Guvenkaya® The Bedrock of Security

Virto Network

Pallet Pass Security Review

Lead Security Engineer: Timur Guvenkaya

Date of Engagement: 9th July 2025 - 18th July 2025

Visit: www.guvenkaya.co

Contents

About Us

Guvenkaya is a security research firm specializing in Rust security, Web3 security of Non-EVM protocols, and Web2 security. With our expertise, we provide both security auditing services and custom security solutions

About Virto

Virto Network is next generation payment infrastructure for impactful communities.

Audit Results

Guvenkaya conducted a security assessment of the Virto network Substrate **Pallet Pass** from 9th July 2025 to 18th July 2025. During this engagement, a total of **8 findings** were reported. 1 of the findings was high, 4 medium and the remaining were low severity. Major issues are not fixed by the Virto Network team yet

Project Scope

Files	Link
Pallet Pass	https://github.com/virto-network/frame-contrib/tree/d78e5e2236b8f8a8c5f07f7fef319729bf959249/pallets/pass
Pass Authenticators	https://github.com/virto-network/pass-authenticators/tree/344a4363b0a887810932d625c63dff896c1f094a

Out of Scope

The audit will include reviewing the code for security vulnerabilities. The audit does not include a review of the tests and dependencies.

Timeline

Start of the audit	Draft report
9th July 2025	21st July 2025

Methodology

- RESEARCH INTO PROJECT ARCHITECTURE
- PREPARING ATTACK VECTORS
- SETTING UP AN ENVIRONMENT
- MANUAL CODE REVIEW OF THE CODE
- ASSESSMENT OF RUST SECURITY ISSUES
- ASSESSMENT OF SUBSTRATE SECURITY ISSUES
- ASSESSMENT OF ARITHMETIC ISSUES
- BUSINESS LOGIC VULNERABILITY ASSESSMENT
- BEST PRACTICES AND CODE QUALITY
- CHECKING FOR CODE REFACTORING/SIMPLIFICATION POSSIBILITIES
- ARCHITECTURE IMPROVEMENT SUGGESTIONS
- PREPARING POCS AND/OR TESTS FOR EACH CRITICAL/HIGH ISSUES

Severity Breakdown

01. Likelihood Ratings

Likely: The vulnerability is easily discoverable and not overly complex to exploit.
Possible: The vulnerability presents some challenges either in discovery or in the complexity of the attack.
Rare: The vulnerability is either very difcult to discover or complex to exploit, or both.
This matrix provides a nuanced view, taking into account both the ease of discovering a vulnerability and the complexity involved in exploiting it.

02. Impact

Severe: Exploitation could result in critical loss or compromise, such as full system control, substantial financial loss, or severe reputational damage.
Moderate: Exploitation may lead to limited data loss, partial compromise, moderate financial impact, or noticeable degradation of services.
Negligible: Exploitation has minimal impact, such as minor data exposure without significant consequences or slight inconvenience without substantial disruption.

03. Severity Ratings

Critical: Assigned to vulnerabilities with severe impact and a likely likelihood of exploitation.
High: For vulnerabilities with either severe impact but only a possible likelihood, or moderate impact with a likely likelihood.
Medium: Used for vulnerabilities with severe impact but a rare likelihood, moderate impact with a possible likelihood, or negligible impact with a likely likelihood.
Low: For vulnerabilities with moderate impact and rare likelihood, or negligible impact with a possible likelihood.
Informational: The lowest severity rating, typically for vulnerabilities with negligible impact and a rare likelihood of exploitation.

CRITICAL HIGH MEDIUM Low Informational

Likelihood Matrix:

Attack Complexity \ Discovery Ease	Obvious	Concealed	Hidden
Complex	Possible	Rare	Rare
Moderate	Likely	Possible	Rare
Straightforward	Likely	Possible	Possible

Likelihood/Impact Matrix:

Likelihood \ Impact	Severe	Moderate	Negligible
Likely	**CRITICAL**	**HIGH**	**MEDIUM**
Possible	**HIGH**	**MEDIUM**	Low
Rare	**MEDIUM**	Low	Informational

Findings Summary

01. Remediation Complexity: This measures how difcult it is to fx the vulnerability once it has been identifed.

Simple: Patches or fixes are readily available and easily implemented.
Moderate: Requires some time and resources to remediate, but well within the capabilities of most organizations.
Difficult: Remediation requires significant resources, specialized skills, or substantial changes to systems or architecture.

