

xG-ALOE
Alliance for Open Enterprise Networks

PLUGFEST 2026
TEST REPORT

Results of the second xG-ALOE Plugfest
17–25 September 2026, Berlin

EANTC
European Advanced Networking Test Center

Fraunhofer HHI

Contents

Acronyms iii

Introduction 1

The Plugfest at a Glance . . . 2

1 Deterministic Timing in 5G 3

1.1 Use Case Description . . . 4

1.2 Test Cases . . . 4

1.3 Technical Challenges and Limitations . . . 9

1.4 Summary . . . 10

1.5 Outlook . . . 10

2 Security Assurance for 5G Campus Components 12

2.1 Test Tool . . . 13

2.2 Security Risks Covered . . . 16

2.3 Test Execution Procedure . . . 16

2.4 Participants . . . 18

2.5 Results . . . 19

2.6 Analysis . . . 30

2.7 Technical Challenges and Limitations . . . 31

2.8 Summary . . . 32

3 Hardware Acceleration Comparison for Private 5G 33

3.1 Device under Test . . . 34

3.2 Test Cases . . . 34

3.3 Testbed Setup . . . 34

3.4 Results . . . 36

3.5 Analysis . . . 42

3.6 Technical Challenges and Limitations 44
3.7 Summary 45
3.8 Outlook 45

Final Summary & Conclusion **46**

Lessons Learned and Next Steps 47

References **48**

Glossary **49**

Acronyms

1PPS	One pulse per second
3GPP	3rd Generation Partnership Project
AGV	Automated guided vehicle
API	Application programming interface
AST	Abstract syntax tree
AVX-512	Advanced Vector Extensions, 512 bit (Intel CPU instruction set)
AWGN	Additive white Gaussian noise
BIOS	Basic input/output system
BMC	Baseboard management controller
CLI	Command-line interface
COTS	Commercial off-the-shelf
CPU	Central processing unit
CSRF	Cross-site request forgery
CUT	Code under test
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DPDK	Data Plane Development Kit
DS-TT	Device-side TSN translator
DTD	Document type definition
DU	Distributed unit
FPGA	Field-programmable gate array
gNB	Next-generation Node B (5G base station)
GNSS	Global navigation satellite system
gPTP	Generalized Precision Time Protocol (IEEE 802.1AS)
LDPC	Low-density parity-check (code)
LLM	Large language model
LoC	Lines of code
MCS	Modulation and coding scheme
MIMO	Multiple input, multiple output
MTIE	Maximum time interval error
MVP	Minimum viable product
NR	New Radio (5G air interface)
NTP	Network Time Protocol
NVD	National Vulnerability Database
NW-TT	Network-side TSN translator
O-DU	O-RAN distributed unit
O-RU	O-RAN radio unit
ORS	Open Radio Systems
OSV	Open Source Vulnerabilities (database)
OWASP	Open Worldwide Application Security Project

PCIe	Peripheral Component Interconnect Express
PHC	PTP hardware clock
PPS	Pulse per second
PTP	Precision Time Protocol (IEEE 1588)
RAN	Radio access network
RMS	Root mean square
RPU	Radio processing unit
RTT	Round-trip time
SARIF	Static Analysis Results Interchange Format
SAST	Static application security testing
SBOM	Software bill of materials
SCA	Software composition analysis
SFP	Small form-factor pluggable
SIB9	System Information Block Type 9
SNR	Signal-to-noise ratio
SPDX	Software Package Data Exchange
SSRF	Server-side request forgery
TDD	Time division duplex
TE	Time error
TLS	Transport Layer Security
TOCTOU	Time of check to time of use
TSN	Time-Sensitive Networking
TWAMP	Two-Way Active Measurement Protocol
UE	User equipment
UPF	User plane function
UTC	Coordinated Universal Time
VDR	Vulnerability disclosure report
vRAN	Virtualized RAN
XML	Extensible Markup Language
XSS	Cross-site scripting
XXE	XML external entity

Introduction

The second xG-ALOE Plugfest took place from 17 to 25 September 2026 at the Fraunhofer Heinrich Hertz Institute (HHI) in Berlin, organized together with EANTC AG, which supported the test plan creation, test execution, results verification, and reporting. Vendors, research institutions, and universities came together to test their private 5G technologies side by side and to see how well their Open RAN components work together in practice. The participating organizations were airpuls, Benetel, brown-iposs, Fraunhofer HHI, Hochschule Offenburg, Open Radio Systems, Technische Universität Dresden, and Universität Passau/CipherCell. The Plugfest is part of the xG-ALOE initiative of Fraunhofer HHI, which offers companies and research groups a shared infrastructure for developing and evaluating such technologies. Especially for startups and small companies, this shared environment offers a practical approach to address integration challenges and improve their products.

Three test scenarios were executed during the event. The first dealt with time synchronization. It measured how closely two UEs agree on time when they are aligned via SIB9, and how accurately a gPTP time reference is delivered across the 5G system. The second scenario looked at the security of the software itself. Four components provided by participants, about 2 million lines of code in total, were analyzed with Universität Passau/CipherCell's Security Test Platform using static analysis, dependency analysis, and autonomous penetration testing. The third scenario examined whether offloading the uplink LDPC decoding to a hardware accelerator pays off for an open-source RAN stack. For this purpose, the ORS AURORA decoder was compared with an AVX-512 software implementation and the Intel ACC100 card in terms of throughput, latency, and power efficiency.

Some technical challenges and limitations were faced during this Plugfest. Working through such problems together, finding their causes, and adjusting both the implementations and the test procedures was a central benefit of the event. The findings are documented in this report so that future plugfests can build on them and resolve similar issues more quickly, and so that participating vendors can further improve and optimize their products. The following sections describe these test scenarios in detail, including their test setups, procedures, and results.

The Plugfest at a Glance

1 Deterministic Timing in 5G

Time synchronization of UEs via SIB9 and transport of gPTP time through the 5G system.

±136 ns
maximum relative time error between two UEs (45 min)

−885 to +770 ns
gPTP client offset in steady state (client-reported, spike intervals excluded)

2 spikes
of up to 220 µs, most likely caused by the PPS connection – to be confirmed

Chapter 1

2 Security Assurance for 5G Campus Components

Source code of four components (about 2 million lines) checked with static analysis, dependency analysis and an autonomous penetration-testing agent.

140
confirmed findings in the source code

138
known vulnerabilities in 2,241 third-party components, 75 of them critical or high

89
findings from autonomous penetration testing

Chapter 2

3 Hardware Acceleration Comparison

Uplink LDPC decoding on ORS AURORA and Intel ACC100 vs. AVX-512 software, with 1 and 10 worker threads.

10 threads
same throughput for all; the software decoder is the most energy-efficient

1 thread
at low SNR the accelerators keep 151–157 Mbit/s (software: 70 Mbit/s), about 1.8× the efficiency

−16 W
AURORA vs. ACC100 at the same throughput (BMC resolution 4 W)

Chapter 3

Participants

airpuls, Benetel, brown-iposs, EANTC AG, Fraunhofer HHI, Hochschule Offenburg, Open Radio Systems, Technische Universität Dresden and Universität Passau/CipherCell. EANTC AG supported the test plan creation, test execution, results verification and reporting.

1 Deterministic Timing in 5G

Private 5G campus networks see increasing deployment and usage, offering deterministic latency, high reliability, and local data sovereignty. Applications such as robot coordination, motion control, distributed data acquisition, and time-correlated sensor data evaluation require a precise, shared time base for all devices. Traditionally, this is achieved using mechanisms such as the Global Navigation Satellite System (GNSS), Network Time Protocol (NTP), or IEEE 1588 Precision Time Protocol (PTP). However, these solutions add installation and configuration complexity and are often unsuitable for indoor or industrial environments where GNSS reception is unavailable.

According to 3GPP specifications, the 5G system can serve as a time distribution infrastructure. The gNodeB (gNB) broadcasts reference time information in System Information Block Type 9 (SIB9) [8], allowing each User Equipment (UE) in the cell to synchronize its local clock to the network time without dedicated signaling. For deterministic and time-sensitive applications, the 5G system can also be integrated into a generalized Precision Time Protocol (gPTP, IEEE 802.1AS [3]) domain using network-side and device-side translators (NW-TT and DS-TT). In this setup, the gPTP grandmaster time is transparently transmitted over the 5G system to gPTP clients connected to the UE, making the 5G network function as a time-aware bridge.

In this test, the effectiveness of SIB9-based time synchronization for UEs in supporting time-sensitive applications within a private 5G campus network is being evaluated. The primary focus is on the relative timing accuracy between two UEs, since many multi-device use cases depend more on inter-device agreement than on absolute alignment to Coordinated Universal Time (UTC).

Additionally, the end-to-end gPTP accuracy between the gPTP grandmaster and the gPTP client behind the UE is being assessed. This evaluation shows how accurately the grandmaster time is reproduced at the device side after transmission through the 5G system, and whether this accuracy meets the requirements of typical industrial use cases.

The involved participants in this scenario were airpuls, Hochschule Offenburg and Technische Universität Dresden.

1.1 Use Case Description

A representative use case for SIB9-based time synchronization is synchronized data acquisition in test and monitoring environments, such as test benches or structural health monitoring. In these scenarios, multiple distributed sensors measuring vibration, strain, or acoustic signals are each connected via their own UE and report timestamped measurement values. For analyses such as phase comparison, propagation delay estimation, or sensor fusion, what matters is not how closely each device is aligned to UTC, but how well the timestamps of the individual devices agree with each other. The relative timing accuracy between UEs, therefore, directly determines the quality of the combined measurement data, and any offset between devices translates into a systematic error in the evaluation. Since the sensors are typically connected to the UE via Ethernet rather than being part of the modem itself, gPTP provides the standard interface through which the UE's time is passed on. The sensors' timestamping clocks are disciplined by the gPTP client, so that the measurement timestamps agree across devices, and the gPTP servo additionally smooths out short-term jitter on the radio link.

A second use case is coordinating multiple mobile robots and automated guided vehicles (AGVs) in indoor industrial environments. When several vehicles jointly carry a load, coordinate their paths in time, or fuse sensor data, such as LiDAR scans, into a shared environmental model, they depend on a common time base. Wired PTP is not applicable to mobile devices, and GNSS is typically unavailable inside halls and buildings. A 5G campus network can close this gap by distributing time wirelessly to all devices in the cell. In this case, gPTP disciplines the onboard controllers behind the UE and allows the 5G system to be integrated as a time-aware bridge into an existing gPTP domain. The mobile devices can thus share the same time base as the stationary infrastructure, such as fixed machines or cameras, which are already synchronized to the gPTP grandmaster over wired connections. The accuracy of the gPTP time delivered behind the UE is therefore decisive for how precisely the onboard systems can be aligned to the common time reference.

Both use cases require that multiple wireless devices agree on time with high precision over extended periods, without a dedicated synchronization infrastructure. In both cases, SIB9 aligns the UE with the 5G system time, while gPTP carries this time to application devices via a standard, widely supported mechanism, enabling integration into an existing time domain without modification. This makes the relative timing accuracy between UEs and the gPTP accuracy between the grandmaster and the client the key metrics for assessing whether SIB9-based synchronization is suitable for such applications.

1.2 Test Cases

To evaluate the two goals of this scenario, two separate test cases were set up, each with its own topology and measurement method. The first scenario isolates the synchronization of the UEs themselves by measuring the relative time error between two UEs aligned to network time via SIB9. The second scenario extends the setup with a gPTP domain and measures the synchronization accuracy between a gPTP grandmaster and a gPTP client, with gPTP messages carried over the 5G system. While the first scenario evaluates the quality of UE-side synchronization on its

own, the second shows the accuracy a device behind the UE experiences.

1.2.1 Test Case 1: Relative Timing Accuracy via SIB9

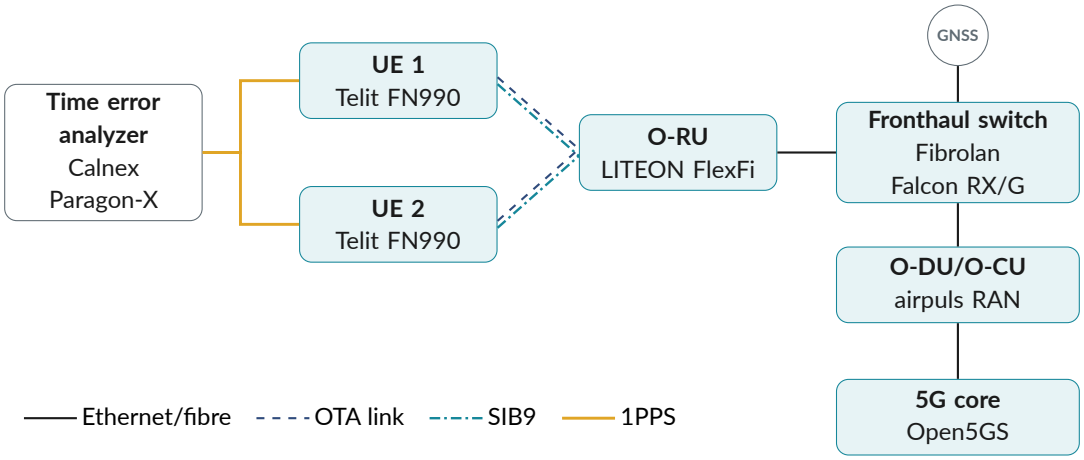

Figure 1: Testbed Setup – Relative Timing Accuracy with SIB9

The first test case (Figure 1) evaluates the relative timing accuracy between two UEs synchronized to network time via SIB9. Both UEs are of the same model (Telit FN990), are attached to the same O-RU, and receive the SIB9 reference time information over the air. The RAN is referenced to a GNSS receiver, which provides the network's common time source.

To measure the synchronization quality independently of any application layer, the 1PPS output of each UE is connected to a time analyzer. The analyzer measures the time interval between the two 1PPS signals, which corresponds to the relative time error between the UEs. Recording this value over an extended period allows both the static offset and its variation over time (jitter and wander) to be evaluated. Since both UEs derive their time from the same SIB9 broadcast, this scenario isolates the RAN's synchronization performance from the UEs. The result directly indicates how well multiple wireless devices agree on time, which is the key metric for the use cases described above.

Results

For the first test case, the relative time error between the two UEs was measured using their 1PPS outputs over 45 minutes at 1-second intervals. Table 1 summarizes the results; here, jitter denotes the deviation of the relative TE from its mean value.

Table 1: 1PPS Relative Time Error Measurement Results for SIB9

Metric	Value [ns]
Mean |relative TE|	48.5
Maximum relative TE	+136
Minimum relative TE	-129
Standard deviation of relative TE (RMS jitter)	39.7
Mean |jitter|	30.7

The results can be put into context with the relevant specifications. In 5G NR TDD systems, 3GPP (TS 38.133 [1]) and ITU-T G.8271 [4] require a maximum time error of 3 µs between neighboring cells. This corresponds to ±1.5 µs with respect to a common reference. The end-to-end requirement of ±1.5 µs is split in G.8271.1 [5] into a network limit of 1.1 µs and a 400 ns allowance for the end equipment.

The relative time error measured between the two UEs is small compared to the requirements of the use cases described above. Over the 45-minute measurement, the mean magnitude of the relative TE was 48.5 ns, with extremes of +136 ns and −129 ns. This corresponds to a Maximum Time Interval Error (MTIE) of 265 ns. For the sensor data acquisition use case, a relative error of this order has little practical effect. At a vibration frequency of 10 kHz, for example, a timing offset of 100 ns corresponds to a phase error of only 0.36°. The maximum relative TE of 136 ns is about a factor of 11 below the 1.5 µs end-to-end limit and about one third of the 400 ns allowance for the end equipment.

