Non-intrusive Runtime Verification of Embedded Software

Inês Gouveia and José Rufino

LaSIGE, Faculdade de Ciencias, Universidade de Lisboa, Portugal,
igouveia@lasige.di.fc.ul.pt, jmrufino@ciencias.ulisboa.pt

Abstract. The increasing development in cyber-physical systems (CPS) has lead to new concerns regarding performance, safety and security, while ensuring timeliness requisites are met. Conventional runtime verification approaches tend to resort to code instrumentation among other intrusive methodologies, affecting the CPS timing characteristics. As such, by taking advantage of reconfigurable logic technologies, we introduce a hardware-based non-intrusive observer entity that is configurable by means of observation points of interest extracted directly from the CPS software components binary code.

Keywords: Runtime Verification, Embedded Software, Operating Systems, Real-Time Embedded Systems

1 Non-intrusive Runtime Verification

Due to their criticality, the software components executing in embedded system platforms, the latter usually consisting of soft-processors deployed in **F**ield **P**rogrammable **G**ate **A**rray (FPGA) devices, depend on constant correct operation. In order to ensure the system specification is strictly followed and that no deviation takes place, runtime verification becomes crucial. However, existing approaches have been embracing techniques such as function call interception and source code annotation to observe the desired points of interest, downgrading performance and other timing characteristics, such as timeliness, by interfering with the normal system functioning and possibly affecting safety along the way.

1.1 TSP Systems' Safety and Security

The increasing systems' complexity has lead to the definition of Time and Space Partitioned (TSP) environments for the execution of applications, with the intent of preserving their timeliness and safety requirements. This compartmentalization implies function separation and that execution of an application in one partition does not affect others with respect to provision of timing guarantees.

Generally, different Operating Systems are deployed in each partition, including Real-Time Operating Systems (RTOSs), for critical functions and generic OSs, for non-critical best-effort functions. Despite the many useful features

present in generic OSs, some timeliness and safety limitations come associated with their use, along with with severe security related issues. Common vulnerabilities include buffer overflows, signedness errors, format strings and code injection. Our purpose is to contribute to eliminate or at least attenuate the impact of some of those vulnerabilities.

Making use of non-intrusive and self-contained techniques, it is the observer's duty to detect errors related to a relevant set of the vulnerabilities enumerated above, as well as other possible violations to the system, namely in the timeliness domain, and act accordingly.

1.2 Observer Implementation

The idea behind the observer is to check predetermined points of interest against a relevant set of invariants in runtime, so that correct behavior can be ensured. Some of these observation points can be extracted at design time, from the binary code, by standard and/or using special-purpose tools, and configured in the observer unit. For dynamically generated information, such as local variables, other methods must be employed, such as non-intrusive monitoring of the stack, where each variable's position is characterized by its offset w.r.t. the stack frame pointer. The observer configuration may be changed in runtime, allowing the addition and/or removal of relevant points of interest, thus allowing its dynamic tuning in terms of the nature and types of events to be monitored.

While the observer entity itself would be responsible for capturing events that match its chosen configuration, an independent monitoring infrastructure should, afterwards, analyze them in order to detect eventual specification violations and issue an alarm signal to initiate a recovery response, if possible, or promote a transition to a safe operating mode, otherwise.

Aiming to provide the stated non-intrusiveness and to be easily integrated in reconfigurable processing cores, the observer will be designed in VHDL, a hardware description language used to describe digital systems such as FPGAs, using a Xilinx XUPV5-LX110T development board as a proof of concept prototype.

In sum, our purpose is, thus, to design a configurable non-intrusive observer to perform runtime verification in cyber-physical systems, while taking into consideration common OS vulnerabilities, aiming to ensure both safety and security properties, while respecting timeliness guarantees.

