

バイナリプログラムの解析 最終レポート

1 はじめに

このレポートは情報システム実験 B K-15「バイナリプログラムの解析」の最終レポートである。実験の期間中に開発したバイナリ解析プログラムについて説明する。ソースコードは https://github.com/Alignof/binary_program_analysis から参照できる。

2 機能の説明

各機能ごとに詳細を説明をする。なお機能の一覧は以下のコマンドで確認できる。

```
$ ./unlibit --help
```

```
unlibit 0.1.0
```

USAGE:

```
unlibit [OPTIONS] <filename>
```

ARGS:

```
<filename>    target file path
```

OPTIONS:

-h, --header	Show header
-p, --program	Show all segments
-s, --section	Show all sections
-d, --disasem	Disassemble ELF/PE
-a, --analyze	Analyze target binary file
--dump	Dump binary file
--diff <other>	Take a diff of the binary files
--histogram	Show byte histogram
--help	Print help information
-V, --version	Print version information

2.1 ELF と PE を解析する機能

まず、ELF と PE の構造を理解するためそれらを解析する機能を付けた。32bit 向けの ELF ファイルを解析するプログラムは書いたことがあったのと授業課題で PE ファイルの解析が出たので ELF と PE 双方で 32bit/64bit の両方をサポートすることにした。

ELF では ELF ヘッダ、セクションヘッダ、プログラムヘッダが表示可能である。PE では MS-DOS ヘッダ、NT ヘッダ、セクションヘッダが表示可能である。

今回は ELF と PE を同じように処理できるように trait で抽象化を行った。ELF と PE のローダを抽象化した trait, Loader と ELF での実装例をリスト 1 に示す。

Listing 1: src/loader.rs

```
1 pub trait Loader {
2     fn mem_data(&self) -> &[u8];
3     fn header_show(&self);
4     fn show_segment(&self);
5     fn show_section(&self);
6     fn disassemble(&self);
7     fn show_all_header(&self);
8     fn analysis(&self);
9 }
10 impl Loader for ElfLoader {
11     fn mem_data(&self) -> &[u8] {
12         &self.mem_data
13     }
14
15     fn header_show(&self) {
16         self.elf_header.show();
17     }
18
19     fn show_segment(&self) {
20         for (id, prog) in self.prog_headers.iter().enumerate() {
21             prog.show(id);
22             println!("\n\n");
23         }
24     }
25
26     fn show_section(&self) {
27         for (id, sect) in self.sect_headers.iter().enumerate() {
```

```

28         sect.show(id);
29         println!("\n\n");
30     }
31 }
32
33 fn disassemble(&self) {
34     for (id, sect) in self.sect_headers.iter().enumerate() {
35         sect.show(id);
36         sect.dump(&self.mem_data);
37         println!("\n\n");
38     }
39 }
40
41 fn show_all_header(&self) {
42     self.elf_header.show();
43
44     println!("\n\n");
45
46     for (id, prog) in self.prog_headers.iter().enumerate() {
47         prog.show(id);
48     }
49
50     println!("\n\n");
51
52     for (id, sect) in self.sect_headers.iter().enumerate() {
53         sect.show(id);
54     }
55 }
56
57 fn analysis(&self) {
58     let mut inst_list_overall = HashMap::new();
59     for func in self.functions.iter() {
60         let mut inst_list = HashMap::new();
61         let call_addrs = func.inst_analysis(&mut inst_list,
↪ &self.mem_data);
62
63         for (name, count) in inst_list.clone() {
64             *inst_list_overall.entry(name).or_insert(0) +=
↪ count;
65         }
66

```

```

67         let mut inst_list =
↳ inst_list.iter().collect::

```

リスト 1: Loader trait

Loader trait では,

- 構造体内部にあるメモリマップのデータを貸し出す `mem_data`
- ヘッダを表示する `header_show`
- セグメントを表示する `show_segment`
- セクションを表示する `show_section`
- ディスアセンブルを表示する `disassemble`
- すべてのヘッダを表示する `show_all_header`
- データを元に命令を解析する `analysis`

の 7 つを共通の振る舞いとして定めた.

