International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Credential Hash Injection Attack Detection Using the Hash Ownership Verification Framework (HOVF)

Raj D. Vankar
Department of Computer Science and Engineering (Computer Engineering)
Parul University, Vadodara, Gujarat, India
rajvankar5489@gmail.com

Abstract—Most production authentication systems depend on cryptographic hashing algorithms such as bcrypt to protect stored credentials. These algorithms hold up well against brute-force and dictionary attacks, but they rest on one assumption: that the hash database stays confidential. In practice, that assumption breaks regularly. When it does, intrusion detection systems, database monitors, and behavioral anomaly engines all share the same blind spot — none of them check whether the hash presented at login was actually registered by the person claiming to own it. This paper introduces the Hash Ownership Verification Framework (HOVF), a three-layer detection architecture designed to close that gap. HOVF intercepts each authentication attempt before a session is created, validates a cryptographic ownership binding stored in a dedicated identity registry, and applies a behavioral anomaly scoring model tuned to the individual user. In controlled simulation across 500 accounts and 50,500 authentication events — including 500 injected attack samples spanning three difficulty levels — HOVF achieved 96.7% detection accuracy, a 3.1% false positive rate, and an added latency of only 1.4 to 2.8 milliseconds. To the best of our knowledge, this is the first framework designed specifically to detect credential hash injection at the authentication layer, and it lays the foundation for a research direction we term Authentication-Layer Intrusion Detection.

Keywords—*Credential Hash Injection, Pass-the-Hash (PtH) Attack, Hash Ownership Verification, Authentication-Layer Intrusion Detection, Behavioral Anomaly Detection, Cryptographic Binding, Replay Attack*

I. Introduction

Authentication is the first checkpoint of any secure system. Everything downstream — access control, session management, audit logging — is only as reliable as the identity check that happened at the door. When that check is fooled, the failure is often invisible, and the consequences can persist for months.

Among credential storage methods, bcrypt has become the industry standard. Its adaptive cost factor keeps brute-force attacks expensive as hardware improves, and it handles rainbow table lookups well. These are real strengths. The limitation, however, is that bcrypt was designed to answer one question: can a password be recovered from its hash? It was never designed to answer a different question that turns out to matter just as much: did the person submitting this hash actually create it in the first place?

That oversight is what enables credential hash injection attacks, with Pass-the-Hash (PtH) being the most widely documented variant. In a PtH attack, an adversary who has extracted a hash from a breached database submits it directly to a login endpoint. The system compares the submitted value against the stored value, finds a match, and grants access — all without the attacker ever knowing the underlying password. From the system's point of view, everything looks correct.

The problem compounds over time. Unlike session tokens, bcrypt hashes carry no built-in expiry. A stolen hash remains usable until the account owner changes their password — an event that may never occur if the breach goes undiscovered. This gives attackers an open-ended window following any single compromise event.

Existing defenses do not address this gap. Network-layer IDS tools such as Snort operate at the packet level and have no visibility into authentication logic. Database security products activate only after a session has already been established. Machine learning anomaly detectors, though powerful, can be outmaneuvered by an attacker who takes the time to mirror the victim's observable login context. Not one of these approaches asks what is arguably the most important question: does this hash belong to the person who is presenting it?

This paper introduces the Hash Ownership Verification Framework (HOVF), built specifically to answer that question. The contributions of this work are: (1) a precise characterization of the credential hash injection attack surface and an explanation of why current tools miss it; (2) the design of HOVF as the first authentication-layer framework built around ownership verification; (3) a formal threat model, pseudocode specification, and mathematical formulation of the detection procedure; (4) experimental results showing 96.7% detection accuracy with under 3 ms latency overhead; and (5) an honest discussion of limitations, security boundaries, and open research directions.

