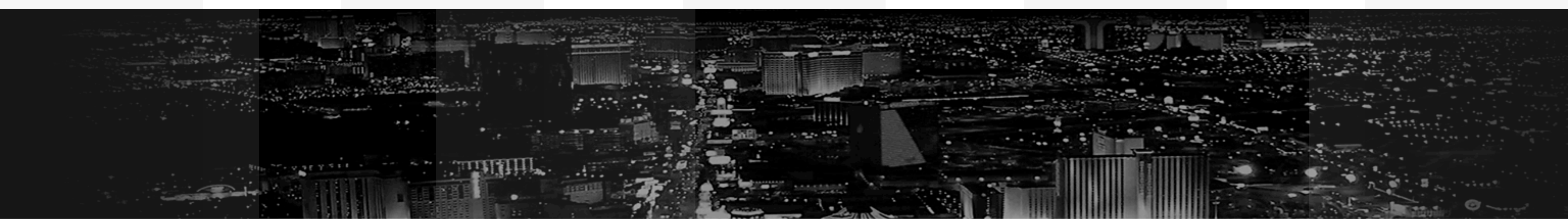
HITBSecConf

Keeping Knowledge Free for Over a Decade

Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti-VM Technologies

Rodrigo Rubira Branco (BSDaemon)
http://twitter.com/bsdaemon
rodrigo *noSPAM* kernelhacking.com



Acknowledgements

- Gabriel Barbosa and Pedro Drimel, part of the team who wrote everything I'm presenting today
- Ronaldo Pinheiro de Lima – Joined our team a bit later in the research process, but gave amazing contributions!
- Peter Ferrier – Amazing papers, great feedback/discussions by email
- Jurriaan Bremer – SSEXY
- Reversing Labs – TitaniumCore

- And ALL of you who wrote papers about those techniques

Where I live…

Nah, I actually live here…

São Paulo

Agenda

- Introduction / Motivation
- Objectives
- Methodology
- Dissect || PE Project
- Executive Summary
- Techniques
- Resources
- Conclusions

Introduction / Motivation

- Hundreds of thousands of new samples every week
- Still, automation is about single tasks or single analysis
- Presentations still pointing tens of thousands in tests (what about the millions of samples?)
- Companies promote research which uses words such as ‘many’ instead of X number

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

Objectives

- Analyze millions of malware samples
- Share the current results related to:
  - Anti-Debugging
  - Anti-Disassembly
  - Obfuscation
  - Anti-VM
- Keep sharing more and better results in our portal (www.dissect.pe):
  - New malware samples are always being analyzed
  - Detection algorithms are constantly being improved
  - The system does not analyze only anti-RE things

Dissect || PE Project

- Scalable and flexible automated malware analysis system
- Receives malware from trusted partners
- Portal available for partners, researchers and general media with analysis data

Dissect || PE – Overview

- Open research malware analysis system for the community
  - Open architecture (documented in IEEE Malware 2010 paper)
  - Works with plugins
- 10 dedicated machines distributed in 3 sites:
  - 2 sites in Brazil (São Paulo and Bauru cities)
  - 1 site in Germany
- Some numbers:
  - Receives more than 150 GB of malwares per day
  - More than 30 million unique samples

Dissect || PE – Partners

Dissect || PE – Backend

- Each backend downloads samples scheduled for analysis (our scheduler algorithms are documented in a IEEE Malware2011 paper)
- Analyze samples
  - Both static and dynamic analysis currently supported
- Analysis results accessible from the portal
  - Sync'ed back from the backend
- Some characteristics:
  - Plugins
  - Network traffic
  - Unpacked version of the malware

Dissect || PE – Plugins

- Samples are analyzed by independent applications named "plugins"
- Easy to add and/or remove plugins
  - Just a matter of copy and remove their files
- Language independent
- Easy to write new plugins:
  - Needed information come as arguments
    - We usually create handlers so the researcher does not need to change his actual code
  - Simply print the result to stdout
    - The backend takes care of parsing it accordingly

Dissect || PE – Plugin Examples

- Python

```
print "My plugin result."
```

- C

```
#include <stdio.h>
int main(int argc, char **argv) {
    printf("My plugin result.\n");
    return 1;
}
```

Dissect || PE – Plugin Types

- Static:
  - Usually executed outside of the VM (we already have an exception for the unpacking plugin)
  - Failsafe: errors do not compromise the system
  - Might get executed in one of two different situations depending on where we copied the plugin:
    - Before the malware is executed
    - After the malware was executed
- Dynamic:
  - Executed inside a Windows system (for now the only supported OS, soon others)

Dissect || PE – Network Traffic

- During dynamic analysis all the network traffic is captured
- Pcap available at the portal
- Dissectors:
  - Analyze the pcap and print the contents in a user-friendly way
  - Supporting IRC, P2P, HTTP, DNS and other protocols
  - SSL inspection (pre-loaded keys)

Methodology

- Used a total of 72 cores and 100 GB of memory
- Analyzed only 32-bit PE samples
- Packed samples:
  - Different samples using the same packer were counted as 1 unique sample
    - So, each sample was analyzed once
  - Analyzed all packers present among the 4 million samples
- Unpacked samples:
  - Avoided samples bigger than 3,9 MB for performance reasons (with some exceptions such as the Flame Malware)

Methodology

- Static analysis:
  - Main focus of this presentation
  - Improves the throughput (with well-written code)
  - Not detectable by malwares
- Dynamic counter-part:
  - It is not viable to statically detect everything
  - Already developed and deployed, but is not covered by this presentation
    - The related results can be found at https://www.dissect.pe

