Evasion and Countermeasures Techniques to Detect Dynamic Binary Instrumentation Frameworks

Ailton Santos Filho[†], **Ricardo J. Rodríguez**[‡], Eduardo L. Feitosa[†]

[†]Institute of Computing
Federal University of Amazonas, Brazil

[‡] Dept. of Computer Science and Systems Engineering
University of Zaragoza, Spain

March 10, 2022

RootedCon 2022
Madrid, Spain

RESEARCH-ARTICLE OPEN ACCESS

Evasion and Countermeasures Techniques to Detect Dynamic Binary Instrumentation Frameworks

Authors: Ailton Santos Filho, Ricardo J. Rodríguez, Eduardo L. Feitosa Authors Info & Claims

Digital Threats: Research and Practice, Volume 3, Issue 2 • June 2022 • Article No.: 11, pp 1–28 • https://doi.org/10.1145/3480463

Online: 08 February 2022 Publication History

doi: 10.1145/3480463

Ailton Santos **Ricardo J. Rodríguez** Eduardo L. Feitosa

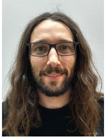
Universidad Zaragoza

$whoami

- **Associate Professor at the University of Zaragoza**
- **Research lines**:
  - Program binary analysis
  - Digital forensics
  - Offensive security
  - Security and survivability analysis with formal models

$whoami

- **Associate Professor at the University of Zaragoza**
- **Research lines**:
  - Program binary analysis
  - Digital forensics
  - Offensive security
  - Security and survivability analysis with formal models
- **Research team** – *we make really good stuff!* ☺
  - `https://reversea.me`
  - `https://twitter.com/reverseame/`
  - `https://t.me/reverseame`



Miguel Martín-Pérez
PhD. student

Daniel Uroz
PhD. student

Razvan Raducu
PhD. student

Pedro Fernández
Technician

Universidad Zaragoza

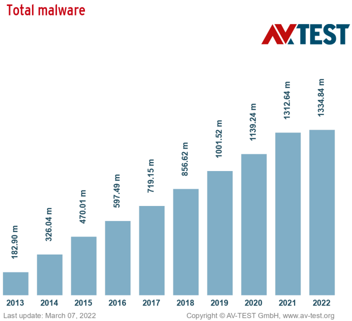
Agenda

Outline

Introduction

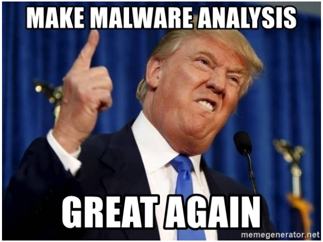
Malicious software (malware) is still an issue...

- 360,000 malware samples analyzed per day in 2017 (Kaspersky)
- 50M malicious samples in the last quarter of 2018 (McAfee Labs)
- +294M targeting Windows only in 2019 (Ugarte-Pedrero et al., 2019)



Credits: `https://www.av-test.org/en/statistics/malware/`

Introduction



Malware analysis

- Determine *what the heck* the malware does as harmful activities
- **Static analysis**
  - Executable files are analyzed without being executed
  - Shortcomings: binary obfuscation (packing, opaque predicates, etc.)
- **Dynamic analysis**
  - Executable files are analyzed when run
  - Shortcomings: very costly (time-consuming)

Universidad Zaragoza

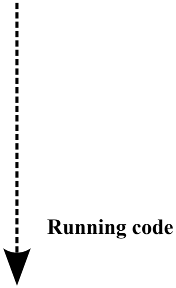
Introduction

Evasive malware

- **Malware capable of detecting analysis environments**
  - Malware changes its behavior when recognizes it, avoiding any malicious action
  - *The longer the malware goes unnoticed, the more revenue cybercriminals earn*
- **Different terminology (and means) in the literature**:
  - Analysis-aware malware (Balzarotti et al., 2010; Rodríguez et al., 2016)
  - Evasive malware (Polino et al., 2017; Ekenstein and Norrestam, 2017)

Introduction
What is Dynamic Binary Instrumentation?

