Catch Me If You Can: A Decade of Evasive Malware Attack and Defense

Alexei Bulazel & Bülent Yener
River Loop Security
Rensselaer Polytechnic Institute (RPI)

SHMOOCON

https://git.io/vbgEz

RPISEC

Bio

- Researcher at River Loop Security
- 2015 RPI/RPISEC graduate
  - Work was began as part of my MS and continued until early 2017
  - Collaboration with advisor Dr. Bülent Yener
- Good to be back at ShmooCon!

"A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web" published at ROOTS (Reversing and Offensive-Oriented Trends Symposium) 2017 in Vienna, Austria

Twitter:@0xAlexei

https://git.io/vbgEz

Introduction

- Automated dynamic malware analysis is essential to keep up with modern malware (and potentially malicious software)
- Problem: malware can detect and evade analysis
- Solution: detect or mitigate anti-analysis

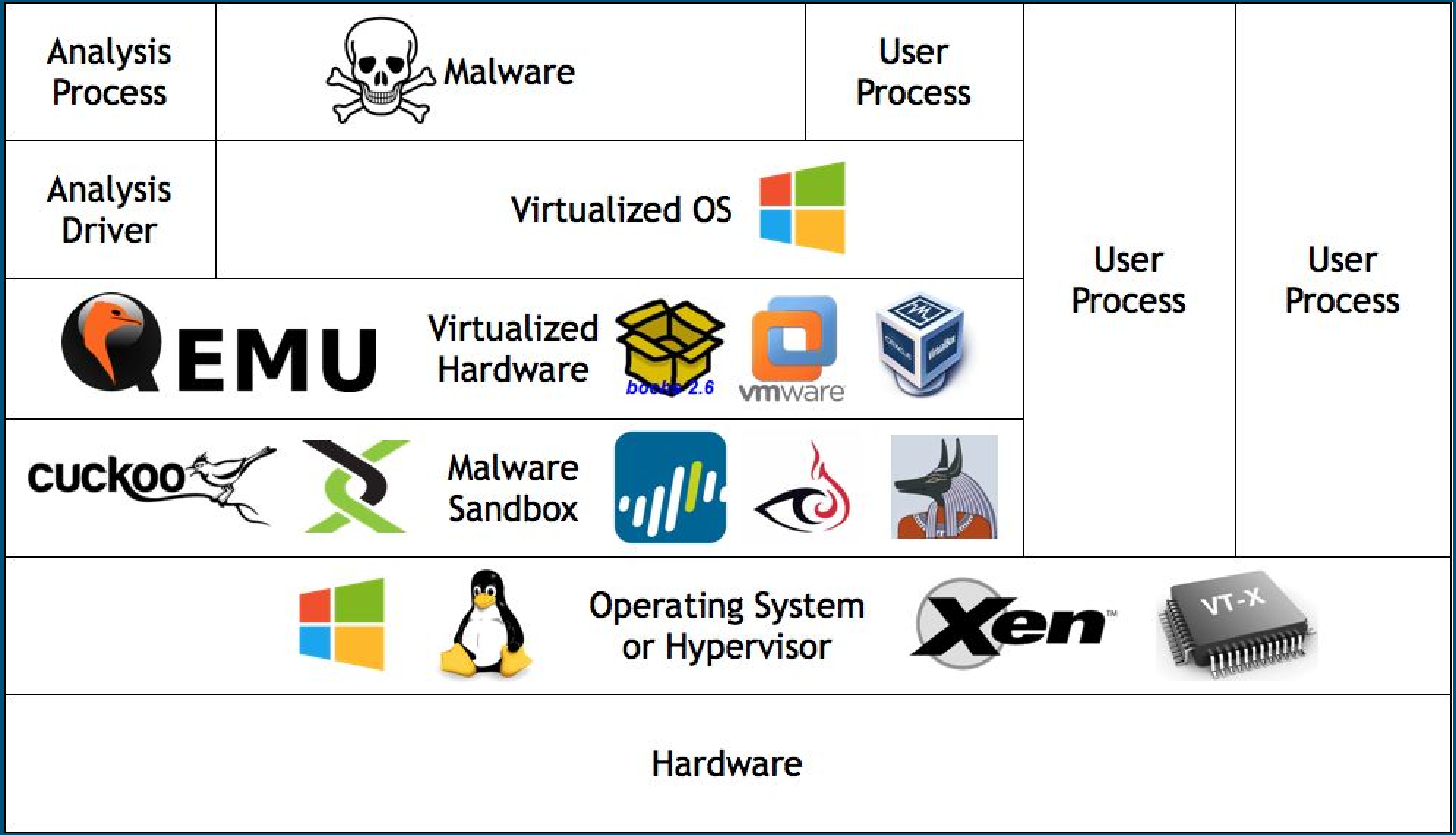
https://git.io/vbgEz

Dynamic Automated Analysis Systems

a.k.a:
"malware sandboxes"
"detonation chambers"

Not in scope:

- anti-debug
- anti-RE
- obfuscation



https://git.io/vbgEz

Motivation

- Evasive malware techniques is a perennial security conference topic
- Defensive techniques to counter evasion are generally relegated to academic publications or trade secrets
- There’s been over a decade of academic work in this field, no one had comprehensively looked at it
- Writing papers is fun

https://git.io/vbgEz

Scope

- Survey of ~200 works on evasive malware techniques, detection, mitigation, and case studies
- Mostly academic works, with a few industry talks and publications

- This presentation - focus on PC-based malware experimentation
- Timeline
  - First papers on evasive malware ~2005
  - Part of a longer lineage, see historical work on software protection

https://git.io/vbgEz

Takeaways

- Evasive malware and defenders continually evolve to counter one another

