aSSIsT: Secure Software for the Internet of Things

Simon Duquennoy, **Bengt Jonsson**, Luca Mottola, Shahid Raza, Kostis Sagonas
Uppsala University and SICS

1 Introduction

Background The Internet of Things (IoT) is now becoming an integrated part of our society's infrastructure. The IoT includes a large number of small embedded devices that are connected to each other and the internet. IoT devices act as the interface between the physical and the digital world, enabling real-time insights into physical world dynamics. IoT systems are being deployed in a range of application domains, including factories, hospitals, as well as smart buildings and homes. Swedish industry and research has played a leading role in this development: IoT technology is an important offering of many Swedish companies, and several key technologies, such as Contiki, have a leading role in IoT system development.

As we become increasingly reliant on IoT systems to perform critical functions, it becomes apparent that security and safety concerns must be taken seriously. Compromised or faulty devices and systems can cause catastrophic damage to individuals, companies, and the society. IoT devices can be targeted by exploits that steal sensitive information and perform coordinated attacks. A number of such attacks has already occurred, including "Mirai", "Linux.Darlloz" and "Zigbee War-flying", and their number will certainly continue to increase with the deployment of new IoT systems (cf. Figure 1).

For mainstream computing systems, software security mechanisms are being developed and deployed since many years. Examples include techniques for preventing and detecting vulnerabilities in software [10, 16], techniques for detecting and mitigating malicious code, etc. Small embedded IoT devices, however, feature peculiar characteristics that render these techniques difficult to apply:

1. Software for IoT devices is chiefly designed to cope with constrained memory, power, processing, and bandwidth resources. This prevents developers from using sophisticated mechanisms for monitoring and intrusion detection; it also encourages programmers to produce highly optimized code, which is more susceptible to exhibit security bugs and makes their detection more difficult.
2. Software for IoT devices often has device-specific constructs, e.g., for accessing resources such as sensors, which are not handled by existing techniques for security analysis, and which vary between different devices.
3. Operating systems for IoT devices rarely provide features, such as memory protection and run-time component replacement [17, 14]. This makes it harder to prevent security breaches and limit their effect, and also entails that security patches must replace the whole binary image on a device, causing undesirable down-time.

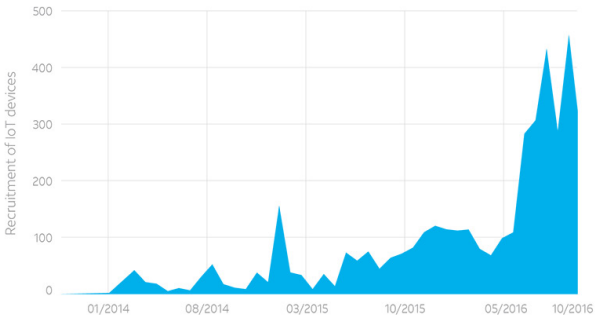

Figure 1: Progression of botnet activity originating from IoT devices, as recorded by AT&T over the past years (from https://www.business.att.com/cybersecurity/cybersecurity-innovation/).

Challenge The existing software infrastructure for IoT represents an impressive effort to equip small devices with advanced functionality. Now that the IoT revolution is here, society and industry face the challenge to use this software basis for building systems with the highest levels of security and safety.

Goal The goal of aSSIsT is to enable IoT system developers to meet this challenge, by providing them with powerful techniques that can detect and remove security vulnerabilities in software, ensure that security protocols provide guaranteed functionality, which prevent, detect, and mitigate intrusions when they occur. These techniques must furthermore be integrated in existing development process for IoT systems, by automating them and by focusing on low-power, battery-operated wireless devices, such as those found in IoT systems.

aSSIsT in a Nutshell aSSIsT will achieve these goals by advancing the state-of-the-art in security for small embedded devices, matching the above characteristics (1–3), along three complementary directions:(i) automated software analysis techniques will detect security problems in software, and guar-

antee their absence for specific components, (ii) automated testing techniques will check correctness of implementations of security services, and (iii) run-time protection mechanisms will provide safeguards so as to isolate software modules, detect and mitigate attacks, should attacks occur that could not be detected with other means.

2 Approach

aSSIsT will achieve its goals by advancing the stat-of-the-art along three directions of crucial importance for security of software, while also considering the specific characteristics (1–3) of IoT systems.

