Automating IoT Security Standard Testing by Common Security Tools

Rauli Kaksonen[1][a], Kimmo Halunen[1,2][b], Marko Laakso[1] and Juha Röning[1][c]
[1]*University of Oulu, Oulu, Finland*
[2]*National Defence University of Finland, Department of Military Technology, Finland*
fi

Keywords: IoT, Security Standard, Testing, Automation, Security Tools, ETSI EN 303 645, ETSI TS 103 701.

Abstract: Cybersecurity standards play a vital role in safeguarding the *Internet of Things* (IoT). Currently, standard compliance is assessed through manual reviews by security experts, a process which cost and delay is often too high. This research delves into the potential of automating IoT security standard testing, focusing on the ETSI TS 103 701 test specification for the ETSI EN 303 645 standard. From the test specification, 56 tests are relevant for the network attack threat model and considered for automation. Results from the research are promising: basic network security tools can automate 52% of these tests, and advanced tools can push that number up to 70%. For full test coverage, custom tooling is required. The approach is validated by creating automation for a real-world IoT product. Test automation is an investment, but the results indicate it can streamline security standard verification, especially for product updates and variants. Data from other testing activities can be used to reduce the effort. Automating the security standard testing would enable the certification of a large number of IoT products for their lifetime.

1 INTRODUCTION

Cybersecurity standards play a vital role in safeguarding the *Internet of Things* (IoT). Effective and independent assessment of security standard compliance is essential to build trust to the systems we depend on. The need to improve the cybersecurity of IoT devices has been widely recognized. The lack of security in IoT will undermine the users' trust and hamper the adoption of new products. (Matheu et al., 2020; Traficom, 2023)

The current security assessment techniques are complex and must be performed by cybersecurity experts. Major challenges are the high cost, delays to product launch, re-certification as products have to be updated, and overall scalability to cover the huge number of products in the market (Matheu et al., 2020). A potential solution would be to replace manual work with automated tool-based verification.

Different products have different security needs and thus the rigor of security certification evaluation should vary (ECSO, 2017). An entry-level security certification may be sector-agnostic and performed as *self-certification*. *Third-party certification* is required for higher-level assurance. An entry-level assessment may only consider the interface of the product, i.e. a black-box assessment. A high-level certification may require inspection of design information, source code, etc. Security assessment can examine the product, the vendor and its processes, or both. The product-based approach allows assessment by customers or third parties without visiting vendor premises. When a new product version is released, a *re-certification* is required. To avoid full re-certification, only the modified parts of the product may be assessed or the vendor is given the freedom to perform updates without invalidating the certificate (ECSO, 2017).

1.1 Certification and Standards

Many different IoT security standards, guidelines, and best practices exist (Cirne et al., 2022; Kaksonen et al., 2022; Khurshid et al., 2022; Matheu et al., 2020). The requirements are usually generic in nature and verification requires the creation of test cases and review actions. This may bring inconsistencies and variation to the evaluations done by different evaluators and for different products.

However, there are some relevant standards which have a test specification. ETSI EN 303 645 is a recognized consumer IoT security standard with accom-

[a] https://orcid.org/0000-0001-8692-763X
[b] https://orcid.org/0000-0003-1169-5920
[c] https://orcid.org/0000-0001-9993-8602

Kaksonen, R., Halunen, K., Laakso, M. and Röning, J.
Automating IoT Security Standard Testing by Common Security Tools.
DOI: 10.5220/0012345900003648
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 10th International Conference on Information Systems Security and Privacy (ICISSP 2024)*, pages 42-53
ISBN: 978-989-758-683-5; ISSN: 2184-4356
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

panying test specification ETSI TS 103 701 (ETSI, 2020; ETSI, 2021). ETSI EN 303 645 is used as the basis for Finnish, Singaporean, and German cybersecurity labels (BSI, 2023a; CSA, 2023; Traficom, 2023). ETSI EN 303 645 can be considered sector-agnostic which requirements are applicable for most consumer IoT products. *The Common Criteria for Information Technology Security Evaluation* (CC) is a third-party certification scheme standardized as ISO/IEC 15408 (CC, 2022). CC is sector-agnostic, but its large set of optional requirements can be mixed and matched to suit different needs. Requirements can be also tailored using operations. The CC *protection profiles* are predefined collections of requirements for specific product sectors. However, the cost and complexity of CC certification seriously hinder its adoption for IoT (Matheu et al., 2020).

There are also national and industry-specific certification schemes. The Federal Office for Information Security (BSI) of Germany has published security requirement guideline TR-03148 and a test specification specific for consumer broadband routers (BSI, 2023c; BSI, 2023b). *ioXt* is an industry-driven security certification scheme by the ioXt Alliance (ioXt Alliance, 2023), which specifies requirements and test cases. It supports both self-certification and third-party certification by base and sector-specific requirements.

1.2 Common Security Tools

There exists a large pool of open source security tools to probe various aspects of IT security (Kaksonen et al., 2021). The most popular tool categories are network scanning, packet capture and injection, web security, and disassembly.

The use of tools is not usually dictated in security standards, but it is up to the evaluator to select the tools, if any. ETSI TS 103 701 especially discussed this: *"Due to the heterogeneity of consumer IoT devices ... test groups in the present document are formulated in a generic manner. Thus, the present document does not describe specific tools or detailed step-by-step instructions"* (ETSI, 2021, p. 10). The continuous emergence of new attacks and vulnerabilities would call for automated monitoring, testing, and mitigation tools (Cirne et al., 2022).

1.3 Related Studies

Model-based testing (MBT) is an often suggested security testing automation approach for IoT systems (Lonetti et al., 2023; Matheu et al., 2020; Matheu García et al., 2024). MBT facilitates automated test case generation. For example, Matheu-Garcia et al. present a risk-based MBT approach for automated IoT cybersecurity certification (Matheu-García et al., 2019). However, the downside is that the generation of comprehensive security test cases requires a very detailed model. It is indicative, that MBT research papers tend to concentrate on testing of specific product components, such as the implementation of a communication protocol (Lonetti et al., 2023; Matheu-García et al., 2019).