Analysis

In the first test case, the low relative time error can be explained by the test's architecture. Both UEs are attached to the same O-RU and evaluate the same SIB9 broadcast and the same downlink frame timing. All error contributions that are common to both devices, such as the GNSS reference, the O-DU/O-RU synchronization, and the fronthaul, therefore cancel out in a relative measurement. The remaining error consists only of UE-specific contributions. These include the estimation of the downlink frame boundary and the compensation of the propagation delay via the timing advance, the resolution and tracking behavior of the modem's internal clock, and the generation of the 1PPS signal. Since these contributions are independent for the two UEs, they add up in the difference. If both UEs behave similarly, the jitter of a single UE is therefore probably lower than the measured 39.7 ns, in the order of 28 ns (39.7 ns / √2). The ratio of the mean magnitude of the jitter to the RMS jitter (30.7 ns / 39.7 ns ≈ 0.77) is close to the value of 0.80 expected for Gaussian noise, which indicates that the jitter is dominated by random noise rather than by occasional outliers. With regard to the specifications, the 1.5 µs allowance for the 5G system, including the UEs, in G.8271.1 is the most relevant budget. The maximum relative time error of 136 ns corresponds to about one tenth of this allowance. This comparison is indicative only, as a relative error is not directly comparable to an absolute limit. Note that this scenario only characterizes the relative error. Since both UEs share the common-mode error, no statement about the absolute accuracy with respect to UTC can be derived from it.

1.2.2 Test Case 2: End-to-End Accuracy of gPTP over a 5G Network

The second test case (Figure 2) evaluates the end-to-end accuracy of a gPTP time distribution carried through the 5G system. A gPTP grandmaster is connected to the 5G core via the NW-TT. The gPTP messages are transported through the 5G core and RAN and delivered over the air interface to the UEs. On the device side, the DS-TT's serial interface receives the UE's time via its 1PPS output, which is aligned to the 5G system time through SIB9, and provides the gPTP interface to the gPTP client. In this configuration, the 5G system acts as a time-aware bridge between the grandmaster and the client.

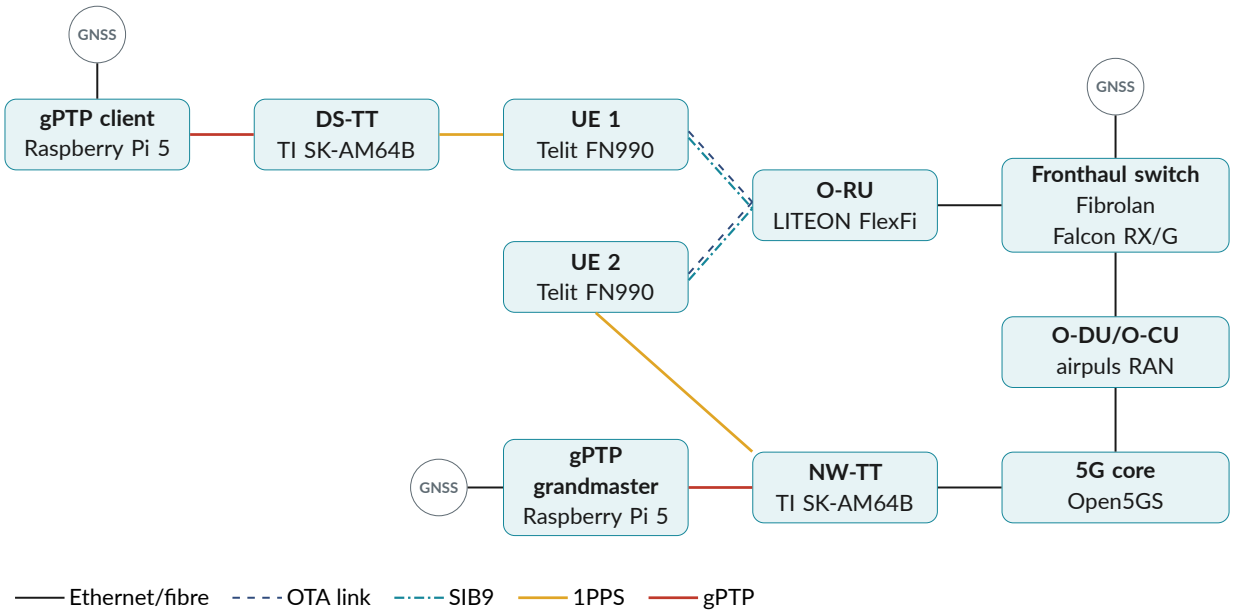

Figure 2: Testbed Setup – End-to-End Accuracy of gPTP over a 5G Network

The synchronization accuracy is determined by comparing the gPTP client's clock with the gPTP grandmaster's clock. The time error used here is the offset from the master as reported by the gPTP client itself; it is therefore not an independent measurement and does not capture path asymmetry or timestamping errors of the client. This measurement covers the entire chain: grandmaster, NW-TT, 5G core, RAN, SIB9 alignment of the UE, DS-TT, and client. It therefore represents the accuracy that an application behind the UE experiences, including the contribution of the gPTP servo on the client side.

Results

For test case two, the time error between the gPTP client and the gPTP Grandmaster, as reported by the client, was measured. The measurement was conducted over a 15-minute period, with eight samples per second. Table 2 summarizes the results.

Table 2: gPTP Time Error (Offset Reported by the gPTP Client) – not including spike intervals

Metric	Value [ns]
Mean |TE|	71.87
Maximum TE	+770
Minimum TE	-885
Standard deviation of TE (RMS jitter)	115.08
Mean |jitter|	86.13

During the measurement, after the first 22 seconds, a large spike in time error, reaching up to 39,000 ns (39 µs), was observed and lasted about 10 seconds before the time error began to decrease again. For the following 8 minutes, the time error was low and stable, with no outliers. After those eight minutes, another 15-second-long spike in the time error, reaching up to 220,000 ns (220 µs), was observed before normalizing again. Following this spike, no more time error spikes

were observed for the remainder of the measurement. Excluding those short spike intervals, the mean magnitude of the TE was 71.87 ns, with extremes ranging from -885 ns to +770 ns. An RMS jitter of 115.08 ns was calculated, while the mean magnitude of the jitter was 86.13 ns.

For time-aware bridge integration, IEEE 802.1AS [3] specifies a maximum error of 1 µs between end stations over up to seven hops, and 3GPP TS 22.104 [2] specifies a synchronicity budget of ≤ 900 ns for the 5G system. Outside the spike intervals, the TE remains within this limit even at the extremes (77% and 89% of 1 µs, or 85% and 98% of 900 ns), although with a narrow margin. The RMS jitter is about three times that of test case 1. This is plausible: in test case 1, error contributions common to both UEs cancel out in the relative measurement, whereas in test case 2, the jitter also includes contributions from the grandmaster, the NW-TT, the DS-TT, and the gPTP servo on the client side. The maximum |TE| of 885 ns therefore indicates that in steady state, with the spike intervals excluded, the system meets the accuracy requirements for typical industrial use cases, while the narrow margin at the extremes leaves little headroom.

Analysis

In the second test case, the low steady-state error results from how time is delivered in this architecture. The DS-TT does not derive its time from the timing of packets crossing the 5G user plane, but from the UE, which is aligned to the 5G system time via SIB9. The packet delay variation of the user plane, therefore, has little influence on the steady-state accuracy, which is mainly determined by the UE synchronization quality and by the timestamping and servo implementation of the gPTP devices. The RMS jitter of 115.08 ns is about three times that of test case 1. If the UE contribution is assumed to be about 28 ns, the remaining ~110 ns would originate from the grandmaster, the NW-TT, the DS-TT, and the gPTP servo. This estimate assumes uncorrelated contributions. The error distribution is also less Gaussian than in test case 1. With about 7,200 samples (15 minutes at 8 samples per second), the extremes of a Gaussian process would be expected at about 3.6σ. The observed extremes of +770 ns and −885 ns correspond to about 6.7σ and 7.7σ, indicating occasional larger deviations, for example, from servo corrections or from recovery after the spike intervals. The available data does not allow these causes to be distinguished. With the extremes at 77% and 89% of the 1 µs limit (85% and 98% of the 900 ns budget), the headroom is small, and little of the error budget remains for additional hops in a larger gPTP domain.

The large time-error spikes, of up to 39 µs after about 22 seconds and up to 220 µs after about eight minutes, are the most noticeable findings of this measurement. They exceed the 1 µs bound by a factor of about 40 and 220, respectively. Together, they cover about 25 s of the 15-minute measurement (about 3%).

For the sensor use case, a 220 µs error would exceed the sampling period of a 10 kHz measurement (100 µs). If such errors originated from the 5G system, individual samples would therefore be invalidated. However, the root cause of the observed spikes has not been conclusively determined. A reference measurement of the source PPS signal, taken without passing through the synchronization setup (cf. Test Case 1), showed no such spikes. This rules out the reference PPS and the reference side of the measurement chain as the cause, and it localizes the spikes to the synchronization path. That path includes both the 5G system and the UE-side PPS output and measurement interface. Our working hypothesis is that the spikes are an artifact of the measurement setup rather than a property of the 5G system. Electrical problems with the PPS signal on

the PPS interface occurred during setup and pretesting, leading to misdetected pulses. This hypothesis has not been verified for the reported measurements, so a contribution of the 5G system cannot be ruled out and must still be verified. As the DS-TT derives its timing from the UE's 1PPS output, missed pulses interrupt the discipline of the PTP hardware clock (PHC), causing it to drift freely until valid pulses are detected again. The magnitude of the spikes is consistent with this explanation: 39 µs within about 10 s corresponds to a frequency offset of roughly 4 ppm and 220 µs within about 15 s to roughly 15 ppm, which is the typical order of magnitude for a free-running crystal oscillator. In addition, the relative time error between the two UEs in the first test case remained stable, with no comparable spikes, indicating that SIB9 synchronization itself was not the source. A contribution of the initial lock-in of the gPTP servo to the first spike, which occurred only 22 s after the start, cannot be excluded either. It is important to note that the statistics in Table 2 exclude the spike intervals. The reported values, therefore, describe the stable operating state, but not the worst-case behavior of the setup.

1.3 Technical Challenges and Limitations

The main technical challenge was the occurrence of large time error spikes in the second test case. In the first test case, the relative time error between the two UEs remained stable, with no spikes, indicating that the UEs' SIB9 synchronization was not the source of the error. Therefore, the PPS connection between the UE and the DS-TT is the most likely cause. This could not be verified because no log data on missed PPS events were available for evaluation.

The duration of the disturbances (about 10 s and 15 s) would be significant for time-sensitive applications if they were caused by the 5G system itself. The statistics in Table 2 exclude these intervals. They therefore describe only the steady state, not the worst-case behavior of the setup.

Several limitations should be considered when interpreting the results. The first test case measures only the relative error between two UEs. Error contributions that are common to both devices, such as the GNSS reference, the O-DU/O-RU synchronization, and the fronthaul, cancel out, so no statement on the absolute accuracy with respect to UTC is possible. Measurements of absolute accuracy were not possible because the UEs sent their 1PPS signal based on the system frame number rather than the start of each new second. Therefore, comparisons to another 1PPS source, such as a Grandmaster, are not possible, as those send their 1PPS signal at the start of each new second. Additionally, at the time of testing, only one UE model (Telit FN990) supported SIB9 reception, so the behavior of other UEs could not be verified.

The two test cases also use different measurement methods (time analyzer on the 1PPS outputs vs. offset to master from the gPTP measurement) and different sampling rates (1 sample/s vs. 8 samples/s), so their values are only comparable to a limited degree. In particular, the stability observed in the first test case does not rule out problems on the PPS path in the second test case. A repeated measurement with a verified PPS connection is needed to confirm that the spikes are a setup artifact. Furthermore, test case 2 measures the entire chain, so the individual contributions of the grandmaster, NW-TT, DS-TT, and gPTP servo cannot be separated from the results.

The measurements lasted 45 minutes (test case 1) and 15 minutes (test case 2). Long-term wander, temperature effects, oscillator aging, and rare events are therefore not covered. The spikes in

test case 2 show that disturbances can occur, and with only 15 minutes of data, their frequency cannot be estimated. Finally, each scenario was measured only once, so the repeatability and reproducibility of the results were not assessed.

1.4 Summary

This test evaluated the effectiveness of SIB9-based time synchronization in a private 5G campus network for time-sensitive applications. For this purpose, the relative time error between two UEs was measured, as well as the end-to-end accuracy of a gPTP time distribution carried through the 5G system.

The relative time error between two UEs synchronized via SIB9 was small. The mean magnitude of the relative TE was 48.5 ns, the extremes were +136 ns and −129 ns, and the RMS jitter was 39.7 ns. The jitter behaved like bounded random noise, without steps or drift, so the effect on use cases such as synchronized data acquisition is negligible.

With gPTP carried over the 5G system, the steady-state accuracy between the grandmaster and client was also good. The mean magnitude of the TE was 71.87 ns, the RMS jitter 115.08 ns, and the extremes were +770 ns and −885 ns, i.e., close to, but within, the 1 µs limit. However, two time error spike intervals of up to 39 µs and 220 µs occurred during the measurement. They are most likely caused by electrical problems with the PPS connection between the UE and the DS-TT, which led to a brief drift in the PHC, rather than by the 5G system itself. This could not be verified and should be confirmed with a repeated measurement. In the steady state, with the spike intervals excluded, the 5G system meets the requirements of typical industrial use cases.

1.5 Outlook

The results show that SIB9-based time synchronization provides high relative synchronization accuracy between UEs, and that the 5G system can deliver gPTP time with sub-microsecond accuracy in the steady state. Several follow-up steps are of interest to confirm these results and to extend them to further use cases.

First, the second test case should be repeated using a verified, electrically robust PPS connection between the UE and the DS-TT. In addition, the PHC servo and the DS-TT should be logged during the measurement, so that missed or misdetected PPS events can be correlated with the time error. This would confirm whether the observed spikes were a setup artifact, as assumed, or had another cause. Longer measurement durations of several hours up to days, as well as multiple repetitions, would also show whether further spikes occur and how the system behaves with regard to long-term wander and temperature effects.

Second, the accuracy should be characterized in absolute terms. In this test, only the relative error between two UEs and the offset between the gPTP client and the grandmaster were measured. By directly comparing the UE's 1PPS output with a GNSS-referenced UTC source, the absolute accuracy of the SIB9 synchronization could be determined. In the second test case, an indepen-

dent measurement of the client clock, for example, with a time analyzer on the PPS outputs of the grandmaster and the client, would remove the dependency on the offset reported by the gPTP implementation itself. Measuring at the individual interfaces (NW-TT, UE, DS-TT) would also allow the error contributions of the single components to be separated.

Lastly, the test setup should be extended to higher-scale deployment scenarios. This could include more than two UEs, several gPTP clients, UEs of different models and vendors, once available, and several cells. For mobile robots and AGVs, mobility and handover between cells are particularly relevant, as SIB9 reception is unavailable during handover. The influence of user-plane load and radio conditions, such as low SNR or interference, on synchronization quality should also be evaluated.

2 Security Assurance for 5G Campus Components

Private 5G networks are increasingly built from open interfaces, open-source software, and components of multiple vendors. This increases flexibility but also increases the amount of software and the number of third-party libraries integrated into a deployment. Every one of these elements can contain security weaknesses that provide an entry point for an attack or allow unintended access to the system. Security measures that focus on communication between network elements are therefore insufficient. The software running within each component of a private 5G network also needs to be covered and tested. Weaknesses discovered only during operation are considerably more costly to fix than those found before deployment.

Analyzing a component's source code makes it possible to identify such weaknesses at an early stage, regardless of a specific installation or configuration. For this purpose, several complementary methods can be combined. Static analysis searches the code for known weakness patterns without executing it. Dependency analysis identifies the integrated third-party components and checks them for known vulnerabilities (CVEs). Autonomous penetration testing actively searches for further weaknesses that are difficult to detect statically. The findings are then reviewed by a security expert to filter out false positives and to prioritize the results.

In this test, the effectiveness of such a combined approach for the source code of private 5G network components is being evaluated. The source code of four components provided by Plugfest participants is analyzed using the Universität Passau/CipherCell Security Test Platform. The primary focus is on identifying potential security weaknesses in the code and its dependencies, and on how well the three analysis modules complement each other.

Additionally, the results are intended to provide participating vendors with a concrete starting point for improving their components. As the purpose of this scenario is to evaluate the platform and not to compare or rate individual vendors, the results are presented anonymously.

2.1 Test Tool

The Security Test Platform consists of a set of individual tools arranged in a pipeline to detect, filter, and rank vulnerabilities. The different stages of the pipeline are shown in Figure 3.