02. Status: This measures how difcult it is to fx the vulnerability once it has been identifed.

Not Fixed: Indicates that the vulnerability has been identifed but no remedial action has been taken yet. This status is crucial for newly discovered vulnerabilities or those awaiting prioritization.
Fixed: This status is applied when the vulnerability has been successfully remediated. It implies that appropriate measures (like patching, confguration changes, or architectural modifcations) have been implemented to resolve the issue.
Acknowledged: This status is used for vulnerabilities that have been recognized, but for various reasons (such as risk acceptance, cost, or other business decisions), have not been fxed. It indicates that the risk posed by the vulnerability is known and has been consciously accepted.
Scheduled: This status indicates that the vulnerability has been acknowledged and a plan is in place to fix it in the future. It signifies that while remediation hasn't yet occurred, the issue has been prioritized and is part of the planned development roadmap.

Finding	Impact	Likelihood	Severity	Remediation Complexity	Remediation Status
GUV-1: DoS of The Main Functionality Through Session Key Hijacking	Moderate	Likely	**HIGH**	Simple	Not Fixed
GUV-2: Phishing and Cross-Site Credential Reuse Attacks Through Missing Origin & rpId Validation	Severe	Rare	**MEDIUM**	Simple	Not Fixed
GUV-3: Arbitrary Device Registration Through Missing Validation in Device Attestation	Severe	Rare	**MEDIUM**	Moderate	Not Fixed
GUV-4: Potential DoS and Resource Exhaustion Through Underestimated Benchmarks and Fixed Weights	Moderate	Possible	**MEDIUM**	Moderate	Not Fixed
GUV-5: Potential Out-of-Memory (OOM) and Degraded Performance Due to Inefficient Parsing	Moderate	Possible	**MEDIUM**	Moderate	Not Fixed
GUV-6: Silent or Unauthorized Authentication Through Missing Authenticator Flags Validation	Negligible	Possible	Low	Simple	Not Fixed
GUV-7: Possible Storage Bloating Due To Unbounded Devices and Session Keys Per User	Negligible	Possible	Low	Simple	Not Fixed
GUV-8: Authentication Through Incorrect Credential Type	Negligible	Possible	Low	Simple	Not Fixed

Findings Details

GUV-1 DoS of The Main Functionality Through Session Key Hijacking - High

In Pallet Pass, session keys are publicly exposed through the SessionCreated event (emitted in add_session_key with the raw session_key value) and are queryable via the SessionKeys storage map. An attacker can monitor events or query storage to obtain existing session keys from other users. Due to insufficient checks in **add_session_key**, the attacker can then call this extrinsic with the stolen key for their own pass account. This triggers **try_remove_session_key**, which removes the key from the original owner's storage (decrementing their consideration count and emitting SessionRemoved), and re-assigns it to the attacker with a new expiration. The uniqueness check !frame_system::Pallet::<T>::account_exists(session_key) only prevents collisions with persistent system accounts, not with existing ephemeral session keys in the pallet's storage. As a result, attackers can hijack keys before their natural expiry, rendering the session key functionality unreliable.

add_session_key:pallet_pass/src/lib.rs

```
    ConsiderationHandler::<
            T::AccountId,
            SessionKeyConsiderations<T, I>,
            T::SessionKeyConsideration,T::AccountId,>::increment(address)?;

        Self::try_remove_session_key(session_key)?;

         let until = duration
            .unwrap_or(T::MaxSessionDuration::get())
            .min(T::MaxSessionDuration::get());
         SessionKeys::<T, I>::insert(session_key.clone(), (address.clone(), until));
        Self::schedule_next_removal(session_key, duration)?;
         ...
       }
```

try_remove_session_key:pallet_pass/src/lib.rs

```
    if let Some(address) = &Self::pass_account_from_session_key(session_key) {
              ConsiderationHandler::<
                T::AccountId,
                SessionKeyConsiderations<T, I>,
                T::SessionKeyConsideration,
                T::AccountId,
              >::decrement(address)?;

             SessionKeys::<T, I>::remove(session_key);

              Self::deposit_event(Event::<T, I>::SessionRemoved {
                session_key: session_key.clone(),
              })
            ...
          }
```