- The fight between malware and analysis systems is likely to continue long into the future

- There are immense challenges to experimental evaluation and the ability to establish ground truth

Read The Paper!

https://git.io/vbgEz

https://github.com/bulaza/Publications/blob/master/ROOTS2017

https://git.io/vbgEz

Presentation Outline

1. Introduction
2. Offense - Detecting Analysis Systems
3. Defense - Detecting Malware Evasion
4. Defense - Mitigating Malware Evasion
5. Discussion
6. Conclusion

SHMOOCON

Offense - Detecting Analysis Systems

- Fingerprint Classes
  - Environmental Artifacts
  - Timing
  - CPU Virtualization
  - Process Introspection
  - Reverse Turing Tests
  - Network Artifacts
  - Mobile Sensors
  - Browser Specific

```
bool beingAnalyzed = DetectAnalysis();

if(beingAnalyzed)
{
    BehaveBenignly();
}
else
{
    InfectSystem();
}
```

Environmental Artifacts & Timing

- Unique distinguishing characteristics of the analysis environment itself
  - Usernames
  - System settings
  - Date
  - Installed software
  - Files on disk
  - Running processes
  - Number of CPUs
  - Amount of RAM

- Timing discrepancies in analysis systems
- Sources:
  - Emulation / virtualization overhead
  - Analysis instrumentation overhead
  - Overhead of physical hardware instrumentation (potentially)
- Challenging to mitigate
  - Garfinkle et al: “extreme engineering hardship and huge runtime overhead”

CPU Virtualization & Process Introspection

- CPU "Red Pills"
- Discrepancies in CPU behavior introduced by virtualization
  - Erroneously accepted/rejected instructions
  - Incorrect exception behavior
  - Flag edge cases
  - MSRs
  - CPUID/SIDT/SGDT/etc discrepancy
- Particularly applicable for emulators

- Discrepancies in internal state
  - Memory or register contents
  - Function hooks
  - Injected libraries
  - Page permission eccentricities
- Commonly used in anti-DBI

