(Malware) Analysis Using Visualization

Hack in the Box, Kuala Lumpur 2013

Wes Brown

wbrown@ephemeralsecurity.com

Ephemeral Security

self.about

- self.name = Wes Brown
- self.company = { current: Ephemeral Security
  previous: ThreatGRID, Inc }
- self.coolstuff = [Mosquito Remote Injectable VM,
  Malnet Malware Analysis LiveCD,
  Supercomputing Analysis of Malware]
- self.proclivities = [Weird Functional Languages]

visualization.about

- Visualization is the organization, rendering, and presentation of data in a visual.
- Meaningful visualization that has more purpose than to impress management is very hard.
- This workshop is intended to show how to use visualization to analyze malware as well as other security topics, and to provide tools to aid in this.

svforth.about

- svforth.name
  - Security Visualization FORTH
- svforth.description
  - FORTH dialect with threads, remote procedure calls, and access to platform language functions and libraries.
- svforth.platforms = [JavaScript, Python]
- svforth.sources = https://github.com/ephsec/svforth

forth.wtf?

- Language of implementation shapes thought patterns, and alter reasoning about a problem.
- Lisp and other functional languages that allow high order functions and lazy evaluations allow the passing of functions that customize the behavior of the function being called.
- Forth has a stack oriented nature and encourages a layered approach to programming using short functions.
- Visualization and analysis revolves around manipulation of linear data that are query results, lending itself to stacks.

svforth.javascript

- SVFORTH’s primary implementation langauge is JavaScript.
- JavaScript’s use of closures and ability to pass anonymous functions (lambdas) as arguments to functions makes implementing a Forth trivial.
- JavaScript runs in the browser, and has a rich set of libraries revolving around visuals.
- Author likes functional languages, and JavaScript really is a functional language in Algol (C, Java) like costume.

svforth.python

- SVFORTH also has an implementation in Python.
- While SVFORTH.JS works and is used with Node.js for server-side tasks, Python is more commonly available making it more useful for the workshop.
- Python also supports passing functions as arguments, and functions as objects.
- In some ways, Python implementation is cleaner due to JavaScript's stupidity with global values.

get(svforth)

- Requirements
  - Modern HTML5 browser
  - Python (Optional)
- Wireless Network
  - AP: ForthLand
  - Password: SVFORTH
- http://svforth.forthland

learn(forth)

- Stack Based (Reverse Polish Notation)
  - Push items to operate on onto stack
  - Forth words operate on stack, typically popping values off the end.
  - Whitespace delimited tokens
  - Every Forth word can be redefined to something else, including primitive stack operations if you are foolhardy enough!

forth learn

Input	Evaluate	Stack
10 20 30 * +		
20 30 * +	10	10
30 * +	20	10, 20
* +	30	10, 20, 30
+	*	10, 600
	+	610
10 / 50 -		
	10	610 10
	/	61
	50	61 50
	-	11

forth.stack

word	stack diagram	description
pop	(a b c) -- (a b)	pops a value off the stack for current fn
push	(a b c) -- (a b c d)	pushes a new value onto the stack
drop	(a b c) -- (a b)	drops a value off the stack without using
dup	(a b c) -- (a b c c)	duplicates value at top of the stack
swap	(a b c) -- (a c b)	swaps top value on stack with value before
nip	(a b c) -- (a c)	removes value before top of stack
rot	(a b c) -- (c a b)	rotates entire stack
-rot	(a b c) - (b c a)	counter-rotates entire stack
depth	(a b c) - (a b c 3)	pushes current depth of stack onto stack
.s	(a b c) -- (a b c)	prints stack

forth.canvas

word	stack diagram	description
:	: word definition ;	defines a Forth word terminated by ;

word	stack diagram	description
canvas	(html-canvas) - ( )	sets the current canvas operated upon
fillcolor	(r g b) - ( )	sets the current color used for drawing
rect	(x1 y1 x2 y2) - ( )	draws a rectangle on current canvas

