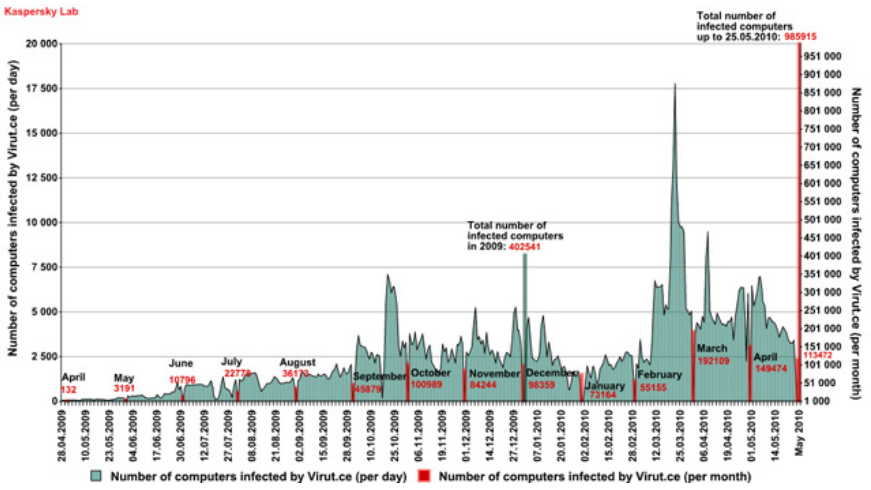
of February 2009, a new variant has appeared with the name called Virut.ce. Experts say that hackers used the previous break to study, perfect infection techniques, ability to hide themselves and new encryption algorithms. From here, the term is used in the rest of the article such as Virut, virus . to refer to Virus.Win32.Virut.ce.

Up to this point, the Virut.ce line is the second most popular variant only after versions of Virus.Win32. *. * Detected on the total number of infected computers:



Statistics of Kaspersky Lab about 20 virus samples were most detected from January 2009 to May 2010

When looking at the data below, people can easily imagine how powerful Virut.ce's progress and spread are over time:



The number of computers infected by Virut.ce during the same time from the month
May 2009 to May 2010

The mode of infection of this virus pattern is mainly via executable files * .exe and HTML, or software crack programs, but we still often call keygen and cracked files already available. More specifically, they often hide in RAR / SFX crack files with names like **codename_panzers_cold_war_key.exe** or **advanced_archive_password_recovery_4.53_key.exe** .

The main functions of viruses

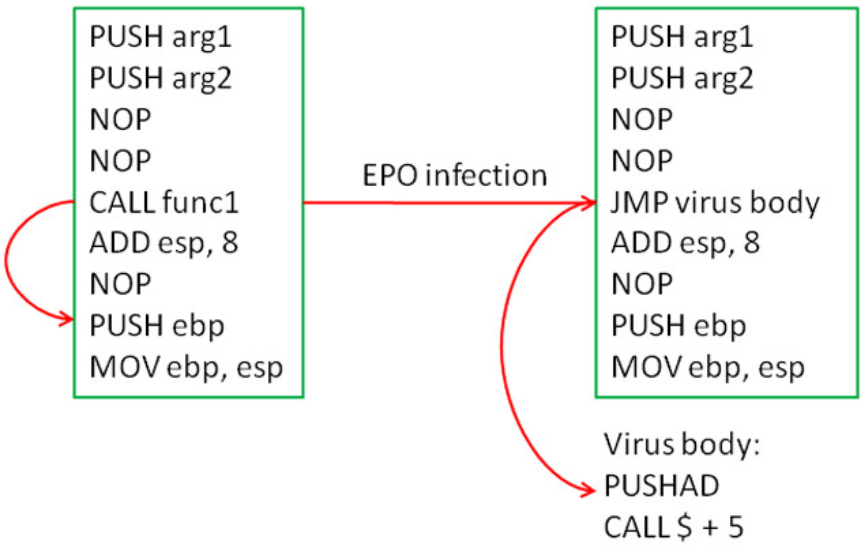
Next, we will go to the most important function part - the payload process (understandably the implementation of the acts of stealing data related to financial purposes from infected users) of virus. This is a very recognizable part, because it is created and used by any virus pattern without exception. Very effectively, it will normally implant a backdoor program first to invade the active area of ??the explorer.exe process (such as services.exe or iexplore.exe), then make connections to the site. Only irc.zief.pl and proxim.ircgalaxy.pl via IRC protocol and wait for the commands and feedback signals. These procedures can easily be detected by popular security programs such as nod32, rising, f-secure .:

The image represents the process of deciphering Virut.ce's static body, including the name of the virus-locked application

In fact, infectious virus samples into the file * .htm, * .php and * .asp are stored right inside the corner of the computer. To do this, they manually added the following lines of code to the basic code: . This command will automatically download the latest version of the virus to your computer via a PDF file vulnerability. These lines will vary according to the corresponding versions of the ce variant. For example, the letter u may be replaced by u, may not affect the browser but will protect the identification process when they work.

