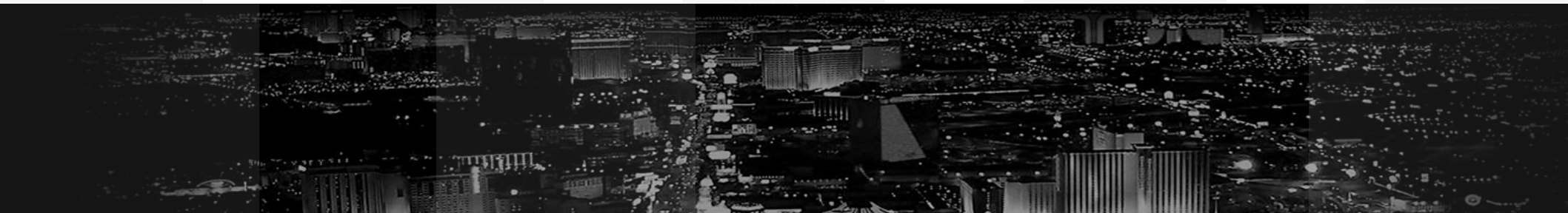
Prevalent Characteristics in Modern Malware

Gabriel Negreira Barbosa (@gabrielnb)
Rodrigo Rubira Branco (@BSDaemon)
{gabriel.negreira.barbosa || rodrigo.branco} *noSPAM* intel.com



Disclaimer

- We do work for Intel, but we do Security Validation and hardware security research (yes, we find bugs in the processor and the platform)
    - To be very honest, we don’t do anything, since there are no bugs!
- We are very hard to convince, but we are thankful to all the great discussions with the Intel Security guys!
- Mistakes, bad jokes are all **OUR** responsibilities

Disclaimer

- We do work for Intel, but we do Security Validation and hardware security research (yes, we find bugs in the processor and the platform)
  - To
- We are l to all the great d
- Mistakes, bad jokes are all **OUR** responsibilities

DON'T TRUST SOMEONE WHO JUST TROWS DATA ON YOU!!

STATISTICS ARE USUALLY MISLEADING!

SINCE THE DATA LIES, WE WILL TRY TO LIE LESS!

We are from Brazil

Respect the sysadmins!

Respect the sysadmins!

Agenda

- Introduction / Motivation
- Objectives
- Methodology (New)
- Executive Summary
- Updates from 2012 Protection Mechanisms Research
- Considered Techniques (New)
- Resources / Conclusions
- Acknowledgments

Introduction / Motivation

- Hundreds of thousands of new samples every week
- Still, prevalence data talks about only few thousands or couple millions samples
- Often, researchers use words such as ‘many’ instead of X number
- Lots of tricks exposed by researchers without data on how prevalent they are (and thus if they are worth to be used as indicator of maliciousness)
- **INDUSTRY-RELATED RESEARCH NEED RESULTS, THUS NOT PROMISING POINTS ARE NOT LOOKED AFTER**

Thanks to our sponsors!!

Before continue, some definitions ...

- Anti-Debugging
  - Techniques to compromise debuggers and/or the debugging process
- Anti-Disassembly
  - Techniques to compromise disassemblers and/or the disassembling process
- Obfuscation
  - Techniques to make the signatures creation more difficult and the disassembled code harder to be analyzed by a professional
- Anti-VM
  - Techniques to detect and/or compromise virtual machines
- Malicious Technique
  - A characteristic we look for in the scope of this research. Not necessarily all software using such technique is malicious

Objectives

- Analyze millions of malware samples
- Share the current results related to:
  - Anti-Debugging (updated from 2012 presentation)
  - Anti-Disassembly (updated from 2012 presentation)
  - Obfuscation (updated from 2012 presentation)
  - Anti-VM (updated from 2012 presentation)
  - Other techniques that are interesting (or not) (new for 2014)
- Keep sharing more and better results with the community
  - More samples analyzed compared to 2012 (12 millions x 4 millions)
  - More than 50 new techniques not related to Anti-RE (new for 2014)
  - Propose improvements in the malware selection methodology

Project Decision

- Simplify everything
  - We coded everything from scratch
  - We put together the detection algorithms for better performance (less I/O)
  - Output goes to the disk, parse results later

- Some numbers, just for the LOL ;)
  - 72 + hours just to parse the results (complications such as function and section lists) -> We will release raw data on that in the future
  - 10 + days running with full capacity to analyze the 12 million samples (considering that many of them got just ignored – discussed later in the methodology)
  - Not easy to ask more questions to the data, but easy to implement, simple enough to anyone to implement -> We limited our data because we forgot some considerations while writing the parsers

Methodology

- Used a total of 80 cores and 192 GB of memory
- Analyzed only 32-bit PE samples from public sources
- Packed samples:
  - In the previous research, different samples using the same packer were counted as 1 unique sample (and analyzed once)
  - Analyzed all packers present among the 12 million samples
    - Dataset comprises the original 4 million samples (after clearing it with feedbacks from McAfee – it did not change the previous statistics though)
- Size limitation of samples:
  - Removed the previous limitation of not analyzing samples bigger than 3,9 MB for performance reasons (with some exceptions such as the Flame Malware) – This could have affected the Brazilian malware results (keep watching)

Methodology - Dataset cleaning

- In 2012 we cleaned the dataset of packers using specific detections and heuristics (data section size x code section size, sections existence and entropy levels)
- McAfee researchers gave the idea of looking for similarities in code sections
  - We came up with an algorithm for that to be sure the previous dataset was not biased (for the new one, we wanted raw numbers, so we did not need it)
    - Extract disassembly of each executable section of each malware
    - Normalize addresses
    - Generate sha1 and search for existence in the database
- Some samples were marked by them as known-good
  - Not enough to affect any of the numbers (around 23 thousand)
  - We eliminated those samples just in case

Methodology

- Static analysis:
  - Main focus of this presentation
  - Improves the throughput (with well-written code) – Results possible to be applied in network in-lined solutions
  - Not detectable by malware
  - Has lots of limitations
- Dynamic counter-part:
  - It is not viable to statically detect everything
  - We expect the numbers on the prevalence to grow if dynamic (and more complete) versions of the detection algorithms are implemented
  - We did not cover this in our research

Methodology

- Malware protection techniques in the previous work:
  - State-of-the-art papers/journals
  - Malware in the wild
  - All techniques were implemented even when there were no public examples of it (github)
- Malware techniques in this work:
  - Coverage of techniques not previously analyzed (standalone implementation instead of using the previous platform)
  - Tricks seem during this 2 year period by different malicious samples (we show the prevalence of the trick in our dataset, we don’t discuss if the existence of it is enough to define maliciousness)
  - Usage of standalone detection mechanisms to avoid previous size limitations
  - Triage of familiarities in malware code to avoid bias in favor of specific malware – around a billion basic blocks to analyze, future results to be released after the conference and not considered in the presented data

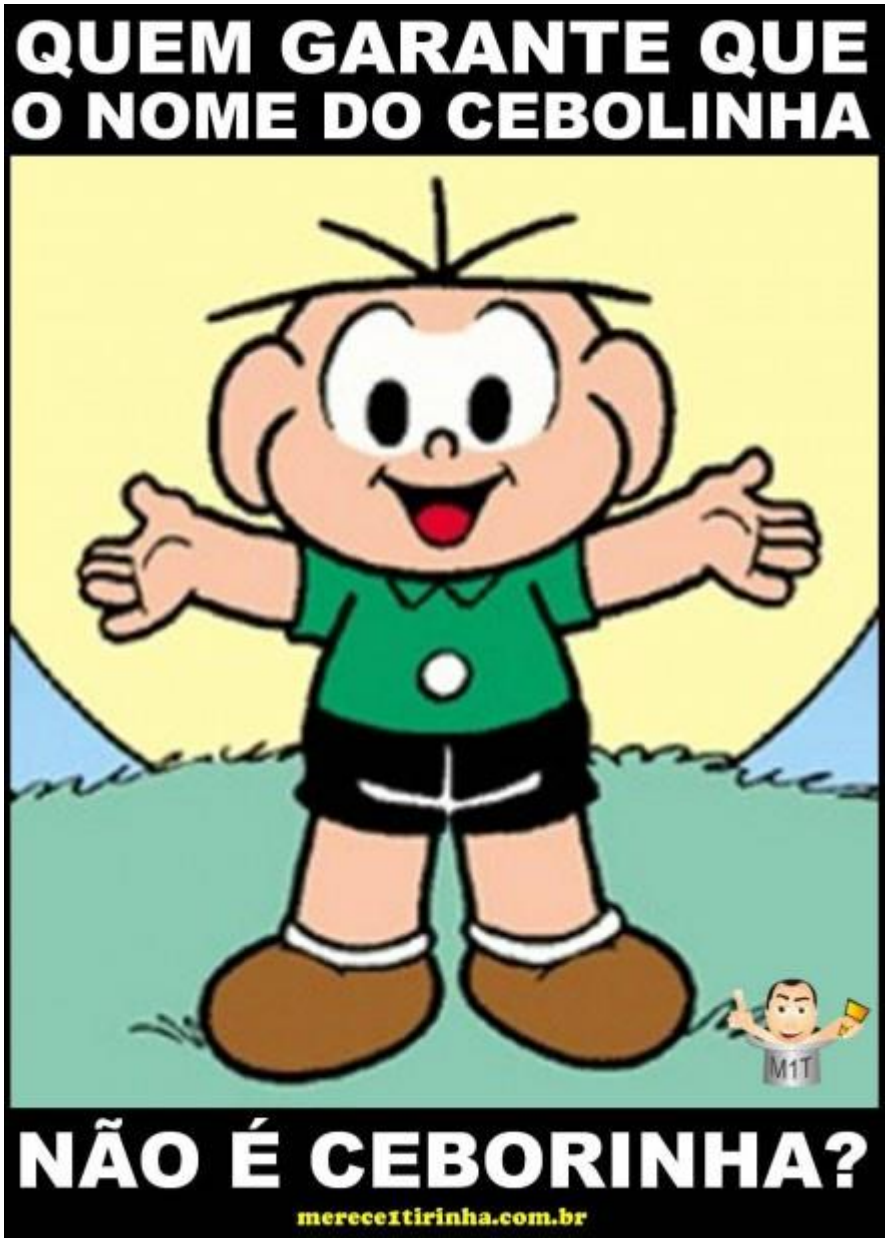
Methodology

- Possible techniques detection results:
  - Detected:
    - Our algorithm detected the presence of the technique
  - Not detected:
    - Our algorithm did not detect the presence of the technique
  - Evidence detected:
    - Our algorithm could not deterministically detect the presence of the technique, but some evidences were found
- For the statistics, we only grouped the results:
  - Detected: We consider the technique is present
  - Not detected OR evidence detected: We consider the technique is NOT present

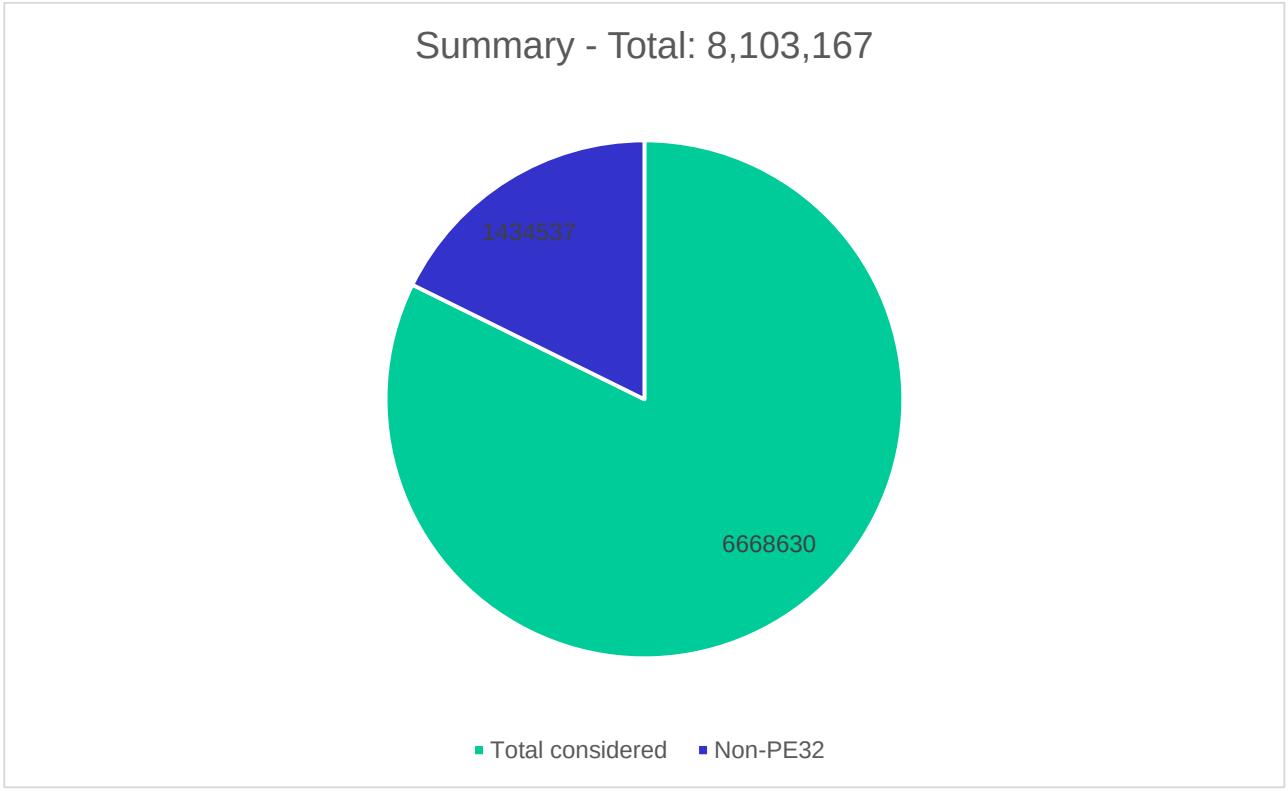
Methodology

- Analysis rely only on executable sections and in the entry point
  - Decreases the probability to analyze data as code
  - Improves the analysis time
  - For now we miss non-executable areas, even if they are referred by analyzed sections (future work?)
- Standalone code:
  - Removed the dependency of the previous platform
  - Permitted more flexibility in the research (collecting the data separately)
  - Avoided re-running the same unneeded code again in the same dataset