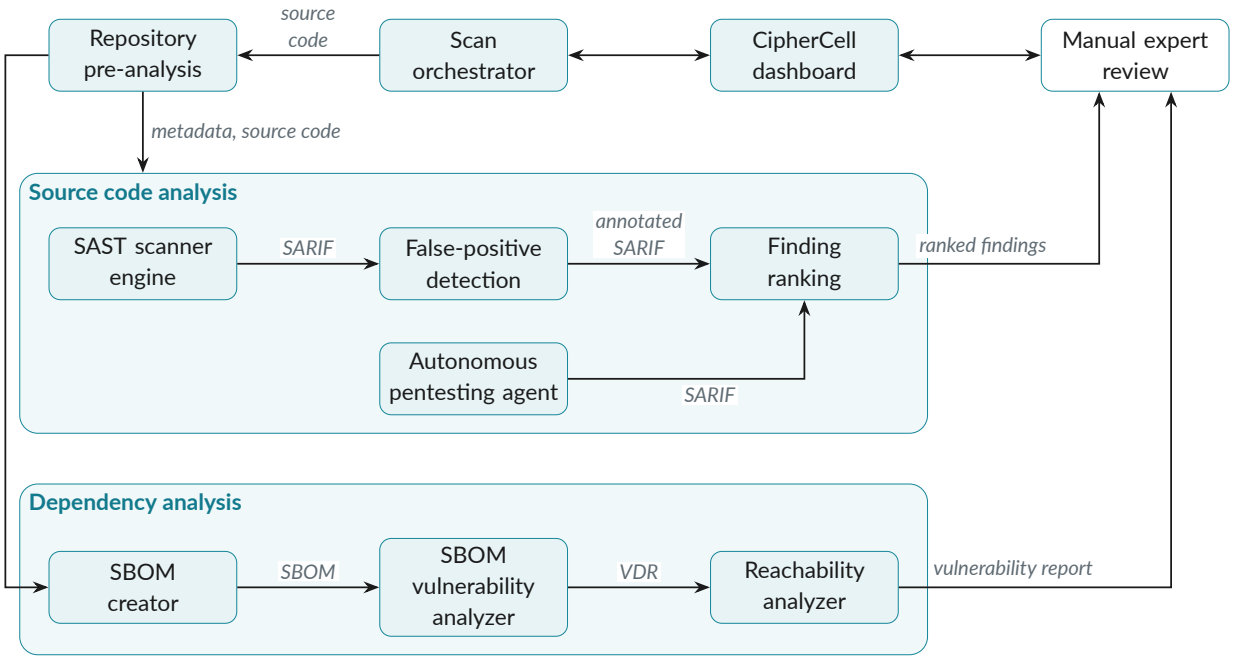

Figure 3: Pipeline Stages of the Security Test Platform

Via a dashboard or a CLI, users can select a repository, which is then automatically checked out. A dedicated Repository Pre-Analysis step detects the programming languages used and analyzes the repository structure. The resulting information is passed to two parallel pipelines: Source Code Analysis and Dependency Analysis. Based on the pre-analysis results, each pipeline selects and executes the appropriate tools for the repository.

Within the xG-ALOE Plugfest, open-source tools were automatically selected and orchestrated. The analyzed projects were large and heterogeneous. The pre-analysis components successfully identified components written in C/C++, Python, HTML, Go, TypeScript, and YAML, as well as Dockerfiles, CMake files, and Makefiles. Overall, nearly 2 million lines of code were analyzed; Figure 4 shows their distribution.

2.1.1 Source Code Analysis

Based on these results, the appropriate open-source scanners are automatically selected for the Static Application Security Testing (SAST) Scanning pipeline. The scanning tools are orchestrated by the SAST Scanner Engine, which executes the selected scanners to detect security vulnerabilities within the projects. The open-source scanners that were executed are listed in Table 3.

The results from all scanners are normalized and deduplicated into a standardized Static Analysis Results Interchange Format (SARIF) [10]. The normalized SARIF results are then analyzed by the False Positive Detection stage, which identifies and flags potential false positives. Locally

Table 3: SAST Tools

Tool	Category	Primary Scope	What It Detects
Gitleaks	Secret scanning	All text files and Git history	Hard-coded secrets such as passwords, API keys, tokens, and private keys using patterns and entropy analysis.
Bandit	SAST / security linting	Python	Security issues identified through Python AST analysis, including dangerous subprocess usage, `eval`/`exec`, insecure deserialization, weak cryptography, and hard-coded credentials.
Trivy	Vulnerability, Software Composition Analysis (SCA), misconfiguration & secret scanning	Dependencies, containers, filesystems, Kubernetes/IaC, and repositories	Known vulnerabilities (CVEs), vulnerable dependencies, insecure configurations, and exposed secrets.
Pylint	Linting / static analysis	Python	Programming errors, code smells, bad practices, and maintainability issues. Some findings may have security implications, but it is not primarily a security scanner.
Semgrep	SAST / rule-based static analysis	30+ languages	Vulnerability patterns such as injection, insecure API usage, weak cryptography, and missing validation.
Gosec	SAST / security linting	Go	Security issues including injection, hard-coded credentials, weak cryptography, insecure TLS configuration, and unsafe file/path handling.
Fallow Security	Security-oriented static analysis	JavaScript / TypeScript	Identifies potentially security-sensitive code locations/candidates for subsequent verification.
Cppcheck	Static analysis/defect detection	C / C++	Undefined behavior, memory/resource errors, out-of-bounds access, null dereferences, leaks, uninitialized variables, and similar defects.

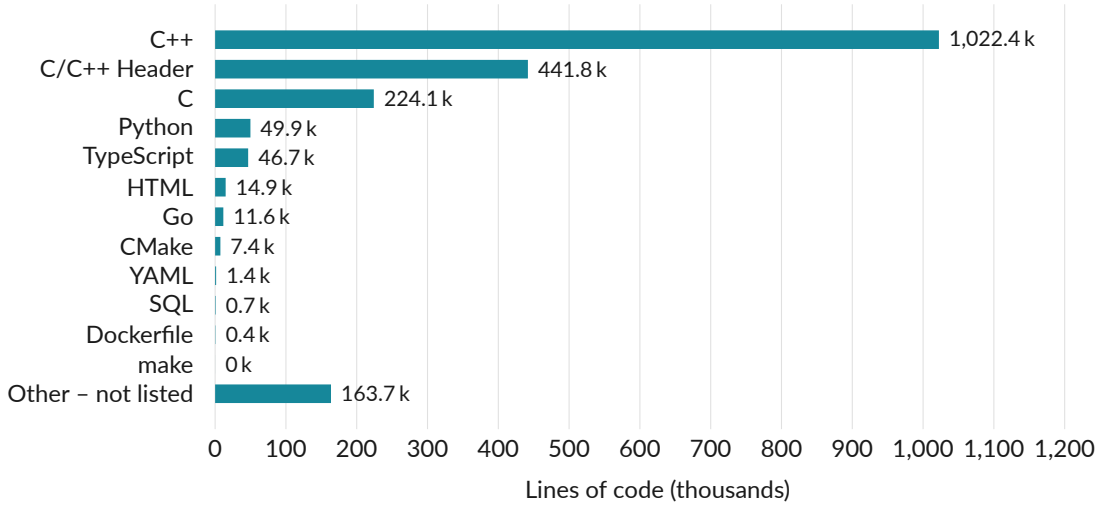

Figure 4: Distribution of analyzed code

hosted LLMs are used to analyze the context of each finding. This analysis combines the static context surrounding a finding with dedicated escalation agents that perform additional analysis for findings that are difficult to classify automatically. Finally, the SAST findings are ranked by severity and context, presented in the dashboard, and also provided via a report-generation tool that includes the scan results and metadata.

2.1.2 Dependency Analysis

The second branch of the pipeline focuses on analyzing vulnerabilities in a project's dependencies. The first step is to identify the project's dependencies and summarize them in a Software Bill of Materials (SBOM). Generating an accurate SBOM requires dedicated dependency-scanning tools.

The SBOM Creator combines multiple open-source SBOM generation tools with custom logic. The results from the individual scanners are aggregated into a single SBOM, with duplicate components removed and the results normalized. The resulting SBOM is generated in the CycloneDX format [11] and further enriched with custom algorithms to improve dependency information. Table 4 lists the scanners integrated into the SBOM Creator.

The generated SBOM is subsequently analyzed against multiple vulnerability databases, including GitHub Advisory Database [12], Google OSV (Open Source Vulnerabilities) [13], and the National Vulnerability Database (NVD) [14], to identify known vulnerabilities affecting the project's dependencies. However, the presence of a registered vulnerability does not necessarily mean that it is exploitable within the project. To assess the actual exploitability of identified vulnerabilities, the Reachability Analyzer determines whether the vulnerable code is reachable from the application. It analyzes whether the vulnerable functionality can be invoked through the API endpoints and execution paths exposed by the project/source code under test. The Reachability Analyzer combines deterministic reachability analysis with AI-driven filtering to reduce false positives and prioritize potentially reachable vulnerabilities, thereby making them more relevant to the application.

Finally, all results from the Source Code Analysis and Dependency Analysis branches are combined

into a single overall security assessment. The vulnerability metrics are normalized to provide a consistent view across both analysis paths. The impact of each vulnerability is further detailed in terms of availability, confidentiality, and integrity, along with additional metrics such as time to exploit, attack vectors, and other relevant characteristics.

Table 4: Scanners integrated into SBOM Creator

Tool	Type	Analysis Targets	Description
cdxgen	SBOM generator (OWASP CycloneDX)	Source code repositories and build files of most ecosystems (e.g. npm, PyPI, Maven/Gradle, Go, Cargo, Conan), container images	Generates a CycloneDX SBOM directly from the source code. Resolves direct and transitive dependencies from manifests and lockfiles, including the dependency tree and evidence where each component was found.
Syft	SBOM generator (Anchore)	Container images, file systems, archives, and binaries	Catalogs the software components actually present in a build artifact, including OS packages (dpkg, rpm, apk), installed language packages, and binaries. Outputs SBOMs in CycloneDX or SPDX format.

2.2 Security Risks Covered

Source code can contain many different types of security weaknesses. These can range from simple implementation errors to more complex issues related to authentication, authorization, input handling, cryptography, sensitive data, dependencies, or unintended functionality.

It is not practical for a single automated analysis platform to cover every possible security risk. The scope of this test is therefore defined by the security weaknesses that the Security Test Platform can detect. The types of vulnerabilities that can be detected in source code analysis depend heavily on the scanner used, as shown in Table 3.

Table 5 provides an overview of the security risk/CWE (Common Weakness Enumeration) [6] categories considered for this test, based on the MITRE CWE-1400 view [9].

2.3 Test Execution Procedure

Step 1: Project Setup

Every scan runs within a project in the Universität Passau/CipherCell UI. A project groups one or more repositories for scanning and reporting. Source code can be connected in one of two ways:

- **Managed ZIP upload:** The source code is provided as a ZIP archive. The testing team creates the project, uploads the archive to the Universität Passau/CipherCell UI, and starts the scan.

Table 5: Security Risks Covered by the Test Tool

Security Risk Category (CWE Category)	Coverage Details
Access Control (CWE-1396)	Incorrect file permissions, missing or insufficient access controls, hard-coded credentials, and selected authentication or authorization weaknesses.
Comparison (CWE-1397)	Incorrect comparisons, overly permissive comparisons, and selected weaknesses involving allowlists or validation logic.
Component Interaction (CWE-1398)	HTTP request smuggling and interpretation conflicts between interacting components.
Memory Safety (CWE-1399)	Buffer overflows, out-of-bounds access, invalid pointer use, and other memory-safety issues in C/C++ code.
Concurrency (CWE-1401)	Race conditions, time-of-check/time-of-use (TOCTOU) issues and other concurrency-related weaknesses.
Encryption (CWE-1402)	Weak hash and encryption algorithms (e.g., MD5, SHA-1, DES, RC4), cleartext transmission, disabled TLS verification, and improper signature verification.
Exposed Resource (CWE-1403)	Insecure temporary files, externally controlled file names, and sensitive data stored in caches.
File Handling (CWE-1404)	Path traversal and insecure file path handling.
Improper Check or Handling of Exceptional Conditions (CWE-1405)	Unchecked return values, missing exception handling, and overly broad exception handling.
Improper Input Validation (CWE-1406)	Basic data-flow analysis of untrusted input and detection of insufficient input validation.
Improper Neutralization (CWE-1407)	Missing output encoding, improper escaping, and insecure encoding/decoding of untrusted data.
Incorrect Calculation (CWE-1408)	Integer overflows and underflows, arithmetic errors, and incorrect calculations.
Injection (CWE-1409)	SQL injection, OS command injection, code injection, template injection, header injection, and cross-site scripting (XSS).
Insufficient Control Flow Management (CWE-1410)	Infinite loops, uncontrolled recursion, and other potentially unsafe control-flow conditions.
Insufficient Verification of Data Authenticity (CWE-1411)	CSRF, downloads without integrity checks, and insufficient verification of data origin or authenticity.
Poor Coding Practices (CWE-1412)	NULL pointer dereferences, use of dangerous functions, active debug code, and other insecure coding practices.
Protection Mechanism Failure (CWE-1413)	Selected patterns indicating missing, incorrectly implemented, or bypassed security protection mechanisms.
Randomness (CWE-1414)	Weak or predictable random number generation and insecure use of randomness.
Resource Control (CWE-1415)	Insecure deserialization (e.g., `pickle`, unsafe YAML loading) and prototype pollution.
Resource Lifecycle Management (CWE-1416)	Uncontrolled resource consumption, resource leaks, and improper management of resources during their lifecycle.
Sensitive Information Exposure (CWE-1417)	Exposed secrets, and sensitive data disclosed through log files, error messages, or other application output.
Violation of Secure Design Principles (CWE-1418)	Execution with unnecessary privileges and selected weaknesses resulting from insecure design decisions.

- **Self-service GitLab connection:** The participant creates a project, connects their GitLab account, imports the repositories to be tested, and starts the scan by clicking Start Scan.

Step 2: Automated Pipeline

Starting a scan triggers a pipeline run that executes several analysis paths in parallel. The following modules are currently integrated:

- **Source code analysis:** Multiple static analysis tools scan the code, and their results are merged. Each finding is then checked in context to filter out likely false positives, and the remaining findings are grouped and ranked by priority.
- **Dependency analysis:** Third-party components are identified and checked for known vulnerabilities. Each vulnerable component is then assessed to determine whether the project actually uses the affected code.
- **Autonomous penetration testing:** An autonomous pentesting agent independently searches for additional vulnerabilities. Its findings are merged into the results of the other modules.

Step 3: Expert Review

A security expert reviews the prioritized findings, confirming or dismissing each one based on their experience, and manually examines the code for further security issues. Based on the confirmed results, a detailed PDF report is generated, along with a summary providing a quick overview of all analyzed areas.

Step 4: Results Review

The participant receives four deliverables:

- **SARIF file:** Lists the vulnerabilities found in the source code itself.
- **VDR (Vulnerability Disclosure Report):** Lists the vulnerabilities found in third-party dependencies.
- **PDF report:** Documents the confirmed findings in detail.
- **Summary:** Gives a quick overview of the results across all analyzed areas.

2.4 Participants

The component security test involved Universität Passau/CipherCell as the provider of the Security Test Platform and four participating vendors/organizations whose source code was assessed: airpuls, brown-iposs, Fraunhofer HHI, and Open Radio Systems.

As the purpose of this scenario is to evaluate the security use case and the capabilities of the Security Test Platform, rather than to compare or evaluate individual vendors, the detailed findings are not directly linked to the participating companies. The codes differ in purpose, goal, size/length,

and the number of third-party/open-source components used, further invalidating any comparisons between them. The results have therefore been anonymized, and the tested components are therefore referred to as CUT 1, CUT 2, CUT 3, and CUT 4 in the following sections.

2.5 Results

This section shows the scan results for each code under test (CUT), split into the three analysis modules. As the test results may contain sensitive information about security weaknesses in individual components, the detailed findings are presented anonymously. Each tested component is therefore assigned an identifier from CUT 1 to CUT 4, and the mapping between the participating vendors and these identifiers is not disclosed in this report.

