CyberTwin: AI Powered Predictive Cybersecurity Digital Twin with Blockchain-Based Security Verification

Sangeetha Priya B[1], Haripriya P[2], Irfan Ali S K[3], Navajeevan A[4], Sathyes B[5], Thanu M[6]
[1,2,3,4,5,6]*Department of Computer Science and Engineering, SNS College of Technology, Coimbatore, Tamil Nadu, India*

***Abstract*—Security tests are usually reported as an attack label, a detector alert and a log file. That is rarely enough to show what the testers expected before the test, what the target actually did, and whether the record was left alone afterwards. This paper presents CyberTwin, a software-defined digital twin used to test three web-application controls in an isolated Docker lab: progressive account lockout, server-side ownership checks against insecure direct object references (IDOR), and input handling against SQL injection (SQLi). Every test follows a Predict–Commit–Execute–Verify (PCEV) lifecycle. A heuristic expectation is recorded and its SHA-256 digest committed before one bounded probe is run. Target-side evidence is then compared with that expectation, and an evidence summary is sealed in a local hash chain or, when the operator chooses, on a private Hyperledger Besu network. In the recorded runs the protected services returned HTTP 423 after three failed logins, HTTP 403 with no disclosure for a cross-user invoice request, and HTTP 400 with zero rows for an injection-style search string. These are case-level demonstrations, not accuracy figures. The classifier is trained on a small synthetic corpus, and latency, tamper-detection and live-chain measurements are left as future work.**

***Index Terms*—Blockchain, digital twin, evidence integrity, Isolation Forest, predict–Commit–Execute–Verify, Random Forest, web-application security.**

I. INTRODUCTION

When a team tests a security control, the write-up usually holds three things: the attack that was run, an alert from some detector, and a log. Read months later, that package leaves open what the team believed would happen before the test began. Expectations written after the event have a way of matching the outcome. Logs can be trimmed, edited or reordered. And a detector that labels a request as SQL injection says nothing about whether the database actually gave up a row.

We built CyberTwin to keep those questions apart in one small, repeatable lab: what did we expect, what happened, and can anyone check that the record has not changed since? The lab is a digital twin in a narrow sense. It is a software copy of a tiny enterprise made of an authentication service, a web gateway with a PostgreSQL invoice store, and a disposable product catalogue, together with a dependency graph over them [1], [2]. It is not a replica of any real organisation, and nothing in it touches real accounts or real data.

The central idea is a lifecycle we call Predict–Commit–Execute–Verify (PCEV). The operator records the expected outcome, the system commits a digest of that expectation and of the test configuration, and only then is a single bounded probe fired at the target. Evidence is collected from the target, compared with the expectation, and sealed. The order is the whole point. A forecast fixed before the probe cannot be quietly rewritten to fit what the probe returned.

The main contributions are:

1) a seven-stage PCEV lifecycle with a commit-before-execute rule enforced by stage gates;
2) a containerised three-scenario lab (credential compromise, IDOR and SQLi) in which a defended outcome, an attempted attack and a confirmed compromise are kept as different states;
3) an evidence pipeline that reports rule findings, a supervised classification and an anomaly score as separate outputs instead of one merged risk number; and
4) a two-path integrity layer, with a SHA-256 digest and SQLite hash chain as the default and an optional manual anchor on a private Besu QBFT network.

We have tried to be strict about scope. The forecaster is a deterministic heuristic, not a trained predictor. The classifier is trained on synthetic data. We report case-level observations rather than detection rates, and where a number does not exist yet we say so. Section II reviews related work, Section III states the threat model, Section IV describes the architecture, Sections V to VII cover the PCEV lifecycle, the evidence analysis and the integrity layer, Section VIII reports what was observed, and Sections IX to XI discuss limits and next steps.

II. RELATED WORK

A. Digital twins for security

Grieves and Vickers described the digital twin as a virtual counterpart that mirrors a physical system through its life [2]. Security uses have grown since then. A recent survey groups them by models, applications and tools and lists the architectural problems that remain open [1]. Most of that work targets cyber-physical systems and networks. CyberTwin is narrower on purpose. It works at the application layer, uses synthetic services only, and cares less about how faithful the twin is than about how an experiment on it is recorded.

