MONTANA STATE UNIVERSITY

11/18/2025

Malicious Code Analysis

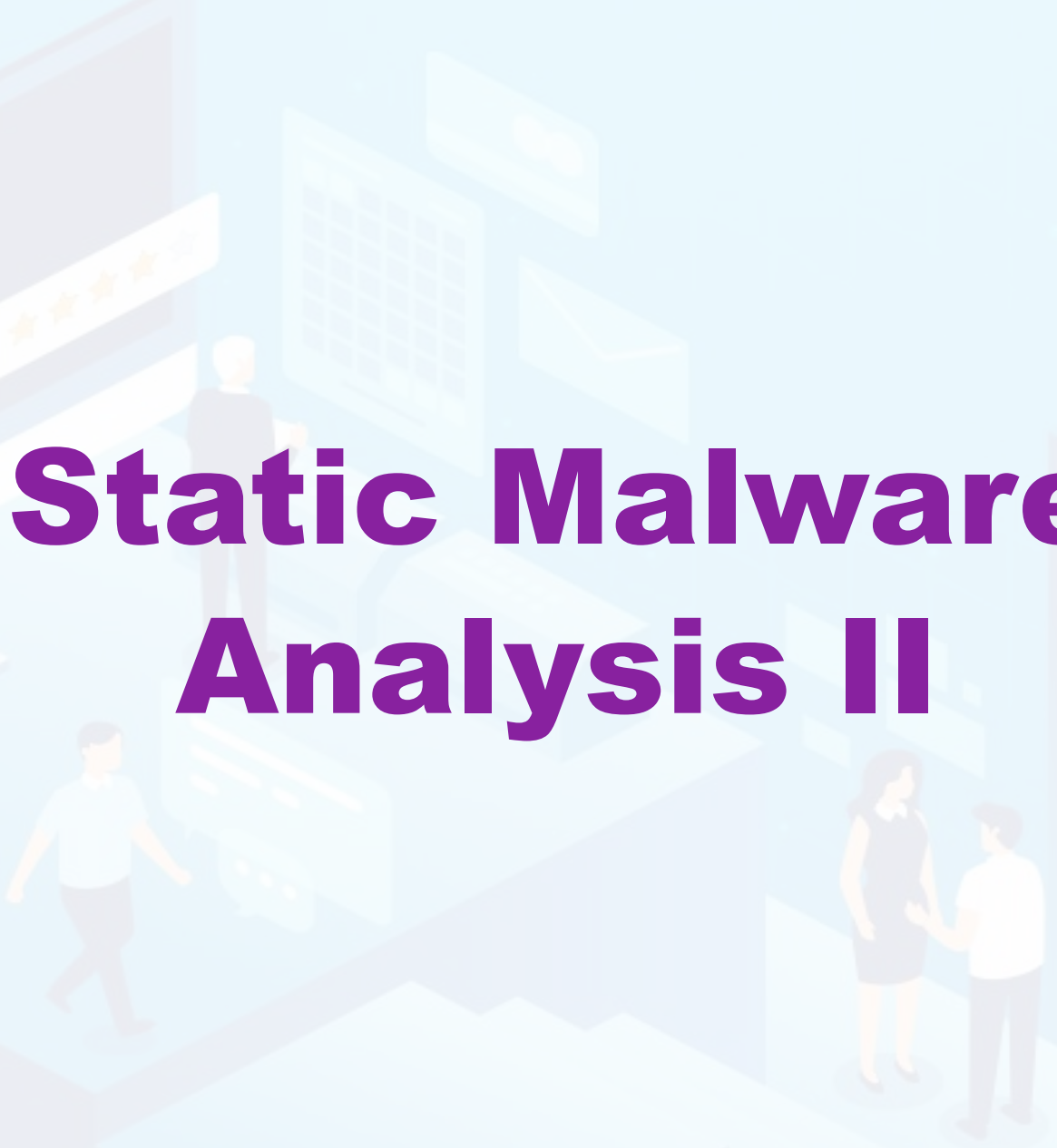
Fangtian Zhong
CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