Methodology

- Malware protection techniques in this work:
  - State-of-the-art papers/journals
  - Malwares in the wild
  - Some techniques we documented are not yet covered by our system:
    - The system is constantly being updated
  - All techniques were implemented even when there were no public examples of it (github)
- Our testbed comprises 883 samples to:
  - Detect bugs
  - Performance measurement
  - Technique coverage

Methodology

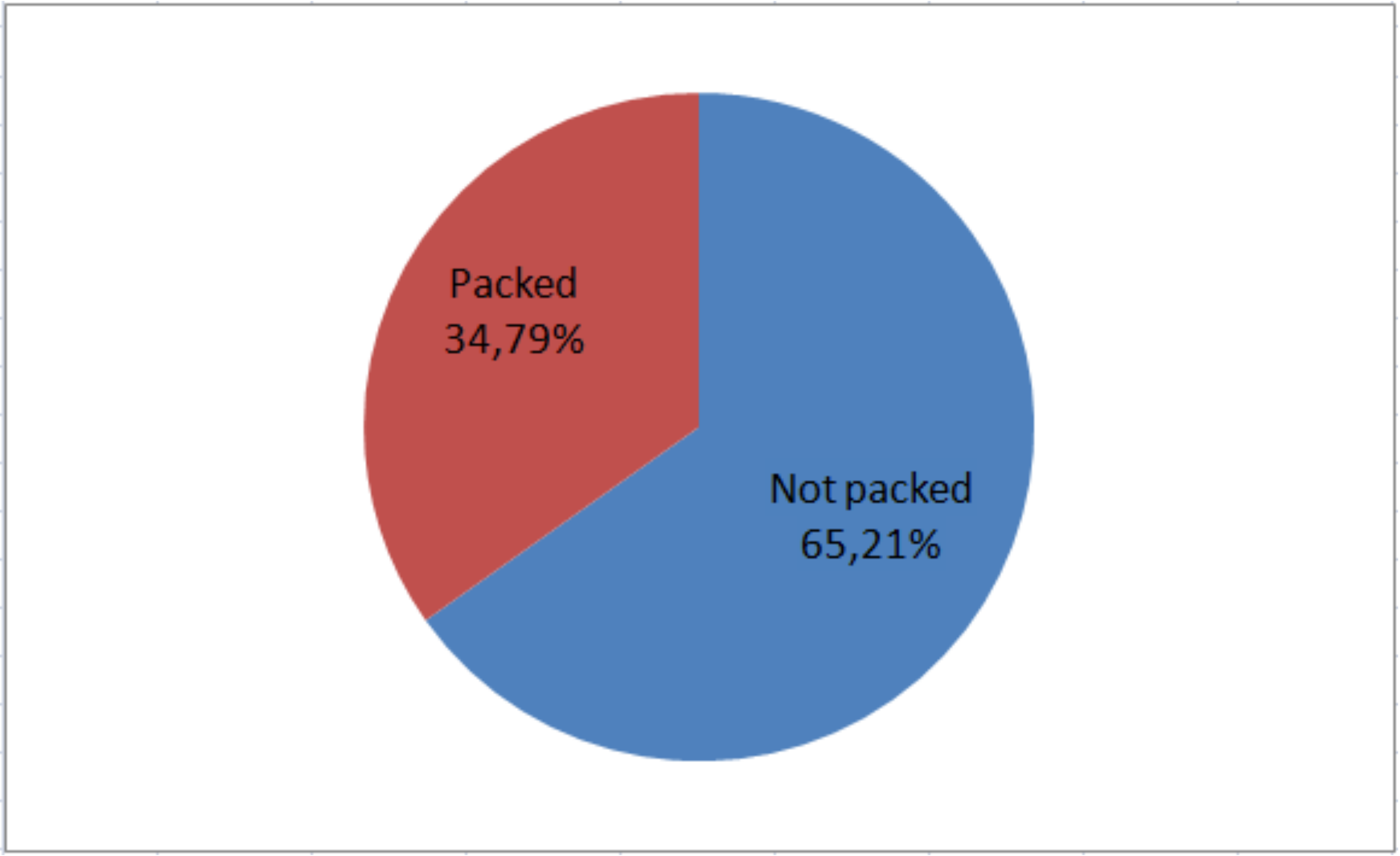
- Possible techniques detection results:
  - Detected:
    - Current detection algorithms detected the malware protection technique
  - Not detected:
    - Current detection algorithms did not detect the malware protection technique
  - Evidence detected:
    - Current detection algorithms could not deterministically detect the protection technique, but some evidences were found