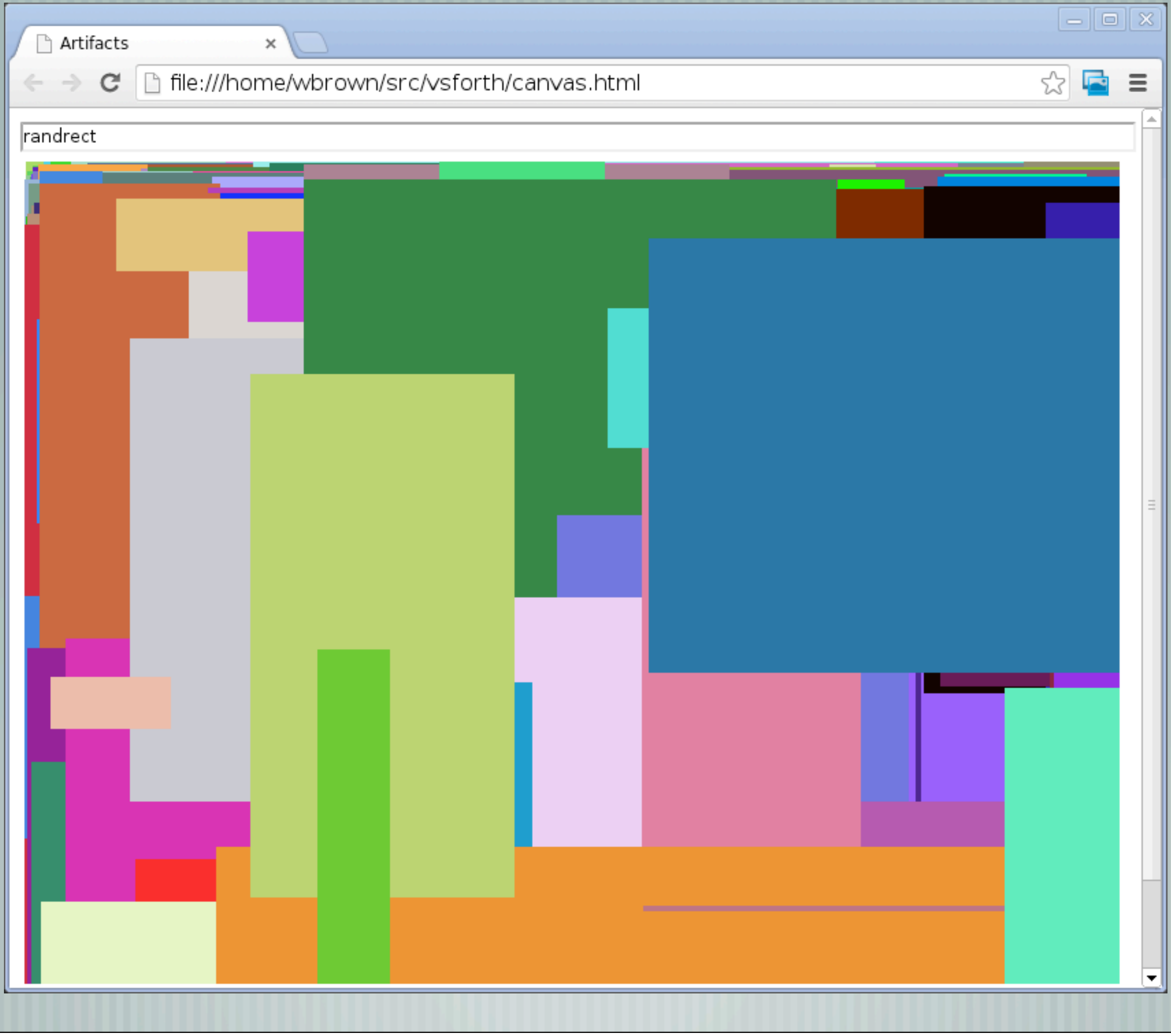
word	stack diagram	description
rand	(low high) - (rand)	generates random number

word	stack diagram	description
loop	loop a b c again	marks the beginning of a loop block
again	loop a b c again	repeats loop block ad infinitium

randrect.forth

word	code
pick-color	: pick-color 0 255 rand 0 255 rand 0 255 rand (red, green, blue) fillcolor ; (set our color)
draw-rect	: draw-rect 0 800 rand 0 600 rand (begin coords) 0 800 rand 0 600 rand (second coords) rect ; (draw our rectangle)
randrect	: randrect canvas pickcanvas (sets canvas on page) 200 tokenresolution (every 200 tokens) begin pickcolor (pick a random color) putrect (draw a random rect) again

randrect.screenshot



randrect

SVFORTH Queries

- Forth also makes a very nice query and filter language.
- Queries fill the stack with results.
- Filters remove items from the stack based on criteria.
- Stack object are heterogenous so different types of data can fill the same stack.

```
twitter 500 from facebook 500 from #anonymous filter loic filter
```

- Pull 500 twitter and 500 facebook posts and filter for #anonymous tags and then further filter for ‘loic’ mentions

Demo: SVFORTH Queries

- This is **not** available in the source code, nor is there public access to the data source being used.
- Various data sources stored in a flat Postgres table.
  - Pastebin
  - Twitter
  - IRC
- Production prototype for PacketNinjas.

forth.more

word	stack diagram	description
[	[code block]	a block of code treated as a stack object
exec	code-block exec	synchronously execute code block
rpc	code-block rpc	executes code block remotely on server
apply	ds apply code-block	applies code block or word to ds

word	stack diagram	description
get-url	(url get-url)	gets objects from server
get-binary	(url get-binary)	obtains URL as binary object on stack

word	stack diagram	description
xml-to-ds	(xml-text xml-to-ds)	converts XML to data structure
ds-get	(ds index ds-get)	gets index from data structure
ds-get-all	(ds index ds-get-all)	iteratively pushes index from ds

getmalware.forth

word	code
get-rss-links	: get-rss-links xml-to-ds (convert our XML RSS to ds) channel ds-get (grab the ‘channel’ element) item ds-get (grab the ‘item’ element) link ds-get-all ; (grab all ‘item’ elements)
get-rss-binaries	: get-binary-links get-url (fetch our RSS feed URL) get-rss-links (parse our links out of RSS) apply get-binary ; (grab our links as binaries)

word	code
get-malware	: get-malware http://svforth/malware.rss get-rss-binaries ;

svforth.so-far

- Forth makes it very simple to extend the existing abilities with small pieces of functiona code.
  - Very much like Unix command line and pipes – do one thing, do it very well.
- We have retrieved links via RSS to malware binaries and fetched them.
- In SVFORTH's dialect, arbitrary and heterogenous objects including binaries can be in the stack.

