



# Virtualization Based Security

Using Endpoint CPU Virtualization to  
Transform Enterprise Security

Mike Burwick, Principal SE

[mike.burwick@bromium.com](mailto:mike.burwick@bromium.com)



**Detection:  
Shark or Dolphin?**





# The **key principles** of attacks remain true



There will always be **software vulnerabilities**

**Malicious code and threats** will always exist

You will get owned  
**You cannot anticipate the next move**

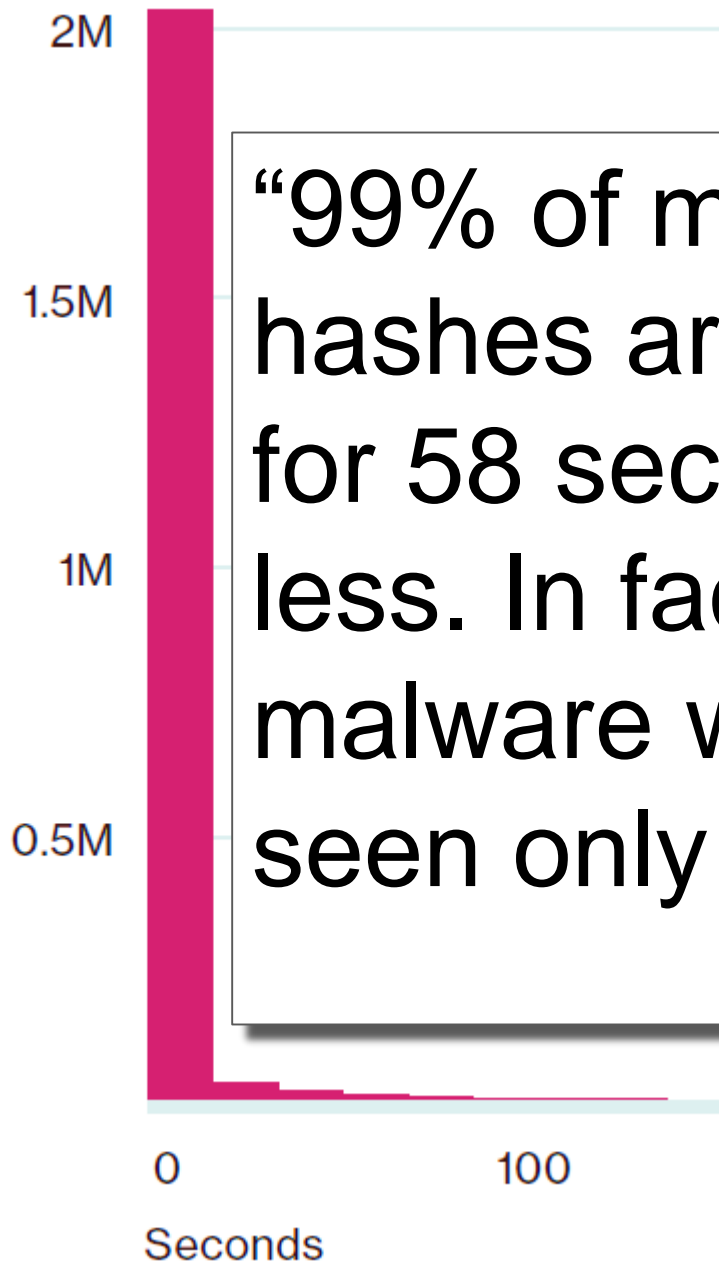
97% of malware is unique to a specific endpoint

March 2016

“99% of malware hashes are seen for 58 seconds or less. In fact, most malware was seen only once.”

Verizon DBIR '16

Count of hashes



Collecting InfoSec with News

NEWS HACKER NEWS SECURITY ARTICLES STUDY HOW TO EXPERT PANEL

HOME » MALWARE BECOMING OVERWHELMINGLY POLYMORPHIC

## Malware becoming Overwhelmingly Polymorphic

Travis Smith On March 4, 2016 — Leave A Comment

Malware and potentially unwanted applications (PUAs) have become overwhelmingly polymorphic, with 97 percent of malware morphing to become unique to a specific endpoint device, according to a report from security specialist Webroot. Travis Smith, senior security researcher at Tripwire have the following comments on it.



# The Failure of Detection

## It is mathematically impossible\* to detect all polymorphic or zero day malware in advance

**On the Infeasibility of Modeling Polymorphic Shellcode.**

Yingbo Song      Michael E. Locasto      Angelos Stavrou  
 Dept. of Computer Science    Dept. of Computer Science    Dept. of Computer Science  
 Columbia University          Columbia University          Columbia University  
 yingbo@cs.columbia.edu    locasto@cs.columbia.edu    angel@cs.columbia.edu

Angelos D. Keromytis      Salvatore J. Stolfo  
 Dept. of Computer Science    Dept. of Computer Science  
 Columbia University          Columbia University  
 angelos@cs.columbia.edu    sal@cs.columbia.edu

**ABSTRACT**  
 Polymorphic malware remains a troubling threat. The ability for malware to automatically transform into semantically equivalent variants frustrates attempts to rapidly construct a single, simple, easily verifiable representation. We present a quantitative analysis of the strengths and limitations of the classic polymorphism and consider its impact on current intrusion detection practices. We focus on the nature of shellcode decoding routines. The empirical evidence we gather helps show that modeling the class of self-modifying code is likely intractable by known methods, including both statistical constructs and string signatures. In addition, we develop and present measures that provide insight into the capabilities, strengths, and weaknesses of polymorphic engines. In order to explore countermeasures to future polymorphic threats, we show how to improve polymorphic techniques and create a proof-of-concept engine expressing these improvements. Our results indicate that the class of polymorphic behavior is too greatly spread and varied to model effectively. Our analysis also supplies a novel way to understand the limitations of current signature-based techniques. We conclude that modeling normal content is ultimately a more promising defense mechanism than modeling malicious or abnormal content.

**Categories and Subject Descriptors**  
 H.1.1 [Models and Principles]: Systems and Information Theory—Value of Information

**General Terms**  
 Experimentation, Measurement, Security

**Keywords**  
 polymorphism, shellcode, signature generation, statistical models

**1. INTRODUCTION**  
 Code injection attacks have traditionally received a great deal of attention from both security researchers and the blackhat community [1, 14], and researchers have proposed a variety of defenses, from artificial density of the address space [5] or instruction set [20, 4] to compiler added integrity checking of the stack [10, 15] or heap variables [34] and “solar” versions of library functions [3]. Other systems explore the use of tainted dataflow analysis to prevent the use of untrusted network or file input [9, 29] as part of the instruction stream. Finally, a large number of schemes propose capturing a representation of the exploit to create a signature for use in detecting and filtering future versions of the attack. Signature generation methods are based on a number of content modeling techniques, including simple string-based signature matching schemes like those used in Snort [36]. Many signature generation schemes focus on relatively simple detection heuristics, such as traffic characteristics [35, 22] (e.g., frequency of various packet types) or identification of the NOP sled [38], while others derive a signature from the actual exploit code [24, 43, 25] or statistical means of packet content [41, 40, 28], including content captured by honey pots [44]. This paper presents a study of the efficacy of contemporary polymorphism techniques, as well as methods to combine and improve them. Our analysis focuses on what we consider the most constrained section of malware, the decoder portion. Since this section of a malware sample or exploit instance must contain executable code, it cannot easily be disguised (unlike most other parts of a malware sample, except, perhaps, the higher order bits of the return address section). We derive our motivation from the challenge of modeling this particular type of malware data. We wondered whether, given unlimited samples of polymorphic code, it is possible to compare and create a set of signatures or a statistical model that could represent this class of code. If so, how costly would such a task be in terms of memory and processing time? In the span of the n-byte space: that those samples of code population, how much overlap is there with the class of benign network traffic? Unlike current research on poly-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.  
 CCS'07, October 29–November 2, 2007, Alexandria, Virginia, USA.  
 Copyright 2007 ACM 978-1-59593-705-2/07/0001...\$5.00.