Reverse Turing Tests & Network Artifacts

- *Computer* decides if *it* is interacting with computer or human
- Passive: mouse movement, typing cadence, process churn, scrolling
- Active: user must click a button
- Wear-and-Tear: evidence of human use, copy-paste clipboard, “recently opened” file lists, web history, phone camera photos

- Fixed IP address
- Network isolation
- Incorrectly emulated network devices or protocols
- Unusually fast internet service

Detection - Discussion

- Variety of sources: underlying technologies facilitating analysis, system configuration, analysis instrumentation

- Easy to use = easy to mitigate
- Difficult to use = difficult to mitigate

- Reverse Turing Tests seem to be growing in relevance, and are extremely difficult to mitigate against

https://git.io/vbgEz

Presentation Outline

1. Introduction
2. Offense - Detecting Analysis Systems
3. Defense - Detecting Malware Evasion
4. Defense - Mitigating Malware Evasion
5. Discussion
6. Conclusion

Detecting Malware Evasion

- Detecting that malware exhibits evasive behavior under dynamic analysis, but not mitigating evasion
  - Comparatively fewer works relative to mitigation work
- Early work - detecting known anti-analysis-techniques
  - 2008: Lau et al.'s DSD-Tracer
- Most works use multi-system execution
  - Run malware in multiple systems and compare behavior offline - discrepancies may indicate evasion in one or more of these systems

Multi-System Execution

- Instruction-level (2009: Kang et al.)
  - Too low level, prone to detect spurious differences

- System call-level (2010: Balzarotti et al. / 2015: Kirat & Vigna - MalGene)
  - Higher level than just instructions
  - MalGene uses algorithms taken from bioinformatics work in protein alignment

- Persistent changes to system state (2011: Lindorfer et al. - Disarm)
  - Jaccard distance-based comparisons

- Behavioral profiling (2014: Kirat et al. - BareCloud)
  - *What* malware did vs. *how* it did it, “hierarchical similarity” algorithms from computer vision and text similarity research

Evasion Detection - Discussion

- Evolution over time to increasingly complex algorithmic approaches, working over increasingly abstracted execution traces

- Multi-system execution is a common solution for evasion detection

- Offline algorithms do not detect evasion in real time

- Detection does not solve the main challenge of evasion, so there is less work in the field compared to mitigation research

https://git.io/vbgEz

Presentation Outline

1. Introduction
2. Offense - Detecting Analysis Systems
3. Defense - Detecting Malware Evasion
4. Defense - Mitigating Malware Evasion
5. Discussion
6. Conclusion

Defense - Mitigating Evasion

- Mitigating evasive behavior in malware so that analysis can proceed unhindered

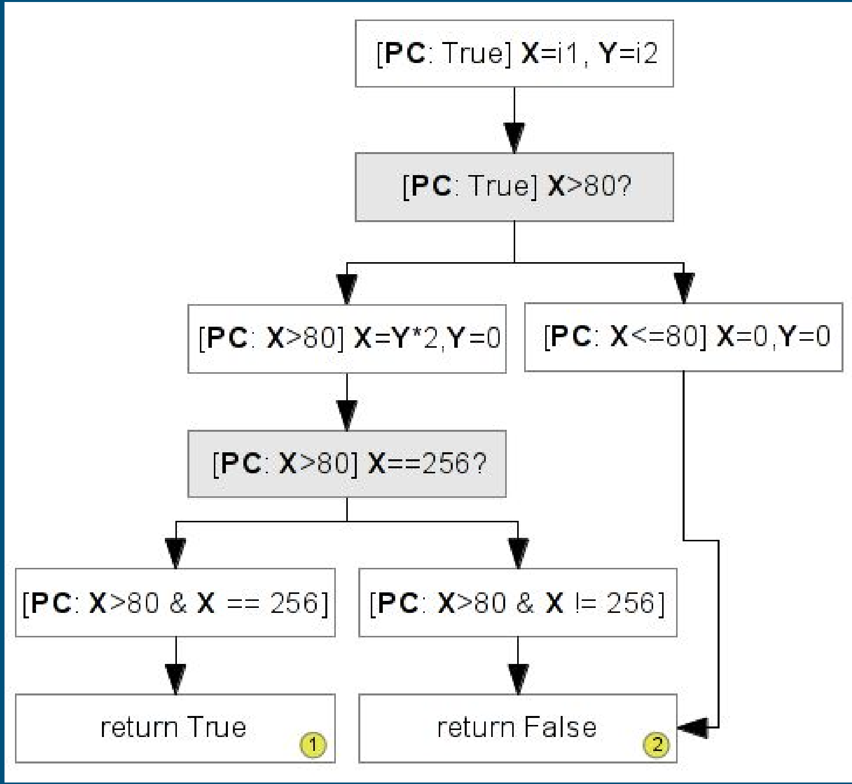
- Early approaches
  - Binary Modification
  - Hiding Environment Artifacts
  - State Modification
  - Multi-Platform Record and Replay
- Path Exploration
- Hypervisor-based Analysis
- Bare Metal Analysis & SMM-based Analysis
- Discussion

