Bin-Carver: Automatic Recovery of Binary Executable Files

By

Scott Hand, Zhiqiang Lin, Guofei Gu and Bhavani Thuraisingham

Presented At

The Digital Forensic Research Conference

DFRWS 2012 USA Washington, DC (Aug 6th - 8th)

DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment. As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.

http:/dfrws.org

Bin-Carver

Automatic Recovery of Binary Executable Files

Scott Hand[†], Zhiqiang Lin[†],
Guofei Gu*, Bhavani Thuraisingham[†]

[†]Department of Computer Science, The University of Texas at Dallas
*Deparment of Computer Science and Engineering, Texas A&M University

2012/08/08

Outline

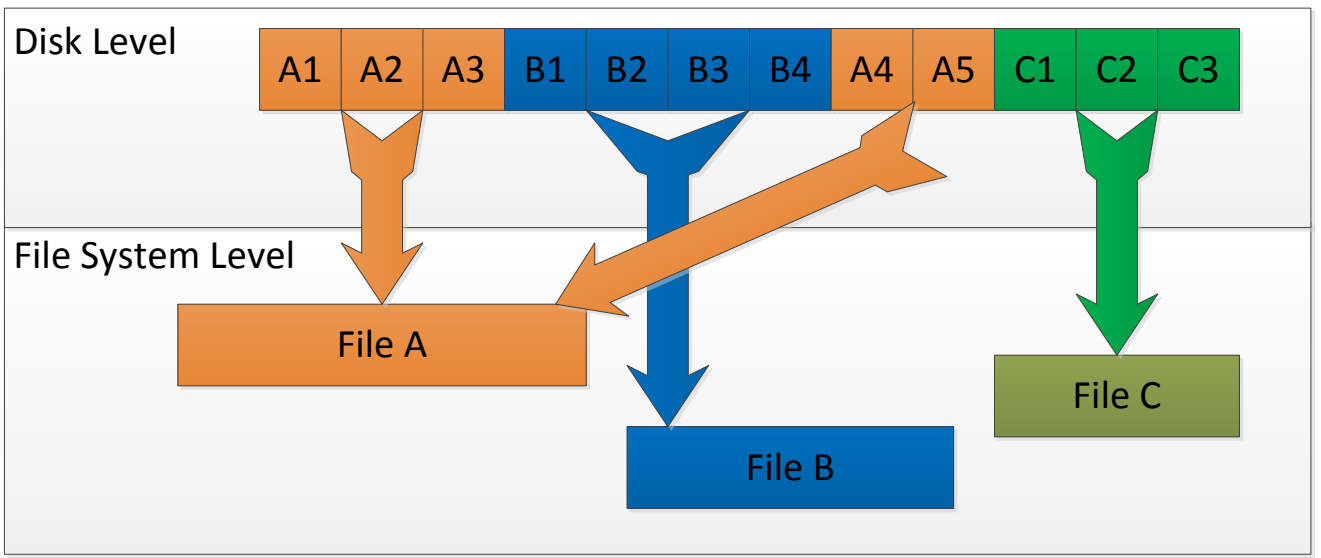
Outline

Outline

What are we trying to accomplish?

Basic Idea

Recover meaningful data (files) from unorganized data (data from disk)

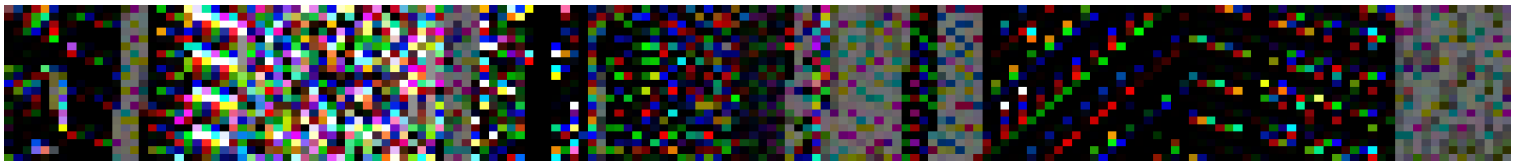


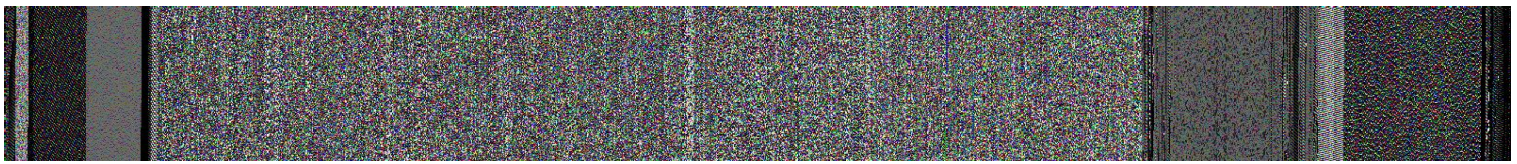
Why do we care?

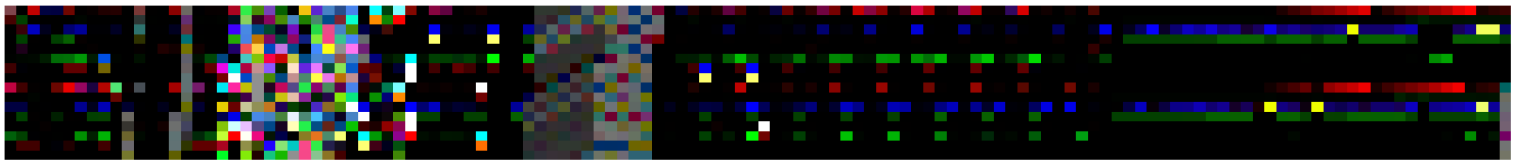
Needed any time file system metadata is not present

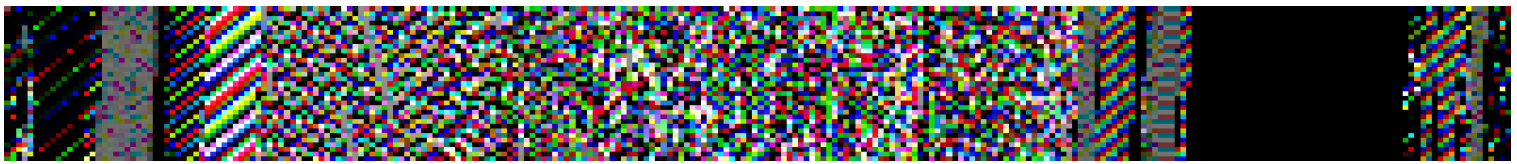
- Deletion
- Corruption
- Not part of file system (VM, embedded in other files, etc.)

Why are binaries special?