General discussion and modes of infection

In essence, Virut.ce uses EPO technology or overwrites entry points to carry out infection, and one or two decoding tools are used in the process. Entry Point Obscuring (EPO) technology works based on the ability to resist identification. Usually, this process is performed by replacing random instructions throughout the main source code of the program and the associated parameters. Below is an example of the process of replacing instructions and working addresses:



The phrase rewriting the entry point - translating is the process of overwriting entry points, implying modifying the file's PE header data, especially overwriting the AddressOfEntryPoint data field in the structure. IMAGE_NT_HEADERS32. Therefore, the executable file will directly start and activate the main components of the virus. As discussed above, virus patterns use only one or two major decryption tools during the infection process - can be temporarily called Init and Main. The Main decoding tool is defined in each file affected by Virut.ce, while the Init section only works occasionally.

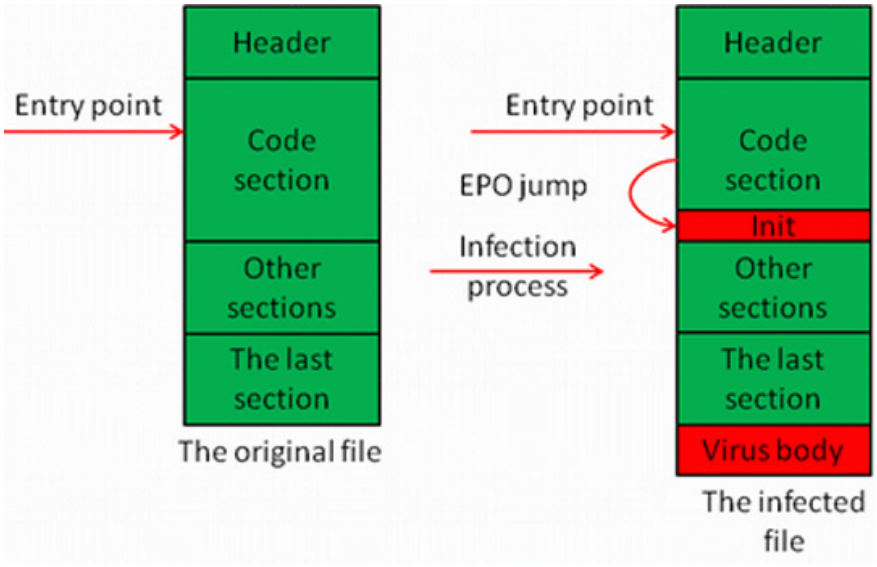
The main purpose of the Init section is to identify and decode the first data layer in the body of the virus according to the process to hand over the autonomous mechanism and activate it. However, the remaining components remain the same even when the decoding process occurs and ends. The Init decryption tool is a small piece of code from 0x100 to 0x900 bytes, which contains a lot of misguided instructions to avoid the identity of security programs. Summarize the compilation process, decoding this can include the following four main steps:

- Initializing and recording the decoded sections of the sections to the register management program
- Perform the operators logically to continue coding the sections with constant key constants
- Increase / decrease the number of pointers pointing to encrypted sections
- Turn back to step 2 until all data is decoded

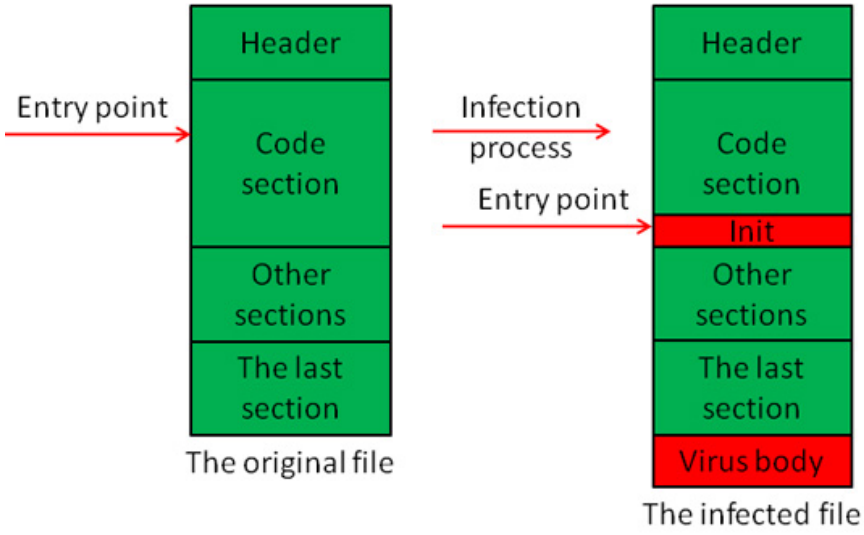
Typically, the main body of the virus will be in the range of 0x4000 to 0x6000 bytes, and defined at the end of the last section, clearly defined by access, execution, and read / write properties. of data.

