White Paper

Cybersecurity
Advanced Threat Detection

# Detecting Process Hijacking and Software Supply Chain Attacks Using Intel® Threat Detection Technology


**Author:**

**Zheng Zhang**
Principal Engineer

Intel Corporation

**Contributors:**

**Suriyaraj Natarajan**
Senior Security Researcher

**Amitrajit Banerjee**
Senior Software Engineer

Microsoft Corporation



Living off the land (LotL) and software supply chain attacks pose a significant security challenge in that they blur the boundary between benign and malicious programs. It is much harder for security products to accurately identify the threat and promptly respond when legitimate programs are attacked and start to misbehave.

Intel® Threat Detection Technology (Intel® TDT) Anomalous Behavior Detection (ABD) can help meet this challenge by leveraging the power of Intel CPU telemetry and machine learning heuristics. Instead of blindly trusting legitimate programs, ABD follows the zero-trust principle [1] and utilizes Intel® Processor Trace (Intel® PT), Last Branch Record (LBR) and Performance Monitor Unit (PMU) telemetries to continuously monitor the behaviors of known programs and verify whether they remain within their normal behavior boundaries. If these programs are attacked or experience unexpected errors, their execution behaviors will deviate. ABD will quickly notice the behavior deviations and generate real-time alerts to notify security products about suspicious activities within benign processes. By combining hardware security knowledge from Intel with the software security expertise from security providers (like Microsoft Defender for Endpoints®), ABD can help to cover the critical blind spot in benign process monitoring and meaningfully assist security products to better detect and stop advanced LotL attacks.

Partnering with Microsoft Defender for Endpoint Research team, ABD had been extensively evaluated using a wide range of benign workloads and attack scenarios. Our test results indicate that ABD can accurately detect common process hijacking techniques and some simulated supply chain attacks with very high signal-to-noise ratios. Through GPU acceleration and software optimization, ABD significantly reduced Intel PT telemetry collection and processing overhead, making it suitable for production deployment.


## 1 Introduction

To avoid being detected and blocked by security products, attackers have increasingly been pivoting toward living off the land (LotL) attacks [2], which hijack and misuse legitimate programs to perform malicious operations. LotL attack techniques include abusing system scripting or management tools (e.g., PowerShell, cmd.exe, WMI...), hijacking trusted system processes (e.g., File Explorer, Winlogon...), and contaminating upstream software supply chains (e.g., software updates or open-source libraries). The LOLBAS project [3] captures a list of benign binaries that are frequently targeted by LotL attacks.

LotL attacks pose a great challenge for existing security products, most of which are designed to detect known malicious files or behavior patterns. Because programs from reputable vendors are often considered benign and trusted, these programs are typically not closely monitored. Hiding malicious code within the context of benign program processes reduces the chances of being detected and allows attackers potentially unrestricted access to critical system resources.

To identify and prevent such sophisticated attacks, security products have adopted a defense-in-depth strategy: they deploy multiple layers of protection, using detection/identification technologies such as exploit detection and protection, behavior monitoring, network monitoring, and endpoint detection & response (EDR),

Table of Contents

to keep track of the behaviors and execution status of suspicious applications and identify artifacts associated known LotL attack techniques [4]. LotL attacks intentionally mix their malicious behaviors with the benign behaviors from the victim processes, so it is challenging to accurately identify LotL attacks without increasing the risk of false positives. As a result, security products sometimes miss the initial signs of attacks until significant damage is done.

To address the evolving threats, the security community has been researching anomaly-based detection solutions [5], which could potentially detect any attacks. Unfortunately, anomaly solutions are currently not widely used in production environments, because many of them tend to have higher false-positive rates (FPR) compared to content-based solutions. In addition, most anomaly detection solutions rely on signals from operating systems, so attackers can evade detection by mutating their OS behaviors and by tampering the integrity of OS signals.

## 1.1 Zero-trust Security

Zero-trust [1] is a network security principle based on the concept "Never trust, always verify." [6] Differing from traditional network security models that assume everyone inside the network perimeter is trusted by default, this model trusts nothing and no one. If a user or device needs access to resources, they always must prove their identity and will be continuously monitored, whether they are inside or outside of the network.

## 1.2 Hardware-Assisted Malware Detection (HMD)

Just as biological viruses can evolve to resist specific vaccines and medicinal treatments, modern malware has evolved to resist traditional network, file, and OS-based security technologies. The security community, including Intel and Microsoft, is developing technologies that harness CPU telemetry, such as performance monitor unit (PMU), last branch record (LBR) and Intel® Processor Trace (Intel® PT), as an additional data source for behavior detection [7] [8] [9] [10] [11]. HMD could help to improve security solutions' robustness and effectiveness, because CPU telemetry directly reflects the computational behaviors of the actual malware payloads. Such HMD-based solutions could also be more immune to common techniques for evading malware detection in that CPU telemetries are not directly accessible by user-mode programs.

## 1.3 Intel Threat Detection Technology

Designed to augment the existing security technologies, Intel Threat Detection Technology (Intel TDT) [12] utilizes a combination of CPU telemetry and machine learning heuristics to detect advanced malware threats, which can't be effectively handled by traditional software-based detection methods, based on their CPU behaviors. Intel TDT is deployed by leading security products (e.g., Microsoft Defender for Endpoint and Blackberry Spark UES Suite) to add an additional layer of protection for hundreds of millions of Windows PCs [13] from threats such as crypto-jacking and ransomware.

## 1.4 Intel TDT Anomalous Behavior Detection (ABD)

This paper presents Intel TDT Anomalous Behavior Detection (ABD), a hardware-based anomaly detection solution to help protect known, benign applications from unknown attacks. It extends the Zero-Trust principle from network security to application security: instead of blindly trusting known benign programs, ABD continuously monitors the runtime status of these applications and verifies that their behaviors are staying within normal boundaries. The ABD solution was then validated extensively with multiple real-world hijacking and ransomware attack examples in collaboration with Microsoft Security Research to evaluate its usage and investigate product integration.

1. **Training:** ABD utilizes control-flow telemetry in Intel® processors (including PT, LBR and PMU) and machine learning algorithms to master the normal control-flow behaviors of benign applications, as they execute in controlled clean environments.
2. **Detection:** ABD monitors the execution status of the monitored applications in production environments, and it detects control-flow deviations at real-time if the applications are attacked or experience unexpected errors.

ABD utilizes multiple CPU telemetry sources for training and detection, differing from previous HMD security solutions, most of which tend to use one type of CPU telemetry. This multi-telemetry approach allows ABD to provide more optimized security coverage for different client and cloud runtime environments because different telemetry sources have different characteristics in security, performance, and virtualization supports.

ABD detects control-flow anomalies in two steps:

1. Checks the correctness of individual control-flow transfers
2. Evaluates the magnitude of control-flow deviations observed within a given time window, generating alerts when deviations exceed a certain threshold.

To reduce the probability of false positives due to incomplete training, Intel also developed a continuous learning algorithm so that ABD can continuously update its models in a production environment through controlled incremental training. This unique capability enables ABD to continuously improve the accuracy of its models using the data observed in the field. ABD continuous learning process can be managed and augmented by security products. By leveraging their long-term contextual threat intelligence, security products can guide ABD on when to commit or roll back its incremental models and improve the quality and integrity of ABD continuous learning process.