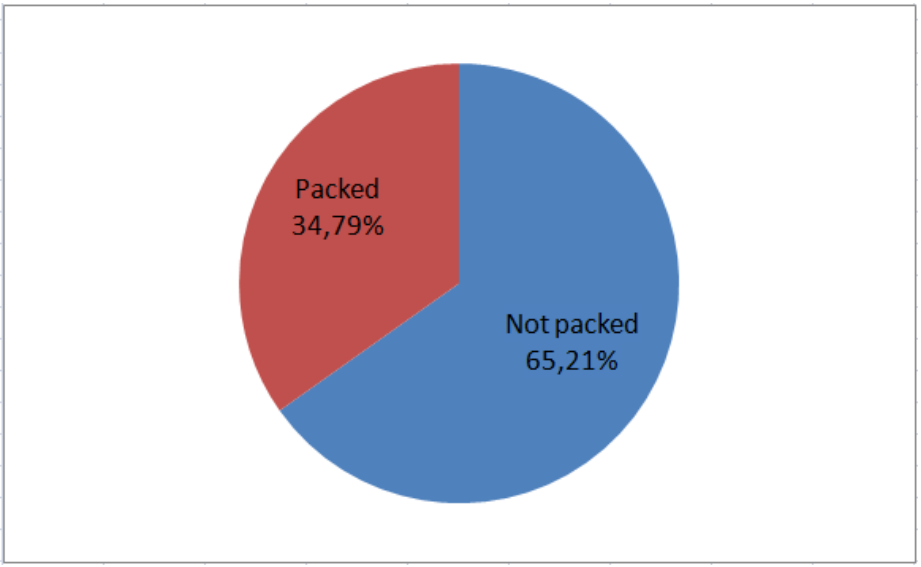
Executive Summary



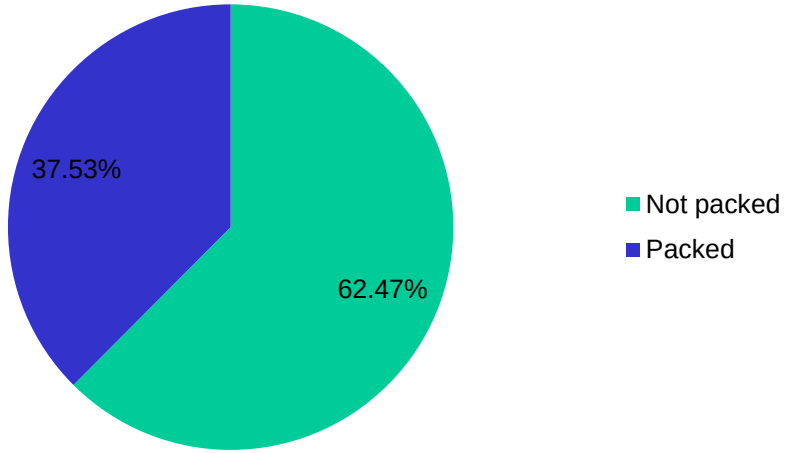
Newly analyzed samples



Packed vs Not Packed (updated)

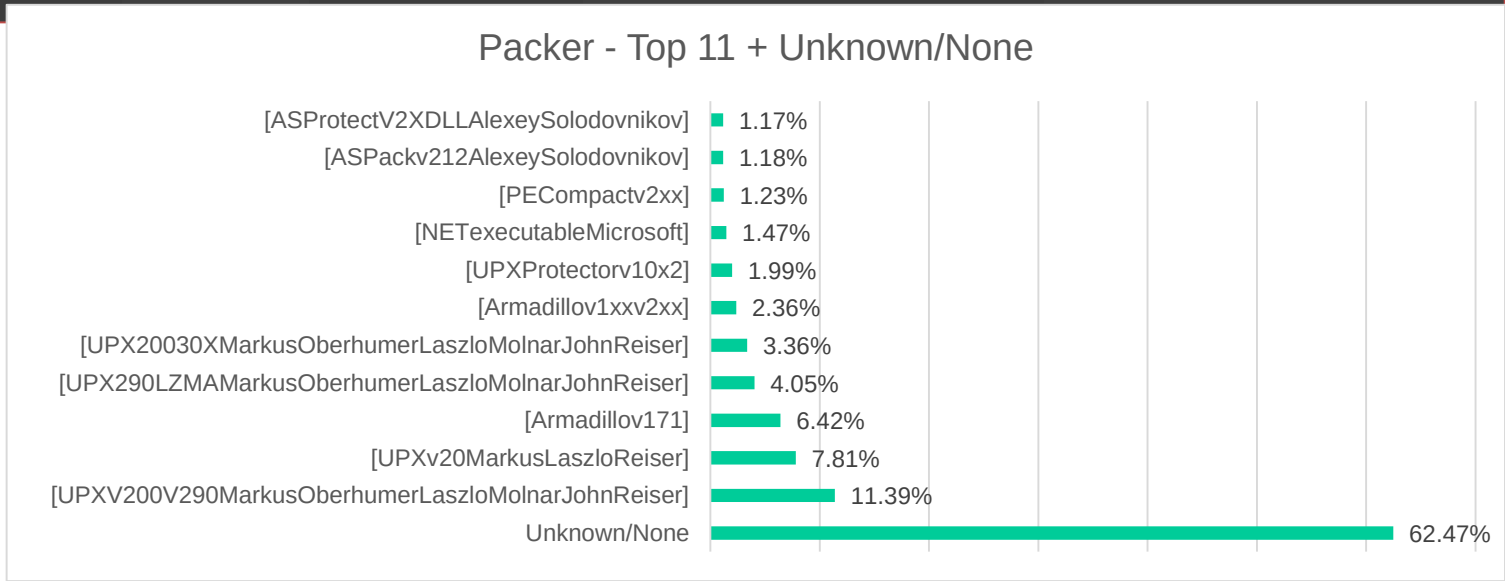


2012 results

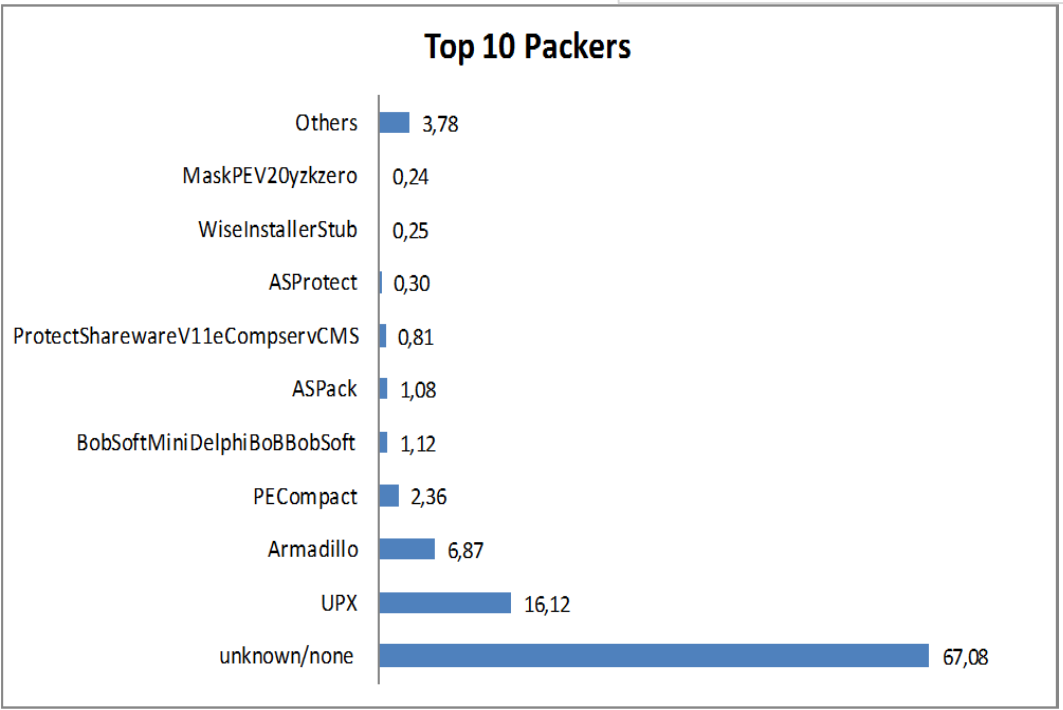


2014 results

Top Packers (updated)



2012 results



2014 results

Protecting Mechanisms of Packers (updated)

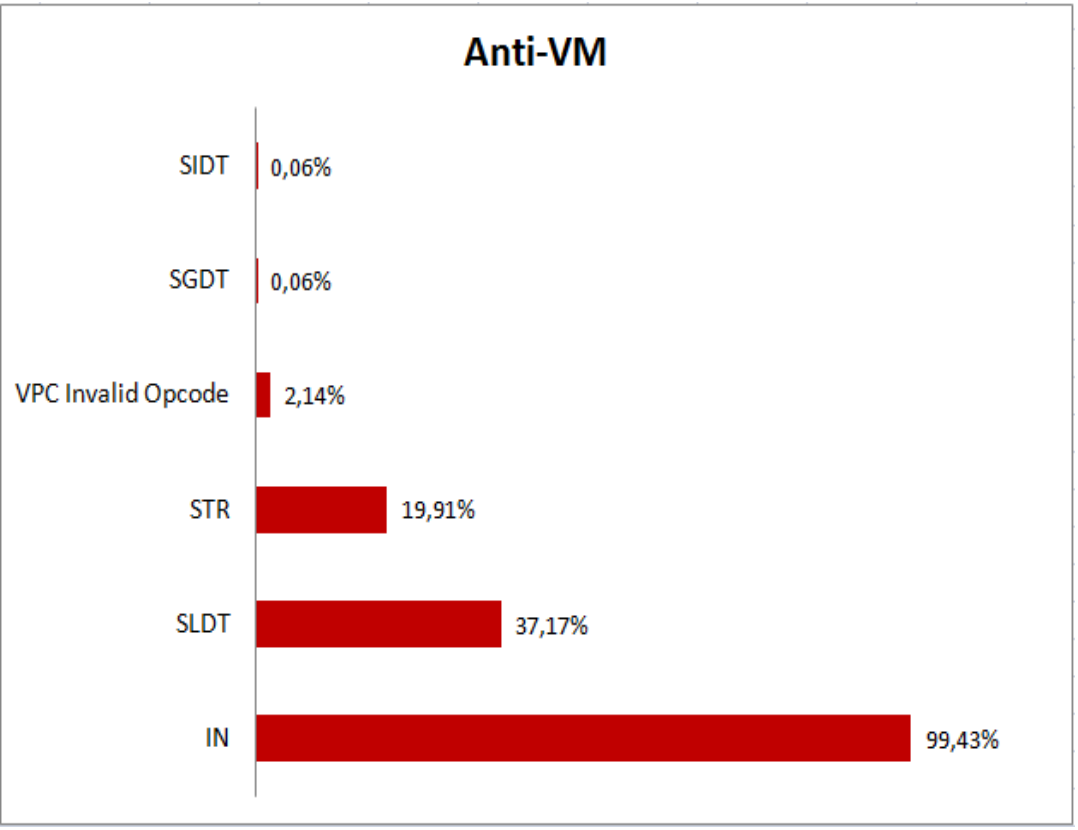
Paper (yes, we wrote one two years ago…)

…

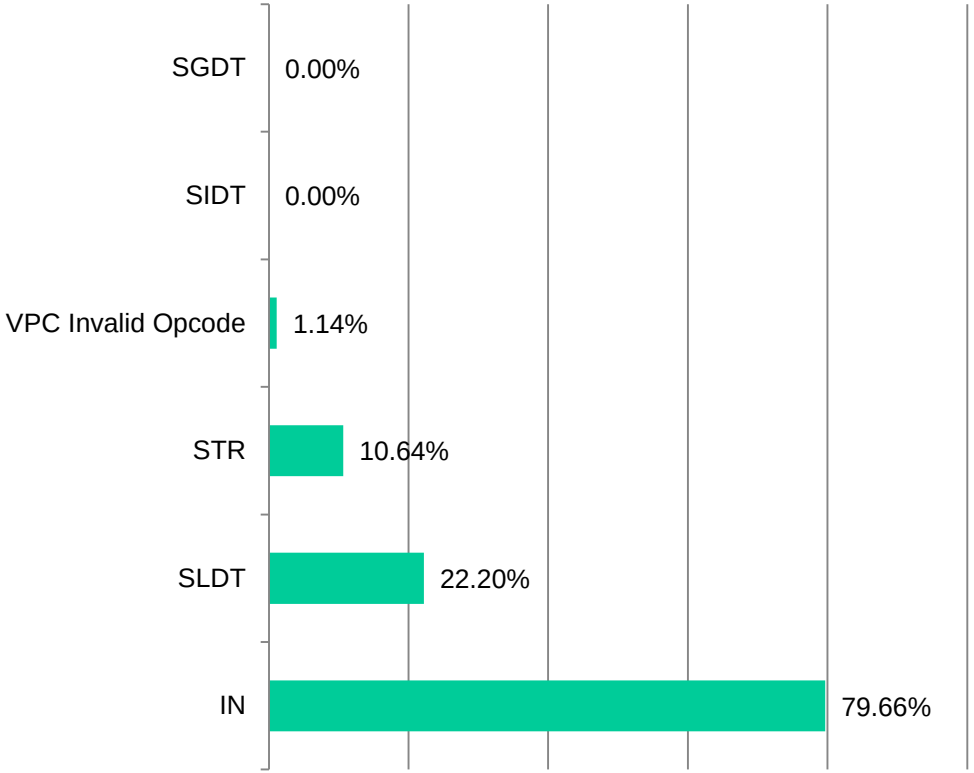
And a new one this year ;)