- **Technique for the dynamic analysis of programs**
- **Adding arbitrary code when running an application**
  - Addition of arbitrary code: *instrumentation*
  - During program execution: *dynamic*
  - On the application: *binary*

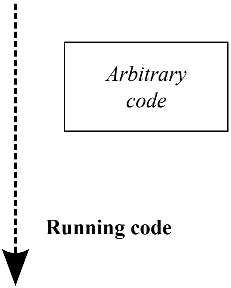


Universidad Zaragoza

Introduction

What is Dynamic Binary Instrumentation?

- **Technique for the dynamic analysis of programs**
- **Adding arbitrary code when running an application**
  - Addition of arbitrary code: *instrumentation*
  - During program execution: *dynamic*
  - On the application: *binary*

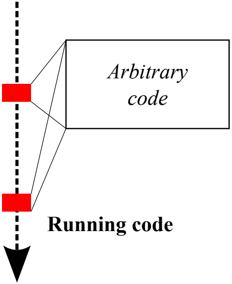


Universidad Zaragoza

Introduction

What is Dynamic Binary Instrumentation?

- **Technique for the dynamic analysis of programs**
- **Adding arbitrary code when running an application**
  - Addition of arbitrary code: *instrumentation*
  - During program execution: *dynamic*
  - On the application: *binary*



Introduction

How Dynamic Binary Instrumentation works?

Different elements

- **DBI engine**
- **Dynamic Binary Analysis tool**
- **DBI framework**
  - Just-In-Time (JIT) compiler
  - Intercepts the execution of the first instructions of the client application
  - Generates a new assembly code directly from the subsequent instructions at runtime
  - The resulting code contains the code to redirect the execution to the analysis code
  - Generally, this code is allocated in a code cache (to eventually reuse it)

Introduction
DBI and security

- **Used in various security solutions**
  - Dynamic taint analysis, malware unpacking, and VM transparency enhancement
  - Detection of anti-instrumentation techniques in evasive malware
- **Malware is actually using some sort of anti-instrumentation techniques** (15.6% of 7K samples used at least one; Polino et al., 2017)
- **Tools to mitigate specific evasive techniques**
  - `PinVMShield`
  - `Arancino`
  - ...

Universidad Zaragoza

Are DBI-based tools adequate tools for malware analysis?

- **Recently questioned by several researchers in the community**
- **Why?**

Are DBI-based tools adequate tools for malware analysis?

- **Recently questioned by several researchers in the community**
- **Why?**
  1. DBI-based tools can be detected through specific evasion
  2. More attack surface increases the probability of exploitation

Are DBI-based tools adequate tools for malware analysis?

- **Recently questioned by several researchers in the community**
- **Why?**

1. DBI-based tools can be detected through specific evasion
2. More attack surface increases the probability of exploitation

Contributions

- **Review of anti-instrumentation and countermeasures techniques**
- **New taxonomy of evasion techniques**
- **Highlight areas of interest for future work and open issues**

Outline

Methodology

Planning step

Research questions

- **Q1** *What are the anti-instrumentation techniques proposed in the literature?*
- **Q2** *What are the proposed countermeasures (if any) to mitigate the anti-instrumentation techniques and thus improve the reliability of DBI frameworks?*

Universidad Zaragoza

Methodology
Planning step

Research questions

Q1 *What are the anti-instrumentation techniques proposed in the literature?*

Q2 *What are the proposed countermeasures (if any) to mitigate the anti-instrumentation techniques and thus improve the reliability of DBI frameworks?*

Search strategies

- Search for articles in the digital library
  - ACM Digital Library, Science Direct, SpringerLink, and IEEEXplore Digital Library
  - Manually scrutinized DBLP of top-notch conferences not indexed in these search databases (e.g., NDSS and USENIX Security)
  - Search string terms: *DBI*, *evasion*, and *malware* (+ alternative terms and synonyms)
- Availability of the consulted articles
- Articles are available in English, in whole or in part