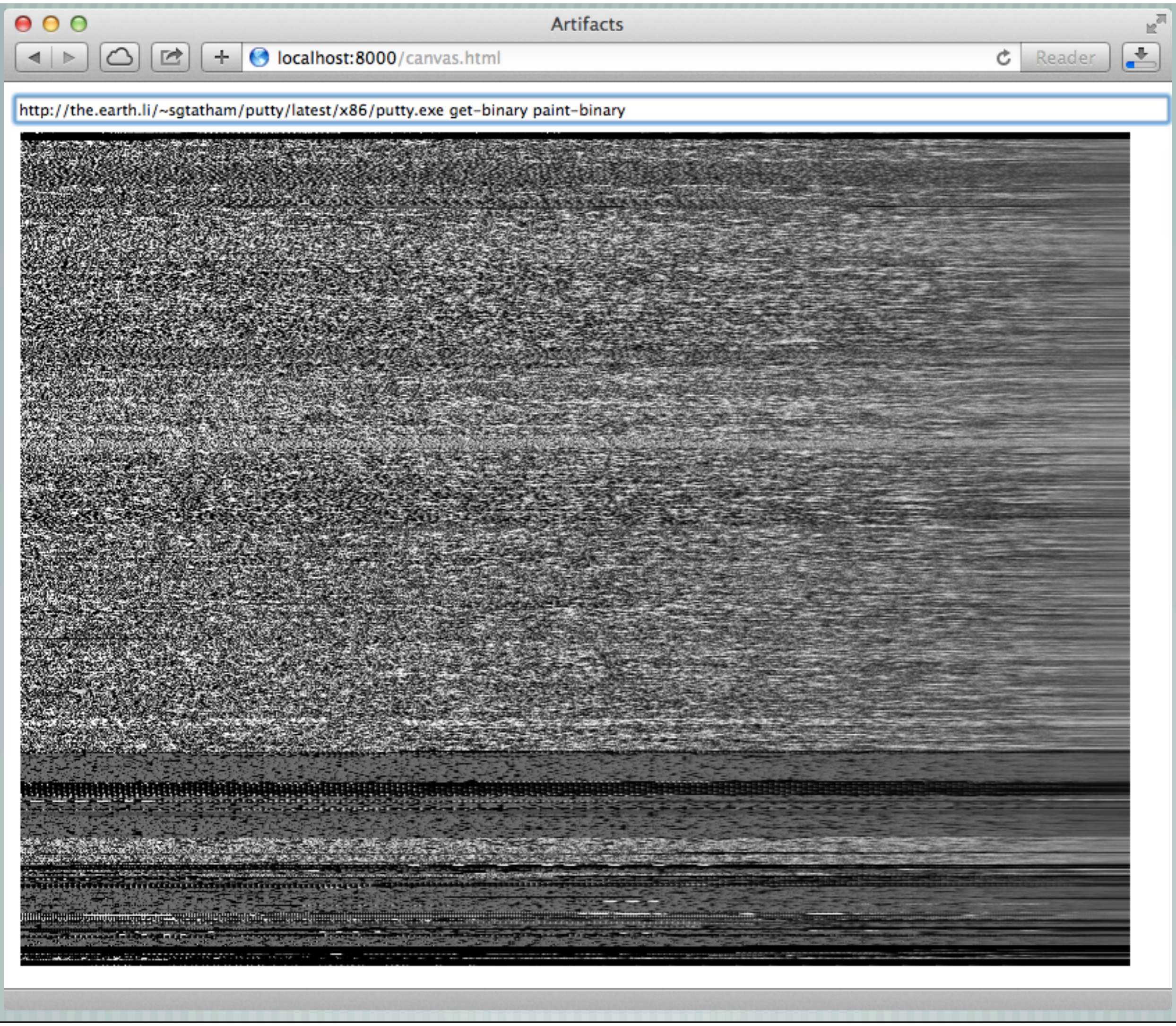
Binary Representation

- Representing binaries visually - how?
- Simplistic view is as 8-bit integers.
  - value 0-255
  - can be represented in 24-bit RGB space as grayscale by assigning the same value across Red, Green, Blue

word	stack diagram	description
get-binary	(url get-binary)	grab url as a binary on the stack
paint-binary	(binary paint-binary)	draws grayscale 8-bit representation

svforth.view.8bit

http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe get-binary paint-binary



http://the.earth.li/~sgtatham/putty/latest/x86/putty.exe get-binary paint-binary

Raw Grayscale Not Useful

- Grayscale view of 8-bit is not very useful visually.
  - Can see some areas where it is empty.
  - Mostly noise to human eyes.
- Useful for machine algorithms to cluster based on unique features.
- How to make it more useful?

Detour: Color Theory (RGB)

- Red, Green, Blue (RGB) is the de-facto standard for representation of colors at the machine and display level.
  - Web-safe colors = (6^3, 216 colors)
  - 24-bit color space (8^3, 16,777,216 colors)
- By setting values for red, green, and blue, we have colors.
- RGB is not how we perceive or think of colors!
- If we map our binaries to RGB directly, it doesn’t work.