Methodology

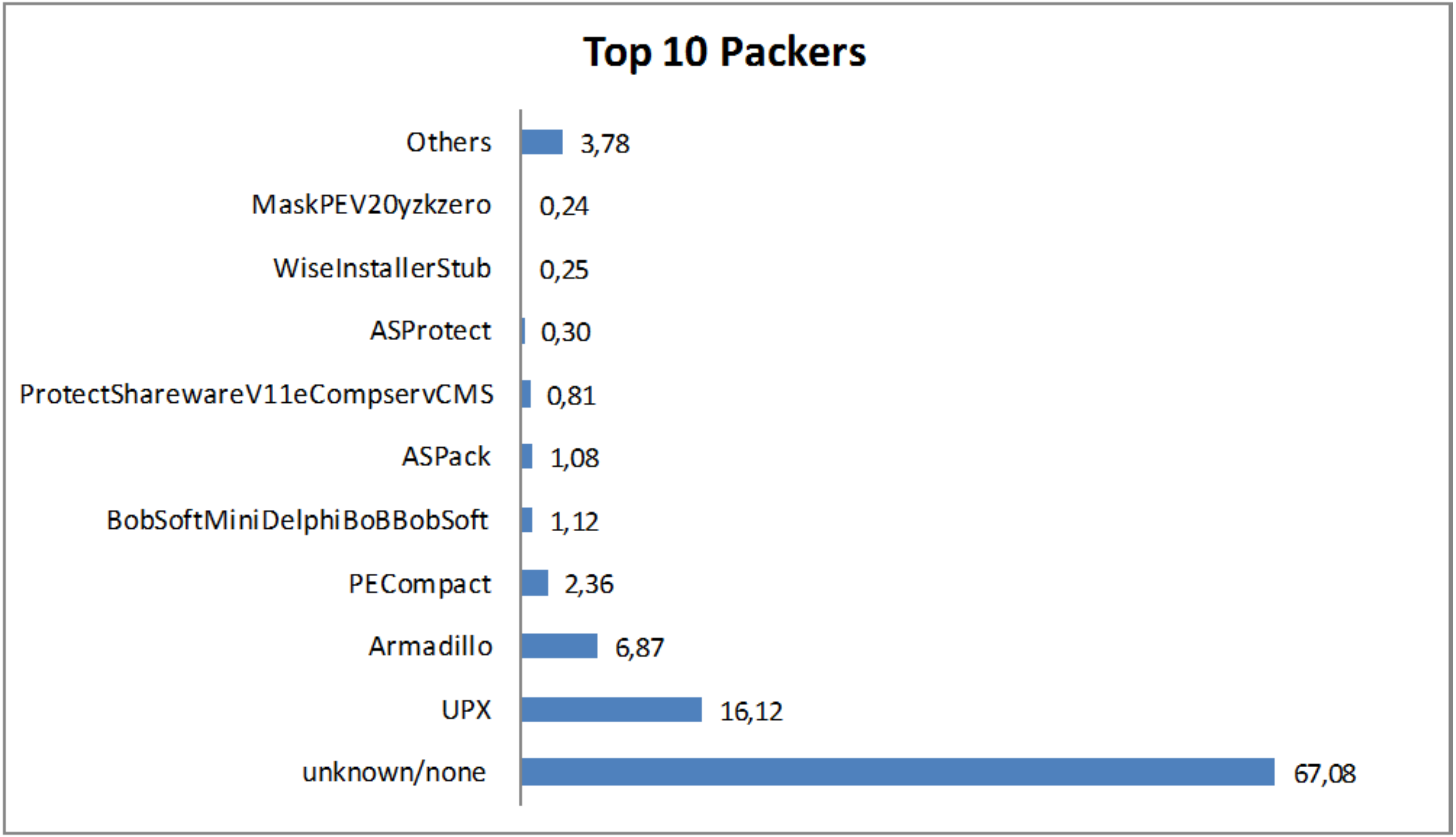
- Analysis rely on executable sections and in the entrypoint one
  - Decreases the probability to analyze data as code
  - Improves even more the analysis time
  - For now we miss non-executable areas, even if they are referred by analyzed sections (future work will cover this)
- Disassembly-related analysis framework:
  - Facilitates the development of disassembly analysis code
  - Speeds up the disassembly process for plugins
  - Calls-back the plugins for specific instruction types
  - Disassembly once, analyze all
  - Care must be taken to avoid disassembly attacks

