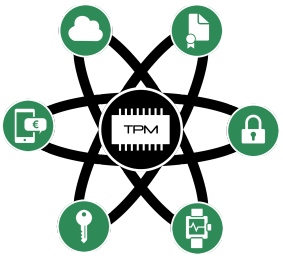


FutureTPM

D4.3

Runtime Risk Assessment, Resilience and Mitigation Planning First Release

Project number:	779391
Project acronym:	**FutureTPM**
Project title:	Future Proofing the Connected World: A Quantum-Resistant Trusted Platform Module
Project Start Date:	1st January, 2018
Duration:	36 months
Programme:	H2020-DS-LEIT-2017
Deliverable Type:	Other
Reference Number:	DS-LEIT-779391 / D4.3 / v0.1
Workpackage:	WP 4
Due Date:	30d June, 2019
Actual Submission Date:	27th August, 2019
Responsible Organisation:	DTU
Editor:	Thanassis Giannetsos
Dissemination Level:	PU
Revision:	v0.1
Abstract:	Deliverable D4.3 describes the complementary functionality of the risk assessment framework delivered in D4.2. More specifically, it describes the run-time mode of operation, and how it handles the cases of devices with “failed” attestation reports towards the re-calculation of the entire risk and threat vector taking into consideration the newly identified vulnerabilities. For the latter, a semi-automatic way for performing complementary system tracing is provided.
Keywords:	Run-time Risk Assessment, Resilience, Mitigation

The project FutureTPM has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 779391.

Editor

Thanassis Giannetsos (DTU)

Contributors (ordered according to beneficiary numbers)

Sofianna Menesidou, Panagiotis Gouvas (UBITECH)
Christos Xenakis, Christoforos Ntantogian, Eleni Veroni, Nikolaos Koutroumpouchos (UPRC)

Executive Summary

Risk management is a key aspect for the efficient operation of the entire ecosystem of assets/devices considered in the context of FutureTPM. As described in Deliverable D4.1 [8], risk analysis methods are used to evaluate the effectiveness of mitigation actions (modeled as control-flow attestation policies) that are associated with a given risk/incident. The implemented **OLISTIC-based FutureTPM Risk Assessment framework** [7] is tailored to the *security, trust and operational assurance* requirements of TPM-enabled devices and is capable of providing a **risk quantification methodology, during design-time,** which is model-driven.

Deliverable D4.3 provides the first release of the Run-time Risk Assessment, Resilience and Mitigation Planning as a complementary service of the overall FutureTPM RA. More specifically, it presents the functionalities that are executed during run-time, in the case of, **policy changing** facts that are likely to occur within the lifetime of the envisioned ecosystem of deployed devices. Such facts can be (indicatively) the disclosure of a zero-day vulnerability or a detected anomaly based on the output of the **Control-Flow Property-based Attestation (CFPA)** toolkit. If any of the enforced security policies fail, from malicious intent or faulty behaviour, the next logical step is to identify what was the cause of this event. This will enable better **situation awareness adaptation** for **re-calculating the overall risks and threats** of the entire ecosystem (considering the newly identied vulnerability) allowing **policy adjustments and the compilation of updated mitigation strategies and control-flow attestation policies**.

The Risk Quantification Engine is based on the OLISTIC platform and handles the unacceptable calculated risks by inferring (using **backward-chaining** techniques) the optimal defense strategies (i.e., properties that have to be re-actively attested) that have to be applied. The risk is calculated based on a number of input constraints such as the: a) **asset listing** of all infrastructure/edge devices and systems, b) **assets relationships** modeled as interdependecy graphs, c) **achievement of specific risk levels**, e.g., . we require the security policy that would minimize the risk of exploiting Asset*x* to 0.1, and d) preferred, high-level security, privacy and trust requirements. Based on the identified vulnerabilities and risk the Security Configuration Policies will be created and interpreted to both a) low-level security attestation policies and/or b) specific system calls to be monitored for providing guarantees against specific vulnerabilities.

During run-time, the low-level properties, that will be attested by the CFPA, and the **Attestation Report will contain the final verdict**. In case of failure, a more in-depth investigation of the systems behaviour is needed in order to to identify the exact attack vector and for the re-configuration of the security policies. Towards this direction, we propose a novel solution for **multi-level detailed tracing**, in order to upkeep the desired assurance and the required details for post-attack investigation. In a nutshell, the FutureTPM Run-time Risk Assessment framework will consider both a) the Backwards Chaining to resolve the constraints and b) the Cascading Effect Analysis in order to perform a re-calculation of the overall risk and threat vector.

The current implementation instance of the FutureTPM Run-time Risk Assessment framework consists of three main components. The **eBPF Runtime Tracer**, where we deploy the necessary eBPF execution hooks during design time, the **FutureTPM Secure Tracing Trust Evidence Collection**, which is responsible for providing the functionalities of multi-level detailed tracing enabling the collection of the necessary information/evidence, in the case of a failed attestation report, and the **Backward Chaining Process** towards the **re-calculation of the ecosystem's risk and threat vector**.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The main goal of this deliverable is to present the first release of the Runtime Risk Assessment, Resilience and Mitigation Planning in combination with the FutureTPM Risk Assessment Framework delivered in D4.2 [7]. The high-level overview is that the Runtime Risk Assessment takes as an input the initial risk and threat vector as identified and quantified, by the OLISTIC-based Risk Quantification Engine and checks whether it is lower than an acceptable threshold or not. The risk is calculated based on a number of input constraints such as the: a) **asset listing** of all infrastructure/edge devices and systems, b) **assets relationships** modeled as interdependecy graphs, c) **achievement of specific risk levels**, and d) preferred, high-level security, privacy and trust requirements.. If this risk is not acceptable, it is analyzed further offline in order to create and enforce a new configuration policy. By configuration policy we mean a high-level access control policy or new interpreted low-level execution properties that needs to be attested. During run time, the low-level properties will be attested by the CFPA and the Attestation Report will contain the final binary result. In addition, if there is a need to enforce a new policy, there is also a possibility to update the tracer with new information needed to be traced and, thus, to deploy new Extended Berkeley Packet Filter (eBPF) hooks.

1.1 Scope and Purpose

The main purpose of this deliverable is to document the complementary functionality of the Runtime Risk Assessment and to provide information on the current implementation status. The goal will be to identify the minimal number of properties need to be attested for achieving a high level of assurance (based on the produced quantifiable risks) while, at the same time, safeguarding the privacy of the attested device.

1.2 Relation to other WPs and Deliverables

In what follows, Figure 1.1 depicts the relationships of the deliverable to the other Work Packages (WPs). As already described, D4.3 is complementary to the Risk Assessment Framework described in Deliverable D4.2 [7].

