프로토타입 오염 패턴 조사를 통한 Node.js 패키지 취약점 분석 연구

최지혁* 강성현** 박성진*** 이동하*** 김찬호*** 김민욱**** 이수영****

*가천대학교(학부생), **LIG넥스원(연구원), ***LIG넥스원(선임연구원),
***가천대학교(학부생), ***AI-Ver.Se labs(연구원), ****고려대학교(학부생),
****한국정보보호교육센터(선임연구원)

Node.js Packages Vulnerability Analysis via A Survey : Prototype Pollution pattern

Ji-Hyeok Choi* Seong-Hyeon Kang** Sung-Jin Park*** Dong-Ha Lee*** Chan-Ho Kim*** Min-Uk Kim**** Su-Young Lee****

*Gachon University(Undergraduate student), **LIG Nex1(Rearch Engineer), ***LIG Nex1(Rearch Engineer), ***Gachon University(Undergraduate student), ***AI-Ver.Se labs(연구원), ****Korea University(Undergraduate student), ****KISEC(Rearch Engineer)

요 약

전 세계적으로 Node.js의 인기가 증가하며 웹 어플리케이션 개발에 있어서 빼놓을 수 없는 요소가 되었다. Node.js에서 사용되는 언어인 자바스크립트는 클래스라는 개념이 없는 대신에 기존의 객체를 복사해 새로운 객체를 생성하는 프로토타입 기반의 언어이다. 본 논문에서는 자바스크립트의 간단한 이론과 자바스크립트의 기반이 되는 프로토타입에 대해 전반적인 이론을 다뤄보고자 한다. 또한 프로토타입에서 발생되는 취약점인 프로토타입 오염이 발생하는 원인 및 패턴에 대해 분석하고 우리가 개발한 툴을 이용해 찾은 취약점에 대해 소개한다.

I. 서론

자바스크립트는 브라우저에서 동적이고 유연한 기능을 가진 가장 인기 있는 프로그래밍 언어 중 하나로 클라이언트 사이드 언어로써 사랑을 받아왔다. 최근 몇 년 동안에는 REST API를 뒷받침하는 서버 사이드 언어로서도 인기가 상승해 풀스택 웹 애플리케이션을 생성할 수 있는 언어가 되었다.[1] 인기가 있는 만큼 취약점이 발생한다면 보안에 심각한 위협을 일으킬 수 있다. 본 논문에서는 자바스크립트에서 발생하는 취약점 중 하나인 프로토타입 오염에 대해서 중점적으로 살펴본다. 이 외에도 ReDos나 Cross-Site Scripting 같은 공격들도 많지만 본 논문에서는 코드 성격에 따라 Dos뿐 아니라 Remote Code Execution까지 이어질 수 있는 프로토타입 오염에 취약점을 중점적으로 다룬다.[2] 우리는 프로토타입 오염이 발생되는 함수들의 특징을 분석한 후에 라이브러리를 대상으로 프로토타입 오염 취약점을 탐지하는 도구를 만들었다. 도구를 통해 4개의 취약점을 탐지했고, 현재 CVE-2023-45827을 발급받았다.

II. 배경지식

2.1 자바스크립트 동작 방식

자바스크립트는 싱글 스레드 기반으로 프로토타입 객체를 상속하는 객체지향언어이다. 자바스크립트를 실행하기 위해 인터프리터가 필요한데 구글의 V8엔진, 사파리의 Webkit, 파이어폭스의 SpiderMonkey 등이 있다. 자바스크립트 엔진의 주요 구성요소로는 메모리 힙과 콜스택이 있다. 코드에서 읽어내려가며 수행할 작업들을 콜스택에 쌓고, 메모리 힙에서 작업 수행에 필요한 것들을 찾아 작업을 수행한다. 싱글 스레드이기 때문에 콜 스택도 한개 뿐이라 Dos공격에 상당히 취약하다.

2.2 프로토타입

자바스크립트는 유연하고 동적으로 언어를 사

용하기 위해 프로토타입이라는 객체를 이용한다. 프로토 타입이란 모든 객체들이 함수와 속성들을 상속 받기 위한 템플릿으로 프로토타입 객체를 가진다는 의미이다. 프로토타입 객체는 또 다시 상위 프로토타입 객체로부터 함수와 속성을 상속받을 수도 있고 그 상위 프로토타입 객체도 같다. 자바스크립트는 특정 객체의 프로퍼티나 함수에 접근하려고 할 때, 해당 객체에 접근하려는 프로퍼티나 함수가 없다면 프로토타입이 가리키는 링크를 따라 자신의 부모 역할을 하는 프로토타입 객체의 프로퍼티나 함수에 접근을 하는데 이런 행위를 프로토타입 체인이라고 한다.

2.3 프로토타입 오염

프로토타입 오염 취약점이란 2.2장에서 언급한 프로토타입 체인 행위에 의해 발생되는 취약점이다.[3]

```
var victim={};
var obj={};
console.log(victim.pp); // undefined
obj['__proto__']['pp']='polluted';
console.log(victim.pp); // polluted
```

(그림 1) Prototype Pollution

(그림 1)을 보면 victim객체와 obj객체가 빈 객체로 선언이 되었다. 그러나 obj에 __proto__속성을 부여해줘 pp라는 프로퍼티에 'polluted'라는 값을 대입해줬다. 그 결과 victim객체에서 pp 프로퍼티에 접근하면서 프로토타입 체인 행위가 일어나 프로토타입 객체의 pp에 접근해 'polluted'가 출력된 것을 확인할 수 있다. 해당 취약점이 클라이언트 사이드에서 발생한다면 Cross-Site Scripting 취약점으로 이어질 수 있다. 또 한 서버 사이드에서 발생한다면 Remote Code Execution 취약점으로 이어질 수 있다.[4]

