



IMMUNEFI

AUDIT COMPETITION



Immunefi /



ens

DATE

September, 14, 2026

TOP AUDITORS

br0nz3p1ck4x3
crypt0n1337
bugtester

REPORT BY

Immunefi

01

Overview

02

Scope of assets

03

Summary

04

Findings



Contents

About Immunefi	3
Immunefi Audit Competitions	4
Terminology	5
Executive Summary	7
Findings — Websites & Apps	13

About Immunefi

Immunefi is the security platform for the onchain economy, and 93% of crypto's critical vulnerability disclosures now flow through it. Founded in December 2020, it protects more than \$180 billion in assets across 650-plus protocols, including MakerDAO, Optimism, Filecoin, LayerZero, Chainlink, Arbitrum, Lido, and Polygon. It has helped prevent more than \$25 billion in hacks and paid more than \$140 million in bounties, more than the rest of the market combined. Its unified security platform combines bug bounties, audit competitions, AI-assisted security, and coordinated vulnerability disclosure, powered by more than 85,000 researchers and a proprietary exploit dataset built since 2020. The thesis: crowdsourced security, supercharged with AI, is the only defense that scales to meet AI-driven attacks.

Learn how you can benefit too at immunefi.com.

Immunefi Audit Competitions

Immunefi has one of the largest communities of security researchers, with over 85,000 members. This community is mobilised through Audit Competitions.

In audit competitions, a prize pool is shared, and the codebase is opened to Immunefi's researcher network for a fixed competition window. Audit Competitions are quantitatively superior to private audits, and these findings are supported by data from 1,178 public audit reports from Trail of Bits, Zellic, and OpenZeppelin.

1. Improving the cost efficiency of bug finding: On average, a critical or high vulnerability costs about \$66,000 in a private audit. This cost is reduced by about 10x in an audit competition, with an average of \$6,548.

2. Improving coverage, scrutiny and findings: An audit assigns 2 to 5 people to your code, while recent Immunefi competitions drew 952, 943 and 1,415 submissions. Hundreds of researchers cover more of the codebase and find more bugs.

The median Immunefi audit competition finds 2 serious findings, compared to 0 for a Tier-1 private audit. The same applies for critical bugs: more than 1 in 2 Immunefi audit competitions find those, compared to only 1 in 6 Tier-1 audits. All in all, Immunefi audit competitions average 6.2 critical or high findings, and Tier-1 audits only about 1.5.

Comparative Analysis between Private Audits and Audit Competitions

	Tier-1 private audit	Immunefi audit competition
Engagements measured	1,178	58
Median serious findings	0	2
Average per engagement	1.5	6.2
Surfaced at least one critical	1 in 6	more than 1 in 2
Cost per critical-or-high	~\$66,000	\$6,548

The best single report across all 1,178 found 21 critical-and-high issues. One Immunefi competition on a \$100K pool found 43. Recent competitions on Immunefi include Firedancer V1, Base Azul, and the Ethereum Protocol Attackathon. Every finding, severity and payout is published at immunefi.com/audit-competition.

Terminology

For the purpose of this assessment we adopt the Immunefi severity classification defined by the ENS Audit Competition programme. Severity is determined by impact: each finding is mapped to a named impact from the programme's Impacts in Scope table, and the severity of that impact is the severity of the finding.

A step-by-step explanation of the bug was required as the proof of concept for every submission; runnable code was not required, although Immunefi could request one for unclear reports or to resolve disputes. Insight is a tier below Low, used for observations with no demonstrated in-scope impact; it cannot be self-selected at submission and is designated by Immunefi or the project team.

The band below is the one that applies to the assets in scope of this competition.

Insights aren't included in the table, since they lack severity/impact.

Websites & Applications

Severity	In-scope impacts
Critical	· Unauthorized takeover of a user's smart account (Hybrid Custody Account) or authenticated session, letting an attacker transfer the victim's names or funds or act as them, such as: – adding, replacing, or escalating an account owner or co-owner – extending or widening a session key's authority beyond its intended scope or lifetime – stealing an authenticated session or bearer token – (Exploiting the 7-day session-key design as described in the Known Issues is not eligible; a weakness that grants authority beyond its stated lifetime or permissions is in scope.) · Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: – resolving or rendering an ENS name to an attacker-controlled address – showing the wrong recipient or amount on a transaction the user then approves · Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: – a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker – redirecting the recipient of a name transfer – leaving a name permanently inaccessible after the flow · Execute arbitrary system commands · Retrieve sensitive data/files from a running server, such as: – /etc/shadow – database passwords – blockchain keys (this does not include non-sensitive environment variables, open source code, or usernames) · Taking down the application/website · Taking state-modifying authenticated actions (with or without blockchain state interaction) on behalf of other users without any interaction by that user, such as: – Changing registration information – Notification/settings changes · Direct theft of user funds

Severity	In-scope impacts
	- Malicious interactions with an already-connected wallet, such as: - Modifying transaction arguments or parameters - Substituting contract addresses - Submitting malicious transactions - Signing or submitting a transaction against the wrong chain (for example when the request's target chain is not validated) - Inducing a blanket token approval (for example setApprovalForAll on the registry or NameWrapper) to an attacker-controlled address - Direct theft of user NFTs - Injection of malicious HTML or XSS through metadata
High	- Injecting/modifying the static content on the target application without JavaScript (persistent), such as: - HTML injection without JavaScript - Replacing existing text with arbitrary text - Arbitrary file uploads, etc - Changing sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: - Email - Improperly disclosing confidential user information, such as: - Email address
Medium	- Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: - Enabling/disabling notifications - Injecting/modifying the static content on the target application without JavaScript (reflected), such as: - Reflected HTML injection - Loading external site data - Redirecting users to malicious websites (open redirect)
Low	- Changing details of other users (including modifying browser local storage) without already-connected wallet interaction and with significant user interaction, such as: - Iframing leading to modifying the backend/browser state (must demonstrate impact with PoC) - Taking over broken or expired outgoing links, such as: - Social media handles, etc. - Temporarily disabling user to access target site, such as: - Locking up the victim from login - Cookie bombing, etc.

Executive Summary

Over the course of 27 days in August 18 – September 14, 2026, ENS ran an audit competition on Immunefi covering five assets across Websites & Apps. 538 reports were confirmed, resolving into **74 unique findings**: 23 Critical, 15 High, 22 Medium and 14 Low, plus 14 Insights.

Summary

Name	ENS
Competition Type	Audit Competition
Competition Period	August 18 – September 14, 2026
Reward Pool	\$70,000 USD
Languages	TypeScript
Scope	137,845 lines of code across five assets
Repository	immunefi-team/audit-comp-ens
Commit	1c9b47f18fcddd2e864dfe385c4171061c9811ae
PoC Requirement	Step-by-step explanation required at every severity; runnable code not required
Confirmed Reports	538
Unique Findings	74

Assets in Scope

Asset	Files	Lines of code
https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager Manager app	686	64,268
https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal Explorer app	574	57,681
https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/workers	93	7,629
https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/packages/transaction-manager	32	6,507
https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/packages/smart-account	16	1,760
Total	1,401	137,845

Issues Found

Severity	Findings
Critical	23
High	15
Medium	22
Low	14
Total	74
Insight	14

Findings index – Websites & Apps

Finding ID	Severity	Report Name
IMM-WA-CRITICAL-89253	Critical	Resolver roles removal revokes only the first grouped name, leaving the removed account's <code>ROLE_SET_ADDR</code> on other names
IMM-WA-CRITICAL-89264	Critical	Transfer flow never revokes the seller's root resolver roles, letting former owners rewrite a sold name's records
IMM-WA-CRITICAL-89270	Critical	Profile route resolves records from the raw URL name and ownership from the normalized name, showing an attacker's address
IMM-WA-CRITICAL-89278	Critical	Transfer route authorizes and labels raw <code>ALICE.eth</code> but normalizes the token lookup, transferring the victim's canonical <code>alice.eth</code>
IMM-WA-CRITICAL-89314	Critical	Migration keeps a gifted V1 name's stale resolver and silently grants its previous manager <code>ROLE_SET_RESOLVER</code> on V2
IMM-WA-CRITICAL-89395	Critical	<code>resolveAddressOrName</code> falls back to the v1 registry controller, redirecting ENS name transfers away from the registrant
IMM-WA-CRITICAL-89431	Critical	Retained successful actor under the shared <code>tx-revoke-roles</code> ID marks a later role removal done without revoking
IMM-WA-CRITICAL-89456	Critical	Renewal route reads expiry from the normalized name but signs the raw URL label, renewing a lookalike name
IMM-WA-CRITICAL-89601	Critical	Missing ENSIP-15 normalization in Manager registration makes users buy a different, permanently unresolvable name
IMM-WA-CRITICAL-90094	Critical	Explorer renewal displays the raw name but signs <code>renew</code> for its normalized twin, extending another owner's registration
IMM-WA-CRITICAL-90190	Critical	Bulk renewal displays the raw indexer name but renews the normalized label, extending a registration the user does not own
IMM-WA-CRITICAL-90379	Critical	<code>ResolverRolesSidebar</code> keys its permission draft to the row index, granting a shifted account an unselected resolver role
IMM-WA-CRITICAL-90544	Critical	<code>setPrimaryNameWithHca</code> normalizes the selected name after confirmation, making Manager render an attacker-controlled Optimism recipient
IMM-WA-CRITICAL-90725	Critical	Deleting a literal <code>[labelhash]</code> subname unregisters the victim subname it encodes, letting a registrar take it over
IMM-WA-CRITICAL-91014	Critical	SDK patch labels same-chain scoped HCA session signatures as owner mode, letting a session key drain HCA tokens

Finding ID	Severity	Report Name
IMM-WA-CRITICAL-91295	Critical	Case-folded address grouping lets an IOST-only delegate's value be shown and copied as the SOL receiving address
IMM-WA-CRITICAL-91335	Critical	Registry roles route applies Registry role labels to a PermissionedResolver, so a "Registrar" grant confers <code>ROLE_SET_ADDR</code>
IMM-WA-CRITICAL-91348	Critical	Unbound SIWE nonce lets an observer redeem a public Safe signature first and steal a seven-day JWT
IMM-WA-CRITICAL-91982	Critical	Migration admits locked names with a custom resolver and <code>CANNOT_SET_RESOLVER</code> burned, leaving their records permanently unwritable
IMM-WA-CRITICAL-92640	Critical	<code>useCanEditRecords</code> authorizes record edits by resolver role, letting a former owner edit a sold name in-app
IMM-WA-CRITICAL-92963	Critical	Profile editor diffs one name's records against another's, turning a Description-only save into a hidden attacker <code>setAddr</code>
IMM-WA-CRITICAL-93026	Critical	Default transfer plan detaches the registry, bricking third-party subnames and letting the new holder re-mint them
IMM-WA-CRITICAL-93081	Critical	Resolver roles sidebar turns an unresolved role scope into the root sentinel, widening a one-name grant to every name
IMM-WA-HIGH-89328	High	Manager checkout total and signed funding permit come from separate unreconciled quotes, so the chain charges more than displayed
IMM-WA-HIGH-89329	High	Uncapped funding permit lets an inflated orchestrator quote pull the owner's full USDC balance on owner-signed HCA actions
IMM-WA-HIGH-89462	High	HCA registration budget omits the first-registration resolver deployment, leaving the reveal underfunded and unrecoverable
IMM-WA-HIGH-90749	High	Resolver-role editor reparses a flat dotted label as a hierarchical name, granting roles on a different name
IMM-WA-HIGH-91441	High	Unverified resolver events let any contract inject persistent text into another name's Explorer history
IMM-WA-HIGH-92196	High	Portal keys pending address-record edits by display symbol, so deleting an alias row clears the canonical Base record
IMM-WA-HIGH-92413	High	Unpinned <code>settlementLayers</code> in the Warp send lets a substituted route session-sign an origin transfer that drains the HCA
IMM-WA-HIGH-92461	High	Registration actor survives a name change, so the Explorer charges an earlier name's price under the new name's total
IMM-WA-HIGH-92540	High	Unscoped <code>useRoleHistory</code> query lets any registry owner forge entries in another name's role history
IMM-WA-HIGH-92665	High	Orchestrator-returned HCA registration is not checked against the requested calldata, installing an attacker-controlled subregistry
IMM-WA-HIGH-92779	High	Owner-signed HCA intents carry no execution verification, so the wallet signs whatever executions the orchestrator returns
IMM-WA-HIGH-92833	High	Manager's third-party V1 renewal leaves a wrapped name's <code>NameWrapper</code> expiry stale, letting a stranger brick it
IMM-WA-HIGH-93029	High	Manager writes L2 address records to mainnet coin types the Explorer never reads, so L2 rows show the default address
IMM-WA-HIGH-93033	High	Address history page trusts unconsented registry ownership, letting anyone plant attacker-authored rows and a forged sender

Finding ID	Severity	Report Name
IMM-WA-HIGH-93046	High	Unchecked <code>register</code> subregistry lets an HCA session signer retain control of the owner's subname resolution
IMM-WA-MED-89249	Medium	Manager migration grants <code>ROLE_SET_RESOLVER</code> to a stale cached V1 manager, letting a former seller hijack the resolver
IMM-WA-MED-89254	Medium	Channel-status writes match on <code>target</code> alone, letting an attacker's pending row disable another user's email notifications
IMM-WA-MED-89627	Medium	Transfer plan never revokes third-party role grants, so a prior grantee can reappoint the transferred name
IMM-WA-MED-90446	Medium	Uncapped push channels and an early-return retry let one user permanently suppress other users' expiry reminders
IMM-WA-MED-91127	Medium	Uncapped favorites overflow the unchunked expiry notification insert, failing shared batches for unrelated users
IMM-WA-MED-91314	Medium	V1 name list inherits the ensjs <code>resolvedAddress</code> relation, listing names that merely point at a user as owned
IMM-WA-MED-91689	Medium	Unnormalized child routes and a spoofable resolver check let an attacker contract become a victim's transaction target
IMM-WA-MED-91913	Medium	Manager Upgrade skips the locked-resolver safety check for the live Sepolia public resolver, wiping a name's records
IMM-WA-MED-92466	Medium	Uncapped <code>description</code> record escaped twice into meta tags lets one name crash the Explorer Worker isolate
IMM-WA-MED-92517	Medium	Pending-change transformer emits a same-key deletion after its replacement, reviving a revoked delegate's fallback address
IMM-WA-MED-92542	Medium	Roles editor's Remove user action revokes undisplayed <code>ROLE_CAN_TRANSFER_ADMIN</code> , freezing the name for its remaining term
IMM-WA-MED-92608	Medium	<code>buildTokenData</code> prices a failed read at <code>0n</code> , so checkout shows \$0.00 while the registrar charges full rent
IMM-WA-MED-92623	Medium	Role holder list replays only the newest 1,000 registry-wide events, letting a seller hide a retained delegate
IMM-WA-MED-92695	Medium	<code>setupControlledResolver</code> omits intent funding and keeps the session signer, so resolver setup cannot complete
IMM-WA-MED-92723	Medium	Missing <code>ccipRead</code> guard in the Explorer Worker client lets an attacker's resolver exhaust its RPC quota
IMM-WA-MED-92778	Medium	Manager's HCA registration verifier ignores the subregistry, certifying an attacker's registration of the name as complete
IMM-WA-MED-92816	Medium	Expiry-discovery cursor on non-unique <code>expiryDate</code> permanently drops names tied at a 1,000-row page boundary
IMM-WA-MED-92825	Medium	Missing registry scoping in the role History query lets anyone insert forged role-change entries into any name's history
IMM-WA-MED-92870	Medium	Manager lower-cases v2 role names before re-querying them, listing a third party's name as managed on any profile
IMM-WA-MED-92995	Medium	Raw-versus-normalized social diff re-authors a prior owner's handles into the replacement resolver on a Discord-only edit
IMM-WA-MED-93007	Medium	Address-page primary-name offer lets a name owner make their name a connected visitor's primary name

Finding ID	Severity	Report Name
IMM-WA-MED-93008	Medium	Transfer flow points <code>addr(60)</code> at the recipient before a transfer the recipient can revert, diverting payments
IMM-WA-LOW-89444	Low	Unauthenticated <code>/wallet/fund</code> with address-keyed controls lets anyone drain the faucet wallet
IMM-WA-LOW-89569	Low	Missing cross-account email uniqueness lets an attacker verify a victim's address on their own account
IMM-WA-LOW-90307	Low	Out-of-range on-chain expiry crashes the Explorer names page of any address sent an unsolicited subname
IMM-WA-LOW-91233	Low	<code>createDiff</code> treats any two upload-worker URLs as equal, leaving a former owner in control of profile media
IMM-WA-LOW-91443	Low	One-byte truncated CBOR ABI record on an unsolicited name hangs the victim's Manager dashboard
IMM-WA-LOW-91444	Low	Oversized coin-type 121 address record on an unsolicited name freezes the victim's Manager dashboard
IMM-WA-LOW-91446	Low	Quadratic primary-contact deduplication freezes the Manager tab when a gifted name is opened
IMM-WA-LOW-91450	Low	Unbounded <code>links</code> record rendering freezes the Manager tab for about 14 seconds on a gifted name
IMM-WA-LOW-91822	Low	Recipient field stays editable while a transfer is preparing, so the name goes to the replaced address
IMM-WA-LOW-91997	Low	Edit-profile save rewrites a previous owner's social records into the replacement resolver while promising to clear them
IMM-WA-LOW-92305	Low	Email verification rate limit keyed on the recipient lets any account block a person's email enrollment
IMM-WA-LOW-92544	Low	Explorer name page tells any visitor they own the name from URL parameters, inducing paid renewals of another's name
IMM-WA-LOW-92586	Low	Text record keyed on an <code>Object.prototype</code> member crashes a name's Manager profile page
IMM-WA-LOW-93124	Low	Predictable SIWE nonce consumed before verification lets an unauthenticated attacker lock a user out of login
IMM-WA-INSIGHT-89255	Insight	Telegram expiry notification interpolates ENS names unescaped into Markdown-parsed message text
IMM-WA-INSIGHT-89300	Insight	In-source SHA-256 comments for the patched <code>@rhinestone/sdk</code> disagree with each other and with the shipped patch
IMM-WA-INSIGHT-89371	Insight	Unverified <code>default.reverse</code> name shown as a forward-matched primary name on inheriting L2 rows
IMM-WA-INSIGHT-89389	Insight	Email channel endpoint reveals addresses held by other users and rate-limits only the literal address string
IMM-WA-INSIGHT-89461	Insight	Temporary registry-wide HCA operator approval survives an interrupted migration and is never revoked
IMM-WA-INSIGHT-89469	Insight	OG image worker's self-host guard compares hosts as raw strings, so a trailing-dot host bypasses it
IMM-WA-INSIGHT-89534	Insight	Session-enable signature commits to session policy only through an opaque salt the signer cannot read

Finding ID	Severity	Report Name
IMM-WA-INSIGHT-89599	Insight	Token page Namehash row hashes the punycode form of non-ASCII names, showing another name's node
IMM-WA-INSIGHT-89705	Insight	<code>getForwardResolutionRequest</code> targets the viewed page's address rather than the connected wallet
IMM-WA-INSIGHT-91732	Insight	Explorer homepage activity feed renders attacker-chosen text-record keys verbatim
IMM-WA-INSIGHT-92307	Insight	API worker persists arbitrarily large transaction payloads with no size limit
IMM-WA-INSIGHT-92553	Insight	Explorer homepage feed presents unverified <code>NameChanged</code> reverse-record strings as genuine ENS name badges
IMM-WA-INSIGHT-92954	Insight	Avatar and header uploads accepted from a wallet that does not own the target name
IMM-WA-INSIGHT-92979	Insight	Telegram auth verification accepts future-dated <code>auth_date</code> values

Findings – Websites & Apps

IMM-WA-CRITICAL-89253

Resolver roles removal revokes only the first grouped name, leaving the removed account's `ROLE_SET_ADDR` on other names

Report ID	#89253
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions · Signing or submitting a transaction against the wrong chain (for example when the request's target chain is not validated) · Inducing a blanket token approval (for example <code>setApprovalForAll</code> on the registry or <code>NameWrapper</code>) to an attacker-controlled address

Description

The Explorer's Resolver Roles view lists the accounts that hold roles on a `PermissionedResolver`, where each role is scoped either to the root resource or to one name's resource. Its "Remove user" confirmation asks "Are you sure you want to remove this user from all roles?" ([ResolverRolesSidebar.tsx#L441-L448](#)), so the removal is expected to revoke every role the row displays, on every resource it covers.

`groupRolesByAccount` groups role entries by the lower-cased account alone, pushing every resource's decoded roles into one `decodedRoles` list and every resolved name into one `resolvedNames` set ([resolverRoles.ts#L179-L221](#)). The sidebar then reduces that list to a single write target:

```
const selectedAccount = row?.original.account as Address
const decodedRoles = (row?.original.decodedRoles ?? []) as ResolverRoleKey[]
const resolvedNames = row?.original.resolvedNames ?? []
```

```
const roleName = resolvedNames.find((n) => n !== '(root)') ?? ''
```

`handleRemoveUser` passes that one `roleName` with the union of `decodedRoles` ([ResolverRolesSidebar.tsx#L307-L323](#)), the removal mutation calls `revokeResolverRoles` once ([ResolverRolesSidebar.tsx#L251-L277](#)), and `prepareRevokeResolverRolesTransaction` builds a `scope: 'name'` revocation for any non-empty name ([revokeResolverRoles.ts#L29-L75](#)).

The resource association is lost between the row and the calldata, however. When one account holds roles on two names served by the same resolver, the row shows both names, but the wallet is asked to sign a single `authorizeNameRoles` call carrying only the first name's DNS encoding, which clears only that name's resource. The transaction succeeds and the flow completes, while the account keeps its roles on every other resource folded into the row. The input is accurate, resource-specific indexed data; the grouping step discards the association. A row that mixes root and name-scoped roles behaves the same way: the non-root name is selected and the root-scoped revocation path is never invoked.

An owner revoking a collaborator who holds `ROLE_SET_ADDR` on `alpha.eth` and `beta.eth` signs a transaction that removes only the `alpha.eth` role, while the dialog promised removal from all roles. The former collaborator then replaces `beta.eth`'s ETH address with an attacker-controlled address, and users or applications resolving `beta.eth` send funds to the attacker. The application submits to the connected wallet a transaction whose arguments cover less than the action it presented, which is a malicious interaction with an already-connected wallet through modified transaction arguments.

Proof of Concept

1. `groupRolesByAccount` receives two accurate entries: one account holding `ROLE_SET_ADDR` on the `alpha.eth` resource and on the `beta.eth` resource of one resolver.
2. It returns a single row whose `resolvedNames` are `alpha.eth` and `beta.eth`, and the sidebar's selection logic picks `alpha.eth`.
3. The real `prepareRevokeResolverRolesTransaction` encodes `authorizeNameRoles` with the DNS encoding of `alpha.eth`, `ROLE_SET_ADDR`, the account and `false`.
4. Applying that single-resource revocation clears the `alpha.eth` role and leaves `ROLE_SET_ADDR` set on `beta.eth`.
5. The account still holds the role needed to set `beta.eth`'s ETH address record after the "Remove user" flow completes.

IMM-WA-CRITICAL-89264

Transfer flow never revokes the seller's root resolver roles, letting former owners rewrite a sold name's records

Report ID	#89264
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

Registration and migration give the owner's wallet authority over the per-HCA PermissionedResolver that serves the name, and the Explorer's transfer flow later moves the name to a new owner. At the registry level, ENSv2 moves the name's roles to the recipient with the ERC-1155 token, and the transfer form tells the sender that with the resolver detached "The recipient starts clean and sets up their own" ([SendNameForm.tsx#L45-L52](#)). Authority over the name's records is expected to follow ownership.