The static analysis table shows the number of true-positive findings in the source code, meaning findings confirmed during review, and the CWE identifiers of the weakness types they represent. The dependency vulnerabilities table shows how many third-party components, i.e., external libraries and frameworks used in the project, were analyzed and how many known vulnerabilities (CVEs) were found in them. It also breaks down those vulnerabilities by severity rating (critical, high, medium, and low) and lists the highest CVSS score among them. The severity rating is derived from the CVSS base score [7], which ranges from 0.0 to 10.0 and is calculated from the exploitability of a vulnerability (attack vector, attack complexity, required privileges, and user interaction) and its impact on confidentiality, integrity, and availability. Scores of 9.0 to 10.0 are rated critical, 7.0 to 8.9 high, 4.0 to 6.9 medium, and 0.1 to 3.9 low. This calculation is based on the CVSS v3.1 specifications by the Forum of Incident Response and Security Teams (FIRST). The autonomous penetration testing table shows the number of findings reported by the pentesting agent, the corresponding CWE identifiers, and the highest severity score, based on the CVSS score.

Together, these give an overview of how many security issues were found and how severe they are, covering the code itself, its dependencies, and active testing. Further details on individual CWEs can be found by searching for the corresponding CWE ID in the MITRE CWE Catalog.

2.5.1 CUT 1

The source code of CUT 1 comprised 444,090 lines of code (LoC) across 3,123 files, corresponding to about 24% of the nearly 2 million lines analyzed in total.

Static Code Analysis

The static code analysis of CUT 1 confirmed 54 true-positive findings. These findings correspond to 11 different weakness types (Table 6).

Regarding the size of the codebase, 54 confirmed findings across 444,090 LoC correspond to about 0.12 findings per 1,000 LoC, or roughly one finding per 8,200 lines. This value includes only findings confirmed during review and weakness types that the integrated scanners can detect. It is therefore not comparable to general defect densities from other sources and should be

Table 6: Static Code Analysis - Findings CUT 1

CWE-250	Execution with Unnecessary Privileges
CWE-1357	Reliance on Insufficiently Trustworthy Component
CWE-1395	Dependency on Vulnerable Third-Party Component
CWE-22	Path Traversal
CWE-918	Server-Side Request Forgery
CWE-276	Incorrect Default Permissions
CWE-476	NULL Pointer Dereference
CWE-457	Use of Uninitialized Variable
CWE-242	Use of Inherently Dangerous Function
CWE-415	Double Free
CWE-676	Use of Potentially Dangerous Function

understood as a reference value for this test.

Five of the eleven weakness types (CWE-476, CWE-457, CWE-242, CWE-415, and CWE-676) concern memory handling and the use of unsafe functions. This is plausible given that the analyzed code is dominated by C and C++. Of these, the double free (CWE-415) is the most security-relevant, since it can lead to memory corruption. The remaining findings concern privileges and permissions (CWE-250, CWE-276), the handling of external input in file paths and requests (CWE-22, CWE-918), and the reliance on third-party components (CWE-1357, CWE-1395). The latter two are closely related to the dependency analysis below. As 54 findings are distributed across only 11 weakness types, individual types recur, suggesting recurring coding patterns rather than isolated mistakes.

Dependency Vulnerabilities

In the dependency analysis of CUT 1, 672 third-party components were identified. These components contain 104 known vulnerabilities (CVEs), of which 1 was rated critical, 51 high, 46 medium, and 6 low. The highest CVSS score among them is 9.2, with an average score of 7.4 across all findings.

Exactly half of the vulnerabilities (52 of 104) are rated critical (1) or high (51), and only 6 are rated low. With 104 vulnerabilities across 672 components, there are, on average, about 0.15 vulnerabilities per component. Because a CVE registration does not necessarily mean the vulnerable code is reachable from the component, these numbers indicate potential exposure rather than confirmed exploitability.

Autonomous Penetration Testing

The autonomous penetration testing agent reported 32 findings for CUT 1, with a highest severity score of 9.4 and an average of 8.7. The findings correspond to 27 different weakness types (Table 7).

The agent reported a considerably broader range of weakness types than the static analysis (27

Table 7: Autonomous Penetration Testing - Findings CUT 1

CWE-770	Allocation of Resources Without Limits or Throttling
CWE-78	OS Command Injection
CWE-400	Uncontrolled Resource Consumption
CWE-195	Signed to Unsigned Conversion Error
CWE-20	Improper Input Validation
CWE-680	Integer Overflow to Buffer Overflow
CWE-693	Protection Mechanism Failure
CWE-347	Improper Verification of Cryptographic Signature
CWE-918	Server-Side Request Forgery
CWE-367	Time-of-Check Time-of-Use Race Condition
CWE-799	Improper Control of Interaction Frequency
CWE-209	Generation of Error Message Containing Sensitive Information
CWE-22	Path Traversal
CWE-674	Uncontrolled Recursion
CWE-79	Cross-Site Scripting
CWE-613	Insufficient Session Expiration
CWE-200	Exposure of Sensitive Information to an Unauthorized Actor
CWE-524	Use of Cache Containing Sensitive Information
CWE-478	Missing Default Case in Multiple Condition Expression
CWE-328	Use of Weak Hash
CWE-488	Exposure of Data Element to Wrong Session
CWE-755	Improper Handling of Exceptional Conditions
CWE-636	Not Failing Securely
CWE-287	Improper Authentication
CWE-639	Authorization Bypass Through User-Controlled Key
CWE-789	Memory Allocation with Excessive Size Value
CWE-125	Out-of-bounds Read

vs. 11), with nearly every finding representing a different weakness type. Several groups can be identified. A first group concerns resource exhaustion and denial-of-service (CWE-770, CWE-400, CWE-789, CWE-674, CWE-799), in which uncontrolled allocation, recursion, or request frequency can overload the component. A second group covers input handling and injection (CWE-78, CWE-79, CWE-20) as well as arithmetic and memory errors that can be triggered through input (CWE-195, CWE-680, CWE-125). A third group concerns authentication, authorization, and session handling (CWE-287, CWE-639, CWE-613, CWE-636). A fourth group covers the exposure of sensitive information through error messages, caches, and sessions (CWE-209, CWE-524, CWE-488, CWE-200). Further findings concern cryptography (CWE-347, CWE-328), a race condition (CWE-367), path traversal and SSRF (CWE-22, CWE-918), a protection mechanism failure (CWE-693), and the handling of exceptional conditions (CWE-478, CWE-755). The highest severity score, 9.4, indicates that at least one finding was rated critical.

CWE Summary

The test identified 38 CWEs (36 distinct CWEs). Mapped to CWE-1400, they span 16 categories, so the weaknesses are spread across the codebase rather than isolated. Access Control (CWE-1396) dominates with six weakness types, covering authentication, authorization bypass, session expiration, default permissions, and SSRF. Memory Safety (CWE-1399), Resource Lifecycle Management (CWE-1416), and Poor Coding Practices (CWE-1412) follow with four weakness types each. Together with Insufficient Control Flow Management (CWE-1410), they indicate low-level robustness problems and a notable denial-of-service exposure.

Design-level gaps appear in Violation of Secure Design Principles (CWE-1418) and Protection Mechanism Failure (CWE-1413). Data handling is affected through Injection (CWE-1409), Improper Input Validation (CWE-1406), and File Handling (CWE-1404). Confidentiality is affected through Sensitive Information Exposure (CWE-1417), Exposed Resource (CWE-1403), and Encryption (CWE-1402). Single weakness types fall under Concurrency (CWE-1401), Exceptional Conditions (CWE-1405), and Comparison (CWE-1397).

Overall, the results point to systemic weaknesses in access control and memory/resource management, as well as injection, input handling, and design flaws.

Overall Assessment of CUT 1

The three modules show different but complementary aspects of CUT 1. The static analysis mainly found weaknesses in the code itself, such as unsafe memory handling and excessive privileges. The penetration testing agent additionally found weaknesses that only appear at runtime or depend on the interplay of components, such as session handling, authentication, race conditions, and resource exhaustion. Two types of weaknesses, CWE-22 (Path Traversal) and CWE-918 (Server-Side Request Forgery), were found by both modules independently, supporting the consistency of the results. The dependency analysis adds the external perspective of known vulnerabilities in third-party components.

2.5.2 CUT 2

The source code of CUT 2 comprised 13,672 LoC in 96 files, which corresponds to less than 1% of the roughly 2 million lines analyzed and is the smallest of the four CUTs.

Static Code Analysis

The static code analysis of CUT 2 confirmed 26 true-positive findings. These findings correspond to 4 different weakness types (Table 8).

Table 8: Static Code Analysis - Findings CUT 2

CWE-250	Execution with Unnecessary Privileges
CWE-703	Improper Check or Handling of Exceptional Conditions
CWE-611	Improper Restriction of XML External Entity Reference
CWE-259	Use of Hard-coded Password

Regarding the size of the codebase, 26 confirmed findings across 13,672 LoC correspond to about 1.9 findings per 1,000 LoC, or roughly one finding per 530 lines. This is the highest density of the four components. As for CUT 1, the value only includes findings confirmed during review and only weakness types that the integrated scanners can detect, and should be understood as a reference value for this test.

The four weakness types cover privileges (CWE-250), credentials (CWE-259), the processing of XML input (CWE-611), and error handling (CWE-703). With 26 findings distributed across only 4 types, each type occurs, on average, about 6.5 times, suggesting recurring patterns rather than isolated mistakes.

Dependency Vulnerabilities

In the dependency analysis of CUT 2, 26 third-party components were identified. These components contain 27 known vulnerabilities (CVEs), of which 1 was rated critical, 19 high, 7 medium, and none low. The highest CVSS score among them is 9.8, with an average score of 7.7 across all findings.

CUT 2 has the fewest third-party components of the four, yet with 27 vulnerabilities in 26 components, there is, on average, about one vulnerability per component, the highest ratio of all components. Most of them are severe: 20 of 27 (about 74%) are rated critical (1) or high (19). As a registered CVE does not necessarily mean that the vulnerable code is reachable from the component, these numbers again show potential exposure rather than confirmed exploitability.

Autonomous Penetration Testing

The autonomous penetration testing agent reported 17 findings for CUT 2, with a highest severity score of 9.4 and an average of 8.7. The findings correspond to 3 different weakness types (Table 9).

The findings are concentrated in very few types of weaknesses, with an average of almost six findings per type, suggesting that the same weaknesses were repeatedly found in the component.

Table 9: Autonomous Penetration Testing - Findings CUT 2

CWE-798	Use of Hard-coded Credentials
CWE-94	Improper Control of Generation of Code (‘Code Injection’)
CWE-611	Improper Restriction of XML External Entity Reference

Hard-coded credentials (CWE-798) correspond to the static analysis finding CWE-259, which describes a more specific variant of the same problem. The XML processing weakness (CWE-611) was found by both modules independently. Code injection (CWE-94) was only found by the penetration testing agent. It is generally regarded as particularly severe, since it can allow an attacker to execute their own code.

CWE Summary

The test identified 7 CWEs (6 distinct CWEs). Mapped to CWE-1400, they fall into four categories. Access Control (CWE-1396) dominates with four weakness types: hard-coded passwords and credentials, and XML external entity (XXE) processing. Each remaining weakness type falls into its own category: Injection (CWE-1409) through code injection, Violation of Secure Design Principles (CWE-1418) through execution with unnecessary privileges, and Improper Check or Handling of Exceptional Conditions (CWE-1405).

Overall, the results point mainly to weak credential management and insufficient restriction of external XML resources. Code injection combined with excessive privileges considerably raises the potential impact of a successful attack.

Overall Assessment of CUT 2

CUT 2 is a small component with a comparatively high density of confirmed findings. The three modules point to related topics. Hard-coded credentials were found by the static analysis (CWE-259) and the penetration testing agent (CWE-798), and XML external entity processing (CWE-611) was found by both modules independently, supporting the consistency of the results. The penetration testing agent additionally found code injection (CWE-94), and the dependency analysis contributes a high number of severe vulnerabilities, including the highest CVSS score of the test. As the findings focus on a few types of weaknesses and the codebase is small, targeted measures, such as secure handling of credentials, secure configuration of XML processing, and updating the affected dependencies, could address a large part of the findings.

2.5.3 CUT 3

The source code of CUT 3 comprised 68,834 LoC across 670 files, corresponding to about 4% of the roughly 2 million lines analyzed across the four CUTs.

Static Code Analysis

The static code analysis of CUT 3 confirmed 34 true-positive findings. These findings correspond to 8 different weakness types (Table 10).

Table 10: Static Code Analysis - Findings CUT 3

CWE-829	Inclusion of Functionality from Untrusted Control Sphere
CWE-250	Execution with Unnecessary Privileges
CWE-494	Download of Code Without Integrity Check
CWE-770	Allocation of Resources Without Limits or Throttling
CWE-497	Exposure of Sensitive System Information to an Unauthorized Control Sphere
CWE-668	Exposure of Resource to Wrong Sphere
CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
CWE-532	Insertion of Sensitive Information into Log File

Regarding the size of the codebase, 34 confirmed findings across 68,834 LoC correspond to about 0.49 findings per 1,000 LoC, or roughly one finding per 2,000 lines. As before, this value only includes confirmed findings and weakness types that the integrated scanners can detect.

Three groups can be identified. The first concerns the trust in external code and its integrity (CWE-829, CWE-494), for example, when functionality is included or downloaded without sufficient verification. The second concerns the exposure of sensitive information (CWE-497, CWE-668, CWE-532), including sensitive data in log files. The remaining findings concern privileges (CWE-250), uncontrolled resource allocation (CWE-770), and open redirects (CWE-601). The latter is typical for components with a web interface. On average, each weakness type occurs about four times, which again suggests recurring patterns.

Dependency Vulnerabilities

In the dependency analysis of CUT 3, 1,508 third-party components were identified. These components contain 7 known vulnerabilities (CVEs), none of which were rated critical; 3 were rated high, 3 medium, and 1 low. The highest CVSS score among them is 8.7, with an average score of 7.2 across all findings.

CUT 3 has by far the largest number of third-party components, but only 7 known vulnerabilities, which corresponds to about one vulnerability per 215 components. With about 22 components per 1,000 LoC, the component relies heavily on third-party packages, which is typical for package-based ecosystems and probably includes transitive dependencies. The low number of vulnerabilities suggests that the dependencies are comparatively up to date. As a registered CVE does not necessarily mean that the vulnerable code is reachable from the component, these numbers, as before, show potential exposure rather than confirmed exploitability.

Autonomous Penetration Testing

The autonomous penetration testing agent reported 14 findings for CUT 3, with a highest severity score of 8.8 and an average of 7.5. The findings correspond to 11 different weakness types (Table 11).

Nearly every finding represents a different type of weakness. A first group concerns web and session security (CWE-863, CWE-352, CWE-79, CWE-614), which aligns with the static analysis

Table 11: Autonomous Penetration Testing - Findings CUT 3

CWE-863	Incorrect Authorization
CWE-352	Cross-Site Request Forgery (CSRF)
CWE-835	Loop with Unreachable Exit Condition ('Infinite Loop')
CWE-347	Improper Verification of Cryptographic Signature
CWE-798	Use of Hard-coded Credentials
CWE-79	Cross-Site Scripting
CWE-614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute
CWE-367	Time-of-Check Time-of-Use Race Condition
CWE-787	Out-of-bounds Write
CWE-829	Inclusion of Functionality from Untrusted Control Sphere
CWE-250	Execution with Unnecessary Privileges

findings for a component with a web interface. A second group concerns trust and integrity (CWE-347, CWE-829, CWE-798), a third group concerns robustness and memory handling (CWE-835, CWE-367, CWE-787), and one finding concerns privileges (CWE-250). The out-of-bounds write (CWE-787) is a memory-safety weakness typically associated with native code. Two weakness types, CWE-829 and CWE-250, were found by both the static analysis and the penetration testing agent.

CWE Summary

The test identified 19 CWEs (17 distinct CWEs). Mapped to CWE-1400, they fall into 11 categories. Access Control (CWE-1396) and Resource Lifecycle Management (CWE-1416) lead with three weakness types each. The first covers open redirect, incorrect authorization, and hard-coded credentials; the second covers inclusion of untrusted functionality and unlimited resource allocation. Two weakness types each fall under Sensitive Information Exposure (CWE-1417), Encryption (CWE-1402), Insufficient Verification of Data Authenticity (CWE-1411), and Violation of Secure Design Principles (CWE-1418). Together they cover exposed system information and log data, weak signature verification and insecure cookie transmission, unverified code downloads and CSRF, and execution with unnecessary privileges. Single weakness types fall under Exposed Resource (CWE-1403), Insufficient Control Flow Management (CWE-1410), Injection (CWE-1409), Concurrency (CWE-1401) and Memory Safety (CWE-1399).

