Strategy for Security Certification of High Assurance Industrial Automation and Control Systems

Thorsten Schulz*, Caspar Gries†, Frank Golatowski* and Dirk Timmermann*
*Institute of Applied Microelectronics and CE, University of Rostock, Germany †SYSGO AG, Klein-Winternheim, Germany
Email: {thorsten.schulz,frank.golatowski,dirk.timmermann}@uni-rostock.de, caspar.gries@sysgo.com

***Abstract*—High assurance Cyber-Physical Systems (CPS) are the supporting pillars of the critical infrastructure. They support the power grid, the water supply, transportation systems and many other devices, where failure or undefined behaviour lead to risk for loss of life, danger to the environment and defective operational safety of production. Rigorous testing practices have assured reliable behaviour even for failure scenarios in their predictable environments. However, previously isolated systems have become connected to the Internet and expose an attack surface that is hard to predict. While the safety of high assurance CPS is well tested with a controlled residual risk, security risks will rise throughout the deployment of a system. Hence, this paper describes research for a testing methodology to tackle emerging threats and preserve certified security assurance.**

I. Introduction

Safety is addressed generically through "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems" (IEC 61508), and also with their refinements in many domain-specific standards, e.g., in avionics (DO-178C, ARP 4754), in automotive (ISO 26262), in railway (IEC 62279 and EN 501xx), in nuclear-power plants, and production plants. Functional safety addresses the adequate reduction of risk that the system can harm its environment. Safety measures reduce risks coming from random and systematic failures, which are well modeled and understood, because they are under the control of the system and equipment manufacturer. However, safety is also tightly coupled to availability and integrity of the system to fulfill its safety policy, i.e., to correctly perform the safety function(s).

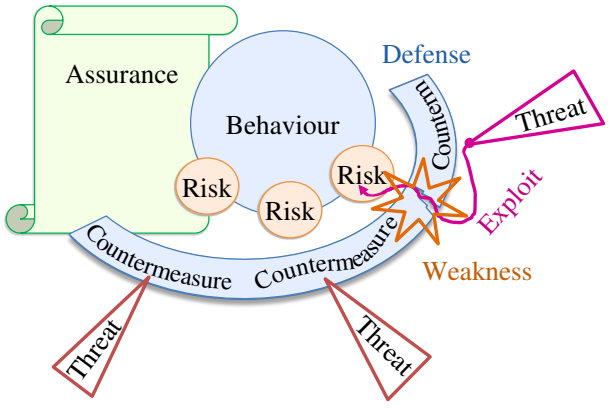

Fig. 1. With the disclosure of an exploit an existing or a new threat can bypass the layer of defense and will trigger a risk situation.

978-1-5386-4155-2/18/$31.00 ©2018 IEEE, 10.1109/SIES.2018.8442081

With the emergence of Commercial-Off-the-Shelf (COTS) products in Industrial Automation and Control Systems (IACS), these properties become tightly coupled to security assurance. In contrast to safety, security addresses risk in the other direction. Security threats are imposed on the system by the environment. The concept of terms is shown in Fig. 1. In this context, assurance is a measure of confidence in the correct behaviour of a system. A vulnerability is a weakness of a system, which can be used to cause a threat. A threat is a method leading to a dangerous event, i.e., risk. The layer of defense is the collection of countermeasures preventing a threat escalating to a risk. An exploit bypasses the layer of defense, using a vulnerability to realize a threat, leading to a dangerous event. It results in loss of integrity of the system and its assurance. Evaluated security techniques can ensure availability and integrity by employing countermeasures to mitigate threats imposing risks on assets.

Additionally, security defense measures have to deal with an environment, which is uncertain and changes or evolves without the control of the supplier. This potentially invalidates static countermeasures against known vulnerabilities. As a result for necessary security certification, high assurance CPS must have an effective infrastructure (not limited) to trace vulnerabilities, receive patches, and re-run tests to demonstrate the required safe behaviour and reassure certification. This process must still adhere to existing safety certification processes.