Universidad Zaragoza

Methodology

Planning step – inclusion (IC) and exclusion (EC) criteria

#	Criterion
IC1	Articles that discuss evasive techniques applicable to DBI frameworks, malware embedded with these techniques, or countermeasures.
IC2	Articles that discuss concepts of dynamic binary instrumentation or characteristics of the DBI frameworks, related to evasive techniques.
EC1	Articles in which the language is different from English or Spanish cannot be selected.
EC2	Articles that are not available for reading and data collection (articles that are only accessible through pay-walls or are not provided by the search engine) cannot be selected.
EC3	Duplicate articles cannot be selected.
EC4	Publications that do not meet any of the inclusion criteria cannot be selected.

Universidad Zaragoza 1542

Methodology
Conducting step

- Preliminary search: **483 articles**
  - 391 from ACM, 91 from Science Direct, 6 from SpringerLink, and only 1 from IEEEXplore
- IC & EC criteria: **57 articles**
- Full-text reading: **7 articles**
- Snowball search on these articles: **10 more artifacts**
  - 5 articles
  - 4 gray research papers
  - 1 tool

Outline

Anti-Instrumentation and Countermeasures Techniques

Towards a new taxonomy

- **6 articles propose taxonomies for DBI evasive techniques**
  - Rodríguez et al. (2016), Sun et al. (2016), Polino et al. (2017), Kirsch et al. (2018), Zhechev (2018), D'Elia et al. (2019)
- **They describe similar concepts**
- Some taxonomy focuses exclusively on Pin, others focus on Pin and DynamoRIO, and others are more general classifications

Universidad Zaragoza

Anti-Instrumentation and Countermeasures Techniques

Towards a new taxonomy

- **6 articles propose taxonomies for DBI evasive techniques**
  - Rodríguez et al. (2016), Sun et al. (2016), Polino et al. (2017), Kirsch et al. (2018), Zhechev (2018), D'Elia et al. (2019)
- **They describe similar concepts**
- Some taxonomy focuses exclusively on Pin, others focus on Pin and DynamoRIO, and others are more general classifications

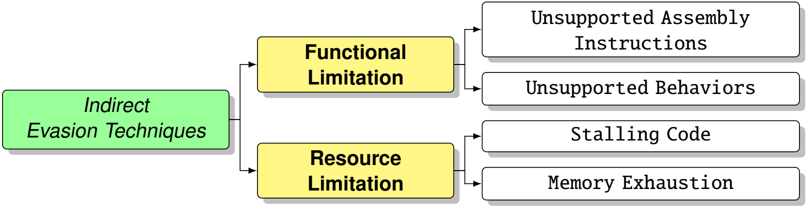
New taxonomy

- **More general, independent of the DBI framework**
- **Direct and indirect nature of anti-instrumentation techniques**
  - Direct: the evasion technique **does** incorporate code artifacts to detect DBI frameworks
  - Indirect: the evasion technique **does not** incorporate any code artifacts

Universidad Zaragoza

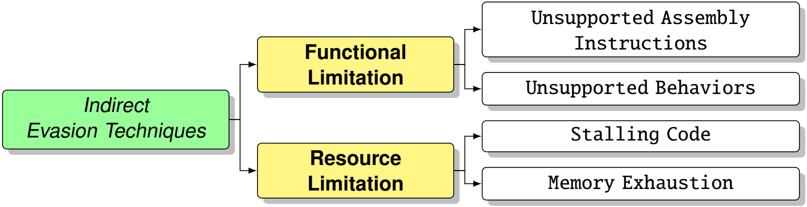
Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques



Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques

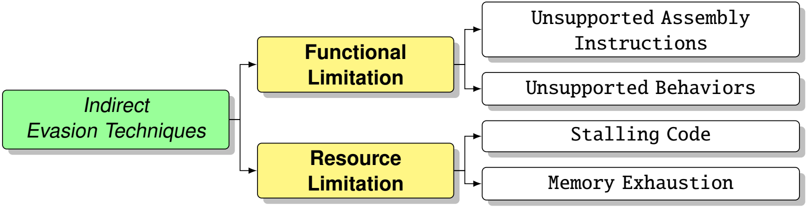


Functional limitation

- Behavioral inconsistencies between bare-metal systems and analysis systems due to lack of handling or implementation of certain behaviors
- Examples: `retf`, `enter`, using the heap as the stack, multi-threading