Because Intel PT trace packets are highly compressed, Intel PT traces need to be decoded before they can be processed by ABD. The Intel PT data volume of some busy workloads can be voluminous, causing substantial overhead in Intel PT processing. To minimize this overhead, we developed a hardware-accelerated Intel PT decoder that offloads the algorithms to the Intel Core processor's integrated GPU.

We evaluated ABD's effectiveness and performance, as it monitored example applications against control-flow and supply chain attack samples that used various attack techniques. In our experiments, ABD reliably detected all attack samples with very high signal-to-noise ratios (SNR) (see Section 4.4, 4.6 and 4.7). The performance test results indicated that the GPU Intel PT decoder significantly reduced the ABD CPU overhead and increased Intel PT decoding throughput (see Section 4.8).

To summarize:

1. *Zero-trust and Hardware-assisted anomaly detection improve the security of benign programs.* ABD demonstrates that, by continuously monitoring known benign programs using CPU control-flow telemetries, ABD can accurately detect the aberrant program behavior caused by unknown control-flow attacks.
2. *Novel integration of multiple CPU telemetries improves the effectiveness and efficiency of hardware-based security solutions.* Instead of relying on a single telemetry, ABD seamlessly integrates multiple hardware telemetries (Intel PT, LBR and PMU), based on the customer requirements and the deployment environments, to monitor program behaviors and detect control-flow anomalies.
3. *Continuous learning improves the accuracy of anomaly detection.* Besides dynamic and static training, Intel TDT also supports managed, continuous online learning of novel control behaviors after deployment. This enables ABD to iteratively adapt its models and reduce the risk of false positives.
4. *GPU-accelerated Intel PT decoding opens door to wide adoption.* The Intel PT processing overhead has hindered the wide adoption of Intel PT-based security solutions. To overcome this obstacle, ABD includes an innovative hardware-based Intel PT decoder that offloads the compute-intensive algorithms to Intel integrated GPUs. That enables high PT-decoding throughput with very low CPU performance overheads (see 4.8.1)

ABD is designed to be integrated with security products (like Microsoft Defender for Endpoint) and deployed to Intel client systems. It provides a unique insight about benign process runtime status and can help to boost security products' capability to detect and stop zero-day process hijacking and software supply chain attacks at early stages.

The rest of this paper is organized as follows:

- Section 2 introduces the relevant Intel telemetries and surveys prior research on hardware-based anomaly detection.
- Section 3 describes the details of the ABD solution.
- Section 4 presents experimental results.
- Section 5 concludes with a discussion of future efforts.

# 2 Backgrounds and Related Works

## 2.1 Control-flow Hijacking Attacks

Control-flow hijacking attacks are designed to change the control-flows of clean applications to execute logic not originally designed by the application developers. There are two types of control-flow hijacking attacks:

- Code injection attacks [14] bring malicious code into a vulnerable program to change its execution flow.
- Code reuse attacks [15] introduce malicious control data into a victim program to change its behavior without injecting malicious code.

Examples of code injection attacks include remote thread injection, process hollowing, DLL (Dynamic Link Library) injection, etc., whereas the most common example of control-flow exploits include Return Oriented Programming (ROP) [16], Data Oriented Programming (DOP) [17] and Counterfeit Object-Oriented Programming (COOP) [18].

## 2.2 Software Supply Chain Attacks

Software supply chain attacks are an emerging threat class. Instead of directly attacking victim systems, they first penetrate the upstream software supply chains to indirectly attack victim systems. There are three common attack techniques [19]:

- *Hijacking Updates:* attackers infiltrate a software vendor's network and implant malware into updates so that downstream customers become infected when they install the tainted software.
- *Undermining Codesigning*: attackers create fake software updates, signed with valid digital certificates from legitimate vendors, and insert the contaminated software in regular updates.
- *Compromising Open-Source Code:* attackers insert malicious code into public open-source libraries that software vendors download and incorporate into their products.

Software supply chain attacks are typically carried out by highly skilled state actors, often associated with advanced persistent threat (APT) campaigns. Due to the stealthy and trusted nature of this attack vector, software supply chain attacks are extremely difficult to detect. They can cause substantial damages, as the recent SolarWinds [20] and Kaysea [21] attacks demonstrate.

| | Intel® Processor Trace | Last Branch Record (LBR) | Performance Monitor Unit |
|---|---|---|---|
| Collection Overhead | Low | Very Low | Very Low |
| Processing Overhead | High | Low | Very Low |
| Telemetry Density | Dense | Sparse | Sparse |
| Supported CFI Methods | Coarse & fine grain CFI | Coarse & fine grain CFI | Coarse grain CFI |
| Support Training | Yes | Limited | No |
| Detection Efficacy | Excellent (for both long and short CF attacks) | Good (for long CF attacks) | Limited (for long CF attacks) |
| Suitable Use Cases | Use cases that request high assurance | Use cases that request high performance | Use cases that request highest performance |

**Table 1.** Hardware telemetry options

## 2.3 CPU Control-flow Telemetries

Modern Intel CPUs have multiple telemetry sources that can provide information about the program control-flows:

- **Last Branch Record (LBR)**: records the traces of the most recent branches executed by the CPU into the LBR stack model-specific registers (MSRs).
- **Performance Monitor Interrupt (PMI):** captures the CPU instruction pointer when the interrupt is triggered, within interrupt trap frames.
- **Intel Processor Trace (PT):** captures the full control-flow traces of executed programs with minimal performance overhead.

Table 1 lists the characteristics and the suitable use cases of these CPU telemetries. Refer to Chapter 17 and Chapter 35 of Intel® 64 and IA-32 Architectures Software Developer's Manual Vol. 3A [22] for more information.

Among these telemetry sources, Intel PT is the only one that can provide full control-flow traces. This makes PT an ideal source for control-flow monitoring. For training, ABD uses Intel PT as the primary telemetry and LBR as the secondary telemetry. All three telemetry sources (Intel PT, LBR and PMU) can be used for detection, either individually or in combination.

## 2.4 Control-flow Integrity

Control-flow integrity (CFI) is a security principle designed to protect programs from control-flow exploits. It restricts and enforces the allowed target addresses of an indirect branch instruction to a small subset of target sites based on control-flow models extracted during source code compilation or binary analysis. Traditional CFI solutions are implemented through compilers and operating systems (for example, Clang CFI [23], Microsoft CFG [24]). These SW-base CFI solutions are proven to be effective in protecting forward edge control-flows (indirect call or jump instructions), but often only provide weaker protection for backward edge control-flow (return instructions).

Intel CPUs (since its 11th Gen Intel® Core™ mobile processors) support Intel® Control-flow Enforcement Technology (Intel® CET), a hardware-based CFI technology [25]. Intel CET helps protect forward edge control-flows through new indirect branch tracking instructions and backward edge control-flows through hardware-based shadow stacks.

CFI solutions are effective in detecting and stopping popular code-reuse exploit techniques (e.g., ROP, COP, JOP) against applications or binaries instrumented by CFI protection. However, CFI is ineffective in protecting un-instrumented applications or defending against exploits and malware attacks that do not violate the CFI policies.

## 2.5 Prior Research in Using Intel PT for Security

Researchers have explored using Intel PT to solve security problems before. For example, Griffin [26], Intel PT CFI [27] and FlowGuard [28] examined how to use Intel PT to assist CFI violation detection and enforcement in real-time. HeNet [29] classified Intel PT traces using Deep Learning Neural Network models to detect control-flow exploits offline. Barnum [30] detected Document Malware Attacks using an Intel PT-based offline anomaly detection. In 2021, CrowdStrike added Hardware Enhanced Exploit Detection, a new Intel PT-based exploit protection feature, into its Falcon sensor [31].

