Secure Edge Computing with Lightweight Control-Flow Property-based Attestation

Nikos Koutroumpouchos*, Christoforos Ntantogian*, Sofia-Anna Menesidou†, Kaitai Liang∓,
Panagiotis Gouvas†, Christos Xenakis*, Thanassis Giannetsos‡
* Department of Digital Systems, University of Piraeus, Greece
‡ Cyber Security, Department of Applied Mathematics and Computer Science, Technical University of Denmark
† Ubitech Ltd, Greece
∓ Center for Cyber Security, University of Surrey, UK
Email: {nikoskoutr@ssl-unipi.gr, dadoyan@unipi.gr, smenesidou@ubitech.eu, k.liang@surrey.ac.uk,
pgouvas@ubitech.eu, xenakis@di.uoa.gr, atgi@dtu.dk}

***Abstract*—The Internet of Things (IoT) is rapidly evolving, while introducing several new challenges regarding security, resilience and operational assurance. In the face of an increasing attack landscape, it is necessary to cater for the provision of efficient mechanisms to collectively verify software- and device-integrity in order to detect run-time software modifications that try to manipulate control-flow integrity. Towards this direction, remote attestation techniques have been proposed in the literature as a promising defense mechanism. However, this family of solutions do not capture the real-time requirements of industrial IoT applications and suffer from scalability and efficiency issues. In this paper, we present a lightweight dynamic control-flow property-based attestation architecture (CFPA) that can be applied on both resource-constraint edge and cloud devices and services. Our architecture is a first step towards a new line of attestation-based mechanisms that enables the provision of control-flow attestation of only these specific, critical software components that are comparatively small, simple and limited in function, thus, allowing for a much more efficient verication. Our goal is to enhance run-time software integrity and trustworthiness with a scalable and decentralized solution eliminating the need for federated infrastructure trust. Based on our findings, we posit open issues and challenges, and discuss possible ways to address them, so that security and privacy do not hinder the deployment of intelligent edge computing systems.**

***Index Terms*—Control-Flow Property-based Attestation; Software-based Attacks; Trusted Component**

I. Introduction

Six decades since the start of the computer revolution, four decades since the invention of the micro-processor, and two decades into the rise of modern Internet, all of the technology required to transform industries through software has finally matured and can be widely delivered at a global scale. No part of the industry is untouched by this transformation; be it automotive [1], [2], smart factories, smart grids [3] or healthcare [4]. And with the advent of Internet of Things (IoT), we have just begun reaping the benefits of this evolution that, however, also brings a number of new challenges (or rather makes old unsolved challenges urgent to be tackled with); with *security*, *resilience* and *operational assurance* being some of the major concerns at both logical extremes of a network, namely the edge and the cloud

Indeed, in the era where "*service is everything and everything is a service*", there is an emerging trend for intelligent edge computing and the cloud to operate in tandem so as to provide flexible design choices that best meet business and operational goals. While this enables the extension of cloud analytics to edge devices and the delivery of a number of specialized software-based services for better managing the generation, processing and exchange of vast amounts of safety-critical data, it also turns them into attractive cyber-attack targets [5]. Therefore, in the face of an increasing attack landscape (Figure 1), it is imperative to ensure their correct and safe operation because by their very nature these software-based components may not always be in trusted custody. This necessitates the provision of efficient mechanisms to collectively verify software- and device-integrity in order to detect software modifications that try to manipulate control-flow integrity [6].

Intensive efforts in academia, industry and standardization bodies have converged to the use of *remote attestation* [7] for verifying the integrity of a software-based service. Typically, it is realized as a challenge-response protocol allowing a trusted verifier to obtain an authentic, and timely report about the software state of a (potentially untrusted and infected) remote device - a prover. A key component in building such trusted computing systems is a highly secure anchor (either software- or hardware-based) that can serve as *root of trust* to provide cryptographic functions, measure and report the behaviour of the running software and/or the computing platform and store data securely. Prominent examples of such trust anchors are Trusted Execution Environments (TEEs like TrustZone) [8], Trusted Platform Modules (TPMs) [2] or more lightweight secure components that require minimal hardware features and assumptions (better suited to to operate in resource-constrained embedded devices) such as SMART, Intel's TrustLite [9] and ARM's TrustZone-M.

In this context, many approaches have been proposed in the literature for enabling devices and services to attest to their integrity securely; this is achieved through various paths that can be categorized in static and dynamic solutions. The static methodologies [10], [11] include functions that

measure the integrity of the device binary images during boot up towards verifying specific configuration properties. These might include low-level concrete properties about the entity's configuration such as the type of firmware running, the version of its configuration file or presence of specific hardware properties, ports and network interfaces, etc.

However, these types of solutions do not capture the real-time requirements of the envisioned industrial IIoT applications since they do not ensure the integrity of the software's execution during run-time and, therefore, cannot capture software attacks that target the program's control flow [12], [13]. These types of attacks are considered the most devastating since they try to exploit memory- [14] and data-related [15] vulnerabilities for altering the execution path of the underlying system processes; either by injecting new malicious code [14] or by dynamically generating malicious programs based on already existing benign code snippets [16]. As a consequence, they can bypass the security of static attestation techniques since the measurement of a binary can remain unchanged even though the software's behaviour has been altered.

Compounding this issue, more advanced dynamic *control-flow attestation* solutions have been proposed that can protect against run-time exploitation techniques [17], [18], [19], [20], [21], [22], [23] by steering away from static measurements and aim to check software behaviour; software that is running as expected by verifying the integrity of the entire control flow. While a number of research efforts have proven the security and trust guarantees provided by such approaches, there are still a number of challenges (Section II) to be conquered especially when it comes to the efficiency and scalability of these techniques that question whether they can be applied in the real-world resource-constrained edge devices.

