International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

A Comprehensive Review of Open-Source Malware Scanners in the Software Supply Chain

Karthikeyan Thirumalaisamy

Independent Researcher, Washington, USA
Email: *kathiru11[at]gmail.com*

Abstract: ***The increasing adoption of open-source software (OSS) makes the software supply chain an attractive target for advanced cyberattacks. The risks of malicious packages or dependencies (also known as dependency confusion) and trojanized components (aka supply chain interception) have been identified as growing risks and we have seen attacks publicly, exposing the risks associated with supply chain security. There are many available security tools, but a systematic overview of open source malware scanners for protecting the software supply chain is missing. This paper classifies and evaluates open-source malware scanners based on their detection paradigm, whether or not they integrate into the software development lifecycle (SDLC), and whether or not they protect the user from supply chain risks. It categorizes the scanners into three types: static analysis tools (e.g. heuristics-based and Semgrep rules), signature-based scanners (i.e. YARA, ClamAV), and behavioral analysis tools to evaluate code in runtime. It also provides an overview and comparison of selected tools based on certain criteria: detection effectiveness, CI/CD integration, and support provided by the OSS community. This paper helps to identify the trade-offs, such as the velocity of discovering potential issues through static analysis, versus the depth of discovery through dynamic analysis, and limitations in the current landscape, such as challenges with obfuscated malware and false positives and offers practitioners and researchers practical information for securing a modern software ecosystem.***

Keywords: Supply chain, Malware Scanning, Behavioral analysis, Threat detection, Static analysis, Dynamic analysis

1. Introduction

Open-source software (OSS) is a vital component of many applications today, spanning applications from enterprise software to consumer-facing software services. There are many attributes that make OSS a place for collaboration and innovation, i.e., teamwork, free cost, and iterative/rapid development, that have made OSS immensely valuable in the global software ecosystem. However, as the OSS movement has exploded to provide quick development solutions there have also been more opportunities for malicious/evil actors to exploit the software supply chain for their illegal activities. Threats are emerging in supply chain security including malicious packages, dependency confusion, and trojanized packages, and clearly documented high-profile incidents clearly light the path to compromises that could be catastrophic. These types of supplsy chain attacks occur due to the trust placed in dependencies, including OSS libraries and their various distribution channels. As a result, malicious code can be inserted into familiar libraries, and if not properly monitored, such code can spread widely throughout software systems and the broader technology ecosystem.

As DevSecOps/DevOps/Security professionals have recognized the vulnerabilities in the software supply chain there have been several security tools created to identify threats and ultimately patch or block threats. One important type of tool is open-source malware scanners, which can be flexible and incorporate standardization and ease-of-use in multiple workflow configurations. Despite the significance of OS malware scanner tools, there is currently no formal, systematic assessment of these tools in the context of providing scanning to protect the software supply chain.

This paper seeks to fill that gap by categorizing and assessing OS malware scanners based on detection methodology, software development life cycle (SDLC) integration and the ability to mitigate supply chain risks. The assessment classifies scanners into three primary areas; static analysis tools (i.e., heuristics-based tools, semgrep rules, etc.), signature-based scanners (i.e., YARA, ClamAV, etc.), and behavioral analysis tools (meaning the evaluation of code is completed when it's executed). Further, selected tools (Socket and GuardDog) are compared in the areas of detection, compatibility with CI/CD pipelines, and community aggregation. This assessment paper will allow the community (DevSecOps, Developers, and Research) to assess the current approaches to strengthen malware detection in the OSS and illuminate the challenges that we still face (i.e., obfuscated malware, false positives, etc.) and provide areas of continued expansion in relation to automated and resilient architectures.

2. Static Analysis Scanners

Static analysis tools recognize malicious or suspicious code patterns without running the code. The tools analyze the source code, bytecode or package metadata to find possible threats according to rules, heuristics or pattern-matching techniques. Static analysis is a great fit for early detection during the software development lifecycle (SDLC) because it can be embedded directly into source repositories and CI/CD pipelines to flag potential issues before software is built or deployed.

