Metrics Gathering for AI-based Trust Assessment

Metrics Gathering for AI-based Trust Assessment

Lorenzo Ferro*, Flavio Ciravegna†, Francesco Zaritto‡, Antonio Lioy§, Luis Landeiro Ribeiro¶
* *Politecnico di Torino*, *Dip. di Automatica e Informatica*, Torino (Italy)
Email: lorenzo.ferro@polito.it
† *Politecnico di Torino*, *Dip. di Automatica e Informatica*, Torino (Italy)
Email: flavio.ciravegna@polito.it
‡ *Politecnico di Torino*, *Dip. di Automatica e Informatica*, Torino (Italy)
Email: francesco.zaritto@polito.it
§ *Politecnico di Torino*, *Dip. di Automatica e Informatica*, Torino (Italy)
Email: antonio.lioy@polito.it
¶*PDMFC*, Lisbon (Portugal)
Email: luis.ribeiro@pdmfc.com

***Abstract*—Technological advances have increased the complexity and volume of system operations. As a result, today systems face broader attack surfaces due to the larger amount of code executed. Ensuring the reliability of applications and operations in such environments requires well-defined security criteria, but establishing them is a non-trivial task. Trusted Computing offers mechanisms to establish trust in a non-repudiable manner, yet it faces inefficiencies and scalability challenges when applied to complex scenarios. In fact, defining a flexible verification method for a complex system is a challenging task. This is especially true when the process focuses solely on verifying individual actions rather than on the overall system's behaviour. AI-based techniques offer a viable countermeasure, since modelling a system's behaviour provides an adaptable approach to determining reliability. An AI model can also analyse a larger amount of data. However, collecting, securely storing, and provisioning relevant information is still a required feature. This work proposes two approaches for the secure collection of system events. The objective is to facilitate the provision of an AI model while ensuring the integrity and authenticity of the data.**

***Index Terms*—Remote Attestation, Trusted Computing, IMA, AI, Trust Assessment, eBPF**

I. Introduction

With the rapid advancement of technology, modern computing nodes are becoming increasingly powerful, enabling the execution of multiple services concurrently on the same host. This capability enhances efficiency and optimises resource use, allowing for more streamlined and cost-effective operations. However, this also introduces new security challenges that must be addressed to ensure the integrity and reliability of these systems.

In today's interconnected world, computing nodes often handle security-critical operations, making it imperative to verify their integrity at runtime continuously. This verification process is crucial to ensure that nodes are not compromised, as any breach could have far-reaching consequences. Traditionally, these verifications are based on the security of tracking the programs executed on the system and allowing an external entity to verify them. The verification process includes comparing the data collected with the expected state of the system, which helps identify any discrepancies or potential security threats.

However, predicting the exact system behaviour becomes increasingly complex in large-scale systems where multiple services run simultaneously. The dynamic nature of such systems makes it challenging to anticipate all possible interactions and dependencies between different services. As a result, conventional verification methods tend to focus only on a subset of system operations, leaving some windows open for undetected attacks. Furthermore, the validation of individual programs may not be sufficient to assess the system's overall security level comprehensively. This limitation highlights the need for more advanced and holistic approaches to system verification.

To address these challenges, gathering all events in a system and analysing them to determine whether they are linked to a compromised host is necessary. With the rise of AI technologies, we can now leverage them to define and assess the system's behaviour from a security-oriented perspective. AI-based behavioural analysis can process more system data and does not require a strict prediction of the system status. The more data provided to the AI model, the more precise the evaluation will be. This approach allows for a more comprehensive and adaptive assessment of the security posture of the system.

Currently, there are no standard ways to collect and store general system information securely. Existing solutions lack flexibility and only provide information about a limited set of events. This limitation hinders the effectiveness of AI-based behavioural analysis, as it relies on comprehensive and accurate data to make informed decisions. For this reason, new solutions for collecting information from the system are required to fully leverage the potential of AI to enhance system

This work was partially supported by the project SERICS (PE00000014) under the NRRP MUR program, funded by the European Union - NextGenerationEU. This work was also supported by the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme with Grant Agreement No.101139198 (iTrust6G project). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or SNS-JU. Neither the European Union nor the granting authorities can be held responsible for them.

security.

In this paper, we propose two different approaches for system data collection. One approach is based on an extension of the standard methods used in the Linux kernel for the collection and secure storage of system events. This method leverages the robustness and reliability of the Linux kernel to ensure that the collected data is accurate and tamper-proof. The other solution is based on eBPF (Extended Berkeley Packet Filter) to trace the events related to system calls more flexibly. eBPF provides a powerful dynamic tracing and monitoring framework, allowing for more granular and adaptive data collection.