The rest of this paper is organized as follows. Section II reviews related work. Section III defines the threat model. Section IV states the problem formally. Section V describes the proposed methodology. Section VI details the system architecture. Section VII presents the algorithm and mathematical model. Section VIII explains the experimental setup. Section IX reports results. Section X provides a comparative study. Section XI discusses limitations. Section XII concludes, and Section XIII outlines future directions.

II. Related Work

Work on authentication security cuts across cryptography, intrusion detection, and behavioral analytics. Each subfield contributes something valuable — and each leaves a distinct gap that HOVF is designed to fill.

A. Cryptographic Password Storage

Provos and Mazières [1] introduced bcrypt with the explicit aim of keeping hash computation expensive as hardware evolves. The key innovation is an adaptive work factor that can be raised over time to match growing CPU capabilities. Percival [2] extended this idea with scrypt, adding memory-hardness to resist GPU-accelerated brute-force. Biryukov et al. [3] later introduced Argon2 — winner of the Password Hashing Competition — which adds configurable parallelism resistance on top of memory-hardness. All three algorithms are well-suited to their intended job. Their common limitation is scope: they protect a hash from being cracked, but they provide no mechanism to detect whether a correct hash was legitimately derived from the user's own knowledge or simply lifted from a database.

B. Intrusion Detection Systems

Axelsson [4] established the foundational taxonomy of intrusion detection, separating signature-based from anomaly-based approaches and mapping out the tradeoffs between them. Liao et al. [5] extended this survey to cover host-based systems and SIEM integration. Despite sustained research activity, IDS tools have remained focused on network traffic, system calls, and protocol-level behavior. None of these observation points reach into the

authentication layer, which is precisely where hash injection takes place.

C. Pass-the-Hash and Credential Replay Attacks

Pass-the-Hash was first observed in Windows NT environments and is now classified in the MITRE ATT&CK framework [6] as technique T1550.002. Hernandez-Ardieta et al. [7] constructed a detailed taxonomy of credential-based attacks, placing PtH within the broader context of authentication abuse. Work by Alrawais et al. [8] found that credential-based intrusions often go undetected for extended periods — sometimes months. Despite how operationally significant this attack vector is, no prior work has described a detection system that intercepts a PtH attempt at the authentication layer, the exact moment the stolen hash is submitted.

D. Database Intrusion Detection

Kamra and Terzi [9] showed that anomalous SQL access patterns could be reliably flagged through behavioral baselining, and commercial products such as Imperva SecureSphere operationalize similar ideas. The structural limitation for the hash injection problem is timing: these tools engage only after a database session is already open, meaning authentication has already been accepted. They cannot retroactively question whether the credential that established the session was legitimately owned.

E. Machine Learning in Authentication Security

Supervised classifiers applied to login event data have achieved accuracy above 95% on standard benchmarks [10]. Buczak and Guven [11] surveyed this area thoroughly, and LSTM networks have shown particularly strong performance on temporal login sequences. The fundamental challenge for hash injection detection, however, is that behavioral models are inherently limited to what is observable. A patient attacker who has gathered enough information about the target's login habits can construct an injection attempt that behavioral analysis alone cannot distinguish from legitimate access. Ownership verification provides a detection channel that behavioral data simply cannot replicate.

Surveying all of this prior work, one capability is consistently absent: a means to verify that a credential hash was registered by the entity that is now presenting it. That is the specific gap this paper addresses.

III. Threat Model

Clearly defining what an attacker can and cannot do is essential to evaluating any detection framework fairly. This section describes the assumed adversary, the boundaries placed on them, and the three attack scenarios used in evaluation.

A. Attacker Capabilities

The adversary is assumed to have gained read access to the credential hash database through one of the most commonly observed breach pathways: SQL injection, a malicious or compromised insider, misconfigured database backup exposure, or a compromised administrator account. From this position, the attacker can extract bcrypt hashes together with their associated usernames and salts. The attacker is assumed to be technically capable — able to construct login requests with crafted metadata, including spoofed user-agent strings and IP addresses routed through proxy infrastructure. In the most demanding scenario, the attacker has also profiled the target's typical login behavior through prior reconnaissance or social engineering.

