

# File Infection Techniques

Wei Wang

# File Infection Techniques

- Executable files (\*.EXE, \*.COM, \*.BAT, etc.) are often the target of viruses
- Executing an infected file usually triggers replication of the virus into other files
- Executable file infection techniques can be categorized broadly by asking where the virus code is placed in the file

# Basic File Infection Goals

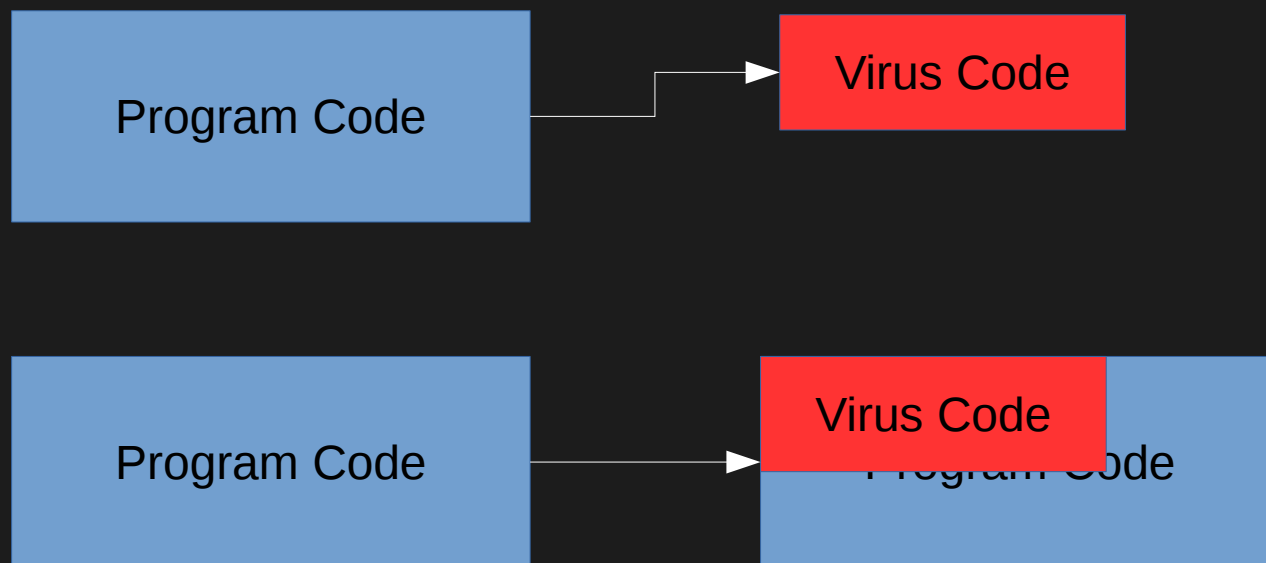
- Enable virus code execution
- Stealth:
  - Preserve original file size
  - Preserve original file functionality – hide from user
  - Pretend to be normal code – hide from anti-virus scanner

# Location: Beginning of the File

- Beginning always refers to the start of the executable, which might follow a header area in some file formats
- A virus can either preserve the original beginning of the file, or destroy it by overwriting
- Destructiveness always reduces the stealthiness of the virus

# Beginning of the File with Destructive Overwrite

- Two primary methods:
  - Replace \*.exe file with virus \*.exe
  - Overwrite only the beginning of a \*.exe that is larger than the virus \*.exe



# Beginning of the File with Destructive Overwrite cont'd

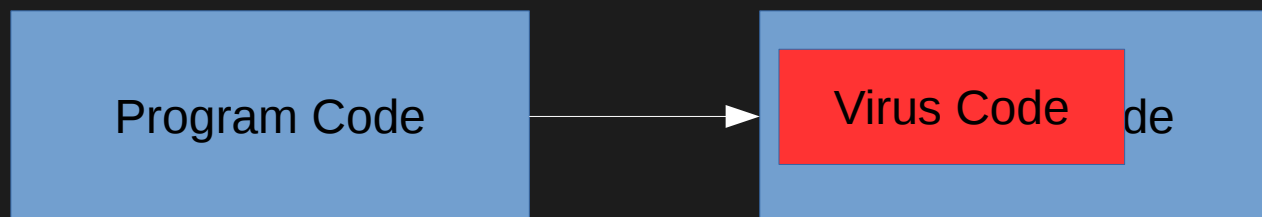
- Neither method is stealthy:
  - The \*.exe has lost its functionality entirely, so the user notices that something is wrong
  - Anti-virus software finds a virus easily right at the beginning of the file
- The first method often changes the file size, making it even less stealthy than the second method