Overall, one theme recurs across several categories: external code and data are not verified well enough, as seen in untrusted inclusions, unverified downloads and weak signature checks. Combined with the access control gaps and excessive privileges, this substantially raises the risk of code execution and privilege escalation.

Overall Assessment of CUT 3

In CUT 3, the weaknesses are mainly in the code itself, not in its dependencies. The static analysis and the penetration testing agent complement each other. They both found the inclusion of untrusted functionality (CWE-829) and excessive privileges (CWE-250), while the penetration

testing agent additionally found weaknesses that only appear at runtime, such as authorization errors, CSRF, and insecure cookie handling. The dependency analysis shows a low exposure despite the high number of components. With the highest severity score of 8.8, the penetration testing agent reported the lowest maximum among the four components, but this value still falls within the high range.

2.5.4 CUT 4

The source code of CUT 4 comprised 1,311,073 LoC across 5,400 files, corresponding to about 71% of the roughly 2 million lines analyzed across the four components, with an average of about 243 lines per file. It is by far the largest of the four components.

Static Code Analysis

The static code analysis of CUT 4 confirmed 26 true-positive findings. These findings correspond to 9 different weakness types (Table 12).

Table 12: Static Code Analysis - Findings CUT 4

CWE-825	Expired Pointer Dereference
CWE-798	Use of Hard-coded Credentials
CWE-676	Use of Potentially Dangerous Function
CWE-476	NULL Pointer Dereference
CWE-682	Incorrect Calculation
CWE-269	Improper Privilege Management
CWE-319	Cleartext Transmission of Sensitive Information
CWE-457	Use of Uninitialized Variable
CWE-190	Integer Overflow or Wraparound

Regarding the size of the codebase, 26 confirmed findings across 1,311,073 LoC correspond to about 0.02 findings per 1,000 LoC, or roughly one finding per 50,000 lines. This is the lowest density of the four components. As before, the value only includes confirmed findings and weakness types that the integrated scanners can detect. It also depends on how much of the code is first-party code and how deeply the findings could be reviewed.

Five of the nine weakness types (CWE-825, CWE-676, CWE-476, CWE-457, and CWE-190) concern memory handling, the use of unsafe functions, and integer arithmetic. Together with CWE-682, which also concerns calculation errors, they make up two-thirds of the weakness types. This is plausible given that the analyzed code is dominated by C and C++. The remaining findings concern credentials and privileges (CWE-798, CWE-269) and the transmission of sensitive data in clear text (CWE-319).

Dependency Vulnerabilities

In the dependency analysis of CUT 4, 35 third-party components were identified. No known vulnerabilities (CVEs) were found in these components. Consequently, no CVSS score is available.

CUT 4 is the only component without known vulnerabilities in its dependencies. However, the number of identified components (35) is small relative to the codebase's 1.3 million LoC, indicating that the CUT has few external dependencies. It may also mean that dependencies are included directly in the source tree or are not declared in build files, in which case a dependency analysis based on manifests and lockfiles can only partially detect them. The result therefore applies to the detected components and to the vulnerabilities known in the databases at the time of the test. It does not prove that the component contains no vulnerable third-party code, nor does it make any statement about weaknesses in the component's own code.

Autonomous Penetration Testing

The autonomous penetration testing agent reported 26 findings for CUT 4, with a highest severity score of 9.8 and an average of 7.0. The findings correspond to 20 different weakness types (Table 13).

Table 13: Autonomous Penetration Testing - Findings CUT 4

CWE-306	Missing Authentication for Critical Function
CWE-190	Integer Overflow or Wraparound
CWE-125	Out-of-bounds Read
CWE-820	Missing Synchronization
CWE-121	Stack-based Buffer Overflow
CWE-776	Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')
CWE-346	Origin Validation Error
CWE-862	Missing Authorization
CWE-191	Integer Underflow (Wrap or Wraparound)
CWE-789	Memory Allocation with Excessive Size Value
CWE-73	External Control of File Name or Path
CWE-779	Logging of Excessive Data
CWE-129	Improper Validation of Array Index
CWE-367	Time-of-Check Time-of-Use Race Condition
CWE-489	Active Debug Code
CWE-362	Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')
CWE-287	Improper Authentication
CWE-200	Exposure of Sensitive Information to an Unauthorized Actor
CWE-770	Allocation of Resources Without Limits or Throttling
CWE-457	Use of Uninitialized Variable

The agent reported a broad range of weakness types, with on average only about 1.3 findings per type. Several groups can be identified. The largest group concerns memory safety and arithmetic errors (CWE-125, CWE-121, CWE-129, CWE-190, CWE-191, CWE-457), which is consistent with the dominance of C and C++. A second group concerns authentication, authorization, and origin validation (CWE-306, CWE-862, CWE-287, CWE-346). A third group concerns concurrency

(CWE-367, CWE-362, CWE-820), and a fourth group concerns uncontrolled resource consumption (CWE-770, CWE-789, CWE-776). The remaining findings concern information exposure and debug functionality (CWE-200, CWE-779, CWE-489) and file path handling (CWE-73). With a severity score of 9.8, CUT 4 has the highest severity score among the four components. Two weaknesses, CWE-190 and CWE-457, were found by both static analysis and the penetration testing agent.

CWE Summary

The test identified 29 CWEs (27 distinct CWEs). Mapped to CWE-1400, they fall into 10 categories. Three categories lead with five weakness types each. Access Control (CWE-1396) covers missing or improper authentication and authorization, improper privilege management, and hard-coded credentials. Memory Safety (CWE-1399) covers out-of-bounds read, stack-based buffer overflow, improper array index validation, expired pointer dereference, and excessive memory allocation. Resource Lifecycle Management (CWE-1416) covers uninitialized variables, XML entity expansion, unlimited resource allocation, and excessive logging.

Incorrect Calculation (CWE-1408) adds four weakness types on integer overflow and underflow. Poor Coding Practices (CWE-1412) contributes three, covering NULL pointer dereference, dangerous functions, and active debug code. Concurrency (CWE-1401) also contributes three, all race conditions. Single weakness types fall under Encryption (CWE-1402: cleartext transmission), Insufficient Verification of Data Authenticity (CWE-1411: origin validation), Exposed Resource (CWE-1403: external control of file paths), and Sensitive Information Exposure (CWE-1417).

Overall, the profile is dominated by low-level implementation flaws typical of native code. These memory, arithmetic, and concurrency errors can lead to crashes, denial of service, or code execution. Fundamental gaps in authentication and authorization make them more serious.

Overall Assessment of CUT 4

CUT 4 shows a clear difference between the modules. The static analysis density is the lowest of the four components, and the dependency analysis found no known vulnerabilities, but the penetration testing agent reported 26 findings with the highest severity score of the test. A low density of static findings and a clean dependency analysis therefore do not imply a low risk in the running system. The two modules overlap in integer arithmetic (CWE-190) and uninitialized variables (CWE-457), while most other weaknesses were only found by one of them. Authentication, authorization, concurrency, and resource consumption issues were found only by the penetration testing agent, underscoring the value of a dynamic approach, since these weaknesses depend on runtime behavior and are difficult to detect statically.

2.5.5 Overview of Results

Table 14 summarizes the results for all CUTs.

Table 14: Results Overview for all CUTs

	CUT 1	CUT 2	CUT 3	CUT 4
LoC/files	444,090 / 3,123	13,672 / 96	68,834 / 670	1,311,073 / 5,400
Static: true-positive findings	54	26	34	26
Static: findings per 1,000 LoC	0.12	1.90	0.49	0.02
Third-party components	672	26	1,508	35
Known vulnerabilities (critical/high)	104 (1 / 51)	27 (1 / 19)	7 (0 / 3)	0
Highest CVSS score	9.2	9.8	8.7	–
Average CVSS score	7.4	7.7	7.2	-
Pentesting: findings/weakness types	32 / 27	17 / 3	14 / 11	26 / 20
Highest pentesting severity	9.4	9.4	8.8	9.8
Average pentesting severity	8.7	8.7	7.5	7.0

2.6 Analysis

The results show that the Security Test Platform was able to process all four components and to deliver findings in all three analysis modules. Across the four CUTs, 140 true-positive findings were confirmed in the source code, 138 known vulnerabilities were identified in 2,241 third-party components, and the autonomous penetration testing agent reported 89 findings. No component was free of findings. Even CUT 4, for which no vulnerable dependencies were found, had 26 confirmed static analysis findings and 26 pentesting findings. These numbers should not be read as a ranking of the components. The size and nature of the codebases differ, and the findings depend on the types of weaknesses the integrated scanners can detect.

In the static analysis, the number of true-positive findings ranges from 26 (CUT 2 and CUT 4) to 54 (CUT 1). The identified weaknesses cover a broad range of CWE types. Recurring themes are privilege and access control issues, with CWE-250 (execution with unnecessary privileges) found in three of the four components and CWE-269 (improper privilege management) in a fourth. Hard-coded credentials (CWE-798 and CWE-259) were found in CUT 2, CUT 3, and CUT 4. Memory and resource handling weaknesses, such as NULL pointer dereferences (CWE-476), use of uninitialized variables (CWE-457), and use of potentially dangerous functions (CWE-676), appear in CUT 1 and CUT 4, which is plausible given that the analyzed code is dominated by C and C++ (about 85% of the lines of code). CUT 2 and CUT 3 show more weaknesses typical for service and web interfaces, such as XML external entity processing (CWE-611), open redirects (CWE-601), and sensitive information in log files (CWE-532).

In the dependency analysis, the picture differs across components. CUT 1 had 104 known vulnerabilities in 672 components, of which 1 was critical and 51 were rated high (highest CVSS 9.2). CUT 2 had 27 vulnerabilities across only 26 components, of which 20 were rated critical or high, including the test's highest CVSS score (9.8). In contrast, CUT 3 had only 7 vulnerabilities across

1,508 components (highest CVSS: 8.7), and CUT 4 had none across 35 components. The number of components therefore says little about the number of vulnerabilities. What matters more is which components are used and how up to date they are. Overall, 75 of the 138 vulnerabilities (about 54%) were rated critical (2) or high (73). A registered CVE does not mean the vulnerable code can be exploited; these numbers indicate potential exposure, not confirmed exploitability.

The autonomous penetration testing agent reported between 14 (CUT 3) and 32 (CUT 1) findings per component, with the highest severity scores ranging from 8.8 to 9.8. In three of the four components, the highest severity was 9.4 or above. The agent also reported a wide range of types of weakness. CUT 1 and CUT 4 showed the highest diversity (27 and 20 different CWEs). The findings of CUT 2, in contrast, were concentrated on three CWEs (CWE-798, CWE-94 and CWE-611), which points to a few specific problems such as hard-coded credentials, code injection, and XML processing.

The modules also complement each other. Several weakness types were found by both the static analysis and the pentesting agent in the same component, for example, path traversal (CWE-22) and server-side request forgery (CWE-918) in CUT 1, XML external entities (CWE-611) in CUT 2, untrusted inclusion of functionality (CWE-829) in CUT 3, and integer overflows (CWE-190) in CUT 4. This overlap indicates that the findings are consistent between the independent methods. At the same time, each module also reported weaknesses that the other did not, for example, race conditions (CWE-367) and missing authentication or authorization (CWE-287, CWE-862, CWE-306) in the pentesting results. This shows the value of combining the three analysis paths. Finally, the pipeline handled heterogeneous projects with about 2 million lines of code in about ten programming languages plus build and configuration files. This demonstrates that automatic tool selection works for complex 5G software stacks.

2.7 Technical Challenges and Limitations

As with any automated security analysis, the Security Test Platform has inherent limitations that should be considered when interpreting the results. They do not invalidate the findings, which were confirmed by expert review, but define what the results can and cannot say.

The range of weaknesses that can be detected is determined by the custom rule sets and the integrated open-source scanners with their rule sets. The depth of analysis varies between programming languages, and some tools focus more on general code quality than on security. Static analysis examines code without executing it and therefore lacks knowledge of a component's runtime behavior, configuration, or deployment environment. Weaknesses that only appear during execution or depend on the overall system design are harder to identify with static methods alone. This is why the autonomous penetration testing agent was included as a dynamic approach. It complements the static analysis and covers weakness types that static methods detect only partially.

The autonomous penetration testing agent is based on LLMs, which come with their own characteristics. Its behavior is not deterministic, so repeated runs on the same component can yield different results. The coverage of the attack surface depends on the strategy the agent chooses and on the available time and computing resources. LLMs are also typically trained on more data about common web and IT technologies than about specialized 5G protocols and interfaces, so

the depth of testing for telecom-specific functionality may be limited. In addition, the agent may misjudge individual weaknesses or their severity. The reported findings and their severity scores should therefore be validated manually, which was the purpose of the expert review step.

Combining the results of multiple scanners requires normalizing their different rule sets, severity scales, and weakness classifications. A single root cause can lead to several findings at different code locations, so the number of reported findings depends on how they are grouped and should not be read as a measure of the number of independent problems. The LLM-based false-positive detection considerably reduces manual effort, although misclassifications cannot be fully ruled out. Some findings can only be conclusively assessed with input from the vendors, since a pattern that appears insecure in isolation may be a deliberate design decision.

Finally, there is no ground truth for the analyzed code, so the number of undetected weaknesses is unknown. The absence of a particular weakness type in the results therefore does not prove that it is absent from the code. The results are best understood as a prioritized list of confirmed and potential weaknesses that serves as a starting point for remediation, and not as a complete security certification of the components.

2.8 Summary

This test evaluated the capabilities of Universität Passau/CipherCell's Security Test Platform for source-code-level security assessment across four components (CUT 1 to CUT 4) of private 5G network solutions. In total, nearly 2 million lines of code, mainly C/C++, were analyzed by the three modules of the platform: static source code analysis, dependency analysis and autonomous penetration testing.

The static analysis confirmed 140 true-positive findings (ranging from 26 to 54 per component). Typical weaknesses included excessive privileges, hard-coded credentials, and memory-handling errors in the C/C++ code. The dependency analysis identified 138 known vulnerabilities in 2,241 components, of which 75 were rated critical or high. These were concentrated in two components. The autonomous penetration testing agent reported 89 findings, with the highest severity scores ranging from 8.8 to 9.8. No component was free of findings. The detailed results are anonymized, and the test is not intended to compare vendors.

Regarding the platform itself, it processed all components, combined the results of the three modules, and found weaknesses across many CWE categories. Several weaknesses were detected independently by more than one module, supporting the consistency of the results, while other findings were detected only by one module, indicating that the combination of methods is useful. The reported numbers indicate potential security weaknesses and do not permit a statement about the overall security level of a component. The results should serve as a starting point for remediation by the vendors.

3 Hardware Acceleration Comparison for Private 5G

RAN processing is one of the most compute-intensive parts of a 5G system. In private and campus networks, this processing often runs on general-purpose servers, where CPU resources and power consumption are important design considerations. Hardware accelerators can offload selected RAN processing tasks from the CPU, improving performance or reducing the resources required by the host system.

However, different accelerator cards can vary in throughput, latency, and power consumption. This test therefore compares the Open Radio Systems (ORS) AURORA Radio Processing Unit (RPU), further referred to as AURORA, with the Intel ACC100 hardware accelerator and with a CPU-only baseline implemented with the AVX-512 instruction set, in terms of throughput, latency, and power efficiency.

The component under test is the LDPC decoder in the uplink. In the current Minimum Viable Product (MVP) implementation, only this step is offloaded by AURORA. Additional steps, such as encoding (downlink) and rate dematching (uplink), are supported by the ACC100. However, in OCUDU, a similar setup was maintained by disabling encoding on the ACC100, since encoding is not implemented in this version of AURORA. In all test cases, rate dematching in the ACC100 was activated.

All configurations were tested with a single COTS UE in a 2x2 MIMO setup, at high SNR (about 20 dB) and low SNR (about 5 dB), with one or ten worker threads assigned to the OCUDU RAN stack, and with downlink or uplink traffic. Each combination ran three times for ten minutes, while the Layer 2 throughput, the TWAMP round-trip time, and the power consumption of the whole server were measured. An additional test case with an emulated Viavi TM500 UE verified the throughput of the COTS UE.

