communication channels and allows fine-grained supervision techniques to be applied to monitor an adversary's behaviour throughout his entire attack. Figure 1 illustrates the overall concept of the AutoHoney(I)IoT framework. As input, the framework requires the firmware of the device to be virtualised as well as a database containing real world device information of common microcontrollers and system on a chips (SoCs). The framework's intended functionality is to analyse a firmware dump and find a Qemu [3] appliance capable of executing the firmware. Furthermore, external peripherals (e.g., network access) are attached as needed.

The analysis is based on a two-step approach. The aim of the first step is to detect the architecture (e.g., ARM, MIPS, PowerPC, x86) of the firmware. The main idea of the second step is to apply fine-grained supervision techniques during the execution to identify the most appropriate processor. Therefore, the firmware is executed first on a generic processor corresponding to the identified architecture. Most likely this will result in an instruction mismatch, memory mismatch or internal peripheral mismatch, since there is a plethora of different embedded processors from various vendors with a wide variety of technical characteristics (e.g., central processing unit types, memory maps). During the emulation an algorithm attempts to identify an admissible embedded processor that is capable of executing the firmware dump. For this purpose, we create a database containing the real world technical characteristics of microcontrollers and SoCs from various vendors. The execution and fine-grained processor selection is repeated until a suitable processor is identified.

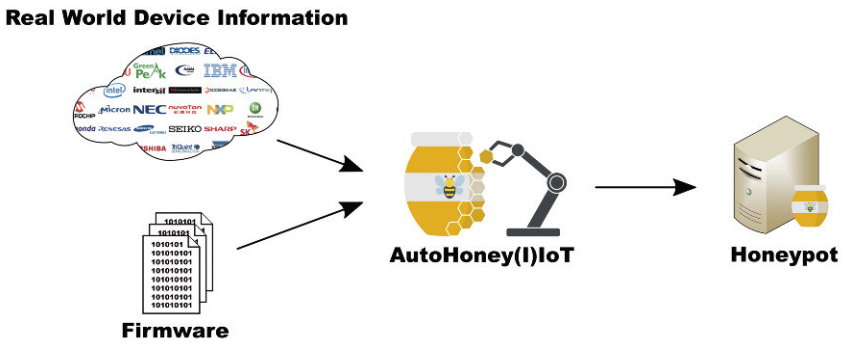

Figure 1: Overview of the AutoHoney(I)IoT framework.

Since a convincing (I)IoT honeypot needs to expose authentic system behaviour and communicate with the outside world, the AutoHoney(I)IoT framework will provide: (i) a method to attach communication interfaces to the emulated embedded processor and bridge them to physical as well as simulated hardware, and (ii) a method to attach custom external peripheral device models to a communication interface whose behaviour can be scripted in a simple way.

The FFG funded project started in July 2019 and is expected to conclude in December 2021. It is jointly realised by SBA Research, FH Technikum Wien, TU Wien and Trustworks GmbH, all situated in Vienna, Austria.

Link:
[L1] https://kwz.me/hEt

References:
[1] K. Angrishi: "Turning Internet of Things(IoT) into Internet of Vulnerabilities (IoV): IoT Botnets", arXiv Prepr. ArXiv1702.03681, 2017.
[2] A. Spognardi et al.: "Analysis of DDoS-Capable IoT Malwares", in Proc. of INSERT, 2017.
[3] F. Bellard: "QEMU, a fast and portable dynamic translator", USENIX Annual Technical Conference, FREENIX Track. Vol. 41. 2005.

Please contact:
Christian Kudera, Georg Merzdovnik, Edgar Weippl, SBA Research, Austria
ckudera@sba-research.org, ,
gmerzdovnik@sba-research.org,
eweippl@sba-research.org

Yogurt: A Programming Language for the Internet of Things (IoT)

by Ivan H. Gorbanov, Jack Jansen and Steven Pemberton (CWI)

As IoT moves from the hands of professionals and academics into those of the general consumers, it becomes increasingly important to provide them with the appropriate tools for interaction. Yogurt is a domain-specific programming language for IoT, designed to tackle the disparity between powerful but complex languages and user-friendly environments with restricted capabilities.

"Yogurt" is an object-oriented declarative programming language for the Internet of Things (IoT). It allows the end-user to program their entire IoT ecosystem through one environment by leveraging the capabilities of Igor [L1], an architecture for unified access to IoT [1]. The language offers high expressiveness by incorporating mechanisms from traditional programming paradigms. Furthermore, the underlying programming model adopts a metaphor close to the users' real-life experience, thus reducing the learning effort required to adopt the language.

