Malware Attribution Using the Rich Header

•••

Seamus Burke Kevin Bilzer RJ Joyce

About Us

www.linkedin.com/in/seamus-h-burke/

www.linkedin.com/in/kevin-bilzer-jr/

www.linkedin.com/in/rj-joyce/

Agenda

1. PE File Metadata
2. The Rich Header
3. Prior Research
4. Packers
5. Metadata Hashing
6. RichPE Metadata Hash
7. Rich Header Tamper Detection
8. Conclusion

PE File Metadata

What is the PE File Format?

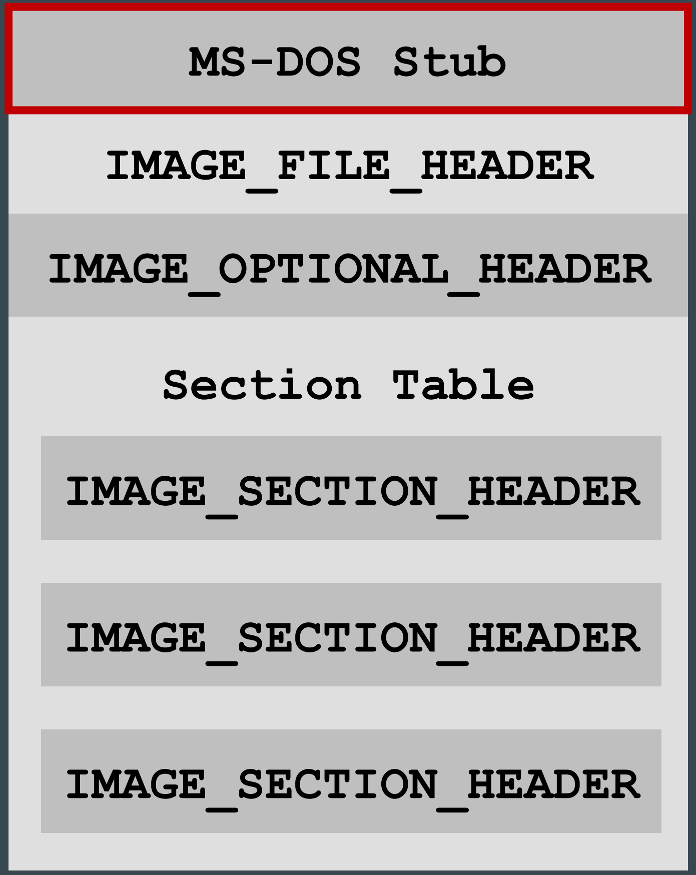
- File format for Windows executables

- About a dozen file types, most notably EXE, DLL, and SYS

- Describes how the executable is loaded into memory

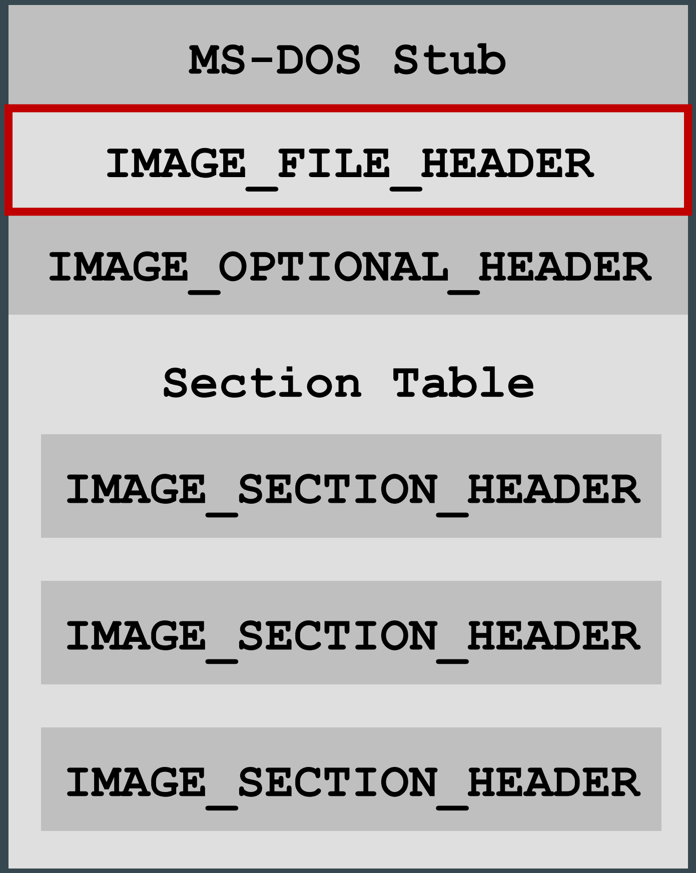
The MS-DOS Stub Header

- Included for legacy DOS compatibility
- “!This program cannot be run in DOS mode.”
- Contains Relative Virtual Address of PE header



The IMAGE_FILE_HEADER

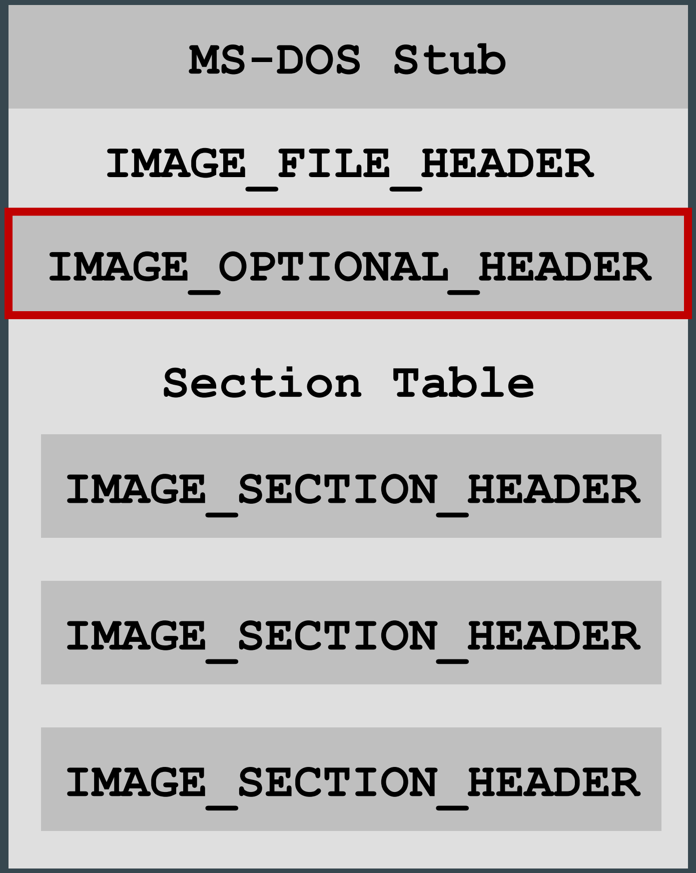
- Basic file information:
  - NumberOfSections
  - TimeDateStamp
  - Characteristics



The IMAGE_OPTIONAL_HEADER

- Not actually optional

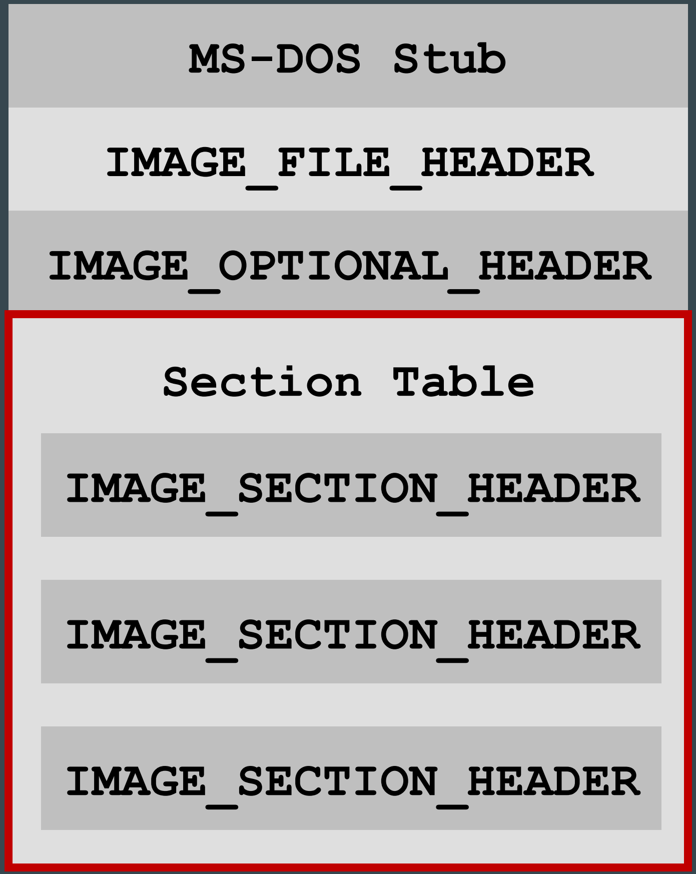
- Contains lots of important file metadata:
    - AddressOfEntryPoint
    - Sizes of various parts of the file that get loaded into memory
    - Versions of linker, OS, image, subsystem