`mem_data` は中身を貸し出すだけなので必要無さそうだが, `trait` はメンバにアクセスするために実装先のオブジェクトを指定する必要があるので `trait` の宣言に共通して書くことができない. PE ではセグメントという概念が存在しないのでセクションヘッダを代わりに表示するようになっている.

また今回は解析に必要なのでディスアセンブルの機能も付けた. ただし, `x86_64` のデコードを手で実装するのは大変なので `iced_x86`[\[1\]](#) というライブラリを使用した. これにより簡単に `x86_64` のデコードや命令の表示が可能になった. 今回は `x86_64` にのみ対応したが, 必要になれば RISC-V 向けの手書きのデコーダが手元にあるのでそれが使える他, Arm など簡単に扱えるツール [\[2\]](#) があるため必要になり次第対応しようと思う.

2.2 バイナリの hex dump をヒートマップで表示できる機能

バイナリをヒートマップで色付けした hex dump で表示できる機能を実装した. 実際の出力を図 1 に示す.

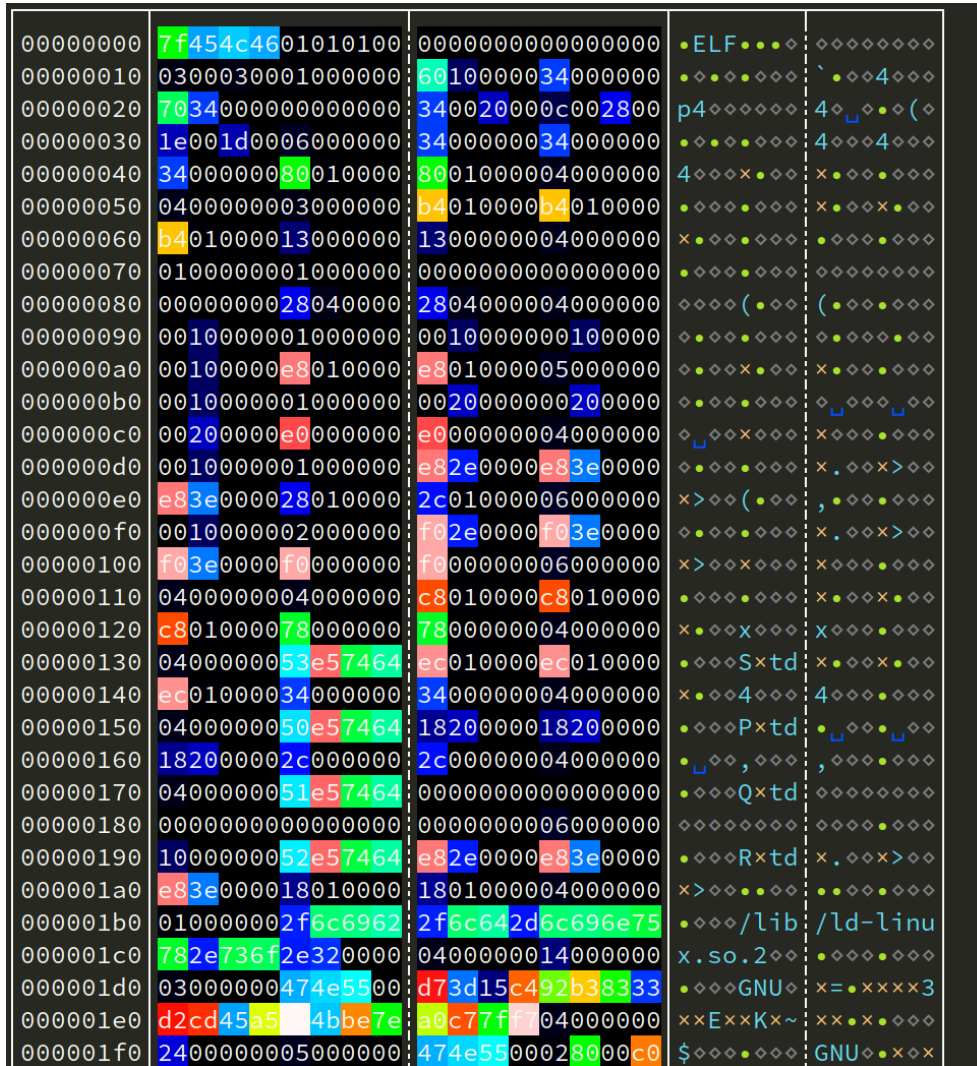


図 1: ヒートマップ表示された hex dump

各バイトの数値によって数字を割り当てて、セクションの境界や性質を直感的に感じることができる。数値の色の対応づけは 0x00 に黒、0xff に白を割り当て数が大きくなるごとに明るい色になるようにしている。具体的には黒→青→水色→緑→黄色→赤→白と段階的に変わっていく。