Such limitations mainly stem from the fact that these types of operational assurance methods try to verify the integrity, during run-time, of the **entire (untrusted) code base** of commodity platforms and operating systems. Considering that competitive IIoT application markets will always produce innovative and *large* systems comprising diverse-origin software-based components, with uncertain security properties, the best one can hope for is that a sub-set of such loaded software functions can be efficiently protected (in near real-time) against sophisticated run-time exploitation attacks [24].

This exact goal sets the challenge ahead: *Can we identify adequate behavioural and execution properties that can capture the chains-of-trust, needed for the correct execution of a system, and that reflect the security- and safety-critical code widgets to be verified from the untrusted code of the commodity platform or the cloud service provider?* This will, in turn, enable the provision of control-flow attestation of only these specific, critical software components that are comparatively small, simple and limited in function, thus, allowing for a much more efficient verification and safe co-existence.

The answer to this question is not straightforward; it poses a series of new technical and research challenges. IIoT applications such as industrial control systems are increasingly looking for consolidation of several critical and non-critical functions into a single device to reduce equipment and factory footprint and consequently the overall cost. This compartmentalization can lead to the identification of the behavioural properties to be attested and the modelling of the critical software components. However, to be trustworthy, these software components must be verified, and to be verified (as aforementioned) they must be small and simple which might denote that a number of basic services should be given up that will greatly underline their usefulness and viability. This is a double-edged sword that calls for a more holistic treatment of such attestation services.

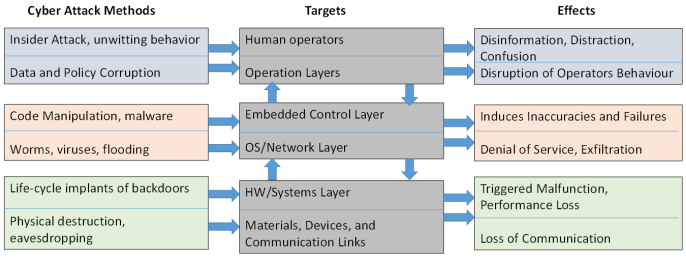

Fig. 1. Potential cyber-attack methods, targets and effects

Contributions: In this paper, we propose the use of *control-flow property-based attestation* (CFPA) to significantly enhance the state-of-the-art in software integrity and trustworthiness. CFPA represents the first step of a new line of research on control-flow attestation that is lightweight enough to run in the resource-constrained edge devices. Through enforceable security policies, we model the behavioural and execution properties (as a sequence of states) of only those safety-critical software functions that need to verified, during run-time, thus reducing the code base to be attested. Such behavioural properties mainly include execution paths to specific memory regions, as a result of the invocation of the functions of interest and their control-flow. As part of this novel decentralized approach, CFPA allows for the efficient monitoring of a program's control-flow through periodically computing authenticated and small control-flow footprints. Our proposed solution is scalable and decentralized, removing the need for federated trust of the infrastructure entities in cloud-based environments. This is clearly a viable approach towards remedying the limitations of existing attestation techniques, nonetheless, there is a need to still overcome a number of open issues towards a holistic end-to-end security approach.

II. Towards Decentralized Roots of Trust

With the advent of Internet of Things (IoT) in both critical and non-critical applications, security has become a major design issue for smart connected edge and cloud devices and services. Traditionally, they relied very often on isolation as only security mechanism and were often found to be completely unprotected. Furthermore, an emerging trend is the creation of even larger "Systems-of-Systems" (SoS) making sound security design more and more challenging and isolation as only security mechanism obsolete.

A key challenge to master this challenge is to establish and manage trust between entities, starting from bi-lateral interac-

tions between two single system components and continuing as such systems get connected to ever larger entities. But *how can we make sound statements on the software security properties of single systems and transfer this to statements on the security properties of such hierarchical compositions of systems?*

This need for the establishment of federated trust between services and devices, in a complex SoS, cannot be solely secured with common centralized solutions like PKI. Recent research [17], [18], [19] has demonstrated the need to move towards decentralized federated safety critical systems that aim to establish roots-of-trust in intelligent edge devices by leveraging software- or hardware-based security mechanisms (e.g., TPMs, TEEs, etc.). Particularly with respect to safety and operational assurance, software components must be enabled to prove statements about their state so that other components can align their actions appropriately and an overall system state can be assessed and verified. Towards this direction, as aforementioned, a number of static and dynamic (control-flow) attestation techniques have been proposed.

When it comes to checking and attesting to the integrity of the control-flow of each service, the overhead on both the device and the network might exceed the functional limits and become cumbersome to the system. One of the first solutions that demonstrated the feasibility of control-flow attestation was C-FLAT [17]. C-FLAT is an attestation scheme that measures the valid execution paths undertaken by embedded devices. However, it requires instrumentation of all control-flow instructions thereby violating legacy compliance. What is more, C-FLAT incurs performance overhead; thus it is suitable only for small size binaries. Subsequent works such as LO-FAT [18], LiteHAX [19], SCAPI [20], SCARR [21], hardware-based attestation [22] and CFIMon [23] aim to reduce this overhead in the edge devices by proposing attestation protocols that are either optimized (control-flow graph shortened) or utilize commonly installed hardware components (hardware accelerators) to ease the strain for these low-powered devices.

Moreover, there are also collective attestation protocols like SANA [25], SEDA [26], DARPA [27] that decongest the network by reducing the traffic introduced by the large number of edge devices trying to attest to their integrity. This is achieved by securely aggregating the attestations of multiple devices in a single cryptographic token through which the verifier will be able to confirm that each of the involved devices proved correctly that their execution flow is as expected.