Part One

01

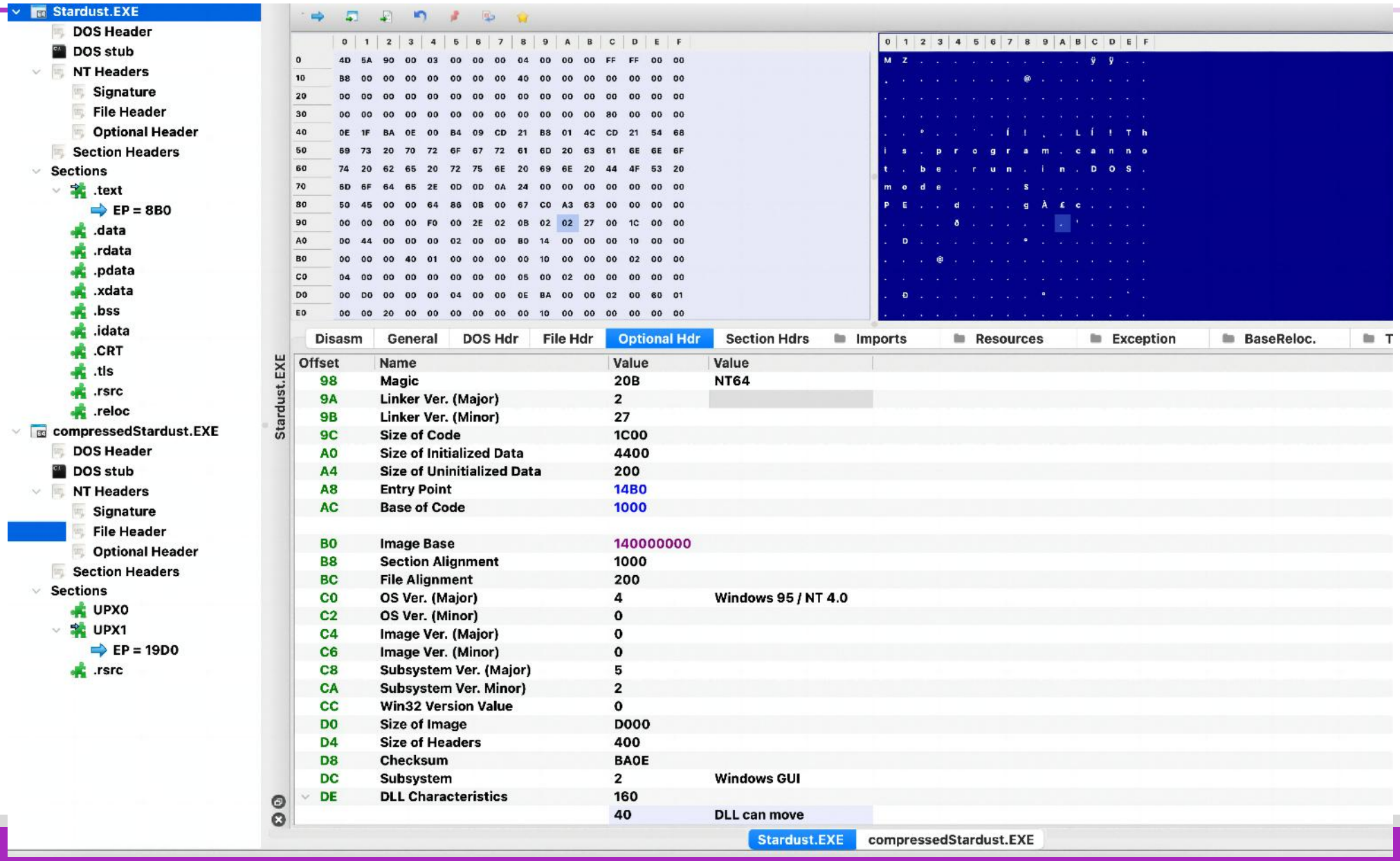
Static Malware Analysis II



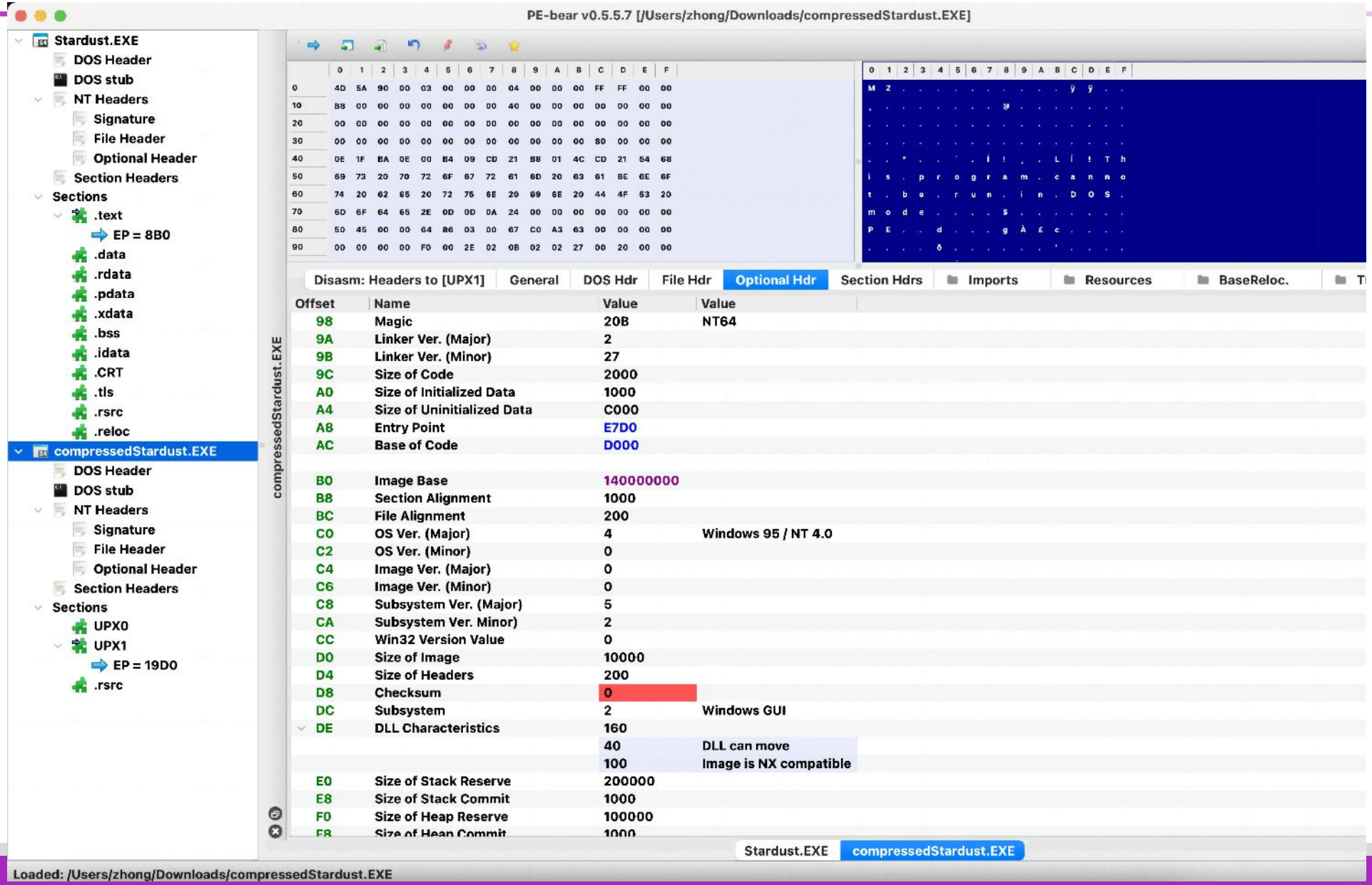
Fixed-point Scanning

- Fixed-point scanning involves analyzing a specific point within the file that is known to be part of the legitimate code. This could be a function or other segment of the code. By analyzing the code at this fixed point, analysts can determine whether it has been modified or replaced with malicious code.