3.1 Device under Test

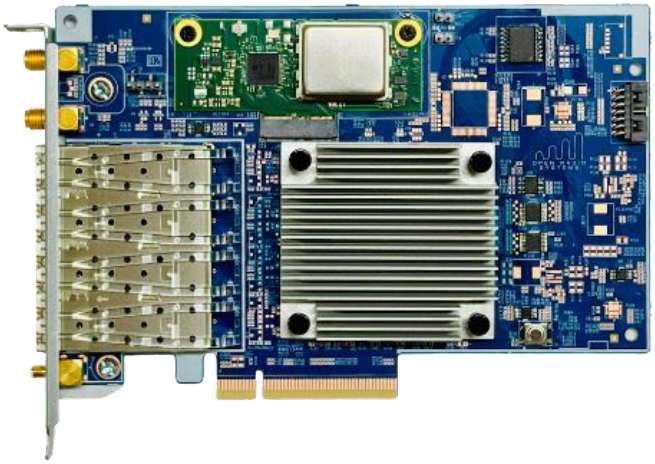

Figure 5: ORS AURORA RPU

According to Open Radio Systems [15], AURORA (Figure 5) will be a fully programmable radio-processing platform and fronthaul bus controller for 5G NR virtual RAN (vRAN) servers in Split 7.2 [16] configuration. It is able to connect to up to 4 O-RUs via the SFP interface.

During the Plugfest, it was used as a look-aside PCIe hardware accelerator card with decoding integrated into the OCUDU [17] RAN stack. It is connected to the host via PCIe Gen 3 x8, which provides a raw transfer rate of 64 GT/s. It is an FPGA-based accelerator; every new feature will be shipped as a configuration update.

3.2 Test Cases

A total of 25 test cases were executed, with each test case being run three times with a duration of ten minutes; 24 test cases constituted a combination of hardware accelerator used (none/ORS AURORA/Intel ACC100), UDP traffic direction (Downlink/Uplink), level of Signal-to-Noise Ratio (high ≈ 20 dB, low ≈ 5 dB), and number of worker threads assigned to OCUDU (1/10), with a single Quectel Commercial-Off-The-Shelf (COTS) UE. Assigned worker threads refer to the server threads/cores allocated to the main data processing (e.g., decoding, rate dematching, ...) of the gNB application, thereby limiting the resources available for these tasks. The additional 25th test case was performed with a single emulated TM500 UE to validate the provided throughput of the Quectel (COTS) UE. For each test case and run, the UE's usable Layer 2 throughput (goodput), the TWAMP Round-Trip Time (RTT), and the server's power consumption were measured.

3.3 Testbed Setup

The testbed setup for the first 24 test cases (Figure 6) consisted of the Server under Test, the EdgePro 2U, which hosted the bare-metal OCUDU RAN on isolated CPU cores, configured to

2x2 MIMO, and the containerized Open5GS [18] Core, with iPerf3 [19] and TWAMP [20] servers running inside the UPF. The server's Fronthaul interface was connected to the Benetel RAN550 O-RU through the Fibrolan Falcon RX switch, which also distributed PTP to the RAN server and the O-RU. Two antennas on the O-RU were then connected to the Quectel RM520N-GL User Equipment (UE) via conductive connections, giving a 2T2R, 2x2 MIMO setup. Between the O-RU and the UE, two different sets of attenuators were installed; each conducted antenna connection to the UE had a fixed 30 dB attenuator and an additional programmable, variable attenuator, which allowed testing of the different SNR scenarios (high/low) by increasing/decreasing the attenuation to simulate an AWGN channel, e.g., a UE close to the cell center or a UE at the cell edge.

The cell operated in band n78 (NR-ARFCN 650000, i.e., 3.75 GHz) with a channel bandwidth of 100 MHz and a subcarrier spacing of 30 kHz, using two antennas in downlink and uplink. The TDD pattern repeats every 10 slots (5 ms) with seven downlink slots, one special slot (six downlink and four uplink symbols), and two uplink slots. The 256QAM MCS table was used. The traffic was generated with iPerf3 as UDP with a packet size of 1398 bytes.

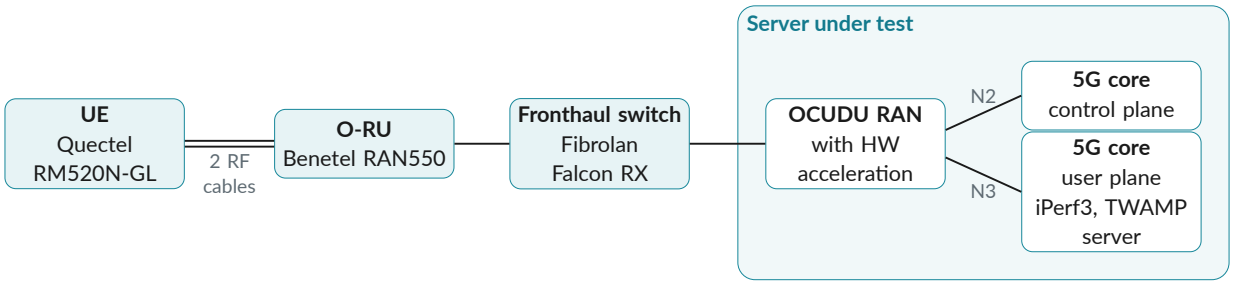

Figure 6: Testbed Setup – COTS UE

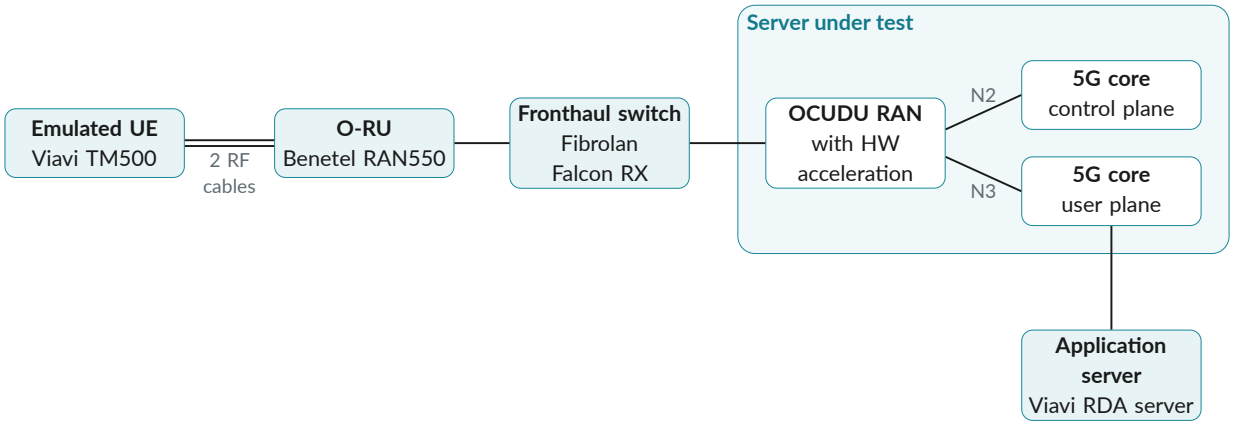

Figure 7: Testbed Setup – Emulated UE

Additional information regarding the different components and their software/firmware versions is listed in Tables 15 and 16.

For the 25th verification test case (Figure 7), the Quectel UE was replaced by an emulated UE from the Viavi TM500, and the dedicated RDA-server from Viavi replaced the iPerf and TWAMP servers running inside the UPF. Other than that, the testbed setup remained the same.

Table 15: 5G System Components Details

Component	Model & Version
UE	Quectel RM520N-GL – RM520NGLAAR03A03M4G Viavi TM500 – NLA 8.15.2
O-RU	Benetel RAN550 – 1v1.5.1-NM-71e2f57
Fronthaul switch	Fibrolan Falcon-RX – 8.24.2.2
gNB	OCUDU – Custom patch based on 26.04
Core	Open5GS – Version from 2026-07-27 (Commit f87da61)

Table 16: Server Component Details

Component	Model/Version
Server CPU	Intel(R) Xeon(R) 6527P
Server RAM	4x Samsung M321R4GA3PB2-CCPKC (32GB at 6400 MT/s)
Server BIOS	Vendor: American Megatrends International, LLC. Version: AGVE_5.35_v1.50
Linux Distro	Ubuntu 24.04.4 LTS
Kernel	6.8.1-1015-realtime
Network Driver	DPDK 22.11.11 (Stable)

3.4 Results

During testing, the UE's usable L2 throughput (goodput) was measured, along with TWAMP End-to-End Round Trip Time and power consumption.

3.4.1 Downlink

Figure 8 shows the results of the downlink throughput tests. In the downlink, the results across the three configurations are close for most test cases. With ten worker threads, all solutions reach approximately 800 Mbps at high SNR and 730–746 Mbps at low SNR.

With a single worker thread, the AVX-512 software solution lags in both SNR scenarios on average. At high SNR with one worker thread, AVX-512 reached 613.3 Mbps compared to 729.9 Mbps (ACC100) and 728.9 Mbps (AURORA), a reduction of about 16%. At low SNR, AVX-512 achieved 590.1 Mbps compared to 680.5 Mbps (ACC100) and 687.9 Mbps (AURORA), a reduction of about 13–14%. ACC100 and AURORA performed nearly identically in these cases, with a difference of ≤ 2% on average.

The TWAMP RTT was also measured during these tests. Figure 9 shows the RTT results for the downlink throughput tests.

The average downlink RTT depends mainly on the number of worker threads and less on the accelerator used. With one worker thread, the average RTT ranges from 22.7 ms to 27.8 ms, and the maximum RTT ranges from 50.0 ms to 69.2 ms. With ten worker threads, the average

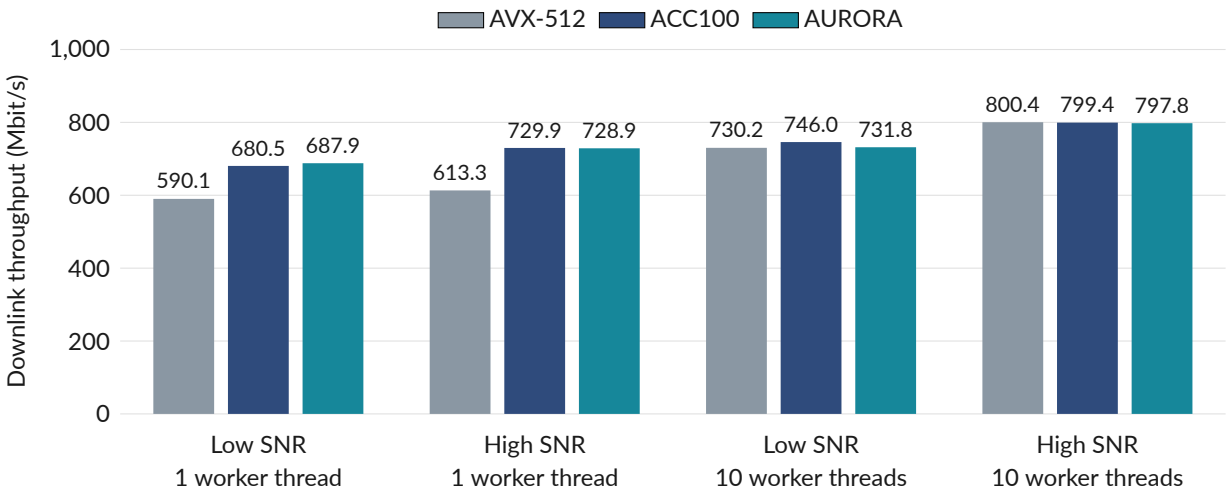

Figure 8: Downlink Throughput Results

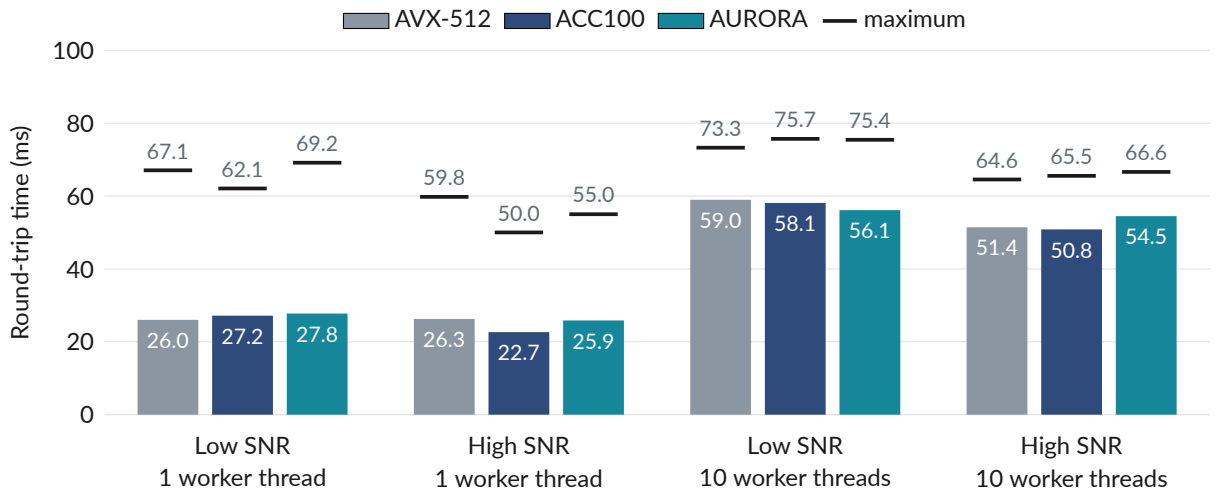

Figure 9: TWAMP Round-Trip Time during Downlink Traffic – Average and Maximum

RTT roughly doubles to 50.8–58.9 ms, while the maximum RTT is only slightly higher at 64.6–75.7 ms. This coincides with the higher throughput reached with ten threads. Between the three configurations, the differences are small, usually within a few milliseconds. ACC100 showed the lowest values at high SNR with one thread (22.7 ms average, 50.0 ms maximum), and AURORA had the lowest average at low SNR with ten threads (56.1 ms). No configuration shows a consistent advantage.

3.4.2 Uplink

The same tests were executed with uplink traffic as well, and Figure 10 shows the throughput results.

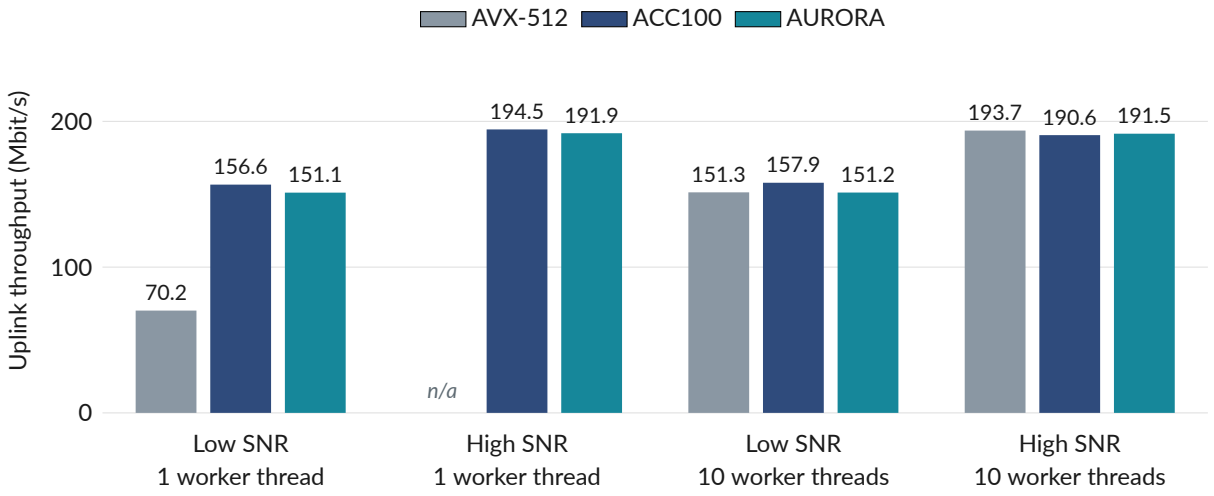

Figure 10: Uplink Throughput Results

In the uplink, the results at high SNR are nearly identical with 10 worker threads. With one worker thread, ACC100 and AURORA reached 194.5 Mbps and 191.9 Mbps, respectively. For the AVX-512 software solution, no results are available, as the UE would consistently and reproducibly lose the connection to the gNB because a single worker thread was not enough to handle decoding.

