

PASTOR MANUL LAPHROAIG's  
INTERNATIONAL JOURNAL OF  
PoC || GTFO,  
CALISTHENICS & ORTHODONTIA  
IN REMEMBRANCE OF  
OUR BELOVED DR. DOBB  
BECAUSE  
THE WORLD IS ALMOST THROUGH!



March 19, 2015

7:2 AA55, the Magic Number

7:3 Laser robots!

7:4 A Story of Settled Science

7:5 Scapy is for Script Kiddies

7:6 Funky Files, the Novella!

7:7 Extending AES-NI Backdoors

7:8 Innovations with Core Files

7:9 Bambaata on NASCAR

7:11 A Modern Cybercriminal

7:12 Fast Cash for Bugs!

---

Heidelberg, Baden-Württemberg:

Funded by Single Malt as Midnight Oil and the  
Tract Association of PoC||GTFO and Friends,  
to be Freely Distributed to all Good Readers, and  
to be Freely Copied by all Good Bookleggers.



Это самиздат; therefore, go ye into all the world, and preach the gospel to every creature!  
€0, \$0, £0. pocorgtfo07.pdf.

**Legal Note:** This telecast is copyrighted by the NFL for the private use of our audience. Any other use of this telecast or of any pictures, descriptions, or accounts of the game without the NFL's consent, is prohibited. Just kidding!

**Reprints:** Bitrot will burn libraries with merciless indignity that even Pets Dot Com didn't deserve. Please mirror--don't merely link! [pocorgtfo07.pdf](#) and our other issues far and wide, so our articles can help fight the coming robot apocalypse.

**Technical Note:** This issue is a polyglot that can be meaningfully interpreted as a ZIP, a PDF, a BPG, or HTML featuring a BPG decoder. We no longer include prior issues in the zip, in order to leave room for more curiosities. Don't be surprised when you stumble upon occasional polyglot матрёшки and chimeras.

**Dedication:** This issue is dedicated to Terry Pratchett, R.I.P.

"I meant," said Ipslore bitterly, "what is there in this world that makes living worthwhile?"  
Death thought about it.  
CATS, he said finally. CATS ARE NICE.

**Printing Instructions:** Pirate print runs of this journal are most welcome, but please do it properly! PoC||GTFO is to be printed duplex, then folded and stapled in the center. Print on A3 paper in Europe and Tabloid (11" x 17") paper in Samland. Secret government labs in Canada may use P3 (280 mm x 430 mm) if they like. The outermost sheet should be on thicker paper to form a cover.

```
1 # This is how to convert an issue for duplex printing.
2 sudo apt-get install pdfjam
3 pdfbook --short-edge pocorgtfo07.pdf -o pocorgtfo07-booklet.pdf
```

Preacherman	Manul Laphroaig
Ethics Advisor	The Grugq
Poet Laureate	Ben Nagy
Editor of Last Resort	Melilot
Carpenter of the Samizdat Hymnary	Redbeard
Funky File Formats Polyglot	Ange Albertini
Assistant Scenic Designer	Philippe Teuwen
Special Correspondent on NASCAR	Count Bambaata
Minister of Spargelzeit Weights and Measures	FX

# 1 With what shall we commune this evening?

Neighbors, please join me in reading this eighth release of the International Journal of Proof of Concept or Get the Fuck Out, a friendly little collection of articles for ladies and gentlemen of distinguished ability and taste in the field of software exploitation and the worship of weird machines. If you are missing the first seven issues, we the editors suggest pirating them from the usual locations, or on paper from a neighbor who picked up a copy of the first in Vegas, the second in São Paulo, the third in Hamburg, the fourth in Heidelberg, the fifth in Montréal, the sixth in Las Vegas, or the seventh from his parents' inkjet printer during the Thanksgiving holiday.

We begin our show tonight in Section 2 with something short and sweet, an executable poem by Morgan Reece Phillips. Funny enough, 0xAA55 is also Pastor Laphroaig's favorite number!

We continue in Section 3 with another brilliant article from Micah Elizabeth Scott. Having bought a BD-RW burner, and knowing damned well that a neighbor doesn't own what she can't open, Micah reverse engineered that gizmo. Sniffing the updater taught her how to dump the firmware; disassembling that firmware taught her how to patch in new code; and, just to help the rest of us play along, she wrapped all of this into a fancy little debugging console that's far more convenient than the sorry excuse for a JTAG debugger the original authors of the firmware most likely used.

In Section 4, Pastor Laphroaig warns us of the dangers that lurk in trusting The Experts, and of one such expert whose witchhunt set back the science of biology for decades. This article is illustrated by Boris Efimov, may he rot in Hell.

In Section 5, Eric Davisson describes the internals of TCP/IP as a sermon against the iniquity of the abstraction layers that—while useful to reduce the drudgery of labor—also cloud a programmer's mind and keep him from seeing the light of the hexdump world.

Ange Albertini is known to our readers for short and sweet articles that quickly describe a clever polyglot file in a page or two. In Section 6, he finally presents us with a long article, a listing of dozens of nifty tricks that he uses in PoC||GTFO, Corkami, and other projects. Study it carefully if you'd like to learn his art.

In Section 7, BSDaemon and Pirata extend the RDRAND trick of PoC||GTFO 3:6—with devilish cunning and true buccaneer daring—to actual Intel hardware, showing us poor landlubbers how to rob not only unsuspecting virtual machines but also normal userland and kernel applications that depend on the new AES-NI instructions of their precious randomness—and much more. Quick, hide your AES! Luckily, our neighborly pirates show how.

Section 8 introduces us to Ryan O'Neill's Extended Core File Snapshots, which add new sections to the familiar ELF specification that our readers know and love.

Recently, Pastor Laphroaig hired Count Bambaata on as our Special Correspondent on NASCAR. After his King Midget stretch limo was denied approval to compete at the Bristol Motor Speedway, Bambaata fled to Fordlandia, Brazil in a stolen—the Count himself says “liberated”—1957 Studebaker Bulletnose in search of the American Dream. When asked for his article on the race, Bambaata sent us by WEFAAX a collection of poorly redacted expense reports<sup>1</sup> and a lovely little rant on Baudrillard, the Spirit of the 90's, and a world of turncoat swine. You can find it in Section 9.

Section 11 is the latest from Ben Nagy, a peppy little parody of Hacker News and New-Media Web 2.0 Hipster Fashion Accessorized Cybercrime in the style of Gilbert and Sullivan. Sing along, if you like!

Finally, in Section 12 we do what churches do best and pass around the old collection plate. We don't need alms of Dollars or Euros, so send those to Hackers for Charity in Uganda.<sup>2</sup> Rather, we pass the plate to ask for your doodles and your sketches, your crazy ideas that work well enough to prove the concept, well enough to light up the mind, well enough to inspire the next lady or gentleman to do something clever and strange.

---

<sup>1</sup>Bambaata, if you're reading this, please call me. Your Amex is beyond its limit after you expensed two “Charlie Miller kitchens,” and we had to reject payment in the amount of \$20,000 USD to “You Better Belize It Bail Bonds.” Oh, and if by chance you happen to be arrested in Brazil, please ask the Federales when the impounded H2HC 2013 conference badges will appear on Ebay. —PML

<sup>2</sup>This isn't a joke, and we're not being snarky. Send money to HFC.

## 2 The Magic Number: 0xAA55

by Morgan Reece Phillips

```
1 [org 0x7c00]           ; make nasm aware of the boot sector offset
3 mov bp, 0x8000          ; move the base of the stack pointer beyond the boot sector offset
5 mov sp, bp              ; move the top and bottom stack pointers to the same spot
7 mov bx, poem
9 call print_str
jmp $                     ; loop forever
11 print_str:             ; define a print "function" for null terminated strings
13   mov al, [bx]
15   cmp al, 0
17   je the_end
19   mov ah, 0x0e           ; set up the scrolling teletype interrupt
21   int 0x10               ; call interrupt handler
23   add bx, 0x1
25   jmp print_str
the_end:
27   ret
29
31 poem:
33   db 0xA, 0xD, \
35   '/*****', \
37   0xA, 0xD, \
39   '** The Magic Number: 0xAA55', \
41   0xA, 0xD, \
43   '*****', \
45   0xA, 0xD,
47   'A word gives life to bare metal', \
49   0xA, 0xD,
51   'Bytes inviting execution', \
53   0xA, 0xD,
55   'Guide to a sector to settle', \
57   0xA, 0xD,
59   'The bootloader', 0x27, 's role is vital', \
61   0xA, 0xD,
63   'Denoted by its locution—', \
65   0xA, 0xD,
67   'A word gives life to bare metal', \
69   0xA, 0xD,
71   'Bytes inviting execution', \
73   0xA, 0xD,
75   '// @linuxpoetry (linux-poetry.com)', \
77   0
79
81 times 510-($-$) db 0    ; write zeros to the first 510 bytes
83 dw 0xaa55                ; write the magic number
```

An MBR/ASM/PDF polyglot variant made by the usual suspects is available in this very polyglot PDF.

### 3 Coastermelt

by Micah Elizabeth Scott

#### 3.1 Getting Inside Your Optical Drive's Head

This is the first of perhaps several articles on the adventures of `coastermelt`, an art-hacking project with the goal of creating cheap laser graffiti using discs burned by Blu-Ray drives with hacked firmware.

##### 3.1.1 Art Hacking Manifesto

If an engineer is a problem solver, hackers and artists are more like problem tinkerers. Some of the most interesting problems are so far beyond the scope of any direct solution that it seems futile to even approach them head-on. It is the artist's purview to creatively approach these problems, sideways or upside down if necessary.

When an engineer is paid to make a tool, is it not the money itself that ultimately decides the tool's function? I believe that to be a hacker is to see tools as things not only to make but to re-make and subvert. By this creative reapplication of technology, research and problem-solving need not be restricted to those who own the means of production.

So says the Maker's manifesto: if you can't open it, you don't own it. I'd like to build on this: if we work together to open it, we all own it. And maybe we can all learn something along the way.

##### 3.1.2 I heard there were laser robots?

Why yes, laser robots! Optical discs may be all but dead as a data storage medium, but the latest BD-RW drives contain feats of electromechanical engineering that leave any commercial 2D or 3D printer in the dust. Using a 405 nm laser, they can create marks only 150 nm long, with accuracy better than 70 nm. Tiny lenses mounted on a fast electromagnetic suspension can keep perfect focus on grooves only 320 nm apart as the disc spins at over 7 m/s.

A specialized system-on-chip generates motor and laser control signals, amplifies and demodulates the light signals captured by a photodiode array, and it does all of this in the service of fairly pedestrian tasks like playing motion pictures and making backups of cat photos.

My theory is that, with quite a lot of effort, it would be possible to create new firmware for a common Blu-Ray burner such that we could burn discs with arbitrary patterns. Instead of the modulated binary data that stays nicely separated into the tracks of a spiral groove, I think we can treat the whole disc surface as a canvas to draw on with sub-100 nm precision.

If this works, it should be possible to create patterns fine enough that they diffract interestingly under red laser illumination. By bouncing a powerful laser pointer off of a specially burned BD-R disc and targeting a flat surface, perhaps we can control the shape of the eventual illumination well enough to project words or symbols.

This is admittedly a very long shot. Perhaps the patterns have nowhere near enough resolution. Perhaps the laser pointer would need to be much too powerful. If this works out, I dream of creating a mobile printing press for light graffiti. If not, I suspect the project may still lead somewhere interesting.

##### 3.1.3 Device Under Test

For `coastermelt` I chose the Samsung SE-506CB optical drive, a portable USB 2.0 burner that's currently quite popular. It retails for about \$80. Inside, I found an MT1939 SoC, an undocumented and highly application-specific chip from MediaTek. It was easy to find some firmware updates which became a starting point for understanding this complicated black box.

My current understanding is that the MT1939 contains a pokey ARM7 processor core along with a lot of strange application-specific peripherals and about 4 MB of RAM. There's also an 8-bit 8051 processor core

in there, which shares access to the USB controller. The USB software stack seems to be confusingly split between the ARM firmware and the tiny 8051 firmware, for still-unknown reasons.

There are two customized and undocumented motor control chips from TI, which drive a stepper motor, brushless motor, and the voice coils that quickly position and focus the lenses. As far as I can tell, these chips just act as high-power load drivers. All of the logic and timing seems to be within that MT1939 chip.

### 3.1.4 How did we get here anyway?

This has been a complex journey full of individual hacks that could each make an interesting story. In my experience, reverse engineering is much like playing a point-and-click or text adventure game. There's a huge world to explore, and so much of your time can be spent on probing the boundaries of that world, understanding who the characters are and what their motivations are, and suffering through plenty of enlightening but frustrating dead-ends.

I wanted to share this process as best I could, in a way that could be documentation for the project, an educational peek into the world of reverse engineering, and an invitation to collaborate. I created a video series<sup>3</sup> with two episodes so far. I won't repeat those stories here; let's go somewhere new.

### 3.1.5 Down the Rabbit Hole

If you take the blue pill, the story ends, and you wake up believing your optical drives only accept standard SCSI commands that read and write data according to the established MMC specifications.

Of course, that is a convenient fairy tale. Firmware updates exist, and so we know the protocol must be Turing-complete already. In this tiny world, our red pill is a patched firmware image that adds a backdoor<sup>4</sup> with enough functionality to implement a simple debugger. After installing the patch,<sup>5</sup> we can go in:

```
backdoor micah$ ./cmshell.py
```

```
-- .-----| | |-----| | |-----| | |-----| | |-----| |
| _ _| _ _| _ _| _ _| _ _| _ _| _ _| _ _| _ _| _ _| _ _|
|_ _|_ _|_ _|_ _|_ _|_ _|_ _|_ _|_ _|_ _|_ _|_ _|_ _|_ _|
--IPython Shell for Interactive Exploration-----
```

Read, write, or fill ARM memory. Numbers are hex. Trailing \_ is short for 0000, leading \_ adds 'pad' scratchpad RAM offset. Internal \_ are ignored so you can use them as separators.

```
rd 1ff_ 100
wr _ 1febb
ALSO: rdw, wrb, fill, watch, find
      bitset, bitfuzz, peek, poke, read_block
```

Disassemble, assemble, and invoke ARM assembly:

```
dis 3100
asm _4 mov r3, #0x14
dis _4 10
ea mrs r0, cpsr; ldr r1, =0xaa000000; orr r0, r1
ALSO: tea, blx, assemble, disassemble, evalasm
```

<sup>3</sup><https://vimeo.com/channels/coastermelt>

<sup>4</sup><https://github.com/scanlime/coastermelt>

<sup>5</sup>There's a Getting Started section in the README that should help.

```
Or compile and invoke C++ code with console output:
```

```
ec 0x42
ec ((uint16_t*)pad)[40]++
ecc println("Hello World!")
ALSO: console, compile, evalc
```

```
Live code patching and tracing:
```

```
hook -Rrcm "Eject button" 18eb4
ALSO: ovl, wrf, asmf, iwt
```

```
You can use integer globals in C++ and ASM snippets,
or define/replace a named C++ function:
```

```
fc uint32_t* words = (uint32_t*) buffer
buffer = pad + 0x100
ec words[0] += 0x50
asm _ ldr r0, =buffer; bx lr
```

```
You can script the device's SCSI interface too:
```

```
sc c ac          # Backdoor signature
sc 8 ff 00 ff    # Undocumented firmware version
ALSO: reset, eject, sc_sense, sc_read, scsi_in, scsi_out
```

```
With a hardware serial port, you can backdoor the 8051:
```

```
bitbang -8 /dev/tty.usb<tab>
wx8 4b50 a5
rx8 4d00
```

```
Happy hacking!      -- Type 'thing?' for help on 'thing' or
~MeS'14           '??' for IPython, '%h' for this again.
```

```
In [1]:
```

Such a strange debugger! At a basic level everything works by *peek* and *poke* in memory with the occasional *call*. The shell is based on the delightful IPython, with commands for easy inline C++ and assembly code. Integer variables and register values are bridged across languages when possible.

### 3.1.6 GO NORTH; LOOK

You have entered a console full of strange commands. The CPU seems to be an ARM. You don't know what it's doing now, but it runs your commands when asked. Before you appears a vast 32-bit address space, mostly empty.

You happen to see a note on the ground, a splotchy Hilbert curve napkin sketch followed by a handwritten table of hexadecimal numbers with uncertain names scrawled nearby.

Flash, 2 MB	00000000 - 001fffff
... write-protected bootloader, 64 kB	00000000 - 0000ffff
... loadable, 1863 kB	00010000 - 001e1fff
... storage, 120 kB	001e2000 - 001fffff
DRAM, 4 MB	01c08000 - 02007fff
MMIO	04000000 - 043fffff

You can peek around at memory, and things seem to be as they appear for the most part. The flash memory can be read and disassembled, interrupt vectors pointing to code that can unfurl into many hours of disassembly and head-scratching. DRAM at this point is like a ghost town, plenty of space to build scaffolding or conduct science.

```
In [1]: ea mov r0, pc; mov r1, sp
r0 = 0x01e4000c, r1 = 0x0200067c
```

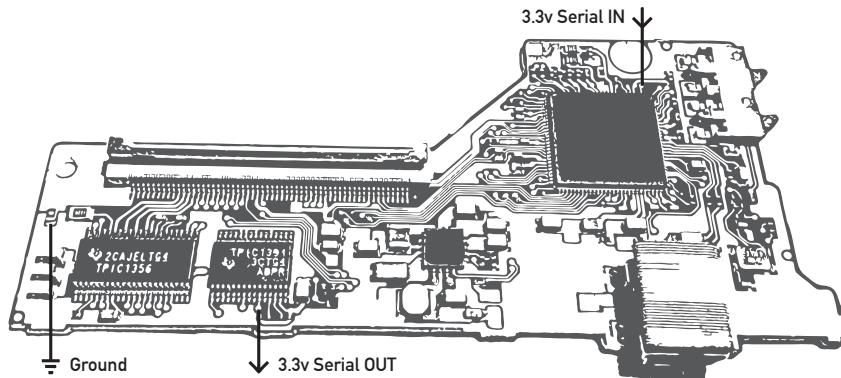
```
In [2]: rdw 2000067c 30
0200067c 01000000 01e40000 01ffc290 00000007 0000000d 01ffc2a8 0004bad7 00000000
0200069c 01ffc290 02000cf8 01ffc290 02000cf8 0001efa9 00000000 00000000 02000cdc
020006bc 01ffb76c 02000c0e 0001ec2f 00000000 02000cdc 01ffb76c 00018c07 00000000
020006dc 00018e31 00000032 02000cdc 00167558 00000000 00000000 00000000 00000000
020006fc 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0200071c 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Using some inline assembly, we find the program counter and stack pointer, and separately we dump the memory where the top of the stack was. These can't tell us what the firmware would have been doing had we not rudely interrupted with our backdoor, but these are breadcrumbs showing us some of the steps the firmware took just before we intervened.

### 3.1.7 30 Gauge Enamel-Coated Freedom

Direct physical access is of course the ultimate hacking tool. With the USB backdoor we can send the ARM processor cutesy little notes asking it or even daring it to run instructions for us, but this will end in heartbreak if we expect to hold the CPU's attention for longer than one fleeting SCSI command.

Heartbreak is a complicated thing though, sometimes it can act like a forest fire leaving the ground fertile for fresh inspiration. If the ARM and the SCSI driver were to never speak again, how could we still contact the ARM? This is where we need to warm our soldering irons. If there's blue wire there's a way. Let's add a serial port for the next step.



### 3.1.8 GET WALKTHROUGH

In the first `coastermelt` video, I got as far as using this serial port to build an alternate debug backdoor that can break free from the control flow in the original firmware.

```
In [1]: bitbang -8 /dev/tty.usbserial-A400378p
* Handler compiled to 0x2e8 bytes, loaded at 0x1e48000
* ISR assembled to 0xdc bytes, loaded at 0x1e48300
* Hook at 0x18ccc, returning to 0x18cce
* RAM overlay, 0x8 bytes, loaded at 0x18ccc
* Connecting to bitbang backdoor via /dev/tty.usbserial-A400378p
* Debug interface switched to <bitbang.BitbangDevice instance at 0x102979998>
  305 / 305 words sent
* 8051 backdoor is 0xef bytes, loaded at 0x1e49000
* ARM library is 0x3d4 bytes, loaded at 0x1e490f0
* 8051 backdoor running
```

In the second video, I introduced a CPU emulator that can run the ARM firmware on your host computer, proxying all I/O operations back to the debug backdoor while of course logging them.