23rd Annual Computer Security Applications Conference

**Limits of Static Analysis for Malware Detection**

Andreas Moser, Christopher Kruegel, and Engin Kirda  
 Secure Systems Lab  
 Technical University Vienna  
 {andy, chris, ek}@seclab.tuwien.ac.at

**Abstract**  
 Malicious code is an increasingly important problem that threatens the security of computer systems. The traditional line of defense against malware is composed of malware detectors such as virus and spyware scanners. Unfortunately, both researchers and malware authors have demonstrated that these scanners, which use pattern matching to identify malware, can be easily evaded by simple code transformations. To address this shortcoming, more powerful malware detectors have been proposed. These tools rely on semantic signatures and employ static analysis techniques such as model checking and theorem proving to perform detection. While it has been shown that these systems are highly effective in identifying current malware, it is less clear how successful they would be against adversaries that take into account the novel detection mechanisms. The goal of this paper is to explore the limits of static analysis for the detection of malicious code. To this end, we present a binary obfuscation scheme that relies on the idea of opaque constants, which are primitives that allow us to load a constant into a register such that an analysis tool cannot determine its value. Based on opaque constants, we build obfuscation transformations that obscure program control flow, disguise access to local and global variables, and interrupt tracking of values held in processor registers. Using our proposed obfuscation approach, we were able to show that advanced semantics-based malware detectors can be evaded. Moreover, our opaque constant primitive can be applied in a way such that it is provably hard to analyze for any static code analysis. This demonstrates that static analysis techniques alone might no longer be sufficient to identify malware.

**1 Introduction**  
 Malicious code (or malware) is defined as software that fulfills the harmful intent of an attacker. The damage caused by malware has dramatically increased in the past few years [8]. One reason is the rising popularity of the Internet and the resulting increase in the number of available vulnerable machines because of security-unaware users. Another reason is the elevated sophistication of the malicious code itself. Current systems to detect malicious code (most prominently, virus scanners) are largely based on syntactic signatures. That is, these systems are equipped with a database of regular expressions that specify byte or instruction sequences that are considered malicious. A program is declared malware when one of the signatures is identified in the program's code. Recent work [2] has demonstrated that techniques such as polymorphism and metamorphism are successful in evading commercial virus scanners. The reason is that syntactic signatures are ignorant of the semantics of instructions. To address this problem, a novel class of semantics-aware malware detectors was proposed. These detectors [3, 10, 11] operate with abstract models, or templates, that describe the behavior of malicious code. Because the syntactic properties of code are (largely) ignored, these techniques are (mostly) resilient against the evasion attempts discussed above. The premise of semantics-aware malware detectors is that semantic properties are more difficult to morph in an automated fashion than syntactic properties. While this is most likely true, the extent to which this is more difficult is less obvious. On one hand, semantics-aware detection faces the challenge that the problem of deciding whether a certain piece of code exhibits a certain behavior is undecidable in the general case. On the other hand, it is also not trivial for an attacker to automatically generate semantically equivalent code. The question that we address in this paper is the following: How difficult is it for an attacker to evade semantics-based malware detectors that use powerful static analysis to identify malicious code? We try to answer this question by introducing a binary code obfuscation technique that makes it difficult for an advanced, semantics-based malware detector to properly determine the effect of a piece of code. For this obfuscation process, we use a primitive known as

1063-9527/07 \$25.00 © 2007 IEEE  
 DOI 10.1109/ACSAC.2007.21      421     

\*[Limits of Static Analysis for Malware Detection](#)  
 Andreas Moser, Christopher Kruegel, and Engin Kirda  
 Secure Systems Lab Technical University Vienna

\*[On the Infeasibility of Modeling Polymorphic Shellcode.pdf](#)  
 Yingbo Song, Michael E. Locasto, Angelos Stavrou  
 Dept. of Computer Science Columbia University

# Br Detection vs Isolation

- Existing security solutions today rely on detection
  - A threat must be detected before it can be blocked
  - Polymorphic & “0 day” malware evades AV, white lists, IPS etc.



- Micro-virtualization protects through “CPU enforced Isolation”
  - Isolates untrusted content from the OS, files, network, etc.
  - Leverages hardware virtualization for maximum protection