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques



Functional limitation

- Behavioral inconsistencies between bare-metal systems and analysis systems due to lack of handling or implementation of certain behaviors
- Examples: `retf`, `enter`, using the heap as the stack, multi-threading

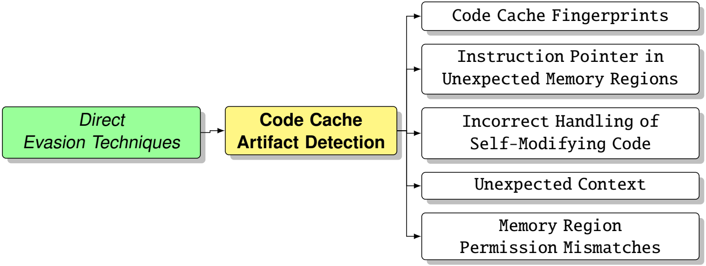
Resource limitation

- Analysis environments have limited processing resources

Universidad Zaragoza

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques – direct evasion techniques

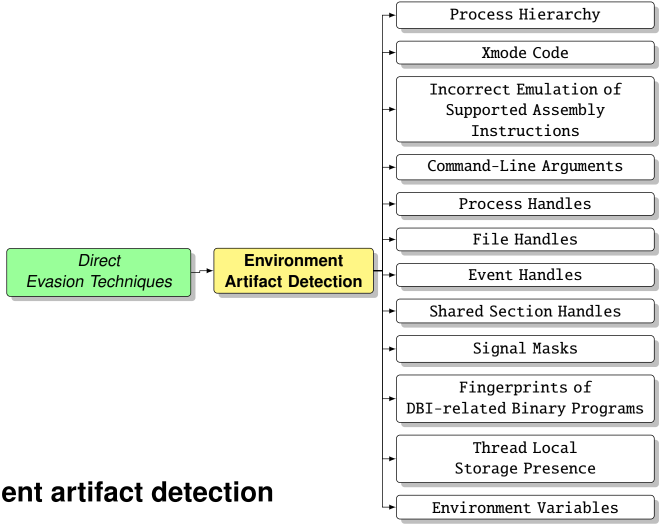


Code cache artifact detection

- Particular artifacts and behaviors that DBI frameworks use in code caches

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques – direct evasion techniques

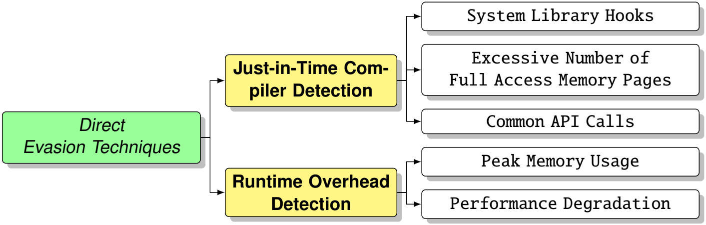


Environment artifact detection

- Environmental artifacts introduced by the DBI framework on the process of the client application

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques – direct evasion techniques

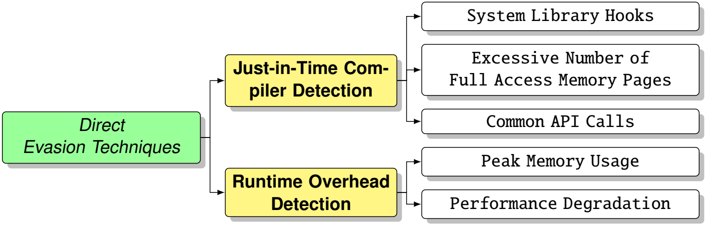


JIT compiler detection

- Constantly used by the DBI framework
- Lot of activity (for instance, when allocating code generated to a code cache)

