Malware

Pericle Perazzo
pericle.perazzo@iet.unipi.it

Virus writers, who are they?

- Programmers who want to prove their skills to the community ('80-'90)
- Spammers
- Criminals searching for passwords and credit card numbers
- Detectives or secret services searching for personal information
- Intelligence agencies for cyber-warfare

Who writes malware?

In the past ('80-'90), the virus writers were principally young male programmers wanting to prove their skills to the community of *black hat*s.

Nowadays, malware is developed mainly for business purposes. Spammers create malware in order to force legitimate email accounts to send spam. Criminals develop spyware, in order to steal passwords and credit card numbers.

Finally, detectives and intelligence agencies develop malware to gather information about people under investigation, to perform industrial espionage or sabotage, or for cyber-warfare purposes.

Terminology

- *Malware*: Software that performs unwanted features
  - *Virus*: A piece of code which infects an executable file and auto-replicates
  - *Worm*: A program which infects a computer and auto-replicates through the Internet
  - *Trojan horse*: A program which performs actions unwanted by the user (does not reproduce)

The *malware* is software that performs features that are unwanted by the user.

A first classification of malware is based on its infection paradigm. A *virus* is a piece of code (not a standalone program) which infects executable files. A *worm* is a standalone program which infects computers (typically through the Internet). Finally, a *trojan horse* is a program which infects computers but does not self-propagate to other hosts.

Actually, (pure) viruses are very rare nowadays. The majority of malware is constituted by trojans and worms.

Terminology

- *Spyware*: Malware that collects information about the user and possibly sends it through Internet
- *Keylogger*: Spyware that records the key strokes
- *Adware*: Software (possibly malware) that displays advertising
- *Rootkit*: Malware that grants privileged access to the host system
- *Dialer*: Malware that performs phone calls (old 56k modem or modern smartphones)

We can make another classification of malware, basing on its objective.

Spyware is the malware which gathers information about the victim and possibly sends them to the cracker via Internet. A *keylogger* is a type of spyware which records the pressed keys. It is very useful to steal passwords. *Adware* shows advertising to the user, and is not necessarily malware. It is considered malware if the user did not give his consent to receive such advertising. A *rootkit* is the malware which grants a remote and privileged access to the victim's computer, typically through a remote shell. Finally, a *dialer* is malware that performs (pay per) phone calls to specific numbers. They were widespread in the '80-'90, when the majority of Internet connections were 56k-based. Nowadays, they made a strong comeback due to the diffusion of smartphones.

Malware functionalities

- Execution and infection (infection location)
- Self-replication (infect another host)
- Privilege escalation
- Manipulation (damage the host)
- Concealment (hide from detection)

The main functionalities of malware are the following:

1) Execution and infection. In order to execute the first time, the malware can leverage on poor precautions of the user, which runs suspect programs without security checks, or even on a security vulnerability of the software, without user's participation. In the first execution the malware *infects* the host (*installation*), that is, the malware makes sure about its successive executions. These features are mandatory for every malware. The following features are optional.

2) Self-replication. The malware propagates itself by infecting other hosts.

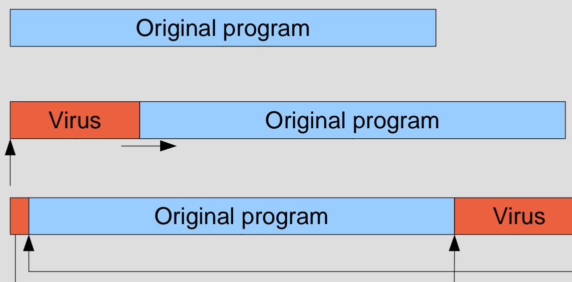
3) Privilege escalation. On the first execution, the malware assumes the privileges of the application it cracked. If this application run with limited privileges, the malware will have to assume administrator's privileges, in order to take the full control of the system. Typically, this is done by leveraging on a vulnerability of some admin-privileged process.