Traditionally IoT devices are programmed through a variety of high-level languages. This, however, requires an in-depth knowledge of computer

programming which often takes years of training and practice to develop. In addition, the decentralised nature of the development of these technologies has led to the incorporation of many communication protocols and formal languages. Therefore, managing a complete system often requires knowledge of multiple programming environments. Both the consumer industry and academia have tried to address this problem by introducing systems which act as a central control point for the customer's IoT systems. Through these all of the user's devices can be programmed through one language. These programming facilities, however, are mostly optimised for usability in order to reduce the learning curve for inexperienced users. This, therefore, leaves a gap between highly expressive but difficult programming languages and user-friendly environments, which lack sufficient expressive power. Consequently, the main purpose of this work is twofold: to extend the work of Jansen and Pemberton [1] by providing a viable programming language for Igor, and to fill the abovementioned gap that currently exists in the programming facilities for IoT devices.

Yogurt's underlying programming model (Figure 1) achieves simplicity by providing abstractions analogous to the real world in order to reduce the gap between what the programmer is trying to achieve and how to achieve it. The "actor" abstraction represents all devices which would make up an IoT system. Each "actor" has a state, which represents what the device is doing in the real world and can perform "actions" which change that state. An "action" gets triggered by a change in the state of its own corresponding actor or that of another one. Conditions can also be added in the form of "guards" to allow the device's behaviour to vary based on its environment. This small set of general abstractions is at a high enough level so as to be easy to conceptualise. Furthermore, they are general enough to be applicable in the increasingly heterogenous collection of devices part of IoT.

The proposed model utilises several mechanisms from established programming paradigms to allow it to tackle the use cases that have come to be expected by users of this technology. The actor abstraction works as a class in object-oriented programming in order to allow the reuse of code, making writing Yogurt programs more efficient. Furthermore, it allows for the use of encapsulation and inheritance. This makes it easier to program more complex devices as a collection of simpler ones. In addition, by forcing the programmer to keep data and corresponding methods together, it is easier to keep track of dependencies and spot any conflicts that may arise from different methods trying to change the same data at the same time. Last but not least, the language is declarative, meaning that the user needs to specify what the result of the program needs to be rather than how exactly to achieve it. This feature brings two major advantages: it further reduces the learning curve as the programmer does not have to worry about concepts such as memory management and it makes the language context independent, meaning that a program which switches the lights on and off would remain the same regardless of the hardware devices used to implement the solution as the result will be the same.

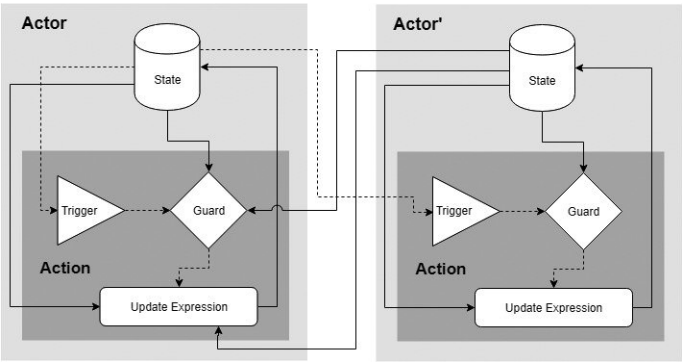

Figure 1: Yogurt programming model abstractions.

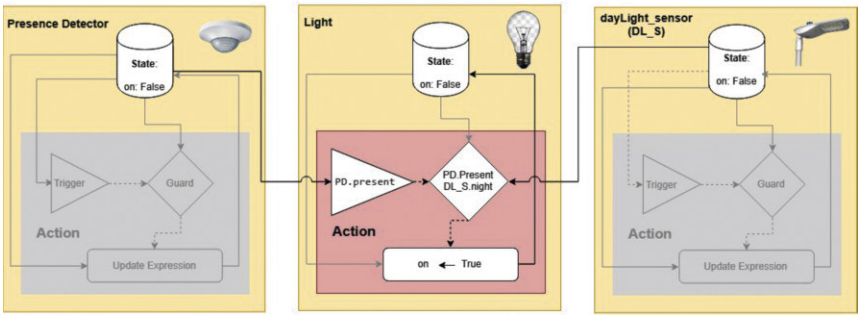

Figure 2: Light, presence sensor, daylight sensor example model.

The proposed textual representation of Yogurt uses human readable keywords to reduce the barrier to adoption and increase code readability. Here is a simple example of a light which turns on with a presence detector (PD) only when a day light sensor senses that it is dark outside:

```
on(PD.present):
        whenall(PD.present = True,
dayLight_sensor.night = True):
                on <- True
```

The current version has been tested using the Discount method for programming language evaluation [2]. Participants confirmed that the language was easy to pick and use because it allows them to think about the task in a way in which they would in the physical world, while suggesting changes that could be made to increase efficiency.

Links:

[L1] https://github.com/cwi-dis/igor

[L2] https://www.dis.cwi.nl/

References:

[1] J. Jansen, S. Pemberton: "An architecture for unified access to the internet of things", XML LONDON 2017 (2017).

[2] S. Kurtev, T. A. Christensen, B. Thomsen: "Discount method for programming language evaluation", in PLATEAU @ SPLASH. 1-8, 2016.

Please contact:

Jack Jansen
CWI, Netherlands
jack.jansen@cwi.nl