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques – direct evasion techniques



JIT compiler detection

- Constantly used by the DBI framework
- Lot of activity (for instance, when allocating code generated to a code cache)

Runtime overhead detection

- Take measurements at runtime and then compare them with a baseline
  - Note that false positives may arise
- DBI parses the code: takes (a lot of) time!
- Not only on execution time, also in the amount of memory used

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques – direct evasion techniques

Evasive Techniques	**Articles and works** [1]	[2]	[3]	[4]	[5]	[6]	[7, 8]	[9]	(this work)	**PoC**	**Classification**†
Unsupported Assembly Instructions	●								●	✗	FL
Unsupported Behaviors	●	●							●	✗	FL
Stalling Code					●				●	✗	RL
Memory Exhaustion									●	✗	RL
Code Cache Fingerprints	●					●	●	●	●	✓	CCAD
Instruction Pointer in Unexpected Memory Regions			●	●	●	●	●	●	●	✓	CCAD
Incorrect Handling of Self-Modifying Code			●			●	●	●	●	✓	CCAD
Unexpected Context	●								●	✗	CCAD
Memory Region Permission Mismatches							●	●	●	✓	CCAD
Process Hierarchy		●		●	●	●		●	●	✓	EAD
Xmode Code	●							●	●	✗	EAD
Incorrect Emulation of Supported Assembly Instructions							●	●	●	✓	EAD
Command-Line Arguments				●	●				●	✗	EAD
Process Handles				●	●			●	●	✗	EAD
File Handles		●						●	●	✗	EAD
Event Handles				●	●			●	●	✗	EAD
Shared Section Handles				●	●			●	●	✗	EAD
Signal Masks		●							●	✗	EAD
Fingerprints of DBI-related Binary Programs		●		●	●			●	●	✗	EAD
Thread Local Storage Presence	●							●	●	✗	EAD
Environment Variables							●		●	✗	EAD
System Library Hooks				●	●	●		●	●	✓	JCD
Excessive Number of Full Access Memory Pages	●			●	●	●	●		●	✓	JCD
Common API Calls				●		●			●	✗	JCD
Peak Memory Usage		●							●	✗	ROD
Performance Degradation		●		●	●	●	●	●	●	✓	ROD

†FL: *Functional Limitation*; RL: *Resource Limitation*; CCAD: *Code Cache Artifact Detection*; EAD: *Environment Artifact Detection*; JCD: *JIT Compiler Detection*; ROD: *Runtime Overhead Detection*

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques – discussion of results

- **Small number of proof of concepts** (PoCs)
  - Only 9 PoCs are provided in the literature
  - 4 papers have made PoC tools available (`eXait`, `PwIN`, and two unnamed tools)

Universidad Zaragoza

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques – discussion of results

- **Small number of proof of concepts** (PoCs)
  - Only 9 PoCs are provided in the literature
  - 4 papers have made PoC tools available (`eXait`, `PwIN`, and two unnamed tools)
- **Transparency property of DBI tools**
  - All techniques look for artifacts in memory and in the system to detect their presence
  - We need **perfect transparency** to get unnoticed

Universidad Zaragoza

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques – discussion of results

- **Small number of proof of concepts** (PoCs)
  - Only 9 PoCs are provided in the literature
  - 4 papers have made PoC tools available (`eXait`, `PwIN`, and two unnamed tools)
- **Transparency property of DBI tools**
  - All techniques look for artifacts in memory and in the system to detect their presence
  - We need **perfect transparency** to get unnoticed
- **Isolation property of DBI tools**
  - All the highlighted techniques interact with resources strictly associated with DBI frameworks (such as code caches and TLS) as a form of detection
- **False positives can occur** when using these detection techniques

Universidad Zaragoza

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques – discussion of results

Conclusions

1. **Significant advances have been made to reduce the attack surface**
2. **DBI are suitable for certain types of security analysis** (taint analysis, symbolic execution, or cryptoanalysis, to name a few)**...**
   but they are unsuitable for others (e.g., sophisticated malware or advanced threats analysis)