Executive Summary

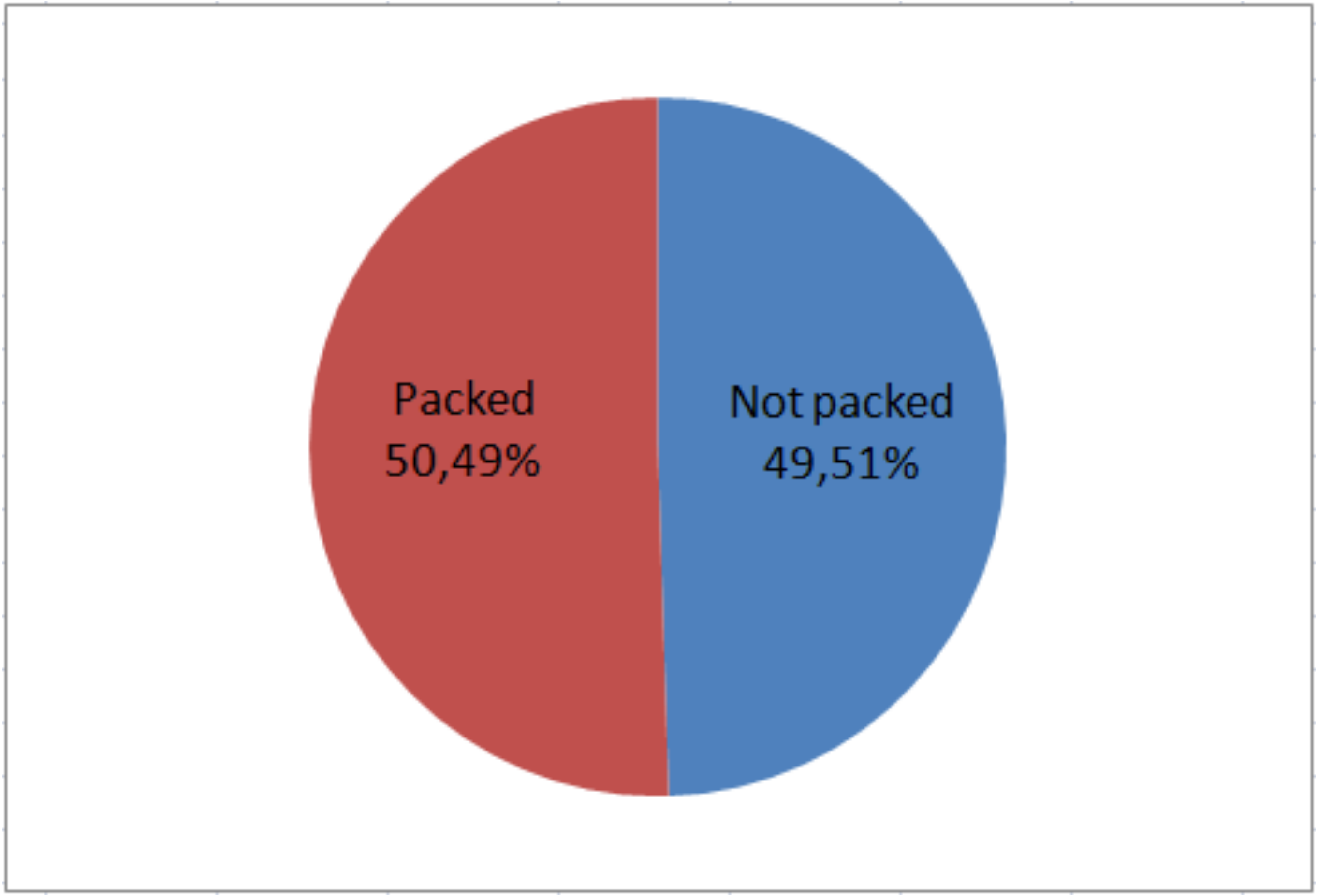
Packed vs Not Packed



Top Packers


Top 10 Packers
Others
3,78
MaskPEV20yzkzero
0,24
WiseInstallerStub
0,25
ASProtect
0,30
ProtectSharewareV11eCompservCMS
0,81
ASPack
1,08
BobSoftMiniDelphiBoBBobSoft
1,12
PECompact
2,36
Armadillo
6,87
UPX
16,12
unknown/none
67,08