数値と色の具体的な対応を図 2 に示す。



図 2: 数値の色の具体的な対応

この配色の問題点は緑色が占める部分の多いことと ascii 範囲である 0x70 0x7f と 0x80 0x8f の見分けが付きづらいということが挙げられるが dump の横には後述する ascii 表示がある上、仮に隣り合っているときにはそもそも ascii かどうかの区別は不要な場合（ascii 領域に非 ascii のバイトが来るとは考えづらい）であるのでこのまま採用した。

もし仮に改善するならば水色の次を紫にして 0x80 からを緑にする方法がある。ただしこれはルールに一貫性が無いことを考慮しなくてはならない。

配色の実装を以下に示す.

Listing 2: visualize.rs

```
1 fn hex_to_color(hex: u8) -> (u8, u8, u8) {
2     const STEP: u8 = 6;
3     let step_up = |start: u8| (hex - start).saturating_mul(STEP);
4     let step_down = |start: u8| 255_u8.saturating_sub((hex -
5     ↪ start).saturating_mul(STEP));
6     let red = match hex {
7         0..=127 => 0,
8         128..=169 => step_up(128),
9         170..=255 => 255,
10    };
11    let green = match hex {
12        0..=41 => 0,
13        42..=83 => step_up(42),
14        84..=169 => 255,
15        170..=211 => step_down(170),
16        212..=255 => step_up(212),
17    };
18    let blue = match hex {
19        0..=41 => step_up(0),
20        42..=83 => 255,
21        84..=127 => step_down(84),
22        128..=211 => 0,
23        212..=255 => step_up(212),
24    };
25    (red, green, blue)
26 }
```

リスト 2: 数値と配色を対応付ける処理

色を黒 (0,0,0), 青 (0,0,1), シアン (0,1,1), 緑 (0,1,0), 黄 (1,1,0), 赤 (1,0,0), 白 (1,1,1) の間の領域に分けてその間で段階的に増加・減少を行っている. この処理はもっときれいに書けそうなので反省点である.

次に dump の横に表示されている記号の表示について説明する. 右側には dump のバイト数と同じだけの記号が置かれている. この表示は普段自分が使っている hexyl[3] という hex viewer を参考に実装したもので以下の意味を持つ.

これにより hex の大まかな種類とバイナリに埋め込まれている ascii 文字を読むことができる. ascii 文字を読めることは解析において非常に重要な情報である.

表 1: 記号と色の意味

表示	記号	意味
灰色のダイヤ	☒	0x00
青色の記号	␣	空白文字 (0x20)
緑色の文字	a	ascii 表示可能なもの
水色の丸	•	0x7f より小さい ascii 表示外の数
黄色のバツ	×	0x80 以上の ascii 表示外の数

2.3 バイナリの diff を取る機能

バイナリの diff を取る必要があることが良くあるが、コマンドライン上で diff や delta のように手軽に diff を取れるツールは見つからない。普段は hexyl の出力を左右に並べて目で比較したり hexyl の出力を delta に与えて比較する方法 (図 3) を取っている。

/proc/self/fd/13 → /proc/self/fd/14

1:

00000000	d0 0d fe ed 00 00 06 66	00 00 00 38 00 00 05 18	x_××××.f	××××××.f
00000000	d0 0d fe ed 00 00 06 6a	00 00 00 38 00 00 05 1c	x_××××.j	××××××.j
00000010	00 00 00 28 00 00 00 11	00 00 00 10 00 00 00 00	××××(××××	××××××××
00000020	00 00 01 4e 00 00 04 e0	00 00 00 00 00 00 00 00	×××.N×××.x	××××××××
00000020	00 00 01 4e 00 00 04 e4	00 00 00 00 00 00 00 00	×××.N×××.x	××××××××
00000030	00 00 00 00 00 00 00 00	00 00 00 01 00 00 00 00	××××××××	××××××××
00000040	00 00 00 03 00 00 00 04	00 00 00 00 00 00 00 02	××××××××	××××××××
00000050	00 00 00 03 00 00 00 04	00 00 00 0f 00 00 00 02	××××××××	××××××××