Analysis of Security of Software: aSSIsT will develop powerful techniques for analyzing software for IoT devices in order to detect security bugs, a.k.a. vulnerabilities, including buffer overflows, memory safety errors, information leakage, run-time errors, authentication bypasses, etc. We will leverage on the recent development of dynamic symbolic analysis techniques, which have been successfully used in automated analysis of security properties for mainstream computing systems (e.g., [22]), in order to develop such techniques for IoT software. For this, we must overcome key challenges, including taking into account specific peculiarities of IoT device platforms, handling complex software features such as interrupts, and developing techniques that allow dynamic analysis to provide guarantees of correctness for specific software components.

Testing & Verification of Protocol Implementations: aSSIsT will develop techniques for efficient automated testing of protocol implementations, in particular for ensuring that they provide security services such as authentication, secrecy, etc. We will build on recent advancements on automated techniques for testing implementations of key communication protocols [13, 27], and will leverage on our recent advances on techniques for directing automated testing [19] to search for violations of specific correctness properties. We will improve the power of such techniques so that they can provide security guarantees for IoT protocol implementations.

Run-time Protection Mechanisms: aSSIsT will design tools, compiler techniques, and run-time functionality that make it harder for attackers to compromise IoT devices, even if they manage to gain access in unintended ways, and for detecting and mitigating successful attacks. Such functionality includes component isolation, which limits the damage that can be incurred by intruders, and techniques for diversification of device software, e.g., by memory randomization, so that an attacker who finds a method for intrusion into one device cannot use the same method for the other ones. We will also contribute monitoring solutions based on an embedded supervisor and advanced event analysis. Only few efforts currently exist in this area, as one must overcome several challenges specific to IoT devices, including the lack of advanced mechanisms for managing memory, such as MMUs (memory management units), in addition to power and memory constraints. Such isolation will also help protect verified components in a code base from interference from untrusted parts, and will enable modular verification of software components.

The above directions of advancements are closely interlinked. Software analysis techniques will work in tandem with protocol testing: testing will check requirements whereas software analysis can provide coverage guarantees and direct testing towards potentially unsafe scenarios. Software analysis will assist in providing component isolation, by analyzing memory usage of concerned component; in return, component isolation will support modular analysis of individual software components.

Demonstrators Techniques developed by aSSIsT will be usable for any C-based software for IoT. We will demonstrate them through three demonstrators:

- *Contiki*, which is the leading open-source OS for IoT devices, and has a wide industry adoption,
- *DTLS* and *TSCH* are two of the most important protocols in the network stack for IoT.

Given their wide industry adoption, these demonstrators will showcase the application of aSSIsT techniques, thereby ensuring rapid industry uptake.

On the Scope of aSSIsT IoT system security must be approached with a holistic perspective, including physical threats to its components, management of access rights, etc. aSSIsT focuses on security mechanisms that can be mitigated on the software level. It can be argued that software vulnerabilities are the most important security threats, since undetected intrusions can cause potentially catastrophic harm to networks and also to entire society infrastructures; hence their detection and removal should be of utmost importance to any organization.

aSSIsT further aims to provide automated techniques that can be fitted into existing development flows in the IoT domain. There have been efforts to develop a software basis for IoT devices using "safe" programming languages, that support the provision of security guarantees, e.g., TinyOS, but so far this has not succeed in equipping small devices with sufficiently powerful IP-connectivity. Another, more drastic approach, would be to develop a software basis, including embedded OSes and network stacks, using techniques from the safety-critical domain, including similar procedures as in avionics, or supported by interactive theorem provers. However, this would be very costly, would take several years to become mature, and may result in a software basis which cannot be adapted to upcoming needs.

Relation to SSF call The proposed project addresses a core topic of the SSF call, which (in Section "Background") describes the security challenges that arise when traditional "best-effort" designs must scale to the needs of the digital society. The SSF call mentions *"anomaly detection and mitigation of security vulnerabilities"* as key topics. These are central concerns of aSSIsT, for the existing IoT software infrastructure. Moreover, aSSIsT techniques will be automated, and can thereby be adopted into existing development processes. Beyond IoT systems, the developed technologies will also be applicable in any sector where protocol correctness and software vulnerabilities are important, such as network servers, software for the powergrid, medical cyber-physical systems, etc.

3 Planned Work

WP1: Analysis of Security of Software
(B. Jonsson, K. Sagonas, S. Duquennoy)

IoT devices typically have a large attack surface, consisting of a wireless interface, and possibly interfaces for reboot and maintenance. Because of their mission-critical role, it is important to ensure the absence of security vulnerabilities that can make devices susceptible to buffer overflow attacks, authentication bypasses, etc., when exposed to some unexpected combination of inputs and environment events. The goal of WP1 is to develop techniques for detecting such problems, and if possible guarantee their absence, that are automated in order to fit into development processes for IoT systems.