At low SNR, the throughput across all configurations is, as expected, lower, at about 151–158 Mbps with 10 threads. With one worker thread, the effect of the hardware acceleration is clearly visible. In this case AVX-512 was able to handle the decoding with only one worker thread without connection loss, because the UE chose a lower MCS due to the lower SNR, but therefore only reached 70.2 Mbps. ACC100 (156.6 Mbps) and AURORA (151.1 Mbps) remained nearly the same with one thread as with ten threads. This corresponds to a reduction of about 54–55% for the AVX-512 software solution. Once again, the ACC100 and the AURORA perform nearly identically, with the maximum difference being about 4%.

The latency tests with the TWAMP RTT were also repeated while only uplink traffic was running. Figure 11 shows these results.

With ten worker threads, the uplink RTT is similar for all configurations. At high SNR, the average RTT is 34.9–36.6 ms, and the maximum RTT is 59.0–64.7 ms. At low SNR, the average RTT is 40.8–43.3 ms, and the maximum RTT is 65.5–74.5 ms. With one worker thread, the differences are

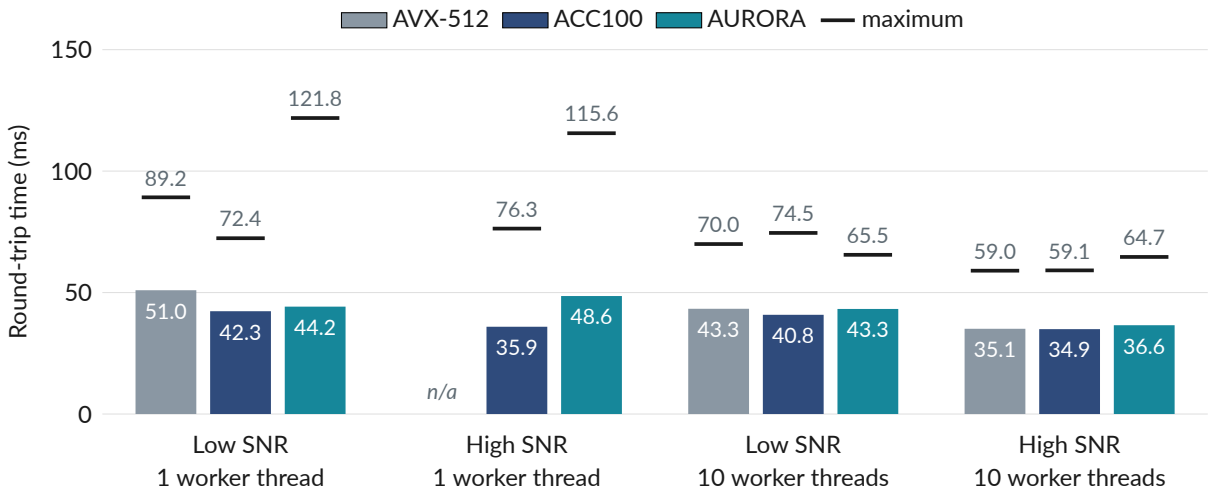

Figure 11: TWAMP Round-Trip Time during Uplink Traffic – Average and Maximum

larger. At high SNR, ACC100 reached 35.9 ms (average) and 76.3 ms (maximum), while AURORA reached 48.6 ms and 115.6 ms; no result is available for AVX-512, as the UE would lose connection to the gNB, as mentioned above.

At low SNR, ACC100 showed the best values (42.3 ms average, 72.4 ms maximum), followed by AURORA (44.2 ms average) and AVX-512 (51.0 ms average, 89.2 ms maximum). AURORA showed a noticeably higher maximum RTT (121.8 ms) in this case, indicating occasional latency spikes with a single worker thread.

3.4.3 Latency without Traffic

For comparison, TWAMP latency measurements were also performed in isolation, meaning no downlink or uplink traffic was present. The results are shown in Figure 12.

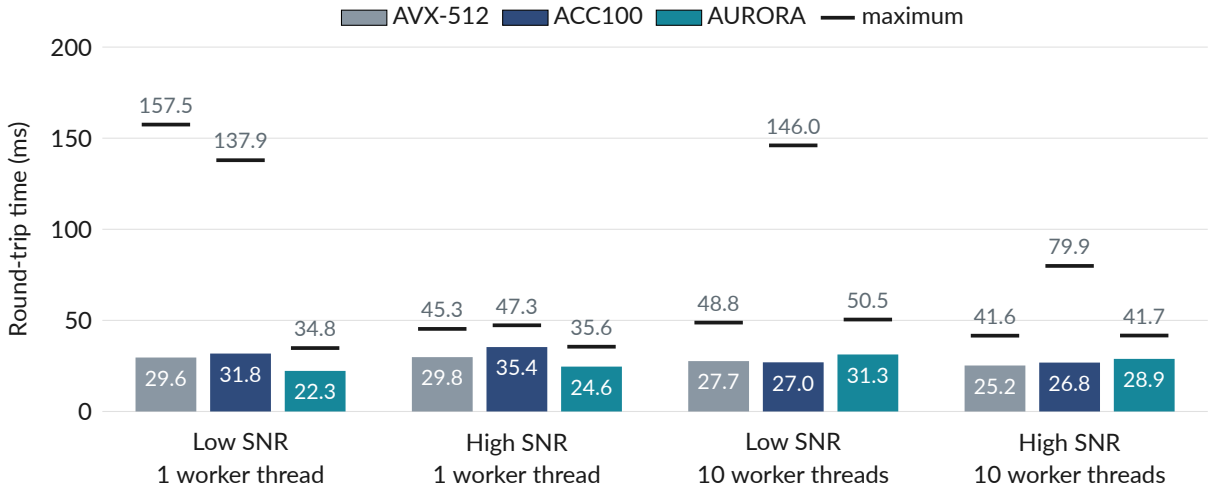

Figure 12: TWAMP Round-Trip Time without Traffic – Average and Maximum

In the TWAMP-only measurements, the average RTT across all configurations ranges from 22.3 ms to 35.4 ms. With one worker thread, AURORA showed the lowest average and maximum RTT in both SNR scenarios (24.6 ms/35.6 ms at high SNR and 22.3 ms/34.8 ms at low SNR). AVX-512

and ACC100 had average values of 29.6–35.4 ms. At low SNR, they also reached maximum values of 157.5 ms (AVX-512) and 137.9 ms (ACC100), both of which were due to single-latency peaks.

With ten worker threads, the average RTT is similar for all solutions (25.2–31.3 ms). Here, AVX-512 and AURORA stay at a maximum of 41.6–50.5 ms, while ACC100 shows peaks of 79.9 ms (high SNR) and 146.0 ms (low SNR).

3.4.4 Power Consumption and Efficiency

In addition to the throughput and latency, the average power consumption of the whole server (including the accelerator cards, if present) was measured. These measurements were collected via the server's Baseboard Management Controller (BMC) over three measurements, each ten minutes long, with a ten-second interval. Table 17 lists the average power consumption of the server in the different test combinations.

Table 17: Average Power Consumption Measurements for different Test Combinations

Configuration	AVX-512 (W)	ACC100 (W)	AURORA (W)
Downlink, High SNR, One Thread	191.91	223.73	207.73
Downlink, High SNR, Ten Threads	192.00	224.00	208.00
Downlink, Low SNR, One Thread	176.62	223.73	207.33
Downlink, Low SNR, Ten Threads	192.00	223.91	208.00
Uplink, High SNR, One Thread	N/A	208.00	207.73
Uplink, High SNR, Ten Threads	192.00	223.82	208.00
Uplink, Low SNR, One Thread	176.44	208.00	207.73
Uplink, Low SNR, Ten Threads	192.00	223.91	208.00

Based on these power readings, the power efficiency was computed as average Throughput per Watt (Mbps/W) for Downlink and Uplink. Figures 13 and 14 show those results.

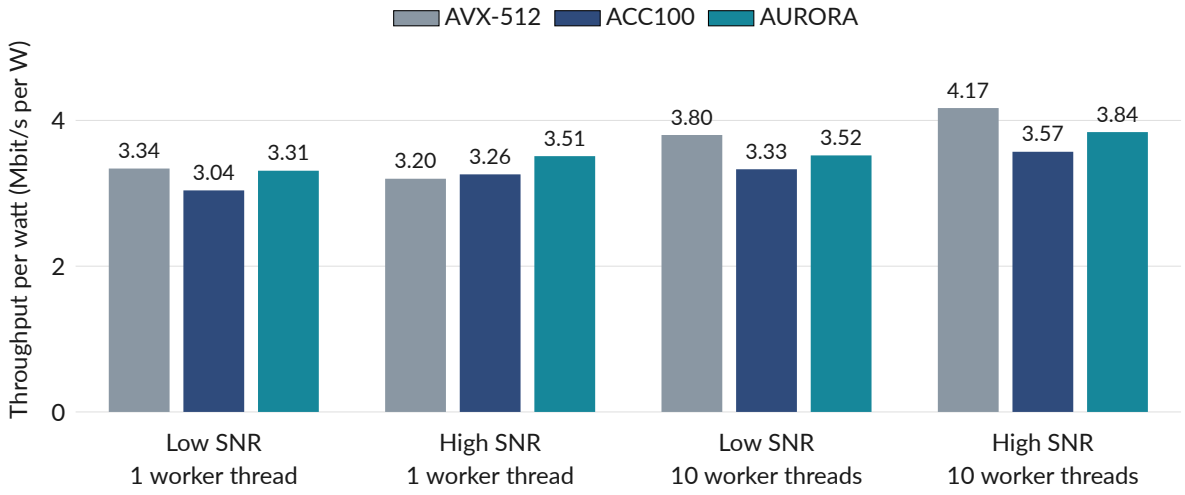

Figure 13: Power Efficiency – Throughput per Watt for Downlink Traffic

In the downlink, the power efficiency of all configurations ranges from 3.0 to 4.2 Mbps/W. With ten worker threads, the AVX-512 software solution is the most efficient configuration in both

SNR scenarios (4.17 Mbps/W at high SNR and 3.80 Mbps/W at low SNR). AURORA follows, at 3.84 Mbps/W and 3.52 Mbps/W, which are about 8% and 7% lower than AVX-512, respectively. ACC100 shows the lowest efficiency (3.57 Mbps/W and 3.33 Mbps/W), about 14% and 12% below AVX-512. Since the throughput of all configurations is nearly identical with ten threads, these differences are mainly caused by differences in power consumption. Since the accelerator cards were not used in the downlink tests, this most likely reflects the idle power consumption of the installed cards.

With one worker thread, the picture is different. At high SNR, AURORA is the most efficient solution (3.51 Mbps/W), followed by ACC100 (3.26 Mbps/W) and AVX-512 (3.20 Mbps/W), i.e., about +10% and +2% compared to AVX-512. Here, the software solution's lower throughput is offset by its lower power consumption. At low SNR, AVX-512 (3.34 Mbps/W) and AURORA (3.31 Mbps/W) are almost equal, while ACC100 is about 9% lower (3.04 Mbps/W).

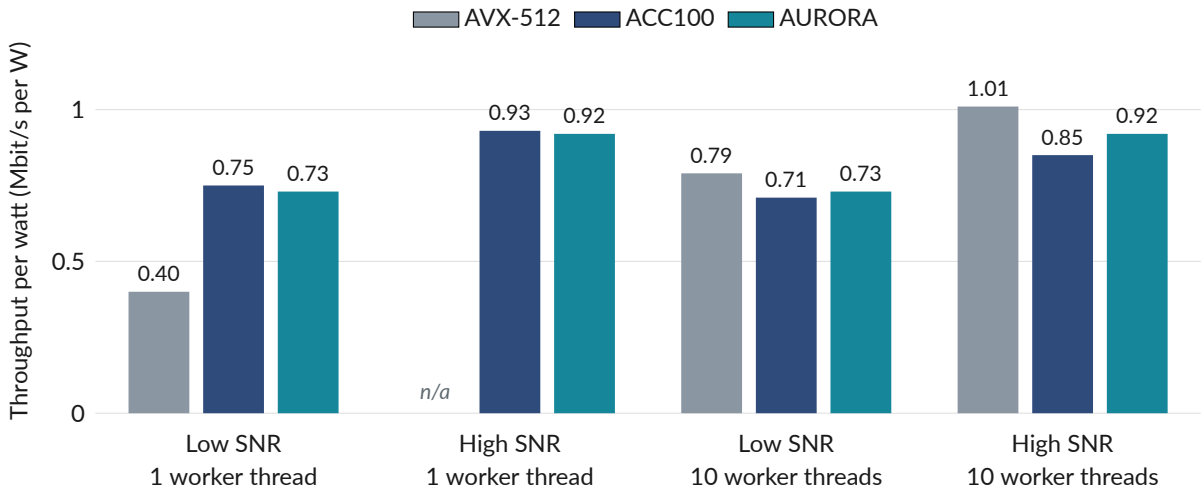

Figure 14: Power Efficiency – Throughput per Watt for Uplink Traffic

In the uplink, power efficiency is significantly lower than in the downlink, ranging from 0.40 to 1.01 Mbps/W, due to lower uplink throughput at comparable power consumption.

With ten worker threads, the ranking is the same in both SNR scenarios. AVX-512 is the most efficient solution (1.01 Mbps/W at high SNR and 0.79 Mbps/W at low SNR), followed by AURORA (0.92 Mbps/W and 0.73 Mbps/W) and ACC100 (0.85 Mbps/W and 0.71 Mbps/W). AURORA is about 9% (high SNR) and 8% (low SNR) below AVX-512, ACC100 is about 16% and 10%. Again, the throughput is nearly the same, so the differences result from the additional power demand of the accelerator cards.

With one worker thread, the AVX-512 solution is far less efficient because it cannot maintain uplink throughput at low SNR. At low SNR, AVX-512 reached only 0.40 Mbps/W, compared to 0.75 Mbps/W (ACC100) and 0.73 Mbps/W (AURORA), an improvement of about 88% and 83%, respectively. At high SNR, no AVX-512 value is available due to the aforementioned UE connection-loss issues; ACC100 (0.93 Mbps/W) and AURORA (0.92 Mbps/W) are nearly identical in efficiency.

3.4.5 Verification with an Emulated UE

Finally, to verify correct UE behavior and functionality, a verification test using an emulated UE from the Viavi TM500 was conducted to measure uplink throughput and RTT at high SNR with ten worker threads while using AURORA. The results from those measurements are shown in Figure 15.

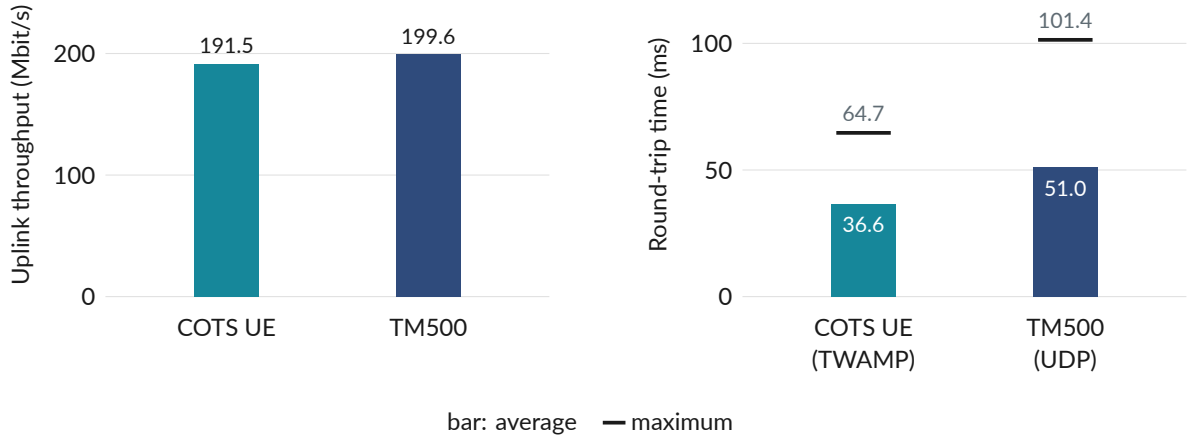

Figure 15: TM500 Verification Compared with the COTS UE – Uplink, High SNR, 10 Worker Threads, AURORA: (a) Throughput, (b) Round-Trip Time

The throughput results from the TM500 are very similar to those from the COTS UE, differing by less than 5%. This small difference is due to the SNR: With the TM500, it was not possible to configure exactly 20 dB; the closest achievable value was 22 dB. Therefore, the radio conditions were slightly better with the TM500 than with the COTS UE, which would explain the small difference in throughput.