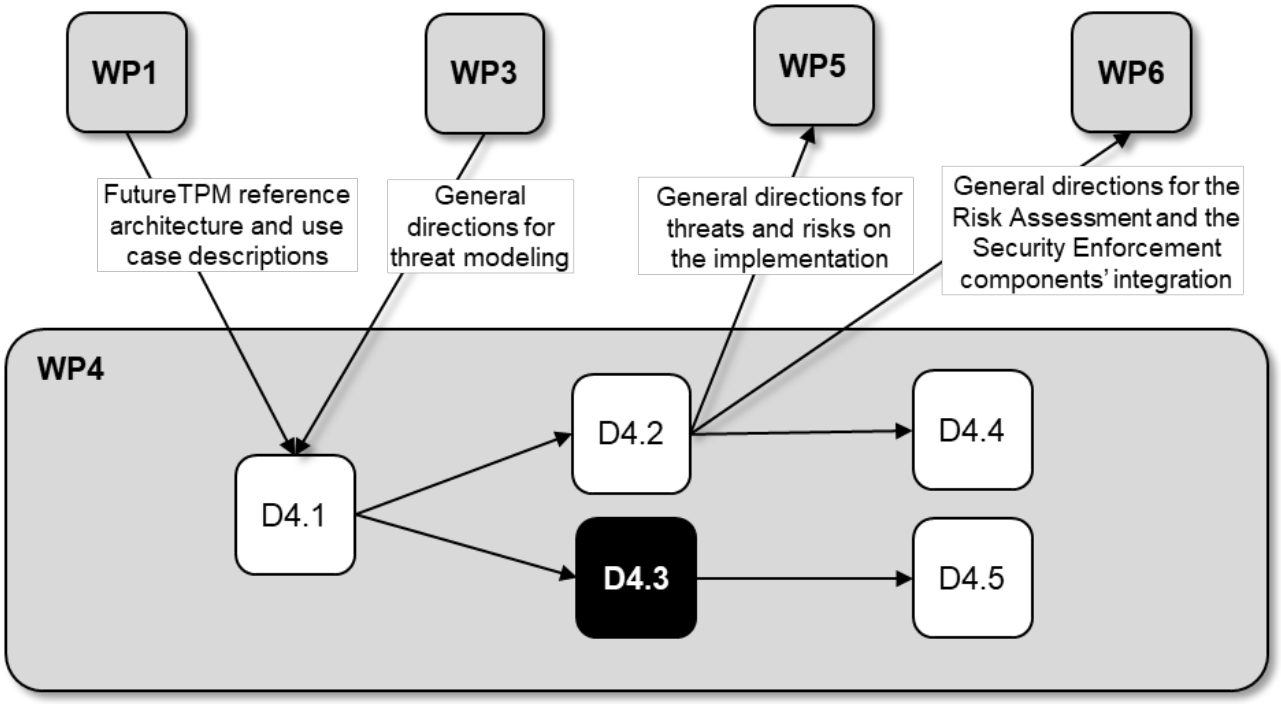


Figure 1.1: Deliverable D4.3 relationship within the FutureTPM project

1.3 Deliverable Structure

This deliverable is structured as follows. In Chapter 2, we describe the mode of operation of the run-time risk assessment including the design and implementation details of the core components. In Chapter 3, the process that takes place in the case of a "failed" attestation report towards a more in-depth investigation of the systems behaviour for post-attack identification. The provision of the semi-automatic way for performing complementary system tracing is described based on the deployment of enriched eBPF execution hooks. Chapter 4 outlines some of the mitigation strategies that are currently been investigate focusing on the memory and control safety of a device's execution. Finally, Chapter 5 concludes the deliverable.

Chapter 2

Run-time Risk Assessment

2.1 OLISTIC-based Run-time Re-Calculation of Risks

Risk management is a key aspect for the efficient operation of the entire ecosystem of assets/devices considered in the context of FutureTPM. As described in Deliverable D4.1 [8], risk analysis methods are used to evaluate the effectiveness of mitigation actions (modeled as control-flow attestation policies) that are associated with a given risk/incident. The implemented **OLISTIC-based FutureTPM Risk Assessment framework** [7] is tailored to the *security, trust and operational assurance* requirements of TPM-enabled devices and is capable of providing a **risk quantification methodology, during design-time,** which is model-driven. This risk quantification leverages the Drools Experts system in order to **assess the threat level of all listed assets**, including also their relationships and interdependecies, (**forward chaining mode**) and to **generate risk and threat impact models** based on threat level goals (**reverse chaining mode**).
Figure 2.1 showcases the high-level overview of the Risk Assessment execution and data flow, which is separated into two phases: **design- and run-time**. As aforementioned, the Risk Quantification Engine is based on the OLISTIC platform and will be used to calculate the overall risk graph of the FutureTPM ecosystem during **design-time**. This risk quantification, calculates the entire risk and threat vector considering **all possible attacks, threats and vulnerabilities**. Based on this output, security policies can be compiled which constitute the optimal defense strategy (Mitigation Strategy) tailored to the caclculated cyber-risks. These are modelled as: a) **low-level control-flow attestation policies**, and b) **specific system calls to be monitored** for providing guarantees against specific remote exploitation vulnerabilities (Section 3.3).
This combination allows us to express fine-grained trust assumptions in a “top-down” manner starting from the description of the applications trust domain, and iteratively refining it to model internal interactions between the entities involved, and the specific operations performed by each device. Such fine-grained trust assumptionsonce expressedcan be used to precisely delimit the contextual interactions under which the TPMs security guarantees are proved to hold, and can be monitored, or even partially enforced, through the Control-Flow Attestation Toolkit. These low-level policies will be received by the Policy Decision Point, to be deployed and enforced by the Policy Enforcement Point to the host devices ecosystem.
Overall,such security policies represent a view of a security conguration which is considered optimal within a time frame and that **satisfy a set of specific constraints**. The constraints, which are taken into consideration in the risk quantification process, are the: a) **asset listing** of all infrastructure/edge devices and systems, b) **assets relationships** modeled as interdependecy graphs, c) **achievement of specific risk levels**, e.g., . we require the security policy that would minimize the risk of exploiting Asset$x$ to 0.1, and d) preferred, high-level security, privacy and trust

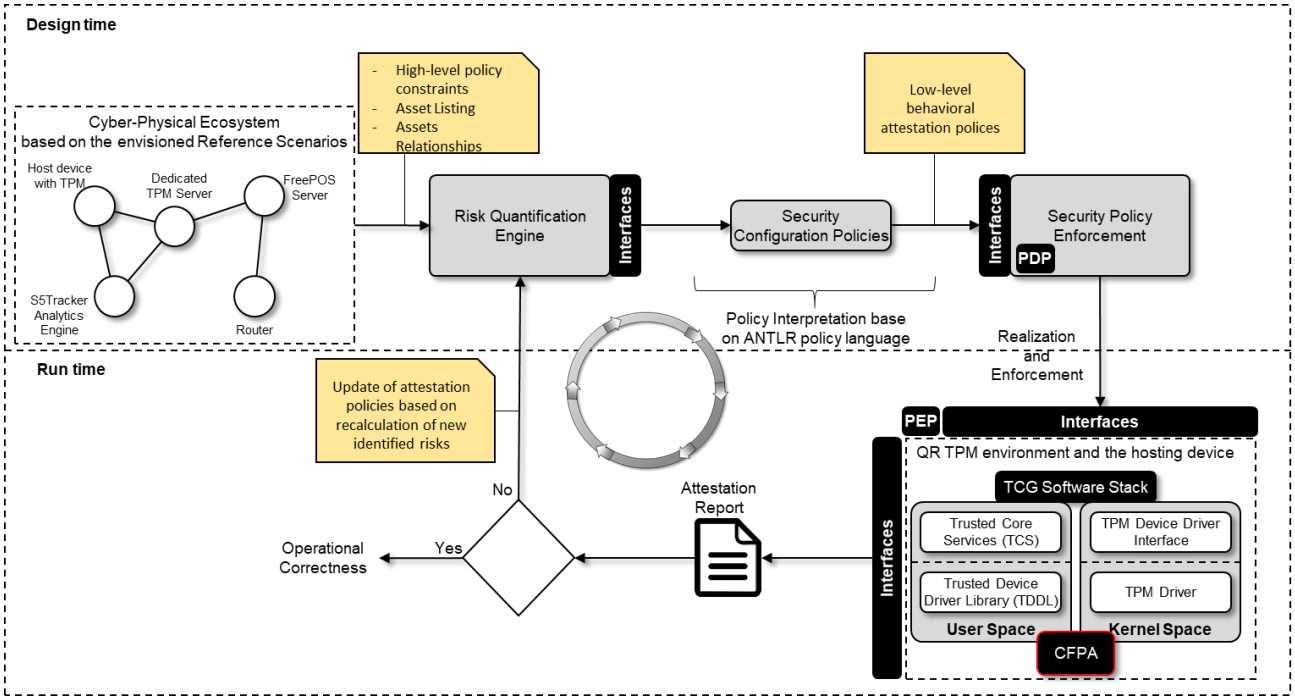


Figure 2.1: Risk Assessment Conceptual Flow

requirements (as modeled in Chapter 3 of Deliverable D4.2 [7]). Therefore, the policy generation is a linear optimization problem which is tackled by a near-optimal constraint satisfaction solver.
However, **policy changing** facts are likely to occur within the lifetime of the envisioned ecosystem of deployed devices: Such facts can be (indicatively) the disclosure of a zero-day vulnerability or a detected anomaly based on the output of the **Control-Flow Property-based Attestation (CFPA)** toolkit. If any of the enforced security policies fail, from malicious intent or faulty behaviour, the next logical step is to identify what was the cause of this event. This will enable better **situation awareness adaptation** for **re-calculating the overall risks and threats** of the entire ecosystem (considering the newly identied vulnerability) allowing **policy adjustments** and the **compilation of updated mitigation strategies and control-flow attestation policies.** This is the main functionality performed by the risk assessment framework **during run-time**.
Recall that the output of the CFPA will provide a Boolean decision regarding a systems conguration and execution integrity with respect to the properties that were attested. If the output is “YES”, then this reflects the appropriate statements on the correct and trustworthy execution of the deployed devices (**Operational Correctness**); otherwise, a “NO” attestation report reflects that a **deviation from the normal device behavior** was detected and the Attestation Toolkit informs the **FutureTPM Secure Tracing & Trust Evidence Collection** (Chapter 3) engine to start a more detailed monitoring and tracing of the devices execution so as to collect more information that can be used for an offline investigation on the exact attack details and point of intrusion. This in turn will be fed to the Risk Quantification for the **re-calculation of the overall risk and threat vector and for the re-configuration of the security policies**.
Towards this direction, the FutureTPM Run-time Risk Assessment enhancement, of the overall RA framework, will be responsible for performing this re-caclculation based on the following techniques: a) the **Backwards Chaining** to resolve the given set of constraints and b) the **Cascading Effect Analysis**.
Reverse Chaining: It uses the same basic approach as forward chaining but in reverse order. In the backward chaining mode the **engine will be provided with a specific goal regarding a risk**

