A Tool for Monitoring and Maintaining System Trustworthiness at Runtime

Abigail Goldsteen[1], Micha Moffie[1], Torsten Bandyszak[2], Nazila Gol Mohammadi[2], Xiaoyu Chen[3], Symeon Meichanetzoglou[4], Sotiris Ioannidis[4], Panos Chatziadam[4]

[1]IBM Research - Haifa, Israel
{abigailt,moffie}@il.ibm.com
[2]paluno - The Ruhr Institute for Software Technology, University of Duisburg-Essen, Germany
{torsten.bandyszak,nazila.golmohammadi}@paluno.uni-due.de
[3]It-Innovation Center, University of Southampton, UK
wxc@it-innovation.soton.ac.uk
[4]Foundation for Research and Technology Hellas, Greece
{simosme,sotiris,panosc}@ics.forth.gr

Abstract. Trustworthiness of software systems is a key factor in their acceptance and effectiveness. This is especially the case for cyber-physical systems, where incorrect or even sub-optimal functioning of the system may have detrimental effects. In addition to designing systems with trustworthiness in mind, monitoring and maintaining trustworthiness at runtime is critical to identify issues that could negatively affect a system's trustworthiness. In this paper, we present a fully operational tool for system trustworthiness maintenance, covering a comprehensive set of quality attributes. It automatically detects, and in some cases mitigates, trustworthiness threatening events. The use of such a tool can enable complex software systems to support runtime adaptation and self-healing, thus reducing the overall upkeep cost and complexity.

Keywords: Trustworthiness, runtime, monitoring, mitigation, adaptation.

1 Introduction

Cyber-physical systems (CPS) are highly-connected, distributed, software-intensive systems that interact with other software as well as a multitude of physical entities. Trustworthiness of CPS is a key factor in their effectiveness. We define trustworthiness as the assurance that a system will perform as expected, or meet certain requirements, as defined by trustworthiness attributes. Different trustworthiness attributes should be considered and not only those related to security or reliability. In addition, different systems may have different requirements with regard to these attributes. The target trustworthiness may be derived from system requirements or service level agreements (SLA).

In addition to designing systems with trustworthiness in mind, monitoring and maintaining trustworthiness at runtime is critical to identify issues that could negatively affect a system's trustworthiness. These could stem from system failures, security attacks or even changes in the system's context. The relevant attributes should be measured and accounted for at all phases of the software lifecycle and corrective actions taken if they are violated.

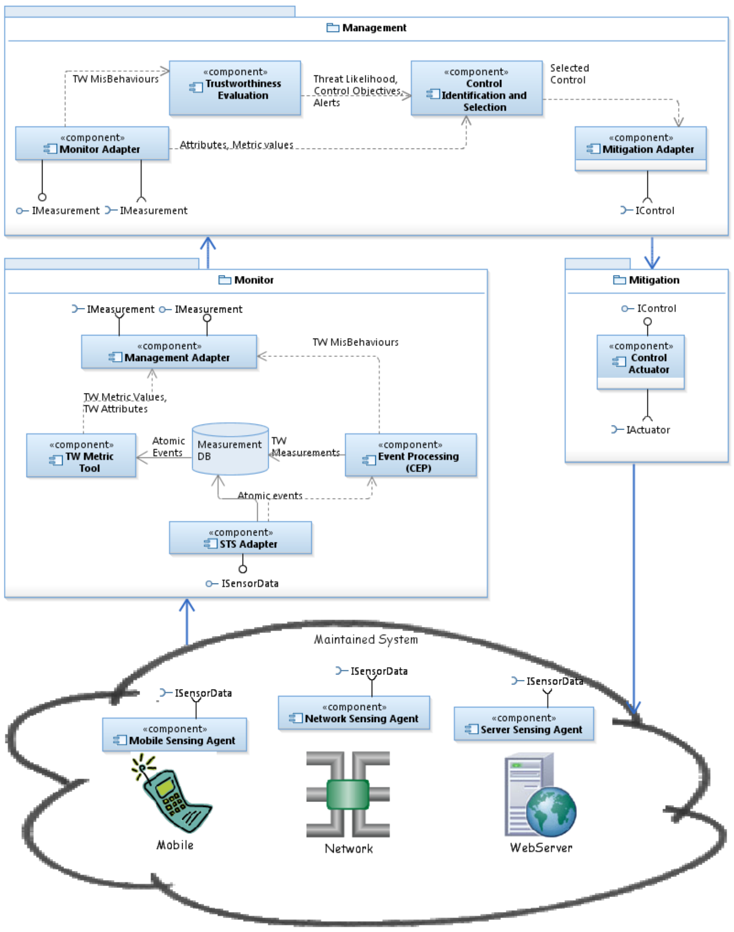
Existing solutions usually monitor a subset of trustworthiness characteristics, or do not propose any mitigating actions that can be performed to alleviate a problem once it has been identified. In this paper, we build upon previous work [1] and present a fully operational tool for system trustworthiness maintenance. The tool covers a comprehensive set of trustworthiness attributes [2], based on a generic ontology suitable for different kinds of software systems. In addition it envelops a complete maintenance cycle starting from raw events collected from the system, through identification of possible threats to the system's trustworthiness, to automatic mitigation of these threats and verification that the issue was corrected. In some cases this whole flow, including execution of controls, can be fully automatic. The use of such a tool enables complex software systems to support runtime adaptation and self-healing with respect to trustworthiness, achieved by automatic identification and mitigation of problems in real-time. This self-adaptivity to both external and internal changes in the system enables reducing the overall upkeep cost and complexity once the system is operational, since less manual intervention is required.

