Analysis, Anti-Analysis, Anti-Anti-Analysis: An Overview of the Evasive Malware Scenario

Marcus Botacin[1], Vitor Falcão da Rocha(1), André Grégio[1,2], Paulo Lício de Geus[1]

[1]Institute of Computing - UNICAMP
{marcus,vitorf,paulo}@lasca.ic.unicamp.br

[2]Federal University of Paraná (UFPR)
gregio@inf.ufpr.br

November 7^{th}, 2017

Topics

1. Part I
   - Analysis and Anti-analysis
2. Part II
   - Tricks and detection methods
3. Part III
   - Tests and Results
4. Part IV
   - Concluding Remarks

Topics

Arms-Race

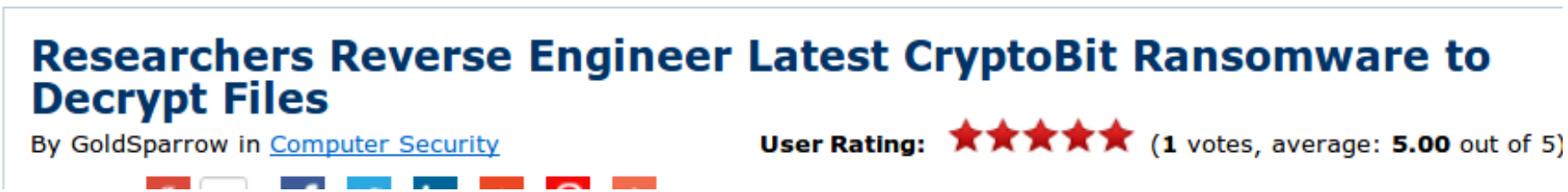

Figura: Enigma: `https://tinyurl.com/kydgwve`

Figura: Themerkle: `https://tinyurl.com/kasuxcr`

Topics

APIs and direct calls.

Attackers I

Listagem 1: Attackers.

```
1 do_malicious();
```

Analysts I

Listagem 2: Analysts.

```
1 if do_malicious is present;
2     detect()
```

APIs and direct calls.

Attackers II

Listagem 3: Attackers.

```
1 if !is_debug():
2     do_malicious();
```

Analysts II

Listagem 4: Analysts.

```
1 if is_debug() is present;
2     detect()
```

APIs and direct calls.

Attackers III

Listagem 5: Attackers.

```
1 typedef struct _PEB {
2 BYTE Reserved1[2];
3 BYTE BeingDebugged;
4 BYTE Reserved2[1];
```

Attackers III

Listagem 6: Analysts.

```
1 mov eax, [fs:0x30]
2 mov eax, [eax+0x0c]
```

APIs and direct calls.

Analysts III

Listagem 7: Analysts.

```
1 def check():
2   if instruction in ['mov', 'movsx', 'movzx']:
3     if 'fs:0x30' in op2:
4         self.found_op1 = op1
5         self.found_keyword = True
6   if self.found_keyword:
7       if instruction in ['cmp', 'cmpxchg', '
          mov', ...]:
8         if '[' + self.found_op1 + '+0xc]' or
              ...:
9       print "Detected!"
```

Anti-Disassembly.

Technique	Description	Detection
PUSH POP MATH	`PUSH` and `POP` a value on/from the stack instead of using a direct `MOV`	Detect a sequence of `PUSH` and `POP` on/from a register.
PUSH RET	`PUSH` a value on the stack and `RET` to it instead of the ordinary return.	Detect a sequence of `PUSH` and `RET`
LDR address resolving	Get loaded library directly from the PEB instead of using a function call	Check memory access referring the PEB offset.
Stealth API import	Manually resolving library imports instead of directly importing them.	Check for a sequence of access/compares of PEBs offsets.
NOP sequence	Breaks pattern matching by implanting NO-OPerations	Detect a sequence of `NOPs` within a given window
Fake Conditional	Create an always-taken branch	Check for branch-succeded instructions which set branch flags
Control Flow	Changing control flow within an instruction block	Check for the `PUSH-RET` instruction sequence
Garbage Bytes	Hide data as instruction code	Check for branch-preceeded data

Anti-Debug.

Technique	Description	Detection
Known Debug API	Call a debug-check API	Check for API imports
Debugger Fingerprint	Check the presence of known debugger strings	CHeck known strings inside the binary
NtGlobalFlag	Check for flags inside the PEB structure	Check for access on the PEB offset
IsDebuggerPresent	Check the debugger flag on the PEB structure	Check access to PEB on the debugger flag offset
Hook Detection	Verify whether a function entry point is a `JMP` instruction	Check for a `CMP` instruction having `JMP` opcode as an argument
Heap Flags	Check for heap flags on the PEB	check for heap checks involving PEB offsets
Hardware Breakpoint	Check whether hardware breakpoint registers are not empty	Check for access involving the debugger context
SS Register	Insert a check when interruptions are disabled	Check for SS register's `POPs`
Software Breakpoint	Check for the `INT3` instruction	Check for `CMP` with `INT3`
SizeOfImage	Change code image field	Check for PEB changes.

Anti-VM.