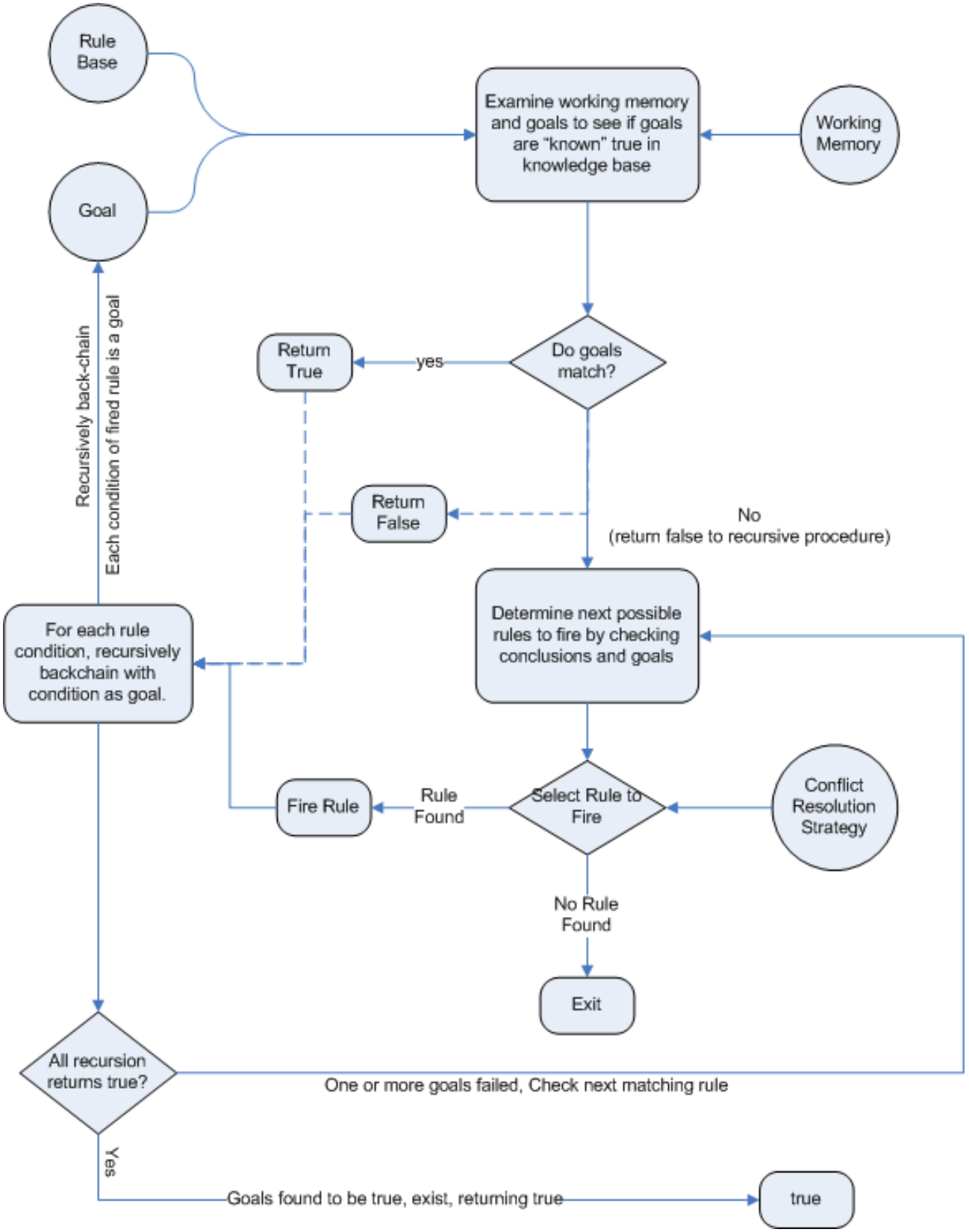


Figure 2.2: Reverse Chaining [16]

level and then it will propose a set of **control elements** that can satisfy this goal. This controls mainly constitute the control-flow attestation policies: the **set of properties that if attested can**

achieve the desired level of assurance. Therefore, the first mode is **rule-driven** (e.g. signature-based vulnerabilities) while the second one is **goal-driven** (e.g. the properties attestation). Goal-driven, meaning that we start with a conclusion which the engine tries to satisfy. If this cannot be achieved, then it searches for conclusions that can be satisfied; these are known as sub goals (e.g. **operational and safety assurance**, etc.), that will help satisfy some unknown part of the current goal. It continues this process until either the initial conclusion is proven or there are no more sub goals. The report that is generated, in both cases, can be interpreted in concrete policies. **This will enable to see which sub-goal is responsible for a failed attestation report, thus, allowing for a more targeted evidence collection.**
The output of the Risk Quantification Engine will be assessed in order to evaluate if a set of specific risks are below acceptable thresholds. The thresholds will be set based on the three reference scenarios. **Drools has been integrated for performing this backward chaining process, which we refer to as derivation queries.** Figure 2.2 presents the activity diagram Backward chaining questionnaire [16].
Cascading Effect Analysis: The detection of possible device abnormal behaviour entail clearly **pro-active** nature. However, efficient reactiveness is a key aspect in cyber-security. Part of the actions that have to be accomplished upon an incident is the evaluation of the **level of penetration within the target ecosystem**. In most of the cases this is a manual task which is performed by an emergency response team in a rather intuitive way. FutureTPM will rely on a **dependency model** which will track **the functional interdependencies among the various assets in order to infer the cascading effects in case of an asset compromise.** This feature would radically increase the efficiency of mitigation actions.

In the case of run-time risk assessment, **near real-time risk quantification** of newly identified attacks will also be performed. In particular, new target values will be set as input to the Quantification Engine which will be invoked in the mode of goal-driven inference engine. This means that it will propose several new control elements, i.e., **mitigation controls that map to existing properties that have to be attested.** These properties may be a subset of the configuration properties that are already defined or can be other newly-identified that can further enable semantic, behavioural remote attestation, i.e., attestation of dynamic, arbitrary and system properties as well as behavior of executable code in an attempt to mitigate the newly discovered run-time vulnerabilities. **This process may lead to a dynamic update of the already defined policies as a response to these newly identified attacks.**

2.2 Implementation Aspects

2.2.1 Core Components & Building Blocks

The implementation of the run-time risk assessment (which is highly interdependent with the **FutureTPM CFPA toolkit** and **Risk Quantification Engine**) includes three main components, as listed below:

- **eBPF Runtime Tracer** This is one of the core building blocks of the CFPA and run-time risk assessment [7] for dynamic tracing: the implemented **eBPF Runtime Tracer** performs a **detailed dynamic tracing** of the kernel shared libraries, low-level code, etc., and an in-depth investigation of the systems behaviour and execution ow. More specically, it provides the trusted anchor with the compiled CFGs that represent the **run-time state of a remote device**, against only those properties of interest included in the deployed control-ow attestation policies, that need to be attested (Section 3.1.1). The tracing takes place through

TPM2 Commands and Timing (≃ ms)			
TPM2_CreatePrimary	11000	TPM2_FlushContext	60
TPM2_Create	40	TPM2_Commit	45
TPM2_ContextSave	50	TPM2_PCR_Extend	55
TPM2_ContextLoad	50	TPM2_GetRandom	15
TPM2_Load	45	TPM2_Hash	15
TPM2_RSA_Encrypt	420	TPM2_Sign	70
TPM2_RSA_Decrypt	450	TPM2_MakeCredential	150
TPM2_EncryptDecrypt	500	TPM2_ActivateCredential	50
TPM2_EncryptDecrypt2	500	TPM2_PolicySecret	20
TPM2_ReadPublic	65	TPM2_Certify	20
TPM2_GetCapability	75	TPM2_VerifySignature	200
TPM2_StartAuthSession	20	TPM2_ECDH_KeyGen	150
TPM2_PCR_Read	55	TPM2_ECDH_ZGen	380
TPM2_PolicyPCR	60	TPM2_ECC_Parameters	90
TPM2_PolicyGetDigest	60	TPM2_EC_Ephemeral	90
TPM2_EvictControl	15	TPM2_ZGen_2Phase	85
TPM2_Unseal	120	TPM2_CertifyCreation	25

Table 2.1: Implemented TPM2 commands on the eBPF Run-time Tracer

the deployment of already defined eBPF hooks, initially identified as low-level behavioural properties, during design-time.