2.1. Static Scanner Approach

Static scanners work by parsing and analyzing the code files and detecting risky constructs for example:

- Unsafe functions or APIs (for example: eval() in JavaScript or os.system() in Python).
- Suspicious imports or executables embedded within the code.
- Obfuscated or base64 encoded payloads.
- Hard coded credentials or external network callback to an untrusted domain.

These scanners typically rely on heuristic rules, abstract syntax tree (AST) analysis, and pattern-based scanning to represent these indicators. Static analysis scanners provide many benefits for securing the software supply chain. They provide fast scanning since no code needs to actually execute, allowing code scanning to take place quickly within the development process. This affords developers a "shift-left" security approach where issues are discovered as early as possible to minimize cost and effort of remediation. Static analysis tools lend themselves well to performing automated security scans in version control hooks and during CI/CD workflows, as they work quickly and are easy to integrate.

However, static analysis tools have some clear drawbacks as well. They lack execution (or runtime) context, meaning that they cannot flag behaviors that only occur when code is running. They can be evaded by obfuscation techniques (code-running techniques), dynamic imports, or payloads that are encrypted, all ways that a malicious actor can perform their action without easily being discovered. Static analysis tools also can produce a lot of false positives (potential security vulnerabilities), especially if rules are too broad or code too restrictive. These flags can confuse developers and may lead to alert fatigue for development and security teams.

2.2. Supply Chain Security Focused Scanners

Supply chain security–focused scanners differ from general-purpose static analysis tools because they specialize in detecting risks that occur within the open-source software (OSS) ecosystem. These scanners focus on detecting malicious dependencies and trojanized packages and dependency confusion attacks which Static Application Security Testing (SAST) platforms normally do not detect. Supply chain scanners protect against software supply chain risks by analyzing package metadata and maintainer behavior and repository practices and suspicious code patterns.

2.2.1. Semgrep

Semgrep is an open-source, pattern-based static analysis tool, and does not execute code. It is faster and easier to maintain across programming ecosystems. Semgrep parses source code into abstract syntax trees (ASTs) and checks the parsed code against rules either defined by the user or made available by the community, which are written in the YAML format. In this manner, Semgrep has the ability to identify risky coding patterns and behaviors that could highlight supply chain dangers such as malicious imports, dependency confusion, obfuscated payloads, and misuse of sensitive APIs in third-party packages. Because Semgrep is easy to use, it can be added to CI/CD pipelines and version control systems, supporting shift-left security by allowing a user to detect threats in their source code during development. When Semgrep was initially developed, its emphasis was on secure coding, but the Semgrep project more recently extended its offerings for supply chain security use cases and its community rulesets (such as those provided from GuardDog and OpenSSF) developed rules targeting Trojanized packages and malicious components in OSS. Semgrep is versatile due to its extensive multi-language support, which allows it to work across a vast majority of software stacks and dependency ecosystems.

Supported Languages: Python, JavaScript, TypeScript, Go, Java, C#, Ruby, PHP, C, C++, Rust, Kotlin, Scala, Swift, Terraform, Dockerfile, YAML, JSON

2.2.2. CodeQL (GitHub)

CodeQL is a query-based static analysis tool that was designed by GitHub, which treats code as data in a relational database. This allows security teams to build custom queries to find those vulnerabilities and risky patterns that they are concerned about as needed throughout the SDLC. CodeQL is different than traditional rule-based SAST tools because it provides declarative queries similar to SQL, which can capture important items like insecure API usage, improper input validation, or risky dependency behavior. From a supply chain security point of view, CodeQL can identify unwanted network calls to known bad sites in third-party repositories, suspicious local system calls, obfuscation of code, hidden scripts in open-source packages, etc. CodeQL has first class integration with GitHub Actions to automate the scanning of repositories and any pull request and contains a large collection of public queries that can be re-used, re-purposed, or extended based on the needs of individual organizations. CodeQL also works across many programming language and ecosystems, making it an ideal candidate for analyzing many types of projects and OSS dependencies.

Supported Languages: C, C++, C#, Java, JavaScript, TypeScript, Python, Go, Ruby

2.2.3. Socket

Socket is a supply chain security-focused analysis platform that finds malicious or risky behavior in open-source packages across ecosystems. Traditionally, security tools (SAST) analyze only the source code or rely on manual analysis. Sockets go deeper by doing deep package inspection through assessing, and evaluating metadata, scripts, code patterns to find threats, such as trojanized packages, dependency confusion, and obfuscated payloads. Socket uses static and heuristic analysis while performing checks to find suspicious install scripts, hidden calls on the internet, odd behavior of maintainers, and more, all risk signals as part of an overall analysis to identify supply chain attacks.