References

1. Edwards, A., Jaeger, T., Zhang, X.: Runtime verification of authorization hook placement for the Linux security modules framework. In: Proc. of the 9th Conf. on Computer and Communications Security. ACM, Washington, DC, USA (Nov 2002)
2. Pinto, R.C., Rufino, J.: Towards non-invasive run-time verification of real-time systems. In: 26th Euromicro Conf. on Real-Time Systems - WIP Session. pp. 25–28. Madrid, Spain (Jul 2014)
3. Watterson, C., Heffernan, D.: Runtime verification and monitoring of embedded systems. Software, IET 1(5), 172–179 (October 2007)

Non-intrusive Runtime Verification of Embedded Software

Inês Gouveia and José Rufino

Ciências ULisboa
Faculdade de Ciências da Universidade de Lisboa

Summary

Software components executing in embedded system platforms (usually soft-processors deployed in FPGA (Field-Programmable Gate Array) devices) depend on constant correct operation, due to their **criticality**. In order to ensure no deviations take place, **runtime verification** becomes crucial.

However, conventional runtime verification approaches tend to resort to **code instrumentation** among other intrusive methodologies, affecting the CPS timing characteristics, and **possibly affecting safety along the way**.

By taking advantage of **reconfigurable logic** technologies, we introduce a **hardware-based non-intrusive observer** entity, configurable by means of observation points of interest extracted directly from the CPS software components' binary code.

TSP Systems' Safety and Security

The increasing systems' complexity has lead to the definition of **Time and Space Partitioned** (TSP) environments for the execution of applications, with the intent of **preserving their timeliness and safety requirements**. This compartmentalization implies function separation and that execution of an application in one partition does not affect others with respect to provision of timing guarantees.

Different Operating Systems are usually deployed in each partition, including Real-Time Operating Systems (RTOSs), for critical functions and generic OSs, for non-critical best-effort functions. While generic OSs' features are useful, some **timeliness and safety limitations** come associated with their use, along with **severe security related issues**.

Common vulnerabilities include **buffer overflows**, **signedness errors**, **format strings** and **code injection**. Our purpose is to contribute to eliminate or at least attenuate the impact of some of those vulnerabilities.

It is the observer's duty to detect errors related to a relevant set of the vulnerabilities enumerated above, as well as other possible violations to the system, namely in the timeliness domain, and act accordingly.

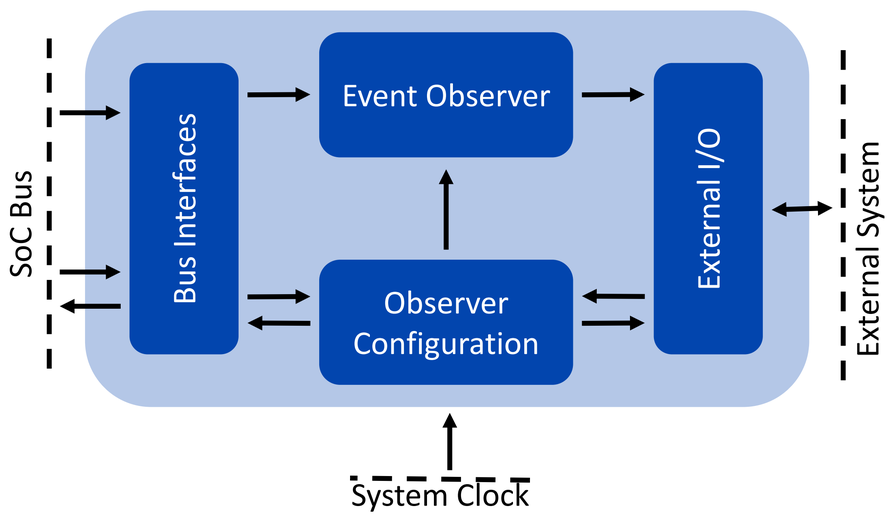

Fig. 1 – Observer Entity Architecture

Observer Implementation

The idea behind the observer is to **check predetermined points of interest** against a set of invariants in runtime. Some of these observation points can be extracted at design time, from the binary code, by standard and/or special-purpose tools. For dynamically generated information, such as local variables, other methods must be employed, such as **non-intrusive monitoring of the stack**, where each variable's position is characterized by its offset w.r.t. the stack frame pointer.

