Malpedia: A Collaborative Effort to Inventorize the Malware Landscape

Daniel Plohmann
daniel.plohmann@fkie.fraunhofer.de

@push_pnx

Martin Clauß
martin.clauß@fkie.fraunhofer.de

Steffen Enders
steffen.enders@tu-dortmund.de

Elmar Padilla
elmar.padilla@fkie.fraunhofer.de

2017-12-07 | Botconf, Montpellier

Fraunhofer FKIE

$whoami
Daniel Plohmann

Fraunhofer FKIE

- Security Researcher @ Fraunhofer (Europe's largest organisation for applied research)

- Research Scope:
  - Malware Analysis
  - Reverse Engineering
  - Automation

Fraunhofer FKIE

Outline

- Summary
- Motivation *(or: how it began)*
- Approach
- The Malpedia Corpus & Platform
- A Comparative Structural Analysis of Windows Malware
- Future Plans / Conclusion

Summary

Summary
TL;DR

- What is Malpedia?
  - A free, independent, pooled resource for confidently labeled, unpacked reference samples for malware families and versions
  - Meta data tracker for info such as references (analysis reports, blogs, …), YARA rules, actors, tied to these families
  - Status (2017-12-01): 2491 samples for 669 families, multi-platform (WIN, ELF, APK, OSX, …)

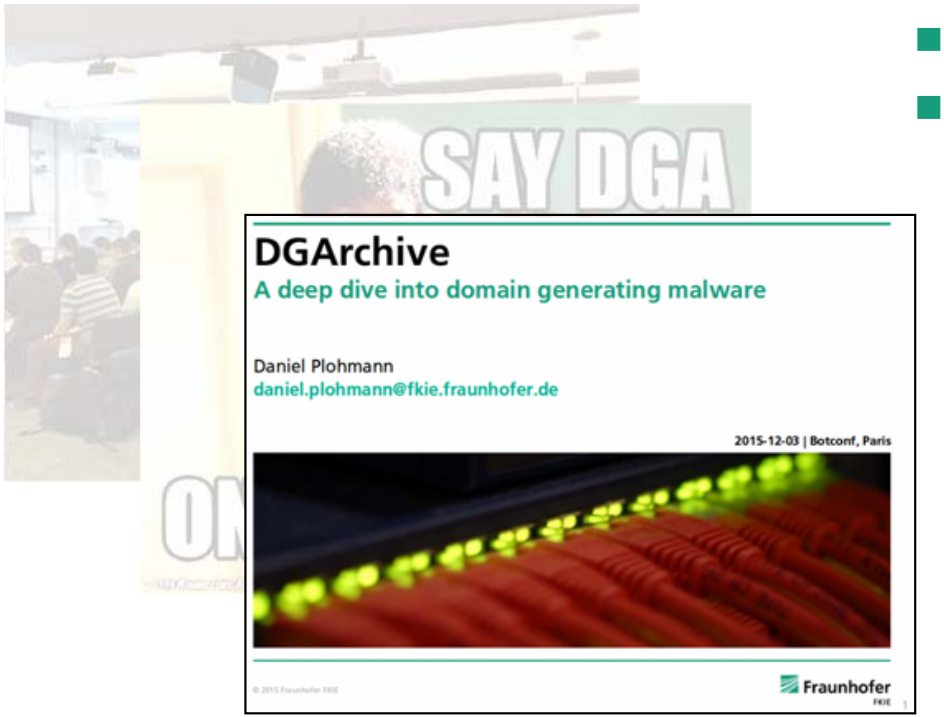
- Our Contributions
  - Definition of requirements for malware corpora and a reference corpus + platform implementing these
  - A Comprehensive, quantitative static analysis of structural features for 446 Windows malware families

Motivation

… or how it began

Fraunhofer
FKIE

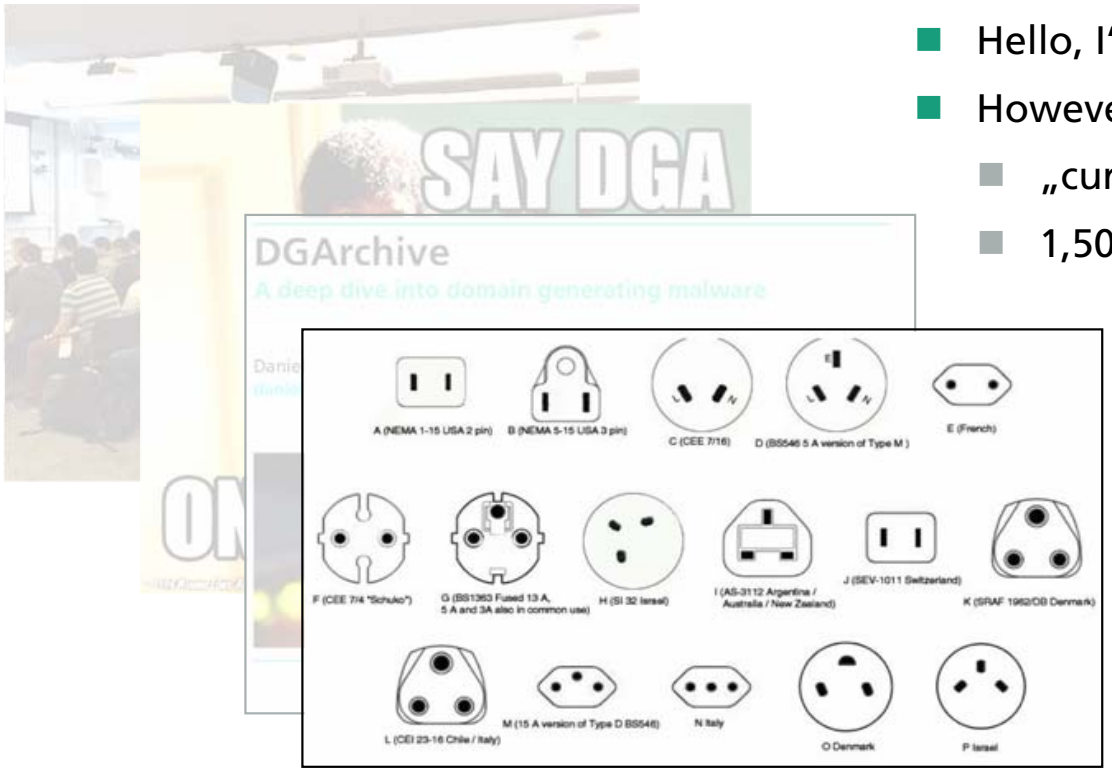
Throwback: Botconf 2015



- Hello, I'm one of the culprits! ☺
- However, from feedback received:
  - „curated, systematized data is cool and helpful"
  - 1,500,000+ requests to DGArchive since launch

[1] https://www.lexsi.com/securityhub/wp-content/uploads/2015/12/BotconfLexsi5.jpg
[2] https://www.lexsi.com/securityhub/wp-content/uploads/2015/12/BotconfLexsi-3.png
[3] https://ih1.redbubble.net/image.129250252.8619/flat,800x800,075,f.u1.jpg
[4] https://www.botconf.eu/wp-content/uploads/2015/12/OK-LT10-Wayne-Crowder-Wrestling-elephants-Tackling-malware-research-issues.pdf

Throwback: Botconf 2015



Plug What? Plug X!

- Hello, I'm one of the culprits! ☺
- However, from feedback received:
  - „curated, systematized data is cool and helpful"
  - 1,500,000+ requests to DGArchive since launch

- Wayne Crowder's lightning talk
  - Malware Name Synonym Dilemma

[1] https://www.lexsi.com/securityhub/wp-content/uploads/2015/12/BotconfLexsi5.jpg
[2] https://www.lexsi.com/securityhub/wp-content/uploads/2015/12/BotconfLexsi-3.png
[3] https://ih1.redbubble.net/image.129250252.8619/flat,800x800,075,f.u1.jpg
[4] https://www.botconf.eu/wp-content/uploads/2015/12/OK-LT10-Wayne-Crowder-Wrestling-elephants-Tackling-malware-research-issues.pdf

Motivation
My situation post-Botconf 2015

- Malware identification seems to be an issue… synonyms as well…
- I had a respectable pile of „okayish but not greatly“ sorted malware investigations.
  - A code-centric inventory with emphasis on static analysis would be awesome!
  - The aspect of preservation sounds like a good idea as well.