The data collected using these approaches is then processed by an AI-based Verifier, which considers different aspects of the system. The Verifier analyses the data to identify any anomalies or potential security threats, providing a more comprehensive assessment of the system's security posture. We compared the two ideas regarding overhead, flexibility, and security guarantees to evaluate their effectiveness in enhancing system security. This comparison highlights the strengths and weaknesses of each approach, providing valuable insights into the best practices for system data collection and analysis in modern computing environments.

The paper is organised as follows. Section II provides background concepts required for understanding the work. Section IV defines the motivations behind the issues addressed. Section V and Section VI describe the two proposed approaches for metrics gathering. Section VII presents a comparative analysis of the two approaches. Section VIII discusses the evaluation methodology and results. Finally, Section IX outlines potential directions for future research and the implications of the proposed solutions.

II. Background

Trusted Computing (TC) [1], [2] is a set of standardised security principles and technologies promoted by the Trusted Computing Group (TCG) [3]. The primary objective of these is to establish *trust* in computing systems by relying on the identification of their individual hardware and software components. Trust [4] in TCG terminology is defined as the expectation that a system will behave in a known and expected manner.

Identifying the target system's components is crucial for understanding its specific behaviour. Once these components are determined, verifying their authenticity and integrity becomes possible, ensuring that the system's expected behaviour corresponds to the detected behaviour. Behaviour validation is the process through which trust in the system is effectively established. It involves computing a *measurement* for each component, precisely the cryptographic hash of its content, and comparing the obtained result with the expected value to ensure integrity.

The reliability of this process is ensured by a foundational security component known as the Root of Trust (RoT) [4] (Fig. 1). The RoT is the essential component that forms the basis for the platform's trust establishment process. It

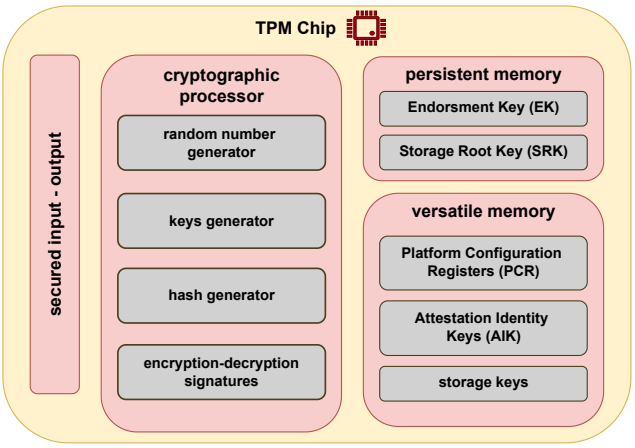

Fig. 1. Architecture of a TPM chip

provides security functions such as trusted boot, measurement, secure storage and reporting. The RoT is assumed to be inherently reliable, as any deviation in its behaviour would be undetectable. In more detail, the TCG guidelines define three distinct types of RoTs, each of which contributes to setting trust in the platform on which they operate:

- RoT for Storage (RTS): the component responsible for storing and protecting data from external access;
- RoT for Measurement (RTM): the component collecting and transmitting system measurements to the RTS;
- RoT for Reporting (RTR): the component ensuring trustworthy reporting of information held in the RTS.

RoTs are typically implemented in dedicated hardware components. A significant example of the latter is represented by the Trusted Platform Module (TPM) [4], a secure crypto-processor which implements both an RTS and an RTR. It is characterised by a set of protected locations known as Platform Configuration Registers (PCRs) while providing the necessary features to access the stored values. The TPM enables data to be stored within PCRs using the *extend* function. This operation involves concatenating the newly provided data with the current content of the selected PCR and then overwriting the PCR with the resulting hash of the combined input. This process efficiently optimises memory usage while sequentially concatenating each new data item to the previous ones:

$$\text{PCR}_{new} = hash(\text{PCR}_{old} \parallel measurement_{new})$$

A low-level, software-based mechanism can also represent a RoT. A relevant example is the Integrity Measurement Architecture (IMA) [5], [6], part of the Linux kernel's integrity subsystem. It is a RTM defining an Measurement List (ML) in which all integrity measurements of files accessed by the system are recorded sequentially, enabling effective operations tracing and integrity monitoring. IMA can work in conjunction with a TPM, using its PCRs to link these measurements and later securely retrieve them for verification.