Socket integrates into your dev workflow, including GitHub, and CI/CD pipelines, and will alert you in real-time when you are introducing risky dependencies. Socket works across multiple ecosystems including npm, PyPI, Go, Ruby, and Java and is effective in today's multi-language projects. Socket blocks package-level threats, rather than relying on source code analysis alone, providing organizations a way to stop malicious OSS components from riding into their software supply chains. Socket also works alongside traditional static analysis tools and is a vital second layer of supply chain defense.

Supported Languages: JavaScript, TypeScript, Python, Go, Ruby, Java

2.2.4. GuardDog

GuardDog is a static analysis tool focused on supply chain security, and is particularly good at identifying malicious or suspicious behavior in open-source packages, with the focus on the Python ecosystem. It performs rule-based detection (much of it based on Semgrep patterns) and scans through package metadata and source code for red flags involving dependency confusion, obfuscated code, hidden install scripts, or unknown system calls. GuardDog aims to identify trojanized packages and other supply chain attacks before they enter the project's dependency list.

The tool is set up to easily be incorporated into development workflows such as CI/CD pipelines and pre-commits so that scanning can occur automatically and early on during development. This is an important consideration for AI agentic development in Python since these AI systems may automatically fetch, import, or execute open-source packages for them. Supply chain attacks in this context could result in unknown sources of code execution through unintentional code execution, unwanted data send (exfiltration), or unwanted model manipulation. Therefore, scanning the supply chain is a critical layer of defense. While GuardDog is primarily aimed at Python packages, it can also facilitate JavaScript scanning in limited contexts. GuardDog's predominant role is to give developers information to take action on targeted alerts to prevent malicious components into software projects, which works best alongside a broader static analysis and supply chain security tools.

Supported Languages: Python**,** JavaScript

2.2.5. Bandit

Bandit is an open-source static analysis tool that finds security problems in Python code. It works by parsing Python source files and looking up the abstract syntax tree (AST) for patterns of common security weaknesses like calls to eval(), weak cryptography, insecure file operations, hard-coded credentials, or unsafe subprocess vulnerabilities. Bandit ships with several built-in, pre-configured security rules, and there is the ability to add additional rules in order to customize the scans according to the needs of the specific project.

While Bandit is more general Python security, it is especially applicable to supply chain security for Python projects. Many of the more recent Python applications, including AI agentic systems, make use of many third-party packages. Bandit can be used to scan project dependencies and find opportunities to misuse those dependencies – either adapted between changes in libraries or version, all of which could be exploited by a malicious package, or a trojanized package as common in supply chain attacks. Integrating Bandit with CI/CD pipelines or pre-commit hooks allows developers to apply early security checks and define their own risk thresholds when pulling in untrusted functionality.

Supported Languages: Python

2.3. Static Scanner Comparison

The table provides a comparative overview of five prominent static analysis and supply chain security tools Semgrep, CodeQL, Socket, GuardDog, and Bandit. It focuses on each tool's primary type and focus, detection methodology, supply chain security use cases, key strengths, and limitations.

Tool	Type / Focus	Detection Methodology	Supply Chain Security Use	Key Strengths	Limitations
Semgrep	Multi-language SAST & pattern-based	AST and pattern-matching rules	Detects suspicious code constructs, obfuscation, and risky imports in dependencies	Highly customizable, fast, multi-language support	Requires well-defined rules, may miss runtime-only behavior
CodeQL	Query-based SAST / security analysis	Code as data in relational DB and custom queries	Detects risky code patterns, and malicious dependency behavior	Powerful query-based analysis, extensive query library and multi-language	Complex setup, steep learning curve
Socket	Supply chain security scanner	Deep package inspection and heuristic analysis	Detects trojanized packages, hidden scripts dependency confusion	Multi-language, package-focused, detects hidden malicious behaviors	Focused on package-level threats, less code-level detail
GuardDog	Python / JS supply chain scanner	Rule-based detection (Semgrep patterns)	Detects trojanized packages, dependency confusion, obfuscated code	Specialized for Python supply chain, lightweight, early detection	Primarily Python-focused, may need custom rules for other ecosystems
Bandit	Python static analysis	AST-based security rules	Detects insecure coding practices in dependencies and potential supply chain risks	Lightweight, easy to integrate, built-in rules for common Python vulnerabilities	Python-only, less effective for complex supply chain attacks