- Related work:
  - Botnets.fr [1] by Éric Freyssinet et al. — wiki-like, 1500+ pages
  - theZoo [2] by Yuval Nativ et al. — labeled, semi-structured, packed
  - VirusBay.io [3] by Ido Naor et al. — knowledge exchange platform, IR-focused
  - ID Ransomware [4] by MalwareHunterTeam — hidden collection, ransomware only

[1] https://botnets.fr
[2] https://github.com/ytisf/theZoo
[3] https://beta.virusbay.io
[4] https://id-ransomware.malwarehunterteam.com/index.php

Approach

Approach

One step back, let's do this systematically

- Goals for a malware corpus
- Review of „Prudent Practices", Rossow et al. 2012
- Derive & refine requirements for a malware corpus, tailored for static analysis

Approach
Goals for a Malware Corpus (with Static Analysis in Mind)

- Representative Data
  - Coverage in multiple angles: temporal, platforms, families, versions, …
- Easily accessible format
  - Clean, unpacked samples
  - Meta information: labels, references, …
- A resource with practical use
  - Topical / Maintained
  - Tooling to work with it, potentially even signatures?
  - Vendor-neutral / consensual ground-truth

Approach

Review of Rossow et al.‘s „Prudent Practices“

Prudent Practices for Designing Malware Experiments: Status Quo and Outlook

Christian Rossow*‡, Christian J. Dietrich*, Chris Grier†‡, Christian Kreibich†‡,
Vern Paxson†‡, Norbert Pohlmann*, Herbert Bos§, Maarten van Steen§
* Institute for Internet Security, Gelsenkirchen
{rossow,dietrich,pohlmann}@internet-sicherheit.de
‡ University of California, Berkeley
† International Computer Science Institute, Berkeley
grier@icsi.berkeley.edu, {christian,vern}@icir.org
§ VU University Amsterdam, The Network Institute
{herbertb,steen}@cs.vu.nl

Abstract—Malware researchers rely on the observation of malicious code in execution to collect datasets for a wide array of experiments, including generation of detection models, study of longitudinal behavior, and validation of prior research. For such research to reflect prudent science, the work needs to address a number of concerns relating to the correct and representative use of the datasets, presentation of methodology in a fashion sufficiently transparent to enable reproducibility, and due consideration of the need not to harm others.

In this paper we study the methodological rigor and prudence in 36 academic publications from 2006–2011 that rely on malware execution. 40% of these papers appeared in the 6 highest-ranked academic security conferences. We find frequent shortcomings, including problematic assumptions regarding the use of execution-driven datasets (25% of the papers), absence of description of security precautions taken during experiments (71% of the articles), and oftentimes insufficient description of the experimental setup. Deficiencies occur in top-tier venues and elsewhere alike, highlighting a need for the community to improve its handling of malware datasets. In the hope of aiding authors, reviewers, and readers, we frame guidelines regarding transparency, realism, correctness, and safety for collecting and using malware datasets.

I. Introduction

Observing the host- or network-level behavior of malware as it executes constitutes an essential technique for researchers seeking to understand malicious code. Dynamic malware analysis systems like Anubis [7], CWSandbox [50] and others [15, 21, 26, 36, 42] have proven invaluable in generating ground truth characterizations of malware behavior. The anti-malware community regularly applies these ground truths in scientific experiments, for example to evaluate malware detection technologies [2, 10, 16, 18, 23, 24, 29, 32, 44, 48, 52–54], to disseminate the results of large-scale malware experiments [5, 11, 42], to identify new groups of malware [2, 8, 38, 41], or as training datasets for machine learning approaches [19, 33, 35, 38, 40, 41, 47, 55]. However, while analysis of malware execution clearly holds importance for the community, the data collection and subsequent analysis processes face numerous potential pitfalls.

In this paper we explore issues relating to prudent experimental evaluation for projects that use malware-execution datasets. Our interest in the topic arose while analyzing malware and researching detection approaches ourselves, during which we discovered that well-working lab experiments could perform much worse in real-world evaluations. Investigating these difficulties led us to identify and explore the pitfalls that caused them. For example, we observed that even a slight artifact in a malware dataset can inadvertently lead to unforeseen performance degradation in practice.

Thus, we highlight that performing prudent experiments involving such malware analysis is harder than it seems. Related to this, we have found that the research community's efforts (including ours) frequently fall short of fully addressing existing pitfalls. Some of the shortcomings have to do with presentation of scientific work, i.e., authors remaining silent about information that they could likely add with ease. Other problems, however, go more deeply, and bring into question the basic representativeness of experimental results.

As in any science, it is desirable for our community to ensure we undertake prudent experimental evaluations. We define experiments reported in our paper as *prudent* if they are are correct, realistic, transparent, and do not harm others. Such prudence provides a foundation for the reader to objectively judge an experiment's results, and only well-framed experiments enable comparison with related work. As we will see, however, experiments in our community's publications could oftentimes be improved in terms of transparency, e.g., by adding and explaining simple but important aspects of the experiment setup. These additions render the papers more understandable, and enable others to reproduce results. Otherwise, the community finds itself at risk of failing to enable sound confirmation of previous results.

In addition, we find that published work frequently lacks sufficient consideration of experimental design and empirical assessment to enable translation from proposed methodologies to viable, practical solutions. In the worst case, papers can validate techniques with experimental results that suggest the authors have solved a given problem, but

- Rossow et al.:
  - “Prudent Practices for Designing Malware Experiments: Status Quo and Outlook”, IEEE S&P 2012
  - Things to consider when doing experiments with malware
- 18 aspects in 4 categories:
  1) Correctness of data set
     - Blending, balance, separation, Artifact treatment, …
  2) Transparency
     - Data set and experiment setup, analysis of results
  3) Realism
     - Real-world relevance, stimuli and access
  4) Safety
     - Deployment and containment

[1] https://www.christian-rossow.de/publications/guidelines-ieee2012.pdf

Fraunhofer FKIE

Approach

Our definition of requirements for a good malware corpus (focused on static analysis)

1) Representative content
2) Cross-platform orientation
3) Unpacked samples
4) Accurate labels and meta data
5) Documentation of data generation
6) Controlled curation and dissemination

Approach
Our definition of requirements: Representative content

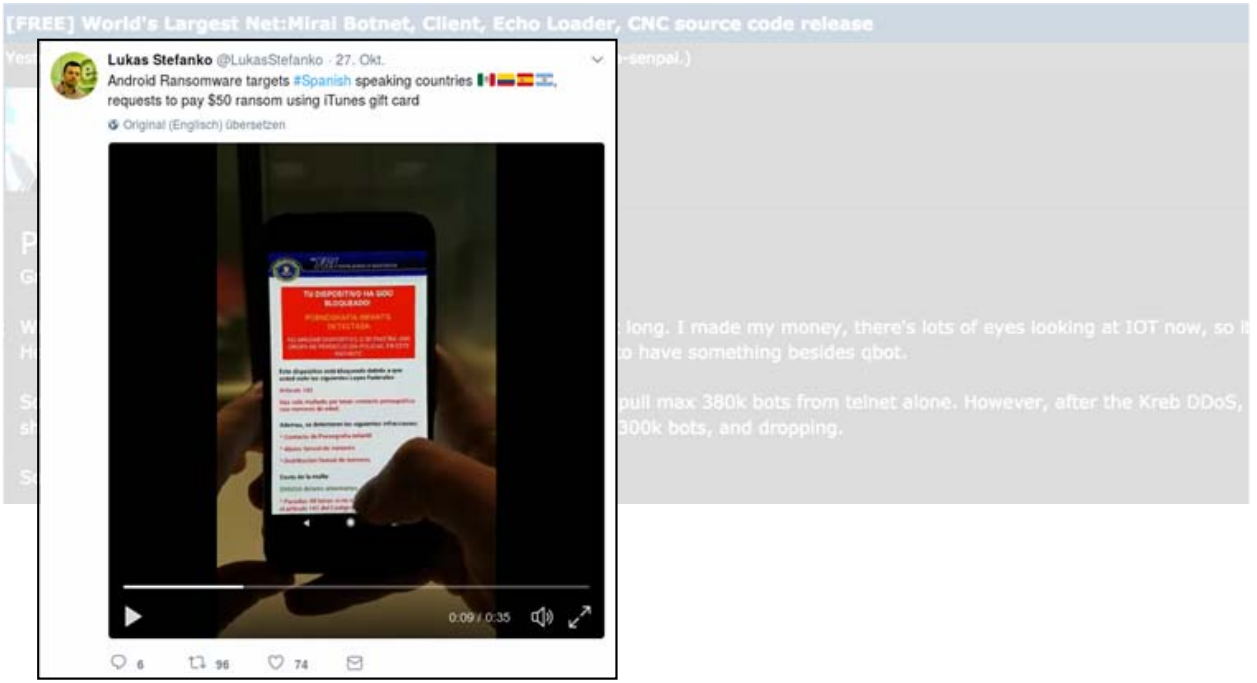
- Representative content
  - Relevant malware: aim to be useful for applied / real-world research
  - Favor quality over quantity
    - Avoid redundancy beyond the „packer barrier“

- Example: Our experiences with malware config data mining since 2012:
  - Close to 100,000 unique Citadel samples
    - But only 21 versions (despite 140+ user IDs)
    - When just interested in code -> 4,500x data reduction factor
  - TinyBanker: 5,700x, Asprox 5,500x, VMzeus 471x, KINS 105x, …

➔ Code maintenance affects us all (even the bad guys ☺)

Approach
Our definition of requirements: Cross-platform Orientation

- Cross-platform Orientation
  - It‘s 2017, let‘s be real. :)



[1] https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/
[2] https://twitter.com/LukasStefanko/status/933283910317899776

Approach
Our definition of requirements: Unpacked Samples

- Unpacked samples:
  - Essentially, enable effective static analysis

- Dumping > Unpacking?
  - Intermediate, partially unpacked stages can be omitted
  - Dumps contain additional run-time only information (decrypted strings, dynamic API usage, …)
  - Easier to automate, and can serve as a normalization

Approach

Our definition of requirements: Accurate Labels and Meta Data

- Accurate labels and meta data
  - Identification of family is the goal
  - Versions, plugins etc. are nice to have (if available)

Approach
Our definition of requirements: Documentation of Data Generation

- Documentation of data generation
  - Accountability and reproducibility
  - Track origin of samples
  - Explain dumping method and environment specifications