Despite that these early PT research demonstrated promising results, there are very few other wide-scale deployments of Intel PT-based security solutions today. The main reason for this is that the performance overhead using software-only solution to process and classify Intel PT traces is too high for deployment in production environments. In addition, the CFI and machine learning models trained with data collected only in lab environments don't scale well in diverse user environments.

ABD is the first implementation and application of its kind that applies intel PT telemetry for real time control-flow anomaly detection at endpoint systems. It addresses the above limitations with the following innovations:

- ABD uses multiple CPU telemetry sources to deliver the optimal performance and efficacy required for production deployment.
- ABD offloads Intel PT decoding logic to Intel Core processor's integrated GPU to free the CPU resources for the other system workloads.
- ABD's continuous learning capability to adapt its models based on new behaviors observed in customer systems help to ensure ABD can scale across diverse user environments and handle OS and software updates.

# 3 System Design

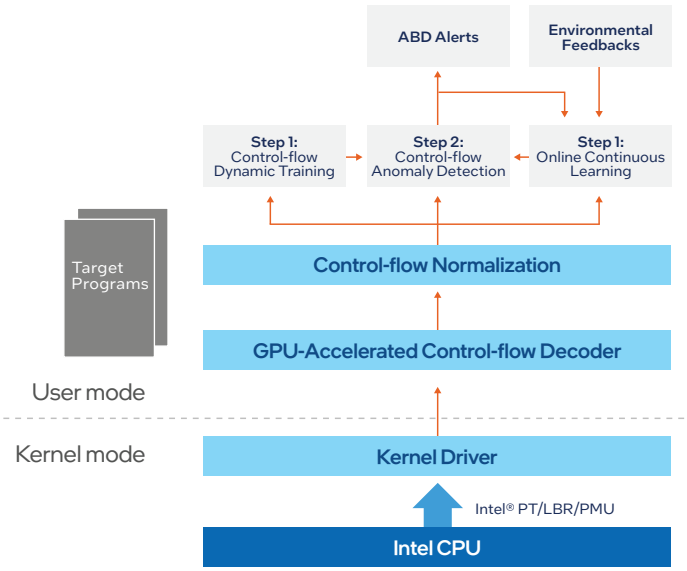



**Figure 1.** System architecture

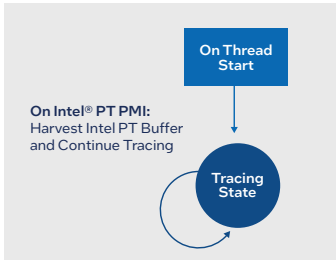
The high-level system diagram of the Intel TDT ABD solution is depicted in Figure 1. It consists of the following components:

- **Kernel Driver** collects OS events and CPU telemetries and forwards them to the user-mode modules.
- **GPU-accelerated Control-flow Decoder** decodes the Intel PT traces and extracts the control-flows of the monitored programs using GPU acceleration on Intel® Core Processor's integrated GPU.
- **Control-flow Normalizer** transfers the control-flows extracted from PT, PMU and LBR telemetries into a normalized format that is suitable for ABD training and detection.
- **Control-flow training** is the process to learn the ABD control-flow models of monitored programs using only benign Intel PT traces.
- **Control-flow anomaly detection** monitors the control-flows of the targeted programs running in production environments, and it detects any significant flow deviations as control-flow anomalies.
- **Continuous learning** receives environmental feedback (from users, security products, next level classifiers…) on previous ABD detection alerts and continuously update CFI models and heuristics.

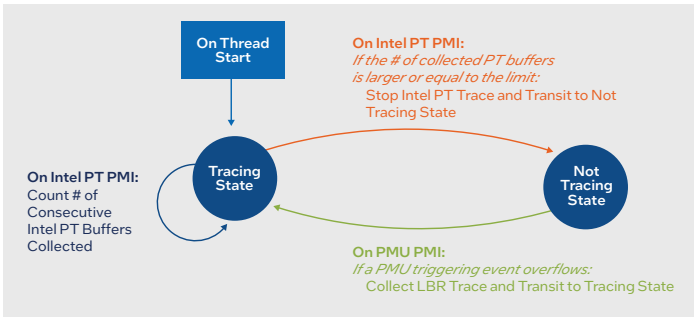
## 3.1 High-Performance Intel PT Tracing via Intel PT Sampling Mode

As described in Section 2.3, collecting Intel PT traces has a very low overhead but decoding Intel PT traces could incur significant processing overhead. To overcome this, existing Intel PT-based security solutions either choose to focus on offline analysis [29] or use trigger-based real-time analysis [26] [28] [31] [32], only when pre-defined events (e.g., critical system calls) have been triggered.

ABD is designed to detect control-flow anomalies at runtime without any pre-assumptions, so it does not rely on predefined triggers. Instead, ABD can continuously collect and process Intel PT traces, and it does so with low performance overhead.




**(a) Intel PT Continuous Modes**




**(b) Intel PT Sampling Modes**

**Figure 2.** Intel PT Continuous and Sampling Modes.

To achieve high performance Intel PT tracing, ABD uses two distinctive Intel PT tracing modes:

1. Intel PT Continuous Mode: in this mode (Figure 5.a), the Kernel Driver continuously collects Intel PT traces of a monitored thread and forwards them to the user-mode modules for processing.
2. Intel PT Sampling Mode: in this mode (Figure 5.b), the Kernel Driver utilized a combination of CPU telemetries for high-performance Intel PT tracing: Intel PT for high-efficacy control-flow tracing, PMU to trigger Intel PT tracing, and LBR for high-performance control-flow tracing.

For each monitored thread, the Kernel Driver maintains a finite state machine (FSM) of two states: Tracing State and Not-Tracing State. When the Kernel Driver starts to trace a new thread, it first enters Tracing Mode, where it will continuously collect Intel PT traces of the thread. When the Kernel Driver has collected a predefined number of Intel PT traces, it will configure one or multiple PMU trigger events, which will generate PMU interrupts on certain suspicious CPU behaviors (like excessive branch mispredictions, cache misses or TLB misses), and transit to the Not-Tracing mode, in which the Kernel Driver will not collect any CPU telemetry. When a PMU trigger event causes a PMU interrupt, the Kernel Driver will switch back to Tracing Mode, gather the LBR telemetry, which contains historic control-flow traces leading to the PMU interrupts, and then resume the Intel PT collection.

ABD training uses Intel PT Continuous Mode to stream Intel PT traces without gap. This allows ABD training to converge quickly. On the other hand, Intel PT Sampling Mode enables high-performance Intel PT tracing with balanced performance overhead and detection efficacy. Therefore, Intel PT Sampling Mode is primarily used in ABD Detection Mode.

## 3.2 GPU-Accelerated Control-flow Decoding

Because Intel PT packets are heavily compressed to minimize hardware collection overhead, Intel PT traces have to be decoded before they can be used for control-flow monitoring. Most prior Intel PT security research relied on an open-source Intel PT decoding library like libipt [33] to decode Intel PT packets. Open-source Intel PT-based fuzzing libraries (e.g., libxdc [34] and Honeybee [35]) have developed more optimized Intel PT decoders to speed-up fuzzing tests.

