CONTRACT WOLF

Security Assessment

Project 32 Dapp

Verified on 02/01/2025

SUMMARY

Project	CHAIN	METHODOLOGY
Project32	Ethereum	Manual & Automatic Analysis

FILES	DELIVERY	TYPE
Single	1/11/2024	Dapp Audit

6	0	0	0	2	4	0
Total Findings	Critical	Major	Medium	Minor	Informational	Resolved

🟥 0 Critical	An exposure that can affect the dapp's functions in several events that can risk and disrupt the code
🟧 0 Major	An opening & exposure to manipulate the code in an unwanted manner
🟧 0 Medium	An opening that could affect the outcome in executing the code in a specific situation
⬜ 2 Minor	An opening but doesn't have an impact on the functionality of the code
🟦 4 Informational	An opening that consists information but will not risk or affect the code
🟩 0 Resolved	ContractWolf's findings has been acknowledged & resolved by the project

STATUS ✔️**AUDIT PASSED**

TABLE OF CONTENTS | Project 32

DISCLAIMER | Project 32

ContractWolf audits and reports should not be considered as a form of project's "Advertisement" and does not cover any interaction and assessment from "Project Code" to "External Code"

ContractWolf does not provide any warranty on its released report and should not be used as a decision to invest into audited projects.

ContractWolf provides a transparent report to all its "Clients" and to its "Clients Participants" and will not claim any guarantee of bug-free code within its **DAPP**.

ContractWolf's presence is to analyze, audit and assess the Client's Dapp to find any underlying risk and to eliminate any logic and flow errors within its code.

Each company or project should be liable to its security flaws and functionalities.

SCOPE OF WORK | Project 32

Project 32's team has agreed and provided us with the files that need to be tested. The scope of audit is the main dapp.

The goal of this engagement is to identify if there is a possibility of security flaws in the implementation of dapp and its systems.

ContractWolf will be focusing on dapp issues and functionalities along with the project claims from smart contract to their website, whitepaper, repository which has been provided by **Project 32**.

AUDITING APPROACH | Project 32

Every line of code along with its functionalities will undergo manual review to check for security issues, quality of logic and dapp scope of inheritance. The manual review will be done by our team that will document any issues that they discovered.

METHODOLOGY

The auditing process follows a routine series of steps :

1. Code review that includes the following :
   - Review of the specifications, sources and instructions provided to ContractWolf to make sure we understand the size, scope and functionality of the DAPP.
   - Manual review of code. Our team will have a process of reading the code line-by-line with the intention of identifying potential vulnerabilities, underlying and hidden security flaws.

2. Testing and automated analysis that includes :
   - Testing the DAPP's function with common test cases and scenarios to ensure that it returns the expected results.

3. Best practices and ethical review. The team will review the dapp with the aim to improve efficiency, effectiveness, clarifications, maintainability, security and control within the dapp.

4. Recommendations to help the project take steps to eliminate or minimize threats and secure the dapp.

TOKEN DETAILS | Project 32

Project 32 is a groundbreaking initiative on Solana, merging blockchain, artificial intelligence, and decentralized finance to democratize access to validator nodes and Maximal Extractable Value (MEV).

Token Name	Symbol	Decimal	Total Supply	Chain
Project 32	32	9	-	Solana

SOURCE

Source	*Sent Via local-files*

FINDINGS | Project 32

6	0	0	0	2	4	0
Total Findings	Critical	Major	Medium	Minor	Informational	Resolved

This report has been prepared to state the issues and vulnerabilities for Project 32 Dapp through this audit. The goal of this report findings is to identify specifically and fix any underlying issues and errors

Backend *(index.tsx)*

ID	Title	File & Line #	Severity	Status
DCW-011	Insecure API Usage	L: 65	Informational	• Pending
DCW-006	Potential SQL Injection	L: 189	Minor	• Pending
DCW-017	Request Limit	--	Informational	• Pending

Frontend*(claim-feature.tsx)*

ID	Title	File & Line #	Severity	Status
DCW-018	Overflow or Precision Loss	L: 20	Informational	• Pending
DCW-012	Error Handling	L: 107	Informational	• Pending
DCW-019	Unintended Behavior	--	Minor	• Pending