# Beginning of the File with Destructive Overwrite cont'd

- The file size change was only significant for stealthy viruses when first-generation anti-virus software depended on keeping track of file sizes
- Both kinds of overwriting viruses can only be repaired by restoring files from a backup
- E.g. the LoveLetter mass mailer worm, after replicating by email, overwrote every file on the system that had one of 32 file extensions: \*.c, \*.cpp, \*.mp3, \*.vbs, etc.
  - It had already replicated to other systems, so it no longer tried to remain stealthy; a common design for worms

# Random Location in the File with Destructive Overwrite

- The Russian Omud virus, also called 8888, overwrote at a random location in the \*.exe file.
  - Anti-virus software must now search the entire file to find it; this defeated early anti-virus software
  - Control might transfer to the virus during execution of the \*.exe, or it might not, or program might crash; stealth came at a price!



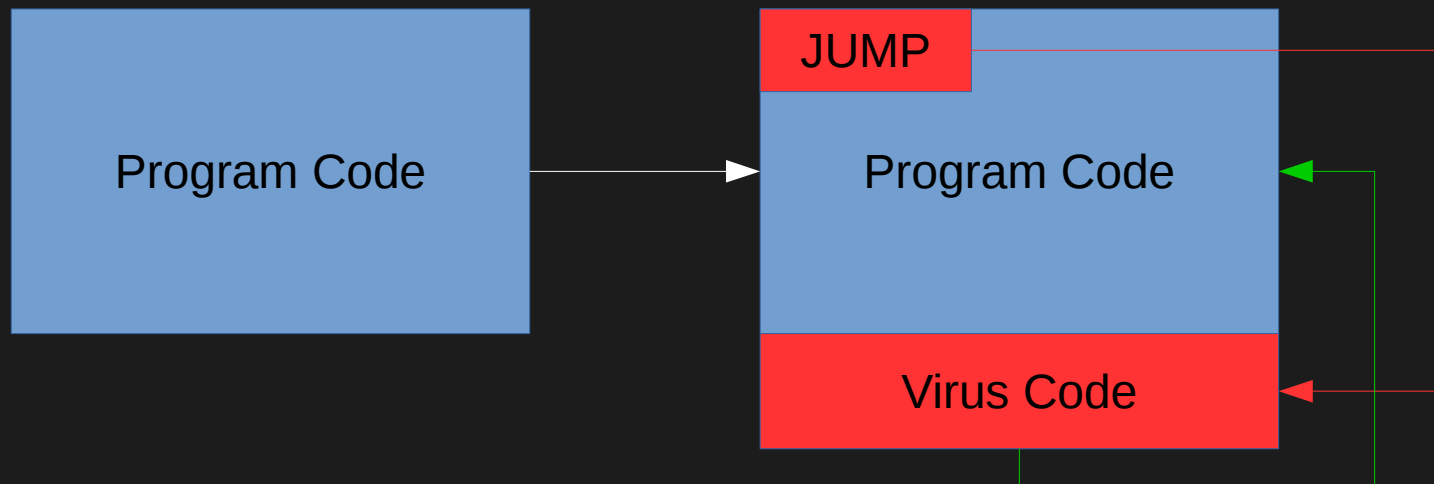


# Appending Viruses

- A jump, or tricky jump, to the virus address is overwritten on the first few bytes of the executable
- Virus code is appended to the file
- Overwritten instructions are saved in the virus
  - When the virus is about to terminate, it executes the saved instructions and jumps back to the spot that followed them
  - Application program functionality is preserved (stealth)
- So common among DOS .COM files that it was called the normal COM virus technique; Vienna and Suicide are famous examples of this kind of virus

# Appending Viruses cont'd

- The jump, or tricky jump, is easily spotted by anti-virus software
- The file size has changed



# Appending Viruses cont'd

- The stealth depends on executing the original application successfully when the virus code has finished, AND not spending too long in the virus code
- In order to execute the application successfully, the virus often copies the application code into a temporary file, then calls `system()` or a similar function to execute the contents of that file
  - Must pass original command-line arguments!

