

Computer Viruses

Christian Klein <kleinc@cs.bonn.edu>

Agenda

- Overview
- History
- More Overview
- Techniques
- After the presentation: heavy work!

Worms and Viruses

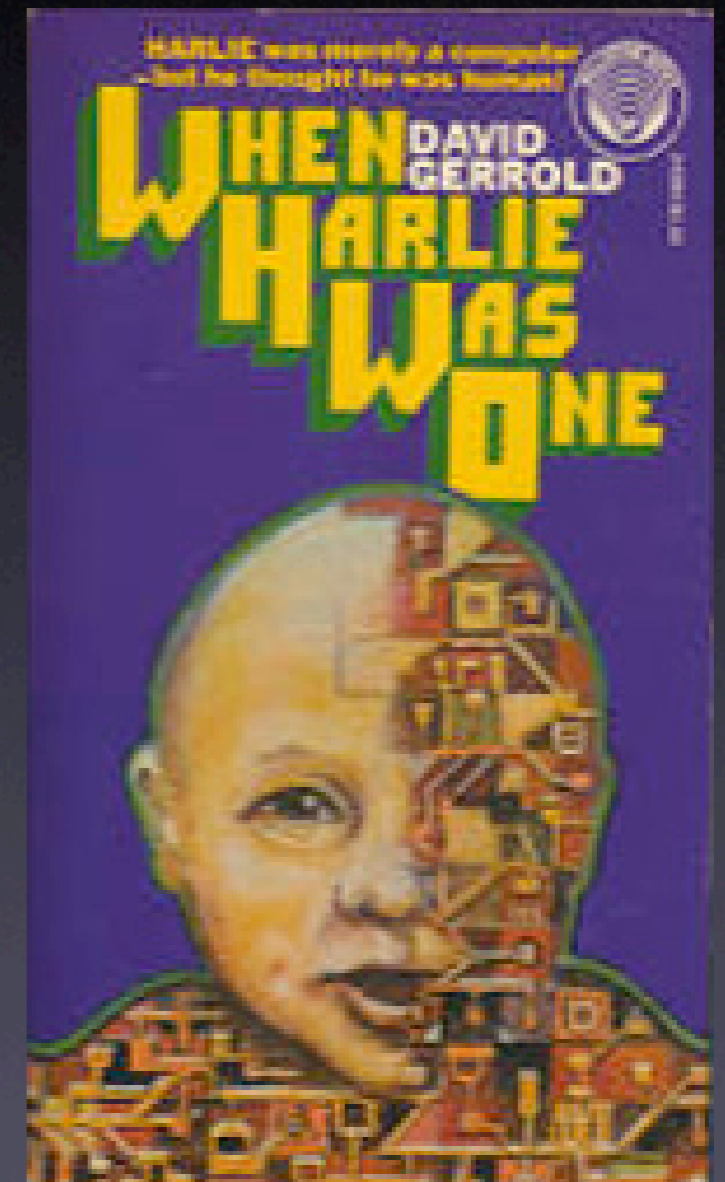
- analogy from medicine
- first mentioned in science fiction literature
- hostile code
- non-vaccinated hosts get infected

The Virus

"Do you remember the VIRUS program?"

"Vaguely. Wasn't it some kind of computer disease or malfunction?"

"Disease is closer. There was a science-fiction writer once who wrote a story about it -- but the thing had been around a long time before that. It was a program that -- well, you know what a virus is, don't you? It's pure DNA, a piece of renegade genetic information. It infects a normal cell and forces it to produce more viruses - viral DNA chains - instead of its normal protein. Well, the VIRUS program does the same thing."



