Towards Auditing of Control-Flow Integrity

Luke Atherton - Supervised by: Konstantinos Markantonakis

Information Security Group, Smart Card and IoT Security Center
Royal Holloway, University of London

Objectives

- Investigate existing methods providing control-flow integrity;
- Propose a solution for enabling the audit of control-flow integrity.

Introduction

- Control-flow integrity (CFI) is a useful measure of secure software execution;
- When using control-flow integrity as a policy it states that the execution flow of an application must follow the control-flow graph generated from the application;
- The problem of enforcing control-flow integrity can be approached from three different directions: *prevention*, *detection* and *attestation*;
- In this paper, we intend to add a fourth method of enforcing CFI - *audit*. We will propose a solution which enables the tracking and storing of control-flow data in audit-friendly reports.

Control-Flow Graphs

Control-flow graphs (CFG) are a method used to formally describe the legitimate paths an application can take during execution. A simple measure of control-flow integrity is to verify whether instructions are processed in an order which abides by the application's CFG.
CFGs consist of vertices - *basic blocks* (BBL), and edges - *transitions*. Basic blocks represent a sequence of instructions which will always run from beginning to end.
Transitions can take several forms including direct branches, where *jump* destinations are hard-coded and indirect branches, where the destination is determined at run-time (the destination is contained within a register or memory address).

CFG Violation

CFG violation occurs when an attacker manipulates the execution so that it does not follow any legitimate routes described in the application's CFG. Taking the CFG shown in Figure 1 which shows the legitimate flows of $BBL\ 1 \rightarrow BBL\ 2 \rightarrow BBL\ 4 \rightarrow BBL5$ and $BBL\ 1 \rightarrow BBL\ 3 \rightarrow BBL\ 5$. An example of a path with violates the CFG is $BBL\ 1 \rightarrow BBL\ 2 \rightarrow BBL\ 3 \rightarrow BBL\ 5$, where the transition $BBL\ 2 \rightarrow BBL\ 3$ is unauthorised.