# Multiple Techniques

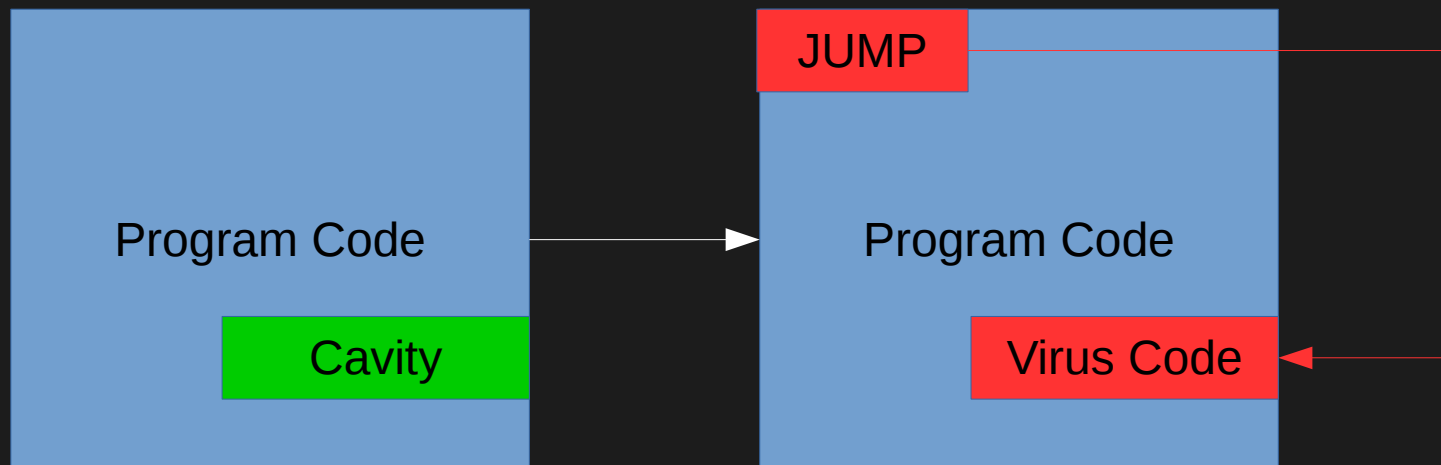
- Many viruses implement, in one virus, several of the techniques we have studied
- 1991 DOS normal COM example: Phantom
  - Appending COM file infector (normal COM)
  - Memory resident: installs itself into high DOS memory, reduces available memory by about 2KB, monitors system activity and infects COM files as they are executed
  - Hooks interrupts 20h and 21h in order to intercept COM file executions
  - Existed in multiple variants with different messages

# Cavity Viruses

- Virus creators often search for space within a file that is filled with zeroes or ASCII blanks
- These spaces, or cavities, can be filled with virus code without changing the file size
- A single cavity might be big enough for the whole virus, or the virus might be distributed into multiple small cavities, loaded into memory by the virus loader code at the head of the virus, connected by jump instructions (a fractionated cavity virus)
- Still need to reach the start of the virus with a jump, or modify the PE entry point