```
In [2]: sim
235 / 235 words sent
* Installed High Level Emulation handlers at 01e00000
- initialized simulation state
[INIT] .....0 ---- >00000000      ldr      pc, [pc, #24]
r0=00000000  r4=00000000  r8=00000000  r12=00000000
r1=00000000  r5=00000000  r9=00000000  sp=00000000
r2=00000000  r6=00000000  r10=00000000 lr=fffffff
r3=00000000  r7=00000000  r11=00000000 pc=00000000
```

Now we can follow in the normal firmware's footsteps, mapping out the tiny islands of I/O scattered through this sea of memory addresses. As the `%sim` command churns away, every instruction and memory access shows up in `trace.log`. In the video you can see a demo where a properly arranged replay of these register writes can trigger motor movement.

This trace log is like a walkthrough, showing us exactly how the normal firmware would use the hardware. It's helpful, but certainly not without its limitations. There's so much data that it takes some clever filtering to get much out of it, and it's quite slow to run the simulation. It's a starting point, though, and it can offer clues and memory addresses to use in other experiments with other tools.

At this point in the project, we have some basic implements of cartography, but there isn't much of a map yet. Do you like exploring? I have the feeling there's some really neat stuff in here. With so much interesting hardware to map out, there's enough adventure to share. Take an interesting journey, and be sure to tell us what you find.

## 4 Of Scientific Consensus and a Wish That Came True

a sermon by Pastor Manul Laphroaig

*Every now and then we see some obvious bullshit being peddled under the label of science, and we wish, couldn't we just put a stop to this? This bullshit is totally not in the public interest—and isn't the government supposed to look after the public interest? Wouldn't it be nice if the government shut these charlatans down?*

*This is the story of a science community that had had this wish come true.*

Once upon a time in a country far far away there was an experimental scientist who managed to solve a number of important real-world problems, or at least managed to convince himself and many other scientists that he did. His work brought journalists to otherwise unexciting scientific conferences and made headlines across the world.<sup>6</sup> He might have ended up in history as a talented experimentalist who challenged contemporary theories to refine themselves by sticking them with examples they didn't quite cover. As his luck would have it, though, he came of age in the time and place where scientific debates were being settled by majority votes and government action.

It so happened that the government of that country was very pro-science. They took to heart the stories of scientists being kept back by ignorant retrogrades and charlatans throughout history, and they would have none of that. They were out to give science the support and protection it deserved, and they looked to it to solve practical problems. So they took a keen interest, and, being well-educated and versed in the scientific method as they were, trusted themselves to tell a true scientific theory from an obviously erring one.

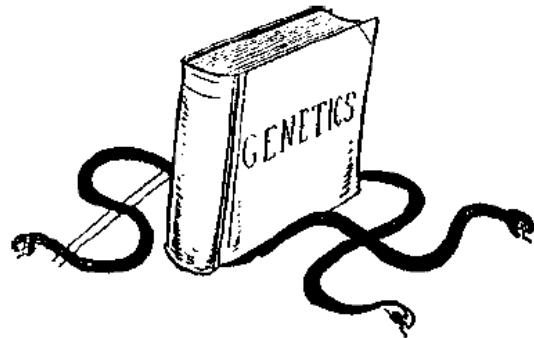
Since scientists continually find themselves in bitter debates, this ability was extremely useful. They had the power to settle such debates to reap all the rewards of having the right science and to stop those scientists in the wrong from wasting people's time and resources. Sometimes the power had to stop them the hard way, to protect the impressionable youth who could otherwise be misled by complicated arguments; but that was all right because, once the debate is settled, isn't it one's duty to protect the young 'uns from harmful influences with all the means at hand?

So our up-and-coming scientist did the right thing: he petitioned the government to suppress the erring opposition, citing his experimental successes and the opposition's failures, obvious waste of effort, and conflicts of interest. Besides his successes, he built a strong moral case against his opponents: while

his school showed exactly how to produce broad impacts for the benefit of humanity, the others mostly proclaimed that the result of any direct human efforts would be at best uncertain, that the current state of Nature might be really hard to change, and yet that humans were rather powerless against its accidental changes.

Clearly, such interpretations of science were perversions that couldn't be tolerated. Moreover, the immediate implications of the opponents' theories obviously benefited the worst political actors of the age—and guess who funded the bulk of their so-called science? The very same regressive forces that sought to forestall Social Progress! Of course, not all of the opposition was knowingly in their pay, but shouldn't Real Scientists know better anyway, especially when the majority has had its say? Surely they have had enough notice.

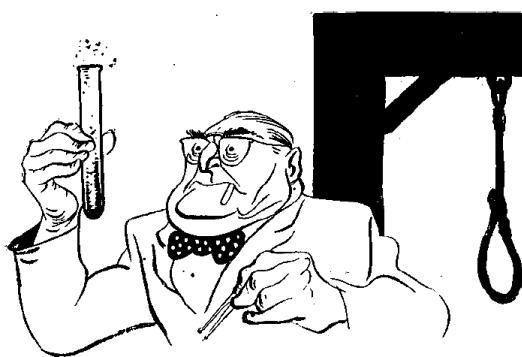
The name of our scientist was Trofim Denisovich Lysenko. The reactionary pseudo-science in the sights of his and his hard-won scientific majority's rightful wrath: so-called *Genetics*. The place was the Soviet Union, 1936–48.



More precisely, it was the Mendelian theory of heredity based on genes, the so-called Weismannism–Morganism. That theory postulated that genes governed heredity, mutated unpredictably under factors such as radiation, and that mutations were hard to

<sup>6</sup>You'll find one such headline from the New York Times on the page 12.

direct for human purposes such as creation of new useful breeds of plants and animals. That was, of course, scandalous: didn't Marxist science already assert that environment was solely responsible for shaping all essential characteristics of life? Surely this "fear and doubt" approach of genetics that proclaimed all human beings to be carriers of countless hopeless mutations did not belong in the world of progressive sciences.



This theory was merely re-arming the racists and eugenacists, intent on suppressing the lower classes!



It was obvious that this "science" was in fact pure fascism, not matter how desperately it tried to distance itself from such anti-science atavisms.



And all of this was under the banner of "pure science", even though obviously financed by and serving the interests of the imperialist ruling class!



There is an old word for what happens when science becomes settled by majority, and the settlement gets enforced by the government. This good old word is *Inquisition*.

Inquisition got started to protect the lay people from destructive ideas that any learned person at the time would easily recognize as false, such as that "witches" could somehow interfere with crops and flocks. It eventually sought the power of the government to enforce its verdicts and to curb the charlatans from confusing those of little knowledge. It got what it sought, and the rest is history. Which, of course, tends to repeat itself.

# FINDS WAY TO CREATE MORE FOOD PLANTS

**Dr. Lyssenko, Russian, Crosses Varieties Having Different Periods of Vegetation.**

## WIDE EFFECT IS FORESEEN

**Tropical Products Now Can Be Grown in North, the Genetics Session at Ithaca Is Told.**

## EVOLUTION SEEN AS CURBED

**Haldane Says Man's Chance Is Not as Favorable Now as When He Lived in Small Tribes.**

**By WILLIAM L. LAURENCE.**

Special to THE NEW YORK TIMES.

ITHACA, N. Y., Aug. 30.—A new discovery in plant breeding, regarded as epoch-making, which permits growing of tropical and sub-tropical fruits in northern climates and allows crossing of varieties of seeds requiring entirely different periods of vegetation was announced at Cornell University today during the last general session of the International Congress of Genetics.

This discovery, which opens the way for creation of many new varieties of fruits and other foods, was made recently in Odessa, Russia, by Dr. T. D. Lyssenko, and was reported here by Dr. N. I. Vavilov, director of the Institute of Plant Industry in Leningrad.

The process involves a relatively simple treatment of the seed before planting and will make it possible to raise such tropical fruits as alligator pears and bananas, and such semi-tropical fruits as grapefruit, oranges and lemons in New York State, New England and the Middle West. It may, if practiced on a large scale, have incalculable economic significance, in view of the fact that States like California and Florida and many tropical and sub-tropical countries have developed such large industries around their fruit products.

New York Times report from the sixth International Congress of Genetics (1932) in Ithaca, NY.

All cartoons in this sermon are by one Boris Efimov, who started his long career in Party Art by lauding Trotsky, then glorifying Stalin and calling for summary executions of "Trotskyite dogs" (which included his brother), did his humble bit in promoting first the heroic Soviet political police in 1930s, and then the "Soviet peace initiatives" and "Soviet democracy" throughout the 1960s and 70s, denouncing the imperialists and the wavering.



*The Great Captain leads us from Victory to Victory!*

One of his last commissions (he was over 85), was to ridicule both those who clamored to speed up Gorbachov's "Perestroika" and those showing too much caution in conducting it—because the right way was to go in lockstep with the Party. (Just like he did in 1987, drawing pig-like Deniers of Lawless Terror worshiping the Great Captain's blood-spattered idol.) When the Party's power ended, he complained that "political cartooning didn't exist anymore."

He passed away in 2008, a paragon of sticking to just the prescribed amount of murderous blood-thirstiness at any given time, a true knight of the Party Line—and, if there is ever a Hell, doubtlessly sticking Hell's engineers with the problem of how to reward such a sterling life achievement of toeing it ever so precisely. There are many shitty jobs in this world and the one beyond, but, believe in Hell or not, that one takes the cake.



*Efimov's Trotsky: Revolutionary Saint to Fascist Enemy!*

## 5 When Scapy is too high-level

by Eric Davisson

Neighbors, we are hackers. Our power comes from the ability to understand and manipulate things at the lowest level we can get our hands on. Verily, a stack-based buffer overflow makes sense to those who understand machine code and assembly, but it makes no sense to those who only use high-level languages, for they know not what a program stack is, nor rejoice in the wonders of the ABI.

Likewise with TCP/IP. Those who only use others' applications to talk to a networked host never learn the miracles of the protocols below. Preach to them the good news of Netcat, and of Scapy in Python or Net::Raw in Perl, neighbors—but forget not that these excellent tools may still mask the true glory of the raw bytes below.

This article will take us a step farther down than these tools do. We will create a proper packet in a pcap file with `xxd`. Let us please the ASCII art gods of TCP in the truly proper way, neighbors!

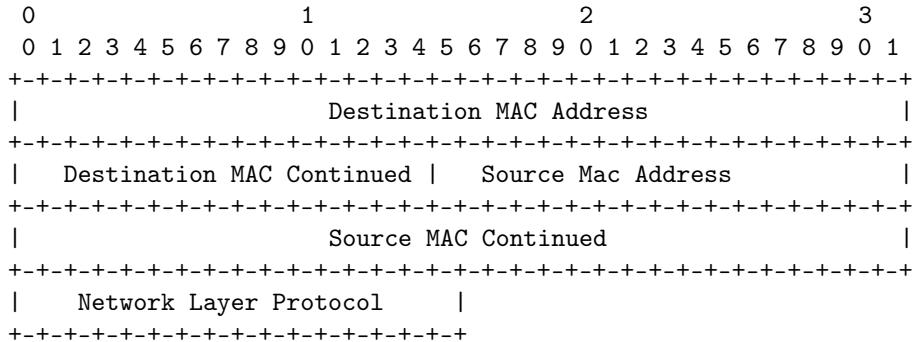
There are books dedicated to TCP/IP, neighbors, such as St. Stevens' *TCP/IP Illustrated Vol. 1*, a very thick and thorough book indeed. But at times when you don't have the Bible a mere tract would suffice; and so here's ours briefest tract on TCP/IP.

Let's begin by compressing the full OSI model to just the four layers that are actually relevant to TCP/IP. From the lowest layer up, we have the Data Link, Network, Transport, and Application layers—but of course it's not what we call these layers that matters, but what bytes they contain.

Each layer has a byte or two that specify which kind of protocol the next layer will be. So the Data Link Layer will specify IPv4 as the Network Layer, which will specify TCP as the Transport Layer, which will specify HTTPS as the Application Layer, and so on. This is really what makes the “stack”, and we will tour it from the bottom up.

### 5.1 The Layers

**Data Link Layer** This is the first and the simplest layer. For most traffic, it has the destination and source MAC addresses and 2 bytes referring to what the Network Layer should be. The most common next protocol would be IPv4 (0x0800). Other possible protocols include IGMP (0x0641), ARP (0x0806), IPv6 (0x86DD), and STP (0x8181).



**Network Layer (RFC791)** Let's assume we are dealing with IPv4. There are many fields in the IPv4 header; the most interesting ones<sup>7</sup> are: Version, Total length, TTL, Source and Destination IP addresses, Checksum, and—the most important to our next layer—the Protocol byte.

That next layer to the IPv4 network layer protocol can also be many things. The most common are TCP (0x06), UDP (0x11), and ICMP (0x01), but there are well over a hundred other choices such as IGMP (0x02), GRE (0x2F), L2TP (0x73), SKIP (0x39), and many others.

<sup>7</sup> The Pastor notes that `fragroute` might beg to differ, and your neighborly IDS might agree. It suffices to say that the IDS evasion party that Rev. Ptacek and Rev. Newsham started in 1998 is still going strong.

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Version  IHL  Type of Service		Total Length	
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Identification	Flags	Fragment Offset	
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Time to Live	Protocol	Header Checksum	
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Source Address			
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Destination Address			
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Options		Padding	
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			

**Transport Layer (RFC793)** The intent of this layer is to handle the transportation of data between two hosts. For UDP, this header is just the source and destination ports, length, and a checksum. For “reliable” connections there’s TCP, of which we’ll talk more later. TCP headers are more complex, since it takes more data to set up a connection with a 3-way handshake and agreed-upon SEQ/ACK numbers. So TCP includes the ports, some flags, a window size, checksum, and some other fields. The destination port is implicitly used to specify what the application layer will be: HTTP (80), HTTPS (443), SSH (22), SMTP (25), and so on.

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Source Port		Destination Port	
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Sequence Number			
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Acknowledgment Number			
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Data	U A P R S F		
Offset	Reserved	R C S S Y I	Window
		G K H T N N	
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Checksum		Urgent Pointer	
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
Options		Padding	
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			
data			
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+			

And now that the gods as ASCII art have been properly pleased, let’s make some packets!

## 5.2 Crafting a Packet

**Link Layer** Let’s choose a destination MAC address of 12:34:56:78:9A:BC and a source MAC address of 31:33:37:31:33:37. We also need to specify the network-layer protocol of IPv4, 0x0800.

**Network Layer (IPv4)** The version is `0x4`, and that's the first nybble of our header. The header length is going to be twenty bytes, as we will use no IP options.<sup>8</sup> The second header nybble is the header length in 32-bit words, and so it will be `0x5` to represent our twenty bytes. So the first byte will be `0x45`, combining the version and the header length. When you next see this byte at the start of an IP packet's hexdump, give it a smiling node like a good neighbor!

The type of service byte doesn't matter unless your site implements special QoS for things like voice and streaming video, so we'll arbitrarily set that to `0x00`. The following field, the total length of this packet, will be 61 bytes (IP+TCP+Payload), `0x003D` in hex. We'll just spoof the IP identification field to be `0x1337`. Next, let's set the IP flags to not fragment (`0b010`) and a fragment offset of zero. As these fields share bytes, the hex result of these two bytes will be `0x4000`. For the next field, the Time-To-Live, let's be generous and give our packet a TTL of 140 (`0x8C`), which is higher than Linux or Windows would set by default.<sup>9</sup>

Our higher-layer protocol will be TCP, `0x06`. Let's skip over the IP checksum for the moment (although we will have to correct that later). The source IP will be `192.168.1.1` (`0xC0A80101`) and the destination IP will be `192.168.1.2` (`0xC0A80102`), an HTTPS server. There will be no options or padding.

To compute the checksum, let's take all our IP header data we filled in so far in two-byte chunks, add it together, then add the overflowing byte back into the result, and subtract from `0xFFFF`. So  $0x4500 + 0x003D + 0x1337 + 0x4000 + 0x8C06 + 0xC0A8 + 0x0101 + 0xC0A8 + 0x0102$  is `0x2A7CD`. `0x2` is the overflow, so we add it back in to get  $0xA7CD + 0x2 = 0xA7CF$ . Subtracting this from `0xFFFF`, we find  $0xFFFF - 0xA7CF = 0x5830$ , our packet's IPv4 checksum.

It's now time to set up our transport layer, TCP.

**Transport Layer (TCP)** Let's say our source port will be `0x1337`, and the destination port will be `0x01BB`, which is decimal 443 for HTTPS. There's no point to any specific SEQ or ACK numbers for this implausible single packet, so we'll just use `0x00000000` and `0x00000000`.

The data offset (TCP header length) and flags share some bytes. We will have 32 bytes in our TCP header, including the 12 bytes of TCP options. 32 bytes are eight 32-bit words, so our data offset field is `0x8`.

We want this packet to have the flags of PUSH and ACK, so setting these bits gives us `0x18`. Combining these two values gives us the 2-byte value of `0x8018`, where the middle zero is a reserved nybble.

As we don't care to specify a window size at the moment, we'll default to `0x0000`—but keep in mind that putting a zero length in a TCP response is a rather evil trick you should only use on spammers and SEOs (look up the SMTP/TCP “LaBrea Tarpit” technique for more details.) We will do the checksum later, as a TCP checksum applies both to the header and to the payload. Since we won't be using the URG flag to mark this packet as urgent, we'll leave the urgent pointer field as `0x0000`.

For the options, we will use two NOPs for padding, to ensure an even number of 32-bit words, `0x0101`. Our option will be a timestamp (`0x08`), with a length of 10 (`0x0A`). Its TSval will arbitrarily be `0xDEADBEEF`, and its TSecr will be `0xFFFFFFFF`.

It is now time for the TCP checksum. A TCP checksum is calculated similarly to the IP one, but it also covers some of the IP fields!<sup>10</sup> The source IP, the destination IP, and the protocol number must all be included. Also included is the size of the TCP section, including the payload data.

$(0xC0A8 + 0x0101 + 0xC0A8 + 0x0102 + 0x0006 + 0x0029) + 0x1337 + 0x01BB + 0x0000 + 0x0000 + 0x0000 + 0x0000 + 0x8018 + 0x0000 + 0x0000 + 0x0101 + 0x080A + 0xDEAD + 0xBEEF + 0xFFFF + 0xFFFF + 0xD796 + 0xC34F + 0x4FC7 + 0xE3C6 + 0xD600$  is `0x963A3` with an overflow of `0x9`.  $0x63A3 + 0x9 = 0x63AC$ , and  $0xFFFF - 0x63AC = 0x9C53$ , our TCP checksum.

**PCAP Metadata** So now we have the packet, but to look at it with the standard dissection tools (Tcpreplay, Wireshark) or to use it with an injection tool (Tcpreplay), we need to create some metadata first.

<sup>8</sup> But if you are looking to light up your local IDS like a Christmas tree, by all means add some later! —PML

<sup>9</sup> But check out `/proc/sys/net/ipv4/ip_default_ttl`; for Windows, you are on your own—and many happy reboots! —PML

<sup>10</sup> Yes, neighbors, it is an OSI layering violation—and it has been extracting its cost, in sweat, blood, and Oday. And if you think you are properly scared, you are not scared enough—just think of that SCADA protocol that has kept your neighborhood's lights on, so far. —PML

We will use the PCAP format, the most common format of packet capture tools.

A PCAP starts with 24 bytes of global file-scope metadata and another 16 bytes of per-packet metadata. The first six of PCAP's 4-byte fields are the magic number (0xA1B2C3D4), the PCAP version (2.4, so 0x00020004), the timezone (GMT, so 0x00000000), the sigfigs field<sup>11</sup> (0x00000000), the snaplen<sup>12</sup> (0x0001000F) and the network's data link type<sup>13</sup> (Ethernet: 0x00000001).

So our global header will be A1B2C3D40002000400000000000000001000F00000001. Fun fact: reversing the order of the magic number to 0xD4C3B2A1 will change the endianness of the PCAP metadata—alerting your packet analyzer that the order of bytes in the capture file from another system should be reversed.

The per-packet data consists of four 4-byte fields: time, microtime, packet length, and captured length. Let's set the time to default day (0x4EBD02CF) and zero out the microtime (0x00000000). Our packet length will be 0x00000004B, and we'll repeat the same value for the capture length.

**Saving the pcap.** Below you see a massively ugly command. We are echoing all of the above hex data in order, starting with the PCAP file's global metadata and following with the packet data. There isn't a single byte of this that we didn't discuss above; it's all there. We pipe it through `xxd` and use the `-r` and `-p` arguments to convert it from hex to actual binary data (`-p` tells `xxd` to expect a continuous hexdump without per-line addresses or offsets, rather than the standard `xxd` output; any whitespace including line breaks is ignored in this mode). Say hello to `lol.pcap`:

```
echo A1B2C3D4 00020004 00000000 00000000 0001000F 00000001 \
    4EBD02CF 00000000 0000004B 0000004B \
    \
    12345678 9ABC3133 37313337 0800 \
    \
    45 00 003D 1337 4 000 8C 06 5830 C0A80101 C0A80102 \
    \
    1337 01BB 00000000 00000000 8 0 18 0000 9C53 0000 \
    01 01 08 0A DEADBEEF FFFFFFFF \
    \
    D796C34F4FC7E3C6D6 | xxd -r -p > lol.pcap
```

Now that you have a PCAP (see also Fig. 1), you can open it up in Wireshark and select each field in the Packet Details section to see the corresponding hex data in the Packet Bytes section. If you want to send a hand-crafted packet over your network, just replay it with something like