Following this direction of improving the performance of device attestation, another line of research is on application partitioning. Approaches on this matter [28] try to dissect each application in coherent sections with each one having different security requirements. In these research works, the aim is to reduce and improve the precision of security measures that mitigate the possible vulnerabilities that each partition might have. A reminiscent of virtual partitioning is the estimation of the attack surface [29] of a binary. In our context, the application partitioning and the attack surface can be utilized to monitor and attest only the most critical -from a security point of view- paths of the binary (e.g., input data handling, etc) reducing in this way the size and complexity of the control-flow graph (CFG). The main drawback of these approaches is that, although it might improve the overall performance of the applied security, it lacks in providing security for future attacks (i.e., zero days). This is common among security measures that just mitigate known families of attacks, since they do not continuously check the program for possible divergences in the control flow. Instead, they just aim at actively mitigating any attack that they are designed to prevent.

A combination of these concepts is of great interest, the future-proofing of control-flow attestation together with the efficiency of application partitioning could lead to a solution that is both easily applicable and highly secure. This is perfectly aligned with the current trend of shifting cloud computing technologies and services into edge devices. The recent advancements on containerization software that allowed running containers (docker) on low resource devices , paves the way for a distributed software architecture in edge devices, where micro-services can be leveraged to streamline the service orchestration process and optimize the use of computing resources on the edge devices [30]. Micro-services introduce a way to deal with creating applications as an arrangement of small autonomous services, a completely different paradigm from monolithic architecture design. As a result, various parts of the same application can be independently implemented, making it conceivable to break down huge applications in small services with a particular target to achieve. As such, a system is required to verify that this particular fine grained, modular, target is indeed achieved.

All the aforementioned researches and observations, provide steps to the correct direction of tackling the performance and viability issues that the ever-increasing number of edge devices to be attested has introduced. As described previously, control-flow property-based attestation proposes another view in the matter of control-flow attestation for verifying the integrity of only those critical software functionalities (e.g., specific micro-services). The contributions of such a CFPA-based architecture are two-fold: first of all, we are using advanced tracing mechanisms for extracting the control-flow graph of a service (i.e., extended Berkeley Packet Filters (eBPFs)) that provides high performance with minimal overhead to the execution times of the device. Second, we redefine the attestation process by checking only the security-wise critical functions (instead of the whole application) of the attested device, resulting in a novel, scalable by design, lightweight, control flow attestation scheme as presented in the following sections.

III. Control-Flow Property-based Attestation

The control-flow attestation process includes the checking of each execution instruction of a security sensitive application. However, given the size of modern applications, the recording and transmission of every execution instruction is becoming infeasible, since this would: (1) result in a very long attestation response, and (2) require the verifying entity to walk through every single instruction/command. Hence we propose control-

flow property-based attestation that compounds these issues and allows for a faster verification process.

The insight behind this approach is that we do not need the attestation of the entire device but only the execution properties of the security sensitive functionalities that are running on the device. The identification of which functionalities should be monitored by the attestation mechanism is implementation dependent; for instance in the case of automatic collision avoidance of a vehicle, the brake Electronic Control Unit (ECU) and the radar sensors are potential targets that need to be checked for their proper functionality to be asserted that the system will work as intended. The aim of this procedure is to check both behavioural properties and low-level concrete properties about the entitys configuration and execution, such as the current firmware version it is running, the version of its configuration file or presence of certain hardware properties, integrity of sensor measurements, execution paths to specific memory regions, ports and network interfaces, etc. Furthermore, some of these properties might need to be attested individually while others might require to be approached as a system of systems that need to be attested by the involved devices as a group.

A. System Model

The system is composed of a network with heterogeneous interconnected edge devices. Each device is either a verifier (Vrf) or a prover (Prv) with the prover devices executing security sensitive operations which is the reason why their system state integrity must be verifiable. Vrf must be able to check if Prv is in a trusted state by checking its state against a good state that Vrf has stored. Both the provers and the verifiers in this system are equipped with a secure hardware device (TPM, TrustZone, SHE[1]) that will support the attestation process by improving its reliability. The attestation process allows each verifier to check whether each device in the network is in a trustworthy state. Should an entity fail to provide a proper attestation based on a set of installed security policies, it is immediately considered untrusted and it is instructed to collect further evidence that can be used to either prove its trustworthiness or to determine a route to meeting the failed security policy. The system can be composed of multiple verifiers and provers, but a typical setup contains only one verifier with a large number of provers.

B. Security Requirements and Threat Model

Trust Assumptions: A common requirement in attestation protocols is that each device in the system is equipped with hardware support for remote attestation [17], [18], [19]. This requirement is needed towards establishing a hardware-based root-of-trust on which the attestation process will measure the integrity of the device/service while also communicating the results of this measurement securely. Furthermore, we require that the device is resistant to non-invasive attacks (i.e, side-channel attacks, fault inducing attacks) while the system should be able to identify offline nodes that have been absent for a long time and could be victims of such exploitation attempts (micro-probing, reverse engineering) [31]. Finally, we require that there are common mechanisms (run-time error detection, ASLR, stack canaries, bound checking etc.) installed that will provide memory, type and control-flow safety which will rid the application of unnecessary known security issues and will let us focus on diminishing greater threats.

[1]Secure Hardware Extension

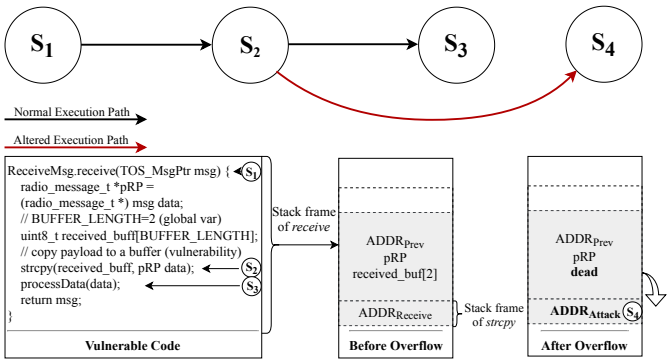

Fig. 2. Normal and Altered Control Flows

Security Requirements: Based on the aforementioned system model and trust assumptions, a CFPA-based architecture is capable of providing a broadened set of security and trust protection (besides the traditional data confidentiality, integrity and availability) under weakened assumptions. In particular, it can achieve:

R1. Memory Safety. All accesses performed by loaded processes/services in the underlying memory map of the host device are "*correct*" in the sense that they respect the: (i) logical separation of program and data memory spaces, (ii) array boundaries of any data structures (thus, not allowing software-based attacks exploiting possible buffer overflows), and (iii) don't access the memory region of another running process that they should not have access to.