Technique	Description	Detection
VM Fingerprint	Check for known strings, such as serial numbers	Check for known strings inside the binary
CPUID Check	Check CPU vendor	Check for known CPU vendor strings
Invalid Opcodes	Launch hypervisor-specific instructions	Check for specific instrutions on the binary
System Table Checks	Compare IDT values	Look for checks involving IDT
HyperCall Detection	Platform specific feature	Look for specific instructions

Topics

Sections

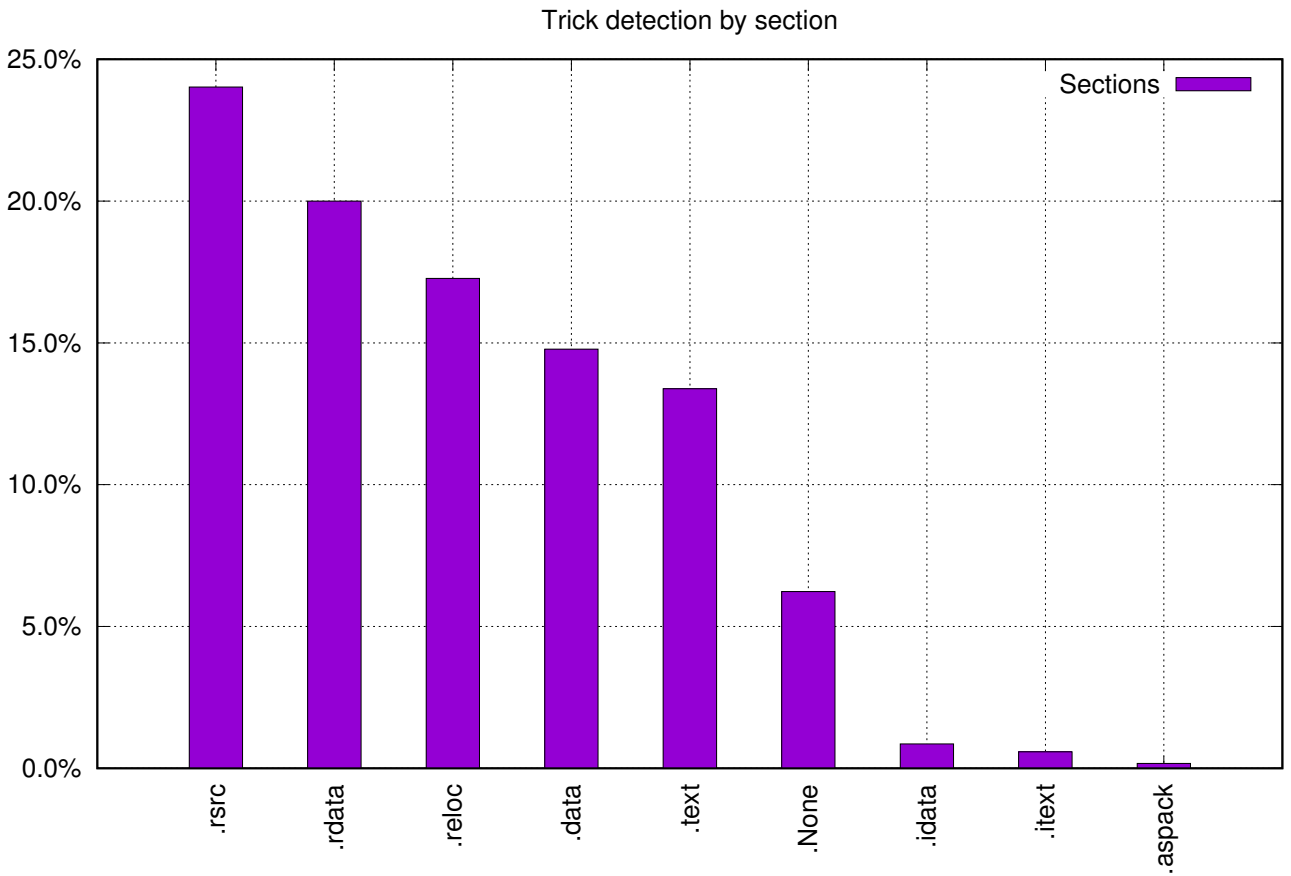

Figura: Tricks by section.

