A brief analysis on trust-based software development

Author - Rodrigo Lima

DETI, University of Aveiro

June 29, 2022

Abstract

The main objective of this report is to reveal the security flaws on the current state on the trust-based software development industry and their social and economic impacts.

Modern software development relies on 3rd-party packages from non-commercial entities, this dependency is based only on trust and status of the maintainer of the given package. This trust poses a threat to modern security.

Given the current growth on reliance on package managers, a threat actor[1] with the right access to a dependency that a given software relies, allows the compromise of the digital infrastructure of any party that relies on that given software, causing a supply chain attack.

Social and economic impacts of this attacks can be attributed to the lack of security on the less-secure elements of the software supply chain, being therefore highly effective on breaching large to medium scale companies, creating havoc on the software industry and affiliated industries.

Keywords— Dependencies, Supply Chain, Security, Vulnerability, npm

Contents

List of Figures

1 Introduction

Current software development relies on open source code developed by third parties. This reliance enables faster software development lifecycles and more robust code, given that in any given task there might be an open source code variant that better approaches the given task compared to if developed in house.

Although the new approach to software development has its advantages, it also has some disadvantages. The security robustness of these products has not increased as their adoption has grown so dramatically, this growth can be seen on Figure 1.

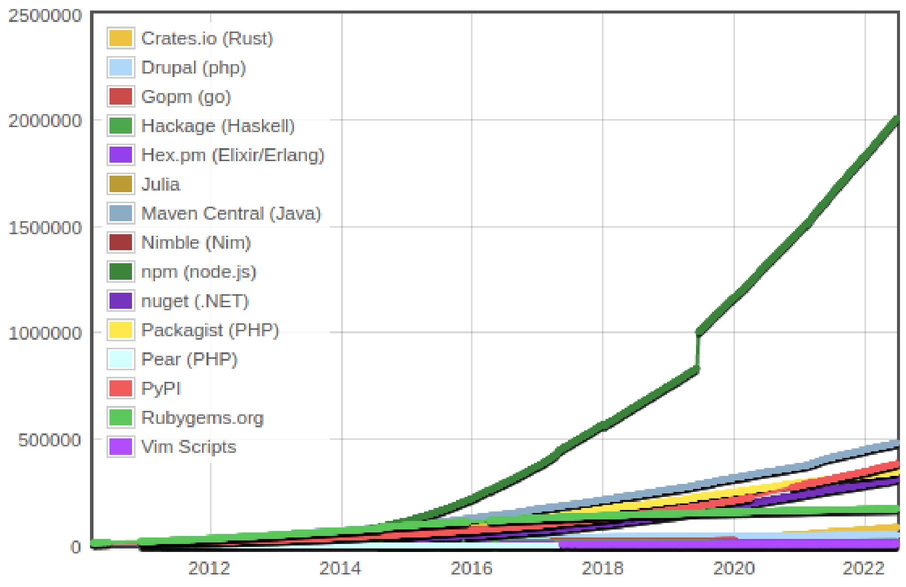


Figure 1: Package Managers Usage Growth[2]

The software supply chain[3] is the selection, development, and distribution of components, or pre-made pieces of software, through the development process until they become finished software. In physical terms, this is similar to how raw materials are transformed into finished products, then put on store shelves.

The dramatic growth on reliance on package managers[4] and dependencies[5] leads to a big attack surface on products that rely on those. If compromise, the supply chain of a given product could be at risk.

With the large interconnection of products and software in the current state of the industry, a single compromise of a product can lead to exponential compromises of products that rely on it. Causing therefore a cascading effect of compromises and affected products even if not directly related to the source of the problem.

While mitigations for this security issue are being taken, the rapid adoption of dependencies is exposing new attack vectors as well as old security attacks due to a lack of formation for developers in this area.

2 Security aspects

The use of 3rd-party packages from non-commercial entities may compromise the security of modern software development. Complex supply chains are also inherent in this software, which exposes vast attack surfaces to threat actors.

A number of components are included in the software supply chain, as shown in Figure 2. Since every component is an attack surface, the final product is only as secure as the least secure component in the chain.