R2. Control-flow Safety. All control transfers are envisioned by the allowed program. This translates to no arbitrary jumps in the code, no calls to random library routines, etc. This information is depicted by the allowed control-flow graphs (CFGs) that are calculated prior to the deployment of a service and are used as a baseline of the normal (trusted) sequence of execution states against which run-time control-flow footprints will be assessed.

R3. Type Safety. All function calls and operations have arguments of correct type, thus, protecting against data-oriented exploits [15].

Control-flow Properties & Policies: We need to identify a set of specific properties to be measured by the assisting hardware for the attestation process to be successful. We define a set of properties to be attested as the minimal required set of execution related attributes that will allow the verifier to confidently detect any control-flow deviations. These properties directly map to specific security-critical subsets of the application control-flow graph that need to be identified beforehand. They might include but not limited to: the current

firmware version it is running, the version of its configuration file or presence of certain hardware properties, integrity of sensor measurements, execution paths to specific memory regions, ports and network interfaces, etc. The aim of this approach is to greatly reduce the size of the attested binaries, thus, reducing the strain and the computational overhead on the edge devices, while sustaining a high level of assurance of trust for the attesting devices.

Such policies can then be modelled as control-flow policies that can be deployed to all devices, acting as verifiers, and will dictate the workflow of the overall CFPA (Section IV). Such policies are in essence regular expressions denoting sets of possible control traces; i.e., sets of normal executions CFGs.

DEFINITION 1. *Control-flow Correctness*: Service S respects a CF policy P if and only if when executed, all control transfers in S respect the already measured CFGs.

A running example: We consider a running example of smart connected vehicles which comprise an ecosystem of thousand ECUs. Each ECU must attest to each other in a web of interconnected nodes that propagate trust from edge nodes (usualy sensors) to more central management nodes. The example follows the functionality of collision avoidance in which the ECUs contained in the tires gather information from the tire sensors and reports them to the ECU that handles the breaks. The break ECU, depending on the status of the tire that is derived from the data sent by the tire ECU, will take action and apply the breaks accordingly. Let a code snippet that runs in the tire ECUs contain a buffer overflow vulnerability (Figure 2). In this example, the tire ECU is the Prv and the breaks ECU is the Vrf. The tire ECU must attest to its running integrity before it is trusted by the breaks ECU: if the properties received from the attestation are consistent with the S_1, S_2, S_3 control-flow state sequence, then the attestation process is successful and any data reported from the tire ECU is trusted. Any other set of properties that might result from any deviating control-flow like S_1, S_2, S_4 will be rejected and will flag the tire ECU as compromised and untrusted.

The properties should contain just enough details to figure any control-flow changes. For instance in Figure 2 we demonstrate a basic code injection scenario where the control-flow is altered. The normal execution flow should start from the function entry (S_1 / ReceiveMsg.receive()), continue to a memory sensitive function (S_2 / strcpy()) and finish with a data processing function (S_3 / processData()). In the scenario of code injection, the attacker will overflow the stack frame up to the point that she can overwrite the return address of the function (see Figure 2) with the address of S_4 ($ADDR_{Attack}$) which is the function that she will redirect the execution flow to. The system we propose, stores a healthy image of the control-flow in the form of its properties and afterwards, during the run-time of the application, it is able to detect any changes made from code injection attacks like the aforementioned one.

The code injection attacks belong in an attack category that directly manipulates control data (return address) that will ultimately change the control flow under the attackers

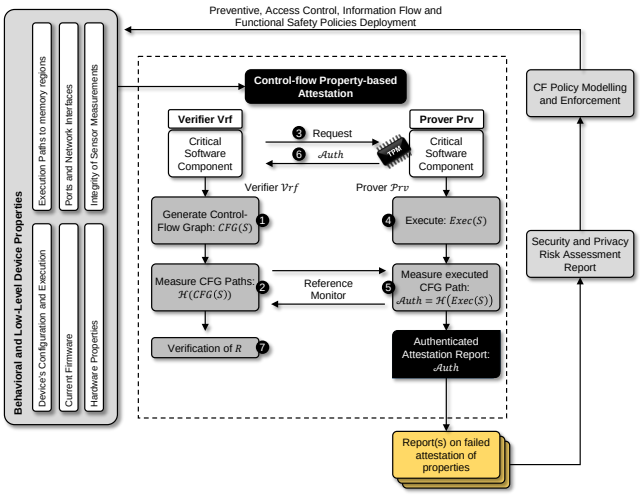

Fig. 3. CFPA Work Flow

will. In the same category, return oriented programming (RoP) aims to once again change control data, but this time instead of injecting code, the attacker tailors his execution target by chaining already installed code in a specific sequence. This category of attacks, can easily be detected by techniques such as control-flow integrity (CFI), code-pointer integrity (CPI)[17] and control-flow attestation which our solution is based on. However, CFI and CPI are not able to catch non-control-data attacks which instead of directly targeting control-data, they focus on corrupting data variables which are used to determine the control-flow that will be taken (e.g. variables in an if statement). This kind of attack is also detectable by our solution, since once again it indirectly manipulates the control-flow and leads to a set of control-flow properties that will fail to be attested. Finally, pure data-oriented programming (DOP) attacks aim to modify only memory load and store operations and do not affect the control-flow at all, hence they cannot be caught by control-flow attestation and they are beyond the scope of this paper.