Sections

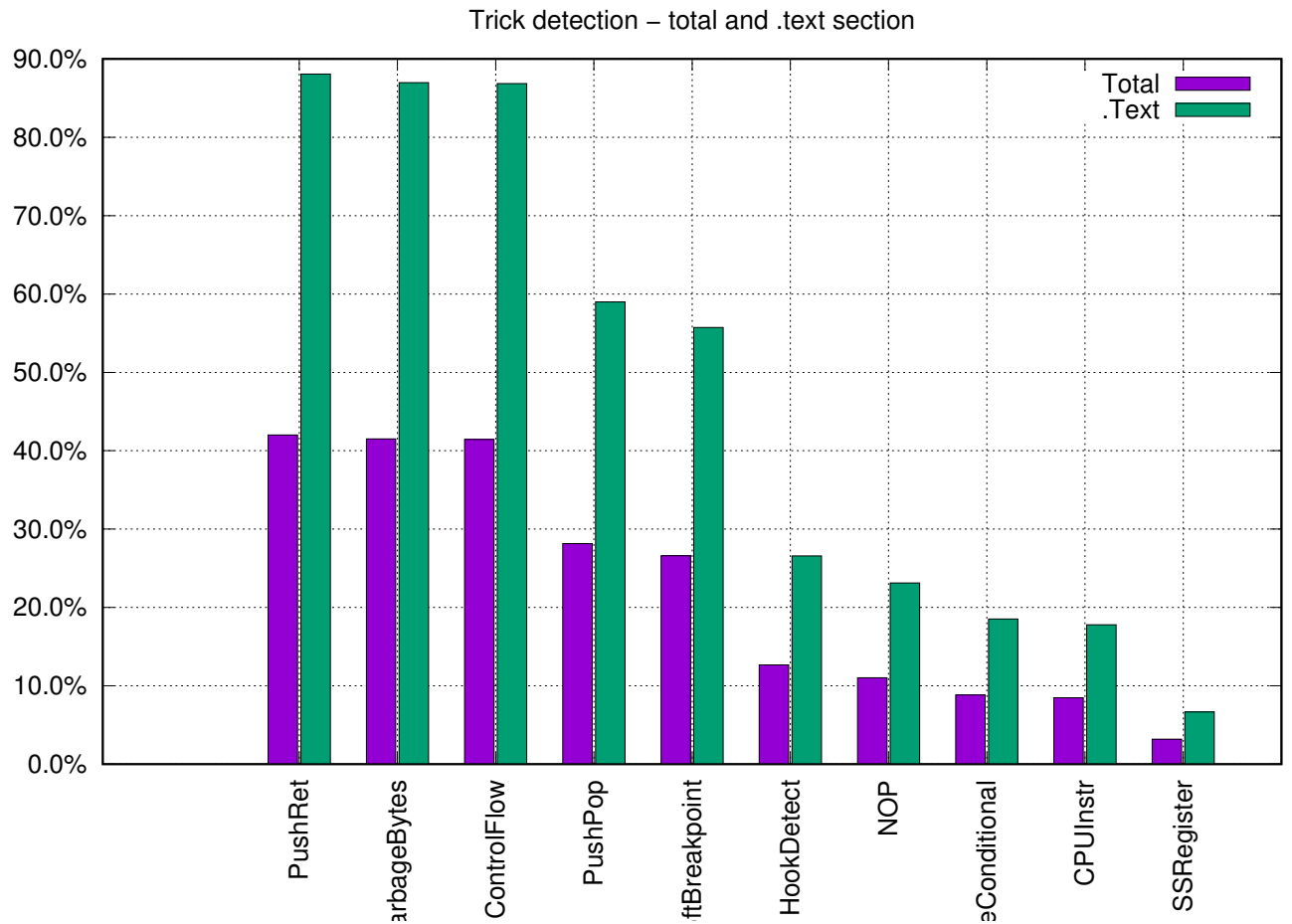

Figura: Tricks - total and `.text` section.