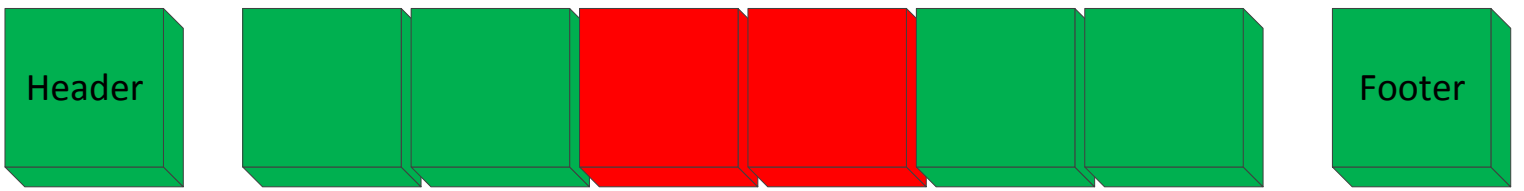


Motivation for focusing on binary executables

- Difficult to carve
  - Heterogeneous contents
  - No explicit footers
- Lots of internal structure
- They're everywhere
- Malware loves to hide

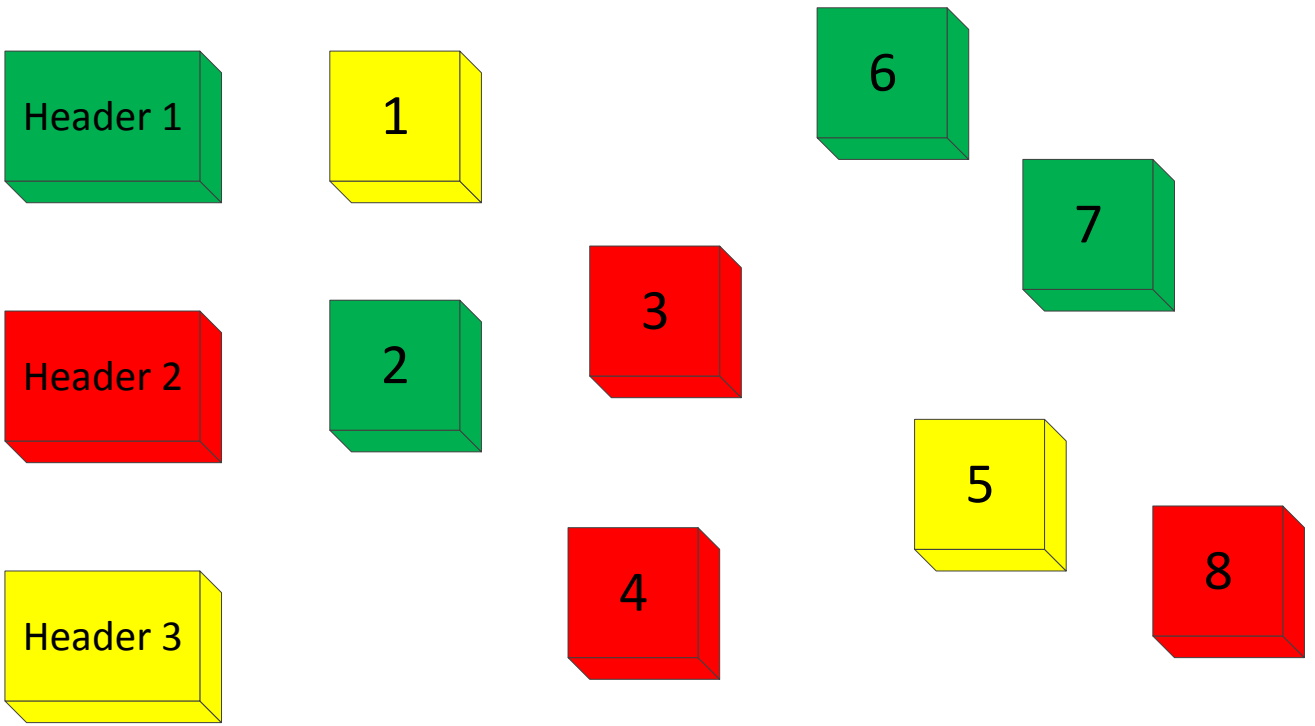
Previous Approaches - Bifragment Carving

Simson Garfinkel - *Carving Contiguous and Fragmented Files with Fast Object Validation*
DFRWS'07

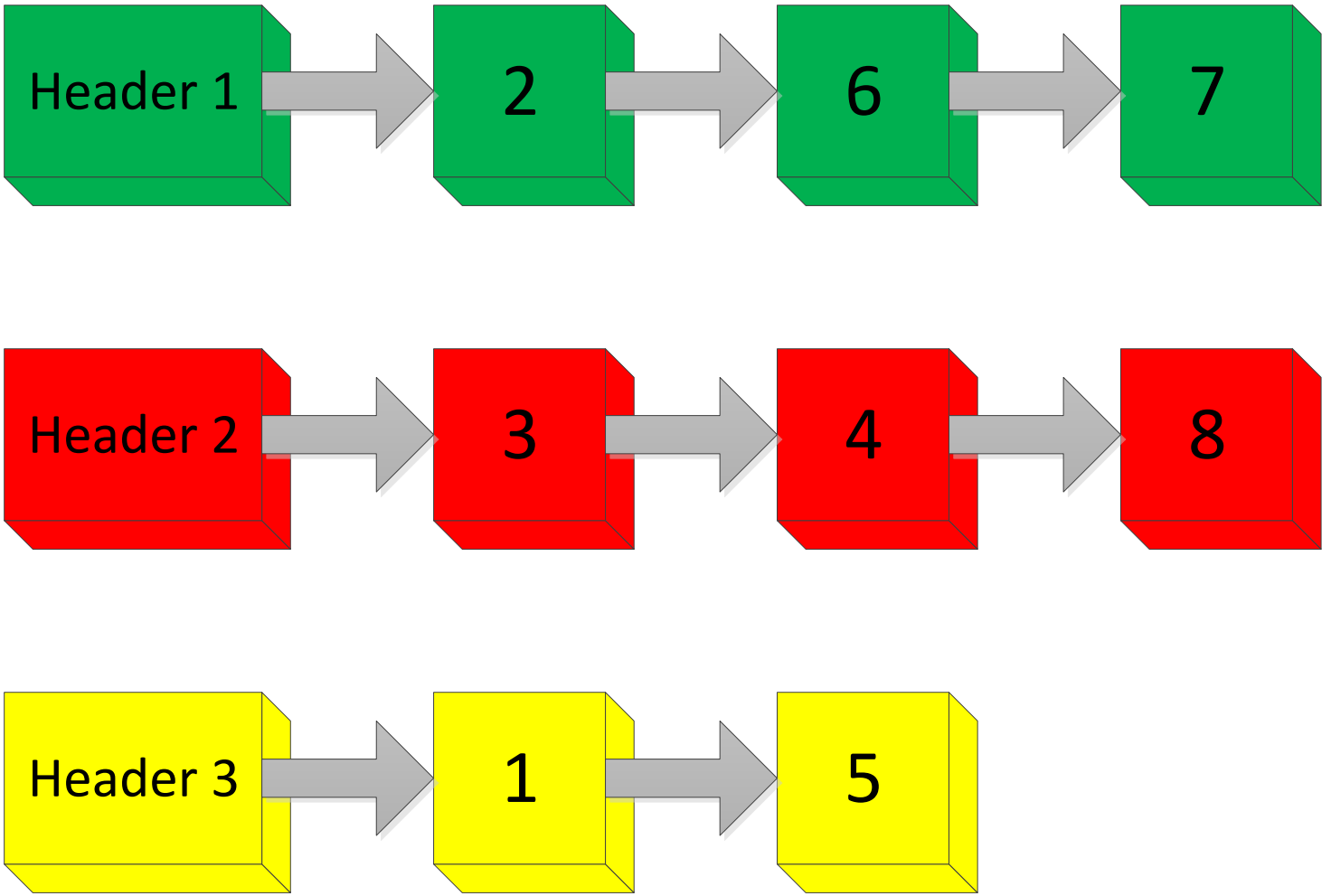


Previous Approaches - Shortest Path

Pal, A. and Shanmugasundaram, K. and Memon, N. *Automated reassembly of fragmented images using greedy algorithms* IEEE Transactions on Image Processing 2006



