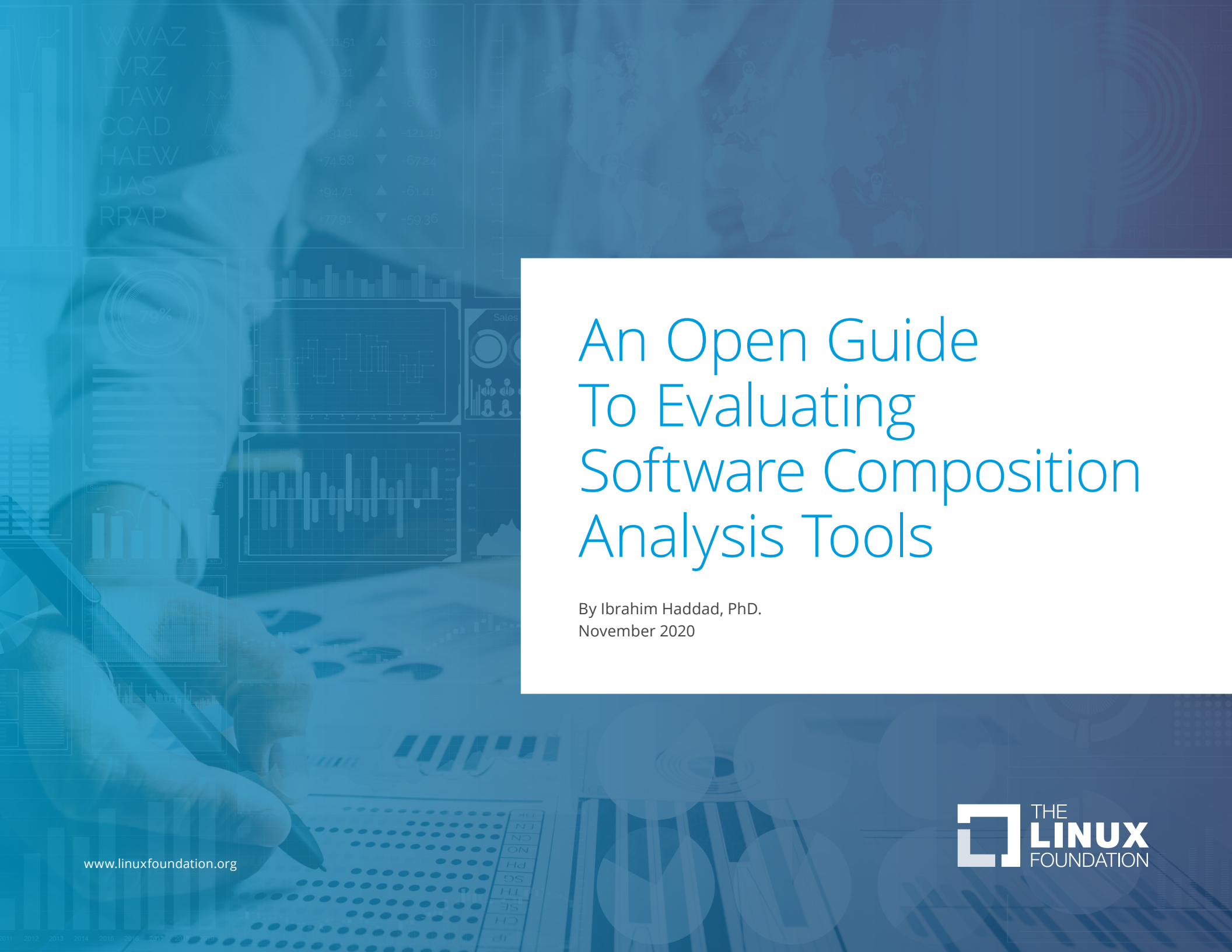

An Open Guide To Evaluating Software Composition Analysis Tools

By Ibrahim Haddad, PhD.
November 2020

www.linuxfoundation.org

THE LINUX FOUNDATION

Introduction

With the help of software composition analysis (SCA) tools, software development teams can track and analyze any open source code brought into a project from a licensing compliance and security vulnerabilities perspective. Such tools discover open source code (at various levels of details and capabilities), their direct and indirect dependencies, licenses in effect, and the presence of any known security vulnerabilities and potential exploits. Several companies provide SCA suites, open source tools, and related services driven as community projects. The question of what tool is most suitable for a specific usage model and environment always comes up. It is difficult to answer given the lack of a standard method to compare and evaluate such tools. Therefore, our goal with this paper is to recommend a series of comparative metrics when evaluating multiple SCA tools.