B. Web-application weaknesses

The three scenarios correspond to entries in the OWASP Top 10:2025, where Broken Access Control is A01, Injection is A05 and Authentication Failures is A07 [3]. For SQL injection, OWASP puts parameterised queries first and adds allow-list validation and least-privilege database accounts as further layers, noting that least privilege limits the damage but does not prevent injection itself [4]. Halfond et al. classify injection attacks by technique and review the defences against each [5]. The tautology-style input used here, of the form ' OR '1'='1, is the simplest class in that taxonomy. For IDOR the expected control is a server-side check that the requester owns the object, whatever identifier the client sends [3].

C. Detection models and calibration

Random forests [6] and Isolation Forest [7] are standard tools, and both are available in scikit-learn [8]. A classifier's output score is not automatically a probability. Platt scaling [9] and later empirical comparisons of calibration methods [10] show that fitting a sigmoid or isotonic map on held-out folds gives better probability estimates, and the Brier score [11] is the usual way to measure them. CyberTwin applies sigmoid calibration through scikit-learn [12] but, as Section VIII explains, we have not yet computed any of these metrics on a held-out set.

D. Tamper-evident records

Schneier and Kelsey showed that audit logs kept on an untrusted machine can be made tamper-evident by chaining entries with cryptographic hashes [13]. NIST SP 800-92 describes log management practice more broadly [14]. Putz et al. place proofs of log records on a permissioned blockchain instead of storing the logs themselves there [15], which is also the pattern we follow: digests go on the chain and content stays off it. The hash function is SHA-256 from FIPS 180-4 [16]. Besu's QBFT consensus descends from Istanbul BFT [17], [18].

What differs in our case is what gets committed. Existing work protects the record of what happened. CyberTwin additionally commits the expectation that existed before anything happened. We have not carried out a systematic search for prior work with exactly this structure, so we claim only the combination, not priority.

III. THREAT MODEL AND DESIGN GOALS

A. What the design is meant to resist

We consider three problems. The first is *hindsight editing*: someone changes the recorded expectation after seeing the result.

The second is *record tampering*: stored evidence or a report is modified, deleted or reordered after it was sealed. The third is *misattribution*: an uploaded log is presented as a live run, evidence from one run is reused for another, or a blocked attempt is counted as a compromise.

B. What it does not resist

A target that reports false telemetry before the hash is computed is outside the model, as is a host administrator who controls both the ledger and the clock. We also do not address key management, denial of service, or testing against real systems. Section IX returns to these limits.

C. Design goals

The goals that follow from this are: (G1) commit the expectation before execution; (G2) run only bounded, operator-started probes against synthetic services; (G3) derive findings from target responses and not from the scenario label the operator picked; (G4) keep rule, classifier and anomaly outputs separate; and (G5) report the true status of every seal, so a local record is never shown as an on-chain one.

IV. SYSTEM ARCHITECTURE

A. Overview

Fig. 1 shows the layers. The operator works in a React 18 dashboard built with Vite. It offers scenario selection, the pre-execution forecast, stage navigation, findings, the impact map, and evidence and report views. A FastAPI service on Python 3.13 validates scenario and run identifiers, controls the PCEV lifecycle, dispatches one bounded probe and assembles the evidence. Analysis uses Pandas and scikit-learn. State is held in three SQLite databases, one each for experiments, consolidated reports and the fallback ledger. Reports are produced as PDF with ReportLab and carry a SHA-256 checksum.