Despite these software optimizations, Intel PT decoding remains compute intensive. Because ABD is designed to run continuously on business client systems, we developed a novel method to offload Intel PT decoding to the integrated graphic processing unit (GPU) in Intel Core processors. In this way, ABD can minimize its CPU overhead, while simultaneously increases its Intel PT decoding throughput.

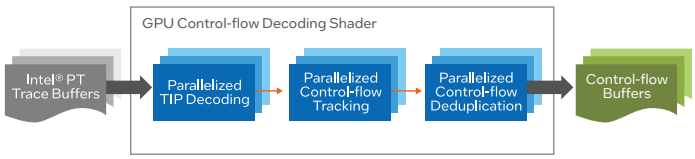



**Figure 3.** GPU-based control flow decoding.

Figure 3 shows the GPU-based control-flow decoding process, which includes the following steps:

1. **Intel PT trace buffering** caches multiple Intel PT trace buffers in a GPU memory buffer. Once enough Intel PT trace data has been accumulated, it forwards the Intel PT trace buffers to GPU Shader for parallelized control-flow decoding.
2. **TIP decoding** utilizes multiple GPU threads to decode the Intel PT packets in parallel. Because ABD training and detection only requires branch target address information contained within Intel PT TIP/FUP/PGE/PGD packets, the TIP decoder is optimized to process only these Intel PT packets and skip through the other unused Intel PT packets.
3. **Control-flow tracking** keeps track of the program flow of indirect branch targets and generates the program control-flows, which consist of tuples of (Previous TIP, Current TIP).
4. **Control-flow deduplicatio**n identifies and consolidates the duplicated control-flows within each Intel PT buffer. The output control-flow buffers consist of a sequence of control-flow frequency tuples (Previous TIP, Current TIP, Frequency).

Compared to the existing open-source Intel PT coding libraries, ABD's GPU-based control-flow decoder offers several distinct advantages:

- Simplified Intel PT decoding logic decodes only required Intel PT packet types and skip the other, unneeded packets.
- Parallelized GPU-based decoding helps to significantly reduce CPU processing overhead, while simultaneously increases the decoding throughput.
- Output decoding results as control-flows helps to meaningfully reduce the processing and memory overhead of ABD training and detection logic.

## 3.3 Control-flow Normalization

To guard against control-flow exploits, modern operating systems use Address Space Layout Randomization (ASRL) techniques to randomize the locations where executable and DLL files are loaded into memory. Due to randomization, the TIP addresses and control-flows of a program can vary at every run.

ASRL is highly effective in making viable control-flow exploits much more difficult, but it can also cause problems for ABD: its models won't be able to generalize from run to run and from system to system. To overcome this limitation, we designed a unique process to normalize the program control-flows with the assistance of critical OS events.

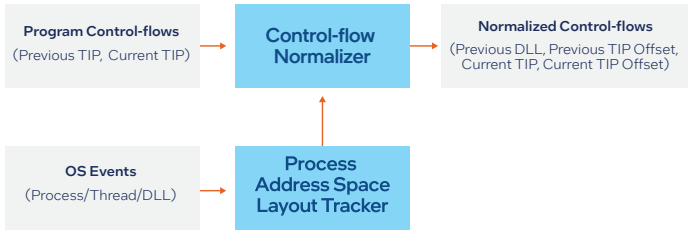



**Figure 4.** Control-flow normalization.

The normalization process is shown in Figure 4:

- **Process Address Space Layout Tracker** keeps track of the memory layout of running processes by using OS process, thread, and DLL events.
- **Control-flow Normalizer** converts the input control-flows (Previous TIP, Current TIP) into the output normalized control-flows (Previous DLL, Previous TIP Offset, Current DLL, Current TIP Offset). The Previous DLL and Current DLL fields are unique identifiers of DLL and EXE modules, to which the Previous and Current TIP addresses belong, and the Previous and Current TIP offsets are the offsets of the TIP addresses within the correspondent DLLs.

After normalization, the resulted control-flows become layout independent and portable among different runs and systems.

ABD also supports a feature to reduce the control-flow resolution: Instead of the default byte-level resolution, the normalized control-flows can have coarser resolutions (e.g., DWORD, QWORD or page levels). Our experiments indicate that a moderate resolution reduction helps to significantly reduce the complexity and the size of ABD models without affecting detection efficacy.

## 3.4 ABD Training and Detection

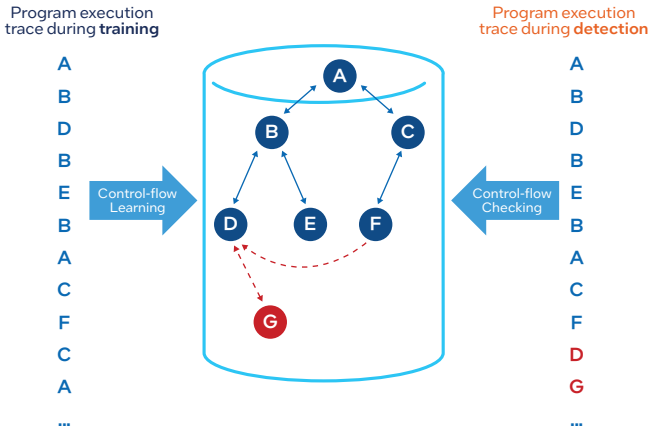



**Figure 5.** Control-flow learning and anomaly detection

Figure 5 depicts the high-level approach of the ABD training and detection phases:

- During the **Training Phase**, ABD monitors the runtime behaviors of monitored programs, when they are executed in clean environments without attack, to learn and build their control-flow graph models.
- During the **Detection Phase**, ABD monitors the control-flows of monitored programs, when they are executed in production environments and could be attacked. ABD continuously verifies whether the program control-flows comply with their respective control-flow models.
- If ABD detects significant control-flow deviations, it generates alerts to notify users and security management software of the suspicious behaviors.

### 3.4.1 Control-flow violations

ABD can identify four different types of control-flow violations:

- *Coarse-grain DLL violation:* if a control-flow target address is not within any known DLLs within the program control-flow model.
- *Coarse-grain TIP violations:* if a control-flow target address is within a known DLL, but the address is not within the program control-flow model.
- *Fine-grain DLL violation:* if the DLL control-flow (Previous DLL, Current DLL) is not within the control-flow model.
- *Fine-grain TIP violation:* if the TIP control-flow (Previous TIP offset, Current TIP offset) is not within the control-flow model.

### 3.4.2 Timer Series Analysis

ABD applies a time series analysis heuristic on the control-flow violations observed within each time window. If control-flow violation rates exceed pre-defined thresholds, ABD will generate control-flow alerts to notify users and security management software about suspicious behaviors.

### 3.4.3 Continuous Learning

Because the configurations and the user behaviors of production systems can be different from those of training systems, the control-flows of monitored programs might also be different. Control-flow models trained solely on the data collected from backend training systems might have false positives in production environments. To reduce false positives, ABD uses continuous learning algorithms to iteratively update the control-flow models based on the program behaviors it observed after deployment. To ensure the integrity of updated control-flow models, the continuously learning process is managed by the security software products and IT admins through ABD model management APIs (to enable, disable, commit and rollback TDT continuous learning).