Previous Approaches - Shortest Path



Previous Approaches - Sequential Hypothesis Testing

Pal, A. and Sencar, H. and Memon, N. - *Detecting file fragmentation point using sequential hypothesis testing* DFRWS'08

Iteratively build up sequences of blocks using statistical hypothesis testing

Common elements

- Fragment edge identification
- Needs edge location heuristics
- Need both header and footer

Outline

Outline

Start with the magic number and expand

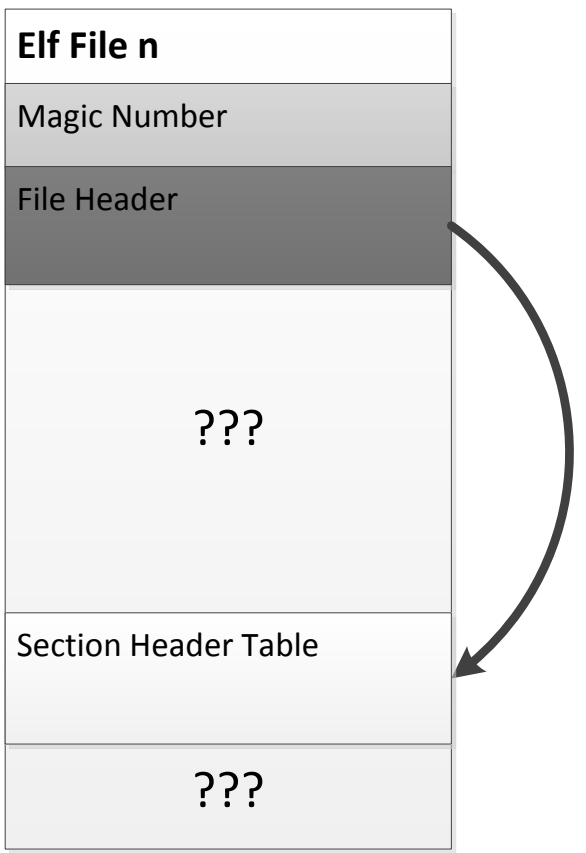
Build a list of ELF file headers by searching for ELF file magic numbers (0x7f,0x45,0x4c,0x46)

Load the ELF header

Luckily the ELF header will always be on the same block as the magic number

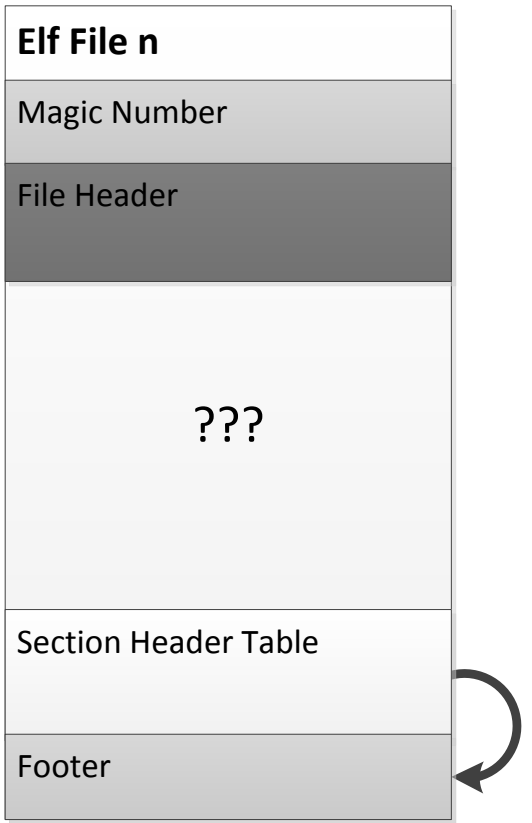
Find the section header table

The header will have a pointer to the section header table (SHT).



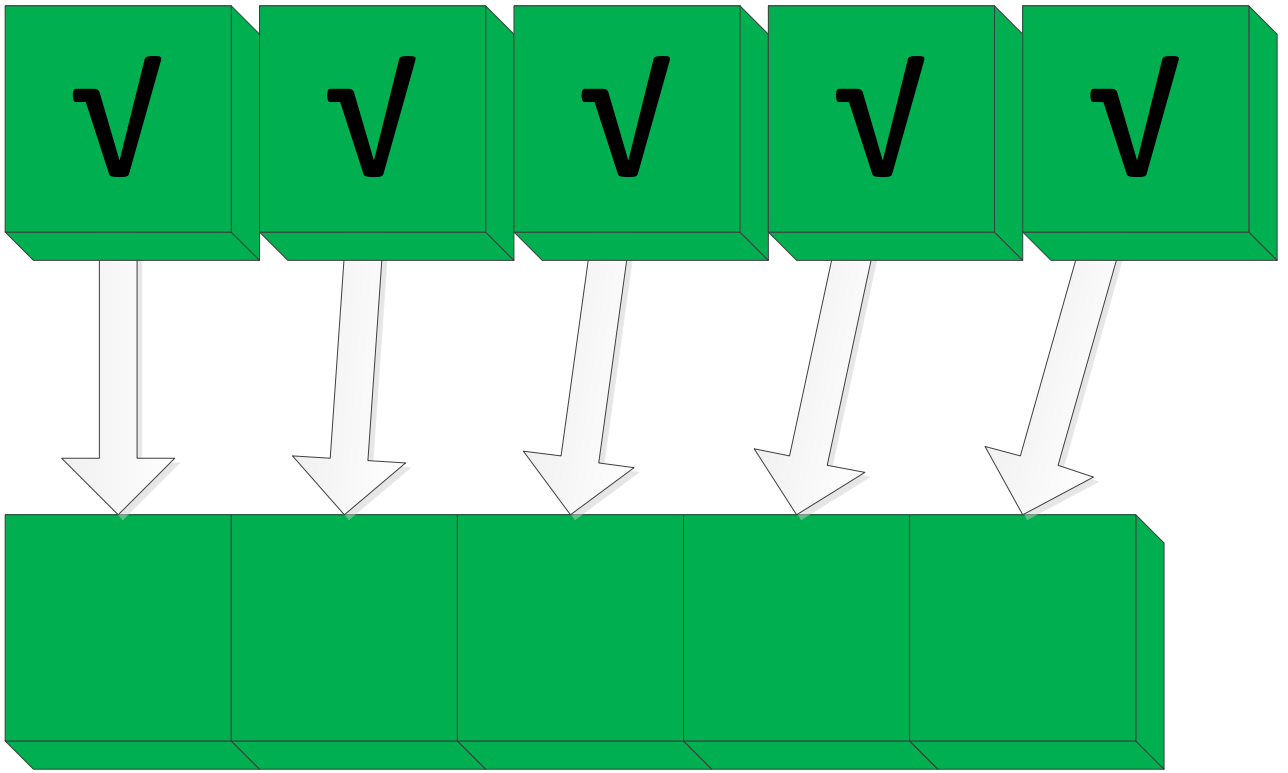
Identify the "footer"

The last part of the ELF file will either be the last section or the SHT. This can be easily checked, the footer identified, and the file size inferred.