IV. An Architectural Blueprint for Lightweight CF Property-based Attestation

A. High-Level Overview

As described in previous sections, CFPA relies on two core pillars: **remote attestation of specific properties** and **enforcement of dynamically adaptable policies**. With this, we claim that an SoS can withstand even a prolonged siege by a pre-determined attacker with known or unknown capabilities as the system can dynamically adapt to its security and safety state. This is substantially more flexible than traditional security mechanisms that often try to maintain and enforce a pre-defined set of policies using static attestation mechanisms.

Policy-based security management is an administrative approach for simplifying access control and security management of networks, services, etc. by establishing policies. Policies are sets of operating rules, usually in the form "*on event, if condition, then action*", that reflect the resource owners

intention of adequately protecting valuable resources. Policy-Based Access Control (PBAC) [32] is an access control model based on policy-based security management, which controls the access to resources by defining the rules and policies.

The CFPA-based architecture builds on concepts already existing in the PBAC area and particularizes them for complex deployments of software components. For instance, access to the basic services offered by these components will only be allowed when they can abide to the defined policies; attest to the integrity of the modelled properties (see the specific case study presented in Figure 2). Policy enforcement is managed through appropriate Policy Decision Points (PDPs) and Policy Enforcement Points (PEPs) [33].

It is imperative that we are able to express policies that: (*i*) when enforced, mitigate the risks of the safety and security critical systems we wish to compose, (*ii*) regard properties that can be attested to by the resource constrained embedded components of which systems are composed, (*iii*) safeguard the privacy of attesting devices by specifying the general principles for attestation data protection, and (*iv*) specify the type of evidence to be collected from a system, in case it fails to attest some of its properties, so as to perform a more in-depth investigation of the systems behaviour to detect if any type of malware is resident to the program and data memory (Section IV-B3). Defined policies must be expressive, deployable, and enforceable and may be dynamically updated if the attack graph is amended with new types of vulnerabilities.

After the correct definition of such policies, the system can proceed to periodically (or on-demand) attest the modelled behavioural and execution properties that mainly include execution paths to specific memory regions, as a result of the invocation of the functions of interest and their control-flow. All correct execution paths (i.e., CFGs) need to have been identified and securely deployed to the verifiers so as to act as the baseline of the normal sequence of states against which the run-time computed control-flow footprints will be assessed.

Following a similar work flow to the most prominent control-flow attestation architectures [17], [18], [19], Figure 3 presents the information flow of CFPA between a "prover" and a "verifier": In a nutshell, Vrf receives the already defined CF policies and computes all legitimate control-flow paths (CFGs) of the software component S of interest (Steps 1 and 2). To trigger the run-time CFPA, as dictated by an already defined security policy, Vrf sends a request to the device which acts as the Prv (Step 3). The Prv device executes these core functionalities that the Vrf desires to attest (Step 4) and the hardware root-of-trust measures the taken control-flow paths (Step 5). Such a measurement could be achieved through a simple hash function (Section IV-B2). Finally, the attestation result is send back to the Vrf for validation (Steps 6 and 7).

Since Prv attests the control-flow of the software component S, Ver can detect run-time attacks, as discussed in Section III. Any deviation from the program's legitimate CFG results in an unrecognized measurement. In order to support this periodic attestation process, an important characteristic of the depicted work flow is the capability of the CFPA to

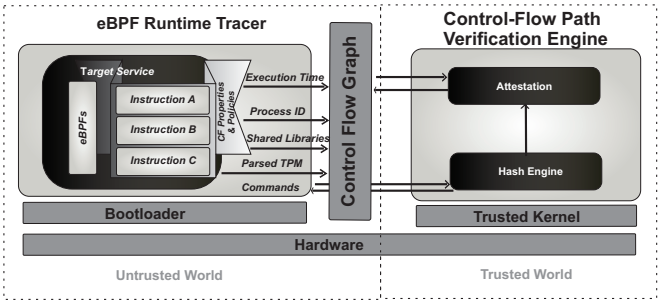

Fig. 4. CFPA Conceptual Architecture

observe the run-time execution of the process and collect and analyze low-level code information. This type of reference monitoring is based on the integration of advanced tracing techniques coupled together with the verification components for confining the execution path if the program is about to violate the deployed CF policy.

These attributes are depicted in Figure 4 that showcases the main components of the core attestation toolkit of CFPA: (a) an eBPF Runtime Tracer and (b) a Control-Flow Path Verification Engine to trace and measure the run-time CF path. The eBPF Runtime Tracer (eRT) is based on the innovative use of extended Berkeley Packet Filters (eBPFs) (Section IV-B1). In recent years, the eBPF subsystem has been added to the most Unix-based operating systems and, thus, does not require any additional kernel module while offering the capability of compiling and injecting new monitoring code dynamically during runtime. This characteristic makes eBPF a perfect candidate to trace the execution flow in the target environments. The Control-Flow Path Verification Engine (CFVE) (Section IV-B2) incorporates both: (a) a static binary analyzer that generates the programs Control Flow Graph (CFG) by identifying basic blocks and their connection, or (b) a dynamic analyzer that produces valid measurements for a set of inputs by tracing execution of the target program. One inherent assumption made is that the Execution Path Verification Engine needs to run in a trusted environment, otherwise an adversary will be able to manipulate the output of the control-flow path computation.

B. CFPA Protocols

1) Tracing: As aforementioned, such techniques are used to collect statistical information, performance analysis, dynamic kernel or application debug information and in general system audits. In dynamic tracing, this can take place without the need of recompilation or reboot. In the context of CFPA, a detailed dynamic tracing of the kernel shared libraries, low-level code, etc., and an in-depth investigation of the systems behaviour and execution flow will be performed to detect any cheating attempts or if any type of (non-previously identified) exploits are resident to the program and data memory. There are several open-source tracers that exist in the literature. Examples include the Unix-based *ftrace tool* that provides static and dynamic tracing. *SystemTap* tracing tool provides dynamic tracing through the use of Kprobes, Jprobes and Uprobes [34].