Universidad Zaragoza

Anti-Instrumentation and Countermeasures Techniques

Anti-instrumentation techniques – discussion of results

Conclusions

1. **Significant advances have been made to reduce the attack surface**
2. **DBI are suitable for certain types of security analysis** (taint analysis, symbolic execution, or cryptoanalysis, to name a few)...
   but they are unsuitable for others (e.g., sophisticated malware or advanced threats analysis)

More efforts are needed
to achieve complete isolation and transparency

Universidad Zaragoza

Anti-Instrumentation and Countermeasures Techniques

Countermeasures techniques

Anti-instrumentation tools

- `PinVMShield` (Rodríguez et al., 2016)
  - GNU/GPL version 3 license
  - Source code available (`https://bitbucket.org/rjrodriguez/pinvmshield/`)
  - Pin + Windows
  - Extended in (A. Santos et al., 2020)
- `Arancino` (Polino et al., 2017)
  - Unspecified license
  - Source code available (`https://github.com/necst/arancino`)
  - Pin + Windows
- Unnamed library (D'Elia et al., 2019)
  - Unspecified license
  - Source code available (`https://github.com/season-lab/sok-dbi-security/`)
  - Pin + Windows
  - Extended in (D'Elia et al., 2020)

Universidad Zaragoza

Anti-Instrumentation and Countermeasures Techniques

Countermeasures techniques

Evasive Techniques	Articles and works					Classification†
	[5]	[6]	[9]	[10]	[11]	
Code Cache Fingerprinting		●	●	●	●	
Instruction Pointer in Unexpected Memory Regions		●	●	●		
Incorrect Handling of Self-Modifying Code		●				CCAD
Unexpected Context		●				
Memory Region Permission Mismatches			●	●	●	
Process Hierarchy		●		●		
Fingerprints of DBI-related Binary Programs	●					EAD
Thread Local Storage Presence					●	
System Library Hooks		●				
Excessive Number of Full Access Memory Pages		●	●			JCD
Common API Calls		●				
Performance Degradation	●			●		ROD

†CCAD: *Code Cache Artifact Detection*; EAD: *Environment Artifact Detection*; JCD: *JIT Compiler Detection*; ROD: *Runtime Overhead Detection*

Anti-Instrumentation and Countermeasures Techniques

Countermeasures techniques – discussion of results

- **Only 12 countermeasures are proposed** (out of 26)
- **Mitigation techniques mainly based on the monitoring of system calls**
  - Main disadvantage: large number of system calls to be monitored
  - Example: Windows API (Ex family + internal Nt calls)
- **Incomplete solutions**
  - Not all the evasion cases are considered by the current countermeasures
- **Large overhead**
  - The lower the level of instrumentation granularity, the greater the overhead
  - Relevant metric in determining whether a countermeasure is usable in the real-world
  - Not studied in all the works

Countermeasures techniques – discussion of results

Conclusions

1. **Some evasion techniques are not mitigated** (at the time of writing)
   - Indirect evasion techniques remain unmitigated
   - Some of the direct evasion techniques remain unmitigated too
     - Based on environment artifacts
     - Based on runtime overhead detection (in particular, *Peak Memory Usage*)
2. **Recall rootkit paradox**: *whenever code wants to run on a system, it must be visible to the system in some way*

Universidad Zaragoza

Anti-Instrumentation and Countermeasures Techniques

Countermeasures techniques – discussion of results

Conclusions

1. **Some evasion techniques are not mitigated** (at the time of writing)
   - Indirect evasion techniques remain unmitigated
   - Some of the direct evasion techniques remain unmitigated too
     - Based on environment artifacts
     - Based on runtime overhead detection (in particular, *Peak Memory Usage*)
2. **Recall rootkit paradox**: *whenever code wants to run on a system, it must be visible to the system in some way*
   - Therefore, all evasion techniques can be detected in some way
   - Although avoiding indirect evasion techniques can be difficult (e.g., mitigation of *Unsupported Assembly Instruction* or *Unsupported Behaviors*)
     - *Fine-grained instrumentation has a large impact on performance, making it impractical for real-world scenarios*
     - *What are the behaviors/assembly instructions not currently supported by DBI frameworks?*
   - **We need instruction-level instrumentation with semantic analysis**
3. **Source code of the tools available to the public**
   - Facilitates their study, use, and improvement