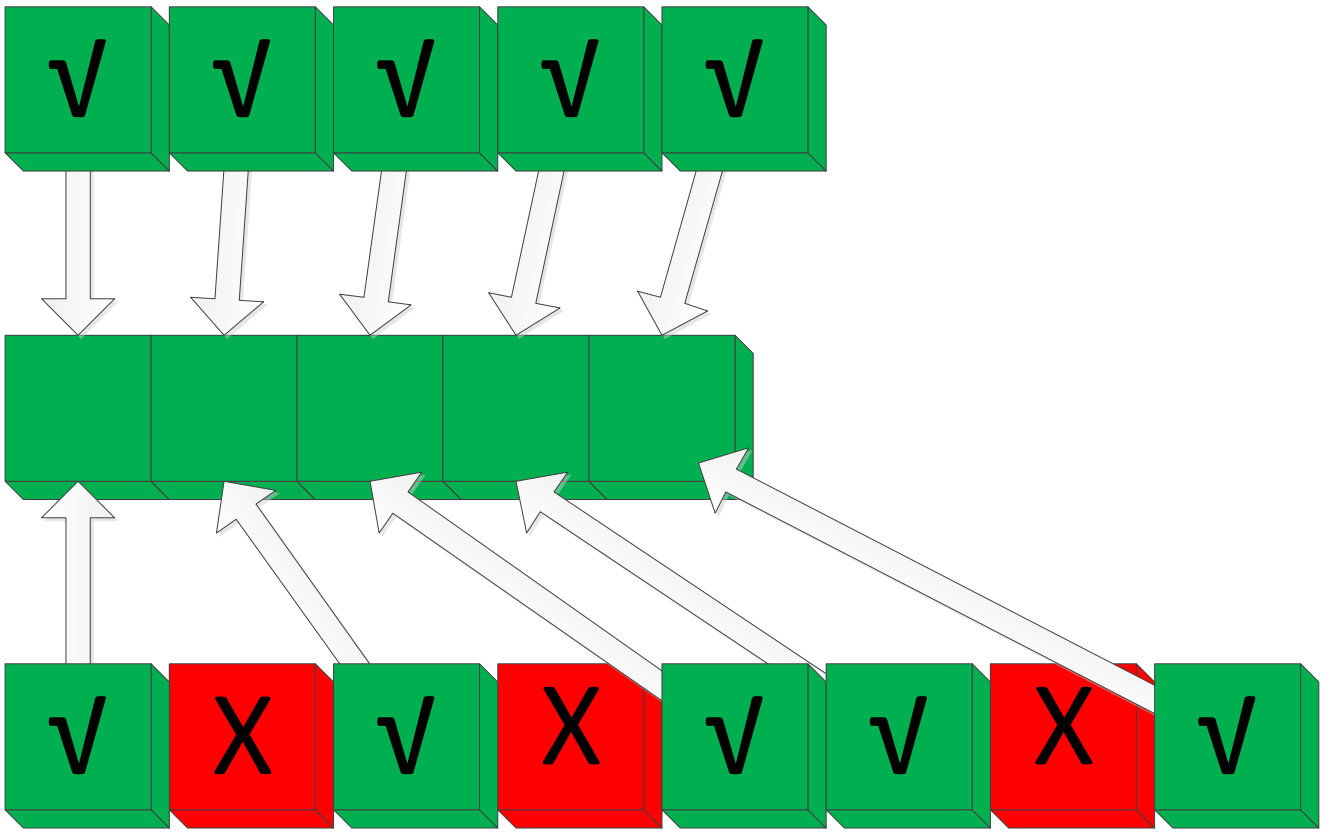
We're done!

Write everything from beginning to end



Uh oh!

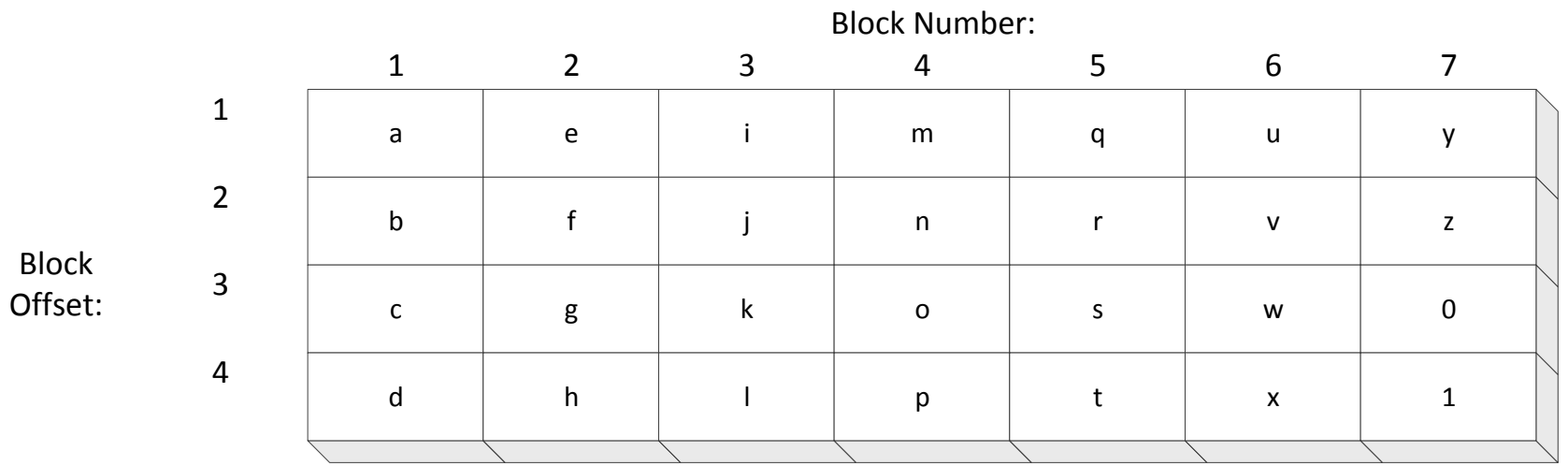
Disaster strikes



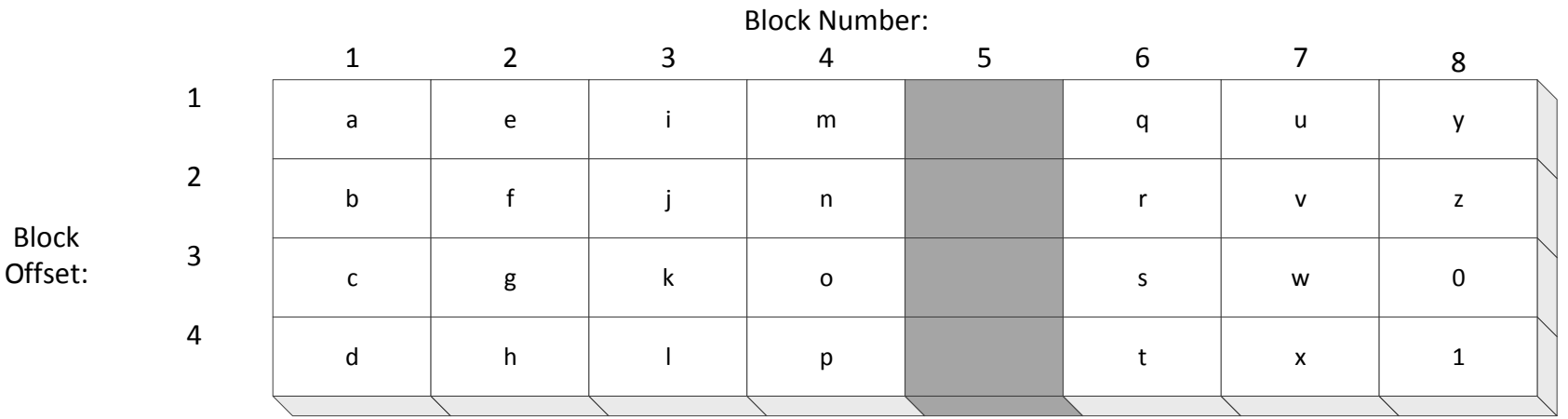
Outline

Outline

Pointers Before Fragmentation



Pointers After Fragmentation



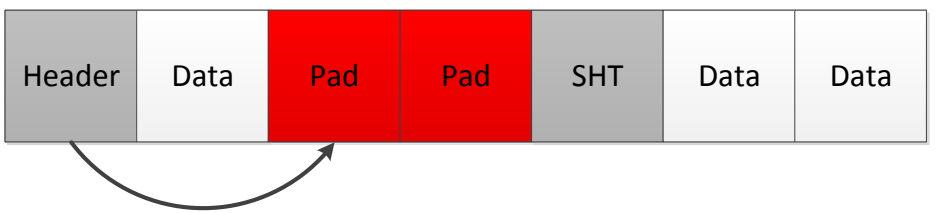
Finding the SHT

Without fragmentation:

Finding the SHT

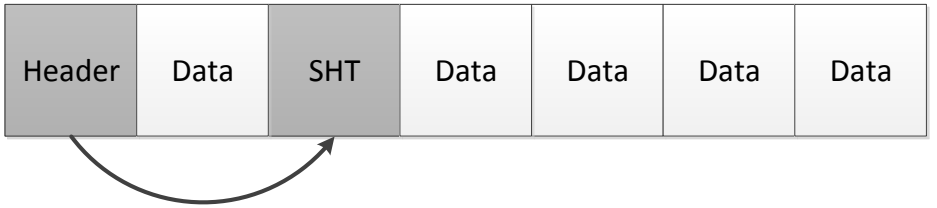
Without fragmentation:

With fragmentation:

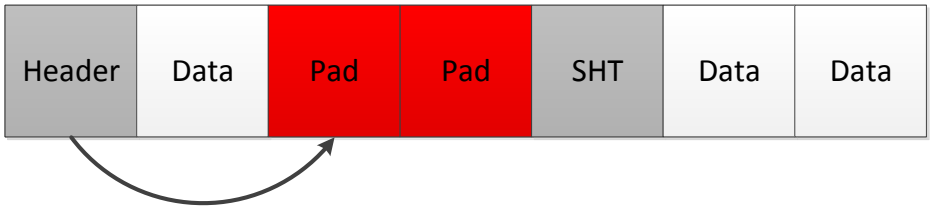


Finding the SHT

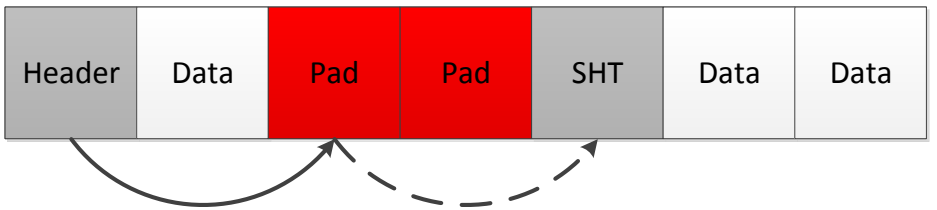
Without fragmentation:



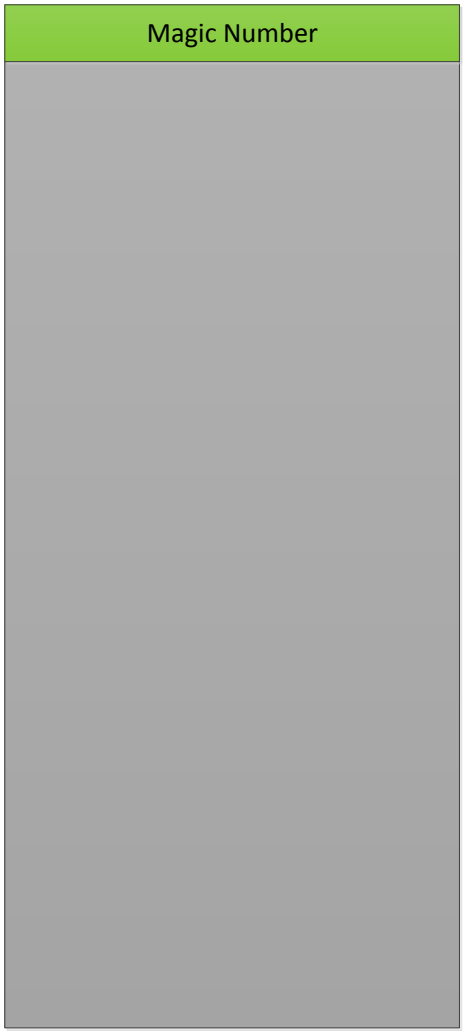
With fragmentation:



After moving forward twice, we find the SHT:

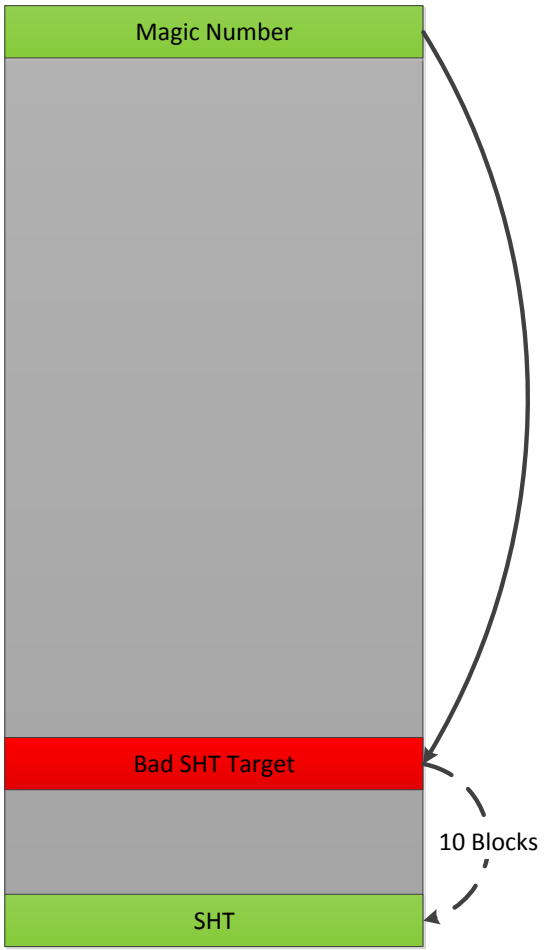


Step-by-step Example



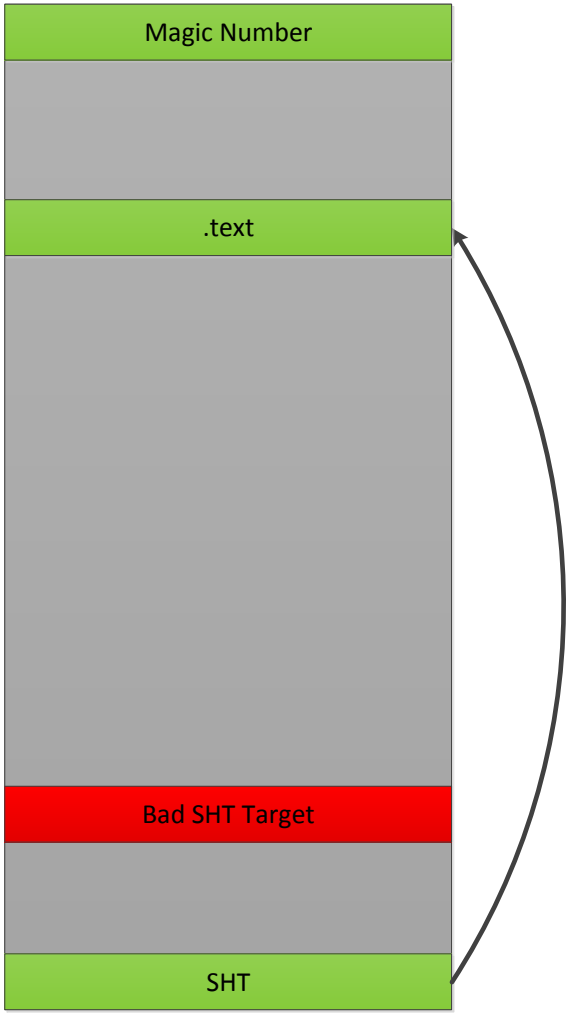
Step-by-step Example

There is fragmentation. SHT is 10 blocks away from its expected location.



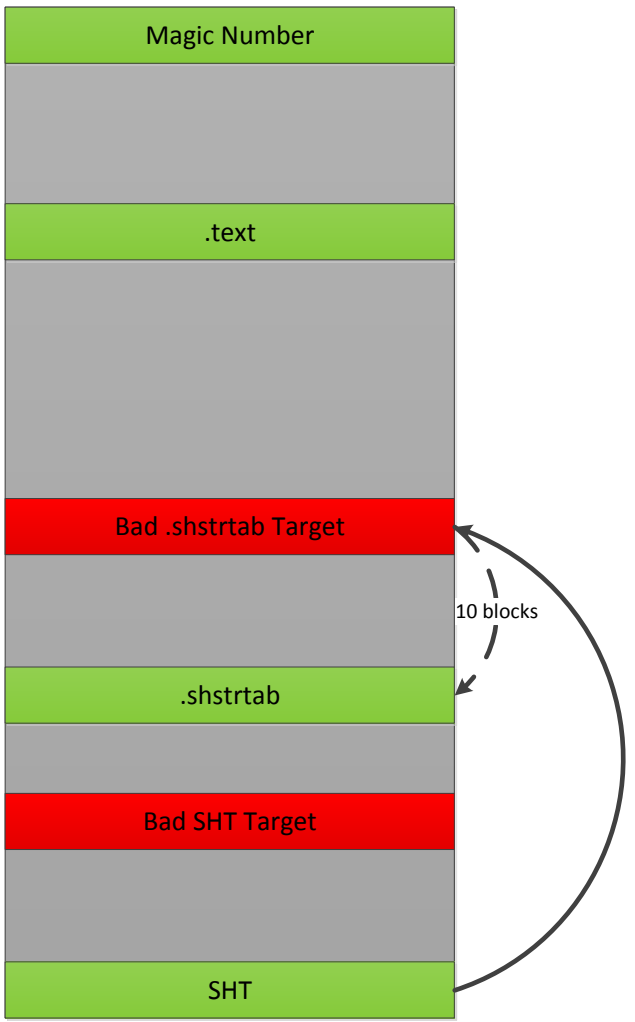
Step-by-step Example

The .text section appears to be in the right place.



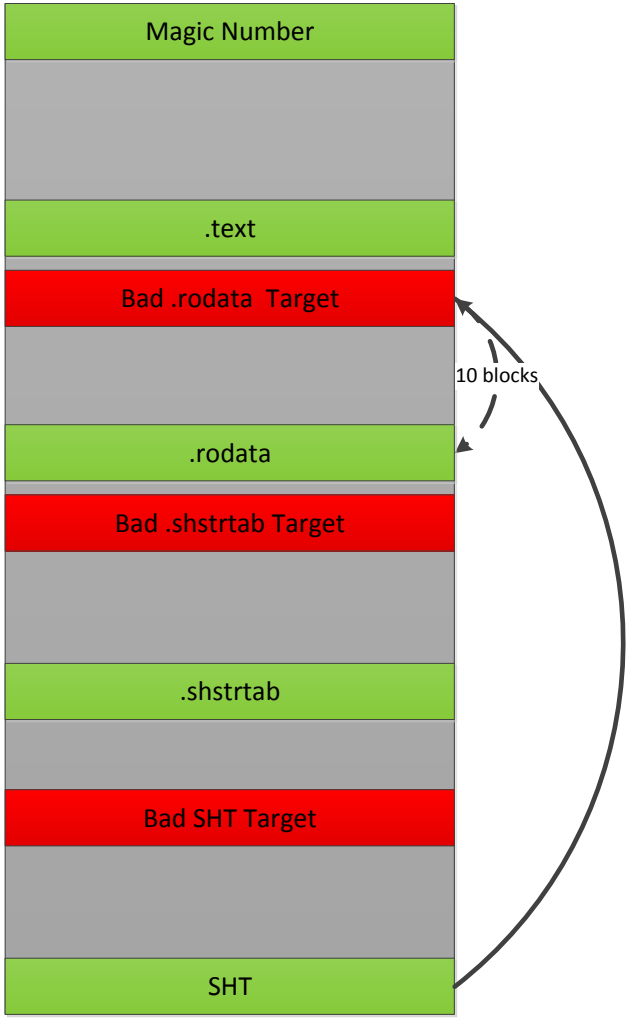
Step-by-step Example

The .shstrtab section is offset by 10 blocks as well.



Step-by-step Example

So is .rodata. .text looks pretty guilty.



Outline

The next step

What next?

Finding fragmentation in the ELF file now becomes finding fragmentation within sections

Targeting .text

Let's focus on .text, as it comprises a large part of the ELF file

Strategy for validating machine code blocks

Taking advantage of internal structure

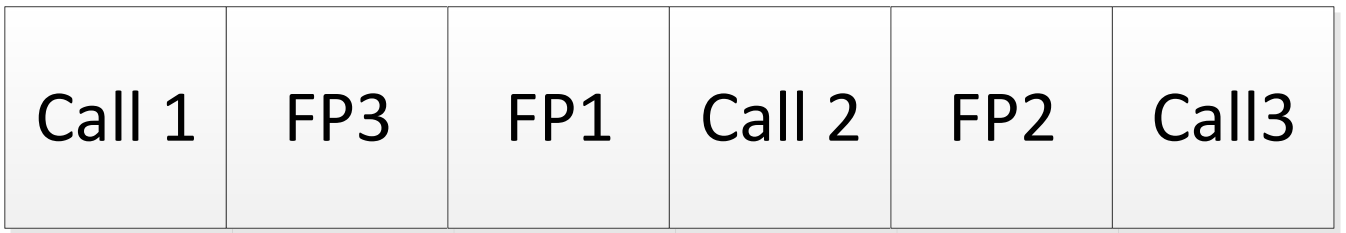
- Explore the structure provided by pointers in the code
- Map a CALL instruction to a function prologue at its target to validate a pair of locations