```
sudo tcpreplay -i eth0 lol.pcap
```

Hack around, change some bytes, and see what happens. Do impossible things, like setting the IPv4 layer's first byte to 0x43, which specifies an IPv4 packet with a 12-byte IP header. This means the IP header doesn't have room for its own IP addresses. What will your little Linksys box do when it gets such a packet? What will your newest shiny box with that fruit logo do? And how much do you dare trust that penguin, really? Well, there is—and there has ever been—only one way to find out :)

---

<sup>11</sup>In theory, this is the accuracy of time stamps in the capture; in practice, typically set to zero.

<sup>12</sup>This is the maximum length of captured packets, in octets, or zero for no limit.

<sup>13</sup>`man 7 pcap-linktype` (from libpcap0.8-dev or equivalent)

0	15	31	47	63				
magic number				pcap version				
A1	B2	C3	D4	00	02	00	04	
timezone				sigfigs				
00	00	00	00	00	00	00	00	
snaplen				data link type				
00	01	00	0F	00	00	00	01	
time				microtime				
4E	BD	02	CF	00	00	00	00	
packet length				captured length				
00	00	00	4B	00	00	00	4B	
Destination MAC								Source MAC
12	34	56	78	9A	BC	31	33	
Source MAC Continued				NLP				
37	31	33	37	08	00			
Ver.n	IHL	ToS	Total Length	Identification	Fl.	Fragment Offset		
45		00	00 3D	13 37	40	00		
TTL	Protocol	Header Checksum			Source Address			
8C	06	58	30	C0 A8	01	01		
Destination Address								
C0	A8	01	02					
Source Port		Destination Port		Sequence Number				
13	37	01	BB	00 00	00 00			
Acknowledgment Number				DOFF	Reserv.	UAPRSF RCSYSI GKHITNN	Window	
00	00	00	00	80	18	00 00		
Checksum		Urgent pointer		Options				
9C	53	00	00	01	01	08 0A		
DE	AD	BE	EF	FF	FF	FF FF		
Raw								
D7	96	C3	4F	4F	C7	E3	C6	
D6								

Figure 1: Crafted PCAP

## 6 Abusing file formats; or, Corkami, the Novella

by Ange Albertini

First, you must realize that a file has no intrinsic meaning. The meaning of a file—its type, its validity, its contents—can be different for each parser or interpreter.

Like beef cuts, which vary with the country's standards by which the animal is cut, a file is subject to interpretations of the standard. The beauty of standards is that there are so many interpretations to choose from!

Because these standards are sometimes unclear, incomplete, or difficult to understand, a variety of abuses are possible, even if the files are considered valid by individual parsers.

A *Polyglot* is a file that has different types simultaneously, which may bypass filters and avoid security counter-measures. A *Schizophrenic* file is one that is interpreted differently depending on the parser. These files may look innocent (or corrupted) to one interpreter, malicious to another. A *Chimera* is a polyglot where the same data is interpreted as different types, which is a more advanced kind of filter bypass.

This paper is a classification of various file techniques, many of which have already been mentioned in previous PoCs and articles. The point here is to have an overview and comparison of them, not to necessarily explain again all of them in detail.

### 6.1 Identification

It's critical for any tool to identify the file type as early and reliably as possible. The best way for that is to enforce a unique, not too short, fixed signature at the very beginning. However, these magic byte signatures may not be perfectly understood, leading to some possible problems.

Most file formats enforce a unique magic signature at offset zero. It's typically—but not necessarily—four bytes. Office documents begin with D0 CF 11 E0, ELF files begin with 7F E L F, and Resource Interchange File Format (RIFF) files begin with R I F F. Some magic byte sequences are shorter.

Because JPEG is the encoding scheme, not a file format, these files are defined by the JPEG File Interchange Format or JFIF. JFIF files begin with FF D8, which is one of the shortest magic byte sequences. This sequence is often wrongly identified, as it's typically followed by FF E0 for standard header or FF E1 for metadata in an EXIF segment.

BZIP2's magic signature is only sixteen bits long, B Z. However it is followed by the version, which is only supposed to be h, which stands for Huffman coding. So, in practice, BZ2 files always start with the three-byte sequence B Z h.

A Flash video's magic sequence is three bytes long, F L V. It is followed by a version number, which is always 0x01, and a mask for audio or video. Most video files will start with F L V 01 05.

Some magic sequences are longer. These typically add more characters to detect transfer errors, such as FTP transfers in which ASCII-mode has been used instead of binary mode, causing a translation between different end-of-line conventions, escaping, or null bytes.

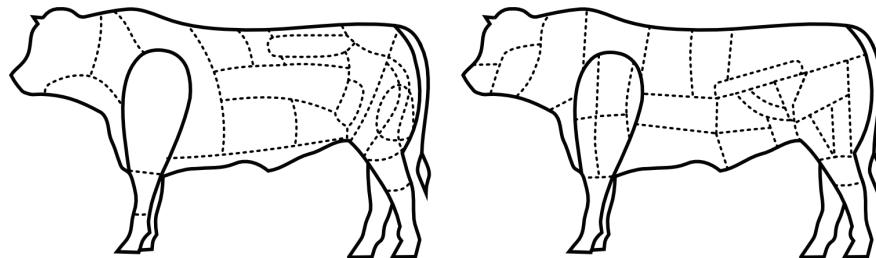


Figure 2: Brazilian and French beef cuts.

Portable Network Graphic (PNG) files always use a magic that is eight bytes long, 89 P N G OD OA 1A OA. The older, traditional RAR file format begins with R a r ! 1A 07 00, while the newer RAR5 format is one byte longer, R a r ! 1A 07 01 00.

Some magic signatures are obvious. ELF (Executable & Linkable Format), RAR (Roshal Archive), and TAR (Tape Archive) all use their initials as part of the magic byte sequence.

Others are obscure. GZIP uses 1F 8B. This is followed by the compression type, the only correct value for which is 0x08 for Deflate, so all these files are starting with 1F 8B 08. This is derived from Compress, which began to use a magic of 1F 8D in 1984, but it's not clear why this was chosen.

Some are chosen for vanity. Philipp Katz placed his initials in ZIP's magic value of P K, while Fabrice Bellard chose 0xFB for the BPG file format.

Some use L33TSP34K sequences, such as D0 CF 11 E0, CA FE BA BE, and CA FE FE ED. It looks cool, but there are not so many words that can be encoded as hex. There aren't so many collisions, but the most common one is of course CA FE BA BE, which is used for Java .CLASS and Universal Mach-O. These are easy to tell apart right after the magic, however. In a Mach-O, the magic signature is followed by the number of architectures as a big-endian DWORD, which means such a fat binary usually starts with CA FE BA BE 00 00 00 02 to indicate support for x86 and PowerPC, just two of the twenty supported architectures.<sup>14</sup>. Conversely, a Java Class puts minor and major version numbers right after the magic, and *major\_version* should be greater than or equal to 0x2D, which indicated JDK 1.1 from 1997.<sup>15</sup>

-----  
Some file formats can be seen as high-level containers, with vastly differing internal file formats. For example, the Resource Interchange File Format (RIFF) covers the AVI video container, the WAV audio container, and the animated image ANI. Thus three different file types (video, audio, animation) are relying on the same outer format, which defines the magic that will be required at offset zero.

## Encodings

Some file formats accept different encodings, and each encoding uses a different Magic signature.

TIFF files can be either big or little endian, with I I indicating Intel (little) endianness and M M for Motorola (big) endianness. Right after the signature is the number forty-two encoded as a 16-bit word—00 2A or 2A 00 depending on the endianness—so the different magics feel redundant! A common T I F F magic before this endianness marker would have been good enough.

32-bit Mach-O files use FE ED FA CE, while 64-bit Mach-O files use FE ED FA CF. The next two fields also imply the architecture, so a 32-bit Mach-O for Intel typically starts with FEEDFACE 00000007 00000003, while a 64-bit file starts with FEEDFACF 01000007 80000003, defining a 64b magic, ABI64 architecture, and Lib64 as a subtype.

Flash's Small Web Format originally used the F W S magic, then its compressed version used the C W S magic. More recently, the LZMA-compressed version uses the Z W F magic. Once again, it doesn't make sense as the signatures are always followed by a version number. A higher bit could have been set to define the compression if that was strictly necessary. In practice, however, it turns out that there is rarely a check for these values. Why do they bother defining a version number and file size if it just works with any value?

While most file formats enforce their magic at offset zero, it's common for archive formats to NOT enforce magic at the start of an archive. 7ZIP, RAR, and ZIP have no such requirement. However, some Unix compressors such as GZIP and BZIP2 do demand proper magic at offset zero. These are just designed to compress data, with the filename being optional (for GZIP) or just absent (BZIP2).

## Specific Examples

TAR, the Tape Archive format, was first used to store files via tape. It's block-based, and for each file, the header block starts with the filename. The magic signature, depending on the exact version of TAR,

<sup>14</sup><http://tinyurl.com/MachO-fat-header>

<sup>15</sup><http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-4.html#jvms-4.1>

is present at offset 0x100 of the header block. The whole header contains a checksum for itself, and this checksum is enforced.

PDF in theory should begin with a standard signature at offset zero, % P D F - 1 . [0-7], but in practice this signature is required only to be within the first kilobyte. This limit is odd, which is likely the reason why some PDF libraries don't object to a missing signature. PDF is actually parsed bottom-up for a complete document interpretation to allow for incremental document modifications. Further, the signature doesn't need to be complete! It can be truncated, either to %PDF-1. or %PDF\0.

ZIP doesn't require magic at offset zero, and like PDF it's parsed from the bottom up. In this case, it's not to allow for incremental updates; rather, it's to limit those time-consuming floppy swaps when a multi-volume archive is created on the fly, on external storage. The index structure must be located near the end of the file.

Even more confusingly, it's common that viewers and the actual extractor will have a different threshold regarding the distance to the end of file. WinRAR, for example, might list the contents of an archive without error, but then silently fail to extract it!

Although standard ZIP tolerates not starting at offset zero or not finishing at the last offset, some variants built on top of the ZIP format are pickier. Keep this in mind when creating funky APK, EGG, JAR, DOCX, and ODT files.

## Bad Magic Signatures

OpenType fonts start with 00 01 00 00, which is actually not a magic signature, but a version number, which is expected to be constant. How pointless is that?

Windows icons (ICO) and static cursors (CUR) are using the same format. This format has no official name, but it always has a magic of 00 00.

## 6.2 Hardware Formats

Hardware-oriented formats typically have no header. They are designed for efficiency, and their parser is implemented in hardware. They are seen not as files, but as images burned into a ROM or similar storage. They are directly read (and executed/interpreted) by a CPU, which often specifies critical data at the very first offsets.

For example, floppy disks and hard disks begin with a 512-byte Master Boot Record (MBR) of executable code that must end with 0xAA55. Video game console ROMs often begin with the initial stack pointer and program counter. The TGA image format, which was designed in 1984 as a raster image format to be read directly by a graphics board, begins with the image's width and height. (Version 2 of TGA has an optional footer, ending with a constant signature.)

However, it's also common that some extra constant structure is required at a specific offset, later in the memory space. These requirements are often enforced in software by the BIOS or bootloader, rather than by a hardware check. For example, a Megadrive (Genesis) cartridge must have the ASCII string "SEGA" at offset 0x100.<sup>16</sup> A Gameboy ROM must contain the Nintendo logo for its startup screen from offset 0x104 to 0x133, one of the longest signatures required in any file format.<sup>17</sup> Super NES ROMs have a header later in the file, called the Cartridge Header. The exact offset of this header varies by the type of ROM, but it is always far enough into the header that polyglot ROMs are easy to create.<sup>18</sup> Examples of such polyglots are shown in Figures 3 and 4.

## Abusing File Signature

Obviously, there is no room for abusing signatures as long as the content and the offset of the signatures are strictly enforced. Signature abuse is possible when parsers are trying to recover broken files; for example,

<sup>16</sup><http://wiki.megadrive.org/index.php?title=TMSS>

<sup>17</sup><http://problemkaputt.de/pandocs.htm#thecartridgeheader>

<sup>18</sup><http://problemkaputt.de/fullsnes.htm>

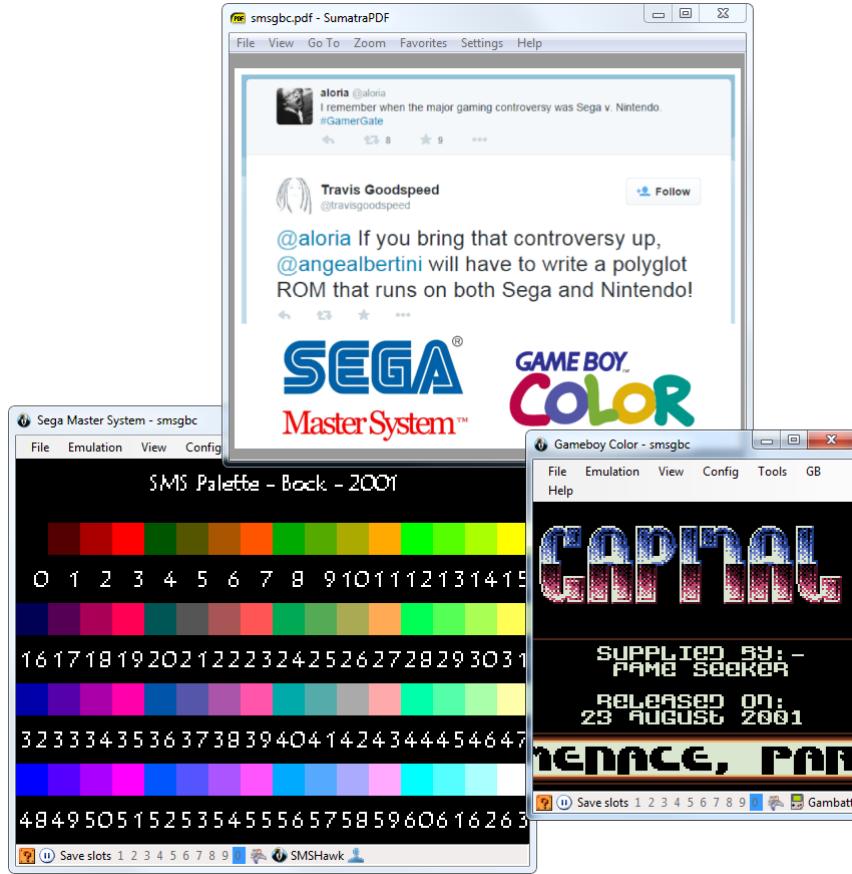


Figure 3: Sega Master System, Gameboy Color & PDF Polyglot

some PDF readers don't require the presence of the PDF signature at all!

Header abuse is also possible when the specification is incorrectly implemented. For example, the GameBoy Pocket—and only the GameBoy Pocket—doesn't bother to fully check the BIOS signature.

### Blacklisting

As hinted previously, PDF can be easily abused. For security reasons, Adobe Reader, the standard PDF reader, has blacklisted known magic signatures such as PNG or PE since version 10.1.5. It is thus not possible anymore to have a valid polyglot that would open in Adobe Reader as PDF. This is a good security measure even if it breaks compatibility with older releases of PoC||GTFO.

However, it's critical to blacklist the actual signature as opposed to what is commonly appearing in files. JPEG File Interchange Format (JFIF) files typically start with the signature, SOI, and an APP0 segment, which make the file start with FF D8 FF E0. However, the signature itself is only FF D8, which can lead to a blacklist bypass by using a different segment or different marker right after the signature. I abused this trick to make a JPEG/PDF polyglot in PoC||GTFO 0x03, but since then, Adobe has fixed their JFIF signature parsing. As such, pocorgtfo03.pdf doesn't work in versions of Adobe Reader released since March of 2014.

Of course, blacklisting can only affect current existing formats that are already widespread. The Z W S signature that we used for PoC||GTFO 0x05 is now blacklisted, but the BPG signature used in PoC||GTFO 0x07 is very recent so it has not been blacklisted yet. Moreover, each signature to be blacklisted has to be added manually. Requiring the PDF signature to appear earlier in the file—even just in the first 64 bytes

instead of a whole kilobyte—would proactively prevent a lot of polyglot types, as most recent formats are dense at the start of the file. Checking the whole signature would also make it even harder, though not respecting your own standard even for checking signatures is an insult to every standard.

### 6.3 File Format Structures

Most file formats are either chunk-based or pointer-based. Chunked files are often some variant of Tag/Length-/Value (TLV), which are versatile and size-efficient. Pointer-based files are better adapted to direct memory mapping. Let's have some fun with each.

#### Chunk Sequences

The information is cut into chunks, which all have the same top-level structure, often defining a type, via a tag, then the length of the chunk data to come, then the chunk content itself, of the given length. Some formats such as PNG also require their chunks to end with a checksum, covering the rest of the chunk. (In practice, this checksum isn't always enforced.)

For even more space efficiency, BZIP2 is chunk based, but at the bit level! Bytes are never padded, and structures are not aligned. It doesn't waste a single bit, but for that reason it's damned near unreadable with a standard hex viewer. Because no block length is pre-encoded, block markers are fairly big, taking 48 bits. These six bytes, if they were aligned, would be 31 41 59 26 53 59, the BCD representation of  $\pi$ .

#### Structure Pointers

The first structure containing the magic signature points to the other structures, which typically don't lie immediately after each other. Pointers can be absolute as in file offsets, or relative to the current structure's offset or to some virtual address. In many cases, relative pointers are unsigned. Typically, executable images use such pointers for their interrupt tables or entry points.

In many chunk-based formats such as FLV, you can inflate the declared size of a chunk without any warnings or errors. In that case, the size technically behaves as a relative pointer to the next chunk, with a lower limit.

### 6.4 Abusing File Format Structures

#### Empty Space

Block-sized formats, such as ISO,<sup>19</sup> TAR, and ROM dumps often contain a lot of extra space that can be directly abused.

In theory, it may look like TAR should have lots of zero bytes, but in practice, it's perfectly fine to have one that's 7-bit ASCII! This makes it possible to produce an ASCII abstract that is a valid TAR. For good measure, the one shown in Figure 5 is not only an ASCII TAR, but also a PDF. The ASCII art comes free.

---

<sup>19</sup>PoC||GTFO 0x05



---

**German GQRP Club Members**  
**MEETING IN MAY 1998**  
**Please contact Rudi before the end of January**  
**Rudi Dell, DK4UH, Weinbietstr. 10, 67459, BOEHL-IGGELHEIM**

---

## Appended Data

Since many formats define an end marker, adding any data after is usually tolerated: after all, the file is complete, parsing can end successfully. However, it's also easy for them to check if they reached the end of the file: in this case (such as BPG or Java Class), no appended data is tolerated at all.

## Trailing Space

Metadata fields are often null-terminated with a maximum length. This gives us a bit of controllable space after the null character. That way, one could fit a PDF signature and stream declaration within the metadata fields of a NES Sound Format (NSF) to get a working polyglot.

This is shown in Figure 6, where the NSF's Title is "SSL Smiley song :-)\\0%PDF-1.5". Similarly, the Author is "Melissa Elliott\\0 9 0 obj <<>>%" and the Copyright is "2014 0xabad1dea"\\0 \\n stream \\n".

The original metadata is preserved, while declaring a PDF file and a dummy PDF object that will cover the rest of the data of the NSF file.

## Non-Critical Space

Some fields are required by a standard, but the parsers will forgive us for violations of the standard. These parsers try to recover information out of corrupt files rather than halting on invalid structures.

JFIF is a clear example. Many JFIF segments clearly define their length, however nothing prevents you from inserting extra data at the end of one segment. This data may be ignored, and the parser will just look for the next segment marker. Since JFIF specifies that all segments are made of FF followed by a non-null byte, as long as your extra data doesn't encode a segment marker for a known segment type, you're fine. Known types include Define Quantization Table FF DB, Define Huffman Table FF C4, Start Of Scan FF DA, and End Of Image FF D9.

In console ROMs, CPU memory space often starts with interrupt vector tables. You can adjust the handler addresses to encode a useful value, or sometimes use arbitrary values for unused handlers.

## Making Empty Space

In a chunk-structured format, you can often add an auxiliary chunk to carve extra space. Forward compatibility makes readers fully ignore the extra chunk. Figure 7 shows a PNG whose "duMb" chunk happens to contain valid PCM audio.

Sometimes, you have to flip a bit to enable structure space that can be abused. Examples include the 512-byte training buffer in the iNES (.nes) ROM format, which is used to hold code for enabling cheats.

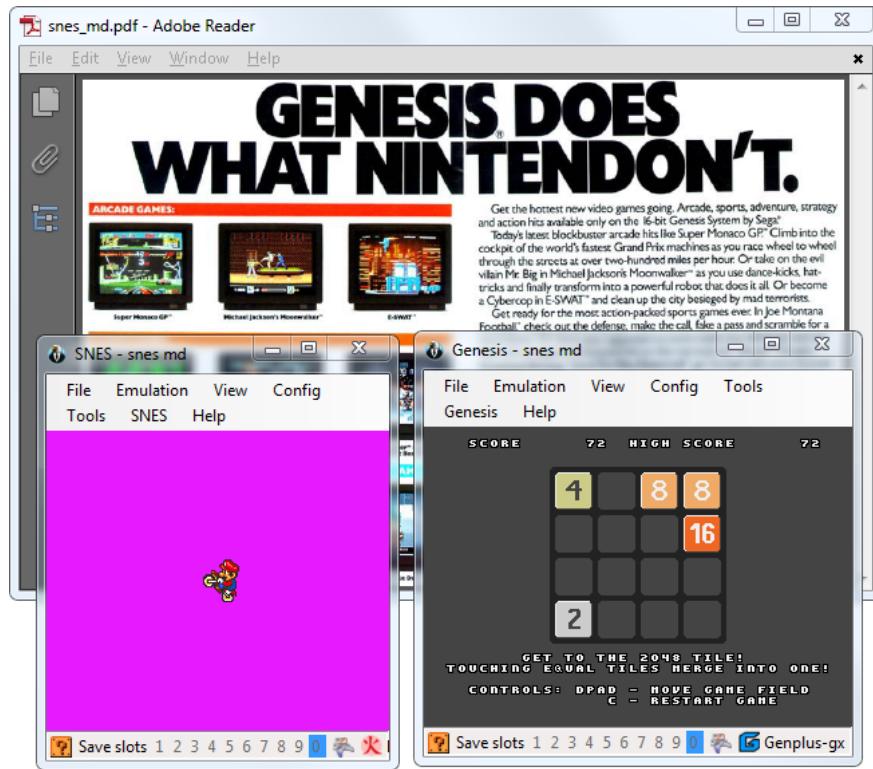


Figure 4: Sega Megadrive, Super Nintendo & PDF Polyglot

