Detecting Python Malware in the Software Supply Chain with Program Analysis

Ridwan Shariffdeen (NUS), Behnaz Hassanshahi (Oracle Labs, Australia), Martin Mirchev (NUS), Ali El Husseini (NUS), Abhik Roychoudhury (NUS)

Risks in the supply chain

Sonatype 2024: **512,847 malicious packages** logged just in the past year, a **156% increase year-over-year**

Known attacks:

- Typosquattin
- Combosquatting
- Dependency confusion
- Repo confusion
- “Slopsquatting”

sqlite → slqite

cmd-utils → utils-cmd

December 31, 2022

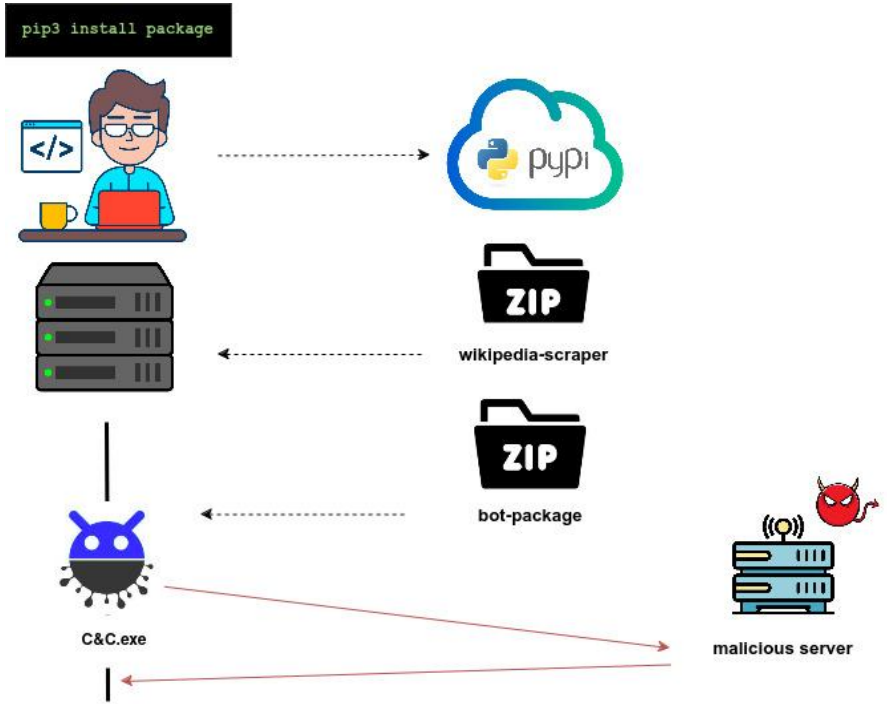
Compromised PyTorch-nightly dependency chain between December 25th and December 30th, 2022.

Kaspersky uncovers year-long PyPI supply chain attack using AI chatbot tools as lure

November 20, 2024

Divide-and-hide attack in the wild

User wants to collect data from Wikipedia.



State of the art approaches for malware detection

1. Pattern-based Static Analysis
Fast ✅
False Positives: ❌ ● Missing intent ○ subprocess.run()
False Negatives ❌ ● Obfuscations ○ eval(‘subprocess.run())

State of the art approaches for malware detection

1. Pattern-based Static Analysis	2. Dynamic Analysis
Fast ✅	Less False positives ✅
False Positives: ❌ ● Missing intent ○ subprocess.run()	False negatives: ❌ ● Complex constraint solving
False Negatives ❌ ● Obfuscations ○ eval(‘subprocess.run())	Sandboxing ❌

State of the art approaches for malware detection

1. Pattern-based Static Analysis	2. Dynamic Analysis	3. Machine Learning
Fast ✅	Less False positives ✅	Fast ✅
False Positives: ❌ ● Missing intent ○ subprocess.run()	False negatives: ❌ ● Complex constraint solving	Limitations of dataset features ❌
False Negatives ❌ ● Obfuscations ○ eval(‘subprocess.run())	Sandboxing ❌	Unbalanced training dataset ❌

State of the art approaches for malware detection

1. Pattern-based Static Analysis	2. Dynamic Analysis	3. Machine Learning
Fast ✅	Less False positives ✅	Fast ✅
False Positives: ❌ ● Missing intent ○ subprocess.run()	False negatives: ❌ ● Complex constraint solving	Limitations of dataset features ❌
False Negatives ❌ ● Obfuscations ○ eval(‘subprocess.run())	Sandboxing ❌	Unbalanced training dataset ❌
● Focus on known patterns and locations ❌ ● Single package Analysis! ❌		