Detour: Color Theory (HSV)

- HSV - Hue, Saturation, Value
  - Hue - Color
  - Saturation - Colorfulness
  - Value - Brightness
- Work with HSV colors, convert to RGB and back.
- Allows humans to think in terms of ‘brighter’, ‘darker’.

svforth.color

word	stack diagram	description
red	(red) -- (h s v)	push h, s, v value for red
green	(green) -- (h s v)	push h, s, v value for green
blue	(blue) -- (h s v)	push h, s, v value for blue
lighten	(h s v lighten)	lighten h s v
darken	(h s v darken)	darken h s v
saturate	(h s v saturate)	increase saturation of h s v
desaturate	(h s v desaturate)	decrease saturation of h s v
rotcolor	(h s v rotcolor)	rotate along the color wheel
-rotcolor	(h s v -rotcolor)	counter-rotate along the color wheel
rgb-to-hsv	(binary rgb-to-hsv)	convert triplets of r, g, b to h, s, v
hsv-to-rgb	(binary hsv-to-rgb)	convert triplets of h, s, v to r, g, b

Visual Encoding

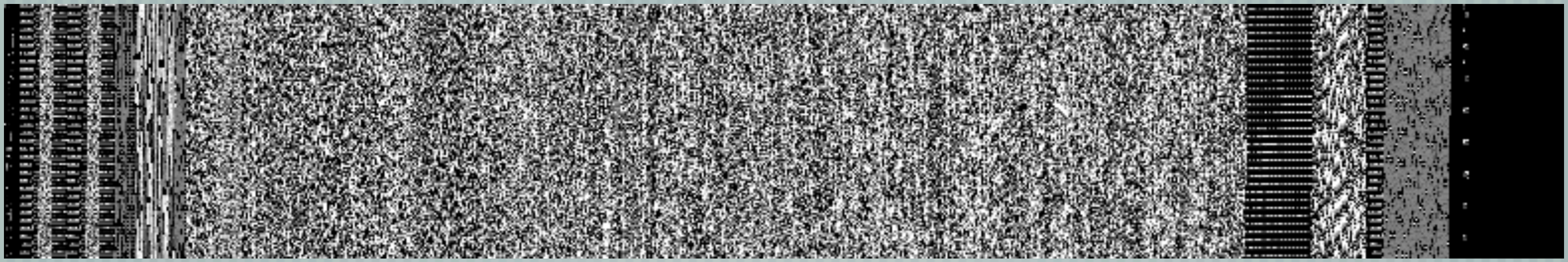
- Now that we can map to HSV – what can we encode to this based on the information in a binary file?
  - PE Sections
    - Windows PE executables have distinct sections.
  - Entropy
    - How much randomness along a set of bytes – detect if encrypted or compressed

Visual Encoding (pt 2)

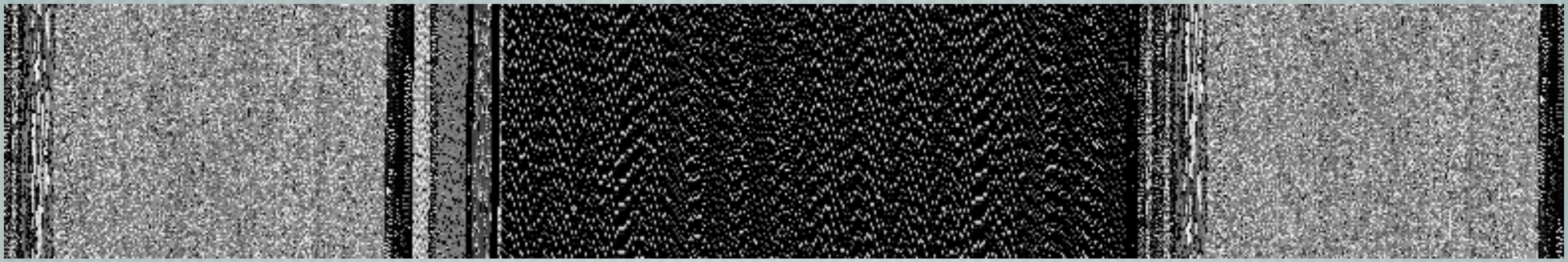
- Also ... what if we represented more meaningful data than just a byte stream?
  - Like, say, a stream of disassembled opcodes with arguments stripped out?
  - The majority of Intel opcodes lay within the 8-bit bounday, and the rest can be safely discarded.

Example 1: 00b8dc50625...