The HCA reveal batch always ends with a wildcard grant to the wallet:

```
// 6. authorizeNameRoles(hex"00", ROLES.ALL, wallet, true) - every session registration
calls.push({
  to: params.resolver,
  value: 0n,
  data: encodeFunctionData({
    abi: permissionedResolverAuthorizeNameRolesSnippet,
    functionName: 'authorizeNameRoles',
    args: ['0x00', ROLES_ALL, params.wallet, true],
  }),
})
```

Migration repeats the same grant with `ROOT_NAME = '0x00'` ([buildAtomicMigrationBatches.ts#L422-L433](#)).

The code documents that this call "maps the empty DNS name to the resolver root resource" ([migrationInvariants.ts#L281-L287](#)), and root roles apply to every node the resolver serves. The transfer plan, however, contains only `set-eth-addr`, `detach-resolver`, `detach-registry` and `transfer-token` steps ([buildTransferPlan.ts#L16-L43](#)), `transferToken` encodes a bare ERC-1155 `safeTransferFrom` ([transferToken.ts#L67-L71](#)), and detaching "never touches the resolver contract itself" ([detachNameResolver.ts#L4-L7](#)).

No step revokes the seller's root grant or gives the recipient any resolver role, and the record-edit hook states the outcome: "After a transfer that keeps the resolver, the previous owner typically retains these roles while the new token owner does not" ([useCanEditRecords.ts#L38-L40](#)). The resolver detach is pre-checked but the sender can turn it off, and it is hidden when the owner lacks registry `ROLE_SET_RESOLVER` ([SendNameForm.tsx#L69-L96](#), [useTransferDetachTargets.ts#L86-L90](#)). The recipient takes no part in that choice. Even after a detach, the seller's roles persist and become live again once any name resolves through that resolver.

After a transfer that keeps the resolver, the seller can at any time re-point the sold name's `addr` record to an attacker address and rewrite its text records, including the avatar and description the Portal renders, while the new owner's own `setAddr` reverts. The grant also covers subnames handed to third parties, which the subname form assigns the parent's resolver ([create-subname.tsx#L192-L207](#)), and unrelated nodes on the same resolver. The seller keeps `ROLE_UPGRADE` and can grant or revoke any account's roles, while the new owner can neither revoke the seller nor grant itself a role. Anyone who then pays the name pays the attacker: the application resolves and renders an ENS name to an attacker-controlled address, so a user sends funds to an unintended recipient.

Proof of Concept

1. On a Sepolia fork, deploy the resolver as the app does, with `deployProxy(PERMITTED_RESOLVER_IMPL, salt, initialize(hca, ROLES_ALL, []))`.
2. As the HCA, call `authorizeNameRoles(hex"00", ROLES_ALL, seller, true); hasRootRoles(ROLES_ALL, seller)` returns true.
3. Transfer the name as `transferToken.ts` does, with a single ERC-1155 `safeTransferFrom`; the resolver's role store is untouched.
4. The buyer's `setAddr(node, 60, buyer)` reverts.
5. The seller's `setAddr(node, 60, attacker)` succeeds and `addr(node, 60)` returns the attacker; the seller's `setText` also succeeds and it still holds `ROLE_UPGRADE`.
6. The seller also writes records on a third-party subname and on an unrelated node served by the same resolver.
7. With `setResolver(tokenId, 0)` before the transfer, the name stops resolving through the old resolver, but the seller still holds root roles on it.

IMM-WA-CRITICAL-89270

Profile route resolves records from the raw URL name and ownership from the normalized name, showing an attacker's address

Report ID	#89270
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Manager's `/$name` profile route assembles one page from several data sources: the name heading, the Owner field, the Registered and Expires dates, the "Main receiving address" card and the text records. All of them describe the name in the URL, and a user reads them together as one name's profile. ENS normalization is many-to-one: `Vitalik.eth`, or `vitalik.eth` with an embedded soft hyphen (U+00AD), normalizes to `vitalik.eth` while hashing to a different node, and the registrar registers label strings verbatim.

The loader fetches records and ownership in parallel from the same raw router parameter:

```
const [profileRecords, ownerData] = await Promise.all([
  queryClient.ensureQueryData(profileRecordsQuery(name)),
  // Fetch, not ensure: `ensureQueryData` serves invalidated data, so a name
  // cached as ownerless pre-registration would redirect its owner away.
  queryClient.fetchQuery(profileOwnerQuery(name)),
])
```

The parameter parser only checks for a length of at least three and the presence of a dot ([route.tsx#L5-L13](#)). `getOwner` normalizes before calling `ensjsv2_getOwner` ([profileOwner.ts#L24-L58](#)), and `getExpiry` and `getRegistration` do the same through `normalizeEth2LdName` ([profileExpiry.ts#L73-L110](#), [profileRegistration.ts#L62-L66](#)), which applies `normalize` from `viem/ens` ([profileName.ts#L14-L21](#)). `getProfileRecords`, however, passes the raw `name` straight to `getRecords` with no name normalization ([profileRecords.ts#L82-L132](#)), and its query is keyed on the raw string ([profileRecords.ts#L165-L169](#)). The Universal Resolver selects the resolver from the raw DNS-encoded name, so the resolver that the attacker's node points to answers the record calls. The page renders the Owner as `ownerReverseName || truncateAddress(owner)` from the normalized lookup ([ProfileDetails.tsx#L113-L123](#)), while the address

card's copy button carries `address.value` from the raw lookup ([ProfileAddressesSection.tsx#L90-L95](#), [ProfileAddressesSection.tsx#L154-L163](#)). Nothing cross-checks the two halves.

An attacker registers an unnormalized variant of a target name and points it at a resolver that answers with the attacker's address. The Manager then serves, on its own domain, a page whose heading reads as the target name, whose Owner, Registered and Expires fields are the genuine target's, and whose "Main receiving address", behind a copy button, is the attacker's. The heading uses `truncate`, which sets `white-space: nowrap`, so a soft hyphen never paints ([ProfileDetails.tsx#L22-L28](#)); the capital-letter variant needs no invisible character at all. Delivery is an ordinary link or a pasted name, and the cost is one registration per squatted name. The application renders an ENS name to an attacker-controlled address, corroborated by the genuine owner's metadata, so funds copied from that card go to an unintended recipient.

Proof of Concept

1. Offline, `namehash` of `vita` + U+OoAD + `lik.eth`, `Vitalik.eth` and `vitalik` + U+FEOf + `.eth` each differs from `namehash('vitalik.eth')`, although all three normalize to `vitalik.eth`.
2. On a local Anvil fork, the victim registers `pocvictim.eth` and sets an ETH address, avatar, description and primary name.
3. The attacker registers the raw label `pocvict` + U+OoAD + `im` directly on the registrar and, on the resolver it points to, writes the attacker's wallet as the ETH address under `namehash('pocvictim.eth')`, copying the victim's text records.
4. Loading `/pocvict%C2%ADim.eth` renders a heading that reads `pocvictim.eth`, the victim's reverse name as Owner, and the victim's Registered and Expires dates.
5. On the same page, the "Main receiving address" card shows and copies the attacker's wallet.
6. Control: with only `pocvictim.eth` registered, `/Pocvictim.eth` keeps Owner and dates populated while the Addresses card disappears, showing the two halves resolve different nodes.

IMM-WA-CRITICAL-89278

Transfer route authorizes and labels raw `ALICE.eth` but normalizes the token lookup, transferring the victim's canonical `alice.eth`

Report ID	#89278
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

In ENSv2 a name is an ERC-1155 token held in its registry, and the Explorer's transfer flow moves it with `safeTransferFrom` on the id returned by `getTokenId(label)` ([transferToken.ts#L3-L7](#)). The token moved must be the name the flow displays and authorizes. The app's validator, `isValidEnsName`, rejects any label that is not normalized ([isNormalized.ts#L39-L50](#)), and the name index page renders "Invalid name" for such input ([index.tsx#L441-L460](#)).

The nested `/$name/ownership/transfer` route does not call that validator, and the parent route uses it only to decide a TLD redirect ([\\$name.tsx#L25-L39](#)). The transfer route resolves the owner of the raw route `name` and, when the connected wallet matches, renders `AuthorizedTransfer` with that raw name ([transfer.tsx#L21-L27, #L102-L111](#)); `resolveEnsOwner` passes the string unchanged to the ensjs V2 `getOwner` and `getNameRegistries` ([resolveEnsOwner.ts#L44-L51](#)). The role check in `useCanTransferName` ([useCanTransferName.ts#L57-L73](#)) and the token lookup both go through `getLabel`, which calls `ens_normalize` ([getLabel.ts#L10-L19](#)):

```
mutationFn: ({ recipient, options }: StartTransferParams) =>
  getEnsTokenId({
    label: getLabel(name),
    registryAddress,
  }).map((tokenId) => ({ recipient, tokenId, options })),
```

[useTransferName.ts#L111-L115](#)

However, raw `ALICE` and canonical `alice` are distinct ENSv2 tokens with distinct owners: the owner gate follows the raw label, while `getLabel("ALICE.eth")` returns `alice`. Every user-facing string keeps the raw name, `Transfer name - ALICE.eth` ([useTransferName.ts#L174-L181](#)) and `Transfer ${name}` ([transferToken.ts#L86-L91](#)), while the calldata carries the token id of canonical `alice`, and no step maps

that id back to the displayed name. Registering raw `ALICE` needs no privilege: the PoC does it through the public `ETHRegistrar` commit/reveal flow.

An attacker registers raw `ALICE`, gifts it to a victim who owns canonical `alice.eth`, and directs the victim to `/ALICE.eth/ownership/transfer` to return the unsolicited token. The victim passes the owner gate as holder of `ALICE`, every screen says `ALICE.eth`, and the wallet signs `safeTransferFrom(victim, attacker, tokenId(alice), 1, 0x)`. Afterwards the attacker owns canonical `alice.eth`, while the victim still holds raw `ALICE`. The same split reaches renewal: `buildRenewIntent` encodes `getLabel(params.name)` while describing the transaction as `Renew ${params.name}` ([useRenewalTransactions.ts#L190-L224](#)). The application displays an incorrect name on a transaction the user then approves, and that transaction hands the victim's canonical name to the attacker.

Proof of Concept

1. Through the public `ETHRegistrar` commit/reveal flow, the victim registers `alice` and the attacker registers raw `ALICE`, which receives a distinct token id.
2. The attacker transfers the `ALICE` token to the victim, and `findOwner` for `ALICE.eth` now returns the victim.
3. `isValidEnsName('ALICE.eth')` returns false, yet the nested transfer route renders `SendNameForm` with raw `ALICE.eth` and runs `useCanTransferName` on it.
4. `useTransferName` calls `getEnsTokenId` with label `alice` while naming the transaction `Transfer name - ALICE.eth`.
5. `transferToken` submits a request described as `Transfer ALICE.eth` whose calldata decodes to `safeTransferFrom(owner, recipient, tokenId(alice), 1, 0x)`.
6. After the victim's transfer to the attacker, canonical `alice` is owned by the attacker and raw `ALICE` remains with the victim.

IMM-WA-CRITICAL-89314

Migration keeps a gifted V1 name's stale resolver and silently grants its previous manager `ROLE_SET_RESOLVER` on V2

Report ID	#89314
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

The Manager migrates a user's ENSv1 `.eth` names into ENSv2 in one atomic `executeByOwner` batch that the owner signs. For an unwrapped name the registrant holds the ERC-721 while the registry `owner` acts as its manager, and the flow presents any preserved manager as the user's own, describing the step as "Restore the existing managers for your names" ([WalletConfirmationStepsDialog.tsx#L84-L87](#)).

The Manager re-exports its classifier from the shared `@ens-apps/migration` package ([classifyNames.ts#L1-L11](#)). From subgraph data, that classifier sets `managerAddress` to the registry owner whenever it differs from the registrant and selects `resolverStrategy: 'keep-v1'` for any resolver outside `KNOWN_PUBLIC_RESOLVERS`; its eligibility check reads only `BaseRegistrar.ownerOf`. For a `keep-v1` name with an unknown resolver, `resolverFor` returns `name.v1ResolverAddress` ([encodeMigration.ts#L32-L44](#)), and `buildAtomicMigrationBatches` writes it into the `MigrationHelper` data ([buildAtomicMigrationBatches.ts#L481-L491](#)). When `managerAddress` is set, the same batch appends `buildRoleGrantCall` ([buildAtomicMigrationBatches.ts#L608-L614](#)):

```
const resource = labelToCanonicalId(name.label)

return {
  to: V2_CONTRACTS.ETHRegistry,
  data: encodeFunctionData({
    abi: ETH_REGISTRY_V2_ABI,
    functionName: 'grantRoles',
    args: [resource, ROLE_SET_RESOLVER, name.managerAddress],
  }),
}
```

[buildRoleGrantCalls.ts#L16-L24](#)

However, a V1 `BaseRegistrar.transferFrom` moves only the NFT: until the recipient calls `reclaim()`, the registry owner and resolver still belong to the previous holder, and the migration treats that stale state as the owner's own configuration. The confirmation UI names neither address, since no migration component renders `managerAddress`, `v1ResolverAddress` or `resolverStrategy`, and post-execution verification expects the grant because its expectations come from the same plan ([buildAtomicMigrationBatches.ts#L615-L620](#)).

An attacker registers `gift.eth`, points it at an attacker-controlled resolver and transfers the ERC-721 to the victim as a gift or sale. When the victim migrates it, alone or in a bulk selection, the batch they sign contains `migrate(label: "gift", owner: victim, resolver: attacker resolver)` and `grantRoles(canonicalId("gift"), 1 < 24, attacker)`. The attacker keeps control of the name's records, so funds sent to `gift.eth` resolve to an attacker-chosen address, and `ROLE_SET_RESOLVER` lets the attacker restore its resolver if the victim replaces it. Where `reclaim()` would have reset the manager before migration, the migration instead makes the previous holder a V2 role holder. This hijacks the ENS name during a Manager v1 to v2 migration, assigning its resolver to an attacker.

Proof of Concept

1. A `gift.eth` subgraph fixture has the victim as registrant, the attacker as registry owner, and an attacker contract outside `KNOWN_PUBLIC_RESOLVERS` as resolver.
2. `buildMigrationPlan` runs the production planning pipeline with the victim as `migrationOwner`, stubbing only a `{ chain: { id } }` public client.
3. Classification returns `tokenType: 'unwrapped'`, `resolverStrategy: 'keep-v1'`, the attacker resolver as `v1ResolverAddress` and the attacker as `managerAddress`.
4. The victim-signed `executeByOwner` batch decodes to `MigrationHelper.migrate` with `owner` set to the victim and `resolver` set to the attacker resolver.
5. The same batch decodes to `ETHRegistry.grantRoles(canonicalId("gift"), 16777216, attacker)`.
6. The plan's step descriptors, the only batch content the UI renders, contain neither attacker address.

IMM-WA-CRITICAL-89395

`resolveAddressOrName` falls back to the v1 registry controller, redirecting ENS name transfers away from the registrant

Report ID	#89395
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

The Explorer app's "name or address" inputs resolve through one hook, `useAddressResolution`, documented as "Shared by every 'name or address' input in the app (transfer recipient, add-user sheets, ...)" ([useAddressResolution.ts#L43-L45](#)). For an unwrapped ENSv1 name the codebase separates two roles: "registrant is Owner, registry owner is Manager" ([names.tsx#L184-L187](#)), and the migration hook describes the registry `owner` as "the controller, which may be a different address, so it is only a last-resort fallback" ([useMigrationStatus.ts#L31-L39](#)). The party a typed name identifies is its registrant.

The hook queries `getResolvedAddress`, which calls `resolveAddressOrName` ([getResolvedAddress.ts#L26-L29](#)). That helper asks the universal resolver for the name's `addr` record and, when there is none, substitutes the name's owner:

```
// Fallback for names that do not set an address record:
// use current ENS owner address so the role can still be granted.
if (!resolvedAddress) {
  const ownerResult = await getEnsOwner({ name: nameOrAddress })
  if (ownerResult.isOk()) {
    resolvedAddress = ownerResult.value?.owner ?? null
  }
}
```

[addUser.handlers.ts#L27-L34](#)

`getEnsOwner` goes through `resolveEnsOwner`, which tries the V2 registry first and otherwise returns the `owner` field of the ensjs v1 `getOwner` result ([resolveEnsOwner.ts#L84-L102](#)). Its return type, `ResolvedEnsOwner`, carries only `owner`, `registryAddress` and `protocolVersion` ([resolveEnsOwner.ts#L20-L24](#)).

For an unwrapped ENSv1 name, however, that `owner` is the registry controller, not the registrant who holds the registrar NFT, and `ResolvedEnsOwner` drops `registrant`, so no consumer can recover it. The fallback's comment scopes it to granting roles, yet it runs for every consumer, and two of them assign an asset. `SendNameForm` takes the resolved address as the transfer recipient, checking only `!!recipient && !isSelf && !isZeroAddress` ([SendNameForm.tsx#L81-L89](#)), and `transferToken` encodes it straight into an ERC-1155 `safeTransferFrom` ([transferToken.ts#L67-L71](#)). The create-subname route passes it as the new subname's `owner` ([create-subname.tsx#L161-L170](#)), with `roleBitmap` defaulting to `DEFAULT_ROLE_BITMAP`, every role ([create-subname.helpers.ts#L25-L27](#)). Where the default status line is used, `AddressNameInput` labels the substituted address "Resolved: 0x...", exactly as it labels a genuine record ([AddressNameInput.tsx#L62-L66](#)).

Controller and registrant diverge most often after a secondary sale the buyer has not `reclaim()`ed. Resolver records authorise on the controller, so the buyer cannot publish an `addr` record to clear the condition, and a seller who keeps the controller slot can hold it open.

A user who sends an ENSv2 name to such a recipient name transfers it to the controller instead of the registrant; the PoC reads the new owner back on-chain after a mined receipt, and the form itself warns that the transfer is irreversible ([SendNameForm.tsx#L116-L117](#)). On the subname surface, the controller becomes owner of a subname inside the victim's own name, holding every role. The flaw redirects the recipient of a name transfer, hijacking the ENS name being sent.

Proof of Concept

1. On a Sepolia fork, `demo.eth` (ENSv1) has its registrar NFT moved to Bob with `BaseRegistrar.transferFrom`, the registry controller left with the seller, and its resolver cleared so it has no `addr` record.
2. `resolveEnsOwner("demo.eth")` misses in the V2 registry and returns the seller's address from the v1 branch.
3. Alice types `demo.eth` as the recipient of her ENSv2 name `hello.eth`, and `resolveAddressOrName` returns the seller rather than Bob.
4. The transfer is mined, and `ensjs.getOwner` reads the owner of `hello.eth` back as the seller.
5. Creating `pay.hello.eth` with `demo.eth` in the Owner field makes the seller its owner with role bitmap `0x1111...1111`.
6. Bob's attempt to set his own `addr` record on `demo.eth` is mined and reverts while the seller holds the controller.
7. Control: with the `addr` record of `demo.eth` pointed at Bob and the controller unchanged, the app resolves to Bob.

IMM-WA-CRITICAL-89431

Retained successful actor under the shared `tx-revoke-roles` ID marks a later role removal done without revoking

Report ID	#89431
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/packages/transaction-manager
Impact	Direct theft of user funds

Description

The transaction manager tracks each transaction as an XState actor in an active map keyed by transaction ID, and the Explorer's transaction modal derives each step's status from that map. A "Remove user" action on a name's Roles page is meant to build and submit a revocation for the selected account, and the modal's Done state is meant to report that this revocation executed.

When an actor reaches `success` or `error`, the subscriber in `startTransaction` archives it and emits telemetry ([transactionManager.ts#L290-L336](#)) but never removes it from `this.transactions`, where `startTransaction` inserted it ([transactionManager.ts#L338-L340](#)); only the explicit `cancelTransaction`, `clear` and `clearAllAndPersistence` paths remove entries ([transactionManager.ts#L348-L362](#), [transactionManager.ts#L480-L509](#)). Closing the modal clears the actor only on error, keeping a success so the user "might want to reopen and see the result" ([TransactionModal.tsx#L56-L65](#)), and a route change only closes the modal ([useTransactionModal.ts#L36-L43](#)). Every role revocation uses the constant `REVOKE_ROLES_TX_ID = 'tx-revoke-roles'`, whatever the name, account or roles ([buildRoleTransactionDescriptors.ts#L12-L13](#), [buildRoleTransactionDescriptors.ts#L83-L95](#)). The overview reads status by descriptor ID ([TransactionsOverviewContent.tsx#L50-L53](#)) and, on success with no next step, completes the flow:

```
onClick={() => {
  if (transactionStatus === 'success' && !hasNextTransaction) {
    activeTransaction.onDone()
    return
  }
}
```

A new removal descriptor therefore inherits any retained successful actor that shares its ID, however. Its `onStart`, the only path to `revokeRoles` ([buildRoleTransactions.ts#L61-L76](#), [buildRoleTransactions.ts#L104-L108](#)), never runs. The modal shows the new target's name beside the earlier transaction's hash, and Done invokes the cleanup `handleDone` ([RolesSidebar.tsx#L104-L110](#)). No calldata, wallet request or on-chain revocation is produced.

An owner who completed an earlier revocation and left its success modal with the close button or by navigating away, rather than clicking Done, later removes a delegate holding `ROLE_SET_RESOLVER` on `victim.eth`. The removal completes with no wallet prompt, and the delegate keeps its role. The delegate can then point `victim.eth` at an attacker-controlled resolver, so funds subsequently sent to the name are delivered to the attacker. The path requires a delegate that is already authorized and turns hostile, and an owner who relies on the success screen; no stored-state tampering, malformed name or contract defect is involved. The false completion leaves resolver authority in place and results in direct theft of user funds.

Proof of Concept

1. A role revocation runs through the real `transactionManager` under `tx-revoke-roles`, reaches `success`, and its actor remains retrievable with its hash.
2. With that actor retained, the Explorer's `TransactionModal` opens for a new "Remove user" descriptor for `victim.eth` built by the production `buildRoleTransactions`.
3. The modal immediately offers Done, and the operation detail shows `victim.eth` beside the earlier transaction's explorer link; no wallet request is made.
4. Clicking Done calls `handleDone` once, while the new descriptor's `revokeRoles` callback is called zero times.
5. `hasRoles` for `victim.eth` still reports `ROLE_SET_RESOLVER` for the delegate, and no revocation transaction for it exists.
6. Contract tests confirm that a holder of that role can call `setResolver` and that a genuine owner revocation removes it.

IMM-WA-CRITICAL-89456

Renewal route reads expiry from the normalized name but signs the raw URL label, renewing a lookalike name

Report ID	#89456
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Manager's `/renew/$name` route renews a single v2 `.eth` name taken from the URL. Its loader derives both the label that will be signed and the expiry that gates and populates the page from the same `name` parameter, so the two are expected to describe one registration.

```
const parsedName = parseRenewableName(name)

if (parsedName.isErr()) {
  throw parsedName.error
}

const expiryData = await queryClient.ensureQueryData(
  profileExpiryQuery(name),
)
```

`parseRenewableName` delegates to `parseName`, which applies only `trim().toLowerCase()` and rejects `/\s/` and a fixed punctuation set ([name-parser.ts#L11](#), [name-parser.ts#L45-L75](#)). `profileExpiryQuery(name)` instead reaches `getExpiry`, which fetches the expiry for `normalizeEth2LdName(name)` ([profileExpiry.ts#L73-L110](#)), and that helper applies `normalize` from `viem/ens` ([profileName.ts#L14-L21](#), [profileName.ts#L62-L73](#)). The loader returns `parsedName.value.label` together with `expiryData.expiry` ([\\$name.tsx#L60-L63](#)), and `submitRenewActor` encodes `renew(cleanLabel, ...)` with only `.eth` stripped and describes the transaction as `Renew ${input.label}.eth` ([registration.actors.ts#L1020-L1031](#), [registration.actors.ts#L1072-L1094](#)).