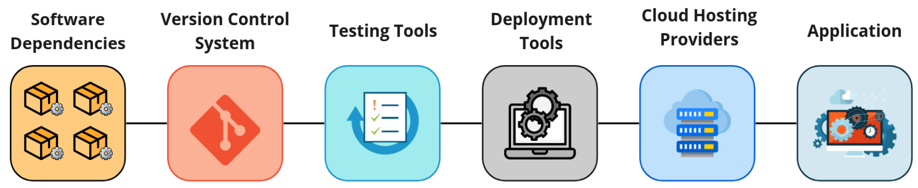


Figure 2: Attack Surfaces in Software Supply Chain

Software dependencies seem to be the weak link in the supply chain at the moment. Since other components are built with security in mind, whereas dependencies are mainly created and maintained by single individuals, this explains the difference.

By being the weakest link in the supply chain, multiple attacks where developed to compromise this component. This attacks range from technical to human engineering.

2.1 Domain Hijacking Attack

Domain hijacking or domain takeover is an attack based on the change of ownership of a domain name without the consent of the owner or by abusing the privileges of the domain's regulatory bodies.

A supply chain attack could be a result of this attack since users can use their domain on the package registration form. This can lead to a change of ownership of the package if the original creator loses ownership of the domain used on the email registered.

A change of domain ownership can be caused by the original owner forgetting to renew their domain name. This may lead to a malicious actor buying it and setting up an email server for this domain and being able to change ownership of the packages created by the original owner. An example of this attack can be seen on Figure 3.

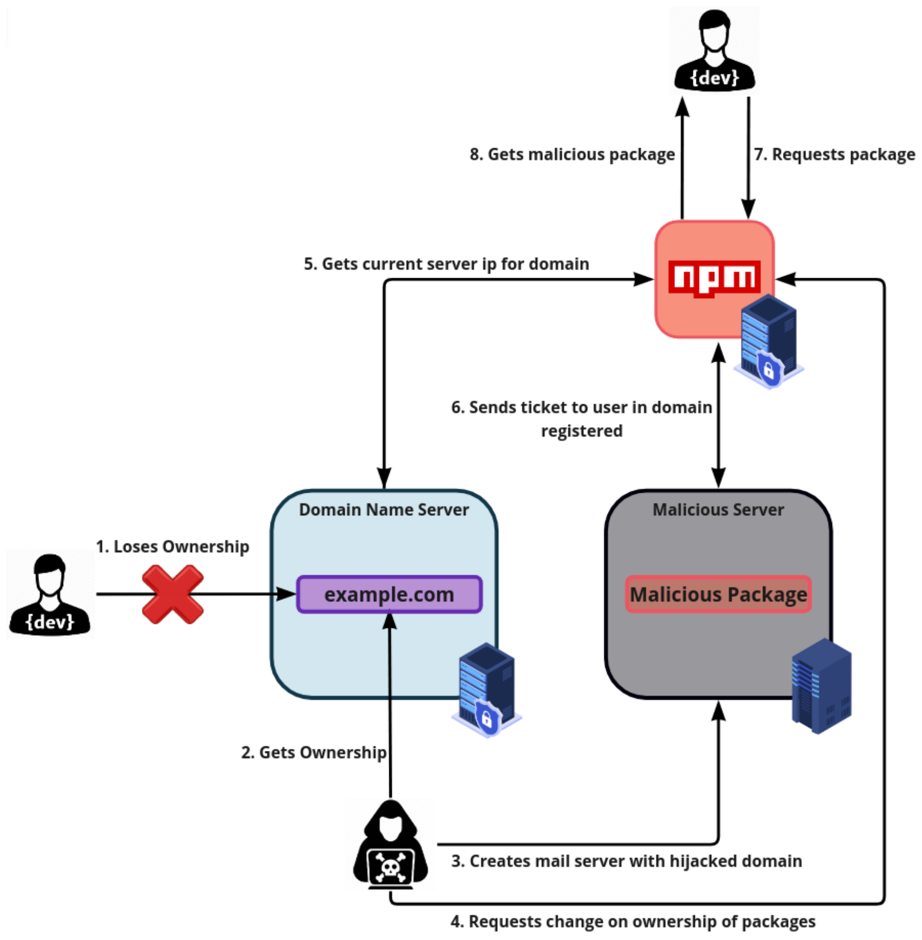

{dev}
8. Gets malicious package
7. Requests package
5. Gets current server ip for domain
npm
6. Sends ticket to user in domain registered
Domain Name Server
Malicious Server
1. Loses Ownership
{dev}
example.com
Malicious Package
2. Gets Ownership
3. Creates mail server with hijacked domain
4. Requests change on ownership of packages

Figure 3: Domain Hijacking Attack

2.2 Dependency Confusion Attack

Dependency confusion attacks are attacks that rely on the behavior of a package manager when multiple viable choices of a given project are given to be distributed.