Approach

Our definition of requirements: Controlled Curation and Dissemination

- Controlled curation and dissemination
  - Ensure structural integrity of repository of corpus over time
  - Limit access to avoid harm

The Malpedia Corpus

How we put this into practice

The Malpedia Corpus
Overview

- Terminology
- Collection approach
- Dump creation and family identification
- Storage and organization
- Data set status

The Malpedia Corpus
Terminology

- Malware Family:
  - All samples that belong to the same project (seen from a developer's point of view)
  - Include (custom) loaders, plugins, …
  - Allows to deal with source code leaks and rewrites

The Malpedia Corpus
Terminology

- Unpacked sample:
  - Direct representative of the family itself, i.e. no third-party code present not related to ist own code (packers, protectors, …)
  - This does not include addressing de-obfuscation
  - Ideally (but not neccessary), the sample can be executed from disk

- Dumped sample:
  - Memory capture of the malware during execution
  - Will potentially contain dynamically initialized data
  - Including auxiliary modules etc. that may be found along

The Malpedia Corpus
Collection Approach

- Quality > Quantity
  - Rather extend coverage to new families than add to already inventorized families
  - Strong emphasis on verification of labels: push quality
  - Prioritization of relevant families (active, impactful, …)
- Provide means for quality assurance
  - Ideally: full, conflict-free YARA coverage
- Data Origins
  - Prefer public sources where possible, i.e. almost everything also on VT etc. (97%+)

The Malpedia Corpus
Dump Creation and Family Identification

- Currently, we (still) focus strongly on Windows
  - Fixed VM states to derive dumps from
    - Known system parameters, especially Windows API versions (upcoming: ApiScout)
  - Windows XP SP3, Windows 7 SP1 64bit
    - „But that's sooo old!“
    - Maybe, but malware feels at home there (fewer OS-embedded protection mechanisms)
  - All MSVCRT, .NET added -> maximize execution success
- Dumping
  - Execute, wait, diff memory, filter. Potentially repeat or interact manually.
  - Clean up dumps if neccessary
- Identification: run YARA or do manual investigation

The Malpedia Corpus
Storage and Organization

The Malpedia Corpus
Data Set Status (31.10.2017, commit bf6532c)

- 1792 samples inventorized into 607 families.
  - 505 families for Windows
  - 34 families for Android
  - 29 families for macOS/OSX
  - 24 families in ELF format
  - 2 families for iOS
  - 1 family for Symbian
  - 12 families that are scripted or for other reasons potentially multi-platform
- Unpacking and dumping
  - Overall 1149 (64.12%) samples are dumped (and partially also unpacked)
    - Windows: 446 (88.32%) of the families covered with one dump or more
  - 221 (12.33%) samples are just unpacked
  - 422 (23.55%) samples are neither dumped or unpacked

The Malpedia Platform

How we make this accessible

The Malpedia Platform
Overview

- Mission statement
- Implementation of trust mechanisms
- Ensurance of contribution quality

- Launch today, accessible here:
  - https://malpedia.caad.fkie.fraunhofer.de

The Malpedia Platform
Mission Statement

- „The primary goal of Malpedia is to provide a community-driven, independent resource for rapid identification and actionable context when investigating malware.
  Openness to curated contributions shall ensure topicality and an accountable level of quality in order to foster meaningful and reproducible research.“

The Malpedia Platform
Implementation of Trust Mechanisms

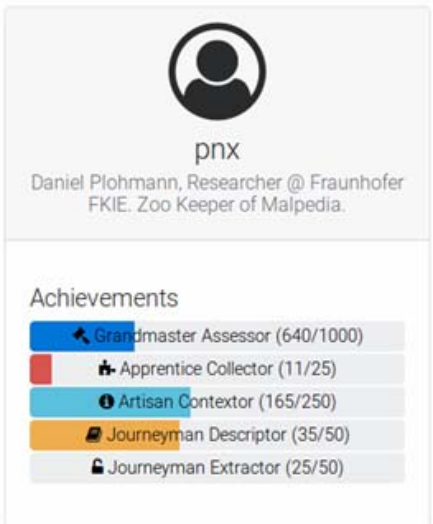
- Adoption of Traffic Light Protocol (TLP)
  - TLP:WHITE for non-critical information (family inventory, aggregated statistics, references, …)
  - TLP:AMBER for all other content (unless designated otherwise)

- Platform access granted after vetting through community members (who are accountable for you)
  - Inspired by the methods used in other trust communities such as Ops-Trust [2]

[1] https://www.first.org/tlp/
[2] https://jeroen.massar.ch/presentations/files/f41lf3st2015-Trident-Toothed-and-Pronged.pdf

The Malpedia Platform
Ensurance of Contribution Quality

- (Double-blind) peer review through volunteering users
  - 2 votes required per proposal



  - Achievements in the making!

[1] http://www.tshirtroundup.com/wp-content/uploads/2011/12/i-want-you-for-kombat.jpg

The Malpedia Platform
Summary & Features

- Baseline data set
  - ~670 families with ~2,500 samples and growing
- Contextual enrichment
  - Tracking of aliases for families
  - References to analysis reports, blog posts, media coverage, …
  - Families tied to actors, using and contributing back to the directory of MISP
- Automation support
  - REST API, including python client to access everything

[1] https://github.com/MISP/misp-galaxy/blob/master/clusters/threat-actor.json

A Comparative Structural Analysis of Windows Malware

Showcasing the data set

Comparative Structural Analysis
Overview

- Data used:
  - 31.10.2017, commit bf6532c
  - 446 families, 1208 dumped files

- Three aspects addressed:
  - PE header
  - Code (very cursory, follow-up planned)
  - Windows API usage

Families :)

7ev3n
Alphabet Ransomware
Asprox
Banatrix
Bredolab
Carberp
Client Maximus
Contopee
Crypto Ransomeware
Cuegoe
DEloader
Dorkbot
DuQu
EvilGrab
Fireball
Ghole
GrabBot
Helminth
HtBot
ISMdoor
Joao
KhRAT
Kuaibu8
LokiBot
MatrixRansom
Mocton
Nabucur
NeutrinoPOS
OpGhoul
PoisonIvy
Pykspa
Ransoc
Retefe
rtm
SedUploader
Sierras
SolarBot
SynFlooder
Thanatos
TorrentLocker
unknown_002
unknown_030
VenusLocker
XAgent
Zeus

9002 RAT
AlphaLocker
Athenago
Bart
BTCWare
Cardinal RAT
CloudDuke
CoreBot
Cryptolocker
Cutwail
Deltas
Dorshel
Duuzer
EvilLoader
FireCrypt
GhostAdmin
Graftor
Heloag
httpbrowser
iSpy Keylogger
JqjSnicker
KillDisk
Lambert
LuminosityRAT
Matsnu
Moker
Nagini
Newcore RAT
Orcus RAT
PolyglotRansom
Qadars
RapidStealer
Revenge RAT
Rurktar
SendSafe
Siggen6
Spora
SynthLoader
Threebyte
TrickBot
unknown_003
unknown_031
Virut
XbotPOS
Zeus Mailsniffer

AbbathBanker
Alphanc
ATMitch
Batel
Buhtrap
Casper
CMSbrute
CoreShell
CryptoLuck
CyberSplitter
DeputyDog
DoublePulsar
Dyre
Xtreme RAT
FlokiBot
Ghost RAT
Gratem
Herbst
Hworm
isr_stealer
JripBot
KINS
LatentBot
Lurk
Mewsei
Mokes
Naikon
NexsterBot
OvidiyStealer
Pony
QakBot
RawPOS
Rincux
Sage
Serpico
Simda
SpyBot
Sys10
ThumbThief
Trochilus RAT
unknown_005
unknown_032
VMZeus
XBTL
ZeusSphinx

Acronym
Alreay
AugustStealer
Bedep
Bundestrojaner
CCleaner Backdoor
CobaltStrike
CradleCore
CryptoMix
CyberGate
DeriaLock
DownDelph
EDA2 Ransomware
FakeRean
Flusihoc
Glasses
H1N1 Loader
Herpes
IAP
isspace
KAgent
KokoKrypt
Lazarus
Luzo
Miancha
MoleRAT Loader
Nanocore
NexusLogger
PadCrypt
PopcornTime
QuantLoader
Razy
RockLoader
Sakula RAT
ShadowPad
Sinowal
sslmm
SysGet
Tidepool
Troldesh
unknown_006
unknown_033
Vreikstadi
Xpan
ZeusSSL

AdamLocker
AMTsol
Aveo
BetaBot
Bunitu
Cerber
Cobian RAT
Crashoverride
Cryptorium
CycBot
Derusbi
Downeks
EhDevel
FakeTC
Fobber
Globe Ransomware
Hamweq
Hesperbot
Ice IX
jaff
Karagany
Konni
Laziok
MadMax
Micropsia
Moonwind
Necurs
Nitol
PandaBanker
Poweliks Dropper
Quasar RAT
RCS
Rofin
Sality
Shakti
Skyplex
Stabuniq
SysScan
Tinba
Trump Ransomware
unknown_008
unknown_034
Wannacry
XsPlus
ZhmMmikatz