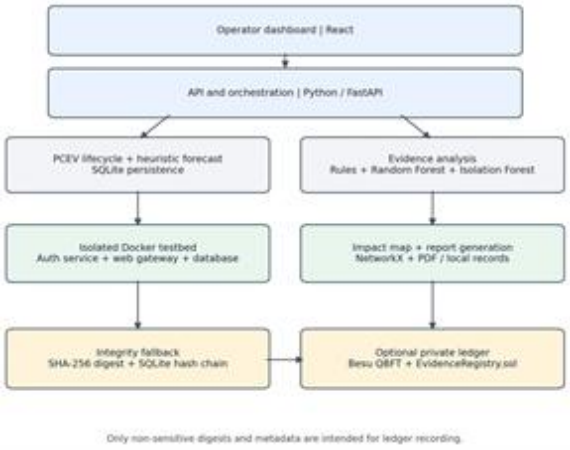

Fig. 1. CyberTwin architecture. Besu anchoring is optional and manual; the local SQLite hash chain is the default integrity path.

A. The lab

The lab runs under Docker Compose [19] on two networks, one for the application and probe tier and one internal network for the database. An authentication service listens on port 5000 at /login. A web service on port 80 exposes /api/records/{id} and /catalog/search. A PostgreSQL 16 container holds synthetic multi-tenant invoices, and separate attacker and client containers stand in for the probe source and a victim workstation. The injection scenario runs against a disposable SQLite catalogue and not against the invoice database. Keeping the two apart means the injection test cannot touch invoice data.

B. Scenarios

Table I lists the three active scenarios. IDOR and SQLi each also have a deliberately vulnerable fixture, which is used only when the operator selects it, to give a contrast with the protected mode.

This paper reports only the modes that were actually exercised. Other attack families appear in older registry entries of the code base, but they are not part of this work.

TABLE I ACTIVE SCENARIOS AND PROTECTED BEHAVIOUR

Scenario and Target	Protected Control	Protected Response
Credential compromise (SCEN-01). Auth service, /login, synthetic accounts	Per-account failed-attempt counter; lockout at three failures (default 300 s)	HTTP 401 on early failures; HTTP 423 once locked
Broken access control, IDOR (SCEN-02). Web service, /api/records/{id}, PostgreSQL 16 invoices	Server-side owner and session check on every record request	HTTP 403, no records disclosed
SQL injection (SCEN-03). Web service, /catalog/search, disposable SQLite catalogue	Syntax-rule rejection plus parameter binding	HTTP 400, zero rows

V. THE PCEV LIFECYCLE

A. Stages

An experiment moves through seven states, listed in Table II and drawn in Fig. 2. Each stage has a gate. The experiment cannot reach EXECUTED unless a commitment exists, and the interface disables the execute button until the forecast has been committed.

TABLE II PCEV STAGES

Stage	What is Recorded
DRAFT	Scenario ID, run ID and normalised parameters
PREDICTED	Heuristic expectation: expected status, outcome and targeted assets
COMMITTED	SHA-256 digest of the canonical forecast and configuration, with a timestamp; the snapshot becomes immutable
EXECUTED	One operator-started bounded probe; actual response telemetry
VERIFIED	Whether the intended defence, or a compromise, is evidenced
COMPARED	Match, residual and discrepancies against the forecast; the forecast itself is never replaced
SEALED	Final evidence digest, sealed locally or on-chain

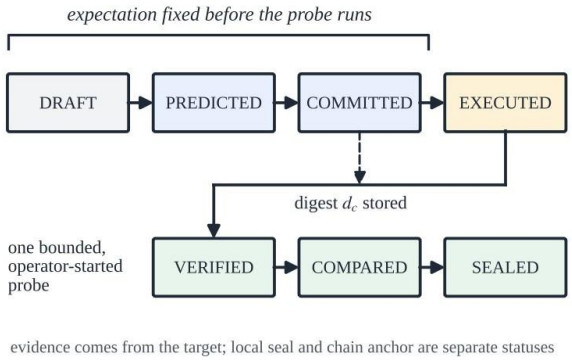

Fig. 2. The PCEV lifecycle. The digest of the expectation is stored before the probe is run.