When a developer requests a package that's not publicly accessible but only reachable inside an internal repository, the package manager supplies that private package. But if the package is reachable internally and publicly, then the package manager will choose the package with the highest version, not giving therefore a priority to the package present in the internal repository.

This unintuitive behavior of the package manager can cause security issues. Due to the use of internal repositories by big companies, if a developer uses a package not publicly available and a malicious actor publishes a package with the same name but with a higher version, then that malicious package would gain priority and enable malicious code to be run instead of the original. An example of this attack can be seen on Figure 4.

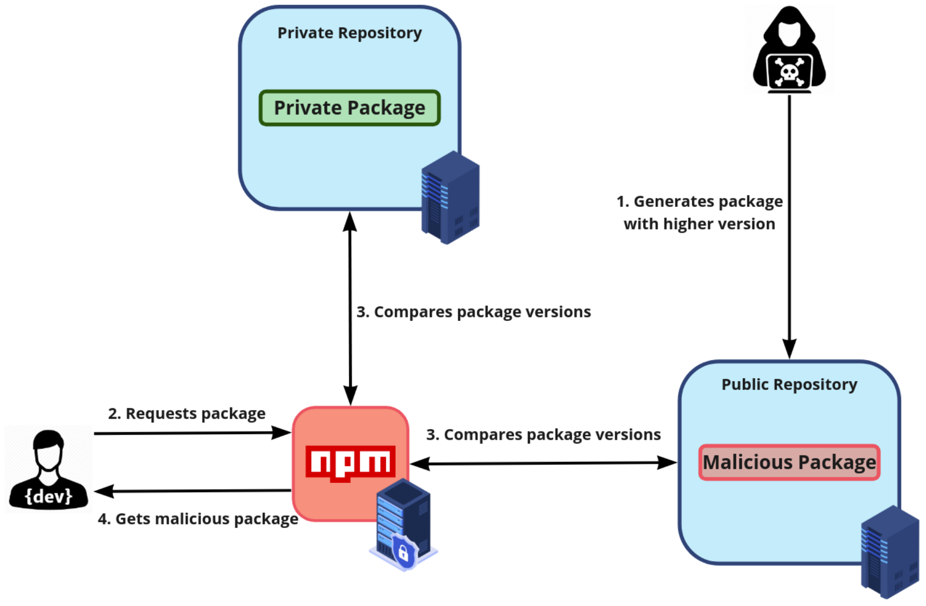

Private Repository
Private Package
1. Generates package with higher version
3. Compares package versions
2. Requests package
npm
3. Compares package versions
Public Repository
Malicious Package
{dev}
4. Gets malicious package

Figure 4: Dependency Confusion Attack example

2.3 Typosquatting Attack

Typosquatting is a type of social engineering attack that can target software packages. It involves imitating a legitimate package by using a package name similar to an official and trusted package. An example of this attack can be seen on Figure 5.

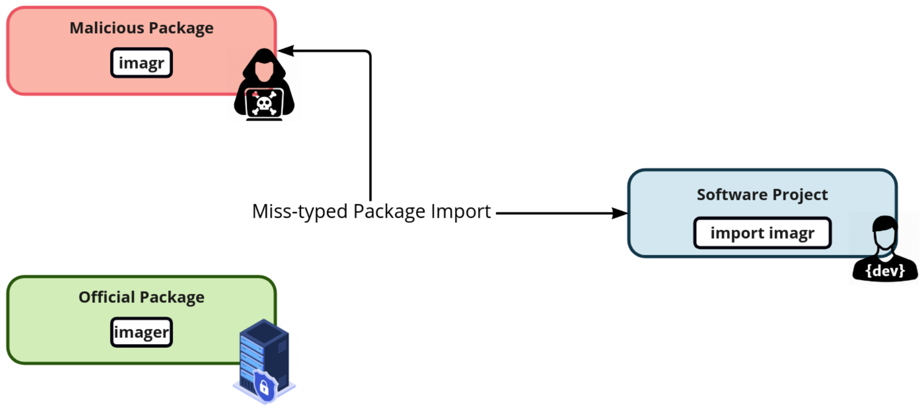


Figure 5: Typosquatting attack example

Due to the notoriety of the targeted package, the hacker is able to get this package to be used. A victim that lacks vigilance and does not systematically observe the package name can find themselves using malicious code instead of the pretended package.