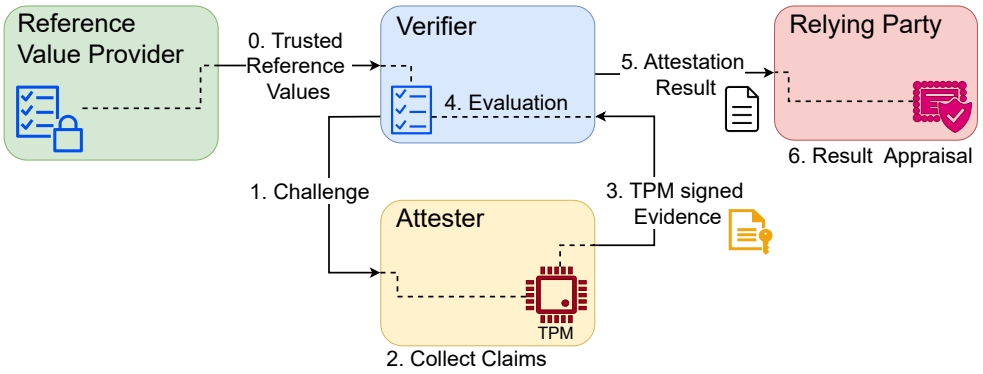

Fig. 2. RA: high level workflow leveraging a TPM as RoT.

Possessing a RoT enables the execution of a convenient method for periodically verifying the trustworthiness and integrity of a platform referred to as Remote Attestation (RA) [7]. RA (Fig.2) is a challenge-response protocol in which the target system (*Attester*) generates a cryptographic proof derived from its collected integrity measurements, accurately reflecting its current state and enabling a trusted remote entity (*Verifier*) to assess its trustworthiness. A third entity, known as the *Relying Party*, can then make proper decisions based on the results provided by the Verifier. The reliability of the RA process derives from the RoT generating and signing the proof using a key uniquely tied to it. This guarantees non-repudiation and securely links the proof to the platform on which it is created, ensuring its authenticity. The Verifier can confidently compare the received proof with trusted reference values representing the expected system state, confirming the platform's security. To standardise RA procedures, the IETF defined the Remote ATtestation procedureS (RATS) Architecture [8] as a dedicated framework for attestation protocols.

III. Related Works

A. Falco

Falco [9] is a real-time tool built to monitor your system and spot unusual activities or potential security threats. At its core, Falco works by tracking events, like system calls, happening in your operating system's kernel. It uses custom rules to identify suspicious behaviour. Falco can also gather additional information by connecting to container runtimes (like Docker) and Kubernetes, giving it a better understanding of what's happening within your containers or cloud environment. Falco consists of a userspace program that interacts with the tool. This configuration defines how it runs, a driver that sends kernel events, plugins for extending functionality, and Falcoctl for managing rules and plugins. The event collection is based on eBPF and a custom kernel module.

The events Falco detects are collected and sent to external systems like Security Information and Event Management (SIEM) platforms or data lakes, where they can be analysed further to detect or respond to security issues. This makes it an essential tool for securing your infrastructure and quickly detecting threats. It doesn't verify or protect the data or guarantee that the data is accurate or collected correctly.

B. Cilium

Cilium [10] is an open-source project that enhances networking, security, and observability for cloud-native environments, particularly Kubernetes clusters. Cilium is built on eBPF and offers high-performance networking, multi-cluster capabilities, advanced load balancing, encryption, and deep security and visibility controls. It decouples security from traditional IP-based models, using service identity and application-layer filtering for stronger isolation and dynamic policy enforcement, which is crucial for the dynamic nature of microservices.

Cilium's core components include the Cilium agent, operator, client tool, and CNI plugin. The agent configures networking, load balancing, policies, and monitoring on each cluster node. The CNI plugin interacts with Kubernetes to set up networking datapaths for pods, while the operator centrally manages cluster-wide tasks. Hubble, an observability platform built on Cilium, provides deep visibility into service communication, network health, and application performance, leveraging eBPF to offer dynamic, low-overhead insights. It answers complex questions about service dependencies, network failures, and security events, ensuring complete transparency across the system.

Cilium supports modern microservices architectures by offering flexible, identity-based security for containerised workloads. It replaces traditional firewalls with more granular controls, securing APIs (e.g., HTTP, gRPC) and service-to-service communication. Additionally, Cilium integrates load balancing, bandwidth management, and troubleshooting capabilities, including metrics collection via Prometheus and event monitoring. Its ability to scale across clusters with minimal infrastructure requirements makes it well-suited for both virtualised and cloud-native environments.

C. Tracee

Tracee [11] is a security and observability tool that leverages eBPF technology to monitor system and application behaviour.

It collects events that provide insights into system activities and security threats, helping you detect suspicious behaviour patterns. Tracee has two types of events: built-in events, which are predefined by Tracee, and custom events, which users can define. Built-in events include syscalls, network activity, security events, LSM (Linux Security Modules), container events, and miscellaneous activities.