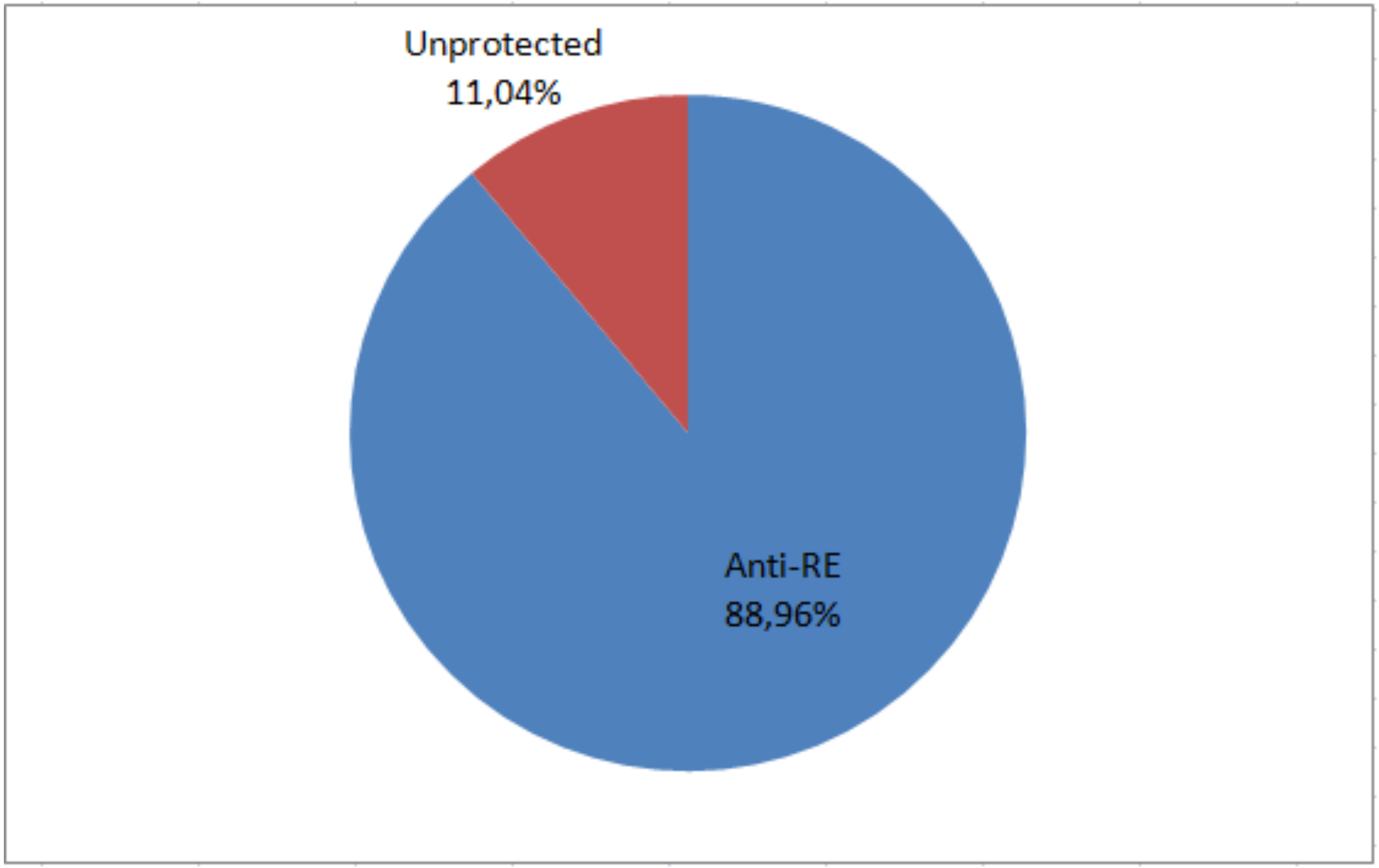
Malware Targeting Brazilian Banks



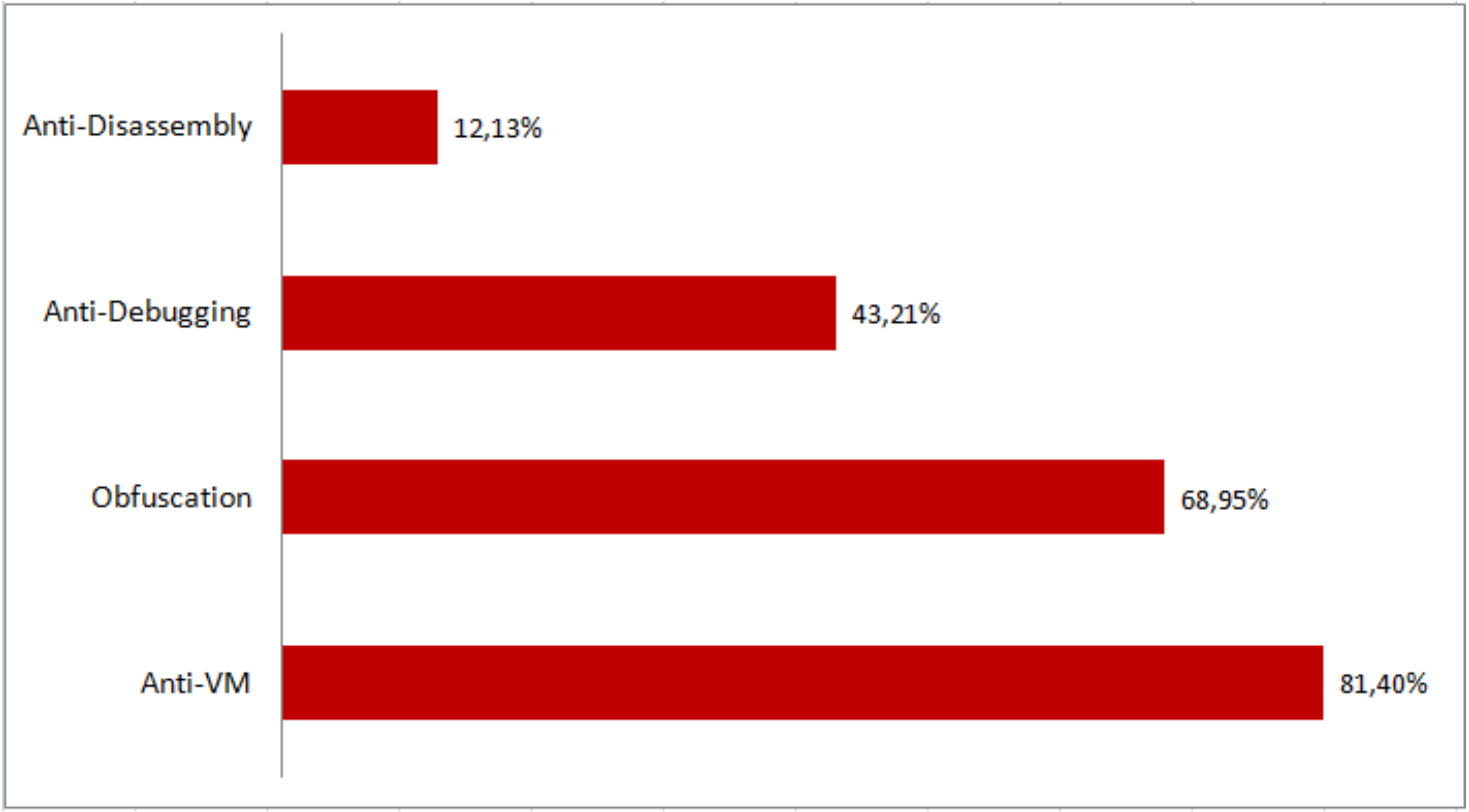
Protecting Mechanisms of Packers

Paper (yes, we wrote one...)