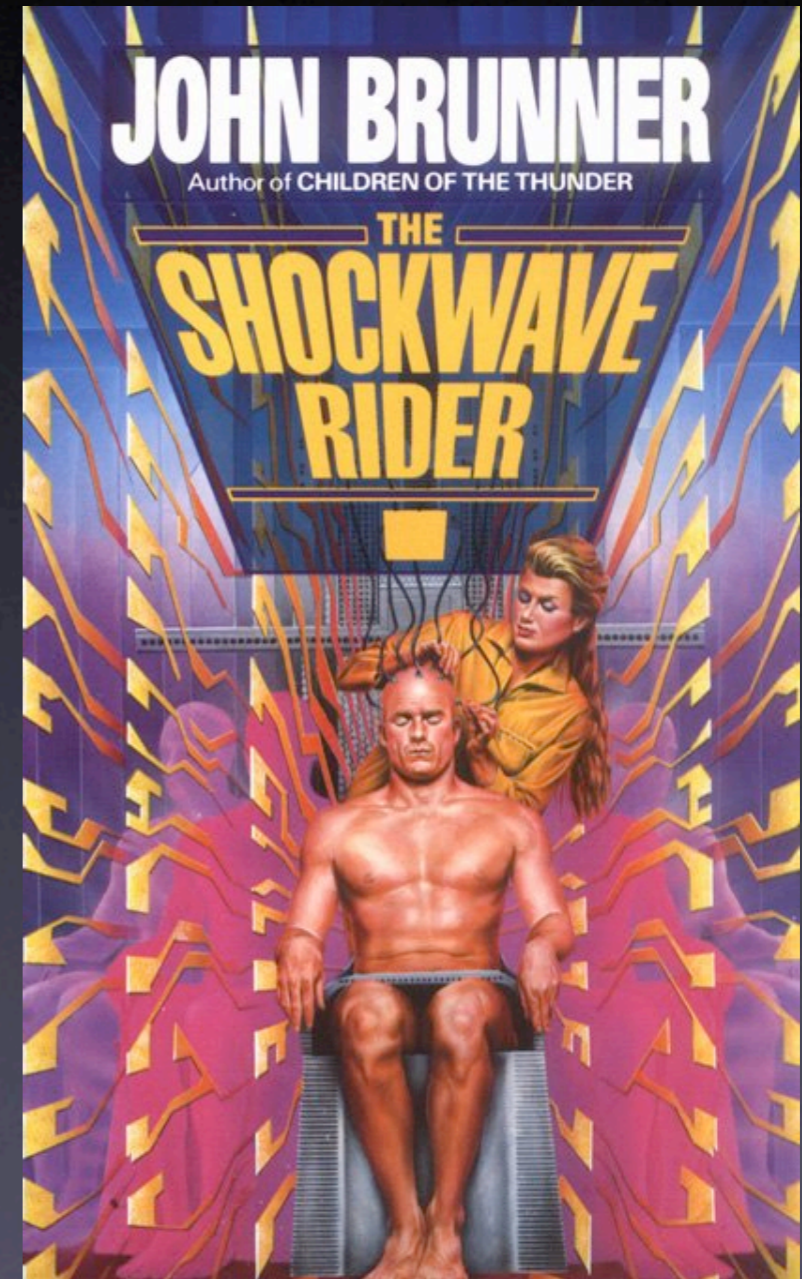
The Worm

I guess you all know about tapeworms... ? Good. Well, what I turned loose in the net yesterday was the.., father and mother of all tapeworms

My newest-my masterpiece-breeds by itself....

By now I don't know exactly what there is in the worm. More bits are being added automatically as it works its way to places I never dared guess existed

And-no, it can't be killed. It's indefinitely self-perpetuating so long as the net exists. Even if one segment of it is inactivated, a counterpart of the missing portion will remain in store at some other station and the worm will automatically subdivide and send a duplicate head to collect the spare groups and restore them to their proper place.