To be released (pending legal approvals)

Anti-VM (updated)

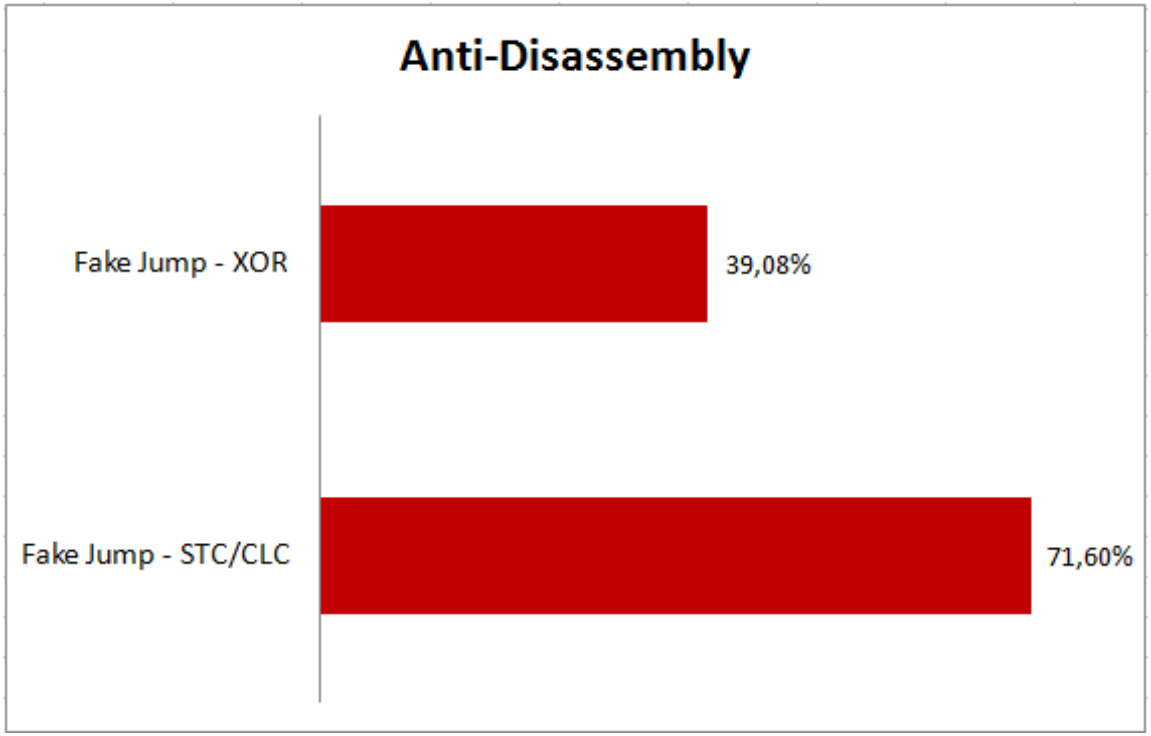


2012 results

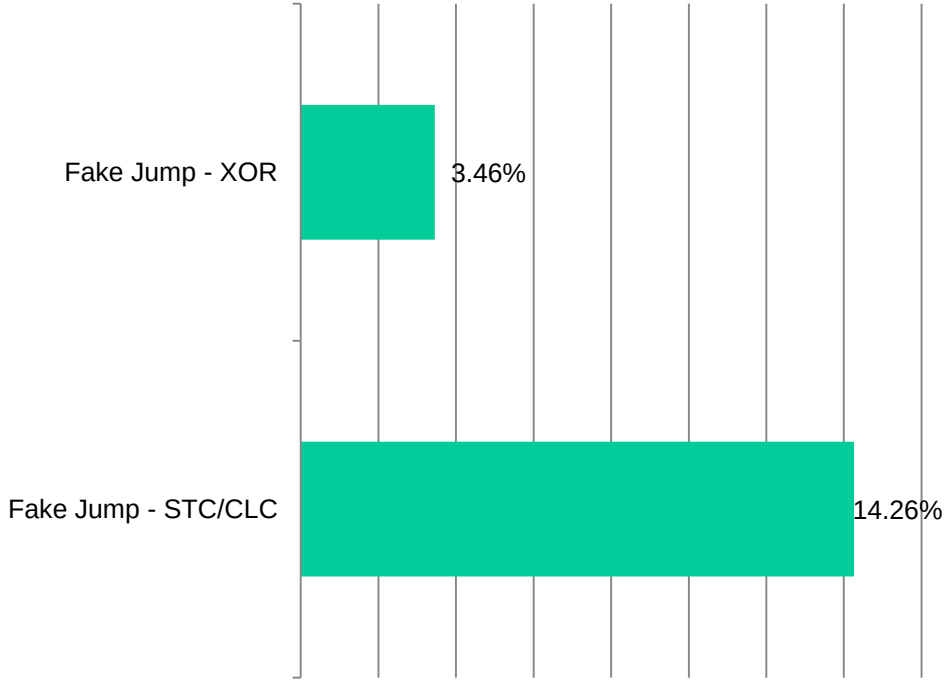


2014 results

Anti-Disassembly (updated)

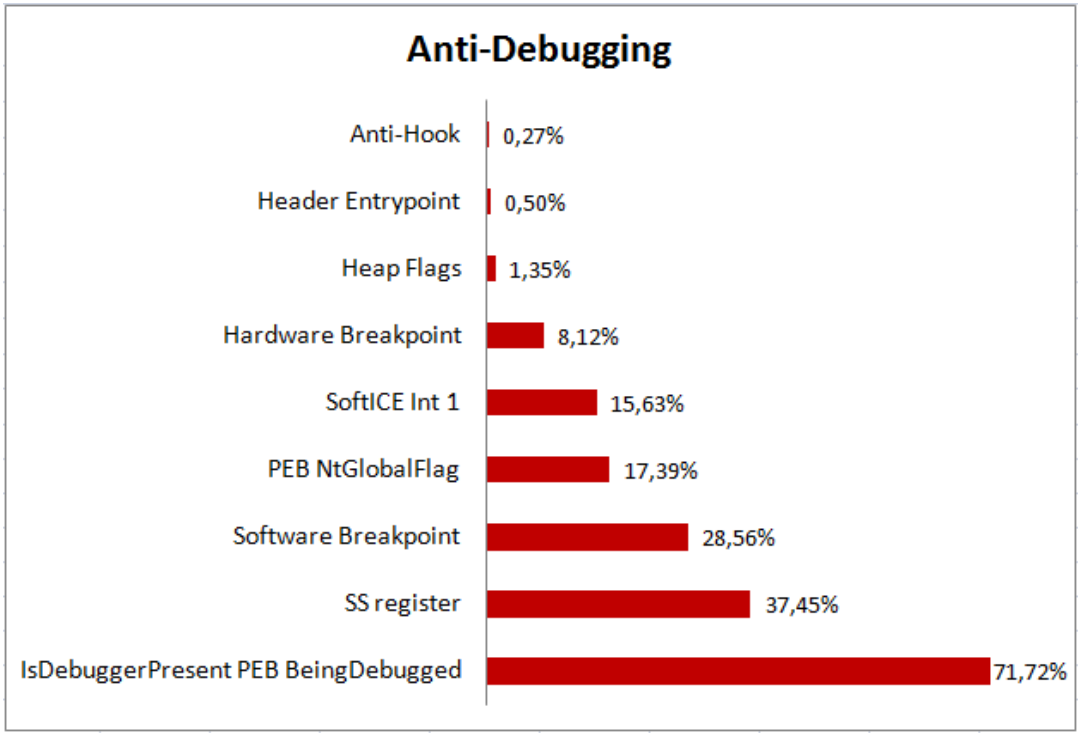


2012 results

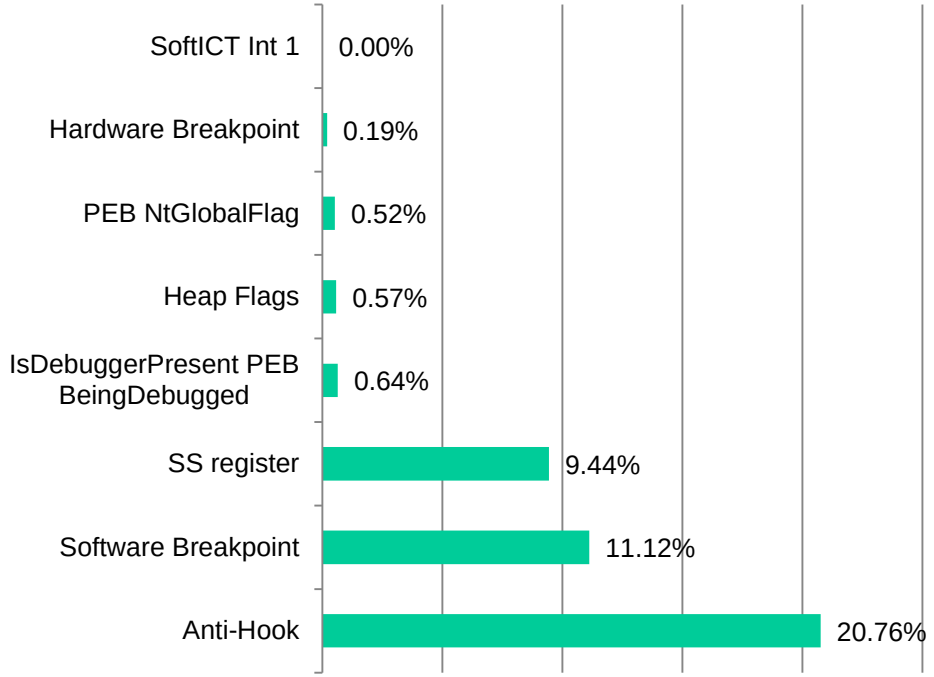


2014 results

Anti-Debugging (updated)

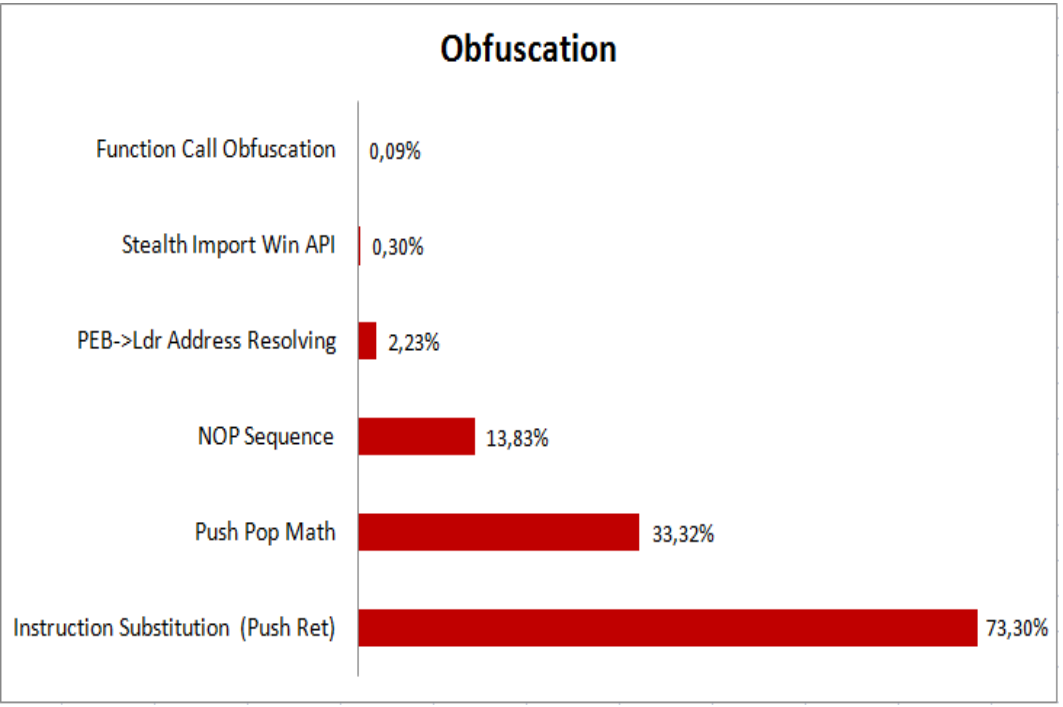


2012 results

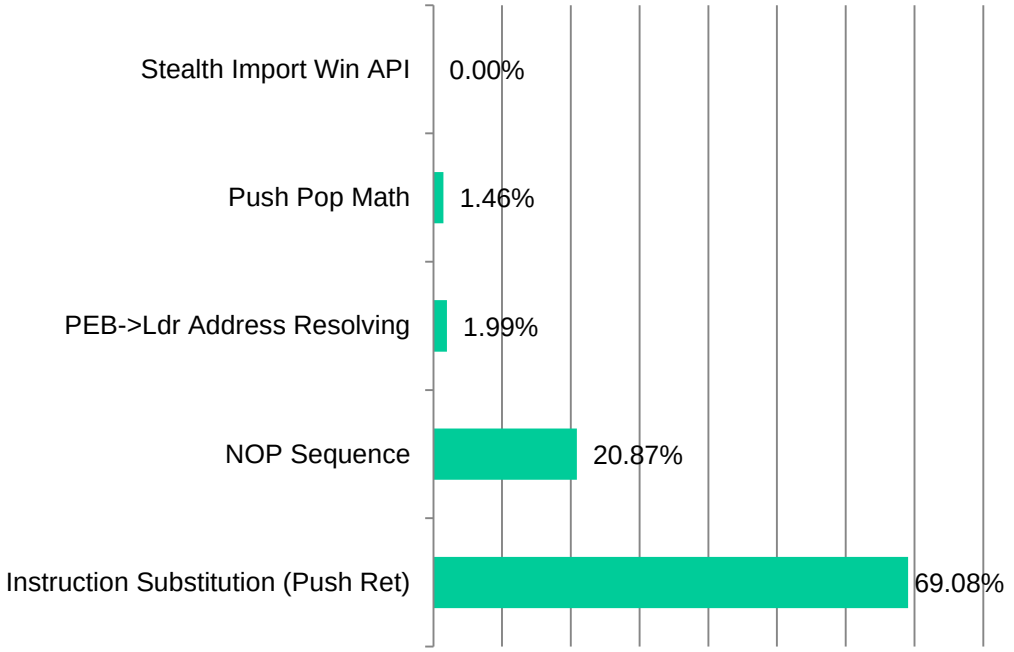


2014 results

Obfuscation (updated)