4) Manipulation. The malware modifies in some way the host and make some damage.

5) Concealment. The malware hides its presence to the user and the anti-virus software.

Execution of viruses

- Infection location:
  - Prepending
  - Appending
  - ...



The old-fashioned '80 viruses acted this way. They appended or prepended on executable files (programs, dynamic libraries, etc.) and run before the legitimate program. They infected executables rather than computers, and they spread from a computer to another through floppy discs containing infected executables.

Pseudo-code of a virus

```
void virus(){
    const char* signature = "1234567";

    do{
        get_random_executable_file();
        if(first_bytes != signature){
            prepend_me_to_file();
        }
    } while(first_bytes != signature);

    if(trigger){
        do_some_damage();
    }

    execute_host_program();
}
```

Avoid multiple infection

Infection

Damage

The image above shows the pseudo-code of a typical virus. The virus code is executed every time the user executes the program. It contains a mechanism to avoid multiple infections of the same executable. The mechanism involves a signature, that is a sequence of bytes by which the virus self-identifies. During the infection phase, the virus seeks for a non-infected executable and infects it. Then, if some particular activation condition is met, it makes some damage. Finally, it runs the original program.

Note that the virus is not a standalone program. It is rather a “fragment of code”, which “becomes alive” by appending itself to another program.

Pseudo-code of a worm

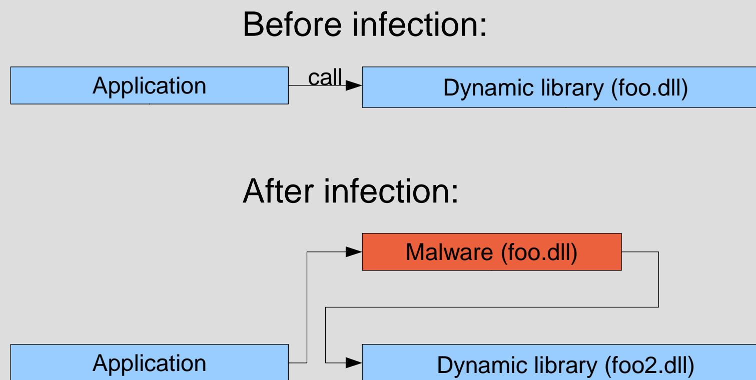
```
                     void main(){ // worm
Avoid                  check_if_already_infected();
multiple               if(already_infected){
infection                  return;
                       }

Infection              infect(); // make sure of successive executions

Privilege              if(!admin_privileges){
escalation                 get_admin_privileges();
                       }

                       for(;;){
Activation                 block_until_some_condition();

Self-replication           send_copies_of_me_over_internet();

Manipulation               do_some_damage();
                       }
                     }
```

The image above shows the pseudo-code of a typical worm. Note that this time the malware is a standalone program (`void main`).

Once the worm is executed, it checks whether the computer is already infected. If it is not, it infects on the system. Then, it checks for administrator privileges, and performs privilege escalation in case. After that, it blocks its execution until an activation condition is met. When it activates, it tries to infect other hosts through Internet and makes some damage.

Infection

- Registering as a start-up program
- Hijacking:



How malware assures its successive executions, every time the computer is booted?

There are many methods to do this. A first one is simply to register itself as a start-up program. As the operative system boots, it runs the malware together with the other services. Another method is the hijacking. The malware substitutes a legitimate library (e.g. a DLL) assuming its name and interface. When an application calls a method of the library, it actually executes the malware, which in turn forwards the call to the legitimate library. In this way the malware executes without the application noticing anything.

Self-replication

- A worm uses security vulnerabilities to infect other computers
- Infection vectors:
  - Web page
  - P2P
  - Any file containing code (executable, document, etc.)
  - Email (with file attached)
  - USB autorun
  - Malformed requests

Typically, a worm uses the following infection vectors to propagate to a host to another:

1) A web site, cracked or controlled directly by the malware writer.

2) Peer to peer channels. Especially through the “fakes”.