Code

```
8049480 <_init>:
8049480:   55                      push   %ebp
8049481:   89 e5                   mov    %esp,%ebp
8049483:   53                      push   %ebx
...
804949d:   e8 de 00 00 00          call   8049580 <__gmon_start__@plt>
...
80494b0 <abort@plt-0x10>:
80494b0:   ff 35 08 e1 05 08       pushl  0x805e108
80494b6:   ff 25 0c e1 05 08       jmp    *0x805e10c
80494bc:   00 00                   add    %al,(%eax)
...
8049580 <__gmon_start__@plt>:
8049580:   ff 25 40 e1 05 08       jmp    *0x805e140
8049586:   68 60 00 00 00          push   $0x60
804958b:   e9 20 ff ff ff          jmp    80494b0 <_init+0x30>
...
8059e84:   e8 f7 f5 fe ff          call   8049480 <_init>
...
```

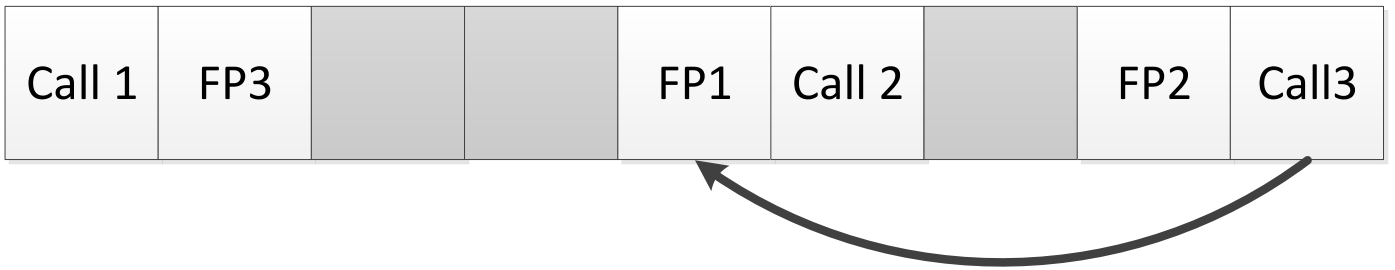
Example

A quick example shows this algorithm handling three calls to three different blocks.
Before fragmentation:



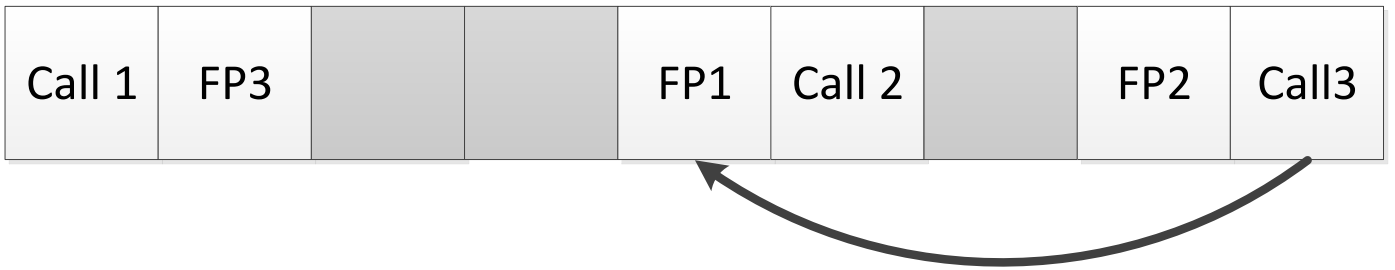
Example - Call 3

Call 3 previously pointed four blocks back to FP3, now it is invalid.

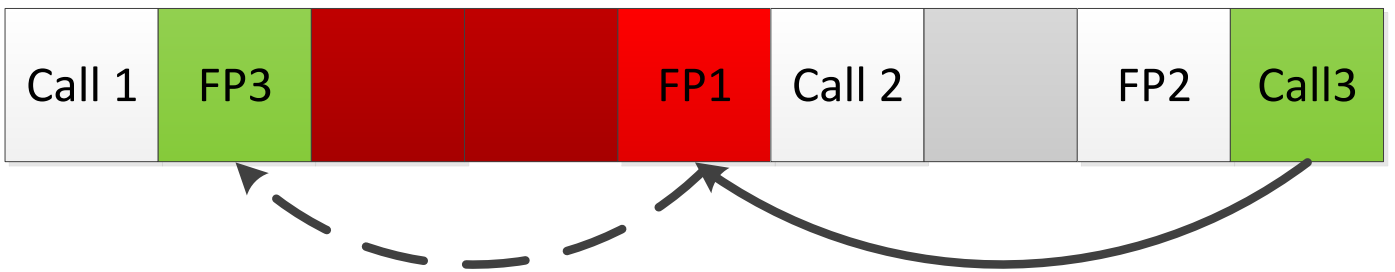


Example - Call 3

Call 3 previously pointed four blocks back to FP3, now it is invalid.



We look backward to find FP3:



Example - Call 2

Call 2 previously pointed one block forward to FP2, now it is invalid.

Example - Call 2

Call 2 previously pointed one block forward to FP2, now it is invalid.

We look forward to find FP2:

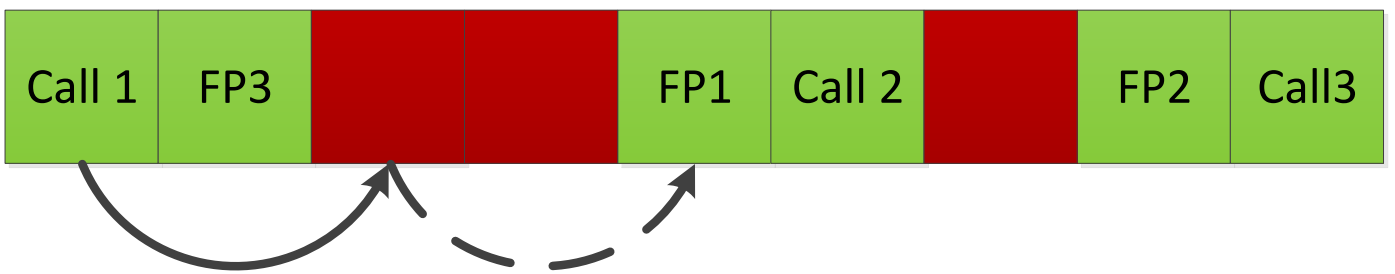
Example - Call 1

Call 1 previously pointed two blocks forward to FP1, now it is invalid.

Example - Call 1

Call 1 previously pointed two blocks forward to FP1, now it is invalid.

We look forward to find FP1:



Other Sections

Other important sections need recovery approaches as well, but many of them (rodata, debug sections, etc.) have predictable structures that lend themselves to data classification approaches.

Outline

Outline

Setup

Bin-Carver

- Prototype was coded in C#
- Python used for collection of accuracy statistics

Test Data

- Tested on 8 different disk images
- Each differed in the number of files as well as the number of deletes and copies executed after its creation

Disks

1. Disk 1 was a small baseline sample, only contained `/bin`
2. Disk 2 contained a larger number of ELF files
3. Disk 3 contained some of the files from disk 2, with some of them deleted before the image was made
4. Disk 4 contained all of disk 2 as well as SO ELF files from `/lib`

Disks

1. Disk 5 had all the files from disk 4 which were then deleted. Half were then picked randomly and copied back.
2. Disk 6 is the same as disk 5 except that only half were deleted
3. Disk 7 repeated the same process as 6, but twice with smaller batches
4. Disk 8 did lots of unpredictable small copy and delete cycles to create the most chaotic image