2 Tool Architecture and Implementation

The tool's architecture (see Figure 1) is based on the concept of autonomic computing and the MAPE-K reference model [3]. It includes a ***Monitor*** component responsible for collecting events from the system assets, storing them in a database and performing initial processing to compute trustworthiness metrics and misbehaviours; a ***Management*** component that receives alerts from the Monitor, identifies relevant threats and controls and selects the best controls to deploy; and a ***Mitigation*** component that actually executes the selected controls.

Events may be sent to the Monitor in several ways. One option is to pre-configure the monitored system to send such events to the maintenance tool. This requires planning the support for runtime maintenance into the system design so that observation and control interfaces are built into the system. More details on the required system analysis at design time, including the identification of relevant threats and controls, the identification of measureable system properties, and the design of respective interfaces can be found in [5]. Another option is to use specialized sensors that collect events in a specific environment (such as a mobile device) or a generic monitoring framework such as Zabbix [4]. The latter option also enables monitoring existing systems and applications.

The main processing sub-component within the Monitor is a ***Complex Event Processor*** (CEP) which receives all low-level events and fires a misbehaviour event whenever a single measurement reaches its pre-defined threshold or some more complex rule is triggered. The CEP depends on system-specific configuration that is based on observable system properties.



Management
TW MisBehaviours
«component» Trustworthiness Evaluation
Threat Likelihood, Control Objectives, Alerts
«component» Control Identification and Selection
Selected Control
«component» Monitor Adapter
Attributes, Metric values
«component» Mitigation Adapter
IMeasurement
IMeasurement
IControl
Monitor
IMeasurement
IMeasurement
«component» Management Adapter
TW MisBehaviours
TW Metric Values, TW Attributes
«component» TW Metric Tool
Atomic Events
Measurement DB
TW Measurements
«component» Event Processing (CEP)
Atomic events
«component» STS Adapter
ISensorData
Mitigation
IControl
«component» Control Actuator
IActuator
Maintained System
ISensorData
ISensorData
ISensorData
«component» Mobile Sensing Agent
«component» Network Sensing Agent
«component» Server Sensing Agent
Mobile
Network
WebServer

Fig. 1. Low-level Architecture of the Maintenance Tool

The main analysis in the Management is performed by a ***Trustworthiness Evaluator*** (TWE) which maintains and incrementally updates a semantic runtime model of the system. The model includes the different system assets and the connections between them, system threats and vulnerabilities, and controls that have been deployed. This model comprises not only software assets, but also physical assets, such as hardware and humans interacting with the system. The TWE utilizes machine reasoning based on a generic threat ontology that incorporates relevant security knowledge. This semantic model should encode as many common attack patterns as possible to be able to correctly identify threats to the system.