For mainstream computing systems, the currently most powerful automated techniques for vulnerability detection are based on dynamic symbolic execution and various further developments, including concolic testing [24] and veritesting [4]. These techniques extend dynamic execution techniques by instrumentation that allows to systematically generate inputs that exercise a high percentage of feasible execution paths of the program code, while at the same time checking security properties using a dedicated runtime checker. For servers and PC-based systems, such security analysis techniques are now the most powerful ones used at, e.g., Microsoft [16, 23]. Tools based on similar technology achieved the top places in 2016 at DARPA's prestigious "cyber grand challenge" [10].

State of the Art Efforts to analyze software on small devices include FIE [12], which uses the symbolic execution engine KLEE [8] to find vulnerabilities in small embedded firmwares executing on the MSP430 family of micro-controllers. It achieved high coverage on small firmwares, but had significant problems to deal with larger codes, such as in Contiki. It is also not able to consider interrupts and event-driven execution in an efficient manner. A previous Contiki verification effort [28], using CBMC, did not manage to achieve high coverage.

Approach Our departure point is symbolic and concolic execution techniques as found, e.g., in the tool FIE [12] and in our own tool CUTER [15], for concolic testing of Erlang programs. We will develop such technology into powerful techniques for automated analysis of security properties of actual IoT device software. For this, we must overcome key challenges, including (i) software that is specialized for a small platform, having specific interactions with peripherals, (ii) the potential explosion in program paths incurred by frequent occurrence of interrupts, and possibly custom thread models, and (iii) designing techniques that provide security guarantees for key software components.

Task 1.1: Handling Device Platform Characteristics IoT device software includes platform specific constructs, e.g., for accessing peripherals, network interfaces, etc. These moves execution outside the domain of the standard C language, and can not be handled by plain symbolic analysis techniques. We will address this challenge by a combination of several techniques.(i) We will combine symbolic analysis with concrete test executions on a platform emulator. (ii) We will develop support for a user or device provider to model accesses to peripherals, such as reading sensor data from a designated memory location, within the C language in a structured way; this allows to apply the power of symbolic analysis. (iii) Leveraging our past work on model learning [9], we will develop learning techniques for inferring models of peripheral behavior by repeated test runs. The techniques will be demonstrated by providing support for analyzing software on the ARM Cortex M3, which is one of the most widespread platforms for IoT, and for which support will soon be available in the COOJA simulator.

Task 1.2: Handling Interrupts and Threading IoT device platforms typically coordinate I/O and overlapping activities by interrupts. Unfortunately, a faithful representation of interrupts may cause an explosion in the number of program paths that must be considered. An analogous challenge stems from various event- or thread-based execution models in embedded OSes. We will address these challenges, inspired by our recent advances in reduction techniques for multithreaded programs [2], which limit the analysis to a small number of representative paths, while still guaranteeing coverage of all paths. The result will allow complete handling of interrupts and event-driven execution with only modest overhead on the software analysis.

Task 1.3: Complete Analysis of Software Components For realistic software systems, symbolic-execution based verification techniques typically are

not able to analyze all possible execution paths, and consequently cannot give absolute guarantees about the absence of security bugs. We will develop approaches by which such guarantees can be given for modest-size components of the software on an IoT device. This will be done by augmenting symbolic execution with extrapolation techniques that can generalize the analysis of a small number of execution paths to a whole class of paths. Instances of this idea have appeared in some works, e.g., memory smudging [12], which we will generalize to become broadly applicable. We will apply these techniques to central components of an IoT OS. For instance, for Contiki, we will focus on its scheduler and memory allocator, so as to provide a verified trusted computing base. At run-time, we leverage on the protection mechanisms from WP3 to isolate the trusted base and components from untrusted code. This will allow other Contiki components, e.g., those involved in specific security protocols, to be verified in a modular manner.

WP2: Testing & Verification of Protocol Implementations (K. Sagonas, B. Jonsson, L. Mottola)

Designing and implementing secure communication protocols is an exceptionally hard problem. In the context of TLS alone, the Transport Layer Security protocol which is the de facto protocol standard for secured Internet and mobile applications, a number of attacks have been discovered in the recent years. They range from cryptographic attacks (e.g., problems when using RC4 and attacks such as FREAK), to serious implementation bugs such as Heartbleed, to timing attacks (e.g., Lucky Thirteen [3] and variations of Bleichenbacher attacks [27]).