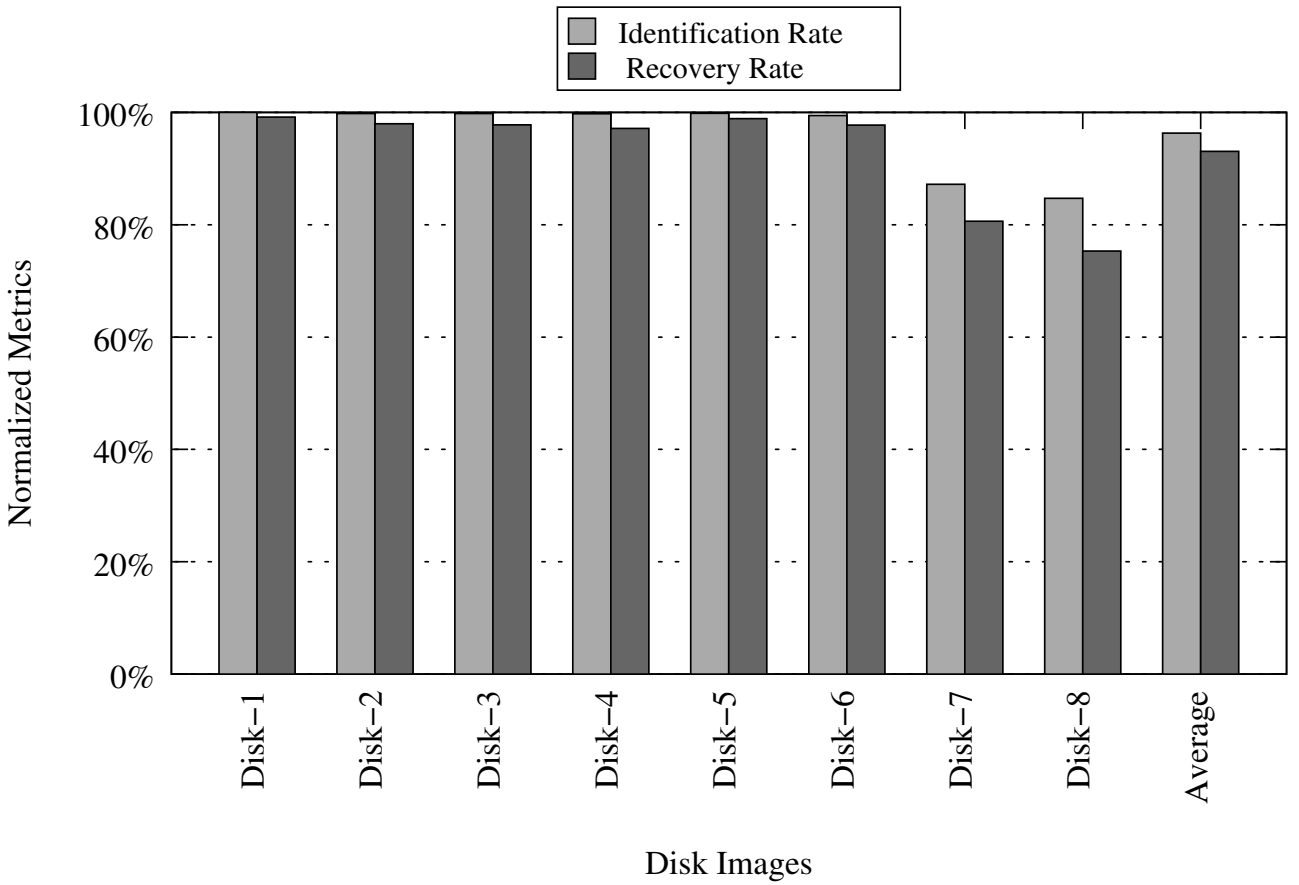
Evaluating accuracy

Effectiveness

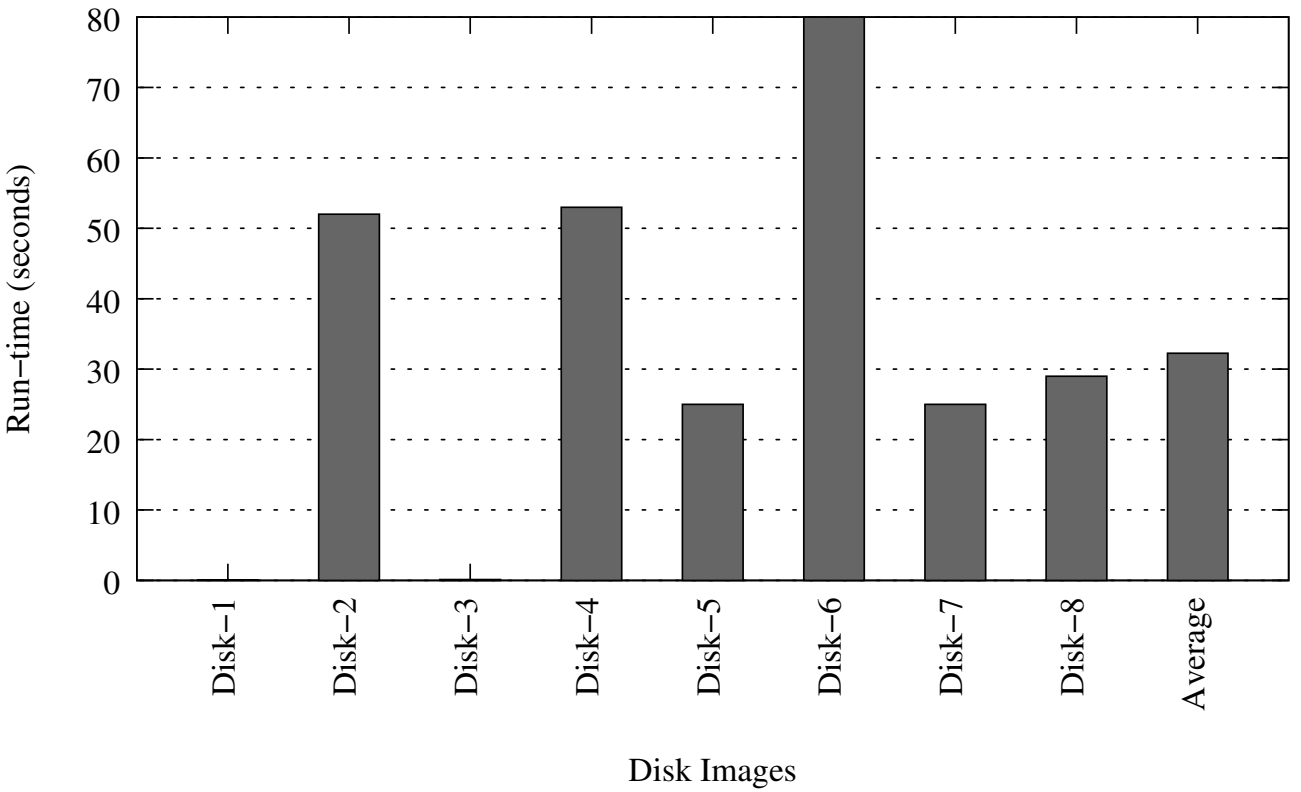
- Identification Rate - number of valid files on the disk we can identify
- Recovery Rate - number of files that were recovered successfully after identification

Outline

Accuracy Metrics



Performance Metrics



Outline

Conclusion

Remarks

- Recovery approaches were shown to be effective
- Hopefully, more research will be done in executable file carving
- Exclusionary carving could benefit other kinds of file carving

Conclusion

Remarks

- Recovery approaches were shown to be effective
- Hopefully, more research will be done in executable file carving
- Exclusionary carving could benefit other kinds of file carving

Limitations and Future Work

- PE Files
- More signatures
- Robustness

Thanks for coming

Any questions?