80:

000004e0	00 00 00 01 00 00 00 02	00 00 00 02 00 00 00 01	××××××××	××××××××
000004f0	68 74 69 66 00 00 00 00	00 00 00 03 00 00 00 0a	htif××××	××××××××_
00000500	00 00 00 1b 75 63 62 2c	68 74 69 66 30 00 00 00	××××.ucb,	htif××××
00000510	00 00 00 02 00 00 00 02	23 61 64 64 72 65 73 73	××××××××	#address
00000520	2d 63 65 6c 6c 73 00 23	73 69 7a 65 2d 63 65 6c	-cells××	size-cel
00000530	6c 73 00 63 6f 6d 70 61	74 69 62 6c 65 00 6d 6f	ls×compa	tible×mo
00000540	64 65 6c 00 73 74 64 6f	75 74 2d 70 61 74 68 00	del×stdo	ut-path×
00000550	62 6f 6f 74 61 72 67 73	00 74 69 6d 65 62 61 73	bootarg×	×timebas
00000560	65 2d 66 72 65 71 75 65	6e 63 79 00 64 65 76 69	e-freque	ncy×devi
00000570	63 65 5f 74 79 70 65 00	72 65 67 00 73 74 61 74	ce_type×	reg×stat
00000580	75 73 00 72 69 73 63 76	2c 69 73 61 00 6d 6d 75	useriscv	,isa×mmu
00000590	2d 74 79 70 65 00 72 69	73 63 76 2c 70 6d 70 72	-type×ori	scv,pmpr
000005a0	65 67 69 6f 6e 73 00 72	69 73 63 76 2c 70 6d 70	egions×	iscv,pmp
000005b0	67 72 61 6e 75 6c 61 72	69 74 79 00 63 6c 6f 63	granular	ity×cloc
000005c0	6b 2d 66 72 65 71 75 65	6e 63 79 00 23 69 6e 74	k-freque	ncy×#int
000005d0	65 72 72 75 70 74 2d 63	65 6c 6c 73 00 69 6e 74	errupt-c	ells×oint
000005e0	65 72 72 75 70 74 2d 63	6f 6e 74 72 6f 6c 6c 65	errupt-c	ontrolle
000005f0	72 00 70 68 61 6e 64 6c	65 00 72 61 6e 67 65 73	rophandl	e×ranges
00000600	00 69 6e 74 65 72 72 75	70 74 73 2d 65 78 74 65	×interru	pts-exte
00000610	6e 64 65 64 00 72 69 73	63 76 2c 6e 64 65 76 00	nded×oris	cy,n×dev×
00000620	72 69 73 63 76 2c 6d 61	78 2d 70 72 69 6f 72 69	riscv,ma	x-priori
00000630	74 79 00 69 6e 74 65 72	72 75 70 74 2d 70 61 72	ty×inter	rupt-par
00000640	65 6e 74 00 69 6e 74 65	72 72 75 70 74 73 00 72	ent×inte	rruptsor
00000650	65 67 2d 73 68 69 66 74	00 72 65 67 2d 69 6f 2d	eg-shift	×reg-io-
00000660	77 69 64 74 68 00		width×	
00000510	00 00 00 02 00 00 00 02	00 00 00 09 23 61 64 64	××××××××	××××_#add
00000520	72 65 73 73 2d 63 65 6c	6c 73 00 23 73 69 7a 65	ress-cel	ls×#size
00000530	2d 63 65 6c 6c 73 00 63	6f 6d 70 61 74 69 62 6c	-cells×	ompatibl
00000540	65 00 6d 6f 64 65 6c 00	73 74 64 6f 75 74 2d 70	e×model×	stdout-p
00000550	61 74 68 00 62 6f 6f 74	61 72 67 73 00 74 69 6d	ath×boot	arg××tim
00000560	65 62 61 73 65 2d 66 72	65 71 75 65 6e 63 79 00	ebase-fr	equency×
00000570	64 65 76 69 63 65 5f 74	79 70 65 00 72 65 67 00	device_t	ype×rego
00000580	73 74 61 74 75 73 00 72	69 73 63 76 2c 69 73 61	status×	iscv,isa
00000590	00 6d 6d 75 2d 74 79 70	65 00 72 69 73 63 76 2c	×mmu-typ	e×orisvc
000005a0	70 6d 70 72 65 67 69 6f	6e 73 00 72 69 73 63 76	pmpregio	ns×riscv
000005b0	2c 70 6d 70 67 72 61 6e	75 6c 61 72 69 74 79 00	,pmpgran	ularity×
000005c0	63 6c 6f 63 6b 2d 66 72	65 71 75 65 6e 63 79 00	clock-fr	equency×
000005d0	23 69 6e 74 65 72 72 75	70 74 2d 63 65 6c 6c 73	#interru	pt-cells
000005e0	00 69 6e 74 65 72 72 75	70 74 2d 63 6f 6e 74 72	×interru	pt-contr
000005f0	6f 6c 6c 65 72 00 70 68	61 6e 64 6c 65 00 72 61	olleroph	andle×ra
00000600	6e 67 65 73 00 69 6e 74	65 72 72 75 70 74 73 2d	nges×int	errupts-
00000610	65 78 74 65 6e 64 65 64	00 72 69 73 63 76 2c 6e	extended	×riscv,n
00000620	64 65 76 00 72 69 73 63	76 2c 6d 61 78 2d 70 72	dev×risc	v,max-pr
00000630	69 6f 72 69 74 79 00 69	6e 74 65 72 72 75 70 74	riority×	nterrupt
00000640	2d 70 61 72 65 6e 74 00	69 6e 74 65 72 72 75 70	-parent×	interrup