Packers

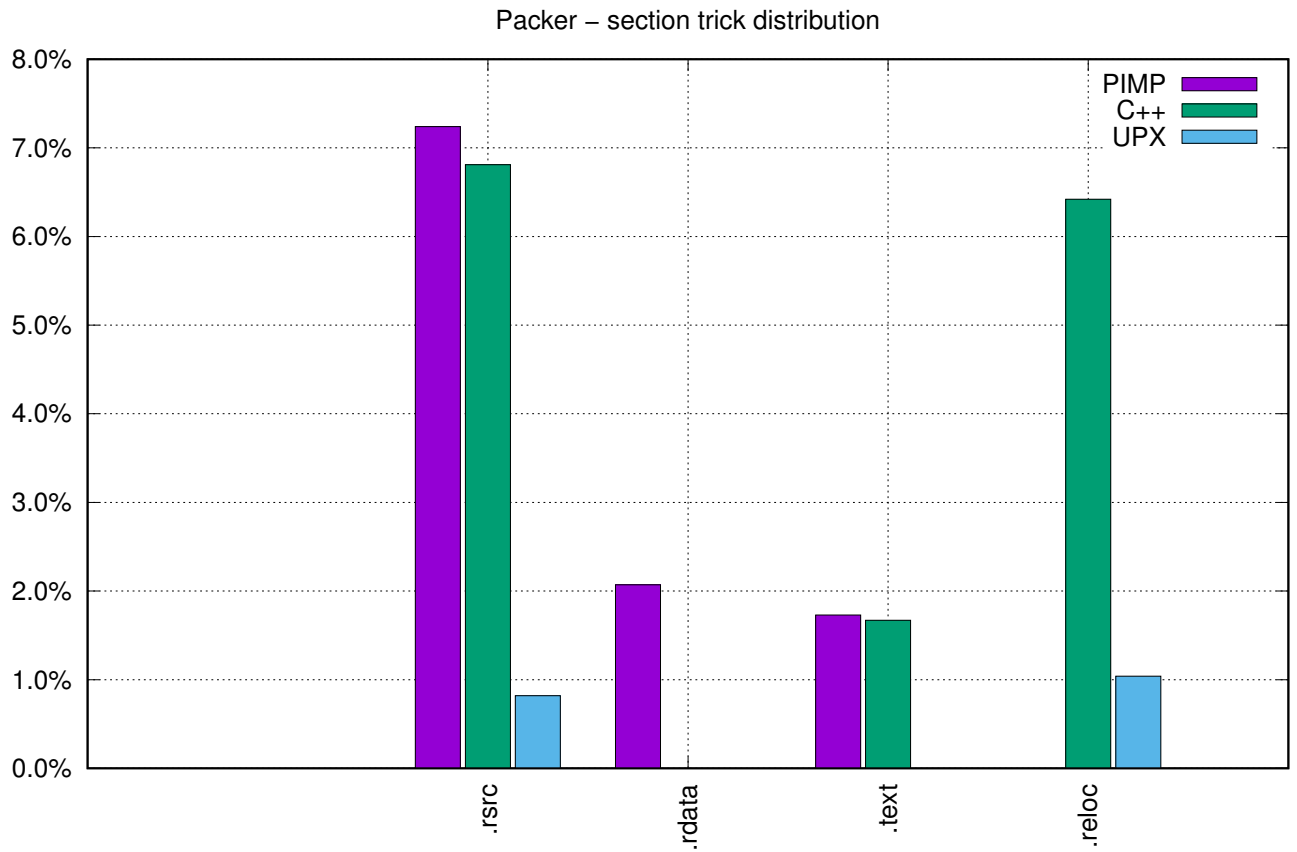

Figura: Packer distribution across binary sections.

Packers

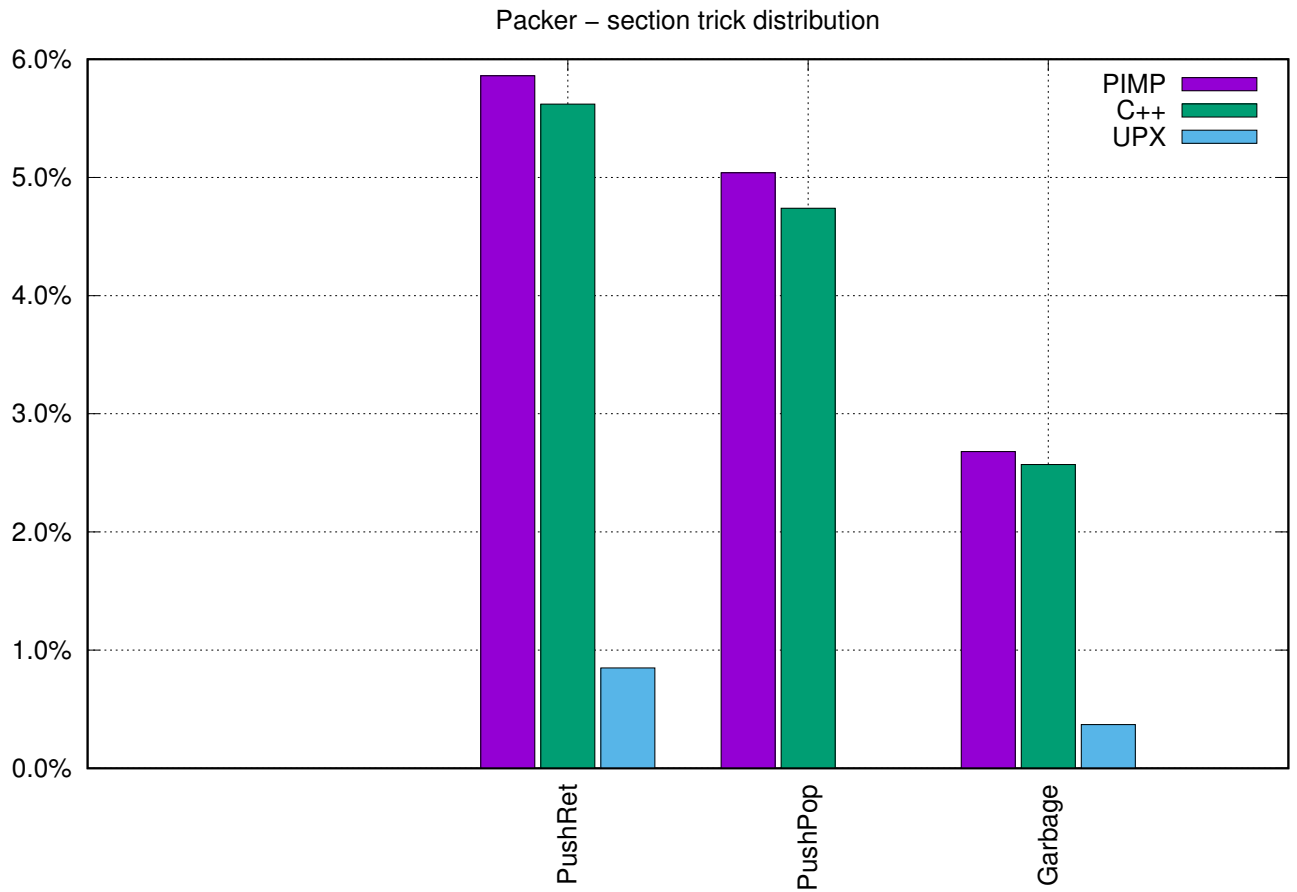

Figura: Tricks detected on distinct packers.