```
$ tar -tf abstract.tar
Binary tricks to evade identification, detection, to exploit encryption and hash collisions
$ []
```

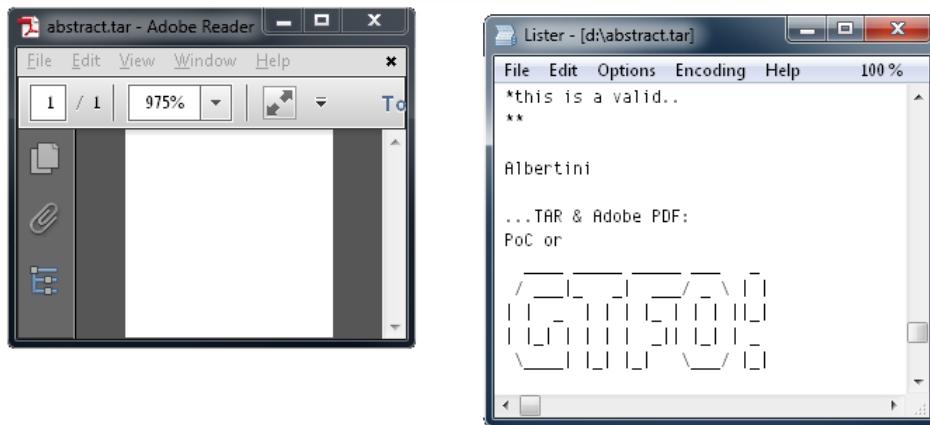


Figure 5: PDF, TAR Polyglot in 7-bit Clean ASCII

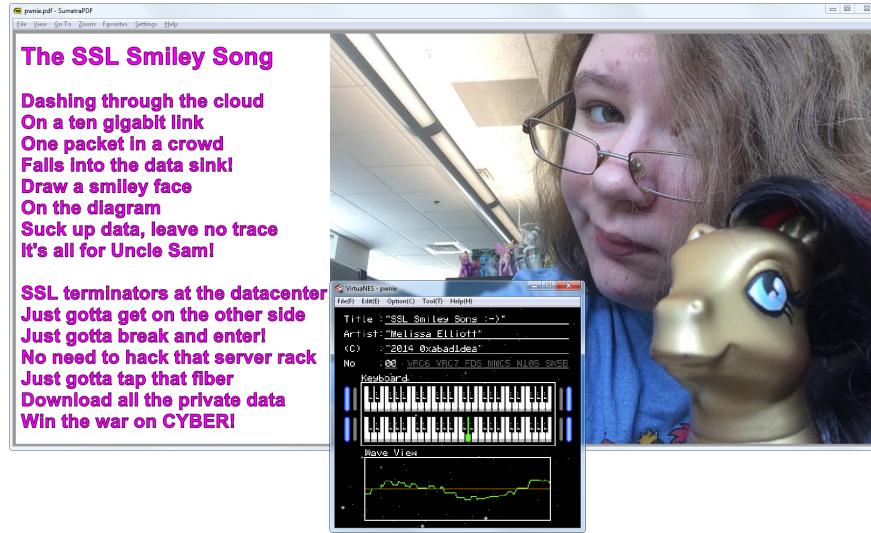


Figure 6: PDF and NES Sound Format polyglot

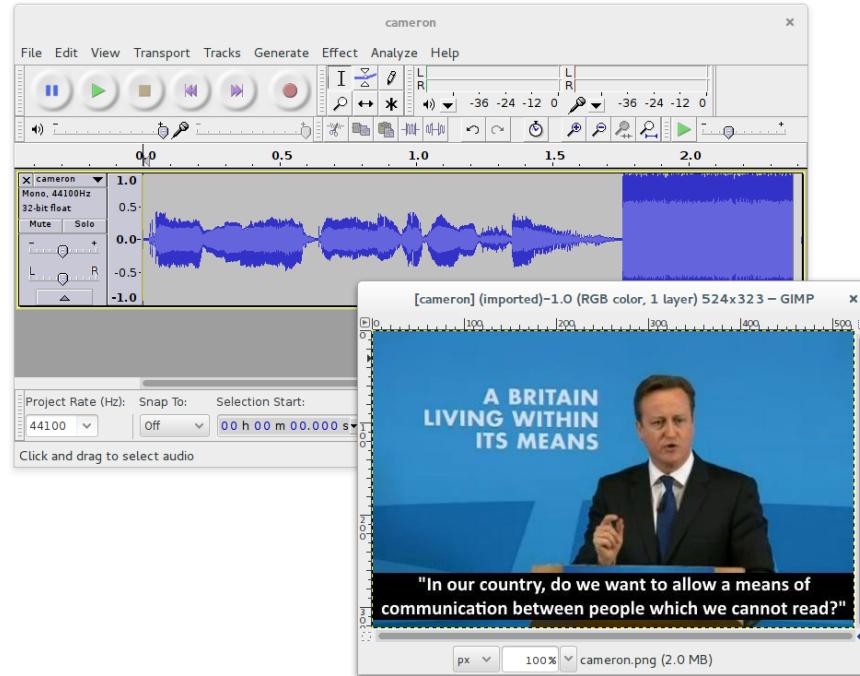


Figure 7: PNG whose “duMb” chunk contains PCM Audio

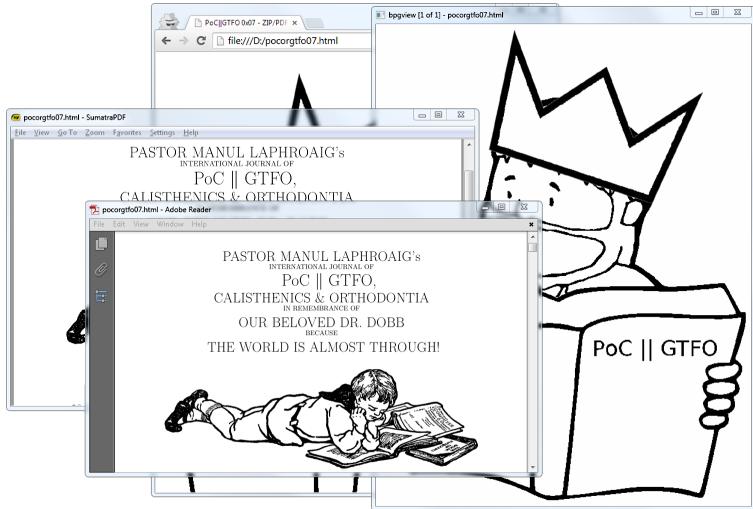


Figure 8: BPG/HTML/PDF Polyglot. ZIP not shown.



*Telpar has a new thermal printer (very quiet), which is 48 columns wide and has four interfaces (TTL parallel and serial, 20 mA, RS-232). The print quality looked very nice . . . and so did the price of \$666. 4132 Billy Mitchell Rd., PO Box 796, Addison TX 75001.*

**X-Y PLOTTERS**  
NOW YOU  
CAN HAVE A  
PLOTTER ALSO  
!!!

**SYLVANHILLS PLOTTERS ARE FOR YOU**

Completely assembled X-Y mechanism & interface. You add drawing table, 24" x 36" sturdy aluminum frame. Resolution is .01 inch. Rugged enough to handle 11x17 inch plots. 17x22 inch plots. 17x22 inch plots. Owners manual (Included with purchase) \$890. SOFTWARE NOW AVAILABLE!!!

**Sylvan Hills Laboratory, Inc.**

**MAC GYVER ARMORY**

**20th Century**

**WRIGLEY'S SPEARMINT CHEWING GUM**  
LASTS  
76mm x 19mm

DNA storage, conductor, sealing material, adhesive, stress relieve, nomnom

**21th Century**

Open source hardware and software. ARM Cortex-A8 800MHz, 512MB RAM, microSD, USB 2.0 OTG, Ethernet/Storage/I2C/I2S/HID/etc device emulation, 65mm x 19mm

**A PDF/ZIP/BPG/HTML polyglot** BPG<sup>20</sup> stands for Better Portable Graphics. It was recently created as an alternative to JPG, PNG, and GIF. BPG images can be lossy or lossless. The format supports animation and transparency.

To give BPG more exposure, this issue is a PDF/ZIP/BPG/HTML polyglot. Also, we're running out of formats that Adobe hasn't blacklisted as polyglots.

BPG's structure is very compact. Some fields' bits are split over different bytes, most numerical values are variable-length encoded, and every attempt is made to avoid wasted space. Besides the initial signature, everything is numerical. 'Chunk types'—called 'extension tags'—are not ASCII like they commonly are in PNG. Information is byte-aligned, so the format isn't quite so greedily compressed as BZIP2.

BPG enforces its signature at offset zero, and it is not tolerant to appended data, so the PDF part must be inside of the BPG part. To make a BPG polyglot, enable use the extension flag to add your own extension with any value other than 5, which is reserved for the animation extension. Now you have a free buffer of an arbitrary length.

Since the author of BPG helpfully provides a standalone JavaScript example to decompress and display this format, a small page with this script was also integrated in the file. That way the file is a valid BPG, a valid PDF, and a valid HTML page that will display the BPG image. You just need to rename the `pocorgtfo07.pdf` to `pocorgtfo07.html`. You can see this in Figure 8.

Thanks to Mathieu Henri for his help with the HTML part.

**Moving Structures Around** In a pointer-chained format, you can often move structures around or even inside other structures without breaking the file. These parsers never check that a structure is actually after or outside another structure.

Technically-speaking, an FLV header defines its own size as a 32-bit word at offset 0x05, big endian. However nothing prevents you from making this size bigger than used by Flash. You can then insert your data between the end of the real header and the beginning of the first header packet.

To make some extra space early in ROMs, where the code's entrypoint is always at a fixed address, just jump over your inserted data. Since the jump instruction's range may be very limited on old systems, you may need to chain them to make enough controllable space.

## Structure Order

To manipulate encryption/decryption via initialization vector, one can control the first block of the file to be processed by a block cipher, so the content of the file in this first block might be critical. It's important then to be able to control the chunk order, which may be against the specs, and against the abilities of standard processing libraries. This was used as AngeCryption in PoC||GTFO 0x03.

The minimal chunk requirements for PNG are IHDR, IDAT, and IEND. PNG specifies that the IHDR chunk has to be first, but even though all image generators follow this part of the standard, most parsers fail to enforce the requirement.

The same is true for JFIF (JPEG) files. The APP0 segment should be first, and it is always generated in this position, but readers don't really require it. In practice, a JFIF file with no APPx segments often produces neither warnings nor errors. Figure 9 shows a functional JPEG that has no APPx segments, neither a JFIF signature nor any EXIF metadata!

## 6.5 Data Encodings

It's common for different file formats to rely on the same data encodings that have been proved reliable and efficient, such as JPEG for lossy pictures or Deflate/Zlib. Thus it's possible to make two different file formats in the same file relying on the same data, stored with the same encoding.

---

<sup>20</sup><http://bellard.org/bpg/>

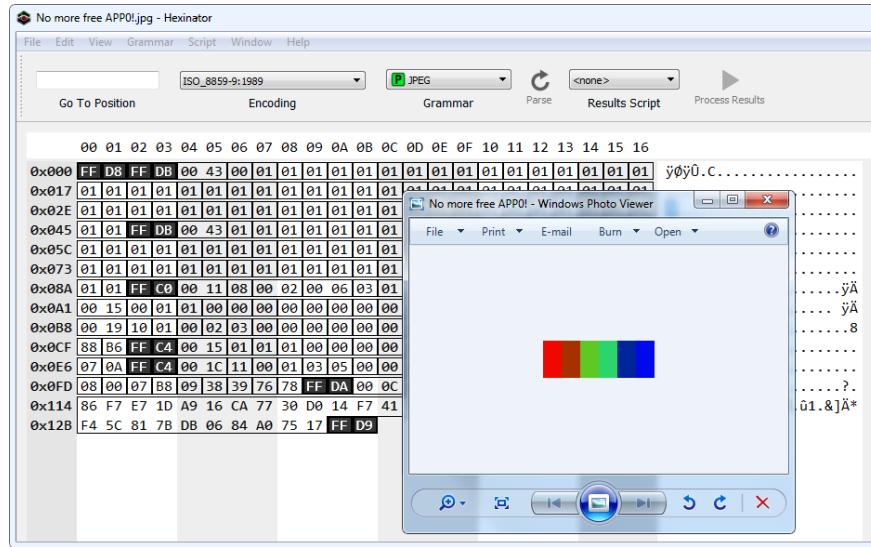


Figure 9: JPEG with no APPx segments.

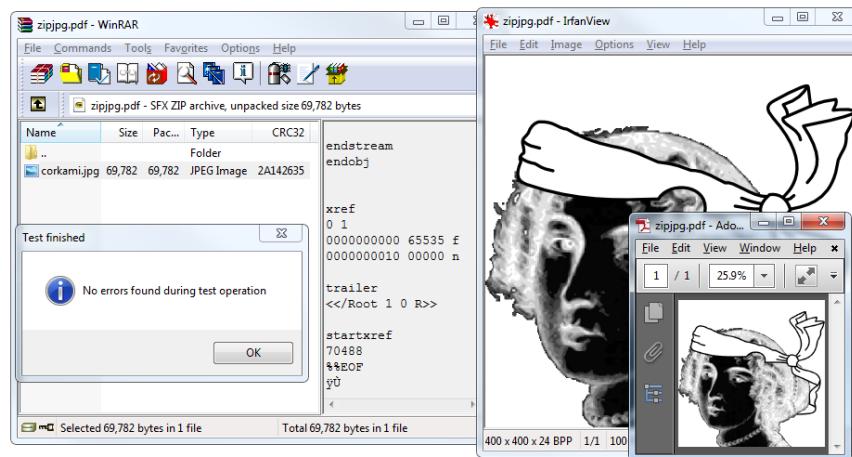


Figure 10: JPG/PDF/ZIP Chimera

Offset	Content 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	JPEG	PDF	ZIP
00000:	FF D8	magic header		
00002:	FF E0 00 10 .J .F .I .F 00 01 01 01 00 48			
00004:	00 48 00 00			
00014:	FF FE 02 1F	comment segment start (length)		
00018:	%PDF-1.4		PDF header & document	
	1 0 obj			
	...			
00140:	20 0 obj «/Length 69786» stream		dummy object start	
00168:	.P .K 03 04			local file header start
00181:	00 9B			filename length
00186:	endstream			lh's filename (abused)
	endobj			
	5 0 obj		image object start	
	«/Width 400 ...			
	stream			
00221:	FF D8 FF E0 00 10 .J .F .I .F 00 01 01 01 00	(end of comment)	image header	stored file data
00235:	48 00 48 00 00	image data (DQT)	—	—
112B5:	FF DB 00 43 ...	end of image	—	—
112B7:	FF D9	segment comment start (not strictly req.)		
	FF FE 00 E6			
112BC:	endstream		end of image object	
	endobj			
	24 0 obj		dummy object start	
	stream			
	...			
112DE:	.P .K			central directory
	01 00			
1130C:	corkami.jpg			filename (correct)
11317:	.P .K 05 06			end of central directory
1132B:	75 00			length of comment
1132E:	endstream		end of dummy object	archive comment
	endobj			
	xref		xref, trailer	
	...			
1139A:	%EOF		end of file line comment	
	%			
113A1:	FF D9	end of image marker	(end of line)	(end of comment)

Table 1: JPG/PDF/ZIP Chimera Layout

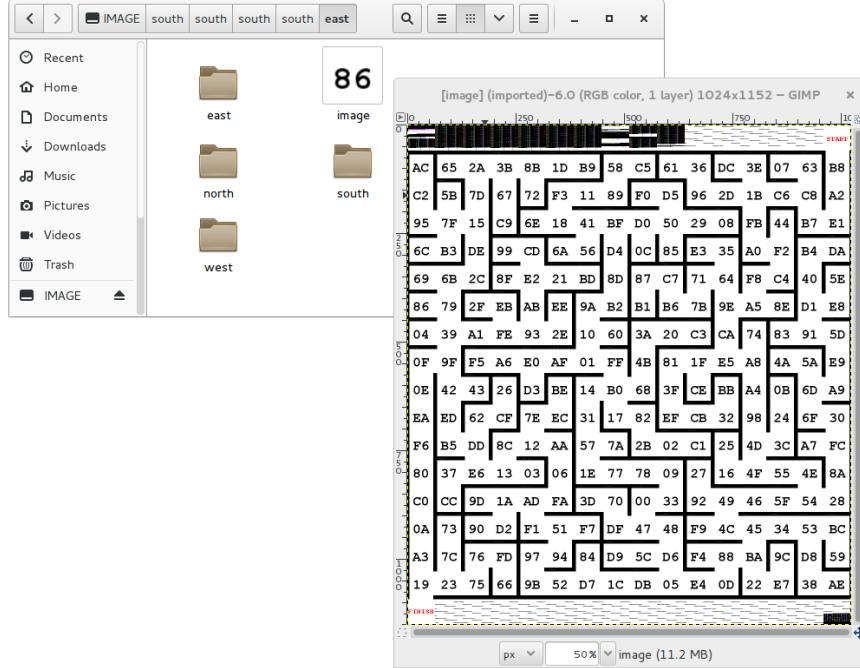


Figure 11: TIFF/EXT2 Chimera

## Abusing Data

**JPG/PDF/ZIP Chimera** For this kind of abuse, it's important to see if what comes directly before the data can be abused, and how the data offset can be abused.

A PDF directly stores JPG image and so does a ZIP archive using no compression, except that a ZIP's Local File Header contains a duplicate of the filename just before the file data itself.

Thus, we can create a single chimera that is at once a ZIP, a JPG, and a PDF. The ZIP has the JPEG image as a JFIF file, whereas the whole file is also a valid JPEG image, and the whole file is also a PDF that displays the image! Even better, we only have one copy of the image data; this copy is reused by each of the forms of the chimera.

There are two separate JFIF headers. One is at the top of the file so that the JFIF file is valid, and a duplicate copy is further in the file, right before the JPEG data, after the PDF header and the ZIP's Local File Header.

Other kinds of chimeras are possible. For example, it's not hard to use TAR instead of ZIP as the outer archive format. A neighbor could also use PNG (Zlib-compressed data, like in ZIP) instead of JPG.

One beautifully crafted example is the Image puzzle<sup>21</sup> proposed at the MIT Mystery Hunt 2015. It's a TIFF and an EXT2 filesystem where all the EXT2 metadata is visible in the TIFF data, and the filesystem itself is a maze of recursive sub-directories and TIFF tiles. This is shown in Figure 11.

## Abusing Data to Contain an Extra Kind of Information

Typically, RGB pixels of images don't need to follow any particular rule. Thus it's easy to hide various kinds of data as fake image values.

This also works in PDF objects, where lossy compression such as JBIG2, CCITT Fax, and JPEG2000 can be used to embed malicious scripts. These are picture formats, but nothing prevents us from hiding

<sup>21</sup><http://web.mit.edu/puzzle/www/2015/puzzle/image/>



Figure 12: Artistic, Valid QR Codes



Figure 13: Barcode-in-Barcode Inceptions

other types of information in them. PDF doesn't enforce these encodings to be specifically used on objects referenced as images, but allows them on any object, even JavaScript ones.

Moreover, image dimensions and depth are typically defined in the header, which tells in advance how much pixel data is required, and appending any extra data *within* the pixel stream—such as in the IDAT chunk of a PNG, which is Zlib-wrapped—will not trigger any problem with viewers. All the original pixels are present, so the image is perfect, and the extra appended data in the pixel stream remains. This can be used to hide data in a PNG picture without any obvious appended data after the IEND chunk.

### Abusing Image Parsing

In some specific cases, such as barcodes, images are parsed after rendering. Even in extreme cases of barcode manipulation, it's still quite easy to see that they could be parsed as barcodes. The examples in Figure 12 come from a SIGGRAPH Asia 2013 paper by fine folks at the City College of London on Half-Tone QR Codes.<sup>22</sup>

However, we usually have no control over the scanning software. This software determines which types of barcodes will be scanned, and in which order they will be parsed. By relying on error code information recovery – and putting a different kind of barcode inside another one! – QR Inception is not only possible, but was thoroughly investigated by the fine folks at SBA Research in Vienna!<sup>23</sup> Some quick examples are in Figure 13.

### Corrupting Data to Prevent Standard Extraction

Although many parsers may refuse to extract a corrupted stream, it's possible that some will parse until corruption is found and attempt to use the undamaged portion. By appending garbage data and corrupting

---

<sup>22</sup>[http://vecg.cs.ucl.ac.uk/Projects/SmartGeometry/halftone\\_QR/halftoneQR\\_sigga13.html](http://vecg.cs.ucl.ac.uk/Projects/SmartGeometry/halftone_QR/halftoneQR_sigga13.html)

<sup>23</sup>unzip pocorgtfo07.pdf abusing\_file\_formats/qrinception.pdf #by Dabrowski et al