Early Approaches

- Binary Modification
  - 2006: Vasudevan et al. - Cobra
  - Emulate code in blocks like QEMU
    - Remove or rewrite malware instructions that could be used for detection
- Hiding Environmental Artifacts
  - 2007: Willems et al. - CWSandbox
    - In system kernel driver hides environmental artifacts
  - Oberheide later demonstrated several detection techniques against CWSandbox
- State Modification
  - 2009: Kang et al.
    - Builds upon detection work
    - “dynamic state modification” (DSM), modifications to state force malware execution down alternative paths
- Multi-Platform Record and Replay
  - 2012: Yan et al. - V2E
    - Kang et al.’s DSMs are not scalable for multiple anti-analysis checks
    - Don’t mitigate individual occurrences of evasion, make evasion irrelevant because systems are inherently transparent

Path Exploration

- 2007: Moser et al.
  - Looks broadly at code coverage and analyzing trigger-based malware
  - Track when input is used to make control flow decisions, change it to force execution down different paths
- 2008: Brumley et al. - MineSweeper
  - Trigger-based malware focused
  - Represents inputs to potential triggers symbolically, while other code is executed concretely



Hypervisor-based Analysis

- 2008: Dinaburg et al. - Ether
  - Catch system calls and context switches from Xen
  - Despite extensive efforts to make analysis transparent, Pék et al. created nEther and were able to detect Ether
- 2009: Nguyen et al. - MAVMM
  - AMD SVM with custom hypervisor
  - Thompson et al. subsequently demonstrated timing attacks that can be used to detect MAVMM and other hypervisor based systems
- 2014: Lengyel et al. - DRAKVUF
  - Xen-based, instruments code with injected breakpoints

Bare Metal Analysis

- 2011, 2014: Kirat et al. - BareBox & BareCloud
  - BareBox - in-system kernel driver
  - BareCloud - post-run disk forensics
- 2012: Willems et al.
  - Hardware-based branch tracing features
  - Analyzed evasive PDFs
- 2016: Spensky et al. - LO-PHI
  - Instrument physical hardware
  - Capture RAM and disk activity at the hardware level
  - Scriptable user keyboard/mouse interaction with USB-connected Arduinos
- SMM-based analysis: all the transparency benefits of bare metal, while restoring introspection
  - Full access to system memory, protection from modification, high speed, protection from introspection
- 2013 & 2015: Zhang et al. - Spectre, MalT
  - Spectre: SMM-based analysis, 100x faster than VMM based introspection
  - MalT - SMM-based *debugging*
- 2016: Leach et al. - Hops
  - SMM memory snapshotting and PCI-based instrumentation