The Section Table

- Contains array of IMAGE_SECTION_HEADERs

- Each contains that section’s:
  - Name
  - VirtualAddress
  - VirtualSize
  - SizeOfRawData
  - Characteristics



MS-DOS Stub

IMAGE_FILE_HEADER

IMAGE_OPTIONAL_HEADER

Section Table

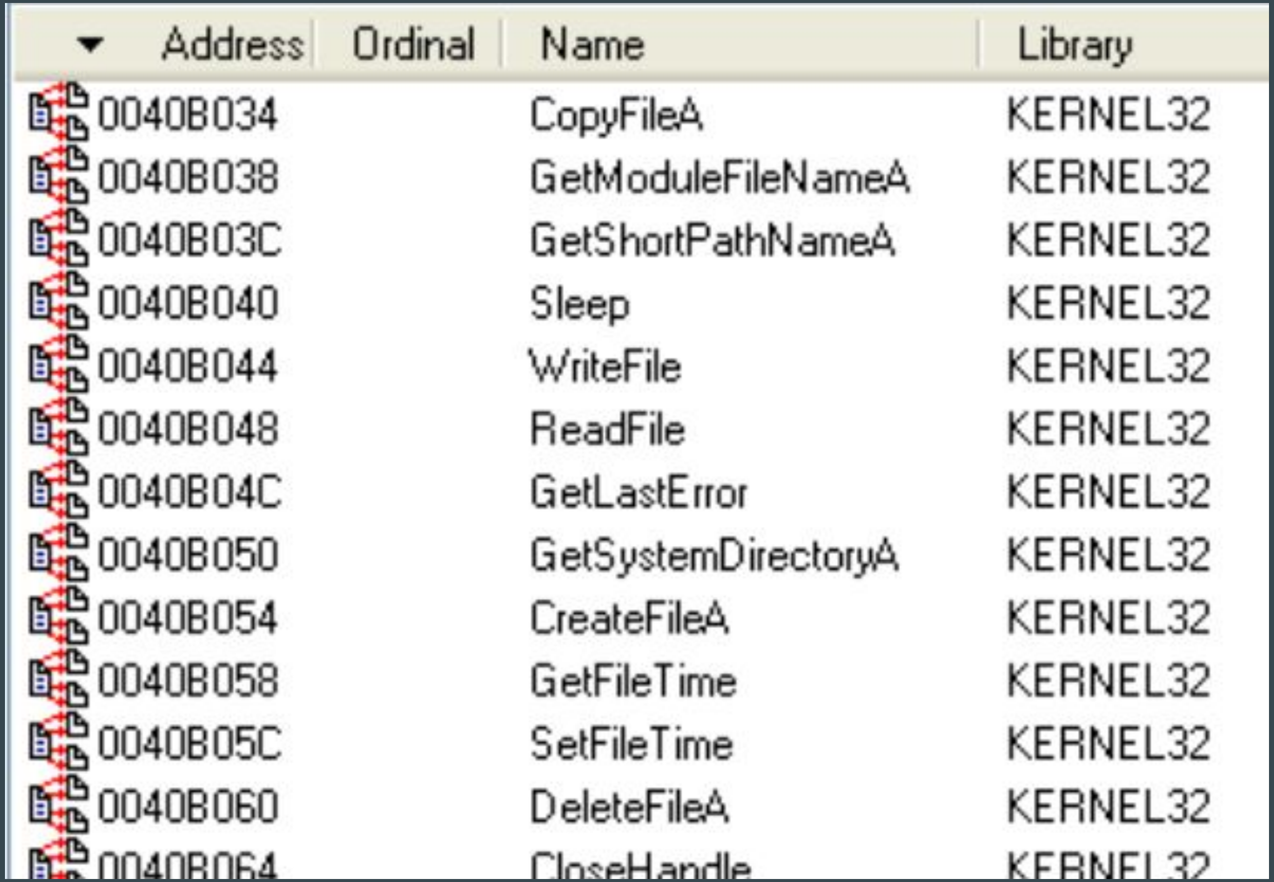
IMAGE_SECTION_HEADER

IMAGE_SECTION_HEADER

IMAGE_SECTION_HEADER

The Import Address Table (IAT)

- Located in the .idata section
- Lists DLLs and functions imported from them



Address	Ordinal	Name	Library
0040B034		CopyFileA	KERNEL32
0040B038		GetModuleFileNameA	KERNEL32
0040B03C		GetShortPathNameA	KERNEL32
0040B040		Sleep	KERNEL32
0040B044		WriteFile	KERNEL32
0040B048		ReadFile	KERNEL32
0040B04C		GetLastError	KERNEL32
0040B050		GetSystemDirectoryA	KERNEL32
0040B054		CreateFileA	KERNEL32
0040B058		GetFileTime	KERNEL32
0040B05C		SetFileTime	KERNEL32
0040B060		DeleteFileA	KERNEL32
0040B064		CloseHandle	KERNEL32

The Rich Header

The Rich Header

- Included in PE files built with the Microsoft compilation toolchain
- Located between the MS-DOS stub and PE header
- Contents are obfuscated, undocumented

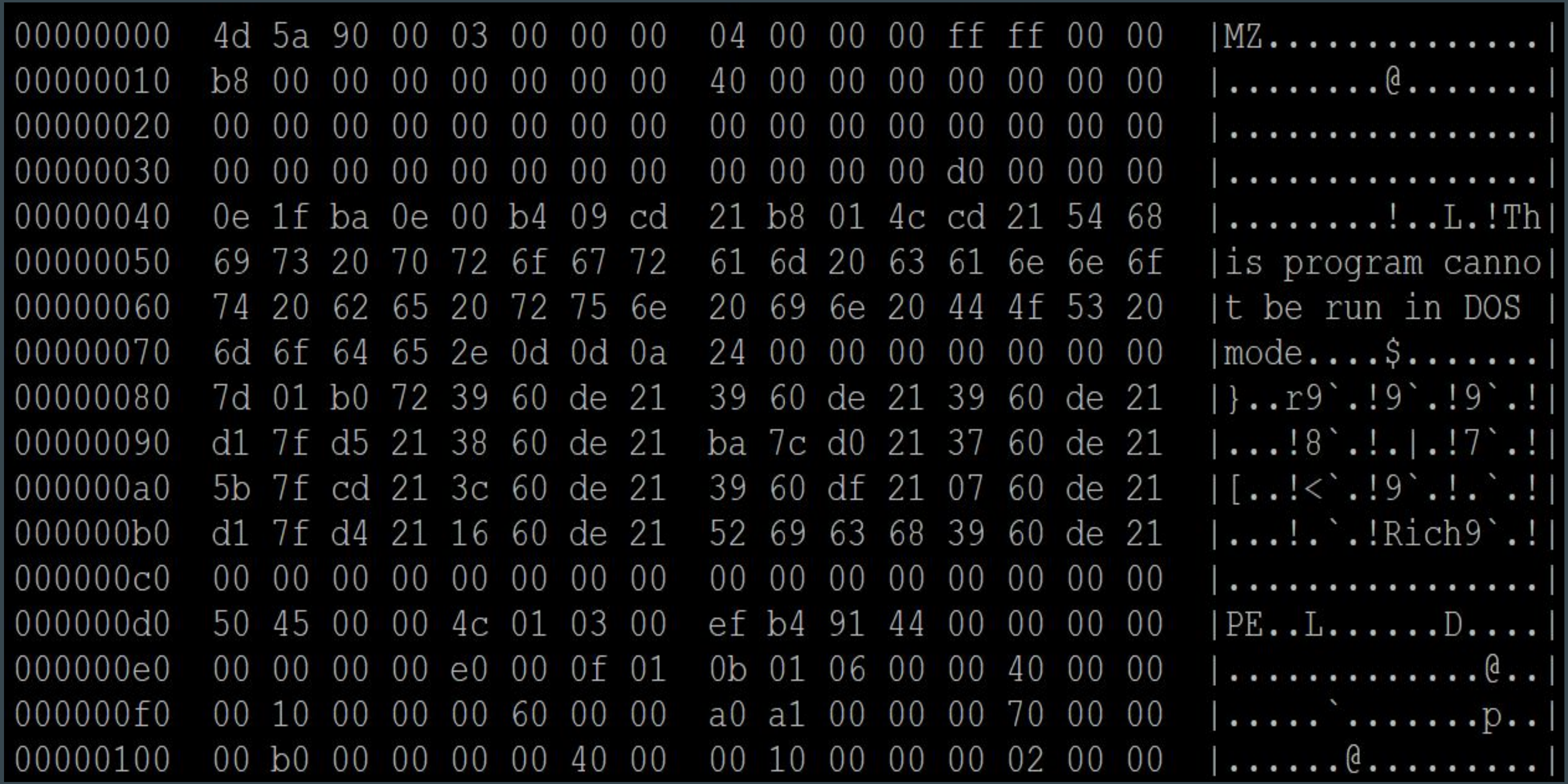
Rich Header Backstory

- Developers afraid Microsoft was storing personal info in Rich header
    - “Devil’s Mark”