Entry-point-Stardust

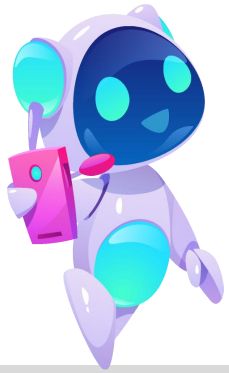


Entry-point-compressedStardust



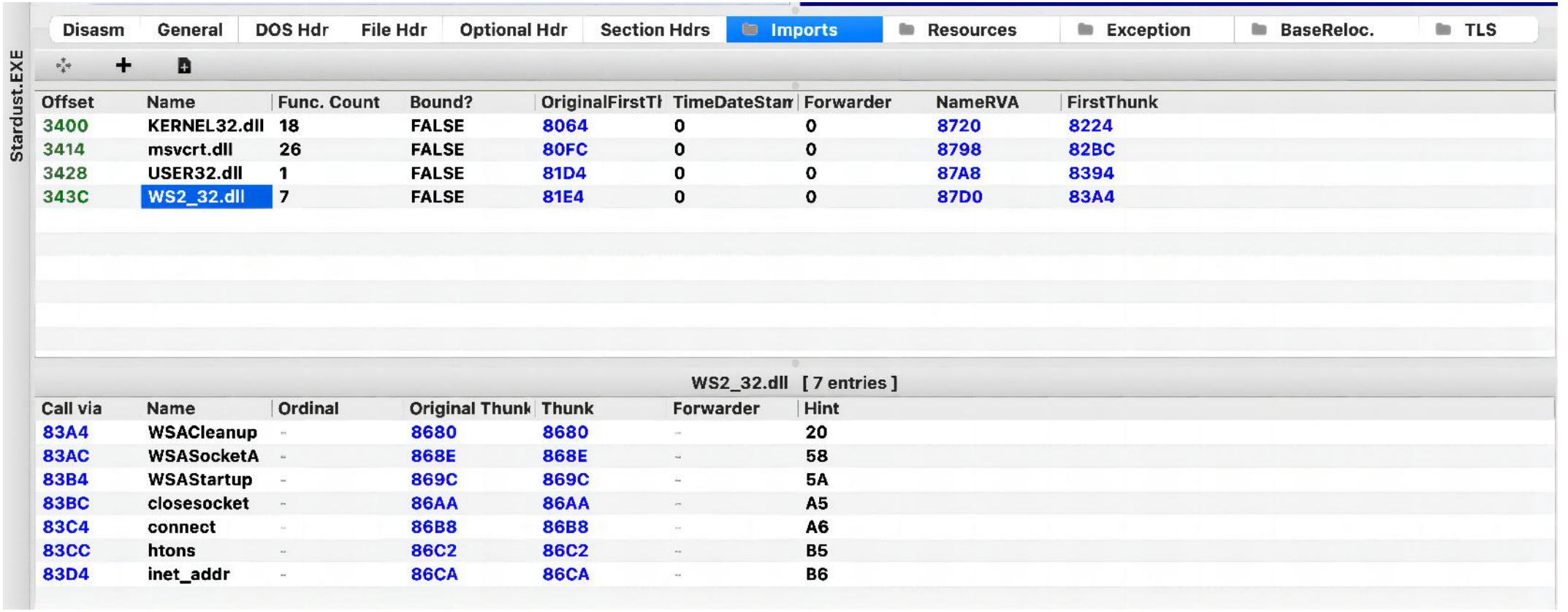
Skeleton Detection

- Skeleton detection is a technique that identifies the core functionality or "skeleton" of a malware sample through its static attributes, such as its code, file structure, signature generation.



Skeleton Detection-Code analysis

- It can help identify the main control flow and key functions used by the malware. This can include the identification of specific API calls or system functions used by the malware.

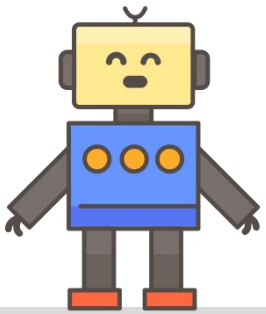


Nearly Exact Identification

- Nearly exact identification is a technique that identifies a specific variant or version of a malware sample. It involves comparing two or more versions of the same malware sample to identify differences between them.

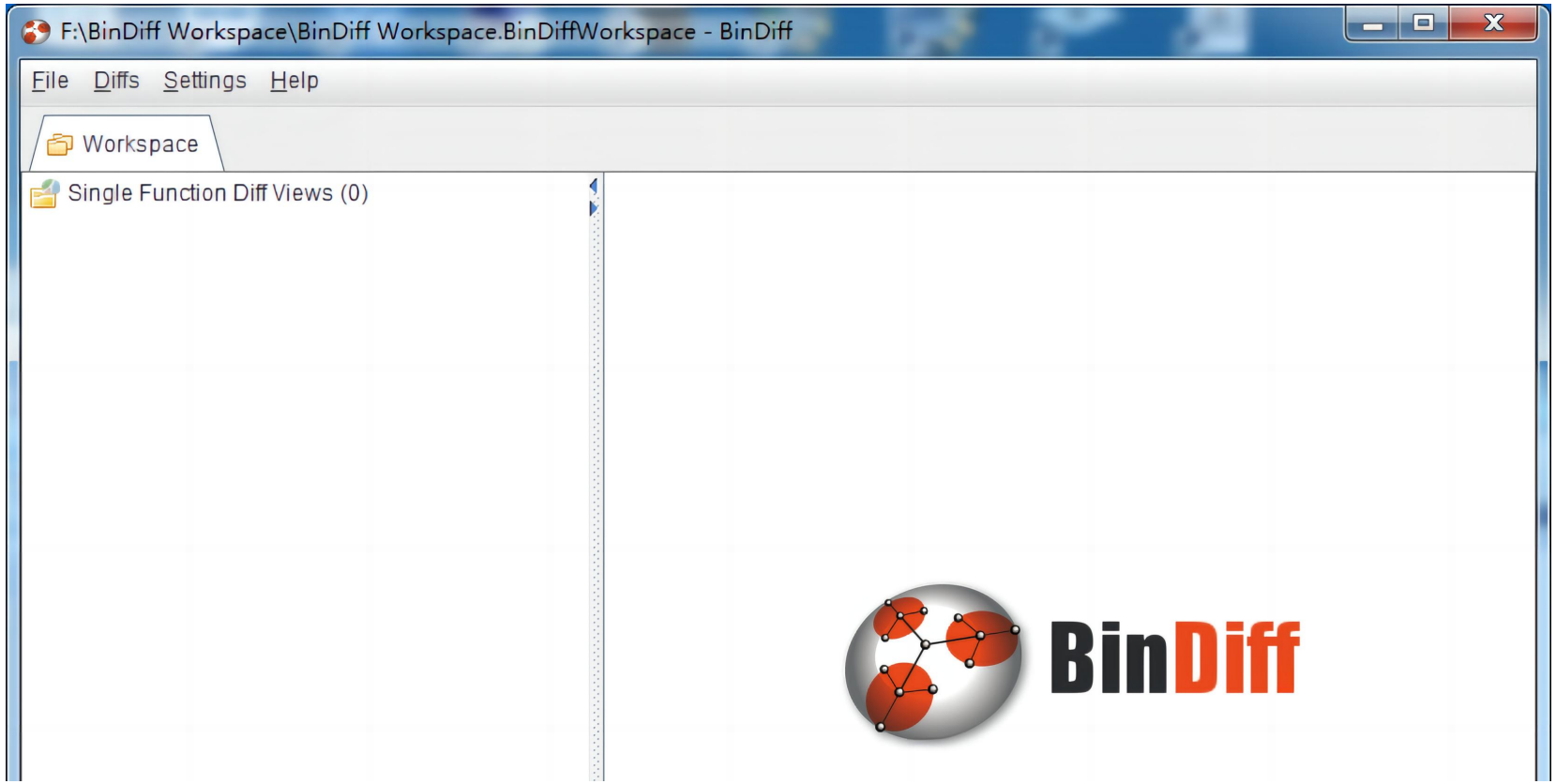
Nearly Exact Identification

- To perform nearly exact identification, analysts typically use techniques such as binary diffing, code comparison, or file comparison.