- **FutureTPM Secure Tracing & Trust Evidence Collection:** This component as described in Chapter 3, is responsible for providing the functionalities of **multi-level detailed tracing** enabling the collection of the necessary information/evidence, in the case of a "failed" attestation report, for **post-attack investigation**. This enhanced tracing is implemented in a semi-automatic manner based on two possible approaches (currently under investigation) (Section 3.3.1): a) **automatic deployment of new, more rich, programmable eBPF hooks**, and b) **deployment, during design-time, of eBPF hooks already capable of enhanced monitoring and tracing of mode of the operational system calls** (to be activated only upon a failed attestation process).

- **Backward Chaining Process**: Functionality towards the **re-calculcation of the ecosystem's risk and threat vector** taking also into consideration the newly identified and investigate vulnerabilities, based on the output of the evidence collection phase.

In the context of the **eBPF Runtime Tracer**, the goal of this initial tracing is to monitor system calls, **produce the necessary CFGs** so that they can be fed to the **CFPA Verification Engine** for acquiring the attestation report (more information on the workflow of the CFPA can be found in Chapter 4 of D4.2 [7]. For instance, in the context of the TSS attestation, information of interest will include: **execution time, process name of invoked libraries, process id, time of internal operations, parsed TPM commands, etc.** This information is the output of the eBPF tracing process.

As was also described in Deliverable D4.1 [8], the first implementation instance of the eBPF Runtime Tracer focused on how to trace all the needed TPM commands (per use case) and identify possible **object, sequence leakage and possible exploitation attempts;** by either manipulating the parameters expected during the invocation of these TPM commands (through the TSS) or by exploiting the insecure nature of the leveraged crypto primitives against quantum adversaries (e.g., ECC has been proven broken and needs to be updated with a different algorithms in the TPM2 Commit that performs the rst part of an ECC anonymous signing operation)[1].
To this end, we evaluate the timing requirements in order to trace the entire TSS. Table 2.1 lists the implemented traces, of the TPM commands, and the corresponding timings per command (of our parser) and Table 2.2 provides the output information of the run-time tracer including example snippets.

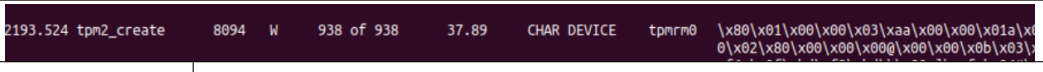
TPM2 Output Information	
Kernel Space	**total time passed** in seconds (e.g. 2193.54) **process name** (e.g. tpm2_create) **process id** (pid) (e.g. 8094) **Virtual File System** (VFS) function hooked (e.g. W) **number of bytes** read (e.g. 938 of 938) **time of the operation** to complete in milliseconds (e.g. 37.89) **type of files** (e.g. CHAR DEVICE) **device** / **file name** (e.g. tpmrm0) **parsed data** from kernel with eBPF
	
User Space	**parsed TPM commands**
	

Table 2.2: Runtime Risk Assessment output information

2.2.2 Run-time Risk Assessment APIs

As described before, the run-time risk assessment mainly revolves around the re-calculation of risks, for the entire ecosystem of devices (**assets, assets listing, and assets interdependency**

[1] In the current implementation version of the FutureTPM Risk Assessment, all TSS-related vulnerabilities have already been modeled.

graphs), taking also into consideration the newly identified threats and vulnerabilities. This process is performed by the **OLISTIC-based Risk Quantification Engine** and, thus, is based on the same APIs as were described in Chapter 2 of Deliverable D4.2 [7].
In this context, the **Backward Inference API** is based on the Risk Assessment API, as depicted in Figure 2.3. The "*create*" method creates an instance of the Risk Assessment Entity. Each asset may be associated with a set of **Vulnerabilities, Threats and Controls**. In order to attach the new collected information from the Evidence Control, in the Risk Quantification Engine, the first three functions (e.g. "*attachRiskassessmentAssetControl*", "*attachRiskassessmentAssetThreat*" and "*attachRiskassessmentAssetVulnerability*") are used.
The calculation of the risk per se is performed by invoking the "*execute*" method. After the calculation of the risk, someone can traverse through the calculated risks using two helper functions. The first is "*getRraaset*" and the second is the "*getRrassetPerIRL*". The former is returning an **aggregate view of the calculated risks** while the latter is returning the **Individual Risk Level (a.k.a. IRL)**, per asset individually.

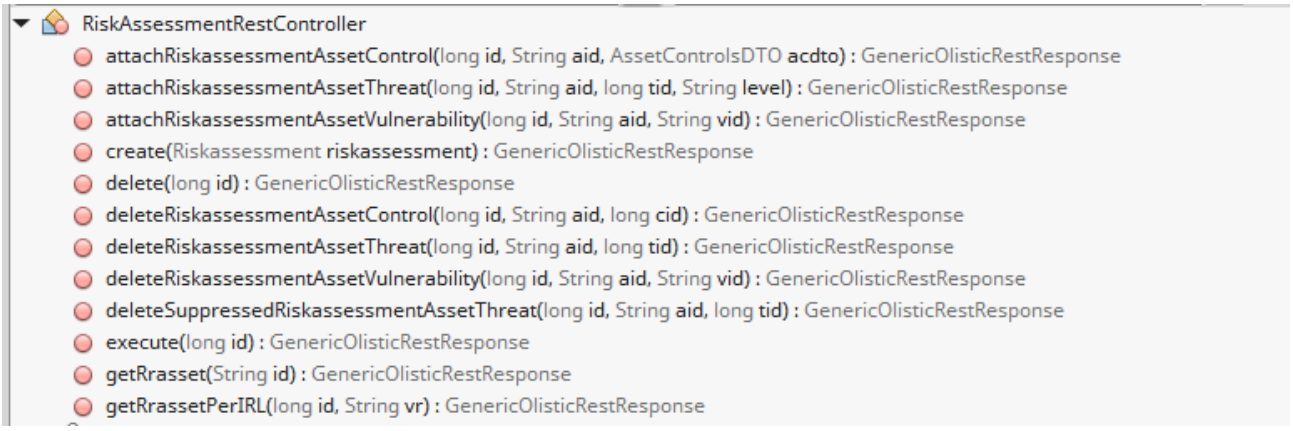


Figure 2.3: FutureTPM Run-time Risk Assessment Framework APIs

Chapter 3

Evidence Collection

3.1 Adaptive Policy-driven Attestation and Trust Evidence Collection

As described before, in FutureTPM there will be certain enforced policies tailored to maintain a certain level of assurance in the form of **memory safety**, **control-flow safety** and **type safety**. These policies, classified as **Control Elements** by the FutureTPM Risk Assessment framework [7], mainly constitute the **optimal defense strategy (Mitigation Strategy)** tailored to the calculated cyber-risks. As it can be easily inferred, such policy making is strongly dependent on the outcome of the Risk Quantification engine. They are modelled as Controls that reflect the **control-flow attestation policies** (including the low-level configuration and behavioural properties) to be deployed and enforced during run-time.
Overall, such security policies represent a view of a security configuration which is considered optimal within a time frame. However, **policy changing facts** are likely to occur within the lifetime of the envisioned ecosystem of deployed devices: Such facts can be (indicatively) the disclosure of a zero-day vulnerability or a detected anomaly based on the output of the **Control-Flow Property-based Attestation (CFPA)** toolkit.
If any of the enforced security policies fail, from malicious intent or faulty behaviour, the next logical step is to identify what was the cause of this event. Towards this direction, what is needed, is the capability to be able to collect more information for answering **security-critical questions regarding the policy failure** and get more insights of **what**, **why**, **how** and by **whom** this problematic operation occured. This will enable better **situation awareness adaptation** for re-calculating the overall risks and threats of the entire ecosystem (considering the newly identified vulnerability) allowing **policy adjustments** and the **compilation of updated mitigation strategies and control-flow attestation policies**.
However, recall that the output of the CFPA will provide a Boolean decision regarding a system's configuration and execution integrity with respect to the properties that were attested. Such a decision is usually not sufficient to understand a device's behaviour when the attestation output is negative; especially when a more complex control-flow graph (CFG) is the reason of the attestation failure (in the case of specific system calls being exploited, mitigation measures are more easily constructed (Chapter 4)). Thus, in this case, a more **in-depth investigation of the system's behaviour is needed** to detect any cheating attempts or if any type of (non-previously identified) malware is resident to the program and data memory.
In the context of FutureTPM, we are solving this issue by providing a **semi-automatic way of performing complementary system tracing** that will be able to provide more information on any

cheating attempts. This is the goal of the **Trust Evidence Collection** component that provides the functionality of data collection (in the case of a "Fail" attestation report) regarding the execution behaviour of a system that can be subsequently used for the **analysis and classification of potential runtime vulnerabilities and attacks** to be fed in the Risk Assessment framework for the re-calculation of the overall system risk vector. The exact nature of evidence to be collected is defined through an offline investigation, once the failed attestation report is received, by security analysts and can vary depending on the type of properties that were attested. For instance, this evidence may include access patterns to different memory regions, libraries, ports and network interfaces, stack frames, etc. The service will provide **run-time monitoring functionalities** for collecting this evidence over some period of time (in the case of a negative attestation outcome), according to specified policies for **attack detection and identification of the point of intrusion** (Section 3.2).