- Speculation that Microsoft was using it to track malware authors

- Article about how to de-obfuscate the Rich header published in 2008 by Daniel Pistelli

What it Looks Like



```
00000000  4d 5a 90 00 03 00 00 00  04 00 00 00 ff ff 00 00  |MZ..............|
00000010  b8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  |........@.......|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 d0 00 00 00  |................|
00000040  0e 1f ba 0e 00 b4 09 cd  21 b8 01 4c cd 21 54 68  |........!..L.!Th|
00000050  69 73 20 70 72 6f 67 72  61 6d 20 63 61 6e 6e 6f  |is program canno|
00000060  74 20 62 65 20 72 75 6e  20 69 6e 20 44 4f 53 20  |t be run in DOS |
00000070  6d 6f 64 65 2e 0d 0d 0a  24 00 00 00 00 00 00 00  |mode....$.......|
00000080  7d 01 b0 72 39 60 de 21  39 60 de 21 39 60 de 21  |}..r9`.!9`.!9`.!|
00000090  d1 7f d5 21 38 60 de 21  ba 7c d0 21 37 60 de 21  |...!8`.!.|.!7`.!|
000000a0  5b 7f cd 21 3c 60 de 21  39 60 df 21 07 60 de 21  |[..!<`.!9`.!.`.!|
000000b0  d1 7f d4 21 16 60 de 21  52 69 63 68 39 60 de 21  |...!.`.!Rich9`.!|
000000c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000d0  50 45 00 00 4c 01 03 00  ef b4 91 44 00 00 00 00  |PE..L......D....|
000000e0  00 00 00 00 e0 00 0f 01  0b 01 06 00 00 40 00 00  |.............@..|
000000f0  00 10 00 00 00 60 00 00  a0 a1 00 00 00 70 00 00  |.....`.......p..|
00000100  00 b0 00 00 00 00 40 00  00 10 00 00 00 02 00 00  |......@.........|
```

MS-DOS Stub

Rich Header

PE Header

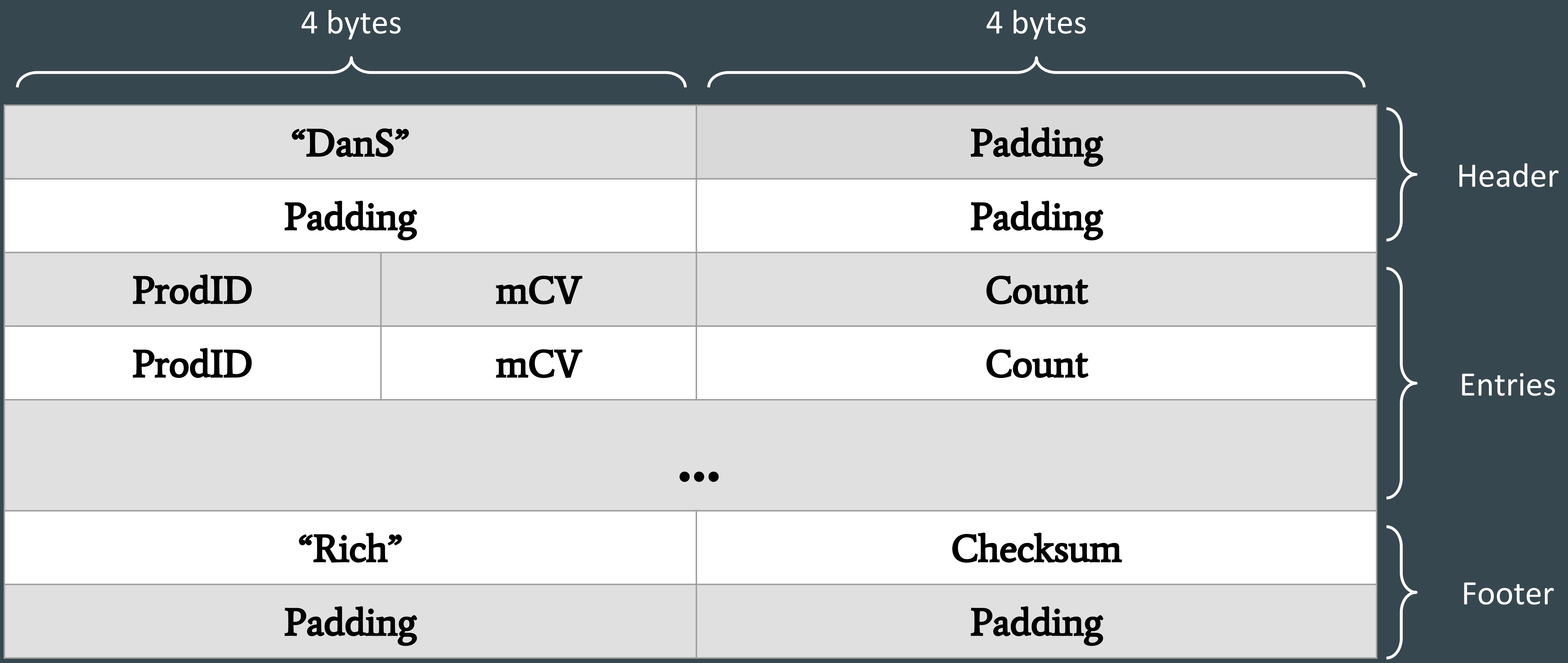
Rich Header Checksum

- The 4 bytes after “Rich” are a checksum

- Linker calculates it based on:
  - MS-DOS stub length
  - MS-DOS stub contents
  - Contents of the Rich header

- Contents of the Rich header are obfuscated using checksum as an XOR key

De-Obfuscated Rich Header

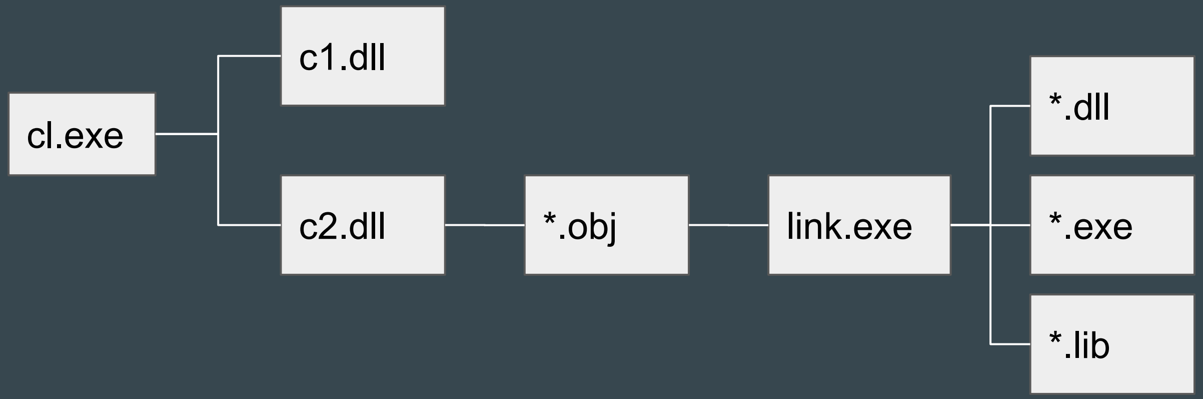


4 bytes		4 bytes	
“DanS”		Padding	Header
Padding		Padding	
ProdID	mCV	Count	Entries
ProdID	mCV	Count	
…			
“Rich”		Checksum	Footer
Padding		Padding	

How the Rich Header is Built

- The backend compiler (c2.dll) inserts the @comp.id into each object it generates during compilation

- The linker reads in all this information, keeps track of how many times each object is used, and builds the Rich header



Prior Research

The Rich Header + Malware Analysis

- Very little public literature that incorporates both
- *Finding the Needle: A Study of the PE32 Rich Header and Respective Malware Triage*, Webster et. al
- *The Devil's in the Rich Header*, Securelist blog
- *Case Studies in Rich Header Analysis and Hunting*, Ropgadget blog

Finding the Needle

- Surveyed over 1 million PE files, ~70% had Rich headers
- How packers affect the Rich header
- Identifying metadata tampering
- Finding related malware samples

The Devil’s in the Rich Header

- OlympicDestroyer worm was an attribution nightmare
- Had Rich header identical to wiper used by Lazarus group
- Article proved that OlympicDestroyer’s Rich header was a false flag
    - mscoree.dll