Characteristics

	Virus	Worm
preservation	propagation	migration
detectable	hard	easy
portability	independent	well suited on some program

History of Viruses

- 1949, von Neumann, *Theory and Organization of Complicated Automata*
- 1960, Robert Morris Sr. and others create game *Darwin* (aka *Core Wars*) at Bell Labs
- 1980, Authorities stop distribution of Jürgen Kraus' diploma thesis *Self-reproduction of Programs*

History of Viruses

- 1982, the first public computer virus *Elk Cloner* found in pirated games at Texas A&M University (<http://www.skrenta.com/cloner/>)
- 1983, first virus for UNIX computers by Fred Cohen (a program that can ‘infect’ other programs by modifying them to include a version [...] of itself)

History of Viruses

- 1986, *the Brain*, the first PC virus, infected the boot sectors of 360k floppies, full stealth - reason: monitoring piracy in Pakistan
- 1986, Ralf Burger presents the first COM infector *VIRDEM* at Chaos Communication Congress
- 1987, Ralf Burger publishes a book about viruses, including source code

History of Viruses

- in first issue of 29a: *“i think that the 'viruses of the future' will have support for spreading over a network. there are plenty of this possibilities under win*. * that can be implemented, and they *will* be implemented :)”*
- 1999, *Melissa*, the first e-mail virus

History of Worms

- 1988, Robert Morris Jr. wrote the *Morris Worm* (there might be a virus loose on the internet, Andy Sudduthm, Harvard)
- exploits in sendmail, fingerd, rsh/rexec and weak passwords
- DEC VAX, 4BSD and Sun 3
- 6000 infected hosts (~ 10 % of the internet)

History of Worms

- Several Linux worms in the past:
 - Adore (2001)
 - Linux.Ramen
 - Lion (2001) - exploit in BIND

Virus spreading

- Linux as desktop computers
- notorious roots
- mounting contaminated file systems (floppies, cds, **memory sticks**)
- network services
- harmful also for the single non-privileged user!

AT&T attack virus

- DOE virus database lists 2 UNIX viruses
 - one hoax
 - #950, called AT&T attack virus
- proof of concept, developed by researchers
- gained full access to the system in 30 min
- never found in the wild

Lotek by Wintermute

- 338 bytes ELF infection
- adds itself to the .note section
- test virus for a presentation at HackMeeting Barcelona '00
- *“to silent those ‘linux-never-can-get-infected’-people”*
- see 29a.5.801

Linux Viruses

- LoTek (harmless)
- PElf, multi platform (harmless)
- Rike (“non dangerous”)
- Bliss A, Bliss B
- Satyr
- Zipworm
- Diesel
- Nuxbee

Executable and Linkage Format

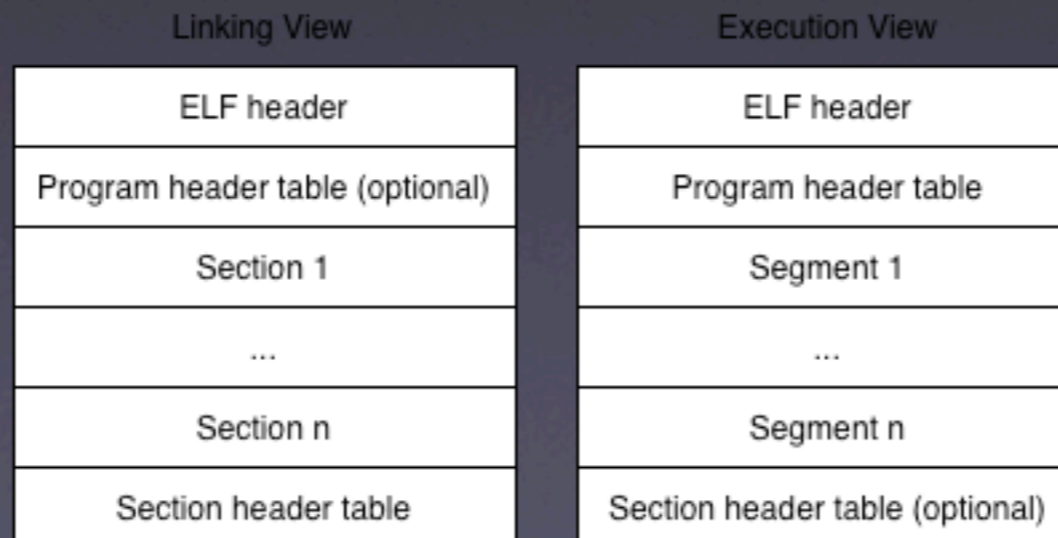
- developed at UNIX System Laboratories
- selected as portable object file format by Tool Interface Standard committee (IBM, SCO, Intel, Microsoft, ...)
- used by nearly all UNIX systems (except MacOS X)