B. Commitment and ordering

Let P be the canonical JSON form of the scenario ID, run ID, parameters and heuristic forecast. The commitment digest is

$$d_C = \text{SHA-256}(\text{canon}(P)) \quad (1)$$

and the intended ordering of the commitment time t_C, the execution time t_X and the sealing time t_S is

$$t_C < t_X \leq t_S \quad (2)$$

Inequality (2) is a design invariant that should be checked against event timestamps and, where the chain is used, against block timestamps. A successful state transition in the database is not by itself evidence that the chain finalised anything. If the parameters of a test change, the system opens a new experiment and a new commitment. The old one stays as it was. Run and scenario identifiers are attached to every record, so a forecast or result from another scenario cannot be picked up by mistake.

C. Verdicts

The VERIFIED stage produces one of three verdicts. CONFIRMED means the evidence shows the intended defence held, FAILED means it shows the defence did not hold, and INCONCLUSIVE means the evidence is not enough to say. Anchoring on the chain is always an explicit operator action. It is never triggered automatically when a report is generated.

VI. EVIDENCE ANALYSIS

A. Evidence from the target

Findings start from what the target returned: HTTP status, failed-login count, lockout state, ownership mismatch, SQL-syntax indicators and returned record counts. The rules read these before any model does, and they are written to avoid over-reading a single signal.

One HTTP 401 is a normal user mistake and not a compromise. An HTTP 423 with an active lockout and no successful login or token supports a defended state, not a breach. On the IDOR endpoint an HTTP 200 is not proof of disclosure until the returned record is checked against the requester's tenant. A legitimate catalogue search that returns authorised rows is an ordinary operation. A claim of database bypass needs direct evidence of rows or changes that should not have been possible, and SQL punctuation in the input is not that evidence.

B. Supervised classifier

The classifier is a scikit-learn random forest with 100 trees, maximum depth 8 and random seed 42, wrapped in sigmoid calibration with three-fold cross-validation [12]. It takes a 17-element telemetry vector, grouped in Table III, and predicts one of four classes: BENIGN, CREDENTIAL_COMPROMISE, BROKEN_ACCESS_CONTROL_IDOR and SQL_INJECTION. The training corpus is synthetic, with 400 samples and 100 per class. A stratified 75/25 split grouped by scenario-template seed gives 300 training and 100 test samples.

TABLE III TELEMETRY FEATURES USED BY THE CLASSIFIER

Group	Features	No.
Response	HTTP status, latency, exact rows returned	3
Endpoint type	Authentication, business-record and catalogue flags	3
Authentication	Failed attempts, total attempts, lockout flag	3
Access control	Ownership mismatch, data disclosure	2
Injection	SQL-syntax rule match, unexpected rows	2
Request shape	Query input, target-record parameter and user-identity header present	3
Completeness	Feature-completeness ratio	1

Because the corpus is generated from templates, the classifier's displayed confidence reflects the generator and not field performance. A value such as 96% shown for one event is a model output for that input. It is not an accuracy figure for the model.

C. Anomaly score and forecaster

A separate Isolation Forest with 100 trees and contamination 0.15 is fitted on a synthetic baseline of ordinary traffic. It returns an anomaly score and a label, ANOMALOUS_TELEMETRY or NOMINAL. This is an outlier measure. It is neither a probability of compromise nor a threat-type classifier, so the interface and the reports show it in its own field. The pre-execution forecaster is a deterministic mapping from scenario parameters, mainly protected versus baseline mode, to an expected status and outcome. It is tagged EXPERIMENTAL_HEURISTIC_FORECAST with the certainty level EXPERIMENTAL_UNCONFIRMED. Its value is that the expectation is written down and later compared. The word "predictive" in the title refers to this recording step and not to a trained attack predictor.

D. Impact map

A directed NetworkX graph [20] models a synthetic company, "Apex Global Enterprise", across SecOps, Finance and E-Commerce, with service, database, identity, session and data-asset nodes. Node states include TARGETED (expected target before execution), DEFENDED (a control blocked the probe, shown by direct evidence), CONFIRMED_AFFECTED (direct evidence of compromise or disclosure), POTENTIALLY_AFFECTED and UNKNOWN. A lockout is a defensive outcome and never raises the affected count. A pending execution is shown as "Pending Execution" and not as affected, and impact is filtered by run and scenario ID so one run cannot leak into another view. The graph explains dependencies. It is not a validated risk model.