lines 1-65

図 3: hexyl の出力を delta に与えて比較する方法

しかし出力してからテキストで比較する方法は元のツールの色の情報が失われてしまう上、1 行ごとの比較には向かない。そこで、今回は上の hex dump の機能を拡張する形でバイナリファイルから直接 diff を取れる機能を開発した。

実際に使用している際のスクリーンショットを図 4 に示す。

000003f0	0000000001000000	0000000300000004	00000000	00000000
00000400	000000fd0000001f	0000000300000004	00000000	00000000
00000410	000001080000000f	0000000300000004	00000000	00000000
00000420	000000b400000001	0000000300000000	00000000	00000000
00000430	000000c500000003	00000004000000da	00000000	00000000
00000440	0000000200000002	000000016e733136	00000000	00000000
00000450	3535304031303030	3030303000000000	550@1000	00000000
00000460	0000000300000009	00000001b6e733136	00000000	00000000
00000470	3535306100000000	0000000300000004	550a	00000000
00000480	000000a400989680	0000000300000004	00000000	00000000
00000490	0000011b00000002	0000000300000004	00000000	00000000
000004a0	0000012c00000001	0000000300000010	00000000	00000000
000004b0	0000006000000000	1000000000000000	00000000	00000000
000004c0	0000010000000003	0000000400000137	00000000	00000000
000004d0	0000000000000003	0000000400000141	00000000	00000000
000004e0	0000000100000002	0000000200000001	00000000	00000000
000004f0	6874696600000000	000000030000000a	htif	00000000
00000500	0000001b7563622c	6874696630000000	ucb,htif	00000000
00000510	0000000200000002	0000000923616464	00000000	00000000
...	...	2361646472657373
00000520	726573732d63656c	6c73002373697a65	ress-cel	ls#size
...	2d63656c6c730023	73697a652d63656c
00000530	2d63656c6c730063	6f6d70617469626c	-cells	c ompatibl
...	6c7300636f6d7061	7469626c65006d6f
00000540	65006d6f64656c00	7374646f75742d70	e model	stdout-p
...	64656c007374646f	75 2d7061 6800
00000550	61746800626f6f74	617267730074696d	ath boot	args tim
...	626f6f7461726773	0074696d65626173
00000560	65626173652d6672	657175656e637900	ebase-fr	equency
...	2d6672 717565	6e63790064657669
00000570	6465766963655f74	7970650072656700	device_t	ype reg
...	63 5f7479706500	726567 73746174
00000580	7374617475730072	697363762c697361	status	r iscv,isa
...	7573007269 6376	2c697361006d6d75
00000590	006d6d752d747970	650072697363762c	mmu-typ	e riscv,
...	2d74797065007269	7363762c706d7072
000005a0	706d70726567696f	6e73007269736376	pmpregio	ns riscv
...	6567696f6e730072	69 63762c706d70
000005b0	2c706d706772616e	756c617269747900	,pmpgran	ularity
...	6772616e756c 72	69747900636c6f63
000005c0	636c6f636b2d6672	657175656e637900	clock-fr	equency
...	6b2d667265717565	6e63790023696e74
000005d0	23696e7465727275	70742d63656c6c73	#interru	pt-cells