Policies in Tracee allow users to specify which events to trace and which workloads they apply to. These policies can be shared across different environments and used in the Tracee CLI and Kubernetes installations. Because Tracee can generate thousands of logs daily, the tool allows you to customise how logs are displayed and stored through its configuration file, making it easier to filter and access the needed data.

D. Standard IMA

IMA [5], [6] is a security feature in the Linux kernel designed to ensure that files remain intact and haven't been tampered with, either by accident or maliciously. It creates a unique hash, or fingerprint, of a file whenever it's accessed. This hash is then compared to a previously stored "good" hash. The file might have been altered if the current hash doesn't match the original, the file might have been altered. The hashes are stored in kernel memory, making them harder for attackers to access or change. Additionally, IMA can use hardware components to securely store the hash values, providing an extra layer of protection against software-based attacks. The main goal of IMA is to maintain the integrity of files, enabling local and remote checks to verify that they haven't been modified. It allows users to customise how it operates through specific policies and works alongside other security features, such as controlling access to resources. IMA tracks more than just files—it also monitors system events like process creation or network activity. However, it mainly focuses on recording file integrity data and doesn't allow much formatting flexibility unless you use a custom kernel. The default structure for storing data in IMA is "ima-ng", which comprises the value of the hash of the file and the absolute path where it is saved.

E. AIDE

Advanced Intrusion Detection Environment (AIDE) [12] is a powerful, open-source tool designed to monitor the integrity of files and directories on a Linux system. It compares the current state of files to a database it maintains, detecting any unauthorised changes or modifications to essential system files. AIDE is secured by Security-Enhanced Linux (SELinux) [13], which adds an extra layer of protection to the system. SELinux enforces mandatory access control, defining strict rules for how different processes, users, and applications can interact with system resources. When a process like AIDE attempts to access a file, SELinux checks the permissions against its security policies. If it cannot determine whether the access is allowed, it consults its security database to verify the request. Based on this verification, SELinux either grants or denies the access request. Through this integration, AIDE benefits from SELinux's robust security framework, ensuring it operates in a secure environment with properly controlled access to system files. This solution is oriented at appraising the security in a system without allowing a remote entity to obtain proof of the system state.

IV. Motivations

The motivation for this work stems from the inherent inflexibility of existing integrity verification methods when applied to complex systems. Conventional validation methods rely on comparing measured values of individual system components against predefined, trusted reference values. This strategy is efficient and effective when applied to a well-defined set of components that undergo minimal structural or behavioural changes over time. Still, it does not scale well when the verification scope is extended to encompass complex systems. In such environments, the number of valid system states grows exponentially, making it impractical to define and maintain a comprehensive set of reference values. This leads to a lack of flexibility and contextual awareness in traditional RA, which ultimately limits its effectiveness for large-scale or rapidly evolving systems.

Several research works have suggested that Artificial Intelligence (AI)-based techniques [14]–[16] could address these challenges. The core idea is to shift from static checks toward a dynamic evaluation based on system behaviour. Proper models can be built to analyse how components interact and behave over time, instead of relying only on fixed reference values. Such models can identify deviations that traditional methods might miss, making them a more robust solution in complex environments. Once trained, they can be deployed in AI-driven verifiers capable of offering a broader, more accurate view of system integrity. However, it is important to emphasise that this paper does not aim to develop an AI-based verifier. Instead, our contribution lies in providing support for such approaches. The aim is the provisioning of rich, context-aware data necessary for training and operating AI models in the RA domain.

In order to achieve this, it is necessary to rethink how system events are captured. Current event logging mechanisms in standard RA frameworks are typically minimalistic, designed to record only essential information for strict checks. Specifically, they usually target predictable system operations. While this approach preserves system resources, it lacks the depth and contextual information needed for integrity assessment in complex and dynamic scenarios. Conversely, an AI-driven model can benefit from more detailed data to obtain higher accuracy and scalability.

To address this gap, this work proposes two distinct approaches for intercepting events relevant to reliability and integrity under Linux systems. The objective is to monitor these events, extract their key properties and characteristics relevant to the analysis, and systematically record them to facilitate their integration into the target trust and security model (Fig. 3). It is important to state that the goal of this work is not to substitute traditional RA techniques but to enhance them through AI-based behavioural analysis.

V. Approach I

Our first idea for gathering information regarding system events is an approach similar to the IMA one and is depicted in Fig. 4. We introduce a new set of kernel-level hooks for different *syscalls* [17] to catch new system events, observe them, and ultimately enable the resumption of their execution. Those hooks allow the log of syscalls with their parameters to give more contextual information. Each event values are recorded into a list formatted as follows:

```
[PCR] [HASH] [ID] [param] [context_info]
```

Where `[PCR]` indicates the PCR extended by this event. The `[HASH]` field is calculated over the following fields (id, parameters, and contextual info). The hash value is linked with the TPM, extending the target PCR, and is used to verify the list's integrity. `[ID]` is the identifier of the syscall needed to understand its type. The `[param]` field includes the syscall's parameters. The `[context_info]` attribute is optional and could be populated with details useful for the syscall evaluation. The event record is mediated through policies that allow specification of the events we are interested in tracing. For example, it is possible to specify the recording of some open sockets to trace the system's open connections. Events are recorded in the kernel's memory, making them more secure than stored on the user side. Users can access this list through a virtual filesystem [18], which provides a bridge to view kernel data.