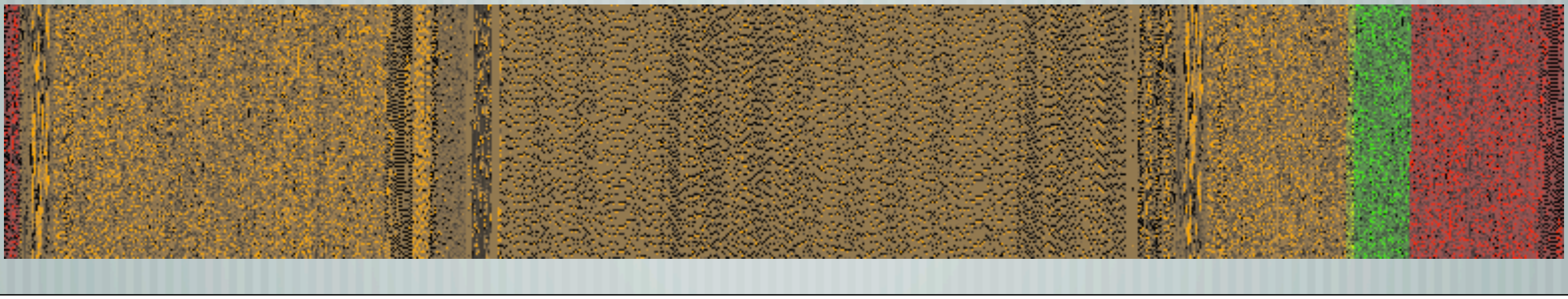
8-bit aligned values to grayscale



8-bit disassembly opcodes to grayscale

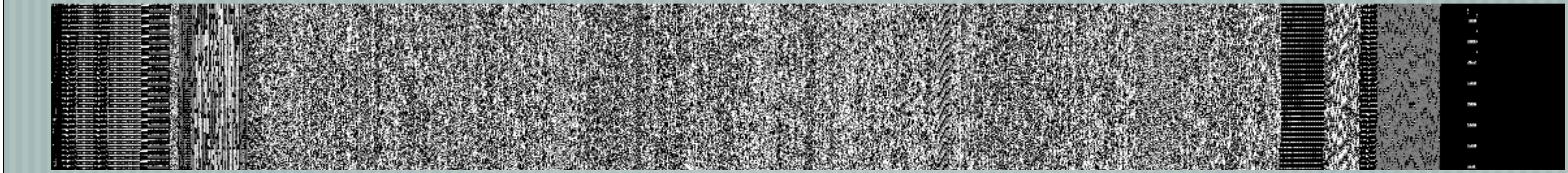


8-bit disassembly opcodes overlaid with PE sections colorized

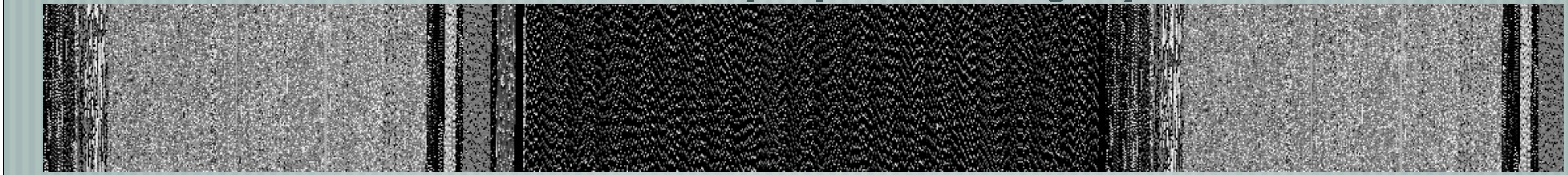


Example 2: 00d0071a86...

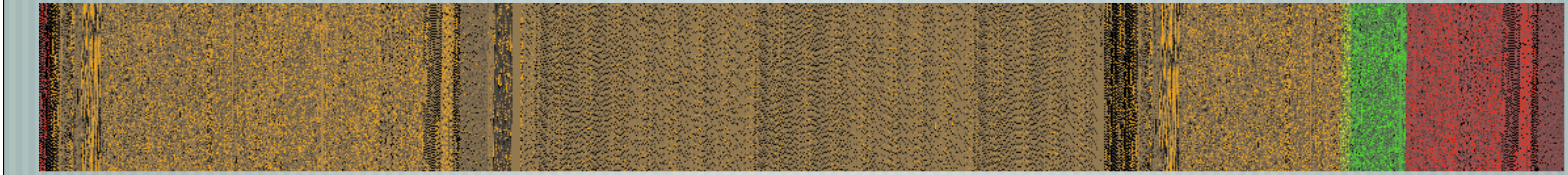
8-bit aligned values to grayscale



8-bit disassembly opcodes to grayscale

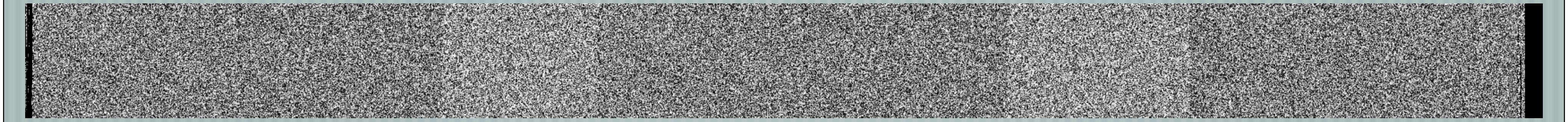


8-bit disassembly opcodes overlaid with PE sections colorized

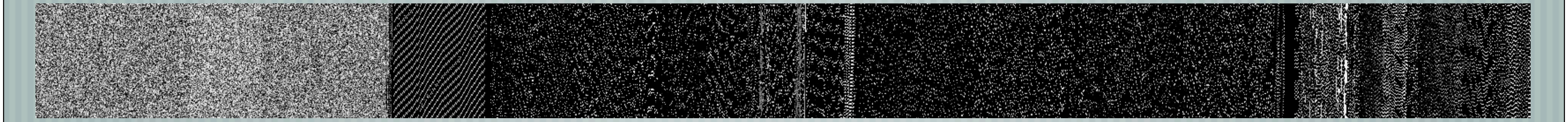


Example 3: 0038fd97d96...