Impact:

- **Denial-of-Service (DoS) of main functionality**: Attackers can continuously hijack all active session keys (e.g., by monitoring events and calling add_session_key with short durations like 1 block), making sessions unusable for legitimate users. Victims must re-authenticate via passkeys/devices, degrading UX and potentially overwhelming the system with re-auth requests.
- **Griefing Vector**: As outlined, attackers can target specific users (e.g., high-value accounts) or broadly disrupt the session feature, rendering it "useless" without constant re-creation. With short durations, attackers avoid risks to their own accounts while forcing expirations.
- **Economic Waste**: Victims lose active sessions prematurely, incurring re-auth costs (e.g., time/gas for new keys).
- **Reputation Damage**: Undermines trust in session keys as a reliable "short-lived" auth mechanism.

POC: **GUV-1**

Recommendation

- **Prevent Key Reuse**: Add a check in add_session_key to ensure the key isn't already in SessionKeys: ensure!(!SessionKeys::<T, I>::contains_key(session_key), Error::<T, I>::SessionKeyInUse);.
- **Obscure Keys in Events**: Emit a hash instead of raw session_key in SessionCreated (e.g., session_key_hash: T::Hashing::hash(&session_key.encode())).

Remediation - Not Fixed

The Virto Network team has not fixed the issue yet.

GUV-2 Phishing and Cross-Site Credential Reuse Attacks Through Missing Origin & rpId Validation - Medium

Pallet Pass's WebAuthn verification lacks checks for the origin (client domain) in client_data JSON and rpIdHash in authenticator_data, allowing credentials generated for one site to be reused on another. Per WebAuthn specs, these must match the expected Relying Party (RP) to prevent cross-origin misuse. The verifier only does signature checks, ignoring these fields.

Impact:

- **Cross-Site Credential Reuse**: Allows auth from unintended domains, bypassing site isolation.
- **Compliance Failures**: Non-spec-compliant; may fail interop with standard WebAuthn clients, causing auth denials or bugs.

POC: GUV-2

Recommendation

To ensure the passkey is genuine and from a verified source, you should:

- **Parse and Validate Client Data**: Ensure origin matches expected RP domain.
- **Validate rpIdHash**: Parse authenticator_data; verify rpIdHash == SHA256(expected rpId).

Remediation - Not Fixed

The Virto Network team has not fixed the issue yet.

GUV-3 Arbitrary Device Registration Through Missing Validation in Device Attestation - Medium

The is_valid method in the DeviceAttestation implementation always returns true, bypassing critical verification steps required for secure device registration in a WebAuthn-based system. This method is part of the DeviceChallengeResponse trait and is invoked during the Authenticator::verify_device process, which handles new device attestations for user accounts. Normally, attestation verification should follow the WebAuthn Level 3 specification, including parsing the attestation object (CBOR-encoded with fields like fmt, attStmt, and authData), validating signatures against certificate chains, checking RP ID and origin bindings, and ensuring the authenticator's authenticity via AAGUID and FIDO Metadata. However, the current implementation skips all these checks.

is_valid:authenticators/webauthn/src/runtime/attestation.rs

```
impl<Cx> DeviceChallengeResponse<Cx> for Attestation<Cx>
      where
        Cx: Parameter + Copy + 'static,
      {
        // TODO: @pandres95, considering that DeviceChallengeResponse is used for
creating a new
        // authentication device, webauth_verify wouldn't work here. We need to
implement a new
        // verification method exclusively for credential creation.
        fn is_valid(&self) -> bool {
          true
        }

        fn used_challenge(&self) -> (Cx, Challenge) {
          (self.meta.context, self.challenge())
        }
```

Impact:

- **Arbitrary Device Registration**: Attackers can register new accounts with malicious public keys, enabling front-running of legitimate users. For existing accounts, combined with phishing, it allows adding backdoor devices, leading to permanent unauthorized access and actions.
- **Phishing Amplification**: Weak attestation accepts cross-origin or fake attestations, making it easier to phish users into registering cloned/malicious devices.
- **Reputation and Compliance Risks**: Violates WebAuthn standards, exposing users to cloning/replay attacks.