Case Studies in Rich Header Analysis

- Using YARA to track malware by Rich header contents

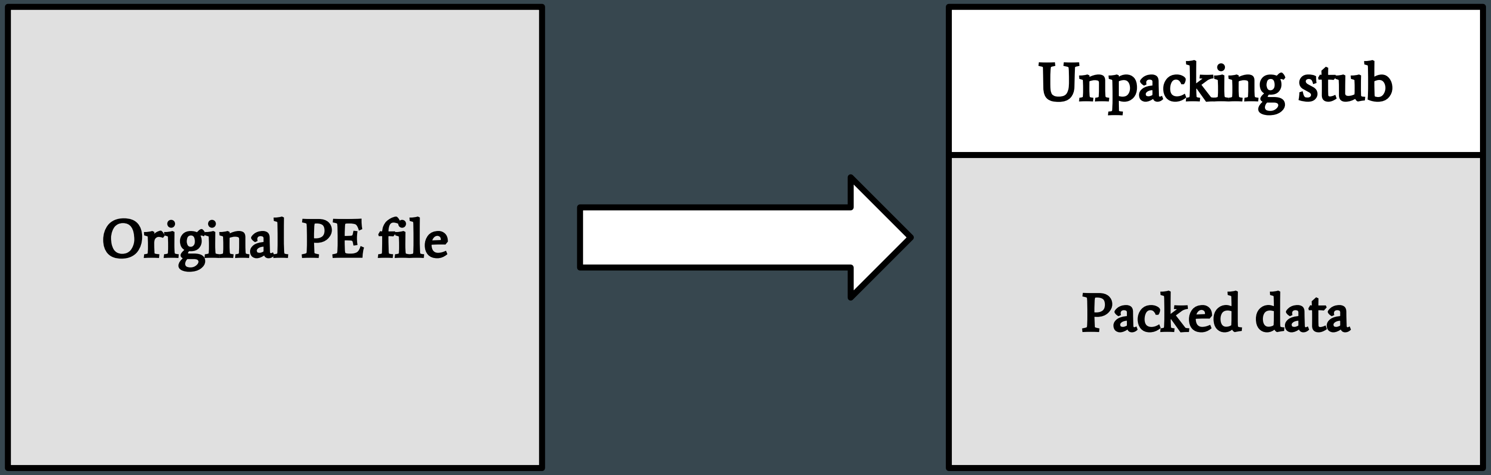
```
import "pe"
import "hash"

rule rich_example {
  condition:
    hash.md5(pe.rich_signature.clear_data) == "[MD5]"
}
```

Packers and the Rich Header

Packers 101

- Obfuscates executable code
- Inhibits static analysis



Finding the Needle Packer Results

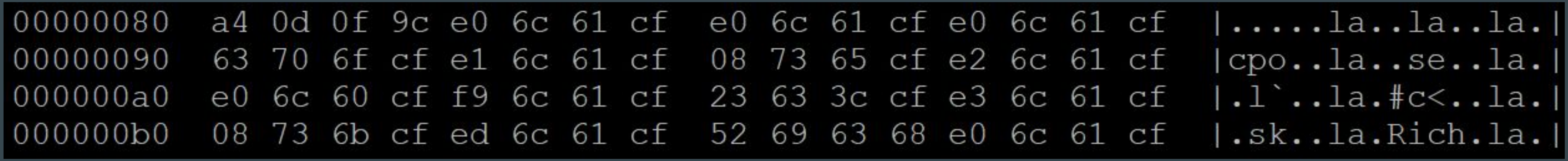
- Surveyed 5 common malware packers
- UPX, ASPack, NSIS do not modify the Rich header
- Found that other packers may corrupt / remove the Rich header

Our Own Findings

Packer Name	Not Modified	Inadvertently Modified	Purposefully Modified
ASPack	X		
PECompact	X		
Petite	X		
Themida	X		
UPX	X		
FSG		X	
Upack		X	
VMProtect		X	
RLPack			X

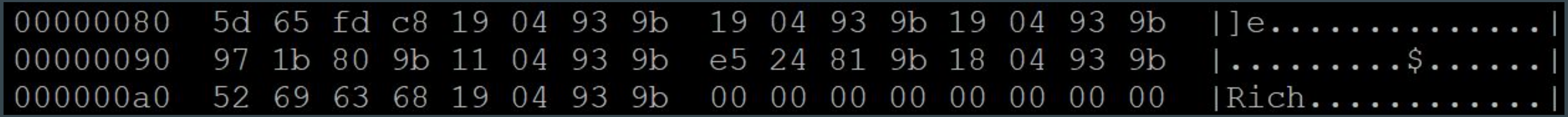
RLPack

Rich header of unpacked sample:



```
00000080  a4 0d 0f 9c e0 6c 61 cf  e0 6c 61 cf e0 6c 61 cf  |.....la..la..la.|
00000090  63 70 6f cf e1 6c 61 cf  08 73 65 cf e2 6c 61 cf  |cpo..la..se..la.|
000000a0  e0 6c 60 cf f9 6c 61 cf  23 63 3c cf e3 6c 61 cf  |.l`..la.#c<..la.|
000000b0  08 73 6b cf ed 6c 61 cf  52 69 63 68 e0 6c 61 cf  |.sk..la.Rich.la.|
```

Rich header of sample packed with RLPack:



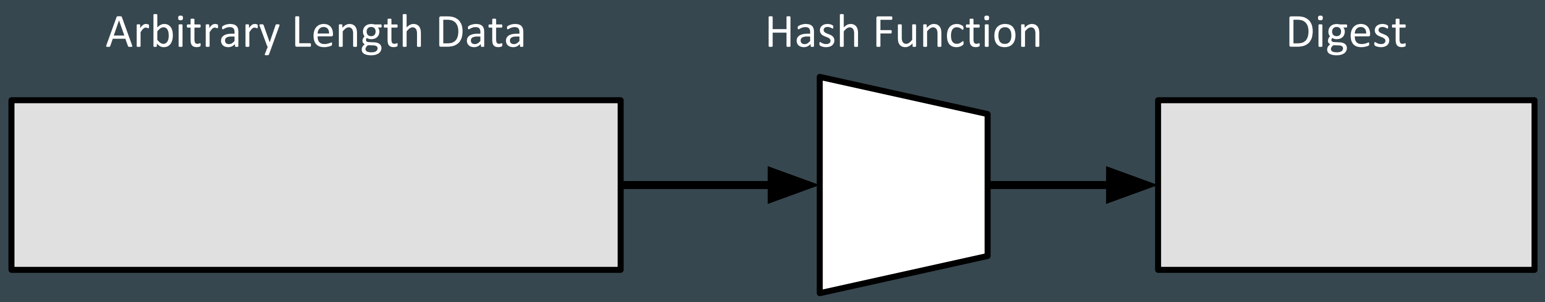
```
00000080  5d 65 fd c8 19 04 93 9b  19 04 93 9b 19 04 93 9b  |]e..............|
00000090  97 1b 80 9b 11 04 93 9b  e5 24 81 9b 18 04 93 9b  |.........$......|
000000a0  52 69 63 68 19 04 93 9b  00 00 00 00 00 00 00 00  |Rich............|
```

Metadata Hashing

What is a Hash Function?

- Maps arbitrary-length data to a fixed-length digest

- Properties of cryptographic hash functions:
  - Deterministic
  - Not reversible
  - Resistant to collisions



What is Metadata Hashing?

- Used to efficiently query malware samples that share metadata
- Use malware metadata as input to a hash function
- Store malware in a database indexed on metadata hash

Imphash

- Hash of functions in the order they are listed in the IAT

- Malware samples with same Imphash likely have similar source code

Imphash Weaknesses

- Low confidence if a malware sample does not have many imports

- Packed malware frequently uses runtime linking

- Changes the imphash, hinders static analysis of imports

Pehash

- Uses metadata from:
  - IMAGE_FILE_HEADER
  - IMAGE_OPTIONAL_HEADER
  - IMAGE_SECTION_HEADERs

- Polymorphic malware from the same family will often share a pehash

Pehash Weaknesses

- Very strict - high confidence but high false negative rate
- Packing a malware sample almost always changes its PE sections
- Changes the pehash because it relies on section metadata

RichPE Metadata Hash

How the RichPE Hash is Computed

Rich Header Entries	ProdID
	mCV
	Masked Count
IMAGE_FILE_HEADER	Machine
	Characteristics
IMAGE_OPTIONAL_HEADER	Subsystem
	MajorLinkerVersion
	MinorLinkerVersion
	MajorOperatingSystemVersion
	MinorOperatingSystemVersion
	MajorImageVersion
	MinorImageVersion
	MajorSubsystemVersion
	MinorSubsystemVersion

Metadata Hashes vs ASPack

	Original File	Packed file
MD5	0c5e9f564115bfcbee66377a829de55f	0c685b6a355eb493e9e07296ba95619c
Imphash	63bf00403dae8328fff132b19e7e9b46	1417f7317798bb198313884c0e6740a4
Pehash	a2793a4e5a7c5c55549b0f6c8551ccb575713eb2	6e4a9338bf5378a218dda0336142872e9d29f8aa
RichPE	8f6dcb3f2e8facfc3f8ba79ff5cdea50	8f6dcb3f2e8facfc3f8ba79ff5cdea50

Metadata Hashes vs PECompact

	Original File	Packed file
MD5	c627e595c9ec6dc2199447aeab59ac03	c1732007b2972d782fc833c424f10f20
Imphash	387de552b3e0b8567609f40c93db20c5	None
Pehash	24c52a685c65c40943cd7b7d1a63f6e772da71eb	8e570144c042fda180677f16c83b26492d93394c
RichPE	7569682a56f9fc1e307cc57e1bd3412a	7569682a56f9fc1e307cc57e1bd3412a