```
$ cat asciizip
xÜBØUpøTZUñnnnnnnnnnnnnnnnnnUÜ5nnnnnn3SUUnUUwCiudIbEAt33wwt3wwØGDDGtwwDDwDtØ3GGpDD3333sØ3333GDdFPWØsookwKgQ1‡$▼
$ printf "\x1f\x8b\x08\0\0\0\0" | cat - asciizip | gzip -dc
PoC||GTFO !
```

Figure 14: ASCII Zlib Stream

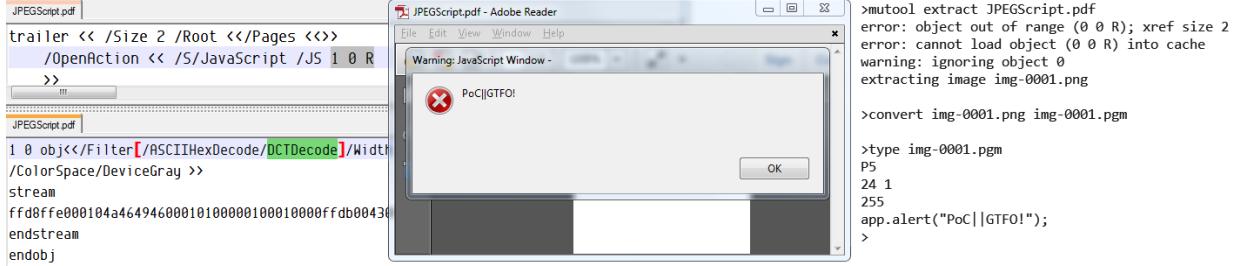


Figure 15: JPEG-Encoded JavaScript

its encoding, we can create a stream that still contains its information, but will not be extracted by purist tools!

Appending garbage, compressing, then truncating the last compressed block is a straightforward way to do this with Zlib and Deflate. Using LZMA without End of Stream markers also works. As mentioned before, you also get the same result by corrupting the CRC32 of a JAR. Most if not all ZIP extractors will fail to open the archive, whereas Java itself will ignore and execute the classes just fine.

In a similar but a bit more unpredictable way, it looks like most Windows viewers open a PNG file with corrupted checksums in critical chunks just fine. Most Linux viewers reject the file completely.

### Abusing Encoding to Bypass Filter

**ASCII Zlib Stream** As Gabor Molnar proved with ascii-zip,<sup>24</sup> it's possible to turn the Huffman coding used in Zlib into an ASCII-only expansion, and thus send a Zlib-compressed binary as a standard ASCII string. An ASCII gzip file using this trick is shown in Figure 14.

Michele Spagnuolo used this same trick in the better-known RosettaFlash attack, the details of which you can find in PoC||GTFO 5:11.

**Lossless JPEG** We can abuse JPEG's lossy compression and turn it lossless. Since it's lossy by definition, it makes sense to expect that it cannot be controlled, so it is often ignored by security software. But, by encoding a greyscale JPEG, chrominance and luminance separation is fully predictable, as there is no more chrominance. Combining this with either 100% quality compression or a specific quantization matrix allows the decompressed data to be predictable and reusable! Dénes Óvári demonstrated PoC of this in VirusBulletin 2015/03,<sup>25</sup> and an example of the technique is shown in Figure 15.

### Altering Data to Contain Extra Information

**Image and Sound** When sound is stored as 32-bit PCM, the 16 lower bits can be modified without much effect on the final sound as 16-bit resolution allows for a comfortable dynamic range of about 96 dB.

The BMP file format allows us to define *both* which color channels are stored, *and* on how many bits those channels are stored. Thus, it's possible to store a 16-bit picture as 32-bit words, leaving 16 bits of each word unused! By combining these two techniques, we can mix picture and sound on the same words: 16 bits of audible sound, 16 bits of visible pixel colors. The sound is ignored for the picture, and the image drops below the threshold of hearing.

<sup>24</sup><https://github.com/molnarg/ascii-zip>

<sup>25</sup>[unzip pocorgtfo07.pdf abusing\\_file\\_formats/vb201503-lossy.pdf](#)

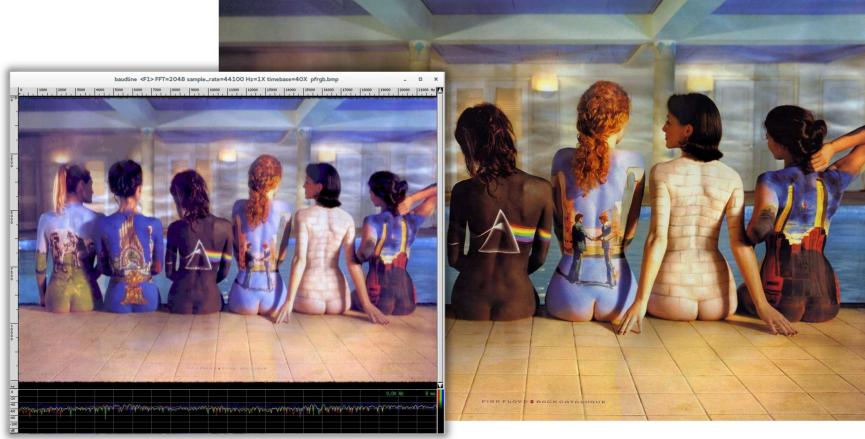


Figure 16: BMP Image with Another Image as RGB Channels in PCM Audio

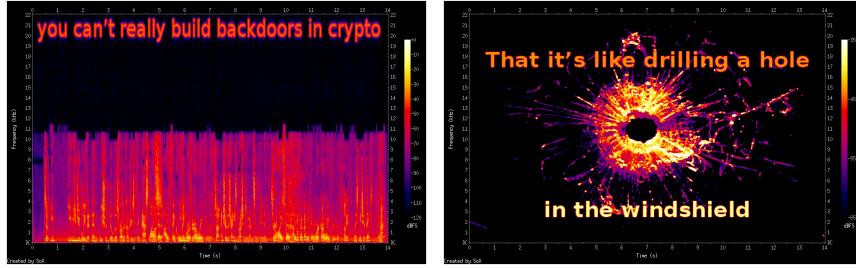


Figure 17: Two Sound Files Combined, with Spectral Images

And if you're cheeky, you can encode another picture in sound, that will be visible via spectrogram view. Or encode some actual sound, with a banner picture encoded in the higher frequencies; this way, the sound is still worth listening to yet also a thin picture is visible in the spectrogram view.<sup>26</sup>

**Sound and Sound** Not only can you combine a BMP and PCM together, you can also encode two different sound signals together by using different endianness and allowing the unchosen signal to drop beneath the noise floor.<sup>27</sup>

Figure 17 demonstrates a single file whose spectrogram is one image as big endian and a different image as little endian. Note that the text in the left interpretation is in inaudibly high frequencies, so it can peacefully coexist with music or speech in the lower frequencies.

**Two Kinds of Schizophrenic PNGs** In a similar way, by altering the Red/Green/Blue channels of each pixel, one gets a similar image but with extra information.

In naive steganography, this is often used to encode external data on the least significant bits, but we can also use this to encode one image within another image and create a schizophrenic picture!

Palettes image formats typically don't require that each color in the palette be unique. By duplicating the same sixteen colors over a 256-color palette, one can show the same image, but with extra information stored by whatever copy of the palette index is used. (Original idea by Dominique Bongard, re-implemented with Philippe Teuwen.)

<sup>26</sup>[http://wiki.yobi.be/wiki/BMP\\_PCM\\_polyglot](http://wiki.yobi.be/wiki/BMP_PCM_polyglot)

<sup>27</sup>[http://wiki.yobi.be/wiki/WAV\\_and\\_soft-boiled\\_eggs](http://wiki.yobi.be/wiki/WAV_and_soft-boiled_eggs)

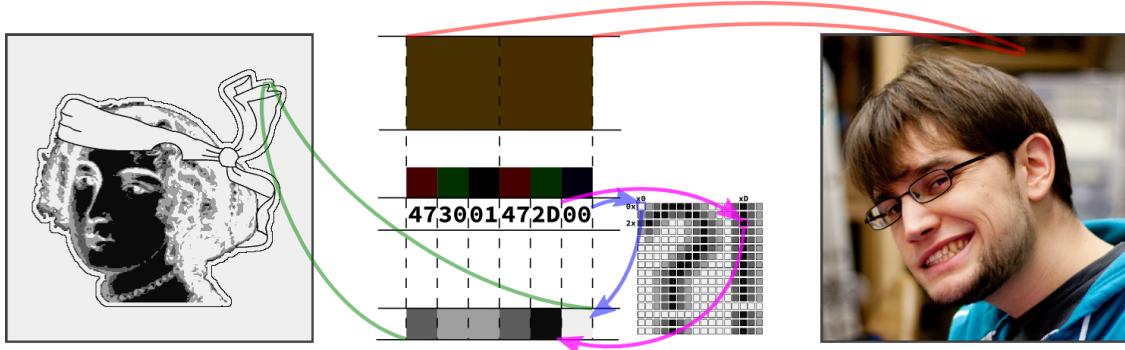


Figure 18: PNG with both Palette and RGB images from the Same Data

By combining both the redundant palette trick and the altered RGB components trick, we can store two images. One image appears when the palette is taken into account, and the other appears when the palette is ignored, and the raw RGB displayed.<sup>28</sup> Note that although an RGB picture with an extra palette isn't necessarily against the specs, there doesn't seem to be any legitimate example in the wild. (Perhaps this could be used to suggest which color to use to render on limited hardware?) As a bonus, the palette can contain itself a third image.

A related technique involves storing two 16-color pictures in the same data by illegally including two palettes. A PNG file having two palettes is *against* the specifications, but many viewers tolerate it. Some parsers take the first palette into account, and some the last, which leads to two different pictures from the same pixel information. This is shown in Figure 19, but unfortunately, most readers just reject the file. (Screenshot by Thijs Bosschert.)

## 6.6 Schizophrenia

### Semi-Constance

**Constant Obstacles Make People Take Shortcuts.** If most implementations use the same default value, then some developer might just use this value directly hardcoded. If a majority of developers do the same, then the variable aspect of the value would break compatibility too often, forcing the value to be constant, equal to its default. Already in DOS time, the keyboard buffer was supposed to be variable-sized<sup>(29)</sup>. It had a default start and size (40:1E, and 32 bytes), but you were supposed to be able to set a different head and tail via (40:1A and 40:1C). However, most people just hardcoded 40:1E, so the parameters for head and tail became not usable.

**BMP Data Pointer** A BMP's header contains a pointer to image data. However, most of the time, the image data strictly follows the headers and starts at offset 0x36. Consequently, some viewers just ignore that pointer and just incorrectly display the data at offset 0x36 without paying attention to the header length.

So, if you put two sets of data, one at the usual place, and one farther in the file, pointed at from the header, two readers may give different results. This trick comes from Gynvael Coldwind.

### Unbalanced Nested Markers

It's a well known fact that Web browsers don't enforce HTML markers correctly. A file containing only `a<b>c` will show a bold "c" despite the lack of `<html>` and `<body>` tags.

<sup>28</sup>[http://wiki.yobi.be/wiki/PNG\\_Merge](http://wiki.yobi.be/wiki/PNG_Merge)

<sup>29</sup>[http://stanislav.org/helpc/bios\\_data\\_area.html](http://stanislav.org/helpc/bios_data_area.html)

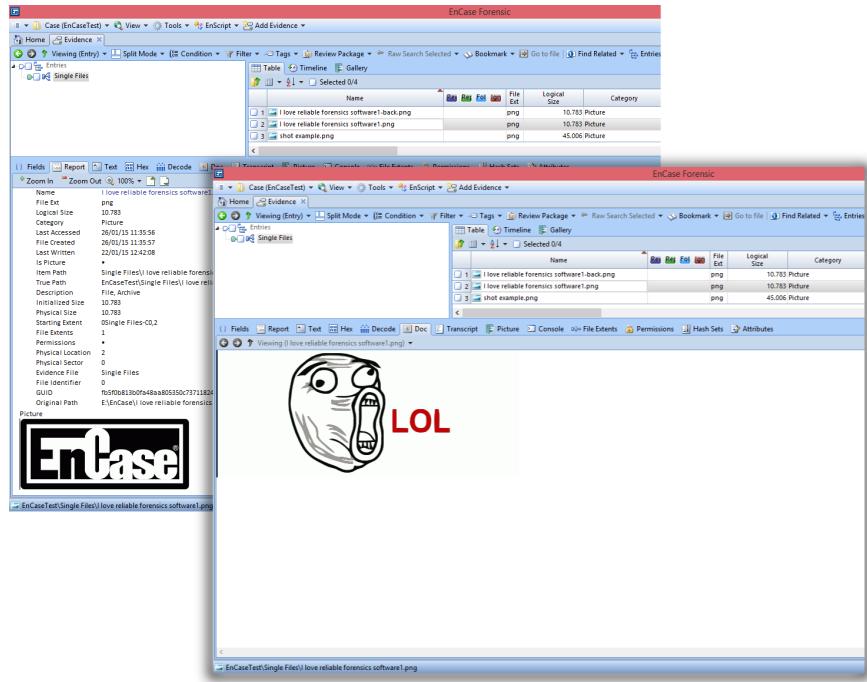


Figure 19: Schizophrenic PNG via Double Palettes, in EnCase Forensic v7

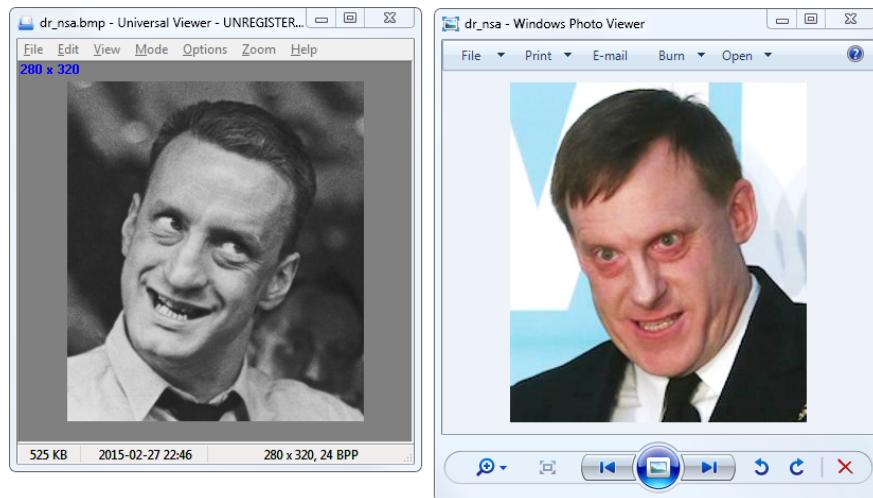


Figure 20: Schizophrenic BMP with Non-Default Data Pointer

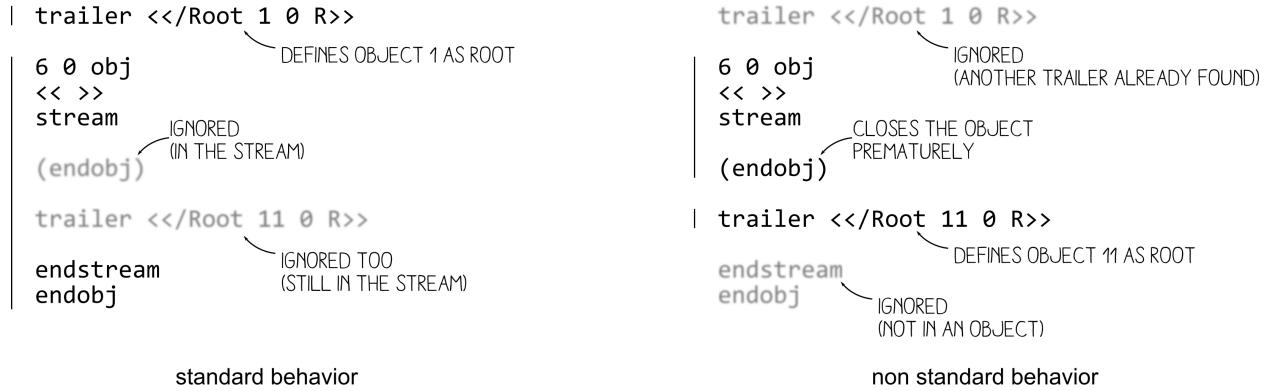


Figure 21: One PDF, Two Interpretations

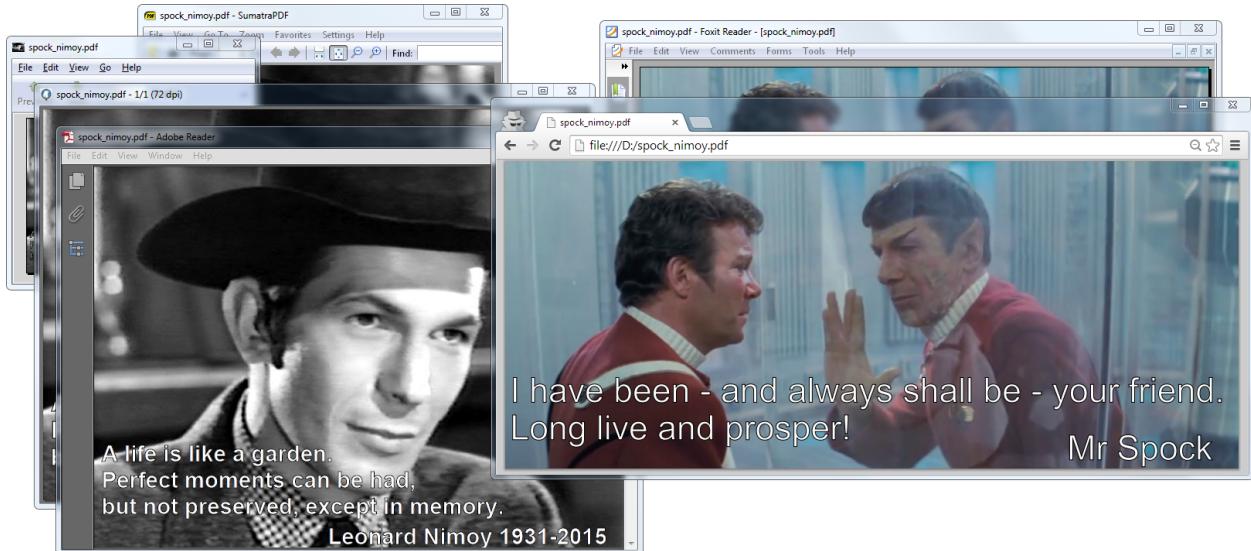


Figure 22: Schizophrenic PDF by Closed String Object (endobj)

In file formats with nested markers, ending these markers earlier than expected can have strange and lovely consequences.

For example, PDF files are made of objects. An object is required to end with `endobj`. Some of these objects contain a stream, which is required to end with `endstream`. As the stream is contained within the object, `endstream` is expected to always come first, and then `endobj`.

In theory, a stream can contain the keyword `endobj`, and that should not affect anything. However, in case some PDF generators should forget to close the stream before the object, it makes sense for a parser to close the object even if the stream hasn't been closed yet. Since this behavior is optional, different readers implement it in different ways.

This can be abused by creating a document that contains an object with a premature `endobj`. This sometimes confuses the parser by cloaking an extra root element different from the one defined in the trailer, as illustrated by Figure 21. Such a file will be displayed as a totally different document, depending upon the reader. Figure 22 shows such a schizophrenic PDF.

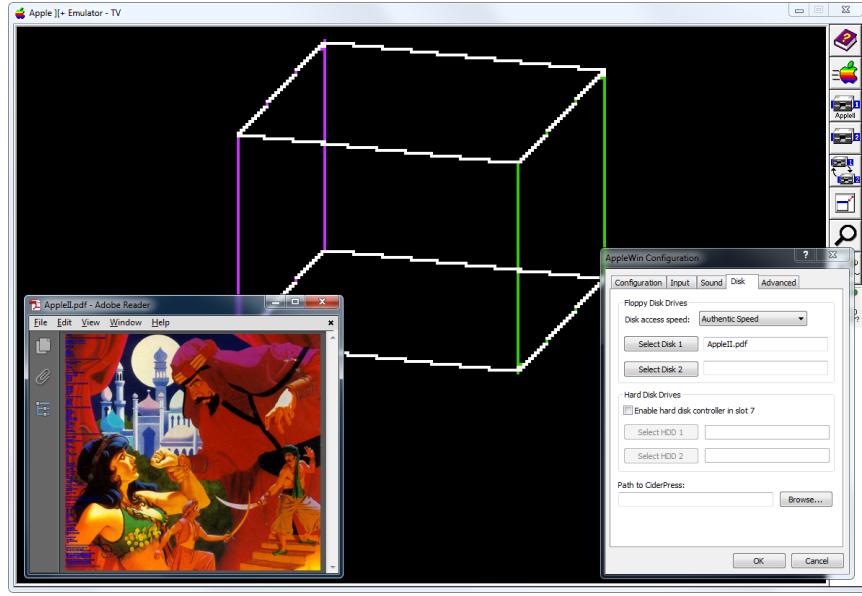


Figure 23: Apple II & PDF Polyglot

## 6.7 Icing on the Cake

After modifying a file, there are checksums and other limitations that must be observed. As with any other rule, there are exceptions, which we'll cover.

**ZIP CRC32** Most extractors enforce a ZIP file's checksums, but for some reason Java does not when reading JAR files. By corrupting the checksums of files within a JAR, you can make that JAR difficult to extract by standard ZIP tools.

**PNG CRC32** PNG also contains CRC32 checksums of its data. Although some viewers for Unix demand correct checksums, they are nearly never required on Windows. No warnings, no nothin'.

**TAR Checksum** Tar checksums aren't complicated, but the algorithm is so old-timey that it warms the heart just a little.

**Truecrypt Header** A Truecrypt disk's header is encrypted according to the chosen algorithm, password, and keyfile. Prior to the header, the disk begins with a random 64-byte salt, allowing for easy manipulation of headers. See my article on Truecrypt, PoC||GTFO 4:11, for a PDF/ZIP/Truecrypt polyglot.

## 6.8 Size Limitation

It's common that ROM and disk images require a specific rounded size, and there is often no workaround to this. You can merge a PDF and an Apple II floppy image, but only if the PDF fits in the 143360-byte disk image.

If you need a bigger size, you can try with hard disk images for the same system, if they exist. In this case, you can put them on a two megabyte hard disk image, with partitioning as required. Thanks to Peter Ferrie for his help with this technique, which was used to produce the polyglot in Figure 23. Shown in that figure is an Apple II disk image of Prince of Persia that doubles as a PDF.

## 6.9 Challenges

**Limitations of Standard Libraries** Because most libraries don't give you full control over the file structure, abusing file formats is not always easy.

You may want to open the file and just modify one chunk, but the library—too smart for its britches—removed your dummy chunk, recompressed your intentionally uncompressed data, optimized the colors of your palette, and ruined other carefully chosen options. In the end, such unconventional proofs of concept are often easier to generate with a small script made from scratch rather than relying on a well-known bulletproof library.

**Normalization** To make your scripts more efficient, it might be worth finding a good normalizer program for the filetype you're abusing. There are lots of good programs and libraries that will not modify your file in depth, but produce a relatively predictable structure.

For PDF, running `mutool clean` is a good way to sand off any rough edges in your polyglot. It modifies very little, yet rebuilds the XREF table and adjusts objects lengths, which turns your hand-made tolerated PDF into one that looks perfectly standard.

For PNG, `advpng -z -0` is a good way to produce an uncompressed image with no line filters.

For ZIP, `TorrentZip` is a good way to consistently produce the exact same archive file. `AdvDef` is a good way to (de)compress Zlib chunks without altering the rest of the file in any way. For example, when used on PNGs, no PNG structure is analyzed, and just the IDAT chunks are processed.

Normalizing the content data's range is sometimes useful, too. A sound or image that consumes its entire dynamic range leaves more room for hidden data in the lower bits.

## Compatibility

If your focus is still on getting decent compatibility, you may pull your hair a lot. The problem is not just the limit between valid and invalid files; rather, it's the difference between the parser thinking "Hey this is good enough." and "Hey, this looks corrupted so let's try to recover what I can."

This leads to bugs that are infuriatingly difficult to solve. For example, a single font in a PDF might become corrupted. One image—and only one image!—might go missing. A seemingly trivial polyglot then becomes a race against heisenbugs, where it can be very difficult to get a good compatibility rate.

## Automated Generation

Although it's possible to alter a generated file, it might be handy to make a file generator directly integrate foreign data. This way, the foreign data will be integrated reproducibly, whereas the rest of the structure is already one hundred percent standard.

**Archives** Archiving a file without any compression usually stores it as is. Please note, however, that some archive formats will escape data in order to prevent stored data from interfering with the outer format.

**PDFLATEX** PDFLATEX has special commands to create an uncompressed stream object, directly from an external file. This is extremely useful, and totally reliable, no matter the size of the file. This way, you can easily embed any data in your PDF.

```
2 \begingroup
  \pdfcompresslevel=0\relax
  \immediate\pdfobj stream
    file {foo.bin}
\endgroup
```

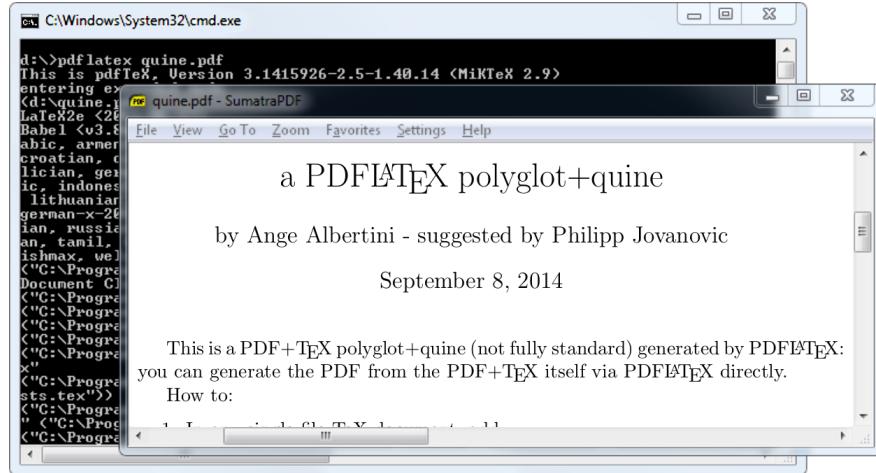


Figure 24: a PDFLATEX/PDF quine

**A PDFLATEX/PDF Polyglot** If your document’s source is a single `.tex` file, then you can make a PDFLATEX quine. This file is simultaneously its own TeX source code and the resulting PDF from compilation. If your document is made of multiple files, then you can archive those files to bundle them in the PDF.

You can also do it the other way around. For his Zeronights 2014 keynote, *Is infosec a game?*, Solar Designer created an actual point and click adventure to walk through the presentation.<sup>30</sup>

It would be a shame if such a masterpiece were lost, so he made his own walkthrough as screenshots, put together as a slideshow in a PDF, in which the ZIP containing the game is attached. This way, it’s preserved as a single file, containing an easy preview of the talk itself and the original presentation material.

**Embedding a ZIP in a PDF** However, if you embed a ZIP in a PDF as a simple PDF object, it’s possible that the ZIP footer will be too far from the end of the file. Objects are stored before the Cross Reference table, which typically grows linearly with the number of objects in the PDF. When this happens, ZIP tools might fail to see the ZIP.

A good way to embed a ZIP in a PDF, as Julia Wolf showed us with napkins in PoC||GTFO 1:5, is to create a fake stream object *after* the `xref`, where the trailer object is present, before the `startxref` pointer. The official specifications don’t specify that no extra object should be present. Since the trailer object itself is just a dictionary, it uses mostly the same syntax as any other PDF objects, and all parsers tolerate an extra object present within this area.

1. PDF Signature
2. PDF Objects
3. Cross Reference Table
4. (*extra stream object declaration*)
  - ZIP Archive
5. Trailer Object
6. `startxref` Pointer

---

<sup>30</sup><http://www.openwall.com/presentations/ZeroNights2014-Is-Infosec-A-Game/>

This gives a fully compatible PDF, with no need for pointer or length adjustment. It's also a straightforward way for academics to bundle source code and PoCs.

**Appended Data** If for some reason you need the ZIP at the exact bottom of the file, e.g. to maintain compatibility with Python's EGG format, then you can extend the ZIP footer's comment to cover the last bytes of the PDF. This footer, called the End of Central Directory, starts with P K 05 06 and ends with a variable length comment. The length is at offset 20, then the comment itself starts at offset 22.

If the ZIP is too far from the bottom of the file, then this operation is not possible as the comment would be longer than 65536 bytes. Instead, to increase compatibility, one can duplicate the End of Central Directory. I describe this trick in PoC||GTFO 4:11, where it was used to produce a Truecrypt/PDF/ZIP polyglot.

Combined with the trailing space trick explained earlier, one can insert an actual null-terminated string before the extraneous data so ZIP parsers will display a proper comment instead of some garbage!

**Fixing Absolute Pointers** When an unmodified ZIP is inserted into a PDF, the pointers inside the ZIP's structures are only valid relative to the start of the archive. They are not correct as seen from the file itself.

Some tools consider such a file to be damaged, with garbage to ignore, but some might refuse to parse it with incorrect addresses. To fix this, adjust the `relative offset of local header` pointers in the Central Directory's entries. You might also ask a ZIP tool to repair the file, and cross your fingers that your tool will not alter anything else in the file by reordering files or removing slack space.

## 6.10 Thoughts

**Polyglots** Polyglot files may sound like a great idea for production. For example, you can keep the original (custom format) source file of a document embedded in a file that can be seen as a preview in a standard format. To quickly sort your SVG files, just ZIP them individually and append them to a PNG showing the preview.

As mentioned previously, ZIP your `.tex` files and embed them in the final PDF. This already exists in some cases, such as OpenOffice's ability to export PDF files that contain the original `.odt` file internally.

A possible further use of polyglots would be to bundle different outputs of the same file in two different formats. PDF and EPUB could be combined for e-book distribution, or a installer could be used for both Linux and Windows. Naturally, we could just ZIP these together and distribute the archive, but they won't be usable out of the box.

Archiving files together is much more natural than making a polyglot file. Although opening a polyglot file may be transparent for the targeted software, it's not a natural action for user.

There are also security risks associated with polyglot construction. For example, polyglots can be used to exfiltrate data or bypass intrusion detection systems. Testing various polyglots on Encase showed that nearly all of them were reported as a single file type, with no warnings whatsoever.

**Offset Start** I see no point in allowing a magic signature to be at any offset. If it's for the sake of allowing a comment early in the file, then the format itself should have an explicit comment chunk.

If it's for the sake of bundling several file types together, then as mentioned previously, it could just be specific to one application. There's no need to waste parsing time in making it officially a part of one format. I don't see why a PE with ZIP in appended data should still be considered to be a standard ZIP; jumping at the end of the PE's physical size is not hard, neither is extracting a ZIP, so why does it sound normal that it still works directly as a ZIP? If a user updates the contents of the archive, it's quite possible that the ZIP tool would re-create an entire archive without the initial PE data.

While it's helpful to manually open WinZip/WinRAR/7Z self-extracting archives, you still have to run a dedicated tool for formats such as Nullsoft Installer and InnoSetup that have no standard tool. Sure, your extraction tool could just look for data anywhere like Binwalk, but this exceptional case doesn't justify the fact that the format explicitly allows any starting offset.

This is likely why some modern tools take a different approach, ignoring the official structure of a ZIP. These extractors start at offset zero and look for a sequence of Local File Headers. This method is faster than the official bottom-up method of parsing, and it works fine for 99% of standard files out there.

Sadly, doing this differently makes ZIP schizophrenia possible, which can be critical as it can break signatures and the complete chain of trust of a standard system.

And yet, how hard would it be to create a new, top-down, smaller Zlib-based archive format, one that doesn't contain obsolete fields such as **number of volumes of the archive?** One that doesn't duplicate file names between Central Directory and Local File Headers?

**Enforcing Values** File structures are like laws: when they are overly complicated and unnecessary, people will ignore them. The PE file format now has tons of deprecated fields and structures, especially by comparison to its long overdue sibling, the Terse Executable file format. TE is essentially the same format, with a lot of obsolete fields removed.

From especially unclear specifications come diverging implementations, slightly different for each programmer's interpretation. The ZIP specifications<sup>31</sup> don't even specify the names of the various fields in the structures, only a long description for each of them, such as **compression method!** Once enough diverging implementations survive, then hard reality merges them into an ugly de facto standard. We end up with tools that are forced to recover half-broken files rather than strictly accepting what's okay. They give us mere warnings when the input is unclear, rather than rejecting what's against the rules.

## 6.11 Conclusion

Let me know if I forgot anything. Suggestions and corrections are more than welcome! I hope this gives you ideas, that it makes you want to explore further. Our attentive readers will notice that compressions and file systems are poorly represented—except for the amazing MIT Mystery Hunt image—and indeed, that's what I will explore next.

Some people accuse these file format tricks of being pointless shenanigans, which is true! These tricks are useless, but only until someone uses one of them to bypass a security layer. At that point everyone will acknowledge that they were worth knowing before, but by then it's too late. It's better to know in advance about potential risks than judge blindly that 'nobody was ever pwned with such a trick'.

As a closing note, don't forget the two great mantras of research and security. First, to stay safe, don't do anything. Second, to make nifty new discoveries, try everything!

<sup>31</sup><https://pkware.cachefly.net/webdocs/APPNOTE/APPNOTE-6.3.3.TXT>

**VOTRAX ANNOUNCES  
VOTALKER IB and AP**

New Levels Of Voice Clarity And Versatility For Personal Computers

Unlimited Phonetic Speech For IBM PC, XT, AT, PS/2, Mac II, Mac Plus, And All True Compatible

Votalker boards are the only Synthetic Speech Generating Systems for Personal Computers that Provide Four Voice Patterns Through On-Board Switches. Both boards provide off-the-shelf preprogrammed voice modes and allow further customization through an on-board filter. Voice modes and filters are activated by switches.

**Other Speech Options:**

- Novel Design Circuit Board with Advanced SC-02 Speech Chip
- Sophisticated Text-to-Speech Translator Datasette
- SuperTalk™ Enhanced Software Operation

**Special Introductory Offer**

\$249 — Votalker IB For IBM PC and XT  
\$179 — Votalker AP for Apple II, Apple IIe, and Apple II Plus

**Other Votrax Products:**

- Disk Local Management System for IBM PC
- Parallel Speech System and Type 'N' Talk Stand-Alone Systems
- Votekit C-Kit for Commodore 64
- Votekit C-Kit for Commodore 68000
- SC-01 and SC-02 Speech Synthesis Chips

**VOTRAX**  
THE PIONEER IN SYNTHETIC SPEECH SYSTEMS

To place an order or learn more about Votalker IB and AP, Call Votrax at (800) 521-1360. In Michigan, Call collect (313) 586-0341.

**Super Cart™**

**CART**

**ATARI CARTRIDGE-TO-DISK COPY SYSTEM \$69\***

Supercart lets you copy ANY cartridge for the Atari 400/800 to diskette, and thereafter run it from your disk drive. Enjoy the convenience of selecting your favorite games from a "menu screen" rather than swapping cartridges in and out of your computer. Each cartridge copied by Supercart functions exactly like the original . . . and more!

**COPY ROUTINE** - Dumps the contents of the cartridge to a diskette (up to 8 cartridges will fit on one disk.)

**MENU ROUTINE** - Auto-loading menu prompts user for a **ONE** keystroke selection

**CARTRIDGE** - "Tricks" the computer into thinking that the original "protected" cartridge has been inserted.

To date there have been no problems duplicating and running all of the protected cartridges that we have on our system. We are sure that you will have no problems with your cartridges.

Supercart is user-friendly and simple to use. **PIRATES TAKE NOTES: SUPERCART IS NOT intended for illegal copying and/or distribution of copyrighted software . . . Sorry!!**

**SYSTEM REQUIREMENTS:**

- Atari 400 or 800 Computer
- At least 16K RAM
- One Disk Drive

Available at your computer store or directly from FRONTRUNNER. DEALER INQUIRIES ENCOURAGED

**TOLL FREE ORDER LINE: (24 Hrs) 1-800-648-7700** In Nevada, or for questions Call: (702) 795-4000

Personal checks allow 2-3 weeks to clear. M/C and VISA accepted

International Freight Forwarding and Shipping:  
**FRONTRUNNER COMPUTER INDUSTRIES**  
316 California Ave., Suite #712, Reno, Nevada 89503 - (702) 795-4800

**Others Make Claims . . . SUPERCART makes copies!!!**

**TRY THIS ON A STANDARD DESKTOP PUBLISHING SYSTEM**

$$\int_{-\infty}^{nZ} \frac{X^y Z^z}{X - tY} dt = \sum_{i=1}^n \left( \frac{(x^4 - i)^3}{\sqrt{Z^2 + y^{10}}} \right)$$

**PC** is professional looking numbers—no matter how complex PC is integrated. That's why many university engineering departments are switching to PC for their desktop publishing needs. In fact, the first PCTEX #1, a very non-PC based program has just been released for the PC. It's called *TYPESET*. And from PC ASIMONE™ . . . You can achieve incredible precision in creating **ANY** typeface using mathematical expressions.

**TYPESET & MANUALS**  
The FORMA is different  
BIG  
BY YOUR FOLD  
INFORMATION  
TRY IT WITH PERSONAL  
LNC

WHERE YOU CAN GET THE FORMA  
CONT'D ON BACK PAGE  
NUMBERS  
Frontier with the formula? Then it's time to move to the BIG. In the world of desktop publishing systems, nothing is more powerful than PCTEX. And *TYPESET* is the most advanced PCTEX compatible system available. And from PC ASIMONE™ . . . You can achieve incredible precision in creating **ANY** typeface using mathematical expressions.

INFOGRAPHIC: "Extremely difficult to read and understand the output of your printer?" And with Biomax's BIG font families, the answer is NO. With the FORMA, the output is also extremely difficult to read and understand. And from PC ASIMONE™ . . . You can achieve incredible precision in creating **ANY** typeface using mathematical expressions.

PCTEX™ is the professional carriage-ready, print-on-the-fly, page layout system.

PCTEX™ SUBTRACTS TIME FROM  
MATERIALS TO PRINTED PAPER.  
MATERIALS TO PRINTED PAPER.

PC TEX is a registered trademark of Frontier Corporation. PC ASIMONE is a registered trademark of ASIMONE Corporation. No. 1 is a registered trademark of ASIMONE Corporation.

## 7 Extending crypto-related backdoors to other scenarios

by BSDaemon and Pirata

This article expands on the ideas introduced by Taylor Hornby's "Prototyping an RDRAND Backdoor in Bochs" in PoC||GTFO 3:6. That article demonstrated the dangers of using instructions that generate a #VMEXIT event while in a guest virtual machine. Because a malicious VMM could compromise the randomness returned to a guest VM, it can affect the security of cryptographic operations.

In this article, we demonstrate that the newly available AES-NI instruction extensions in Intel platforms are vulnerable to a similar attack, with some additional badness. Not only guest VMs are vulnerable, but normal user-level/kernel-level applications that leverage the new instruction set are vulnerable as well, unless proper measures are in place. The reason for that is due to a mostly unknown feature of the platform, the ability to disable this instruction set.

### 7.1 Introduction

From Intel's website,<sup>32</sup>:

Intel AES-NI is a new encryption instruction set that improves on the Advanced Encryption Standard (AES) algorithm and accelerates the encryption of data in the Intel Xeon processor family and the Intel Core processor family.

The instruction has been available since 2010.<sup>33</sup>

Starting in 2010 with the Intel Core processor family based on the 32nm Intel micro-architecture, Intel introduced a set of new AES (Advanced Encryption Standard) instructions. This processor launch brought seven new instructions. As security is a crucial part of our computing lives, Intel has continued this trend and in 2012 and [sic] has launched the 3rd Generation Intel Core Processors, codenamed Ivy Bridge. Moving forward, 2014 Intel micro-architecture code name Broadwell will support the RDSEED instruction.

On a Linux box, a simple `grep` would tell if the instruction is supported in your machine.