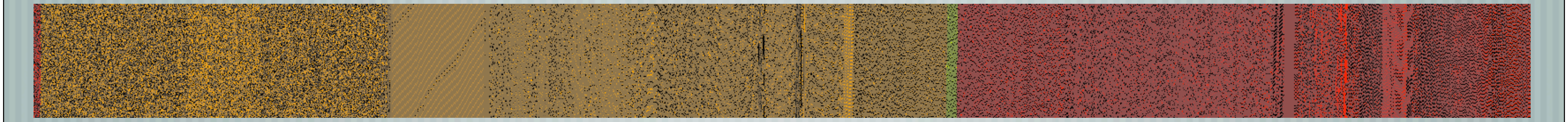
8-bit aligned values to grayscale



8-bit disassembly opcodes to grayscale

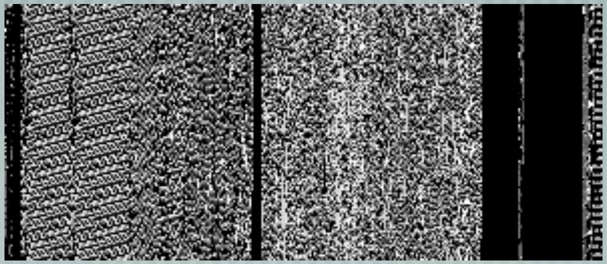


8-bit disassembly opcodes overlaid with PE sections colorized

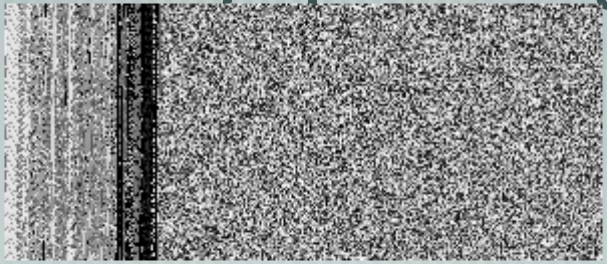


Example 4: 231ee964ade..