2.4 Mitigations

Supply chain attacks have multiple attack vectors, but mitigation is possible. Key concerns should be addressed to mitigate supply chain security attacks. Some mitigations can be the following:

- Packages that are abandoned should be migrated from or taken over
- Packages used shouldn't be too recent, they should have a lengthy track record of security
- Unexpected behaviors and inconsistencies should be reported
- Packages should be reviewed before being upgraded
- Environment variables should be segregated to their step in the software lifecycle, they shouldn't be the same throughout the whole lifecycle
- Developers should be educated on the attack surfaces and vectors presented throughout the whole lifecycle
- Span between dependency update tools that pull request (PR) updates and production usage should have enough time so that the new version can be reviewed and verified

By keeping these mitigations in mind, most attacks on the supply chain can be mitigated while also securing other parties involved.

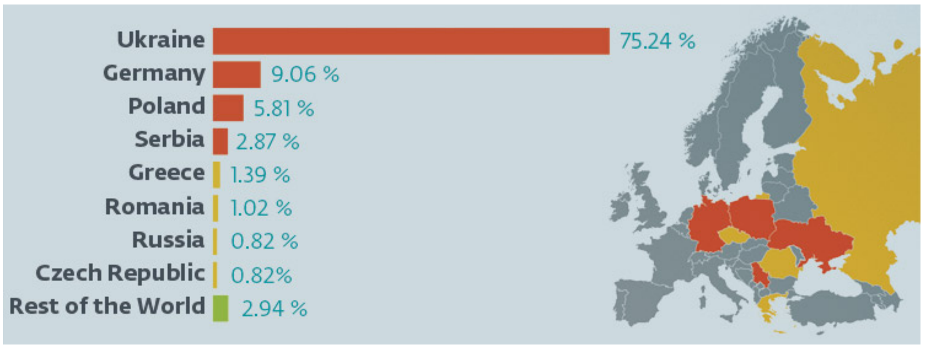
3 Social and Economic Impacts

The software development environment is not new to compromises in the supply chain. A significant number of compromises are traced to supply chain attacks, and large numbers of these compromises have real-world ramifications outside of the software development environment.

3.1 NotPetya / M.E.Doc

It was discovered in spring 2017 that the core code of the financial package "M.E.Doc" used in Ukraine was infected with the NotPetya virus and was subsequently downloaded by subscribers. An attacker possibly hacked the provider's code or rerouted download requests to a different server on the provider's system. According to press reports at the time, it was a supply chain attack, but the attack vector wasn't specified.

Ukraine felt its impact the most, but its area of effect was global. Over 300 companies were taken down by NotPetya in a matter of hours. An estimated $10 billion in damages were caused by this virus, which spread extremely quickly and reached 65 countries. Images of the consequences can be seen on Figures 6.



(a) NotPetya infection spread[6]

(b) NotPetya infected terminals

Figure 6: NotPetya compromise consequences

3.2 SolarWinds

In 2020, the global supply chain is believed to have been attacked by an assault on the IT infrastructure business enterprise SolarWinds.

Before FireEye detected the cyber breach in December 2020, SolarWinds Orion software had been compromised in March and June 2020. Out of the 33,000 customers who used SolarWinds Orion, 18,000 were vulnerable due to the compromise of the software, causing havoc to the modern industry.