Attacks against communication protocols specific to IoT systems are less studied. In addition to being susceptible to most security threats against the TCP/IP stack in a wired network, an IP-based IoT network possesses a number of additional vulnerabilities (e.g., wireless medium unreliability, spectrum use, power management, limited bandwidth, etc.), and involves dealing with network and physical layer issues, which make IoT-specific protocols more challenging to test and verify.

State of the Art For TCP/IP-based protocols, the large number of recent attacks has motivated researchers to provide further security analyses of protocols and incorporate them in tools. In recent scientific studies, authors have considered testing and verification of TLS state machines [6], have developed tools for testing protocol security properties when messages are sent in arbitrary orders, and even tools for modifying specific message fields [7].

Techniques like protocol state fuzzing [13] and systematic fuzzing [27] have also been considered recently. Many such techniques also involve a *learning* component that learns the states of the protocol or the format of messages to make the fuzzing process more powerful and automatic. Still, relatively little attention has been paid to security breaches that are specific to IoT systems and to protocols that are implemented using UDP instead of TCP.

Approach We will further develop lightweight black-box techniques for testing correctness and security properties of IoT protocol implementations. We have developed techniques for property-based testing (PBT) of sensor networks [20], support for testing to take place in COOJA (a cross-level sensor network simulator for Contiki-based IoT systems), and recently a general framework for *targeted property-based testing* [19], a powerful testing technique that extends the generation component of a PBT tool with a search-based testing component, thereby combining the advantages of the two techniques and allowing the testing process to be faster and significantly more effective. Most notably, it frees the developer from the burden of having to manually write generators tailored to each individual property.

We have already applied targeted PBT not only to test correctness but also non-functional properties (e.g., energy consumption), the technique is currently restricted only to situations where the unit under test is *stateless* and the property of interest involves a single IoT node rather than a whole network configuration.

Task 2.1: Targeted Property-Based Testing We will first extend targeted PBT to also work in situations where network-wide and/or *stateful* testing of IoT protocol implementations is needed. The latter mechanism is required for testing network security protocols, such as TLS or DTLS, which typically implement a state machine with different states for establishing connections. In doing so, we also plan to explore the applicability of domain-specific state abstraction techniques that project partners devised earlier for both the IoT [21] and other domains [5].

Task 2.2: Systematic Fuzzing Once the mechanisms for stateful testing are in place, the second task is to employ them to perform *property-based systematic fuzzing* of IoT protocols. We will address the challenge to learn relatively accurate state machine models of implementations of complex IoT protocols, such as DTLS, leveraging partners previous work on state machine learning [9]. We can thereafter address the challenge how to appropriately fuzz the state machine model, using search-based techniques, in order to discover violations of security properties effectively.

Task 2.3: Verification via Compositional Testing We will employ techniques such as *(stateless) model checking* and *compositional testing* to IoT protocol implementations, so that we provide stronger guarantees than property-based testing

and systematic fuzzing can give us. The goal here is to be able to *verify* that implementations indeed preserve important properties of their protocols, such as conformance to their standard. In this task, we plan to leverage on the fact that central components of the IoT system software will have already been completely analyzed by Task 1.3, which will allow us to employ a modular approach to verification via compositional testing techniques.

WP3: Run-time Protection Mechanisms

(S. Duquennoy, S. Raza, L. Mottola)

Because of size, cost, and energy constraints, IoT systems rarely provide advanced functionality to ensure the dependable operation of software components. Example of these are hardware Memory Management Units (MMUs, available on computer CPUs) or Address Space Layout Randomization (ASLR), designed for dynamic loading, in systems with MMUs. IoT OSes, therefore, provide no means to isolate and protect different software components from intrusion attempts. Malicious software components may, for example, deliberately corrupt a node's memory to gain control on a device.

Remedying these issues is not at all trivial. Runtime monitoring and mitigation mechanisms from the OS research are unsuitable in the IoT's radically different application model (dynamic loading vs integrated firmware) and underlying hardware (with limited memory management). What is required is to devise mechanisms able to strike a balance between the level of protection achieved and accommodating the resource constraints of IoT devices. The ideal solution would bear minimal impact on development process and run-time operation, but still ensure a level of protection and ability to monitor that is close to that of mainstream systems.