For a label containing a code point that ENSIP-15 removes rather than rejects, such as U+200B, the two pipelines disagree, however. U+200B is not matched by `/\s/` and is not in the punctuation set, so it survives into the signed label, while the expiry lookup and the `canRenewV2Name` gate ([\\$name.tsx#L52-L54](#)) resolve the victim's real name. The page header renders `${label}.eth` verbatim ([PricingDomainHeader.tsx#L7-L9](#)), and

the transaction description uses the same raw label; both display as the victim's name because the character has no width. Punycode does not reach this state, since `normalize` throws on an `xn--` label and the route errors before rendering.

An attacker registers `vitalik` followed by U+200B directly on the registrar and sends the owner of `vitalik.eth` a renewal link for the crafted name. The page shows `vitalik.eth` with its genuine current expiry, the victim confirms a transaction described as renewing `vitalik.eth`, and `ETHRegistrar.renew` extends the attacker's name with the victim's funds. The victim's own name is not renewed and continues toward grace and expiry, while every value the flow displays points to the name the victim meant to renew. The application shows the wrong name on a transaction the user then approves, and the payment extends an unintended registration.

Proof of Concept

1. `parseRenewableName` on `vitalik` + U+200B + `.eth` returns the eight-code-point label that still contains U+200B.
2. `normalizeEth2LdName` on the same input returns `vitalik.eth`, and `canRenewV2Name` passes with the victim name's expiry.
3. The shipped `Route.options.loader`, called with the crafted name, returns the U+200B label alongside the expiry the chain holds for `vitalik.eth`.
4. Control: for `vitalik.eth` the loader's label matches the normalized name, and `xn--ls8h.eth` is rejected by `normalizeEth2LdName`.
5. The attacker registers the crafted label, and the victim opening `/renew/vitalik%E2%80%8B.eth` sees `vitalik.eth` with its real expiry and confirms.
6. `renew` executes with the crafted label, extending the attacker's name while the expiry of `vitalik.eth` is unchanged.

IMM-WA-CRITICAL-89601

Missing ENSIP-15 normalization in Manager registration makes users buy a different, permanently unresolvable name

Report ID	#89601
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

Manager's registration path takes a name from the `/register/$name` URL or the landing search, checks its availability, quotes a price and builds the commit and register calls. Other ENS clients reach a name only through its ENSIP-15 normalized form, and the repository's own review rule states that "User-supplied ENS names must pass through `normalize` from `viem/ens` before being used in contract calls or queries" ([config.json#L190-L193](#)).

Every stage of the path handles the raw string instead. `normalizeDomainNameFromUrl` only strips and re-appends `.eth` and truncates ([domain.ts#L27-L33](#)), and the search input goes through `normalizeQuery`:

```
export const normalizeQuery = (query: string): string => {
  const trimmed = query.trim().toLowerCase()
  if (!trimmed) return ''
  return trimmed.endsWith('.eth') ? trimmed : `${trimmed}.eth`
}
```

`validateENSName` checks spaces, dots, length and an ASCII character blocklist ([nameUtils.ts#L90-L143](#)), `parseName` again only trims and lowercases ([name-parser.ts#L46](#)), and `checkRealNameAvailability` passes the name to `ensjs.getAvailable`, which queries `isAvailable` with the leaf label as given ([nameChainContractService.ts#L28-L49](#)). The EOA path passes `name.replace('.eth', '')` to `makeCommitment` and `register` ([registration.actors.ts#L141-L158](#), [registration.actors.ts#L243-L252](#)); the HCA reveal batch keys `setAddr` on `namehash` of the raw label and passes the same label to `register` ([registration-calls.ts#L211-L224](#), [registration-calls.ts#L280-L298](#)). Within `apps/manager`, `normalize` is imported only by `profileName.ts` and `setPrimaryName.ts`, and neither touches the label before the availability check, the commitment or `register`.

However, `toLowerCase()` is not ENSIP-15 normalization: it does not fold fullwidth forms, circled letters, ligatures or superscripts, does not strip ignorables such as the soft hyphen, and does not reject forbidden characters. Such a label passes validation, is reported available and priced, and is registered verbatim, while every conformant client maps the same input to its normalized form, a different node.

On Sepolia at block 11523539, six of the tested inputs were available as typed while their normalized form was already taken: `v i t a l i k` is quoted 8.000021 USDC/year while `vitalik` reverts as unavailable, and `են` is billed at the three-character tier of 640.000005 USDC/year. The user pays and receives a label that no ENSIP-15 client can look up, send to or use as a primary name, and its `addr` record is written under a node no conformant resolver queries. The two primary-name branches also disagree: the EOA branch normalizes through `submitPrimaryNameForward`, pointing the reverse record at `vitalik.eth`, a name the user does not own, and throws after payment for labels ENSIP-15 rejects, while the HCA branch writes the raw name ([registrationUi.machine.ts#L508-L511](#), [registrationUi.machine.ts#L577-L578](#), [setPrimaryName.ts#L237](#)). The registration flow thereby bricks the name the user paid for, leaving it permanently inaccessible after the flow.

Proof of Concept

1. Apply Manager's `normalizeQuery` and `toLabel` and, separately, `ens_normalize` to each probe input, then query `ETHRegistrar.isAvailable` on Sepolia for both labels; six inputs are available raw and taken normalized.
2. Read `getRegisterPrice` for `v i t a l i k` (8.000021 USDC/year) and `են` (640.000005 USDC/year); the same call for `vitalik` reverts.
3. Pass Manager's label for the input `fire` to the production `buildRevealBatch`: the `register` calldata contains the raw label and not `fire`.
4. The same batch's `setAddr` is keyed on `namehash("fire.eth")`, not `namehash("fire.eth")`.
5. Call `submitPrimaryNameForward` with `v i t a l i k .eth`: the encoded `setName` argument is `vitalik.eth`, while `buildRevealBatch` encodes `setNameWithHCA` with the raw name.
6. The same EOA call throws for a label containing U+200D, which `register` accepts.

IMM-WA-CRITICAL-90094

Explorer renewal displays the raw name but signs **renew** for its normalized twin, extending another owner's registration

Report ID	#90094
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Explorer's renew flow lets the owner of a **.eth** name select it from their names list, confirm the extension and pay for it. The name the user sees and confirms must be the name the signed transaction renews. Because the production registrar accepts non-normalized labels, a raw name such as **PDIV1787437457033.eth** and its normalized form **pdiv1787437457033.eth** are two distinct registrations that different accounts can own.

The only shape check between the names list and the confirm screen is a regex in **isExtendable2LD** ([nameExtension.ts#L66](#)), consumed by **useCanExtend** ([useCanExtend.ts#L67](#)). The confirm screen renders **selectedName.name** verbatim ([ExtendNameConfirmation.tsx#L32-L33](#)), as does the multi-name card ([MultiNameConfirmCard.tsx#L35](#)). The renewal intent, by contrast, derives its on-chain name from **getLabel**, which returns the first label of **ens_normalize(name)** ([getLabel.ts#L10-L20](#)):

```
function buildRenewIntent(params: RenewParams): CustomTransactionIntent {
  // ensjs splits the label without normalizing, so pass a normalized 2LD name.
  const writeParams = renewNameWriteParameters(
    params.publicClient as unknown as Parameters<
      typeof renewNameWriteParameters
    >[0],
    {
      name: `${getLabel(params.name)}.eth`,
      duration: BigInt(params.duration),
    }
  )
}
```

However, when a name differs from its normalized form, the display layer shows one node while the action layer renews another, and nothing on the path from **useCanExtend** through **ExtendNameConfirmation** to **buildRenewIntent** checks that **ens_normalize(name)** equals **name** before confirmation. The user confirms **PDIV1787437457033.eth**, but the signed **renew** names **pdiv1787437457033**.

In the executed end-to-end run on a Sepolia fork, the victim owned `PDIV1787437457033.eth` and a second account owned `pdiv1787437457033.eth`, both minted through the production commit/reveal flow. The victim confirmed an extension of their own name through the shipped UI, with every screen naming that name, and paid 8,005,501 USDC base units; the second account's expiry grew by 31,557,600 seconds (about 365 days) while the victim's expiry did not change. Whoever holds the normalized twin captures the payment, and a holder who registers the twin in advance captures every later renewal, while the victim's real name silently moves toward expiry. When no one owns the twin, the renewal reverts after confirmation. The transfer flow carries the same divergence, deriving the token ID from `getLabel(name)` ([useTransferName.ts#L112-L113](#)). The application therefore shows the wrong name on a transaction the user then approves, so the user's payment funds an unintended recipient's registration.

Proof of Concept

1. On a Sepolia fork, register `pdiv1787437457033.eth` to a second account and `PDIV1787437457033.eth` to the victim through the production commit/reveal flow.
2. As the victim, open the Portal names list, which shows `PDIV1787437457033.eth` verbatim, and choose Extend.
3. The confirm screen heading reads `PDIV1787437457033.eth`; confirm and sign, and the approve and renew transactions report success.
4. The decoded `renew` calldata carries the label `pdiv1787437457033`.
5. The second account's expiry increases by 31,557,600 seconds, the victim's expiry is unchanged, and the victim's USDC balance has decreased.
6. Control: the same flow on the already-normalized `pctr11787437492969.eth` renews the victim's own name, and the signed label matches the displayed one.

IMM-WA-CRITICAL-90190

Bulk renewal displays the raw indexer name but renews the normalized label, extending a registration the user does not own

Report ID	#90190
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Manager dashboard lets a user select owned v2 `.eth` names and renew them in one flow. `toBulkRenewName` maps each selected row to a `BulkRenewName` payload that carries a `displayName` for the confirmation dialog and a `label` for the `ETHRegistrar.renew` calldata ([NamesTable.tsx#L81-L93](#)). The two fields are meant to identify the same registration: the function's own comment says the normalized name is used so that the label does not "hash to the wrong label".

`displayName` comes from `resolveDomainLabel`, which returns `domain.name ?? domain.normalizedName ?? domain.id`, the verbatim label the indexer read on chain ([utils.ts#L42-L46](#)). The calldata label comes from `domain.normalizedName`, its ENSIP-15 fold:

```
const displayName = resolveDomainLabel(domain)
// Use the canonical normalized name for the on-chain label - the display name
// may be unnormalized (e.g. mixed case) and would hash to the wrong label.
const name = domain.normalizedName ?? displayName
return {
  displayName,
  label: name.replace(/\.eth$/i, ''),
}
```

The dialog renders `row.displayName` ([NamesBreakdown.tsx#L40](#)), while the price quote ([useBulkRenew.ts#L155](#)), the avatar and profile preview ([useBulkRenew.ts#L186-L194](#)) and the submitted `label` ([useBulkRenewSubmit.ts#L76-L83](#)) all use the normalized value. `encodeRenewData` passes that label to `renew` with only `.eth` stripped ([registration.actors.ts#L1020-L1031](#)).

The comment has the direction reversed, however. The v2 `ETHRegistrar` registers the label string as supplied, so the raw label is the on-chain name and the normalized label hashes to a different, separately ownable registration. ENS normalization is many-to-one, and some of the characters it removes, such as

U+200B, have no visible width, so the row and the dialog can render a name identical to the one being renewed while the two hash differently. `renew` carries no ownership check, so the transaction succeeds. The dashboard query filters only on `owner_in` ([useOwnedDomains.ts#L36-L46](#)) and never reads the indexer's `isNormalized` field, so nothing marks such a row.

`register` accepts an arbitrary owner, so an attacker can mint a lookalike label directly into a victim's dashboard, choosing one whose normalized form is a name the attacker already holds. "Select all" builds its set from the full owned-names list rather than the visible page ([NamesTable.tsx#L137-L151](#), [NamesTable.tsx#L185-L197](#)), so the planted row is swept in with the user's own renewals. The same split occurs with no attacker for any row whose `name` differs from its `normalizedName`, such as a mixed-case label. On a local Sepolia fork, the victim paid 8,000,021 raw USDC units (6 decimals), the registration shown in their row gained 0 seconds, and a registration owned by a third party gained 31,536,000 seconds. The application shows the wrong name on a transaction the user then approves, and the payment extends an unintended registration.

Proof of Concept

1. On a local Sepolia fork, the attacker registers `pocTarget2608` to a third-party address and the lookalike `pocTarget` + U+200B + `2608` directly to the victim's address.
2. The project's indexer, queried with the dashboard's `owner_in` filter, returns the lookalike row with `name` differing from `normalizedName` and `isNormalized: false`.
3. The real `NamesTable` renders the row as the raw name, and "Select all" plus Renew hand the dialog a raw `displayName` with a normalized `label`; a mixed-case `Vitalik.eth` row splits the same way.
4. `useBulkRenewSubmit` calls `submitRenewActor` with the normalized label, and the resulting `renew` calldata decodes to that label.
5. The victim's `renew` succeeds on the fork and 8,000,021 raw USDC units leave the victim's balance.
6. The lookalike the victim owns gains 0 seconds, while the third party's `pocTarget2608` gains 31,536,000 seconds.

IMM-WA-CRITICAL-90379

`ResolverRolesSidebar` keys its permission draft to the row index, granting a shifted account an unselected resolver role

Report ID	#90379
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Subdomain takeover without already-connected wallet interaction

Description

The Explorer's Roles page lists the accounts holding roles on a resolver and lets its administrator open a sidebar to edit or remove one account's roles. A permission draft opened for an account must stay bound to that account: removing or reordering one row must never carry its selected permissions into another account's transaction.

The shared `DataTable` accepts only `columns` and `data` and calls `useReactTable` without a `getRowId` ([DataTable.tsx#L28-L47](#)), so TanStack identifies rows by array index, and that positional `row.id` also keys each rendered row ([DataTable.tsx#L83-L87](#)). `ResolverRolesTable` keeps the selected `editingRow` and the mounted sidebar across data refreshes ([ResolverRolesTable.tsx#L97-L138](#)). The sidebar reads the account and original roles from whichever row it receives ([ResolverRolesSidebar.tsx#L165-L173](#)) but re-initializes the editable draft only when the row ID changes:

```
// Sync edited permissions when the selected row changes
const [prevRowId, setPrevRowId] = useState<string | null>(null)
if (row && row.id !== prevRowId) {
  setPrevRowId(row.id)
  setEditedPermissions(roleToPermissions(row.original.decodedRoles))
}
```

`computeRoleChanges` grants every permission present in the draft and absent from the account's current roles ([rolesToPermissions.ts#L66-L106](#)), `handleSaveChanges` snapshots those grants with the current account ([ResolverRolesSidebar.tsx#L295-L305](#)), and `grantResolverRoles` encodes them as a root grant to that account ([grantResolverRoles.ts#L45-L74](#)).

When the administrator removes account A, however, the chain-backed refetch drops A and shifts its successor B into A's index while the page and sidebar stay mounted. Opening B switches the displayed account and original roles to B, but the reused `row.id` suppresses the draft reset, so A's checked permissions remain selected. The draft is bound to an array position, not to an account.

Any ordinary edit to B is then diffed against A's stale draft. In the reproduced flow A holds Text and Address, B holds only Text, and the administrator selects only Content Hash for B; Portal submits `grantRootRoles(257, B)`, where `257 = ROLE_SET_ADDR | ROLE_SET_CONTENTHASH`. The Save review names only the action and the account, with no role delta ([ResolverRolesSidebar.tsx#L339-L349](#)), so nothing reveals the extra grant. B needs no access to the administrator's browser or wallet and can stay passive until the grant is mined. It then holds a root `ROLE_SET_ADDR` that no one selected for it, and uses it to rewrite the name's ETH address record to an address of its choosing (`0x1111111111111111111111111111111111111111` in the PoC). This is a subdomain takeover without already-connected wallet interaction: the application's own Save flow hands B control over where the name resolves.

Proof of Concept

1. On a Sepolia fork, seed account A with root roles `TEXT | ADDR` (17) and account B with `TEXT` (16) on the resolver.
2. Open A in the Roles sidebar and remove it; Portal sends `revokeRootRoles(17, A)` and, after the refetch, B moves into row 0 without a page or sidebar remount.
3. Open B: the sidebar shows B's account and Text role, with Address still checked from A's draft.
4. Select only Content Hash and save; the wallet request decodes to `grantRootRoles(257, B)` and B's on-chain bitmap becomes `273`.
5. Send `setAddr` as B: it succeeds and the name's ETH record now points to the address B chose.
6. Control: after the owner revokes only the two unintended roles, the same `setAddr` from B reverts.
7. Control: repeating a predecessor removal but reloading the page before opening the successor leaves Address unchecked and Save disabled.

IMM-WA-CRITICAL-90544

`setPrimaryNameWithHca` normalizes the selected name after confirmation, making Manager render an attacker-controlled Optimism recipient

Report ID	#90544
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

Manager's primary-name flow lets a user choose one of their owned names in `ChoosePrimaryNameDialog` and claim it as the primary name of their address through an owner-signed HCA intent. The flow rests on one invariant: the registration the user selects, the registration they confirm and the primary identity they sign are the same.

The Dashboard labels each owned domain with `resolveDomainLabel`, which prefers the indexer's raw `name` field over `normalizedName` ([utils.ts#L42-L46](#)). The dialog renders that label, stores it as the selection and, on confirmation, passes it unchanged to `submitPrimaryName` ([ChoosePrimaryNameDialog.tsx#L81-L96](#), [ChoosePrimaryNameDialog.tsx#L467-L471](#)). Registration does not canonicalize labels either: `parseName` only trims and lowercases ([name-parser.ts#L46](#)) and the HCA reveal batch hands the label to `register` as-is ([registration-calls.ts#L280-L298](#)). A decomposed NFD label such as `e\u0301-hca-1811111111.eth` and its composed NFC twin `\u00E9-hca-1811111111.eth` therefore render identically but are separate registrations that different accounts can own. The HCA branch of `useSetPrimaryName` calls `setPrimaryNameWithHca` ([useSetPrimaryName.ts#L65-L74](#)), which re-derives the name before encoding it:

```
// Normalized claim string: a non-canonical name would fail the bidirectional
// check at resolution time and read as "no primary name".
const withEthSuffix = (name: string) =>
  normalize(name.endsWith('.eth') ? name : `${name}.eth`)
```

The result becomes the `setNameWithHCA` argument of the owner-signed request ([setPrimaryName.ts#L140-L165](#)).

However, `normalize` runs only after the raw name has been displayed, checked, selected and confirmed, and nothing verifies that the normalized string still designates the registration the user chose, or one they own.

The only guard checks that the wallet controls `ownerAddress` ([setPrimaryName.ts#L142-L149](#)). A victim who selects their NFD name signs a primary-name claim for the attacker's NFC name, with no visible difference on screen.

The attacker keeps the NFC name's ETH (coin type 60) record pointed at the victim, so Manager's reverse lookup, a `getEnsName` call with the default coin type ([profileReverseName.ts#L30-L41](#)), returns the attacker-owned name for the victim's address. The attacker then sets the NFC name's Optimism record (coin type `2147483658`) to their own address, which leaves the ETH match intact. A sender who opens the victim's address profile follows its "View profile" link to the NFC name ([AddressProfileHeader.tsx#L233-L240](#)), where Manager takes ETH as the main address and lists every differing record as chain-specific ([ProfileView.helpers.ts#L229-L277](#)); the Optimism card's `CopyableButton` copies the attacker's full address ([ProfileAddressesSection.tsx#L127-L151](#)). The attacker still owns the NFC name and can rotate that record at any time without further victim interaction. Manager thus displays and pre-fills an incorrect address, so that a user sends funds to an unintended recipient: Optimism payments meant for the victim reach the attacker.

Proof of Concept

1. On a local Sepolia fork, the attacker registers the NFC name `\u00E9-hca-1811111111.eth` and the victim registers the visually identical raw NFD name `e\u0301-hca-1811111111.eth`; the registry reports two different owners.
2. The attacker sets the NFC name's ETH (60) record to the victim's address and its Optimism (`2147483658`) record to the attacker's address.
3. The NFD name selected in `ChoosePrimaryNameDialog` reaches `setPrimaryNameWithHca`, and the owner-signed intent fills successfully with the NFC name as the decoded `setNameWithHCA` argument.
4. Manager's `getEnsName` for the victim returns the attacker-owned NFC name, and an independent ENSjs strict check reports `match=true`.
5. The NFC profile shows the victim as the main ETH address and the attacker as the Optimism chain-specific address; clicking that card writes the attacker's full address to the clipboard.
6. Control: repointing the NFC name's ETH record to the attacker produces `ReverseAddressMismatch` and removes the victim's primary name.

IMM-WA-CRITICAL-90725

Deleting a literal `[labelhash]` subname unregisters the victim subname it encodes, letting a registrar take it over

Report ID	#90725
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Subdomain takeover with already-connected wallet interaction

Description

The Explorer's Subnames page lets an account holding `ROLE_UNREGISTER` on a parent registry delete child names, and both the inline confirmation and the transaction modal name the row the user selected. The codebase recognises that bracket notation is ambiguous: `getLabelRegistrationError` rejects `[...]` labels because "a literal `[...hash...]` label would later be indistinguishable from a genuine encoded labelhash (a known ENS spoofing vector)" ([isNormalized.ts#L59-L78](#)).

For ENSv2 names, `getSubnames` asks the indexer for each child's `name`, `labelName` and `labelhash` ([useSubnames.ts#L61-L73](#)). The route then maps every child into a `SubnameRow` holding only `name`, `owner` and `canDelete` ([subnames.tsx#L372-L378](#); [SubnamesTable.tsx#L36-L41](#)). On delete, it rebuilds the label from the displayed name:

```
const getLabel = useCallback(
  (subname: string) => {
    const suffix = `.${name}`
    if (subname.endsWith(suffix)) {
      return subname.slice(0, -suffix.length)
    }
    return subname.split('.')[0]
  },
  [name],
)
```

[subnames.tsx#L177-L186](#)

Both the submitted mutation and the gas-estimate intent pass `getLabel(subname.name)` ([subnames.tsx#L207-L212](#), [#L263-L272](#)), and `prepareDeleteSubnameTransaction` hands that string to `deleteSubnameWriteParameters` ([delete-subname.helpers.ts#L35-L41](#)). The modal title stays `Delete` `${subname.name}` ([subnames.tsx#L286](#)), and the inline confirmation reads `Remove {row.original.name}?` ([SubnamesTable.tsx#L418](#)). The page offers deletion only when the connected account holds `ROLE_UNREGISTER` ([subnames.tsx#L370](#)), and that gate works as intended: the deleting user is genuinely authorized.

However, a literal label `[H]` and the encoded-labelhash notation for hash `H` are the same string. The encoder reads `[H]` as the precomputed hash `H`, and the mined PoC transaction decodes to `unregister(H)`, while the selected row's token key is `B = keccak256("[H]")`, which the indexer returned and the route discarded. The registration guard covers only the Explorer's own inputs; a registrar calling the registry directly can still register `[H]`.

An attacker holding only delegated `ROLE_REGISTRAR` on a parent registers the literal label `[H]`, where `H` is the labelhash of a valuable victim-owned child. When the victim deletes that visibly unusual row, the confirmation and modal name `[H].parent.eth`, but the wallet signs `unregister(H)`: the victim's child becomes available and the attacker's `[H]` row stays registered. The attacker then registers `valuable` to itself with the same registrar role, and holds it until a parent administrator unregisters and reissues the label. The victim's legitimate unregister authority is redirected to a different token, producing a subdomain takeover with already-connected wallet interaction.

Proof of Concept

1. On a private Sepolia fork, the victim owns `valuable` (token key `H`) under a parent registry and holds `ROLE_UNREGISTER`, while the attacker holds only root `ROLE_REGISTRAR` and its direct `unregister(H)` is rejected.
2. The attacker registers the literal label `[H]` through a direct registry call, under token key `B = keccak256("[H]")`.
3. The indexer returns both children with their distinct labelhashes `H` and `B`.
4. The victim opens the Subnames page and deletes the `[H]` row; the inline confirmation and transaction modal both name `[H].bracket-parent-1889999999.eth`.
5. The mined transaction decodes to `unregister(H)`: `valuable` becomes available (status `0`) while `[H]` stays registered to the attacker (status `2`).
6. The attacker registers `valuable` with its registrar role and is read back as its owner.
7. Control: for an ordinary label, the literal hash and the builder's hash match.