The timeline of the attack can be seen below on Figure 7:

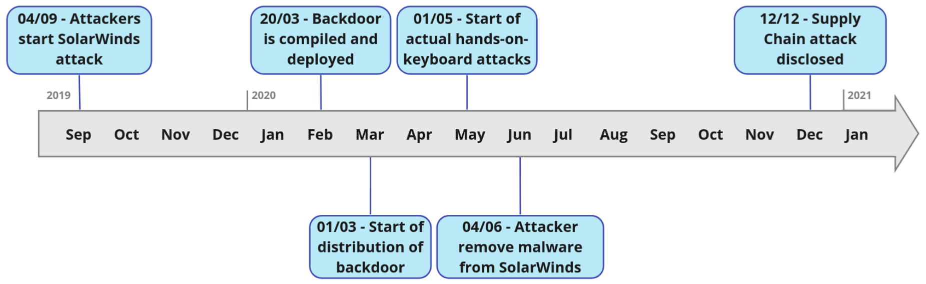

04/09 - Attackers start SolarWinds attack
20/03 - Backdoor is compiled and deployed
01/05 - Start of actual hands-on-keyboard attacks
12/12 - Supply Chain attack disclosed
2019
2020
2021
Sep Oct Nov Dec Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec Jan
01/03 - Start of distribution of backdoor
04/06 - Attacker remove malware from SolarWinds

Figure 7: SolarWinds breach timeline

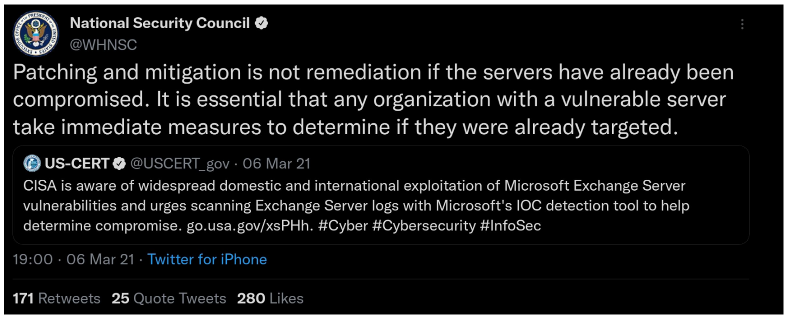
3.3 Microsoft Exchange Server

According to Microsoft, attackers gained access to a subset of Azure, Intune, and Exchange components in February of 2021. The previous subset of components contained no production credentials, the repositories were secured in December, and attacks ceased in January.

A backdoor was used to compromise more than 20000 US organizations in March of 2021. There were a number of flaws in the Microsoft Exchange Server that enabled this backdoor to be installed. Patches for these flaws were released on March 2, 2021, however, only 10% of compromised organizations had implemented them by March 5.

On April 14, 2021, the FBI completed a covert cyber operation to remove the web shells from afflicted servers.

A cautionary warning from NSA was given, as seen in Figure 8.



National Security Council
@WHNSC

Patching and mitigation is not remediation if the servers have already been compromised. It is essential that any organization with a vulnerable server take immediate measures to determine if they were already targeted.

US-CERT @USCERT_gov · 06 Mar 21
CISA is aware of widespread domestic and international exploitation of Microsoft Exchange Server vulnerabilities and urges scanning Exchange Server logs with Microsoft's IOC detection tool to help determine compromise. go.usa.gov/xsPHh. #Cyber #Cybersecurity #InfoSec

19:00 · 06 Mar 21 · Twitter for iPhone

171 Retweets 25 Quote Tweets 280 Likes

Figure 8: NSA mitigation warning

4 Proof of Concept

To provide a deeper look at the state of security in modern software development, a full dependency security analysis was made on a widely used package manager tool in search of Domain Hijacking vulnerabilities.

4.1 Motivation

By analysing current trends of usage, the package manager chosen to be analysed was npm. This choice was based on the large size of adoption to this tool, as seen in Figure 9, and due to previous analyses been already made on this tool for this security issue.