3.1.1 Runtime Monitoring and eBPF-based Tracing

Run-time monitoring and trust evidence collection will be based on the use of **eBPF execution hooks**. As described in Deliverables D4.1 [8] and D4.2 [7], eBPFs are one of the core building blocks of the CFPA for **dynamic tracing**: the implemented **eBPF Runtime Tracer** performs 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. More specifically, it provides the trusted anchor with the compiled CFGs that represent the runtime state of a remote device, against only those properties of interest included in the deployed control-flow attestation policies, that need to be attested. The intuition behind the use of eBPFS is that they are **lightweight** enough for such a detailed tracing and can provide near real-time low-level code inspection, thus, capturing the requirements of the envisioned applications.
Following the same approach, the same type of **eBPF-based tracing enablers** are implemented for **multi-level detailed tracing** (Section 3.3.1). These will be triggered upon an attestation failure and will be either deployed or activated for providing more insights on the state of running system resources. As aforementioned, the primary interest is the analysis of the execution flow of a device which represents one of the main threat vectors in FutureTPM [6]. In this respect and in contrast to other state-of-the-art tracers (see below), **eBPFs provide the uncommon characteristic that it can be linked to any kernel function**, including system calls and I/O. The eBPF is **fully programmable** and this represents another priority requisite for **dynamic adaptation of tracing and inspection tasks**.
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 [23]. 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.
Linux Trace Toolkit Next Generation (LTTng) tracing adds up considerably the collective tracing impact on the target software for long runs, in resource constrained and high throughput environments, such as embedded network nodes and production servers [27]. An other example of dynamic tracing is DTrace. However, a visual survey of the DTrace code reveals that the DTrace compiler offers very limited optimizations compared with the eBPF bytecode [27]. An important aspect regarding tracing is the need for filtering due to the large number of generated data [27],

DTrace, LTTng and eBPF has been addressed such a need.

3.2 Attacks and Vulnerabilities Detection & Investigation

There is much research on the topic of attack and threat detection that theoretically aligns with the system tracing of FutureTPM. Most of these bibliographic entries propose novel methodologies of how the attack detection will take place but do not elaborate on how this threat detection will escalate when an attack happens in order to capture any additional trust evidence. A common method of attack detection is tracing system calls and identifying any abnormal behavior which will then be classified as an attempted attack. In [18] the researchers propose the collection of system calls in the form of audit trails, that is, collected sequences of system calls during run-time. These sequences can be used either during run-time to immediately detect any threats or offline to detect attacks after the fact that can be thought as trust evidence collection. In the ideal case,the system can analyze the trail online as it is created, flag any unusual, anomalous, or prohibited behavior immediately, and then initiate a response. If it must examine the trail off line, this can take place routinely during off-peak hours or when unusual behavior has been detected by some other means. There is a chance, in this case, that a particularly successful intruder could corrupt the trail and hide the intrusion. For this reason, a computationally fast on-line method is useful.
Furthermore, in [22] there is a similar approach in terms of system call sequencing but there is the addition of clustering and machine learning mechanisms for the automatic detection of abnormal behavior. The research describes a clustering methodology for the distinction between normal and malicious system behavior as well as a marokovian model for analyzing each system call sequence and classifying it accordingly. This approach, although automated, it is much more resource heavy and poses the risk of possible false positives and more importantly false negatives that could lead to undetected attacks. Finally an interesting case in [24] where the researchers propose the utilization of hardware acceleration to improve the performance of attack detection in embedded devices.
Another more advanced methodology despite system call tracing is attack detection through control flow monitoring. There is a wide range of research on this area due to the nature and size of the system control flow graph. Mapping the entire control flow graph of an application can be tedious since there can be many branches and the control flow tree can go uncontrollably. That is why, when it comes to checking and attesting the integrity of a service control-flow, 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 [2]. 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 significant performance overhead; thus, it is suitable only for small size binaries. Subsequent works such as LO-FAT [11], LiteHAX [10], SCAPI [17], SCARR [29], hardware-based attestation [31] and CFIMon [30] aim to reduce this overhead 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 low-powered devices.
Most of the aforementioned control-flow attestation mechanisms support single-device attestation where a trusted party, the *verifier*, is able to check the integrity of a remote device, the *prover*. However, as aforementioned, applying such techniques results in a huge overhead. Towards this direction, collective attestation protocols like SANA [3], SEDA [4], DARPA [15] and CAMIE [20]

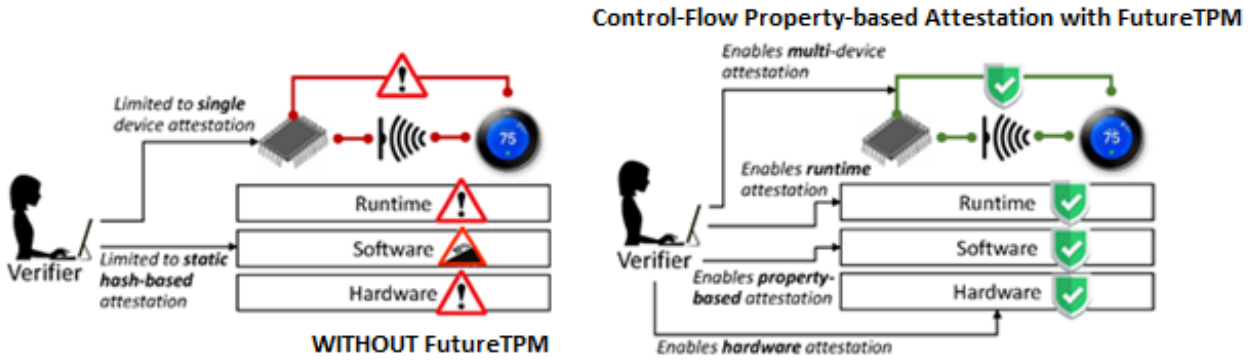


Figure 3.1: FutureTPM Methodology and Advancement wrt to Remote Attestation

have also been proposed that decongest the network by distributing the computational and communication burden to all provers, thus, 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.

3.3 FutureTPM Methodology

Leveraging any of the previously described mechanisms, as a standalone component, is not enough in the case of FutureTPM as there is no definition for an **escalation plan**; just detecting an attack is not enough for a complete, holistic security system managing the complex security and operational assurance properties as the ones revolving around the use of TPMs. Thus, as has been elaborated, we prompted to use both techniques: a) the **tracing system calls**, and b) a newly designed **control-flow property-based monitoring**.
As described in the previous section and also depicted in Figure 3.1, there does not yet exist a comprehensive design nor an effective as well as efficient implementation for enabling dynamic attestation. Moreover, existing attestation approaches are limited to single device attestation, whereas in FutureTPM we target attestation of systems-of-systems to address the emerging class of interconnected embedded devices in the Internet-of-Things. To provide strong security assurance in this context, we bridge this gap within FutureTPM and developed novel dynamic attestation mechanisms, particularly focusing on **behavioural-based attestation dynamic properties of software and hardware for systems-of-systems**, through the deployment of eBPF execution hooks.