Manufacturer Usage Description (MUD) is originally intended for IoT devices to signal the kind of network access they require to function properly (Rekhter and Li, 2019). So far, MUD has not been embraced by the industry and use has not caught up. However, the research community has noted it and many extensions have been proposed to make it more comprehensive for MBT and other uses (Lonetti et al., 2023; Matheu García et al., 2024).

Penetration testing is an expert analysis method, where the expert analyzes the system to find vulnerabilities (Johari et al., 2020). The tester assumes the role of an attacker. *Fuzzing* is a popular and effective security testing method and it can be performed even over the network (Takanen et al., 2018).

The amount of research focusing on the use of common tools for IoT security testing is limited. Kaksonen et al. use common security tools to test security requirements derived from different security specifications (Kaksonen et al., 2023a). Lally and Sgandurra provide a mapping from vulnerabilities identified in the *OWASP IoT* project into testing instructions, methodologies, and tools to use (OWASP, 2018; Lally and Sgandurra, 2018). Rollo lists some tools and techniques applicable to high and medium-level security certification (Rollo, 2017). Dupont et al. presented tool-based automation of risk assessment in an agile iterative development process (Dupont et al., 2023). Tools are run on each iteration for continuous and automatic risk assessment. Tools also are used to automate penetration testing (Abu Waraga et al., 2020; Akhilesh et al., 2022; Siboni et al., 2019). Fuzzing is by nature a tool-based approach (Takanen et al., 2018). Usually, the common tools have a supporting role in research, e.g. for capturing protocol traffic (Matheu et al., 2020).

The use of proprietary functionality, even for common tasks like authentication, makes it difficult to automate IoT security assessment. This is described by Waraga et al. as *"Assessing the security of IoT devices is difficult due to the wide variety and functionality of IoT devices"* (Abu Waraga et al., 2020, p. 15). The same issue is raised by Matheu et al. (Matheu et al., 2020).

None of the aforementioned publications propose implementing security standard testing using the common security tools.

1.4 Research Objectives

The focus of our research is the use of common security tools for IoT security standard testing. As full automation is unlikely to be feasible, we first identify the tests for which automation is a priority. Then, we study which types of common tools are applicable and what is achievable coverage for the test automation.

The structure of the remainder of this article is as follows. In Section 2, we examine the security test specification ETSI TS 103 701 to understand which tests should be automated, which common tools are applicable, and what is achievable automation coverage. In Section 3, we create a proof-of-concept testing framework and present test automation for a real-world IoT product. Finally, we provide discussion and conclusions.

2 STANDARD TESTING AUTOMATION

In this section, we go through a security standard verification and consider automating it. The threat model used in this research is an attack over a network, either from the Internet or from the local network. We exclude non-network-based interfaces from the security analysis. As the most common vulnerabilities in IoT are related to *HyperText Transport Protocol* (HTTP) servers and other network-exposed services, we think this is a reasonable starting point for IoT security testing (Kaksonen et al., 2023b; OWASP, 2018).

Verdicts of automated tests are assigned without manual work by the evaluator. As someone needs to create the automation, this does not mean automatic certification. However, automation may perform repetitive tasks, such as running the same checks for many interfaces or services. Automation also supports continuous standard compliance checks during iterative product development. Especially useful automation should be in re-certification after product updates or certifying product variants.

2.1 Test Specification ETSI TS 103 701

In our analysis, we use ETSI TS 103 701 which is a test specification for security standard ETSI EN 303 645 (ETSI, 2020; ETSI, 2021). The standard is well recognized and the use of an existing specification prevents us from biasing tests to match the available tools. The standard covers customer IoT device security and device interactions with associated services, such as backend services or mobile applications. The associated services themselves are out of scope. ETSI TS 103 701 specifies the following verification process steps (ETSI, 2021):

1. Identification of the *device under test* (DUT)
2. Defining the pro-forma *implementation conformance statement* (ICS)
3. Providing the required *implementation extra information* (IXIT)
4. Verification of the ICS
5. Security assessment and test verdict assignment
6. Assignment of the overall verdict

Steps 1-3 are performed by the vendor and 4-6 by the *test laboratory*. ICS specifies the standard requirements implemented by DUTs and vendor processes. Pro-forma ICS is the proposal from the vendor, which the security assessment then accepts or rejects. IXIT provides extra information required in the security assessment, e.g. network topology, services, software components, and so on. The test laboratory then verifies the requirements and assigns the verdicts.

The ETSI EN 303 645 is made up of 67 security *provisions*, i.e. requirements. Provisions can be mandatory, optional, or recommended. ETSI TS 103 701 defines groups of *test cases* to test the conformance for the provisions. There are *conceptual* and *functional* test cases. A conceptual test checks the conformity of ICS to requirements and a functional test probes the DUT or other artifacts. Many of the tests are difficult to verify automatically without expert judgment. For example, consider the following provision (ETSI, 2020, p. 19).

> **Provision 5.5-1.** The consumer IoT device shall use best practice cryptography to communicate securely.

The provision can be tested by the following test cases and *test units* (ETSI, 2021, p. 52-54):

- Test case 5.5-1-1 (conceptual).
  a) Appropriate security guarantees are specified
  b) The security guarantees are achieved by the used communication mechanism
  c) The communication mechanism uses best practice cryptography
  d) There are no known vulnerabilities for the communication mechanism
- Test case 5.5-1-2 (functional).

a) The implementation is using the specified communication mechanism

The test units 5.5-1-1 a) and b) require expert opinion about the appropriateness and sufficiency of the product design. The verdict assignment is based on the intended usage of the product and IXIT. The test units 5.5-1-1 c) and d) are more mechanical comparison to best practices and known vulnerabilities. They are subject to change as the best practices and attacks evolve. The test unit 5.5-1-2 a) checks that the product implementation is indeed using the specified mechanisms.

Table 1: ETSI TS 103 701 test categories and test unit counts (C is conceptual, F is functional).

Test units		C	F
All, T_A	226	124	102
IXIT and ICS review, T_X	106	106	-
Cross-references, T_C	14	8	6
Product security tests, T_P	106	10	96
- **Security perimeter tests, $\mathbf{T_S}$**	**56**	10	46
- Product UI tests, T_U	50	-	50

Table 2: ETSI TS 103 701 test targets referenced in the product tests T_P.