Mitigation - Discussion

- Two broad categories: active and passive mitigation
  - Active: detect-then-mitigate
  - Passive: build inherent transparency
- Passive approaches have been more prevalent
  - Hypervisors, bare metal, etc
- Bare metal is the cutting edge in academic research, but it may not be scalable to industry applications
  - Promising, but not a panacea against any class of attacks other than CPU-based

https://git.io/vbgEz

Presentation Outline

1. Introduction
2. Offense - Detecting Analysis Systems
3. Defense - Detecting Malware Evasion
4. Defense - Mitigating Malware Evasion
5. Discussion
6. Conclusion

Discussion

- Offensive Research
  - Reverse Turing Tests
  - Detecting Bare Metal Analysis
- Defensive Research
  - Improving Bare Metal Analysis
  - Heuristic Evasion Detection
  - Passing Reverse Turing Tests
- Research Evaluation
  - Establishing Ground Truth
  - Challenges in research evaluation

Offensive Research

- Reverse Turing Tests
  - Difficult to mitigate against
  - Increasingly relevant as analysis systems become transparent
  - Look to anti-cheating research for online gaming
- Detecting bare metal analysis
  - Still vulnerable to everything except CPU-based attacks
  - Look to detecting analysis instrumentation

Defense - Improving Bare Metal Analysis

- Improving bare metal analysis - efficient, introspection, and stalling mitigation
    - Efficiency
        - 2016: Vadrevu and Perdisci - MAXS - improve efficiency by 50% on average with less than 0.3% information loss in analysis
    - Introspection
        - SMM needs further research
    - Stalling mitigation
        - Difficult to mitigate against with current bare metal systems
        - Performance tracing technologies may provide a direction forward

Defense - Heuristic Evasion Detection

- Can the behaviors involved in evasion before conditional branching occurs be detected heuristically?

- Inspirations
  - Code *fragility* may indicate maliciousness
  - Heuristic detection in enterprise and personal AV/endpoint products
  - Stalling detection techniques
  - Anti-anti-DBI heuristics

Defense - Passing Reverse Turing Tests

- Believably simulating human presence as reverse Turing Tests become more prevalent

- Inspirations:
  - UNVEIL’s fake file system creation
  - LARIAT information assurance testbed
  - Biometric spoofing research

Meta - Establishing Ground Truth

- *Unknown-unknowns*: researchers don't know what they don't know
- Human malware analysis is not scalable
- "Bootstrapping" corpora - use previously generated analysis reports as ground truth
  - Problematic: differences in execution environment and time may lead to spurious differences
- Collection in the wild
  - Challenging for *evasive* malware
  - Collection sources may reveal biases

Meta - Challenges in Research Evaluation

- Evaluated works range from evaluating one lab-created malware sample to analyzing millions captured in the wild
- Impossible to empirically compare research, or reproduce results

- 2012: Rossow et al. - evaluated the “methodological rigor and prudence” of 36 papers involving malware experimentation from 2006-2011
    - We re-emphasize all of the author’s points and recommend researchers read their paper closely

https://git.io/vbgEz

Conclusion

- Surveyed in paper and backup slides: mobile and web analysis, case studies

- Continual evolution of offense and defense, will to continue into the future
- Cutting edge defenses may not be scalable
- Immense challenges to experimental evaluation and ground truth

ROOTS'17: *"A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion: PC, Mobile, and Web"*

- Friends who helped us edit: Rolf Rolles, James Kukucka, Aaron Sedlacek
- RPI support: Jeremy Blackthorne and Dr. Greg Hughes
- Program committee and our anonymous reviewers - particularly #4
- Dr. Sergey Bratus and ROOTS PC

SHMOOCON

@0xAlexei

yener@cs.rpi.edu

Backup Slides

Mobile Analysis - Fingerprinting

- Vulnerable to all of the previously mentioned classes of fingerprinting
- Almost all papers use Android
- Mobile specific:
    - Mobile wear-and-tear artifacts: photos on camera roll, text messages, installed apps, battery life and charge cycles
    - Mobile sensors: multiple radios (cellular, Wi- Fi, Bluetooth, NFC), cameras, microphones, GPS, accelerometers, gyroscopes, thermometers, barometers, proximity sensors, light sensors, magnetic sensors, humidity sensors, air pressure sensors, geomagnetic field sensors, voltmeters
    - Sensor interaction: device vibration can be measured with accelerometer, audio output can be picked up by microphones, CPU strain can increase temperature and strain battery