Packers

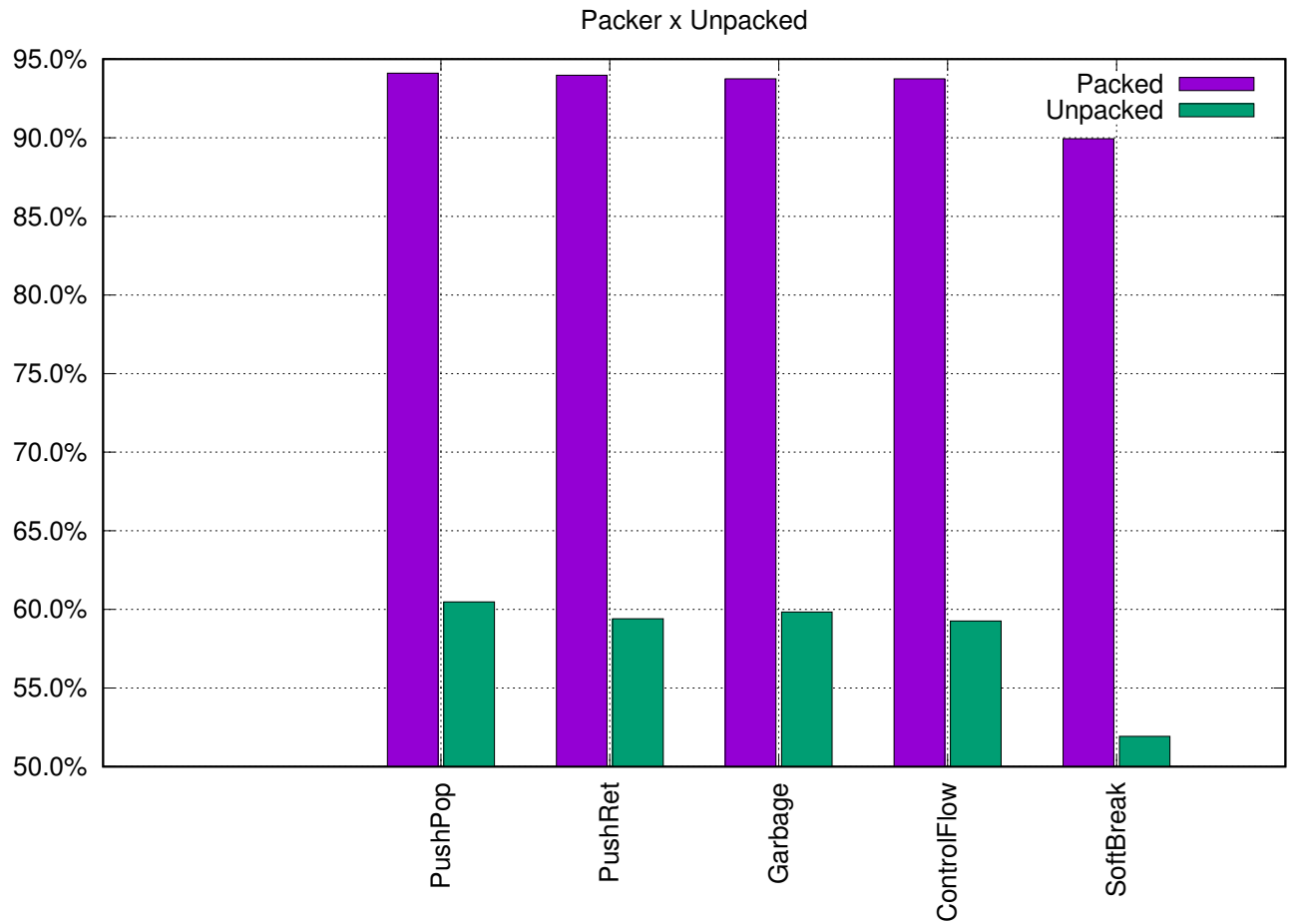

Figura: Packer influence on trick detection.