Those events are recorded in a separate list, and a different PCR is extended compared to IMA events. This allows us to not interfere with the standard IMA behaviour but gather more information that is used in addition to the IMA ones. Due to the nature of the linkage with the TPM, it is necessary to have the entire list to verify its integrity. However, storing the list in kernel memory for the entire system execution could overload the kernel memory. The kernel memory is limited, and it is not a good practice to store kernel data at the user level [19]. Still, the saturation of kernel memory will lead to kernel panic and system failure. IMA address this

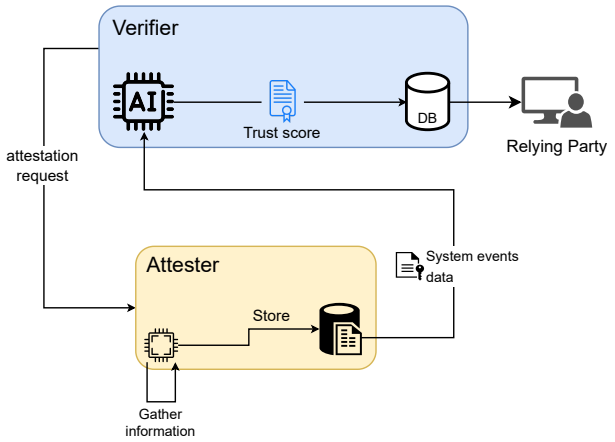

Fig. 3. Proposed RA Architecture.

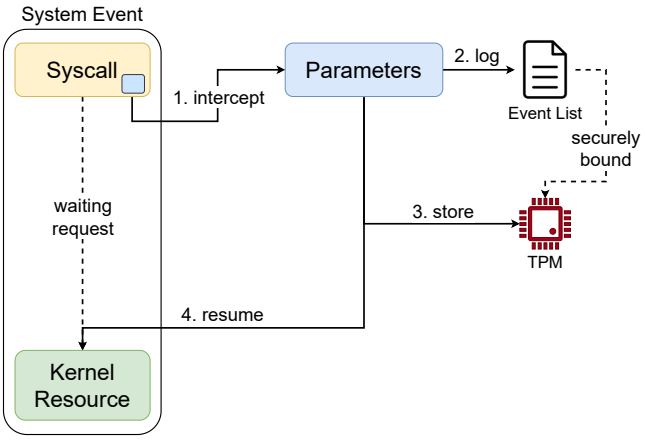

Fig. 4. System event detection: high-level workflow.

issue by measuring events regarding only newly accessed or modified files. In this way, re-accessing a file does not result in its reinsertion into the list, limiting its dimension. While applicable to a scenario regarding just files, recording data for all syscalls is unfeasible because each call is a different event. For this reason, to eliminate the list dimension, we introduce the concept of *list aggregate*, illustrated in Fig. 5. The list aggregate is the incremental hash resulting from iterating the PCR extension over all measurements belonging to the current Event List. This value is stored at the top of the list as it contains an aggregate of removed values that have already been verified in previous attestation rounds. It serves as the foundation for a new list version, reflecting the previous list state. Using this value, the list's integrity can be efficiently reconstructed without requiring storage of the entire list in memory. Since RA is a periodic operation, we need to verify just the new entries at each iteration, allowing the deletion of already verified ones.

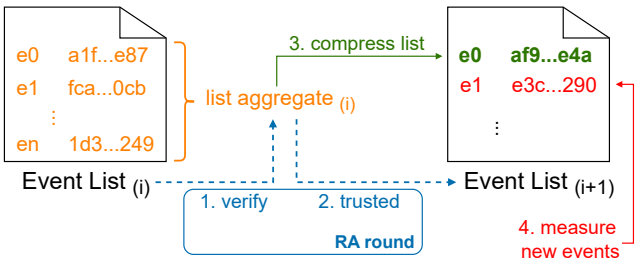

Fig. 5. Event List compression: high-level workflow.

This approach allows for the gathering of system information and securely storing it. When transmitting these data to the AI-based Verifier, it can first validate their integrity and origin using the information provided with a TPM Quote before proceeding with their analysis. This offers additional guarantees and provides a more reliable assessment.