Universidad Zaragoza

Outline

Challenges and Open Issues

More efforts are needed

- **Better DBI frameworks: complete isolation and full transparency**
- **Requirements needed when analyzing an application**

Challenges and Open Issues

More efforts are needed

- **Better DBI frameworks: complete isolation and full transparency**
- **Requirements needed when analyzing an application**

Gaps found – *students interested in this topic, email me* ☺

- **Lack of countermeasures** (only 12 out of 26)
- **Lack of experimentation in real-world scenarios**
- **Lack of evaluation on the impact of countermeasures**
- **Lack of comparison between countermeasures**
- **Lack of proofs of concept**

Challenges and Open Issues

More efforts are needed

- **Better DBI frameworks: complete isolation and full transparency**
- **Requirements needed when analyzing an application**

Gaps found – *students interested in this topic, email me* ☺

- **Lack of countermeasures** (only 12 out of 26)
- **Lack of experimentation in real-world scenarios**
- **Lack of evaluation on the impact of countermeasures**
- **Lack of comparison between countermeasures**
- **Lack of proofs of concept**

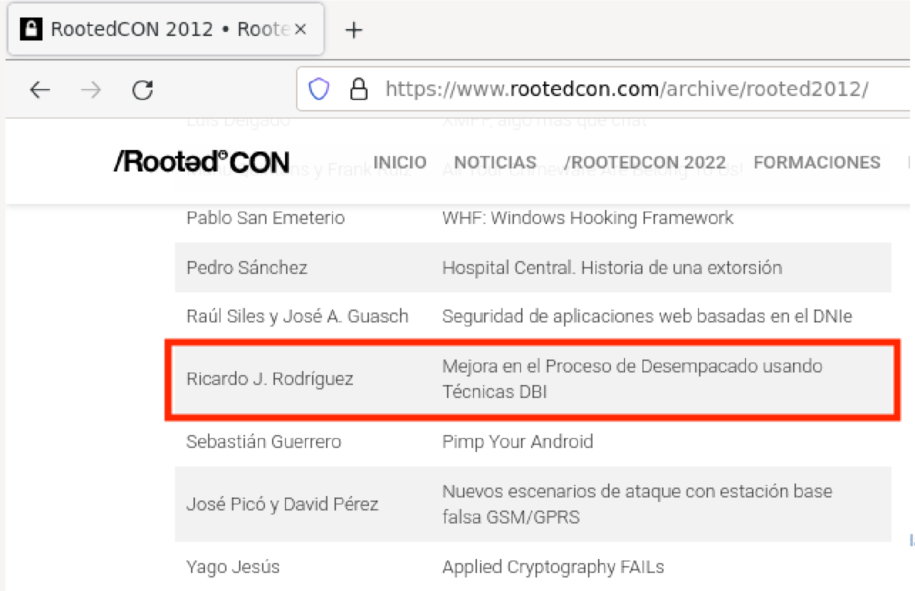
Keep working, folks!

(and please, **make your research and tools available to the public** ♥)

Universidad Zaragoza

Special Thanks

10 years ago... and presenting works in 7 editions of RootedCON...



Special Thanks

10 years ago... and presenting works in 7 editions of RootedCON...

Universidad Zaragoza

Outline

References

Balzarotti et al., 2010 Davide Balzarotti, Marco Cova, Christoph Karlberger, Engin Kirda, Christopher Kruegel, and Giovanni Vigna. 2010. Efficient detection of split personalities in malware. In Proceedings of the Network and Distributed System Security Symposium. The Internet Society, 16 pages.