Or in another way, we can imagine the above process according to the following diagram:

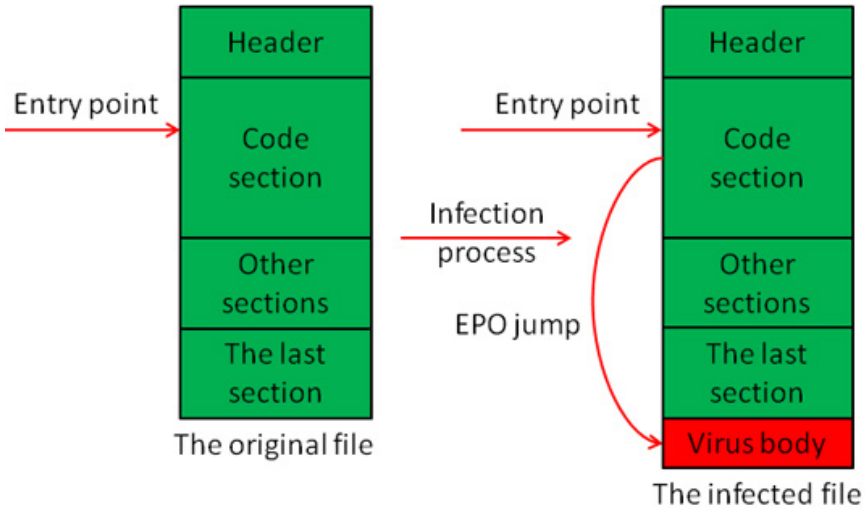
- Init Decryptor and EPO process:



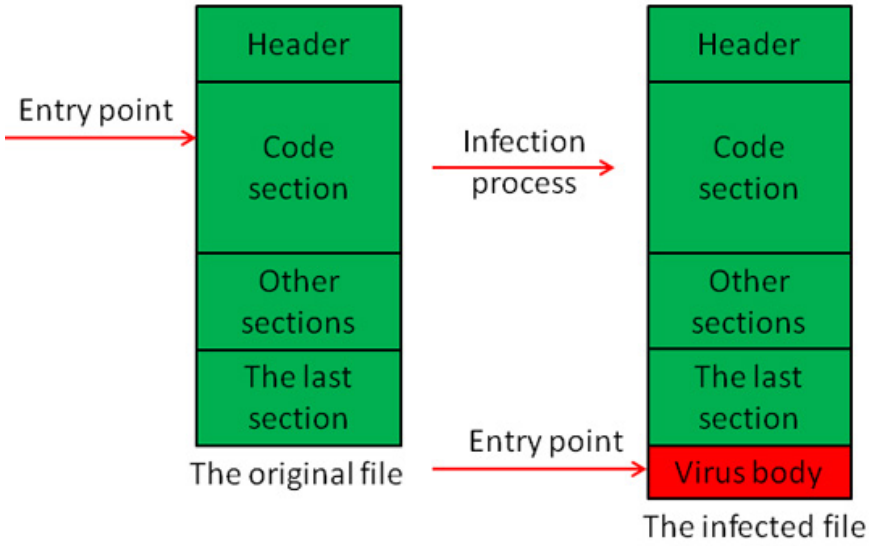
- Init Decryptor and edit EP:



- Particularly EPO process:

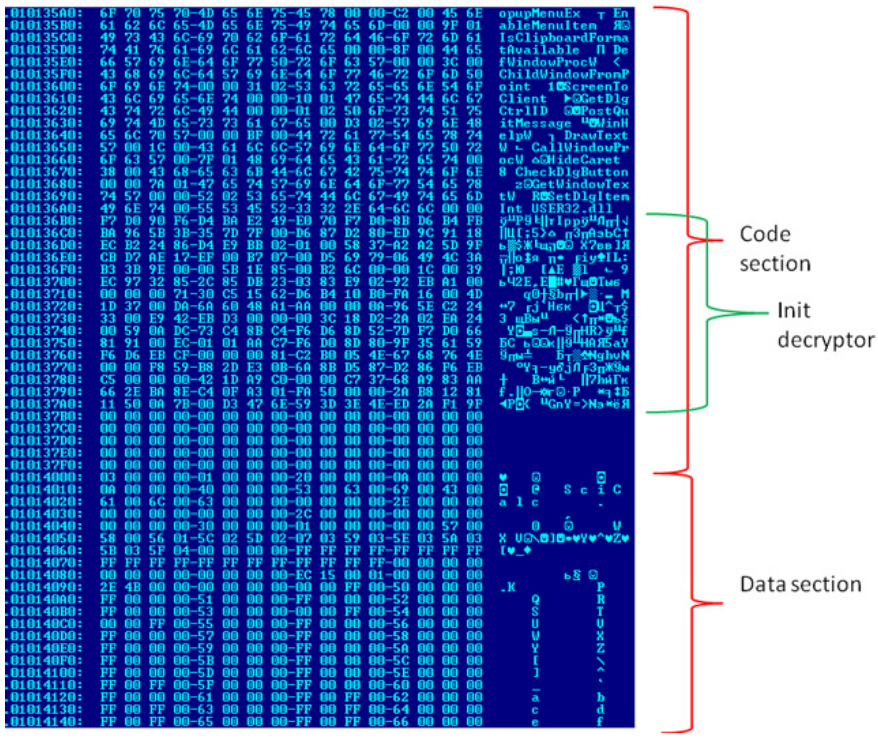


- The process of rewriting entry points:

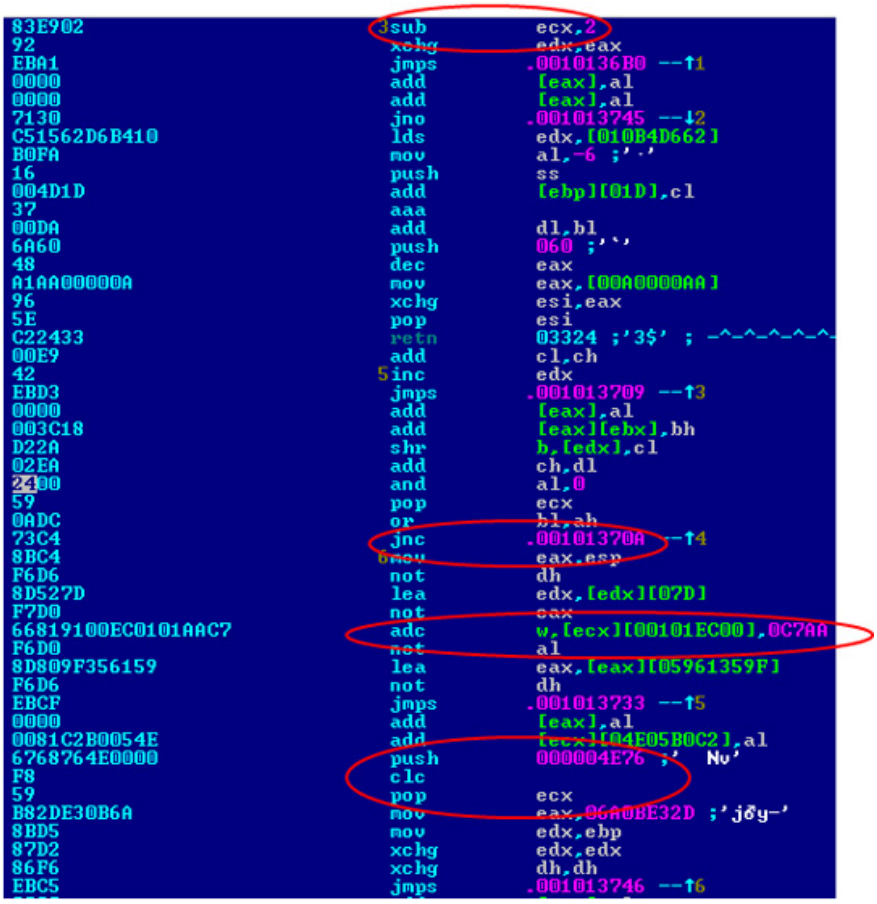


The main decoding piece of the Virut.ce virus body

Before going into the main discussion of the virus payload method, let's look at the Init decrypting tool in an infected file:



Part of the file has been infected by Virus.Win32.Virut.ce with Init decryptor



Disassembled code of Init decryptor
The first image shows the infected code sections of the calc.exe file. The outer part of the code sections is marked, and the Init decryptor section is also marked. The image below shows the disassembled code of the Init decoding tool. The four modes mentioned above are circled in a red oval.

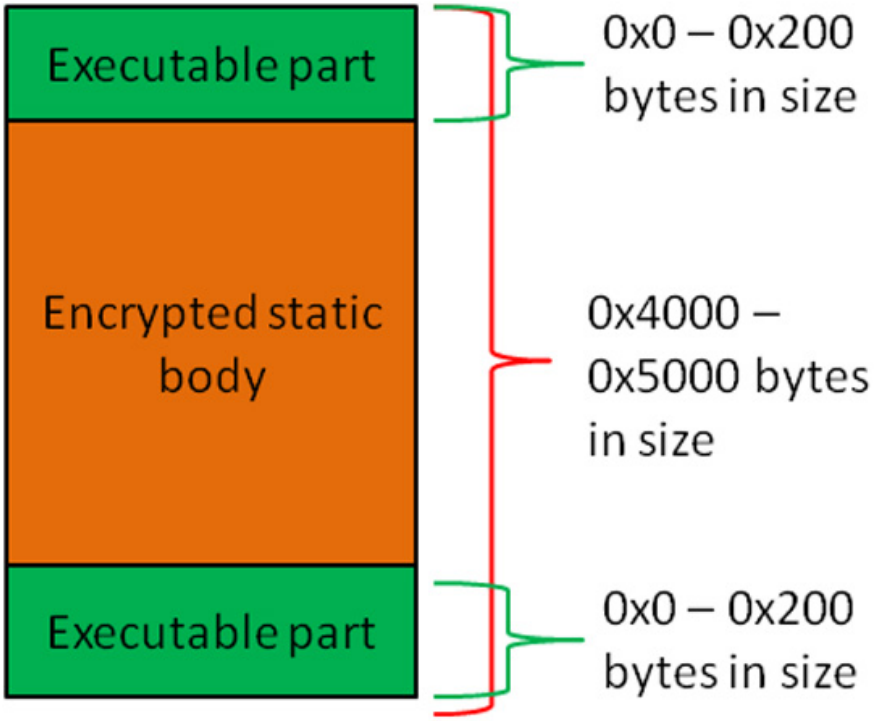
In this example, the ECX registry manager is filled with continuous push / pop actions and decoded by adc. However, these processes are not always the same. Virut.ce has grown very rapidly over the past year, as evidenced by how easy it is to integrate Init decoding technology. If the information of the body virus is recorded in the registry once modified (mov reg, dword converted to push dword; pop reg), the procedures for decoding also change more than 1 time (arranged in order) :