By directly accessing the kernel, exact measurement policies can be defined, allowing for targeted assessment of specific

characteristics for each event of interest. This approach's efficiency and strong security come at the expense of flexibility. Since it relies on a custom kernel version, distributing and deploying the proposed solution as a ready-to-use service is non-trivial.

VI. Approach II

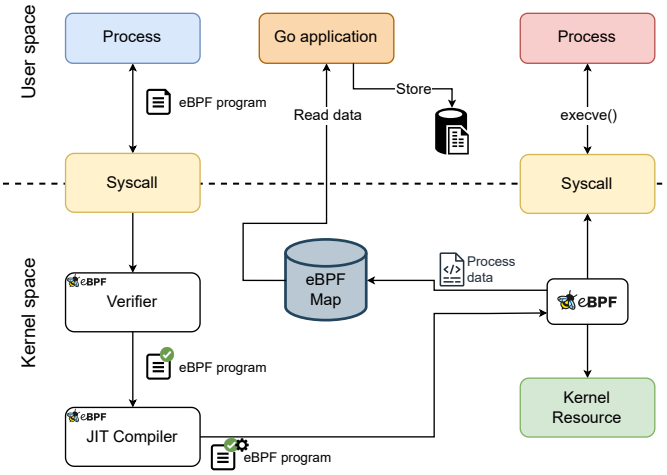

Fig. 6. eBPF overview.

The second approach we propose for collecting system events follows a similar procedure but is implemented using a different technique. It primarily relies on extended Berkeley Packet Filter (eBPF) [20], a technology that enables the safe and efficient extension of kernel capabilities without modifying its source code or loading additional modules. It provides a sandboxed environment for executing code that monitors and interacts with the kernel without compromising system stability or security. eBPF operates by attaching programs to predefined hooks within the kernel, which are invoked in response to specific events. These hooks provide a flexible mechanism for capturing real-time data, making eBPF a valuable tool for monitoring integrity-sensitive operations. As an example, eBPF can be used to intercept system calls, capturing both the parameters passed to these calls and the metadata associated with the invoking process (Fig. 6). In this way, it is possible to efficiently extract the data and compute measurements, which are then structured and aggregated in user space before being supplied to a Verifier.

The general workflow of the eBPF-based system event collection proceeds as follows. When a specific system call (such as *execve*) is invoked by a process running in user space, the kernel triggers the corresponding eBPF program. The eBPF program then executes in kernel space, collecting process-related information regarding the system behaviour, such as `Process ID (PID)`, `User ID (UID)`, process start time, and `CPU ID`. Once the data is collected, the eBPF program stores it in a high-performance ring buffer. The ring buffer, which is optimised for low-latency data transfer, acts as temporary storage that allows efficient communication between the kernel and user space. In this way, collecting data with minimal impact on system performance is possible. Furthermore, this overcomes the issues related to the saturation of kernel memory associated with the IMA-based approach (Sec. V). At this point, a userspace application retrieves the event data from the buffer. Afterwards, it processes the information by aggregating the data into batches for efficient handling. The data is signed before being reported by the system, but without strong hardware guarantees. This solution records and handles events on the user side, allowing storage in a flexible format.

VII. Approach Comparision

The workflow for collecting system information based on eBPF is notable for its high flexibility and expressive capabilities. This method is particularly advantageous in modern computing environments where dynamic and detailed system monitoring is crucial. eBPF allows for the precise definition of the data to be extracted through hooks, which are mechanisms that intercept function calls or system events. This capability enables interaction with the kernel without necessitating any modifications to it, thus maintaining system stability and integrity.

One of the key benefits of this eBPF-based approach is that it processes the resulting list of measurements in user space rather than kernel space. User space refers to the part of the system where application software runs, as opposed to kernel space, where the core operating system functions are executed. By handling measurements in user space, this method mitigates the risk of memory saturation in kernel space, an issue when using kernel modules like IMA. Memory saturation in kernel space can lead to system crashes or performance degradation, making the eBPF approach a more reliable option for extensive data collection tasks.

However, this flexibility comes with certain security trade-offs that need to be carefully considered. Managing measurements in user space exposes them to potential security vulnerabilities related to access control and authenticity verification. In the user space, data is more accessible to various processes and applications, increasing the risk of unauthorised access or tampering. Unlike the approach described in Section V, which might involve more secure and isolated methods, the measurements obtained through eBPF are not immediately secured in an RTS. This implies that the integrity guarantees about those values are lower, and it should be considered during an AI-based evaluation.

Even if an RTS is available for securing the measurements, the process of transferring data from kernel space to user space introduces a security vulnerability. This transfer creates a window of opportunity where the legitimacy of the data being recorded cannot be guaranteed. During this transfer, there is a risk of feeding false measurements into the RTS. These false measurements, despite being invalid, could be accepted

as legitimate and attested by the RTS itself, leading to potential security breaches.