# Cavity Viruses cont'd

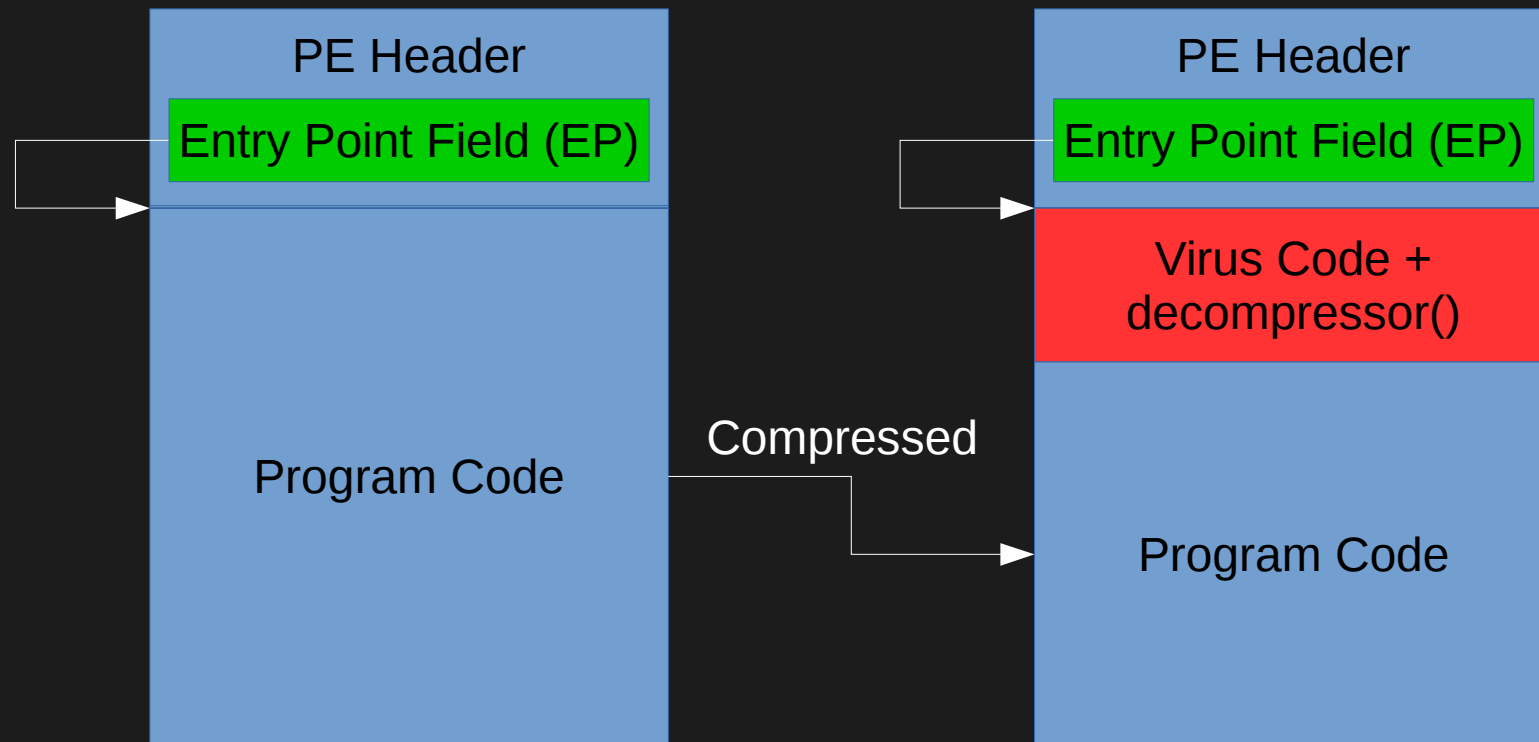
- Jump, or modified PE entry point, detectable by anti-virus software
- Disinfection can be difficult (was the original cavity full of zeroes, or ASCII blanks?)



# Compressing Viruses

- Application code is compressed
- Virus code plus decompressor code fits into the space that was saved
- Can keep the file size from changing
- Might not even change the entry point!

# Compressing Viruses Illustration





# Compressing Viruses cont'd

- How can a compressing virus be detected and disinfected?
- The virus code might even be compressed, so that only the decompressor code is recognizable as normal code
- However, a self-extracting archive would have a similar appearance and be quite legitimate
- File size and entry point could be unchanged
- Application behavior could be preserved

# Detecting a Compressing Virus

- When a virus outbreak occurs, reports come in to major anti-virus software vendors from their customers
- More expensive system scans than are normal for a background anti-virus program might reveal that known applications now have unintelligible executables
- Disassembly tools are used to examine the code, and human intelligence is needed to find the decompressor code
- A copy of the virus code can be decompressed using the decompressor
- The virus design is then figured out by walk-throughs

# Detecting a Compressing Virus cont'd

- Which other files on the system are targeted for infection can now be determined by examining the virus code
- A code pattern is devised that describes unique instruction sequences in the decompressor code
- The system is scanned to verify that this code pattern is not found in uninfected files
- The virus code pattern database is updated, and customers download the update
- More on pattern recognition shortly