- ADD / SUB [mem], dword;
- ROL / ROR [mem], bytes;
- ADC / SBB [mem], bytes;
- ADD / SUB [mem], bytes;
- ADD / SUB [mem], word;
- ADC / SBB [mem], word;
- ADC / SBB [mem], dword;

Restore the original code

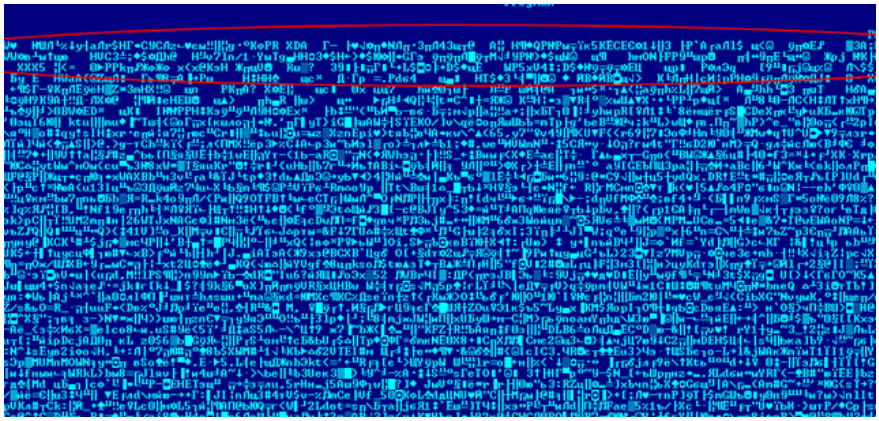
Theoretically, the entire source code of the main body part can be divided into three groups, according to the following main tasks: restoration process from function / entry points, body decoding process in static form, and the performance of the payload process.

Before conducting a thorough review of each component, take a look at the entire body part of the virus and the relevant sections:



The overall structure of Virut.ce's main body

As shown in the figure, we can see that the main body part is concatenated to the end of the last sectin code and forms two parts: the static body part decoder and the code execution unit. The executable portion of the executable contains code to enforce modes against simulation, restore the original entry point parameter, and the entire static body decoding mechanism. These sections are scattered across the entire body or can be fixed at the beginning or end, or divided into 2. It is also fortunate that the executable part is covered quite carefully. This will make detection or identification much more complicated:



The code contains the main component of Virus.Win32.Virut.ce

The image above describes the code that contains the main component of any file infected by *Virus.Win32.Virut.ce* . The area that is zoned with the red oval is the part that performs the task, or it is easily

recognized because there are many empty byte characters. And in this section, the virus is not in a hurry to use the decryption mechanism during the infection process, all sections look the same and are encrypted together.

Next, look at the dedicated data blocks to restore the original part of the data. This logical process can be divided according to the following steps:

- Perform CALL procedure (fixed addresses with small deviations)
- Restore the original content of the data with the register manager
- Add specific addresses to point to kernel32.dll and EBX register
- Calculate the number of pointers to the addresses of the CALL function calls in step 1
- Continue to perform the operator on the address called from step 4

Note that the virus uses EPO technology only when it recognizes that API-function functions are being called from kernel32.dll. The procedure to call this function can be identified through 'calls' from the operation code 0x15FF or 0xE8, with subsequent JMP instructions (0x25FF). If any function is identified as being replaced by JMP (0xE9), it only points to chart 1 (above image), then the function's address is replaced from kernel32.dll in the EBX register manager. If the entry points have just been modified, the values ??at [ESP + 24] are replaced in the EBX register - this is the address of the specified application returning to the kernel32.dll kernel. Next, the values ??stored in the register will be used to record addresses when the system outputs the corresponding actions of the DLL. If EPO technology is applied, the value at [ESP + 20] will store the information of the called functions corresponding to the API-function, but also store the entry point address. original.

If the obfuscation process is excluded, the simplest way to restore entry points and other functions will be as follows (in the case of GetModuleHandleA has been replaced):

CALL $ + 5
PUSHAD
MOV EBX, [GetModuleHandleA]
XOR [ESP + 20h], Key

This code perfectly matches the general logic of the entire data block. Next, let's see how they change over different stages.

The first stage - CALL, does not change much through the stages. In essence, it looks like CALL $ + 5, then CALL $ + 6 (7,8), and continues to be CALL $ + 0xFFFFFFFx . with the same mechanism as the callback function. It can be assumed that this process is not very important, and for certain cases, the removal mechanism will be applied, but the truth is not so. In fact, this address is used to store the entry point / original functions as well as when pointing to the address of the decryption keys and the start of the static code.