3.3.1 Multi-Level Detailed Tracing

These kernel hooks, initially, are identified as low-level behavioural properties and are deployed to trace system calls. The goal of this initial tracing is to monitor system calls, **produce the necessary CFGs and finally acquire the attestation report**. In the case of a failed attestation report, FutureTPM has defined a methodology for **increasing the level of monitoring** in order to collect additional evidence and information on the incident for the assistance in finding the province of the attack as well as in the development of new enforceable policies that should be able to catch this newly identified threat that caused the attack in the first place. Effectively, we propose a novel solution for **multi-level detailed tracing** that, depending on the situation, **mon-**

itors the security-critical components of each system in different levels in order to upkeep the desired assurance while also providing the required details for **post-attack investigation**.
This multi-level detailed tracing is implemented in a semi-automatic manner. Towards this direction, there are two possibilities explored within FutureTPM. The first is the **automatic deployment of new, more rich, programmable eBPF hooks** after the reception of a failed attestation report. In this case, the security analyst(s) (performing the offline investigation) assess the attestation report and determines whether more information is needed for identifying the **cause and type of attack** - in which case she also defines the set of policies describing the type of information/evidence to be collected. After this process, new programmable ebPF kernel hooks are seamlessly and automatically deployed (Section 3.3.2) to the target device for fulfilling these policies. As previously noted, the eBPF is implemented in the kernel, thus, needs a userland **control agent for remote control and exportation of the data**. Such agents are going to be implemented as part of the FutureTPM RA framework. Overall, this approach enable us to get the necessary evidence without **affecting the performance of the target system** since these rich eBPF hooks (that will incur a higher penalty in the system execution as more system calls need to be monitored in near real-time) will be deployed. However, there is the inherent limitation of a **large attack time window**: Since the detailed tracing commences after the attestation process has failed, this means that the attack is already in place which in turn results to the additional requirement of "letting" the attack to run further (for a larger amount of time) in the device until the detailed tracing has finished. Depending on the application at hand, such an approach (which will of course require the isolation of the attacked device during the evidence collection so as other adjacent assets are not affected) might not be feasible.
Compounding this issue, another approach that is currently been investigated is the **deployment, during design-time, of eBPF hooks already capable of enhanced monitoring and tracing of most of the operational system calls**. In this case, **only those execution hooks** that are necessary for tracing the CFGs to be attested will be active during the run-time risk assessment phase (i.e., including the execution of the CFPA mechanism) and the **enriched eBPFs will be activated** only if the evidence collection is triggered; thus, resulting to the transmission of the necessary information about the device that got compromised, vulnerabilities found in any of the internal system components or their configurations, configuration changes, etc. This approach while less efficient, as the penalty of increased tracing is incurred from the beginning, doesn't necessitate a large attack time window. Further investigation on the feasibility and usability of these two mechanisms will be done during the first testing phase of the FutureTPM demonstrators and will be documented in the second version of this deliverable (D4.5).
Summarizing the process of the FutureTPM **tracing hooks deployment**, **multi-level detailed tracing** and **trust evidence collection** mechanisms, the conceptual work-flow is depicted in Figure 3.2.The process of this deployment is as follows:

1. An attack or malfunction occurs causing an **enforced policy to fail** as detected by the output of the **Control-Flow Property-based Attestation** mechanism;

2. The security analyst(s) asseses if she needs **additional information related to the attack incident**. If she can directly deduce what was the cause of the attack then the process ends here;

3. The security analyst(s) **defines the required information** (through adequate policies) and **directly deploys** (Case 1 in Figure 3.2) or activates (Case 2 in Figure 3.2) **the enriched eBPF execution hooks** for collecting the necessary additional information;

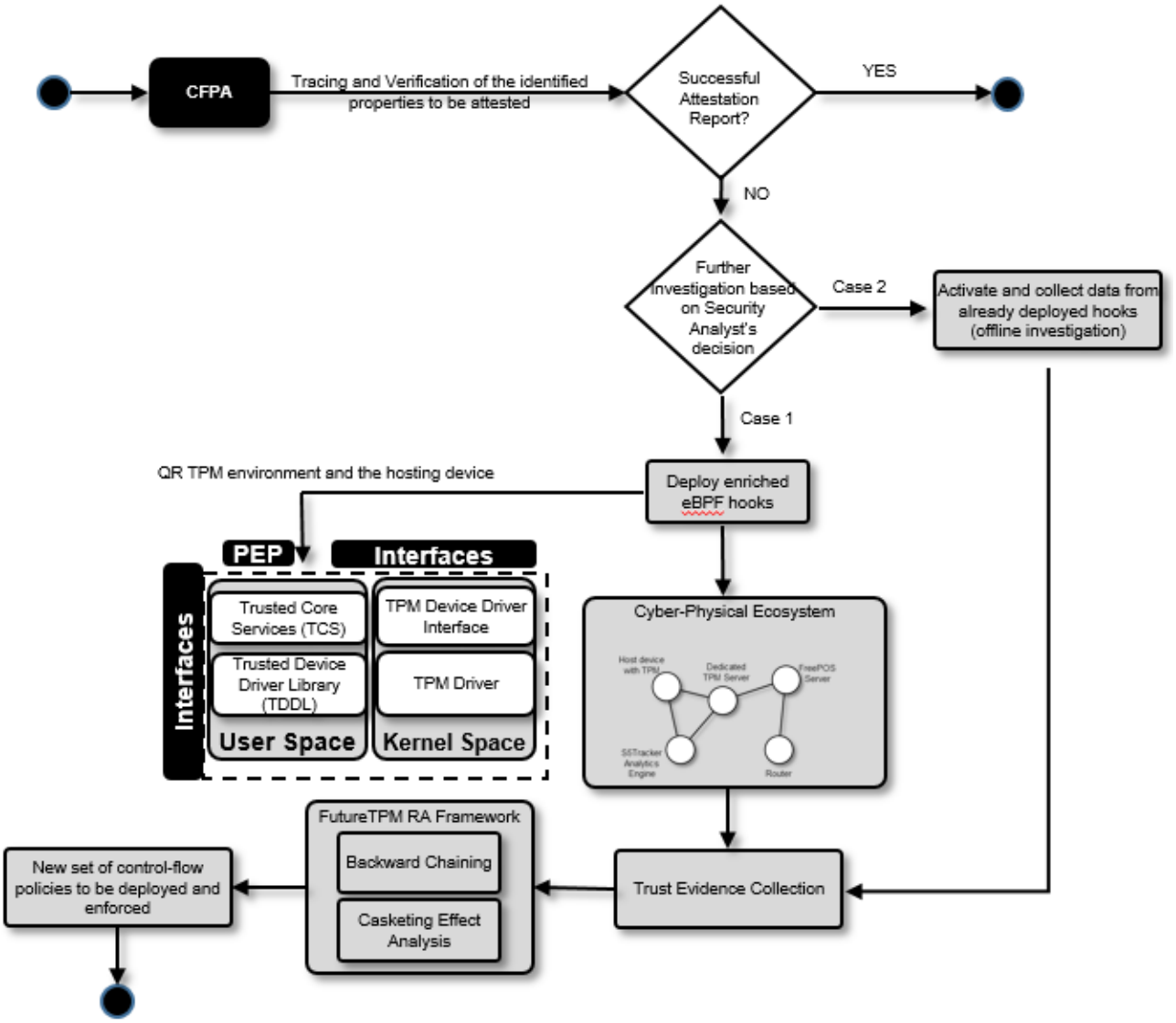


Figure 3.2: Conceptual Work-flow of Multi-level Detailed Tracing & Trust Evidence Collection

4. All the information collected is fed to the **FutureTPM Run-time Risk Assessment framework** for performing a re-calculation of the overall risk and threat vector (considering **Backwards Chaining** and **Cascading Effect Analysis** (Chapter 2)). Compilation and enforcement of new attestation policies that should be able to cover it in the future.