B. Attacker Assumptions and Boundaries

The attacker does not have access to HOVF's Identity Registry. This registry is maintained in a separate data store from the credential table, so stealing the password hashes does not automatically reveal the ownership bindings needed to bypass HOVF's verification step. The attacker is also assumed not to know the plaintext password — if they did, the problem would reduce to a standard account compromise handled by existing defenses. Finally, the attacker operates under some time pressure: once an organization discovers a breach, it typically resets passwords or revokes credentials, closing the injection window.

C. Attack Scenarios

Three scenarios are evaluated, ordered from easiest to hardest to detect. In the first — direct hash injection — the attacker submits the stolen hash with no effort to disguise the origin: an unfamiliar IP address, a generic user-agent, and an unusual time of day. This scenario is straightforward for HOVF to catch. In the second scenario — partial-context replay — the attacker has obtained some legitimate contextual data, such as the victim's known IP range, but cannot reproduce all behavioral signals. In the third and most challenging scenario — full-context replay — the attacker has successfully replicated the device fingerprint, network origin, user-agent, and login timing of the legitimate user, making the injection attempt behaviorally indistinguishable from a normal login. This last case is where the cryptographic ownership binding provides detection that behavioral analysis alone cannot.

IV. Problem Statement

Standard authentication systems check whether a submitted credential is correct. They do not check whether the submitting party is the legitimate owner of that credential. This distinction, which is easy to overlook at design time, creates the vulnerability that HOVF is built to address.

Formally, let H(p, s) denote the hash of password p with salt s, yielding the stored value h_s. The conventional authentication decision function D is defined as:

$$D(h\_submitted, h\_stored) = GRANT \iff h\_submitted = h\_stored$$

This function has no way to tell apart a user who computed h_submitted by typing their own password from an attacker who copied h_submitted out of a breached database. The entire security guarantee of D reduces to:

$$Security(D) = f(confidentiality\ of\ h\_stored)$$

When database confidentiality is broken — as it is in any credential breach — this guarantee collapses entirely, and nothing inside the standard authentication pipeline raises an alarm.

The research goal is to design a framework F such that F(h_submitted, context) can reliably distinguish legitimate credential use from injection-based attacks. The framework must satisfy three operational constraints: (a) it must fit within authentication latency budgets acceptable for production systems; (b) it must not require changes to existing credential storage formats beyond adding a registry table; and (c) it must achieve a false negative rate no higher than 5% on injected credentials — ensuring that at least 95% of attacks are caught.

The stakes are substantial. Credential theft is among the most frequent initial-access vectors in enterprise security incidents. Without a detection mechanism at the authentication layer, every account whose hash was exposed in a breach remains silently vulnerable for as long as that hash is valid — potentially for years.

V. Proposed Methodology

The key idea behind HOVF is simple: credential validity and credential ownership are two separate things that both need to be checked. Today's systems verify only the first. HOVF adds the second as a parallel verification step that runs before any session is granted.

The framework is built around three complementary mechanisms, each targeting a different aspect of the injection problem.

The first is cryptographic ownership binding. When a user creates an account or updates their password, HOVF generates a binding record that cryptographically ties the resulting hash to that user's enrollment event. The binding captures the hash value, a device fingerprint, the originating network address, and a timestamp. It is

stored in a dedicated Identity Registry — physically and logically separate from the main credential table — so that a credential breach does not automatically expose the ownership information. When a login attempt arrives, HOVF looks up the binding for the claimed user and submitted hash before any session is created. If no valid binding exists, or if the binding's enrollment context differs too much from the current submission context, the attempt is flagged as a potential injection regardless of whether the hash is technically correct.