E. Uploaded evidence

Operators can also upload. json, jsonl, csv, tsv, .log or .txt files of up to 5 MiB. The same rules and models analyse them, but the record keeps the provenance tag UPLOADED_UNVERIFIED. An upload cannot create an execution run or a forecast-versus-actual match. Packet captures and Windows event logs are not supported.

VII. EVIDENCE INTEGRITY LAYER

A. Local hash chain

The fallback ledger links each record to the one before it. In general form, for evidence digest d_i and previous link h_{i-1},

$$h_i = \text{SHA-256}(h_{i-1} \parallel d_i) \quad (3)$$

with a fixed genesis value for h_0. Changing, deleting or reordering an earlier entry changes every later link, so a mismatch shows up when a verifier holds a later trusted head. Without such an independent copy the host owner could recompute the whole chain, which is why the chain shares a failure domain with its host unless copies are exported (Fig. 3).

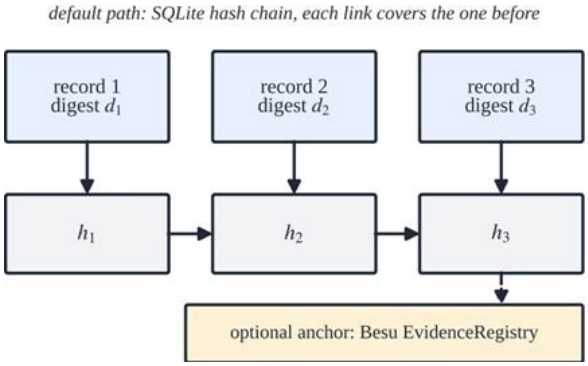

Fig. 3. Hash-chain links and the optional chain anchor. Only digests and limited metadata leave the host.

B. Private chain anchor

The optional path uses Hyperledger Besu 24.7.0 with QBFT consensus, four validators and chain ID 13371,

all running in Docker on one host. An EvidenceRegistry contract written in Solidity 0.8.24 stores evidence digests, timestamps and verdict metadata, and enforces that a commitment comes before its seal. Cleartext passwords, user records and attack strings are not written to the chain. Like other BFT protocols, QBFT needs $n \geq 3f + 1$ validators to tolerate f faulty ones [17], so four validators tolerate one. Transactions go through JSON-RPC on the local host.

C. Three statuses that must not be confused

TABLE IV EVIDENCE STATUSES

Status	Means	Does not mean
Local seal (hash chain)	Digest was computed and written to the fallback ledger	A chain transaction was sent or finalised
Chain commitment	A pre-execution digest was recorded with a transaction hash and block	The target sensor was truthful or the attack succeeded
Chain seal	A post-execution digest and verdict were recorded after the commitment	The model is accurate, or the chain is decentralised

A report marked AUDIT_SEALED_LOCAL, or one whose transaction hash is null or pending, is a local record. It is never described as on-chain sealing.

D. Verification procedure

An auditor checks a run in five steps:

1) recompute the canonical digest of the stored forecast and configuration and compare it with d_C;
2) check that $t_C < t_X \leq t_S$ in the stored records and, where used, in block timestamps;
3) recompute the hash chain from the genesis value and compare its head with an exported or anchored copy;
4) recompute the SHA-256 of the PDF report and compare it with the recorded checksum; and
5) confirm that the displayed status matches the real path, local or with a transaction receipt.

Table V states what each kind of alteration should do under this procedure. These are expected outcomes from the design. The tamper tests that would confirm them have not been run yet.