```
1 bsddaemon@bsddaemon.org:~# grep aes /proc/cpuinfo
2 flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
3 pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm
4 constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc
5 aperfmpf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3
6 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx
7 f16c rdrand lahf_lm ida arat epb xsaveopt pln pts dtherm tpr_shadow vnmi
     flexpriority ept vpid fsgsbase smep erms
```

A little-known fact, though, is that the instruction set can be disabled using an internal MSR on the processor. It came to our attention while we were looking at BIOS update issues and saw a post about a machine with AES-NI showing as disabled even though it was in, fact, supported.<sup>34</sup>

Researching the topic, we came across the MSR for a Broadwell Platform: `0x13C`. It will vary for each processor generation, but it is the same in Haswell and SandyBridge, according to our tests. Our machine had it locked.

MSR 0x13C
2 Bit Description
0 Lock bit (always unlocked on boot time, BIOS sets it)
1 Not defined by default, 1 will disable AES-NI
2-32 Not sure what it does, not touched by our BIOS (probably reserved)

Discussing attack possibilities with a friend in another scenario—related to breaking a sandbox-like feature in the processor—we came to the idea of using it for a rootkit.

<sup>32</sup><http://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard-aes-/data-protection-aes-general-technology.html>

<sup>33</sup><https://software.intel.com/en-us/node/256280>.

<sup>34</sup>"AES-NI shows Disabled", <http://en.community.dell.com/support-forums/servers/f/956/t/19509653>

## 7.2 The Idea

All the code that we saw that supports AES-NI is basically about checking if it is supported by the processor, via CPUID, including the reference implementations on Intel's website. That's why we considered the possibility of manipulating encryption in applications by disabling the extension and emulating its expected results. Not long after we had that thought, we read in the PoC||GTFO 3:6 about **RDRAND**.

If the disable bit is set, the AES-NI instructions will return #UD (Invalid Opcode Exception) when issued. Since the code checks for the AES-NI support during initialization instead of before each call, winning the race is easy—it's a classic TOCTOU.

Some BIOSes will set the lock bit, thus hard-enabling the set. A write to the locked MSR then causes a general protection fault, so there are two possible approaches to dealing with this case.

First, we can set *both* the disable bit *and* the lock bit. The BIOS tries to enable the instruction, but that write is ignored. The BIOS tries to lock it, but it is ignored. That works unless the BIOS checks if the write to the MSR worked or not, which is usually not the case—in the BIOS we tested, the general protection fault handler for the BIOS just resumed execution. For beating the BIOS to this punch, one could explore the BIOS update feature, setting the **TOP\_SWAP** bit, which let code execute *before* BIOS.<sup>35</sup> Chipsec toolkit<sup>36</sup> **TOP\_SWAP** mechanism is locked.

For a Vulnerable Machine,

```

1 ##### BIOS VERSION 65CN90WW
OS      : uefi
3 Chipset:
VID:     8086
5 DID:    0154
Name:   Ivy Bridge (IVB)
7 Long Name: Ivy Bridge CPU / Panther Point PCH
[-] FAILED: BIOS Interface including Top Swap Mode is not locked

```

For a Protected Machine,

```

2 OS      : Linux 3.2.0-4-686-pae #1 SMP Debian 3.2.65-1+deb7u2 i686
Platform: 4th Generation Core Processor (Haswell U/Y)
        VID: 8086
4        DID: 0A04
CHIPSEC : 1.1.7
6 [*] BIOS Top Swap mode is disabled
[*] BUC = 0x00000000 << Backed Up Control (RCBA + 0x3414)
8 [00] TS      = 0 << Top Swap
[*] RTC version of TS = 0
10 [*] GCS = 0x00000021 << General Control and Status (RCBA + 0x3410)
[00] BILD    = 1 << BIOS Interface Lock Down
12 [10] BBS     = 0
14 [+]
[+] PASSED: BIOS Interface is locked (including Top Swap Mode)

```

The problem with this approach is that software has to check if the AES-NI is enabled or not, instead of just assuming the platform supports it.

Second, we can NOP-out the BIOS code that locks the MSR. That works if BIOS modification is possible on the platform, which is often the case. There are many options to reverse and patch your BIOS, but most involve either modifying the contents of the SPI Flash chip or single-stepping with a JTAG debugger.

Because the CoreBoot folks have had all the fun there is with SPI Flash, and because folk wisdom says that JTAG isn't feasible on Intel, we decided to throw folk wisdom out the window and go the JTAG route. We used the Intel JTAG debugger and an XDP 3 device. The algorithm used is provided in the attachment 3.

To be able to set this MSR, one needs Ring0 access, so this attack can be leveraged by a hypervisor against a guest virtual machine, similar to the **RDRAND** attack. But what's interesting in this case is that it can also be leveraged by a Ring0 application against a hypervisor, guest, or any host application! We used a Linux Kernel Module to intercept the #UD; a sample prototype of that module is in attachment 6.

<sup>35</sup>"Using SMM for other purposes", Phrack 65:7

<sup>36</sup><https://github.com/chipsec/chipsec>

## 7.3 Checking your system

You can use the Chipsec module that comes with this article to check if your system has the MSR locked. Chipsec uses a kernel module that opens an interface (a device on Linux) for its user-mode component (Python code) to request info on different elements of the platform, such as MSRs. Obviously, a kernel module could do that directly. An example of such a module is provided with this article.

Since the MSR seems to change from system to system (and is not deeply documented by Intel itself), we recommend searching your OEM BIOS vendor forums to try and guess what is that MSR's number for your platform if the value mentioned here doesn't work. Disassembling your BIOS calls for the `wrmsr` might also help. Some BIOSes offer the possibility of disabling the AES-NI set in the BIOS menu, thus making it easier to identify the code (so dump the BIOS and diff). By default, the platform initializes with the disable bit unset, i.e., with AES-NI enabled. In our case, the BIOS vendor only set the lock bit.

## 7.4 Conclusion

This article demonstrates the need for checking the platform as whole for security issues. We showed that even "safe" software can be compromised, if the configuration of the platform's elements is wrong (or not ideal). Also note that forensics tools would likely fail to detect these kinds of attacks, since they typically depend on the platform's help to dissect software.

## Acknowledgements

Neer Roggel for many excellent discussions on processor security and modern features, as well for the enlightening conversation about another attack based on disabling the AES-NI in the processor.

## Attachment 1: Patch for Chipsec

This patch is for Chipsec (<https://github.com/chipsec/chipsec>) public repository version from March 9, 2015. A better (more complete) version of this patch will be incorporated into the public repository soon.

```
diff -rNup chipsec-master/source/tool/chipsec/cfg/hsw.xml chipsec-master.new/source/tool/chipsec/
      cfg/hsw.xml
2 --- chipsec-master/source/tool/chipsec/cfg/hsw.xml 2015-01-23 16:07:19.000000000 -0800
+++ chipsec-master.new/source/tool/chipsec/cfg/hsw.xml 2015-03-09 19:13:55.949498250 -0700
4 @@ -39,6 +39,10 @@
<!-->
6 <!-- ##### -->
<registers>
8   <register name="IA32_AES_NI" type="msr" msr="0x13c" desc="AES-NI Lock">
+     <field name="Lock" bit="0" size="1" desc="AES-NI Lock Bit" />
10    <field name="AESDisable" bit="1" size="1" desc="AES-NI Disable Bit (set to disable)" />
+   </register>
12 </registers>
14 </configuration>
\ No newline at end of file
16 </configuration>
diff -rNup chipsec-master/source/tool/chipsec/modules/hsw/aes_ni.py chipsec-master.new/source/tool/
      /chipsec/modules/hsw/aes_ni.py
18 --- chipsec-master/source/tool/chipsec/modules/hsw/aes_ni.py 1969-12-31 16:00:00.000000000 -0800
+++ chipsec-master.new/source/tool/chipsec/modules/hsw/aes_ni.py 2015-03-09 19:22:12.693518998
-0700
20 @@ -0,0 +1,68 @@
+CHIPSEC: Platform Security Assessment Framework
22 +#Copyright (c) 2010-2015, Intel Corporation
+#
24+#This program is free software; you can redistribute it and/or
+#+modify it under the terms of the GNU General Public License
26+#as published by the Free Software Foundation; Version 2.
+#
28+#This program is distributed in the hope that it will be useful,
+#+but WITHOUT ANY WARRANTY; without even the implied warranty of
30+#MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+#+GNU General Public License for more details.
```

```

32+#
33+#+You should have received a copy of the GNU General Public License
34+#+along with this program; if not, write to the Free Software
35+#+Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
36+#
37+#+Contact information:
38+#+chipsec@intel.com
39+#
40+
41+
42+
43+
44+### \addtogroup modules
45+## --chipsec/modules/hsw/aes_ni.py -- checks for AES-NI lock
46+#
47+
48+
49+from chipsec.module_common import *
50+from chipsec.hal.msr import *
51+
52+TAGS = [MTAG_BIOS,MTAG_HWCONFIG]
53+
54+class aes_ni(BaseModule):
55+
56+    def __init__(self):
57+        BaseModule.__init__(self)
58+
59+    def is_supported(self):
60+        return True
61+
62+    def check_aes_ni_supported(self):
63+        return True
64+
65+    def check_aes_ni(self):
66+        self.logger.start_test( "Checking if AES-NI lock bit is set" )
67+
68+        aes_msr = chipsec(chipset.read_register( self.cs, 'IA32_AES_NI' )
69+                          chipsec(chipset.print_register( self.cs, 'IA32_AES_NI', aes_msr ) )
70+
71+        aes_msr_lock = aes_msr & 0x1
72+
73+        # We don't really care if it is enabled or not since the software needs to
74+        # test - the only security issue is if it is not locked
75+        aes_msr_disable = aes_msr & 0x2
76+
77+        # Check if the lock is not set, then ERROR
78+        if (not aes_msr_lock):
79+            return False
80+
81+        return True
82+
83+    #
84+    # run( module_argv )
85+    # Required function: run here all tests from this module
86+    #
87+    def run( self, module_argv ):
88+        return self.check_aes_ni()

```

## Attachment 2: Kernel Module to check and set the AES-NI related MSRs

If for some reason you can't use Chipsec, this Linux kernel module reads the MSR and checks if the AES-NI lock bit is set.

```

1 #include <linux/module.h>
2 #include <linux/device.h>
3 #include <linux/highmem.h>
4 #include <linux/kallsyms.h>
5 #include <linux/tty.h>
6 #include <linux/ptrace.h>
7 #include <linux/version.h>
8 #include <linux/slab.h>
9 #include <asm/io.h>
10 #include "include/rop.h"
11 #include <linux/smp.h>

```

```

12 #define _GNU_SOURCE
14 #define FEATURE_CONFIG_MSR 0x13c
16 MODULE_LICENSE("GPL");
18 #define MASK_LOCK_SET 0x00000001
20 #define MASK_AES_ENABLED 0x00000002
21 #define MASK_SET_LOCK 0x00000000
22
23 void * read_msr_in_c(void * CPUInfo)
24 {
25     int *pointer;
26     pointer=(int *) CPUInfo;
27     asm volatile("rdmsr" : "=a"(pointer[0]), "=d"(pointer[3]) : "c"(FEATURE_CONFIG_MSR));
28     return NULL;
29 }
30
31 int __init
32 init_module (void)
33 {
34     int CPUInfo[4]={-1};
35
36     printk(KERN_ALERT "AES-NI testing module\n");
37
38     read_msr_in_c(CPUInfo);
39
40     printk(KERN_ALERT "read: %d %d from MSR: 0x%08x \n", CPUInfo[0], CPUInfo[3], FEATURE_CONFIG_MSR);
41
42     if (CPUInfo[0] & MASK_LOCK_SET)
43         printk(KERN_ALERT "MSR: lock bit is set\n");
44
45     if (!(CPUInfo[0] & MASK_AES_ENABLED))
46         printk(KERN_ALERT "MSR: AES_DISABLED bit is NOT set - AES-NI is ENABLED\n");
47
48     return 0;
49 }
50
51 void __exit
52 cleanup_module (void)
53 {
54     printk(KERN_ALERT "AES-NI MSR unloading \n");
55 }
```

### Attachment 3: In-target-probe (ITP) algorithm

Since we used an interface available only to Intel employees and OEM partners, we decided to at least provide the algorithm behind what we did. We started with stopping the machine execution at the BIOS entrypoint. We then defined some functions to be used through our code.

```

1 get_eip(): Get the current RIP
2 get_cs(): Get the current CS
3 get_ecx(): Get the current value of RCX
4 get_opcode(): Get the current opcode (disassembly the current instruction)
5 find_wrmsr(): Uses the get_opcode() to compare with the '300f' (wrmsr opcode) and
6     return True if found (False if not)
7 search_wrmsr():
8     while find_wrmsr() == False: step() -> go to the next instruction (single-step)
9 find_aes():
10    while True:
11        step()
12        search_wrmsr()
13        if get_ecx() == '0000013c':
14            print "Found AES MSR"
15            break
```

### Attachment 4: AES-NI Availability Test Code

This code uses the CPUID feature to see if AES-NI is available. If disabled, it will return “AES-NI Disabled”. This is the reference code to be used by software during initialization to probe for the feature.

```

1 #include <stdio.h>
3 #define cpuid(level, a, b, c, d) \
5     asm("xchg{l}\t{\%}\{ebx, %1\n\t" \
6         "cpuid\n\t" \
7         "xchg{l}\t{\%}\{ebx, %1\n\t" \
8         : "=a" (a), "=r" (b), "=c" (c), "=d" (d) \
9         : "0" (level))
11 int main (int argc, char **argv) {
12     unsigned int eax, ebx, ecx, edx;
13     cpuid(1, eax, ebx, ecx, edx);
14     if (ecx & (1<<25))
15         printf("AES-NI Enabled\n");
16     else
17         printf("AES-NI Disabled\n");
18     return 0;
}

```

## Attachment 5: AES-NI Simple Assembly Code (to trigger the #UD)

This code will run normally (`exit(0)` call) if AES-NI is available and will cause a #UD if not.

```

Section .text
2     global _start
4 _start:
5     mov ebx, 0
6     mov eax, 1
7     aesenc xmm7, xmm1
8     int 0x80

```

## Attachment 6: #UD hooking

There are many ways to implement this, as ‘Handling Interrupt Descriptor Table for fun and profit’ in Phrack 59:4 shows. Another option, however, is to use Kprobes and hook the function `invalid_op()`.

```

#include <linux/module.h>
2 #include <linux/kernel.h>
4 int index = 0;
module_param(index, int, 0);
6
#define GET_FULL_ISR(low, high) ( ((uint32_t)(low)) | (((uint32_t)(high)) << 16) )
8 #define GET_LOW_ISR(addr) ( (uint16_t)((uint32_t)(addr)) & 0x0000FFFF )
#define GET_HIGH_ISR(addr) ( (uint16_t)((uint32_t)(addr)) >> 16) )
10
12 uint32_t original_handlers[256];
13 uint16_t old_gs, old_fs, old_es, old_ds;
14
15 typedef struct idt_gate_desc {
16     uint16_t offset_low;
17     uint16_t segment_selector;
18     uint8_t zero; // zero + reserved
19     uint8_t flags;
20     uint16_t offset_high;
21 } idt_gate_desc_t;
22 idt_gate_desc_t *gates[256];
23
24 void handler_implemented(void) {
25     printk(KERN_EMERG "IDT Hooked Handler\n");
26 }
27
28 void foo(void) {
29     __asm__("push %eax"); // placeholder for original handler
30
31     __asm__("pushw %gs");
32     __asm__("pushw %fs");
33     __asm__("pushw %es");
34     __asm__("pushw %ds");
35     __asm__("push %eax");

```

```

36     __asm__( "push %ebp");
37     __asm__( "push %edi");
38     __asm__( "push %esi");
39     __asm__( "push %edx");
40     __asm__( "push %ecx");
41     __asm__( "push %ebx");
42     __asm__( "movw %0, %%ds" : : "m"(old_ds));
43     __asm__( "movw %0, %%es" : : "m"(old_es));
44     __asm__( "movw %0, %%fs" : : "m"(old_fs));
45     __asm__( "movw %0, %%gs" : : "m"(old_gs));
46
47     handler_implemented();
48
49     // place original handler in its placeholder
50     __asm__( "mov %0, %%eax": : "m"(original_handlers[index]));
51     __asm__( "mov %eax, 0x24(%esp)");
52
53     __asm__( "pop %ebx");
54     __asm__( "pop %ecx");
55     __asm__( "pop %edx");
56     __asm__( "pop %esi");
57     __asm__( "pop %edi");
58     __asm__( "pop %ebp");
59     __asm__( "pop %eax");
60     __asm__( "popw %ds");
61     __asm__( "popw %es");
62     __asm__( "popw %fs");
63     __asm__( "popw %gs");
64
65     // ensures that "ret" will be the next instruction for the case
66     // compiler adds more instructions in the epilogue
67     __asm__( "ret");
68 }
69
70 int init_module(void) {
71     // IDTR
72     unsigned char idtr[6];
73     uint16_t idt_limit;
74     uint32_t idt_base_addr;
75     int i;
76
77     __asm__( "mov %%gs, %0": "=m"(old_gs));
78     __asm__( "mov %%fs, %0": "=m"(old_fs));
79     __asm__( "mov %%es, %0": "=m"(old_es));
80     __asm__( "mov %%ds, %0": "=m"(old_ds));
81
82     __asm__( "sidt %0": "=m" (idtr));
83     idt_limit = *((uint16_t *)idtr);
84     idt_base_addr = *((uint32_t *)&idtr[2]);
85     printk("IDT Base Address: 0x%08x, IDT Limit: 0x%08x\n", idt_base_addr, idt_limit);
86
87     gates[0] = (idt_gate_desc_t *) (idt_base_addr);
88     for (i = 1; i < 256; i++)
89         gates[i] = gates[i - 1] + 1;
90
91     printk("int %d entry addr %08x, seg sel %08x, flags %08x, offset %08x\n", index, gates[index], (uint32_t)gates[index]->segment_selector, (uint32_t)gates[index]->flags, GET_FULL_ISR(gates[index]->offset_low, gates[index]->offset_high));
92
93     for (i = 0; i < 256; i++)
94         original_handlers[i] = GET_FULL_ISR(gates[i]->offset_low, gates[i]->offset_high);
95
96     gates[index]->offset_low = GET_LOW_ISR(&foo);
97     gates[index]->offset_high = GET_HIGH_ISR(&foo);
98
99     return 0;
100 }
101
102 void cleanup_module(void) {
103     printk("cleanup entry %d\n", index);
104
105     gates[index]->offset_low = GET_LOW_ISR(original_handlers[index]);
106     gates[index]->offset_high = GET_HIGH_ISR(original_handlers[index]);
107 }

```

## 8 Innovations with Linux core files for advanced process forensics

by Ryan O'Neill,  
who also publishes as Elfmaster

### 8.1 Introduction

It has been some time since I've seen any really innovative steps forward in process memory forensics. It remains a somewhat arcane topic, and is understood neither widely nor in great depth. In this article I will try to remedy that, and will assume that the readers already have some background knowledge of Linux process memory forensics and the ELF format.

Many of us have been frustrated by the near-uselessness of Linux (ELF) core files for forensics analysis. Indeed, these files are only useful for debugging, and only if you also have the original executable that the core file was dumped from during crash time. There are some exceptions such as `/proc/kcore` for kernel forensics, but even `/proc/kcore` could use a face-lift. Here I present *ECFS*, a technology I have designed to remedy these drawbacks.

### 8.2 Synopsis

ECFS (Extended core file snapshots) is a custom Linux core dump handler and snapshot utility. It can be used to plug directly into the core dump handler by using the IPC functionality available by passing the pipe '|' symbol in the `/proc/sys/kernel/core_pattern`. ECFS can also be used to take an *ecfs-snapshot* of a process without killing the process, as is often desirable in automated forensics analysis for whole-system process scanning. In this paper, I showcase ECFS in a series of examples as a means of demonstrating its capabilities. I hope to convince you how useful these capabilities will be in modern forensics analysis of Linux process images—which should speak to all forms of binary and process-memory malware analysis. My hope is that ECFS will help revolutionize automated detection of process memory anomalies.

ECFS creates files that are backward-compatible with regular core files but are also prolific in new features, including section headers (which core files do not have) and many *new* section headers and section header types. ECFS includes full symbol table reconstruction for both `.dynsym` and `.syms` symbol tables. Regular core files do not have section headers or symbol tables (and rely on having the original executable for such things), whereas an *ecfs-core* contains everything a forensics analyst would ever want, in one package.

Since the object and `readelf` output of an *ecfs-core* file is huge, let us examine a simple *ecfs-core* for a 64-bit ELF program named `host`. The process for `host` will show some signs of virus memory infection or backdooring, which ECFS will help bring to light.

The following command will set up the kernel core handler so that it pipes core files into the `stdin` of our core-to-ecfs conversion program named `ecfs`.