Executable and Linkage Format

- 3 main types for object files
 - a relocatable file (.o) holds code and data suitable for linking with other object files
 - an executable (a.out) holds program suitable for execution
 - a shared object (.so) holds code and data suitable for linking

Executable and Linkage Format

- Linking view sections are stored in execution view segments



the ELF header

- an ELF header resides at the beginning of each file and holds a “road map” describing the organization of the file

```
typedef struct {
    unsigned char e_ident[EI_NIDENT];
    Elf32_Half    e_type;
    Elf32_Half    e_machine;
    Elf32_Word    e_version;
    Elf32_Addr    e_entry;
    Elf32_Off     e_phoff;
    Elf32_Off     e_shoff;
    Elf32_Word    e_flags;
    Elf32_Half    e_ehsize;
    Elf32_Half    e_phentsize;
    Elf32_Half    e_phnum;
    Elf32_Half    e_shentsize;
    Elf32_Half    e_shnum;
    Elf32_Half    e_shstrndx;
} Elf32_Ehdr;
```

the ELF header

```
Terminal — ssh
chris@untergrund:~ $ hexdump -C dnsfinger | head -n 5
00000000  7f 45 4c 46 01 01 01 09 00 00 00 00 00 00 00 00 |.ELF.....|
00000010  02 00 03 00 01 00 00 00 d8 85 04 08 34 00 00 00 |.....?...4...|
00000020  e0 13 00 00 00 00 00 00 34 00 20 00 06 00 28 00 |?.....4. ....|.
00000030  19 00 16 00 06 00 00 00 34 00 00 00 34 80 04 08 |.....4...4...|
00000040  34 80 04 08 c0 00 00 00 c0 00 00 00 05 00 00 00 |4...?...?...|
chris@untergrund:~ $
```

the ELF header

```
Terminal — ssh
chris@untergrund:~ $ hexdump -C dnsfinger | head -n 5
00000000  7f 45 4c 46 01 01 01 09  00 00 00 00 00 00 00 00  |.ELF.....|
00000010  02 00 03 00 01 00 00 00  d8 85 04 08 34 00 00 00  |...?..4...|
00000020  e0 13 00 00 00 00 00 00  34 00 20 00 06 00 28 00  |?.....4. ...|.
00000030  19 00 16 00 06 00 00 00  34 00 00 00 34 80 04 08  |.....4..4...|
00000040  34 80 04 08 c0 00 00 00  c0 00 00 00 05 00 00 00  |4...?..?.....|
chris@untergrund:~ $
```

the ELF header

```
Terminal — ssh
chris@untergrund:~ $ hexdump -C dnsfinger | head -n 5
00000000  7f 45 4c 46 01 01 01 09  00 00 00 00 00 00 00 00 |.ELF.....|
00000010  02 00 03 00 01 00 00 00  d8 85 04 08 34 00 00 00 |.....?..4..|
00000020  e0 13 00 00 00 00 00 00  34 00 20 00 06 00 28 00 |?.....4. ....|
00000030  19 00 16 00 06 00 00 00  34 00 00 00 34 80 04 08 |.....4..4..|
00000040  34 80 04 08 c0 00 00 00  c0 00 00 00 05 00 00 00 |4...?...?.....|
chris@untergrund:~ $
```