Recommendation

- **Implement Full Attestation Verification**: Replace is_valid = true with spec-compliant logic in DeviceChallengeResponse per this specification: https://www.w3.org/TR/webauthn-3/#sctn-registering-a-new-credential
- **Enhance Challenge Binding**: Use non-empty xtc in verify_device to prevent precomputation.
- **Add Uniqueness and Limits**: Enforce unique devices per account; cap devices per account (e.g., via BoundedVec).

Remediation - Not Fixed

The Virto Network team has not fixed the issue yet.

GUV-4 Potential DoS and Resource Exhaustion Through Underestimated Benchmarks and Fixed Weights - Medium

The benchmarks and weights in the pallet are not correctly configured to handle the variability introduced by WebAuthn-specific paths, particularly in functions like register and authenticate that involve parsing variable-sized fields such as client_data (a Vec<u8> containing JSON). The benchmarks rely on fixed-size, small inputs generated via BenchmarkHelper for SubstrateKey (sr25519-based) attestations, which are typically 100-200 bytes and do not exercise the WebAuthn variant. As a result, the weights are fixed and do not scale parametrically with input size.

Impact:

- **Denial of Service (DoS) Vulnerability**: Attackers can submit extrinsics with large client_data which are charged the same low fixed weight but incur high actual execution time and memory. This can exceed block production limits, cause node crashes, or fill blocks with resource-intensive transactions, disrupting network consensus.
- **Economic Exploitation**: Undercharged weights allow spamming without proportional gas fees, enabling cheap DoS attacks that waste validator resources (CPU, memory) while refunding minimal costs to the attacker.
- **Degraded Performance**: In production, unaccounted WebAuthn paths lead to slower block times or validation failures under load, especially if extensions inflate data sizes. This erodes pallet reliability for WebAuthn users.
- **Incomplete Security Model**: Fixed weights ignore adversarial inputs, potentially allowing griefing where large extrinsics pass validation but strain nodes, leading to operational costs and reduced scalability.

Recommendation

- **Update Benchmarks for Variability**: Revise benchmarks to include WebAuthn-specific paths with parametric inputs.
- **Implement Parametric Weights**: Calculate weights as base_weight + per_byte_weight * client_data_len, incorporating measured costs for decoding, parsing, and verification.
- **Add Input Bounds**: Enforce limits on Vec<u8>fields (e.g., via BoundedVec<u8, ConstU32<4096>>) to cap maximum sizes and prevent extreme cases.
- **Separate Paths in Benchmarks**: Benchmark SubstrateKey and WebAuthn variants independently to ensure accurate weights for each authenticator type.

Remediation - Not Fixed

The Virto Network team has not fixed the issue yet.

GUV-5 Potential Out-of-Memory (OOM) and Degraded Performance Due to Inefficient Parsing - Medium

The find_challenge_from_client_data function (and related get_from_json_then_map) employs a simplistic, non-robust parsing approach for extracting keys like "challenge" from clientDataJSON (a Vec<u8>). It converts the entire input to a string via String::from_utf8 then performs O(n) operations like split and linear iteration with find_map scanning substrings for matches. This allocates memory proportional to the input size (O(n)) and iterates in O(n) time worst-case, making it vulnerable to large or adversarial inputs.

find_challenge_from_client_data:src/runtime/helpers.rs

```
pub fn find_challenge_from_client_data(client_data: Vec<u8>) -> Option<Challenge> {
        get_from_json_then_map(client_data, "challenge", |challenge| {
            base64::decode_engine(challenge.as_bytes(),
&BASE64_URL_SAFE_NO_PAD).ok()
          })
        }
        ...
        // get_from_json_then_map function declaration
       let json = String::from_utf8(json).ok()?;

       let value = json
         .split(",")
         .find_map(|kv| kv.contains(key).then_some(kv.split_once(":")?.1))
         .map(|v| v.trim_matches(|c: char| c.eq(&' ') || c.eq(&'"')))
         .and_then(map)?;
          ...
```