This paper is a significantly improved version of Chapter 12 from the Open Source Compliance in the Enterprise (2nd Edition) to document and publicize metrics to compare and evaluate SCA tools, collect feedback and drive towards a standardized model of comparison and evaluation. Please note that no size fits all. There are many tools on the market with varying features, maturity levels, deployment models, etc. As you embark on this journey, we highly recommend that you identify the top desired features for your specific environment and requirements, then test and score the tools against those metrics.

Evaluation Metrics

Metric	Description
(1) KNOWLEDGE BASE	• Size of the knowledge base is typically measured as the number of open source projects, and the number of files tracked. This knowledge base stores information about open source software; the larger the database is, the more open source code you will be able to identify as you scan. • List major repositories tracked (e.g., all of NPM, SourceForge, etc..) • What ecosystems are being tracked (e.g., R, Delphi) • What source languages are in scope (based on extension and repository type). Ideally, the scanner should be language agnostics; however, very few vendors provide that support, hence the need to clarify what languages are supported. • Distinguish between package level detection (e.g., Maven) and "Java" support, e.g., you can find jar dependencies but don't actually scan .java source files for copyright/license info) • Frequency of update of the knowledge base. More frequent updates are desired to keep up with the fast pace of open source development. Compliance services and tools providers update their databases regularly. Some companies update three or four times per year; other companies do it at a much higher frequency (up to daily). Ideally, you would want to have the largest and most updated database to increase your chances of identifying newly created open source code. • How long does it take for a customer request to be added to the knowledgeable, is there an SLA for requests? What is the process?

(2) DETECTION CAPABILITIES	• Whole components
	• Ability to correct/configure the analyzer—SW projects are complex with different build setups, and need a way to configure the tool to capture reality.
	• What is the detection methodology used? Different analysis approaches have pros and cons; the tool creator should summarize how their detection works for each used scanner.
	• Partial snippets—ranging from few lines to a partial file
	• What options are offered to correct and verify its results? Does it support the ability to rank results (e.g., P1 or Serious, etc.)?
	• Ability to auto-identify code with proper origin and license without the need of a compliance engineer directing the tool on what's a correct match and what's a false positive. Many of the source code scanning engines, especially those with snippet support, do generate a significant number of false positives that need to be investigated and must be resolved manually. The endless hours of manual labor generated by these false positives is an ongoing problem with some of the most known products in the market today. When evaluating such products, we recommend prioritizing scanning engines capable of auto-identifying source code snippets leading to the least amount of false positives that you need to vet manually.
	• Supports which type of analysis (distinguish between package level detection and "exact" style file detection used to discover single file copies of the source, binaries, multimedia files):
	  • Source scanners (code -> which OSS package(s)?
	  • Binary scanners (binary -> which OSS package(s)?)
	  • Snippet scanners (code fragment -> copied from which OSS package(s))
	  • Dependency scanners (code -> which dependencies are included via a package manager)
	  • License scanners (code -> OSS licenses?)
	  • What languages are supported? If a language is supported, is that for snippet analysis as well? Package level only? Exact file matching?

(2) DETECTION CAPABILITIES (cont.)	• Security scanners (code -> vulnerabilities?) • Other vulnerabilities detection techniques include search terms, email/URL detection, web service detection, etc.
(3) EASE OF USE	Ease of use is important because if all your engineers have access and use the scanning tool (versus only compliance engineers), you may avoid compliance problems way before they arise and before engineers integrate the new code with your build system. You would want an easy to use tool that minimizes the learning curve and avoid the need for costly professional training. • Intuitive design and user interface • Availability of local client or browser plugin • Availability of mobile client • Requires minimal to no training to run but training is provided to " but understanding how to examine and evaluate the results Please note that ease of use is a very subjective criterion and hard to quantify or qualify. However, some tools are a lot easier to use and navigate than others.