3. Signature-Based Scanners

Signature-based scanners identify malicious or suspicious code by comparing files, packages or binaries to a database of known malware signatures or patterns. While static analysis tools would detect a general risky construct (such as a use of the eval function) signature-based scanners flag maliciousness based on previously identified threats. These scanners can operate on either source code, compiled binaries or package metadata and are very useful in quickly identifying known trojanized packages or previously seen malware in the software supply chain. Signature-based scanning can run in CI/CD pipelines, artifact repositories or container registries, providing automated checks before deployment. Signature-based scanners have the advantage of high confidence when detecting known threats but may not detect malware that is a novel construct or purposely obfuscated malware. And when used with static analysis or

behavioral analysis these signature-based scanners are most effective.

3.1. Signature Scanner Approach

Signature-based scanners rely on matching code, packages, or binaries against a set of known malware signatures - representing patterns, hashes, or signature characteristics that signify previously seen malware. It looks for threats by finding an exact or near-exact match with previously identified malicious code, which can be:

- Files/messages/packages containing known trojanized scripts or malware payloads.
- Executables or scripts associated with previously identified versions of malicious binaries.
- Metadata/hash values of previously seen compromised dependencies.
- Patterns of code/scripts previously associated with dependency confusion or supply chain attacks.

Signature-based scanners can be very reliable in detecting known threats and allow fast scanning without executing code, making it ideal for workflows such as CI/CD pipelines, artifact repositories, and version control hooks because you can allow companies to automatically block or even flag malicious packages before deploying them.

Signature-based scanners have limitations too. Since they only identify known malware and don't execute code, they are susceptible to new malware and/or obfuscated malware that hasn't been added to an organization's signature database yet. Attackers themselves can evade detection if they change the code enough to avoid a full signature match, but not so much that they break functionality, for example encrypting or obfuscating the payload, changing the methods, or using dynamic imports. Additionally, if the signature is too broad it may get too many false positives, marking benign packages as malicious which causes development slowdowns at best or over time an alert fatigue in a security function.

3.2. Supply Chain Security Focused Scanners

Signature-based scanners focused on supply chain security are targeted types of tools, and they have the sole purpose of detecting known malicious packages and previously discovered malware in the software supply chain. While traditional signature scanners identify the source code or binaries for any application, supply chain focused scanners are specifically for open-source dependencies/package ecosystems, and are therefore useful in blocking trojanized packages, dependency confusion attacks, and propagating other, supply chain-related threats into an application.

3.2.1. ClamAV

ClamAV is one of the most popular open-source antivirus engines. ClamAV is able to detect known malware, such as trojans and malicious packages, by matching file content or package binaries against a popular signature database. ClamAV can also scan many types of packages and compressed formats, making it useful when examining OSS dependencies and container artifacts prior to deployment. The signature definitions are updated often, which allows for fast detection of something that has already been seen. If the malware is new or obfuscated however, ClamAV will not be able to capture it unless a relevant signature is available.

ClamAV executes the following sequence of operations when it scans files and packages:

- The system decompresses archives containing .zip, .tar, .jar and .deb files to enable full analysis of all contained files.
- The system uses signature database comparison to check each file against its stored entries. The system detects trojanized setup.py scripts in Python packages through its signature database which contains known malicious code fragments.
- The system uses heuristic checks to identify known malware variants through basic rules that detect encoded payloads and exploit markers.
- The system produces an infection report when it detects a match which gets sent to CI/CD pipelines and security dashboards for processing

Supported Languages: ClamAV is not language-specific like Semgrep or CodeQL. Instead, it scans files, binaries, and archives regardless of the programming language.