Metadata Hashes vs Petite

	Original File	Packed file
MD5	995442f722cc037885335340fc297ea0	f366ca2f54ed38c555d8230071611212
Imphash	8e6265b4d84471cbb32c119bcd93dc47	318e98359811909d24ad34aac812aa63
Pehash	e88e1fc9d900ccd770deb492ff855f499d2fb238	abd8ad4daab0aa3b1dc36a19cde3c23fafe77868
RichPE	7ef53a9bafca2bd6f35f1692697e28d8	7ef53a9bafca2bd6f35f1692697e28d8

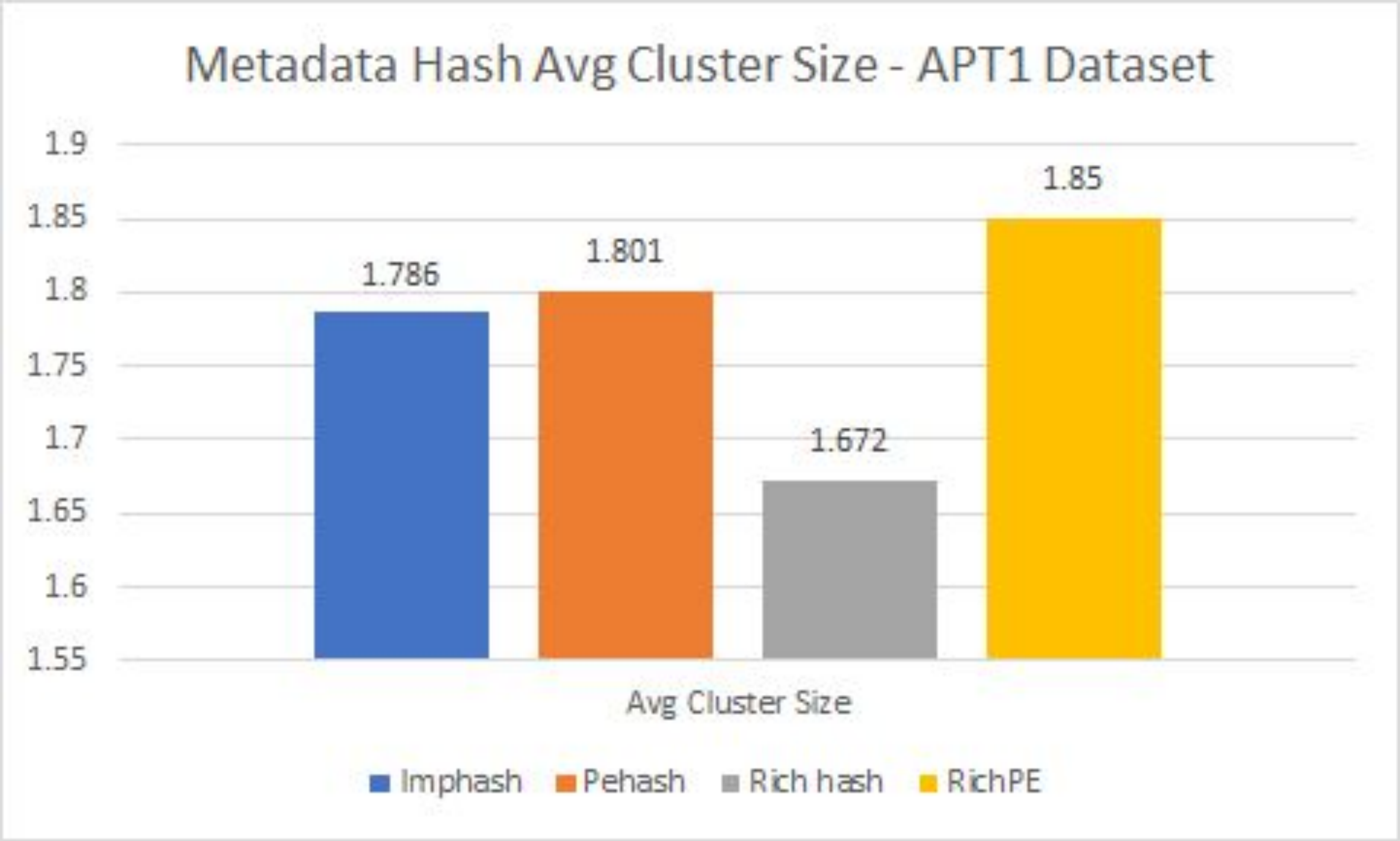
Metadata Hashes vs Themida

	Original File	Packed file
MD5	6b31344b40e2af9c9ee3ba707558c14e	99ac9ccb4f0db04f24f2c80d8f6e46b3
Imphash	5776b1400eb618f9f213ae9dee30ce2f	baa93d47220682c04d92f7797d9224ce
Pehash	48be6716fa7c6d8bb7256138690662badf1fe5e9	ee636745bdab534b65a6a447525a84ba2d9ca10a
RichPE	e315a4477592046c9cbf7652003732b1	e315a4477592046c9cbf7652003732b1

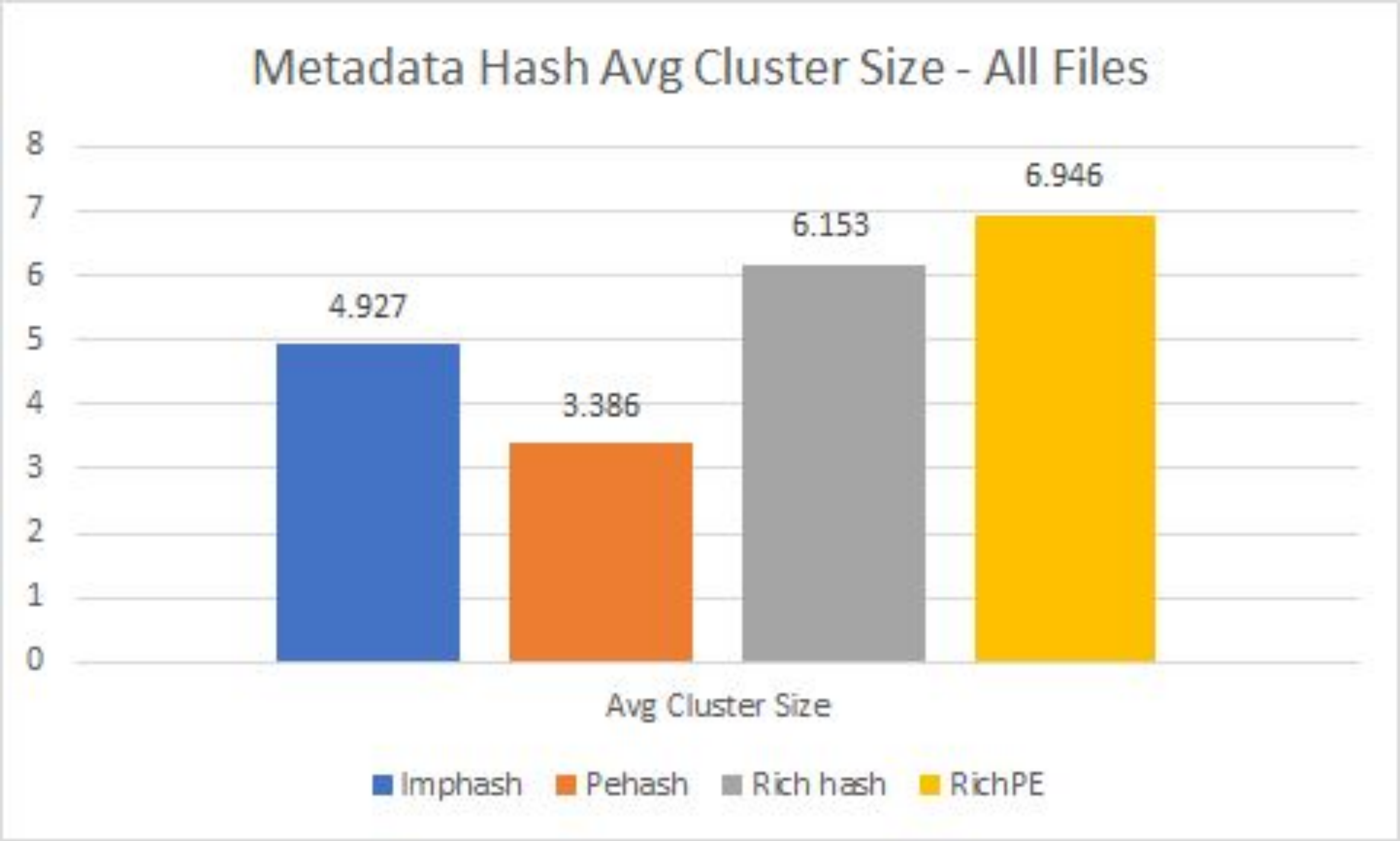
Metadata Hashes vs UPX

	Original File	Packed file
MD5	6b97b3cd2fcfb4b74985143230441463	f7fe21ab370efc5c4b4478627189f011
Imphash	ba47a0478b3cdd3b7d2c2438b409a2ca	e21e17ff820bc123b050075aae0d0a6b
Pehash	eea113541bb30c2a955f9253fcd65bad609e2e6d	896c92b861ae1b4457ecce0002dedbeb3e8a13ad
RichPE	3ee78f0e6bd1d24dc1a7820e95f9b604	3ee78f0e6bd1d24dc1a7820e95f9b604