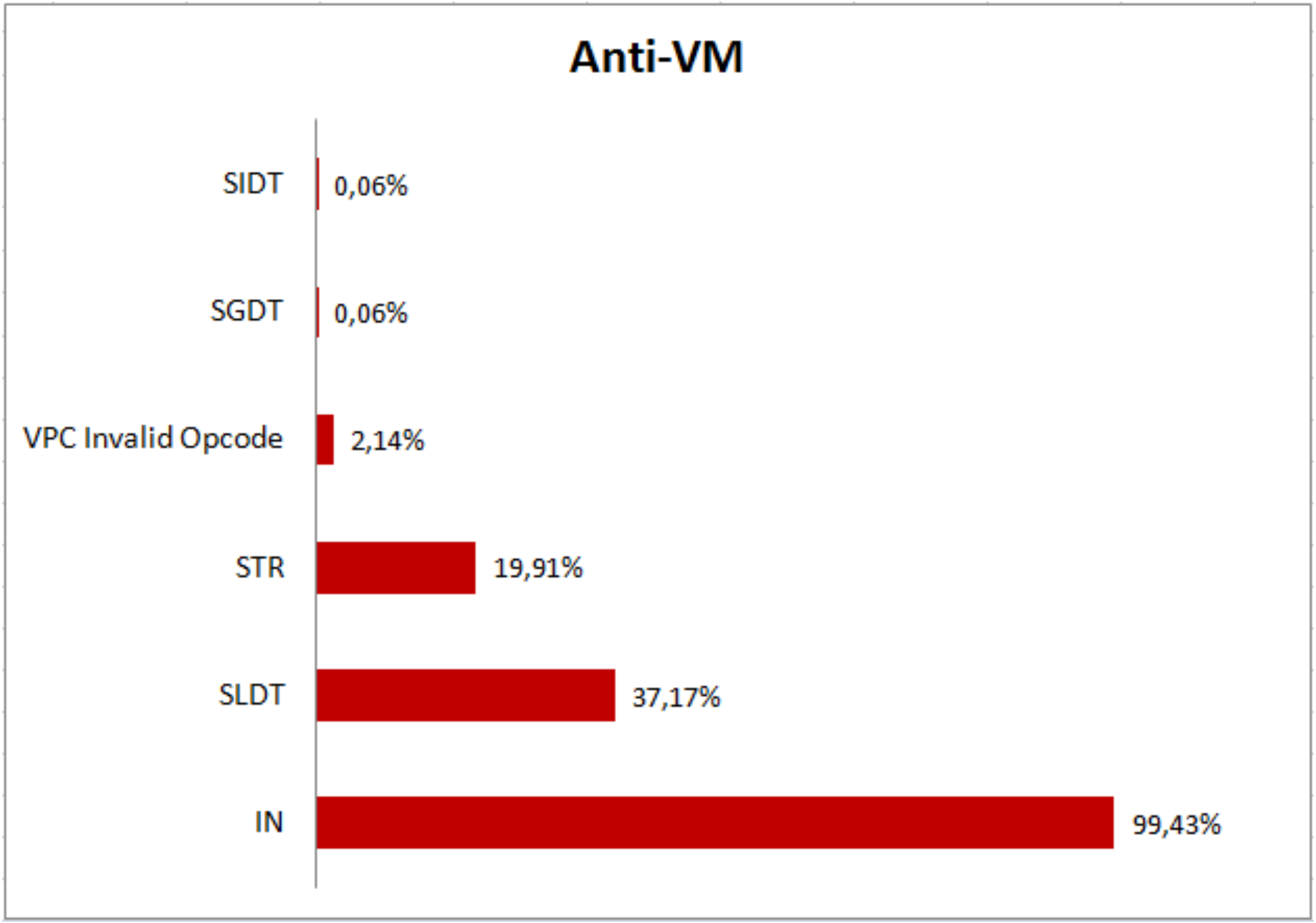
Protected Samples



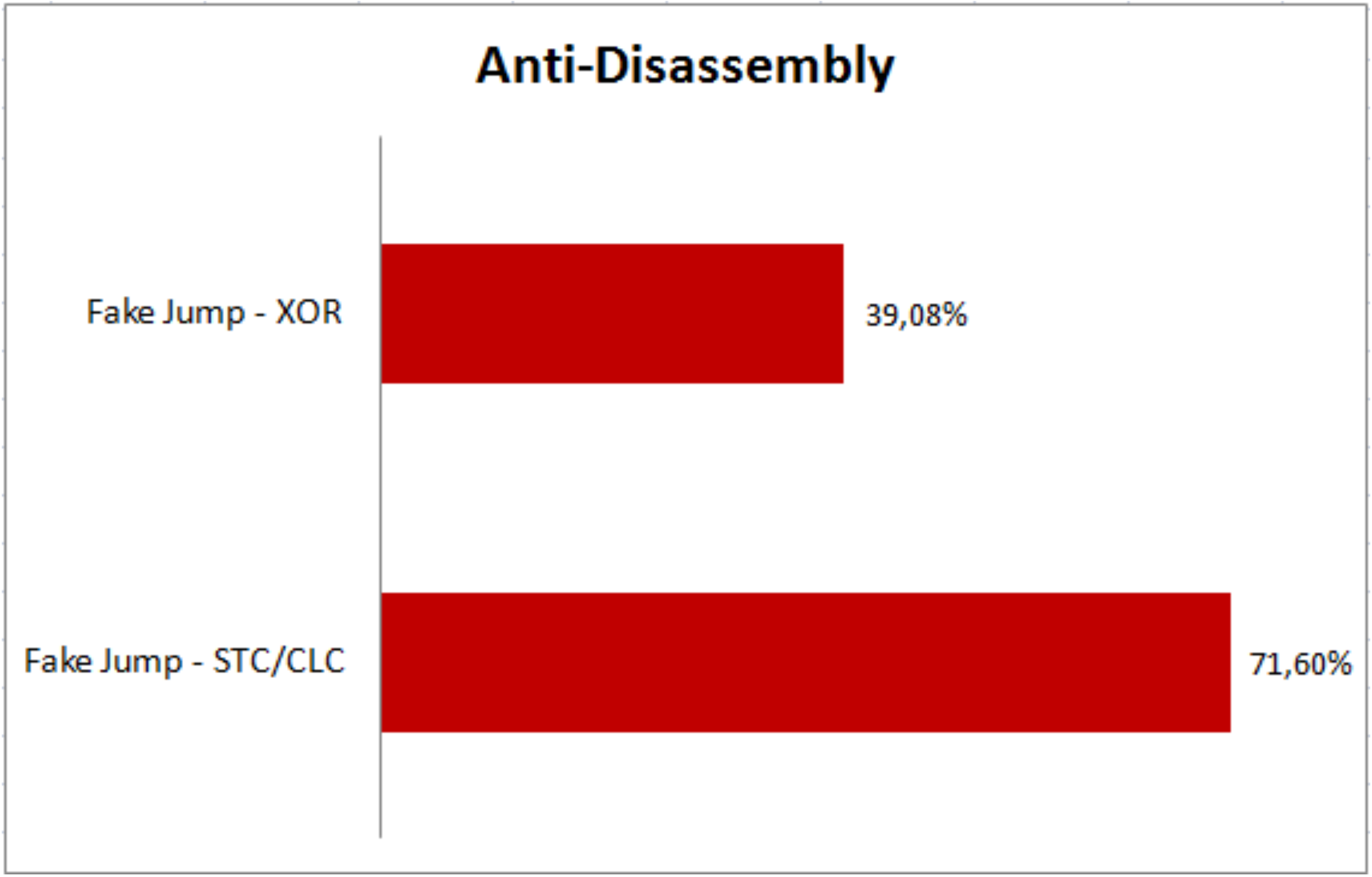
Anti-RE Categories