The two main key points of this method is that the security analyst will either require or not any additional tracing. The **standard tracing kit** will be running on the end device (*prover*) generating attestations of its control-flow graphs and sending them to the *verifier*. When a policy fails, then the security analyst will read the attestations generated by the prover and will assess if she needs additional tracing to identify the source of the event. Depending on her choice she will either end the evidence collection process or **deploy/activate any additional tracing** required to gather the required data. For example, when a buffer overflow attack occurs that will alter the control flow graph (and thus, failing the corresponding behavioral property policy) of the monitored application, then there might be the need for additional information such as what memory regions where overwritten, what memory regions where over-read or which entity started the malfunction. On the other hand when an unidentified application is found to run on a restricted device (thus

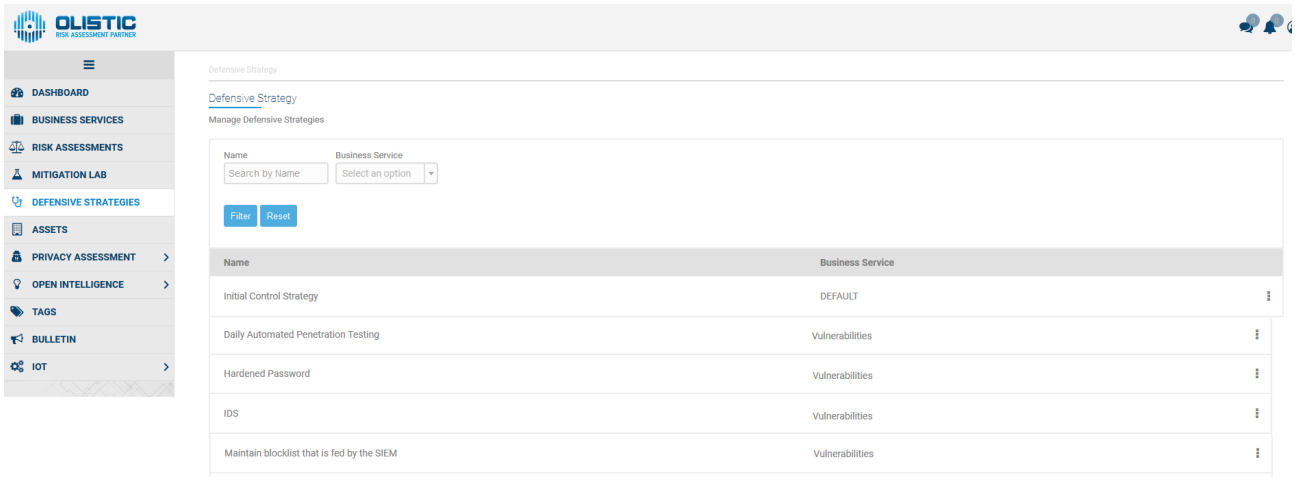


Figure 3.3: Execution hooks as part of the Defense Strategies in OLISTIC-based Risk Assessment

failing the corresponding configuration property policy), then from the already existing attestation the security analyst can black-list this application without any additional information. In both cases, **new policies will be compiled that will be able to catch this attack** before it happens and prevent it from breaching the security of the end device.

3.3.2 Automatic Deployment of eBPF hooks

As already aforementioned, if the calculated risk (based on the output of the **Risk Quantification Engine**) is not acceptable or the result of an attestation report has failed (based on the output of the **CFPA**), then a new configuration and security policy needs to be enforced. More specifically, the configuration policy includes a high-level access control policy or new interpreted low-level execution properties that needs to be attested. In the case there is a need to enforce a new policy, there is also a possibility to **update the tracer and deploy new eBPF hooks automatically** (Case 1 of Figure 3.2). This subsection focuses on the automatic deployment of such eBPF hooks.

This process will work according to an **Event-Condition-Action pattern**, where **deployment events** are trigerred by the failure of an attestation report or a calculated risk with high Impact, being above a threshold, and **actions entail the deployment and/or modification of the eBPF execution hooks** (monitored data, frequency, granularity, filtering, etc.), **re-configuration of the control-flow attestation policies, etc.** The description of data to be monitored and collected and the automatic deployment of the kernel hooks are expressed by policies, which also represent the "smartness" of the **FutureTPM Secure Tracing** component and encompasses both reaction and prevention actions as well as defensive policies.

This Secure Tracer instantiates and configures these programmable execution hooks to be deployed to the envisioned ecosystem of devices. For instance, if the focus is mainly against **remote memory and data exploitation attacks**, the tracer will instantiate the hooks for **monitoring system calls** that try to access the data variable of interest (in the device's memory) and also are capable of compiling in real-time the necessary **Control-Flow and Data-Flow Graphs** (CFGs and DFGs, respectively) to be then attested by the CFPA. This controller is also capable of **re-programming the execution environment of existing eBPF hooks**, leveraging various means of abstraction. The main technical challenge here is the translation of the evidence-related poli-

cies (definition of evidence to be collected) into configurations and code for the heterogeneous set of security hooks. In the current architecture, this is realized by selecting pre-defined programs and configuration files from an internal library, but the long-term ambition would be the definition of dynamic code generation and run-time compiling.
All these control elements and tracing components are part of the **Security Policy Enforcement** mechanism that is highly interdependent with the FutureTPM RA framework. In the current implementation, eBPF hooks that are instantiated during design-time to capture the requirements of the extrcted control-flow attestation policies can be deployed as part of the defense strategies of the OLISTIC-based risk assessment toolkit (Figure 3.3).
Recall that the Security Policy Enforcement strategy [9], [12], [13] implements a **policy-driven approach for allowing proofs of a system's integrity based on the attested properties**. This architecture specifies how **composition of large scale "Systems-of-Systems"** is to be controlled via layered and cross-domain attestation decisions. The outputs of the FutureTPM RA framework dictate the control-flow attestation policies required to mitigate the identified risks. Such policies are **expressive**, **deployable** and **enforceable** within the Security Policy Enforcement architecture and may be dynamically updated if the attack graph (produced and maintained by the RA framework) is amended with new types of vulnerabilities.

Chapter 4

Mitigation Strategies

As stated above, this multi-level tracing aims to alleviate any restrictions that comes with classic static monitoring techniques, while maintaining a small overhead, towards the compilation of **optimal mitigation strategies** reflected through appropriate control-flow attestation policies. In the context of FutureTPM, as described in Deliverable D4.2 [7], the main focus is on mitigation measures against attacks that target the memory and control safety of a device's execution. In what follows, we list some interesting, state-of-the-art lines of research that are currently being investigated, in the context of FutureTPM, so as to be taken into consideration when constructing the eBPF tracing hooks to be deployed.

Currently, as described in Deliverable D4.2 [7], FutureTPM designed and implemented a novel control-flow property-based attestation scheme for attesting specific execution properties in the target device. This enables the verifier to detect run-time attacks based on code reuse. Enforcement techniques for control-flow correctness, such as control-flow integrity (CFI) [1], do not provide information about the executed control-flow path on collaborating devices. Control-flow attestation, however, gives the verifier information about the executed control flow, which enables the verifier to detect not only attacks that do not conform to a software programs control-flow graph, like return-oriented programming (ROP) [26], but also a subset of attacks that lead to a valid but unintended program execution, i.e., non-control data attacks [5], [14]. Moreover, control-flow attestation enables the verifier to determine the appropriate reaction, in case of an attack. With CFI, a violating device would be stopped and possibly crashed, which is particularly dangerous in safety critical applications. With control-flow attestation, in contrast, contextual reactions can be implemented, e.g., excluding a compromised device from the collaboration and the entire communication.

In [25], the authors have created new instructions for the OpenRISC and RISC-V architectures in order to be used as countermeasures against **memory integrity attacks**. These instructions are divided in two categories; one for **return address protection** and one for the security of the **security-sensitive memcpy() instruction**. Additionally, the researchers have proposed a hardware stack that should be more difficult to exploit than the traditional software stack. Furthermore, in [19] there is a similar approach, where the researchers propose a Built-in Secure Register Bank (BSRB) which is a hardware based register that will store the return address of each sub-routine with the purpose of defending against control-flow attacks.

The researchers of [28] have proposed a framework for current ARM mobile devices that can detect **application control-flow manipulation** attempts by looking at the history of executed control-flow altering instructions on the processor. This history examination provides enough information to implement the state-of-the-art fine-grained control policies, without additional binary instrumentation. Moreover, this framework is designed to work with existing hardware and have

a minimal impact on performance. This solution is dependant on the ARM TrustZone technology as a trust anchor for integrity of the monitoring process.
The closest research to the FutureTPM multi-level tracing is found in [21], where the researchers implement a monitoring solution for distributed systems. This research identifies the problem of static solutions that employ only pre-defined monitoring that targets specific security-sensitive system attributes and proposes a novel scheme that has two basic contributions. First of all the security analyst is able to filter and analyze the results of the entire system even when crossing components or machine boundaries of the distributed system. The second and most meaningful contribution is that the security analyst can deploy new monitoring on the running system seamlessly in order to investigate anything he might require. **FutureTPM proposes a more complete technique that also supports dynamic policies derived from any additional monitoring**.
Another advanced line of research, in the context of remote attestation, is the enrichment of the threat model to also include **data oriented attacks** for **verifying data trustworthiness**. More specifically, for the correct operation of complex systems, each device must be able to verify that the data coming from other devices is correct and has not been maliciously altered. Towards this direction, the FutureTPM CFPA toolkit will be enhanced to also include the **attestation of Data-Flow Graphs (DFGs)** - on top of the already considered control-flow graphs - towards enhanced data-flow integrity. This will also require the construction of **data-flow monitoring eBPF hooks** for accessing and monitoring the data of interest. In the context of data-flow integrity, the term data integrity is used to describe that **all** operations on data and variables obey a program's DFG. Such DFGs will also be expressed as data-flow attestation policies focusing solely on data that has been explicitly selected to be controlled. This data inherits its integrity from the integrity of the software modules that processed it.
This will be achieved by decomposing the underlying embedded software (in the target device) into small interacting software modules and attest the control-flow of those modules that are relevant for data exchanged in a given interaction. The control-flow attestation guarantees that the data is only processed in a benign execution path. Overall, the main goal of the enhanced CFPA variant would be to enable efficient and secure interaction/collaboration of embedded devices in an autonomous system achieving:

- **Code integrity on devices.** Unintentional/malicious alternation of the code running on a device can be detected based on the compiled control-flow attestation policies.
- **Data integrity on devices.** Unintentional/malicious alternation of the data on a device (before being sent out to other devices) can be detected. This means data can only be modified in a non-malicious way. This is necessary as devices do not only exchange raw data but mostly processed data.
- **Data integrity and authenticity during transportation.** Malicious alternation of the data when traversing from one device to another must be detected.