State of the Art For mainstream systems, the prevention of software attacks at runtime is a mature field, with established solutions such as address obfuscation and ASLR, which were recently proposed for embedded systems too [1]. These solutions, however, require availability of a hardware MMU. Other options have been conceptually proposed for small embedded systems [18], such as link-time reordering, stack shuffling, and system call diversification. Still, to achieve a practical design, a number of challenges remain to be addressed, such as adapting to custom threading models of IoT systems and keeping the energy overhead acceptable.

In place of an MMU, a Memory Protection Unit (MPU), which protects memory regions, may be available in low-power microcontrollers. Current IoT software, however, lacks MPU support, in part because segregating the kernel from user space proves challenging as a design afterthought. FreeRTOS does provide basic MPU support, but it is seldom used and poorly maintained, because developing for an MPU is difficult [11].

Attacks against a network node may be detected and countered using standard security components such as anti-virus, firewalls and local and network intrusion detection systems. Some IoT devices are powerful enough to use standard desktop/server software for this. In other cases the security software must be adapted to a resource constrained environment. For example, IDS and firewall solutions where designed for low-power IoT networks [26]. While this is a first step, more attacks have to be covered, potentially with distributed intelligence. Furthermore, there is a need for detection of malicious code in constrained devices, like the ones we consider.

Task 3.1: Memory Isolation An important step is to enable memory isolation, by means of compiler tools, OS mechanisms, and building on an MPU and/or Trusted Execution Environments (TEE). TEEs now equip low-power, MMU-less platforms (e.g., ARM TrustZone on the Cortex-M23), and provide hardware isolation of resources, including memory and bus transactions, interrupts and peripherals. We will tackle the challenge of achieving fine-grained, yet efficient protection. To this end, we will devise tools that infer properties on memory access patterns of different modules, and optimize memory assignment to the MPU's (sub-)regions.

To tackle the code base maintainability challenge with MPUs [11], we will provide the developer with means (e.g., through code annotations) to specify constraints such as privileges, security and sharing properties. We will incorporate TEE support in our mechanisms, to handle a truly isolated trusted kernel, and its connection to the untrusted world.

Task 3.2: Exploit Mitigation Building on our software isolation solutions, we will explore mitigation strategies for attacks such as buffer overflow, code injection, and return-to-libc. In particular, we will solve the challenge of achieving high entropy for memory randomization [25], by reasoning at the function or even sub-function level, rather than page or library level. Such approaches are practical in IoT OSes, where the code is considerably smaller than in conventional OSes, and where firmwares are integrated rather than dynamically loaded.

Address randomization could be done per-device, which prevents attacks to be generalized, but still does not protect single devices. We will therefore also explore boot-time randomization, so that a crashed device always reboots with a different memory layout. This will require solutions that preserve high entropy while keeping the procedure embeddable. Finally, we will design complementary solutions to achieve functionality such as stack shuffling, boot-time tampering detection, and return address protection, in constrained environments.

Table 1: The three demonstrators of aSSIsT.

Contiki	The Contiki OS will operate as a testbed for the memory protection and monitoring techniques of **WP3**. Such techniques will be OS-agnostic on a conceptual level, fostering their general applicability independent of the specific software platform. On the other hand, using Contiki as their demonstrator bears key advantages: (i) as it is written in pure C, Contiki provides a quick path to a working implementation by avoiding the idiosyncrasies of many IoT OSes that employ dialects of C, and (ii) because of its wide industry adoption, a Contiki-specific implementation provides a stepping stone for a rapid industry uptake of the project's results.
DTLS	The Datagram Transport Layer Security protocol is derived from TLS that secures communication between web browsers and servers. In contrast to TLS, DTLS runs over UDP rather than TCP and hence fits resource-constrained devices. It is one of the most important higher layer security protocols in the IoT as it is used by CoAP, the IoT's web protocol. As TLS, DTLS provides integrity, authentication and confidentiality.
TSCH	Time-Slotted Channel Hopping is a MAC-layer protocol that provides industrial-grade reliability (e.g., 99.999% delivery) in IoT scenarios. TSCH has attracted the attention of major industry players including ABB, Volvo Trucks, Linear Technologies, Toshiba, ST Microelectronics and NXP. In the industrial scenarios where these companies plan to employ TSCH, security is of utmost importance.