Test target	Test target description
AuthMech	Authentication mechanisms
UserInfo	User documentation and other info
UpdMech	Software update mechanisms
ReplSup	Isolation and hardware replacement
SecParam	Stored security parameters
ComMech	Communication mechanisms
NetSecImpl	Implementations of network and security functions
SoftServ	Implementations of service authorization and access control
Intf	Network, physical, and logical interfaces of the DUT
SecBoot	Secure boot mechanisms
ExtSens	External sensing capabilities
ResMech	Resilience mechanisms for outages
TelData	Telemetry data
DelFunc	User data deletion functionalities
UserDec	User decisions
UserIntf	User interfaces
ExtAPI	Externally accessible APIs
InpVal	Data input validation methods

There are a total of 226 test units. Table 1 shows the test unit categories, the symbol denoting the category, and the number of test units in it divided into conceptual and functional tests. The symbol refers to both conceptual and functional test units. All test units are denoted with T_A and $|T_A| = 226$. *IXIT and ICS review T_X* tests are for reading and validating the IXIT and ICS. The specification contains some *cross-reference* tests T_C which refer to other test units. We ignore them to avoid double counting. The *product security tests T_P* exercise or probe the product or its components or source code, e.g. the aforementioned 5.5-1-1 c) and d) and 5.5-1-2 a).

The threat model is an attack through a network, and the relevant tests target the network interfaces and the protection layers behind them. We call these *security perimeter tests T_S*. The other product tests are different *User interface (UI)* tests T_U. The focus of this research is the 56 T_S tests. This does not mean that T_U is not important, as e.g. a bad UI may lead users to make decisions bad for the system security.

ETSI TS 103 701 IXIT is made up of 29 different types of information. For example, all communication mechanisms are listed under "*11-ComMech: Communication Mechanisms*" with *identifier, description, security guarantees, cryptographic details, and resilience mechanisms* specified for each mechanism (ETSI, 2021, p. 101). The specification does not provide any generic term for these, we use the term *test target*. The test targets are used in the test specifications to point the focus of the test, e.g. the aforementioned 5.5-1 test units refer to *ComMech*. Table 2 lists the test targets used in T_P.

2.2 Automation of ETSI TS 103 701

We examined the automation potential of ETSI TS 103 701 tests by studying the test unit descriptions, test targets, and features of the available security tools (ETSI, 2021).

The choice of proper tools is mostly affected by the test target. For example, a network tool can probe network interfaces and communication, while a static analysis tool can probe software. Table 3 lists the test targets from T_P, T_S, and T_U with the tool types that can be used to automate the tests. The test and tool types are listed in rows and the test targets in columns. When a test type applies to a test target, the number of relevant test units is given in the table cell. The *Share* column gives the proportion relative to T_P tests, e.g. T_S tests are 53% of T_P and tests by *basic tools* are 27% of T_P.

The tools are divided into *basic*, *advanced*, and *custom* tools. The **basic tools** are network-based and should be easy to deploy in most cases. *Network scan* tools are used to find network hosts and services. There are also search engines for hosts and services on the Internet. *Network capture* tools record network traffic for analysis. *Protocol-specific* tools can check the correctness and use of the best practices on a protocol basis, such as HTTP, *Trasport Layer Security* (TLS), or *Secure SHell* (SSH). An important group

Table 3: ETSI TS 103 701 product test targets mapped to the product security tests T_P, security perimeter tests T_S, types of tests and test tools, and UI tests T_U. Shares given as the percentage of T_P.

Test target	AuthMech	UserInfo	UpdMech	ReplSup	SecParam	ComMech	NetSecImpl	SoftServ	Intf	SecBoot	ExtSens	ResMech	TelData	DelFunc	UserDec	UserIntf	ExtAPI	InpVal	Share
Product security tests $\mathbf{T_P}$	11	18	11	10	3	4	3	3	9	3	3	5	4	11	5	1	1	1	**100%**
Security perimeter tests $\mathbf{T_S}$	8	3	8	5	3	4	3	3	7	2	1	5	2				1	1	**53%**
Basic tools																			27%
Network scan and capture	2		1			1			7	1							1		
Protocol-specific	3		3			3		1											
Web page availability		3		2							1								
Advanced tools																			9%
MITM			1																
Input validation																		1	
Password brute-forcing	1																		
Code analysis							3												
Network/power switch				1								3							
Custom tools																			16%
Access control								2											
Functional	2			2								2	2						
Unauthorized update			3																
Internal properties					3					1									
UI tests $\mathbf{T_U}$																			**47%**
Document review		11		3							1		2			1			
UI usage	3	4	3	1						1				11	5				
Physical inspection				1					2		1								

are the various *Web security tools*. *Web page availability* tool, which can be a simple script, checks that a document is available on the Internet. Together the basic tools cover 52% of T_S and 27% of T_P.

The **advanced tools** require more work to be deployed, as they must be configured for the purpose and can be intrusive and not suitable for all environments. *Man in the middle* (MITM) tools try to break encryption by infiltrating the network connections. *Input validation* tools try to find flaws in the parsing of external input. This is often done by *fuzzing*. *Password brute-forcing* tools check that password-guessing attempts over the network are throttled. *Code analysis* refers to various tools inspecting source code and other development artifacts. *Software composition analysis* (SCA) tools are used to identify the used software components and libraries. SCA tools may also check if the software contains known vulnerabilities. *Network/power switches* are devices which allow programmatically switch network connections or power on and off. They can be used to simulate outages to test DUT behavior in error situations. Together the advanced and basic tools cover 70% of T_S and 37% of T_P.

The **custom tools** must be specifically created on a per-product basis. These tools often require to drive DUT into specific states and coordination with the backend. *Access control* testing verifies that access to critical resources is appropriately granted or denied depending on the provided credentials. *Functional* tests check that a DUT performs an intended functionality. *Unauthorized update* testing tools inject modified update packages to check for the proper response. *Internal* tools check DUT internal components, such as secure storage of critical parameters. If custom tools can be implemented for all tests, the aggregate T_S tool coverage would be 100% and T_P coverage 53%.