# Disinfecting a Compressing Virus

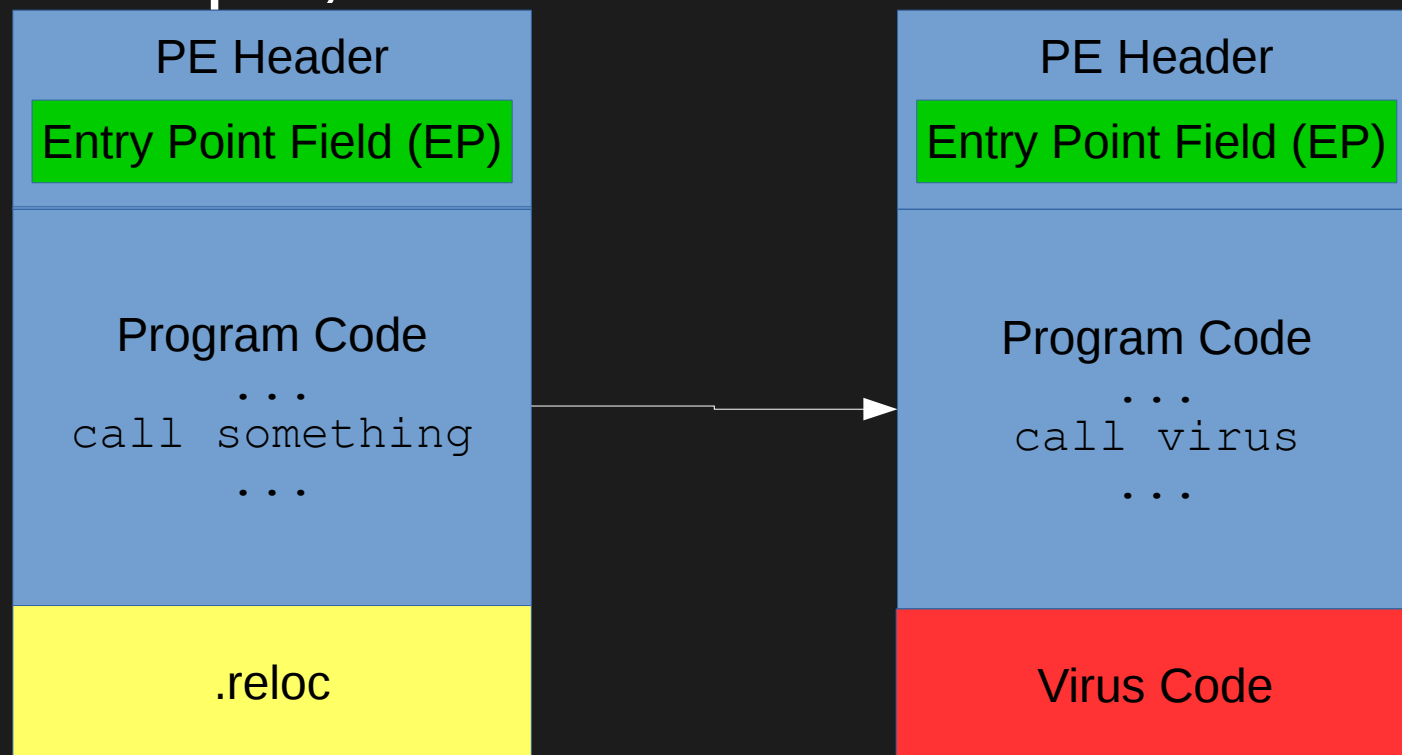
- With the virus and decompressor understood, the decompressor algorithm can be applied to the compressed application code
- The virus code and decompressor are removed
- The anti-virus software might maintain a database of cryptographic checksums for application executables
- If the disinfecting application now matches its stored checksums, success is declared
- Otherwise, restore the file from backup

# Entry-Point Obscuring (EPO) Viruses

- Anti-virus software closely examines PE file headers, entry points, and the initial code executed at the entry point
- A stealthy virus must be designed to avoid changes to any of these locations
- The EPO virus obscures its own entry point by finding a call instruction in the targeted PE file and “hijacking” the call so that the virus code is called instead

# EPO Virus cont'd

- A function call within the application becomes a call to the virus code.
- For example,



# EPO Virus cont'd

- The virus code saves the registers in order to preserve the parameters that were being passed. Also saves the original call target address.
- When the virus finishes executing, it restores the registers and does a tricky jump back to the original call target

# EPO Virus cont'd

- How can a virus find a function call?
- The binary opcodes can be scanned. However, constant data in the code section can happen to have the same value as a call opcode
- The most well-designed viruses examine the field that gives the target of the call.
  - What does the virus do with this field?
  - How does this help the virus? (answers in the last slide)



# EPO Virus cont'd

- The .reloc section in PE format gives information to be used if the program has to be relocated during execution, i.e. reloaded at a different load point because the system had to defragment memory or some other reason
- Relocation during execution is unusual, so the .reloc section usually sits unused, e.g. in statically linked executables
- Unfortunately, this provides a large cavity for viruses to use and still leave the file size unchanged