TABLE V EXPECTED EFFECT OF ALTERATIONS (DESIGN ANALYSIS, NOT YET TESTED)

Alteration	Expected to be caught by	Condition
Change payload bytes of a stored record	Step 1 digest mismatch; chain mismatch (step 3)	Committed digest or later head is held elsewhere
Delete a record from the middle of the chain	Chain break at the next entry (step 3)	A later trusted head exists
Reorder or change timestamps	Ordering check (step 2); chain mismatch (Step 3)	Block timestamps available if on-chain
Replace a record and recompute the whole chain	Only by an external copy or chain anchor	Without one, not detected
Falsify telemetry before hashing	Not detected by this layer	Outside the threat model

VIII. EVALUATION AND OBSERVATIONS

A. Scope

The evidence in this paper supports a functional demonstration, not a benchmark. It comes from recorded runs of the three scenarios (response status, counts and ledger references) and from test summaries produced during development. In the environment of the last full test pass Docker Desktop was not running for part of the time, so live probes and JSON-RPC calls to Besu could not be repeated there. The block numbers in Table VI were read from reports generated earlier. We did not re-check them against a running node while preparing this paper. We have no host CPU or memory profile, no repeated timing measurements and no independent evaluation dataset.

B. Scenario observations

TABLE VI RECORDED OBSERVATIONS IN PROTECTED MODE

Test and input	Observed	What it supports
Credential threshold. Three failed synthetic logins	HTTP 423, lockout reported, no successful login or token; impact state DEFENDED; chain block #40309	Consistent with progressive lockout in this run. Not resistance to credential stuffing or distributed

		attempts.
IDOR. user_a requests INV-2026-003, owned by user_b	HTTP 403, nothing disclosed, zero affected assets; chain block #40729	Consistent with server-side ownership checks for this case. Not complete access-control coverage.
SQLi-like input. Hardware' OR '1'='1	HTTP 400, zero rows; chain block #41177	Rejection of this payload on this path. Not general injection immunity.
Benign search Hardware	HTTP 200 with authorised rows (two in the recorded run). Rule result benign, yet the Isolation Forest flagged an anomaly in one snapshot	An unresolved threshold or baseline issue, not evidence that the query was malicious.

These are case-level demonstrations. We do not pool them into an accuracy score, because the number and independence of the evaluated samples have not been established. The SQLi result concerns one payload and not all of SQL syntax. The IDOR result concerns one ownership mismatch. The credential result concerns one fixed threshold in a synthetic service.

One earlier IDOR run is worth stating, because it shows the record doing its job. That run used the deliberately vulnerable fixture, returned another tenant's synthetic invoice and was sealed with a FAILED verdict. We kept it. It is the reason this paper does not say that every IDOR run was blocked, and an append-only record is only useful if unwelcome results stay in it.

C. Software tests

Table VII lists test results taken from separate runs. They are not added together. Counts for some groups differ between snapshots, for example for the impact-state and frontend suites, so each figure below comes from a single recorded snapshot. The full Python run is reported as it was, with its errors, because those errors were environment-related and not passes.

TABLE VII TEST RESULTS FROM RECORDED SNAPSHOTS

Suite	Recorded result	What it supports
Frontend regression scripts	37 / 37 passed	Selected UI state and flow behaviour
Hardhat Evidence Registry unit tests	19 / 19 passed (in memory)	Contract logic only, not a live Besu deployment
Threat classifier; scenario registry; impact states; evidence upload	22; 9; 15; 25 passed	Component behaviour on the synthetic set-up
PCEV core	14 passed, 1 error (Besu offline)	Needs a rerun with the runtime online
Besu acceptance	19 passed, 7 failed (Besu offline)	Live chain integration is not demonstrated
Full Python discovery	182 passed, 12 skipped, 9 errors, 1 failure (Docker unavailable)	Integration status is incomplete

Passing unit tests support component behaviour under the tested conditions. They do not establish end-to-end reliability or detector accuracy.

D. Planned measurements

The following are planned and none has been completed:

1) 100 controlled PCEV cycles across the three scenarios, reporting median (P50) and 95th-percentile (P95) time for digest creation, chain commitment, probe execution, model inference and sealing, including offline-chain cases rather than dropping them;
2) a classification report and 4×4 confusion matrix on the 100-sample grouped hold-out, then a calibration curve and Brier score [11] computed from that set;
3) the tamper experiments of Table V, run against both the local chain and the anchored path;
4) an end-to-end browser check from scenario selection through to the report and optional anchor; and
5) a rerun of the safe test suites with Docker and Besu running, against a fixed commit with logs archived.