ELF section header

```
// from /usr/include/sys/machine/elf32.h
typedef struct {
    Elf32_Word    sh_name;
    Elf32_Word    sh_type;
    Elf32_Word    sh_flags;
    Elf32_Addr    sh_addr;
    Elf32_Off     sh_offset;
    Elf32_Word    sh_size;
    Elf32_Word    sh_link;
    Elf32_Word    sh_info;
    Elf32_Word    sh_addralign;
    Elf32_Word    sh_entsize;
} Elf32_Shdr;
```

ELF sections

- `.bss` - uninitialized data
- `.data` - initialized data
- `.got` - global offset table
- `.note` - compatibility information
- `.text` - instructions
- `.strtab` - string table
- `.symtab` - symbol table

```
typedef struct {  
    Elf32_Word    sh_name;  
    Elf32_Word    sh_type;  
    Elf32_Word    sh_flags;  
    Elf32_Addr    sh_addr;  
    Elf32_Off     sh_offset;  
    Elf32_Word    sh_size;  
    Elf32_Word    sh_link;  
    Elf32_Word    sh_info;  
    Elf32_Word    sh_addralign;  
    Elf32_Word    sh_entsize;  
} Elf32_Shdr;
```

ELF program header

```
// from /usr/include/machine/elf32.h
typedef struct {
    Elf32_Word    p_type;
    Elf32_Off     p_offset;
    Elf32_Addr    p_vaddr;
    Elf32_Addr    p_paddr;
    Elf32_Word    p_filesz;;
    Elf32_Word    p_memsz;
    Elf32_Word    p_flags;
    Elf32_Word    p_align;
} Elf32_Phdr;
```

ELF segments

“A program header table, if present, tells the system how to create a process image.”

- a segment contains one or more sections
- PT_LOAD is most interesting for us

ELF attacks

- all 3 types of ELF files are attackable
 - especially executables
- known attacks
 - text or data segment infection
 - injection into alignment
 - PLT infection

Virus in own segment

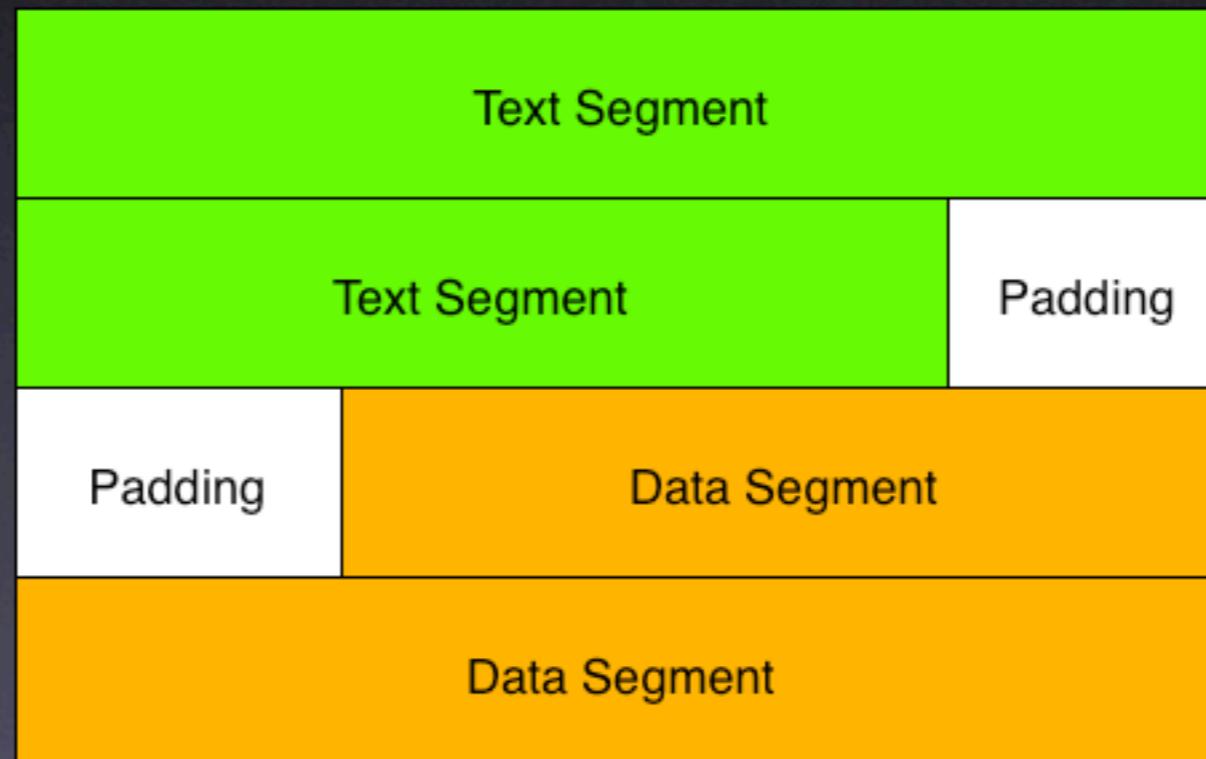
- LoTek is stored in .note segment
- entry point is redirected