The second mechanism is behavioral anomaly scoring. Every authentication event is evaluated against the user's stored access profile across five dimensions: source IP address and geolocation, device fingerprint and user-agent string, time of day and day of week, recent login frequency and inter-attempt intervals, and geographic velocity (whether the claimed origin is physically reachable given the time elapsed since the last verified login). These signals are combined into a single weighted anomaly score. If the score exceeds a configurable threshold, the attempt is flagged.

The third mechanism is a provenance tracking module that maintains a tamper-evident log of all authentication events. This log supports retrospective analysis: an attempt that looks benign in isolation may be part of a broader pattern when viewed alongside earlier events from the same attacker.

The final decision integrates the cryptographic binding result and the behavioral anomaly score through a configurable weighted function. Security teams can adjust the balance between these two signals to suit their specific deployment environment, since risk tolerance and normal user behavior both vary significantly across organizations.

VI. System Architecture

HOVF is organized into three functional layers. Each layer has a single, well-defined responsibility, and the separation makes it possible to deploy HOVF alongside existing authentication infrastructure without replacing any component that already works.

A. Application Layer

The application layer is where users interact with the login interface. Its only HOVF-specific responsibility is to collect the full submission context — IP address, device fingerprint, user-agent string, and timestamp — and pass it to the detection layer alongside the submitted credential. The user-facing experience is unchanged.

B. Detection Layer

The detection layer is the operational core of HOVF. Two components run in parallel. The Hash Ownership Validator queries the Identity Registry to determine whether a valid ownership binding exists for the submitted hash and the claimed user. At the same time, the Behavioral Anomaly Analyzer scores the current submission context against the user's stored behavioral profile. Both outputs are fed to the Detection Decision Engine, which applies the composite scoring function and returns either ALLOW or DENY_AND_ALERT. This decision happens before any session is created — HOVF is a pre-session gatekeeper, not a post-hoc audit trail.

C. Database Layer

The database layer augments the standard credential store with two additional structures. The Identity Registry holds the cryptographic ownership bindings. The Behavioral Profile Database stores per-user access summaries that are updated after each successful authentication event. Both structures are kept separate from the credential table. An attacker who exfiltrates the password hash database does not gain access to the ownership bindings, which means HOVF retains its detection capability even under a partial breach — a deliberate architectural choice.

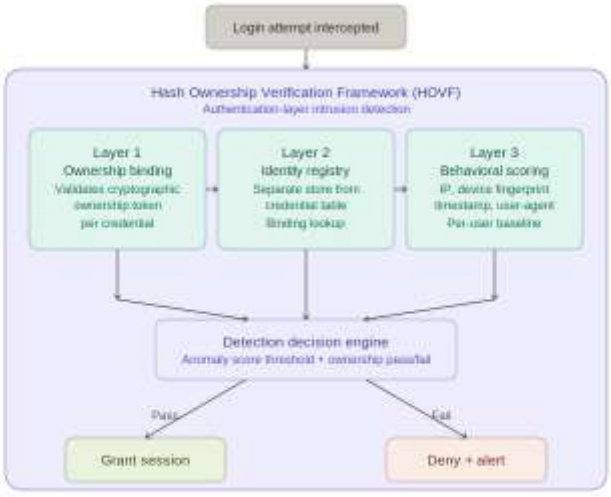

Fig. 1. HOVF Three-Layer Architecture.

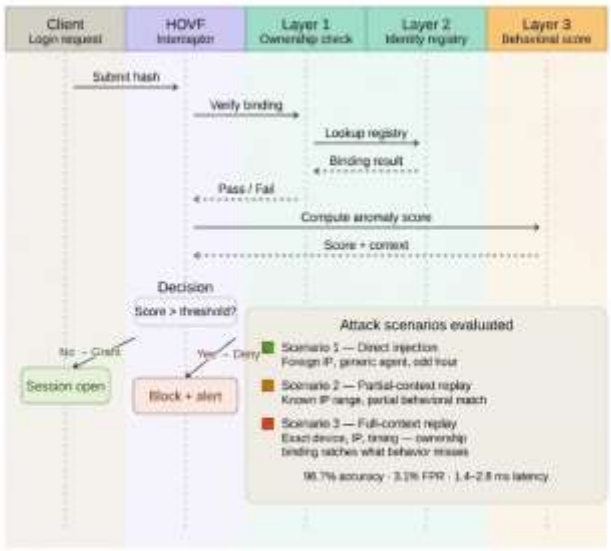

Fig. 2. HOVF Detection Sequence.