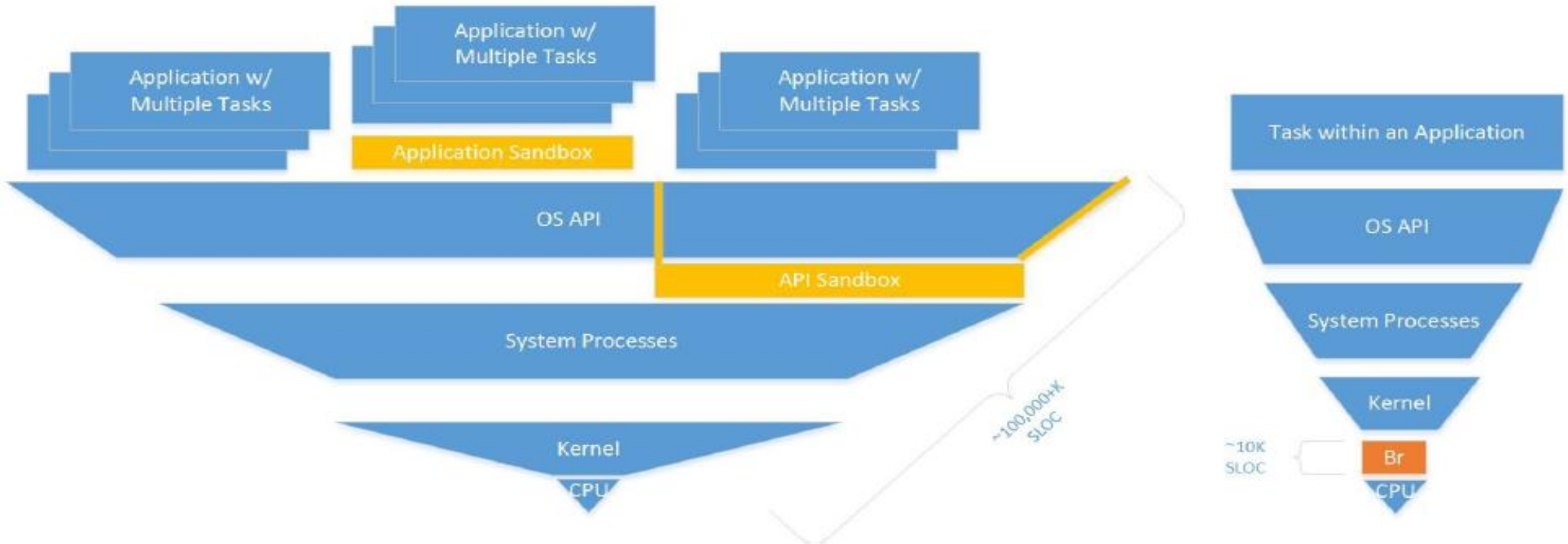
# Br Software Isolation Techniques





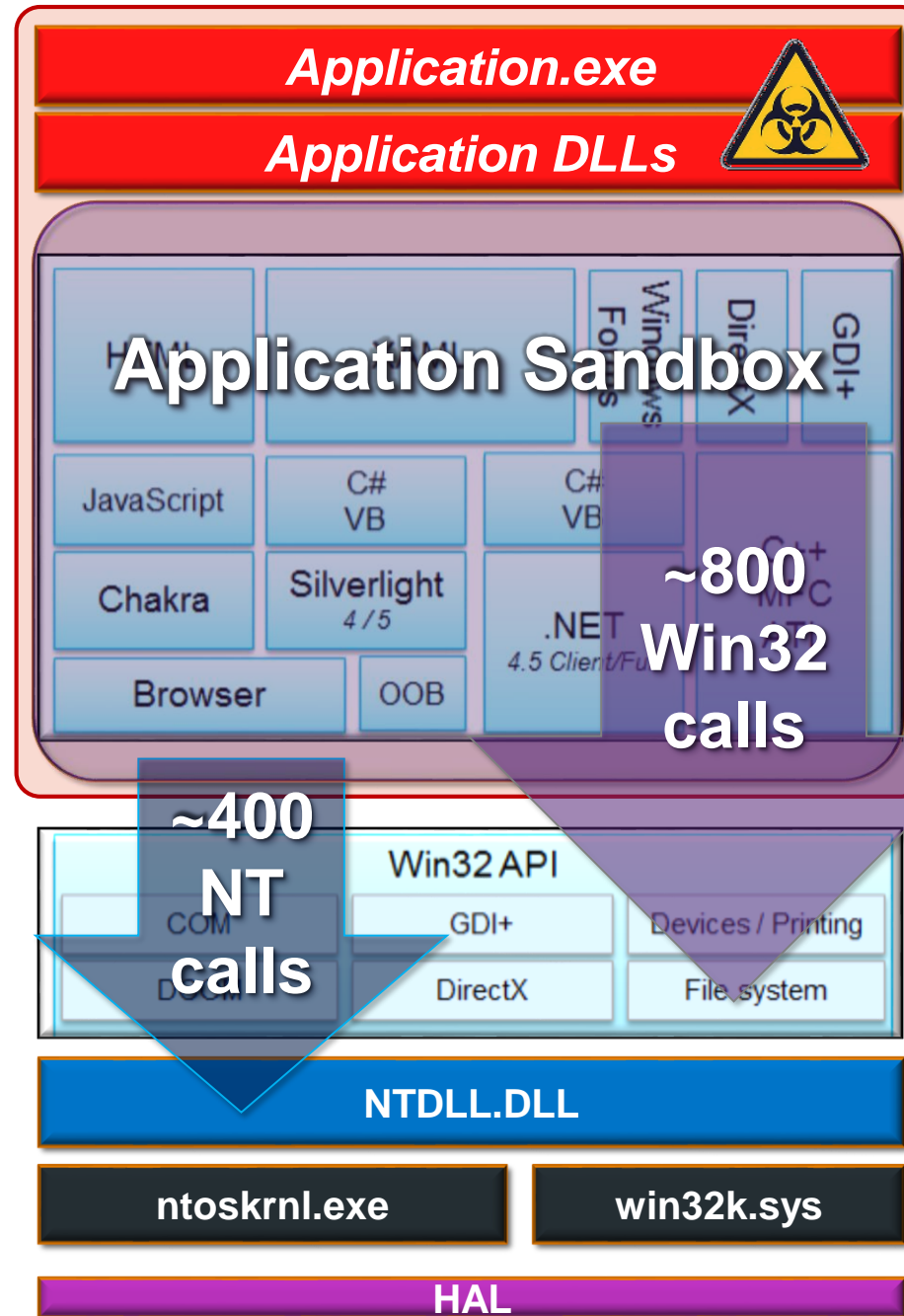
# Software vs Hardware Isolation

## Attack Surface Comparison between Sandboxing & Micro-virtualization



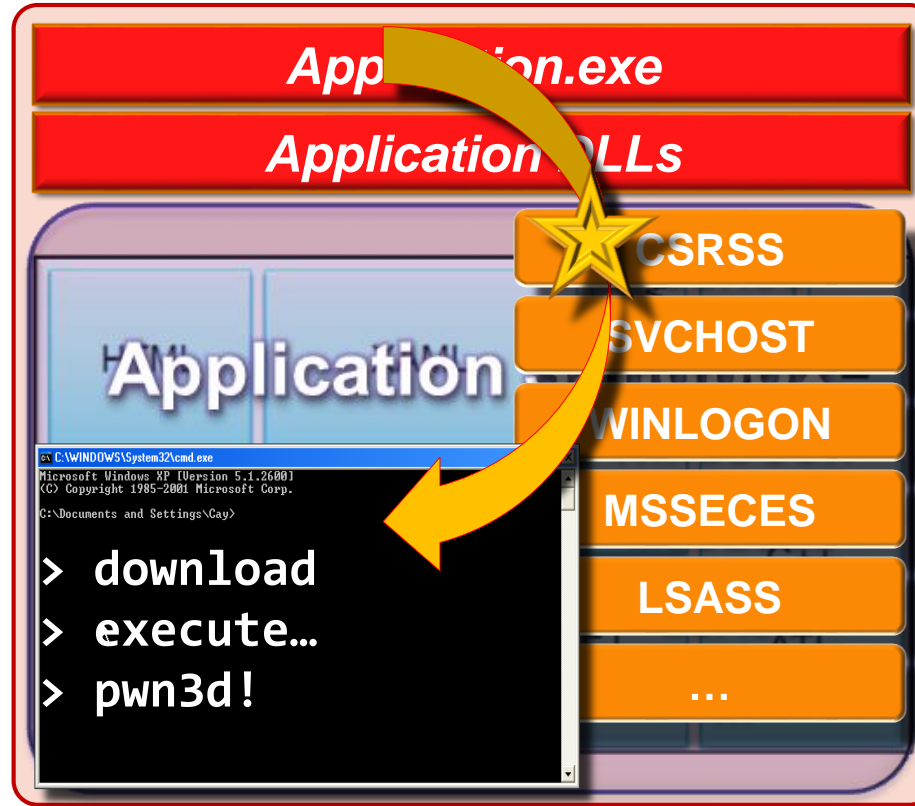


**Goal: Protect the OS kernel by isolating any malicious user task (application)**

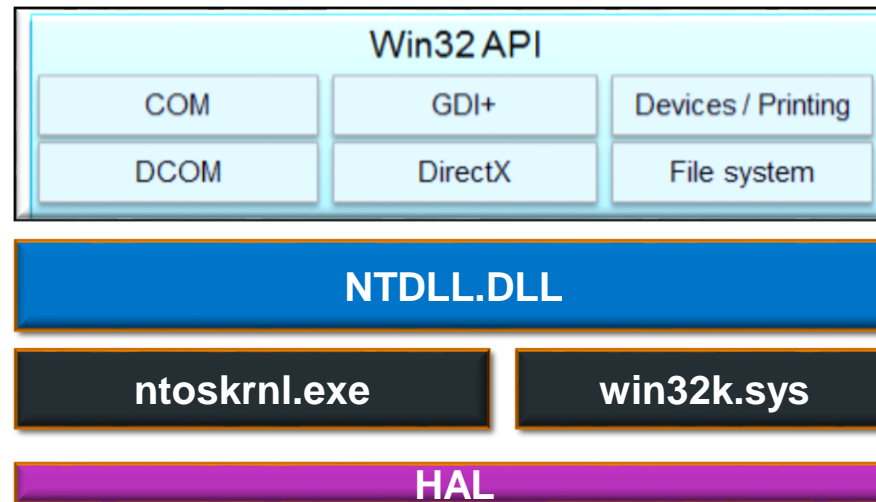