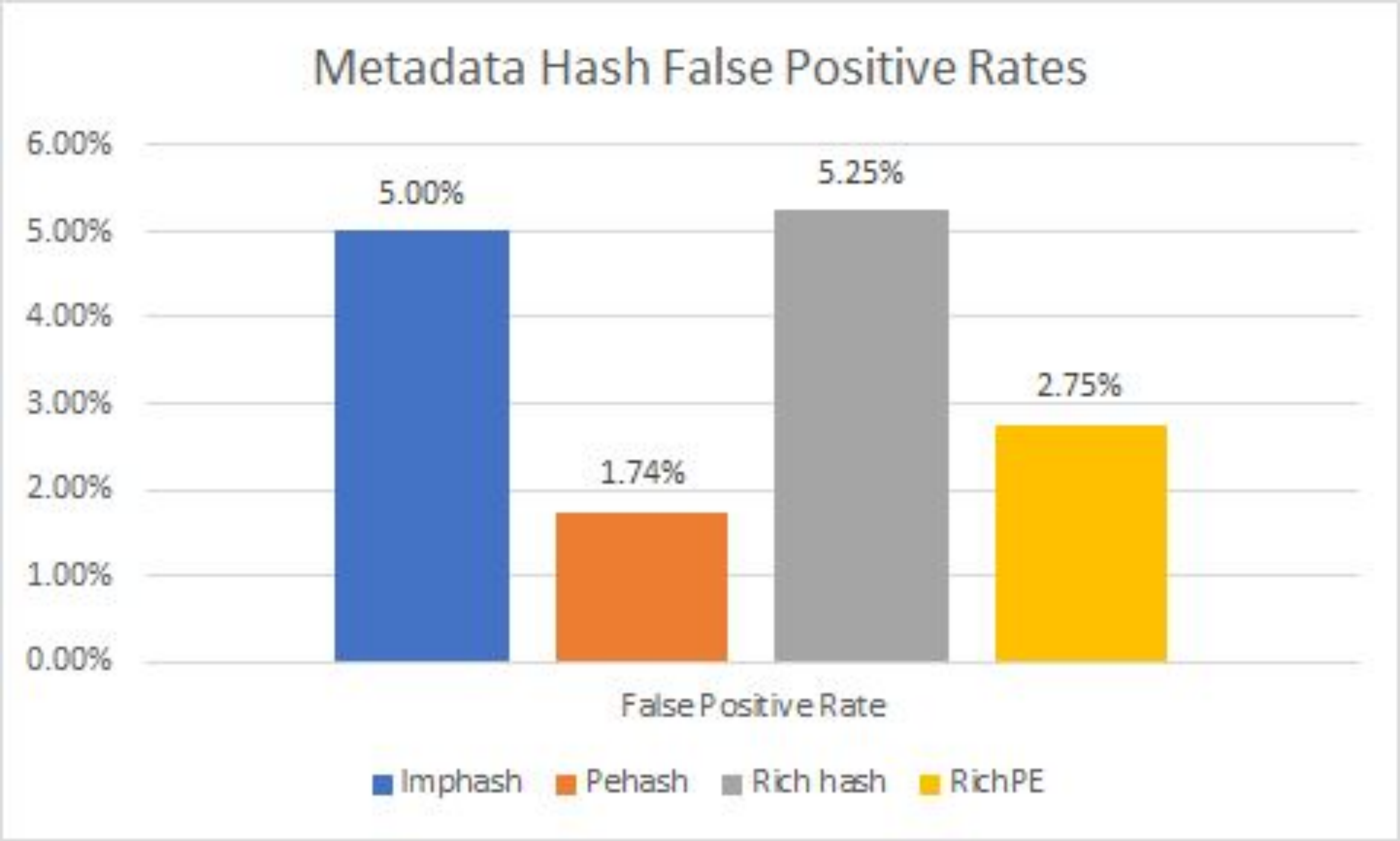
Metadata Hash Stats - APT1 Dataset



Metadata Hash Stats - All Files



RichPE Hash Accuracy



Other Cool RichPE Findings

- Identified probable APT1 malware in the VirusShare dataset
- Identified unpacked and packed malware of the same family
- Identified malware samples of the same family packed with different packers

RichPE Weaknesses

- Doesn’t work on malware without a Rich header
- Not all packers leave the Rich header alone
- Still in proof-of-concept stage, need to do more testing

Rich Header Tamper Detection

Motivation

- Adversaries are already spoofing Rich headers as false flags
- How easy is it for an adversary to spoof a Rich header?
- How challenging is it to detect?

Checking Rich Header Validity

- Must have valid checksum
  - Otherwise, MS-DOS stub or Rich header has been modified

- Cannot contain duplicate ProdID + mCV entries

Checking Rich Header Validity

- Typically, last entry in Rich header is the linker version

- Can verify this against IMAGE_OPTIONAL_HEADER:
  - MajorLinkerVersion
  - MinorLinkerVersion

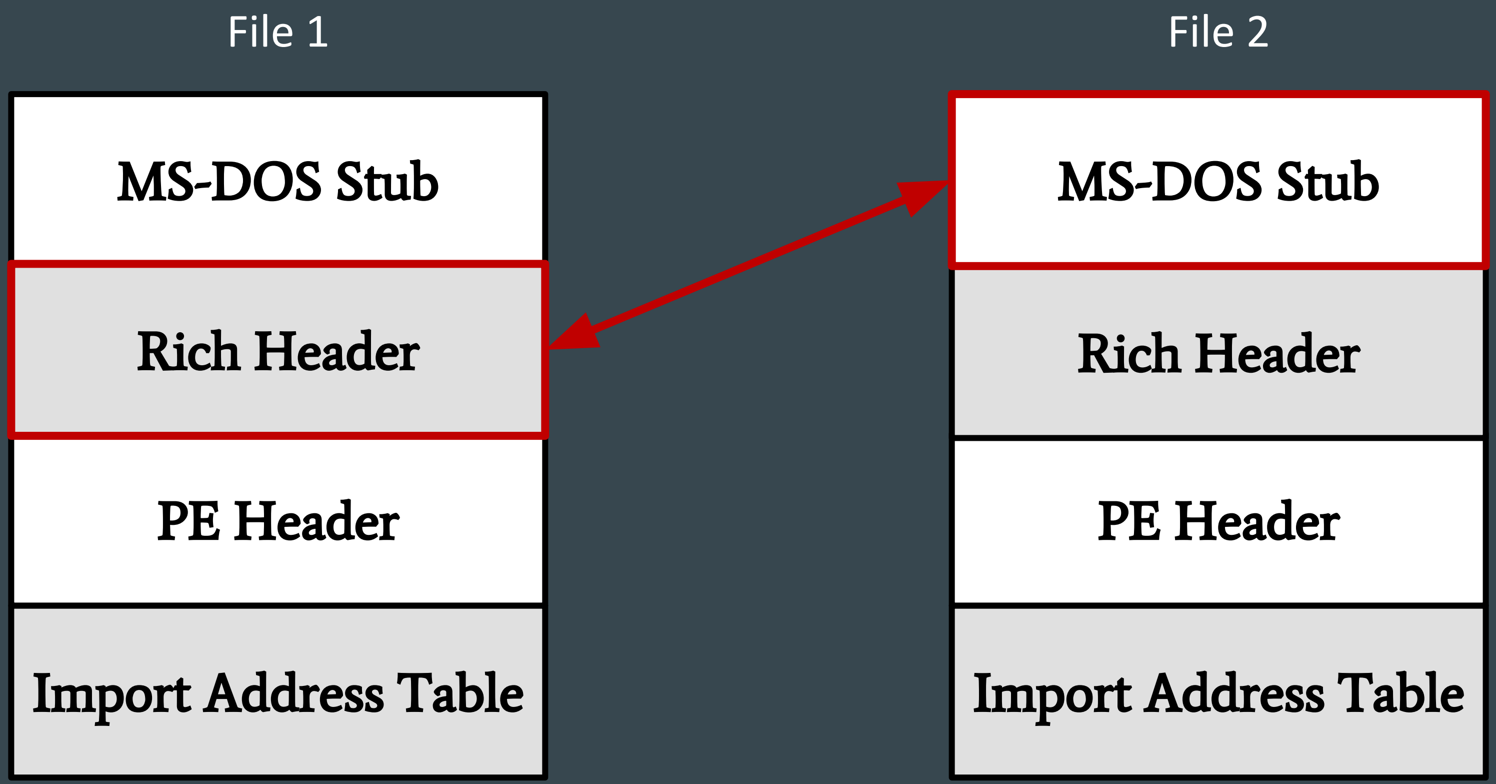
- If they don't match, either the Rich header or PE header has been modified