The remaining 47% of T_P are **UI tests**. *Document review* ensures that the required aspects are clearly presented in the user documentation. *UI usage* refers to the actual use of the DUT user interface. *Physical inspection* is required to check the physical properties of the DUT casing. It has been assumed that UI tests must be performed manually, but with additional effort they could be automated, as well. A *change detection* tool can check if a document has been updated since a review. *UI automation* tools can be used to drive UI which can be controlled programmatically, e.g. web UI. With change detection tools, the product test T_P automation coverage can be raised to 70% and by adding UI automation up to 96%.

The problem with test coverage is that without UI automation, a user must actively exercise the product to go through its different states and functions. This may be alleviated if the security testing takes advantage of actions performed in other testing. Network captures may be taken while the product is used for other purposes. Non-intrusive tests, e.g. network scans, may be performed in parallel with other uses of the product. Code auditing may take place for normal quality control, but its results can be reused as evidence of security tests. Security testing done by only observing the DUT is called *passive* security testing (Mallouli et al., 2008).

2.3 Mapping Tool Capabilities and Tests

Security tools are unlikely to be specifically designed for security standard testing and the standard tests may not be designed to be automated. Thus, the *tool capabilities* and test verdict assignments are not necessarily well aligned and must be mapped.

Mapping the two can be fairly straightforward, e.g. consider the aforementioned test units 5.5-1-2 a) and 5.5-1-1 c) and d) (ETSI, 2021, p. 52-54). An up-to-date protocol-specific tool can check that the protocol is used, best practice cryptography is applied, and no known vulnerabilities are present.

More mapping logic is required e.g. for the seven *Intf* network interface tests. The tests aim to check that the interfaces match IXIT, no undocumented interfaces exist, interfaces are enabled or disabled as stated, and no extra information is disclosed. For them, the matching tool type is *Network scan and capture*, but these tools produce lists of observed network nodes, services, ports, connections, etc., not verdicts. The automation must compare the tool output to the network interface definition *Intf* and assign test pass or fail verdicts as appropriate.

Automating some test units requires interpretation of the test purposes. For example, there are eight *UpdMech* update mechanism perimeter test units. The test unit 5.3-2-2 states that the test lab must a) "*devise functional attacks to misuse the update mechanism*" and then b) "*attempt to misuse each update mechanism on the base of the devised adverse actions and assess whether the design of the mechanism (...) effectively prevents the misuse*" (ETSI, 2021, p. 34). Automating the test unit a) is likely to require the creation of a custom test tool for misuse attempts of the update mechanism. The tool should be capable of inserting a malicious update package and checking that the update mechanism rejects it. However, common security tools can be used to check the secure delivery of the update packages over the network using standard protocols. Thus, an entry-level b) test is possible with basic tools, e.g. by TLS protocol tool.

The update mechanism test 5.3-6-2 a) is a check if updates are enabled by default, which is checked by observing an update attempt from traffic capture. The tests 5.3-7-1 c) and d) are about proper encryption for update protection, which are checked using protocol-specific tools. The tests 5.3-9-1 c) and d) are for integrity and authenticity verification of the update, which requires the custom tool. Finally, test 5.3-10-1 c) aims to verify that there are no undocumented update channels via network interfaces. It is considered a pass, if there is no unknown network traffic from a DUT. An undocumented update channel inside other expected traffic would not be detected.

It is noteworthy, that an unexpected connection in network capture will make many test cases fail, such as several *Intf* interface tests and the *UdpMech* test 5.3-10-1 c). Thus, there are test failures which are *false positives* and *true positives*, but determining which is which requires an expert analysis. However, this is only required if an impact assessment of the failure is required.

3 CASE STUDY

We implemented a proof-of-concept framework for studying the automation of ETSI TS 103 701 tests. Then we applied the framework in a case study of real-world IoT product testing of Ruuvi Tags and Gateway (Ruuvi Innovations Ltd, 2023). The manufacturer was updated on our activities.

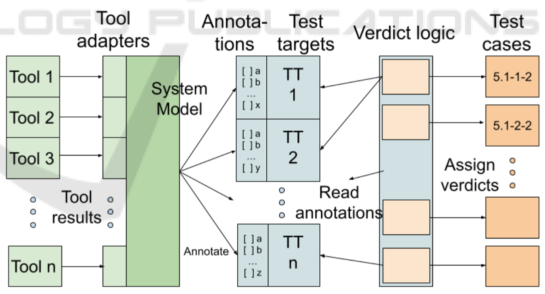

Figure 1: The proof-of-concept framework overview. Tool results are read by tool adapters and mapped into annotations on the test targets. Test case verdict logic reads annotations to assign test verdicts.

3.1 Tool Data Collection and Verdict Assignment

We designed and implemented a proof-of-concept framework which digests output from different security tools, maps the output into test targets, and assigns test verdicts. The framework is passive, the test tools must be invoked separately. Running the tools is outside of the scope of this research.

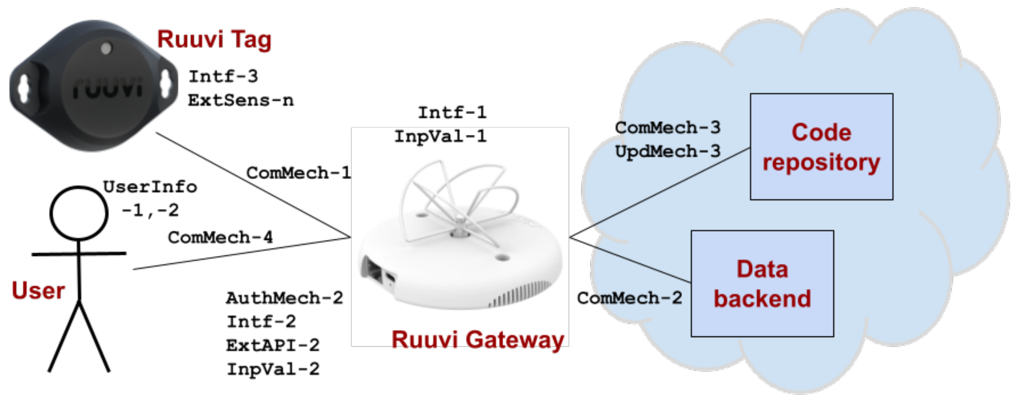

Figure 2: Ruuvi system overview with test targets relevant to the case study.