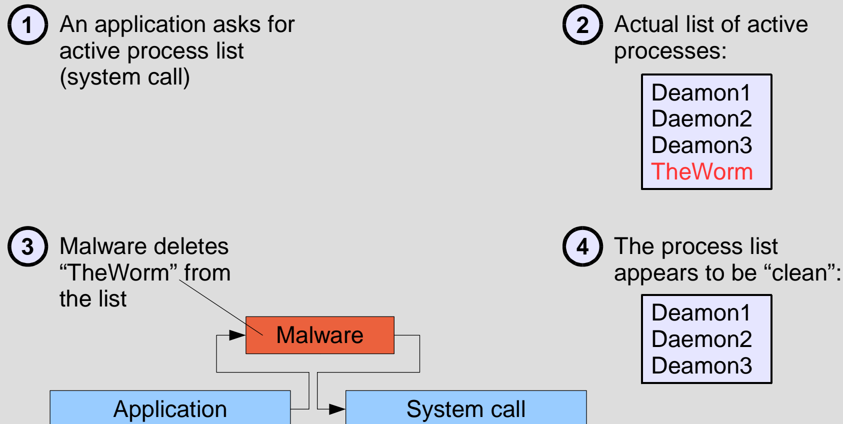
3) Any file containing code. Thus, not only executables but every file format which can contain scripts or macros, like Word documents, etc.

4) Emails with attached an infected file.

5) The autorun functionality of the USB flash drives.

6) Malformed requests which leverages on vulnerabilities (e.g. buffer overflow) of the receiving applications.

Concealment



Some malware takes actions in order to hide its infection. Several techniques are possible. The image above shows an example:

1) An application (or the user) wants to view the list of the active processes.

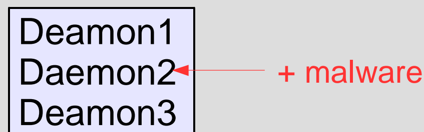
2) In the process list, among legitimate processes, there is the worm's process. The malware wants to hide it.

3) The malware hijacks the library call that retrieves the active process list. It intercepts the system call and cancels its name from the actual list.

4) The application (or the user) views the "clean" list.

Concealment

- Concealment through Dll-Injection
  - Inject malicious code into a legitimate process
  - Effects:
    - Conceal from detection
    - Grant access to process's resources



Another technique is called Dll-Injection. In this case, the worm injects its code, usually a Dll image, inside another legitimate process. In this way, the malware conceals from detection and accesses the resources of the injected process.

Anti-virus

Anti-virus functionalities

- Recognize malware
  - Prevent infection
  - Detect infection
- Remove malware
- Other functions, like firewall, intrusion detection, anti-spam, anti-phishing

The main functionalities of anti-virus software are the following three:

1) *Prevent* malware's infection

2) *Detect* malware's infection

3) *Remove* malware's infection

The first two features require the ability to *recognize* malware from legitimate software, before and after the infection.

Modern anti-virus software has other features, among which firewall, intrusion detection, anti-spam and anti-phishing.

Malware detection

Is it possible to develop
a perfect malware detector?

Cohen's theorem

“A perfect malware detector is impossible”

Proof (by contradiction):

(1)
```
bool is_malware(·);
```

(2)
```
void my_prog(){
    if(!is_malware(my_prog)){
        behave_like_malware();
    }
}
```

(3)
```
is_malware(my_prog)
```
→ ~~false~~
→ ~~true~~

Fred Cohen (the inventor of the word “computer virus”) developed the following theorem: “a perfect malware detector is impossible”. A proof by contradiction follows.

1) Let us suppose a function `is_malware(·)` exists, which examines a program or a piece of code and returns `true` or `false` whether the code is malware or not. Let us suppose that `is_malware` is perfect, that is it does not give any false negatives nor false positives.