### 3.4.4 Static Learning

Although ABD primarily relies on dynamic training, it also supports control-flow training through static binary analysis: it statically analyzes the executable and library files loaded by monitored programs to extract static control-flows. Because dynamic training might not be able to execute all possible code paths of complex programs, the resulting ABD dynamic model might have insufficient code coverage, which could cause false positives in production environments if untrained code paths are triggered.

Most existing CFI solutions require application recompilation to extract the control-flow information and to embed the CFI verification instructions into the binaries. Because ABD relies on CPU telemetry to monitor program control-flows, it does not require applications to be recompiled. Instead, ABD developed an extension to the open-source reverse engineering tool Ghidra [36] to statistically disassemble application binaries and extract program control-flows. With this unique process, ABD can support legacy applications and coexist with existing CFI solutions.

## 3.5 Handling OS and Software Updates

For dynamic systems like consumer and enterprise devices, OS and software updates can happen frequently, and sometimes these updates are not fully managed by users or IT admins. Because ABD models are application and OS specific, false positive rates could increase after a software update.

To mitigate this risk, ABD detects OS and software changes at runtime and notifies the ISV products, when program updates are detected. ISVs and IT admins will determine whether these changes are legitimate or not. For legitimate OS and software changes, ISVs can choose to update ABD models using the backend training process or allow the models to be updated locally through the ABD training and continuous learning processes outlined in Section 3.4.

# 4 Experimental Results

This section presents the results of ABD performance and efficacy tests.

## 4.1 Experimental Setup

We developed an ABD prototype for Windows 10 and later OSs. The prototype consists of a Kernel Driver, a user-mode application and a set of ABD model and configuration files. The experiments are conducted from Windows root partitions without Hyper-V on systems with integrated Intel® GPUs on 10th gen Intel® Core™ processors or later. These evaluations were performed in partnership with the Microsoft Defender for Endpoint research team. They served as a proof-of-concept on how ABD could be leveraged by security products to detect and prevent hijacking and ransomware threats.

## 4.2 Threat Model

In this paper, we tested ABD on a wide range of unprivileged user mode control-flow attacks. We used both attack simulators that model known attack techniques and real-world malware samples known to attack benign applications. Refer to Table 2 for more information about the simulated attacks and Table 3 for more information about the live malware samples.

| Attack Type | Attack Details |
|---|---|
| **Remote thread injection attack** | This simulator models the MITRE Adversarial Tactics, Techniques & Common Knowledge (ATT&CK) technique T1055 [37]. This technique is widely used by many malware families to inject malicious code into benign processes.<br>This simulator is developed in-house and is capable of injecting malicious payloads (such as cryptomining and side channel attacks) into benign processes. |
| **DLL Side-Loading** | This simulator models the MITRE ATT&CK technique T1574.002 [38]. Attackers aim to take advantage of weak Dynamic Link Library (DLL) references and the default Windows search order by placing a maliciously crafted DLL file (masquerading as a legitimate/clean DLL file) for automatic loading by the legitimate application.<br>This simulator is based on the testing tool [39] developed by the open-source Atomic Red Team [40]. |
| **Supply Chain backdoor attack** | This simulator models the MITRE ATT&CK technique T1195 [41] which had been used by the high profile SolarWinds and Kaseya supply chain attacks.<br>This simulator models a scenario that an attacker embeds a custom hidden backdoor to a notepad++ [42] process. The backdoor remains dormant until triggered by users. Once activated, the backdoor establishes a remote shell session with an attacker-controlled server. Through the remote shell, the attacker can remotely execute any command, and it will be executed by the custom backdoor. Attackers can also push additional payloads to victim machines (also executed by the backdoor). An example payload could be a Meterpreter payload, which allows execution of more advanced commands through the Metasploit framework (see below). |
| **Metasploit [43] attack** | Metasploit is an open-source offensive security tool that aids security researchers for vulnerability scanning and penetration testing. It is also widely abused by bad actors for exploitation, command & control (C2) and lateral movements [44].<br>This simulator models the 2nd stage attack after the initial successful supply chain attack. Attackers use Metasploit to carry out human-operated hacking attacks (e.g., privilege escalation, process migration and information stealing) within benign process contexts. |
| **Zero-day exploit attack** | PrintNightmare (CVE-2021-1675 and CVE-2021-34527) [45] is a critical vulnerability in the Windows Printer service that could be remotely exploited by attackers for remote code execution (RCE) and privilege escalation.<br>This simulator is based on the open-source PrintNighmare proof of concept (POC) [46] and simulates the remote exploitation of the Windows Printer service. |

**Table 2.** Tested Attack Simulators

| Malware Sample | Malware Details |
|---|---|
| **Qakbot** | SHA: 1073e9f9582472f445080f4017b5054f7b6db5476b6fca53c8255a002f2dfdb8<br>Qakbot [47] has been a prevalent malware family since 2007. Beside stealing information, the malware is also used as a "malware installation as a service" botnet, and it is frequently associated as a precursor to many ransomware attacks. Process hijacking is a critical building block of Qakbot kill chains. During its infection process, Qakbot can attack multiple benign processes for discovery, data stealing & exfiltration and malware download & installation.<br>The Qakbot sample we tested is known to inject into a benign process, msra.exe. It performs malicious activities such as downloading additional modules for further infection. |
| **Cobalt Strike** | SHA:950008035d225dd5f4c3a229082f1206eb9bce8c4aa4822b130db065da54e224<br>Cobalt Strike [48] is a commercial offensive security tool that simulates adversary attacks and red team operations. Although Cobalt Strike is a legitimate software tool, it has been pirated and widely used by many malwares and advanced persistent threat (APT) campaigns. For example, the infamous SolarWinds [20] attack used Cobalt Strike for its second stage exploitation.<br>Cobalt Strike's process injection capability [49] allows attackers to execute code within an arbitrary process. In this experiment, we simulated a Cobalt Strike attack that injected into dllhost.exe. |
| **Trickbot** | SHA: 10001f3da1757be0861df508d85434fbf6c6c422ad9e388d8e37cc7861a0bbea<br>Trickbot [50] is a credential theft trojan with sophisticated reconnaissance and persistence capabilities. It is also a malware-as-a-service botnet that is known to deploy additional payloads (such as Ryuk ransomware) into victim networks.<br>Trickbot relies on process injection techniques to run arbitrary code within the context of benign processes. The sample we tested injected into SVCHost.exe to perform the malicious action. |

**Table 3.** Tested Malware Samples

The applications used in this test are benign system and benchmark workloads (e.g., notepad, Edge, PCMark10...). For the software supply chain test, we developed a custom backdoor, and we embedded it into an example open-source project, notepad++.

We further assumed that during the training phases, these benign workloads and the infected notepad++ were not attacked. As a result, ABD was able to learn normal behaviors and create ABD models for these applications.

## 4.3 Training Convergence Test

This experiment is designed to measure the speed of ABD training convergence. We trained ABD models for a selected set of benign workloads in multiple iterations. In each iteration, we simulated typical user behaviors. The training continues until ABD detects zero false positives in two consecutive iterations.