The functional overview of the framework is given in Figure 1. The output from the tools is read and converted to normalized format by the *tool adapters*. Then, the *system model* is used to find out the relevant test target by an address or other location information. For example, a host service may be identified by IP address, protocol, and port. The system model must contain the network nodes, services, network connections, protocols, software components, etc. The acquisition of the system model is beyond this paper, it can be created for this purpose or a model can be reused (Kaksonen et al., 2023b; Lonetti et al., 2023).

Once the test target is resolved, it is annotated based on the tool results. Annotation indicates the checks performed by the tool. The annotations a tool adapter may assign is dependent on the tool's capabilities. For example, a TLS test tool adapter may assign the annotations `protocol-tls= true`, `tls-best-practices= true`, and `tls-no-known-vulns= true`. The tool can identify TLS services or connections, check if they follow TLS best practices, and whether known vulnerabilities are detected. Table 4 lists the tool capabilities used in the case study. Finally, *verdict logic* contains rules to assign verdicts to tests based on the annotations in the respective test targets. For example, the above annotations may be read by the logic for the aforementioned 5.5-1-1 c) and d) and 5.5-1-2 a).

The proof-of-concept framework is available as open source software. It is hosted in GitHub `https://github.com/ouspg/tcsfw.git`.

3.2 Ruuvi Tags and Gateway IoT

As a case study, we use the proof-of-concept framework for security testing of Ruuvi Gateway and Tags IoT devices (Ruuvi Innovations Ltd, 2023). Figure 2 shows the overview of the Ruuvi system relevant to the case study, including the relevant test targets.

The system is made up of Ruuvi *Bluetooth Low-Energy* (BLE) Tags, which broadcast sensor data **ExtSens**, such as temperature and humidity, over the communication mechanism **ComMech-1**. The Gateway collects the data and uploads it to a cloud service using TLS over **ComMech-2**. The Gateway gets automatic updates **UpdMech-3** over the connection **ComMech-3**. The Gateway is set up and controlled by the User using a browser over an HTTP connection **ComMech-4**. The user is authenticated by a factory-generated password using the mechanism **AuthMech-2**. The user is provided with the *vulnerability disclosure policy* **UserInfo-1** and *published support period* **UserInfo-2**, available on the Internet. The Gateway has HTTP API **ExtAPI-2**. The Gateway has a BLE interface **Intf-1** and Ethernet/WiFi interface **Intf-2**. The Tags have BLE interface **Intf-3**. The Gateway validates input from the BLE broadcasts **InpVal-1** and HTTP **InpVal-2**. The Gateway provides additional connectivity options, but they are not enabled by default and are not included in this presentation.

We target automating the security perimeter tests T_S. The test targets *DelFunc*, *UserDec*, and *UserIntf* only have tests in T_U, thus they are excluded. Also, *ReplSup* was excluded, as it is not relevant for Ruuvi which components can be updated. This leaves 14 test targets to test, of which we implemented automation for seven. The security perimeter tests relevant to the 14 test targets T_S' contains 51 test units.

The Ruuvi product has Finnish Cybersecurity Label, thus it should fulfill selected requirements from ETSI EN 303 645 (Ruuvi Innovations Ltd, 2022). However, the acceptance process does not use ETSI TS 103 701 and ICS and IXIT have not been created. The information we use in the case study is collected from the statement of compliance and by inspecting the product and its documentation (Ruuvi Innovations Ltd, 2022; Ruuvi Innovations Ltd, 2023).

Table 4: Case study tool capabilities, the test targets (and the number of test units) they apply to, and the used tools. The capabilities with tool references a) to f) are automated and marked by green color. Red capabilities are not automated.

Tool capabilities	**Test targets (test units)**
Only the defined connections are present (a	*ComMech*(1), *UpdMech*(2)
Only the defined network nodes are present (b	*Intf*(4)
Only the defined services are present (b	*AuthMech*(2), *ExtAPI*(1), *Intf*(3)
Document is available on the Internet (c	*ExtSens*(1), *ReplSup*(2), *UserInfo*(3)
DUT basic functions work	*ReplSup*(2), *ResMech*(2)
Code review has been performed	*NetSecImpl*(2)
Bad input is rejected (fuzzing)	*InpVal*(1)
MITM attacks are detected (d	*ComMech*(1), *UpdMech*(1)
Malicious update is rejected	*UpdMech*(3)
Boot with modified firmware fails	*SecBoot*(1)
Critical parameter modification	*SecParam*(1)
Default password is strong	*AuthMech*(2)
TLS protocol is used (e TLS best practices are followed (e No TLS vulnerabilities are found (e	*ComMech*(3), *UpdMech*(3)
Only the defined SW is used (SCA)	*NetSecImpl*(1)
Data storage is secure	*SecParam*(2)
Telemetry data is properly used	*TelData*(2)
Web best practices are used (f	*InpVal*(1)
Network or power disconnect	*ReplSup*(1), *ResMech*(3)
Defined auth. protocol is used	*AuthMech*(1), *SoftServ*(1)
Auth. best practices are followed	*AuthMech*(1)
Auth. brute-forcing is prevented	*AuthMech*(1)
No auth. vulnerabilities are found	*AuthMech*(1)
Access is granted/rejected properly	*SoftServ*(2)
Used tools: a) Tcpdump, Hcidump, b) Nmap, Censys, Tcpdump, Hcidump, c) Python script, d) Mitmproxy, e) Testssl.sh, f) ZAP	

Common security tools are used to implement the test automation. Table 4 lists the tool capabilities, the test targets (with test unit counts), and the names of the used tools. The capabilities with a tool references a) to f) are automated. For example, the capability "*Only the defined connections are present*" is applied to one communication mechanism *CommMech* test unit and two update mechanism *UpdMech* test units. This capability is provided using tools *Tcpdump* and *Hcidump*.

The used tools and their descriptions and home pages are listed in Table 6. The local networks are scanned with the *Nmap* tool. The remote backend information is fetched from the *Censys* Internet search engine. IP network traffic is captured by *Tcpdump* and BLE traffic by *Hcidump* tool. TLS services are checked using *Testssl.sh*. TLS session man-in-the-middle attacks are attempted using *MITMProxy*. The availability of web-based documentation is checked by a Python script. Web services are scanned by *ZED attack proxy (ZAP)*. The collection of network traffic requires the product to be used to trigger the various product functions. The following actions are performed.