Next is step 3, often changing and editing more than step 1, if we look more closely, we can continue to divide into different ways:

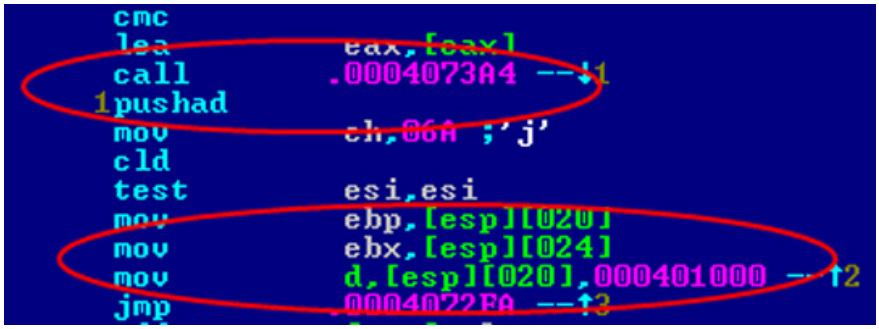
- MOV EBX, [ApiFunc] / MOV EBX, [ESP + 24h];
- PUSH [ApiFunc] / [ESP + 24h]; POP EBX;
- SUB ESP, xxh; PUSH [ESP + 24h + xx]; POP EBX;
- LEA EBX, [ESP + xxh]; MOV EBX, [EBX + 24h - xx];
- ADD ESP, 28h; XCHG [ESP - 4], EBX;

The above example list can help us better visualize how this period develops over time. In addition, there are intermediate effects of ESP and EBX register managers.

After the PUSHAD function is called, the ESP register - specified to point to the stack - will be reduced by the value 0x20, then ESP + 20h will switch to storing the value provided by the CALL function called finally out. A reasonable operation operator will be applied to this value and require additional numbers to be collected.

Here are some sequences that operate in order to describe the actions above:

- XOR / AND / OR / ADD / SUB [ESP + 20h], const;
- MOV [ESP + 20h], const;
- LEA EBP, [ESP + x]; MOV / OR / ADD / SUB / XOR EBP, const; XCHG [EBX + 20h - x], EBP;
- MOV EBX, ESP; PUSH const; POP [EBX + 20h];
- PUSH const; POP [ESP + 20h].



Screenshot of the file infected by Virus.Win32.Virut.ce, with the code taking over the recovery function to the original entry points marked with a red oval

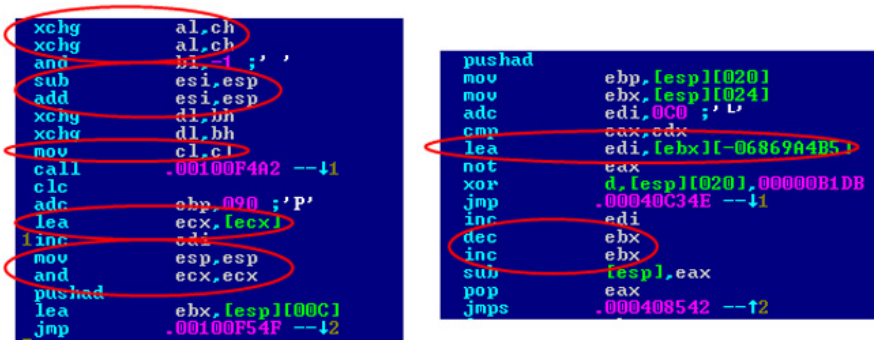
To make a clear distinction, all of the above code examples do not include the obfuscation process. However, that phase is used in all file sections that have been added by the virus, including the Init decoder and the entire executable code in the main body section. And this process completely prevents the identification of viruses from security programs, by changing the overall appearance of the data code without affecting the general operation. Here are some specific examples in which they are used to hide the identification process without affecting the functional operations:

- XCHG reg1, reg2; XCHG reg2, reg1; (used together)
- SUB reg1, reg2; ADD reg1, reg2; (used together)
- MOV reg, reg; OR reg, reg; AND reg, reg; XCHG reg, reg; LEA reg, [REG];
- CLD, CLC, STC, CMC, etc.

In it, 'reg1' and 'reg2' represent different registers, while 'reg' refers to the same register process in the same single expression.

Logical operators are associated with a custom second-class operator.

There are also ADC reg, const; SBB reg, const; XOR reg, const .



The components of the obfuscation process are marked with a red oval

The figure at the left clearly shows that junk-style instructions take up 70-80% of the entire file's data code.

Decode the main body part

The department responsible for executing the decrypted code will boot after the virus completes its initial operations such as restoring the original code, initializing the object name and storing the addresses of the The corresponding function is used directly from the system library DLLs and anti-cycle.

If the Main decoding process is considered to be a part or a separate separator, the entire code generated by this process is completely meaningless, such as instructing RETN to be called for management and control. Random position change. Before the official decoding process takes place, RETN (0C3h) will be replaced by the CALL function (0E8h). We can imagine this process as follows:

ADD / SUB / XOR [EBP + xx], bytereg

Accordingly, EBP will be pointed to the address of the CALL function, and bytereg is only one of the registered byte values.