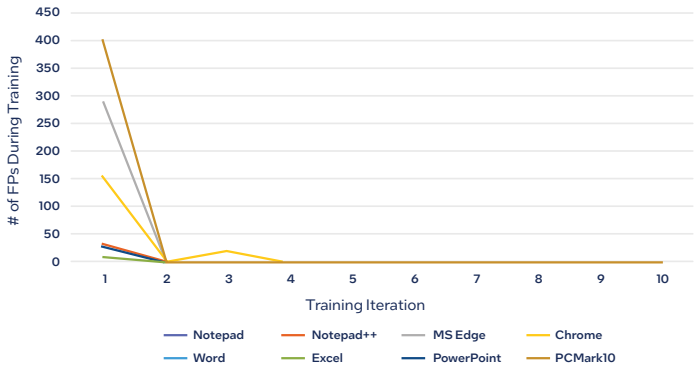



**Figure 6.** Training convergence tests

The results of the convergence tests are visualized in Figure 6. They show that ABD converges quickly on these workloads. Except for PCMark10, all workloads converged within four training iterations. The FP detection of PCMark10 follows a similar curve and drops close to zero from iteration two, but as a composite benchmark consisting of multiple complex workloads, PCMark10 continued to have few random FPs until the 10th iteration.

## 4.4 False Positive Test

This experiment is designed to evaluate risks of false positives with ABD when monitoring common benign workloads. We chose a subset of representative workloads in Windows client systems and trained converged ABD models with them. We ran ABD in detection mode and measured the averages and standard deviations of its control-flow violation rates.

The false positive results are shown in Table 4, indicating the violation rates of these workloads are close to zero.

| Workload | Average of control-flow violation rates | Standard deviation of control flow violation rates |
|---|---|---|
| Notepad | 0.004157532 | 0.00762242 |
| Notepad++ | 0.000730498 | 0.0009794 |
| Edge | 0.002421217 | 0.00818848 |
| Chrome | 0.003553968 | 0.01944568 |
| Word | 0.002093558 | 0.00146446 |
| Excel | 0.010827592 | 0.02282338 |
| PowerPoint | 0.002883157 | 0.00680268 |
| PCMark10 | 0.004746845 | 0.02119948 |

**Table 4.** False Positive Test Results

## 4.5 Model Size Test

To deploy ABD at endpoints, its models need to be downloaded and stored in local systems. In this experiment, we want to better understand the download and storage requirements of ABD models. We measured the sizes of ABD model files of the workloads that we tested in the training convergence test. We measured the size of compressed model files (which will more closely reflect the download size requirements). Because multiple application models could be stored in a model file, we also calculate the average model sizes per application.

The test results are summarized in Table 5. Although most of tested workloads involve more than one application, PCMark10 includes an extraordinarily large number of applications. That is expected because PCMark10 is a composite benchmark that consists of many sub-benchmarks. The model file sizes are dependent on both the number of applications in the model and the application complexity. The average uncompressed model size is 1.64MB per application, whereas the average compressed model size is 298KB per application.

| Workload | Number of Apps in the model file | Uncompressed model file size (MB) | Compressed model file size (MB) | Average uncompressed model size per app (MB) | Average compressed model size per app (MB) |
|---|---|---|---|---|---|
| Notepad | 1 | 1.188 | 0.22 | 1.188 | 0.22 |
| Notepad++ | 4 | 3.388 | 0.598 | 0.847 | 0.1495 |
| Edge | 2 | 4.576 | 0.82 | 2.288 | 0.41 |
| Chrome | 1 | 3.564 | 0.631 | 3.564 | 0.631 |
| Word | 1 | 2.096 | 0.398 | 2.096 | 0.398 |
| Excel | 2 | 1.98 | 0.381 | 0.99 | 0.1905 |
| PowerPoint | 2 | 3.968 | 0.718 | 1.984 | 0.359 |
| PCMark10 | 151 | 30.932 | 3.913 | 0.204848 | 0.025914 |

**Table 5.** Results of Model Size Tests

## 4.6 Detection Test on Simulated Attacks

This experiment is designed to evaluate the effectiveness of ABD in detecting common control-flow and supply chain attacks. We tested the simulated attacks listed in Table 2. The ABD detection results on these attacks are given in Table 6.

| Attack Type | Victim Process | ABD Detection |
|---|---|---|
| Remote thread injection | Notepad | Yes |
| Remote thread injection | Word.exe | Yes |
| DLL Side-Loading | GUP.exe [39] | Yes |
| Supply chain attack with Metasploit backdoor | Notepad++ | Yes |
| Meterpreter migration attack | Notepad++ | Yes |
| PrintNightmare remote execution exploit | Spooler.exe | Yes |

**Table 6.** False Positive Test Results

For comparison, we also evaluated some popular endpoint security products to verify whether they were able to detect these attacks. We found that most of the tested security products do not detect any of these attacks, although some were able to detect the Metasploit backdoor and migration attacks. We also observed that the custom-made backdoor simulator that we developed was not detected by any of the tested security products. Because Metasploit is a well-known penetration testing tool and has been used by real-word attacks before, we suspect some of these Metasploit detections might be related to certain signatures or heuristics designed to specifically detect known Metasploit samples or behaviors. Therefore, these Metasploit detections might not be a reliable indicator of how robustly these products can proactively detect zero-day attacks.

## 4.7 Detection Test on Real World Malicious Samples

This experiment is designed to be a reality check test to verify if ABD can have similar efficacy on detecting real-world malware attacks. We gathered recent, prevalent malware samples known to attack clean processes (to gain privileged access and evade detection). The test results are shown in Table 7, showing that ABD is effective in detecting these malware samples.

| Malware Classification | Victim Process | ABD Detection |
|---|---|---|
| Qakbot | msra.exe | Yes |
| Cobalt Strike | Dllhost.exe | Yes |
| Trickbot | Svchost.exe | Yes |

**Table 7.** ABD Detection Results on Malware Samples

Although security products do have other means to detect these LotL attacks (by monitoring application OS behaviors and static artifacts), these detection methods typically require deep understanding about each attack vector and frequent updates to handle new variants. The hardware-based anomaly detection approach used by ABD is very different from existing detection methods. It has proven to be capable of generically detecting many LotL techniques without prior knowledge of these attacks. We believe ABD can be a valuable addition to enhance security products' capabilities to proactively detect zero-day LotL attacks.

## 4.8 Preliminary Performance Test Results

To accurately estimate the performance overhead of ABD detection and the impact of GPU-accelerated Intel PT decoding, we conducted two experiments:

- Compare the performance of CPU and GPU Intel PT decoders
- Measure the ABD performance overhead when monitoring benchmark workloads

### 4.8.1 Intel PT Decoding Performance Test

We tested decoding throughput, CPU utilization and GPU utilization of ABD-based Intel PT decoders when decoding the same Intel PT dump file. For comparison purposes, we also tested the throughput of libipt [33] Intel PT decoder on the same Intel PT dump file.