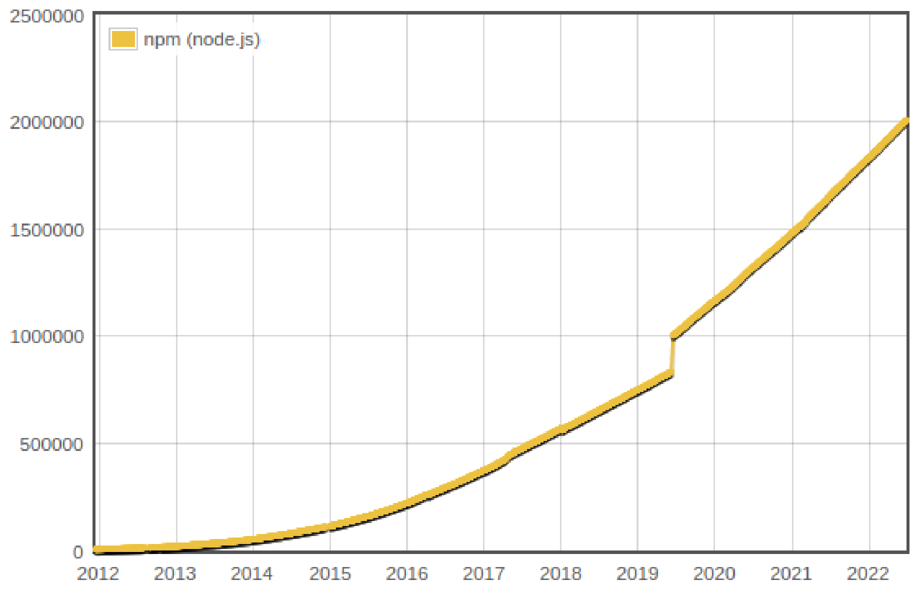


Figure 9: npm Package Manager Usage Growth[2]

Based on this analysis, a reasonable conclusion can be drawn about the current state of the attack surface which is vulnerable to Domain Hijacking.

In addition, it will allow us to see the biggest risks to software development, by identifying the most vulnerable dependencies.

Finally, the information from this study will be used to predict current trends and compare them with those from previous studies.

4.2 Implementation

To analyse the full npm packages repository for Domain Hijacking Attacks, some precautions were needed.

To verify each package for the vulnerability we require to make a lookup for the domain used on the email present on the package registry. This will be a whois request. On a single target this request isn't problematic but due to the target having 2007614 packages, as of time of writing, we could get our private ip address blocked by the server due to the multiple request made. To mitigate this issue a generic usage server was rented to execute the requests.

With the server configured, the next step was on how to obtain the registries of the packages present on the npm repository. To obtain this records npm provides a public API endpoint where the pretended records are available. The API endpoint provided is "https://skimdb.npmjs.com/registry/_all_docs".

To automate the analysis of the packages and filtration, an open source tool was used. The tool used was npmdomainchecker[7] enabling us to retrieve the records of packages that seem vulnerable to the Domain Hijack Attack, while highlighting packages that have a large number of downloads and that are vulnerable.

The resulting output of the tool was later extracted from the probing server to be later analysed on a dedicated computer. The architecture used for the Proof of Concept can be seen on Figure 10.