IX. DISCUSSION

The practical value of CyberTwin is that a scenario comes with an evidence trail. In the observed cases, status codes, lockout state, ownership mismatch and row counts say more than a scenario label would

Keeping the expectation separate from the post-run evidence also makes a wrong forecast visible instead of letting the report overwrite it. Table VIII sets this against a typical log-plus-detector workflow. The entries describe how the design is meant to behave. They are not measured benefits, and whether PCEV reduces hindsight bias in practice still has to be tested.

TABLE VIII DESIGN COMPARISON WITH A TYPICAL LOG-AND-DETECTOR WORKFLOW

Property	Typical log and detector	CyberTwin (by design)
Expectation recorded before the test	Seldom	Yes; digest committed first
Expectation editable after the test	Usually,	No; a changed test is a new experiment
Defended, attempted and compromised told apart	Left to the analyst	Separate impact states with evidence rules
Uploaded versus live evidence	Often mixed	Provenance tag UPLOADED_UNVERIFIED
Tamper evidence	Only with log shipping or write-once storage	Hash chain, plus optional chain anchor

The two analysis models will sometimes disagree, and the benign "Hardware" search shows it: the rules saw an ordinary query while the Isolation Forest saw an outlier. We show both signals instead of hiding one, and treat the case as a reason to review the threshold and baseline and to write down a policy for conflicting outputs. It also shows why displayed classifier confidence should not be quoted as accuracy.

On integrity, the SQLite chain is a useful fallback, but it sits in the same failure domain as its host unless copies are exported. A four-validator Besu network on a single development machine does not give the independence or availability of a consortium run by separate organisations [17]. The chain tells a verifier that a digest existed at a given time and has not changed. It does not say that the telemetry behind the digest was honest.

Ethically, every test runs against local synthetic services with synthetic accounts and records. The vulnerable fixtures are comparison conditions inside that lab and are not meant to be pointed at any external system.

X. LIMITATIONS AND THREATS TO VALIDITY

1) Narrow synthetic workload. Three scenarios, synthetic accounts, invoices and catalogue rows. The results may not carry over to real identity providers, ORM layers, database engines or complex business logic.
2) Limited model evaluation. The random forest was trained on 400 synthetic samples. No held-out report, leakage analysis or calibration result is available yet, so we make no accuracy claim.
3) Heuristic forecast. The PCEV forecaster is deterministic. Calling the system predictive refers to recorded expectation, not to demonstrated predictive AI.
4) Runtime verification gap. Docker and live Besu were unavailable for part of the last full test pass. Local sealing was observed in some runs, while on-chain status was pending or unverified in them. In-memory unit tests do not replace live end-to-end evidence.
5) Trust boundaries. Hashes and consensus protect recorded digests under stated assumptions, not the accuracy of the source telemetry. Compromised sensors, host administrators, clock manipulation and key management need further threat modelling.
6) No performance benchmark. Without host specifications and repeated timings, we make no statement about speed or latency.
7) Reproducibility and consistency. The snapshots were taken at different stages of development, and some of their details differ, such as the number of rows returned for a legitimate query (two in one record, four in another). A versioned commit, an environment manifest, the dataset-generation procedure and raw logs should accompany the final paper.

XI. CONCLUSION AND FUTURE WORK

We presented CyberTwin, a digital-twin prototype for testing credential compromise, broken access control and SQL injection in an isolated lab. Its PCEV lifecycle ties a pre-execution expectation and its digest to a bounded probe, the evidence collected from the target, a comparison verdict and a final record. The recorded cases show the workflow end to end: a 423 lockout, a 403-ownership check and the rejection of an injection-style input.

The evidence supports a working prototype and not a validated predictive security system. The next steps are to rerun the live integration tests with Docker and Besu running, freeze a software and dataset version, evaluate the classifier on properly separated held-out runs, look into the benign-query anomaly, and measure PCEV latency and tamper detection. A later study should also test whether committing the expectation first measurably improves auditability compared with ordinary post-event logging.