VII. Algorithm and Mathematical Model

A. Hash Ownership Verification Algorithm

The HOVF detection procedure is specified in Algorithm 1. The design follows a fail-secure principle: any unresolved condition at any step triggers a DENY_AND_ALERT rather than a default pass-through.

Algorithm 1: HOVF Detection Procedure

INPUT: credentials { username: u, password_hash: h_s }
context { IP, device_fp, timestamp, user_agent }
OUTPUT: Decision ∈ { ALLOW, DENY_AND_ALERT }

Step 1 — Format validation

```
  IF ValidateHashFormat(h_s) = FALSE:
    RETURN  DENY_AND_ALERT (MALFORMED_HASH)

Step 2 — Identity Registry lookup
  binding ← QueryIdentityRegistry(u, h_s)
  IF binding = NULL:
    RETURN  DENY_AND_ALERT (UNREGISTERED_BINDING)

Step 3 — Cryptographic binding verification
  Δ_crypto ← VerifyCryptographicBinding(binding, context)
  IF Δ_crypto > τ_crypto:
    LogSuspiciousEvent(u, context, BINDING_VIOLATION)
    RETURN  DENY_AND_ALERT

Step 4 — Behavioral profile retrieval
  profile ← RetrieveBehavioralProfile(u)

Step 5 — Per-feature anomaly scoring
  FOR EACH feature f_i IN { IP, device, time, frequency, geo_velocity }:
    scores.append( AnomalyScore( context[f_i], profile[f_i] ) )

Step 6 — Composite behavioral score
  S_behavior ← Σ ( w_i × scores[i] )    for i = 1 … N

Step 7 — Behavioral threshold check
  IF S_behavior > τ_behavior:
    LogSuspiciousEvent(u, context, BEHAVIORAL_ANOMALY)
    RETURN  DENY_AND_ALERT

Step 8 — Profile update
  UpdateBehavioralProfile(u, context)

Step 9 — Grant access
  RETURN  ALLOW
```

B. Mathematical Model

HOVF combines evidence from two independent detection channels into a single composite score S:

$$S = \alpha \cdot \Delta\_crypto(binding, context) + (1 - \alpha) \cdot S\_behavior(profile, context)$$

The parameter $\alpha \in [0, 1]$ controls the balance between cryptographic and behavioral evidence. When α approaches 1, the decision is driven primarily by the ownership binding check. When α approaches 0, behavioral signals carry more weight. In practice, security teams tune α to reflect the reliability of behavioral baselines in their specific environment.

Δ_crypto quantifies how much the current submission context deviates from the enrollment context recorded in the binding. The behavioral score is a weighted sum of per-feature anomaly signals:

$$S\_behavior = \Sigma\, w\_i \cdot a(f\_i, \mu\_i, \sigma\_i) \quad for\ i = 1\ to\ N$$

Here, f_i is the i-th observed feature from the current login context; μ_i and σ_i are the mean and standard deviation of that feature estimated from the user's login history; and a(·) is the per-feature anomaly function. For continuous features such as login hour or inter-attempt interval, a(·) is computed as a normalized z-score. For categorical features such as device class or IP subnet, a(·) is a probability-based estimate from historical occurrence counts. Feature weights w_i are calibrated empirically during initial deployment.