Hercule - Using whole-program analysis for the supply chain

Multi-stage inter-package static analysis:

1. Integrity analysis - differential analysis between distributed package and public source;
2. Behavioral analysis - data-flow analysis on inter-package level;
3. Transitive Dependency analysis - flag using known (or unknown) malicious dependencies;

Hercule Architecture

Stage 1: Integrity Analysis

Intuitions:

- Packages that do not indicate a repository are considered suspicious;
- Differences between local and uploaded source is suspicious;
- Locate files, which do not have code clones or has differences with source;
- Comparisons should be done on AST to reduce noise;

Stage 2: Behavior Analysis

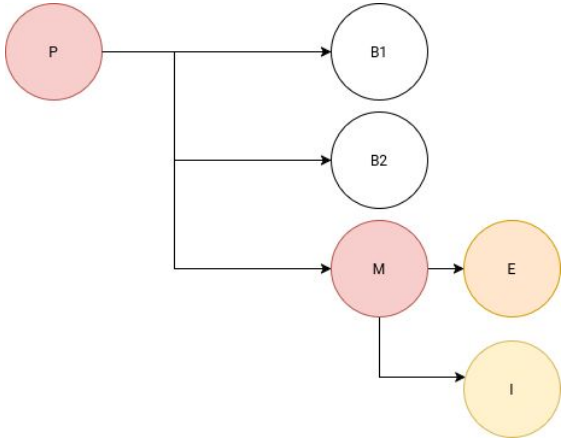
Intuitions:

- Focus on behavior of code, not syntactic patterns;
- Dataflow analysis overcomes obfuscations;
- Track only sources and sinks from suspicious locations:
  - Locations gathered from Integrity analysis
- Focus on commonly sources and sinks:
  - Obfuscated strings, APIs providing sensitive information;
  - Network requests, Process executions;

Stage 3: Dependency Analysis

Intuitions:

- Analyze all dependencies - implicit and explicit;
- Keep collection of known malicious packages;
- Tracking path of malicious behavior for provenance;
- Example package dependency tree:
  - Yellow Node - Implicit Dependency;
  - Orange Node - Explicit Dependency;
  - Red Node - Malicious package;



Comparison with tool used in Practice - Oracle's Macaron

Experiment:

- 1000 packages released on 10th October 2024

Results:

- Macaron identified 5 packages as potentially malicious.
- One true positive and flagged by Hercule.
- Analysis time - less than 2 minutes per subject.

Opportunities for combining Hercule and Macaron

Evaluation

- Malicious Datasets:
  - Maloss (242/255)
  - BackStabber (345)
  - PyPIMalRegistry (2935)
- Benign Datasets:
  - Popular packages (100)
  - Trusted packages (1200)

Baseline Tools:

- Bandit4Mal
  - Rule-based static analysis
- GuardDog
  - Rule-based static analysis
- LastPyMile
  - Integrity analysis using project history
- MalOSS*
  - Hybrid (Static + Dynamic) Analysis

Research Questions

- RQ1: How effective is HERCULE in detecting malicious packages compared to existing state-of-the-art techniques?
- RQ2: What is the contribution of each step in HERCULE in detecting malicious packages?
- RQ3: How effective is HERCULE in reducing false positives for benign packages?
- RQ4: How efficient is HERCULE across benign and malicious packages?