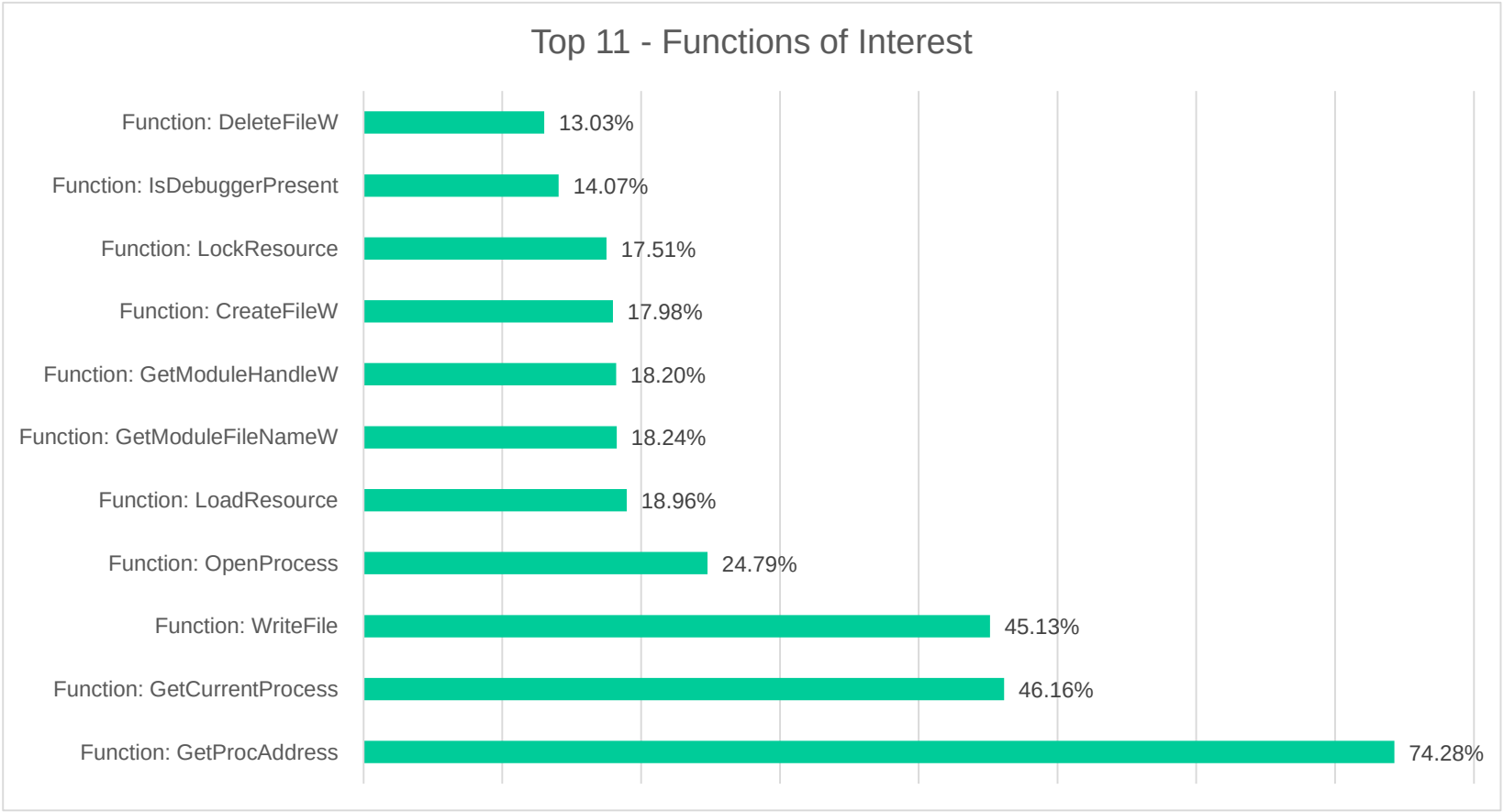


2012 results

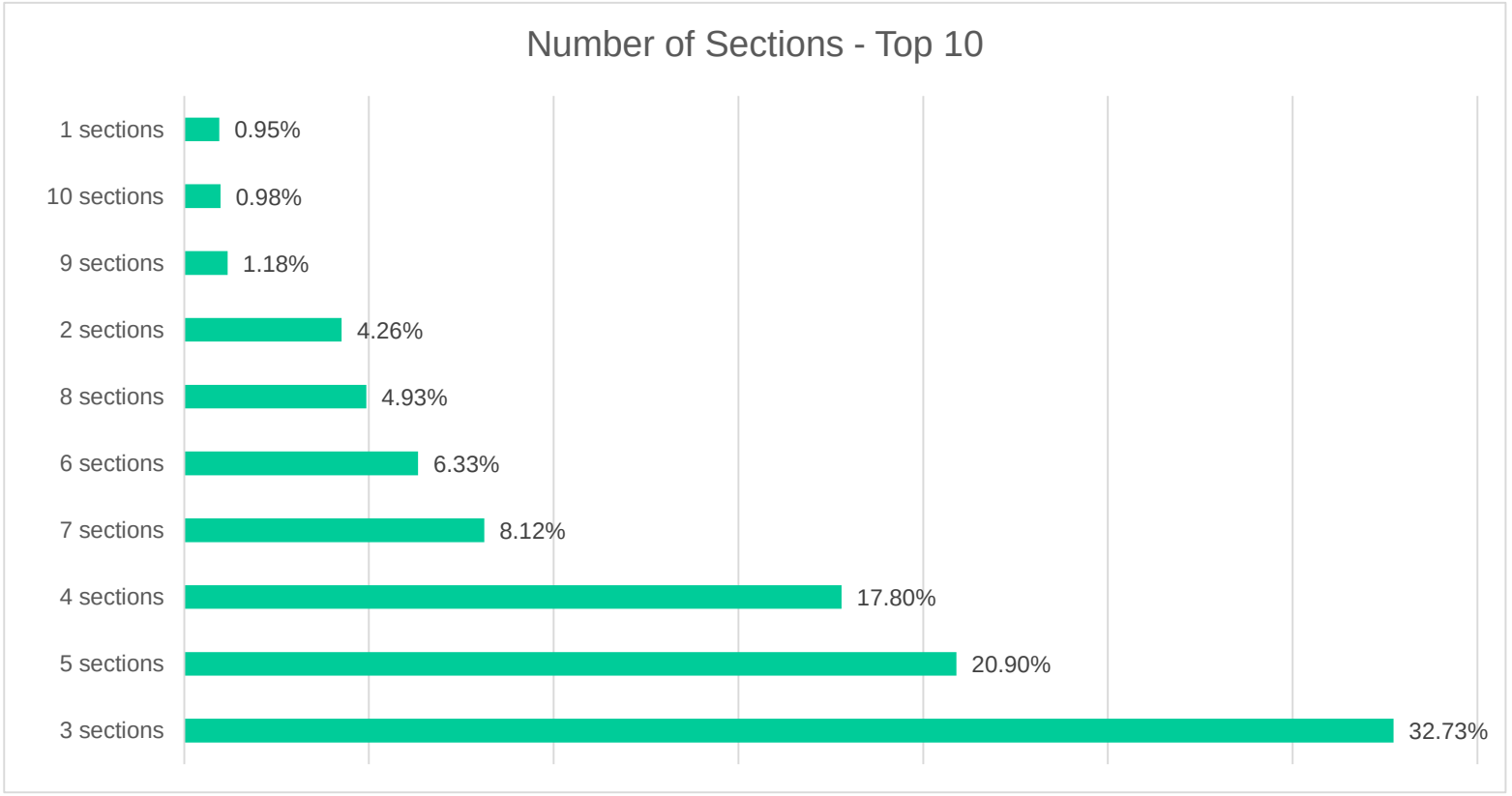


2014 results

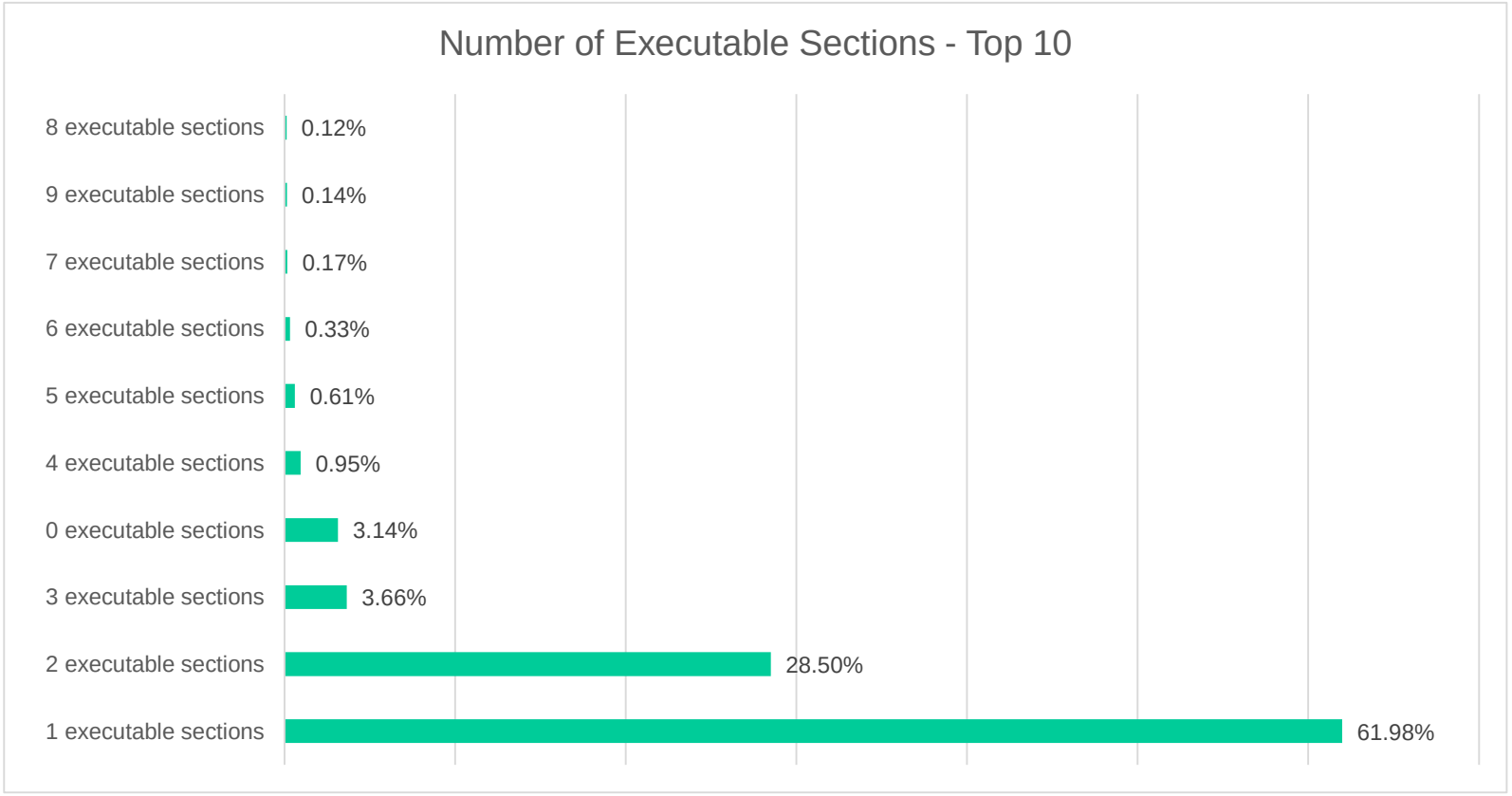
Blacklisted functions presence (new)



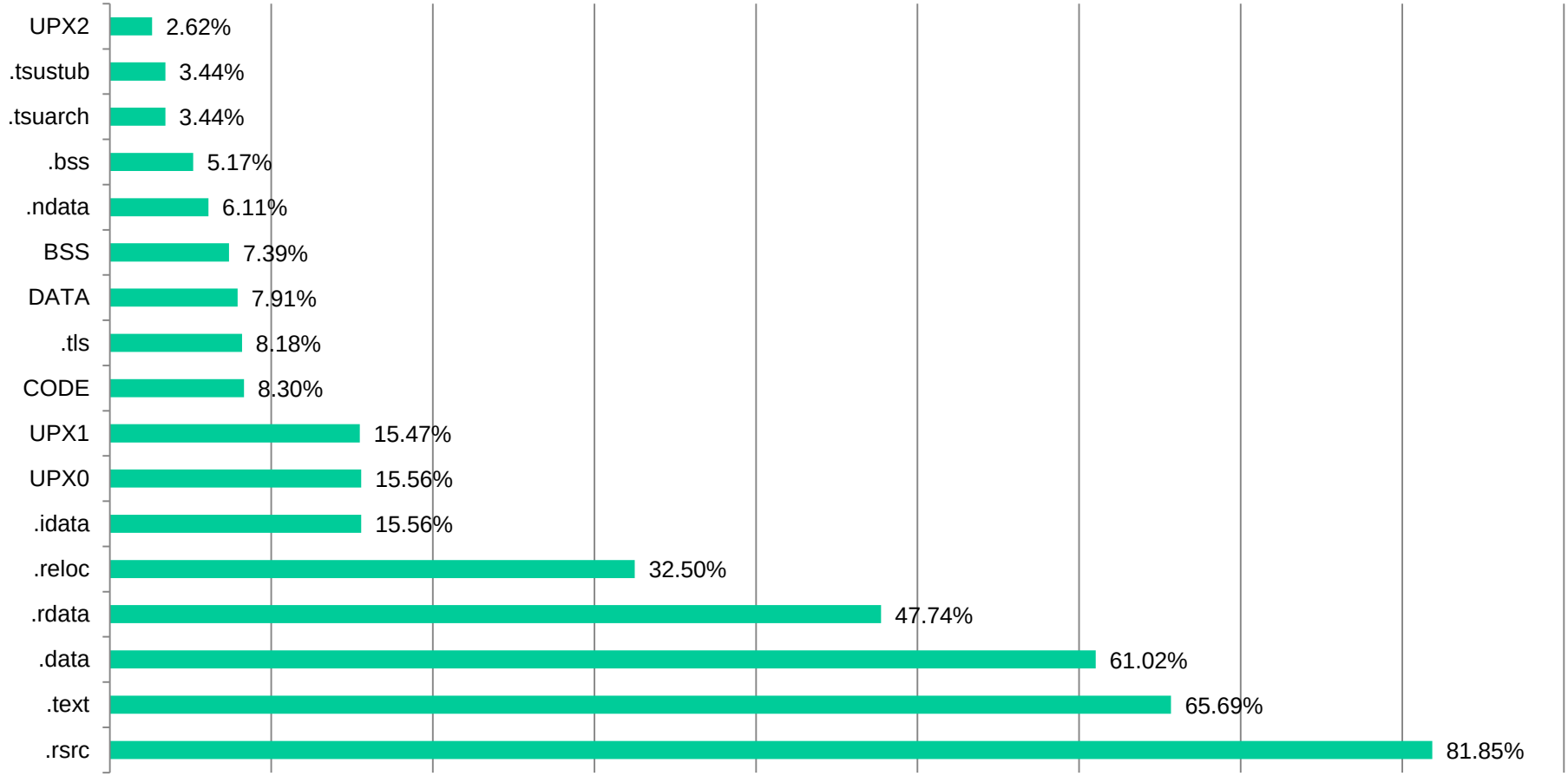
Number of sections (new)