Adylkuzz
Andromeda
Ayegent
BlackEnergy
Buzus
ChChes
Cockblocker
CredRaptor
CryptoShield
DarkComet
Devils RAT
DownRage
Elise
Fanny
Formbook
GlobeImposter
Hancitor
HiZor RAT
Idkey
jager_decryptor
KasperAgent
KoobFace
Limitail
Magala
Mimikatz
Morphine
NetRepser Keylogger
njRAT
Petrwrap
PowerDuke
Radamant
RedAlert
Rokku
Samsam
ShapeShift
Slave
StegoLoader
Teerac
TinyLoader
Tsifiri
unknown_013
Unlock92
Waterspout
Xswkit
ZLoader

AgentTesla
Apocalypse Ransomware
AzorUlt
BlackRevolution
c0d0so0
Chinad
CodeKey
Crylocker
Cryptowall
DarkPulsar
DiamondFox
Dreambot
Enfal
Fast POS
Furtim
GodzillaLoader
HappyLocker
Hiddentear
ImminentMonitor RAT
jaku
Kazuar
Kovter
Listrix
Maktub
Mirai
Moure
NetSupportManager RAT
Nymaim
Petya
PowerSniff
Ramdo
RedLeaves
RokRAT
Satana
Shifu
SmokeLoader
Strongpity
TeleBot
TinyNuke
Turnedup
unknown_020
Upatre
WinMM
XTunnel

Alice ATM
Ardamax
Babar
BlackShades
Cabart
Chir
CoinMiner
CrypMic
CryptoWire
DarkShell
Dimnie
Dridex
Erebus
Feodo
GameoverDGA
GooPic
Harnig
HighTide
Infy
jasus
Kegotip
KrBanker
LockPOS
ManameCrypt
Miuref
MultigrainPOS
NetTraveler
Oddjob
Pittytiger RAT
Prikormka
Ramnit
Remcos
Rombertik
SathurBot
ShimRAT
Snifula
SuppoBox
Tempedreve
TinyTyphon
UACme
unknown_023
Urausy
WINSloader
Yahoyah

Alina POS
Arefty
BadEncript
Bolek
CadelSpy
Chthonic
ComodoSec
Crypt0l0cker
Cryptxxxx
DarkTrack RAT
DirCrypt
Dropshot
Etumbot
FileIce Ransomware
GameoverP2P
Gozi
Havex RAT
HiKit
ISFB
Jigsaw
Kelihos
KrDownloader
Locky
Manifestus Ransomware
MM Core
Murofet
Netwire
Odinaff
Ploutus ATM
Pteranodon
Ranbyus
Remexi
Romeos
Screenlocker
Shujin
SNS Locker
Swift
Terminator RAT
TinyZBot
Uiwix
unknown_026
UrlZone
WndTest
ZeroAccess

AlmaLocker
Arik Keylogger
BadNews
Bravonc
Carbanak
Citadel
Conficker
CryptoFortress
CsExt
Daserf
DMA Locker
DualToy
EvilBunny
FinFisher
Geodo
GPCode
Hawkeye Keylogger
HLUX
ISMagent
Jimmy
Keylogger (APT3)
Kronos
LockyDecryptor
MatrixBanker
MobiRAT
Mutabaha
Neutrino
Opachki
PlugX
Pushdo
Ranscam
RemsecStrider
Roseam
Sedreco
Shylock
Socks5Systemz
SyncCrypt
TeslaCrypt
Tofsee
unknown_001
unknown_029
Vawtrak
Woolger
ZeroT

Comparative Structural Analysis

PE Header: Short Recap

```
Offset    Hexdump                                                 Text
00000000: 4D5A9000 03000000 04000000 FFFF0000 B8000000 00000000 40000000 00000000  MZ..........ÿÿ..ž.......@.......
00000020: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 C0000000  ............................À...
00000040: 0E1FBA0E 00B409CD 21B8014C CD215468 69732070 726F6772 616D2063 616E6E6F  ..º..Ž.Í!ž.LÍ!This program canno
00000060: 74206265 2072756E 20696E20 444F5320 6D6F6465 2E0D0D0A 24000000 00000000  t be run in DOS mode....$.......
00000080: 498D63B3 0DEC0DE0 0DEC0DE0 0DEC0DE0 608D6190 68886481 2D9E6283 669F2CC1  I.c³.ì.à.ì.à.ì.à`.a.h.d.-.b.f.,Á
000000A0: 52696368 0DEC0DE0 00000000 00000000 00000000 00000000 00000000 00000000  Rich.ì.à........................
000000C0: 50450000 4C010500 7CF2275A 00000000 00000000 E0000221 0B010A00 00000100  PE..L...|ò'Z........à..!........
000000E0: 00000200 00000000 CAFE0100 00100000 00200200 00000010 00100000 00020000  ........Êþ....... ..............
00000100: 05000100 00000000 05000100 00000000 00501000 00040000 00000000 03004001  ..................P............@.
00000120: 00001000 00100000 00001000 00100000 00000000 10000000 00BD0E00 4B000000  ..........................œ..K...
00000140: <data directories>
```

1) MZ Magic	5) Machine	9) Linker Info	13) Data Directories
2) PE Magic	6) Num Sections	10) OS Required	
3) Dos String	7) Timestamp	11) SubSystem	Above Diagram may or may not include one or more eastereggs.
4) Rich Header	8) Characteristics	12) DllCharacteristics	

Comparative Structural Analysis
PE Header Availability

Field / Test	Samples		Families	
	True	False	True	False

- Let's have a look in how many cases we have a PE header at all!
  - Check for MZ, PE, and structural integrity

Comparative Structural Analysis
PE Header Availability

Field / Test	Samples		Families	
	True	False	True	False
Extractable Info	1122 (92.88%)	86 (7.12%)	422 (94.62%)	24 (5.38%)
pefile.py parsable	1111 (91.97%)	97 (8.03%)	419 (93.95%)	27 (6.05%)

- Good news: almost always a PE header available!

Comparative Structural Analysis
PE Header Availability

Field / Test	Samples		Families	
	True	False	True	False
Extractable Info	1122 (92.88%)	86 (7.12%)	422 (94.62%)	24 (5.38%)
pefile.py parsable	1111 (91.97%)	97 (8.03%)	419 (93.95%)	27 (6.05%)
MZ Magic	1111 (91.97%)	97 (8.03%)	419 (93.95%)	27 (6.05%)
PE Magic	1114 (92.22%)	94 (7.78%)	419 (93.95%)	27 (6.05%)

- Good news: almost always a PE header available!
- Even presence of header magics is quite common!

Comparative Structural Analysis
PE Header Availability

Field / Test	Samples		Families	
	True	False	True	False
Extractable Info	1122 (92.88%)	86 (7.12%)	422 (94.62%)	24 (5.38%)
pefile.py parsable	1111 (91.97%)	97 (8.03%)	419 (93.95%)	27 (6.05%)
MZ Magic	1111 (91.97%)	97 (8.03%)	419 (93.95%)	27 (6.05%)
PE Magic	1114 (92.22%)	94 (7.78%)	419 (93.95%)	27 (6.05%)

- Good news: almost always a PE header available!
- Reasons for no extractable info / no PE header:
  - 30 families: no header (18x position-independent shellcode, 7x data, 5x dynamic IAT at start)
  - 7 families: modified header (5x nulled, 2x XOR'ed)
  - *Disparity of 37 vs. 24 families as shown above: not all samples for a family may be affected this way and some are even in multiple ways.*

Comparative Structural Analysis
PE Header Availability

Field / Test	Samples		Families	
	True	False	True	False
Extractable Info	1122 (92.88%)	86 (7.12%)	422 (94.62%)	24 (5.38%)
pefile.py parsable	1111 (91.97%)	97 (8.03%)	419 (93.95%)	27 (6.05%)
MZ Magic	1111 (91.97%)	97 (8.03%)	419 (93.95%)	27 (6.05%)
PE Magic	1114 (92.22%)	94 (7.78%)	419 (93.95%)	27 (6.05%)
DOS String	1003 (83.03%)	205 (16.97%)	403 (90.36%)	43 (9.64%)

- DOS Strings:
  - “This program cannot be run in DOS mode.”
  - “This program must be run under Win32”
  - “This program must be run under Win64”

} Borland / Delphi…

Comparative Structural Analysis
PE Header Availability

Field / Test	Samples		Families	
	True	False	True	False
Extractable Info	1122 (92.88%)	86 (7.12%)	422 (94.62%)	24 (5.38%)
pefile.py parsable	1111 (91.97%)	97 (8.03%)	419 (93.95%)	27 (6.05%)
MZ Magic	1111 (91.97%)	97 (8.03%)	419 (93.95%)	27 (6.05%)
PE Magic	1114 (92.22%)	94 (7.78%)	419 (93.95%)	27 (6.05%)
DOS String	1003 (83.03%)	205 (16.97%)	403 (90.36%)	43 (9.64%)
Rich Header	766 (64.41%)	442 (36.59%)	272 (60.99%)	174 (39.01%)

- Rich Header
  - Proprietary standard
  - Only available for Microsoft Visual Studio
  - Recently covered in detail by Webster et al. [1]

[1] https://www.sec.in.tum.de/assets/Uploads/RichHeader.pdf

Comparative Structural Analysis
PE Header Fields

Field / Test	Samples (86 (7.12%) not applicable)		Families (24 (5.38%) not applicable)	
	True	False	True	False

- Next, let's have a look at some selected values for 422 families (i.e. all that have extractable headers)

Comparative Structural Analysis
PE Header Fields: Bitness and Type

Field / Test	Samples (86 (7.12%) not applicable)		Families (24 (5.38%) not applicable)	
	True	False	True	False
32bit	1117 (92.47%)	5 (0.41%)	421 (94.39%)	1 (0.22%)
DLL	341 (28.23%)	781 (64.65%)	120 (26.91%)	302 (67.71%)

- Vast majority of malware in the current state of the data set is 32bit
  - Partially a result of me having used 32bit Windows for dumping a lot.
  - Unpacked files also contain 64bit modules for 15 families.
- More than every 4th sample / family has its main modules as a DLL!