Despite these security concerns, the flexibility of eBPF allows for the collection of data regarding various events and more contextual information. This capability is particularly useful in complex systems where understanding the context of events is crucial for accurate monitoring and analysis. An AI model can quickly retrieve and verify the list of measurements, making it possible to detect the absence of data or any modifications. The comprehensive nature of the collected data, combined with the lack of knowledge about the collection policies or the structure of the AI model, makes it challenging to perform undetected modifications. However, it is not entirely impossible, and thus, an additional integrity guarantee is required, particularly for more security-relevant events.

A potential solution to address these security concerns could be a hybrid approach that combines the strengths of both methods. This hybrid approach would involve using TPM to ensure the integrity of more critical events, leveraging its hardware-based security features to provide a high level of trust. Simultaneously, the contextual information provided by eBPF can be used for other data, offering flexibility and detailed insights. By feeding the combination of these two types of events into an AI model, a complete view of the system's behaviour can be obtained.

This comprehensive view enables a thorough analysis, ensuring both flexibility and security. This hybrid approach provides a robust framework for system information collection and analysis, making it suitable for modern computing environments where both performance and security are paramount.

VIII. Tests and Evaluation

We performed a comparison between the overhead added by the two approaches for data collection. The tests were performed on a physical node running Ubuntu 24.04 LTS, equipped with an Intel i5-5300 processor clocked at 2.30 GHz, four cores, eight threads, 16 GB of RAM, and a TPM 2.0 chip. The kernel version is 6.8, with native eBPF support and our patch to support the syscall record.

In Fig. 7, we assess the performance implications of our monitoring solutions by evaluating the overhead introduced to the *execve* system call [21]. The execve syscall executes a program specified by a given pathname, replacing the current process image with a new process image. This evaluation is conducted using a C program specifically designed to continuously invoke the *execve* syscall, creating a more demanding scenario than typical real-world usage. This stress-testing approach ensures that our measurements account for a higher frequency of syscall triggers, providing a robust evaluation of performance under extreme conditions.

Initially, we established a baseline by measuring the average execution time of the *execve* syscall without any monitoring mechanisms active. Following this, we conducted the same experiment with our two proposed monitoring solutions: one leveraging IMA and the other utilising eBPF.

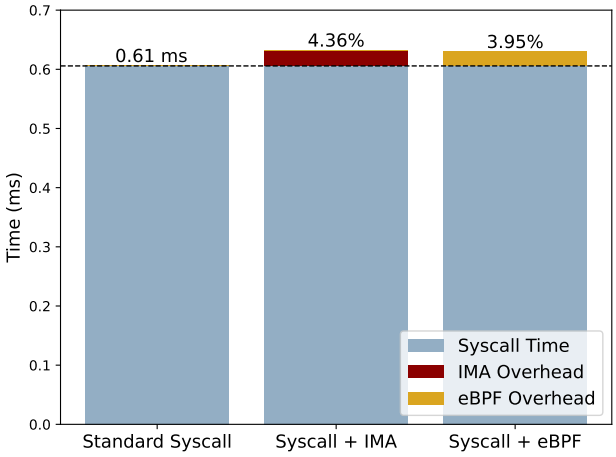

Fig. 7. Solutions overhead comparison.

Our findings indicate that both monitoring solutions introduce similar and acceptable overheads relative to the syscall execution time, each accounting for approximately 4% of the total syscall duration. The IMA-based solution incurs additional latency primarily due to its interaction with the TPM, which is responsible for securely storing and extending integrity measurements. This process, while secure, adds a layer of complexity that slightly increases execution time. On the other hand, the eBPF-based solution, which does not involve TPM interactions, experiences overhead due to additional processing requirements and context switches between kernel and user space. Despite these differences, both solutions maintain a low performance impact, which is crucial for their intended frequent and minimally disruptive operation.

A notable consideration for the IMA-based approach is the potential challenge of managing the storage of recorded values within the kernel list, which could become saturated over time. However, our testing reveals that this issue arises only after prolonged periods, even on a standard desktop computer. In server environments, which typically have more kernel memory available, the likelihood of encountering this problem is significantly reduced. This underscores the robustness of both solutions, particularly in environments where system calls are part of more complex and time-consuming workflows, further diminishing the relative impact of the observed overheads.

In summary, our results demonstrate that both the IMA-based and eBPF-based monitoring solutions are effective and efficient, suitable for deployment even in scenarios involving short-lived operations. Their minimal performance impact ensures that they can be integrated into systems with high-frequency syscall usage without significant disruption.