**Task**

**Goal: Protect the OS kernel by isolating any malicious user task (application)**



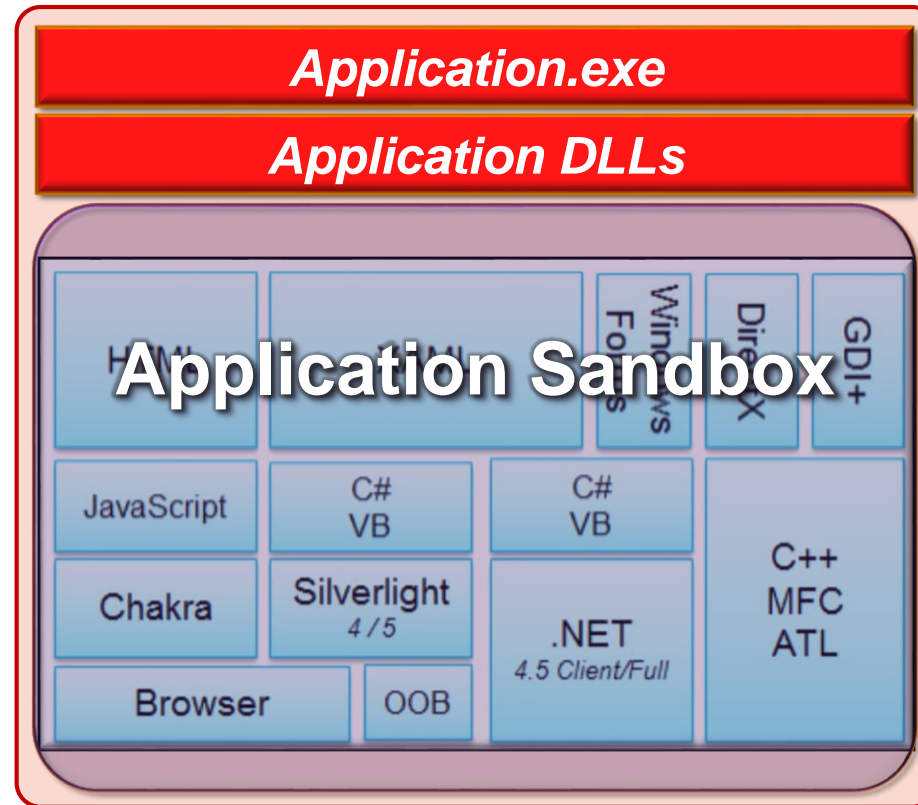
Millions of LOC  
Task



**CVE-2015-5119  
(Hacking Team)**

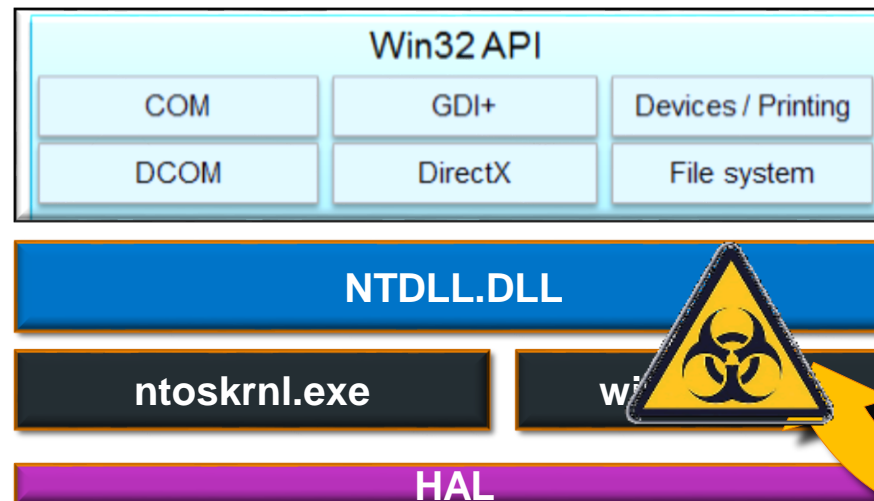


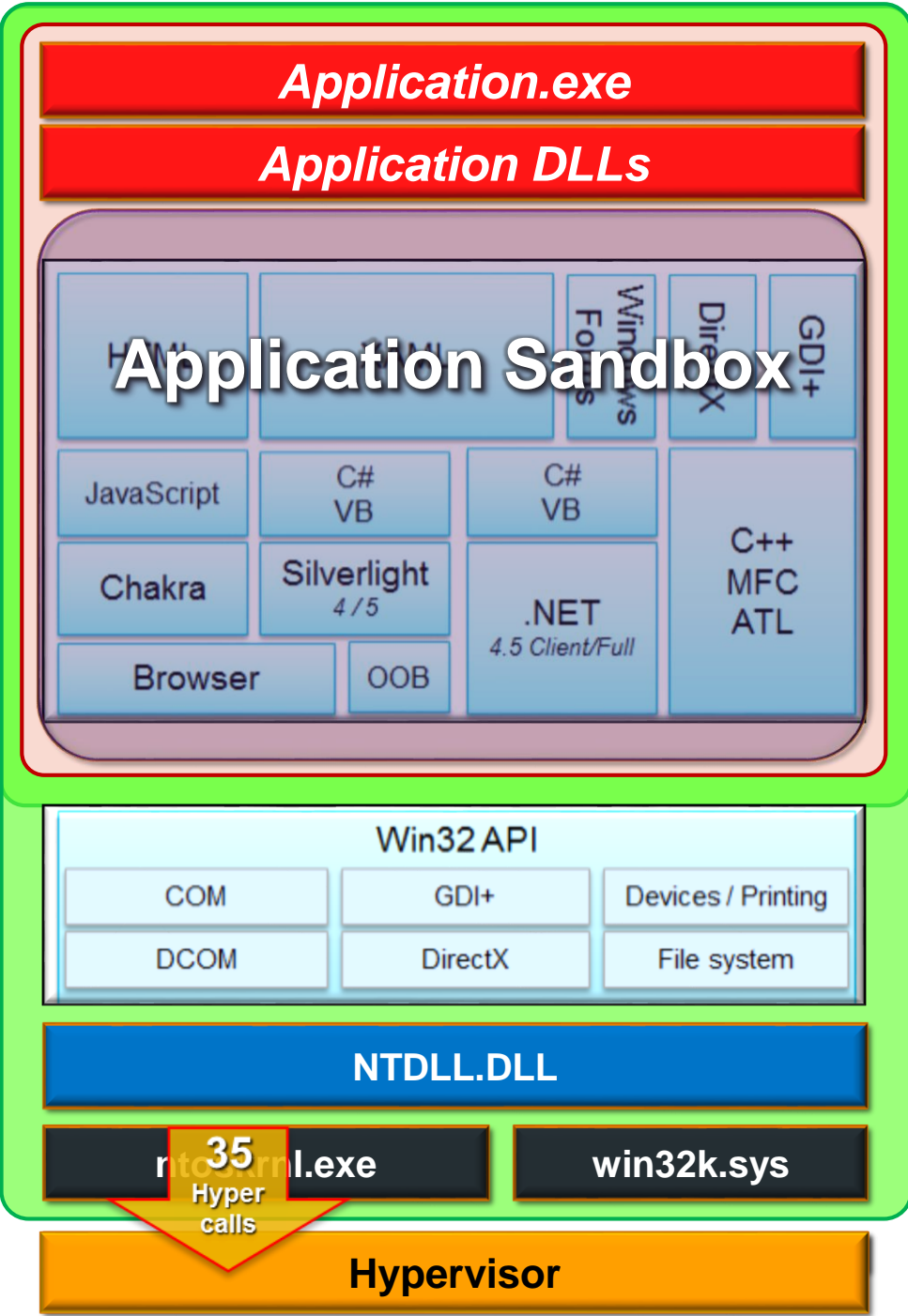
**Goal: Protect the OS kernel by isolating any malicious user task (application)**