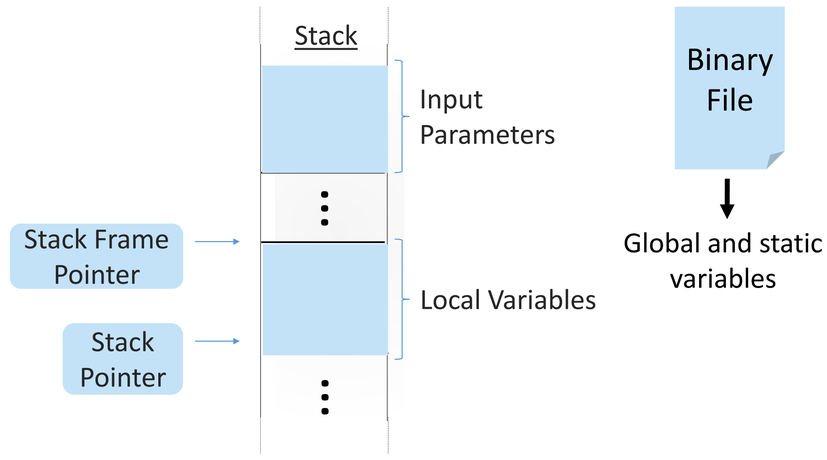

Fig. 2 – Observation Points' Location

The observer configuration may be changed in runtime, allowing the addition and/or removal of relevant points of interest, thus allowing its dynamic tuning. While the observer entity itself would be responsible for capturing events that match its chosen configuration, an independent infrastructure should, afterwards, analyse them in order to detect eventual specification violations and issue an alarm signal to initiate a recovery response or promote a transition to a safe operating mode.

Aiming to provide the stated non-intrusiveness and to be easily integrated in **reconfigurable processing cores**, the observer will be designed in VHDL, a hardware description language used to describe digital systems such as FPGAs, using a **Xilinx XUPV5-LX110T development board as a proof of concept prototype.**

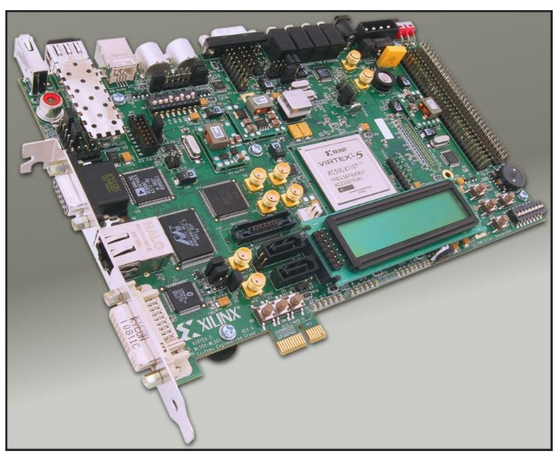

Fig. 3 – Xilinx XUPV5-LX110T board

In sum, our purpose is, thus, to design a configurable non-intrusive observer to perform runtime verification in cyber-physical systems, while taking into consideration common OS vulnerabilities, aiming to ensure both safety and security properties, while respecting timeliness guarantees.

References

1. Edwards, A., Jaeger, T., Zhang, X.: Runtime verification of authorization hook placement for the Linux security modules framework. In: Proc. of the 9th Conf. on Computer and Communications Security. ACM, Washington, DC, USA (Nov 2002)
2. Pinto, R.C., Rufino, J.: Towards non-invasive run-time verification of real-time systems. In: 26th Euromicro Conf. on Real-Time Systems - WIP Session. pp. 25–28. Madrid, Spain (Jul 2014)
3. Watterson, C., Heffernan, D.: Runtime verification and monitoring of embedded systems. Software, IET 1(5), 172–179 (October 2007)

LaSIGE, Faculdade de Ciencias, Universidade de Lisboa, Portugal,
igouveia@lasige.di.fc.ul.pt, jmrufino@ciencias.ulisboa.pt