DECLARATIONS

Funding: [to be completed by the authors]. *Conflicts of interest:* [to be completed by the authors]. *Author contributions:* [to be completed by the authors]. *Data and code availability:* the implementation and synthetic test data are held locally; [repository or access conditions to be added by the authors]. *Ethics:* all tests used isolated lab services with synthetic accounts and records, and no external target was tested.

DECLARATION OF GENERATIVE AI USE

A generative AI tool (Claude, Anthropic) was used to assist with drafting, language editing and reference checking of this manuscript. The authors reviewed and verified all content and take full responsibility for it.

REFERENCES

[1] R. Qureshi, A. Asensio, M. Imran, J. Garcia, and X. Masip-Bruin, "A survey on security enhancing digital twins: Models, applications and tools," *Computer Communications*, vol. 238, Art. no. 108158, 2025, doi: 10.1016/j.comcom.2025.108158.

[2] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in Transdisciplinary Perspectives on Complex Systems, F.-J. Kahlen, S. Flumerfelt, and A. Alves, Eds. Cham, Switzerland: Springer, 2017, pp. 85–113, doi: 10.1007/978-3-319-38756-7_4.

[3] OWASP Foundation, "OWASP Top 10: 2025," [Online]. Available: https://top10.owasp.org/2025/0x00_2025-Introduction/. [Accessed: Oct. 28, 2026].

[4] OWASP Foundation, "SQL injection prevention cheat sheet," OWASP Cheat Sheet Series, [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. [Accessed: Oct. 28, 2026].

[5] W. G. J. Halfond, J. Viegas, and A. Orso, "A classification of SQL-injection attacks and countermeasures," in Proc. IEEE Int. Symp. Secure Software Engineering (ISSSE), Arlington, VA, USA, Mar. 2006, pp. 13–15.

[6] L. Breiman, "Random forests," Machine Learning, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.

[7] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in Proc. 8th IEEE Int. Conf. Data Mining (ICDM), Pisa, Italy, 2008, pp. 413–422, doi: 10.1109/ICDM.2008.17.

[8] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[9] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in Advances in Large Margin Classifiers, A. J. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, Eds. Cambridge, MA, USA: MIT Press, 1999, pp. 61–74.

[10] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in Proc. 22nd Int. Conf. Machine Learning (ICML), Bonn, Germany, 2005, pp. 625–632.

[11] G. W. Brier, "Verification of forecasts expressed in terms of probability," Monthly Weather Review, vol. 78, no. 1, pp. 1–3, Jan. 1950.

[12] scikit-learn developers, "CalibratedClassifierCV," scikit-learn documentation, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html. [Accessed: Oct. 28, 2026].

[13] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," ACM Transactions on Information and System Security, vol. 2, no. 2, pp. 159–176, May 1999.

[14] K. Kent and M. Souppaya, Guide to Computer Security Log Management, NIST Special Publication 800-92. Gaithersburg, MD, USA: National Institute of Standards and Technology,

Sep. 2006. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf.

[15] B. Putz, F. Menges, and G. Pernul, "A secure and auditable logging infrastructure based on a permissioned blockchain," Computers & Security, vol. 87, Art. no. 101602, 2019, doi: 10.1016/j.cose.2019.101602.

[16] National Institute of Standards and Technology, Secure Hash Standard (SHS), FIPS PUB 180-4. Gaithersburg, MD, USA, Aug. 2015, doi: 10.6028/NIST.FIPS.180-4.

[17] H. Moniz, "The Istanbul BFT consensus algorithm," arXiv:2002.03613, 2020.

[18] Hyperledger Besu contributors, "Configure QBFT consensus," Hyperledger Besu documentation, [Online]. Available: https://besu.hyperledger.org/private-networks/how-to/configure/consensus/qbft. [Accessed: Oct. 28, 2026].

[19] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," Linux Journal, vol. 2014, no. 239, Art. no. 2, Mar. 2014.

[20] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. 7th Python in Science Conf. (SciPy)*, Pasadena, CA, USA, 2008, pp. 11–15.