PENETRATION ATTACKS | Project 32

Dapp Weakness Classification and Test Cases

ID	Description	Status
DCW-001	Malware Scan	Passed
DCW-002	Phishing	Passed
DCW-003	Missing HTTP Headers	Passed
DCW-004	Valid SSL Certificate	Passed
DCW-005	Firewalls(Drop & Deny)	Passed
DCW-006	Potential SQL Injection	Minor
DCW-007	Framework Version	Passed
DCW-008	Gas Griefing	Passed
DCW-009	Address Approval	Passed
DCW-010	Address Draining	Passed
DCW-011	Insecure API Usage	Informational
DCW-012	Error Handling	Informational
DCW-013	Memory Leak	Passed
DCW-014	Lack of Input Validation	Passed
DCW-015	Potential Backdoor	Passed
DCW-016	Sensitive Data Exposure	Passed
DCW-017	Request Limit	Informational
DCW-018	Overflow or Precision Loss	Informational
DCW-019	Unintended Behavior	Minor

FIXES & RECOMMENDATION

DCW-011 | Insecure API Usage

The code uses `axios` to make API calls to `/api/getClaim` and `/api/claimConfirm` without ensuring the endpoints are secure (e.g., using HTTPS). Additionally, sensitive data like publicKey are sent in plaintext if `HTTPS` is not enforced.

Recommendation*(or logic)*:

- Ensure all API endpoints use HTTPS to encrypt data in transit.
- Validate the backend API URL to prevent SSRF (Server-Side Request Forgery) attacks:

```
const backendUrl = process.env.NEXT_PUBLIC_API_URL;
if (!backendUrl || !backendUrl.startsWith("https://")) {
  throw new Error("Invalid API URL");
}
```

DCW-006 | Potential SQL Injection

The Code is using raw SQL queries, which may introduce SQL injection risks:

```
const [existingClaim] = (await prisma.$queryRawUnsafe(
  `SELECT * FROM wallets WHERE wallet_address = $1 FOR UPDATE`,
  walletAddress
)) as Wallet[];
```

While Prisma protects against SQL injection, `$queryRawUnsafe` can be dangerous if user input is concatenated improperly.

Recommendation*(or logic)*:
Use Prisma's parameterized queries securely:

```
const [existingClaim] = await prisma.$queryRaw<Wallet[]>(
  `SELECT * FROM wallets WHERE wallet_address = ? FOR UPDATE`,
  walletAddress
);
```

DCW-006 | Request Limit

Backend's endpoints allow unlimited requests making them vulnerable to brute force attacks.

Enforce a code that will protect against **denial-of-service (DoS)** attacks.

Recommendation*(or logic)*:
Use a rate limiter, such as `express-rate-limit`:

```
import rateLimit from "express-rate-limit";

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100, // limit each IP to 100 requests per windowMs
  message: "Too many requests, please try again later."
});

app.use(limiter);
```

DCW-018 | Overflow or Precision Loss

The function `secsToDate(secs: BN)` converts a **BN** (BigNumber) to a number before formatting. If the number is too large, JavaScript's `Number` type could cause precision loss.

Recommendation*(or logic)*:

Use `.toString()` for safer conversion:

```
function secsToDate(secs: BN) {
  try {
    const timestamp = parseInt(secs.toString(), 10);
    if (isNaN(timestamp) || timestamp < 0) return "Invalid date";
    return datetimeFormat.format(new Date(timestamp * 1000));
  } catch (error) {
    console.error("Error formatting date:", error);
    return "Invalid date";
  }
}
```

DCW-012 | Error Handling

If an error occurs during `handleClaim()`, it logs the error but provides **no feedback** to the user.

Recommendation*(or logic)*:

Show an error toast using the `useToast()` hook:

```
const toast = useToast();

const handleClaim = async () => {
  try {
    setIsClaimLoading(true);
    await claimToken.mutateAsync();
    toast.success("Tokens successfully claimed!");
  } catch (error: any) {
    console.error("Claim error:", error);
    toast.error("Error claiming tokens. Please try again.");
  } finally {
    setIsClaimLoading(false);
  }
};
```

DCW-019 | Unintended Behavior

Claim Button May Be Enabled When It Shouldn't

- The button logic disables claiming **only in some cases**, but a **malicious user could still trigger the claim process manually** in the browser console.

Recommendation*(or logic)*:

- Add **backend validation** to verify claim eligibility.
- Add **explicit frontend validation**:

```
const isClaimDisabled =
  isClaimLoading ||
  !locking.wallet_address || // Ensure valid wallet
  locking.token_amount <= 0 || // Ensure claimable amount exists
  (locking.transfer_status !== "processing" &&
    locking.claimed_at !== null &&
    locking.transfer_status?.toLowerCase() !== "failed");
```

AUDIT COMMENTS | Project 32

Dapp audit comment for a non-technical perspective

🟢 Project has been marked as SAFE to be interacted with by any SVM wallets *(02-01-2025)*
🟢 DAPP has no backdoors
🟢 DAPP cannot drain wallets via approval

CONTRACTWOLF

Blockchain Security - Smart Contract Audits