IMM-WA-CRITICAL-91014

SDK patch labels same-chain scoped HCA session signatures as owner mode, letting a session key drain HCA tokens

Report ID	#91014
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/packages/transaction-manager
Impact	Unauthorized takeover of a user's smart account (Hybrid Custody Account) or authenticated session, letting an attacker transfer the victim's names or funds or act as them, such as: adding, replacing, or escalating an account owner or co-owner · extending or widening a session key's authority beyond its intended scope or lifetime · stealing an authenticated session or bearer token · (Exploiting the 7-day session-key design as described in the Known Issues is not eligible; a weakness that grants authority beyond its stated lifetime or permissions is in scope.)

Description

In the standalone-HCA model the wallet signs one scoped session authorization, after which an ephemeral session key signs Intents without a wallet prompt ([signer.types.ts#L35-L43](#)). That session is bounded, including refund caps baked into its salt ([DEBUGGING_INTENTS.md#L146-L152](#)). The deployed IntentExecutor tells the two principals apart by the outer signature mode: execution-emissary modes are session-key operations and must settle refunds through `Paymaster.settleGasRefund_requireCallback()`, while plain ERC-1271 is the trusted owner and settles directly through `settleGasRefund()`. The callback exists so that a session key cannot drain account funds through refund parameters.

The transaction manager hands an active session to `@rhinestone/sdk` as `experimental_session` with `verifyExecutions: true` ([warp-transport.actor.ts#L114-L121](#); [registration.hca.actors.ts#L288-L297](#)). The monorepo applies its own patch to `@rhinestone/sdk@1.8.0` ([package.json#L58-L62](#)), and the patch rewrites the signature-mode choice in `resolveSignatureMode`:

```

return anyVerifyExecutions
-   ? types_1.SIG_MODE_EMISSARY_EXECUTION_ERC1271
+   ? hasHcaFundingSession && standaloneHcaRecipient
+   ? types_1.SIG_MODE_ERC1271_EMISSARY_EXECUTION
+   : standaloneSameChain
+   ? types_1.SIG_MODE_ERC1271
+   : (0, hca_1.isStandaloneHca)(config)
+   ? types_1.SIG_MODE_EMISSARY_EXECUTION
+   : types_1.SIG_MODE_EMISSARY_EXECUTION_ERC1271
: types_1.SIG_MODE_ERC1271;

```

However, for a standalone HCA whose chains all equal the target chain, the patched branch returns `SIG_MODE_ERC1271`, outer mode `0x0201`, although the signature still comes from the scoped session key; stock 1.8.0 returns `0x0205` for the same input, as the PoC's differential run of both packages shows. The IntentExecutor reads only that outer mode, classifies the session signature as the owner's, and skips the mandatory callback.

A holder of a valid session builds an approval-only operation for the configured payment token and Paymaster, which the session policy accepts without any ENS action, signs refund fields at the session's maximum rate, and chooses the refund recipient. `HCAOwnerAndSessionValidator` accepts the signature and the Paymaster transfers the HCA's payment token to that recipient. On a Sepolia fork, the unmodified patched-SDK payload moved 11,595,920 token units from the HCA to the recipient, and each fresh nonce repeats the debit until the session expires, is revoked, or the balance runs out. The refund recipient is not bound to the signed intent, so changing only the recipient leaves the signed amount intact. Every owner-configured ceiling on token, rate, expiry, revocation and nonce still holds; the owner-signed cap bounds the amount but does not confer the owner's principal, so the bypass lies in the mode classification alone. The session key's authority is widened beyond its intended scope, an unauthorized takeover of the HCA that lets the attacker transfer the victim's funds.

Proof of Concept

1. The runner installs `@rhinestone/sdk@1.8.0`, applies the monorepo's patch, and an offline oracle reports outer mode `0x0201` for the patched build with `verifyExecutions: true` against `0x0205` upstream.
2. The patched SDK generates a 586-byte scoped-session signature for an approval-only operation, with outer mode `0x0201`.
3. On a Sepolia fork at block 11,586,585, the SDK-derived HCA is deployed through the production factory and the fixed refund session is enabled.
4. The unchanged payload passes `HCAOwnerAndSessionValidator`, the IntentExecutor skips the callback, and the Paymaster moves 11,595,920 token units from the HCA to the chosen recipient.
5. Two fresh nonces produce two debits, and changing only the unsigned recipient preserves the signed refund amount.
6. Control: session mode `0x0206` reverts with `Unauthorized()` when the callback is absent, and a direct transfer, an over-limit rate and post-revocation use all revert.

IMM-WA-CRITICAL-91295

Case-folded address grouping lets an IOST-only delegate's value be shown and copied as the SOL receiving address

Report ID	#91295
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

A Manager profile shows a "Main receiving address" card followed by "Chain-specific addresses". The main card carries the icons of "every chain whose address resolves to the same value as the main address", and the chain-specific list holds the records that resolve to a different value ([ProfileView.helpers.ts#L246-L277](#)). Grouping is therefore only correct when the grouped records designate the same destination.

Manager requests address records in the order of `addressRecords`, which places a short preferred list first and sorts every other coin by coin type ([address.tsx#L42-L49](#)), and passes the resolved coins through in that order ([profileRecords.ts#L116-L130](#), [transformRecords.ts#L170-L177](#)). With no ETH record, the main address is `addresses[0]` ([ProfileView.helpers.ts#L229-L237](#)). The grouping helpers then compare values after lowercasing:

```
const mainAddress = getMainAddressCandidate(records)
if (!mainAddress) return []

const mainValue = mainAddress.value.toLowerCase()
return getAddressItems(records)
  .filter((address) => address.value.toLowerCase() === mainValue)
  .map(omitSourceIndex)
```

`getChainSpecificAddresses` applies the same projection to drop matching records from the chain-specific list ([ProfileView.helpers.ts#L268-L277](#)), and the main card's `CopyableButton` copies the main record's `address.value` while rendering the grouped chain icons ([ProfileAddressesSection.tsx#L90-L123](#), [ProfileAddressesSection.tsx#L159-L178](#)).

Lowercase equality is a valid shortcut for case-insensitive hex, but Base58 is case-sensitive, and the helpers apply it to every coin type. IOST (coin type 291) and SOL (coin type 501) are both 32-byte Base58 records, and `9c6hybhQ6Aycep9jaUnP6uL9ZYvDjUp1aSkFWPUFjtpj` (A) and `9C6hybhQ6Aycep9jaUnP6uL9ZYvDjUp1aSkFWPUFJtpj` (B) differ only in the case of two letters, yet decode to different valid Ed25519 points. Manager treats them as one destination.

An account holding only the IOST-291 address role on a `PermissionedResolver` cannot write coin type 501, but it can read the legitimate SOL value B, change two letters to obtain A, and write A as the IOST record. Because 291 sorts before 501 and the profile has no ETH record, Manager selects A as the main address, shows both the IOST and SOL icons on that card, hides the SOL card holding B, and copies A when the payer clicks Copy. A payer who relies on the SOL icon sends native SOL to A instead of B, while the SOL record itself stays unchanged. Control of A's private key is not claimed; the funds still go to a destination the name's owner never set as its SOL address. The application thus displays and pre-fills an incorrect address, so that a user sends funds to an unintended recipient.

Proof of Concept

1. Use a `PermissionedResolver` name with no ETH record and the legitimate SOL (501) address B.
2. Grant Mallory only `authorizeAddrRoles(name, 291, mallory, true)`; Mallory's `setAddr` for coin type 501 fails and B stays unchanged.
3. As Mallory, call `setAddr(node, 291, bytes(A))`.
4. Open the profile in Manager: the main card shows A with both the IOST and SOL icons, and no SOL card for B is rendered.
5. Click the main card's Copy button: the clipboard contains A, not B.
6. Control: with a SOL value that does not case-collide, the SOL card stays visible and copies its own value.

IMM-WA-CRITICAL-91335

Registry roles route applies Registry role labels to a `PermissionedResolver`, so a "Registrar" grant confers `ROLE_SET_ADDR`

Report ID	#91335
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Explorer's `/registry/$address/roles` route lists an ENSv2 registry's role holders and, for a caller holding `ROLE_REGISTRAR_ADMIN`, offers an Add User sheet that grants registry-wide roles. Its permission vocabulary is the Registry's: bit 0 is `ROLE_REGISTRAR`, labelled "Registrar" with the description "Can register new labels" ([permissions.ts#L90-L95](#)). The labels the user selects are expected to describe the authority the resulting transaction grants on the contract it targets.

The route is reached through a first-party link: a name's Roles page takes element zero of the registry-discovery result and links to `/registry/$address/roles` ([NameRegistryRolesOverviewTable.tsx#L37-L50](#)). The route queries the indexer's `registry(address)` record ([roles.tsx#L27-L31](#)) and shows "not a known registry contract" only when it is absent ([roles.tsx#L56-L67](#)). The record carries no implementation or contract-kind field ([useRegistry.ts#L17-L30](#)), and the code treats presence as proof of type:

```
// null = indexer has no record for this address (not a registry, or not yet
// indexed). Distinct from a registry with zero labels/roles.
return ok(registry)
```

When the caller passes the `ROLE_REGISTRAR_ADMIN` check ([roles.tsx#L37-L44](#)), Add User is rendered ([roles.tsx#L73-L91](#)), and `prepareGrantRegistryRolesTransaction` encodes the selected roles at `ROOT_RESOURCE = 0n` with `to` set to the route address ([grantRegistryRoles.ts#L45-L83](#)).

The route never verifies that the address is a Registry, however. A Portal-deployed `PermissionedResolver` exposes the same `EnhancedAccessControl` `grantRootRoles(uint256,address)` entrypoint but assigns bit 0 to `ROLE_SET_ADDR`. The app already contains an implementation-aware check, `getIsPermissionedResolver`, which compares the EIP-1967 implementation slot against `ensPermissionedResolverImpl`

([useIsPermissionedResolver.ts#L35-L67](#)) and is used on resolver screens, but `roles.tsx` does not reference it. Public operations are enough to put a resolver on this route: the attacker registers a decoy `.eth` name owned by the victim with the victim's resolver as its subregistry, discovery returns that address, and the indexer lists it as a registry.

When the victim, who holds the resolver's bit-128 admin role, adds the attacker as an operator of the decoy and ticks only "Registrar", Portal sends `grantRootRoles(1, attacker)` to the resolver. On a local Sepolia fork, the captured wallet request byte-equalled the mined input, the attacker's root `ROLE_SET_ADDR` changed from false to true, and the attacker's `setAddr` then redirected the victim's genuine name `vladz.eth` to the attacker's address through the unchanged resolver. The grant covers the resolver's root scope, so every name served by that resolver is exposed. The application displays one permission while the user approves a transaction that grants another, and names resolve to an attacker-controlled address.

Proof of Concept

1. On a local Sepolia fork, `vladz.eth` resolves through PermissionedResolver P to the victim, the victim holds P's bit-128 admin role, and an attacker `setAddr` reverts.
2. The attacker commits and registers a decoy `.eth` name owned by the victim with P as its subregistry, and the indexer writes a registry row for P from that block.
3. The decoy's Roles page shows a View link to `/registry/P/roles`, which renders P as a registry with Add User enabled.
4. The victim adds the attacker with only "Registrar / Can register new labels" selected and confirms once, and Portal reports "Transaction Complete".
5. The mined transaction to P decodes to `grantRootRoles(1, attacker)`, and the attacker's root `ROLE_SET_ADDR` changes from false to true.
6. The attacker's `setAddr` succeeds, and the Universal Resolver answer for `vladz.eth` changes from the victim to the attacker with the resolver unchanged.

IMM-WA-CRITICAL-91348

Unbound SIWE nonce lets an observer redeem a public Safe signature first and steal a seven-day JWT

Report ID	#91348
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Unauthorized takeover of a user's smart account (Hybrid Custody Account) or authenticated session, letting an attacker transfer the victim's names or funds or act as them, such as: adding, replacing, or escalating an account owner or co-owner · extending or widening a session key's authority beyond its intended scope or lifetime · stealing an authenticated session or bearer token · (Exploiting the 7-day session-key design as described in the Known Issues is not eligible; a weakness that grants authority beyond its stated lifetime or permissions is in scope.)

Description

Manager signs users in to the API worker with Sign-In with Ethereum: the client requests a nonce, builds a SIWE message for the connected account, signs it and posts the message, signature and nonce to `/auth/login` ([auth.ts#L30-L56](#)). The worker answers with a JWT that every `requireAuth` route accepts. The login is meant to hand that session to the client that requested the nonce and obtained the signature.

The worker stores the nonce as a bare key-value pair:

```
export const createNonce = (env: CloudflareBindings) => {
  const nonce = generateSiweNonce()

  return intoKVResult(
    env.KV.put(KV_KEY.AUTH.NONCE(nonce), nonce, {
      expirationTtl: 60 * 30, // 30 minutes
    }),
  ).map((_) => nonce)
}
```

`/auth/login` accepts `address`, `message`, `signature` and `nonce` and nothing else ([auth/index.ts#L22-L46](#)).

`createJWT` consumes the nonce, checks the SIWE domain and URI, verifies the signature (through ERC-1271 for a contract account), resolves the user and signs a JWT ([auth/index.ts#L74-L123](#)), which `signJWT` issues with a seven-day lifetime ([jwt.ts#L6](#), [jwt.ts#L41-L54](#)).

However, nothing binds redemption to the client that requested the nonce: the signed transcript is itself the credential, and whoever presents it first receives the session. For a Safe connected through WalletConnect,

that transcript is public before the dapp receives it. Safe Wallet's off-chain message flow stores the raw SIWE message and, once the threshold is met, exposes the completed `preparedSignature` through the credentialless Safe Transaction Service API, then waits `HIDE_DELAY = 3000` milliseconds before returning the signature to the dapp.

An unauthenticated observer who knows the victim's Safe address polls its public messages, copies `message`, `preparedSignature` and the nonce embedded in the message, and calls `/auth/login` first. The worker verifies the proof against the Safe, deletes the nonce and issues the observer the victim's seven-day JWT; the victim's identical request then fails with `INVALID_NONCE`. At the two credentialless requests per second that Safe documents, the poller gets about six reads within the three-second window. The victim only connects the Safe and approves the expected ENS sign-in message; the attacker needs no key, existing token, browser access or on-chain transaction. The `requireAuth` middleware checks only the HS256 signature and payload shape ([auth.ts#L7-L35](#)), so the stolen token works on the favorites, notifications, notification-channel, preference and transaction routes, and signing in again does not revoke it. This is an unauthorized takeover of the user's authenticated session through a stolen bearer token, letting the attacker act as the victim for seven days.

Proof of Concept

1. Deploy a Safe-compatible ERC-1271 wallet on local Anvil and obtain a nonce from the production `createNonce`.
2. Build the ENS SIWE message for the Safe and have its owner sign the Safe EIP-712 digest; the wallet-side and handler-side digests match.
3. Take only the `message` and `preparedSignature` pair, as the Safe Transaction Service exposes it, and parse the nonce from the message.
4. Call the production `createJWT` with the copied transcript; it returns a JWT that the production `verifyJWT` accepts with the victim's address and user ID.
5. Submit the same transcript as the victim: `createJWT` fails with `INVALID_NONCE`.
6. Controls: a modified message, a signature over the wrong Safe EIP-712 domain and a disallowed SIWE domain (`INVALID_DOMAIN`) are all rejected.

IMM-WA-CRITICAL-91982

Migration admits locked names with a custom resolver and **CANNOT_SET_RESOLVER** burned, leaving their records permanently unwritable

Report ID	#91982
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

The Manager migrates locked ENSv1 names into ENSv2 through the **LockedMigrationController**. Its own preflight documents the constraint on such names: "Locked migration cannot apply the app's resolver choice when CANNOT_SET_RESOLVER is burned. The controller replaces the old resolver only when PublicResolverSet certifies it, so prove that membership before asking the wallet to sign anything" ([migrationInvariants.ts#L161-L166](#)). The flow then presents the selected names as safe to move: "Your names are ready to upgrade" ([SelectNamesStep.tsx#L98-L100](#)), and the wallet step promises to "Migrate the selected names and restore their records" ([WalletConfirmationStepsDialog.tsx#L106-L108](#)).

The Manager re-exports the shared **@ens-apps/migration** classifier ([classifyNames.ts#L1-L11](#)), which assigns **keep-v1** to every transferable locked name with **CANNOT_SET_RESOLVER** burned, whatever its resolver, and does not treat a custom resolver as a reason for ineligibility. The membership check is limited by this predicate:

```
const requiresPinnedPublicResolverMembership = (
  name: ClassifiedName,
): name is ClassifiedName & { readonly v1ResolverAddress: Address } =>
  (name.tokenType === 'locked-2ld' || name.tokenType === 'locked-child') &&
  hasFuse(name.fuses, FUSES.CANNOT_SET_RESOLVER) &&
  isKnownPublicResolver(name.v1ResolverAddress)
```

[migrationInvariants.ts#L154-L159](#)

The locked-resolver record-safety inventory in **buildMigrationPlan** returns no candidate when **resolverFor** yields the V1 resolver unchanged ([buildMigrationPlan.ts#L160-L193](#)), which it does for a **keep-v1** name whose resolver is not a known public one ([encodeMigration.ts#L32-L40](#)). **lockedNameOwnerRoleBitmap**, which

mirrors `LockedWrapperReceiver`, adds `ROLE_SET_RESOLVER` only when that fuse is not burned ([buildAtomicMigrationBatches.ts#L366-L374](#)).

However, a custom owner-aware V1 resolver therefore never reaches `PublicResolverSet.includes`, produces no record-safety candidate, and leaves the name eligible. Such a resolver authorizes writes through V1 ownership, yet the migration sends the V1 wrapped token to the Graveyard, while the V2 owner receives no `ROLE_SET_RESOLVER`. That bitmap is safe for a known public resolver that accepts the new owner, but not for a preserved owner-aware resolver, and nothing in the flow warns about the combination.

A normal owner migrating such a name through the supported Manager flow keeps the ENSv2 token, and the existing records still resolve, but record updates and resolver replacement both revert. A stale address or contenthash therefore continues resolving indefinitely; renewal preserves the broken state, and expiry only makes the name publicly registerable rather than restoring control. No attacker transaction is involved. The migration leaves the name permanently inaccessible after the flow, bricking an ENS name during a Manager v1 to v2 migration.

Proof of Concept

1. A Manager unit test classifies a transferable locked `.eth` name with `CANNOT_UNWRAP` and `CANNOT_SET_RESOLVER` burned and a custom owner-aware resolver as eligible, with `resolverStrategy: 'keep-v1'`.
2. The resolver-membership invariant returns without reading `PublicResolverSet`, and the owner role bitmap lacks `ROLE_SET_RESOLVER`.
3. In the ENSv2 contracts, migrating that name through `LockedMigrationController` moves the V1 wrapped token from the user to the Graveyard, while the V2 token stays with the user.
4. The custom resolver and the old text record still read, but updating the record reverts.
5. Replacing the resolver reverts because `ROLE_SET_RESOLVER` is missing.
6. After renewal past the original expiry, the name keeps the same resolver and the same missing role.

IMM-WA-CRITICAL-92640

`useCanEditRecords` authorizes record edits by resolver role, letting a former owner edit a sold name in-app

Report ID	#92640
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/packages/smart-account
Impact	Taking state-modifying authenticated actions (with or without blockchain state interaction) on behalf of other users without any interaction by that user, such as: Changing registration information · Notification/settings changes

Description

The Explorer's `/$name/edit-records` route gives a connected wallet the record editor for a name, from which it saves `addr`, `text` and `contenthash` changes. Who may edit is decided by `useCanEditRecords`. Record-editing authority is expected to follow ownership of the name, and on non-permissioned resolvers the hook does fall back to token ownership ([useCanEditRecords.ts#L41](#)).

Registration ends the HCA reveal batch with `authorizeNameRoles('0x00', ROLES_ALL, wallet, true)`, a root grant over the per-HCA resolver ([registration-calls.ts#L321-L330](#)), and the transfer flow moves only the ERC-1155 token, without touching resolver roles ([buildTransferPlan.ts#L16-L43](#)). For a `PermissionedResolver`, `useCanEditRecords` queries the connected account's `ROLE_SET_*` roles and computes `isOwner` ([useCanEditRecords.ts#L75-L106](#)), then decides:

```
const canEdit = (() => {
  if (!enabled || !connectedAddress || isLoading || !resolverAddress) {
    return false
  }

  if (isPermissioned) return hasResolverRole

  // Public / non-permissioned resolvers: token owner can edit.
  return isOwner
})()
```

The edit route renders `EditRecordsContent` when `canEdit` is true and a "Permission Denied" message otherwise ([edit-records.tsx#L75](#), [edit-records.tsx#L143-L172](#)); `EditRecordsContent` saves through `useSaveRecords`, and `saveRecords.ts` builds the write with ensjs `setRecordsWriteParameters` ([saveRecords.ts#L127-L140](#)).

However, on a permissioned resolver `isOwner` never enters the decision: resolver roles alone grant the editor. The hook's own docstring notes that after a transfer that keeps the resolver, "the previous owner typically retains these roles while the new token owner does not" ([useCanEditRecords.ts#L35-L42](#)). Authorization therefore follows the retained roles, not the current owner, and a wallet holding the resolver-wide root grant passes this check for every name that resolver serves, whether or not it owns them.

After transferring a name with the resolver kept, which is a supported transfer option, the former owner opens the name's edit page, receives the full record editor and rewrites the records through the application, with no direct contract call. The new owner who opens the same page is refused permission to edit their own name and cannot correct the rewritten records in the app. On live Sepolia, the former owner of `bugtester.eth` rewrote its ETH address record to `0x00000000000000000000000000000000dEaD` after the buyer `0xd07e71Dec7A008A3A03677AC0f55dA561F2483Fc` had become the owner, in transaction `0x725e3f0036124d56c1ad3549fe556c963598afaf356527c07ab255fdc2057bc6`, while still holding root roles on the resolver. The former owner thus takes state-modifying authenticated actions on behalf of the new owner without any interaction by that user, changing the name's records and where it resolves.

Proof of Concept

1. The former owner registers `bugtester.eth` on Sepolia, receiving root resolver roles, and transfers it through the Explorer with the resolver kept; `ownerOf` now returns the buyer.
2. Connected as the former owner, open `/bugtester.eth/edit-records`: the record editor renders.
3. Edit the ETH address, save and sign; the record-write transaction succeeds.
4. Connected as the buyer, open the same page: the app shows "Permission Denied" for the buyer's own name.
5. Read the name: the owner is the buyer, `addr(60)` is `0x00000000000000000000000000000000dEaD`, and `hasRootRoles` for the former owner returns true.

IMM-WA-CRITICAL-92963

Profile editor diffs one name's records against another's, turning a Description-only save into a hidden attacker `setAddr`

Report ID	#92963
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immune-fi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions · Signing or submitting a transaction against the wrong chain (for example when the request's target chain is not validated) · Inducing a blanket token approval (for example <code>setApprovalForAll</code> on the registry or <code>NameWrapper</code>) to an attacker-controlled address

Description

The Manager profile editor writes the difference between the records it opened with and the current form values to a name's resolver. Opening the dialog resets both sides from the same `records`, calling `form.reset(records)` and `dialogActor.send({ type: 'OPEN', records })` together ([EditProfileDialog.tsx#L65-L70](#)). A save is correct only when the baseline, the edited values, the name and the resolver all describe one name.