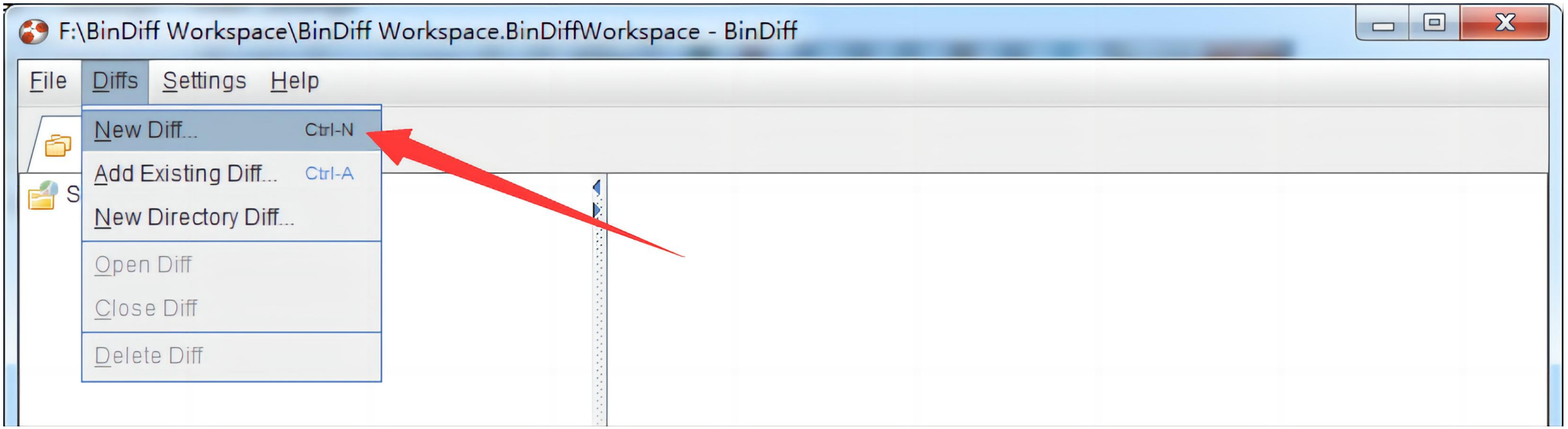


BinDiff

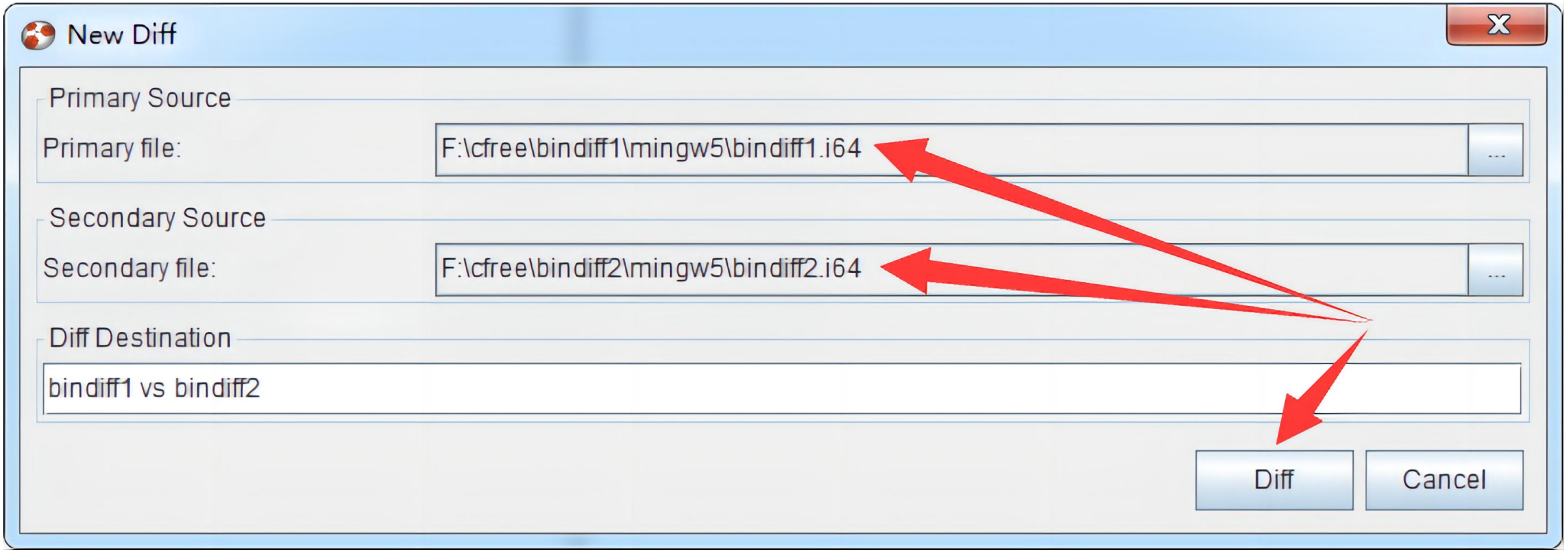
- BinDiff uses a unique graph-theoretical approach to compare executables by identifying identical and similar functions.