(4) OPERATIONAL CAPABILITIES	• Speed of source code scans: Speed of source code scans is a pain point for many products on the market today. For instance, one specific company designed and developed its own database that is perfectly suitable for manipulating the type of such data. As a result, they have lightning-fast scans that are exponentially faster than other existing tools. Furthermore, scans' speed is particularly useful when you integrate the scanning tool with your continuous integration process. One aspect of being aware of is the question of speed when files are being skipped. Another is the question of whether actual copyright and license detection are happening or scan only of repository/package management files such as pom.xml. • Ability to use the tool for scans related to M&A activities without a licensing lock on usage models: Some tool vendors impose limitations via their licensing agreement on your ability to use the tool in scenarios outside just scanning code related to ongoing development efforts. You need to be aware of this fact and make sure that you can use the tool, for instance, for any M&A transaction your company is considering. • Support for different audit models: There are three audit models (discussed in the following chapter): traditional, blind, and DIY. All companies support the traditional model. Very few support DIY. Only one supports the blind audit model, which provides the most secure and private auditing model in M&A scenarios. • Programming language agnostic: Some tools are, by the admission of their creators, very good working with specific programming languages, and not so with others. This is interesting, as you would expect any scanning and identification engine to be agnostic to programming languages. Most tools are not; very few are agnostic to languages. • Ability to re-use scan clarification across the organization • Build system (CD/CI) agnostic
(5) INTEGRATION CAPABILITIES	• Provides APIs for easy integration and command-line interface (CLI): Using a scanning tool is not limited to UI-based usage. Ideally, companies want to integrate the tool with their existing development and build systems and processes. Such a scenario is doable if the scanning tool supports APIs and a CLI that would allow system administrators to interact with the tool outside the UI. • Support UI integration capabilities • Ability to integrate an organization's compliance policies within the tool and have the rule flag code as it relates to the declared policies and rules

(6) SECURITY VULNERABILITIES DATABASE

- Size of the vulnerabilities database—the number of vulnerabilities tracked across all projects: This database contains information about known security vulnerabilities that enable the tool to detect security-related problems in the source code. Please note the use of "source code" and not specifically open source code in the previous sentence. The reason being that developers may copy code snippets from open source components into proprietary or third-party components. If the copied code contained a known security vulnerability, then when you scan the proprietary component, your engine should be able to flag the vulnerability.
- Frequency of update of the vulnerability database: Service providers update their databases regularly. The more frequent the update cycle, the better it is to find vulnerabilities as soon as they have been identified.
- Number of sources of vulnerabilities information: Multiple sources can be used to populate the database of security vulnerabilities in open source components. When evaluating compliance tools that offer this service, we recommend investigating this aspect and exploring the updates' actual mechanics. The various sources (direct and indirect) are used to collect information on security vulnerabilities and on which basis recommendations are presented to fix those vulnerabilities.
- Any additional research conducted by the tool provider to validate vulnerabilities' alerts
- Precision (The rate at which vulnerabilities are True Positives). There are 4 levels of True Positives:
    1. The vulnerable software has been correctly mapped to a dependency that is actually used in our proprietary software.
    2. The dependency is used in a critical environment (runtime).
    3. The proprietary software calls the vulnerable part of the dependency in a critical environment.
    4. The vulnerability is exploitable.
- Recall (How much of the potential universe of True vulnerabilities is found and correctly matched to the proprietary software?). In reality, this is impossible to know—comparisons between different solutions to estimate what solutions have the highest recall for a particular technology stack.
- Capability of contextual vulnerability prioritization. General vulnerability severity scores, such as CVSS3, may be inaccurate depending on the proprietary software's environment. Users should be able to contextualize the severity of vulnerabilities to prioritize working with the resolution of those security threats more accurately.

(7) ADVANCED VULNERABILITIES DISCOVERY METHOD	Support for advanced vulnerability discovery—identifying a vulnerability when vulnerable code was copied into a new component (requires support of identifying source code snippets)
(8) ASSOCIATED COSTS	Several cost parameters need to be taken into consideration: • Infrastructure cost: IT infrastructure costs related to hosting the solution or using it via the cloud. It involves the usage of servers that customers need to buy, set up, and maintain, including the cost to upgrade that infrastructure. Depending on its size, the cost of a dedicated system administrator. • Operational cost: Cost related to managing the results that the tool provides. That involves inspecting and interpreting the results and taking appropriate action. A tool that auto-identifies false positives will lower the labor cost to identify those thousands of false positives manually. • Yearly licensing cost: The cost of the yearly software license for using the tool (cost per seat, unlimited seats), cost to access to SDK so you can integrate your internal tools with the scanning engine, and possibly the cost of any private customization that you want to introduce to fit your needs. • Initial cost to integrate with existing engineering/IT tools and infrastructure: Integration costs are hard to estimate, but they typically evolve about the ability to integrate the tool into your workflows and processes with minimal disruptions. • Ability to export project and other information, either for migration to a new system or to preserve knowledge if you leave a vendor • Lock-in cost (the cost to factor if you need to exit the solution and adopt something else): Companies often ignore or do not pay enough attention to the lock-in factor and costs associated with building the whole compliance environment around a specific tool. When choosing a new tool, we recommend putting enough consideration into this aspect. • Cost of engineering customization to meet your specific needs