The certMILS project has been set up to develop a security certification methodology according to Common Criteria and IEC 62443 for a trustworthy Multiple Independent Levels of Security (MILS) platform [1]. There are more definitions for MILS platforms, such as defined by Alves-Foss [2] and D-MILS [3], which follow different approaches. The fundamental architecture in the certMILS approach is composed of multiple components: Hardware, a separation kernel and multiple partitions for applications of varying security and safety level. Each component may be under the control of a different role of responsibility: supplier, system integrator, or operator. By following the compositional security certification approach, complexity is reduced and the platform is accessible for many industrial safety domains. The project's current work in progress are the development of the certification methodology, creation of the MILS Platform Protection Profile and evaluation of security testing techniques. The latter is the focus of this publication within the following sections. The requirements for security, and robustness testing to maintain assurance and counter emerging threats are described in the next section. The third section will look at the state of the art tooling available from the IT world and early usability

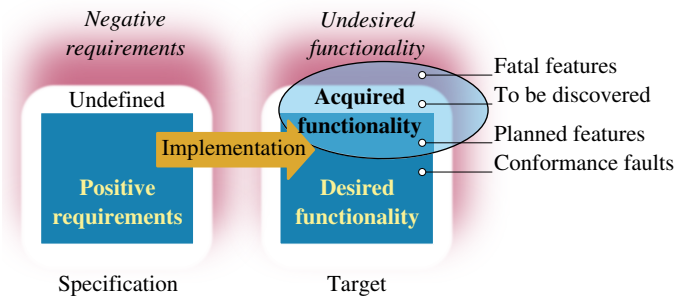

Fig. 2. Non-formal implementation of requirements lead to mismatch of desired and acquired functionality. (Redesigned from [5].)

evaluations for high assurance IACS.

II. Security Testing and Certification

In the web application and general IT domain, large vendors and interest groups have introduced security frameworks, essentially to protect their users and products from abuse: The OWASP Testing Framework, Microsoft SLAM and SAGE [4], MS Security Development Lifecycle (MS SDL), etc. Google has established the OSS-Fuzz service donating server resources to rigorously test open source software. The MS-SDL security framework suggests the development lifecycle to be accompanied with multiple tools:

Phase	Tool
early requirements and design-phase	threat-modelling
implementation phase	static analysis tools
verification phase	penetration-testing and dynamic analysis tools (e.g., fuzz-testing)

Security testing is part of robustness testing within the verification phase. One methodology is fuzz-testing. It invokes the target with random and intelligently mutated input data. Oehlert [6] references fuzz-testing as a robustness testing technique "to better ensure the absence of exploitable vulnerabilities" by "checking large numbers of boundary cases" that functional testing cannot cover. It adds negative test cases to verify that a software or "product does not do something it should not do". In Fig. 2, fatal "features", i.e., risks, are the result of the acquired functionality of a system mismatching the space of desired functionality through a conventional implementation process sourcing from the requirements specification. Takanen et al. [7] provide an in-depth publication on this topic. Indeed, the space of undesired functionality is much larger than of desired functionality, covered by functional testing for conformance. Consequently, fuzz-testing becomes a time consuming technique.

Over the last years, different improved techniques go beyond the "dumb" brute-force fuzzing. E.g., "Taint-Style Vulnerabilities" [8] were the target of Shastry et al in 2017. First the application is fuzz-tested for vulnerabilities. Then, identified vulnerability code patterns were matched throughout the remaining code base to find tainted repetitions. Though this approach yields many false positives, which required further analysis techniques.

So far, the previous tools and processes of general security testing do not suffice for the certified CPS domain. For testing of IACS, Takanen [7] proposes to differentiate the fuzz-testing into Ethernet communication-, logic-, and I/O-processing. As mentioned earlier, *any* stress imposed on a system must not break the safety policy. Challenges specific to IACS were identified by [7]:

- protocol diversity and implementation ambiguity,
- equipment access and configuration complexity,
- test simulations either with and without load,
- grey-box access to SUT (System-Under-Test),
- multi-way redundancy, watch-dogs, fail-safe modes, communication failover
- performance constraints of tested system.

The most prominent tool to address this domain's challenges at the time of the publication of "Fuzzing for Software Security Testing and Quality Assurance" [7] in 2008 was Wurldtech Achilles. In the meantime, as detailed by Hohenegger et al. [9], Achilles has evolved into a formally approved certification process for exhaustive black-box network security and network robustness testing. Achilles certification is based on IEC 62443-2-4 [10].