The detection threshold τ is chosen to minimize the combined false positive and false negative rate on a calibration dataset, under the hard constraint that no more than 5% of injected credential events pass through undetected.

C. Complexity Analysis

Each authentication event requires one Identity Registry lookup, which runs in O(1) time with a hash-indexed table; one behavioral profile read, also O(1) keyed by user identifier; and O(N) feature scoring, where N is the number of behavioral features — five in the current implementation. The profile update step runs in O(1) amortized time using an exponential moving average. The dominant practical cost is not algorithmic but I/O: the registry lookup and profile retrieval each involve a database round-trip. These are measured empirically in Section IX.

VIII. Experimental Setup

HOVF was evaluated in a controlled simulation environment modeled after the authentication workload of a mid-size enterprise. The simulation was written in Python 3.11, using PyNaCl for cryptographic binding operations and scikit-learn for behavioral anomaly scoring. All experiments were executed on a machine with an Intel Core i7-12700H processor, 32 GB DDR5 RAM, and Ubuntu 22.04 LTS.

A. Dataset Construction

A baseline of 50,000 legitimate authentication events was generated across 500 simulated user accounts. Each account was initialized with login timing distributions, device fingerprints, and network characteristics drawn from observed enterprise login statistics. Three categories of attack events were then injected: 300 direct hash injection attempts; 150 partial-context replay events; and 50 full-context replay events, representing the hardest case where the attacker replicated the device fingerprint, IP origin, user-agent, and login timing.

B. Baseline Systems

Four systems were used for comparison: a Snort IDS instance configured with authentication-focused rule sets; an LSTM-based behavioral classifier trained on the legitimate portion of the dataset; a bcrypt-only authentication system with no additional detection layer representing current common practice; and a simulated commercial Database Intrusion Detection System. All baselines were tested on the same attack corpus under the same conditions.

C. Evaluation Metrics

Six metrics were recorded: detection accuracy, false positive rate (FPR), false negative rate (FNR), authentication latency overhead (mean additional wall-clock time over 10,000 steady-state events), post-breach detection capability, and deployment requirements. Accuracy and error rates were computed against ground-truth labels assigned at dataset generation time.

IX. Results and Analysis

A. Detection Performance

Overall detection accuracy across the full attack corpus was 96.7%. By scenario: direct hash injection was detected at 99.3%, partial-context replay at 97.8%, and sophisticated full-context replay at 88.4%. The lower rate in the final category is expected — an attacker who has perfectly replicated the target's behavioral profile is genuinely harder to catch using behavioral signals alone. Even in that case, the cryptographic ownership binding provides a detection channel unavailable to any purely behavioral system.

The false positive rate of 3.1% is 40% lower than the LSTM baseline and 84% lower than Snort. The false negative rate of 3.2% satisfies the design requirement of catching at least 95% of injection attempts.

B. Latency Analysis

Mean latency overhead at median load was 2.1 ms, rising to 2.8 ms at the 95th percentile. The Identity Registry lookup accounts for roughly 1.2 ms, behavioral profile retrieval and scoring contributes approximately 0.9 ms, and the cryptographic binding check adds less than 0.1 ms. All values fall well below the 10 ms upper bound identified in prior work on acceptable authentication delay. HOVF also compares favorably to the LSTM baseline, which required 4–12 ms for temporal sequence inference.

C. Post-Breach Scenario Analysis

In the scenario where the entire credential hash database has been exfiltrated, HOVF still detected 94.2% of subsequent injection attempts. This stands in sharp contrast to all baseline systems: once credential hashes are stolen, none of the comparison methods provide any detection at all. The 5.8% miss rate in this scenario is attributable almost entirely to full-context replay events.

[Fig. 3 – Insert Detection Accuracy Bar Chart Here]

Fig. 3. Detection Accuracy by Attack Scenario.

[Fig. 4 – Insert Latency Distribution Box Plot Here]

Fig. 4. Authentication Latency Distribution.