The dialog seeds its XState actor with `records` and separately gives `records` to TanStack Form as `defaultValues` ([EditProfileDialog.tsx#L20-L22, #L39-L41](#)). The actor's `savedRecords` changes only on `OPEN` or `SAVE_SUCCEEDED` ([EditProfileDialog.machine.ts#L235-L236, #L286-L289](#)). `ProfileView` queries records for the live route name ([ProfileView.tsx#L93-L95](#)) with a zero `staleTime` ([router.tsx#L32-L36](#)), and the `$name` route renders the same unkeyed `ProfileView` when the parameter changes ([index.tsx#L165-L173](#)). On save, `useEditProfileDialogSave` sends `SAVE_REQUESTED` with the current route `name` ([useEditProfileDialogSave.ts#L381-L396](#)), and `getPendingSave` combines it with the retained snapshot:

```
params: {
  name: deps.name,
  before,
  after,
  signer: { type: 'eoa', walletClient },
  accountAddress: walletAccount.address,
  publicClient: deps.publicClient,
  chainId: deps.chainId,
  resolverAddress: savedRecords.resolverAddress,
```

[EditProfileDialog.machine.ts#L183-L191](#)

`buildRecordsUpdateCalls` diffs `before` against `after` and encodes every difference for `name` in one call to `resolverAddress` ([profileRecordTransactions.ts#L372-L456](#)).

However, nothing binds those four values to one name. An untouched form adopts refetched records, after a browser reconnect in the PoC, while `savedRecords` keeps the opening snapshot, and a browser Back swaps the route name under the still-open, dirty editor. The header shows the new name beside Save Profile ([EditProfileDialogHeader.tsx#L60-L82](#)), and the body uses the diff only to enable saving ([EditProfileDialogBody.tsx#L74-L97](#)), without listing record changes.

An attacker keeps `ROLE_SET_RESOLVER` on a SOURCE name gifted to the victim and pointed at TARGET's `PermissionedResolver` with no ETH record. While the victim's SOURCE editor is open, the attacker re-points SOURCE to a resolver whose ETH record is the attacker, and on reconnect the hidden form value becomes that address. The victim edits only Description, presses Back to TARGET and saves; the wallet request carries both `setText(TARGET, "description", ...)` and `setAddr(TARGET, 60, attacker)`, although the attacker holds no TARGET role. After mining, TARGET resolves to the attacker, Manager shows that address as its "Main receiving address", and a payer's 0.001 ETH reaches the attacker and survives the owner's later repair. Manager has modified the arguments of a transaction the connected wallet signs, a malicious interaction with an already-connected wallet.

Proof of Concept

1. On Anvil, TARGET is created for the victim with its ETH record set to the victim; the attacker holds zero TARGET roles and a direct write to TARGET's resolver is rejected.
2. SOURCE, pointed at TARGET's `PermissionedResolver` with no ETH record, is gifted to the victim while the attacker keeps a SOURCE-only `ROLE_SET_RESOLVER`.
3. The victim visits TARGET, then SOURCE, and opens SOURCE's editor on General with Save Profile disabled and no address field mounted.
4. With the browser offline, the attacker re-points SOURCE to a resolver whose ETH record is the attacker; after reconnect, Save Profile becomes enabled with no visible change.
5. The victim edits only Description, presses Back once, sees TARGET in the header and clicks Save Profile; the captured request contains `setText(TARGET, "description", "Updated public profile")` and `setAddr(TARGET, 60, attacker)`.
6. After mining, ensjs and `UniversalResolverV2` resolve TARGET to the attacker, and a payer copying Manager's "Main receiving address" sends it 0.001 ETH.
7. The victim's repair restores TARGET's resolution, while the payment remains in the attacker's balance.

IMM-WA-CRITICAL-93026

Default transfer plan detaches the registry, bricking third-party subnames and letting the new holder re-mint them

Report ID	#93026
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

In the Explorer, every subname under a v2 parent resolves through the parent's subregistry pointer, and the subname's ERC-1155 token lives inside that subregistry. The registry view offers to deploy or set a registry only for a name whose subregistry slot is empty: `RegistryTreeItem` renders `ConfigureRegistryForm` only when the last level's address is `zeroAddress` ([RegistryTreeItem.tsx#L54-L57](#), [RegistryTreeItem.tsx#L184-L210](#)). Because the registry contract itself permits replacing an occupied slot, this gate is what keeps a live subregistry, and the subname tokens in it, from being swapped out through the UI.

The "Send name" form starts with all three plan options enabled:

```
const [options, setOptions] = useState<Record<OptionKey, boolean>>({
  setEthAddress: true,
  detachResolver: true,
  detachRegistry: true,
})
```

`buildTransferPlan` turns `detachRegistry` into a `detach-registry` step that runs before the token transfer ([buildTransferPlan.ts#L22-L43](#)), and `detachNameRegistry` encodes `setSubregistry(label, 0x0)` on the parent registry ([detachNameRegistry.ts#L66-L70](#)). A seller only has to enter a recipient and confirm.

The detach, however, clears the only pointer to subnames that may belong to independent third parties, and it re-opens the empty-slot gate for whoever holds the parent next. The subname holder cannot repair it: `ConfigureRegistryForm` checks `ROLE_SET_SUBREGISTRY` on the parent label in the parent registry ([ConfigureRegistryForm.tsx#L89-L101](#)), a role that moves to the recipient with the parent token. After the transfer, the recipient's Portal renders "Configure registry" with the deploy option preselected

([ConfigureRegistryForm.tsx#L67](#)), and two transactions point the parent at a fresh, empty registry in which the victim's label is unregistered.

A routine, default-configured transfer thus makes every third-party subname under the parent stop resolving, while each token remains with its owner inside a registry nothing references. The new holder, or a seller who transfers to a second address they control, can then register the victim's label in the new registry to any address, so the name resolves to that address and the original token cannot resolve again unless the parent holder repoints. On a Sepolia fork pinned at block 11,616,799, one default transfer orphaned two independently owned subnames under `shark-cat.eth`, and the buyer, who held no role before the transfer, re-minted `example.shark-cat.eth` to an address of their choosing. The transfer flow hijacks a name held by someone outside it and leaves the owner's original name permanently inaccessible.

Proof of Concept

1. On a Sepolia fork, the seller runs the shipped "Send name" flow with defaults untouched; the plan lists Detach resolver, Detach registry and Transfer name, and all three transactions mine.
2. `getSubregistry` for `shark-cat` returns `0x0`, and `/example.shark-cat.eth` renders "Name not found" while the old registry still shows the victim as the token owner.
3. The victim's own attempt to restore the subregistry reverts with `EACUnauthorizedAccountRoles` for `ROLE_SET_SUBREGISTRY`.
4. In the buyer's Portal instance, "Configure registry" appears only after the transfer, and two UI transactions deploy a fresh registry and point the parent at it.
5. The buyer registers `example` in the new registry to a chosen receiver, as a contract-level call because the UI's indexer-fed availability check lags on the fork.
6. The profile page now resolves to the receiver; the new-registry owner is the receiver and the old-registry owner is still the victim.
7. Control: in the old registry the victim's label stays taken, and registering it reverts with `LabelAlreadyRegistered("example")`.

IMM-WA-CRITICAL-93081

Resolver roles sidebar turns an unresolved role scope into the root sentinel, widening a one-name grant to every name

Report ID	#93081
Severity	Critical
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Taking state-modifying authenticated actions (with or without blockchain state interaction) on behalf of other users without any interaction by that user, such as: Changing registration information · Notification/settings changes

Description

The Explorer's resolver roles page (`/resolver/$address/roles`) lists the accounts holding roles on a resolver and opens `ResolverRolesSidebar` to edit one account's permissions. Grants are either name-scoped or global, and the save helper encodes that choice in its `name` parameter: "Dotted name (e.g. "myname.eth") or empty string for ROOT_RESOURCE (all names)" ([grantResolverRoles.ts#L18-L22](#)). An edit to a role scoped to one name is meant to stay on that name.

`ResolverRolesTable` groups the indexer's role rows with `groupRolesByAccount(roles, buildResourceToNameMap(nodes))` ([ResolverRolesTable.tsx#L103](#)). The map hashes each name in the resolver's `nodes` list with `computeNameResource`, and a role's name joins `resolvedNames` only when its `resource` is found in that map ([resolverRoles.ts#L147-L160](#), [#L193-L207](#)). The sidebar then derives the scope:

```
const resolvedNames = row?.original.resolvedNames ?? []

const roleName = resolvedNames.find((n) => n !== '(root)') ?? ''
```

[ResolverRolesSidebar.tsx#L167-L169](#)

`handleSaveChanges` passes `roleName` as the grant's `name` ([ResolverRolesSidebar.tsx#L295-L303](#)), and `prepareGrantResolverRolesTransaction` sends an empty name to `scope: 'root'` ([grantResolverRoles.ts#L45-L61](#)), for which the pinned ensjs builder emits `grantRootRoles`.

However, `resolvedNames` is empty whenever the role's resource cannot be inverted from the node list, whether through indexer lag, a name missing from the unpaginated `nodes` selection, or a hash the map cannot reverse, and the `?? ''` fallback then yields exactly the sentinel the save path reads as root. The indexer already returns each role's `name` ([useResolverOverview.ts#L14-L22](#)), but the grouping never reads it. The sidebar also renders its scope caption only when `resolvedNames` is non-empty

([ResolverRolesSidebar.tsx#L391-L397](#)), so the operator sees neither "Global roles (all names)" nor the intended name.

An operator holding the resolver's admin bits who adds a role to such a row signs `grantRootRoles`, and the grantee receives that role across every name the resolver serves. In the PoC, on a `PermissionedResolver` deployed through the genuine `VerifiableFactory`, the grantee could not write `victim.eth` before the grant and afterwards set its `url` text record to an attacker value, although the operator never selected that name. On a resolver serving names of several owners, those owners never took part in the grant. The grantee is thereby taking state-modifying authenticated actions on behalf of other users without any interaction by those users.

Proof of Concept

1. On a Sepolia fork, a `PermissionedResolver` is deployed through the genuine `VerifiableFactory` with the operator as `ALL_ROLES` admin, and the grantee's `setText` on `victim.eth` fails.
2. The grantee's `ROLE_SET_TEXT` role on `alice.eth` is grouped by `groupRolesByAccount` against a node list that lacks `alice.eth`, yielding `resolvedNames: []` and `roleName: ""`.
3. `prepareGrantResolverRolesTransaction` and the pinned ensjs builder select `grantRootRoles`.
4. The operator broadcasts that call, and the grantee's root roles and its role on the resource of `victim.eth` both flip from `false` to `true`.
5. The grantee calls `setText` on `victim.eth`, and its `url` record reads back `https://attacker.example`.
6. Control: with `alice.eth` present in the node list, the app selects `authorizeNameRoles` and the grantee can neither reach nor write `victim.eth`.

IMM-WA-HIGH-89328

Manager checkout total and signed funding permit come from separate unreconciled quotes, so the chain charges more than displayed

Report ID	#89328
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

On the Manager's USDC registration path through the Hybrid Custody Account, the wallet funds the HCA by EIP-2612 permit with a budget of rent plus the commit and register leg costs. The checkout states the invariant: its budget query "Mirrors what `registrationUi.machine.ts` hands the machine for `START_REGISTRATION`, so the figure on this screen is the one the permit is sized from" ([TokenPickerContent.tsx#L148-L154](#)).

When the checkout's budget quote is absent or fails, `displayTotal` falls back to `pricingData`, the rent alone ([TokenPickerContent.tsx#L434-L438](#)), and the "up to" hedge disappears. `START_REGISTRATION` carries no `hcaBudget` ([registrationUi.machine.ts#L544-L562](#)), so `computingHcaBudget` quotes the budget again and `signingFundingPermit` signs for that result ([registration.machine.ts#L721-L725](#)).

However, no budget value crosses from the screen to the machine, and nothing compares the signed `value` with the figure displayed. The only check before signing is the wallet-balance preflight, whose own comment records that "on Sepolia an 8.00 USDC name has run to a ~20.20 USDC budget" ([registration.hca.actors.ts#L621-L637](#)).

In the PoC on the repository's local e2e stack, with the checkout's quote requests aborted and the sign-time quote returning 20.00 USDC per leg, the modal showed `Total $16.52 USD` with no hedge, while the wallet's USDC fell by 56.52 USDC, 3.4 times the displayed total, through a normal success screen. The per-leg figure was injected because the local mock orchestrator quotes zero fees. The application thus shows the wrong amount on a transaction the user then approves.

IMM-WA-HIGH-89329

Uncapped funding permit lets an inflated orchestrator quote pull the owner's full USDC balance on owner-signed HCA actions

Report ID	#89329
Severity	High
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/packages/transaction-manager
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions · Signing or submitting a transaction against the wrong chain (for example when the request's target chain is not validated) · Inducing a blanket token approval (for example setApprovalForAll on the registry or NameWrapper) to an attacker-controlled address

Description

Standalone-HCA actions after registration, such as setting a primary name, pay their fee in USDC out of the HCA. The funding helper exists to "quote the fee, and when the HCA cannot cover it, pull exactly the shortfall in from the owner's wallet with an EIP-2612 permit + `transferFrom` pair" ([hca-intent-funding.ts#L10-L16](#)).

`planHcaIntentFunding` obtains `quotedFeeUsdc` from the orchestrator's `prepareTransaction` response ([hca-intent-funding.ts#L120-L137](#)) and signs a permit for the difference against the HCA balance ([hca-intent-funding.ts#L255-L302](#)).

However, the planner applies no ceiling to the quoted fee. The only check before the wallet signs is the preflight `walletBalance < input.value` ([registration.hca.actors.ts#L627-L637](#)), which rejects a permit the wallet cannot cover, so the permit's effective bound is the wallet's entire USDC balance.

`setPrimaryNameWithHca` runs owner-signed, on a path whose own comment says it "applies no action policy at all — which is also what makes the funding permit legal here" ([setPrimaryName.ts#L181-L198](#)). In the end-to-end PoC with a real EIP-2612 USDC, a quote equal to the balance produced a signed permit for 5,000 USDC and moved the victim's full 5,000 USDC to the relayer. Exploitation requires the orchestrator's quote to be malicious, compromised, intercepted or faulty, and the wallet prompt shows the permit value with the user's own HCA as spender. An inflated quote thus modifies the arguments of the permit the already-connected wallet signs, authorizing far more than the action costs.

IMM-WA-HIGH-89462

HCA registration budget omits the first-registration resolver deployment, leaving the reveal underfunded and unrecoverable

Report ID	#89462
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/packages/smart-account
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

A standalone-HCA registration funds itself once: the commit leg carries a single EIP-2612 permit sized by `estimateHcaBudget`, and the reveal leg pays the registrar from that balance. The budget's comment states the requirement: the orchestrator prices the reveal "purely on `destinationGasUnits`", so "an under-sized limit under-funds the permit and the fill then fails for insufficient USDC" ([budget.ts#L108-L119](#)).

That gas number comes from `registerLegGasLimit`, which returns a fixed 450,000 ([budget.ts#L46-L53](#)) plus an allowance for a primary name only. `buildRevealBatch` prepends `VerifiableFactory.deployProxy` for the account's `PermissionedResolver` whenever `resolverDeployed` is false, which holds on every account's first registration ([registration-calls.ts#L248-L267](#)).

However, `registerLegGasLimit` takes no `resolverDeployed` input, so the funded gas ignores the largest conditional call. On a Sepolia fork, the first-registration batch executed at 455,422 gas before the account and validator overhead a real fill adds.

The user signs the permit, the commit lands and is paid for, and the reveal is funded short. The reveal batch contains no `permit` or `transferFrom` to top the account up, and every reveal failure resubmits the same batch ([registration.machine.ts#L1204-L1210](#)). No retry re-runs the budget or the permit, so every attempt meets the same shortfall while the commitment ages out. An account's first registration therefore ends with a paid commitment that cannot be revealed and a name that is never registered, bricking the name's Manager registration flow.

IMM-WA-HIGH-90749

Resolver-role editor reparses a flat dotted label as a hierarchical name, granting roles on a different name

Report ID	#90749
Severity	High
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Subdomain takeover without already-connected wallet interaction

Description

The Explorer's `ResolverAddUserSheet` lets a resolver admin grant roles on one name, and its confirmation reads "Grant resolver roles for" the selected name (`ResolverAddUserSheet.tsx#L321`). The transaction it builds must authorize the object the admin selected and confirmed.

The sheet reduces every node to its display string before selection and hands that name to `prepareGrantResolverRolesTransaction` (`ResolverAddUserSheet.tsx#L327-L334`), which passes any non-empty string to ENSJS `grantResolverRolesWriteParameters` with `scope: 'name'` (`grantResolverRoles.ts#L45-L61`).

However, a display string is not a reversible identity. A single raw label `alpha.beta` registered under `.eth` appears as `alpha.beta.eth`, the same spelling as the hierarchical name `alpha` under `beta.eth`, and the encoder treats every dot as a label boundary. Once the sheet keeps only `n.name`, nothing remains from which to encode the flat object, so the builder always emits the hierarchical packet.

An attacker registers the flat label `alpha.beta`, assigns it to a victim who owns the hierarchical `alpha.beta.eth`, points it at the victim's `PermissionedResolver`, and asks for Set Address Manager on the delivered name. The victim confirms one ordinary Add User flow. In a run of the real Explorer UI on a local chain with a reproduced indexer response, the captured calldata was the three-label packet of the victim's name. The attacker gained `ROLE_SET_ADDR` on that name and her `setAddr` made it resolve to her address. The victim can revoke the role once noticed, but nothing in the flow shows that the grant landed elsewhere. The attacker takes over the victim's name's address record with no further action from the victim's wallet, a subdomain takeover without already-connected wallet interaction.

IMM-WA-HIGH-91441

Unverified resolver events let any contract inject persistent text into another name's Explorer history

Report ID	#91441
Severity	High
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Injecting/modifying the static content on the target application without JavaScript (persistent), such as: HTML injection without JavaScript · Replacing existing text with arbitrary text · Arbitrary file uploads, etc

Description

The Explorer's history page presents the actions taken on a name. For resolver events, `fetchV1NameHistory` states it reads events "from every resolver the name has ever pointed at" ([fetchV1NameHistory.ts#L40-L45](#)). The page therefore relies on every returned resolver row belonging to a resolver the Registry assigned to that name.

The fetcher queries `resolvers(where: { domain: $id })` and each row's events, including `TextChanged { key value }` ([fetchV1NameHistory.ts#L101-L119](#)), and `adaptV1Events` sets the address from `resolverId` as the event's `contractAddress` ([adaptV1Events.ts#L258-L276](#)). The timeline renders the event as `Set text record` with its key and value ([descriptors.ts#L83-L94](#)).

However, the ENSNode `0.36.0` indexer creates a `Resolver` row for any contract that emits a resolver event signature, keyed on the emitter and the `node` it supplies, with no check of Registry assignment. The Explorer never compares the emitter with the Registry's resolver for the node, so a row created by an unrelated contract is rendered as a canonical text-record change on the victim's name.

An attacker who is neither owner, operator nor resolver of the name deploys a contract that emits the four-argument `TextChanged` event with the victim's namehash. In the PoC, the history page of `purple.tomato.eth` showed `Set text record com.twitter → FORGED_WITHOUT_RESOLVER_AUTHORITY` with the emitter labelled `resolver`, while the Registry resolver and the legitimate `text(node, "com.twitter")` value stayed unchanged. The content remains while the event stays inside the returned history window, and React escapes it as plain text. Any contract can thus inject persistent static content, without JavaScript, into the history of a name it has no authority over.

IMM-WA-HIGH-92196

Portal keys pending address-record edits by display symbol, so deleting an alias row clears the canonical Base record

Report ID	#92196
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions · Signing or submitting a transaction against the wrong chain (for example when the request's target chain is not validated) · Inducing a blanket token approval (for example setApprovalForAll on the registry or NameWrapper) to an attacker-controlled address

Description

The Explorer's Edit Records page saves a name owner's address-record edits and deletions as one resolver transaction, whose encoded slot must equal the numeric slot the user selects.

`recordsToTableData` keys each address row by the encoder's display `symbol`, keeping the numeric coin type only as `id` ([recordsToTableData.ts#L36-L40](#)); on save, the transformer resolves each deleted row's key-derived identity back to a record and emits its `id` as the coin to clear ([transformPendingChanges.ts#L120-L130](#)).

However, the display symbol is not unique per coin type. Because the pinned `@ensdomains/address-encoder` derives chain ids with 32-bit operators, the Base Sepolia coin type `2147568180` and the alias `6442535476` both display as `Unknown Chain (84532)` while remaining distinct resolver slots. Both rows share one identity, and `find` returns the first, so when the canonical record precedes the alias, deleting the alias clears the canonical Base record.

A prior owner of a wrapped V1 name can arrange this before transferring it by writing, in order, a default EVM record pointing at an attacker address, the Base record, then the alias; `mergeCoinTypes` preserves the indexer's first-write order ([mergeRecordKeys.ts#L17-L27](#)). When the new owner deletes the `Unknown Chain (84532)` row, the wallet signs `setAddr(node, 2147568180, 0x)`. In three fork runs, `forwardAddress` then returned the attacker's default address as the name's Base address. The owner can restore the record with one write once the change is noticed. The application thus modifies the coin-type argument of the transaction the user approves.

IMM-WA-HIGH-92413

Unpinned `settlementLayers` in the Warp send lets a substituted route session-sign an origin transfer that drains the HCA

Report ID	#92413
Severity	High
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/packages/transaction-manager
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions · Signing or submitting a transaction against the wrong chain (for example when the request's target chain is not validated) · Inducing a blanket token approval (for example <code>setApprovalForAll</code> on the registry or <code>NameWrapper</code>) to an attacker-controlled address

Description

On the Standalone-HCA flow, the reveal leg is a Rhinestone intent signed by the ephemeral session key, with no wallet prompt ([warp-transport.actor.ts#L107-L121](#)). The session's authority is bounded by the on-chain validator's registration policy, whose payment-token branch allows only `approve` ([DEBUGGING_INTENTS.md#L166-L168](#)).

`submitRevealBatchActor` wraps the reveal calls with `buildUserPaidRequest`, which sets nothing about settlement ([registration.hca.actors.ts#L993-L1010](#)).

However, the Warp send never pins `settlementLayers` and never compares the returned settlement layer, `destinationOps` or `preClaimOps` against the reveal calls it built. A route whose Permit2 element's `preClaimOps` call `MockUSDC.transfer(attacker, 19_000_000)` is signed by the session key as returned, and the validator's policy walks only the destination operations, so the transfer is never checked against the allowlist.

On a Sepolia fork, that signature moved 19.00 USDC from the HCA to the attacker. The amount is bounded only by the HCA's USDC balance at reveal, and the consumed Permit2 nonce also leaves the name unregistered. The PoC fed the SDK such a route's `intentOp` without altering the live orchestrator. A substituted route therefore modifies the arguments of a transaction signed on the user's behalf, and the account submits a malicious transfer the user never reviewed.

IMM-WA-HIGH-92461

Registration actor survives a name change, so the Explorer charges an earlier name's price under the new name's total

Report ID	#92461
Severity	High
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

On the Explorer's `/register` route, the form, checkout summary and payment section derive from the `name` prop read from the search parameter ([RegisterName.tsx#L160-L179](#)), so the name and total on screen are meant to be the name the approved sequence registers and the amount it charges.

`useRegistrationTransactions` creates the actor with only the chain as input ([useRegistrationTransactions.ts#L80-L82](#)). The machine copies `name` and price into its context only when it accepts `START_REGISTRATION` ([registration.machine.ts#L537-L560](#)), and later steps build the register transaction from that context ([registration.machine.ts#L1379-L1398](#)). A navigation prompt warns only that leaving may interrupt a running registration ([RegisterName.tsx#L67-L76](#)).

However, the actor is not keyed to the current name, and a same-route change of the `name` parameter neither cancels nor recreates it. Once the warning is accepted, the page renders the new name, its price and `Registering...`, while the running machine still holds, and later submits, the earlier name.

In the PoC on a Sepolia fork, a registration started for `qzm2.eth`; after searching for `low-price-mturyzm2.eth`, the page showed that name with `Total $8.01`, and approving the wallet requests registered `qzm2.eth` and debited 160.109598 USDC. The flow requires the user to accept the navigation warning and three wallet authorizations, and the final calldata carries the plaintext label `qzm2`. The application thus shows the wrong name and amount on a transaction the user then approves.