```
# echo '|/opt/ecfs/bin/ecfs -i -e %e -p %p -o /opt/ecfs/cores/%e.%p' > /proc/sys/kernel/core_pattern
```

Next, let's get the kernel to dump an *ecfs* file of the process for `host`, and then begin analyzing this file.

```
1 $ kill -11 `pidof host`
```

### 8.3 Section header reconstruction example

```
1 $ readelf -S /opt/ecfs/cores/host.10710
```

There are 40 section headers, starting at offset 0x23ffff0:

Section Headers:						
[Nr]	Name	Type	Address	Offset	Flags	Link Info Align
	Size	EntSize				
[ 0]		NULL	0000000000000000	0000000000000000	0	0 0 0
[ 1]	.interp	PROGBITS	0000000000400238	00002238		
[ 2]	.note	NOTE	0000000000000000	000004a0	A 0 0 1	
[ 3]	.hash	GNU_HASH	0000000000400298	00002298		
[ 4]	.dynsym	DYNSYM	00000000004002b8	000022b8		
[ 5]	.dynstr	STRTAB	0000000000400360	00002360		
[ 6]	.rela.dyn	RELA	00000000004003e0	000023e0		
[ 7]	.rela.plt	RELA	00000000004003f8	000023f8		
[ 8]	.init	PROGBITS	0000000000400488	00002488		
[ 9]	.plt	PROGBITS	00000000004004b0	000024b0		
[10]	.text	PROGBITS	0000000000400000	00002000		
[11]		PROGBITS	0000000000400000	00002000		
[12]	.eh_frame_hdr	PROGBITS	0000000000400724	00002724		
[13]	.eh_frame	PROGBITS	0000000000400758	00002758		
[14]	.dynamic	DYNAMIC	000000000040078c	00002790		
[15]	.got.plt	PROGBITS	000000000040078c	00002790		
[16]	.data	PROGBITS	0000000000600000	00003000		
[17]	.bss	PROGBITS	0000000000601000	00004000		
[18]	.heap	PROGBITS	0000000000601000	00004000		
[19]	ld-2.19.so.text	SHLIB	0000000000601058	00004058		
[20]	ld-2.19.so.relro	SHLIB	0000000000601058	00004058		
[21]	ld-2.19.so.data.0	SHLIB	0000000000601058	00004058		
[22]	libc-2.19.so.text	SHLIB	0000000000601058	00004058		
[23]	libc-2.19.so.unde	SHLIB	0000000000601058	00004058		
[24]	libc-2.19.so.relr	SHLIB	0000000000601058	00004058		
[25]	libc-2.19.so.data	SHLIB	0000000000601058	00004058		
[26]	evil_lib.so.text	INJECTED	0000000000601058	00004058		
[27]	.prstatus	PROGBITS	0000000000601058	00004058		
[28]	.fdinfo	PROGBITS	0000000000601058	00004058		
[29]	.siginfo	PROGBITS	0000000000601058	00004058		
[30]	.auxvector	PROGBITS	0000000000601058	00004058		
[31]	.exepath	PROGBITS	0000000000601058	00004058		
[32]	.personality	PROGBITS	0000000000601058	00004058		
[33]	.arglist	PROGBITS	0000000000601058	00004058		
[34]	.stack	PROGBITS	0000000000601058	00004058		
[35]	.vdso	PROGBITS	0000000000601058	00004058		

```

77 [36] .vsyscall PROGBITS fffffffffff600000 0023e000
    0000000000001000 0000000000000000 WA 0 0 8
79 [37] .symtab SYMTAB 0000000000000000 00240b81
    0000000000000078 0000000000000018 38 0 4
81 [38] .strtab STRTAB 0000000000000000 00240bf9
    0000000000000037 0000000000000000 0 0 1
83 [39] .shstrtab STRTAB 0000000000000000 002409f0
    000000000000191 0000000000000000 0 0 1

```

As you can see, there are even more section headers in our *ecfs-core* file than in the original executable itself. This means that you can disassemble a complete process image with simple tools that rely on section headers such as `objdump`! Also, please note this file is entirely usable as a regular core file; the only change you must make to it is to mark it from `ET_NONE` to `ET_CORE` in the initial ELF file header. The reason it is marked as `ET_NONE` is that `objdump` would know to utilize the section headers instead of the program headers.

```

1 $ tools/et_flip host.107170 <- this command flips e_type from ET_NONE to ET_CORE (And vice versa)
$ gdb -q host host.107170
3 [New LWP 10710]
Core was generated by `ecfs_tests/host'.
5 Program terminated with signal SIGSEGV, Segmentation fault.
#0 0x00007fb0358c375a in ?? ()
7 (gdb) bt
#0 0x00007fb0358c375a in ?? ()
9 #1 0x00007fff51da1580 in ?? ()
#2 0x00007fb0358c3790 in ?? ()
11 #3 0x0000000000000000 in ?? ()

```

For the remainder of this paper we will not be using traditional core file functionality. However, it is important to know that it's still available.

So what new sections do we see that have never existed in traditional ELF files? Well, we have sections for important memory segments from the process that can be navigated by name with section headers. Much easier than having to figure out which program header corresponds to which mapping!

```

1 [18] .heap PROGBITS 000000000093b000 00005000
    000000000021000 0000000000000000 WA 0 0 8
3 [34] .stack PROGBITS 00007fff51d82000 00000000
    00000000000021000 0000000000000000 WA 0 0 8
5 [35] .vdso PROGBITS 00007fff51df0000 0023c000
    0000000000002000 0000000000000000 WA 0 0 8
7 [36] .vsyscall PROGBITS fffffffffff600000 0023e000
    0000000000001000 0000000000000000 WA 0 0 8

```

Also notice that there are section headers for every mapping of each shared library. For instance, the dynamic linker is mapped in as it usually is:

```

2 [19] ld-2.19.so.text SHLIB 0000003000000000 00026000
    0000000000023000 0000000000000000 A 0 0 8
4 [20] ld-2.19.so.relro SHLIB 0000003000222000 00049000
    0000000000001000 0000000000000000 A 0 0 8
6 [21] ld-2.19.so.data.0 SHLIB 0000003000223000 0004a000
    0000000000001000 0000000000000000 A 0 0 8

```

Also notice the section type is `SHLIB`. This was a reserved type specified in the ELF man pages that is never used, so I thought this to be the perfect opportunity for it to see some action. Notice how each part of the shared library is given its own section header: `<lib>.text` for the code segment, `<lib>.relro` for the read-only page to help protect against `.got.plt` and `.dtors` overwrites, and `<lib>.data` for the data segment.

Another important thing to note is that in traditional core files only the first 4,096 bytes of the main executable and each shared libraries' text images are written to disk. This is done to save space, and, considering that the text segment presumably should not change, this is usually OK. However, in forensics analysis we must be open to the possibility of an RWX text segment that has been modified, e.g., with inline function hooking.

## 8.4 Heuristics

Also notice that there is one section showing a suspicious-looking shared library that is not marked as the type **SHLIB** but instead as **INJECTED**.

2	[26]	evil.lib.so.text	INJECTED	00007fb0358c3000	00215000
				00000000000000002000	A 0 0 8

“#define SHT\_INJECTED 0x200000” is custom and the **readelf** utility has been modified on my system to reflect this. A standard **readelf** will show it as <unknown>.

This section is for a shared library that was considered by *ecfs* to be maliciously injected into the process. The *ecfs* core handler does quite a bit of heuristics work on its own, and therefore leaves very little work for the forensic analyst. In other words, the analyst no longer needs to know jack about ELF in order to detect complex memory infections (more on this with the PLT/GOT hook detection later!)

Note that these heuristics are enabled by passing the **-h** switch to **/opt/bin/ecfs**. Currently, there are occasional false-positives, and for people designing their own heuristics it might be useful to turn the *ecfs*-heuristics off.

## 8.5 Custom section headers

Moving on, there are a number of other custom sections that bring to light a lot of information about the process.

2	[27]	.prstatus	PROGBITS	0000000000000000	0023f000
	[28]	.fdinfo	PROGBITS	0000000000000000	0023f150
4		000000000000c78	PROGBITS	0000000000000000	0023f000
6	[29]	.siginfo	PROGBITS	0000000000000000	0023fdc8
8		0000000000000080	PROGBITS	0000000000000000	0023f000
10	[30]	.auxvector	PROGBITS	0000000000000000	0023fe48
12		00000000000000130	PROGBITS	0000000000000000	0023f000
14	[31]	.exepath	PROGBITS	0000000000000000	0023ff78
	[32]	0000000000000024	PROGBITS	0000000000000000	0023ff00
	[33]	.personality	PROGBITS	0000000000000000	0023ff9c
		0000000000000004	PROGBITS	0000000000000000	0023ff00
		[33]	.arglist	PROGBITS	0000000000000000
				0000000000000001	0023ffa0

I will not go into complete detail for all of these, but will later show you a simple parser I wrote using the **libecfs** API that is designed specifically to parse *ecfs-core* files. You can probably guess as to what most of these contain, as they are somewhat straightforward; i.e., **.auxvector** contains the process' auxiliary vector, and **.fdinfo** contains data about the file descriptors, sockets, and pipes within the process, including TCP and UDP network information. Finally, **.prstatus** contains **elf\_prstatus** and similar structs.

## 8.6 Symbol table resolution

One of the most powerful features of *ecfs* is the ability to reconstruct full symbol tables for all functions.

2	\$	readelf -s host.10710
		Symbol table '.dynsym' contains 7 entries:

```

4 Num:      Value          Size Type Bind Vis Ndx Name
5   0: 0000000000000000    0 NOTYPE LOCAL DEFAULT UND
6   1: 000000300106f2c0    0 FUNC  GLOBAL DEFAULT UND fputs
7   2: 0000003001021dd0    0 FUNC  GLOBAL DEFAULT UND __libc_start_main
8   3: 000000300106edb0    0 FUNC  GLOBAL DEFAULT UND fgets
9   4: 00007fb0358c3000   0 NOTYPE WEAK  DEFAULT UND __gmon_start__
10  5: 000000300106f070    0 FUNC  GLOBAL DEFAULT UND fopen
11  6: 00000030010c1890    0 FUNC  GLOBAL DEFAULT UND sleep
12
Symbol table '.syntab' contains 5 entries:
13 Num:      Value          Size Type Bind Vis Ndx Name
14  0: 0000000004004b0    112 FUNC GLOBAL DEFAULT 10 sub_4004b0
15  1: 000000000400520     42 FUNC GLOBAL DEFAULT 10 sub_400520
16  2: 00000000040060d   160 FUNC GLOBAL DEFAULT 10 sub_40060d
17  3: 0000000004006b0   101 FUNC GLOBAL DEFAULT 10 sub_4006b0
18  4: 000000000400720      2 FUNC  GLOBAL DEFAULT 10 sub_400720

```

Notice that the dynamic symbols (.dynsym) have values that actually reflect the location of where those symbols should be at runtime. If you look at the .dynsym of the original executable, you would see those values all zeroed out. With the .syntab symbol table, all of the original function locations and sizes have been reconstructed by performing analysis of the exception handling frame descriptors found in the PT\_GNU\_EH\_FRAME segment of the program in memory.<sup>37</sup>

## 8.7 Relocation entries and PLT/GOT hooks

Another very useful feature is the fact that *ecfs-core* files have complete relocation entries, which show the actual runtime relocation values—or rather what you should *expect* this value to be. This is extremely handy for detecting modification of the global offset table found in .got.plt section.

```

1 $ readelf -r host.10710
2
3 Relocation section '.rela.dyn' at offset 0x23e0 contains 1 entries:
4   Offset           Info      Type            Sym. Value   Sym. Name + Addend
5 000000600ff8  000400000006 R_X86_64_GLOB_DAT 00007fb0358c3000 __gmon_start__ + 0
6
7 Relocation section '.rela.plt' at offset 0x23f8 contains 6 entries:
8   Offset           Info      Type            Sym. Value   Sym. Name + Addend
9 000000601018  000100000007 R_X86_64_JUMP_SLO 000000300106f2c0 fputs + 0
10 000000601020 000200000007 R_X86_64_JUMP_SLO 0000003001021dd0 __libc_start_main + 0
11 000000601028 000300000007 R_X86_64_JUMP_SLO 000000300106edb0 fgets + 0
12 000000601030 000400000007 R_X86_64_JUMP_SLO 00007fb0358c3000 __gmon_start__ + 0
13 000000601038 000500000007 R_X86_64_JUMP_SLO 000000300106f070 fopen + 0
14 000000601040 000600000007 R_X86_64_JUMP_SLO 00000030010c1890 sleep + 0

```

Notice that the symbol values for the .rela.plt relocation entries actually show what the GOT should be pointing to. For instance:

```
000000601028 000300000007 R_X86_64_JUMP_SLO 000000300106edb0 fgets + 0
```

This means that 0x601028 should be pointing at 0x300106edb0, unless of course it hasn't been resolved yet, in which case it should point to the appropriate PLT entry. In other words, if 0x601028 has a value that is not 0x300106edb0 and is not the corresponding PLT entry, then you have discovered malicious PLT/GOT hooks in the process. The libecfs API comes with a function that makes this heuristic extremely trivial to perform.

---

<sup>37</sup>I cover this nifty technique in more detail at [http://www.bitlackeys.org/#eh\\_frame](http://www.bitlackeys.org/#eh_frame).

## 8.8 Libecfs Parsing and Detecting DLL Injection

Still sticking with our `host.10710_ecfs-core` file, let us take a look at the output of `readecfs`, a parsing program I wrote. It's a very small C program; its power comes from using `libecfs`.

```
1 $ ./readecfs ../infected/host.10710
2 - read_ecfs output for file ../infected/host.10710
3 - Executable path (.exepath): /home/ryan/git/ecfs/ecfs_tests/host
4 - Thread count (.prstatus): 1
5 - Thread info (.prstatus)
6     [thread 1] pid: 10710
7
8 - Exited on signal (.siginfo): 11
9 - files/pipes/sockets (.fdinfo):
10    [fd: 0] path: /dev/pts/8
11    [fd: 1] path: /dev/pts/8
12    [fd: 2] path: /dev/pts/8
13    [fd: 3] path: /etc/passwd
14    [fd: 4] path: /tmp/passwd_info
15    [fd: 5] path: /tmp/evil_lib.so
16
17 assigning
18 - Printing shared library mappings:
19 ld-2.19.so.text
20 ld-2.19.so.relro
21 ld-2.19.so.data.0
22 libc-2.19.so.text
23 libc-2.19.so.undef
24 libc-2.19.so.relro
25 libc-2.19.so.data.1
26 evil_lib.so.text // HMM INTERESTING
27
28 .dynsym: - 0
29 .dynsym: fputs - 300106f2c0
30 .dynsym: __libc_start_main - 3001021dd0
31 .dynsym: fgets - 300106edb0 // OF IMPORTANCE
32 .dynsym: __gmon_start__ - 7fb0358c3000
33 .dynsym: fopen - 300106f070
34 .dynsym: sleep - 30010c1890
35
36 .symtab: sub_4004b0 - 4004b0
37 .symtab: sub_400520 - 400520
38 .symtab: sub_40060d - 40060d
39 .symtab: sub_4006b0 - 4006b0
40 .symtab: sub_400720 - 400720
41
42 - Printing out GOT/PLT characteristics (pltgot_info_t):
43 gotsite: 601018 gotvalue: 300106f2c0 gotshlib: 300106f2c0 pltval: 4004c6
44 gotsite: 601020 gotvalue: 3001021dd0 gotshlib: 3001021dd0 pltval: 4004d6
45 gotsite: 601028 gotvalue: 7fb0358c3767 gotshlib: 300106edb0 pltval: 4004e6 // WHAT IS WRONG HERE?
46 gotsite: 601030 gotvalue: 4004f6 gotshlib: 7fb0358c3000 pltval: 4004f6
47 gotsite: 601038 gotvalue: 300106f070 gotshlib: 300106f070 pltval: 400506
48 gotsite: 601040 gotvalue: 30010c1890 gotshlib: 30010c1890 pltval: 400516
49
50 - Printing auxiliary vector (.auxilliary):
51 AT_PAGESZ: 1000
52 AT_PHDR: 400040
53 AT_PHENT: 38
54 AT_PHNUM: 9
55 AT_BASE: 0
56 AT_FLAGS: 0
57 AT_ENTRY: 400520
58 AT_UID: 0
59 AT_EUID: 0
60 AT_GID: 0
61
62 - Displaying ELF header:
63 e_entry: 0x400520
64 e_phnum: 20
65 e_shnum: 40
66 e_shoff: 0x23fff0
67 e_phoff: 0x40
68 e_shstrndx: 39
69
70 --- truncated rest of output ---
```

Just from this output alone, you can see so much about the program that was running, including that at some point a file named `/tmp/evil_lib.so` was opened, and—as we saw from the section header output earlier—it was also mapped into the process.

```
2 [26] evil_lib.so.text INJECTED 00007fb0358c3000 00215000
     0000000000000000 0000000000000000 A 0 0 8
```

Not just mapped in, but injected—as shown by the section header type `SHT_INJECTED`. Another red flag can be seen by examining the line from my parser that I commented on with the note “WHAT IS WRONG HERE?”

```
gotsite: 601028 gotvalue: 7fb0358c3767 gotshlib: 300106edb0 pltval: 4004e6
```

The `gotvalue` is `0x7fb0358c3767`, yet it should be pointing to `0x300106edb0` or `0x4004e6`. Notice anything about the address that it’s pointing to? This address `0x7fb0358c3767` is within the range of `evil_lib.so`. As mentioned before it *should* be pointing at `0x300106edb0`, which corresponds to what exactly? Well, let’s take a look.

```
1 $ readelf -r host.10710 | grep 300106edb0
000000601028 000300000007 R_X86_64_JUMP_SLO 000000300106edb0 fgets + 0
```

So we now know that `fgets()` is being hijacked through a PLT/GOT hook! This type of infection has been historically somewhat difficult to detect, so thank goodness that ECFS performed all of the hard work for us.

To further demonstrate the power and ease-of-use that ECFS offers, let us write a very simple memory virus/backdoor forensics scanner that can detect shared library (DLL) injection and PLT/GOT hooking. Writing something like this without `libecfs` would typically take a few thousand lines of C code.

```
-- detect_dll_infection.c --
2 #include "../libecfs.h"
4 int main(int argc, char **argv)
6 {
8     ecfs_elf_t *desc;
10    ecfs_sym_t *dsyms, *lsyms;
12    char *progrname;
14    int i;
16    char *libname;
18    ecfs_sym_t *dsyms;
19    unsigned long evil_addr;
21
22    if (argc < 2) {
23        printf("Usage: %s <ecfs_file>\n", argv[0]);
24        exit(0);
25    }
26
27    desc = load_ecfs_file(argv[1]);
28    progrname = get_exe_path(desc);
29
30    for (i = 0; i < desc->ehdr->e_shnum; i++) {
31        if (desc->shdr[i].sh_type == SHT_INJECTED) {
32            libname = strdup(&desc->shstrtab[desc->shdr[i].sh_name]);
33            printf("[!] Found maliciously injected shared library: %s\n", libname);
34        }
35    }
36    pltgot_info_t *pltgot;
37    int ret = get_pltgot_info(desc, &pltgot);
```

```

32     for ( i = 0; i < ret; i++) {
33         if ( pltgot[i].got_entry_va != pltgot[i].shl_entry_va && pltgot[i].got_entry_va !=
34             pltgot[i].plt_entry_va)
35             printf("![!] Found PLT/GOT hook, function 'name' is pointing at %lx instead
36             of %lx\n",
37             pltgot[i].got_entry_va, evil_addr = pltgot[i].shl_entry_va);
38     ret = get_dynamic_symbols(desc, &dsyms);
39     for (i = 0; i < ret; i++) {
40         if (dsyms[i].symval == evil_addr) {
41             printf("![!] %lx corresponds to hijacked function: %s\n", dsyms[i].symval,
42             &dsyms[i].strtab[dsyms[i].nameoffset]);
43             break;
44     }
45 }

```

This program analyzes an *ecfs-core* file and detects both shared library injection and PLT/GOT hooking used for function hijacking. Let's now run it on our *ecfs* file.