**CVE-2015-2426**

**The attacker directly compromises the kernel, bypassing the sandbox**



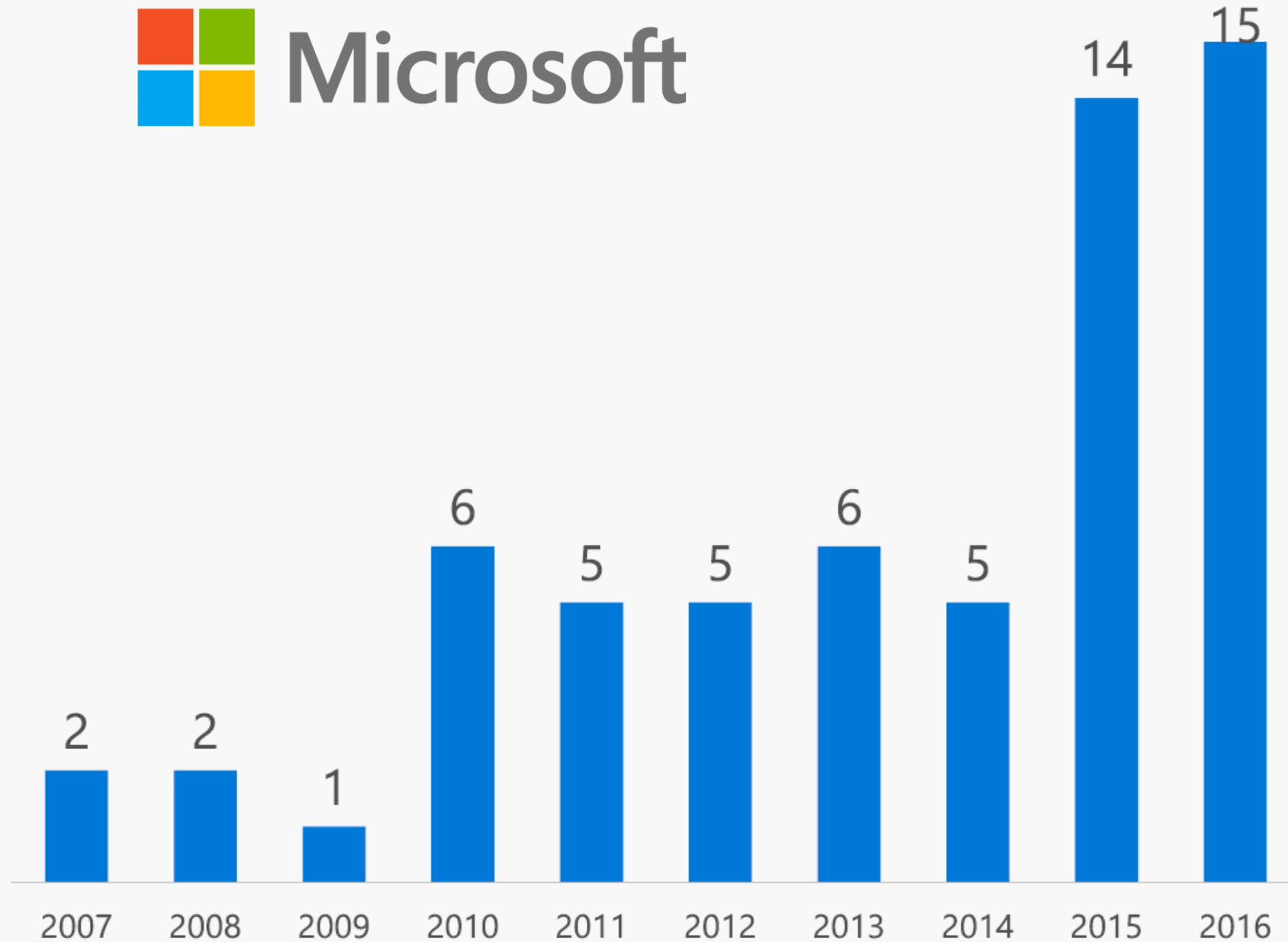


VM

# Current threat landscape

Driving the need for hardware based isolation

Our research indicates that there has been a dramatic increase in kernel exploits over the past two years



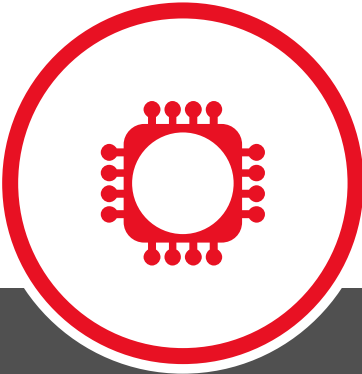


# Windows 10 Device Guard



Device integrity

UEFI Secure Boot prevents device tampering and ensures OS starts with integrity



Cryptographic processor

TPM processor provides tamper proof integrity validation and prevents unauthorized access to sensitive information



Virtualization

Processor based virtualization isolates critical system components and data and protects even in the event full system compromise



Biometric sensors

Using a biometric for authentication increases the level of difficulty for an attacker to the highest level

On Windows 10

50% of the time  
is spent in the  
browser





## Device Guard uses micro-virtualization to hardware-isolate two Windows Services

- ❖ Credential Guard isolates credential hashes to reduce pass-the-hash attacks
- ❖ Hypervisor enforced Code Integrity enforces white-listing for applications and the OS kernel

Windows Defender Application Guard will isolate the Edge browser (in RS2)