Malware and Goodware

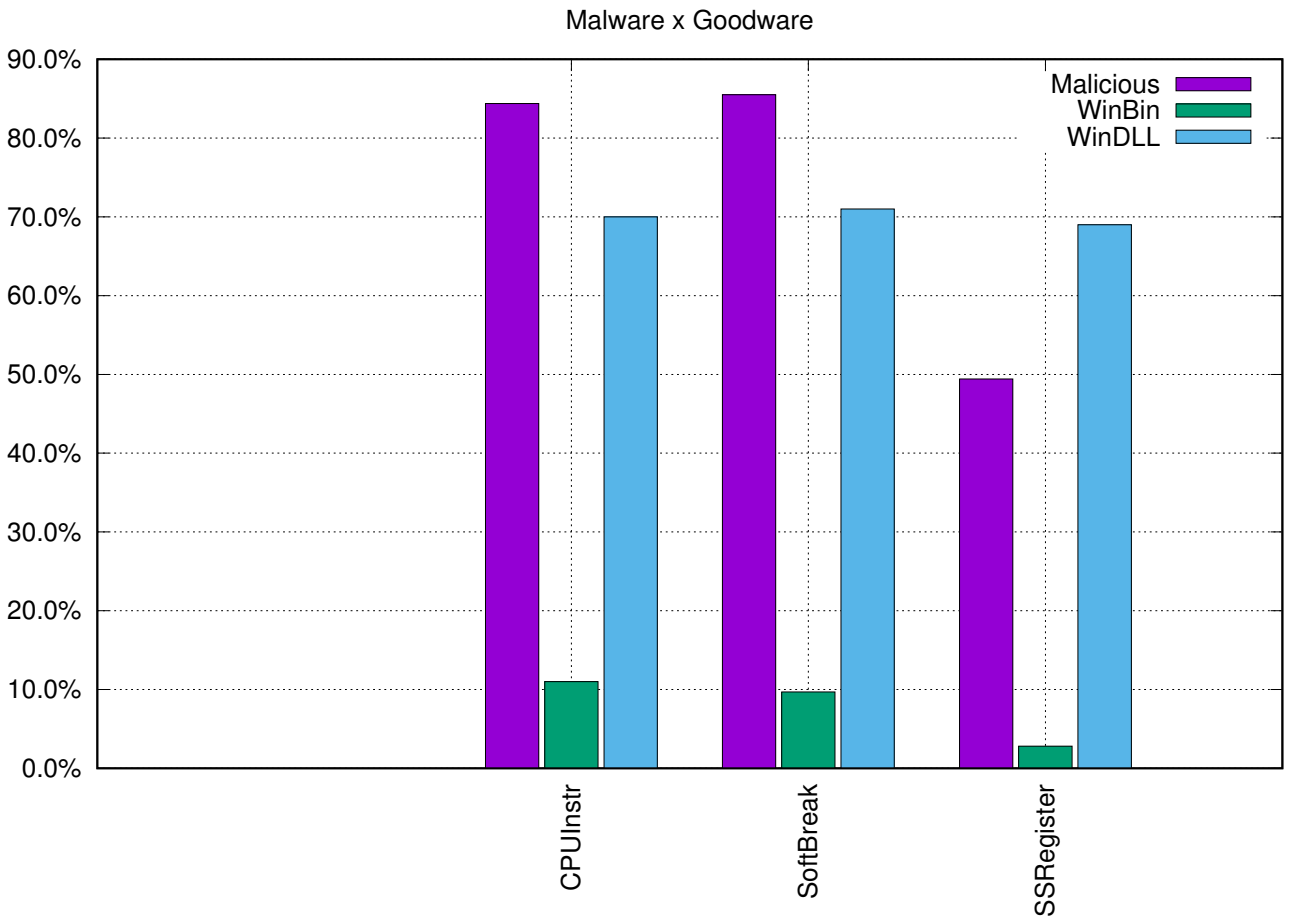

Figura: Tricks detection on malware and goodware.

Comparing Scenarios

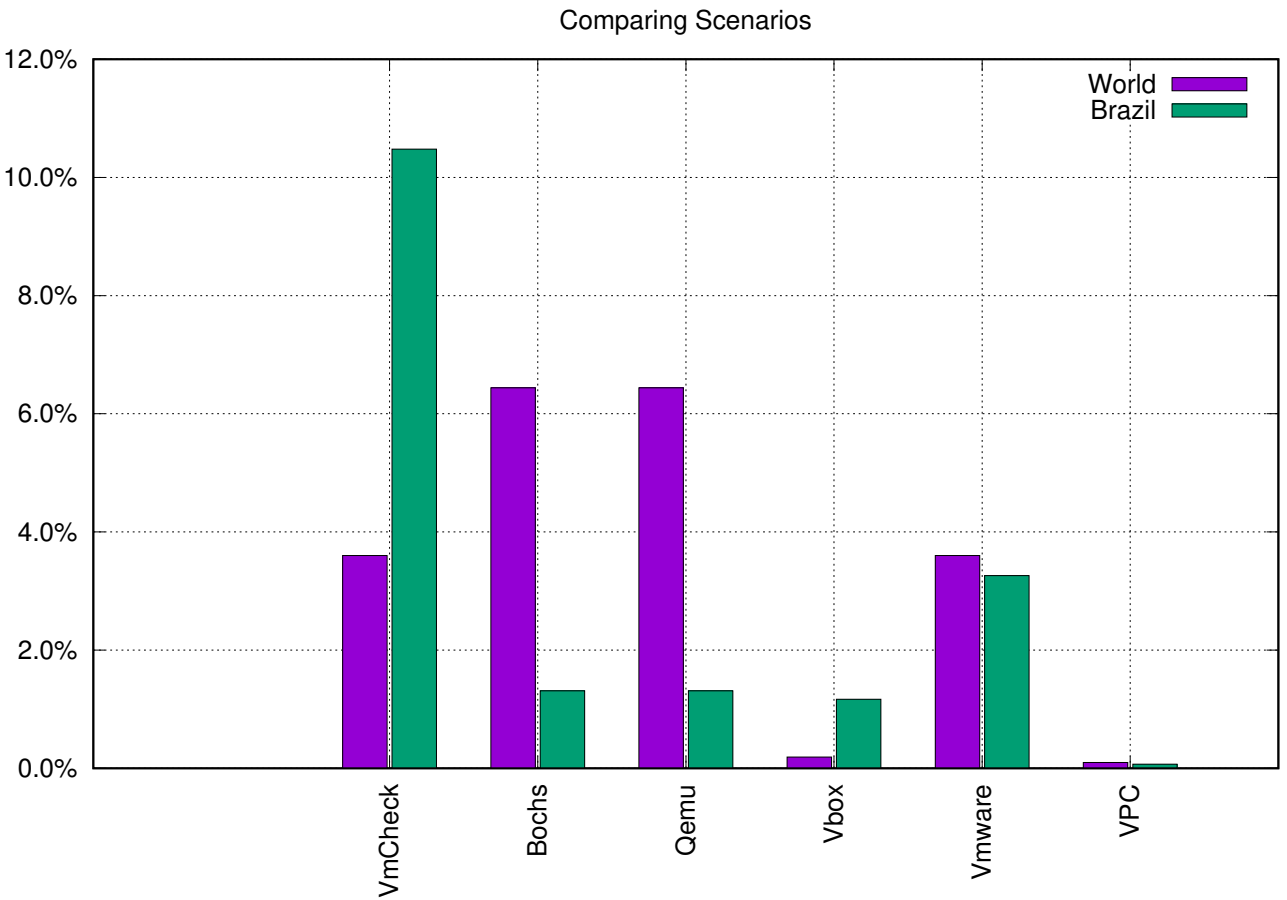

Figura: Comparing scenarios: PEframe detection.