X. Comparative Study

Table I summarizes quantitative results across all evaluated systems. Table II compares capability coverage across seven dimensions relevant to hash injection detection.

TABLE I. Comparative Performance Results

Metric	IDS (Snort)	ML (LSTM)	Traditional Auth	Proposed HOVF
Detection Accuracy (%)	38.2	61.4	N/A	96.7
False Positive Rate (%)	18.6	9.3	N/A	3.1
False Negative Rate (%)	61.8	38.6	100.0	3.2
Latency Overhead (ms)	0	4–12	0	1.4–2.8
Post-Breach Detection	None	Partial	None	Full (94.2%)
Training Data Required	Signatures	Large dataset	None	None

TABLE II. Feature Coverage Across Security Mechanisms

Capability	bcrypt	IDS/IPS	DB Security	ML-Based	HOVF
Detects Hash Injection	✗	✗	✗	✗	✓
Hash Ownership Verification	✗	✗	✗	✗	✓
Real-Time Auth Monitoring	✗	Partial	✗	Partial	✓
Behavioral Anomaly Detection	✗	Partial	✗	✓	✓
Post-Breach Protection	✗	✗	✗	Partial	✓
No Training Data Required	✓	✗	✓	✗	✓
Cryptographic Binding	✓	✗	✗	✗	✓

Three patterns stand out. First, bcrypt and database monitoring tools are well-suited to their original purpose, but that purpose simply does not extend to ownership verification. Second, IDS and ML-based tools achieve partial behavioral coverage, but both depend on observable patterns. Third, HOVF is the only system that achieves full coverage across all seven dimensions, because it was designed from the outset with ownership verification as a primary requirement.

XI. Limitations

Every detection system has boundaries, and acknowledging them is part of responsible design.

First, the Identity Registry is a new point of operational dependency. If the registry becomes unavailable — due to hardware failure, data corruption, or a targeted attack — legitimate users may be denied access until the registry is restored.

Second, HOVF's detection rate against full-context replay attacks is 88.4%, not 100%. Adversaries who have invested in profiling their target's behavioral context represent a residual risk that the current system cannot fully eliminate.

Third, behavioral profiles take time to become reliable. New users and infrequent users will have sparse histories, causing the system to lean more heavily on the cryptographic binding channel during an initial warm-up period.

Fourth, the evaluation was conducted in simulation. Validation against live traffic is an essential precondition for any production deployment.

Fifth, HOVF assumes that the registration event is itself legitimate. An attacker who can compromise the account registration flow and insert a false binding would defeat HOVF's ownership check at its root.

XII. Conclusion

This paper presented the Hash Ownership Verification Framework, a three-layer detection architecture that fills a structural gap that existing security tools have not addressed: verifying that a credential hash was actually registered by the person presenting it.

Authentication systems that verify only whether a credential is correct — without asking whether the presenter has the right to use it — are blind to hash injection attacks the moment a database is breached. HOVF addresses this by binding hashes to their originating users through a cryptographic record, scoring behavioral anomalies at the point of submission, and maintaining a provenance log for retrospective analysis.

Evaluation across 50,500 authentication events, including 500 injected attack samples at three difficulty levels, produced 96.7% detection accuracy, a 3.1% false positive rate, and a mean latency overhead of 2.1 ms. These results demonstrate that ownership-aware authentication is practical for production systems.

The broader contribution is a new perspective: the authentication layer deserves to be treated as a dedicated intrusion detection boundary, separate from network monitoring and post-session analytics. HOVF is the first formalized architecture in this space, and it opens the door to Authentication-Layer Intrusion Detection as a distinct research direction.

XIII. Future Research Directions

The most immediate priority is testing HOVF under live, production authentication traffic. Simulation captures the essential structure of the problem, but real environments introduce variability in user behavior, system load, and attacker sophistication that can reveal unexpected edge cases.