3.2.2. YARA

YARA is an open-source software used by malware researchers and security teams to find and classify malware, based on patterns defined in user-created rules. YARA is fundamentally distinct from ClamAV in that YARA does not rely solely on a centralized database of pre-defined signatures. The rule language of YARA enables users to create signatures which describe malware characteristics for effective detection of malware variants and suspicious supply chain components.

A standard YARA rule contains functionality to detect:

- The tool searches for malicious function names and suspicious API calls and obfuscated payloads through string pattern matching.
- The tool searches for specific byte sequences that exist within binary files and package contents.
- The tool examines file metadata through its analysis of size and structure and import table information.
- The system uses logical operators to check for multiple suspicious strings and file sizes exceeding 50KB.

In the context of the software supply chain, YARA can be used for packages, dependencies, and build artifacts to identify trojanized OSS components or known malware families. Take for instance a YARA rule that could identify a Python dependency, that has embedded reverse shell code or a JavaScript package that attempts to exfiltrate environment variables. Security teams can also create YARA rules for dependency confusion attacks to find the characteristics common in malicious impostor packages.

Supported Languages: Yara is not language-specific like Semgrep or CodeQL. Instead, it scans files, binaries, and archives regardless of the programming language.

3.3. Signature based Scanners Comparison

The table provides a comparative overview of two prominent signature-based analysis and supply chain security tools ClamAV, and YARA. It focuses on each tool's primary type and focus, detection methodology, supply chain security use cases, key strengths, and limitations.

Tool	Type / Focus	Detection Methodology	Supply Chain Security Use	Key Strengths	Limitations
ClamAV	Open-source antivirus engine, signature-based detection of known malware	Centralized malware signature database matching	Scans OSS dependencies, build artifacts, and containers for known trojanized or malicious packages	Fast and automated scanning, regularly updated DB, easy CI/CD integration	Limited to known signatures, misses zero-days, evasion via obfuscation
YARA	Rule-based malware research and detection engine	Custom rules (string, byte, regex, or logic-based patterns)	Detects trojanized packages, suspicious code, or binaries through tailored OSS-focused rules	Highly customizable, effective for malware variants, ecosystem-agnostic	Requires expertise to write/maintain rules, no central DB and can be noisy

4. Behavioral analysis tools

Behavioral analysis tools mainly discover malicious activity by examining how code behaves syntactically as it runs. Traditional methods of looking at code for structure, known signatures, etc. will miss some dependencies that may look okay, yet display malicious activity at runtime. Most behavioral analysis tools perform their analysis in a sandboxed space and measure characteristics towards the entire run time behavior: system calls; file reads/writes; network calls; memory usage; and privilege escalation. Tools from a behavioral analysis model can be a great inclusion to security considerations in the software supply chain particularly around discovering trojanized packages or dependencies: a bad Python package may provide an accurately inspect in a static context, the actual threats are in the execution context of the code execution (e.g., downloads remote payload; exfiltrates secrets; dynamic load from unknown or hidden - included is the ability of threat actors to perform data-injection).

4.1. Behavioral Scanner Approach

Behavioral scanners identify malicious code through its operational behavior during execution instead of using pre-defined signature detection. Behavioral scanners enhance security through defense-in-depth because they operate independently of signature-based detection methods. The scanners operate by monitoring program execution within controlled environments such as sandboxes and containers and virtual machines. The system monitors for any abnormal or unwanted actions that occur during program execution. The system monitors following main types of suspicious behavior:

- The system detects unusual network activities which include domain name connections to untrusted servers and command-and-control server interactions.
- The system tracks file system activities which include file hiding and configuration file alteration and executable file modification.
- The system detects when a program attempts to elevate its privileges through system calls which bypass standard security restrictions.
- Behavioral scanners detect hidden payloads through dynamic code loading which becomes visible only when the program starts running.

Behavioral scanners provide exclusive protection for the software supply chain because they detect malicious dependencies and trojanized packages which static and signature scanning methods cannot identify during execution. The Python dependency obfuscation technique reveals its hidden payload only after the installation script runs. Behavioral scanners track runtime sessions through clock monitoring which enables them to detect zero-day malware and polymorphic code and advanced obfuscation methods that evade static analysis and signature detection systems.