The TWE uses a Bayesian network approach to analyze threat activity likelihood given the reported system behaviour. Bayesian networks are a powerful tool for constructing models of phenomena involving uncertainty. Bayesian models can combine expert knowledge with observational data, and can be refined over time through learning from observation. In our case, we encode the expert knowledge (e.g., prior threat likelihoods, causational probabilities, etc.) in the model, and use runtime observed events as evidence to reason the threat activity likelihood and perform predictive analysis. As a result of system events, the TWE performs a comprehensive threat analysis, including: what are the possible threats given the observed system status, what are their likelihoods and are there threats that constitute a secondary effect due to the activity of other threats. Some more details about the TWE reasoning engine can be found in [1].

The trustworthiness evaluation process is triggered each time a meaningful system event is detected by the Monitor, including misbehaviours and system topology updates. Threats whose likelihood passes a pre-defined threshold are output, along with control options for mitigating each threat. Then the ***Control Identification and Selection*** component selects one or more controls to deploy, taking into consideration both cost and different trustworthiness metrics. More information about the trustworthiness-cost optimization problem can be found in [7]. The Mitigation, which encompasses system-specific information on how to execute each control, actually deploys the selected controls.

Each of the above components needs to be configured to support the specific system that will be monitored: the Monitor is configured to recognize the events arriving from the system (or monitoring framework) and derive metric values and misbehaviours; the TWE receives a model of the deployed system assets and how they are connected; and the Mitigation is configured with system-specific information on how to execute each control. The amount of monitoring information required from the system depends on the attack patterns we want to detect. The more complex the patterns, the more information will be needed from the system in order to detect them.

3 Initial Evaluation

In this section we show how the trustworthiness runtime maintenance tool was used to monitor select trustworthiness characteristics in a Distributed Attack Detection and Visualization (DADV) system, illustrated in Figure 2. The goal of this evaluation was to validate that our tool can in fact increase a system's trustworthiness.

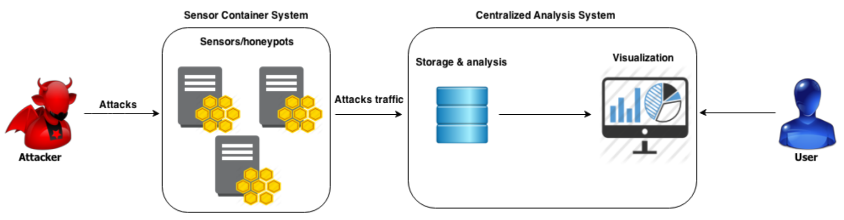


Fig. 2. DADV system architecture

The DADV sensors are low interaction honeypots that run as virtual machines (VMs) within a Sensor Container System (SCS), which is essentially a hypervisor that hosts and manages the VMs. The traffic collected by the sensors is forwarded to a Centralized Analysis System (CAS) for storage and analysis. Each sensor is configured to monitor a set of unused IP addresses in an organization's corporate network. The sensors run simulated vulnerable network services (e.g., SSH, HTTP) in a controlled environment (a sandbox within the VM) in order to attract attackers and gather information about zero-day attacks. However, a skilled attacker could evade the sandbox and take control of the sensor VM. A compromised sensor is a major concern as it poses a serious threat to the corporate network.

The runtime maintenance tool was used to monitor the health of the DADV sensors with the goal of detecting compromised sensors. Under normal conditions, the sensor VMs' resource load (CPU, memory consumption, etc.) is relatively low. Furthermore, sensors should not open network connections to other devices in the network. The existence of an outgoing network connection is indicative of a compromised sensor, which must be blacklisted and switched off immediately. A sensor with very high resource utilization is also suspected as compromised.