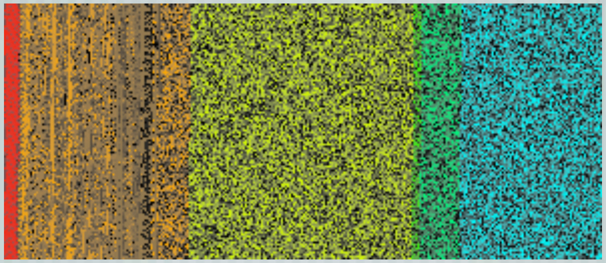
8-bit aligned values to grayscale



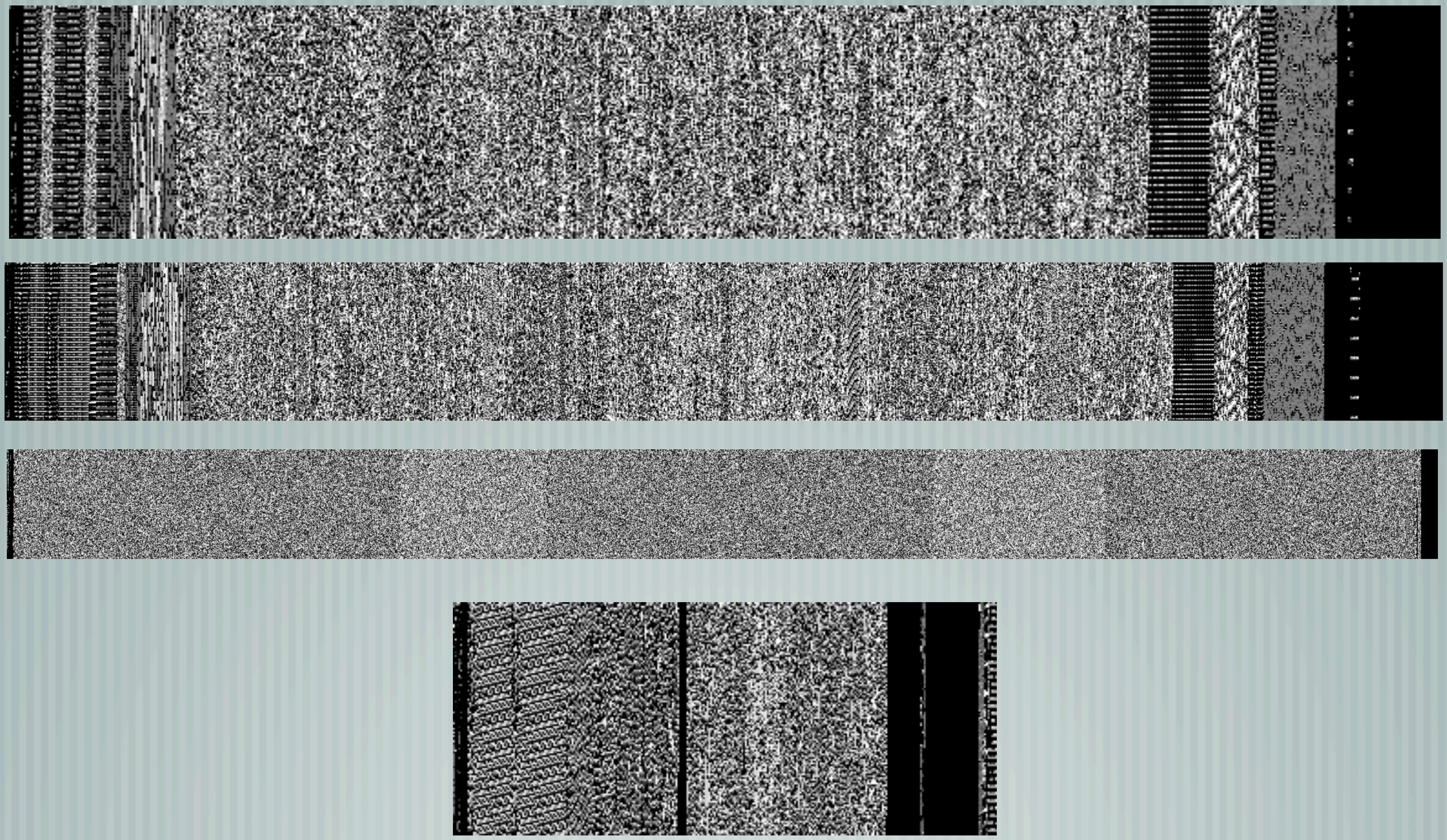
8-bit disassembly opcodes to grayscale



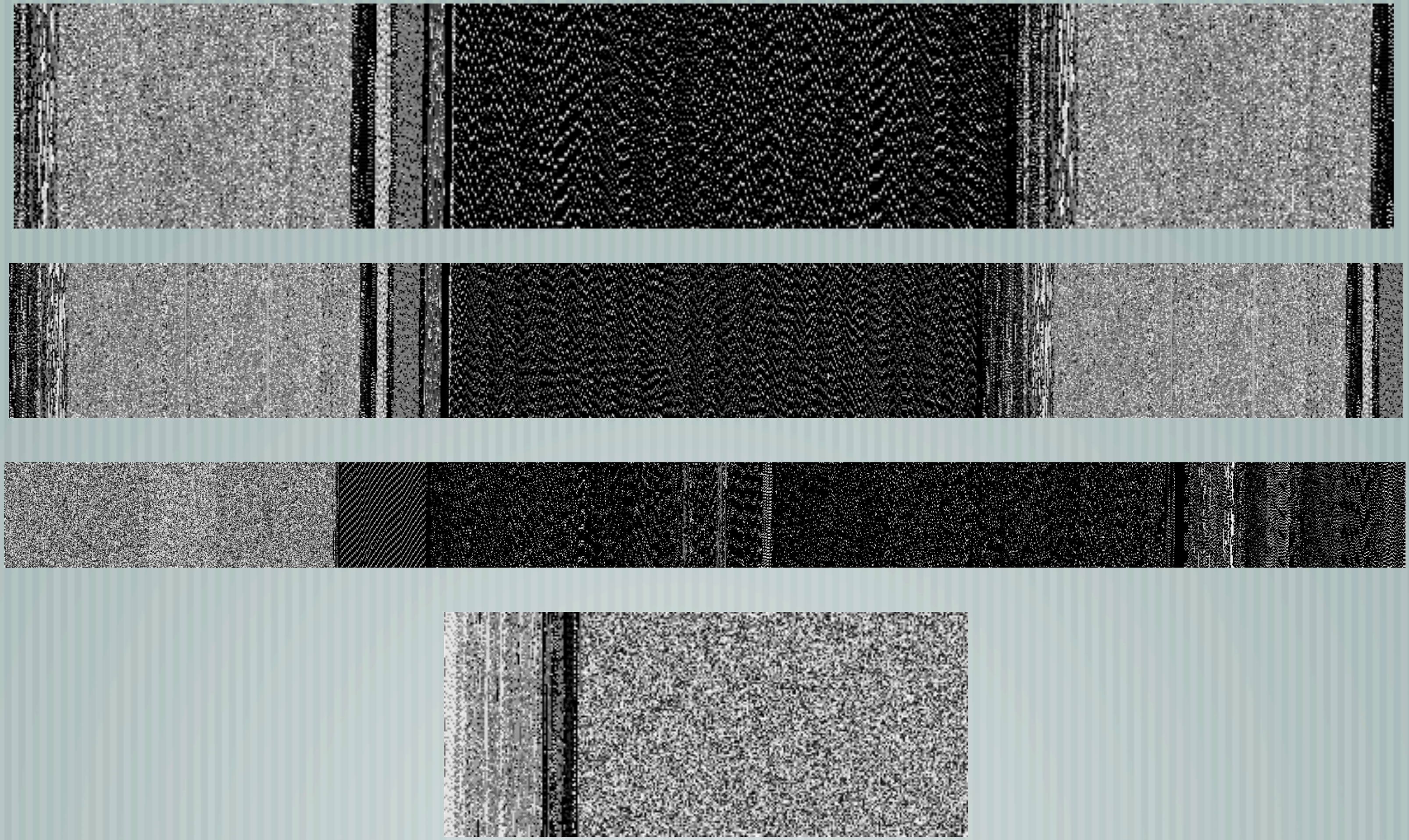
8-bit disassembly opcodes overlaid with PE sections colorized