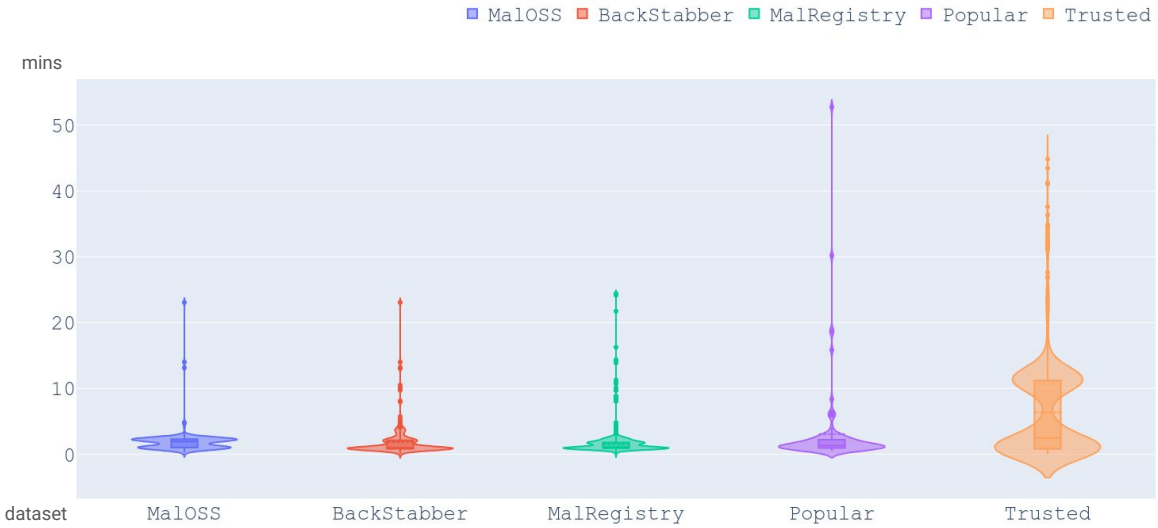
RQ1: Analyzing Packages

Technique	Precision	Recall	F1-Score	FP-Rate	Accuracy
HERCULE	0.966	**0.933**	**0.949**	0.089	**92.721**
MALOSS	0.867	0.834	0.850	0.345	78.557
GUARDDOG	**0.978**	0.687	0.807	**0.042**	76.027
BANDIT4MAL	0.701	0.716	0.708	0.826	56.968

Comparison with multiple state-of-the-art tools shows that HERCULE outperforms existing techniques with an F1-score of 0.949.

Takeaway: A good balance between precision and recall.

RQ4: Overall efficiency



HERCULE completes the analysis of malicious packages on average in 1.61 minutes and benign packages in 6.08 minutes.

Conclusion

- Combine integrity analysis and behavioural analysis to reduce False Positives;
- Inter-package analysis is required to discover sophisticated attacks;
- Implicit and explicit dependencies must be taken into account for higher degree of security;
- Behavioural analysis is more effective than historical analysis for filtering FP;

Hercule Architecture

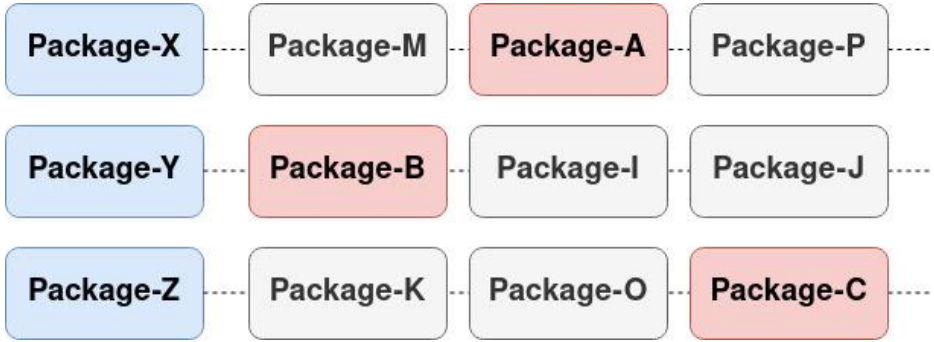
Tool Availability

Hercule Repository

Zenodo artifact

Hercule Architecture

Example attack: Divide-and-hide attack



Packages X,Y,Z and their dependencies

1. User installs Package-X;
2. User installs Package-Y;
3. User installs Package-Z;
4. Package-C uses Package-A and Package-B to execute the attack;

Requires whole environment analysis to be detected!

RQ2: Contribution of each step

Dataset	# Pkgs	Identified Repository	Integrity Analysis Violated	Behavior Analysis Violated	Dependency Analysis Violated
MalOSS	242	101	219	209	6
BackStabber	345	147	291	310	9
MalRegistry	2985	1433	2807	2678	25
Popular Packages	100	89	30	1	1
Trusted Packages	1200	1081	1023	113	0
Overall	4872	2851	4370	3311	41

All three analyses enhance HERCULE's ability to:

- Accurately label packages
- Achieve manageable false positives
- Maximize the detection of malicious packages

RQ3: Reduction of false positives for benign packages

Dataset	BANDIT4MAL	LASTPYMILE	HERCULE*
Popular Packages	19900	3562 (-82.10%)	2241 (-88.74%)
Trusted Packages	44353	24309 (-45.19%)	8553 (-80.72%)
Overall	64253	27871 (-56.62%)	10794 (-83.20%)

The lightweight AST differentiation-based integrity analysis achieves a pruning ratio of 83.20% over Bandit4Mal at a fraction of the time.

Takeaway: AST-based integrity analysis is better than hash-based analysis for filtering signals.