III.연구

3.1 프로토타입 오염 취약점 패턴

우리는 취약점 분석을 하기에 앞서 프로토타입 오염이 자주 나오는 패턴과 기존에 제보되었던 취약점의 패턴에 대해 분석했다.

```
function merge(target, source) {
    for (let key in source) {
      if(Object.is(target[key]) &&
Object.is(source[key])) {
        merge(target[key], source[key]);
      } else {
        target[key] = source[key];
      }
    }
    return target;
  }
```

(그림 2) deep merge algorithm

(그림 2)는 두 객체를 병합하는 deep merge 알고리즘이다. source의 인자로 __proto__가 존재한다면 (그림 1) 예시처럼 __proto__ 속성이 부여되어 프로토타입 오염 취약점이 발생한다.

```
module.exports = function(obj, prop, val) {
  if (!isObject(obj)) {
    return obj;
  }
  if (Array.isArray(prop)) {
    prop = [].concat.apply([], prop).join('.');
  }
  if (typeof prop !== 'string') {
    return obj;
  }
  var keys = split(prop, {sep: '.', brackets:
true});
  var len = keys.length;
  var idx = -1;
  var current = obj;
  return obj;
};
```

(그림 3) CVE-2021-23440

(그림 3)은 프로토타입 오염 취약점이 발생한

set-value함수 코드이다. 첫 번째 인자인 obj객체에 두 번째 인자 prop를 key로 넣고 세 번째인 val을 value로 넣어주는 알고리즘 이다. prop에 __proto__가 존재하면 obj객체에 __proto__ 속성이 부여되어 프로토타입 오염이 발생한다.

(그림 2), (그림 3)을 통해 취약점이 발생하는 패턴분석 결과 프로토타입 오염이 발생하는 함수들의 특징을 확인할 수 있었다. merge 역할을 하는 함수의 경우, 일반적으로 target, source 두 개를 인자로 갖고 있다. 그리고 set 역할을 하는 함수의 경우, target, key, value 세 개의 인자를 가지고 있는 패턴을 확인했다. 우리는 앞의 분석을 통해 라이브러리를 대상으로 프로토타입 오염을 진단하는 도구를 만들어 유의미한 결과를 도출했다.

3.2 취약점 분석

3.1장의 연구를 통해 프로토타입 오염이 발생하는 패턴을 파악하고 발견한 패턴을 바탕으로 Node.js 패키지에 취약점을 탐지 해보았다.

```
var modls=Object.keys(mod);
for(i in modls){
    var obj={};
    func=mod[modls[i]];
    cnt=func.length;
    try{
        func(obj, {'__proto__.polluted':
'nodeBoB!'});
        if(obj['polluted']==='nodeBoB!'){
            console.log('[+] prototype_pol
lution_success on',modls[i]);
            console.log('{}.polluted
:',{}.polluted);
            delete obj.__proto__.polluted;
            continue;
        }
    }catch(err){
    }
    try{
        func(obj, '__proto__.polluted',
'nodeBoB!');
        if(obj['polluted']==='nodeBoB!'){
            console.log('[+] prototype_pol
lution_success on',modls[i]);
            console.log('{}.polluted
:',{}.polluted);
            delete obj.__proto__.polluted;
            continue;
        }
    }catch(err){
    }
}
```

(그림 4) pp detection tool

(그림 4)는 3.1장 연구를 통해 발견한 패턴을 기반으로 만든 취약점 점검 도구 중 패턴 탐지 알고리즘이다. 모듈이 내장하고 있는 모든 함수를 추출해낸 후, __proto__속성을 부여하며 함수를 호출하는 알고리즘이다. 프로토타입 오염을 발생하면 함수 명과 오염된 프로퍼티를 출력한다. 우리는 (그림 4)와 같은 알고리즘을 통해 4개의 라이브러리에서 4개의 취약점은 탐지했고 현재 CVE-2023-45827을 발급받았다.

IV.결론

본 논문은 자바스크립트의 동작 방법과 프로토타입 이론을 다루는 것부터 시작해서 프로토타입 오염이 발생하는 패턴 분석에 대한 내용을 다룬다. 그리고 분석 결과를 토대로 프로토타입 오염 취약점을 탐지하는 것을 목표로 하였다. 자바스크립트는 최근 매우 각광받는 언어가 되었으며, 타 언어에 비해 높은 점유율과 성장률을 보이고 있다.[5] 클라이언트 사이드 뿐 아니라 서버 사이드 또한 점유율이 상승하고 있는 만큼 Node.js 패키지에 대한 보안이 중요하다. 본 논문에서 다룬 클라이언트와 서버 사이드 모두에서 발생하는 취약점에 대한 연구결과는 Node.js 패키지들을 사용하는 데 있어 안정성을 한 층 더 높일 수 있을 것을 기대한다.

[참고문헌]

[1] Doglio, Fernando, “RESP API Development with Node.js”

[2] Jianwei Hou, “Detecting Node.js Prototype Pollution Vulnerabilities via Object Lookup Analysis”, ACM , Aug. 2021

[3] Olivier Arteau, “Prototype pollution attack in NodeJS application”, NSec, Sep. 2018

[4] Mikhail Shcherbakov, “Silent Spring: Prototype Pollution Leads to Remote Code Execution in Node.js”, USENIX, 2023

[5] 정주영, “Design and Implementation of High Availability Web Application Server System Architecture Based on Node.JS“, 한국정보기술학회 , Dec. 2017