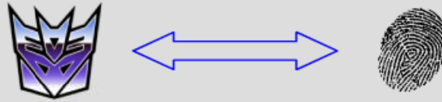
2) Then, it is possible to build a program named `my_prog`, which executes `is_malware` on itself. It behaves like a malware if the function return `false`, it does nothing otherwise.

3) `is_malware(my_prog)` cannot return neither `true` or `false`. If it return `true`, `my_prog` will not be malware, thus it will be a false positive. If it return `false`, `my_prog` will be a malware, thus it will be a false negative.

Hence, such a perfect malware detector is not possible.

Signature-based detection

- *Malware signature*: byte-pattern that identifies malicious code (or a family of)
- Signature remains the same:
  - In the infection vector
  - In the infected processes

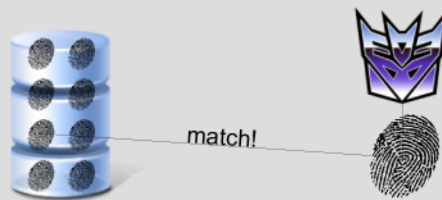


The real-life anti-virus software tries to recognize malware by means of two main techniques: signature-based detection and anomaly-based detection.

A signature is a sequence of instructions in the code of a piece of malware, which univocally identifies it. The presence of a signature reveals the presence of malware inside an infected file or a running process.

Signature-based detection

- Use handmade databases of malware's signatures
- Databases must be periodically updated



The signature-based detection relies on a database of malware's signatures. Every suspect program is checked to contain such signatures. Obviously, signature database is periodically updated by the anti-virus company.

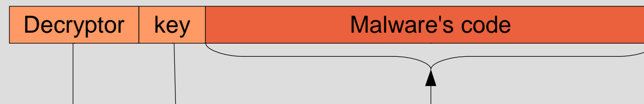
Signature-based detection

- Signature-based detection
  - Efficient
  - No false positives
  - Identification of malware
  - No protection against new malware
  - No protection against polymorphic malware

Such a method is efficient, gives very rare false positives, and identifies the specific malware, rather than detecting its presence only. The identification is particularly important for the sake of the removal procedure. Different malware requires different removal procedures. This is the main recognizing method of every anti-virus software.

The main drawback of this method is that it cannot recognize new malware, whose signature has not been isolated yet. Another drawback is that it does not protect against polymorphic malware, as we will see in the following.

Self-encrypting malware



On first execution:

```
retrieve_key();
decrypt_malware_code();
execute_malware_code();
```

On self-replication:

```
generate_random_key();
crypt_malware_code();
send_infection();
```

The malware can self-encrypt itself in order to avoid detection. The image above shows an example of self-encrypting malware. The decryptor and key are stored in the clear before the cyphered malware's code. During the first execution, the decryptor recover the malware's code and executes it. During the self-replication phase, the malware chooses a new random key and encrypts its code.

Note that a strong encrypting algorithm is not needed here. The cyphering aims only at avoid the detection, rather than protecting the confidentiality. Very simple encrypting algorithms like XOR masks are often used.

Self-encrypting malware

- Self-encryption
  - Efficient
  - Malware detection system can match the decryptor as a signature

This concealment technique is easy and efficient, but malware can still be recognized by means of the code of the decryptor. In fact, such a code does not vary from an infection to another and can be used as a signature. However, this technique is still useful for the malware, because it significantly reduces the space where a signature can be found.

Polymorphic malware

- Changes its form from generation to generation
- Does not change its behaviour

A more advanced technique is polymorphism, which consists in changing the code of the malware without changing its behaviour.