```

1 $ ./detect_dll_infection host.10710
2 [!] Found maliciously injected shared library: evil_lib.so.text
3 [!] Found PLT/GOT hook, function 'name' is pointing at 7fb0358c3767 instead of 300106edb0
4 [!] 300106edb0 corresponds to hijacked function: fgets

```

With just simple forty lines of C code, we have an advanced detection tool capable of detecting an advanced memory infection technique, commonly used by attackers to backdoor a system with a rootkit or virus.

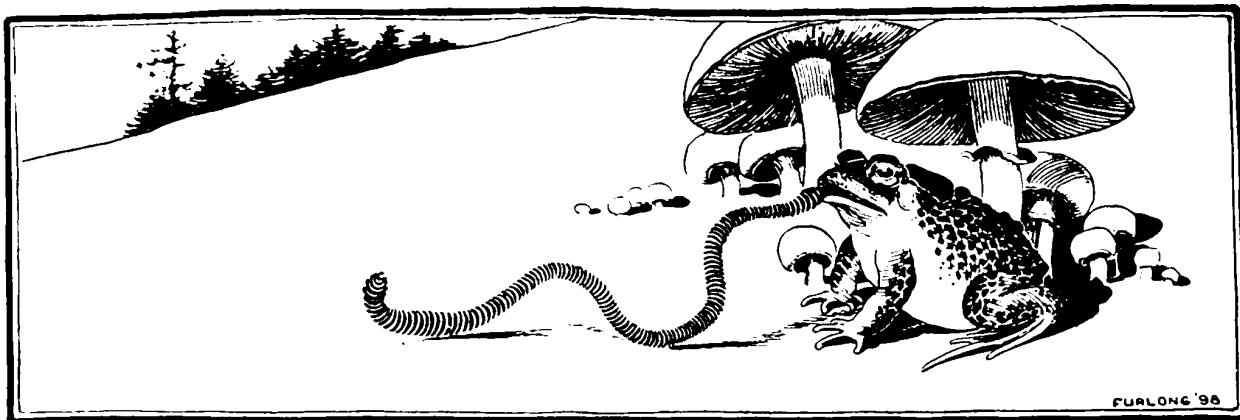
## 8.9 In Closing

If you liked this paper and are interested in using or contributing to ECFS, feel free to contact me. It will be made available to the public in the near future.<sup>38</sup>

Shouts to Orangetoaster, Baron, Mothra, Dk, Sirus, and Per for ideas, support and feedback regarding this project.

---

<sup>38</sup><http://github.com/elfmaster/ecfs>



## THE ALGORITHM SEES THE INTERNET THE WAY DMITRY SKLYAROV SEES A POORLY ENCRYPTED DRM FILE.

Every time you cough, a hunk of code or a piece of some obscure url comes shooting out. You can't see it, but it's there. Probably there is some on your shoes. A little string of binary  $G-(v,e)$ : code, or maybe the "r" and "g" from a dot org, right there on your burgundy cap-toes. The reason is that you're drowning in a sea of information. Heed not the worrisome findings of the recent ODP coastline study—by the time glacial melt brings the ocean to your doorstep, your lungs will already be full of html.

**WE DON'T HAVE TO TELL YOU THE WORLD WIDE WEB IS AN ANARCHIC FORM OF POPULIST HYPERMEDIA.**

But we WILL tell you it's a hypertext corpus of unfathomable intricacy, and it's expanding faster than a flat universe in a cosmologically significant vacuum energy density. For the love of Gödel, just look at the thing! Millions of participants with as many agendas, cranking out hyperlinked content like there's no tomorrow. In fact, at this rate, the disappearance of tomorrow, or at least a universally accepted definition thereof, is actually a valid concern.

**SEARCH IS AN UNDERSTATEMENT.  
ODYSSEAN QUEST IS MORE LIKE IT.**

So how are you supposed to find anything in this great rolling miasma of ones and zeros? Text-based searches are not so good. If you believe otherwise, consider the word facial. A search engine that takes nothing more than the word itself into account will return  $A = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ . Look, even if you do manage to cluster a word into five different meanings, there's still the fact that each individual meaning yields nearly infinite search results. And a quindecillion divided by five is still two hundred quattuordecillion.

**ALL OF A SUDDEN, "WHO KNOWS?" IS AN ASTUTE QUESTION.**

Searching the Internet, it turns out, is not much different from searching the real world. The best thing to do is ask someone who knows. An authority on the subject. But who are the authorities, and what qualifies them as such in the first place? A web page can't just declare itself an authority. If authority could be generated endogenously, Louis de Branges would have verified his own proof of the Riemann Hypothesis. Neither should authority be conferred from one page to another. This means you'd be OK letting Herman Mudgett pick your primary care guy. Last in the triumvirate of really-bad-ways-to-determine-authority is the notion of popularity. Surprisingly, this is the method employed by today's most widely used search engines. They find sites with the most links and present them as authorities. This is roughly analogous to handing the Fields Medal to your high school homecoming queen.

**THE ANSWER CAME FROM BOOKS. WEIRD.**

So what's the solution to search? While computer science was trying to coax an answer from its collective hard drive, it was sitting right there in the stacks all along. Who could have guessed that when Eugene Garfield went all bibliometric and devised a system to find out how much a journal mattered by counting the number of times that journal was cited in other publications, he consciously invented the beginnings of a system that might work in search. Then Gabriel Pinski and Francis Narin took it a step further by suggesting some citations should carry more weight than others, and let's face it, being cited in the Spring '96 issue of *Social Text* (pages 217–252, to be precise) isn't exactly a literary feather in your cap. But taking into account the quality of citations is only half the answer in search.

Because compared to the neatly governed world of identifying and publishing, the Internet is completely insane. Fluid. Variable. Heterogeneous. Anarchic in anonymity. Replete with conflicting agendas. So counting inbound links isn't enough. Not even close. To search effectively in these circumstances, you have to don some serious math goggles and take a look at the big picture.

**THE ALGORITHM SEES GALAXIES, BUT IT'S BLIND AS A BAT.**

The heavy hitters of search all use the same mathematically myopic approach—counting links back to authoritative Web pages. But the only way to tell what's really going on is to take a step back and

look for patterns in the sites that point back to authorities. And when you do, you quickly see that there is another layer to the puzzle—sites that point to more than one authority, or hub pages, if you will. These hubs and their surrounding authorities form little galaxies of relevant information, something that makes the hair stand up on the back of any self-respecting searchophile's neck. It's the difference between checking out the Big Dipper from a lawn chair in your back yard and peering into Fornax with Hubble's Ultra Deep Field. But an algorithm that could detect these galaxies would be virtually impossible to pull off, since it would have to assess both inbound and outbound information, and continually calculate the relationship between the two, in real time.

**THE ALGORITHM IS RELATIVELY SIMPLE,  
IF YOU'RE SOME KIND OF SAVANT.**

It works like this. For each search query, an index  $G$  of Web pages is found. For each page  $p$ , you associate a non-negative authority weight  $a(p) \rightarrow \bar{a} = A\bar{a}$ , and a non-negative hub weight  $h(p) \rightarrow \bar{h} = A\bar{h}$ . This  $\bar{a} = A\bar{a}$  will lead you to the rather obvious conclusion that when  $p$  points to lots of pages with big  $a$  values, it should get a big  $h$  value (inverse weighted popularity). And when  $p$  is pointed to by lots of pages with big  $h$  values, it should get a big  $a$  value (weighted popularity). From here, you simply fire up an iterative singular value decomposition operation and wrap things up by banging out an orthonormal basis of eigenspace for each and obtaining the eigenvectors for the matrices in question. That's it.

**IT'S A GOOD THING ROBERT FROST NEVER  
WROTE AN ALGORITHM.**

Taking the road less traveled is fine if you're stumbling around the New England countryside, being whimsical or whatever. But when you're searching online, that kind of thing gets you eaten by wolves. Because dismissing where others have gone can quickly get you lost in a forest of irrelevant results. But while you are learning from the Algorithm, the Algorithm is learning too. It studies the way anonymous groups of users search and forms an aggregate view of which results those users find the most valuable. This sends relevance through the roof and gets you to your desired destination without the slightest hint of lupine intercession. Sure, "The Road Traveled Every Five Minutes" would make a lousy poem, but it makes a gorgeous piece of code.

**THE ALGORITHM APPROACHES ARTIFICIAL  
INTELLIGENCE, BUT IT HAS NOTHING AGAINST  
PEOPLE NAMED SARAH CONNOR.**

Yes, the Algorithm is an omniscient, evolving organism devoid of all feeling, but in no way should this freak you out. In fact, it's cause for celebration. Because the Algorithm comes in peace.  $\bar{a} = A\bar{a}$  It's here to revolutionize search by identifying a topic, finding experts on that topic and assessing the popularity of pages among those experts, simultaneously, in the blink of an eye, whenever you want. It's here to narrow or expand your search based on concept—something no other search engine can do. Never again will you wade into the perpetually updated, subject-centric world of blogs without technology that actually comprehends subjects. The Algorithm knows that Usher Syndrome is transmitted by an autosomal recessive gene, not a subwoofer. And never again will you get "results" consisting merely of ten blue links, rather than the rich aggregate of images, video, conceptually related search topics and pure expert insight the Algorithm delivers.

**THE ALGORITHM UNDERSTANDS THAT  
COLLECTIVE WISDOM IS NOT NECESSARILY  
COLLECTED FROM EVERYONE.**

Based solely on the number of participants, the Web is undoubtedly the world's largest source of pure wisdom. But this doesn't mean there is wisdom inherent in every participant or every page. The Algorithm is acutely aware of this. It realizes that somewhere between James Surowiecki's *The Wisdom of Crowds* and Charles Mackay's *Madness of Crowds* lies the sweet spot. It sees everything but knows just what to look for. It scour the convoluted expanses of cyberspace and brings back an instantaneous convergence of wisdom collected, waiting for the day you're ready.



THE ALGORITHM

©2007 Ask.com

## 9 Bambaata speaks from the past.

by Count Bambaata, Senior NASCAR Correspondent

"Myths and legends die hard in America. We love them for the extra dimension they provide, the illusion of near-infinite possibility to erase the narrow confines of most men's reality. Weird heroes and mould-breaking champions exist as living proof to those who need it that the tyranny of 'the rat race' is not yet final."

*Gonzo Papers, Vol. 1: The Great Shark Hunt: Strange Tales from a Strange Time*, Hunter S. Thompson, 1979.

It's been an interesting ride for someone who has witnessed nearly all of the perspectives and colliding philosophies of the computer security practice. Having met professionals and enthusiasts of other fields of knowledge built upon the foundations of scientific work, I could say few other industries are as swarmed with swine and snake oil salesmen as computer security. I guess the medium lends itself to such delusions of self-worth and importance. Behind a screen, where you can't see the white of the eyes of the people you interact with, anything is possible.

It doesn't help it that, deprived of other values as important as human contact, true friendship and uninterested genuine camaraderie, fame and financial success dictate the worth of the individual. Far from being the essence of the so-called American dream, where the individual succeeds thanks to persistence and true innovation, in computer security, and more specifically, in the area of security I will be addressing in this letter, success comes from becoming a virtual merchant of vacuum and nothingness, charging a commission for doing absolutely nothing, bringing absolutely no innovation, unfortunately at tax payers expense, as we will see later. An economy built upon the mistakes of others, staying afloat only so as long as such mistakes are never addressed and true solutions remain undeveloped and underutilized.

Going back to the early 2000s, there were two major perspectives on publication and distribution of security vulnerabilities. On one side, those against it (not for economical reasons but a philosophy taking from the times when "hacking" actually meant to hack, not for publicity or profit, but curiosity and technical prowess). These "black hats" perhaps represented the last remnants of a waning trend of testing the widely extended practice of capitalizing security vulnerabilities in a perpetual state of fear and confusion taking advantage of the (then mostly) ignorant user base of networked computers. Opposing them, a large mob in the industry proclaimed the benefits and legitimacy of "full" and "responsible"

disclosure. These individuals claimed the right moral choice was to make information about exploitation of vulnerabilities (and the flaws themselves) publicly available.

They were eager to call out "black hats" with disdain, as dangerous amoral people whose intentions ranged from everything between stealing banking credentials, spreading viruses or, well, fucking children if they ran out of expletives and serious sounding accusations for the press. No accusation was too far-fetched. Underneath, an entire network of consulting firms thrived on the culture of fear carefully built with hype. Techniques and vulnerabilities known to the anti-disclosure community for years surfaced, leading to events such as the swift sweep of format string vulnerabilities that led to a bug class nearly phasing out of existence within less than two years. Back then, some of the members of the industry were able to market IDS products to customers keeping a straight face. And the swine only got better at that game.

As much as groups such as Anonymous and others have prostituted whatever was left of that original "antisecurity" community and its philosophy, whose purpose had nothing to do with achieving fame out of proclaiming themselves as some sort of armchair bourgeoisie revolutionaries, today the landscape is, if you pardon the expression, hilarious. Fast forward to a post-9/11 America, with the equities problem (COMSEC versus SIGINT) leaning to the side of SIGINT. The consulting houses from the old days and a swarm of new small shops appeared in the radar to supply a niche necessity created as an attempt to address the systematic compromise and ravaging of defense industry corporations and federal government networks.

Welcome to the vulnerability market. Flock after flock of vultures fly in circles in a market where obscurity, secrecy and true loyalty are no longer desirable traits, but handicaps. If you are discreet, and remain silent and isolated from the other "players", the buyers will play you out. In a strange mix of publicity

hogs and uncleared greed-crazed freaks, middlemen thrive as the intelligence community desperately tries to address the fact that we are lagging a decade behind the people ravaging our systems, gooks and otherwise. Middlemen provide a much needed layer of separation, while hundreds of thousands of dollars, amounting up to millions, are spent without congressional supervision. Anything goes with the market. Individuals who would never be accepted to participate in any kind of national security-impacting activities live lavish lifestyles, dope addled and confident that their business goes undisturbed. Quite simply, these opportunist swindlers are hustling the buck while the status quo remains unaffected. Just to name one example, Cisco has had its intellectual property stolen several times. Of those compromises, none involving “black hats” resulted in its technology magically appearing at Huawei headquarters. Picture a pubescent 25 year old Chinese virgin incessantly removing “PROPRIETARY” copyright banners from Cisco IOS source, as he laughs hysterically slurping up noodles from a Ramen shake n’ bake cup. The tale of Abdul Qadeer Khan, or a certain Crown Corporation, are lullabies compared to the untold stories that, quite probably, some day will be declassified for our grandsons to read, provided that full-blown Idiocracy hasn’t ensued, and (excuse the language), nobody gives a flying fuck anymore.

Let’s gaze back at the past, something is wrong here. Where did the responsible disclosure geeks go? It was a majestic party. Everyone was having a ball. Suddenly, everyone left and nobody bothered to clean the mess. Perhaps they found a new spiritual path, retiring to a tranquil life enjoying the fruits of the late 1990s and early to mid 2000s, carefree and happy to leave the snake oil salesman life behind. Did they take vows of poverty, donated all they had to the Salvation Army, or the Dalai Lama, and left for Bhutan? Not quite. Please, let me, your humble host, guide you to Crook Planet. It’s a strange place. I used to like it in here. Where I come from, they say when you earn someone’s trust and friendship, it’s a lifelong deal. You break it, and you wish you had never been friends with the poor bastard. In a way, it is better to be wronged by someone you don’t know than being played by someone you considered “a friend.” The word has reasonably dropped value these days. It’s short of meaning “someone I hang out with, can get reasonably drunk with, but that’s about it.” A long time ago, a friend and mentor told me a real friend is the calm guy bothering himself to go visit

you in jail. Everyone else bails out. But that fellow goes there. Like a grandmother, without the weeping. You shake hands. Share a few old stories. Implicitly, you know he’s your only chance. But we’re drifting slightly from our route. Crook Planet, it was. Yes. If you were wondering where all those ethical evangelists of the responsible disclosure creed went, well, wonder no more. They’ve gone silent, because that’s where the dough is at. Keeping silent. Not among them, despite the NDAs in place, because they know that remaining silent, makes them vulnerable when facing buyers. There is irony about the turns of history. Here we are, trading mechanisms and tools to subvert technology, when years ago we considered their publication perfectly valid. And there is a need for offensive capabilities. Are American corporations and its federal government under attack? Yes, they are. Does the market, as it is lined out right now, help the tradecraft and improve the status quo? No, it doesn’t. But millions are plunging into the pockets of people whose interest, was, is and will always be that we, including the government, remain insecure. People have developed defensive technology that can render certain paths of abuse completely unreliable. The reaction of the greed-crazed freaks in the market, which I and others in similar positions have on record, ranged from negative to cocky (“It will drive up the prices, good for us”). Well, you greedy swine, this was never about the money. At least, it wasn’t for me. The kind of offensive capabilities I and my company developed could have netted us immense return on investment if used illegally. And so would yours.

The crude truth is that, by current market prices, they don’t even come close to the risk-reward equation our adversaries have. Whether it is sixty thousand or a quarter million for an exploit yielding high privilege access to a modern operating system, the price is still dramatically ridiculous if compared to the value of the intelligence and trade secrets that can be stolen from domestic corporations and the government itself. The market fails to address any of the problems we face today, while it creates a very real threat. Are we protecting ourselves against the exploits being traded among different agencies and defense contractors? Not a chance. We could see offensive security as the realm of smart men, whose greed exceeded their talents, and made them shit in their own nests. Those teenagers who were shrugged off by the industry in the early 2000s (despite the fact that they managed to publish personal informa-

tion of industry professionals and routinely compromised their systems, assumed to be, at the very least, slightly more secure than those of the laymen) compromised Fortune 50 corporations and obtained trade secrets ranging from proprietary operating system source code to design documents. For free, at zero cost. The first hackers unlocking the Apple iPhone had proprietary schematics of Samsung devices. Today, you can acquire the schematics of any phone in the markets of Shenzhen, China. The most public cases of “whistle blowers” have been individuals with top level clearances. As wave after wave of swine beat on their chests and chant patriotic lures, they salivate for a piece of the defense budget, hoping policy never changes. The problem, clearly, isn’t the need for offensive capabilities. They are necessary. The Cold War never quite went cold. What we don’t need, though, is swine playing the prom queens for us. Because it is only a matter of time until this entire clusterfuck of a party backfires on us, and it’s going to be an interesting crash landing when they start dodging the liabilities. These people do not care about the status quo. They are milking the cow, for as long as it lasts, just like it happened when disclosing information had any sizable “return on investment.” Once the hush money goes away, they might as well go back to the old tale of responsible disclosure. Crook Planet is also Turncoat Planet.

Everyone is willing to remain silent, for a fee. Developing security mitigations to protect both the defense industry and the layman is frowned upon. Talking about the market is frowned upon. Disclosing that former “ethical security researchers” are in it and silent for the big bucks is frowned upon. Acknowledging that the adversary is ahead of us because we are greedy swine hustling for tax payers’ money is frowned upon. It’s all bad for “business.” This hyped up “cyber war” of sorts, unless we do something about it, and do it now, is going to be about as successful as the “War on Drugs” and the “War on Terror.” Billions going into the deep pockets of people whose creed is green, and made out of dollar bills, but are too dumb to figure out, that in the scheme of things, they are their (and our) own worst enemies.

So much for sworn commitment to defend the Constitution and laws of the United States against all enemies, foreign and... Domestic? For a fee. Thank-

fully, the federal government and its institutions aren’t exclusively packed with swine and salesmen. There are, too, good people, no different than you or me, whose goal is to help their fellow men. Baudrillard called America “the last primitive society on Earth.” A society capable of swift change, of both great and depraved actions. Like good ole’ Hunter said, “In a nation run by swine, all pigs are upward-mobile and the rest of us are fucked until we can put our acts together: Not necessarily to Win, but mainly to keep from Losing Completely.” We better get this act together, soon.

I have managed to arrive at this point still remaining a gentleman. No names were called out. But if something happened, if I had the wrong hunch, professionally or personally, if I was disturbed in any way, or those whom are dear to me, let it be clear enough, that I’m not driven by wealth nor power, and even though I’ve never supported organizations like WikiLeaks,<sup>39</sup> I’m this fucking close to picking up a phone and start slipping letters into mail boxes.

All these years, when companies such as Microsoft created databases filled with files on the scene (thanks to their “Outreach” program, a theme park version of a COINTELPRO), and contractors and firms did the same, my own files grew in size, not with gossip, but a very different kind of dirt. “To live outside the law you must be honest,” as the Dylan song goes.

The question is: are we feeling lucky? Well... Are we?

Sincerely yours,

*Count Bambaatae*  
P.S. DONATIONS ACCEPTED:  
P.S. NO HONOR AMONG S.F.  
“BLACKHATS” ONLY  
FUCKING GREED. !!  
{- BLOOD DIAMONDS  
- KUWAITI GOLD  
- ILLEGAL CONTRABAND  
- OFFSHORE BANK INTROS.}

Count Bambaata, Head of the  
Department of Swine Slaughtering and  
Angry Letters Filled With Expletives

<sup>39</sup>With their eerie fixation on demonizing America, as much as we owe domestic swine for letting them have any dirt in first place, let’s not confuse things here and dodge the blame.

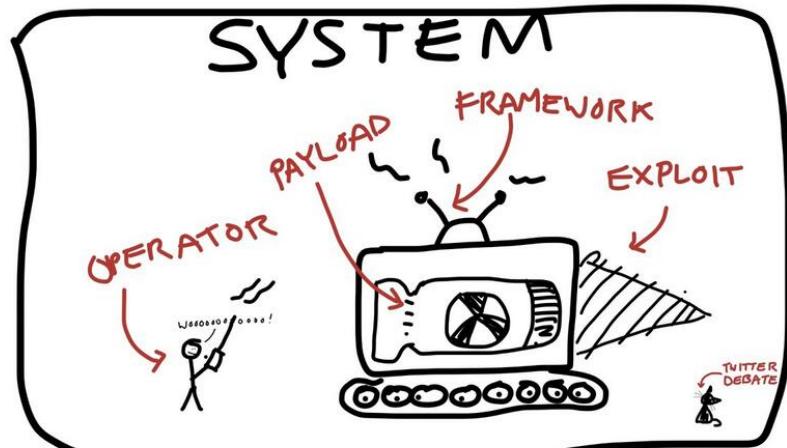
## 10 Public Service Announcement

We dedicate this page to public service, offering a handy cheat sheet for all the would-be regulators of 0-day sales and cyberbullets, so that they don't keep embarrassing themselves in public by misusing the words of our profession. If you know such an aspiring regulator, please feel free to cut this page out on the dotted line and mail it to them!

### Zero-day Cyberbullet Regulation Cheat Sheet & Fashion Advice

If Cyber is your style, *Zero-day Regulation* is “in” this legislative season. Annoyingly, cyberbullet merchants-of-death use too much technical jargon to hawk their deadly Turing-complete wares, and it’s all too easy to mix them all up. Now that would be embarrassing, wouldn’t it?

But despair not! With this handy cheat sheet you will soon be legislating cyberbullet export restrictions on evil cyberhackers like a cyberpro!



And remember: whatever your proposal, neither IMSI Catchers nor Rogue Wi-Fi Access Points are “exploits.” Exploits are what you used to jailbreak your iPhone to load the apps that you want but Apple doesn’t; never confuse the two!

## 11 Cyber Criminal's Song

*Arranged for an Anonymized Voice and the HN chorus  
by Ben Nagy  
(with abject apologies to G&S)*

I am the very model of modern Cybercriminal  
I've knowledge hypothetical that's technical and chemical  
And conduct most becoming, both grammatical and ethical!

I build my site with PHP so coders are replaceable  
I keep it all behind, like, seven proxies and a firewall  
And Tor is such secure so wow - my webs are much unbreakable!  
I'm careful with my secret life, I haven't told a single soul  
(Except three guys on Xbox Live and Chad whose .torrc I stole)

[CHORUS]  
SERIOUSLY, THANKS CHAD, THAT CONFIG IS TOTALLY SWEET

My cash is stored in bitcoin, the transactions are untraceable  
I read on Hacker News that the cryptography's exceptional  
And so, on matters technical, theoretical, and chemical  
I am the very model of modern Cybercriminal!

I'm totes well versed in Haskell and I love the lambda calculus  
I know Actionscript and Coffeescript and XML and CSS  
And OCaml and Rust and D and Clojure plus some Common LISP  
My daring Cyberlife is like The Matrix with a modern twist!  
(But to stay close the metal I prefer to roll with node.js)

[CHORUS]  
TO STAY CLOSE TO THE METAL WE PREFER TO ROLL ON NODE JSSSSSS

For matters pharmaceutical I'm well researched on Erowid  
From Aderall to Zolpidem and Dexedrine to Dicodid  
From re-uptake inhibitors to analgesic opioids  
I know the pharmacology of all the drugs the world enjoys  
Good Sir, in fields theoretical, chemical, and technical  
I am the very model of modern Cybercriminal!

I downloaded all five seasons of The Wire from The Pirate Bay  
And studied all their OPSEC and legalities of what to say  
If interviewed by cops and, well, I must admit it's child's play  
How do these people make mistakes? Such staggering naïveté!

[CHORUS]  
WE'D NEVER MAKE SUCH NOOB MISTAKES WE LAUGH AT YOUR NAÏVETÉ

My records are impeccable, I keep them all in triplicate  
I know what day I paid for my new Tesla or my contract hits  
I run GNUCash on Linux my finances are so intricate  
And all backed up to Google Docs which makes me a Cloud Syndicate.

[CHORUS]  
WE'RE REALLY VERY SORRY BUT WELL ACTUALLY IT'S GNU/LINUX

Then, I can quote Sun Tzu or Nietzsche highlights from the Internet  
My strategies are therefore quite profound much like my intellect  
Yes, for all things theoretical, technical and chemical  
I am the very model of a modern Cybercriminal!

In fact, when I know what is meant by "cover" and "concealment"  
When I can keep my Facebook, Yelp and Tinder in a compartment  
Or when I know the difference 'tween a public and a private key  
Stop logging in to check my recent sales from the library  
When I can keep my mouth shut in a bar just momentarily  
In short, when I have frankly any skills that go beyond my screen  
You'll say no better Cybercriminal the world has ever seen!

Though criminally weak, you'll find I'm plucky and adventurey  
And though my reading starts at the beginning of the century  
On matters theoretical, technical and chemical  
I am totally the model of a modern Cybercriminal!

[CHORUS]  
THE VERY VERY MODEL OF THE MODERN CYBER CRIMINAL!

ASSEMBLE  
*King Midget*

Highway runabout. Low cost. 45 miles per hour. 90 miles per gallon. Just bolt together our factory built units. See how easy it is. Send \$1 (refunded first order) for 24 page assembly book with blueprints, drawings, photos. PLUS detailed illustrated circular.

Circular only—25c.

SAFE •  
PRACTICAL •  
ECONOMICAL •

MIDGET MOTORS      Athens, Ohio

## 12 Fast Cash for Bugs!

*by Pastor Manul Laphroaig, Proselytizer of Weird Machines*

Howdy, neighbor! Is that a fresh new PoC you are hugging so close? Don't stifle it, neighbor, it's time for it to see the world, and what better place to do it than from the pages of the famed International Journal of PoC or GTFO? It will be in a merry company of other PoCs big and small, bit-level and byte-level, raw binary or otherwise, C, Python, Assembly, hexdump or any other language. But wait, there's more—our editors will groom it for you, and dress it in the best Sunday clothes of proper church English. And when it looks proudly back at you from these pages, in the company of its new friends, won't that make you proud? So set that little PoC free, neighbor, and let it come to me, [pastor@phrack.org](mailto:pastor@phrack.org)!

---

Do this: write an email telling our editors how to do reproduce \*ONE\* clever, technical trick from your research. If you are uncertain of your English, we'll happily translate from French, Russian, or German. If you don't speak those languages, we'll dig up a translator.

Like an email, keep it short. Like an email, you should assume that we already know more than a bit about hacking, and that we'll be insulted or—WORSE!—that we'll be bored if you include a long tutorial where a quick reminder would do.

Don't try to make it thorough or broad. Don't use Powerpoint bullet-points or OpenOffice Unicode; we'll typeset it for you.

Do pick one quick, clever low-level trick and explain it in a few pages. Teach me how to make music that also parses as PSK31, RTTY, or WeFax. Show me how to reverse engineer SoftStrip barcodes. Don't tell me that it's possible; rather, teach me how to do it myself with the absolute minimum of formality and bullshit.

Like an email, we expect informal (or faux-biblical) language and hand-sketched diagrams. Write it in a single sitting, and leave any editing for your poor preacherman to do over a bottle of fine scotch. Send this to [pastor@phrack.org](mailto:pastor@phrack.org) and hope that the neighborly Phrack folks—praise be to them!—aren't man-in-the-middling our submission process.