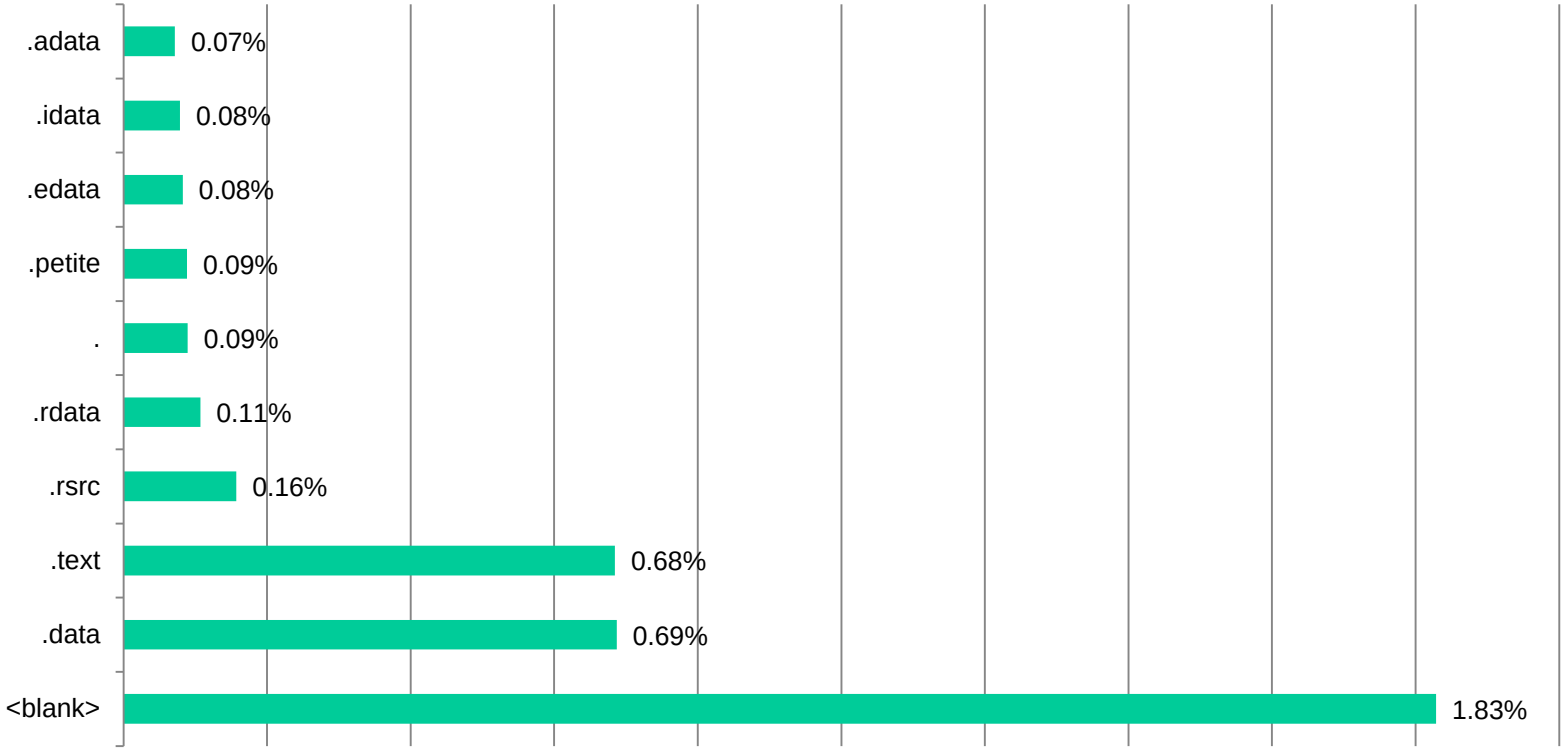
Number of executable sections (new)



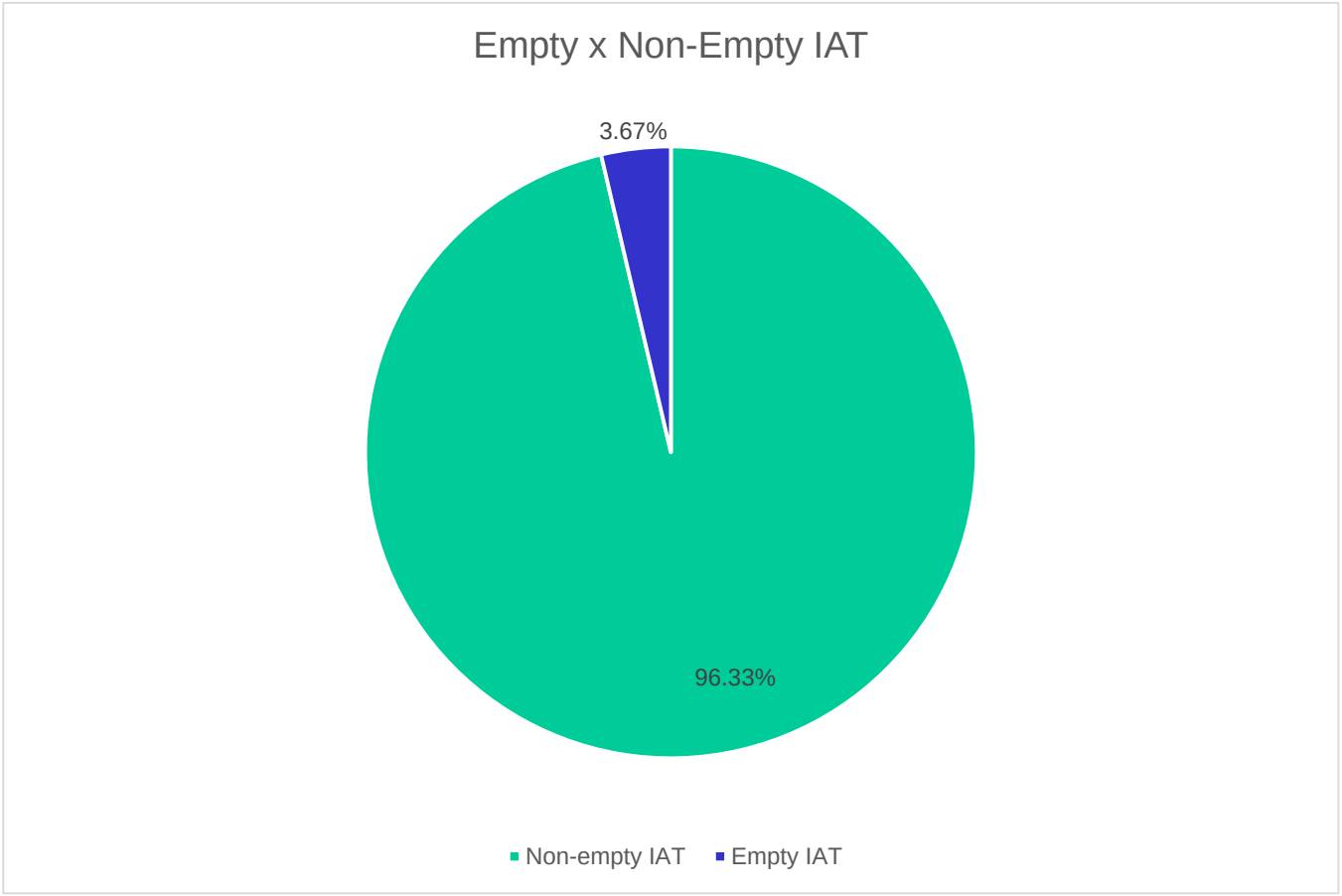
Section Names (new)



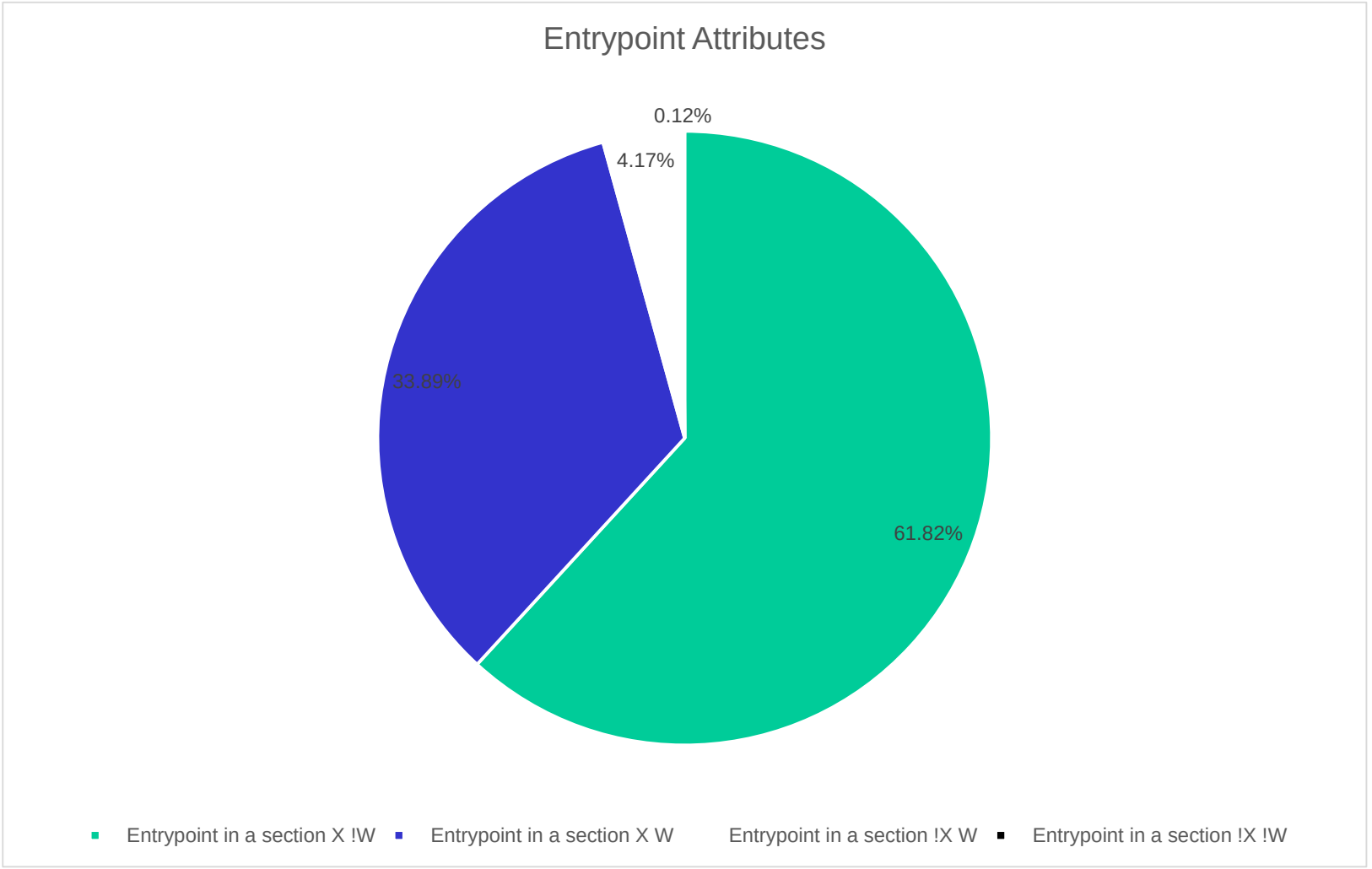
Repeated Section Names (new)