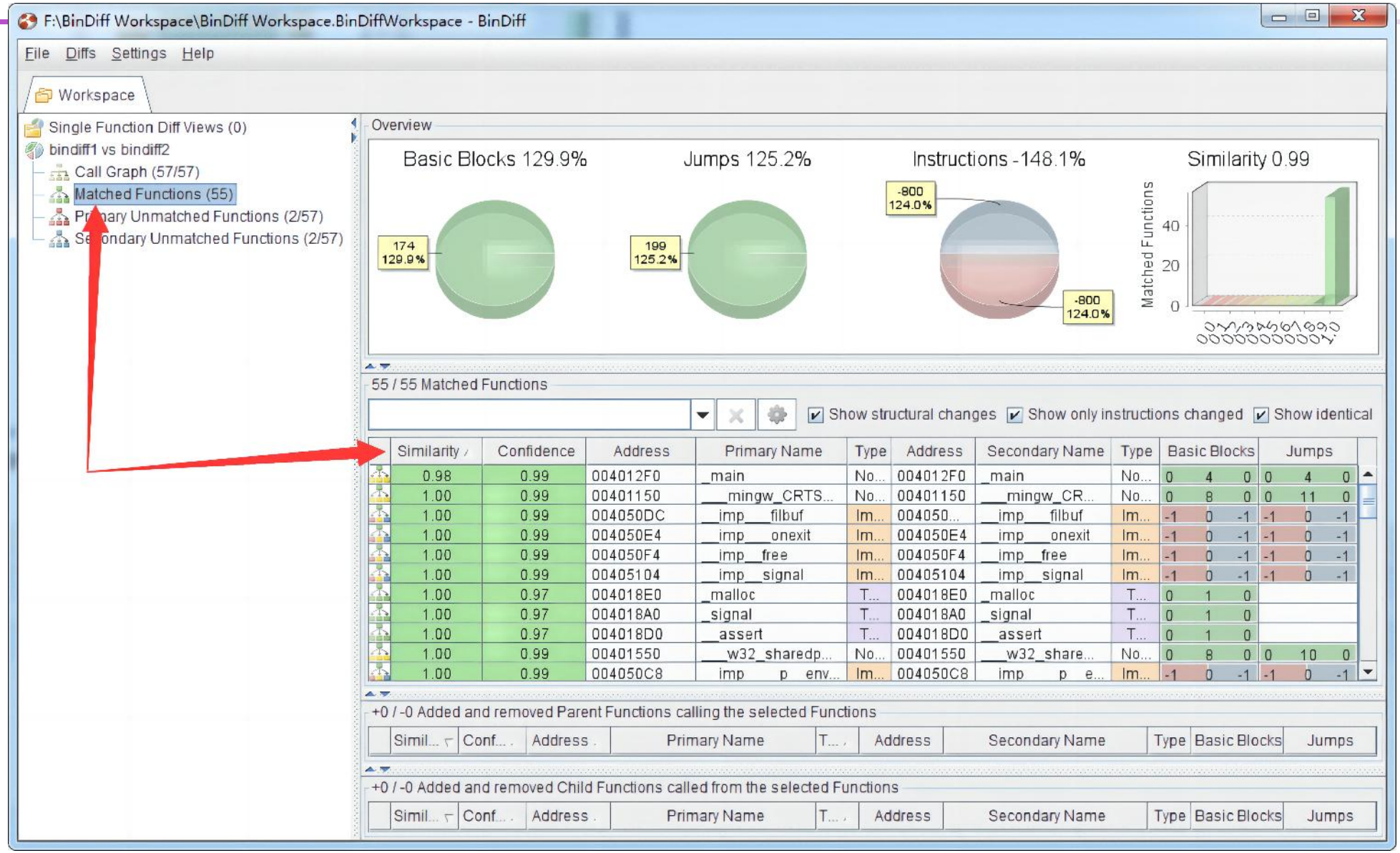
BinDiff


F:\BinDiff Workspace\BinDiff Workspace.BinDiffWorkspace - BinDiff
File
Diffs
Settings
Help
New Diff... Ctrl-N
Add Existing Diff... Ctrl-A
New Directory Diff...
Open Diff
Close Diff
Delete Diff

BinDiff


New Diff
Primary Source
Primary file:
F:\cfree\bindiff1\mingw5\bindiff1.i64
...
Secondary Source
Secondary file:
F:\cfree\bindiff2\mingw5\bindiff2.i64
...
Diff Destination
bindiff1 vs bindiff2
Diff
Cancel

BinDiff


F:\BinDiff Workspace\BinDiff Workspace.BinDiffWorkspace - BinDiff
File Diffs Settings Help
Workspace
Single Function Diff Views (0)
bindiff1 vs bindiff2
Call Graph (57/57)
Matched Functions (55)
Primary Unmatched Functions (2/57)
Secondary Unmatched Functions (2/57)
Overview
Basic Blocks 129.9%
Jumps 125.2%
Instructions -148.1%
Similarity 0.99
174 129.9%
199 125.2%
-800 124.0%
-800 124.0%
Matched Functions
55 / 55 Matched Functions
Show structural changes
Show only instructions changed
Show identical
Similarity
Confidence
Address
Primary Name
Type
Address
Secondary Name
Type
Basic Blocks
Jumps
0.98 0.99 004012F0 _main No... 004012F0 _main No... 0 4 0 0 4 0
1.00 0.99 00401150 ___mingw_CRTS... No... 00401150 ___mingw_CR... No... 0 8 0 0 11 0
1.00 0.99 004050DC __imp___filbuf Im... 004050... __imp___filbuf Im... -1 0 -1 -1 0 -1
1.00 0.99 004050E4 __imp___onexit Im... 004050E4 __imp___onexit Im... -1 0 -1 -1 0 -1
1.00 0.99 004050F4 __imp__free Im... 004050F4 __imp__free Im... -1 0 -1 -1 0 -1
1.00 0.99 00405104 __imp__signal Im... 00405104 __imp__signal Im... -1 0 -1 -1 0 -1
1.00 0.97 004018E0 _malloc T... 004018E0 _malloc T... 0 1 0
1.00 0.97 004018A0 _signal T... 004018A0 _signal T... 0 1 0
1.00 0.97 004018D0 __assert T... 004018D0 __assert T... 0 1 0
1.00 0.99 00401550 ___w32_sharedp... No... 00401550 ___w32_share... No... 0 8 0 0 10 0
1.00 0.99 004050C8 imp p env... Im... 004050C8 imp p e... Im... -1 0 -1 -1 0 -1
+0 / -0 Added and removed Parent Functions calling the selected Functions
Simil... Conf... Address Primary Name T... Address Secondary Name Type Basic Blocks Jumps
+0 / -0 Added and removed Child Functions called from the selected Functions
Simil... Conf... Address Primary Name T... Address Secondary Name Type Basic Blocks Jumps

BinDiff

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int get_num();
void print_bad();

int main() {
  int num;
  while(1) {
    num = get_num();
    if (num == 1337) {
      printf("Accepted!\n");
    }
    else {
      print_bad();
    }
  }
  return 0;
}

int get_num() {
  char buf[16];
  printf("Enter secret number : ");
  fgets(buf, 16, stdin);
  return atoi(buf);
}

void print_bad() {
  printf("Nope!\n");
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int get_num();
void print_good();
void print_bad();

int main() {
  int num;
  while(1) {
    num = get_num();
    if (num == 1337 || num == 7331 || ((num*3+3321)*2 + 7331)/3 == 7333) {
      print_good();
    }
    else {
      print_bad();
    }
  }
  return 0;
}
int get_num() {
  char buf[16];
  printf("Enter secret number : ");
  fgets(buf, 16, stdin);
  return atoi(buf);
}
void print_good() {
  printf("Accepted!\n");
}

void print_bad() {
  printf("Nope!\n");
}
```

BinDiff-function

```
proj = angr.Project('bindiff_a', auto_load_libs=False)
cfg = proj.analyses.CFGFast()
proj2 = angr.Project('bindiff_b', auto_load_libs=False)
cfg2 = proj2.analyses.CFGFast()
anaz = proj.analyses.BinDiff(proj2, cfg_a=cfg, cfg_b=cfg2)


for func_addrs in anaz.differing_functions:
    addr1 = func_addrs[0]
    addr2 = func_addrs[1]
    print(cfg.kb.functions[addr1].name, cfg2.kb.functions[addr2].name)

for func_pair in anaz.identical_functions:
    print(proj.kb.functions[func_pair[0]].name, proj2.kb.functions[func_pair[1]].name)
```

BinDiff-block