Behavioral scans appear most effective for detecting malicious activities because they face specific implementation hurdles. The execution of these scans needs to occur within a monitored environment. The execution of code within an environment-controlled system leads to performance degradation during runtime. The scanning process through behavioral methods operates at a slower pace than static scanning methods which remain essential for modern threat protection. The operation of behavioral scanners depends on sandboxing or monitoring systems which need whitelisting and kernel-level code implementation to add complexity to the system. Some advanced malware programs contain detection capabilities which identify sandbox environments, so they delay their malicious activities until the behavioral scanner stops monitoring their operations. Behavioral scanners serve as an essential defense-in-depth tool because they detect runtime threats which static scanning and other scanning methods fail to identify.

4.2. Supply Chain Security Focused Scanners

Behavioral scanners that are focused on supply chain security can identify malicious activity that is only observable at runtime. Unlike static or signature-based methods, behavioral scanners run code in secure environments (like sandboxes, containers, or kernel monitor mode) and will look for suspicious behaviors (e.g. making unexpected network connections, downloading hidden payloads, tampering with the file system, escalating privileges). Behavioral scanners will be an effective approach to detect obfuscated or zero-day malware that is the result of malicious activity in open-source dependencies, where the intent is hidden until runtime.

4.2.1. Falco

Falco exists as an open-source runtime security tool which

Sysdig created before becoming a CNCF project. The tool tracks system calls (syscalls) on Linux hosts and containers through eBPF (extended Berkeley Packet Filter) or kernel modules. Falco uses real-time system event monitoring to identify security threats which traditional static and signature-based scanners cannot detect.

Falco provides essential supply chain security because it tracks actual system activities that occur when code and dependencies execute inside containers and Kubernetes workloads. The tool detects security threats when a malicious package attempts to perform any of the following actions:

- The system detects when a package tries to establish network connections with unverified external servers.
- The system detects when a container attempts to modify sensitive files located in unauthorized directories.
- The system detects when a build or runtime process creates a new shell process.
- The system detects when code attempts to load dynamically or when privilege escalation occurs.

The system detects suspicious behavior which can trigger alerts and block specific actions based on its integration configuration. The tool includes multiple predefined rules that identify abnormal system activities including "containers should not write to /etc/passwd" and organizations can develop custom rules to monitor specific supply chain threats like package installation scripts that connect to external servers or create unusual files.

The tool provides continuous runtime environment monitoring through its integration with Kubernetes and container runtimes and CI/CD pipelines which enables protection of OSS dependencies in deployment environments. The tool functions as a strong supply chain defense system because it implements behavioral detection to identify hidden zero-day malware and obfuscated threats within dependencies.

4.2.2. Tracee

Tracee is an open-source runtime security and forensics tool developed by Aqua Security. It uses eBPF (extended Berkeley Packet Filter) technology to trace and monitor events within the Linux kernel, allowing it to view the entire execution path without requiring intrusive kernel modules. Tracee focuses on runtime monitoring to discover anomalies and threats during code execution, as opposed to relying on static or signature-based scanners which are intrinsically limited to a set of known "bad" behaviors.

Specifically for supply chain security, Tracee will assist organizations in discovering malicious behavior hidden in open-source dependencies or trojanized packages, probably in a subset of behavior that only appears during execution. In particular, Tracee can identify when a dependency:

- Makes unusual network calls to unknown domains
- Attempts to spawn shell commands during installation or run,
- Loads dynamic code or libraries not specified in a manifest
- Attempts to read or exfiltrate sensitive files from the host/container
- Attempts privilege escalation via suspicious syscalls

Tracee ships with a catalogue of built-in detection rules and signatures for various known malicious behaviors. It also supports custom rule writing through Rego policies (via Open Policy Agent), making it very flexible and adaptable to monitor a particular supply chain risk. Tracee is well-suited to containerized environments, typically Kubernetes, where many supply chain attacks originate from compromised images or dependencies. If organizations were to run Tracee during either builds or deployments, they would obtain unprecedented runtime monitoring into the patterns of third-party packages and be able to limit either the attack or flag the behavior before production use. In summary, Tracee provides a very lightweight and powerful behavioral analysis layer to complement static and signature-based scanners in their ever persistent fight against obfuscated malware, zero-day exploits, and runtime supply chain attacks.