Checking Rich Header Validity

- Rich header entry with ProdID 1 is named “Import0”
- Never less than the number of imported functions in the IAT
- If it is, either the Rich header or IAT has been modified

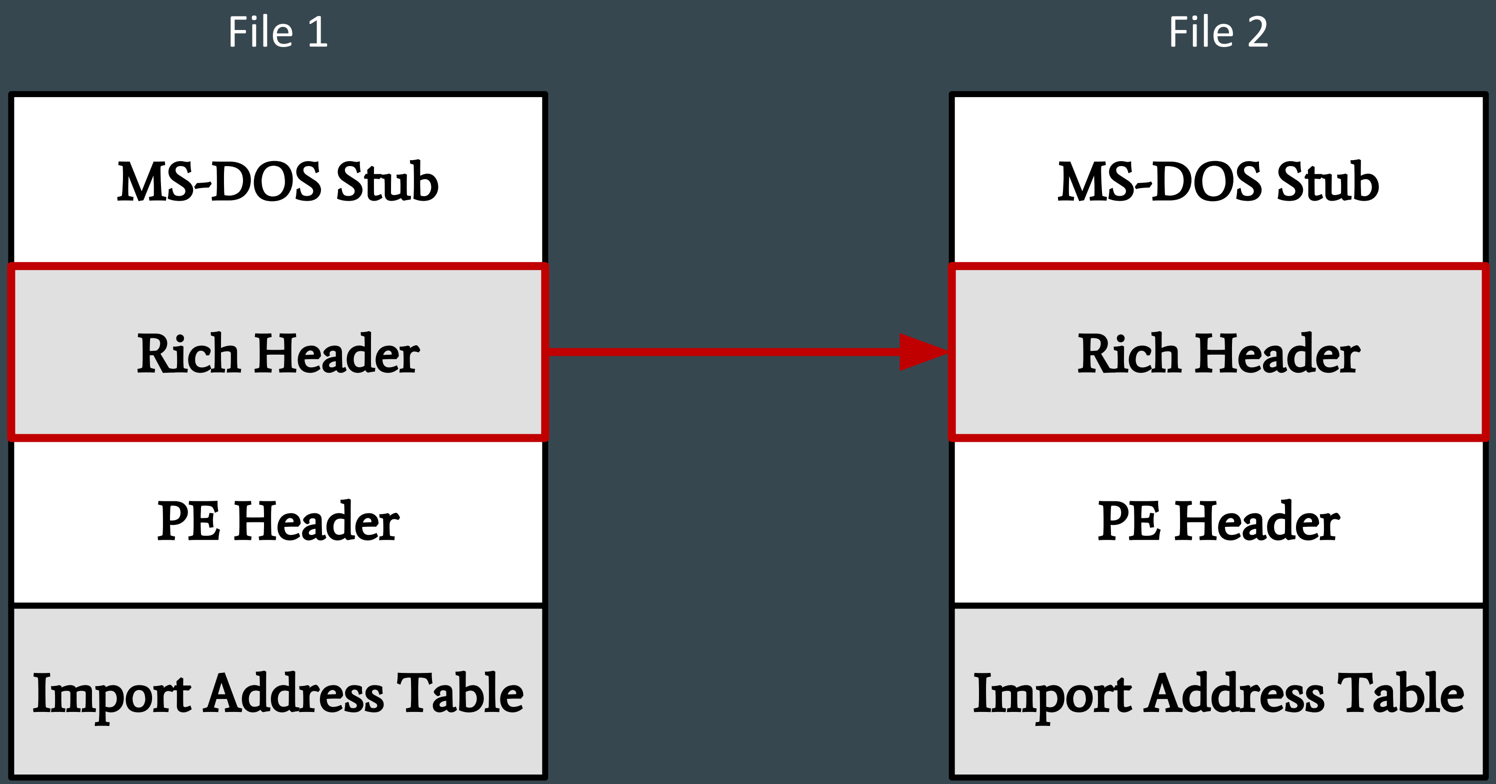
Spoofing a Rich Header?

1. Compute checksum from file 2’s MS-DOS stub and file 1’s Rich header



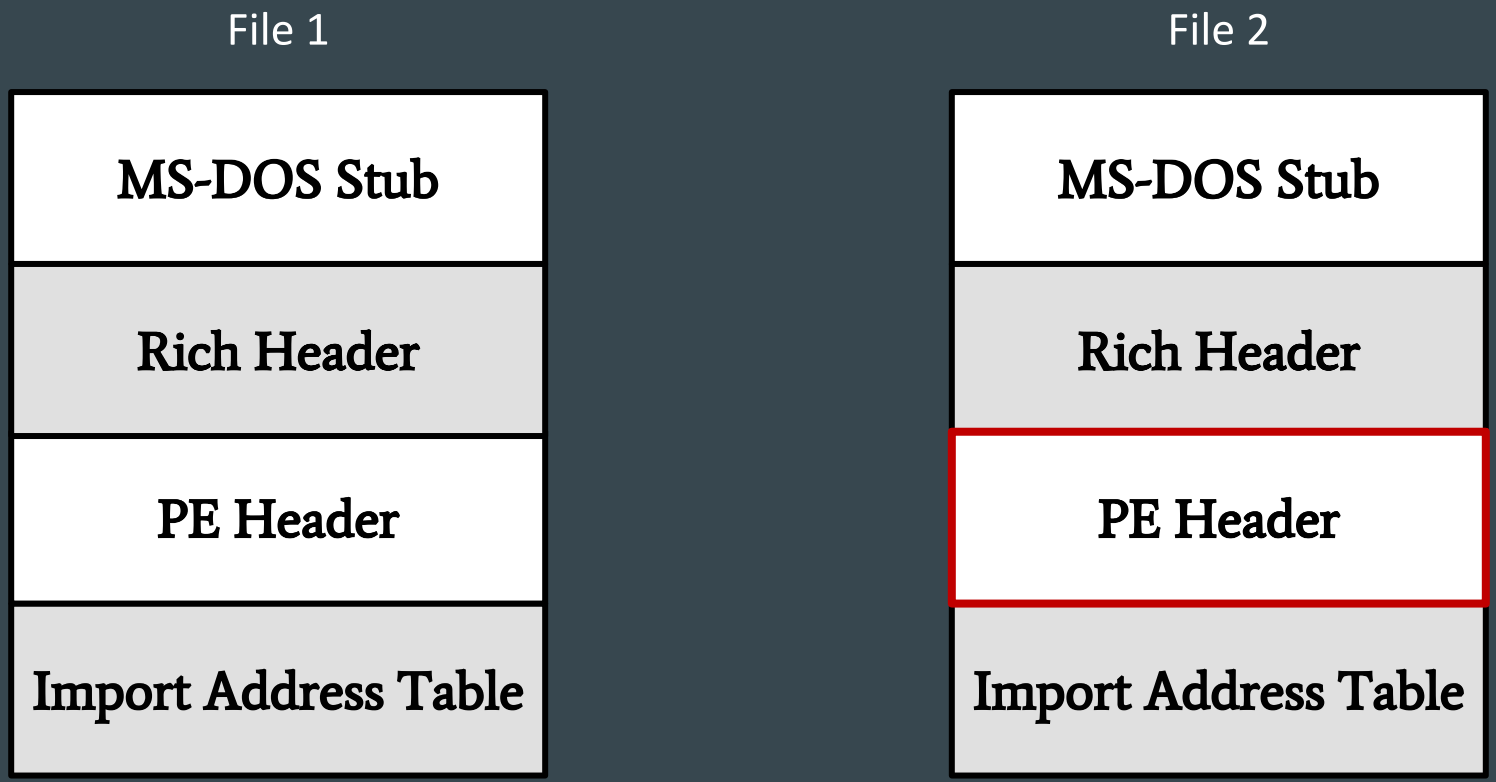
Spoofing a Rich Header?

2. XOR contents of file 1's Rich header with checksum, insert into file 2



Spoofing a Rich Header?