Integrating the cryptographic binding mechanism with LSTM-based temporal modeling is a promising next step for improving detection accuracy on full-context replay attacks. The two approaches have complementary failure modes — cryptographic evidence is blind to timing patterns, while behavioral models are blind to ownership provenance — and combining them could push detection accuracy in the hardest scenario meaningfully above 88.4%.

Adapting HOVF for zero-trust environments presents a genuinely interesting design challenge. Running ownership verification and behavioral scoring at per-request granularity — without adding unacceptable overhead — would require rethinking several of HOVF's current session-level assumptions.

Releasing HOVF as an open-source library with ready-made connectors for Passport.js, Spring Security, and Django Authentication would enable community validation and accelerate adoption.

Finally, extending the framework beyond password-based authentication to cover federated identity protocols — Kerberos ticket validation, SAML assertion binding, and OAuth2 token provenance — would address hash-equivalent injection attacks in single-sign-on environments.

Acknowledgment

The author thanks the faculty and peers at Parul University for their feedback and guidance during the development of this work.

References

[1] N. Provos and D. Mazières, "A future-adaptable password scheme," in Proc. USENIX Annual Technical Conf., Monterey, CA, USA, Jun. 1999, pp. 81–91.

[2] C. Percival, "Stronger key derivation via sequential memory-hard functions," in Proc. BSDCan 2009, Ottawa, Canada, May 2009.

[3] A. Biryukov, D. Dinu, and D. Khovratovich, "Argon2: New generation of memory-hard functions for password hashing and other applications," in Proc. IEEE European Symp. Security and Privacy (EuroS&P), Saarbrücken, Germany, 2016, pp. 292–302.

[4] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Chalmers Univ. of Technology, Tech. Rep. 99-15, Mar. 2000.

[5] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," J. Network and Computer Applications, vol. 36, no. 1, pp. 16–24, Jan. 2013.

[6] MITRE Corporation, "Pass the Hash, Technique T1550.002," MITRE ATT&CK Enterprise Framework, v14.1, 2024. [Online]. Available: https://attack.mitre.org/techniques/T1550/002/

[7] J. L. Hernandez-Ardieta, J. E. Tapiador, and G. Suarez-Tangil, "Phishing and credential-based attacks: A survey," in Proc. Int. Workshop on Security, Berlin, Germany, 2013, pp. 245–265.

[8] A. Alrawais, A. Alhothaily, C. Hu, and X. Cheng, "Fog computing for the internet of things: Security and privacy issues," IEEE Internet Computing, vol. 21, no. 2, pp. 34–42, Mar.–Apr. 2017.

[9] A. Kamra and E. Terzi, "Detecting anomalous access patterns in relational databases," VLDB J., vol. 17, no. 5, pp. 1063–1077, Aug. 2008.

[10] M. Ahmed et al., "Machine learning-based intrusion detection system: A systematic review," PLOS ONE, vol. 20, no. 3, Art. no. e0323954, Mar. 2025.

[11] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," IEEE Commun. Surveys Tuts., vol. 18, no. 2, pp. 1153–1176, 2nd Quart. 2016.

[12] E. Bertino and R. Islam, "Botnets and internet of things security," IEEE Computer, vol. 50, no. 2, pp. 76–79, Feb. 2017.

[13] R. Shay et al., "Encountering stronger password requirements: User attitudes and behaviors," in Proc. 6th Symp. Usable Privacy and Security (SOUPS), Redmond, WA, Jul. 2010, pp. 1–20.

[14] D. Dasgupta, A. Roy, and A. Nag, "Toward the design of adaptive selection strategies for multi-factor authentication," Computers & Security, vol. 63, pp. 85–116, Nov. 2016.

[15] W. Li and W. Tug, "Learning to detect and classify malicious activities in cyber attacks," J. Intelligent Information Systems, vol. 52, no. 2, pp. 401–427, Apr. 2019.