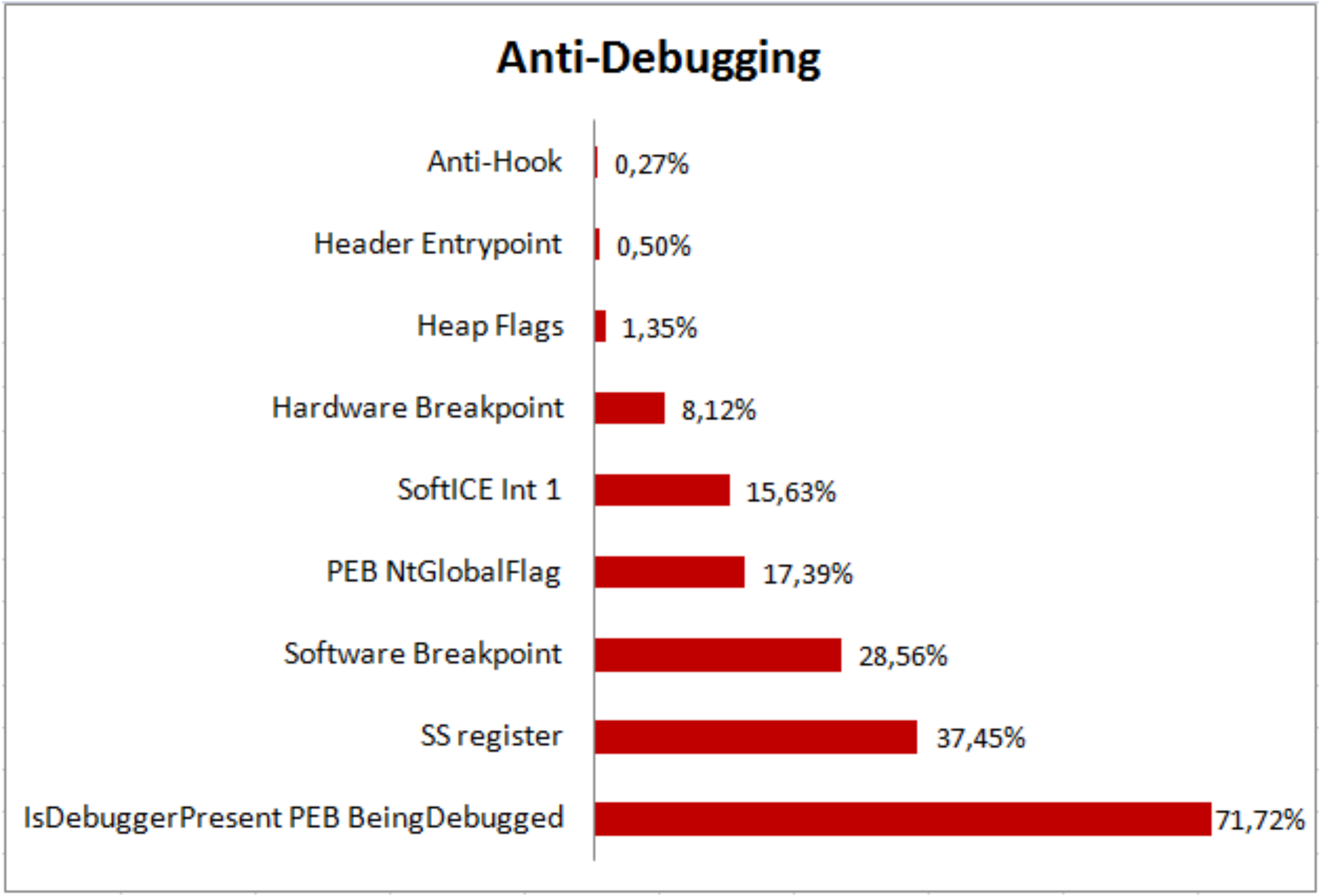
Anti-VM


Anti-VM
SIDT
0,06%
SGDT
0,06%
VPC Invalid Opcode
2,14%
STR
19,91%
SLDT
37,17%
IN
99,43%