(9) SUPPORT FOR DEPLOYMENT MODELS	Support for various deployment models: • On-site only • Cloud only • Hybrid What information about your code and projects leave your networks? This should be crystal clear to the end-user: • Actual source and binary files content • Partial file content / string • Hashes • Inventory lists • Policy information • Compliance / non-compliance status
(10) REPORTING CAPABILITIES	• Ability to generate required compliance notices: Are notices based on actual scan results or pulled only from the knowledge base license information? • What about subcomponents or subfiles? Are actual copyrights/licenses included in notices? • What about notices for snippets of open source code? • Support for various reporting capabilities - the ability to export to various formats such as Excel / Spreadsheet. (including the availability of a sample detailed report) • Support for open standard formats (SPDX, SARIF, CVE, CVSS, etc.)

Conclusion

This paper was created out of necessity due to a lack of a unified way to evaluate source code scanning and license identification tools. We hope that you find it helpful in capturing the important aspects of such tools when you are embarking on an evaluation journey of multiple tools, trying to decide which tool is more suitable for your specific needs. If you have suggestions for other metrics that should be covered, or updates to existing metrics, please feel free to directly provide your feedback via the live document on Google drive.

Contributors

The author would like to express his sincere gratitude to all the contributors to this paper. Their feedback and contributions helped shape this paper and improve it significantly.

- Thomas Steenbergen, Head of Open Source at HERE Technologies
- Gilles Gravier, Director, Senior Open Source and Blockchain Strategy Advisor, Wipro
- Jeff Luszcz, Founder and CTO, Palamida
- Gandharva Kumar, Senior Engineering Manager, GOJEK
- Emil Wåreus, Head of Data Science, Debricked

Linux Foundation Open Source Compliance Resources

The Linux Foundation hosts several community-driven projects focusing on collaborative approaches to managing licensing and compliance. These range from development of best practices, to specifications for inter-organizational exchanges of information, to the software tools needed to automate those exchanges. In particular, we would like to specifically mention the following:

- **Open Compliance Program:** The Open Compliance Program website is a starting point for developers and lawyers, particularly those who are new to open source compliance considerations, to learn more about the tools and best practices that can make compliance easier.

- **ACT** (Automating Compliance Tooling) seeks to improve software tooling for detecting and complying with open source licenses. Its goal is to improve the interoperability of open source compliance tools to enable compliance workflows that can be optimized for each company's unique build and release process.

- **OpenChain** defines the key requirements for an organization's open source compliance program. It establishes a conformance program where companies can self-certify to these requirements, with the goal of improving transparency and communication of compliance information across supply chains.

- **SPDX** (Software Package Data Exchange) is a specification for communicating Software Bill of Materials information in a standardized, human- and machine-readable format. It enables better communication of information, including license and copyright details, between organizations and interoperability between compliance tools.

- **Open Source Licensing Basics for Software Developers:** Free online training on open source licensing and compliance tailored specifically for developers.

- **Whitepapers and blog posts:** The Linux Foundation regularly develops and publishes content with recommendations for how to address open source legal issues that arise in day-to-day practice. Here are some representative examples:

    - Docker containers and license compliance: Blog—Whitepaper

    - Guide to Open Source Software for Procurement Professionals: Blog—Whitepaper

- Copyright Notices in OSS projects: Blog
- Summary of GDPR concepts for OSS projects: Whitepaper
- Practical GPL Compliance: Whitepaper
- Open Source Compliance in the Enterprise: Ebook
- Assessment of Open Source Practices as Part of Due Diligence in Merger and Acquisition Transactions: Ebook

Disclaimer

The opinions expressed in this paper are solely the author's and do not necessarily represent the views of current or past employers. The author would like to apologize in advance for any error or omission and is open for feedback and updates via the online document.

Author

Ibrahim Haddad (Ph.D.) is the Executive Director of the LF AI & Data Foundation. He has held technology and portfolio management roles at Ericsson Research, the Open Source Development Labs, Motorola, Palm, Hewlett Packard, Samsung Research, and the Linux Foundation. He is known for his writing and speaking on topics ranging from legal compliance to using open source as an R&D tool to drive collaboration and innovation.

Email: ibrahim@linuxfoundation.org

LinkedIn: linkedin.com/in/ibrahimhaddad

Twitter: @IbrahimAtLinux

Web: ibrahimatlinux.com

The Linux Foundation promotes, protects and standardizes Linux by providing unified resources and services needed for open source to successfully compete with closed platforms.

To learn more about The Linux Foundation or our other initiatives please visit us at **www.linuxfoundation.org**