IMM-WA-HIGH-92540

Unscoped `useRoleHistory` query lets any registry owner forge entries in another name's role history

Report ID	#92540
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Injecting/modifying the static content on the target application without JavaScript (persistent), such as: HTML injection without JavaScript · Replacing existing text with arbitrary text · Arbitrary file uploads, etc

Description

The Explorer's `/$name/roles` screen opens a `RolesSidebar` whose **History** table lists every role grant and revocation recorded for a role holder on the name, read by the name's own admin. The sibling hook `useNameRoleAccounts` states the invariant: a resource-only filter "would conflate role grants across unrelated registries", so role events must be scoped with `events(contractAddress:)` ([useNameRoleAccounts.ts#L56-L60](#)).

The table's rows come from `getRoleHistory`, which pages through `EACRolesChanged` events selected by type and block cursor only, then keeps those matching a `resource` computed as `toResourceHex(labelToCanonicalId(label))` from the leading label, sorted newest-first ([useRoleHistory.ts#L84-L86](#)).

However, `labelToCanonicalId` depends on the label string alone, and the query carries no `contractAddress`. Any registry holding a subname with the same leading label emits `EACRolesChanged` at the identical resource, and the panel accepts those events as the victim name's history.

An attacker who owns any `.eth` name deploys her own registry, creates a subname carrying the victim's label, and grants and then revokes roles on her own address. On a Sepolia fork, the app's filter placed the forged revocation as the newest row for the attacker's account under the victim's name, while `hasRoles(ROLE_SET_RESOLVER)` for that account read `true` before and after. The attacker chooses every field of a forged row, and the rows persist as real chain events. The result is persistent, attacker-authored content injected without JavaScript: the admin's permission audit trail reports a revocation that never happened.

IMM-WA-HIGH-92665

Orchestrator-returned HCA registration is not checked against the requested calldata, installing an attacker-controlled subregistry

Report ID	#92665
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/packages/transaction-manager
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

Manager's HCA reveal builder requests `ETHRegistrar.register` with a zero subregistry, so a newly registered name starts with no child registry. Before the commit, the budget estimate sends the orchestrator a reveal-shaped quote carrying the label ([registration.hca.actors.ts#L391-L431](#)).

However, neither the session's destination salt nor the registration policy binds the label, secret or subregistry, and Manager never compares the returned executions with the calls it requested:

`verifyExecutions` checks them only against the policy. `ETHRegistrar.makeCommitment` does not bind `msg.sender`, and the HCA owner is publicly readable, so a second commitment for the same label and victim owner can carry another secret and subregistry.

A Byzantine or faulty orchestrator that learns the label from the quote commits that tuple, then returns it in place of Manager's reveal once both commitments mature. The session signs it, the policy accepts the unchanged owner and resolver, and the name is registered with the attacker's registry as its subregistry. `verifyHcaRegistrationActor` checks only status, owner and resolver, so Manager reports success. In a Foundry run through the real HCA validator with a mock intent executor, the attacker then creates and controls `pay.<name>`. The victim can later repair the pointer, but the registration assigns the victim's name's registry to an attacker, hijacking an ENS name during a Manager registration flow.

IMM-WA-HIGH-92779

Owner-signed HCA intents carry no execution verification, so the wallet signs whatever executions the orchestrator returns

Report ID	#92779
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/packages/transaction-manager
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions · Signing or submitting a transaction against the wrong chain (for example when the request's target chain is not validated) · Inducing a blanket token approval (for example setApprovalForAll on the registry or NameWrapper) to an attacker-controlled address

Description

Manager submits standalone-HCA actions as Rhinestone intents: the SDK sends the app's calls to the orchestrator, which returns the operation the wallet signs. On the session path the transport requests `verifyExecutions: true`, which binds the signed executions to the on-chain validator's registration policy.

With no session, `sdkParams` omits `signers` entirely ([warp-transport.actor.ts#L163](#)), and the app calls `account.sendTransaction`, which prepares, signs and submits in one step.

However, the owner path has no execution-equivalence check. The app never compares the orchestrator's executions with the requested `calls`, and in the pinned `@rhinestone/sdk 1.8.0`, `verifyExecutions` applies only to a session signer, so the owner signs the operation the orchestrator returned; the deployed `HCAOwnerAndSessionValidator` skips `_checkRegistrationPolicy` for a 65-byte owner signature.

The owner path is reached in ordinary operation: before a session exists, after one expires, and whenever the owner-mismatch guard drops it. On that path a malicious or compromised orchestrator can return executions that differ from the requested registration, such as a transfer of the HCA's assets, and the user's owner signature authorizes them; no substituted intent was filled end to end. This is a malicious interaction with an already-connected wallet: the wallet signs and submits transaction arguments other than the ones the app requested.

IMM-WA-HIGH-92833

Manager's third-party V1 renewal leaves a wrapped name's `NameWrapper` expiry stale, letting a stranger brick it

Report ID	#92833
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Taking state-modifying authenticated actions (with or without blockchain state interaction) on behalf of other users without any interaction by that user, such as: Changing registration information · Notification/settings changes

Description

Manager lets a non-owner renew someone else's name. `ProfileRenewAction` shows a `ThirdPartyRenewalDialog` when the viewer is not the owner, and the dialog promises that renewing "will add time to the current owner's registration" ([ThirdPartyRenewalDialog.tsx#L65-L69](#)).

For a V1 name the dialog routes to `/renew-v1/$name`, and the renewer is chosen from the protocol version alone. The renewal machine passes it to `submitRenewActor`, which sends `renew(label, duration, paymentToken, referrer)` from the payer's wallet ([registration.actors.ts#L1020-L1031](#)).

However, this choice does not distinguish an unwrapped V1 registration from one held by `NameWrapper`. For a wrapped name, the V1 renewal advances the BaseRegistrar expiry but leaves the NameWrapper expiry, which governs the wrapped owner and fuses, unchanged, and Manager displays the renewed registrar expiry.

An unrelated wallet can therefore pay Manager's 100-year maximum on a victim's wrapped name, with no action from the victim. On a Sepolia fork, the renewal moved the wrapper expiry by zero; once it passed, `NameWrapper.ownerOf` returned the zero address, the victim's `setResolver` call reverted, and the registration stayed unavailable with 36,434 paid days remaining. Acquisition attempts from the payer's wallet all reverted, so the payer gains nothing while the victim loses the name for the paid term. This is a state-modifying authenticated action taken on behalf of another user without any interaction by that user, changing the name's registration information.

IMM-WA-HIGH-93029

Manager writes L2 address records to mainnet coin types the Explorer never reads, so L2 rows show the default address

Report ID	#93029
Severity	High
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Explorer's address page lists the address a name resolves to on each network, including five L2 rows. Each row is keyed on an ENSIP-11 coin type, and because the Explorer runs against Sepolia, L2 coin types derive from the testnet chain ids ([networks.ts#L12-L17](#)). The Manager's Addresses tab is where an owner sets those per-network records, so the two apps must agree on each network's coin type.

The Explorer builds its rows with `getCoinTypeForReverseRegistrarChainId(chainId, 'sepolia')` ([networks.ts#L29-L50](#)), while the Manager offers Optimism, Base and Arbitrum through the `evmChainOptions` list ([addressPickerRecords.ts#L21-L28](#)) and Linea and Scroll through the chain picker ([address.tsx#L5-L12](#)). When a row has no record, `forwardAddress` returns the ENSIP-19 default record ([address.tsx#L58-L67](#)).

However, every coin type the Manager offers for these five networks decodes to a mainnet chain id (Base is `2147492101`, chain 8453), while the Explorer reads the Sepolia slot (Base is `2147568180`, chain 84532). A record written through the Manager therefore lands in a slot the Explorer never reads, and each L2 row falls back to the default address with no sign that it is a fallback.

In the PoC, a sender paid the Base row's displayed address 2 ETH on the L2, and the funds could not be recovered: the address was the owner's smart account, with no code and no private key on the L2. The loss requires a chain-specific L2 record that differs from the owner's EVM address, which the tab's network chip cannot create from a clean profile, and a contract rather than an EOA as the default record. The Explorer thus renders an incorrect address for the name, and the sender pays an unintended recipient.

IMM-WA-HIGH-93033

Address history page trusts unconsented registry ownership, letting anyone plant attacker-authored rows and a forged sender

Report ID	#93033
Severity	High
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Injecting/modifying the static content on the target application without JavaScript (persistent), such as: HTML injection without JavaScript · Replacing existing text with arbitrary text · Arbitrary file uploads, etc

Description

The Explorer gives every Ethereum address a public history page at `/addr/<address>/history`, presented as that address's own ENS history.

The v1 leg, `getV1HistoryForAddress`, asks the subgraph for the domains whose `owner` or `registrant` is the address, then reads each domain's registry, registration and resolver events. The resolver leg selects `AuthorisationChanged { owner target isAuthorized }`, whose `owner` is a bare string (`useV1HistoryForAddress.ts#L162-L166`). An expanded row shows `extractFromAddress(eventDetails) ?? row.original.from` as the sender (`EventsTableRow.tsx#L27-L29`), and `extractFromAddress` returns `owner` whenever it is a string.

However, registry ownership is not consent. The ENS registry's `setSubnodeRecord` is authorised on the parent only, so the owner of any name can create a subname whose owner is the victim's address and whose resolver the attacker deployed. The page never checks that the address created or accepted any domain it lists, and the string `AuthorisationChanged.owner` replaces the transaction's real sender.

The attacker fills the victim's page with rows emitted by her own resolver, and one `AuthorisationChanged` event prints an address of her choice in the "From" column. The victim signs nothing and needs no ENS name; on the deployed Explorer, a victim address went from "No history yet" to a page of attacker-written rows. Because the domain query's `first: 100` carries no `orderBy`, the subgraph returns the 100 lowest namehashes, so grinding labels below the victim's genuine domains evicts the real history. The rows persist until each planted subname is burned or transferred. This is persistent replacement of the application's displayed content with arbitrary attacker text, without JavaScript.

IMM-WA-HIGH-93046

Unchecked **register** subregistry lets an HCA session signer retain control of the owner's subname resolution

Report ID	#93046
Severity	High
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/packages/smart-account
Impact	Unauthorized takeover of a user's smart account (Hybrid Custody Account) or authenticated session, letting an attacker transfer the victim's names or funds or act as them, such as: adding, replacing, or escalating an account owner or co-owner · extending or widening a session key's authority beyond its intended scope or lifetime · stealing an authenticated session or bearer token · (Exploiting the 7-day session-key design as described in the Known Issues is not eligible; a weakness that grants authority beyond its stated lifetime or permissions is in scope.)

Description

In the Standalone-HCA flow, a registration session key is meant to register a name to the account owner with the session-bound resolver. The client builds the reveal's **ETHRegistrar.register** call with the wallet as owner and **ZERO_ADDRESS** as **subregistry** ([registration-calls.ts#L280-L298](#)).

What a session key may register is decided not by the client but by the **HCAOwnerAndSessionValidator** the client is configured against ([manifest.ts#L105](#)), whose registration policy requires each **register** owner to equal the account owner and every **register** in the batch to use the allowed resolver.

However, the validator reads only the owner and resolver words of the **register** calldata and never reads the **subregistry** word. A session signer can therefore reveal a registration that keeps the legitimate owner and resolver but supplies an attacker-controlled registry, and the validator approves it.

On a Sepolia fork, the injected registry pointer made **pay.injectparent.eth** resolve to an attacker-selected address, a 1 ETH payment to that name reached it, and the pointer survived both session revocation and expiry. The attack requires a valid registration session the owner enabled; the owner can remove the pointer with **setSubregistry**, but payments already sent are not reversible. The session key thus gains authority beyond its intended scope and lifetime, and the attacker keeps acting over the owner's subname resolution.

IMM-WA-MED-89249

Manager migration grants `ROLE_SET_RESOLVER` to a stale cached V1 manager, letting a former seller hijack the resolver

Report ID	#89249
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

During v1 to v2 migration, Manager restores each unwrapped name's V1 Registry manager by granting it `ROLE_SET_RESOLVER` on V2. That address comes from the V1 subgraph's `owner { id }` ([v1SubgraphClient.ts#L22-L49](#)), cached client-side for five minutes ([useV1Names.ts#L12-L23](#)), and `buildRoleGrantCall` encodes it directly ([buildRoleGrantCalls.ts#L11-L27](#)). The live eligibility checks read token ownership, approvals and V2 status, but never the live V1 `ENSRegistry.owner(node)`. The batch's post-state expectation then requires that same address to hold the role ([buildAtomicMigrationBatches.ts#L608-L625](#)), and the confirmation dialog says only "Restore the existing managers for your names" without naming it ([WalletConfirmationStepsDialog.tsx#L87](#)).

A seller who sells an unwrapped `.eth` name while remaining its Registry manager therefore stays listed as manager if the buyer reclaims that role and migrates within the cache window, or before the subgraph indexes the change. The buyer's signed migration grants the seller `ROLE_SET_RESOLVER`, and the seller can then point the name at an attacker-controlled resolver so that it resolves to an attacker address. Using Manager's classification, eligibility and batch builders, with a live Registry owner lookup that would have returned the victim but was never called, the proof of concept decoded the `executeByOwner` calldata and found `grantRoles(resource, ROLE_SET_RESOLVER, FORMER_MANAGER)` with verification expecting that role. The migration flow thus assigns control of the name's resolver to an attacker.

IMM-WA-MED-89254

Channel-status writes match on **target** alone, letting an attacker's pending row disable another user's email notifications

Report ID	#89254
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: Enabling/disabling notifications

Description

A notification channel's identity in the api-worker schema is **(user_id, channel, target)**, so two accounts may each hold a row for the same email address ([notifications.ts#L73-L79](#)). The SendGrid webhook, however, selects the row to update by address alone, with no user, channel type or ordering ([sendgrid.ts#L172-L174](#)).

Its **bounce**, **dropped**, **spamreport** and **unsubscribe** branches then set that row's **status** ([sendgrid.ts#L184-L231](#)). The DLQ hard-bounce handler and the push **410** handler go further and update every row sharing the **target** ([dlq.ts#L201-L209](#), [push.ts#L117-L124](#)). An attacker can create the collision: the email-attach route rejects an address only when another account holds it as **verified** ([email.ts#L58-L72](#)), so a victim's not-yet-verified address can be attached as a **pending** row, which also sends the victim an ENS verification email. Once the victim verifies the same address on their own account, both rows are legal, and a spam report, unsubscribe or transient bounce on any message to that address disables whichever row **findFirst** returns. The reproduction compiled the statements with the lockfile's **drizzle-orm** 0.45.2 and confirmed the predicate carries neither **user_id** nor **channel**, and no **ORDER BY**. When the victim's row is the one selected, it reads **bounced** and the victim silently stops receiving name-expiry warnings, which changes non-sensitive details of other users by disabling their notifications.

IMM-WA-MED-89627

Transfer plan never revokes third-party role grants, so a prior grantee can repoint the transferred name

Report ID	#89627
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

The Explorer's transfer form is a cleanup checklist: three toggles default to on, and the resolver toggle promises that "The recipient starts clean and sets up their own" ([SendNameForm.tsx#L36-L61](#), [#L69-L73](#)). `buildTransferPlan` compiles the options into at most four step kinds ([buildTransferPlan.ts#L16-L43](#)).

Roles, however, are keyed on the label rather than the token: `grantRoles` derives its resource from `labelToCanonicalId(label)` ([grantRoles.ts#L70-L71](#)), so a grant survives the ERC-1155 transfer, and no step revokes it. The flow reads roles only for the seller's own account ([useTransferDetachTargets.ts#L67-L74](#)), and the ownership page lists another controller only for ENSv1 names ([index.tsx#L133-L135](#)). On a Sepolia fork, a seller granted a third party `ROLE_SET_RESOLVER` through the app, then transferred the name with every toggle on; the app ran `setResolver(..., 0x0)`, `setSubregistry(..., 0x0)` and `safeTransferFrom`, after which the seller's bitmap was `0x0` and the third party's still `0x1000000`. After the buyer set up their own resolver, the third party, holding zero tokens, repointed the name's resolver through the app's own `changeResolver` helper, and the app's `getEthAddress` query returned the attacker for the buyer's name; an address with no grant making the same call reverted with `EACUnauthorizedAccountRoles`. Payments sent to the name then reach the attacker while the buyer stays owner, which hijacks the name's resolver after a transfer flow.

Severity note. The retained role exists only if the previous owner granted it before the transfer, and the new owner can revoke it once aware of it, which bounds the exposure to payments sent before then.

IMM-WA-MED-90446

Uncapped push channels and an early-return retry let one user permanently suppress other users' expiry reminders

Report ID	#90446
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: Enabling/disabling notifications

Description

The api-worker's expiry consumer turns each `(event, user)` pair into an in-app notification plus one external delivery per verified channel. `POST /notifications/channels/push` deduplicates only the caller's exact endpoint and stores each new one as `status: 'verified'` ([push.ts#L70-L77](#), [#L100-L115](#)), so one account can hold thousands of invented endpoints. `processExpiryEvents` first inserts notifications with `onConflictDoNothing` on the idempotency key ([event-ingestion.ts#L249-L262](#)), then loads every verified channel of the inserted users ([#L279-L291](#)) and writes the whole cross-user fanout in one statement ([#L407-L410](#)), outside any transaction. If that write fails, every message is retried ([#L601-L605](#)). The retry, however, treats an idempotency conflict as finished work ([#L268-L273](#)).

The messages are then acknowledged ([#L608-L610](#)) and the missing deliveries are never rebuilt. With 10,922 fake channels and a favorite on the victim's name, one event yields 10,923 delivery rows, or 65,538 bind parameters, above PostgreSQL's 65,535 ceiling; against a local PostgreSQL 17 the insert failed with `08P01`, the notification rows stayed committed, and the retries returned zero rows for the `7d`, `1d` and `expired` stages. A 219-channel variant reaches 65,706 parameters when 50 events share a consumer batch. The victim keeps the in-app row but receives no email, push or Telegram reminder, with no interaction on their part, which changes non-sensitive details of other users by disabling their notifications.

IMM-WA-MED-91127

Uncapped favorites overflow the unchunked expiry notification insert, failing shared batches for unrelated users

Report ID	#91127
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Taking state-modifying authenticated actions (with or without blockchain state interaction) on behalf of other users without any interaction by that user, such as: Changing registration information · Notification/settings changes

Description

The api-worker's expiry consumer creates one notification per recipient of each event in a queue batch, and for stages that include favorites the recipients are the owner plus every user who has favorited the name ([event-ingestion.ts#L75-L106](#)). Favoriters are loaded with no limit ([#L159-L188](#)), and `PUT /favorites/:name` requires only authentication, with no cap per name or per user ([favorites/index.ts#L21-L36](#)). Every recipient of every event in the batch accumulates in `notificationsToInsert` with four fields per row ([event-ingestion.ts#L190-L235](#)), and the whole array goes to a single unchunked insert ([#L249-L262](#)). That statement, however, cannot carry more than PostgreSQL's 65,535 bind parameters, which a batch with more than 16,383 recipients exceeds. When it throws, the handler retries every valid message in the batch ([#L601-L605](#)), and the event-ingestion consumer allows three retries ([wrangler.jsonc#L76-L81](#)), and each retry that still carries the poisoned name fails the same way. An attacker who signs in with 17,000 gasless SIWE identities and favorites one name therefore fails every batch that name lands in, and the owners and favoriters of the other names grouped with it get no expiry notification; the favorite rows persist, so each later stage repeats the failure. The reproduction drove the shipped `collectRecipientsForEvent` with 17,000 favoriters of one name and asserted 17,000 recipients, or 68,000 parameters, without executing the insert against PostgreSQL. The attacker's own writes thus change the notification outcome of other authenticated users without any interaction by those users.

IMM-WA-MED-91314

V1 name list inherits the `ensjs resolvedAddress` relation, listing names that merely point at a user as owned

Report ID	#91314
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Explorer lists an address's names on its names page, the dashboard and the search box's "Names you own" section ([SearchResultsList.tsx#L289-L292](#)). The V1 half comes from `getV1NamesForAddress`, which forwards its parameters to `ensjs's getNamesForAddress` unchanged ([useV1NamesForAddress.ts#L18-L31](#)), and the call sites pass only `{ address }` ([names.tsx#L170-L175](#)). The `ensjs` default relation set, however, includes `resolvedAddress`, the address a name's `addr` record points at, which that name's owner sets unilaterally. The V2 reader filters on `owner` ([useV2NamesWithRolesForAddress.ts#L55](#)), and Manager's V1 reader lists only `owner`, `registrant` and `wrappedOwner` ([v1SubgraphClient.ts#L56-L59](#)).

The names table maps each row to owner and manager flags without reading `relation.resolvedAddress` ([names.tsx#L177-L191](#)), so an attacker's name whose `addr` record points at the victim appears among the victim's names, with Extend available. On a Sepolia fork, one `setAddr` on a live V1 name was enough: renewing it from the victim's list cost the victim 8.000021 USDC and extended the attacker's registration by one year, with its registrant unchanged. The application thus displays an incorrect ownership detail, on the strength of which the user submits a renewal transaction for someone else's registration.

Severity note. The renewal payment goes to the registrar and extends the name shown on the row; none of the victim's own names or assets is affected, so the loss is bounded by the price of one renewal term.

IMM-WA-MED-91689

Unnormalized child routes and a spoofable resolver check let an attacker contract become a victim's transaction target

Report ID	#91689
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions · Signing or submitting a transaction against the wrong chain (for example when the request's target chain is not validated) · Inducing a blanket token approval (for example setApprovalForAll on the registry or NameWrapper) to an attacker-controlled address

Description

The Explorer rejects non-normalized names only in the `$name/index` route ([index.tsx#L441-L460](#)); the parent `$name` route performs no equivalent check ([\\$name.tsx#L25-L45](#)), so `$name/edit-records` loads for a raw NFD name that renders like a victim's NFC name. That page reads the raw name's resolver and gates editing on `useCanEditRecords` ([edit-records.tsx#L73-L75](#)). `getIsPermissionedResolver` trusts a resolver whose own EIP-1967 implementation slot holds a known implementation address ([useIsPermissionedResolver.ts#L46-L66](#)), and for such a resolver `canEdit` is that resolver's own `hasRoles` answer ([useCanEditRecords.ts#L75-L113](#)).

Both inputs, however, come from the contract being checked. An attacker who owns the NFD name can point it at a non-proxy contract that writes the expected implementation into that slot and answers `true` for the victim. The save flow then uses that resolver as the transaction target ([saveRecords.ts#L144-L149](#)) under a modal titled "Save records". In the proof of concept, the victim opened the `/edit-records` URL for the NFD name, saved a text record and approved the wallet request; the mined transaction had `from` equal to the victim and `to` equal to the attacker's contract, while the index route for the same name showed "Invalid name". This is a malicious interaction with an already-connected wallet: the application substitutes an attacker-controlled contract address as the target of what it presents as an ordinary record update.

Severity note. The substituted call reaches attacker code but does not act on any asset the victim controls, and it requires the victim to open the crafted URL, make an edit and approve the wallet request.

IMM-WA-MED-91913

Manager Upgrade skips the locked-resolver safety check for the live Sepolia public resolver, wiping a name's records

Report ID	#91913
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

For a locked name with `CANNOT_SET_RESOLVER` burned, the migration receiver can rotate a certified public resolver to the shared `PublicResolverV2`, into which the plan cannot replay records, so `assertLockedResolverReplacementRecordSafety` permits that rotation only after proving the V1 record inventory empty ([buildMigrationPlan.ts#L280-L290](#)), and Manager probes `PublicResolverSet` membership before signing. Both gates key off `isKnownPublicResolver`, a static list that omits the live Sepolia public resolver `0xE99638b40E4Fff0129D56f03b55b6bbC4BBE49b5`, which `PublicResolverSet.includes()` reports as a member. For that resolver, `resolverFor` returns the V1 address unchanged ([encodeMigration.ts#L34-L40](#)), so `lockedResolverReplacementsWithoutAtomicReplay` drops the name as not being replaced ([buildMigrationPlan.ts#L175-L184](#)), the safety check returns without reading any records, and `requiresPinnedPublicResolverMembership` skips the probe ([migrationInvariants.ts#L154-L159](#)). No record replay is built, yet the wallet dialog reads "Migrate the selected names and restore their records" ([WalletConfirmationStepsDialog.tsx#L107](#)).