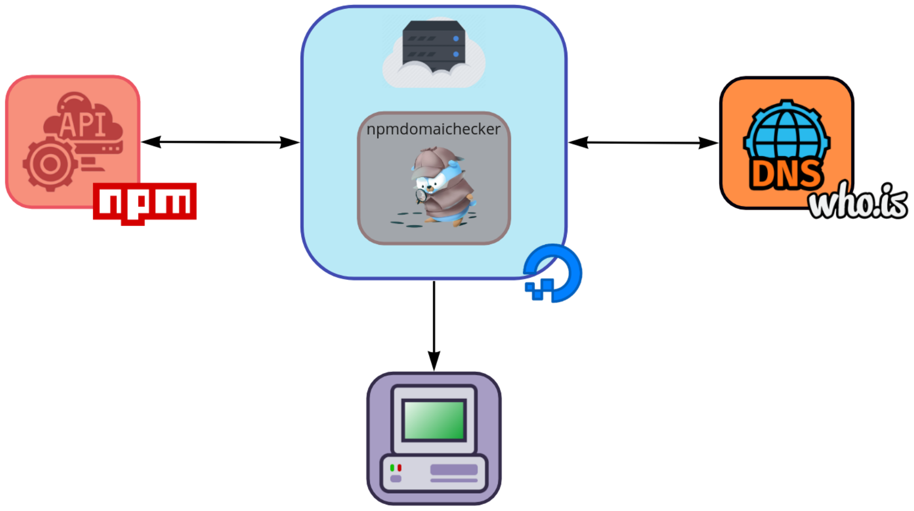


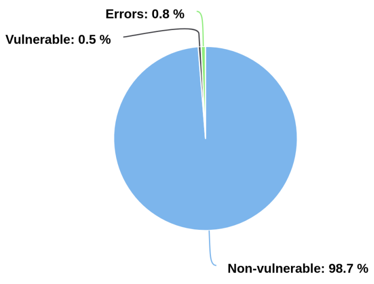
Figure 10: POC Architecture

4.3 Discussion and Implications

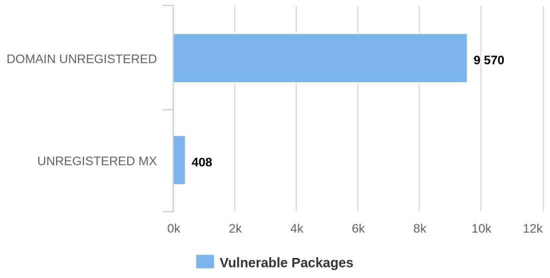
After analysing all of the npm repository packages, some errors were found. Due to malformed registries and non standard naming schemes present on the npm registry, it was found that from the 2007614 packages analysed, 15326 gave errors. The number of packages that gave errors were 0.8% of all packages analysed.

From the successful requests obtained, 9978 packages were found to be vulnerable to Domain Hijacking attacks. The vulnerable packages were found to have either an unregistered domain or a domain with an unregistered MX record, enabling therefore the possibility of takeover of the ownership of the package.

The types of results obtained can be seen on Figures 11.



(a) Types of results obtained



(b) Vulnerabilities found on packages

Figure 11: npm analysis obtained results

From the vulnerable packages found, 37 packages had more than 100000 downloads, the number of downloads can give a perspective of usage, in this case these packages had a high adoption. The number of downloads from each of the top 37 most downloaded packages found can be seen on Figure 12, the package's names were redacted and instead numeric IDs were given.

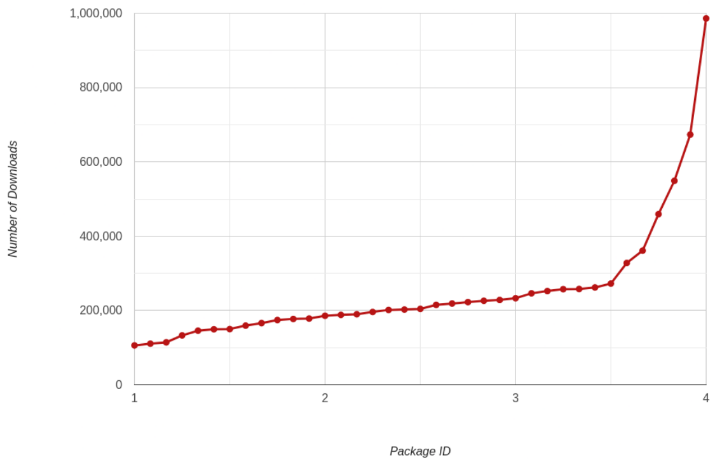


Figure 12: Top 37 most downloaded vulnerable packages

By taking a deeper look at the most popular, potentially vulnerable, package, we can predict the worst-case scenario associated with it becoming compromised.

The name of the package will be redacted, due to security concerns. The official npm package page shows us that the most downloaded from the packages analyzed are currently being used and don't show any signs of drops in adoption, as seen in Figure 13.

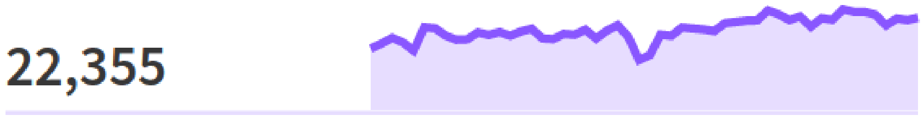


Figure 13: Most downloaded vulnerable package weekly downloads

It is also shown that this package has other dependent packages, which would enable in case of compromise, the compromise of those dependent packages.

To compromise a package, an attacker would require that the official owner of the target repository doesn't have Two Factor Authentication active, or 2FA for short. Adding this additional layer of authentication prevents a third party from taking over an account, even if the email address is compromised.