This experiment was conducted on 11th gen Intel® Core™ mobile system with 4 cores and Win10 21H1 OS. Four different PT decoding configurations were tested:

- The **libipt CPU Single Thread** configuration decodes the PT dump files using the standard libipt library in a single thread.
- The **ABD CPU Single Thread** configuration decodes the PT dump files using ABD's optimized CPU PT decoder in a single thread.
- The **ABD CPU Multi Thread** configuration decodes the PT dump files using ABD's optimized CPU PT decoder in multiple threads.
- The **ABD GPU** configuration decodes the PT dump files using ABD's optimized GPU PT decoder.

| Decoder Tests | Decode Throughput | CPU Util | GPU Util |
|---|---|---|---|
| Libipt CPU Single Thread | 112.9 MB/s | 12.50% | n/a |
| ABD CPU Single Thread | 266.8 MB/s (2.36x) | 12.50% | n/a |
| ABD CPU Multi Thread | 583.7 MB/s (5.17x) | 44.19% | n/a |
| ABD GPU | 341.4 MB/s (3.02x) | 0.25% | 88.77% |

**Table 8.** Preliminary Intel PT Decoding Tests

| Workload | Intel PT continuous mode | | Intel PT sampling mode<br>▪ Every 1M instructions<br>▪ 1 PT buffer/trigger | | Intel PT sampling mode<br>▪ Every 50M instructions<br>▪ 1 PT buffer/trigger | |
|---|---|---|---|---|---|---|
| | CPU decoder | GPU decoder | CPU decoder | GPU decoder | CPU decoder | GPU decoder |
| Speedometer (Edge) | Score: -18.4%<br>CPU Util: 15.8%<br>GPU Util: 0% | Score: -16.4%<br>CPU Util: 12.1%<br>GPU Util: 16.1% | Score: -17.9%<br>CPU Util: 15.5%<br>GPU Util: 0% | Score: -16.4%<br>CPU Util: 9.7%<br>GPU Util: 13.5% | Score: -2.4%<br>CPU Util: 1.3%<br>GPU Util: 0% | Score: -2.9%<br>CPU Util: 0.9%<br>GPU Util: 0.6% |
| Jestream2 (Edge) | Score: -23.5%<br>CPU Util: 19.3%<br>GPU Util: 0% | Score: -16.7%<br>CPU Util: 16.7%<br>GPU Util: 23.5% | Score: -20%<br>CPU Util: 16.7%<br>GPU Util: 0% | Score: -15.0%<br>CPU Util: 6.7%<br>GPU Util: 14.2% | Score: -3.0%<br>CPU Util: 1.0%<br>GPU Util: 0% | Score: -3.0%<br>CPU Util: 0.9%<br>GPU Util: 1.0% |
| PCMark10 | Score: -5.6%<br>CPU Util: 7.0%<br>GPU Util: 0% | Score: -6.5%<br>CPU Util: 4.3%<br>GPU Util: 9.2% | Score: -5.6%<br>CPU Util: 5.9%<br>GPU Util: 0% | Score: -5.6%<br>CPU Util: 3.9%<br>GPU Util: 9.0% | Score: -4.3%<br>CPU Util: 0.9%<br>GPU Util: 0% | Score: -0.4%<br>CPU Util: 0.9%<br>GPU Util: 1.05% |
| Average | Score: -15.8%<br>CPU Util: 14.0%<br>GPU Util: 0% | Score: -13.2%<br>CPU Util: 11.0%<br>GPU Util: 16.3% | Score: -14.5%<br>CPU Util: 12.7%<br>GPU Util: 0% | Score: -12.3%<br>CPU Util: 6.8%<br>GPU Util: 12.2% | Score: -3.2%<br>CPU Util: 1.1%<br>GPU Util: 0% | Score: -2.1%<br>CPU Util: 0.9%<br>GPU Util: 0.9% |

**Table 9.** Preliminary Results of ABD Performance Tests

The results of Intel PT decoding tests are shown in Table 8. The results indicate:

1. The optimized CPU Intel PT decoder in ABD can decode 2.36 times faster than the libipt decoder.
2. CPU-based Multi Thread Intel PT decoder has the highest decoding throughput.
3. GPU-based Intel PT decoder can decode 3 times faster than the baseline libipt decoder with only 0.25% CPU utilization.

This experiment shows that the GPU-based Intel PT decoder can significantly reduce CPU utilization and meaningfully increase Intel PT decoding throughput.

### 4.8.2 ABD Detection Performance Tests

In this experiment, we evaluated the performance overhead of ABD when monitoring some common enterprise benchmark workloads on an an 11th gen Intel® Core™ mobile system with 4 CPU cores and Win10 20H2 OS. We tested CPU and GPU utilization observing benchmark impacts with Intel PT continuous and Intel PT sampling modes. The test results are listed in Table 9.

Table 9 indicates that Intel PT sampling mode has a big impact on the monitoring overhead of ABD. With the optimized Intel PT sampling configuration, performance overhead can be reduced from ~15% to ~3% on these stress benchmark workloads. On the other hand, the GPU-based Intel PT decoding is very effective in reducing CPU utilization. When ABD runs in Intel PT continuous mode and Intel PT sampling mode with very frequent triggers, the GPU-based Intel PT decoding can reduce 20% to 50%+ of CPU utilization for ABD and ~15% of the benchmark overhead. When running in an Intel PT sampling model with very infrequent triggers, the GPU decoder overhead is consistently lower than the CPU decoder overheads (although both CPU and GPU decoders have very low overhead).

## 5 Conclusion

Intel TDT Anomalous Behavior Detection (ABD) is a hardware-based control flow monitoring and anomaly detection solution. We introduced ABD architecture, system optimization and ABD training and detection algorithms. Our test results showed that ABD can detect a wide range of attack and exploitation techniques with few false positives. We also presented ABD performance results which demonstrated that GPU offloading by ABD can significantly reduce CPU utilization. ABD can be integrated with security products (like Microsoft Defender for Endpoint) to plug the critical gap in benign process monitoring and improve their capability to proactively detect zero-day control flow and supply chain attacks. On the other hand, the contextual intelligence from security products can help to protect the integrity of the ABD continuous training phase and make ABD suitable to be deployed at complex customer environments.

As the next steps, we are planning to investigate how ABD can be used to detect attacks to OS kernels and how ABD can be applied in Cloud and IOT environments.

## Acknowledgement

We thank the Intel TDT team for developing the Intel ABD solution, and we thank the Microsoft Defender for Endpoint team for testing ABD and sharing valuable feedback.

# 6 References


[1] National Institute of Standards and Technology, "Zero Trust Architecture," August 2020. [Online]. Available: https://csrc.nist.gov/publications/detail/sp/800-207/final.

[2] The Driz Group, "What Is Living Off the Land Attack and How to Prevent Such Attack," The Driz Group, 21 June 2021. [Online]. Available: https://www.drizgroup.com/driz_group_blog/what-is-living-off-the-land-attack-and-how-to-prevent-such-attack.

[3] "LOLBAS," [Online]. Available: https://lolbas-project.github.io/.

[4] MITRE, "MITRE ATT&CK Matrix for Enterprise," MITRE, [Online]. Available: https://attack.mitre.org/.

[5] Wikepedia, "Network behavior anomaly detection," [Online]. Available: https://en.wikipedia.org/wiki/Network_behavior_anomaly_detection.

[6] Alper Kerman, "Zero Trust Cybersecurity: 'Never Trust, Always Verify'," NIST, 28 October 2020. [Online]. Available: https://www.nist.gov/blogs/taking-measure/zero-trust-cybersecurity-never-trust-always-verify.

[7] X. Wang and R. Karri, "NumChecker: Detecting kernel control-flow modifying rootkits by using hardware performance counters," in Proceedings of the 50th Annual Design Automation Conference, DAC 2013, Austin, TX, 2013.

[8] A. Tang, S. Sethumadhavan and S. Stolfo, "Unsupervised detection of anomalous processes using hardware features," in RAID 2014: Recent Advances in Intrusion Detection, 2014.

[9] Z. Pan, J. Sheldon, C. Sudusinghe, S. Charles and P. Mishra, "Hardware-Assisted Malware Detection using Machine Learning," in 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021.