1. The Ruuvi Tags and Gateway are powered up.
2. The user connects to special setup WiFi of the Gateway, configures the system, and disconnects.
3. The Gateway is connected to the Data backend and Code repository for data upload and updates.

For data collection, we use a Raspberry Pi 3 as WiFi gateway between the Gateway, User browser, and Internet (Raspberry Pi Ltd, 2023). The setup allows to record and control all traffic between DUTs and the Internet. The tools are run manually, either from a normal computer or from the Raspberry Pi, and the tool output is collected for analysis. The resulting network traffic is captured and stored with other tool outputs. The data is analyzed by the framework to assign verdicts for T_S' test units.

Table 5: Ruuvi case study test targets, covered test units / all units, and test target explanations T_S'. The green and red color marks the targets covered and not covered by the tools, respectively.

Test targets	**Test units**	**Explanation**
AuthMech	2 / 8	HTTP server at Gateway
UserInfo	3 / 3	Two Internet-provided documents
UpdMech	5 / 8	Gateway updates by TLS
SecParam	- / 3	
ComMech	4 / 4	BLE, HTTP, 2 x TLS
NetSecImpl	- / 3	
SoftServ	- / 3	
Intf	7 / 7	Gateway and Tag interfaces
SecBoot	- / 2	
ExtSens	1 / 1	Tag sensors
ResMech	- / 5	
TelData	- / 2	
ExtAPI	1 / 1	Gateway HTTP API
InpVal	- / 1	Gateway HTTP and BLE
Tests T_S'	**45%**	Covered 23 / 51 test units

3.3 The Achieved Coverage

Table 5 summarizes the achieved coverage. From the 51 Ruuvi security perimeter tests T_S', 23 are automated for 45% coverage.

Table 6: The used security tools, short tool descriptions, and tool home pages.

Name	Description	Home page
Apktool	Android APK analysis tool	https://ibotpeaches.github.io/Apktool/
Black Duck	SCA service (commercial)	https://protecode-sc.com/
Censys	Internet search service	https://search.censys.io
Hcidump	Bluetooth Low Energy recorder	https://www.bluez.org
Nmap	Network scanner	https://nmap.org
Ssh-audit	SSH test tool	https://github.com/jtesta/ssh-audit
Testssl.sh	TLS test tool	https://testssl.sh
Tcpdump	Traffic recorder	https://www.tcpdump.org
ZAP	Web security scanner	https://www.zaproxy.org/

More automation could be implemented to increase the coverage. Adding authentication protocol and brute-force tests, static code analysis, and input validation test would raise the $T_S^{'}$ coverage to 63%.

With custom tools for testing access control, unauthorized update handling, security boot, and telemetry, the coverage would be 80%.

Adding automation tests for network and power interruptions, e.g. using controllable power and network switches, would add 10 %-points.

ETSI TS 103 701 includes a review of the vendor processes for monitoring vulnerabilities and issuing security updates, but no tests to look for vulnerabilities in the product software. The scope is limited to IoT devices, the mobile applications and backend servers are excluded. We implemented the following security testing automation beyond these limitations:

- Verification of the backend service security posture by *Censys*, *Testssl.sh*, and *Ssh-audit* tools.
- Using a SCA tool to extract the software bill of materials (SBOM) for verification and known vulnerability checking using *Black Duck* service (commercial).
- Inspecting communication content between User browser and other services by *HAR* data collected by Web browser (Odvarko, Jan, 2007).
- Unpacking Ruuvi Mobile Application by *Apktool* tool to check the application permissions.
- Parsing of release history of software components in Code Repository by *Github* API to resolve the update frequency (GitHub Inc., 2023).

The used tools and their descriptions are in Table 6.

4 DISCUSSION

We analyzed the ETSI TS 103 701 security test specification to study the proportion of the tests which could be automated with common security tools (ETSI, 2021). Of the 226 test units in the specification, 106 are product tests and 120 are inspections of the conformance documentation. Of the product tests, 53% are security perimeter tests relevant to the threat model of network-based attacks and 47% are different types of UI tests. Basic network security tools allow the automation of 52% of the security perimeter tests and 70% coverage is reachable by the use of advanced tools. With custom tools, the perimeter automation coverage can rise towards 100%. The UI tests are made up of user documentation reviews, actual UI tests, and physical inspection. These could also be covered by automation, if suitable tools are available or can be created.

Secondly, we created a proof-of-concept framework, which assigns test verdicts based on output from common security tools and performed a case study with Ruuvi real-world IoT product. In the case study, 45% of the security perimeter test verdict assignments were automated using network-based tools. This could feasibly rise to 90% by introducing more advanced and custom tools.

The proof-of-concept framework we created is itself passive, it reads output from different tools. Security testing effort may be decreased by mixing the invocation of the security tools with other testing activities. For example, traffic captures can be recorded whenever the system is exercised. This lessens the cost of security testing and increases the chances that security is assessed. The downside is that the test coverage is decreased if not all product functionality is exercised. However, some security tests appear better than no security tests. When higher coverage is required, dedicated security testing must be performed.

Automated security tests enable security verification without security experts and security self-certification. Tests that failed during the verification must be resolved. A failure may indicate a security vulnerability, a security-neutral change in a DUT, or a flaw in the automation. As many tests are based on the same result data set, a single root cause may manifest itself as many test case failures, e.g. an unexpected network connection may cause many different interface tests to fail. The proportion of flaws by bugs in the test automation is hard to estimate but in a study of unit tests in Java code 31-53% of failures were attributed to tests rather than being a real bug (Hao et al., 2013). Discovered vulnerabilities must be fixed by the vendor, but fixing the tests may require the original author of the automation to be available. The resolution of test failures in automated self-certification requires a new process, warranting a future research topic.

4.1 Efficiency and Coverage

The creation of automation for security tests is additional work. The exact effort is hard to estimate, as it depends on the product and extend of test automation. Impact of the required work can be managed by starting from testing of basic network features and extending towards full certification automation. Regular testing by the automation helps to avoid regression in security.

The impact on cost and delay is compensated when several test targets, product versions, or configurations must be tested. The use of security tools reduces the effort to create the tests. Work can be reduced by reusing tests between projects. The reuse can be by the vendor or the testing laboratory, or by having a library of test cases for a standard. A solid benefit from automation should come from recertification of the product after updates, which may be self-certification if the coverage of the automatic test cases is sufficient. The certification of product variants should receive similar benefits.