Comparing Scenarios

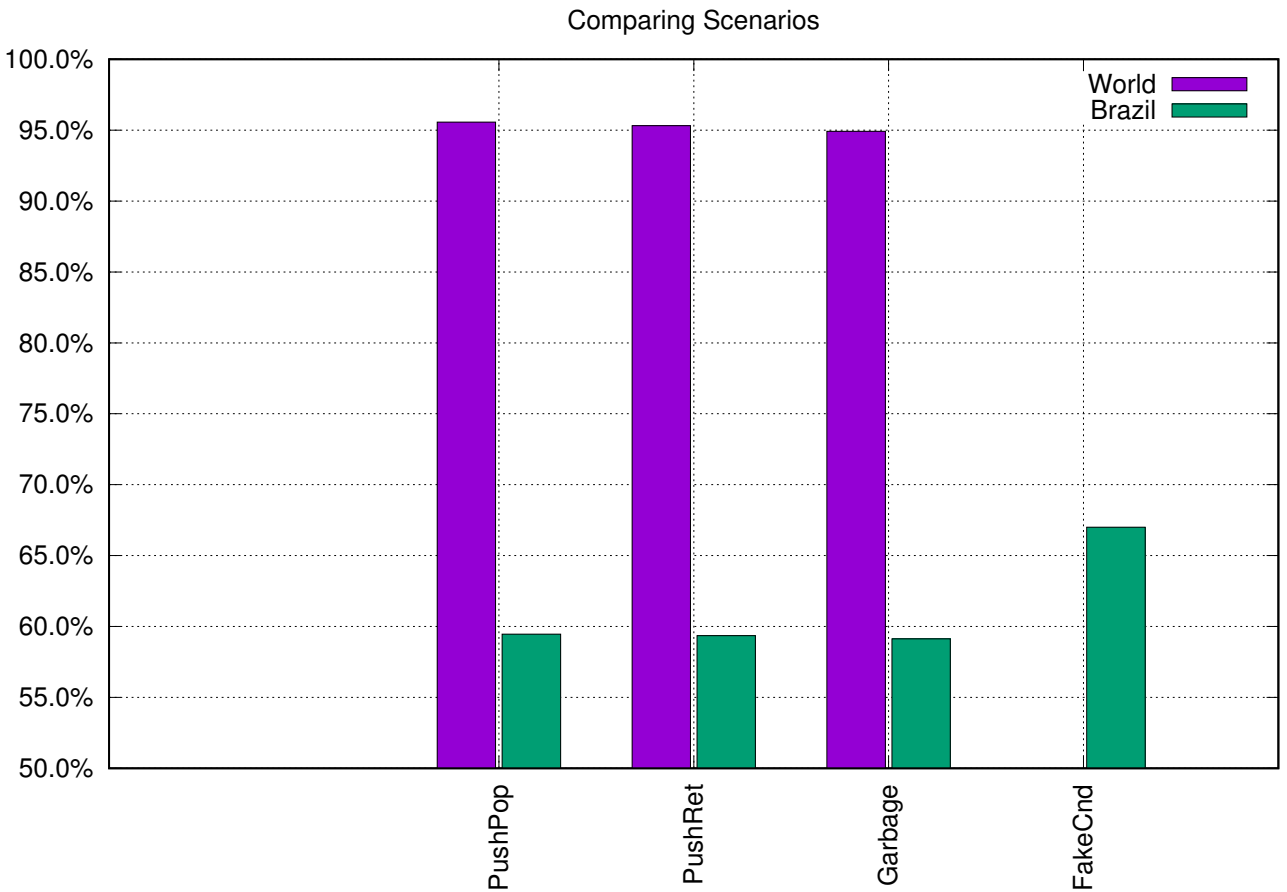

Figura: Comparing scenarios: Tricks detection.

Comparing Scenarios

More about the brazilian scenario

- Uma Visão Geral do Malware Ativo no Espaço Nacional da Internet entre 2012 e 2015 - SBSEG 2015
  Marcus Botacin, André Grégio, Paulo Lício de Geus
  (`http://siaiap34.univali.br/sbseg2015/anais/WFC/artigoWFC02.pdf`)