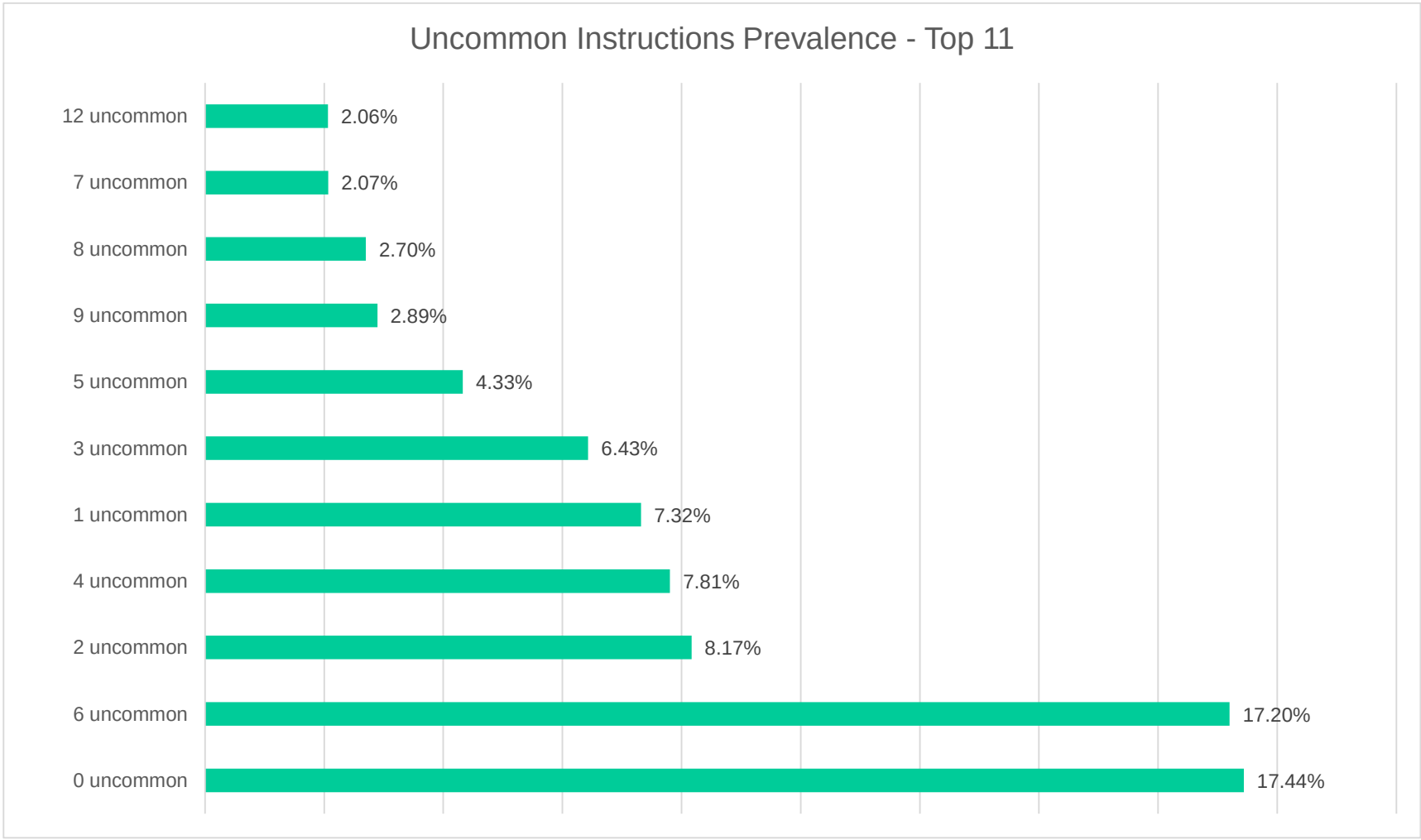
Empty IAT (new)



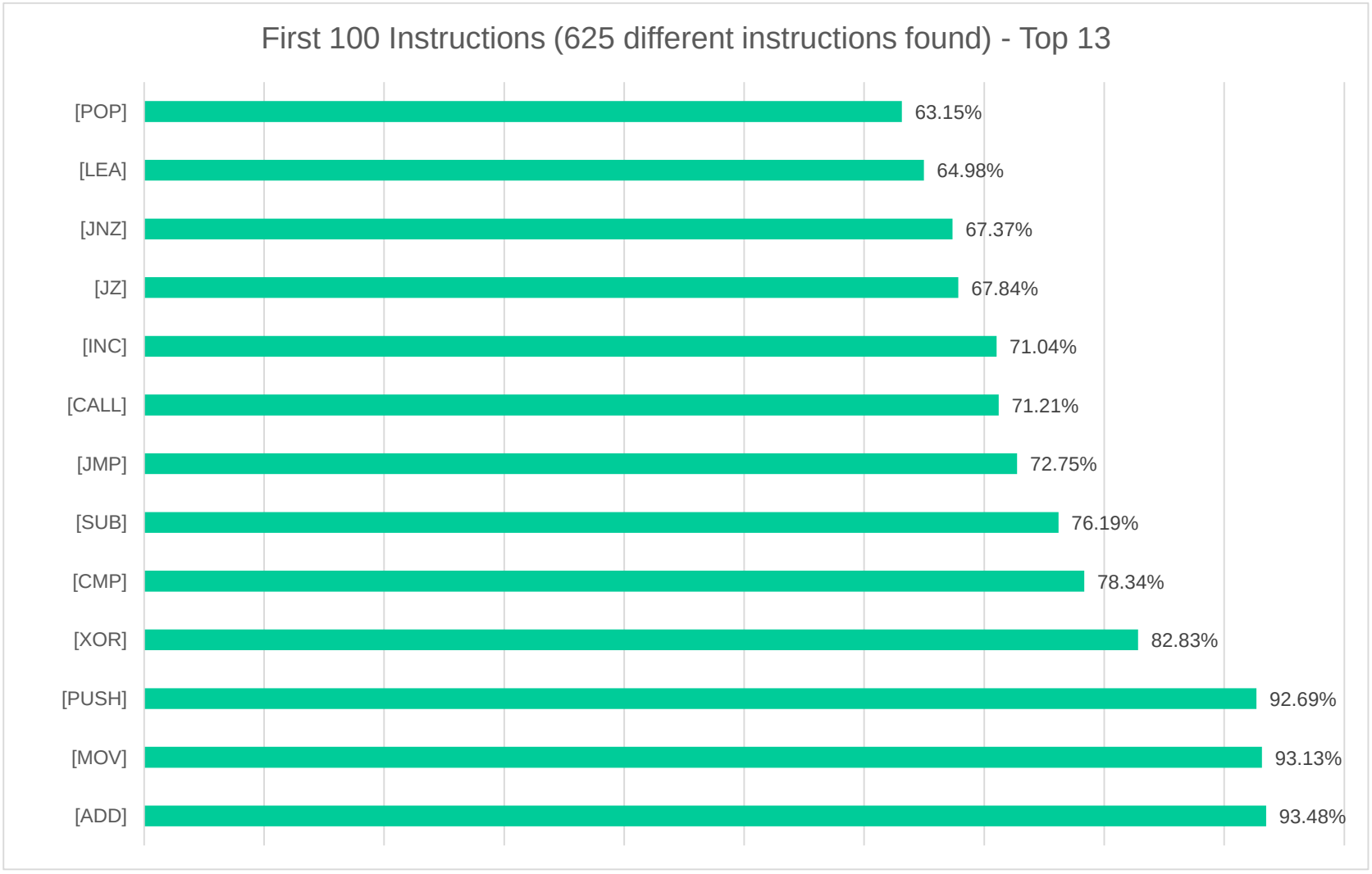
Entry-point attributes (new)



Entry-point uncommon insns (new)

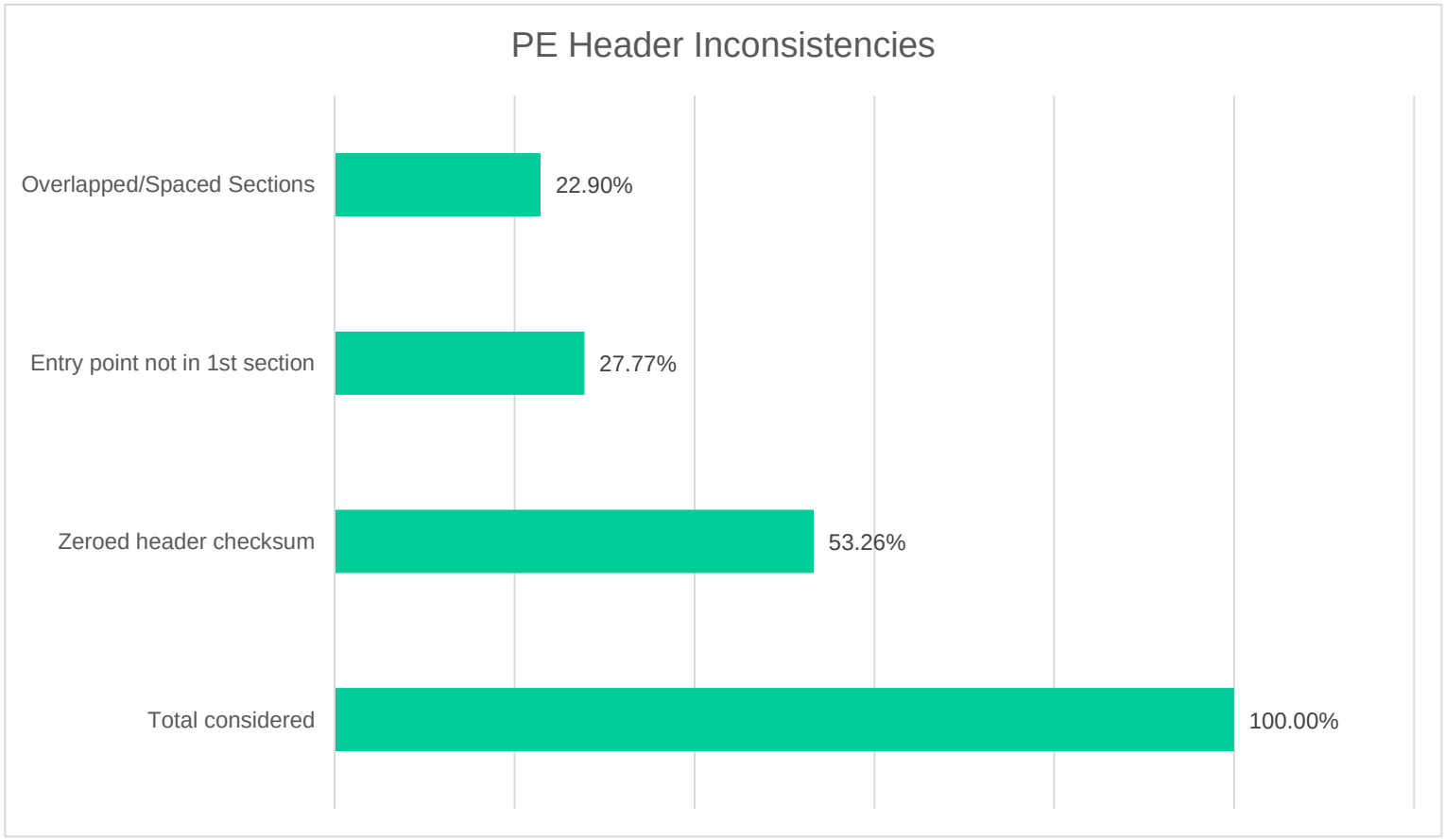


Entry-point insns (new)

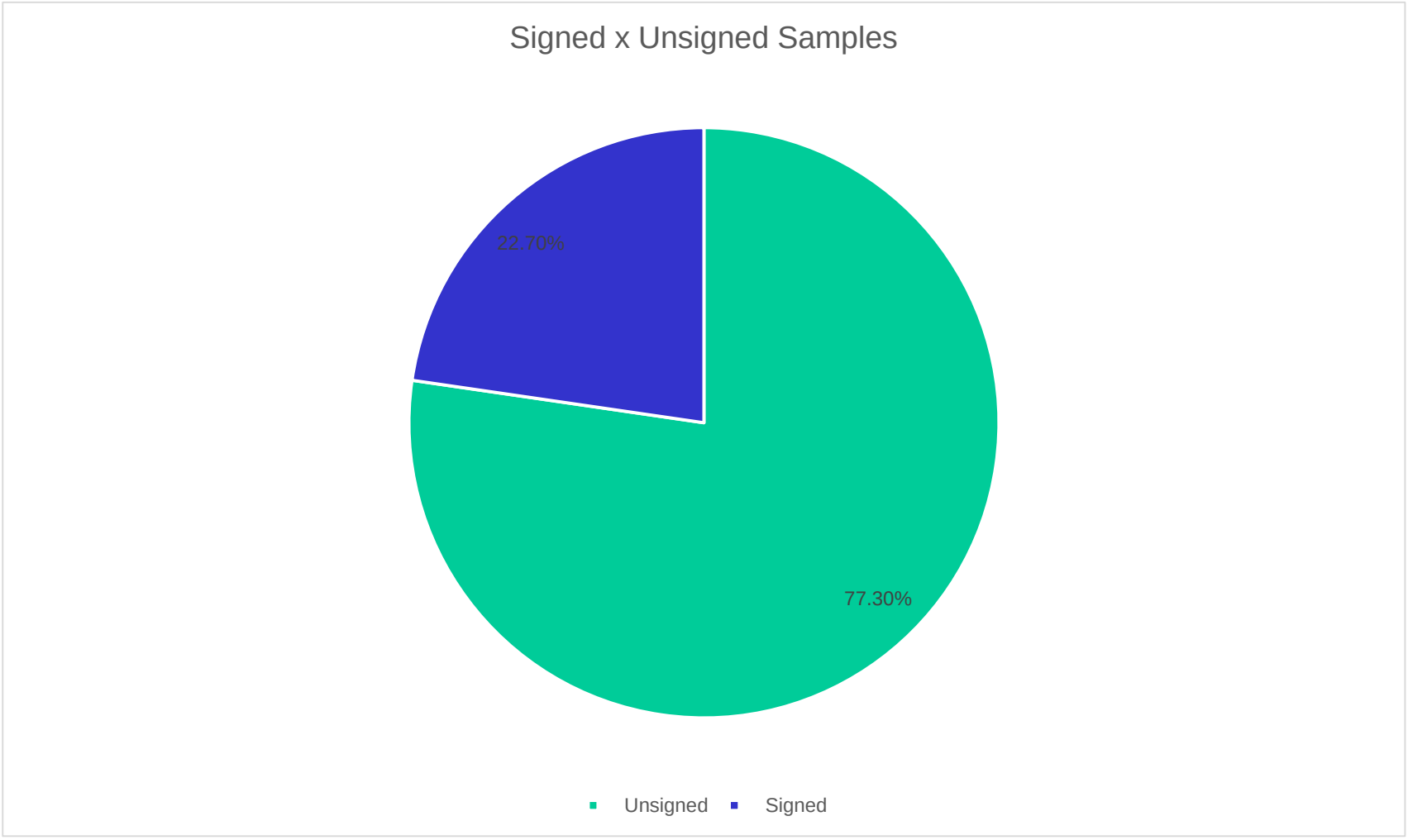


Entry-point sleep() calls (new)