text segment infection

- direct injection into machine code
- problems:
 - extending backwards: headers in text segment might have to move
 - extending forwards: segments might overlap
 - new segment: even easier to spot
- solution: page padding at segment borders

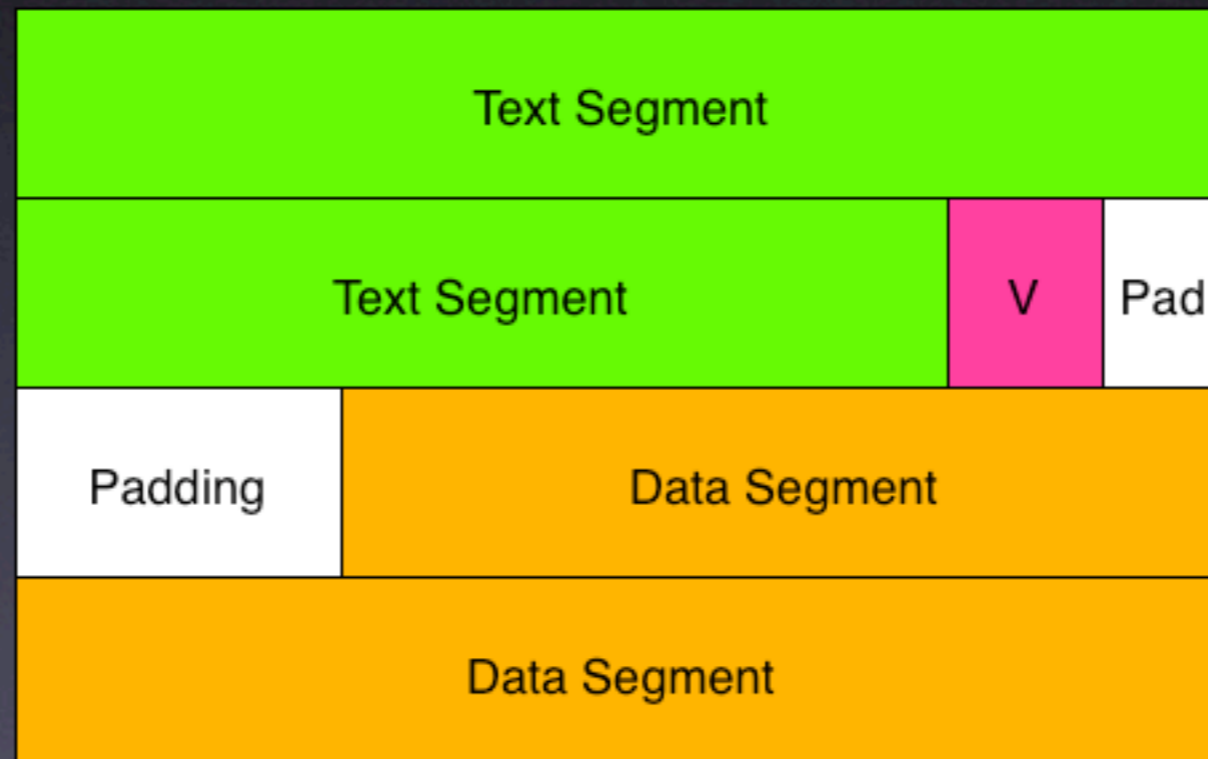
text segment infection

layout of an ELF executable



text segment infection

layout of an ELF executable



text segment infection

- writing viral code
 - can be done in C (not only ASM)
 - no special limitations (zero bytes, ...)
- infection routine
 - should not take too long
 - should avoid fork

text segment infection

- how to get viral code into code flow
 - new entry point := `phdr.v_addr + phdr.p_filesze`
 - after execution, jump to original entry point
 - alternative: hook into `_start`

text segment infection

“Some object file control structures can grow, because the ELF header contains their actual size.”

text segment infection

- getting code into executable
 - update various length fields, p_shoff in Elf32_Hdr, p_filesz, p_memsz in Elf32_Phdr
 - update various offsets, p_offset in Elf32_Phdr, sh_offset in Elf32_Shdr
- $p_vaddr \% \text{PAGESIZE} \approx p_offset \% \text{PAGESIZE}$

text segment infection

- new code is not associated with any section
 - suspicious
 - not strip safe

text segment infection

Full algorithm:

- Increase `e_shoff` (by `PAGESIZE`) in ELF header
- Locate the text segment program header
 - Increase `p_filesz`
 - Increase `p_memsz`
 - Patch the entry point
- Increase `sh_lein` in last section header in text segment
- For each `shdr` (`phdr`) who's section is after the insertion
 - Increase `sh_offset` (`ph_offset`)
- Insert the hostile code into the file

data segment infection