Trick Blocking

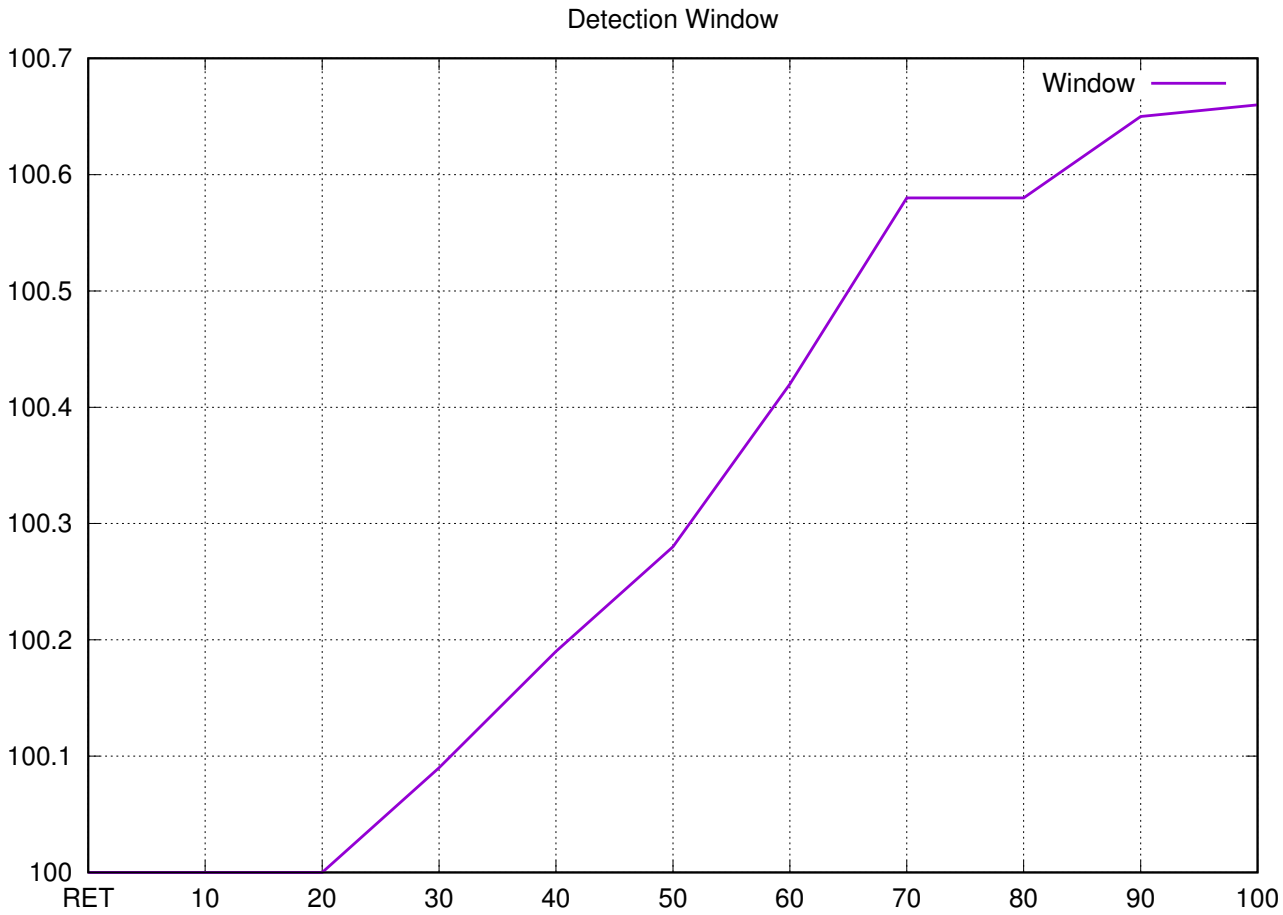

Figura: Evaluating block window effect on trick detection.

Trick Alignment

Tabela: Evaluating the occurrence of misaligned tricks.

Trick	Aligned	Unaligned
CPU	182	287
FakeJMP	63	203

Compiler-based evasion

Tabela: Compilation-based evasion.

ShellCode	Unarmored	ROPinjector
1[1]	4/57	0/57
2[2]	15/58	0/57
3[3]	9/57	0/54
4[4]	7/58	0/54
5[5]	9/53	0/53

[1] http://shell-storm.org/shellcode/files/shellcode-898.php
[2] http://shell-storm.org/shellcode/files/shellcode-874.php
[3] http://shell-storm.org/shellcode/files/shellcode-627.php
[4] http://shell-storm.org/shellcode/files/shellcode-568.php
[5] http://shell-storm.org/shellcode/files/shellcode-714.php

AV Detection

Shellcode	SC1		SC2		SC3	
Technique	W/o Trick	W/ Trick	W/o Trick	W/ Trick	W/o Trick	W/ Trick
Fakejmp	10/58	6/57	20/58	17/58	15/58	10/57
PushRet		7/57		17/58		10/58
NOP		6/57		17/57		10/58

Topics

Summary

Summary

- Arms race.
- Static detection can help detection.
- Tricks can be improved.
- Static analysis is theoretically limited.

Concluding Remarks

Handling anti-analysis dynamically

- Análise Transparente de Malware com Suporte por Hardware - SBSEG 2016
  Marcus Botacin, Paulo Lício de Geus, André Grégio
  (`http://sbseg2016.ic.uff.br/pt/files/anais/completos/ST8-3.pdf`)

Agradecimentos

- CNPq, pelo financiamento via Proj. MCTI/CNPq/Universal-A edital 14/2014 (Processo 444487/2014-0)
- CAPES, pelo financiamento via Proj. FORTE - Forense Digital Tempestiva e Eficiente (Processo: 23038.007604/2014-69).
- Instituto de Computação/Unicamp
- Departamento de Informática/UFPR

Contato:

marcus@lasca.ic.unicamp.br
vitorf@lasca.ic.unicamp.br
paulo@lasca.ic.unicamp.br
gregio@inf.ufpr.br