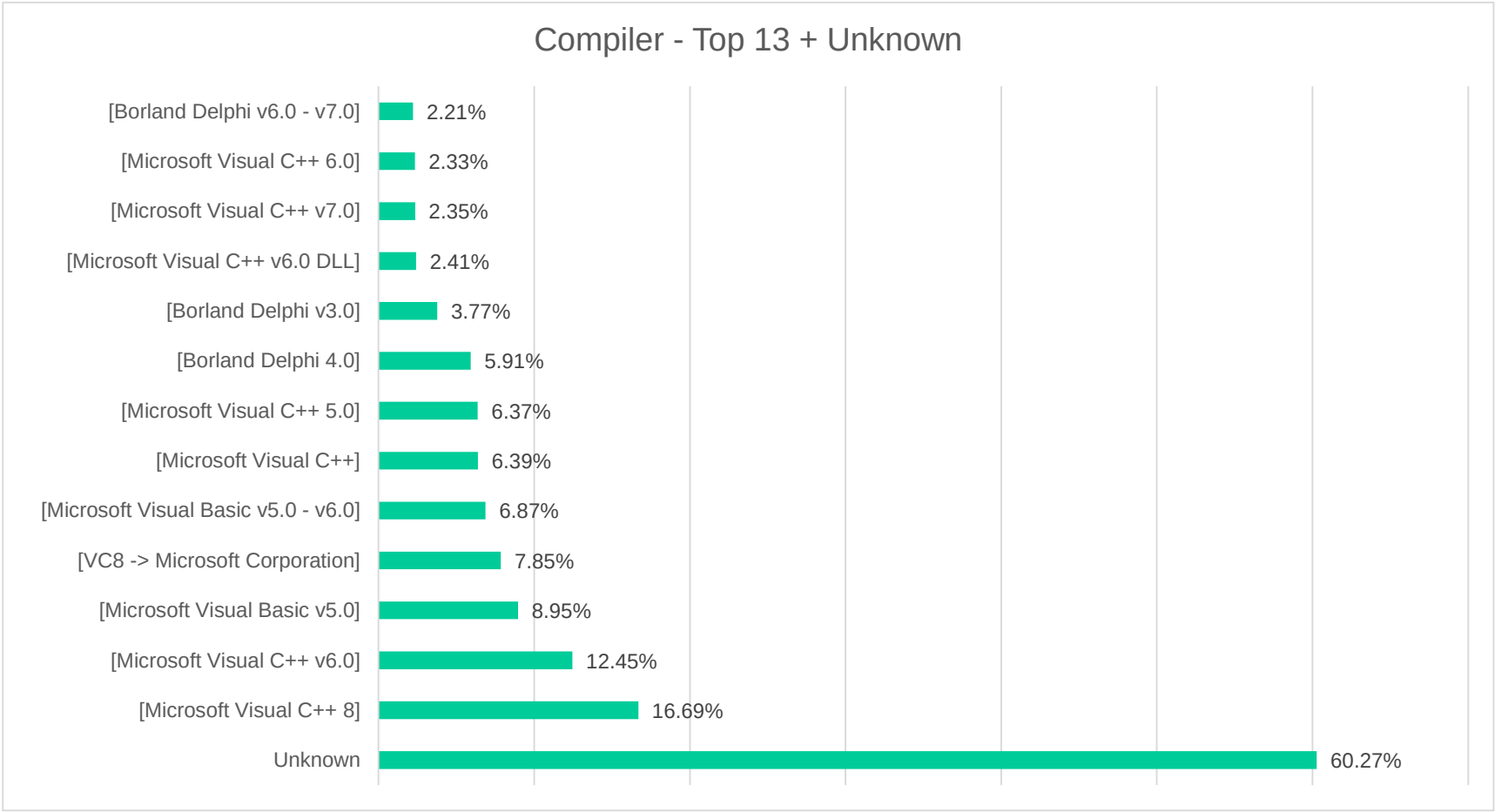
PE Header Fields (new)



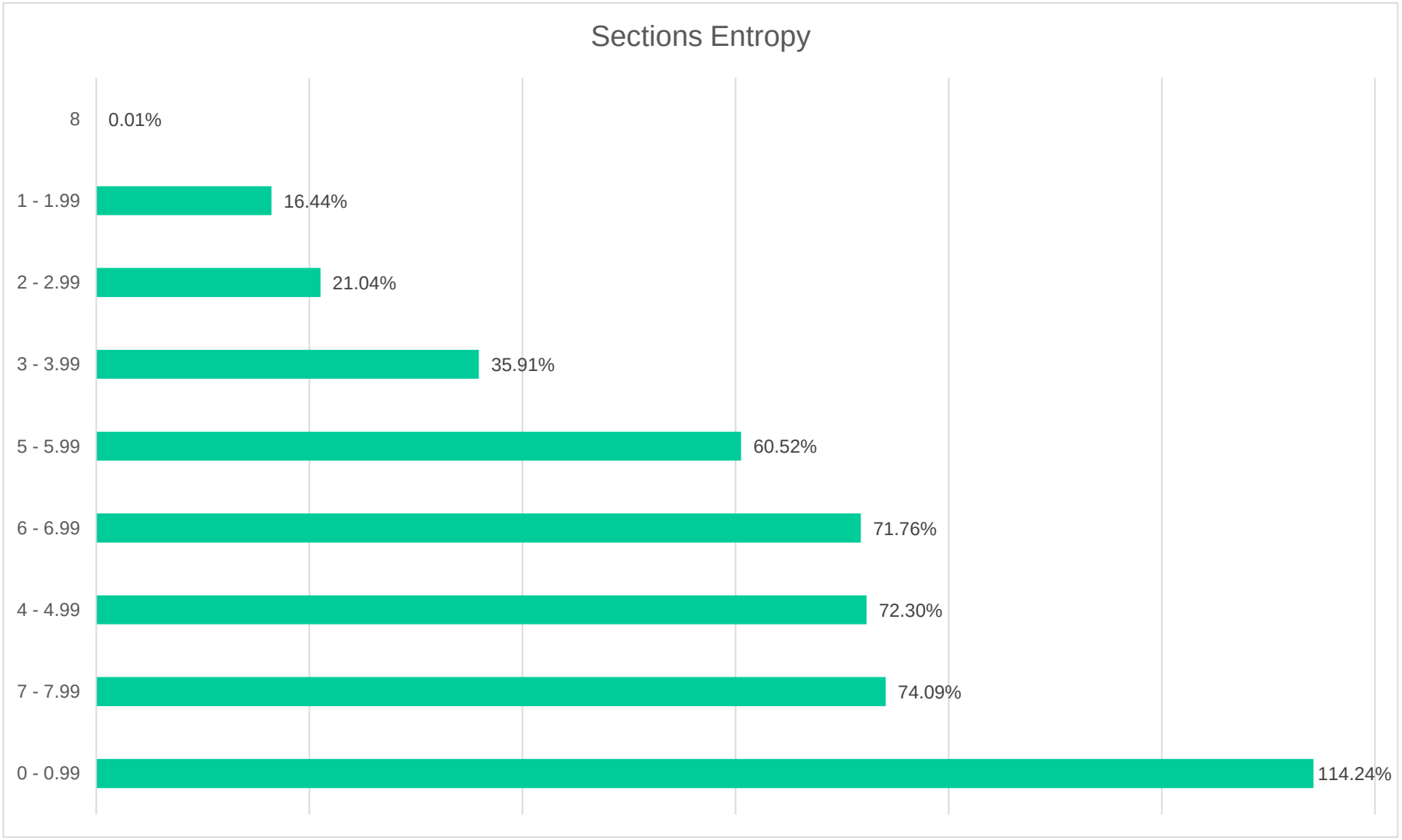
Signed binaries (new)



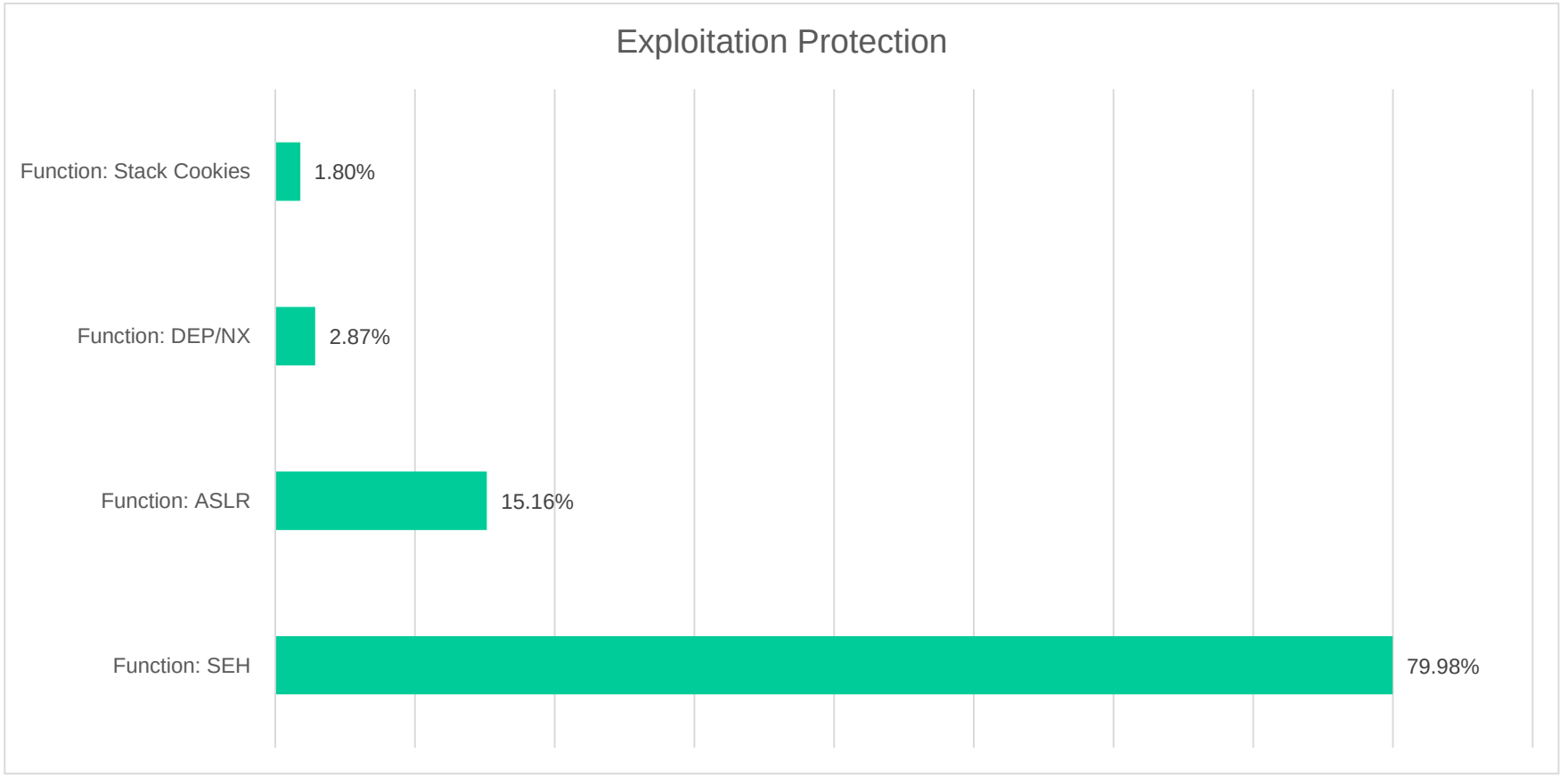
Compilers (new)



Sections Entropy (new)



Vulnerability Prevention Usage (new)



Anti-Debugging Techniques (2012)

- Studied and documented 33 techniques
- Currently scanning samples for 31 techniques
  - Detected: Marked in green
  - Evidence: Marked in yellow
  - Not covered: Marked in black

Anti-Debugging Techniques

- PEB NtGlobalFlag (Section 3.1)
- IsDebuggerPresent (Section 3.2)
- CheckRemoteDebuggerPresent (Section 3.3)
- Heap flags (Section 3.4)
- NtQueryInformationProcess – ProcessDebugPort (Section 3.5)
- Debug Objects – ProcessDebugObjectHandle Class (Section 3.6)
- Debug Objects – ProcessDebugFlags Class [1] (Section 3.7)
- NtQuerySystemInformation – SystemKernelDebuggerInformation (Section 3.8)
- OpenProcess – SeDebugPrivilege (Section 3.9)
- Alternative Desktop (Section 3.10)

Anti-Debugging Techniques

- Self-Debugging (Section 3.11)
- RtlQueryProcessDebugInformation (Section 3.12)
- Hardware Breakpoints (Section 3.13)
- OutputDebugString (Section 3.14)
- BlockInput (Section 3.15)
- Parent Process (Section 3.16)
- Device Names (Section 3.17)
- OllyDbg – OutputDebugString (Section 3.18)
- FindWindow (Section 3.19)
- SuspendThread (Section 3.20)

Anti-Debugging Techniques

- SoftICE – Interrupt 1 (Section 3.21)
- SS register (Section 3.22)
- UnhandledExceptionFilter (Section 3.23)
- Guard Pages (Section 3.24)
- Execution Timing (Section 3.25)
- Software Breakpoint Detection (Section 3.26)
- Thread Hiding (Section 3.27)
- NtSetDebugFilterState (Section 3.28)
- Instruction Counting (Section 3.29)
- Header Entrypoint (Section 3.30)
- Self-Execution (Section 3.31)
- Hook Detection (Section 3.32)
- DbgBreakPoint Overwrite (Section 3.33)