These have traditionally been TRAP-based tracing methods. However, it has been shown that leveraging such techniques consumes a significant amount of resources in the host device for large software runs, thus, making their integration in feasible for the resource-constrained edge devices envisioned in our scenario. Another example of dynamic tracing is DTrace but a visual survey of its code reveals that it offers very limited optimizations compared with the eBPF bytecode.

In CFPA, the goal is to leverage a general tracer that can be integrated in both edge and virtual (cloud-based) devices. For that reason, we prompt for investigating the novel use of eBPFs although their initial scope is mainly for packet monitoring. Overall, eBPFs are lightweight enough and can provide near real-time low-level code inspection, thus, capturing the requirements of IIoT applications. By incorporating eBPFs, we can also trace low-level system behaviour such as execution time, process name of invoked libraries, process id and time of internal operations. This feature is very important as it enables the in-depth investigation of the system's operation in case of an identified deviation from the normal CFG. The main idea is that the installed eBPF probes will be programmed to intercept internal operations towards producing a run-time control-flow path. eBPFs are used to capture the execution of specific software components in both physical and virtual devices so that we can check and attest the integrity of the execution behaviour based on already defined policies from embedded devices to cloud services.

2) Attestation Protocol: The internal control-flow attestation protocol information flow is presented in Figures 3 and 4. The CFVE measures the run-time control-flow path leveraging the output of the execution of the eRT tracer. This output will include information regarding the process execution time, the process name, the process id, the number of bytes read/write, the time of the operation to complete, the type of traced files (e.g. Directory, Symbolic Link, Character Device, Socket etc.) and the the parsed data from kernel. The main idea is that Vrf can check whether the reported state is trustworthy, i.e., whether only known and benign software is loaded on Prv. The measures of the CFVE engine are based on advanced crypto primitives such as run-time property-based attestation. As already pointed out, Vrf receives and stores the security CF policies and the control-flow graph paths of the core functions that need to be attested (Step 1). Each path is pre-measured separately and corresponds to a hash value $H(CFG(A))$ (Step 2) which is generated by a target collision resistant hash function taking $CFG(A)$ as input. Due to the collision resistant feature, any two different inputs, for the hash function, will lead to two distinct outputs. This indicates that any modification on the path flow will be easily identifying by comparing the hash value of the modified path with the one of the original path. The CFVE engine, which may be seen as a log engine, provides a chain of hash values that may reflect change/update on any input control-flow graph (presented by Prv). For the same Prv, from a series of generated hash values, a hash chain can be formulated to help Vrf maintain a history of the prover's statuses. This hash chain is a core aspect of CFPA since it provides additional levels of security guarantees: for instance, one may trace back to the prover's previous and even original status, or search specific status in the chain to match some given control-flow graph. Vrf challenges the prover with a specific flow path of a functionality (Step 3). At this stage Prv runs this functionality (Step 4) and the CFVE calculates the measurement of the selected control-flow path and responds to Vrf (Step 5). Here the prover must guarantee that the flow should avoid the possibility of MITM attacks. To do so, Prv may make use of cryptographic digital signatures (e.g., Schnorr signatures [35]). We have to make the inherent assumption that Prv has already had a public- and private-key pair and the public key is shared with Vrf. In this case, the prover just needs to sign the flow with its secret key and further deliver the flow and the corresponding signature to the verifier such that Vrf is then able to verify the integrity of the flow using the prover's public key. After checking the integrity, Vrf will check the validity of the response and then compare the new measurement with the pre-deployed one to identify any deviation from the trusted flow of the selected core function.

3) Trust Evidence Collection: Considering the complexity of the intelligent edge and cloud deployments where the deployed software may be updated to a more sophisticated version with new capabilities, the threat model has to also take into consideration "zero-day" attacks that were not evaluated during design-time through static analysis. In addition, a decision about a systems integrity might not be sufficient to understand the systems behaviour when the attestation output is negative. In this case, another stream of assurance functionality is needed which entails a more in-depth investigation for detecting any cheating attempts. This is the goal of the *trust evidence collection* phase: based on collected evidence regarding the behaviour of a system, run-time verification mechanisms are applied for monitoring and verifying both the execution behaviour of a single system as well as the communication patterns of a set of attesting systems against a set of specific requirements as identified in the already deployed policies. This evidence may include access patterns to different memory regions, libraries, ports and network interfaces, stack frames, etc. The service will collect this evidence over some period of time (in the case of a negative attestation outcome), according to specified policies, and will feed it to the run-time verification engine for further investigation.

C. Challenges

One of the main challenges in managing device and network security in todays heterogeneous and scalable infrastructures is the lack of adequate containment and sufficient trust when it comes to the behaviour of a remote system that generates and processes mission-critical and/or sensitive data. An inherent property is the codification of trust among computing entities that potentially are composed of heterogeneous hardware and software components, are geographically and physically widely separated, and are not centrally administered or controlled. By leveraging the artefacts of traditional secu-

rity infrastructure (such as digital signatures, certificates and assurance statements) coupled with advanced crypto primitives (such as run-time property based attestation) and building upon emerging trusted computing technologies and concepts, CFPA will convey trust evaluations and guarantees for each network entity. Another challenging task in control-flow attestation techniques is how to trace deviations from the valid execution path and how to trace code-reuse techniques such as return-oriented programming and data-oriented programming that do not divert the programs control-flow or modify its binaries. As already pointed out, a deeper level of investigation is needed and eBPFs can be used to monitor the low-level system behaviour apart from the control- flow on the fly without re-deployment. These advantages make the integration of such a tracing technique highly desirable and effective despite some known limitations such as limited program size and limited support for loops [36].