Comparative Structural Analysis
PE Header Fields: Security

Field / Test	Samples (86 (7.12%) not applicable)		Families (24 (5.38%) not applicable)	
	True	False	True	False
32bit	1117 (92.47%)	5 (0.41%)	421 (94.39%)	1 (0.22%)
DLL	341 (28.23%)	781 (64.65%)	120 (26.91%)	302 (67.71%)
SafeSEH	939 (77.73%)	183 (15.15%)	326 (73.09%)	96 (21.52%)
NX	529 (43.79%)	593 (49.09%)	234 (52.47%)	188 (42.15%)
ASLR	663 (54.88%)	459 (38.00%)	257 (57.62%)	165 (37.00%)
NX+ASLR	475 (39.32%)	647 (53.56%)	219 (49.10%)	203 (45.52%)

- SafeSEH (whitelisted exception handler functions) quite common
- NX+ASLR not so much.

Comparative Structural Analysis
PE Header Fields: Timestamps

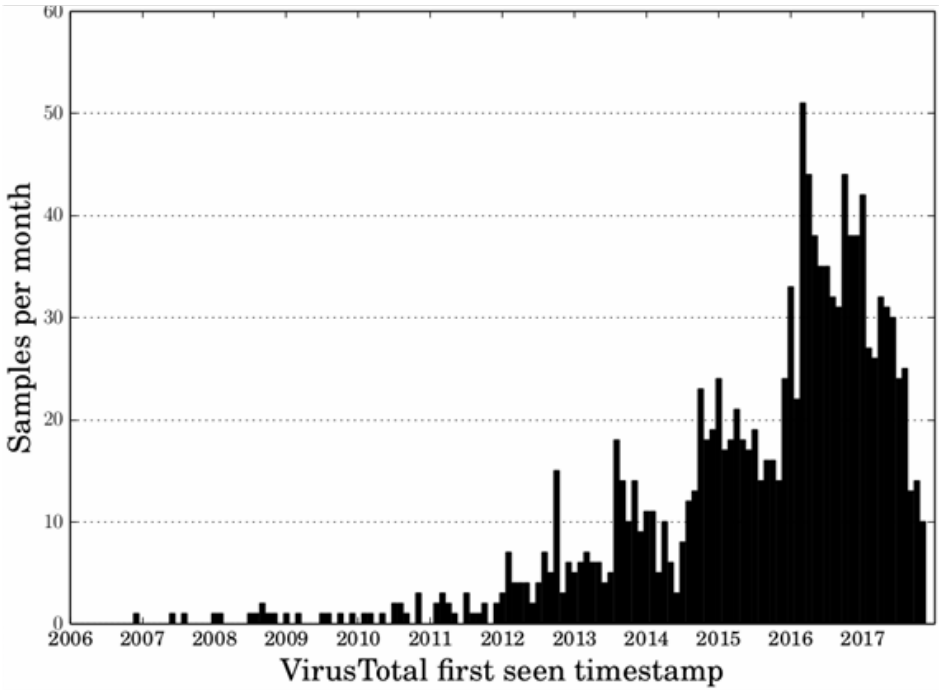
Field / Test	Samples (86 (7.12%) not applicable)		Families (24 (5.38%) not applicable)	
	True	False	True	False
32bit	1117 (92.47%)	5 (0.41%)	421 (94.39%)	1 (0.22%)
DLL	341 (28.23%)	781 (64.65%)	120 (26.91%)	302 (67.71%)
SafeSEH	939 (77.73%)	183 (15.15%)	326 (73.09%)	96 (21.52%)
NX	529 (43.79%)	593 (49.09%)	234 (52.47%)	188 (42.15%)
ASLR	663 (54.88%)	459 (38.00%)	257 (57.62%)	165 (37.00%)
NX+ASLR	475 (39.32%)	647 (53.56%)	219 (49.10%)	203 (45.52%)
Timestamp	1060 (87.75%)	62 (5.13%)	398 (89.24%)	24 (5.38%)

- For almost 90%, the timestamp is set to „not 0“ or „Delphi 4 Time“ (0x2A425E19 -> 1992-06-19 22:22:17) [1]
  - Next up: we look at timestamps in some more detail

[1] http://www.hexacorn.com/blog/2014/12/05/the-not-so-boring-land-of-borland-executables-part-1/

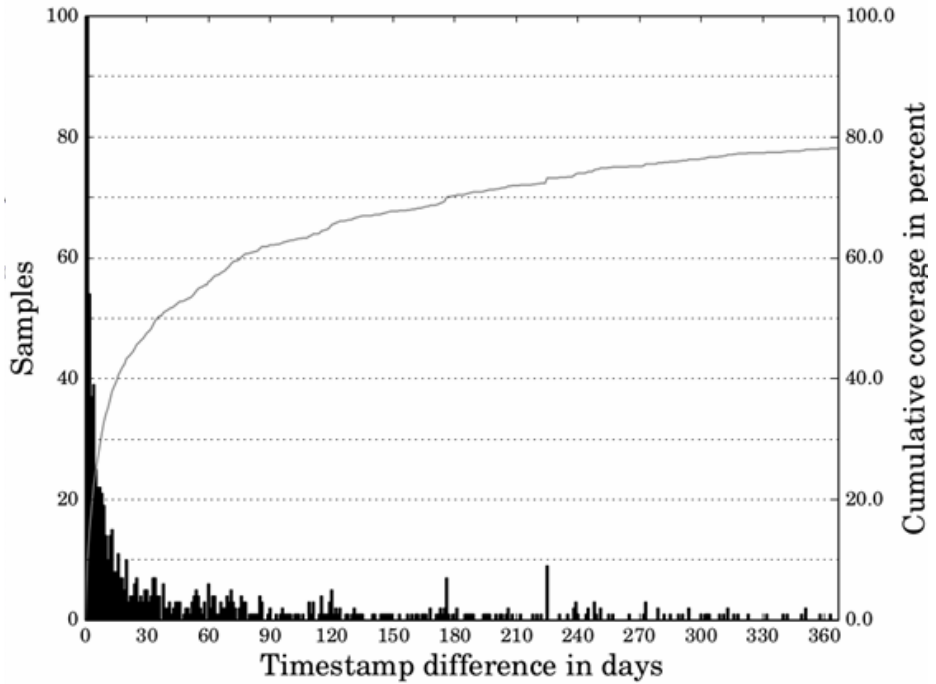
Comparative Structural Analysis
PE Header Fields: Timestamps

- „Age“ of Malpedia corpus
- 1173 / 1208 (97.10%) of files are on VirusTotal and thus have a first seen date.
  - 91.65% of files in the data set first seen between 2013 and 2017.
  - Oldest file: 2006-12-11 (a sample of Gozi)



Comparative Structural Analysis
PE Header Fields: Timestamps

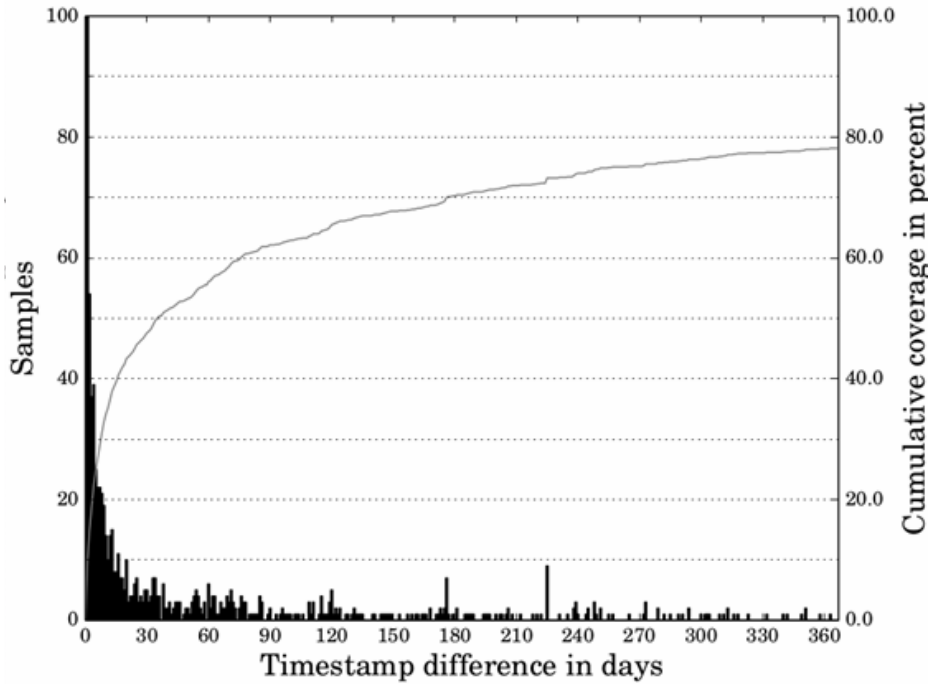
- Estimating PE timestamp accuracy
- 1208 dumps -> filtering
  - 1122 (86 without PE header)
  - 1060 (62 with null or Delphi Timestamp)
  - 1030 (30 without VT timestamp)
  - 992 (38 where VT < PE)
  - 949 (2006-12-11 < Timestamp < now)
- 949 candidate timestamp pairs, 367 (82.29%) families!
  - 10.43% within one day
  - 32.24% within one week
  - 48.78% within one month
  - 78.30% within one year