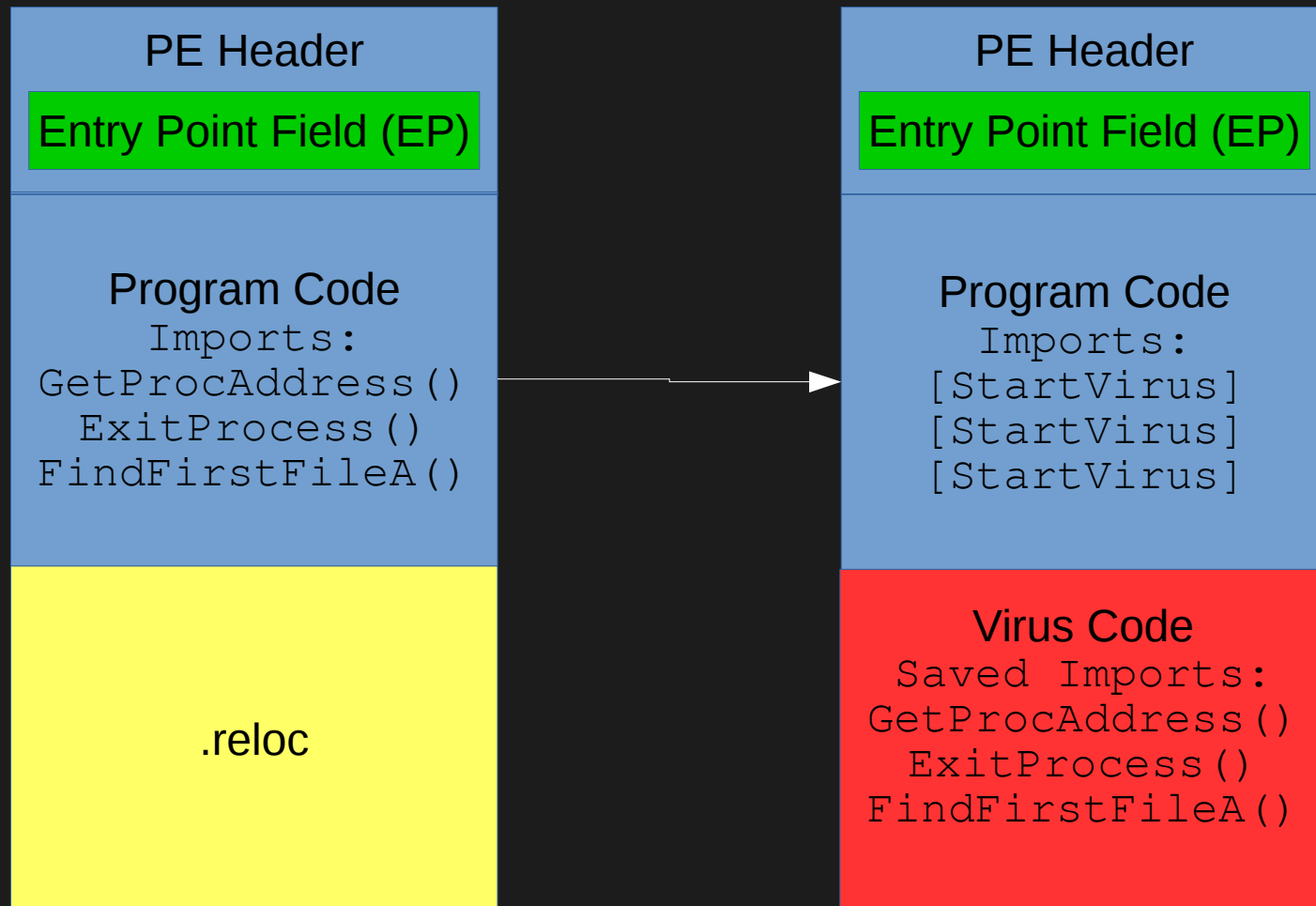
# Detecting Call-Hijacking Viruses

- The .reloc section is examined by modern anti-virus software to see if it looks like a legitimate .reloc section
- Code patterns such as saving state, tricky jumps, etc., can be detected in the .reloc section
- Some EPO viruses are accidentally destructive; hard to re-enter the application successfully in some cases

# Import Table-replacing EPO Virus

- The IAT (import address table) is the function pointer table that records the API (application program interface) that the user application is using
- Several function pointers can be saved in the virus body, then replaced with pointers to the virus code
- After the virus code is memory-resident, it can restore the IAT in memory so that the API is preserved and stealth is maintained

# Import Table-replacing EPO Virus cont'd



# Semi-EPO: Unknown Entry Points

- Windows PE format has another entry point besides the main entry point – thread local storage (TLS) entry points
- Windows loader search for TLS entry points and execute it first before the main entry point
- TLS entry points can be changed by virus to point to virus code
- Example: W32/Chiton

# Analyzing the Target of a Call Instruction

- The virus can check if the target address has function prologue instructions, which are “push %ebp; mov %esp, %ebp”