On a Sepolia fork using Manager's `classifyName`, `resolverFor` and `buildMigrationHelperCall`, the migration left V2 `getResolver` pointing at the empty `PublicResolverV2`, with `addr(60)` and the `url` text record empty, while the records remained on `0xE996...`. The owner holds no `ROLE_SET_RESOLVER` to point the name back, and the post-migration check expects `0xE996...`, so the batch fails verification and is not resubmitted. The Manager v1 to v2 migration thus leaves the name resolving to nothing, with no in-app path to restore the records it promised to keep.

IMM-WA-MED-92466

Uncapped `description` record escaped twice into meta tags lets one name crash the Explorer Worker isolate

Report ID	#92466
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Taking down the application/website

Description

For every HTML request to a name page, the Explorer Worker reads the name's `description` text record and returns it with no length cap ([ens.ts#L165-L168](#)), although the same Worker caps the avatar at 5 MiB and truncates the displayed name. `handleNamePage` passes the value to `buildMetaTags` ([worker.ts#L271-L277](#)), which escapes it twice, for `og:description` and `twitter:description`, and joins all tags into one in-memory string ([worker.ts#L47-L61](#)). `escapeHtml` turns each `"` into the six-character `"`, so a record of quotes becomes a string twelve times its size.

Nothing on this path, however, bounds or catches the allocation: `worker.ts` contains no `try` block, and a V8 heap-limit abort is not a catchable exception in any case, so it terminates the isolate and every request it is serving ([worker.ts#L398-L402](#)). An attacker needs only a name they own with a large `description`, which the Worker's client also fetches over CCIP-Read from an attacker-run gateway without any on-chain write, and a single request or a posted link that an unfurler fetches. In the proof of concept, the pinned `escapeHtml` and `buildMetaTags` ran under a 128 MiB heap, Cloudflare's published per-isolate limit: a quote-filled record of about 2 MiB aborted the process with a heap out-of-memory error and 0 of 8 concurrent requests completed, while the same size of plain text completed all 8. One attacker-owned name thus becomes a repeatable trigger for taking down the application for every request sharing the isolate.

IMM-WA-MED-92517

Pending-change transformer emits a same-key deletion after its replacement, reviving a revoked delegate's fallback address

Report ID	#92517
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

In the Explorer's record editor, which turns pending edits into one `setRecords` batch, deleting an existing Base address adds its id to `deletedIds` while adding a new one appends to `newRecords` ([useEditRecordsState.ts#L58-L89](#)). `transformPendingChangesToSetRecords` emits new records first and deletions last, as an empty value, without reducing entries that share a coin type ([transformPendingChanges.ts#L79-L95](#), [#L119-L138](#)).

`saveRecords` passes that ordered list to `setRecordsWriteParameters` ([saveRecords.ts#L110-L136](#)). A delete-and-re-add of the same key therefore becomes `setAddr(Base, C)` followed by `setAddr(Base, empty)`, and the second call erases the replacement. On `PermissionedResolver` an absent Base record falls back to the Default address, however, and that value may have been written earlier by a delegate the owner has since revoked. On Sepolia and Base Sepolia forks, the revoked delegate's later writes failed, the owner's save emitted the two calls in that order, the resolver returned the delegate's address A, and `CoinRecord` displayed and copied A ([CoinRecord.tsx#L72-L77](#)); a payer using the copied value sent `0.01 ETH` to A. Reversing only the two operations left C resolved and paid. The application renders the name's Base address as the former delegate's, so a user who relies on it sends funds to an unintended recipient.

IMM-WA-MED-92542

Roles editor's Remove user action revokes undisplayed `ROLE_CAN_TRANSFER_ADMIN`, freezing the name for its remaining term

Report ID	#92542
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

On `/$name/roles`, each role holder's sheet renders four permissions (Renew, Set Subregistry, Set Resolver, Unregister), keyed by a type that excludes every `_ADMIN` role (`permissions.ts#L11-L40`), with the Admin checkboxes permanently disabled (`RolesSidebar.tsx#L271-L275`). The sheet's **Remove user** button, however, sends the account's full decoded role list to the revoke transaction without filtering it (`RolesSidebar.tsx#L125-L141`). For an ordinarily registered `.eth` 2LD that list includes `ROLE_CAN_TRANSFER_ADMIN`, which has no row in the sheet at all, along with the resolver and subregistry admin roles, shown only as disabled Admin checkboxes. On the owner's own row, the confirmation says only that deletion "would prohibit you from adding more users" (`RolesSidebar.tsx#L344-L346`). The sibling `RegistryEditUserSheet` filters with `getRemovableRoles` and confirms sole-admin lockouts before removal (`RegistryEditUserSheet.tsx#L161-L178`); `RolesSidebar` imports neither helper.

On a Sepolia fork, after the app's own `revokeRoles` landed, `safeTransferFrom` reverted with `TransferDisallowed`, `setResolver` and `setSubregistry` reverted, and `grantRoles` to restore the roles reverted with `EACCannotGrantRoles` for the owner, the issuing registrar and the registry's root role holder. The owner keeps the token but cannot transfer it or change its resolver or subregistry until the registration expires. The application thereby displays an incorrect transaction detail, and the user submits a revocation far broader than the one shown.

IMM-WA-MED-92608

`buildTokenData` prices a failed read at `0n`, so checkout shows \$0.00 while the registrar charges full rent

Report ID	#92608
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Explorer's single-name payment picker quotes each token's price before registration or renewal, and `buildTokenData` substitutes `DEFAULT_PRICE`, whose `total` is `0n`, for any price that is not a valid `TokenPrice` ([tokenData.ts#L13-L19, #L56](#)). `PaymentTokenPicker` checks only `isLoading` before passing each query's `data` in ([PaymentTokenPicker.tsx#L109-L115, #L144-L149](#)), so a price read that settles in error reaches the list as a free token. Every downstream gate then compares against that zero. `PaymentTokenList` enables the button and prints "available" on an empty wallet ([PaymentTokenList.tsx#L34-L64](#)).

`PaymentTokenSection` hands `price.total` to the flow ([PaymentTokenSection.tsx#L42-L46](#)), and `needsApproval` compares the allowance with that saved zero ([useRegistrationTransactions.ts#L137-L140](#)), which drops "Approve payment" from the step list. The multi-name renewal picker, however, blocks the same condition, stating that an errored read "must NOT be treated as 0" ([MultiNamePaymentTokenPicker.tsx#L119-L137](#)). On a Sepolia fork, the defect arm displayed `$0.00`, the wallet was still asked to sign `approve(registrar, 8800023)` for a step the modal no longer listed, and the registrar pulled `8000021` USDC base units, taking the buyer from `1000000000` to `91999979`; the control arm, with the read resolving, displayed `$8.00` and charged the same amount. The application shows the wrong amount on a transaction the user then approves.

Severity note. The amount charged is the registrar's correct rent price, paid to the legitimate recipient, and the trigger depends on a price read failing, for which no attacker-controlled cause has been shown.

IMM-WA-MED-92623

Role holder list replays only the newest 1,000 registry-wide events, letting a seller hide a retained delegate

Report ID	#92623
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

The role-management page, the control a new owner uses to find and revoke surviving delegates, is built by `getNameRolesAccountsFromIndexer`, which requests the newest `MAX_EVENTS` (1,000) `EACRolesChanged` events for the whole registry and only then discards those for other resources ([useNameRoleAccounts.ts#L50-L110](#)). It neither paginates nor checks completeness, and an empty result counts as success, so the on-chain fallback runs only when the GraphQL call throws ([useNameRoleAccounts.ts#L139-L150](#)). An empty map makes `RolesTable` render "No role holders yet" with no edit action ([RolesTable.tsx#L48-L59](#)).

Third-party registry roles survive a transfer, however, and the transfer form, which says detaching the resolver and registry lets the recipient start clean, does not revoke them ([SendNameForm.tsx#L45-L60](#)). A seller can grant `ROLE_SET_RESOLVER` and `ROLE_SET_SUBREGISTRY` to an alternate account, transfer the name, then send one 500-entry self-batch transfer on another name they own, which emits 1,000 newer role events without changing its state (about 7.39 million gas). The buyer's role page then shows no holders, and the hidden delegate calls `setResolver` and `setSubregistry`. Run against the official v2 registry and a local indexer, the proof of concept showed an empty list while the role stayed active on-chain, and both pointers were replaced with attacker values. The name is thereby hijacked after transfer, with its resolver and subregistry assigned to an attacker.

IMM-WA-MED-92695

`setupControlledResolver` omits intent funding and keeps the session signer, so resolver setup cannot complete

Report ID	#92695
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

The Manager's own save machine documents why an HCA intent fails after registration: a session-signed one hits `HCAOwnerAndSessionValidator`, whose action policy allowlists only the registration selectors, and an owner-signed one needs USDC for the intent fee "that registration leaves the HCA without" ([EditProfileDialog.machine.ts#L160-L169](#)). `setPrimaryName` meets both by stripping the session and calling `planHcaIntentFunding` before it builds its intent ([setPrimaryName.ts#L189-L198](#)). `setupControlledResolver`, which Edit Profile and the primary-name dialog call to deploy an owned resolver and point the name at it ([useEditProfileDialogSave.ts#L253-L262](#), [ChoosePrimaryNameDialog.tsx#L277-L288](#)), does neither ([setupControlledResolver.ts#L114-L121](#)).

The function takes no `walletClient`, declares no `auxiliaryFunds`, and forwards the caller's `signer`, session included, to `transactionManager.startTransaction` ([setupControlledResolver.ts#L123-L133](#)). The batch, however, includes `setResolver`, which is not a registration call, and draws its fee from an HCA balance that registration leaves empty. The reproduction's unit test confirmed that `planHcaIntentFunding` is never called, that the request carries no `auxiliaryFunds` or permit calls, and that a session-bearing signer reaches the transaction manager unchanged. A smart-account user with an active session, or whose HCA holds no USDC, therefore cannot complete resolver deployment or profile setup through either Manager flow, which leaves the name without a resolver they control.

IMM-WA-MED-92723

Missing `ccipRead` guard in the Explorer Worker client lets an attacker's resolver exhaust its RPC quota

Report ID	#92723
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Taking down the application/website

Description

The Explorer's Cloudflare Worker resolves names server-side for OG images and page metadata: the unauthenticated routes `GET /og/<name>.png` and `GET /<name>` both call `fetchEnsData` ([worker.ts#L169-L173](#), [worker.ts#L250-L253](#)), which reads records through the client from `createClient` ([ens.ts#L140-L148](#)). That client sets no `ccipRead` option ([clients.ts#L9-L24](#)), so viem's ERC-3668 lookup stays enabled: when a resolver reverts with `OffchainLookup`, viem fetches every URL in the resolver-supplied list, continuing past failures, with no timeout, scheme or host restriction.

The Worker routes the avatar URL on the same request through `checkFetchableUrl`, which rejects non-HTTPS, IP-literal and self-host URLs ([safe-fetch.ts#L74-L93](#)), but the gateway fetch happens inside viem and never reaches it. Any name owner can point a name at a resolver whose URL list is long or loops back to the Worker's own `/og/` route. On a local deployment of the pinned tree, one request produced 8,122 upstream `eth_calls`, against 2 for the control, and never returned; against a metered 500-call quota, a single request exhausted it and the next legitimate request had its RPC calls refused. A 1,200-entry list produced 1,200 subrequests in one invocation, above Cloudflare's Free-plan cap of 50. Once the quota is drained, pages are served without resolved ENS data, taking down the application's server-side name resolution.

IMM-WA-MED-92778

Manager's HCA registration verifier ignores the subregistry, certifying an attacker's registration of the name as complete

Report ID	#92778
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Hijacking or bricking an ENS name during a Manager flow (registration, v1 to v2 migration, or transfer), such as: a registration or migration that assigns the name, its ownership, registry, or resolver to an attacker · redirecting the recipient of a name transfer · leaving a name permanently inaccessible after the flow

Description

Because receipt polling can fail after a reveal has landed, the registration machine routes every polling error from `waitingForRhinstoneBundle` to `verifyingRegistration` ([registration.machine.ts#L1225-L1240](#)), and a `verified` result moves it to `success` and clears the error ([registration.machine.ts#L1448-L1474](#)). The HCA verifier reads only `getState` and `getResolver` ([registration.hca.actors.ts#L928-L946](#)).

It never reads the subregistry, the commitment, or the fill that produced the state. An attacker who knows the label and wallet in advance can mature their own commitment and register the label directly to the victim with the deterministic HCA resolver but a hostile subregistry. If that transaction lands after Manager builds the reveal and before the fill, the victim's reveal reverts with `NameNotAvailable`, yet the attacker's state passes all three checks and Manager renders `Registration Complete!` ([RegistrationCompletionBanner.tsx#L4-L20](#)). On a pinned Sepolia fork using Manager's reveal builder and an owner-authorized session, the proof of concept showed the reveal reverting, the verifier returning `{ verified: true }`, no code at the expected resolver, and `pay.<label>.eth` resolving through the attacker's subregistry to an attacker address. The Manager registration flow thereby assigns the name's registry to an attacker while reporting success.

Severity note. The victim keeps ownership of the name itself, and the attack depends on advance knowledge of the label and wallet and on winning a narrow inclusion window.

IMM-WA-MED-92816

Expiry-discovery cursor on non-unique `expiryDate` permanently drops names tied at a 1,000-row page boundary

Report ID	#92816
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: Enabling/disabling notifications

Description

The hourly expiry-discovery cron pages through the names entering each notification stage so that owners and, for `7d`, `1d` and `expired`, favoriting users get a reminder ([stages.ts#L9-L30](#)). Its subgraph query filters on a strict lower bound and sorts by `expiryDate` alone, with `PAGE_SIZE = 1000` ([indexer.ts#L9-L28](#)).

After enqueueing a page, `processStage` sets `cursorEnd` to the last returned domain's `expiryDate` ([expiry-discovery/index.ts#L111-L112](#), [#L136](#)), and `runExpiryDiscoveryCron` persists it as the stage cursor ([#L204-L207](#), [#L224](#)). `expiryDate`, however, is not unique: names registered or renewed in the same block with the same duration share it exactly. When a full page ends inside such a group, the tied names left off the page can never satisfy `expiry_gt` again, and the cursor only moves forward. In the reproduction, the unmodified cron ran five times against a local subgraph serving 997 unique names, five names sharing one `expiryDate` across the 1,000-row cut, and one later name; `boundary-4.eth` and `boundary-5.eth` were never returned or passed to `EVENT_INGESTION_QUEUE.sendBatch`, while every other name was enqueued exactly once. The owners and favoriting users of the dropped names silently stop receiving that stage's email, push or Telegram reminder, with no interaction on their part, which changes non-sensitive details of other users by disabling their notifications.

IMM-WA-MED-92825

Missing registry scoping in the role History query lets anyone insert forged role-change entries into any name's history

Report ID	#92825
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: Enabling/disabling notifications

Description

The Explorer's History panel, the record of who was granted which roles on a name, is populated by `getRoleHistory`, which requests every `EACRolesChanged` event filtered only by type and block number ([useRoleHistory.ts#L46-L72](#)). The query neither filters on nor selects `contractAddress`, and the results are narrowed client-side by a resource derived from the label alone, `toResourceHex(labelToCanonicalId(label))` ([useRoleHistory.ts#L84-L86](#)). The sibling hook `useNameRoleAccounts` documents that a resource-only filter "would conflate role grants across unrelated registries" and scopes its own query by `contractAddress` ([useNameRoleAccounts.ts#L52-L77](#)).

The History hook, however, applies no such scoping, and a label's canonical id is identical in every registry. Any account can deploy a registry through the Explorer's own deployment flow, register the victim's label in it, and call `grantRoles` with an account and role bitmap of its choosing. The emitted event carries the victim's resource and appears in the victim's History panel beside the genuine entries, with a real transaction hash and timestamp; `IndexerEACEvent` has no field that could tell them apart ([filterEventsByResource.ts#L5-L11](#)). The proof of concept executed this on a Sepolia fork against the deployed factory, registry implementation and ETHRegistry, in a single 124,549-gas transaction. The entries are on-chain and permanent, and they are shown to anyone who opens the panel. This changes non-sensitive details of other users without wallet interaction, falsifying the audit trail an owner consults when deciding which accounts to revoke.

IMM-WA-MED-92870

Manager lower-cases v2 role names before re-querying them, listing a third party's name as managed on any profile

Report ID	#92870
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Injecting/modifying the static content on the target application without JavaScript (persistent), such as: HTML injection without JavaScript · Replacing existing text with arbitrary text · Arbitrary file uploads, etc

Description

The Manager's public address profile lists the v2 names an address manages, built from the indexer's `roles(account:)` rows, which carry each name as registered and nothing else identifying it ([getDashboardRoleAssignments.ts#L23-L30](#)). `getRoleBitmapByName` keys every row through a lower-casing fold ([v2NameRoles.ts#L19-L20](#), [#L31-L54](#)).

`getManagedOnlyRoleNames` returns those folded keys ([v2NameRoles.ts#L101-L116](#)), `getProfileAddressNames` queries `domains(where: { name_in })` with them ([profileAddressNames.ts#L52-L68](#), [#L82-L83](#)), and `applyProfileV2RoleAssignments` tags every returned domain with `MANAGER_ONLY_ROLES` ([v2NameRoles.ts#L136-L147](#)). The folded key, however, can name a different domain from the one the role is on. An attacker who registers the mixed-case label `crypt0` and grants a role on it to any address, which needs no consent from the grantee, makes the Manager fetch `crypto.eth`, a real name owned by an unrelated party, and render it with a `Manager` pill ([DashboardPills.tsx#L41-L45](#)). For every visitor other than the profile's own address, the owned/managed filter is skipped ([AddressProfileNamesList.tsx#L182](#)). The state was planted on Sepolia with one registration and one `grantRoles` call, and the shipped build, with no wallet connected, listed `crypto.eth` under a `Manager` pill as the only name of an address that owns nothing. The row stands until the attacker retracts the role or the name expires, which injects persistent static content into the application without JavaScript.

IMM-WA-MED-92995

Raw-versus-normalized social diff re-authors a prior owner's handles into the replacement resolver on a Discord-only edit

Report ID	#92995
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Injecting/modifying the static content on the target application without JavaScript (reflected), such as: Reflected HTML injection · Loading external site data

Description

When a transferred name's inherited resolver is not writable, Manager's **Replace & Save** dialog promises that continuing "clears them and keeps only the changes you save" ([ResolverSetupConfirmDialog.tsx#L38-L47](#)). On every submit, **EditProfileDialogBody** passes the whole form through **normalizeProfileRecords** before saving ([EditProfileDialogBody.tsx#L72-L74](#)), which rewrites every social entry to its display value ([transformRecords.ts#L78-L93](#)), so nine keys such as **com.twitter** lose their profile URL and keep the bare handle, while **com.discord** has no normalizer. The save machine, however, serializes the raw resolver snapshot as **before** and the normalized form as **after** ([EditProfileDialog.machine.ts#L127-L133](#)), and **computeRecordChanges** writes every key whose values differ ([profileRecordTransactions.ts#L172-L182](#)). **setupControlledResolver** batches those writes behind an optional resolver deploy and **setResolver** in one fee-paying intent ([setupControlledResolver.ts#L96-L121](#)). In the reproduction, a prior owner stored URL-form values such as **https://twitter.com/attacker**; a Contact-tab change to **com.discord** alone produced ten **setText** calls, nine carrying **attacker**, and on local canonical ENS contracts the new resolver kept all nine while untouched non-normalizing keys stayed empty. Manager then shows three of them as fallback contact cards, and Portal links them under fixed social origins ([SocialRecord.tsx#L128-L137](#)), so the recipient's profile shows content another party chose, modifying the static content on the target application without JavaScript.

IMM-WA-MED-93007

Address-page primary-name offer lets a name owner make their name a connected visitor's primary name

Report ID	#93007
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immuneFi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions · Signing or submitting a transaction against the wrong chain (for example when the request's target chain is not validated) · Inducing a blanket token approval (for example setApprovalForAll on the registry or NameWrapper) to an attacker-controlled address

Description

On the Explorer's `/$name/address` page, which reports whether each network's address reverse-resolves back to the name, a mismatch banner carries a "Set primary name" button. The button renders when `status === 'mismatch'` and the connected wallet equals the row's address ([AddressResolutionSidebar.tsx#L351, #L454-L458](#)); for the Mainnet row that address is the name's own coin-60 record ([address.tsx#L58-L67](#)). Clicking it writes the URL's name into the visitor's own reverse record ([AddressResolutionSidebar.tsx#L531-L543](#)).

The offer, however, never asks whether the visitor has any authority over that name. The sheet computes `useCanEditRecords` with `ROLE_SET_ADDR` ([AddressResolutionSidebar.tsx#L485-L488](#)) but uses it only for the Address field, and the forward table leaves `showMoreButton` at its default `true` ([SidebarTriggerRow.tsx#L12](#)), whereas the reverse table gates it on the connected account ([ReverseResolutionTable.tsx#L62-L64](#)). The owner of `mallory.eth` points its ETH record at a victim who already has a primary name and sends the page link; the victim sees the banner and the button. In the reproduction, four in-app clicks handed the transaction manager `setName("mallory.eth")` for the Sepolia `ReverseRegistrar`, from the victim's address, while the same run's `hasRoles` check returned `false`. Once signed, the victim's address reverse-resolves to a name whose records the attacker controls: a malicious interaction with an already-connected wallet that submits a transaction whose only argument the attacker chose.

IMM-WA-MED-93008

Transfer flow points `addr(60)` at the recipient before a transfer the recipient can revert, diverting payments

Report ID	#93008
Severity	Medium
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Explorer's V2 transfer form offers to point the name's ETH address at the recipient and to detach its resolver, both enabled by default ([SendNameForm.tsx#L69-L73](#)). When the owner lacks registry `ROLE_SET_RESOLVER`, the detach option is hidden ([useTransferDetachTargets.ts#L84-L90](#)) and `effectiveOptions.detachResolver` becomes `false` ([SendNameForm.tsx#L91-L96](#)), so `buildTransferPlan` emits `set-eth-addr` followed by `transfer-token` ([buildTransferPlan.ts#L27-L40](#)). The option list, however, reads the raw state ([SendNameForm.tsx#L185-L203](#)).

The ETH-address switch therefore stays on and locked, labelled "Not needed while the resolver is being detached", although no detach step runs. The address update and the ERC-1155 `safeTransferFrom` are separate confirmed transactions, so a recipient contract that switches its receiver hook to reject once the first confirms makes only the transfer revert. In the proof of concept, the transfer failed, the victim still owned the name and its resolver, and `addr(name, 60)` pointed at the recipient; an independent payer resolved the name, sent `0.001 ETH`, and the recipient withdrew it. The application thereby resolves the victim's name to an attacker-controlled address, so that payers send funds to an unintended recipient.

IMM-WA-LOW-89444

Unauthenticated `/wallet/fund` with address-keyed controls lets anyone drain the faucet wallet

Report ID	#89444
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Taking down the application/website

Description

The api-worker's `POST /wallet/fund` faucet signs Sepolia mint transactions with the operator's key and requires no authentication ([wallet/index.ts#L97-L107](#)); its two anti-abuse controls, a balance gate and a KV lock, are both keyed on the `address` the caller supplies ([#L121-L135](#)). A fresh address on each request, however, has zero balances and an uncontended lock, so every anonymous request costs the funder a transaction; the PoC mined 50 transactions for 50 requests. Concurrent requests also race the funder's nonce, and neither `waitForTransactionReceipt` call has a timeout, so 16 of 20 concurrent requests hung until the Workers runtime cancelled them. Once the funder is drained every call returns `Internal server error`, and the faucet, including the gas drip that wallet-paid v1-to-v2 migrations depend on, stays unavailable until an operator refills the wallet.