V. CFPA Road-Map: Open Issues Towards Edge Assurance Services

Secure Communication Channel. The CFPA high-level architecture, as depicted in Figure 4, denotes the need of continuous interactions between untrusted and trusted worlds of the host device. The communication mode requires us to build up a secure bridge to deliver messages between the eBPF runtime tracer and the verification engine. By secure we mean that (1) the source of a message can be identified (bidirectional identity authentication) and (2) the integrity and secrecy of a message can be protected from malicious network attackers. To achieve the first requirement, i.e. message source identification, we can leverage digital signatures. The digital signature scheme enables a message sender to digitally sign the outgoing message (using the secret key of the sender) so that the receiver is able to verify (via the public key of the sender) the "ownership" of the message it receives. Considering the run-time efficiency, we plan to investigate the use of Schnorr signatures [35], which are the most efficient and secure signature schemes in the literature; both for the tracer and verification engine. A bidirectional identity authentication is also of high importance for both Prv and Vrf (Figure 3) so that both of them can confirm that they "speak" to "the right entity" in each round of verification.

For securing a message in a communication channel, we may consider to either set up communications via private channel, for example, SSL/TSL, or encrypt messages through public channel (e.g., using RSA encryption [37] to safeguard device status). While message secrecy may not be needed, one may just deploy a signature scheme in the tracer and engine for identity authentication; otherwise, novel signcryption [38], which provide signature and encryption as-a-whole service simultaneously, may prove a desirable option.

In all cases, the local key management requirement is still apparent. This means that we have to make an inherent semi-trust assumption towards the tracer (located in the untrusted world) and the engine of being capable of securely storing their own keys. Recall that the engine is in the trusted kernel, s.t. we only need to consider the case for the tracer. It may be challenging to enforce each part of the run-time tracer to embed a secure key management tool. After a deeper investigation of the eBPF tracer, one can see that the most frequent message exchange is located at CFG. Therefore, one of our next actions will be to investigate how to embed lightweight key management into the CFG towards achieving enhanced identity authentication and data security.

Trusted Computing Base (TCB). The careful design and implementation of TCB components are of paramount importance to the overall security of the device, including the CFPA process. The components of the TCB are designed in such a way so that, when other parts of the system are exploited, they cannot allow any further type of misbehaviour. Ideally, a TCB should be as small as possible so that an exhaustive examination of its code base (by means of manual or computer-assisted software audit) becomes feasible; especially in the context of resource-constrained edge devices.

Trusted Codebase Trade-off. Another open problem stems from the question of how much trust can we actually put on the trusted kernel. In our design, the requirement of a trusted kernel is to provide a fully trusted execution environment for the verification engine. In practice, however, the trusted kernel might be intruded and manipulated by network attackers. In this case, we may need to enhance the trust models contributed by the kernel via an additional layer of attestation between the kernel and the engine. Apart from this single-sided trust, the verification engine should also be designed in a cost-effective way meaning that we should not prompt for a heavy load of functionalities to be executed within the isolated environment of the CFVE since this will compromise the overall performance and efficient. This sets the challenge ahead: what operational functions is it reasonable to place within the "trusted world" of a trusted component without compromising the overall performance? The same question can be reversed for the context of the "untrusted world" provided by a host: what types of services can be placed in this model without compromising the overall security and trustworthiness? These are interesting challenges for future work, where implementation and experimentation will be performed to evaluate the feasibility of the CFPA solution, and identify a trusted component capable of performing the required functionality.

VI. Conclusions

The current state-of-the-art shows a significant lack of adequate containment and trust when it comes to the behaviour of a remote system, while sophisticated software attacks that target the program's control flow bypass the security of static attestation techniques. In this paper we proposed CFPA as a first step of a new line of research on control-flow attestation: a lightweight dynamic control-flow attestation that can protect against run-time penetrations on resource-constrained edge devices. CFPA relies on both remote attestation of specific properties and the dynamic deployment and enforcement of security policies. The main contributions of such a novel

architecture are two-fold: a) the usage of eBPFs as a tracing mechanism for extracting the CFG and b) the examination and verification of only critical software components for scalability and performance reasons. Furthermore, by taking into consideration the salient characteristics of control-flow attestation along with the requirements of the involved stakeholders, we identified a number of open research challenges. It is our strong belief that if these challenges are tackled now while CFPA is still at an early stage, then, this emerging security mechanism can reach its full potential.

VII. Acknowledgment

This work was supported by the European Commission, under the ASTRID and FutureTPM projects; Grant Agreements no. 786922 and 779391, respectively.

References

[1] S. Gisdakis, M. Lagan, T. Giannetsos, and P. Papadimitratos, "Serosa : Service oriented security architecture for vehicular communications," in *2013 IEEE Vehicular Networking Conference (VNC), 2013 IEEE :*, ser. IEEE Vehicular Networking Conference, 2013, pp. 111–118.

[2] J. Whitefield, L. Chen, T. Giannetsos, S. Schneider, and H. Treharne, "Privacy-enhanced capabilities for vanets using direct anonymous attestation." in *IEEE Vehicular Networking Conference*, 2017, pp. 123–130.

[3] T. Dimitriou, T. Giannetsos, and L. Chen, "Rewards: Privacy-preserving rewarding and incentive schemes for the smart electricity grid and other loyalty systems," *Computer Communications*, vol. 137, pp. 1 – 14, 2019.

[4] T. Giannetsos, T. Dimitriou, and N. R. Prasad, "People-centric sensing in assistive healthcare: Privacy challenges and directions," *Security and Communication Networks*, vol. 4, no. 11, pp. 1295–1307, 2011.

[5] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial internet of things," in *Proceedings of the Annual Design Automation Conference*, 2015, pp. 54:1–54:6.

[6] B. Zeng, G. Tan, and G. Morrisett, "Combining control-flow integrity and static analysis for efficient and validated data sandboxing," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, 2011, pp. 29–40.

[7] F. Brasser, K. Rasmussen, A.-R. Sadeghi, and G. Tsudik, "Remote attestation for low-end embedded devices: the prover's perspective," in *Proceedings of the Design Automation Conference*, 2016.