However, the IEC 62443 series of standards has a much wider scope, to cover the whole design of cyber-security robustness and resilience in IACS. The series is organized into four groups addressing (1) general topics, (2) policies and procedures, (3) the system level and (4) the component level. IEC 62443-4-<1,2> focus on device product development requirements and technical security requirements. IEC 62443-3-3 is related to "System security requirements and security levels". These two aspects can be certified by accredited labs using the ISASecure certification schemes:

EDSA Embedded Device Security Assurance [11],

SSA System Security Assurance [12], and

SDLA Security Development Lifecycle Assurance [13].

Most importantly, as a security standard, they define the functional security for safety, availability, integrity and confidentiality for the system and its components. Details about secure implementations are not specified, but the expected behavior. To phrase this more explicitly, IEC 62443 is a standard and makes no obvious technical suggestions. By using the MILS approach, especially the certMILS reference solution building on compositional certification, details can be defined to ease a certification scheme.

The Common Criteria (CC) [14] certification scheme has introduced an *a*ssurance class for *co*mposed (ACO) IT products, where most evaluation work, compared to a monolithic evaluation, is focused on the evaluation of the security of interfaces between components. Though this approach still has impediments, when reaching for higher assurance levels [9]. Here, analysis has also shown that some lines of similarities between the schemes of CC and IEC 62443 can be drawn.

For example, IEC 62443 has the concept of zoned network components. The hypothesis anticipates to map this concept

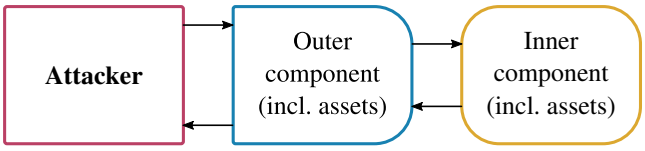

Fig. 3. Relation and information flow between an untrusted component (i.e., attacker) and the assets in a layered composed system.

for certification of a networked composition to a MILS system. Such a networked composition would require efficient verification and testing to demonstrate non-bypassability of the security policies.

For this testing, first a threat analysis models the attacker and the tested components of the composed system. A layered composition, as shown in Fig. 3, is a simpler version of the network type composition [1] mentioned before. The network type can be modelled as an overlay or a fusion of multiple layered evaluations. In this layered model, the outer component has to be robust against malicious inputs. In contrast, the inner component does not have to provide maximum robustness, as the outer component is trusted to only pass valid data. In a generalized MILS system designed as a networked composition, the attacker, i.e., the untrusted component, can have different mappings. E.g., an unprivileged partition component, as implemented by the system integrator or the operator role, cannot generically be trusted by the separation kernel component supplier.

Typically, the hardware access layer, also referred to as platform support package (PSP), is added by the system integrator. It is a component to be composed with the separation kernel component. On the one hand, this makes the MILS system very flexible for COTS hardware and its obsolescence throughout the product's lifecycle. On the other hand, drivers, especially hardware drivers, require elevated access rights and most must run in the kernel domain. For technical and performance reasons, this interface cannot provide the same robustness, as the system-call interface to the user space components. Testing of this information flow for security and non-bypassability is analyzed in the next chapter.

III. Technical Approach for Ongoing Work

In an interconnected CPS, the emergence of security vulnerabilities is realistically inevitable. These general "future" vulnerabilities require these systems to support a patching methodology throughout the lifecycle. Though a small patch must not demand a full system re-certification. As discussed before, a MILS system does address this requirement through partitioning into, e.g., (at least) security relevant components that may need patching and components fulfilling safety functions. Accordingly, to maintain confidence in the correctness of the system, testing can and must be specific to certain aspects of the distinctive components of a MILS system. An additional component that specializes a more generic system or a patched component must proof confidence with testing before returning the system to operations.

The previous chapter concluded with rigorous testing being one of the important techniques to gain confidence in the correctness of a high-assurance system. Testing tools for functional testing are established and well available. Functional testing demonstrates correct behaviour of specified requirements (see earlier in Fig. 2). However, due to often incomplete specifications and ambiguity in requirement interpretation leading to implementation mismatch, undesired behaviour can occur and must be traced with robustness testing.