```
for blocks in anaz.differing_blocks:
    block1, block2 = blocks
    print(hex(block1.addr))
    print(hex(block2.addr))
    print(proj.factory.block(block1.addr).pp())
    print(proj2.factory.block(block2.addr).pp())
```

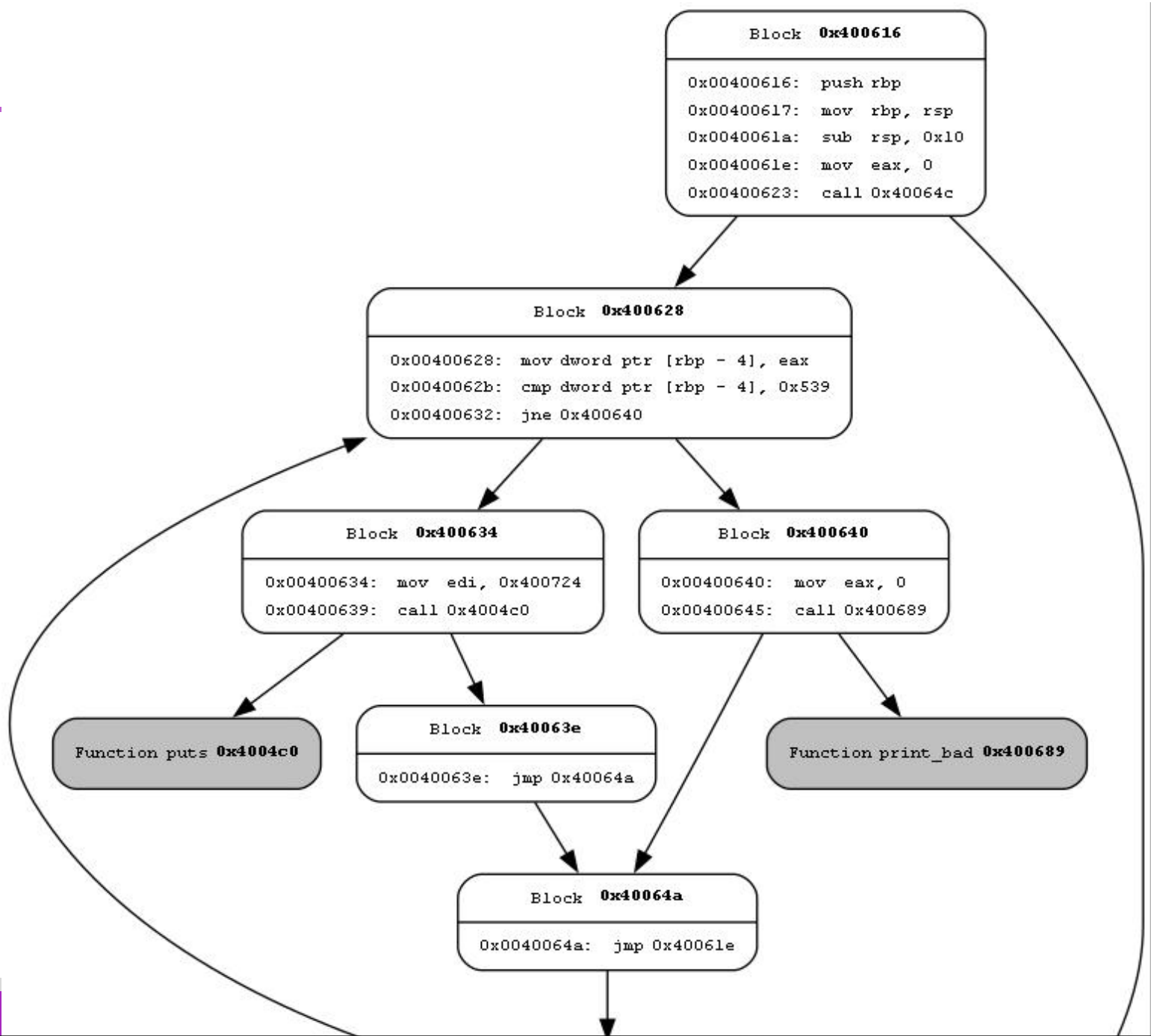
BinDiff-block

```
def analyze(b, name):
    for func in b.kb.functions.values():
        if func.name.find(name) == 0:
            plot_func_graph(b, func.transition_graph, "%s_%s_cfg" % (name, b.filename), asminst=True, vexinst=False)