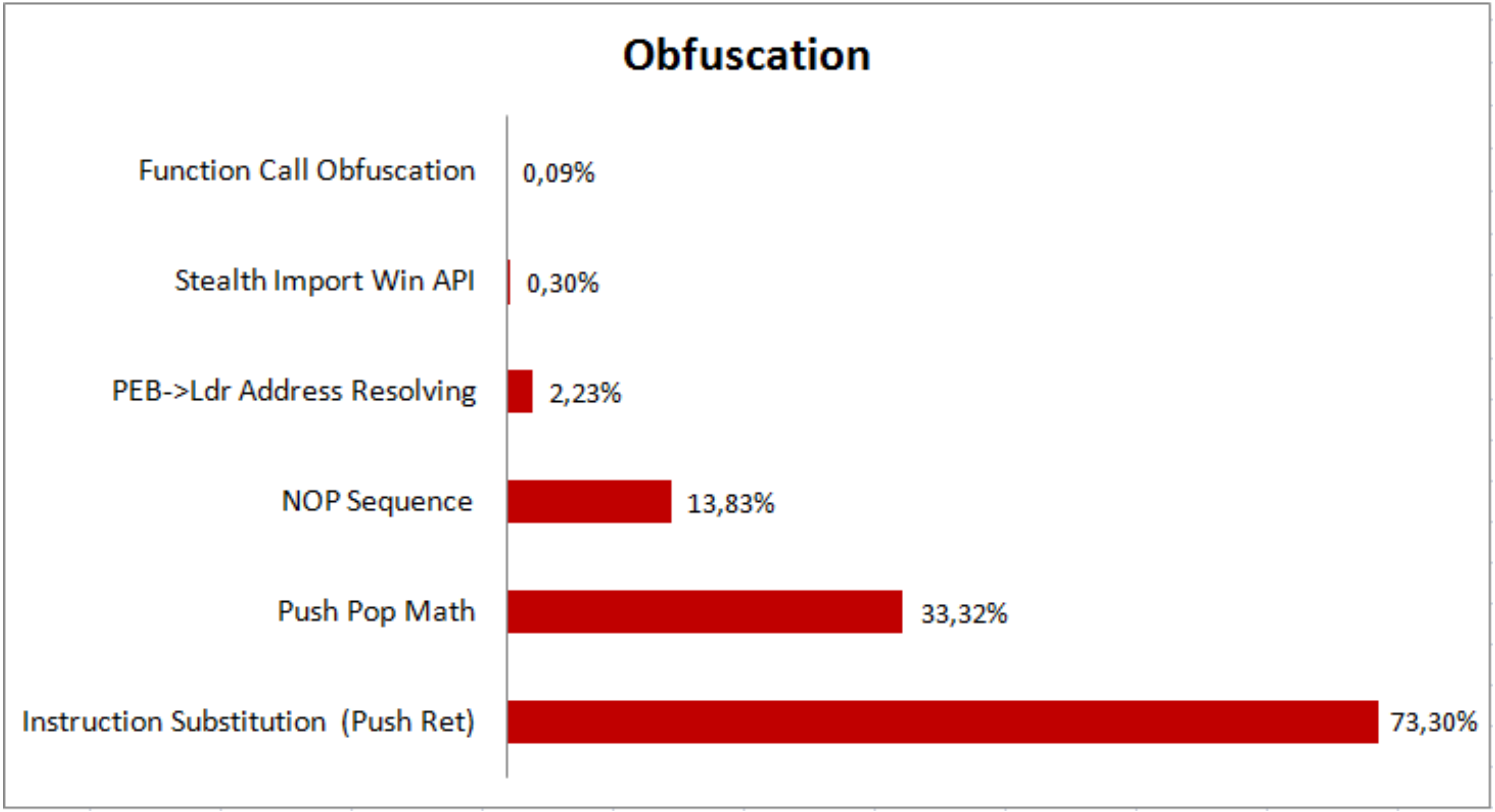
Anti-Disassembly



Anti-Debugging



Obfuscation



Anti-Debugging Techniques

- 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

- 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

- 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

- 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)

New Techniques

- We understand that malware is quickly evolving, thus there is a need for analysis to go at least as fast
- SSEXY
  - SSE obfuscation tool released in Hack in The Box Amsterdam (17-20 of May) by Jurriaan Bremer
  - In June we already had a plugin to detect it
- Flame
  - The industry positioned it as completely new, embedding a LUA interpreter for rapid development of new capabilities
  - We implemented a plugin for the detection of embedded LUA as soon as the news came out and we can TELL you that there is no other malware containing LUA
    - We do not have to assume it as we have analysis results
- Google's "Go"
  - On September, 18, Flora Liu posted in the Symantec blog they were able to find a Malware using the Google's language 'GO'. We wrote a plugin to check if this was the case for any other malware samples and can tell you this was not

New Techniques

- On July, 25, 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)

Resources

- Sample code for the different techniques we detect are available on github:
  - https://github.com/rrbranco/blackhat2012

- Updated versions of the paper and presentation are going to be available at:
  - http://research.dissect.pe

Resources – Portal

- Portal URL: http://www.dissect.pe
- Any interested researcher / contributor / journalist can have access to the portal (drop me an email)
- We are constantly updating the statistics and developing/ improving analysis algorithms

Conclusions

- We analyzed millions of malware samples and showed scientific results about their usage of protection techniques
- There are more techniques to implement and some algorithms to improve
  - We still have a lot to do... and so do you! Help us!
- The portal (www.dissect.pe) is always updated with new and better results:
  - More detection techniques
  - More analyzed samples

HITBSecConf
Keeping Knowledge Free for Over a Decade

THE END ! Really !?

Rodrigo Rubira Branco (BSDaemon)
http://twitter.com/bsdaemon
rodrigo *noSPAM* kernelhacking.com