As of the time of writing the npm team, only require a maintainer of the top 100 projects to enable 2FA[8]. This requirement mitigates only a few of the total number of projects that can be hijacked. According to the findings, the project found to be potentially vulnerable isn't in the top 100, so it has the potential to be hijacked. In case of compromised, the dependant projects would also become compromised, starting therefore a supply chain attack that can better observed on figure 14.

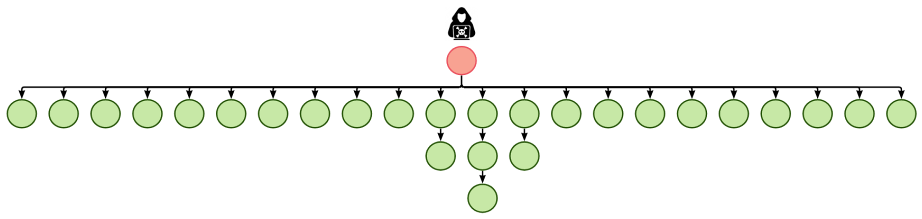


Figure 14: Most downloaded vulnerable package compromise scenario

If this package were compromised, it would compromise multiple packages as well as projects that adopted that package as one of the components in the software.

The analysis made to the npm repository can be compared to a previous analysis made by the team at jfrog[9]. By comparing the results we can conclude that current mitigations from the npm team had an effect on mitigating this security issue.

The previous analysis obtained a vulnerable package with 31 million downloads, whereas the analysis made currently only obtained a vulnerable package with 1 million.

Although the most popular vulnerable package is less adopted the growth of the number of possible vulnerable packages increased 310%, from the prior 3210 packages to 9978. We can attribute this growth to the growing popularity of the npm tool, but it does show that this security problem remains unresolved.

5 Conclusion

With the analysis of the npm repository for vulnerable packages to Domain Hijacking, a better understanding of the current state in the world of trust-based software programming.

The analysis done was only superficial in nature, due to the size of the scope of the problem. This limitation excluded attacks on lower-level components of the supply chain, those components being compilers, interpreters, and hardware-level components.

The compromise of the excluded components would have similar consequences to the ones described for dependency compromises, although harder to test on a large scale, this is an attack surface that may compromise industries.

The compromise of supply chain components can lead to industry disruption and lives at risk. In a similar way to the NotPetya attack, secure software is only as secure as its weakest element. This is because if the software is applied to public services or essential services the daily lives of citizens can be at risk.

To address these concerns security measures are being applied, however, a better understanding of the ramifications of the problem still needs to be researched further.

References

[1] Contributors to Wikimedia projects, “Threat actor - Wikipedia,” May 2022, [Online; accessed 27. Jun. 2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Threat_actor&oldid=1086693746

[2] “Modulecounts,” Jun. 2022, [Online; accessed 26. Jun. 2022]. [Online]. Available: http://www.modulecounts.com

[3] Contributors to Wikimedia projects, “Software supply chain - Wikipedia,” Jun. 2022, [Online; accessed 27. Jun. 2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Software_supply_chain&oldid=1094450681

[4] ——, “Package manager - Wikipedia,” Jun. 2022, [Online; accessed 27. Jun. 2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Package_manager&oldid=1094360477

[5] ——, “Dependency (project management) - Wikipedia,” May 2022, [Online; accessed 27. Jun. 2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Dependency_(project_management)&oldid=1089006059

[6] “NotPetya: From Russian Intelligence, With Love,” Jun. 2022, [Online; accessed 28. Jun. 2022]. [Online]. Available: https://www.bankinfosecurity.com/notpetya-from-russian-intelligence-love-a-10589

[7] firefart, “npmdomainchecker,” Jun. 2022, [Online; accessed 28. Jun. 2022]. [Online]. Available: https://github.com/firefart/npmdomainchecker

[8] “npm enrolls Top 100 package maintainers into mandatory 2FA,” Feb. 2022, [Online; accessed 29. Jun. 2022]. [Online]. Available: https://therecord.media/npm-enrolls-top-100-package-maintainers-into-mandatory-2fa

[9] “npm Package Hijacking via Domain Takeover,” Jun. 2022, [Online; accessed 29. Jun. 2022]. [Online]. Available: https://jfrog.com/blog/npm-package-hijacking-through-domain-takeover-how-bad-is-this-new-attack