Task 3.3: Control, Monitoring, and Analysis In order to further protect against both network attacks and malicious code, aSSIsT will design solutions for control, monitoring, and analysis, with minimal effect on performance and stability. An example of a *control* component is IoT firewall technology adaptable to the context of resource-constrained devices and to new security situations. Thoroughly *monitoring* the activities on devices and network can be prohibitively expensive for constrained units. With efficiency being the main challenge, we plan to replace continuous scanning by trapping on exceptions (enabled by Task 3.1) whenever meaningfully possible. The *analysis* of monitored data needs to keep both communication and on-node computation costs low. We will therefore develop solutions where local analysis on the device is combined with off-loaded analysis, e.g., on the cloud, in a manner that strikes optimal balance between analysis power and message traffic and power consumption.

Demonstrators

We will showcase the results of the project through the three demonstrators of Table 1.

Time Plan and Resources

Our research will be driven by the demonstrators and their application and will be performed in close collaboration with the original developers of IoT protocols available in Contiki and, whenever appropriate, on application code from companies of the reference group. Table 2 contains a breakdown of main actions during the duration of the project.

Available Resources Our demonstrators will make use of the analysis tools and systems software developed by the project partners, in particular the Contiki OS that stems from SICS and is now used widely in both academia and industry. The DTLS and TSCH implementations for the demonstrator have been provided by SICS researchers (co-PI Simon Duquennoy has implemented TSCH for Contiki). In addition, we have infrastructures for testing and benchmarking: both SICS and UU have testbeds ranging from 25 to 50 sensor nodes. Techniques for analysis and testing will in some parts further develop software in partners' widely-used tools, including PropEr, Concuerror, and Nidhugg.

Requested Resources The proposed project budget lists the required additional resources to carry out the project plan, viz. resources for(i) 25% time for each PI, (ii) 11 person years of PostDoc efforts, and (iii) 4 Ph.D. students, all except one recruited at project start at 80% effort during the project.

4 Consortium

The partners of the project are organized into two groups: a **Research Consortium**, responsible for carrying out the technical work, and a **Reference Group**, which will review project progress, and help identify spin-off activities.

Research Consortium

The Research Consortium brings together world-leading key competencies for achieving the project goals, in verification, testing, programming language technology, wireless networking, embedded operating systems, and security, hosted by Uppsala University and SICS.

Uppsala University, Department of Information Technology hosts research activities with a unique breadth; covering areas that include wireless networking, embedded systems, verification, and programming language technology. Involved groups leads the UPMARC linnaeus centre of excellence, and have recently led the SSF-funded projects CoDeR-MP and ProFuN. The department hosts the program office for the VINNOVA strategic innovation program **Internet of Things**. The groups involved in aSSIsT are:

- **Verification group** (Bengt Jonsson, Parosh Abdulla, Mohammed Faouzi Atig, Philipp Rümmer,

Table 2: Action plan for completing tasks during the duration of aSSIsT.

Year	Actions (Tx.y denotes Task x.y)
1	Techniques for effective analysis of software on IoT devices **(T1.1)**, demonstrated by application to Contiki's components. Design and initial prototype of tool for stateful targeted property-based testing (TPBT) **(T2.1)**. Design and initial implementation of memory isolation **(T3.1)** and on-device monitoring **(T3.3)** solutions.
2	Techniques for effective analysis of interrupts and event-driven programs **(T1.2)**, applied to Contiki. Integration of analysis with the "emulate" platform for Contiki simulation. Application of TPBT tool to DTLS protocol **(T2.1)**. Design and initial prototype of a tool for systematic fuzzing **(T2.2)**, of exploit mitigation solutions **(T3.2)** and cloud-offloaded analysis **(T3.3)**. Refined implementation and evaluation of the Y1 mechanisms of **WP3**.
3	Techniques for complete analysis of software components **(T1.3)**. Integration of software analysis with protocol implementation testing **(T1.1&2.1)**. Evaluation of the tool for systematic fuzzing on IoT protocols DTLS and TSCH **(T2.2)**. Extension of memory isolation **(T3.1)**, exploit mitigation and supervisor **(T3.2)** with TEE. Evaluation of correctness and performance of Y2 mechanisms of **WP3**. Start of the integration with Contiki.
4	Development of a framework for compositional testing **(T2.3)** and application to complete verification of security properties **(T1.3&2.3)** in components of Contiki. Iteration of the design and implementation of the Y1–Y3 mechanisms of **WP3**, based on the correctness and performance evaluations. Contiki integration continues.
5	Integration of software analysis, memory isolation, and protocol testing tools within a simulator. Final integration and demonstration in Contiki. Completed evaluation on Contiki code base of correctness and performance.