[8] J. Winter, "Trusted computing building blocks for embedded linux-based arm trustzone platforms," in *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing*, ser. STC '08, 2008, pp. 21–30.

[9] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in *Proceedings of the EuroSys Conference*, 2014, pp. 10:1–10:14.

[10] Y. Li, J. M. McCune, and A. Perrig, "Viper: Verifying the integrity of peripherals' firmware," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11, 2011, pp. 3–16.

[11] A. Seshadri, M. Luk, and A. Perrig, "Sake: Software attestation for key establishment in sensor networks," *Ad Hoc Netw.*, vol. 9, no. 6, pp. 1059–1067, Aug. 2011.

[12] G. Thanassis, D. Tassos, and P. N. R., "Weaponizing wireless networks: An attack tool for launching attacks against sensor networks," in *Black Hat Europe 2010*, Barcelona, Spain, April 12-15, 2010.

[13] T. Giannetsos and T. Dimitriou, "Spy-sense: Spyware tool for executing stealthy exploits against sensor networks," in *Proceedings of the 2Nd ACM Workshop on Hot Topics on Wireless Network Security and Privacy*, ser. HotWiSec '13, 2013, pp. 7–12.

[14] T. Giannetsos, T. Dimitriou, I. Krontiris, and N. R. Prasad, "Arbitrary code injection through self-propagating worms in von neumann architecture devices," *Comput. J.*, vol. 53, no. 10, pp. 1576–1593, Dec. 2010.

[15] S. Chen, J. Xu, E. C. Sezer, P. Gauriar, and R. K. Iyer, "Non-control-data attacks are realistic threats," in *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, 2005, pp. 12–12.

[16] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-oriented programming: Systems, languages, and applications," *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 1, pp. 2:1–2:34, Mar. 2012.

[17] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-flat: Control-flow attestation for embedded systems software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 743–754.

[18] G. Dessouky, S. Zeitouni, T. Nyman, A. Paverd, L. Davi, P. Koeberl, N. Asokan, and A.-R. Sadeghi, "Lo-fat: Low-overhead control flow attestation in hardware," in *Proceedings of the 54th Annual Design Automation Conference*, 2017, pp. 24:1–24:6.

[19] G. Dessouky, T. Abera, A. Ibrahim, and A.-R. Sadeghi, "Litehax: Lightweight hardware-assisted attestation of program execution," in *Proceedings of the International Conference on Computer-Aided Design*, 2018, pp. 106:1–106:8.

[20] F. Kohnhäuser, N. Büscher, S. Gabmeyer, and S. Katzenbeisser, "Scapi: A scalable attestation protocol to detect software and physical attacks," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2017, pp. 75–86.

[21] F. Toffalini, A. Biondo, E. Losiouk, J. Zhou, and M. Conti, "Scarr: A novel scalable runtime remote attestation," 2018.

[22] T. Zhang, X. Zhuang, S. Pande, and W. Lee, "Anomalous path detection with hardware support," in *Proceedings of the Int. Conf. on Compilers, architectures and synthesis for embedded systems*, 2005.

[23] Y. Xia, Y. Liu, H. Chen, and B. Zang, "Cfimon: Detecting violation of control flow integrity using performance counters," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, 2012, pp. 1–12.

[24] Z. Zhou, M. Yu, and V. D. Gligor, "Dancing with giants: Wimpy kernels for on-demand isolated i/o," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, ser. SP '14, 2014, pp. 308–323.

[25] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A.-R. Sadeghi, and M. Schunter, "Sana: Secure and scalable aggregate network attestation," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 731–742.

[26] N. Asokan, F. Brasser, A. Ibrahim, A.-R. Sadeghi, M. Schunter, G. Tsudik, and C. Wachsmann, "Seda: Scalable embedded device attestation," in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 964–975.

[27] A. Ibrahim, A.-R. Sadeghi, G. Tsudik, and S. Zeitouni, "Darpa: Device attestation resilient to physical attacks," in *Proceedings of the 9th ACM Conference on Security Privacy in Wireless and Mobile Networks*, 2016, pp. 171–182.

[28] D. Geneiatakis, G. Portokalidis, V. P. Kemerlis, and A. D. Keromytis, "Adaptive defenses for commodity software through virtual application partitioning," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 133–144.

[29] C. Theisen, K. Herzig, P. Morrison, B. Murphy, and L. Williams, "Approximating attack surfaces with stack traces," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, 2015, pp. 199–208.

[30] M. Alam, J. Rufino, J. Ferreira, S. H. Ahmed, N. Shah, and Y. Chen, "Orchestration of microservices for iot using docker and edge computing," *IEEE Communications Magazine*, vol. 56, pp. 118–123, 2018.

[31] S. Skorobogatov, *Physical Attacks and Tamper Resistance*. Springer New York, 2012, pp. 143–173.

[32] R. Yavatkar, D. Pendarakis, and R. Guerin, "A framework for policy-based admission control," RFC 2753, 2000.

[33] C. Feltus, D. Khadraoui, and J. Aubert, "A security decision-reaction architecture for heterogeneous distributed network," in *2010 International Conference on Availability, Reliability and Security*, 2010, pp. 1–8.

[34] V. Prasad and J. Keniston, "Locating system problems using dynamic instrumentation," 2010.

[35] C. Schnorr, "Efficient identification and signatures for smart cards," in *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology*, 1989, pp. 239–252.

[36] S. Miano, M. Bertrone, F. Risso, M. Tumolo, and M. V. Bernal, "Creating complex network services with ebpf: Experience and lessons learned," in *Proceedings of IEEE High Performance Switching and Routing (HPSR18)*, June 2018.

[37] R. L. Rivest, A. Shamir, and L. M. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[38] Y. Zheng, "Digital signcryption or how to achieve cost(signature & encryption) << cost(signature) + cost(encryption)," in *Advances in Cryptology - CRYPTO '97, 17th Annual International Cryptology*, 1997, pp. 165–179.