The latency results of the TM500 again differ from those of the COTS UE: while the average RTT for the COTS UE in uplink with high SNR, 10 worker threads, and the AURORA was 36.6 ms, the average RTT measured with the TM500 was 51.0 ms, which is considerably higher. This is because with the COTS UE, TWAMP was used for RTT measurements, whereas for the TM500, RTT values from UDP traffic were used. As TWAMP-Test packets are themselves carried over UDP, the difference is more likely caused by the different measurement endpoints (Viavi RDA server vs. TWAMP server in the UPF), the larger packet size, and queueing behind the uplink traffic; the two RTT values are therefore not directly comparable. The use of UDP RTT instead of TWAMP for the TM500 measurements was due to time and setup constraints, which prevented the TWAMP server for the TM500 from being prepared in time. Regardless, the results from the emulated UE closely resemble those of the COTS UE used in the tests, thereby verifying the COTS UE's functionality.

3.5 Analysis

The results show that hardware acceleration affects throughput only when the software decoder is constrained by available processing resources. With ten worker threads, all three configurations reach nearly the same throughput in both directions and at both SNR levels. The differences

are within about 2% in the downlink and about 4% in the uplink and are in the range of normal measurement variation. In this configuration, the AVX-512 software implementation has enough processing power to keep up with the accelerator cards.

With one worker thread, this changes. In the uplink at low SNR, the higher number of LDPC iterations pushes the software decoder to its limit, and its throughput drops by about 54–55% compared to the accelerated configurations (70.2 Mbps vs. 151.1–156.6 Mbps). ACC100 and AURORA remain at the level they reached with ten threads, indicating that LDPC decoding no longer limits the system when offloaded. The same effect is observed in the downlink tests with one thread, where AVX-512 is about 13–16% slower than the accelerated configurations. The accelerators were not used for downlink offloading, so the cause of this difference cannot be determined from these measurements; CPU load profiling per worker thread would be needed to attribute it. At high SNR in the uplink with one thread with AVX-512, the system was unable to maintain the connection to the UE, indicating that the limited number of CPU worker threads is the bottleneck. With an accelerator, on the other hand, the system was able to maintain a stable connection with nearly the same performance as that of ten worker threads.

Between ACC100 and AURORA, the throughput is very similar. In most test cases, the difference is below 1.5%. The largest differences occur in the uplink at low SNR (about +3.6% for ACC100 with one thread and +4.4% with ten threads). In summary, AURORA offloads the uplink LDPC decoding with the same effectiveness as the ACC100, even though ACC100 additionally performs rate dematching.

The RTT results are influenced more by the number of worker threads and the traffic load than by the accelerator used. In the downlink, the average RTT is 22.7–27.8 ms with one thread and 50.8–58.9 ms with ten threads. The higher values with ten threads coincide with higher throughput, so they most likely result from queueing under higher load rather than slower processing. The three configurations differ by only a few milliseconds, without a consistent winner.

In the uplink with ten threads, latency is similar across all configurations (average: 34.9–43.3 ms). With one thread, the differences become visible. ACC100 has the lowest average and maximum RTT in both SNR scenarios (35.9/76.3 ms at high SNR and 42.3/72.4 ms at low SNR). AURORA is close at low SNR (44.2 ms average), but it shows noticeably higher maximum values (115.6 ms at high SNR and 121.8 ms at low SNR), which indicates occasional latency peaks. The software solution is the slowest on average at low SNR with one thread (51.0 ms) because processing is the bottleneck. While higher peaks were observed with the AURORA, the measurements alone do not allow a conclusion about the cause.

The TWAMP-only measurements, which were not affected by iPerf traffic, show a different picture. With one thread, AURORA had the lowest average and maximum RTT in both SNR scenarios (22.3–24.6 ms average and 34.8–35.6 ms maximum). With ten threads, ACC100 showed higher maximum values (79.9 ms and 146.0 ms) than AVX-512 and AURORA (41.6–50.5 ms). As the average values remain similar, these are individual outliers and not a general trend. Across all measurements, no configuration offers a consistent latency advantage, and the accelerators do not noticeably increase the latency.

Based on the power consumption measurements, the average power of the entire server was mostly 192 W for AVX-512, with two exceptions measuring 176 W; 208 W for AURORA; and 224

W for ACC100, except for two test cases where 208 W were measured. This indicates that the accelerator cards mostly add a constant power demand of 16 W (AURORA) and 32 W (ACC100), independent of the load. Note that the BMC reports power in steps of 4 W, so these differences correspond to four and eight quantization steps. In the downlink, this additional power is consumed even though the cards do not perform downlink offloading.

As a result, the power efficiency depends on two factors: the throughput gain through the accelerator and its additional power demand. If the throughput is equal, as with ten worker threads, the AVX-512 software solution is the most efficient configuration, with 4.17/3.80 Mbps/W in the downlink and 1.01/0.79 Mbps/W in the uplink (high/low SNR). AURORA is about 7–9% lower, and ACC100 is about 10–16% lower, which corresponds to their additional power demand. If the software solution is limited, as with one worker thread, the accelerators are more efficient. In the uplink at low SNR, ACC100 and AURORA reach 0.75 and 0.73 Mbps/W compared to 0.40 Mbps/W for AVX-512, an improvement of about 83–88%. In the downlink at high SNR, AURORA is about 10% more efficient than AVX-512 (3.51 vs. 3.20 Mbps/W), whereas at low SNR, AVX-512 and AURORA are on a similar level.

Between the two accelerators, AURORA is more power efficient (Mbps/W) in most cases by about 3–9% due to its 16 W lower power demand at nearly the same throughput. In the two uplink test cases with one worker thread, the power consumption of both accelerators is identical, so the efficiency is on a similar level. At high SNR, ACC100 and AURORA reach 0.93 and 0.92 Mbps/W. At low SNR, ACC100 is slightly ahead (0.75 vs. 0.73 Mbps/W) due to its higher throughput.

3.6 Technical Challenges and Limitations

Several limitations should be considered when interpreting the results. First, downlink offloading was still being tested for AURORA during the Plugfest, so the accelerator cards were not used in the downlink test cases. The downlink results, therefore, do not show the performance of hardware-accelerated downlink processing, and the observed differences mainly reflect the additional power demand of the installed cards.

Second, the offloaded functionality was not identical for the two accelerators. The current AURORA MVP implementation only offloads the uplink LDPC decoding, whereas the ACC100 additionally performs rate dematching in the uplink. A fully equivalent comparison requires rate dematching to be available on AURORA as well.

Furthermore, the server's CPU settings were not power-optimized: the CPU and all its cores were set to run at maximum performance at all times. Higher power savings could therefore have been achieved but could not be shown with this setting. The power values themselves come from the server's BMC, which reports power only in steps of 4 W, so small differences in power and efficiency cannot be resolved reliably.

Lastly, multi-UE behavior and scalability were not assessed. All test runs used a single UE, so the results do not reflect how the configurations would perform with a larger number of UEs simultaneously connected or active. Effects such as increased scheduling complexity, higher processing load, and changes in resource utilization under multi-UE conditions are outside the scope of this

assessment.

3.7 Summary

The test scenario shows that in a modern RAN server system with enough CPU cores, decoding is not the bottleneck. But if the number of CPU cores/worker threads is reduced, decoding limits the throughput. In addition, the connection stability between gNB and UE suffers. Hardware-accelerated decoding overcomes these issues, allowing a lower-scaled RAN server system with relatively low additional power consumption.

Between the two tested hardware accelerators, AURORA and ACC100 offer comparable throughput, while AURORA has lower power consumption and therefore greater power efficiency in nearly all configurations.

3.8 Outlook

The results show that the benefit of hardware acceleration depends strongly on the available CPU resources and the accelerator's power demand. Several follow-up steps are therefore of interest.

First, the tests should be repeated on a more power-efficient and smaller reference system. In this test, the accelerator cards added a constant power demand of 16 W (AURORA) and 32 W (ACC100) to a server consuming 192 W without acceleration. On a smaller platform with lower baseline consumption, the ratio of accelerator power to total system power changes. This directly affects the power efficiency and could shift the point at which acceleration pays off.

Second, according to Open Radio Systems, the functional scope of the AURORA implementation will be extended to an All-In-One 5G system component. Offloading downlink processing, such as encoding, will enable a hardware-accelerated downlink performance comparison and allow a like-for-like comparison with the ACC100, together with rate dematching in the uplink. In addition, AURORA will support full inline acceleration with fronthaul interfaces to O-RUs. In such a design, the accelerator would also handle the fronthaul traffic, and the data path would no longer pass through the host CPU for each processing step. This added functionality could be evaluated against servers/systems using other accelerators.

Third, the scalability of a single accelerator should be investigated. This includes using one AURORA card for several radio units, for example, four, as well as sharing one card between several DUs or gNB instances. Together with multi-UE scenarios, this would show how the accelerator behaves under higher load and whether its computational resources are efficiently distributed across several cells.

Lastly, larger MIMO configurations should be tested. The higher number of antennas and spatial layers considerably increases the processing load. Since the software decoder with one worker thread already reached its limit in the 2x2 MIMO setup, the relevance of hardware acceleration is expected to increase significantly in such scenarios.

Final Summary & Conclusion

The second xG-ALOE Plugfest gave participants the opportunity to evaluate their technologies in a multi-vendor environment and examine questions relevant to private 5G networks in practice. The three scenarios in this report addressed the quality of time distribution, the security of the software, and the efficiency of the RAN, and each produced concrete results and clear starting points for further work.

The timing scenario showed that SIB9-based synchronization can provide a very precise common time base. The relative time error between two UEs remained within ±140 ns, and the gPTP time distribution across the 5G system achieved steady-state accuracy below 1 µs. Two short time error spikes occurred during the gPTP measurement. They are most likely caused by electrical issues with the PPS connection in the test setup rather than by the 5G system itself, but this should be confirmed with repeated, longer measurements.

In the security scenario, the Security Test Platform analyzed four participant components totaling about 2 million lines of code. It confirmed 140 true-positive findings in the source code, identified 138 known vulnerabilities in 2,241 third-party components, and reported 89 findings from the autonomous penetration testing agent. The combination of the three modules proved useful, since each found weaknesses the others did not. The results are a snapshot, do not allow a ranking of the components, and are intended as a starting point for remediation by the vendors.

In the hardware acceleration scenario, the ORS AURORA decoder was compared with an AVX-512 software baseline and the Intel ACC100 card. With ten worker threads, all three configurations reached nearly the same throughput and latency, and the software solution was the most energy-efficient. With a single worker thread under low-SNR conditions, the uplink throughput of the software decoder dropped by more than 50%, whereas both accelerators maintained their performance and nearly doubled power efficiency. AURORA offloaded the uplink LDPC decoding as effectively as the ACC100 with lower power demand, although it was still an MVP implementation, and may therefore be suitable for systems with limited CPU resources.

Lessons Learned and Next Steps

Across all scenarios, the Plugfest also showed where the current limits are: a suspected electrical issue with the PPS connection, an early development stage, and the restriction to a single UE in the acceleration tests meant that some questions could only be answered partially. Working through these issues together helped participants to understand their causes and to improve their implementations and test procedures. Future plugfests can build on these findings, for example, by incorporating multi-UE scenarios, hardware acceleration in the downlink, longer timing measurements with a GNSS-referenced grandmaster, and regular security analyses during development. Together, the results help make private 5G networks more efficient, precise, and secure, and provide a solid basis for the next steps of the initiative.

In all test scenarios, EANTC AG supported the test plan creation and test execution, analyzed and verified the results, and wrote the test report for xG-ALOE based on the verified results.

References

[1] 3GPP TS 38.133, “NR; Requirements for support of radio resource management.”

[2] 3GPP TS 22.104, “Service requirements for cyber-physical control applications in vertical domains.”

[3] IEEE Std 802.1AS, “IEEE Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications.”

[4] ITU-T Recommendation G.8271, “Time and phase synchronization aspects of telecommunication networks.”

[5] ITU-T Recommendation G.8271.1, “Network limits for time synchronization in packet networks with full timing support from the network.”

[6] MITRE, “Common Weakness Enumeration (CWE).” https://cwe.mitre.org/

[7] FIRST, “Common Vulnerability Scoring System v3.1: Specification Document.” https://www.first.org/cvss/v3.1/specification-document

[8] 3GPP TS 38.331, “NR; Radio Resource Control (RRC); Protocol specification.”

[9] MITRE, “CWE VIEW-1400: Comprehensive Categorization for Software Assurance Trends.” https://cwe.mitre.org/data/definitions/1400.html

[10] OASIS, “Static Analysis Results Interchange Format (SARIF) Version 2.1.0.” https://docs.oasis-open.org/sarif/sarif/v2.1.0/sarif-v2.1.0.html

[11] OWASP Foundation, “CycloneDX Bill of Materials Standard” (standardized as ECMA-424). https://cyclonedx.org/

[12] GitHub, “GitHub Advisory Database.” https://github.com/advisories

[13] Google, “OSV – Open Source Vulnerabilities.” https://osv.dev/

[14] NIST, “National Vulnerability Database (NVD).” https://nvd.nist.gov/

[15] Open Radio Systems, “AURORA Radio Processing Unit – Data Sheet.” https://openradiosystems.com/assets/downloads/ors-aurora-data-sheet.pdf

[16] O-RAN Alliance WG4, “Control, User and Synchronization Plane Specification” (O-RAN.WG4.CUS, fronthaul split 7.2x).

[17] OCUDU Ecosystem Foundation (Linux Foundation), “OCUDU – Open Source CU/DU.” https://ocudu.org/

[18] Open5GS, “Open Source 5G Core.” https://open5gs.org/

[19] ESnet, “iPerf3.” https://software.es.net/iperf/

[20] IETF RFC 5357, “A Two-Way Active Measurement Protocol (TWAMP).” https://www.rfc-editor.org/rfc/rfc5357

Glossary

Time error (TE)
Difference between the time of a clock and a reference time. Its constant part is the mean offset; its dynamic part (jitter and wander) is the variation over time.

Relative time error
Time error between two devices instead of against a common reference. Error contributions common to both devices cancel out.

MTIE
Maximum time interval error: the largest peak-to-peak variation of the time error within an observation window.

Time-aware bridge
Network element that forwards gPTP time and corrects for its own residence time. Here, the 5G system acts as one bridge between NW-TT and DS-TT.

Grandmaster
Clock that provides the reference time for a PTP/gPTP domain.

gPTP servo
Control loop in a gPTP client that steers the local clock towards the grandmaster time.

Worker thread
CPU thread assigned to the main data processing of the gNB (e.g. decoding, rate dematching).

Look-aside accelerator
Accelerator card to which the host CPU sends selected processing steps (here: uplink LDPC decoding) and receives the results back, as opposed to an inline accelerator in the data path.

Goodput
Usable throughput at layer 2, without retransmissions and protocol overhead.

Throughput per watt
Average throughput divided by the average power of the whole server, in Mbit/s per W.

True positive
Tool finding that the expert review confirmed as a real weakness.

Weakness type vs. finding
A finding is a single occurrence in the code; a weakness type (CWE ID) groups findings of the same kind.

Reachability analysis
Check whether vulnerable code in a dependency can actually be invoked from the application.

CWE-1400 view
MITRE view that sorts all CWE weaknesses into 22 categories (e.g. Access Control, Memory Safety).

Plugfest
Joint measurement campaign in which participants test their components together in a multi-vendor setup.

© 2026 xG-ALOE / Fraunhofer HHI.
All rights reserved.

While every reasonable effort has been made to ensure accuracy and completeness of this publication, the authors assume no responsibility for the use of any information contained herein. All trademarks, brand names and logos mentioned in this report are the property of their respective owners.



xG-ALOE – an initiative powered by Fraunhofer Heinrich-Hertz-Institut, HHI

Einsteinufer 37, 10587 Berlin, Germany

https://xg-aloe.de/
info@xg-aloe.de

In cooperation with xG-RIC.

November 2026