analyze(proj, "main")
analyze(proj2, "main")
```

BinDiff


Block 0x400616
0x00400616: push rbp
0x00400617: mov rbp, rsp
0x0040061a: sub rsp, 0x10
0x0040061e: mov eax, 0
0x00400623: call 0x40064c
Block 0x400628
0x00400628: mov dword ptr [rbp - 4], eax
0x0040062b: cmp dword ptr [rbp - 4], 0x539
0x00400632: jne 0x400640
Block 0x400634
0x00400634: mov edi, 0x400724
0x00400639: call 0x4004c0
Block 0x400640
0x00400640: mov eax, 0
0x00400645: call 0x400689
Function puts 0x4004c0
Block 0x40063e
0x0040063e: jmp 0x40064a
Function print_bad 0x400689
Block 0x40064a
0x0040064a: jmp 0x40061e

BinDiff

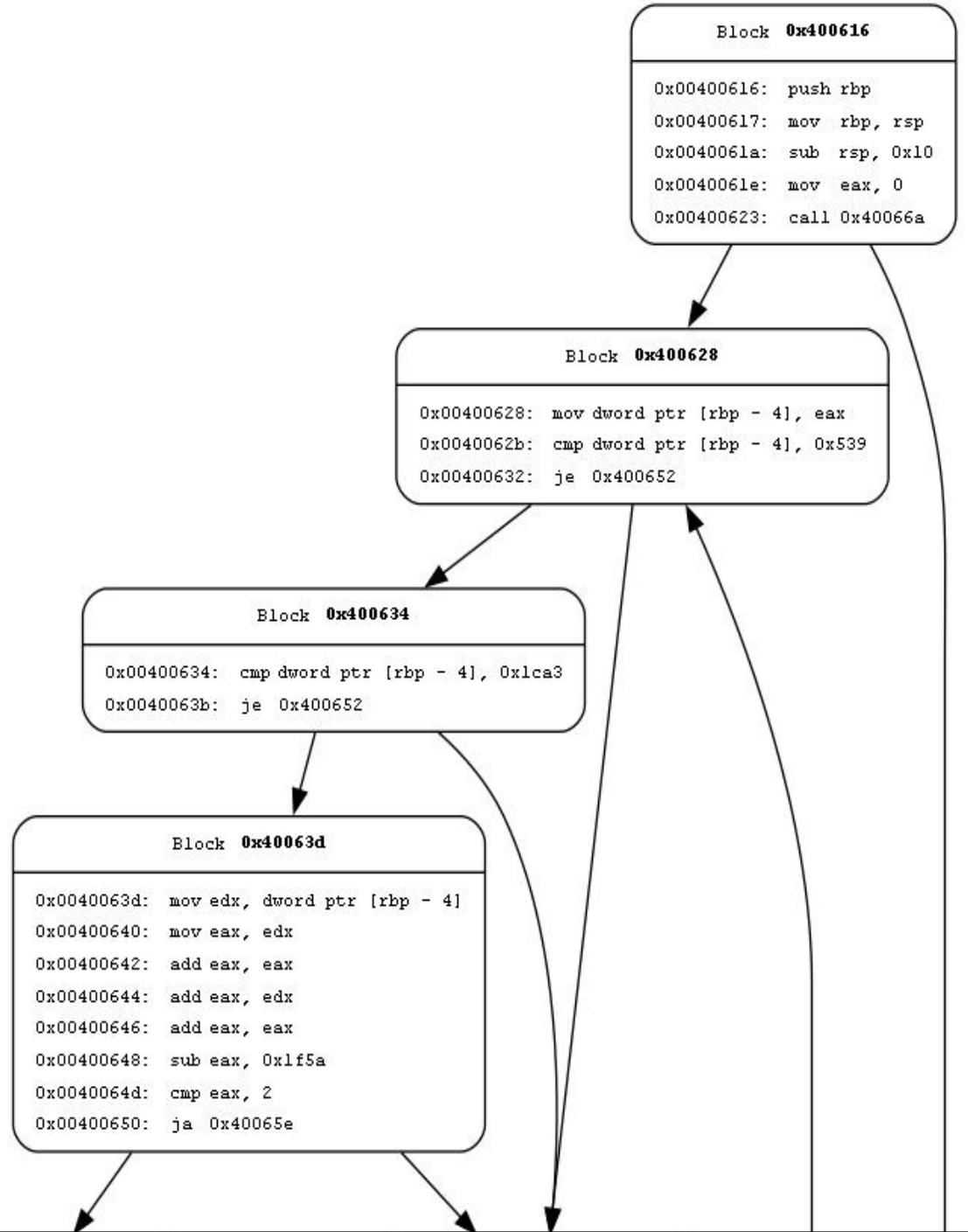

Block 0x400616
0x00400616: push rbp
0x00400617: mov rbp, rsp
0x0040061a: sub rsp, 0x10
0x0040061e: mov eax, 0
0x00400623: call 0x40066a
Block 0x400628
0x00400628: mov dword ptr [rbp - 4], eax
0x0040062b: cmp dword ptr [rbp - 4], 0x539
0x00400632: je 0x400652
Block 0x400634
0x00400634: cmp dword ptr [rbp - 4], 0x1ca3
0x0040063b: je 0x400652
Block 0x40063d
0x0040063d: mov edx, dword ptr [rbp - 4]
0x00400640: mov eax, edx
0x00400642: add eax, eax
0x00400644: add eax, edx
0x00400646: add eax, eax