[10] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh and D. Ponomarev, "Hardware-Based Malware Detection Using Low-Level Architectural Features," EEE Transactions on Computers, vol. 65, no. 11, pp. 3332-3344, 2016.

[11] Intel, "Intel Threat Detection Technology," Intel, [Online]. Available: https://www.intel.com/content/www/us/en/architecture-and-technology/threat-detection-technology-brief.html.

[12] Microsoft, "Defending against cryptojacking with Microsoft Defender for Endpoint and Intel TDT," Microsoft, 26 April 2021. [Online]. Available: https://www.microsoft.com/security/blog/2021/04/26/defending-against-cryptojacking-with-microsoft-defender-for-endpoint-and-intel-tdt/.

[13] OWASP, "Code Injection," owasp.org, [Online]. Available: https://owasp.org/www-community/attacks/Code_Injection.

[14] T. Bletsch, Code-Reuse Attacks: New Frontiers and Defenses, Ph.D. Thesis, North Carolina State University, 2011.

[15] E. Buchanan, R. Roemer and S. Savage, "Return-Oriented Programming: Exploits Without Code Injection," in Black Hat USA 2008 Briefings, 2008.

[16] H. Hu, S. Shinde, S. Adrian, Z. L. Chua, P. Saxena and Z. Liang, "Data-Oriented Programming: On the Expressiveness of Non-control Data Attacks," in 2016 IEEE Symposium on Security and Privacy (SP), 2016.

[17] F. Schuster, T. Tendyck, C. Liebchen, L. Davi, A.-R. Sadeghi and T. Holz, "Counterfeit Object-oriented Programming: On the Difficulty of Preventing Code Reuse Attacks in C++ Applications," in 2015 IEEE Symposium on Security and Privacy, 2015.

[18] CISA; NIST, "DEFENDING AGAINST SOFTWARE SUPPLY CHAIN ATTACKS," Cybersecurity and Infrastructure Security Agency, 2021.

[19] Fireeye, "Highly Evasive Attacker Leverages SolarWinds Supply Chain to Compromise Multiple Global Victims With SUNBURST Backdoor," Fireeye, 13 December 2020. [Online]. Available: https://www.mandiant.com/resources/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.

[20] P. Paganini, "An in-depth analysis of the Kaseya ransomware attack: here's what you need to know," cybernews.com, 19 July 2021. [Online]. Available: https://cybernews.com/security/kaseya-ransomware-attack-heres-what-you-need-to-know/.

[21] Intel, Intel® 64 and IA-32 Architectures Software Developer Manuals, vol. 3A, Intel, 2021.

[22] "Control Flow Integrity," Clang, [Online]. Available: https://clang.llvm.org/docs/ControlFlowIntegrity.html.

[23] "Control Flow Guard," Microsoft, 7 January 2021. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/secbp/control-flow-guard.

[24] Patel, Baiju V., "A Technical Look at Intel's Control-flow Enforcement Technology," Intel, [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/technical-look-control-flow-enforcement-technology.html.

[25] X. Ge, W. Cui and T. Jaeger, "GRIFFIN: Guarding Control Flows Using Intel Processor Trace," in Proceedings of the 22nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2017.

[26] Y. Gu, Q. Zhao, Y. Zhang and Z. Lin, "PT-CFI: Transparent Backward-Edge Control Flow Violation Detection Using Intel Processor Trace," in CODASPY '17: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, 2017.

[27] Y. Liu, P. Shi, X. Wang, H. Chen, B. Zang and H. Guan, "Transparent and Efficient CFI Enforcement with Intel Processor Trace," in 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017.

[28] L. Chen, S. Sultana and R. Sahita, "HeNet: A Deep Learning Approach on Intel® Processor Trace for Effective Exploit Detection," in 2018 IEEE Security and Privacy Workshops (SPW), 2018.

[29] C. Yagemann, S. Sultana, L. Chen and W. Lee, "Barnum: Detecting Document Malware via Control Flow Anomalies in Hardware Traces," in 22nd Information Security Conference (ISC'19), 2018.

[30] Timo Kreuzer; Yarden Shafir; Satoshi Tanda; Blair Foster, "CrowdStrike Strengthens Exploit Protection Using Intel CPU Telemetry," Crowdstrike, 28 December 2021. [Online]. Available: https://www.crowdstrike.com/blog/introducing-falcon-hardware-enhanced-exploit-detection/.

[31] H. Hu, C. Qian, C. Yagemann, S. P. H. Chung, W. R. Harris†, Kim, Taesoo and W. Lee, "Enforcing Unique Code Target Property for Control-Flow Integrity," in CCS '18: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018.

[32] "libipt," [Online]. Available: https://github.com/intel/libipt.

[33] "libxdc," [Online]. Available: https://github.com/nyx-fuzz/libxdc.

[34] "Honeybee," [Online]. Available: https://github.com/trailofbits/Honeybee.

[35] "Ghidra," [Online]. Available: https://ghidra-sre.org/.

[36] Mitre, "Process Injection, Technique T1005," MITRE, [Online]. Available: https://attack.mitre.org/techniques/T1055/.

[37] MITRE, "Hijack Execution Flow: DLL Side-Loading," MITRE, [Online]. Available: https://attack.mitre.org/techniques/T1574/002/.

[38] Atomic Red Team, "T1574.002 - DLL Side-Loading," [Online]. Available: https://github.com/redcanaryco/atomic-red-team/blob/master/atomics/T1574.002/T1574.002.md.

[39] "Atomic Red Team," atomicreadteam.io, [Online]. Available: https://github.com/redcanaryco/atomic-red-team.

[40] MITRE, "Supply Chain Compromise, Technique T1195," [Online]. Available: https://attack.mitre.org/techniques/T1195/.

[41] "Notepad++," [Online]. Available: https://notepad-plus-plus.org/.

[42] "Metasploit," Rapid7, [Online]. Available: https://www.metasploit.com/.

[43] Recorded Future, "Adversary Infrastructure Report 2020: A Defender's View," 2021.

[44] "Windows Print Spooler Remote Code Execution Vulnerability," Microsoft, 1 July 2021. [Online]. Available: https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-34527.

[45] cub0x0, "CVE-2021-1675 / CVE-2021-34527," [Online]. Available: https://github.com/cube0x0/CVE-2021-1675.

[46] Microsoft 365 Defender Threat Intelligence Team, "A closer look at Qakbot's latest building blocks (and how to knock them down)," Microsoft, 9 December 2021. [Online]. Available: https://www.microsoft.com/security/blog/2021/12/09/a-closer-look-at-qakbots-latest-building-blocks-and-how-to-knock-them-down/.

[47] Cobalt Strike, "Cobalt Strike," HelpSystems, [Online]. Available: https://www.cobaltstrike.com/.

[48] Raphael Mudge, "Cobalt Strike's Process Injection: The Details," HelpSystems, 21 August 2019. [Online]. Available: https://www.cobaltstrike.com/blog/cobalt-strikes-process-injection-the-details-cobalt-strike/.

[49] CIS, "TrickBot: Not Your Average Hat Trick – A Malware with Multiple Hats," Center for Internet Security, [Online]. Available: https://www.cisecurity.org/blog/trickbot-not-your-average-hat-trick-a-malware-with-multiple-hats/.




0222/DCC/MZ/PDF