Polymorphic malware

- Code obfuscation: **nop** insertion

```
00401005    8BF0            MOV ESI,EAX
00401007    3E:8A00         MOV AL,BYTE PTR DS:[EAX]
0040100A    84C0            TEST AL,AL
0040100C  v 74 46           JE SHORT Test.00401054
0040100E    53              PUSH EBX
0040100F    3E:8F05 74F940( POP DWORD PTR DS:[40F974]
00401016    D3DB            RCR EBX,CL
00401018    0FCB            BSWAP EBX
0040101A    68 56104000     PUSH Test.00401056
0040101F    5B              POP EBX
00401020    3E:8903         MOV DWORD PTR DS:[EBX],EAX
00401023    43              INC EBX
00401024    0FBDC2          BSR EAX,EDX
00401027    A9 46A978DC     TEST EAX,DC78A946
0040102C    8BC2            MOV EAX,EDX
0040102E    52              PUSH EDX
0040102F    B6 86           MOV DH,86
00401031    B3 27           MOV BL,27
00401033    B8 7CFAA17F     MOV EAX,7FA1FA7C
00401038  v EB 01           JMP SHORT Test.0040103B
0040103A    90              NOP
0040103B    0FBCC2          BSF EAX,EDX
0040103E    3E:C705 FC8841( MOV DWORD PTR DS:[4188FC],0
00401049    2D 210DE8B9     SUB EAX,B9E80D21
0040104E    69DA E577D49D   IMUL EBX,EDX,9DD477E5
```

```
00401005    8BF0            MOV ESI,EAX
00401007    3E:8A00         MOV AL,BYTE PTR DS:[EAX]
0040100A    84C0            TEST AL,AL
0040100C  v 74 49           JE SHORT Test.00401057
0040100E    53              PUSH EBX
0040100F    3E:8F05 74F940( POP DWORD PTR DS:[40F974]
00401016    90              NOP
00401017    D3DB            RCR EBX,CL
00401019    0FCB            BSWAP EBX
0040101B    68 59104000     PUSH Test.00401059
00401020    5B              POP EBX
00401021    3E:8903         MOV DWORD PTR DS:[EBX],EAX
00401024    90              NOP
00401025    43              INC EBX
00401026    0FBDC2          BSR EAX,EDX
00401029    A9 46A978DC     TEST EAX,DC78A946
0040102E    8BC2            MOV EAX,EDX
00401030    52              PUSH EDX
00401031    90              NOP
00401032    B6 86           MOV DH,86
00401034    B3 27           MOV BL,27
00401036    B8 7CFAA17F     MOV EAX,7FA1FA7C
0040103B  v EB 01           JMP SHORT Test.0040103E
0040103D    90              NOP
0040103E    0FBCC2          BSF EAX,EDX
00401041    3E:C705 FC8841( MOV DWORD PTR DS:[4188FC],0
0040104C    2D 210DE8B9     SUB EAX,B9E80D21
00401051    69DA E577D49D   IMUL EBX,EDX,9DD477E5
```

This can be done in many ways. For example by simply inserting `nop` operations at random.