IX. Conclusion

This work explored two approaches for securely gathering system metrics to facilitate AI-based trust assessment. The first approach integrates kernel-level hooks and TPM interactions to ensure the integrity and authenticity of collected events. The

second approach leverages eBPF, offering a more flexible and lightweight solution without modifying the kernel source code.

Through experimental evaluation, we demonstrated that both approaches introduce acceptable overhead while providing robust security guarantees. The TPM-based approach ensures strong cryptographic integrity verification but requires kernel modifications, making it less flexible. Thus, this approach is well-suited for scenarios where strict integrity guarantees are necessary. In contrast, the eBPF-based approach provides a more deployable and adaptable solution at the cost of reduced direct linkage to the hardware trust anchor. This makes it ideal for environments requiring rapid deployment and minimal system modifications, such as cloud-based environments.

Future work will focus on refining these approaches by improving performance, extending the types of captured events, and integrating advanced AI models for dynamic trust assessment. In particular, a promising research direction involves designing and evaluating specific AI-driven techniques for the Verifier. These models could leverage behavioural analysis, profiling, and anomaly detection to enhance the accuracy of trust assessments. Additionally, exploring hybrid methods that combine the strengths of both approaches could lead to more efficient and scalable trust verification mechanisms.

REFERENCES

[1] C. Mitchell, *Trusted computing*. IET, 2005, vol. 6.

[2] A. Lioy and G. Ramunno, "Trusted computing," in *Handbook of Information and Communication Security*. Springer, 2010, pp. 697–717.

[3] Trusted Computing Group (TCG), https://trustedcomputinggroup.org/.

[4] Trusted Computing Group (TCG), "Trusted Platform Module Library Part 1: Architecture," 2024, https://trustedcomputinggroup.org/wp-content/uploads/TPM-2.0-1.83-Part-1-Architecture.pdf.

[5] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and Implementation of a TCG-based Integrity Measurement Architecture," in *13th USENIX Security Symposium*, San Diego (CA, USA), August 13, 2004, pp. 223–238.

[6] Integrity Measurement Architecture (IMA), https://sourceforge.net/p/linux-ima/wiki/Home/.

[7] G. Coker, J. D. Guttman, P. A. Loscocco, A. L. Herzog, J. K. Millen, B. O'Hanlon, J. D. Ramsdell, A. Segall, J. Sheehy, and B. T. Sniffen, "Principles of remote attestation," *Int. J. Inf. Sec.*, vol. 10, no. 2, pp. 63–81, 2011.

[8] H. Birkholz, D. Thaler, M. Richardson, N. Smith, and W. Pan, "Remote ATtestation procedureS (RATS) Architecture," RFC-9334, Jan. 2023.

[9] Falcosecurity, "Falco Website," https://falco.org/.

[10] Isovalent, "Cilium Website," https://cilium.io/.

[11] Acquasecriyt Group, "Tracee Website," https://aquasecurity.github.io/tracee/latest/.

[12] Red Hat, "Advanced Intrusion Detection Environment (AIDE)," https://www.redhat.com/en/blog/linux-security-aide.

[13] ——, "Security-Enhanced Linux (SELinux)," https://www.redhat.com/it/topics/linux/what-is-selinux.

[14] S. K. Gallagher, A. Whisnant, A. D. Hristozov, and A. Vasudevan, "Reviewing the role of machine learning and artificial intelligence for remote attestation in 5G+ networks," in *Future Networks World Forum (FNWF)*, on-line, October 12-14, 2022, pp. 602–607.

[15] J. Guo, A. Marshall, and B. Zhou, "A Trust Management Framework for Detecting Malicious and Selfish Behaviour in Ad-Hoc Wireless Networks Using Fuzzy Sets and Grey Theory," in *IFIP Advances in Information and Communication Technology*, Copenhagen (Denmark), June 29 - July 1, 2011, pp. 277–289.

[16] F. Wang, C. Huang, J. Zhao, and C. Rong, "IDMTM: A Novel Intrusion Detection Mechanism Based on Trust Model for Ad Hoc Networks," in *Int. Conf. on Advanced Information Networking and Applications*, Kitakyushu (Japan), March 22-28, 2008, pp. 978–984.

[17] Linux Manual Page, "Syscalls," https://www.man7.org/linux/man-pages/man2/syscalls.2.html.

[18] Linux Foundation, "Virutal-Filesystems in the Linux Kernel," https://www.kernel.org/doc/html/latest/filesystems/vfs.html.

[19] K. Luke, "Interaction Between the User and Kernel Space in Linux," 2017.

[20] eBPF, https://ebpf.io/what-is-ebpf/.

[21] Linux Foundation, "Execve man page," https://www.man7.org/linux/man-pages/man2/execve.2.html.