4.2.3. Cuckoo Sandbox

Cuckoo Sandbox is a popular open-source automated malware analysis system that works by executing suspicious files, executables, or packages inside an isolated sandbox (or virtual machine) and observing their behavior. It is distinct from static or signature-based tools because Cuckoo observes what actually happens when the code executes, making it a powerful system for detecting obfuscated or previously unknown threats.

Cuckoo Sandbox functions as a tool for software supply chain security through its ability to analyze untrusted open source dependencies and packages and build artifacts before their use in production. Cuckoo Sandbox monitors Python and JavaScript and containerized packages through execution which results in the registration and recording of:

- The system tracks all file system operations that include new file generation and modifications to critical file locations and hidden payload delivery.
- The system tracks network activities which include suspicious domain connections and command-and-control callbacks and data exfiltration attempts.
- The system tracks all process activities which include shell entry points and hidden script executions and privilege elevation attempts.
- The system tracks API and system calls that show signs of security bypass attempts and malicious code loading operations.

Cuckoo operates as a flexible solution which supports multiple guest operating systems including Windows and Linux and macOS and Android. The behavioral indicators and logs and memory dumps and network captures from Cuckoo reports enable users to conduct additional analysis for confirming malicious operations. Cuckoo Sandbox delivers its best performance against zero-day exploitation and polymorphic malware and highly obfuscated packages which evade static and signature-based scanning systems. The sandbox environment of Cuckoo requires dedicated infrastructure for its automated system operation because it generates significant overhead but provides better results than Falco or Tracee workstations do. The supply chain security field benefits from Cuckoo Sandbox as a strong behavioral analysis solution. The tool serves best for pre-deployment package analysis in this context and generates detailed

runtime evidence for digital forensic investigations and incident response operations.

4.3. Behavioral based Scanners Comparison

The table provides a comparative overview of three prominent behavioral analysis and supply chain security tools Falco, Tracee and Cuckoo Sandbox. It focuses on each tool's primary type and focus, detection methodology, supply chain security use cases, key strengths, and limitations.

Tool	Type / Focus	Detection Methodology	Supply Chain Security Use	Key Strengths	Limitations
Falco	Runtime security and behavioral monitoring tool	Monitors system calls and kernel events using eBPF/ptrace	Detects suspicious runtime behaviors in containers, Kubernetes workloads, and build environments	Real-time monitoring, predefined and customizable rules and lightweight for containers	Limited to observable syscalls, requires kernel/eBPF support, may miss very subtle attacks
Tracee	Runtime security and forensic analysis tool	Uses eBPF to trace Linux kernel events	Detects anomalies from untrusted OSS dependencies or build artifacts in containerized environments	Fine-grained kernel event tracing, custom rules via Rego/Open Policy Agent and low overhead	Linux only requires rule expertise, limited Windows/macOS support
Cuckoo Sandbox	Automated malware analysis sandbox	Execute files/packages in an isolated sandbox and monitors behavior	Analyzes OSS packages or build artifacts for malicious activity before integration	Multi-OS support (Windows, Linux, macOS, Android), detailed behavioral reports, detects obfuscated/zero-day malware	High resource overhead, slower than lightweight runtime monitors, requires sandbox infrastructure

5. Conclusion

Due to the broad adoption of open source software and increasingly sophisticated cyberattacks on software supply chains that target dependencies, building artifacts, and containerized (runtime) software environments, the security of the software supply chain is of paramount importance. This paper provided a comprehensive overview of open-source malware scanners, grouping them into static analysis tools, signature-based scanners, and behavioral analysis scanners, analyzing them for their utility in finding supply chain threats. Each category of tool has its strengths: static analysis tools based on the source code are faster and can be integrated early on in any CI/CD pipeline; signature-based scanners are effective at identifying well-known malware or compromised packages; behavioral scanners are important as they show what malicious or problematic behaviors can occur only during run-time (including obfuscated or zero-day behaviors) upon execution.