Wang Yi) is world-renowned in the field of automated verification for among others, its contributions to automated verification of infinite-state and timed systems, including the UPPAAL model checker, for its work on verification of multithreaded software software under different concurrency memory models, and on techniques for learning protocol models from tests. Members received the CAV (Computer-Aided Verification) Award, the most prestigious academic award in the area, both in 2013 (Wang Yi) and in 2017 (Parosh Abdulla and Bengt Jonsson).

- **Programming Language Technology group** (Kostis Sagonas, Dave Clarke, Tobias Wrigstad) is known for work on design and implementation of high-level languages for concurrent programming (Erlang and Encore), for type systems ensuring safety of memory accesses in OO-languages, for the HiPE native code compiler for Erlang and for various widely-used tools (Dialyzer, TypEr, PropEr, Concuerror, CutEr) for static analysis, testing, and stateless model checking (SMC) of programs. Members have recently applied SMC to the code of RCU, a key component of the Linux Kernel, discovered bugs that were previously unknown in older kernels, and verified the key correctness property of RCU in recent kernels. They are currently working towards integrating this work into the Linux test framework.

The groups have a strong track record of collaboration, resulting in high-impact techniques and tools for testing and verification of multithreaded software, including Concuerror and Nidhugg.

RISE SICS is Sweden's leading research institute for applied ICT, founded in 1985. The research is based on cutting-edge new technology with a time horizon stretching beyond other companies' own R&D efforts. The groups involved in aSSIsT are

- **Networked Embedded Systems group** (Simon Duquennoy, Luca Mottola, Thiemo Voigt) is known for its strong track record in wireless sensor networks, and its contribution of Contiki, today's leading open-source OS for constrained IoT devices. The group has major contributions to robust embedded systems and low-power communication, and publishes regularly and receive awards at the flagship conferences in the field, ACM SenSys and IEEE/ACM IPSN. Among the many awards of Luca Mottola is the ACM SigMobile Research Highlight in 2016. Simon Duquennoy is active on formal verification in IoT software: he was a co-applicant of the H2020 Vessedia project that explores static analysis techniques for IoT.
- **Security Lab** (Shahid Raza, Ludwig Seitz) is the largest cybersecurity group in Sweden, with 19 members and two affiliates, begin experts in IoT security, software security, formal methods, cryptography, standardization, privacy (technical and social aspects), cloud security and virtualization. Shahid Raza has driven the security research around the Contiki operating system. Currently, the lab is involved in 15 research projects, funded by H2020, VINNOVA, Eurostars, Celtic-Plus, EIT, ARTEMIS, SSF, and Swedish industry.

The groups have a strong track record of collaboration, e.g., producing Anquiro, a model checker for sensor networks code. Furthermore, the Contiki operating system and the simulation tools Cooja and MSPSim stem from SICS. Uppsala and SICS have previously led SSF-funded projects Promos and ProFuN on software for wireless networks.

Reference Group

The reference group comprises key commercial actors with a core business in IoT system develop-

ment. Partners are:

ASSA ABLOY is the global leader in door opening solutions. ASSA ABLOY is developing embedded software for a wide range of products and are interested in the developed solutions within this project as it can improve software security. *Contact: Tomas Jonsson*, +46 (0)708950666

Intel Sweden AB For Intel, IoT is a key business. For Intel security for embedded devices is of utmost importance. Intel supports the Zephyr OS and can benefit from the work done in this project. *Contact: Björn Runåker*, +46 (0)72 536 69 11.

LumenRadio AB provide wireless solutions targeted for the most business critical applications where reliability and security are key issues. They build products on top of Contiki and have incorporated the Contiki simulator Cooja in their development process. Hence, most of the developed solutions within this project would be directly applicable for them. *Contact: Michael Karlsson, R&D Manager*, +46 (0)708 97 95 97.

Upwis develops low-power sensors and routers for sensing applications. The project will benefit from Kjell Brunberg's huge network in Swedish industry. Also, Upwis uses Contiki and hence can make direct use of many of the developed solutions. *Contact: Kjell Brunnberg*, +46 (0)73 3660707.

Yanzi has deployed the worlds largest automatically provisioned IoT system for smart offices, with over 1000 sensors. Yanzi has strong collaborations with large industrial players such as Intel, IBM and Microsoft. Like LumenRadio, they build products on top of Contiki and most of the developed solutions in aSSIsT would be directly applicable for them. *Contact: J. Eriksson*, +46 (0)70 3213841.