Fuzz-testing, a robustness testing technique, has lately been most successful to discover hidden security vulnerabilities. However, there are challenges: Firstly, fuzz-testing tools need specific adaptation to the interfaces of the test target, i.e., the component. Secondly, the failure, i.e., the crash of a component, needs to be handled. Prominent tools like AFL [15] and libFuzzer [16] handle user-space fuzzing very well, and can also work on high assurance (user-space) applications. Fuzzing kernel-space components, such as the separation kernel or the hardware layer of a MILS system, require more complex approaches. Lastly, even though current fuzzing suites provide powerful tooling to dissect the results of fuzz-testing, this is non-trivial without additional, provided guidance. IEC 62443 describes the roles beyond the component supplier, such as the system integrator and the operator. For example, the integrator may develop an additional hardware driver for their specialized product, and needs to test it for integration. However, typically the integrator is not an expert on internals of the separation kernel component of the MILS system. Further on, operators may have to apply an urgent security patch along maintenance operations, for which *automated* tests need to be provided.

For the security certification approach of the certMILS project must master these different challenges. At first, the approach needs be addressed in a Common Criteria Protection Profile (PP). Since there exists no adequate PP for a MILS separation kernel [17], the Base MILS Platform PP is being written by certMILS project members and published for comments of the community. It consists of the target-of-evaluation (ToE) description, security problem definitions, sec. objectives and sec. requirements. It is assumed, that the underlying approach can also successfully be used for IEC 62443 evaluations. Concurrently, the robustness testing techniques for the different component aspects are being evaluated and implemented. This includes fuzz-testing for user-space application components, for the system-call interface, for the system service component (for, e.g., inter-process communication setup and resource management) and for separation kernel extensions, such as HW-drivers and platform abstractions (PSP). Even in COTS operating systems (OS) testing of the latter is non-trivial, still, indispensable due to the missing context-switch barrier.

For generic IT systems, kernel fuzzing is addressed by the publishers of TriforceAFL [18]. They are using the full system emulation framework QEMU [19] to fuzz-test black-box OS kernels. Even though claiming "arbitrary" OS can be tested, it is silently presumed, the OS is compatible with the emulated environment. Furthermore, this approach is of small use for hardware access layer (PSP) fuzzing, as the full system hardware is emulated by QEMU. Typically, QEMU provides hardware emulation in a generic way that does not necessarily correspond to the actual physical hardware used in IACS. Such fuzzing of the PSP may not provide results adequately related to the operational product.

An improved approach, recently published as kAFL [20], instead uses kernel-based system virtualization [19]. Its pri-

mary goal was significant speed improvement over an emulated system. In a virtualization approach, the testing-hypervisor could also pass-through virtualized parts of the target system compared to an emulated system (see [21]). Though community feedback has shown that this approach is expected raise many technical complications and may still be of limited use for hardware fuzzing in practice.

The authors of kAFL supplied an example of a vulnerable kernel driver that panics on a "magic" input pattern. The example was ported to a simple MILS system based on the PikeOS kernel and fuzz-tested on an average laptop without any special powers. The kAFL fuzzer traced the magic patterns successfully in a matter of hours. Further experiments, supporting own code instrumentation hardware-independent techniques, are in progress.

IV. Conclusion

Upcoming work for the certMILS project needs to merge the state of the art approaches into a tools suite that, most importantly, is maintainable for prolonged periods with limited resources compared to major IT companies. The security testing suite required for the certification methodology must also be "light-weight", to be well evaluable by certification assessors. Some of the technical solutions in the previous chapter introduced heavy modifications to the system emulator QEMU or use a self-developed fuzzer, which may prove expensive to maintain. In case of IACS development with cooperative development partners, simpler approaches based on white-box testing of kernel components are currently favored for the progress of the certMILS project. Instead of architecture specific run-time processor trace information, compiler-based code instrumentation of the tested kernel component can be applied for code-coverage feedback of fuzzing-test-cases. This is similar to the original AFL approach, but must be adapted for the special environment of the MILS separation kernel.

The collection of fuzz-testing guidance, a system fuzzing framework for kernel and PSP, a user-space-interface fuzzer ("syscall" fuzzer) and additional existing tools, such as Achilles, or Nessus, will ease demonstration of preservation of assurance for the complete lifecycle, maintaining the security certification for a high assurance MILS system according to Common Criteria or IEC 62443.

Acknowledgment

Thanks to all project partners for their great contributions.

This work is part of the certMILS project, funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 731456.

References

[1] S. Tverdyshev, H. Blasum, B. Langenstein, J. Maebe, B. D. Sutter, B. Leconte, B. Triquet, K. Müller, M. Paulitsch, A. S.-F. von Blomberg, and A. Tillequin, *MILS Architecture*, 2013. DOI: 10.5281/zenodo.45164.