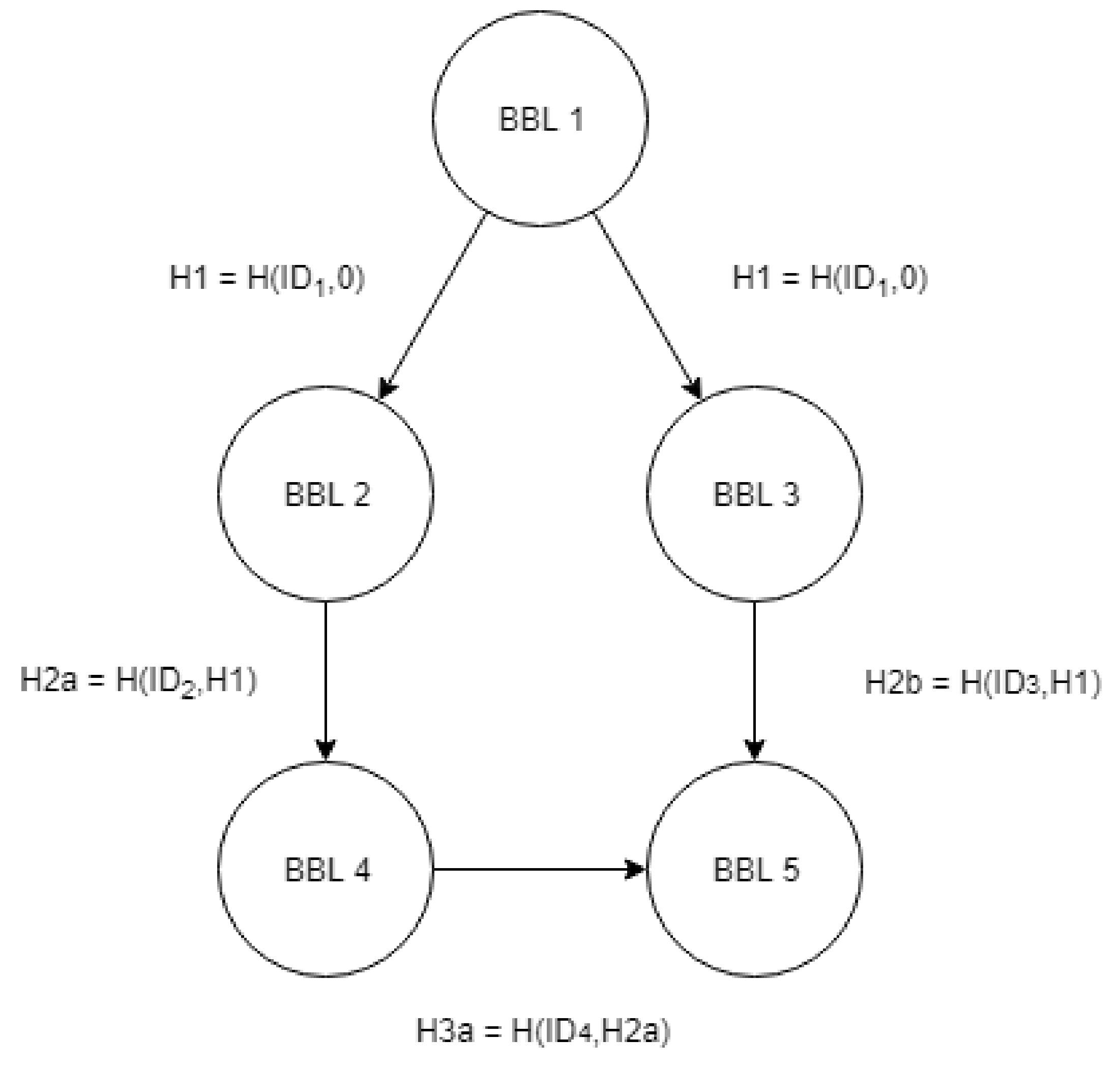


Figure 1:CFG hash

Control-Flow Integrity

Control-flow integrity can be provided using several methods:
Prevention Theoretically CFI could not be compromised. E.g. Read $\oplus$ Write or deterministically encrypted instructions [1].
Detection CFI compromise is detected during execution. E.g. stack canaries or shadow stacks [2].
Attestation Variation of solutions have been examined, where control-flow is tracked at real time and used in an attestation protocol.