Therefore, we can assume that the actual start cycle after the RETN decoding process will be changed to CALL. Sequencing is the obfuscated process - the rest of the body of the virus. Not only do you use a large number of algorithms, and many of these are much more complex than the rest with the Init decoder. Usually, there will be between 2 and 6 algorithms used to combine. And in these algorithms, the EDX registry manager contains the decryption key, and with EAX contains the entire virtual address where the start of the static body is. Applications that manage register functions containing instructions for the corresponding function may look like the following:

MOVZX / MOV dx / edx, [ebp + const] LEA eax, [ebp + const]

The algorithms are mainly used as in the example below:

ROL DX, 4
XOR [EAX], DL
IMUL EDX, EDX, 13h

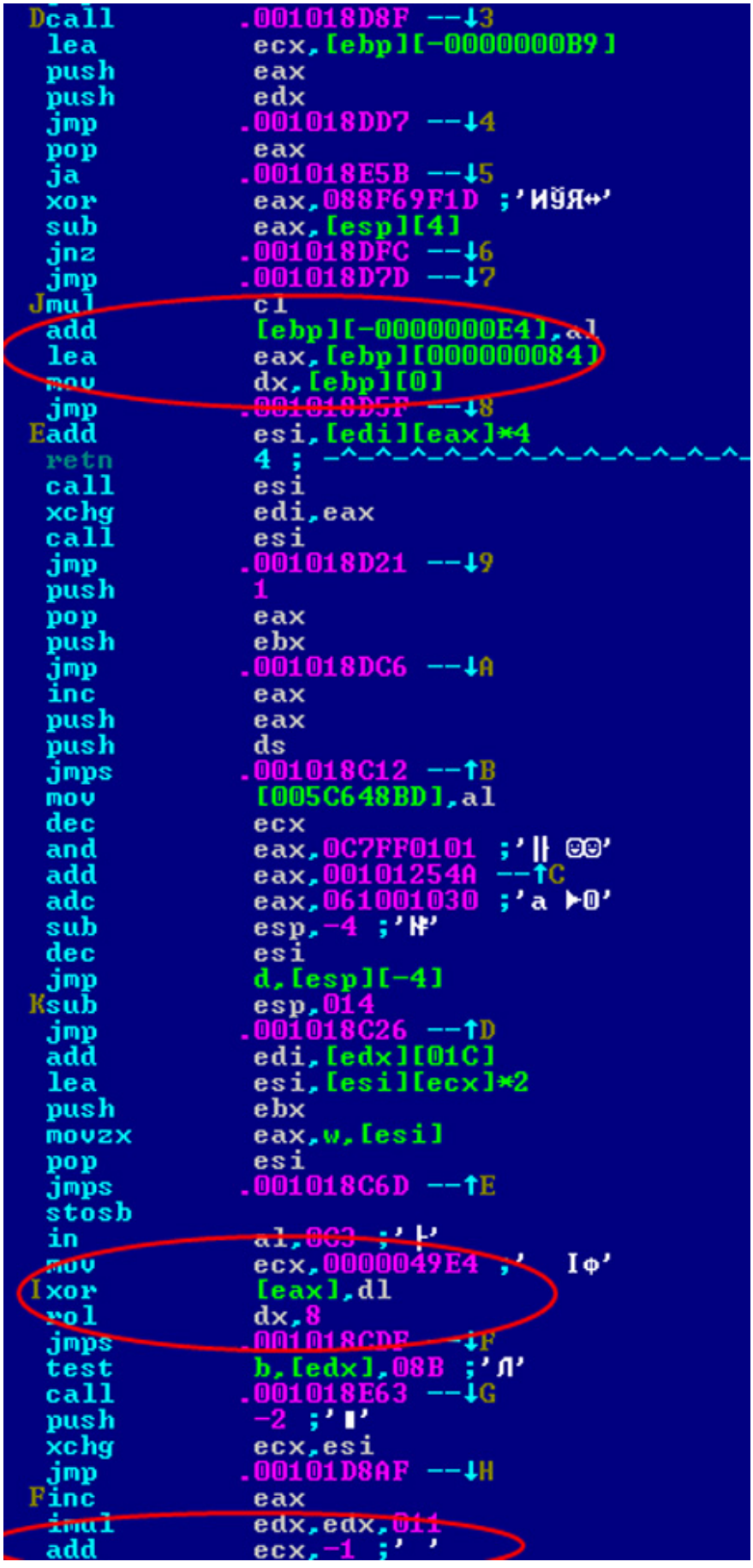
ADD [EAX], DL
ROL DX, 5
IMUL EDX, 13h

XOR [EAX], DH
ADD [EAX], DL
XCHG DH, DL
IMUL EDX, 1Fh

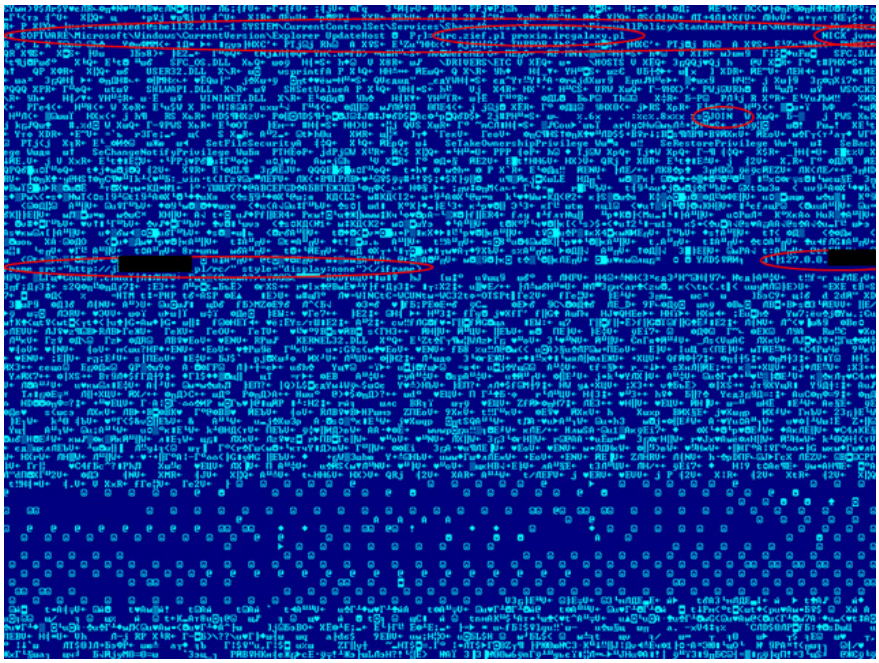
XOR [EAX], DH
XCHG DH, DL
ADD [EAX], DH
IMUL EDX, 1Fh

XOR [EAX], DL
ADD [EAX], DH
IMUL EDX, 2Bh
XCHG DH, DL

Of course, these instructions will change over time.


Dcall .001018D8F --↓3
lea ecx,[ebp][-0000000B9]
push eax
push edx
jmp .001018DD7 --↓4
pop eax
ja .001018E5B --↓5
xor eax,088F69F1D ;'ИЎЯ↔'
sub eax,[esp][4]
jnz .001018DFC --↓6
jmp .001018D7D --↓7
Jmul ci
add [ebp][-0000000E4],al
lea eax,[ebp][000000084]
mov dx,[ebp][0]
jmp .001018D5F --↓8
Eadd esi,[edi][eax]*4
retn 4 ;
call esi
xchg edi,eax
call esi
jmp .001018D21 --↓9
push 1
pop eax
push ebx
jmp .001018DC6 --↓A
inc eax
push eax
push ds
jmps .001018C12 --↑B
mov [005C648BD],al
dec ecx
and eax,0C7FF0101 ;'╢ ☺☺'
add eax,001012540 --↑C
adc eax,061001030 ;'a ►0'
sub esp,-4 ;'№'
dec esi
jmp d,[esp][-4]
Ksub esp,014
jmp .001018C26 --↑D
add edi,[edx][01C]
lea esi,[esi][ecx]*2
push ebx
movzx eax,w,[esi]
pop esi
jmps .001018C6D --↑E
stosb
in al,0C3 ;'├'
mov ecx,00000049E4 ;' Iφ'
Ixor [eax],dl
rol dx,8
jmps .001018CDF --↓F