...	6572727570742d63	656c6c7300696e74
000005e0	00696e7465727275	70742d636f6e7472	interru	pt-contr
...	6572727570742d63	6f6e7472 6c6c65
000005f0	6f6c6c6572007068	616e646c65007261	oller	phandle ra
...	72007068616e646c	650072616e676573
00000600	6e67657300696e74	657272757074732d	nges	interru
...	00696e7465727275	7074732d65787465
00000610	657874656e646564	0072697363762c6e	extended	riscv,n
...	6e64656400726973	63762c6e64657600
00000620	6465760072697363	762c6d61782d7072	dev	riscv,max-pr
...	72697363762c6d61	782d7072696f7269
00000630	696f726974790069	6e74657272757074	riority	i interrupt
...	747900 6e746572	727570742d706172
00000640	2d706172656e7400	696e746572727570	-parent	interrupt
...	656e740069 65	7272757074730072
00000650	7473007265672d73	6869667400726567	ts	reg-s hift reg
...	65672d7368696674	007265672d696f2d
00000660	2d696f2d77696474	6800	-io-widt	h
...	77 64746800

図 4: diff の機能のスクリーンショット

差分のある行のみ，下に比較対象の dump を表示している．これにより 1 行単位でどこが違うかを簡単に比較できる．また，差分のあるバイト列を比較元のファイルで

は緑に比較先のファイルでは赤にハイライト表示している．差分の無い部分は表示もハイライトもしていない．これにより一層差分のある部分が明快になった．

前述の通りこの機能は hex viewer の機能に追加して実装されたので ascii 表示はハイライトを保ったまま表示される．なお ascii 表示は比較元のもののみを表示している．どちらも表示するパターンやここにも diff をつけるパターンも試したが，見やすさを優先した結果現在の表示形式になった．

2.4 ヒストグラムを表示する機能

授業でバイナリヒストグラムを作成する課題があったのでこれを機能の一部として組み込むことにした．課題中では画像形式で出力をしていたが，これを CUI で表示するように変更した．表示には Unicode の Block Elements を使用した [4]．

表示のための実装をリスト 3 に示す．

Listing 3: visualize.rs

```
1 pub fn create_byte_histogram(mem_data: &[u8]){
2     use colored::Colorize;
3
4     let mut histogram = (0..=255)
5         .collect::
```



```

25     const BAR: [&str; 8] = ["", "| ", "█ ", "██ ", "███ ", "████ ", "█████ ",
↪    "██████"];
26     let mut histogram: Vec<(&u8, &u32)> =
↪    histogram.iter().collect();
27     histogram.sort_by(|a, b| (a.0).cmp(b.0));
28     for (hex, count) in histogram.iter() {
29         print!("{hex:02x}: ");
30         if **count < 512 {
31             (0..(*count / 8)).for_each(|_| print!("█"));
32             println!("{}", {count}, BAR[(*count % 8) as usize]);
33         } else {
34             (0..=67).for_each(|_| print!("█"));
35             println!("████ {count}");
36         }
37     }
38     for (hex, count) in histogram.iter() {
39         let percent = **count as f64 / max_count as f64;
40         let count = (percent * 255.0) as u8;
41         let (r, g, b) = hex_to_color(count);
42         print!("{}", format!("{hex:02x}").on_truecolor(r, g, b));
43
44         if *hex % 16 == 15 {
45             println();
46         }
47     }
48     println();
49 }