Comparative Structural Analysis
PE Header Fields: Timestamps

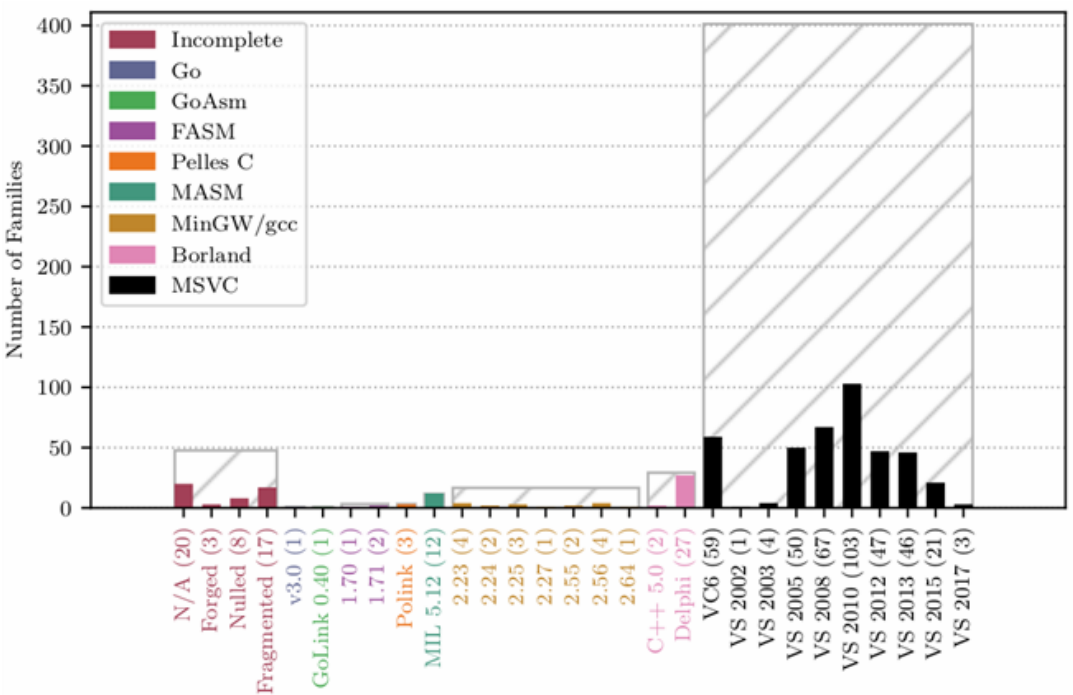
- Estimating PE timestamp accuracy: outliers

- 206 samples with more than 1 year difference:
  - 75 (47 families) with APT background
  - 59 (12 families) with (leaked) builder
  - 26 (5 families) with forgery across header fields
  - 46 (36 families) not trivially explained



Comparative Structural Analysis
PE Header Fields: Linker / Rich Header

- Using Detect-It-Easy [1] as reference
- De-duplication on family level: 513 values for 422 families
- MSVC is absolutely dominant
  - VC6 was released 1998! O_o
  - However, VC6 msvcrt.dll is default on Windows.
- Rich Header: 766 samples
  - 735 (95.95%) have MSVC linker info
  - 721 (98.10%) have matching PID.



[1] https://github.com/horsicq/Detect-It-Easy

Fraunhofer FKIE

Comparative Structural Analysis
PE Header Fields: Data Directories

- Not presented here, information in the paper.

Comparative Structural Analysis
Code Analysis: Control Flow Graphs

	min	25%	50%	75%	max	Mean
Functions	3	181.25	438.50	1,107.00	26,360.00	1,300.92
BBlocks	25	1,437.25	4,261.83	9,765.78	337,386.50	10,954.92
Instructions	30	9,362.75	23,833.62	54,632.33	1,848,787.00	63,390.37
Function Calls	2	446.54	1377.08	3,593.75	113,008.50	4,582.13

- Results obtained with in-house recursive disassembler (to be released)
- Pearson Correlation Coefficient shows notable (linear) correlation between these values

Comparative Structural Analysis
Code Analysis: PDB Information

- Microsoft's proprietary standard for storing meta information used to enrich debugging
  - 163 samples of 111 families (24.98%) have PDB information
- 32 contain non-default user name paths (i.e. != „user", „Administrator", …)
- 49 references that may indicate author's naming choice
  - 40 of them are directly tied to this family was referred to in analysis reports

Comparative Structural Analysis
Windows API Usage Analysis: Overview

- Tool: ApiScout [1]
  - Library for painless (Windows) API reconstruction in known environments
- API Information Availability
  - Usage Styles (Import Table, Dynamic Loading, Obfuscation)
- DLL and API Usage Frequencies

[1] https://github.com/danielplohmann/apiscout

Comparative Structural Analysis
Windows API Usage Analysis: ApiScout

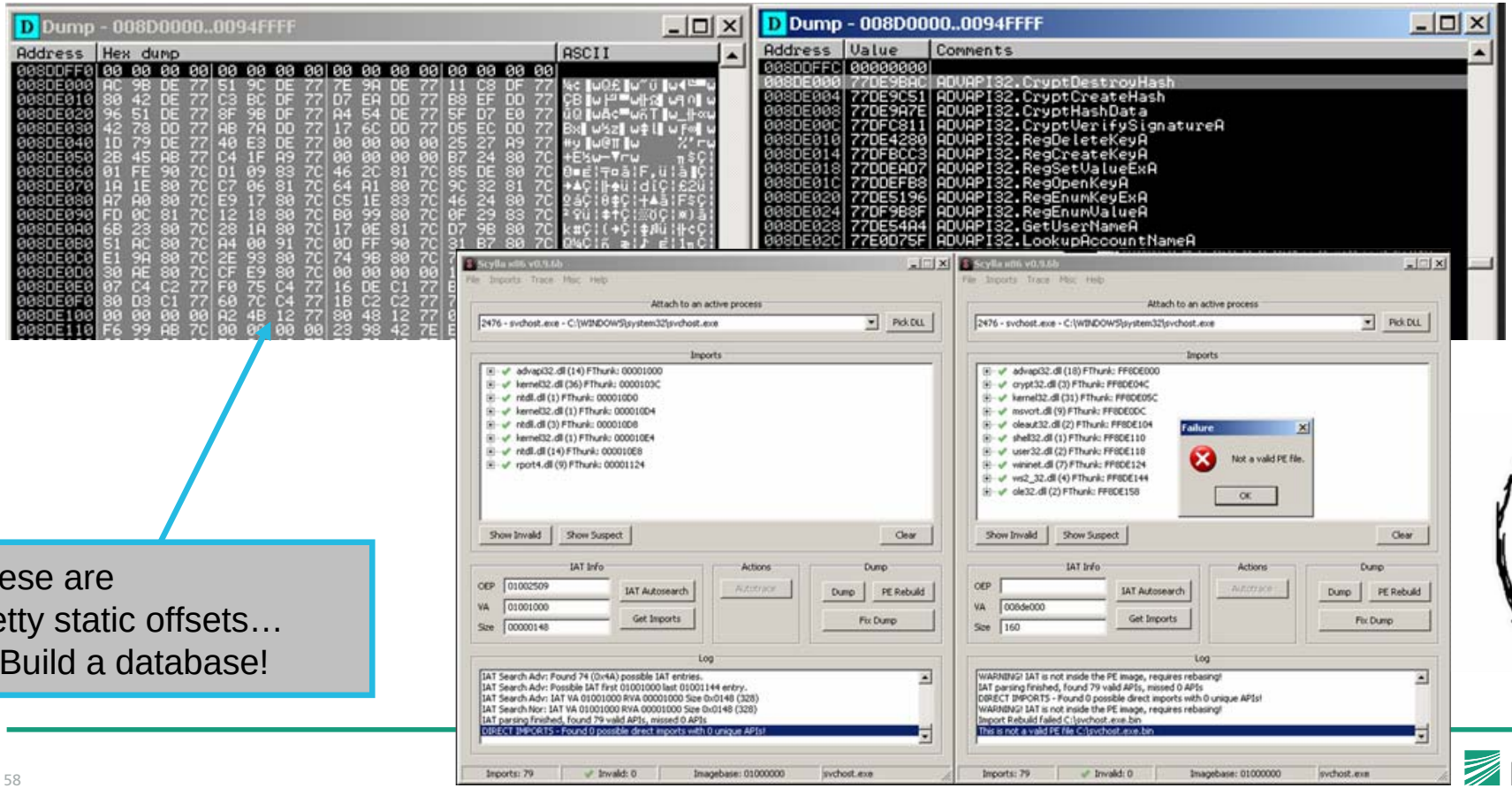
- Tool: ApiScout [1]
  - Library for painless (Windows) API reconstruction in known environments