- just the same

hiding code in instruction alignment

- put instructions into alignment between functions
 - usually between 0 and 15 bytes
 - GCC: instruction padding
- JMP after each injection
- relocation needed
- only code injection, no data

hiding code in instruction alignment

- finding free areas in executables
 - find JMP or RET instruction that is followed by padding
- use pattern matching
- use infelf (FreeBSD)

hiding code in instruction alignment

Example:

```
C3      retn                ; end of function
90      nop                ; alignment
90      nop
90      nop
55      push %ebp          ; function prologue
8B EC  movl %esp, %ebp    ; of next function
```


Procedure Linkage Table

```
.PLT0:pushl 4(%ebx)
      jmp  *8(%ebx)
      nop; nop
      nop; nop
.PLT1:jmp  *name1@GOT(%ebx)
      pushl $offset
      jmp  .PLT0@PC
.PLT2:jmp  *name2@GOT(%ebx)
      pushl $offset
      jmp  .PLT0@PC
      ...
```

redirects position-independant function calls to absolute addresses

PLT infection

- mark the text segment writable
- save the PLT entry
- replace PLT entry with viral library call

PLT infection

- execute the viral library function
- restore the original PLT entry
- call the library function
- if PLT changed, save entry again
- replace the entry again

Resources

- 29a Labs E-Zine: <http://vx.netlux.org/29a>
- 40hex: <http://www.etext.org/CuD/40hex/>
- VX Heavens: <http://vx.netlux.org/>
- Silvio Cesare: <http://web.archive.org/web/20031129164118/www.big.net.au/~silvio/>
- <http://www.lwfug.org/~abartoli/virus-writing-HOWTO/>
- Little Black Book of Computer Viruses

Slides

- <http://c0re.23.nu/~chris/presentations/Virus-2005.pdf>
- Also check <http://felinemenace.org/~mercy/public/ruxcon/ELF%20fairytale.pdf>

Thank you!