```

リスト 3: ヒストグラムを表示するためのプログラム

HashMap を用意して分布を数え上げ、個数に応じて四角を出力している。分布に応じた四角が想定の画面の幅を上回りそうなときはフェードアウトの処理を入れ個数だけを表示する。この方法の問題点としてファイルサイズが非常に巨大な場合はすべての分布が幅を超えてしまうことが挙げられる。対策が必要な場合は全体のファイルサイズや分布に応じてスケールを変えることで対応する必要がある。

また、最大値を基準にその割合で表示を変える方法も検討したが、ファイルの傾向として 0x00 が極端に多いのでそれを基準にすると他の分布の表現力が極端に落ちてしまう可能性があったので採用しなかった。

ヒストグラムは縦に並べて表示されるがこれだと 256 行分の出力があるので一覧性に乏しい。そこで前述のヒートマップを使って簡単に概要を観ることができる機能も

付けた。

出力を図 5,6 に示す。



図 5: ELF のヒストグラム



図 6: 左の ELF を UPX にかけたもの

左の図 5 がある ELF ファイルのバイナリヒストグラムをヒートマップで表示したものである。前述の通り 0x00 が多いことが分かる。ついで 0xff や 0x48 が多い。

右の図 6 は左の図で使った ELF ファイルを UPX にかけた後にバイナリヒストグラムを取ったものである。0x00 が多いのは変わらないが、黒がほとんどなく、色の分布がかなり広がっていることが視覚的に分かる。ただし図を見れば分かるようにまったく均一ではなく下位 4bit が 0 だったり 0xf だったりする左右の端に縦の帯が出来ている。また 0x40 の倍数の行は横に帯が出来ている。先頭 0x00 0x08 までが多いことも分かる。

このように簡単に分布を体感することができる。

3 おわりに

今回の実験期間を通してバイナリ解析に関わるいくつかの機能を備えたプログラムの開発を行った。ELF や PE の解析ツールの再発明だけでなく、自分の欲しい独自の機能を実装した

今後の展望としてまず、車輪の再発明として実装したバイナリ解析ツールの機能の充実化を図りたい。学習のために普段から触れている ELF と普段あまり触れない PE ファイルの両方に対応した解析ツールを書いたが、まだまだ足りない機能も多い。機能を実装すれば実装するほど詳しくなることができるので更に開発を進めていきたい。特に File Attributes のまわりは不足しているのできちんと勉強したい。

また、今回実装した機能のうちヒートマップ機能付きの hex viwer とバイナリの diff を取るツールは個人的に良いものが出来たと考えているのでレポジトリを別で作って名前を付けて開発をしようと思う。2 つとも小さな機能のプログラムなので hex viwer に diff の機能を追加する形で統合して 1 つのツールとして開発をしようと考えている。section 情報を併記するなどの機能を付けても面白いかもしれない。

今回の経験を活かして radare2 のようなもっと大きな解析ツールを開発することも考えている。実際に開発するなら今作っているような小さなプログラムから少しずつ機能を足してインクリメンタルに実装していくことになると思う。モダンで力のある解析ツールが GUI の ghidra の一強になっているのが個人的に気に入らないので、CUI から使える拡張性の高いモダンな解析ツールには意欲がある。今開発しているエミュレータに一区切りついたら作りたいものに加えておこうと思う。

参考文献

- [1] iced_x86 - Rust. 2022 年 2 月 8 日閲覧.
https://docs.rs/iced-x86/latest/iced_x86/
- [2] bad64 - Rust. 2022 年 2 月 8 日閲覧.
<https://docs.rs/bad64/latest/bad64/index.html>
- [3] bad64 - Rust. 2022 年 2 月 8 日閲覧.
<https://docs.rs/bad64/latest/bad64/index.html>
- [4] Unicode Block “Block Elements” . 2022 年 2 月 9 日閲覧.
<https://www.compart.com/en/unicode/block/U+2580>