Automated security tests should be at least as thorough as the manual tests would be. Ideally, the automation increases coverage by performing more checks than a manual tester can accomplish. Further, a protocol-specific security tool may be able to perform more extensive verification than a test created for one product only. An up-to-date tool tests the product against the latest best practices and known vulnerabilities. However, the use of a generic tool may leave some aspects of a security requirement untested, unless the test is customized or accompanied by manual verification. In the general case, the efficiency and coverage of automation in security standard testing remains an open question.

The use of open source and other tools for security standard testing raises the question of how much trust we can place on them. This should be compared to the level of trust we are placing in the evaluators performing the assessment work. A maintained and commonly used tool, e.g. *Nmap*, can likely be trusted to give accurate information. Results from a tool without a good reputation should be more carefully reviewed. It would be best to have at least two tools for a testing capability in case one of them becomes unmaintained and is no longer updated. As there should be a regular assessment of the quality of work of the evaluators, there should be regular reviews about the quality of used tools. To promote the quality of open source tools used in commercial certifications, the projects behind them should be supported.

4.2 Improving Security Testing Automation Coverage

Although we automated a significant portion of the relevant tests, there is room for improvement. To raise the coverage, we propose several improvements:

- IoT products should use more standard components and protocols that have common tools available for testing. For example, authentication may be difficult to test due to nonstandard implementation.
- Security requirements should be made more testable and automation-friendly. For example, a description of the attack surface should be required, as it can be verified by tools.
- New tools should be created to fill the gaps between requirements and automated testing.

There are IoT frameworks which provide common functionality for IoT products (Ammar et al., 2018; CSA, 2022). In the best case, this could increase the testability of IoT, if the frameworks are accompanied by tools to verify the proper use of the framework security features. This could cover authentication, updates, data encryption, and other framework functions.

Most existing IoT security standards share a common set of requirement categories, but there is a lot of variation in the requirement details (Kaksonen et al., 2022). The majority of the requirements are present only in one or a few specifications. This makes it hard to comply with many specifications. If those requirements can be checked by automation, then automation mitigates the problem, but if they all call for custom tools, then they hinder automation.

All security standards and test specifications should be available in machine-readable format. For example, the Common Criteria is available as XML documents (CC, 2022). This provides a good starting point for automation, as the information does not need to be extracted from PDF documents or such. The authors of this paper have found the extraction of machine-readable data from specification documentation especially cumbersome and error-prone.

4.3 Comparison to Related Studies

The existing research of IoT security testing automation emphasizes MBT to generate the tests (Lonetti et al., 2023). We employ common security tools, which enables us to simplify the overall system model, as custom test cases get replaced by the capabilities of the tools. This approach allows us to present a more comprehensive automation of security standards, while the existing research predominantly offers only partial experiments. Test tools are used by some existing research (Lally and Sgandurra, 2018; Kaksonen et al., 2023a; Abu Waraga et al., 2020). Compared to them, we map the tool capabilities to security standard tests.

We agree that it is difficult to test the IoT systems due to their heterogeneity (Abu Waraga et al., 2020; Matheu et al., 2020). We also agree that the use of standard IoT technologies would support increasing test automation for IoT security testing (Lonetti et al., 2023). However, we see that this could happen through new security tools which can test standard implementations, rather than using standard models for test case generation. A share of the future research should be directed towards improving the testability of the IoT, e.g. by standard components, protocols, and frameworks, rather than trying to test whatever constructions the IoT vendors come up with.

4.4 Threats to Validity

We base our research on ETSI EN 303 645 security standard and ETSI TS 103 701 test specification. All findings may not apply to other standards. When using security standards that don't specify test cases, their design is additional work. As the ETSI EN 303 645 emphasizes generic network security requirements, there are many well-known high-quality tools to use to implement the automation. The creation of automated tests may be far more challenging for sector-specific standards which focus on processes and design security, or emphasize the security of internal components. Future research on automation of other security standards would provide more insight into this question.

Even with well-specified tests, there is ambiguity on the exact evidence required to assign a verdict. This may increase or decrease the coverage of automation, depending on the chosen interpretation. We chose to automate the tests from the established specification ETSI TS 103 701 to avoid biasing tests which are easy to automate. From these tests, we suggest the use of a tool when it can significantly cover the intent of the test. In the future, if the tool developers and standard requirement authors get more aligned, it will be easier to see which tests can be automated by tools and which cannot be.

5 CONCLUSIONS

This paper appears to be the first attempt to automate IoT security standard testing using common security tools. We achieved high automation coverage in the security perimeter tests relevant to the threat model of network attacks. We created a proof-of-concept framework for automated security testing and provided a real-world case study. Our research shows that automation could help to solve some of the major problems in IoT certification, the lack of expert personnel and the cost and delays associated with the certification of multiple product versions and configurations.

ACKNOWLEDGMENTS

This work is supported by Finnish Scientific Advisory Board for Defence (MATINE/2500M-0152).

REFERENCES

Abu Waraga, O., Bettayeb, M., Nasir, Q., and Abu Talib, M. (2020). Design and implementation of automated IoT security testbed. *Computers & Security*, 88:101648.

Akhilesh, R., Bills, O., Chilamkurti, N., and Mohammad Jabed, M. C. (2022). Automated Penetration Testing Framework for Smart-Home-Based IoT Devices. *Future Internet*, 14(10):276.

Ammar, M., Russello, G., and Crispo, B. (2018). Internet of Things: A survey on the security of IoT frameworks. *Journal of Information Security and Applications*, 38:8–27.

BSI (2023a). Baseline Requirements for consumer IoT devices. Federal Office for Information Security, https://www.bsi.bund.de/dok/ciot-standard.

BSI (2023b). BSI TR-03148-P: Test Specification Conformance Tests for Secure Broadband Routers. Test specification, Federal Office for Information Security, Germany.

BSI (2023c). BSI TR-03148: Secure Broadband Router. Requirements, Federal Office for Information Security, Germany.

CC (2022). Common Criteria for Information Technology Security Evaluation, Parts 1-5, 2022, Revision 1. Standard.