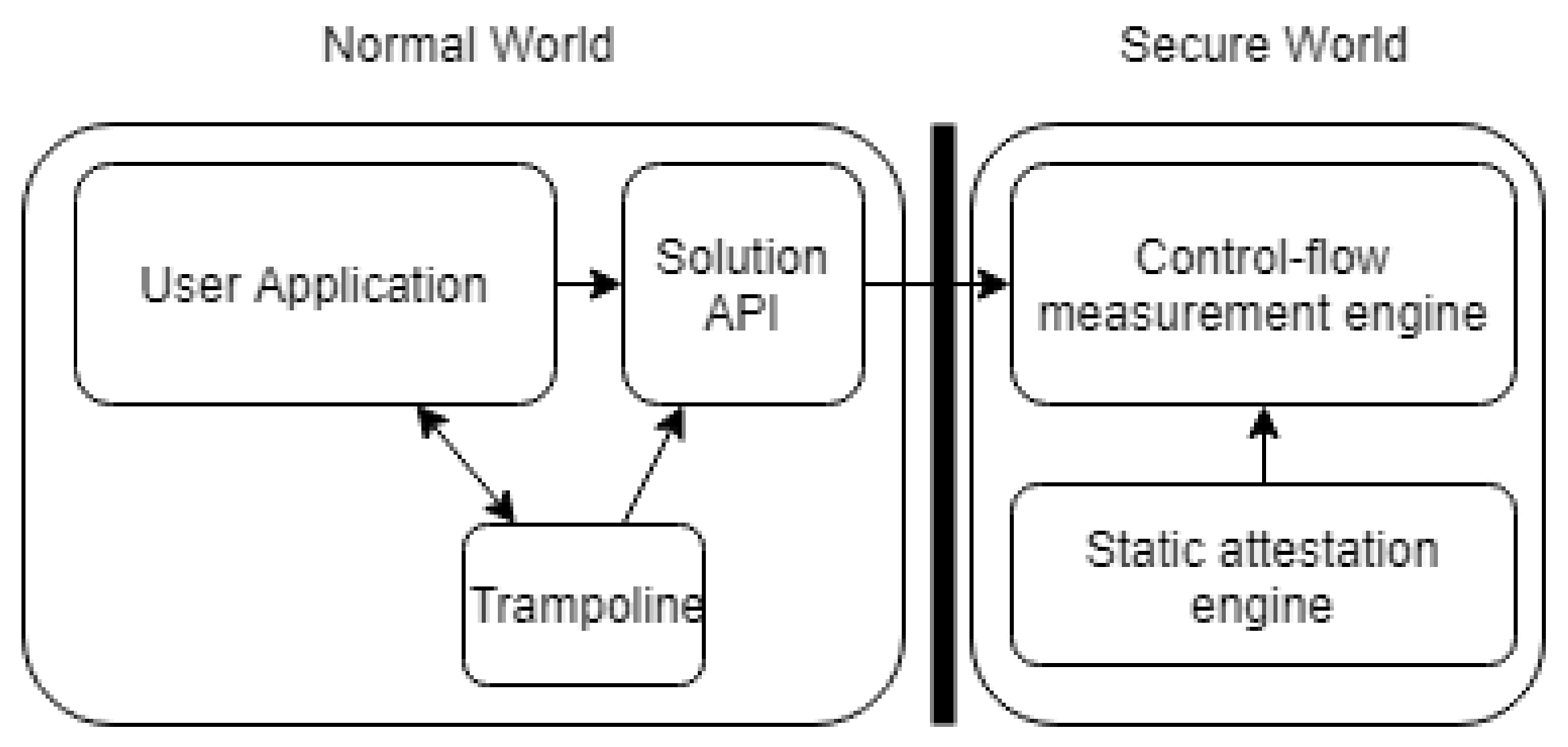
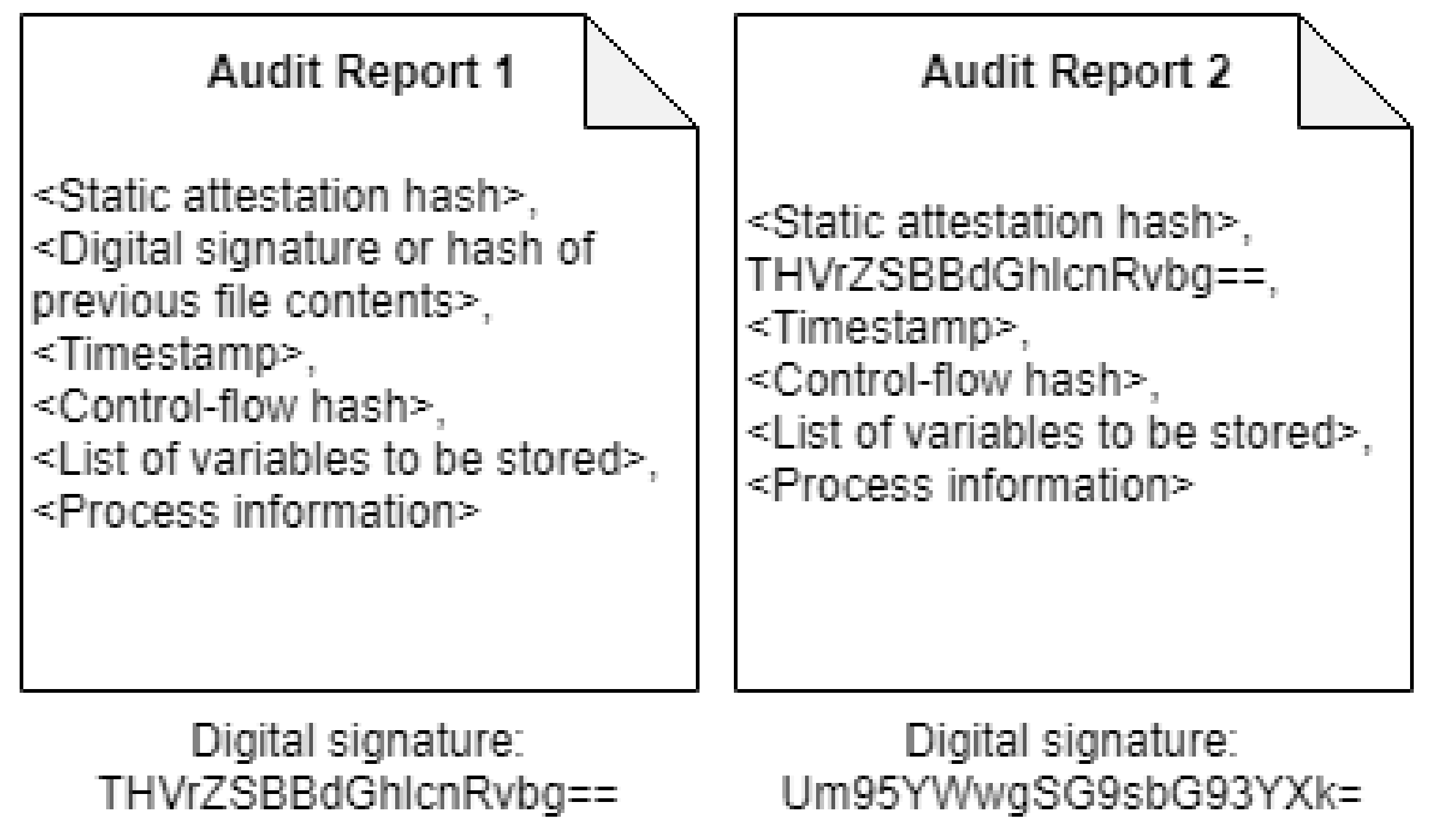