Anti-Disassembly Techniques (2012)

- Studied and documented 9 techniques and variations
- Currently scanning samples for 8 techniques and variations
  - Detected: Marked in green
  - Evidence: Marked in yellow
  - Not covered: Marked in black

Anti-Disassembly Techniques

- Garbage Bytes (Section 4.2.1)
- Program Control Flow Change (Section 4.2.2)
  - Direct approach
  - Indirect approach
- Fake Conditional Jumps (Section 4.2.3)
  - XOR variation
  - STC variation
  - CLC variation
- Call Trick (Section 4.2.4)
- Flow Redirection to the Middle of an Instruction (Section 4.2.5)
  - Redirection into other instructions
  - Redirection into itself

Obfuscation Techniques (2012)

- Studied and documented 14 techniques and variations
- Currently scanning samples for 7 techniques and variations
  - Detected: Marked in green
  - Evidence: Marked in yellow
  - Not covered: Marked in black

Obfuscation Techniques

- Push Pop Math (Section 4.3.1)
- NOP Sequence (Section 4.3.2)
- Instruction Substitution (Section 4.3.3)
  - JMP variation
  - MOV variation
  - XOR variation
  - JMP variation (Push Ret)
- Code Transposition (Section 4.3.4)
  - Program control flow forcing variation
  - Independent instructions reordering variation

Obfuscation Techniques

- Register Reassignment (Section 4.3.5)
- Code Integration (Section 4.3.6)
- Fake Code Insertion (Section 4.3.7)
- PEB->Ldr Address Resolving (Section 4.3.8)
- Stealth Import of the Windows API (Section 4.3.9)
- Function Call Obfuscation (Section 4.3.10)

Anti-VM Techniques (2012)

- Studied and documented 7 techniques and variations
- Currently scanning samples for 6 techniques and variations
  - Detected: Marked in green
  - Evidence: Marked in yellow
  - Not covered: Marked in black

Anti-VM Techniques

- CPU Instructions Results Comparison (Section 5.1)
  - SIDT approach
  - SLDT approach
  - SGDT approach
  - STR approach
  - SMSW approach
- VMWare – IN Instruction (Section 5.2)
- VirtualPC – Invalid Instruction (Section 5.3)

Malicious Techniques - Blacklisted Functions

- Studied and documented 73 functions
- Currently scanning samples using 73
  - We detect the import or the string presence, not necessarily the usage
  - Important to emphasize again: We care about precedence, not maliciousness (most functions WILL be seem in benign software)

Kernel32.dll Functions

SetVDMCurrentDirectories (not documented by Microsoft) (Section Y)

CreateDirectoryW (Section Y)

WriteFile (Section Y)

CreateFileW (Section Y)

GetTempPathW (Section Y)

LockResource (Section Y)

FindFirstFileW (Section Y)

GetCurrentProcess (Section Y)

GetModuleHandleW (Section Y)

Kernel32.dll Functions

OpenProcess (Section Y)

GetModuleFileNameW (Section Y)

GetProcAddress (Section Y)

LoadResource (Section Y)

RemoveDirectoryW (Section Y)

FindNextFileW (Section Y)

DeleteFileW (Section Y)

EnumProcesses (Section Y)

EnumProcessesModules (Section Y)

WinHttpReceiveResponse (Section Y)

WinHttpSendRequest (Section Y)

WinHttpOpenRequest (Section Y)

Gdi32.dll Functions

D3DKMTOpenAdapterFromDeviceName (not documented by Microsoft) (Section Y)

D3DKMTOpenAdapterFromGdiDisplayName (not documented by Microsoft) (Section Y)

D3DKMTOpenAdapterFromHdc (not documented by Microsoft) (Section Y)

D3DKMTOpenKeyedMutex (not documented by Microsoft) (Section Y)

D3DKMTOpenResource2 (not documented by Microsoft) (Section Y)

D3DKMTOpenResource (not documented by Microsoft) (Section Y)

D3DKMTOpenSynchronizationObject (not documented by Microsoft) (Section Y)

D3DKMTPollDisplayChildren (not documented by Microsoft) (Section Y)

D3DKMTPresent (not documented by Microsoft) (Section Y)

D3DKMTQueryAdapterInfo (not documented by Microsoft) (Section Y)

D3DKMTQueryAllocationResidency (not documented by Microsoft) (Section Y)

Gdi32.dll Functions

D3DKMTQueryResourceInfo (not documented by Microsoft) (Section Y)

D3DKMTQueryStatistics (not documented by Microsoft) (Section Y)

D3DKMTReleaseKeyedMutex (not documented by Microsoft) (Section Y)

D3DKMTReleaseProcessVidPnSourceOwners (not documented by Microsoft) (Section Y)

D3DKMTRender (not documented by Microsoft) (Section Y)

D3DKMTSetAllocationPriority (not documented by Microsoft) (Section Y)

D3DKMTSetContextSchedulingPriority (not documented by Microsoft) (Section Y)

D3DKMTSetDisplayMode (not documented by Microsoft) (Section Y)

D3DKMTSetDisplayPrivateDriverFormat (not documented by Microsoft) (Section Y)

Miscellaneous Functions

DhcpDeRegisterParamChange (dhcpcsvc.dll) (Section Y)

K32EnumProcessModules (Section Y)

K32EnumProcess (Section Y)

K32EnumProcesses (Section Y)

K32GetModuleBaseNameA (Section Y)

K32GetModuleFileNameExA (Section Y)

K32GetProcessImageFileNameA (Section Y)

Miscellaneous Functions

DeviceProblemTextA (Section Y)

DeviceProblemWizardA (Section Y)

DeviceCreateHardwarePage (Section Y)

DevicePropertiesA (Section Y)

Atexit (Section Y)

SetDllDirectory (Section Y)

SfcIsFileProtected (sfc.dll) (Section Y)

LockSetForegroundWindow (Section Y)

CICreateCommand (Section Y)

Miscellaneous Functions

FDIDestroy (cabinet.dll) (Section Y)

FDICopy (cabinet.dll) (Section Y)

FDICreate (cabinet.dll) (Section Y)

FindFirstUrlCacheEntryA (Section Y)

FindNextUrlCacheEntryA (Section Y)

Malicious Techniques - Miscellaneous

- Studied and documented 21 miscellaneous techniques
- Currently scanning samples using 10 of them:
  - Entry Point Instructions (Section Y)
  - Entry Point Sleep Function Calls (Section Y)
  - Entry Point in the First Section (Section Y)
  - Exploitation Protection Mechanisms (Section Y)
  - LoadLibrary() usage / Empty Import Data (Section Y)
  - Section Entropy (Section Y)
  - Section Amount (Section Y)
  - Section Names (Section Y)
  - Section Properties (W, X) (Section Y)
  - PE Header validations (illegal values, non-ascii section names, empty section names, invalid checksums)

Miscellaneous that we ignored for now

- StringFromGUID2, CoCreateGUID (ole32.dll) together with CreateMutex
- NtShutdownSystem (and ntshutdown.pdb)
- Missing File Date (in the version information):
  - CompanyName matches (weird strings)
  - FileDescription matches
  - LegalCopyright matches
  - OriginalFilename (different then the binary name)
  - Find company names such as Microsoft but not signed binaries
- Overlays (PEStudio does a great job at that)
  - Smart Installer, 7zSFX, Nullsoft Scriptable Install System (NSIS), AutoIT, Spoon (Xenode), RAR, EXE, DLL

Miscellaneous that we ignored for now

- Known SID string
- WMI Filters
- Imports
- Empty DOS Stub
- Microsoft image without debug symbols
- Microsoft image but with outdated debug info (not RSDS)
- Find weird strings (bad word lookup) in the app manifest

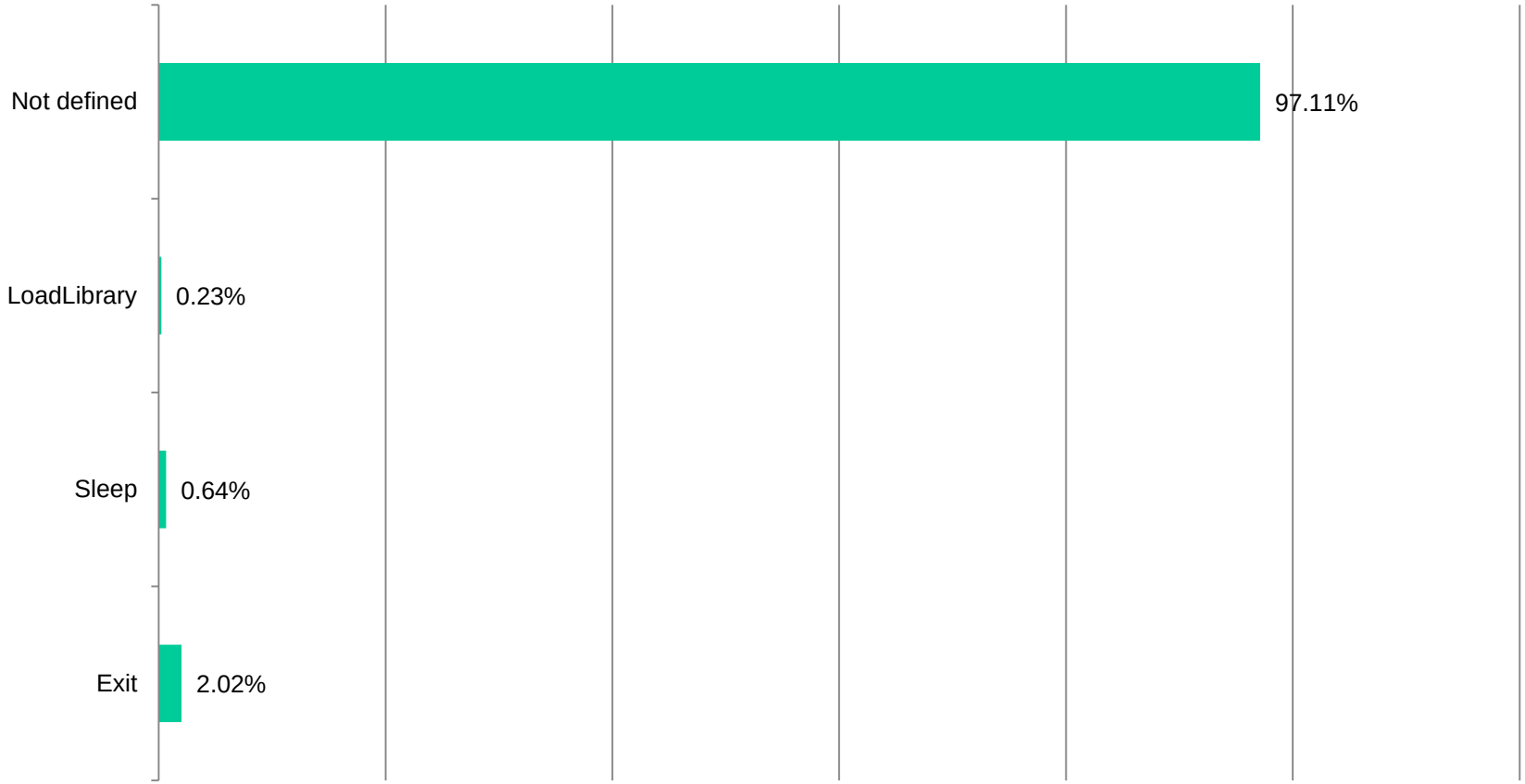
Advanced Malware?

- On July, 25, 2012 Morgan Marquis-Boire and Bill Marczak released a paper about the FinFisher Spy Kit. Their paper mentions many protection techniques used by the code:
  - A piece of code for crashing OllyDBG
  - DbgBreakPoint Overwrite (Covered in Section 3.33)
  - IsDebuggerPresent (Covered in Section 3.2)
  - Thread Hiding (Covered in Section 3.27)
  - Debug Objects - ProcessDebugObjectHandle Class (Covered in Section 3.6)

Bonus – What the malware does?

- We asked ourselves the question:
  - Ok, so the malware tries to define it is been analyzed… so what?
  - If most samples simply do not perform anything malicious under analysis, why not mimic an analyzed system to avoid infections?
- Quite impossible to be scientific answering that:
  - Maybe we missed something
  - Maybe the malware would do something malicious anyway
  - We tried to identify malware exit()ing/sleep()ing the execution few instructions (100) after the detection happened
    - Same block (100 instructions)
    - First flow change follow too (100 instructions)
    - Not taken (100 instructions)

Bonus – What the malware does?



Resources

- Updated versions of the paper and presentation are going to be available at:
  - http://www.kernelhacking.com/rodrigo/docs/blackhat2014-paper.pdf
  - http://www.kernelhacking.com/rodrigo/docs/blackhat2014-presentation.pdf
- Sample code for the different protection mechanisms we detect is available on github (we updated it after the latest presentation):
  - https://github.com/rrbranco/blackhat2012

Future?

- Generate the same prevalence numbers for known-good binaries
  - Define discrepancies
  - Define possibilities for detection
- We can do that, but we don’t have the huge database
  - Download from CNET?
  - Ideas??

Conclusions

- We analyzed millions of malware samples and showed results about the prevalence of different characteristics
- Demonstrated that static analysis is a powerful mechanism and should be explored more
- There are a lot more to do. Hopefully this will demonstrate that sharing prevalence data is helpful
  - We still have a lot to do... and so do you! Help us! Share your ideas, data, samples

Acknowledgements

- Ronaldo Pinheiro de Lima and Pedro Drimel provided great contributions to our research in 2012
- McAfee guys – very insightful discussions and ideas on how to validate a not biased sample base (we actually implemented the code differential thanks to those discussions)

And ALL of you who wrote papers about those techniques

THE END ! Really !?

Gabriel Negreira Barbosa (@gabrielnb)
Rodrigo Rubira Branco (@BSDaemon)
{gabriel.negreira.barbosa || rodrigo.branco} *noSPAM* intel.com