For example, in discussing tools such as Semgrep, CodeQL, Bandit, ClamAV, YARA, Falco, Tracee, and Cuckoo Sandbox, our paper concluded that combining code-level inspection and run-time monitoring will enhance supply chain security profiling and remediation. Behavioral scanners like Falco and Tracee are especially valuable in the context of containerized environments; Cuckoo Sandbox provides an in-depth forensic-level look at a package behavior and running before deploying. We also detected several important trade-offs: static and signature-based tools are the fastest, but do not catch unknown or obfuscated threats; behavioral tools can provide better visibility of malicious behaviors, but they will be detrimental to performance and resource utilization.

In conclusion, no single tool can completely secure the software supply chain, and instead, a layered defense-in-depth strategy that includes multiple static, signature-based, and behavioral scanning during the software development lifecycle is best. A common thread we have seen is that incorporating these tools into automated pipeline workflows, along with version control, artifact repositories, and run-time environments, will reduce the risk of including malicious artifacts into production and public. Further research should address automation as a priority focus, cross-language and interoperability, AI based threat detection, and accurately identifying polymorphic or obfuscated malware. Ultimately, these advances will increase the security posture of the software supply chain and contribute to a more resilient and trustworthy software ecosystem.

References

[1] Rakesh Singh Kunwar, Priyanka Sharma, Malware Analysis: Tools and Techniques, In Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP), pp. 1–8, 2016. https://doi.org/10.1145/2905055.2905361

[2] John Oluwafemi Ogun, Advancements in automated malware analysis: evaluating the efficacy of open-source tools in detecting and mitigating emerging malware threats to US businesses, International Journal of Science and Research Archive, 2024, 12(02), 1958–1964. https://doi.org/10.30574/ijsra.2024.12.2.1488

[3] Robiah Y, SitiRahayu S., MohdZaki M, Shahrin S., Faizal M. A., Marliza R., A New Generic Taxonomy onHybrid Malware Detection Technique, (IJCSIS) International Journal of Computer Science and Information Security, Vol. 5, No. 1, 2009. https://arxiv.org/abs/0909.4860

[4] Uppal, Dolly, Vishakha Mehra, and Vinod Verma. "Basic survey on malware analysis, tools and techniques." International Journal on Computational Science and Application (IJCSA) 4.1 (2014): 103-112. https://wireilla.com/papers/ijcsa/V4N1/4114ijcsa10.pdf

[5] Falco, Detect security threats in real time. [Online]. Available: https://falco.org/

[6] Dave Wichers, itamarlavender, will-obrien, Eitan Worcel, Prabhu Subramanian, kingthorin, coadaflorin, hblankenship, GovorovViva64, pfhorman,

GouveaHeitor, Clint Gibler, DSotnikov, Ajin Abraham, Noam Rathaus, Mike Jang, Source Code Analysis Tools. [Online]. Available: https://owasp.org/www-community/Source_Code_Analysis_Tools

[7] Alastair Wilkes, Static Code Analysis Approaches for Handling Code Quality, 2023. [Online]. Available: https://www.launchableinc.com/blog/static-code-analysis-approaches-for-handling-code-quality/

[8] Stuart Foster, What Is Static Analysis? Static Analysis Tools + Static Code Analyzers Overview, 2023. [Online]. Available: https://www.perforce.com/blog/sca/what-static-analysis

[9] Wikipedia, List of tools for static code analysis. [Online]. Available: https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

[10] VMRAY, Dynamic Analysis, 2024. [Online]. Available: https://www.vmray.com/glossary/dynamic-analysis/

[11] Github, Semgrep. [Online]. Available: https://github.com/semgrep/semgrep

[12] Github, About code scanning with CodeQL. [Online]. Available: https://docs.github.com/en/code-security/code-scanning/introduction-to-code-scanning/about-code-scanning-with-codeql

[13] Bandit, Welcome to Bandit. [Online]. Available: https://bandit.readthedocs.io/en/latest/

[14] ClamAV, ClamAV Documentation. [Online]. Available: https://docs.clamav.net/

[15] YARA, Welcome to YARA's documentation!. [Online]. Available: https://yara.readthedocs.io/en/latest/

[16] Aquasecurity, Tracee: Runtime Security and Forensics using eBPF. [Online]. Available: https://aquasecurity.github.io/tracee/v0.6.4/

[17] Cuckoo, What is Cuckoo?. [Online]. Available: https://cuckoo.readthedocs.io/en/latest/introduction/what/