IMM-WA-LOW-89569

Missing cross-account email uniqueness lets an attacker verify a victim's address on their own account

Report ID	#89569
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Changing details of other users (including modifying browser local storage) without already-connected wallet interaction and with significant user interaction, such as: Iframing leading to modifying the backend/browser state (must demonstrate impact with PoC)

Description

The api-worker enforces one account per verified email with a guard that refuses an address already **verified** on another account ([email.ts#L58-L72](#)). That guard, however, runs only when a channel is requested and ignores **pending** rows, the database constraint is scoped to `(user_id, channel, target)` ([notifications.ts#L73-L79](#)), and the unauthenticated `/verify` handler marks the token's channel **verified** without checking for a rival row ([#L194-L201](#)). An attacker can therefore add a victim's not-yet-linked address to their own account, and when the victim follows the standard ENS verification mail and presses its confirmation button, the attacker's channel becomes verified. ENS notifications for the attacker's names then reach the victim's inbox, and the victim's own attempt to link the address is refused, while the attacker's row stays invisible and undeletable from the victim's session. The change takes two deliberate clicks on a mail the victim did not request.

IMM-WA-LOW-90307

Out-of-range on-chain expiry crashes the Explorer names page of any address sent an unsolicited subname

Report ID	#90307
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Taking down the application/website

Description

The Explorer's names page converts each v2 name's on-chain `uint64` expiry with `new Date(expiryDate * MS_PER_SECOND)` ([names.tsx#L194-L196](#)), and the expiry cell passes any truthy date to `dateToPlainDate` ([columns.tsx#L77-L94](#)). An expiry above 8,640,000,000,000 seconds, however, yields an `Invalid Date`, which is still truthy, so the `!expiryDate` guard does not fire and `Temporal.PlainDate.from` throws a `RangeError` during render. Because a v2 subregistry lets its owner mint a subname to any address with a caller-chosen expiry, an attacker can push one such name to a victim, and the router-wide error component then replaces `/addr/<victim>/names` on every load until the victim transfers the token away with other tooling. The PoC measured 170,595 gas per victim once the attacker's registry is deployed. The page a holder uses to view and renew their names stays unavailable for that address.

Severity note. The failure is confined to one route for one targeted address; the rest of the application keeps working.

IMM-WA-LOW-91233

`createDiff` treats any two upload-worker URLs as equal, leaving a former owner in control of profile media

Report ID	#91233
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Changing details of other users (including modifying browser local storage) without already-connected wallet interaction and with significant user interaction, such as: Iframing leading to modifying the backend/browser state (must demonstrate impact with PoC)

Description

Manager treats an avatar or header change as image-only when both the old and new record values start with `AVATAR_UPLOAD_BASE_URL`, and in that case finalises the signed uploads without sending any record transaction ([createDiff.ts#L209-L219](#), [useEditProfileDialogSave.ts#L455-L468](#)). The prefix test, however, does not bind the URL to the current network, name or image kind ([#L22-L25](#)). A seller who points a name's `avatar` and `header` at the official upload objects of a second name they keep therefore keeps those references through the buyer's Edit profile, Upload and Save: the buyer's uploads succeed, but in the PoC a clean resolver reread still returns the seller's two URLs. The seller can later upload new bytes to their own name and change the avatar and header Manager renders for the buyer's profile, with no authority over the buyer's name. Reaching that state requires the buyer's full edit-and-sign flow.

IMM-WA-LOW-91443

One-byte truncated CBOR ABI record on an unsolicited name hangs the victim's Manager dashboard

Report ID	#91443
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Temporarily disabling user to access target site, such as: Locking up the victim from login · Cookie bombing, etc.

Description

Manager's default dashboard runs the full `profileRecordsQuery` for every active v2 name on its first page (`MyNamesList.tsx#L255-L259`), although each row uses only `theme` and `avatar`, and that query always requests ABI data (`profileRecords.ts#L122-L130`). An ABI of content type `4` is CBOR-decoded, but the decoder in `@ensdomains/address-encoder` does not check for end of input: the single byte `0xbf` opens an indefinite-length map whose break byte never arrives, so the decode loop never exits. An attacker sets that one-byte ABI record, revokes their resolver roles and transfers the name to a victim, and the victim's visit to `/`, which redirects to `/dashboard`, hangs the renderer with no click on that name. In two fresh-browser PoC runs the renderer stayed blocked for over ten seconds until the tab was killed, while an equal-size `0xa0` control stayed responsive. The dashboard hangs on every visit for as long as the victim holds the name.

IMM-WA-LOW-91444

Oversized coin-type 121 address record on an unsolicited name freezes the victim's Manager dashboard

Report ID	#91444
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Temporarily disabling user to access target site, such as: Locking up the victim from login · Cookie bombing, etc.

Description

Manager's default dashboard runs the full `profileRecordsQuery` for every active v2 name on its first page (`MyNamesList.tsx#L255-L259`), and that query always adds Manager's fixed coin inventory to the request, whatever the indexer reports (`profileRecords.ts#L116-L120`). The returned bytes, however, reach the format-specific decoders with no length bound, and the ZEN (coin type 121) decoder checks only a two-byte prefix before a quadratic Base58Check conversion. An attacker transfers a name whose resolver returns a 131,072-byte `0x2089`-prefixed value for coin 121, and the victim's ordinary visit to `/`, which redirects to `/dashboard`, decodes it on the renderer thread without any click on that name. In two fresh-browser PoC runs the renderer and `Open navigation` stayed blocked for 43.9 and 45.4 seconds, against at most 71 ms for the controls. The victim is temporarily locked out of their Manager dashboard on every visit while they hold the name.

IMM-WA-LOW-91446

Quadratic primary-contact deduplication freezes the Manager tab when a gifted name is opened

Report ID	#91446
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Temporarily disabling user to access target site, such as: Locking up the victim from login · Cookie bombing, etc.

Description

Manager always requests the `domains.ens.primary-contacts` text record ([text.tsx#L31-L41](#)) and normalises it by filtering supported keys, deduplicating them with `validKeys.indexOf(key) === index`, and only then applying the three-contact limit ([records.ts#L139-L147](#)). Neither step bounds the array, and the deduplication is quadratic when the first occurrences of each key lie far apart. An attacker transfers a name whose resolver returns 270,000 supported keys grouped as 90,000 each of `mail`, `email` and `phone`, forcing 24,300,270,000 comparisons when the victim opens the name from their Manager address page. In the PoC the renderer stayed blocked for 23.9 seconds after the response and `Back` was delayed 23.4 seconds, against 1.4 seconds for an equal-size interleaved control. The victim's Manager tab is temporarily unusable each time the name is opened.

IMM-WA-LOW-91450

Unbounded `links` record rendering freezes the Manager tab for about 14 seconds on a gifted name

Report ID	#91450
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Temporarily disabling user to access target site, such as: Locking up the victim from login · Cookie bombing, etc.

Description

Manager's profile read requests every text key the indexer reports, including `links`, whose parser schema-validates every element of the array ([transformRecords.ts#L51-L65](#)); `ProfileLinksSection` then mounts one `LinkPreview` per link, and each generates and percent-encodes a 64-rectangle SVG pattern ([ProfileLinksSection.tsx#L24-L25](#)). No byte, item-count or rendering bound applies at any stage. An attacker whose resolver returns 60,000 valid links for a name can transfer that name to a victim; one ordinary click on it from the victim's dashboard then mounts all 60,000 cards synchronously. In the PoC the renderer and Manager's own `Open navigation` control stayed unresponsive for more than 14 seconds after the response had arrived, against 79 ms for an equal-size rejected control. The victim's Manager tab is temporarily unusable each time the name is opened.

IMM-WA-LOW-91822

Recipient field stays editable while a transfer is preparing, so the name goes to the replaced address

Report ID	#91822
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Explorer's transfer form hands the recipient in its input to `startTransfer`, whose `prepareMutation` looks up the token ID asynchronously and then saves `{ recipient, tokenId, options }` as the parameters every transfer step is built from ([useTransferName.ts#L109-L122](#)). While that lookup is pending the submit button is disabled and reads `Preparing...`, but `AddressNameInput` receives no `disabled` prop and stays editable ([SendNameForm.tsx#L123-L152](#)), and nothing compares the saved recipient with the field when the lookup returns. A user who corrects the address during preparation therefore sees the corrected recipient in the form while `safeTransferFrom` is encoded with the one they replaced. The transaction modal names neither address, and in the executed PoC the ERC-1155 name token reached the replaced address while the form still showed the correction. The application displays one recipient and submits an irreversible transfer to another.

Severity note. The stale recipient is an address the user entered themselves, and a wallet that decodes `safeTransferFrom` shows it at the signing prompt, where the transaction can still be rejected.

IMM-WA-LOW-91997

Edit-profile save rewrites a previous owner's social records into the replacement resolver while promising to clear them

Report ID	#91997
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Injecting/modifying the static content on the target application without JavaScript (persistent), such as: HTML injection without JavaScript · Replacing existing text with arbitrary text · Arbitrary file uploads, etc

Description

When the owner of a transferred name cannot write its inherited resolver, Manager's Edit Profile dialog normalises every social record in the form and diffs the result against the raw on-chain snapshot ([EditProfileDialogBody.tsx#L73-L74](#)), then confirms with "Continuing clears them and keeps only the changes you save" before batching that diff into a replacement resolver ([ResolverSetupConfirmDialog.tsx#L38-L46](#)). The baseline, however, is never normalised, so any social record the previous owner stored in non-canonical form, such as `https://twitter.com/attacker` for `com.twitter`, differs from its display value and becomes a `setText` the new owner never requested. In the executed PoC, editing only `description` produced ten `setText` calls, nine of them carrying the previous owner's handles into the new owner's fresh resolver, which Manager and Portal then render as that profile's own social links. Attacker-chosen text is thereby persisted on-chain and displayed as the new owner's static profile content.

IMM-WA-LOW-92305

Email verification rate limit keyed on the recipient lets any account block a person's email enrollment

Report ID	#92305
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: Enabling/disabling notifications

Description

`POST /notifications/channels/email` caps verification mails at three per hour through `checkAndConsumeEmailVerificationRateLimit`, which stores its counter under the normalised recipient address ([email-verification-rate-limit.ts#L25-L30](#)). The limit, however, is keyed on the target rather than on the authenticated caller, and it runs only after the handler has already written the `user_channels` and `channel_verifications` rows ([email.ts#L74-L140](#)). Any authenticated account can therefore name a person's address three times and exhaust its quota, and while that address is not yet verified on another account the cross-account guard does not stop these requests. When the owner then tries to enrol, they receive a 429 and no verification email, and three requests per hour keep them from enabling email notifications indefinitely, with no interaction on their part.

IMM-WA-LOW-92544

Explorer name page tells any visitor they own the name from URL parameters, inducing paid renewals of another's name

Report ID	#92544
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Malicious interactions with an already-connected wallet, such as: Modifying transaction arguments or parameters · Substituting contract addresses · Submitting malicious transactions · Signing or submitting a transaction against the wrong chain (for example when the request's target chain is not validated) · Inducing a blanket token approval (for example setApprovalForAll on the registry or NameWrapper) to an attacker-controlled address

Description

The Explorer's `/$name/` route renders `RegistrationSuccessBanner`, which states "You are the owner of {name}" beside a registration term and a "Paid" figure, whenever the query string carries `registered=true`, accepting any finite `duration` and any `paid` string ([index.tsx#L51-L59](#), [RegistrationSuccessBanner.tsx#L26-L36](#)). Nothing on that path compares the visitor with the on-chain owner, although the same component queries that owner and displays it a few rows lower. A crafted link therefore makes the genuine Explorer tell any visitor, connected or not, that they own someone else's name. The page's primary `Extend` action carries no ownership condition ([#L374-L379](#)), and completing it sends `approve()` and `renew()` from the visitor's wallet; in the executed PoC the victim paid 8.021939 USDC to add a year to the attacker's registration, after dismissing the extend modal's generic non-owner disclaimer. The application thereby induces an already-connected wallet to submit a transaction on a premise it fabricated.

IMM-WA-LOW-92586

Text record keyed on an `Object.prototype` member crashes a name's Manager profile page

Report ID	#92586
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Taking down the application/website

Description

Manager resolves text-record definitions through `recordIndex`, a plain object built with `Object.fromEntries` and read by `getRecordDef` without an own-property check ([utils.ts#L11-L18](#), [#L82-L85](#)), and the profile read fetches every text key the indexer reports on the resolver. For a key such as `constructor`, however, `getRecordDef` returns an inherited `Object.prototype` member, which is truthy, and `transformProfileRecords` then spreads `acc[record.section]`, which is `undefined`, and throws a `TypeError` ([transformRecords.ts#L140-L146](#)). The transform runs while the profile page renders, so one `setText(node, "constructor", "x")` replaces `{name}` with the route's error component for every visitor. Because the Edit Profile dialog is mounted inside that page, a name transferred or sold in this state cannot be repaired in Manager and needs a direct resolver call. Until then, the name's Manager profile route is unavailable to everyone.

IMM-WA-LOW-93124

Predictable SIWE nonce consumed before verification lets an unauthenticated attacker lock a user out of login

Report ID	#93124
Severity	Low
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Temporarily disabling user to access target site, such as: Locking up the victim from login · Cookie bombing, etc.

Description

The api-worker issues SIWE login nonces from viem's `generateSiweNonce()` and consumes them in `verifyAndConsumeNonce`, the first step of `createJWT` ([auth/index.ts#L32-L55](#)). In viem 2.52.2 consecutive nonces are overlapping 96-character windows of one `Math.random()` buffer, so an attacker who requests a nonce just before and just after a victim recovers the victim's nonce verbatim from the union of the two. The nonce, however, is deleted from KV before the message, domain, URI or signature is checked ([#L74-L95](#)), so an unauthenticated `POST /auth/login` that names a candidate with no valid message or signature burns it. The victim's genuine wallet-signed login then fails with `Invalid nonce`; the PoC denied 20 consecutive attempts against the real router. Recovering the nonce needs attacker and victim on the same worker isolate, while the burn works from anywhere, and repeating the cycle locks the victim out of login for as long as the attacker continues.

IMM-WA-INSIGHT-89255

Telegram expiry notification interpolates ENS names unescaped into Markdown-parsed message text

Report ID	#89255
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Injecting/modifying the static content on the target application without JavaScript (persistent), such as: HTML injection without JavaScript · Replacing existing text with arbitrary text · Arbitrary file uploads, etc

Description

The `name-expiry` template interpolates `payload.name` into a message sent with `parseMode: 'Markdown'` without escaping Markdown metacharacters ([telegram.ts#L33-L39](#)), and the indexer-sourced name reaches it with no charset restriction. An attacker who registers a label containing backticks and asterisks and transfers the name to a victim can place attacker-authored bold text inside a genuine expiry alert from the project's bot, with no action by the victim.

IMM-WA-INSIGHT-89300

In-source SHA-256 comments for the patched `@rhinestone/sdk` disagree with each other and with the shipped patch

Report ID	#89300
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/packages/smart-account
Impact	—

Description

`manifest.ts` records the SDK patch SHA-256 as `5e0a5f32...` (`manifest.ts#L17`) and `session-storage.ts` logs a bump to `7603298e...`, but the shipped patch file hashes to `2731d300...`, the value `pnpm-lock.yaml` pins. No source comment identifies the patch that ships, so a reviewer checking which SDK build was audited, or whether stored sessions need re-authorization, gets a wrong answer; the lockfile still enforces patch integrity, with no runtime effect.

IMM-WA-INSIGHT-89371

Unverified `default.reverse` name shown as a forward-matched primary name on inheriting L2 rows

Report ID	#89371
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The reverse-resolution query fetches the `default.reverse` name with `allowMismatch: true` but keeps only its `name` and discards `match` ([useReverseResolution.ts#L236-L243](#)), so every L2 row that inherits it renders **True** under *Forward match* and a **Primary name** badge even when the forward check failed. Because `DefaultReverseRegistrar.setName` accepts any string, any address can make the Explorer present a name it does not own as its verified primary name, with no action from that name's owner.

IMM-WA-INSIGHT-89389

Email channel endpoint reveals addresses held by other users and rate-limits only the literal address string

Report ID	#89389
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: Enabling/disabling notifications

Description

POST /notifications/channels/email returns an *already linked to another account* error before any write, send or quota use ([email.ts#L59-L72](#)), and rate-limits only on the trimmed, lowercased address, so `+tag` variants of one inbox get fresh quotas. Any signed-in wallet can test whether an email belongs to an ENS user and send unbounded ENS mail to one inbox; a spam report on that mail can mark a victim's channel `bounced`, disabling their notifications, because the webhook matches channels by address alone.

IMM-WA-INSIGHT-89461

Temporary registry-wide HCA operator approval survives an interrupted migration and is never revoked

Report ID	#89461
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Direct theft of user NFTs

Description

The migration grants `ETHRegistry.setApprovalForAll(hca, true)` as a temporary approval, but `revokeTemporaryOperatorApprovals` runs only after `executeRemainingAtomicBatches` with no `try/finally` ([migrationService.ts#L1189-L1212](#)), and later sessions omit the already-granted approval from `preflight.migrationApprovals`, the list the revocation filters. A declined prompt, failed batch or closed tab therefore leaves a permanent operator approval from the EOA to its HCA over every v2 name it holds, which no Manager path revokes, exposing those names to transfer by anyone who later controls the HCA.

IMM-WA-INSIGHT-89469

OG image worker's self-host guard compares hosts as raw strings, so a trailing-dot host bypasses it

Report ID	#89469
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Temporarily disabling user to access target site, such as: Locking up the victim from login · Cookie bombing, etc.

Description

`checkFetchableUrl` blocks an avatar URL pointing back at the worker only when `url.host` string-equals the request's own host ([safe-fetch.ts#L88-L90](#)), but the URL parser keeps a trailing dot, so `app.ens.domains.` passes while resolving to the same worker. Two attacker-owned names whose avatars cross-point at each other's `/og/` card with alternating trailing dots make each preview request recurse through nested worker invocations until the 20-second fetch timeout aborts it, so those cards never render.

IMM-WA-INSIGHT-89534

Session-enable signature commits to session policy only through an opaque salt the signer cannot read

Report ID	#89534
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/packages/smart-account
Impact	Unauthorized takeover of a user's smart account (Hybrid Custody Account) or authenticated session, letting an attacker transfer the victim's names or funds or act as them, such as: adding, replacing, or escalating an account owner or co-owner · extending or widening a session key's authority beyond its intended scope or lifetime · stealing an authenticated session or bearer token · (Exploiting the 7-day session-key design as described in the Known Issues is not eligible; a weakness that grants authority beyond its stated lifetime or permissions is in scope.)

Description

`computeDestinationSessionSalt` folds the session's constraints (`validUntil`, resolver, refund token and three refund caps) into one `keccak256` salt (`session.ts#L107-L138`), while no readable field of the signed `MultiChainSession` payload carries policy, so an honest 24-hour enable and one authored with `validUntil = 2^48-1` differ only in opaque bytes. Script in the Manager origin at enable time can obtain the owner's signature for an attacker-keyed session that the deployed validator honors beyond the stated 24-hour lifetime and 100 USDC refund ceiling.

IMM-WA-INSIGHT-89599

Token page Namehash row hashes the punycode form of non-ASCII names, showing another name's node

Report ID	#89599
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

The Token Info page computes its **Namehash** row as `namehash(asciiEncode(name))`, where `asciiEncode` applies the URL parser's IDNA transform ([token.tsx#L286-L288](#)), so for any non-ASCII name it hashes a punycode `xn--` label that ENSIP-15 rejects rather than the name itself. The page therefore shows, beside a **Normalized** badge, the node of a different, separately registrable name; for `🚀.eth` that node is owned on mainnet by an unrelated third party.

IMM-WA-INSIGHT-89705

`getForwardResolutionRequest` targets the viewed page's address rather than the connected wallet

Report ID	#89705
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Causing the application to display or pre-fill an incorrect address, name, or transaction detail, so that a user sends funds or submits a transaction to an unintended recipient, such as: resolving or rendering an ENS name to an attacker-controlled address · showing the wrong recipient or amount on a transaction the user then approves

Description

`getForwardResolutionRequest` uses the route's address, not the signer, as the `setAddr` target ([useReverseResolutionMutations.ts#L121-L131](#)), and the sidebar's **Set primary name** button has no identity check; only the table's `canModify` gates it, rendering the **More** control that opens the sidebar only when the connected account equals the page address. An owner's forward record rests on that display gate alone: signing the builder's request for another address's page sets the name's ETH record to that address.

IMM-WA-INSIGHT-91732

Explorer homepage activity feed renders attacker-chosen text-record keys verbatim

Report ID	#91732
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immune-fi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Injecting/modifying the static content on the target application without JavaScript (reflected), such as: Reflected HTML injection · Loading external site data

Description

For `TextChanged` events, `formatActivityEvent` appends the `key` field of the event's JSON data to the row text, guarded only by truthiness, with no allowlist, length cap or character filter ([formatActivityEvent.ts#L114-L118](#)). Since `setText` accepts any key on a name the attacker controls, and assuming the indexer relays the key verbatim, one transaction places arbitrary plain text on the Explorer landing page's unfiltered Recent Activity feed, shown to every visitor.

IMM-WA-INSIGHT-92307

API worker persists arbitrarily large transaction payloads with no size limit

Report ID	#92307
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	Taking down the application/website

Description

POST /transactions validates payload only as a record of string keys to unknown values (TransactionPayloadSchema) and writes it straight to the database on insert and on conflict (transactions/index.ts#L32-L47), and the worker applies no body size limit. An authenticated user can submit one oversized request; in a self-hosted test deployment, a single ~30 MB payload left the API and its PostgreSQL database saturated and unresponsive to other users for over 20 minutes.

IMM-WA-INSIGHT-92553

Explorer homepage feed presents unverified `NameChanged` reverse-record strings as genuine ENS name badges

Report ID	#92553
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/portal
Impact	Injecting/modifying the static content on the target application without JavaScript (persistent), such as: HTML injection without JavaScript · Replacing existing text with arbitrary text · Arbitrary file uploads, etc

Description

For `NameChanged` events, `formatActivityEvent` copies `data.name` into `entityFromData` without any forward-resolution check ([formatActivityEvent.ts#L132-L140](#)), and because reverse nodes carry no indexed name, the Recent Activity table renders that value with the same name badge and `/<name>` link as genuine activity. Any address can set its reverse record to a name it does not own, such as `vitalik.eth`, and appear on the Explorer homepage as that name updating its primary name; one account replaced all 15 feed rows this way.

IMM-WA-INSIGHT-92954

Avatar and header uploads accepted from a wallet that does not own the target name

Report ID	#92954
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/apps/manager
Impact	Changing non-sensitive details of other users (including modifying browser local storage) without already-connected wallet interaction and with up to one click of user interaction, such as: Enabling/disabling notifications

Description

`uploadSignedImage` sends only `expiry`, `dataURL`, `sig` and `unverifiedAddress`

([profileImageUpload.ts#L156-L161](#)), so the target name travels only in the endpoint path and the signed `Upload` message, neither of which ties the signer to the name's owner. On staging, a fresh wallet that did not own a Sepolia name signed an upload for it and the service returned `200` and served its image, so any wallet can replace another name's uploaded avatar or header.

IMM-WA-INSIGHT-92979

Telegram auth verification accepts future-dated `auth_date` values

Report ID	#92979
Severity	Insight
Category	Websites & Apps
Asset	https://github.com/immunefi-team/audit-comp-ens/tree/audit-comp-ready/worker_s
Impact	—

Description

`verifyTelegramAuth` rejects auth data only when `Date.now() - data.auth_date * 1000` exceeds `maxAge` ([auth.ts#L66-L71](#)), so a future `auth_date` yields a negative age and passes the freshness check. A correctly signed payload dated in the future therefore stays valid beyond the `maxAge` window; producing one still requires the bot token, so the gap is not exploitable on its own.