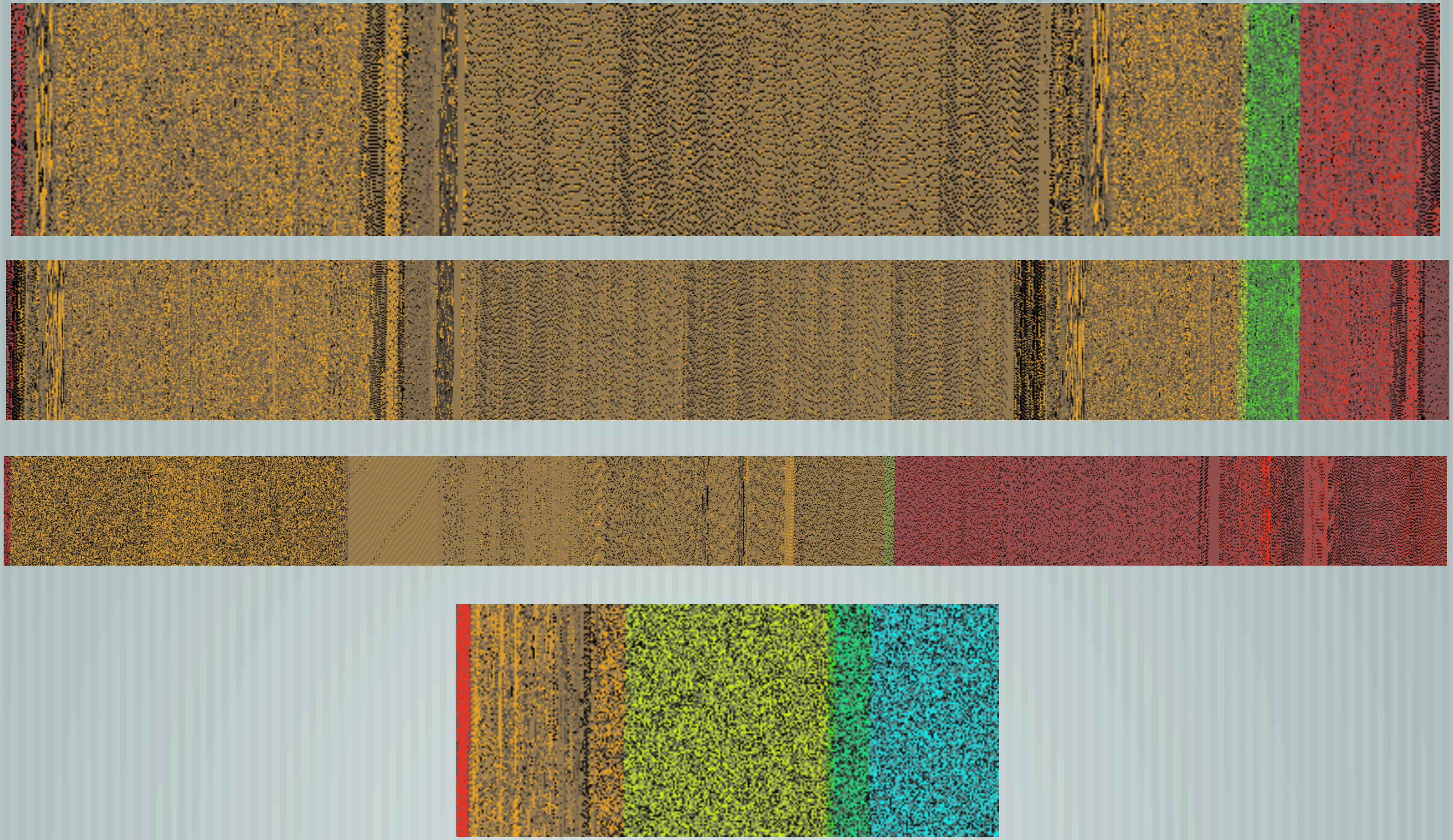
Side-by-Side: 8-bit aligned



Side-by-Side: Opcodes



Side-by-Side: PE Colorized



Intriguing Conclusions

- One thing that really stands out is that there were binaries that were fundamentally the same, structurally, despite being dramatically different sizes.
- This is something that jumps out on visual inspection, with the right view of the data. Comparing a grayscale raw binary image would not have made the difference or similarity apparent here.

Much More Work To Do

- Ongoing process to incorporate visualization shown into SVFORTH.
- Production usage of SVFORTH in a security analysis context.
- Optimization of JavaScript code and image handling.
- Object views of stack, allowing pivots on views.

Cool Stuff To Do

- asm.js
  - Traditional Forth was itself a compiler, compiling Forth words to the native assembler of the platform.
  - Why not take this in that direction, and use asm.js, which is a subdialect of JavaScript?
- D3.js
  - More easy visualization and historgram by using D3

Thanks To

- Daniel Clemens of PacketNinjas for allowing me the freedom to explore interesting solutions to his problem.
- Daniel Nowak of Spectral Security for his valuable feedback and insights into visualization and security analysis.

Questions?

- Any final questions, feedback?

Thank You

- Source code of SVFORTH so far is available online.
  - https://github.com/ephsec/svforth
- Paper covering SVFORTH is available in GitHub markdown:
  - https://github.com/ephsec/svforth/blob/master/doc/svforth.md
- wbrown@ephemeralsecurity.com