Falcón & Riva, 2012 [4] Francisco Falcón and Nahuel Riva. 2012. Dynamic Binary Instrumentation Frameworks: I know you're there spying on me. Retrieved November 14, 2020 from `https://www.coresecurity.com/corelabs-research/open-source-tools/exait`.

Li & Li, 2014 [2] Xiaoning Li and Kang Li. 2014. Defeating the transparency features of dynamic binary instrumentation. In Proceedings of the BlackHat USA.

Hron & Jermář, 2014 [3] Martin Hron and Jakub Jermář. 2014. SafeMachine malware needs love, too. Retrieved November 14, 2020 from `https://www.virusbulletin.com/uploads/pdf/conference_slides/2014/sponsorAVAST-VB2014.pdf`.

Sun et al., 2016 [1] Ke Sun, Xiaoning Li, and Ya Ou. 2016. Break out of the truman show: Active detection and escape of dynamic binary instrumentation, 2016. In Proceedings of the Black Hat Asia.

Rodríguez et al., 2016 [5] Ricardo J. Rodríguez, Inaki Rodriguez Gaston, and Javier Alonso. 2016. Towards the detection of isolation-aware malware. IEEE Latin America Transactions 14, 2 (2016), 1024–1036.

Polino et al., 2017 [6] Mario Polino, Andrea Continella, Sebastiano Mariani, Stefano D'Alessio, Lorenzo Fontana, Fabio Gritti, and Stefano Zanero. 2017. Measuring and defeating anti-instrumentation-equipped malware. In Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment. Michalis Polychronakis and Michael Meier (Eds.). Springer International Publishing, Cham, 73–96.

Ekenstein & Norrestam, 2017 Gustaf Ekenstein and David Norrestam. 2017. Classifying Evasive Malware. Master's thesis. Lund University.

Universidad Zaragoza

References

Kirsch et al., 2018 [7] Julian Kirsch, Zhechko Zhechev, Bruno Bierbaumer, and Thomas Kittel. 2018. PwIN – pwning intel piN: Why DBI is unsuitable for security applications. In Proceedings of the 23rd European Symposium on Research in Computer Security. Javier Lopez, Jianying Zhou, and Miguel Soriano (Eds.), Lecture Notes in Computer Science. Springer International Publishing, Cham, 363–382.

Zhechev, 2018 [8] Zhechko Zhechev. 2018. Security Evaluation of Dynamic Binary Instrumentation Engines. Master's thesis. Department of Informatics, Technical Universtity of Munich.

D'Elia et al., 2019 [9] Daniele Cono D'Elia, Emilio Coppa, Simone Nicchi, Federico Palmaro, and Lorenzo Cavallaro. 2019. SoK: Using dynamic binary instrumentation for security (and how you may get caught red handed). In Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security. ACM, New York, NY, 15–27.

Ugarte-Pedrero et al., 2019 Xabier Ugarte-Pedrero, Mariano Graziano, and Davide Balzarotti. 2019. A close look at a daily dataset of malware samples. ACM Transactions on Privacy Security 22, 1 (Jan. 2019), Article 6, 30 pages

D'Elia et al., 2020 [10] Daniele Cono D'Elia, Emilio Coppa, Federico Palmaro, and Lorenzo Cavallaro. 2020. On the dissection of evasive malware. IEEE Transactions on Information Forensics and Security 15 (2020), 2750–2765.

Santos et al., 2020 [11] Ailton Santos Filho, Ricardo J. Rodríguez, and Eduardo L. Feitosa. 2020. Reducing the attack surface of dynamic binary instrumentation frameworks. In Proceedings of the Developments and Advances in Defense and Security, Vol. 152. Springer, 3–13.

Evasion and Countermeasures Techniques to Detect Dynamic Binary Instrumentation Frameworks

Ailton Santos Filho[†], **Ricardo J. Rodríguez**[‡], Eduardo L. Feitosa[†]

[†]Institute of Computing
Federal University of Amazonas, Brazil

[‡] Dept. of Computer Science and Systems Engineering
University of Zaragoza, Spain

March 10, 2022

RootedCon 2022
Madrid, Spain