Cirne, A., Sousa, P. R., Resende, J. S., and Antunes, L. (2022). IoT security certifications: Challenges and potential approaches. *Computers & Security*, 116:102669.

CSA (2022). Matter security and privacy fundamentals, v. 1.0. Technical report, Connectivity Standards Alliance.

CSA (2023). Cybersecurity Labelling Scheme. Cyber Security Agency of Singapore, https://csa.gov.sg/.

Dupont, S., Yautsiukhin, A., Ginis, G., Iadarola, G., Fagnano, S., Martinelli, F., Ponsard, C., Legay, A., and Massonet, P. (2023). Product Incremental Security Risk Assessment Using DevSecOps Practices. In *Computer Security. ESORICS 2022 International Workshops*, pages 666–685, Cham. Springer International Publishing.

ECSO (2017). European Cyber Security Certification, A Meta-Scheme Approach v1.0. WG1 – Standardisation, certification, labelling and supply chain management, ESCO.

ETSI (2020). Cyber Security for Consumer Internet of Things: Baseline Requirements v2.1.1. ETSI EN 303 645, ETSI.

ETSI (2021). Cyber Security for Consumer Internet of Things: Conformance Assessment of Baseline Requirements v1.1.1. ETSI TS 103 701, ETSI.

GitHub Inc. (2023). GitHub REST API documentation. https://docs.github.com/en/rest.

Hao, D., Lan, T., Zhang, H., Guo, C., and Zhang, L. (2013). Is this a bug or an obsolete test? In Castagna, G., editor, *ECOOP 2013 – Object-Oriented Programming*, pages 602–628, Berlin, Heidelberg. Springer Berlin Heidelberg.

ioXt Alliance (2023). ioXT Internet of secure things. https://www.ioxtalliance.org/.

Johari, R., Kaur, I., Tripathi, R., and Gupta, K. (2020). Penetration Testing in IoT Network. In *2020 5th International Conference on Computing, Communication and Security (ICCCS)*, pages 1–7.

Kaksonen, R., Halunen, K., Laakso, M., and Röning, J. (2023a). Transparent security method for automating IoT security assessments. In *Information Security Practice and Experience*, pages 138–153, Singapore. Springer Nature Singapore.

Kaksonen, R., Halunen, K., and Röning, J. (2022). Common Cybersecurity Requirements in IoT Standards, Best Practices, and Guidelines. In *Proceedings of the 7th International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS,*, pages 149–156. INSTICC, SciTePress.

Kaksonen, R., Halunen, K., and Röning, J. (2023b). Vulnerabilities in IoT Devices, Backends, Applications, and Components. In *ICISSP - 9th International Conference on Information Systems Security and Privacy*. INSTICC, SciTePress.

Kaksonen, R., Järvenpää, T., Pajukangas, J., Mahalean, M., and Röning, J. (2021). 100 Popular Open-Source Infosec Tools. In *36th IFIP International Conference on ICT Systems Security and Privacy Protection (SEC)*, volume AICT-625 of *ICT Systems Security and Privacy Protection*, pages 181–195. Springer International Publishing.

Khurshid, A., Alsaaidi, R., Aslam, M., and Raza, S. (2022). EU Cybersecurity Act and IoT Certification: Landscape, Perspective and a Proposed Template Scheme. *IEEE Access*, 10:129932–129948.

Lally, G. and Sgandurra, D. (2018). Towards a Framework for Testing the Security of IoT Devices Consistently. In Saracino, A. and Mori, P., editors, *Emerging Technologies for Authorization and Authentication*, pages 88–102, Cham. Springer International Publishing.

Lonetti, F., Bertolino, A., and Di Giandomenico, F. (2023). Model-based security testing in IoT systems: A Rapid Review. *Information and Software Technology*, 164:107326.

Mallouli, W., Bessayah, F., Cavalli, A., and Benameur, A. (2008). Security Rules Specification and Analysis Based on Passive Testing. pages 2078–2083.

Matheu, S. N., Hernández-Ramos, J. L., Skarmeta, A. F., and Baldini, G. (2020). A Survey of Cybersecurity Certification for the Internet of Things. *ACM Comput. Surv.*, 53(6).

Matheu-García, S. N., Hernández-Ramos, J. L., Skarmeta, A. F., and Baldini, G. (2019). Risk-based automated assessment and testing for the cybersecurity certification and labelling of IoT devices. *Computer Standards & Interfaces*, 62:64–83.

Matheu García, S. N., Sánchez-Cabrera, A., Schiavone, E., and Skarmeta, A. (2024). Integrating the manufacturer usage description standard in the modelling of cyber–physical systems. *Computer Standards & Interfaces*, 87:103777.

Odvarko, Jan (2007). HTTP Archive 1.2 Specification. Software is hard, http://www.softwareishard.com/blog/har-12-spec/.

OWASP (2018). OWASP Top 10 Internet of Things. https://wiki.owasp.org/index.php/OWASP_Internet_of_Things_Project.

Raspberry Pi Ltd (2023). Raspberry PI. https://www.raspberrypi.com/.

Rekhter, Y. and Li, T. (2019). RFC 8520: Manufacturer Usage Description Specification. RFC, RFC Editor.

Rollo, J. (2017). D1.2 List of tools and techniques applicable for high and medium assurance for efficient assurance. Report DS-01-731456 / D1.2 / V1.0, Project: Compositional security certification for medium to high-assurance COTS-based systems in environments with emerging threats.

Ruuvi Innovations Ltd (2022). *Statement of compliance for the Cybersecurity Label, 2022-06-06*.

Ruuvi Innovations Ltd (2023). Ruuvi home page. https://ruuvi.com.

Siboni, S., Sachidananda, V., Meidan, Y., Bohadana, M., Mathov, Y., Bhairav, S., Shabtai, A., and Elovici, Y. (2019). Security Testbed for Internet-of-Things Devices. *IEEE Transactions on Reliability*, 68(1):23–44.

Takanen, A., Demott, J., Miller, C., and Kettunen, A. (2018). *Fuzzing for Software Security Testing and Quality Assurance, Second Edition*.

Traficom (2023). Finnish Cybersecurity Label. Finnish Transport and Communications Agency Traficom, https://tietoturvamerkki.fi/en/.