Chapter 5

Conclusions

This final section will act as a synopsis of this deliverable and summarize its findings. The scope of this deliverable was to document the Run-time Risk Assessment. More specifically, Deliverable D4.3 highlighted the conceptual work-flow of Run-time Risk Assessment including the Implementation components and APIs, the detailed Evidence Collection and the Mitigation Strategies.
During design-time, the Risk Quantification Engine **quantifies all possible risks based on a number of input constraints**. The produced **security configuration policies** are interpreted to low-level control-flow attestation policies and/or specific system calls to be monitored for providing guarantees against specific vulnerabilities. **During run-time**, these low-level properties will be attested by the CFPA and the Attestation Report will result in a binary verdict. **A more in-depth investigation of the system's behaviour is necessary in case of a "failed" attestation report**.
During this in-depth Evidence Collection, a novel multi-level tracing and the mitigation measures for post-attack investigation is described including the automatic deployment of enriched eBPFs hooks. **This multi-level tracing aims to alleviate any restrictions that comes with classic static monitoring techniques, while maintaining a small overhead.**

Chapter 6

List of Abbreviations

Abbreviation	Translation
APT	Advanced Persistent Threat
BSRB	Built-in Secure Register Bank
CFPA	Control-Flow Properties-based Attestation
CFG	Control-Flow Graph
CFI	Control-Flow Integrity
CPS	Cyber-Physical Systems
CRL	Cumulative Risk Level
CVSS	Common Vulnerability Scoring System
DFG	Data-Flow Graph
eBPF	Extended Berkeley Packet Filter
FAIR	Factor Analysis of Information Risk
IRL	Individual Risk Level
PBAC	Policy-based Access Control
PID	Process ID
PRL	Propagated Risk Level
RA	Risk Assessment
RMF	Risk Management Framework
RM	Resource Manager
TEF	Threat Event Frequency
TPM	Trusted Platform Module
VFS	Virtual File System
WP	Work Package

References

[1] Martín Abadi, Mihai Budiu, Úlfar Erlingsson, and Jay Ligatti. Control-flow integrity principles, implementations, and applications. *ACM Trans. Inf. Syst. Secur.*, 13(1):4:1–4:40, November 2009.

[2] Tigist Abera, N. Asokan, Lucas Davi, Jan-Erik Ekberg, Thomas Nyman, Andrew Paverd, Ahmad-Reza Sadeghi, and Gene Tsudik. C-flat: Control-flow attestation for embedded systems software. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 743–754, 2016.

[3] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. Sana: Secure and scalable aggregate network attestation. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 731–742, 2016.

[4] N. Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. Seda: Scalable embedded device attestation. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, pages 964–975, 2015.

[5] Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. Non-control-data attacks are realistic threats. In *Proceedings of the 14th Conference on USENIX Security Symposium - Volume 14*, SSYM'05, pages 12–12, 2005.

[6] The FutureTPM Consortium. FutureTPM use cases and system requirements. Deliverable D1.1, 2018.

[7] The FutureTPM Consortium. Futuretpm risk assessment framework - first release. Deliverable D4.2, 2019.

[8] The FutureTPM Consortium. Threat modelling & risk assessment methodology. Deliverable D4.1, February 2019.

[9] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 483–486. ACM, 2017.

[10] Ghada Dessouky, Tigist Abera, Ahmad Ibrahim, and Ahmad-Reza Sadeghi. Litehax: Lightweight hardware-assisted attestation of program execution. In *Proceedings of the International Conference on Computer-Aided Design*, pages 106:1–106:8, 2018.

[11] Ghada Dessouky, Shaza Zeitouni, Thomas Nyman, Andrew Paverd, Lucas Davi, Patrick Koeberl, N. Asokan, and Ahmad-Reza Sadeghi. Lo-fat: Low-overhead control flow attestation in hardware. In *Proceedings of the 54th Annual Design Automation Conference*, pages 24:1–24:6, 2017.

[12] Trusted Computing Group. Tcg trusted network communications, federated tnc. Deliverable, 2009.

[13] Trusted Computing Group. Trusted network connect (tnc). Deliverable, 2011.

[14] H. Hu, S. Shinde, S. Adrian, Z. Chua, P. Saxena, and Z. Liang. Data-oriented programming: On the expressiveness of non-control data attacks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 969–986, may 2016.

[15] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. Darpa: Device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security Privacy in Wireless and Mobile Networks*, pages 171–182, 2016.

[16] jBoss. In *Drools Expert User Guide*, 2011.

[17] Florian Kohnhäuser, Niklas Büscher, Sebastian Gabmeyer, and Stefan 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*, pages 75–86, 2017.

[18] Andrew P Kosoresow and SA Hofmeyer. Intrusion detection via system call traces. *IEEE software*, 14(5):35–42, 1997.

[19] Sean Kramer, Zhiming Zhang, Jaya Dofe, and Qiaoyan Yu. Mitigating control flow attacks in embedded systems with novel built-in secure register bank. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 483–486. ACM, 2017.

[20] JongHyup Lee. Collective attestation for manageable iot environments. *Applied Sciences*, 8(12), 2018.

[21] Jonathan Mace, Ryan Roelke, and Rodrigo Fonseca. Pivot tracing: Dynamic causal monitoring for distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 35(4):11, 2018.

[22] Federico Maggi, Matteo Matteucci, and Stefano Zanero. Detecting intrusions through system call sequence and argument analysis. *IEEE Transactions on Dependable and Secure Computing*, 7(4):381–395, 2008.

[23] Vara Prasad and Jim Keniston. Locating system problems using dynamic instrumentation. 2010.

[24] Mehryar Rahmatian, Hessam Kooti, Ian G Harris, and Elaheh Bozorgzadeh. Hardware-assisted detection of malicious software in embedded systems. *IEEE Embedded Systems Letters*, 4(4):94–97, 2012.

[25] Debapriya Basu Roy, Manaar Alam, Sarani Bhattacharya, Vidya Govindan, Francesco Regazzoni, Rajat Subhra Chakraborty, and Debdeep Mukhopadhyay. Customized instructions for protection against memory integrity attacks. *IEEE Embedded Systems Letters*, 10(3):91–94, 2018.

[26] Hovav Shacham. The geometry of innocent flesh on the bone: Return-into-libc without function calls (on the x86). In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 552–561. ACM, 2007.

[27] Suchakrapani Datt Sharma and Michel Dagenais. Enhanced userspace and in-kernel trace filtering for production systems. *Journal of Computer Science and Technology*, 31(6):1161–1178, 2016.

[28] Darius-Andrei Suciu and Radu Sion. Droidsentry: Efficient code integrity and control flow verification on trustzone devices. In *2017 21st International Conference on Control Systems and Computer Science (CSCS)*, pages 156–158. IEEE, 2017.

[29] Flavio Toffalini, Andrea Biondo, Eleonora Losiuouk, Jianying Zhou, and Mauro Conti. Scarr: A novel scalable runtime remote attestation. 2018.

[30] Yubin Xia, Yutao Liu, Haibo Chen, and Binyu Zang. Cfimon: Detecting violation of control flow integrity using performance counters. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–12, 2012.

[31] Tao Zhang, Xiaotong Zhuang, Santosh Pande, and Wenke Lee. Anomalous path detection with hardware support. In *Proceedings of the Int. Conf. on Compilers, architectures and synthesis for embedded systems*, 2005.