Polymorphic malware

- Code obfuscation: ineffective operations

```
00401005    8BF0             MOV ESI,EAX
00401007    3E:8A00          MOV AL,BYTE PTR DS:[EAX]
0040100A    84C0             TEST AL,AL
0040100C  v 74 46            JE SHORT Test.00401054
0040100E    53               PUSH EBX
0040100F    3E:8F05 74F940(  POP DWORD PTR DS:[40F974]
00401016    D3DB             RCR EBX,CL
00401018    0FCB             BSWAP EBX
0040101A    68 56104000      PUSH Test.00401056
0040101F    5B               POP EBX
00401020    3E:8903          MOV DWORD PTR DS:[EBX],EAX
00401023    43               INC EBX
00401024    0FBDC2           BSR EAX,EDX
00401027    A9 46A978DC      TEST EAX,DC78A946
0040102C    8BC2             MOV EAX,EDX
0040102E    52               PUSH EDX
0040102F    B6 86            MOV DH,86
00401031    B3 27            MOV BL,27
00401033    B8 7CFAA17F      MOV EAX,7FA1FA7C
00401038  v EB 01            JMP SHORT Test.0040103B
0040103A    90               NOP
0040103B    0FBCC2           BSF EAX,EDX
0040103E    3E:C705 FC8841(  MOV DWORD PTR DS:[4188FC],0
00401049    2D 210DE8B9      SUB EAX,B9E80D21
0040104E    69DA E577D49D    IMUL EBX,EDX,9DD477E5
```

```
00401005    8BF0             MOV ESI,EAX
00401007    3E:8A00          MOV AL,BYTE PTR DS:[EAX]
0040100A    84C0             TEST AL,AL
0040100C  v 74 4D            JE SHORT Test.0040105B
0040100E    53               PUSH EBX
0040100F    3E:8F05 74F940(  POP DWORD PTR DS:[40F974]
00401016    D3DB             RCR EBX,CL
00401018    0FCB             BSWAP EBX
0040101A    68 5D104000      PUSH Test.0040105D
0040101F    5B               POP EBX
00401020    3E:8903          MOV DWORD PTR DS:[EBX],EAX
00401023    43               INC EBX
00401024    0FBDC2           BSR EAX,EDX
00401027    A9 46A978DC      TEST EAX,DC78A946
0040102C    8BC2             MOV EAX,EDX
0040102E    90               NOP
0040102F    90               NOP
00401030    42               INC EDX
00401031    52               PUSH EDX
00401032    FE0C24           DEC BYTE PTR SS:[ESP]
00401035    4A               DEC EDX
00401036    B6 86            MOV DH,86
00401038    B3 27            MOV BL,27
0040103A    B8 7CFAA17F      MOV EAX,7FA1FA7C
0040103F  v EB 01            JMP SHORT Test.00401042
00401041    90               NOP
00401042    0FBCC2           BSF EAX,EDX
00401045    3E:C705 FC8841(  MOV DWORD PTR DS:[4188FC],0
00401050    2D 210DE8B9      SUB EAX,B9E80D21
00401055    69DA E577D49D    IMUL EBX,EDX,9DD477E5
```

Or by changing instruction or sequence of instruction with equivalent ones, that have the same final effect.

Polymorphic malware

- Code obfuscation: register reassignment

```
00401005   8BF0               MOV ESI,EAX
00401007   3E:8A00            MOV AL,BYTE PTR DS:[EAX]
0040100A   84C0               TEST AL,AL
0040100C v 74 46              JE SHORT Test.00401054
0040100E   53                 PUSH EBX
0040100F   3E:8F05 74F940     POP DWORD PTR DS:[40F974]
00401016   D3DB               RCR EBX,CL
00401018   0FCB               BSWAP EBX
0040101A   68 56104000        PUSH Test.00401056
0040101F   5B                 POP EBX
00401020   3E:8903            MOV DWORD PTR DS:[EBX],EAX
00401023   43                 INC EBX
00401024   0FBDC2             BSR EAX,EDX
00401027   A9 46A978DC        TEST EAX,DC78A946
0040102C   8BC2               MOV EAX,EDX
0040102E   52                 PUSH EDX
0040102F   B6 86              MOV DH,86
00401031   B3 27              MOV BL,27
00401033   B8 7CFAA17F        MOV EAX,7FA1FA7C
00401038 v EB 01              JMP SHORT Test.0040103B
0040103A   90                 NOP
0040103B   0FBCC2             BSF EAX,EDX
0040103E   3E:C705 FC8841     MOV DWORD PTR DS:[4188FC],0
00401049   2D 210DE8B9        SUB EAX,B9E80D21
0040104E   69DA E577D49D      IMUL EBX,EDX,9DD477E5
```