3. Edit MajorLinkerVersion and MinorLinkerVersion to match Rich header



File 1

MS-DOS Stub

Rich Header

PE Header

Import Address Table

File 2

MS-DOS Stub

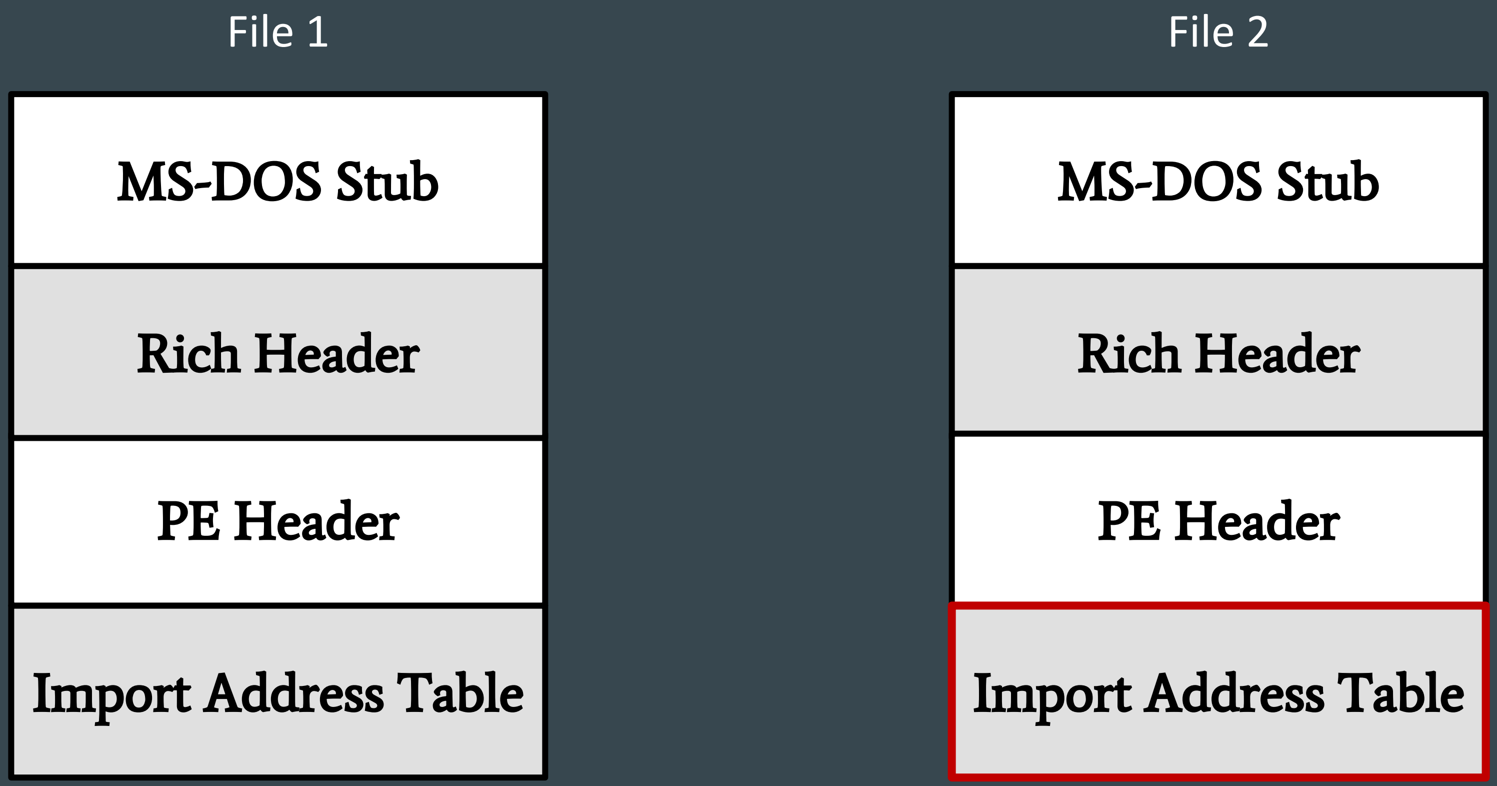
Rich Header

PE Header

Import Address Table

Spoofing a Rich Header?

4. Modify IAT to pass Import0 count check (runtime linking)



File 1

MS-DOS Stub

Rich Header

PE Header

Import Address Table

File 2

MS-DOS Stub

Rich Header

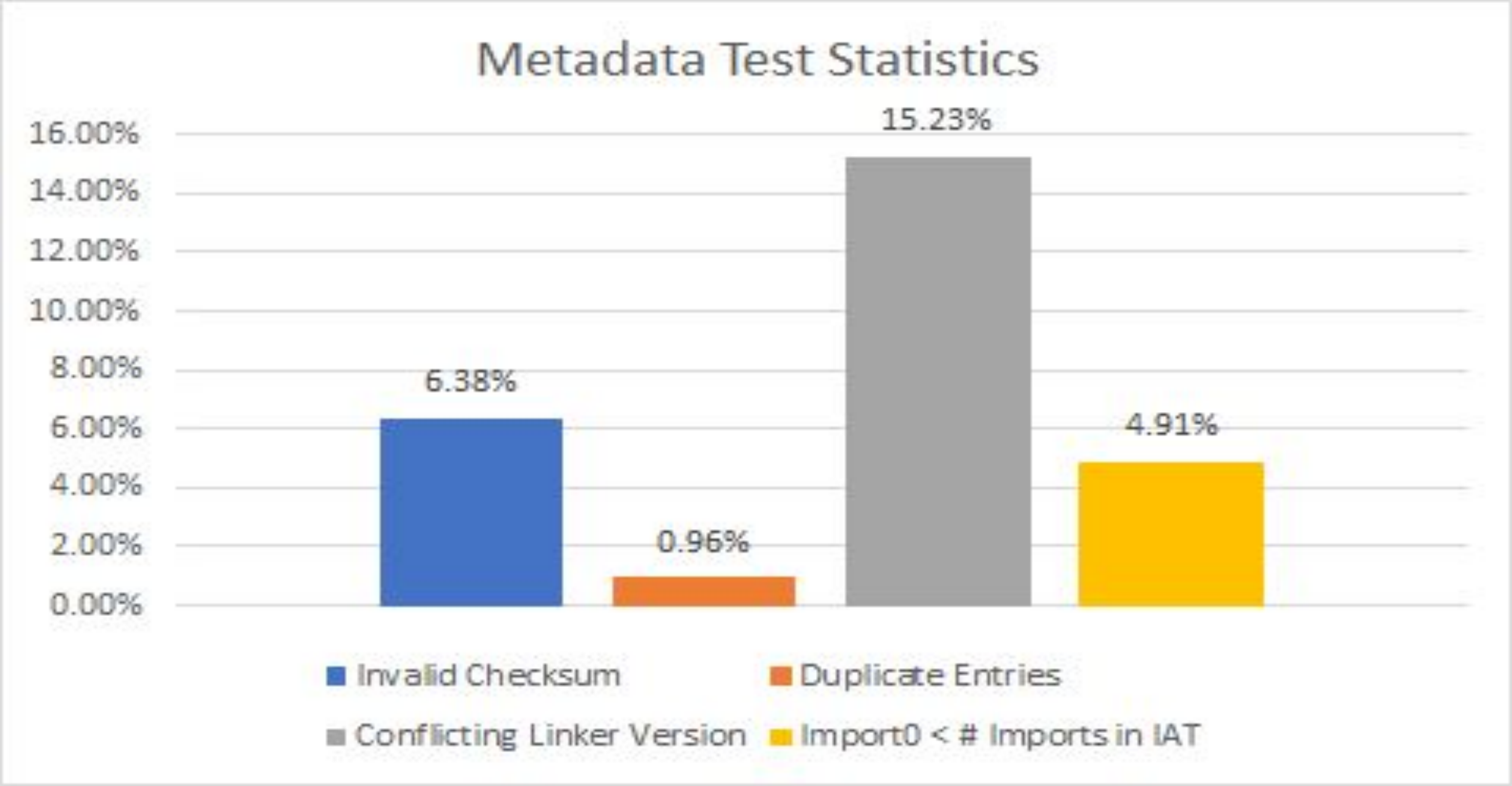
PE Header

Import Address Table

Rich Header Spoofing Feasibility?

- Passing most of the metadata checks is trivial
- Altering the IAT is complicated but doable
- Spoofed Rich header would probably not stand up to manual analysis
    - *The Devil's in the Rich Header*

Invalid Metadata Test Stats



OlympicDestroyer vs Basic Metadata Tests

- Passes checksum, duplicate entries, linker tests
- Fails import count tests

RLPack vs Basic Metadata Tests

- Passes all of our tests
- Sets PE header linker version to 5:12 (matches Rich header)
- Doesn’t have an Import0 entry
  - Uncommon but not necessarily an indicator of tampering

Conclusion

Acknowledgements

- Dr. Charles Nicholas and Dr. Haibin Zhang, UMBC CSEE Dept
- Matt Elder and Bill La Cholter, Johns Hopkins APL
- The ShadowServer Foundation
- Mila Parkour, DeepEnd Research

Source Code

- https://github.com/RichHeaderResearch/RichPE

Questions?

References

- https://www.ntcore.com/files/richsign.htm
- https://bytepointer.com/articles/the_microsoft_rich_header.htm
- https://www.sec.in.tum.de/i20/publications/finding-the-needle-a-study-of-the-pe32-rich-header-and-respective-malware-triage
- https://securelist.com/the-devils-in-the-rich-header/84348/
- http://ropgadget.com/posts/richheader_hunting.html
- https://docs.microsoft.com/en-us/previous-versions/ms809762(v=msdn.10)
- https://github.com/erocarrera/pefile/blob/master/pefile.py
- https://yara.readthedocs.io/en/v3.8.1/

Pefile Python Library

- Awesome Python library for parsing the PE header
- Can use it to parse a file's Rich header
- https://github.com/erocarrera/pefile