0x00400648: sub eax, 0x1f5a
0x0040064d: cmp eax, 2
0x00400650: ja 0x40065e

Image Based Method

- Image-based malware detection is a technique used to identify malware as an image file. This technique involves analyzing the contents of the image file to detect any embedded malicious code or hidden payloads.

Malware Samples to Images

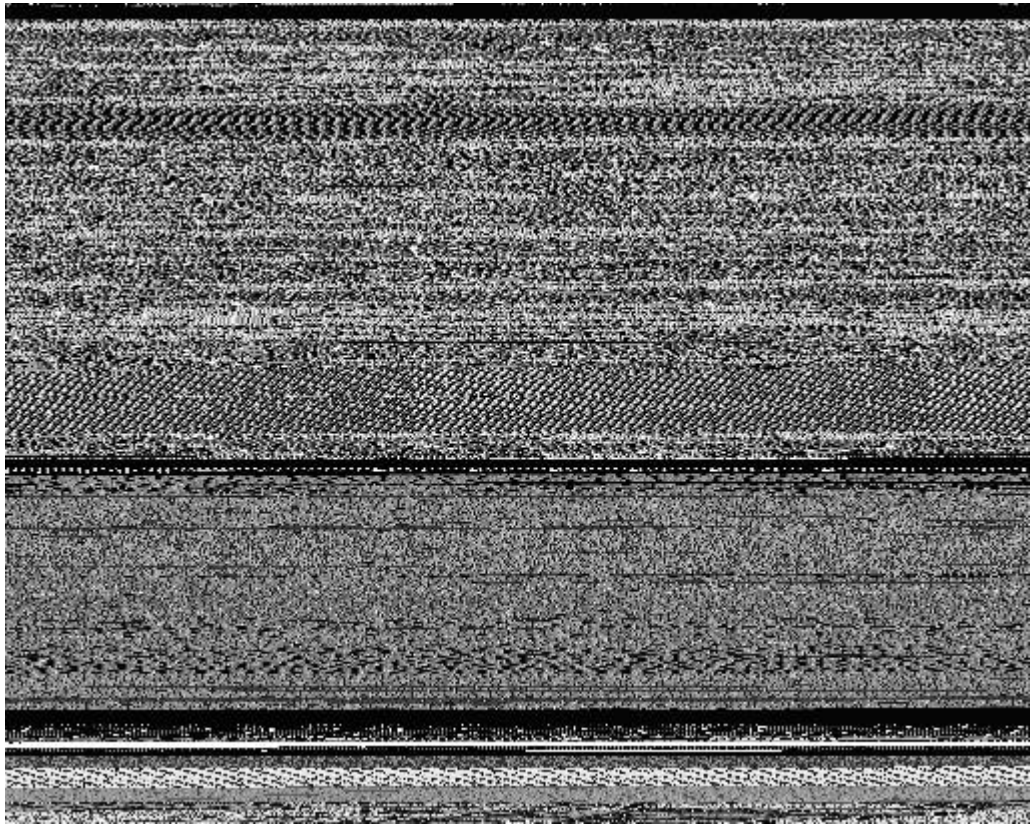
Correspondence between the malware sample file size and the converted imaged width

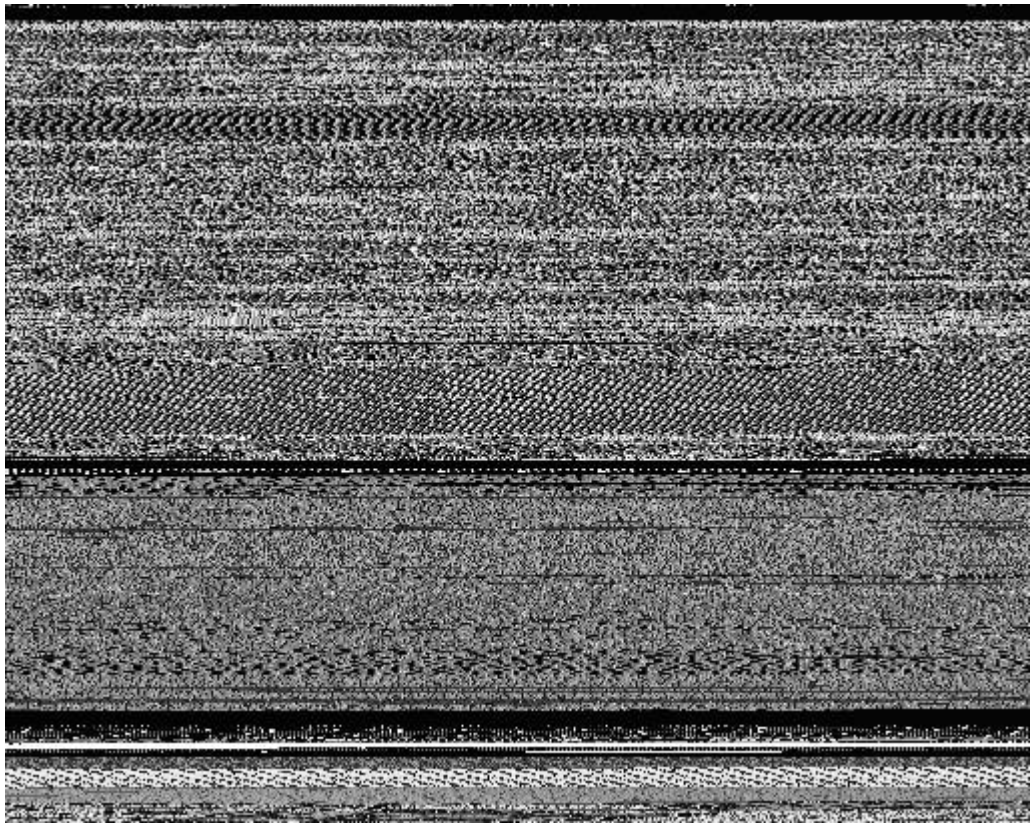
File Size	Width	Height
≤10KB	32	(0, 312]
10KB-30KB	64	(156, 468]
30KB-60KB	128	(234, 468]
60KB-100KB	256	(234, 390]
100KB-200KB	384	(260, 520]
200KB-500KB	512	(390, 976]
500KB-1000KB	768	(651, 1302]
≥1000KB	1024	(976,∞)

1. Sequentially reads the binary data in bytes and converts each byte into a number ranging in [0-255]

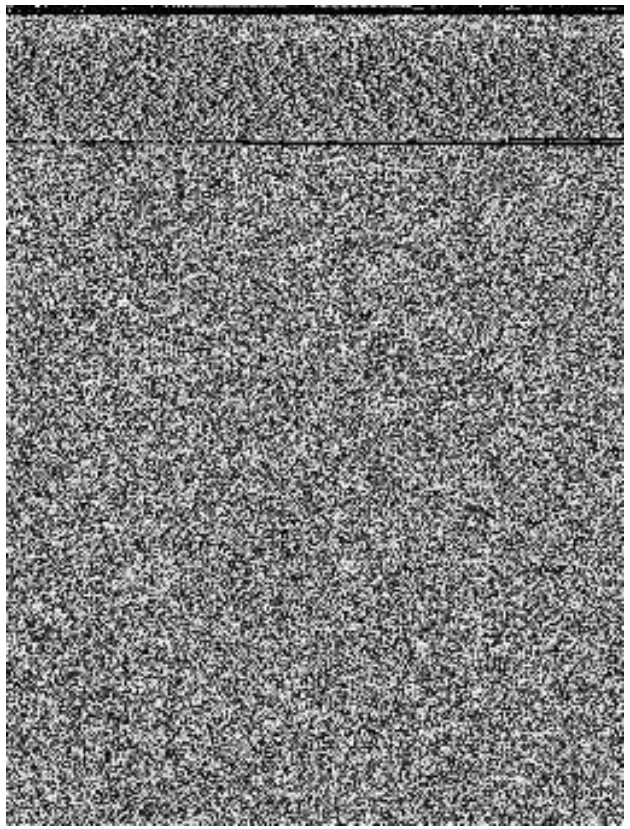
2. Reshape the image data by following a recommended fixed width with a variable height

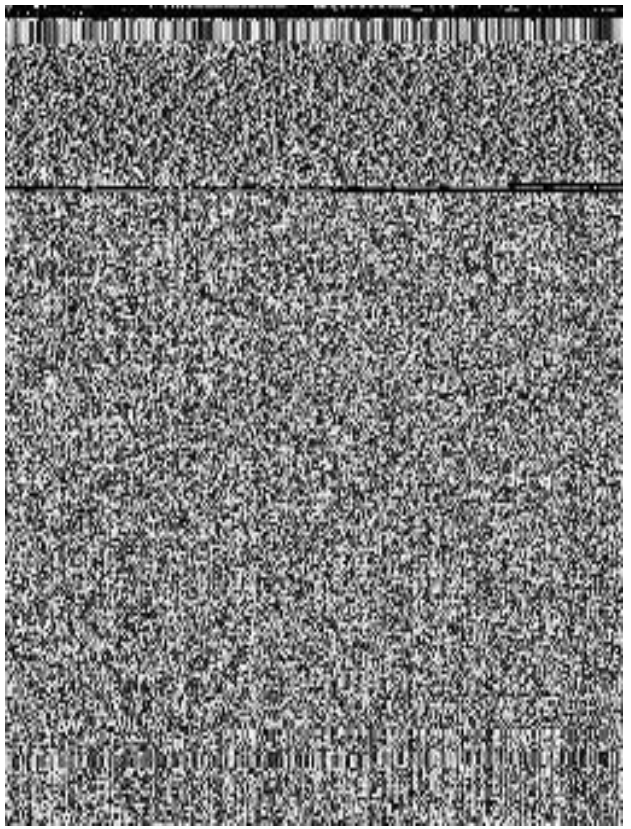
AdialerC7c2-AdialerC948





AllapleA43ad-AllapleAf4a





THE END

Fangtian Zhong

CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

11/18/2025