The reference group will:(i) review and provide feedback on the project's progress and planned work; (ii) evaluate and test selected project outcomes; (iii) ensure the relevance for current development of IoT technologies and standards; (iv) help identify additional suitable partners for further collaborations; (v) help identify suitable spin-off projects. The research consortium and reference group will meet twice a year in half-day seminars, featuring presentations of project progress, relevant technology development, and discussion of new activities.

References

[1] F. Abdi et al. Restart-based security mechanisms for safety-critical embedded systems, 2017. Quanser product(s): 3 DOF Helicopter.

[2] P. Abdulla, S. Aronis, B. Jonsson, and K. Sagonas. Optimal dynamic partial order reduction. In POPL, pp. 373–384, 2014. ACM.

[3] N. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In S&P, pp. 526–540. IEEE, 2013.

[4] T. Avgerinos, A. Rebert, S. K. Cha, and D. Brumley. Enhancing symbolic execution with veritesting. *Comm. of the ACM*, 59(6):93–100, 2016.

[5] L. Baresi, C. Ghezzi, and L. Mottola. Loupe: Verifying publish-subscribe architectures with a magnifying lens. *IEEE Trans. on Soft. Eng.*, 37(2):228–246.

[6] B. Beurdouche et al. A messy state of the union: Taming the composite state machines of TLS. *Comm. of the ACM*, 60(2):99–107, Feb. 2017.

[7] B. Beurdouche et al. FLEXTLS: A tool for testing TLS implementations. USENIX, 2015.

[8] C. Cadar, D. Dunbar, and D. Engler. KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In *OSDI*, pp. 209–224. USENIX, 2008.

[9] S. Cassel, F. Howar, B. Jonsson, and B. Steffen. Active learning for extended finite state machines. *Formal Asp. Comput.*, 28(2):233–263, 2016.

[10] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley. Unleashing mayhem on binary code. In *S&P*, pp. 380–394. IEEE, 2012. See also http://archive.darpa.mil/cybergrandchallenge/.

[11] A. A. Clements et al. Protecting bare-metal embedded systems with privilege overlays. In *S&P*, 2017.

[12] D. Davidson et al. FIE on firmware: Finding vulnerabilities in embedded systems using symbolic execution. In *USENIX Security*, pp. 463–478, 2013.

[13] J. de Ruiter and E. Poll. Protocol state fuzzing of TLS implementations. In *USENIX Security*, pp. 193–206. USENIX, 2015.

[14] A. Dunkels, B. Gronvall, and T. Voigt. Contiki-a lightweight and flexible operating system for tiny networked sensors. In *LCN*, pp. 455–462, 2004.

[15] A. Giantsios, N. Papaspyrou, and K. Sagonas. Concolic testing for functional languages. *SCP*, 2017.

[16] P. Godefroid, M. Levin, and D. Molnar. SAGE: whitebox fuzzing for security testing. *Comm. of the ACM*, 55(3):40–44, 2012.

[17] J. Hill et al. System architecture directions for networked sensors. *OS Review*, 34(5):93–104, 2000.

[18] A. Koivu et al. Software security considerations for IoT. In *iThings*, pp. 392–397, IEEE, 2016.

[19] A. Löscher and K. Sagonas. Targeted property-based testing. In *ISSTA*, 2017. ACM.

[20] A. Löscher, K. Sagonas, and T. Voigt. Property-based testing of sensor networks. In *SECON*, pp. 100–108. IEEE, 2015.

[21] L. Mottola, T. Voigt, et al. Anquiro: Enabling efficient static verification of sensor network software. In *ICSE Workshops*. ACM, 2010.

[22] D. Ramos and D. Engler. Under-constrained symbolic execution: Correctness checking for real code. In *USENIX Security*, pp. 49–64. USENIX. 2015.

[23] Security Risk Detection. https://www.microsoft.com/en-us/security-risk-detection/.

[24] K. Sen. Concolic testing. In *ASE*, ACM, 2007.

[25] H. Shacham et al. On the effectiveness of address-space randomization. In *CCS*, pp. 298–307, 2004.

[26] D. Shreenivas, S. Raza, and T. Voigt. Intrusion detection in the RPL-connected 6LoWPAN networks. In *IoTPTS*, pp. 31–38, 2017.

[27] J. Somorovsky. Systematic fuzzing and testing of TLS libraries. In *CCS*, pp. 1492–1504, 2016. ACM.

[28] T. Vörtler et al. Formal verification of software for the Contiki OS considering interrupts. In *DDECS*, pp. 295–298. 2015. IEEE.