CG  
Credential  
Guard



HVCI  
Code  
Integrity



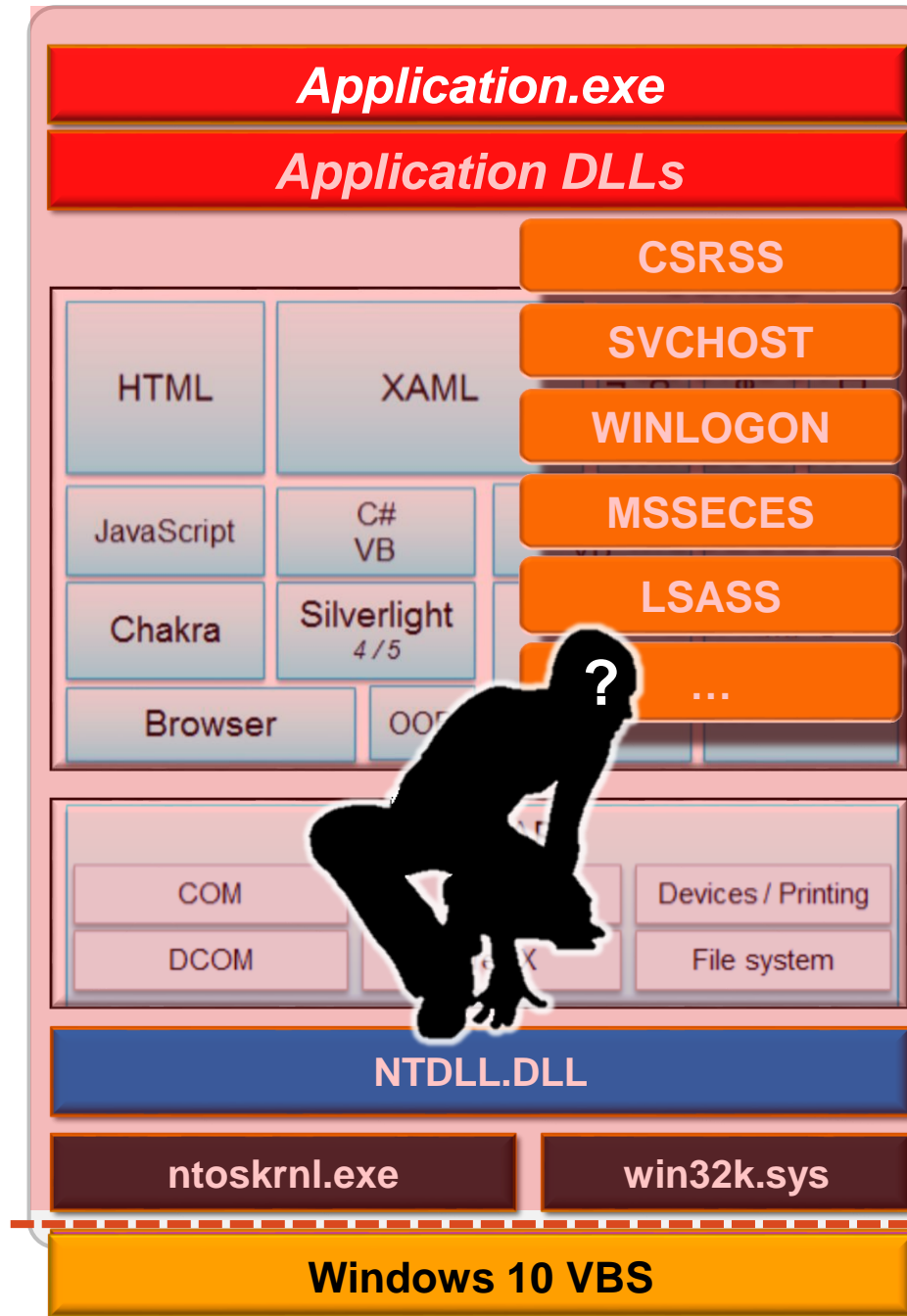
UNIFIED EXTENSIBLE  
FIRMWARE INTERFACE



Windows 10 Enterprise License

## Result

- A compromised PC is less valuable to attackers
- The device still has access to enterprise network and infrastructure (AD, Exchange, Intranet sites, shares)
- Files/data can be stolen
- A keylogger / screen scraper can steal non-Windows credentials



## Windows 10 Credential Guard

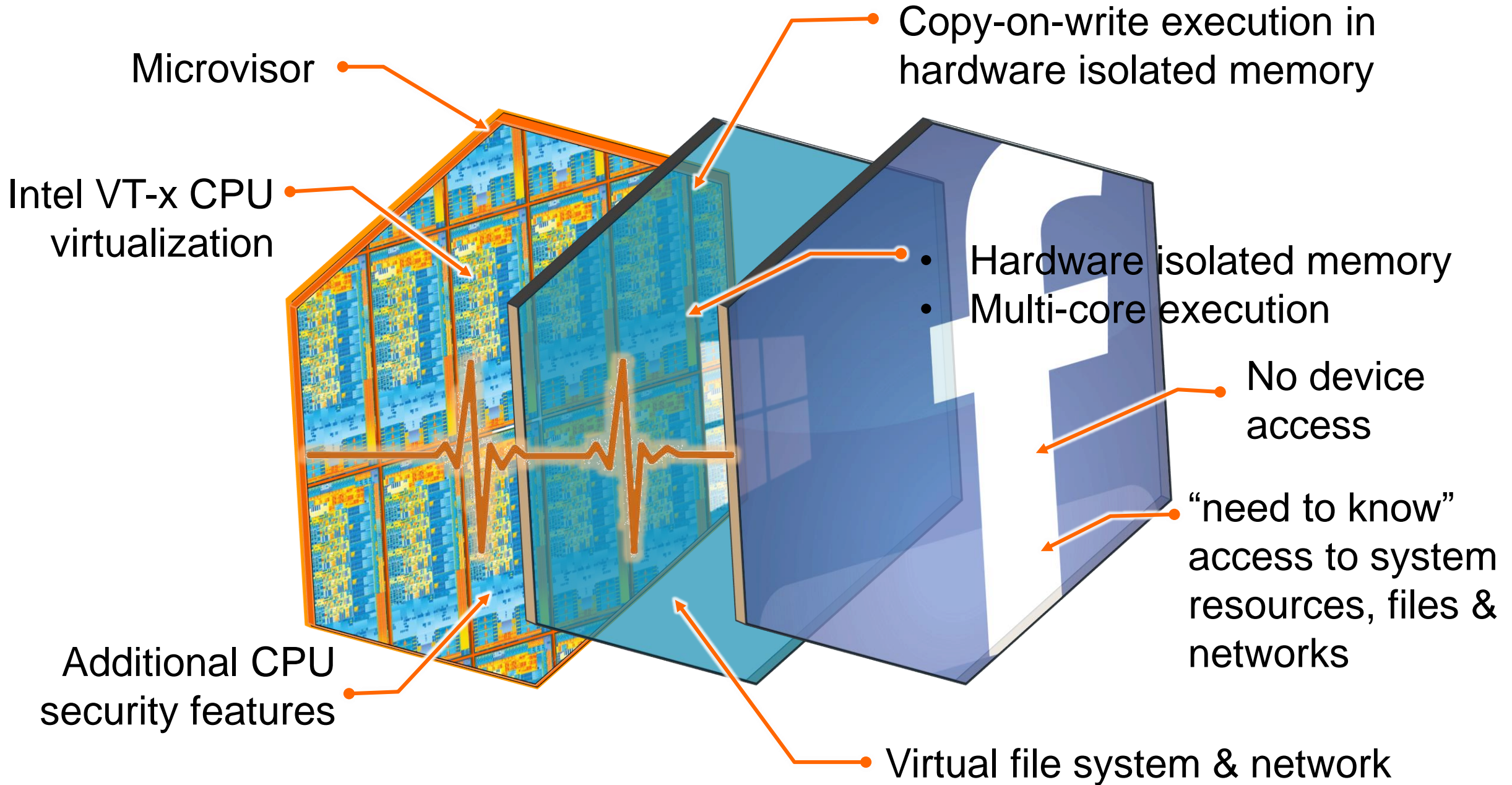
- Protects credentials using hardware virtualization
- Even if the OS is compromised, key data can't be stolen
- Eliminates credential theft and pass-the-hash attacks

Hardware Isolation  
(client Hyper-V)