Mobile Analysis - Defense

- App model means (commodity) mobile malware is fundamentally different than PC malware - it doesn't just "run"

- We could not find any papers on just detecting mobile evasion

- Bare metal analysis:
  - BareDroid - fast system restoration by reverting system drive partitions, monitoring with SELinux
  - A5 - network signature collection with bare metal and virtualized systems
  - Challenges: charging, resetting, introspection

Mobile Analysis - Offensive Research

- Seminal work: Oberheide & Miller’s talk on active reconnaissance attacks against Android Bouncer

- Lots of papers on Android analysis detection - almost all just continuing Oberheide & Miller-style attacks

- Advanced fingerprinting research: biometric-based human interaction detection (e.g., variation in touchscreen input) - PIDetector

Web Analysis Systems

- More generally used to look at browser-based exploits, not so much in-browser malicious code
- Two general categories:
  - Instrumented real browsers
    - Look for OS actions indicative of exploitation, e.g., new file or process creation
  - Browser emulators
    - More brittle - emulate the essential parts of a browser - HTTP, HTML, JS, etc with an emulator, look for patterns indicative of exploitation, e.g., a heap spray, or invocation of a function with known bad arguments
- Approaches: submission-based, web crawling, browser plugin analysis

Web Analysis - Fingerprinting

- Use of alternative scripting languages - e.g., ActiveX or VBScript
- Emulators: JS runtime specific bugs, HTML parsing differentials, DOM incompleteness
- HTML5 canvas fingerprinting against *specific* systems
- Shellcode-time detection - detect analysis during shellcode execution
- Referrer checks - only serve content if coming from a specific referrer

Web Analysis - Defense

- Detecting evasion: Kapravelos et al., Revolver
  - Collect web pages *identified as* malicious and benign by existing web analysis systems
  - Compare script ASTs and compare pairwise - if a “benign” script has the same AST as a “malicious” one with the addition of additional control flow nodes, the added nodes are assumed to be evasion checks
- Evasion mitigation
  - Cova et al. - JSAND: force the invocation of JS functions in web malware if not called during regular execution
  - Kolbitsch et al. - Rozzle: multi-path JS forced execution
  - Kim et al. - J-Force: forced path exploration by recording branch outcomes and mutating them. Dynamic creation of DOM elements as needed to prevent crashes on forced paths.

Quantifying Evasion In The Wild

- Extremely difficult to do - lots of “unknown-unknowns” about evasion
    - Likely better done by commercial AV companies seeing millions of samples a day then by academics
- Chen et al. - 2.7% of samples exhibited VMM evasion 2006-2007
- Bayer et al. - 0.03% of samples had a specific anti-Anubis check in 2009
- Chen et al. - anti-analysis became more prevalent 2009-2014, but less prevalent for APTs
- Kapravelos et al. - ~26% evasion rate in web samples

Suggestions for Experimentation

- Establish ground truth
  - Verify analysis results for at least a portion of the malware with a human analyst
- Make multi-execution system similar
  - Minimize differences in environment causing spurious differences in execution
  - Discuss any unavoidable differences
- Be explicit about malware origins
  - Malware corpora may have inherent skews
    - VirusTotal - wild samples caught by defenders, or offensive actors doing testing
    - APTs - hard to catch

Meta - Game Theory Formalizations

- Cat-and-mouse game of analysis system vs. malware
  - Strategy dependent on the “worthiness” of the adversary
  - Save advanced techniques for the most advanced opponent
- Stackelberg games
  - Allocation of analysis resources by analysis system with randomized strategy while malware deploys a purely deterministic evasion strategy