The sensors send health-statistic events to the Monitor every ten seconds. If an outgoing connection is initiated, the CEP triggers a *Promiscuous* misbehaviour, which is forwarded to the TWE. The TWE analysis outputs several relevant threats, of which the one with the highest likelihood is *Unauthorised Communications*, and proposes the *Blacklisting* control objective to mitigate it. Based on system-specific configuration, this is mapped to the DADV-specific control *Stop Sensor*, which is performed automatically by the Mitigation by calling an HTTP POST method of the SCS.

When a high load is detected on a sensor, the *Overloaded* misbehavior is triggered. This can lead to two possible threats: either the sensors are simply under a high load, which can be mitigated by adding an additional VM to the pool, or the sensor is under attack and should be blacklisted, similar to the above scenario. The likelihood of each threat occurring is computed in the TWE based on additional system behaviours.

We conducted a small-scale experiment with 27 live administrators. Each administrator used two alternative versions of the DADV system, one integrated with the maintenance tool and one not. Results showed that the integrated system was better at early detection and mitigation of compromised sensors (was able to mitigate 80% of the attacks vs. 63% in the regular system), and was found by administrators to be more trustworthy (91% of administrators preferred the integrated version over the regular one). This demonstrates that a system's trustworthiness, as well as users' perceived trust and acceptance of a system, can be substantially increased with the use of such a tool.

4 Summary

We demonstrated a tool for runtime monitoring of software systems, and specifically cyber-physical systems, to enable automatically detecting and mitigating events that may threaten the system's trustworthiness. The tool was tested and validated on two trustworthiness-critical applications: a Fall Management system for Ambient Assisted Living and a Distributed Attack Detection and Visualization system for Cyber-Crisis Management. An initial proof-of-concept was also performed on real sensor data from an electric company.

The tool covers a large range of trustworthiness metrics and can be adapted to many types of systems. It supports runtime adaptation and self-healing of critical systems, thus reducing the overall upkeep costs and complexity and increasing system uptake and retention. This approach requires detailed knowledge about the system to configure the different components, sensors able to accurately observe events that may affect trustworthiness, as well as "hooks" into the system to support automatic deployment of the mitigating actions.

Acknowledgements. This work was supported by the FP7 project OPTET (gr. no. 317631).

5 References

1. Gol Mohammadi, N., Bandyszak, T., Moffie, M., Chen, X., Weyer, T., Kalogiros, C., Nasser, B., Surridge, M.: Maintaining Trustworthiness of Socio-Technical Systems at Run-Time. In: 11th International Conference on Trust, Privacy & Security in Digital Business. LNCS, vol 8647, pp. 1-12. Springer, Heidelberg (2014)
2. Gol Mohammadi, N., Paulus, S., Bishr, M., Metzger, A., Könnecke, H., Hartenstein, S., Weyer, T., Pohl, K.: Trustworthiness Attributes and Metrics for Engineering Trusted Internet-Based Software Systems. In: Helfert, M., Desprez, F., Ferguson, D., Leymann, F. (eds) Cloud Computing and Services Science. CCIS, vol. 453, pp. 19–35. Springer, Heidelberg (2014)
3. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. In: IEEE Computer 36(1), pp. 41–50 (2003)
4. Zabbix, The Enterprise-class Monitoring Solution for Everyone, http://www.zabbix.com/
5. Bandyszak, T., Gol Mohammadi, N., Bishr, M., Goldsteen, A., Moffie, M., Nasser, B., Hartenstein, S., Meichanetzoglou, S.: Cyber-Physical Systems Design for Runtime Trustworthiness Maintenance Supported by Tools. In: 1st International Workshop on Requirements Engineering for Self-Adaptive and Cyber Physical Systems (2015).
6. Surridge, M., Nasser, B., Chen, X., Chakravarthy, A., Melas, P.: "Run-Time Risk Management in Adaptive ICT Systems," In: 8th International Conference on Availability, Reliability and Security (ARES), pp.102-110 (2013)
7. Kalogiros, C., Kanakakis, M.: "Profit-maximizing security level of ICT systems", To be published: 3rd International Conference on Human Aspects of Information Security, Privacy and Trust (2015)