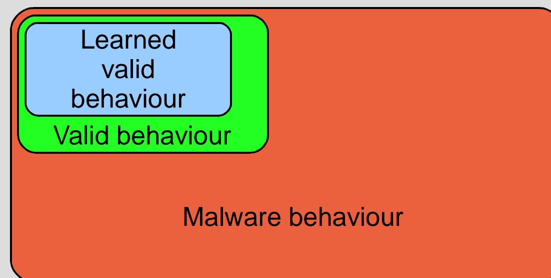
```
00401005   8BF3               MOV ESI,EBX
00401007   3E:8A1B            MOV BL,BYTE PTR DS:[EBX]
0040100A   84DB               TEST BL,BL
0040100C v 74 48              JE SHORT Test.00401056
0040100E   52                 PUSH EDX
0040100F   3E:8F05 74F940     POP DWORD PTR DS:[40F974]
00401016   D3DA               RCR EDX,CL
00401018   0FCA               BSWAP EDX
0040101A   68 58104000        PUSH Test.00401058
0040101F   5A                 POP EDX
00401020   3E:891A            MOV DWORD PTR DS:[EDX],EBX
00401023   42                 INC EDX
00401024   0FBDD8             BSR EBX,EAX
00401027   F7C3 46A978DC      TEST EBX,DC78A946
0040102D   8BD8               MOV EBX,EAX
0040102F   50                 PUSH EAX
00401030   B4 86              MOV AH,86
00401032   B2 27              MOV DL,27
00401034   BB 7CFAA17F        MOV EBX,7FA1FA7C
00401039 v EB 01              JMP SHORT Test.0040103C
0040103B   90                 NOP
0040103C   0FBCD8             BSF EBX,EAX
0040103F   3E:C705 FC8841     MOV DWORD PTR DS:[4188FC],0
0040104A   81EB 210DE8B9      SUB EBX,B9E80D21
00401050   69D0 E577D49D      IMUL EDX,EAX,9DD477E5
```

EAX → EBX
EBX → EDX
EDX → EAX

Or by exchanging every occurrence of a register (e.g. EAX) with another (e.g. EBX).

Anomaly-based detection

- Training phase: register statistics about the normal behaviour
- Detection phase: using the training data, try to recognize abnormal behaviours



Another technique is the anomaly-based detection. This technique tries to detect malware by discriminating the "normal" behaviour of a system from the "abnormal" one.

It works in two phases. During the *learning phase*, the anti-virus records statistical data about what it's consider the "normal" behaviour. It's important that the learning phase is run on non-infected systems. Then, during the *detection phase*, the anti-virus uses the collected data to detect malware's behaviour.

Anomaly-based detection

- Protects against new malware
- Protects against polymorphic malware
- More complex and inefficient
- False negatives and false positives
- No identification
- Requires malware execution

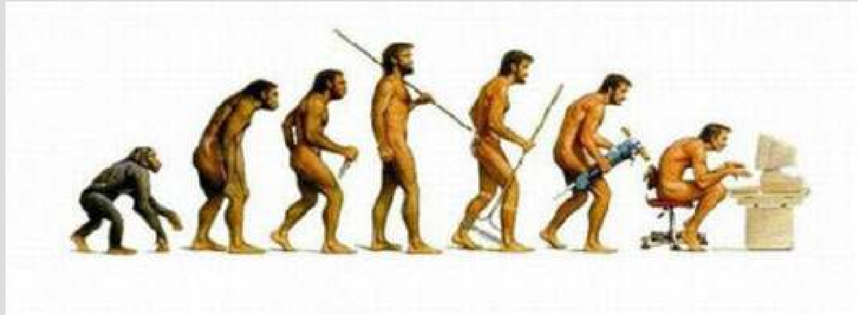
This technique can protect against new malware and polymorphic malware, but has some drawbacks. It is more complex compared to signature-based techniques. It gives high percentages of false negatives and false positives. It does not offer an identification for the detected malware. Finally, it requires to execute the malware. The last drawback is avoided by means of the *sandbox* technique.

Sandbox technique

- Run the suspect program in a controlled and isolated environment (typically a virtual machine)
- Search for signatures or anomalies

A sandbox is a controlled and isolated environment, typically a virtual machine, where the anti-virus can run suspect programs in order to detect malicious behaviours. The malware must be unaware to be running inside a sandbox.