Impact:

- **Out-of-Memory (OOM) Errors**: Large inputs during String::from_utf8 or iteration can cause allocation failures, crashing the node or extrinsic execution, leading to failed transactions or consensus disruptions.
- **Degraded Performance and DoS**: O(n) time for huge JSON causes excessive CPU usage, slowing block validation/production and enabling cheap DoS by spamming large extrinsics that tie up resources without proportional weight charges.
- **Resource Exhaustion**: In validators, sustained large inputs consume memory/CPU, reducing throughput, increasing latency, or evicting other data, impacting overall network health and scalability.

Recommendation

- **Use Efficient Parsing**: Replace naive string operations with a proper JSON parser that validates structure, handles duplicates/escapes, and extracts keys without full buffering.
- **Implement Incremental Parsing**: Scan for keys without converting to a full string (e.g., byte-level searching), reducing memory to O(1) where possible.
- **Enforce Size Limits**: Bound client_data (e.g., BoundedVec<u8, ConstU32<8192>>) to a reasonable max based on WebAuthn specs, rejecting larger inputs early.
- **Add Early Validation**: Check UTF-8 validity and size before parsing; fail fast on anomalies to minimize resource use.

Remediation - Not Fixed

The Virto Network team has not fixed the issue yet.

GUV-6 Silent or Unauthorized Authentication Through Missing Authenticator Flags Validation - Low

No parsing/validation of authenticator_data flags: UP (User Present), UV (User Verified), BE (Backup Eligible), BS (Backup State). Specs require UP (always), UV (if needed), and BE/BS consistency to ensure user interaction and detect backups/clones. Verifier ignores these, accepting silent auth.

Impact:

- **Silent Auth Without Consent**: No UP/UV allows auth without touch/biometrics, enabling scripted attacks (e.g., malware on device).
- **Backup/Cloning Undetected**: No BE/BS checks fails to handle multi-device risks (e.g., insecure backups).
- **Security Downgrade**: Bypasses user verification, weakening passkey protection against physical access attacks.

POC: GUV-6

Recommendation

- Follow flag verification per this specification: https://www.w3.org/TR/webauthn-3/#sctn-verifying-assertion

Remediation - Not Fixed

The Virto Network team has not fixed the issue yet.

GUV-7 Possible Storage Bloating Due To Unbounded Devices and Session Keys Per User - Low

No bounds on devices or session keys per user allow unlimited additions, leading to storage bloat. Costs exist (deposits via linear StoragePrice), but first device/session are free (FirstItemsFree), and privileged origins (Root/Communities) skip entirely (SkipConsideration). Attackers can loop registrations (pay per cycle) + free additions, orphaning data. Alternatively, a single account can add unlimited number of keys or devices.

Impact:

- **Storage Bloat**: Unlimited devices and sessions per account can fill storage maps, increasing node costs and degrading performance.

Recommendation

- Introduce configurable limits, e.g., MaxDevicesPerAccount u32 = 5; MaxSessionsPerAccount u32 = 5. Check in extrinsics.

Remediation - Not Fixed

The Virto Network team has not fixed the issue yet.

GUV-8 Authentication Through Incorrect Credential Type - Low

The verifier doesn't check the type field in client_data JSON (must be "webauthn.get" for assertions per specs). It hashes client_data without parsing, accepting any type (e.g., "webauthn.create" for registrations misused as assertions).

Impact:

- **Type Confusion Attacks**: Allows "create" credentials (for registration) to auth as assertions, potentially creating invalid sessions or bypassing checks.
- **Compliance Issues**: Violates specs; may cause interop failures with clients enforcing types.

POC: **GUV-8**

Recommendation

- **Parse Client Data Type**: Deserialize client_data; ensure type == "webauthn.get".

Remediation - Not Fixed

The Virto Network team has not fixed the issue yet.