- Idea: API function offset bruteforcing based on databases

[1] https://github.com/danielplohmann/apiscout

Comparative Structural Analysis

Windows API Usage Analysis: ApiScout

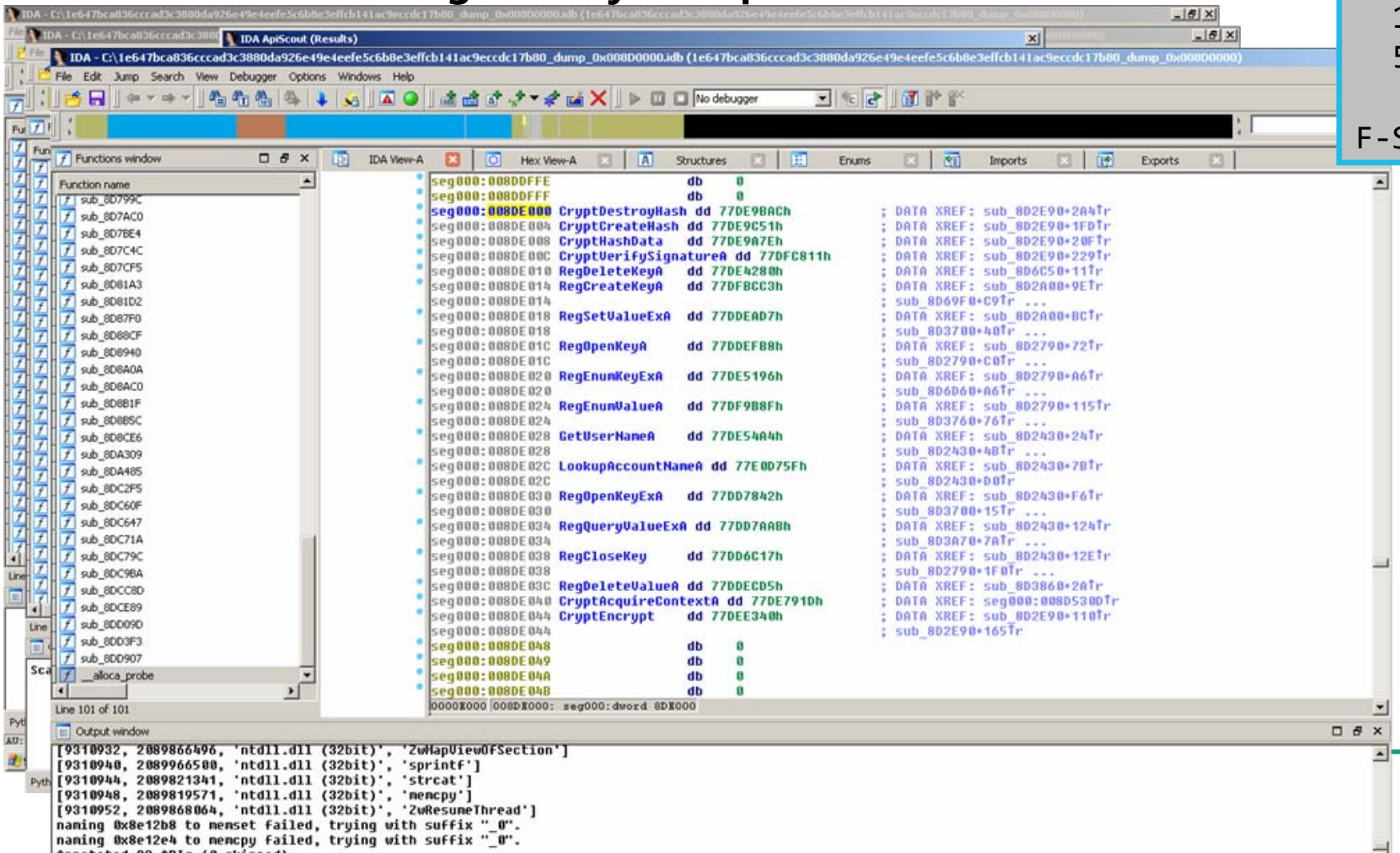


These are
pretty static offsets…
-> Build a database!

Fraunhofer FKIE

Comparative Structural Analysis

Windows API Usage Analysis: ApiScout



Accuracy Evaluation:

Sample:
- 15 Windows Bins
- 5,367 API Imports

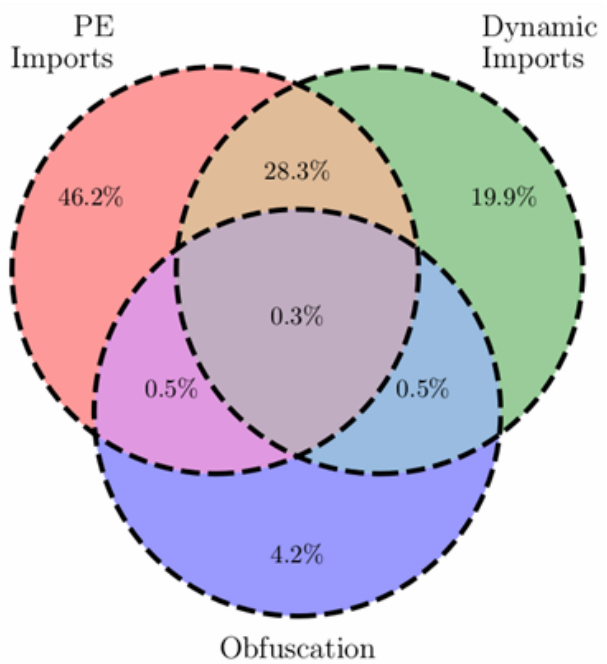
F-Score: 0.995

Currently not IDA 7.0 compatible :(

Fraunhofer FKIE

Comparative Structural Analysis

Windows API Usage Analysis: API Information Availability



- Across 382 families *(40 ignored -> .net)*
- PE Imports:
  - From PE Header Import Table only
- Dynamic:
  - LoadLibrary / GetProcAddress
    ApiHash -> Cache
- Obfuscation:
  - Custom Jump Table *(Andromeda)*
    Offset-based Hook Avoidance *(Chthonic)*
    On-Demand Table *(Dridex)*
    Dynamic Resolving *(Shifu)*
    Imports on Stack / Heap *(PIVY, Cryptowall)*
    XORed Imports *(Qadars)*

Covered by ApiScout [1]

[1] https://github.com/danielplohmann/apiscout

Fraunhofer FKIE

Comparative Structural Analysis
Windows API Usage Analysis: API Information Availability

- Quiz: What are the most common API calls / DLLs used in malware? :)

1) kernel32.dll!Sleep	330 (86.39%)
2) kernel32.dll!CloseHandle	326 (85.34%)
3) kernel32.dll!GetModuleHandle	323 (84.55%)
4) kernel32.dll!CreateFile	314 (82.20%)
5) kernel32.dll!WriteFile	312 (81.68%)
6) kernel32.dll!GetProcAddress	312 (81.68%)
7) kernel32.dll!GetModuleFileName	307 (80.37%)

1) kernel32.dll	363 (95.03%)
2) ntdll.dll	352 (92.15%)
3) advapi32.dll	302 (79.06%)
4) user32.dll	293 (76.70%)
5) shell32.dll	220 (57.59%)
6) ws2_32.dll	206 (53.93%)
7) wininet.dll	161 (42.15%)

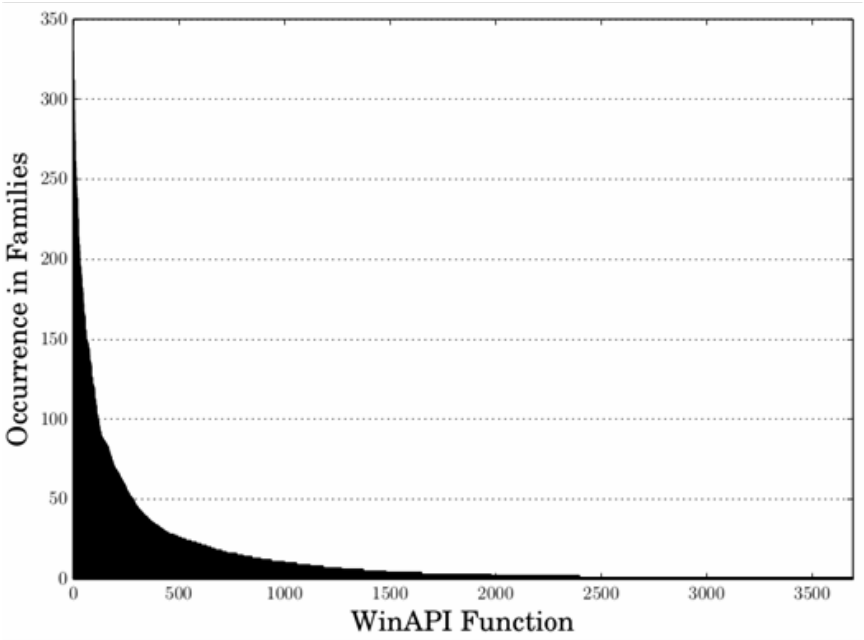
0-592 APIs, mean: 150.26, med: 122.25
(per family)

About 3.700 of 50.000 Indexed APIs observed being used

0-24 DLLs, mean: 8.32, med: 8

Comparative Structural Analysis
Windows API Usage Analysis: API Information Availability

- Occurrence frequency per Windows API function



- Discounting .NET and API-obfuscated families:
  - Only 2 API functions > 90%
  - Only 44 API functions > 50%
  - API function at position 150 appears in 23.89%
  - 3,320 (89.90%) of API functions <= 10%
- Observation: API compositions are highly specific per family
  - Favorable for tools like ImpHash [1], ImpFuzzy [2], …