Malware's evolution



Blaster case

- Blaster worm: August 2003
- Infects through Windows' DCOM-RPC (buffer overflow)
- Contains the joke string:

  I just want to say LOVE YOU SAN!!
  billy gates why do you make this possible ?
  Stop making money and fix your software!!

- DDoS against windowsupdate.com on 15th of each month

Blaster (2003), also known as Lovesan, was a computer worm famous for its quick diffusion.

Stuxnet case

“Stuxnet is the type of threat we hope to never see again„

Symantec Security Response team (2010)

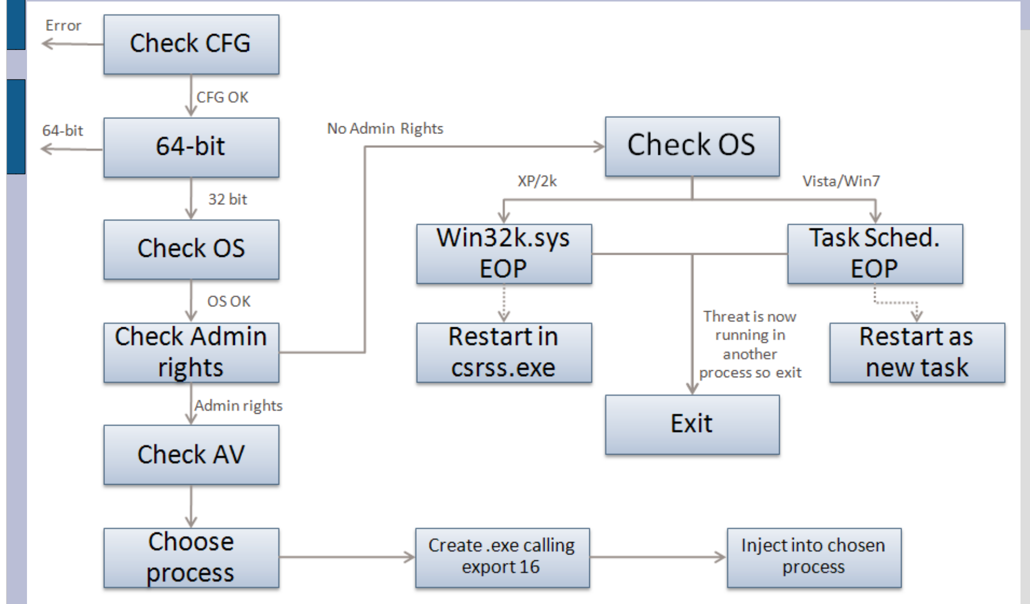
Stuxnet case

- Discovered in June 2009 (1st variant), March 2010 (2nd variant), April 2010 (3rd variant)
- Exploits 4 zero-days vulnerabilities to self-replicate and perform privilege escalation
- Uses 2 compromised digital certificates for driver installation
- Can sabotage industrial control systems (gas pipelines, power plants)
- 60% of infections in Iran

Stuxnet (2009-2010) and its brother Duqu (2011) are the first computer worms developed for cyber-warfare. Stuxnet exploited four zero-day (that is, previously unknown) vulnerabilities and two compromised digital certificates. The development of Stuxnet required a lot of money. Security experts believe that probably a government or an intelligence agency are involved in its development.

Stuxnet case

- Self-replicates through 6 different infection vectors
- Self-updates through a peer-to-peer mechanism
- Performs Dll-Injection on different processes, depending on the security software installed
- Do nothing if industrial control system is not found
- Programmed to self-removal on 24 June 2012



The image above shows the execution scheme of Stuxnet. It uses two different zero-day vulnerabilities to take administrator privileges, depending on the operative system of the host (WinXP-2K or Vista-Win7). After that, it Dll-injects different processes depending on the anti-virus software installed.

References

- D. Ferbrache *A Pathology of Computer Viruses,* Springer-Verlag
- E. Skoudis, L. Zeltser Malware: Fighting Malicious Code,
- N. Falliere, L.O. Murchu, E. Chien *W32.Stuxnet Dossier*, Symantec Security Response