Figure 2:ARM TrustZone Implementation

Control-flow Monitoring

The paper proposes a solution which enables control-flow monitoring utilising functionality provided by ARM TrustZone. Figure 2 shows the control-flow measurement engine which is placed within the *secure world*, with user applications existing in *normal world* having been instrumented to execute via "tramplolines", as used in [3]. Trampolines inform the measurement engine of each transition made.
A transition triggers the measurement engine to create a hash representing the control-flow taken, as shown in Figure 1.



Audit Report 1

<Static attestation hash>,
<Digital signature or hash of previous file contents>,
<Timestamp>,
<Control-flow hash>,
<List of variables to be stored>,
<Process information>

Digital signature:
THVrZSBBdGhlcnRvbg==

Audit Report 2

<Static attestation hash>,
THVrZSBBdGhlcnRvbg==,
<Timestamp>,
<Control-flow hash>,
<List of variables to be stored>,
<Process information>

Digital signature:
Um95YWwgSG9sbG93YXk=

Figure 3:Audit files

Audit Files

As well as containing the control-flow hash, the audit files will contain an initial attestation report; the digital signature of the previous report and operating environment information such as important variables and currently running processes.

Benefits

The proposed solution enables the a new approach to control-flow integrity within embedded systems, providing:

- Historic evidence of control-flow;
- Binding of variables to a control-flow snapshot;

Conclusion

The proposed solution enables the tracing and recording of control-flow through the re-writing of binaries to divert execution to a monitoring engine. Further suggested work includes implementation of the solution, both in its current form and using hardware. The use of CFGs should be compared with other techniques such as generalized path signature analysis (GPSA).

References

[1] Robert P Lee, Konstantinos Markantonakis, and Raja Naeem Akram. Ensuring Secure Application Execution and Platform-Specific Execution in Embedded Devices. *ACM Transactions on Embedded Computing Systems*, 18(3):1–21, apr 2019.

[2] Nick Christoulakis, George Christou, Elias Athanasopoulos, and Sotiris Ioannidis. HCFI. In *Proceedings of the Sixth ACM on Conference on Data and Application Security and Privacy - CODASPY '16*, pages 38–49, New York, New York, USA, 2016. ACM Press.

[3] 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 - CCS'16*, pages 743–754, New York, New York, USA, 2016. ACM Press.

Contact Information

- Web: https://scc.rhul.ac.uk/
- Email: luke.atherton.2018@live.rhul.ac.uk
- Twitter: @LucialHz