[2] J. Alves-Foss, P. W. Oman, C. Taylor, and W. S. Harrison, "The MILS architecture for high-assurance embedded systems," *International Journal of Embedded Systems*, vol. 2, no. 3-4, pp. 239–247, 2006. DOI: 10.1504/IJES.2006.014859.

[3] D-MILS Project. (2013). Distributed MILS for Dependable Information and Communication Infrastructures, [Online]. Available: http://www.d-mils.org/.

[4] P. Godefroid, M. Y. Levin, and D. Molnar, "SAGE: Whitebox Fuzzing for Security Testing," *Commun. ACM*, vol. 55, no. 3, pp. 40–44, Mar. 2012, ISSN: 0001-0782. DOI: 10.1145/2093548.2093564.

[5] A. Takanen, "Fuzzing For Software Security Testing & Quality Assurance," in *EuroStar 2009*, Stockholmsmässan, Sweden, 2009.

[6] P. Oehlert, "Violating assumptions with fuzzing," *IEEE Security Privacy*, vol. 3, no. 2, pp. 58–62, Mar. 2005, ISSN: 1540-7993. DOI: 10.1109/MSP.2005.55.

[7] A. Takanen, J. DeMott, and C. Miller, *Fuzzing for Software Security Testing and Quality Assurance*. Norwood: Artech House Publishers, 2008, ISBN: 9781596932159.

[8] B. Shastry, F. Maggi, F. Yamaguchi, K. Rieck, and J.-P. Seifert, "Static Exploration of Taint-Style Vulnerabilities Found by Fuzzing," Jun. 1, 2017. arXiv: 1706.00206v1.

[9] A. Hohenegger, H. Blasum, S. Tverdyshev, L. Garcia, A. Á. de Sotomayor, B. C. Sillero, K. Tomáš, G. Krummeck, H. Kurth, S. Persson, R. Hametner, M. Paultisch, P. Tummeltshammer, and H. Michal, "certMILS D1.1 Regulative Baseline: Compositional Security Evaluation," 2017, t.b. publ. on zenodo.org.

[10] IEC TC65 WG10, Ed., *IEC TS 62443-2-4:2015 Industrial communication networks - Network and system security - Part 2-4: Requirements for IACS solution suppliers*, Geneva: IEC, 2015.

[11] *ISASecure - IEC 62443-4-2 - EDSA Certification*, 2017. [Online]. Available: http://www.isasecure.org/en-US/Certification/IEC-62443-EDSA-Certification.

[12] ISA Security Compliance Institute, *SSA-300 system security ass. – ISASecure certification requirements v.1.4*, 2016. [Online]. Available: http://www.isasecure.org/en-US/Certification/IEC-62443-SSA-Certification.

[13] ISA Security Compliance Institute, Ed., *ISASecure - IEC 62443-4-1 - SDLA Certification*, 2017. [Online]. Available: http://www.isasecure.org/en-US/Certification/IEC-62443-SDLA-Certification.

[14] Common Criteria. (2017). CC Portal, Schemes. https://www.commoncriteriaportal.org/ccra/schemes.

[15] M. Zalewski. (2017). American fuzzy lop, [Online]. Available: http://lcamtuf.coredump.cx/afl/.

[16] LLVM-Foundation. (2017). libFuzzer – a library for coverage-guided fuzz testing.

[17] S. Tverdyshev, "Security by design: Introduction to mils," MILS Workshop Embedded World Conference, 2017. DOI: 10.5281/zenodo.571164.

[18] J. Hertz and T. Newsham, "Project Triforce: Run AFL on Everything!" NCC Group, Tech. Rep., 2016.

[19] F. Bellard. (2017). QEMU - the FAST! processor emulator, [Online]. Available: https://www.qemu.org/.

[20] S. Schumilo, C. Aschermann, R. Gawlik, S. Schinzel, and T. Holz, "kAFL: Hardware-Assisted Feedback Fuzzing for OS Kernels," in *USENIX Security Symposium 2017*, USENIX Association, 2017, pp. 167–182.

[21] A. Shah, A. M. Kay, M. Ben-Yehuda, and B.-A. Yassour. (2008). PCI Device Passthrough for KVM. K. Forum, Ed., [Online]. Available: https://www.linux-kvm.org/images/d/d0/KvmForum2008$kdf2008_14.pdf.