**Br** Bromium<sup>®</sup>

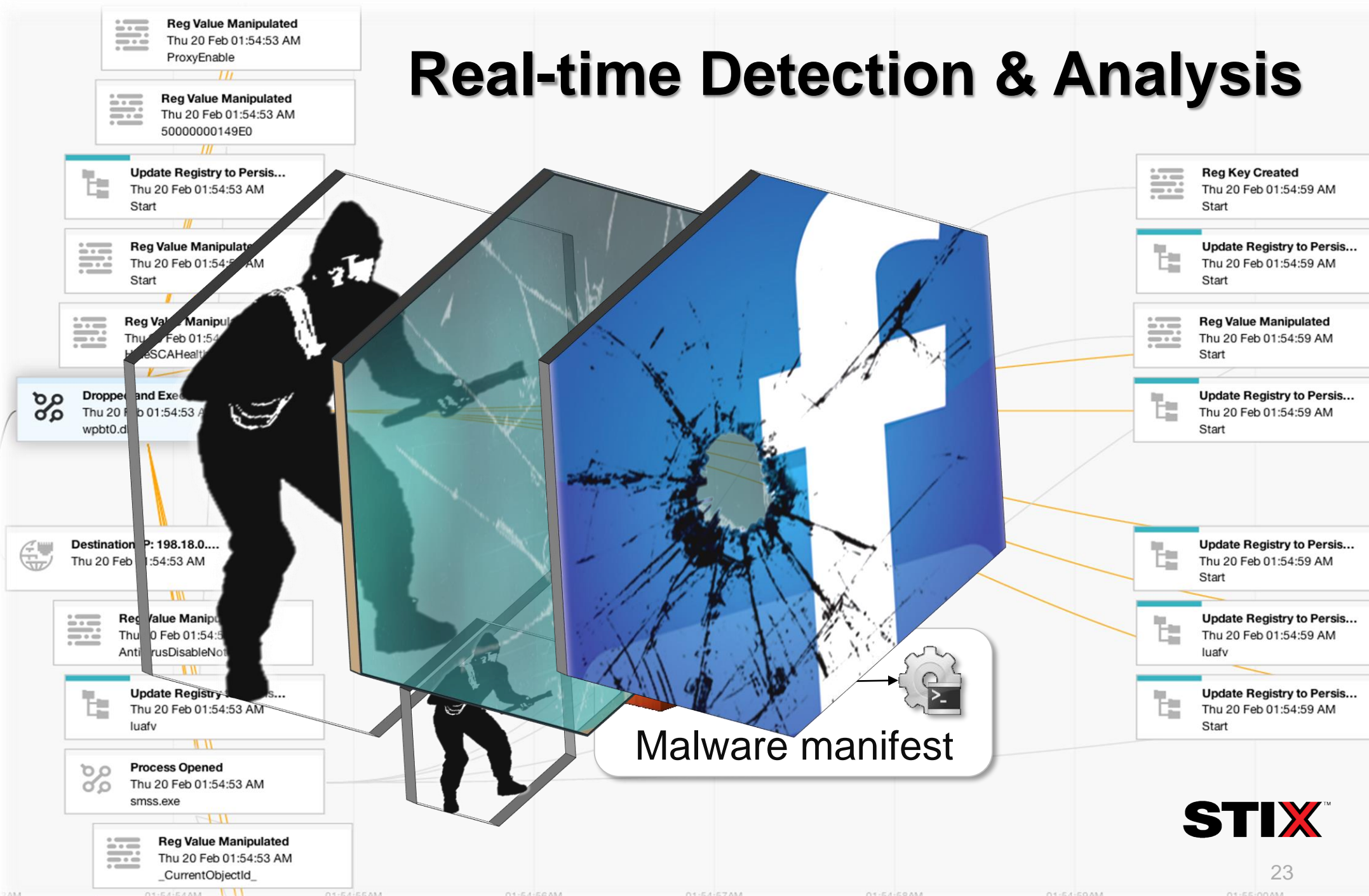


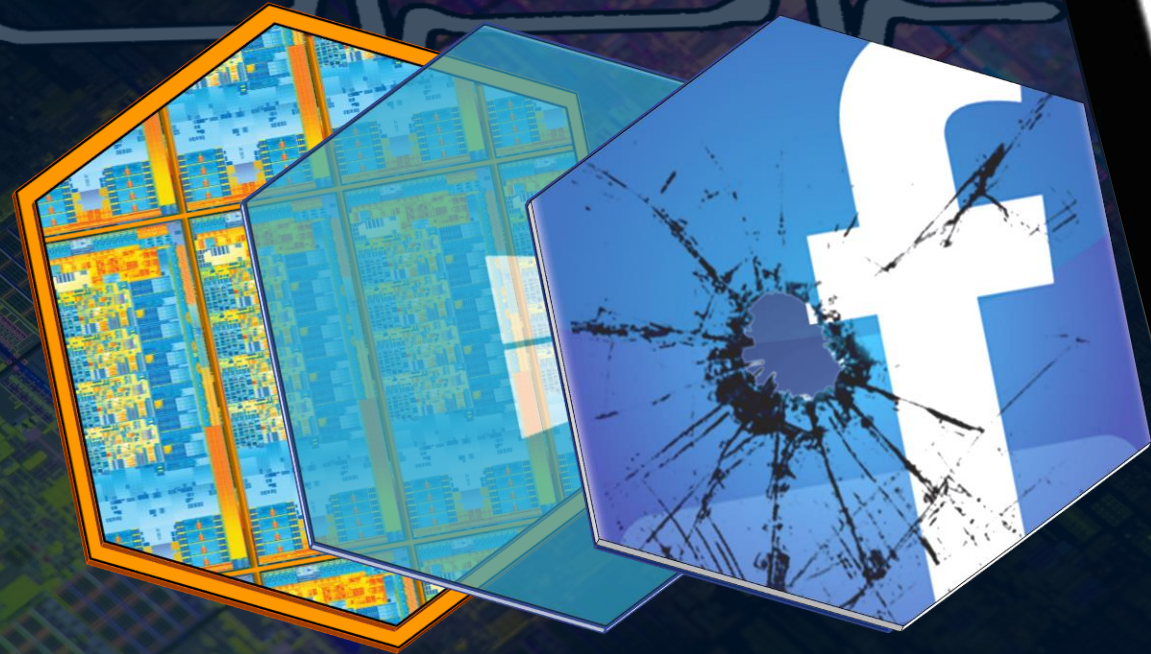


Protect • Detect • Respond



# Real-time Detection & Analysis





Self remediation eliminates persistence





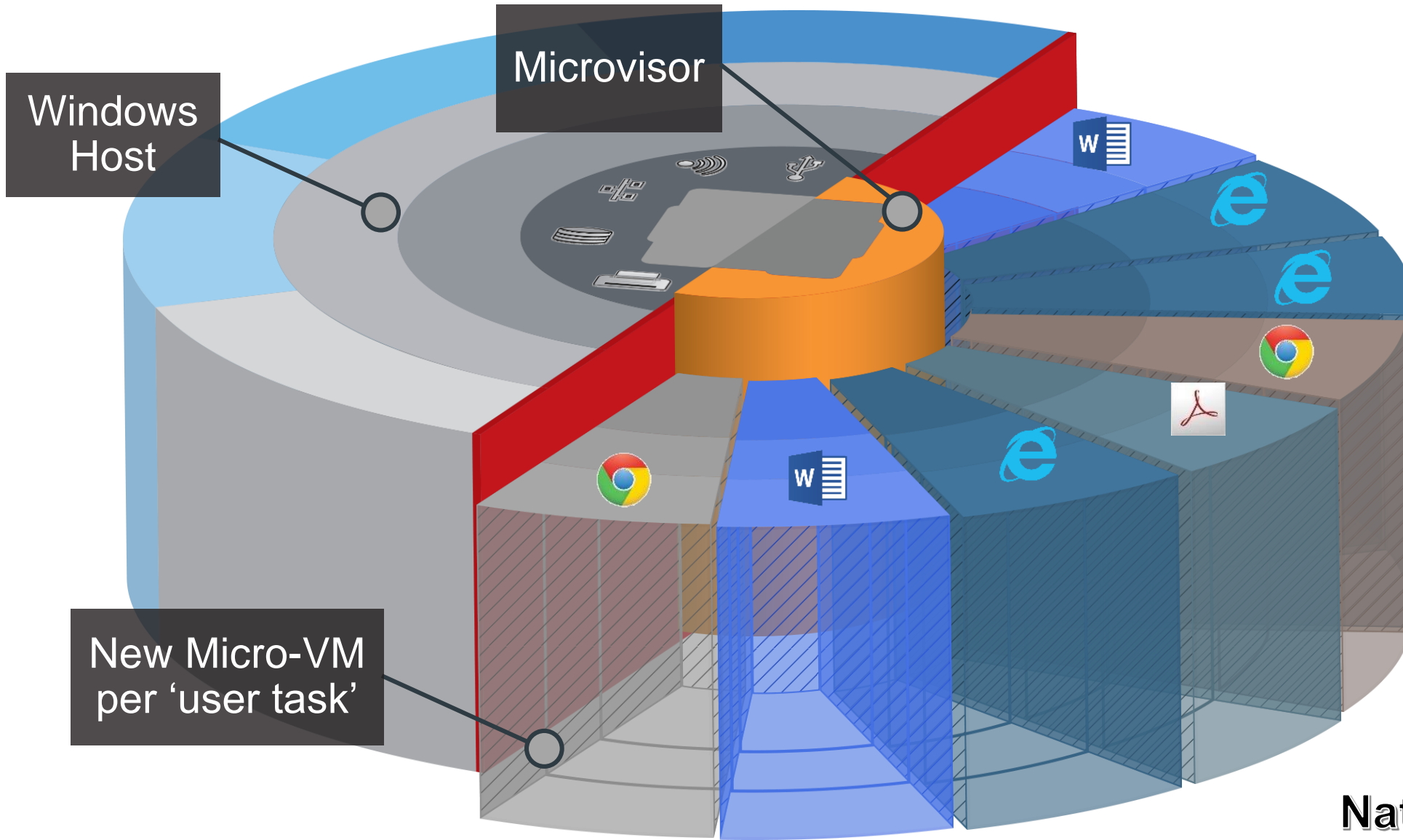
# Top 4 Ransomware Attack Vectors

Weaponized e-mail  
attachments

Embedded hyperlinks  
within emails

Browser (drive-by) attacks

Web application  
vulnerabilities

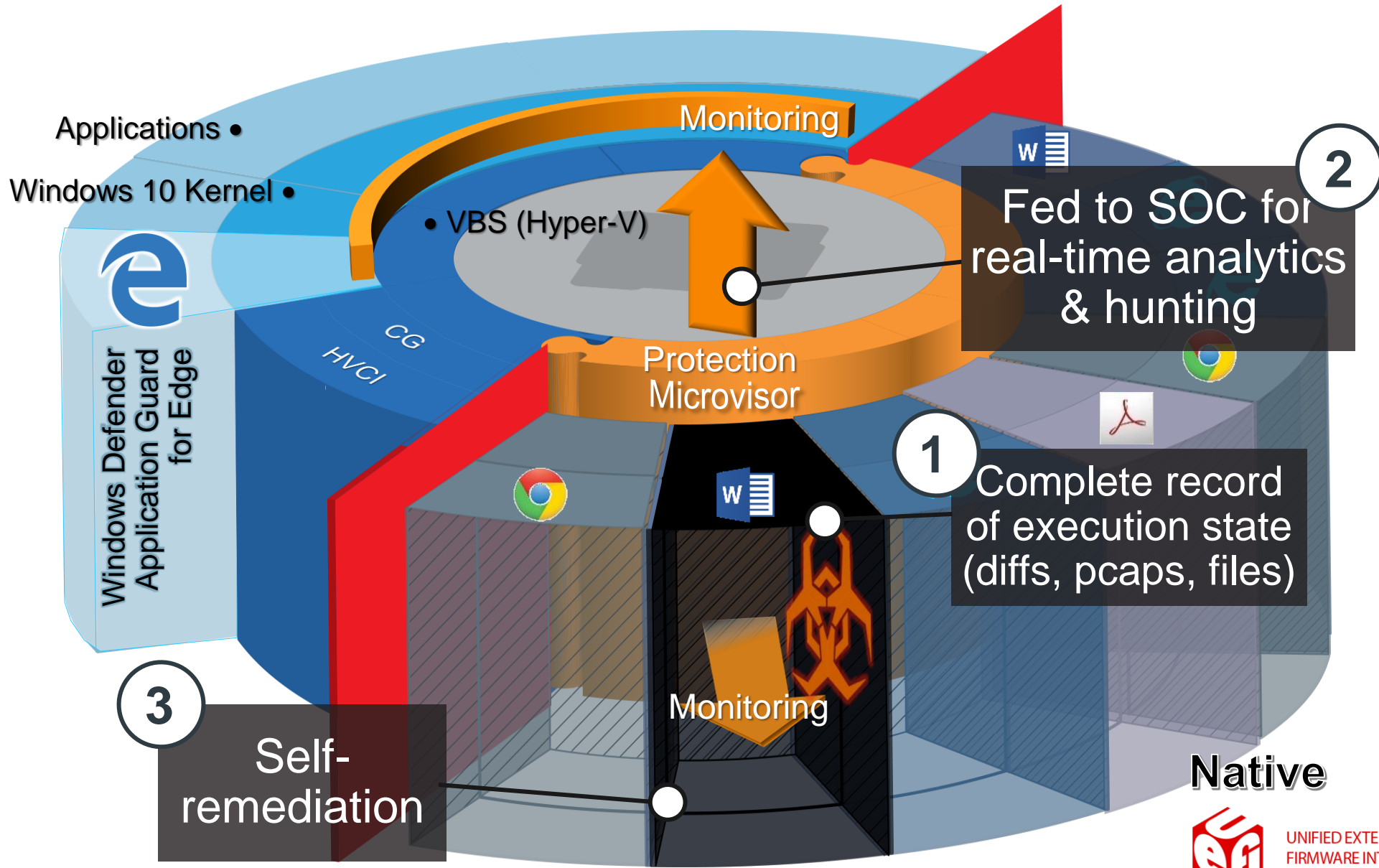


Windows Host

Microvisor

New Micro-VM per 'user task'

**Native  
BIOS Boot**  
(Windows 7, 8, 10)



Windows 10 Enterprise License

# Br Bromium Complements Microsoft Security



## Windows 10

- CG for pass-the-hash attacks
- HVCI kernel code integrity
- WDAG isolates Edge
- ATP for cloud based EDR

## Br Bromium®

- Protection & self-remediation for
  - Network-based attacks
  - File or data loss
  - Key-loggers & screen scrapers
  - Ransomware
  - Persistent APTs
  - Pass-the-hash attacks
  - All malicious execution
- Tamper-proof real-time monitoring (EDR) of isolated tasks and the Windows desktop



**Br** Micro-virtualization for the masses...



&



**Bromium**®

introducing

**HP Sure Click**

Browsing Security Solution

HP Sure Click provides hardware-enforced security for web browsers,  
protecting your PC from websites infected with malware, ransomware or viruses<sup>5</sup>.



# Next Steps

- Learn how hardware-enforced isolation is transforming Microsoft security  
<http://bit.ly/VBSPartnership>
- Watch this demo how Bromium stops ransomware on Windows 7, 8, or 10:  
<http://bit.ly/GoodbyeRansomware>

**Evaluate your top threat vectors then talk to us.**