test b,[edx],08B ;'Л'
call .001018E63 --↓G
push -2 ;'▮'
xchg ecx,esi
jmp .00101D8AF --↓H
Finc eax
imul edx,edx,011
add ecx,-1 ;' '

Part of the code is separated from the Main decoder
To continue discussing the process of executing the infected data file, let's move on to the implementation of the payload process with the code inside the static body. Typically, this process will start with the CALL instruction, first to calculate and identify virtual addresses, and then be applied to the actual addressing.



The static body part of the virus after being decoded

Lines marked with a red oval indicate the parts of the payload mechanism for the virus. For example: 'JOIN' and 'NICK' are IRC, ' *irc.zief.pl* ' and ' *proxim.ircgalaxy.pl* ' lines that are remote IRC servers where the virus performs reciprocating operations, ' *SYSTEMCurrentControlSet ServicesSharedAccess ParametersFirewallPolicy StandardProfileAuthorizedApplications List* 'is a registry key containing information about trusted Windows firewall programs.

Conclude

It can be said that Virut.ce has a quite diverse and dangerous infection mechanism, with polymorphic and obfuscation technologies. However, its payload mechanism is quite complex and cannot be underestimated. And the virus pattern could be considered the most successful combination of malicious technologies and techniques, created by hackers. In addition, there are a number of malicious programs that use obfuscated techniques for different purposes, including technology against mechanism for building and simulating actions . but Virut.ce is a perfect copy. most of the above features. It can be said that this is not a complete and detailed article about the Virut.ce template. Until April 2010, no new variant of Virut.ce has been discovered, which does not mean that its evolution has stopped. On the other hand, up to this point all Kaspersky Lab products have the ability to detect and destroy Virus.Win32.Virut.ce, even if its new variant appears.