[1] https://www.fireeye.com/blog/threat-research/2014/01/tracking-malware-import-hashing.html
[2] http://blog.jpcert.or.jp/2017/03/malware-clustering-using-impfuzzy-and-network-analysis---impfuzzy-for-neo4j-.html

Comparative Structural Analysis
Windows API Usage Analysis: API Context Groups

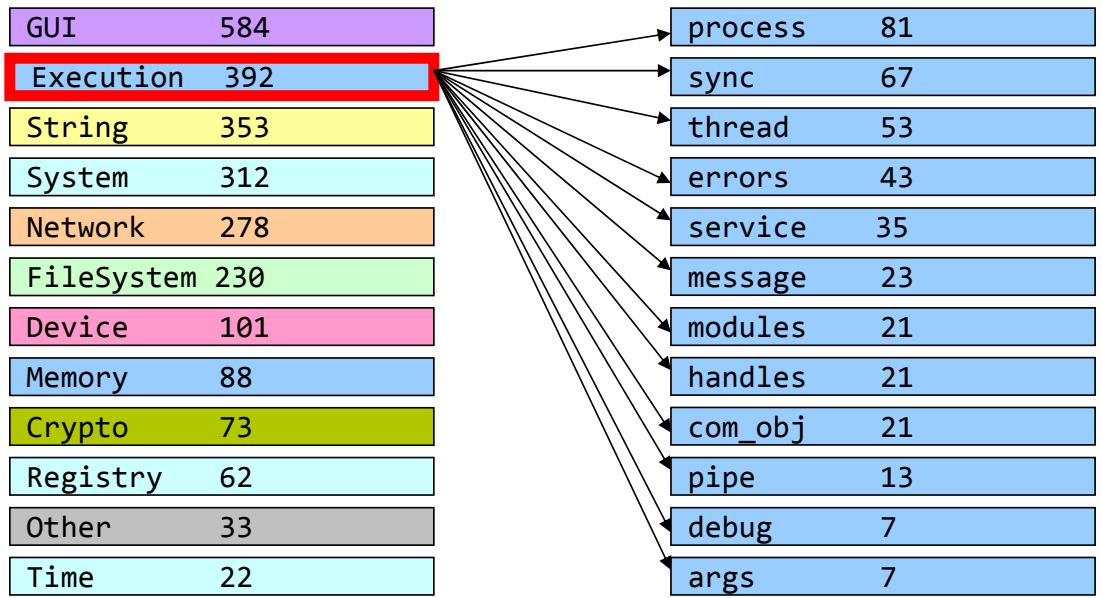
- A better way to approach this: API Context Groups
  - Manually labelled ~3.000/3.500 APIs, primary (12) and secondary class (77)

GUI	584
Execution	392
String	353
System	312
Network	278
FileSystem	230
Device	101
Memory	88
Crypto	73
Registry	62
Other	33
Time	22

Comparative Structural Analysis

Windows API Usage Analysis: Api Context Groups, Example: Execution

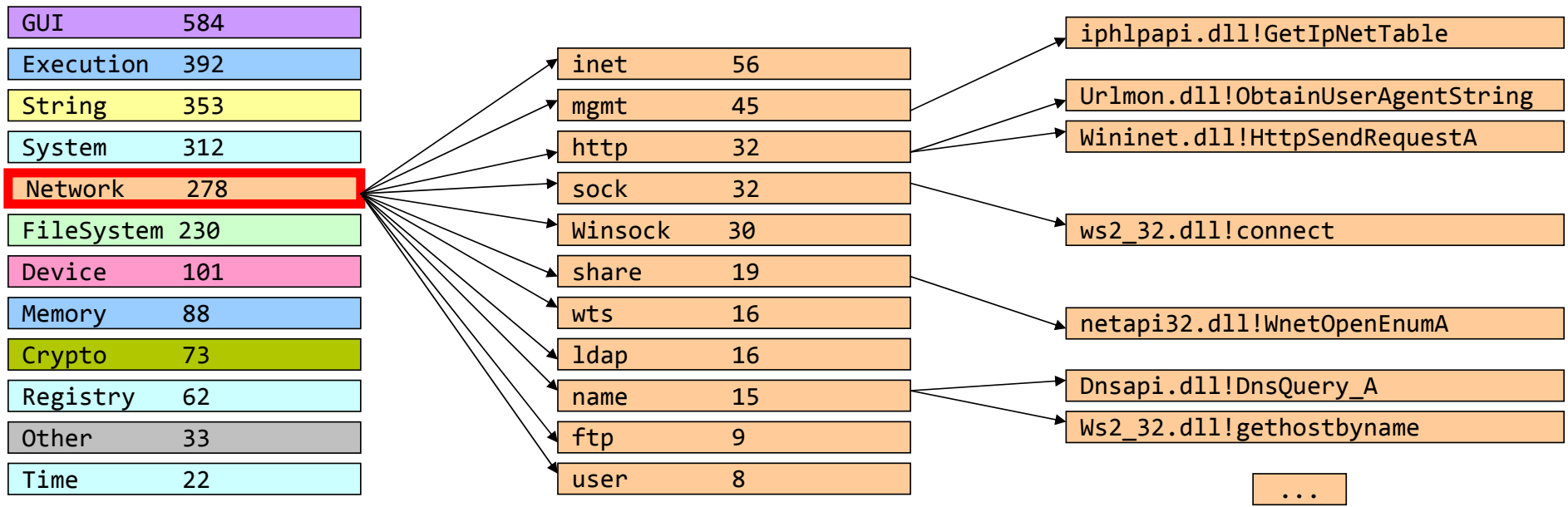
- A better way to approach this: API Context Groups
  - Context: Execution



Comparative Structural Analysis

Windows API Usage Analysis: Api Context Groups, Example: Network

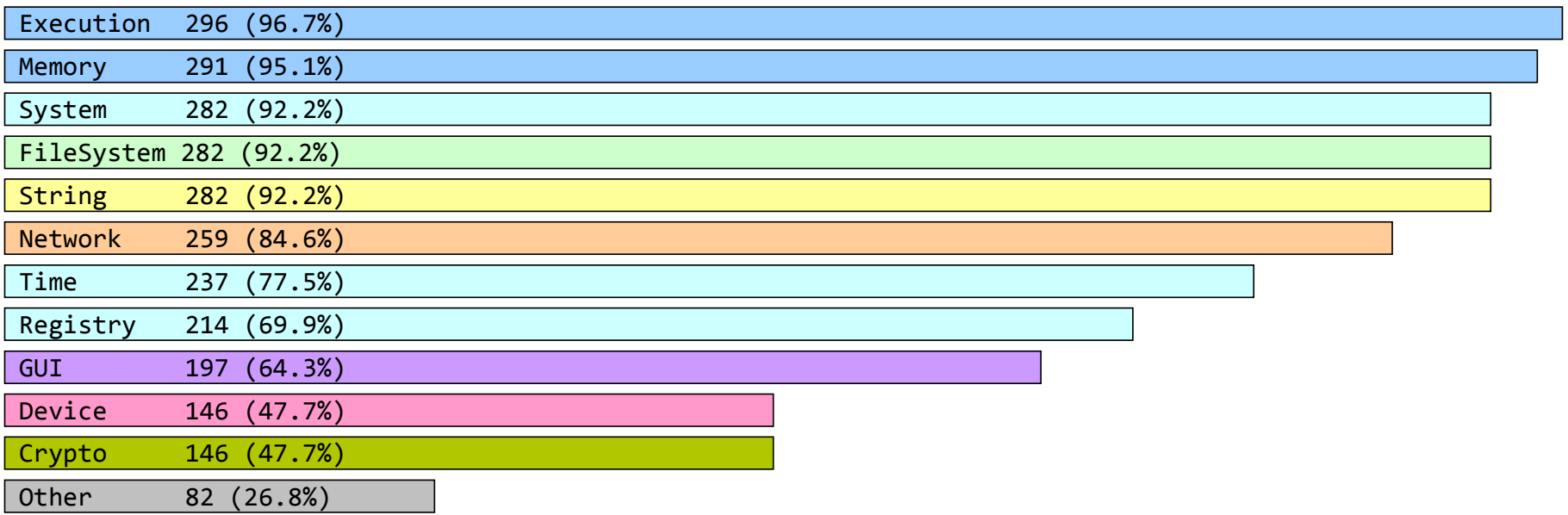
- A better way to approach this: API Context Groups
  - Context: Network



Fraunhofer FKIE

Comparative Structural Analysis
Windows API Usage Analysis: Api Context Groups, Frequencies

- Appearance of context groups across all families



Future Plans / Conclusion

Future Plans
Malpedia

- Corpus / Platform
  - Maintain data set and extend coverage
  - Get some users / activity going – join us for the greater good!
  - Integrate further services around identification

- Research
  - We only scratched the surface…
  - Code/Function indexation: unique vs. shared code, robust means for identification, …
  - Diving deep on functionality carried by the code

Conclusion
Malpedia

- **Vision: A free, curated, high-quality malware corpus for research**
  - Vetted + community-driven effort
- Public launch is today. Yay!
  - https://malpedia.caad.fkie.fraunhofer.de
- Want Access?
  - Talk to me (Know Met Trust (KMT) -> ensures K&M already)
  - Get an invite by another existing member that can vouch for you

Thank You for Your Attention!

Daniel Plohmann
daniel.plohmann@fkie.fraunhofer.de


@push_pnx
@malpedia

Fraunhofer
FKIE