THE LINUX FOUNDATION

Improving Trust and Security in Open Source Projects

Mark Curphey & David A. Wheeler

A Publication of The Linux Foundation
Third Edition • February 2020

www.linuxfoundation.org

Contents

Summary 3
Overview 4
The Eight Best Practices 6
- Roles and Responsibilities 6
- Security Policy 8
- Know Your Contributors 9
- The Software Supply Chain 11
- Technical Security Guidance 14
- Security Playbooks 16
- Security Testing 17
- Secure Releases and Updates 20

A Certification Scheme 22
Other Security Issues That Need Investment & Help 23
- Security Build Certificate 23
- The Lack of Good Open Source Security Testing Tools 24
- Open Source Package Distribution is a Risk to The Internet 25
- Vulnerability Disclosure is Broken 26

Summary

This document is a proposal to the Linux Foundation (the 'Foundation') to build and operate a program I am calling the Trust and Security Initiative (TSI) and a set of recommendations for other security issues that need investment and help.

The proposed TSI describes a collection of **Eight Best Practices** with specific tasks supporting them that should be used by open-source teams to secure the software they produce as well as a **Certification Scheme** to verify adoption and validate products being shipped that follow it. It is anticipated that the TSI could be adopted introspectively to secure and verify the Foundations projects, could be adopted by Foundation members to secure and verify software they produce and could be made available to the world at large to be adopted and extended as they see fit, raising the collective security bar of the worlds software.

This document is written to provide enough detail to describe a potential end-state and facilitate meaningful discussion but does not itself contain the complete material to operate the program. This proposal builds on previous industry work implementing software security at scale and leans heavily on Microsoft's Trustworthy Computing (TWC) initiative and their Secure Development Lifecycle (SDL). Unlike the SDL which was originally focused on how to secure Microsoft products and was aligned to the way MSFT built closed source software at the time, this proposal is tailored to the open-source community and specifically to modern software development teams, embracing Continuous Delivery and Cloud Native Computing. As such it is opinionated and not intended to be a one size fits all solution to all software development teams.

This proposal also acknowledges (or is of the opinion) that developers generally want to focus on innovation and therefore promotes wide-spread adoption by favoring low cost, low friction practices, especially automation integrated with Linux Foundation projects such as Spinnaker and Kubernetes.

An organization implementing all of the best practices to the highest levels of assurance set out in the TSI will not be immune from security issues such is the complex nature of security but they will undoubtedly raise their bar and provide a higher level of confidence in their software to their userbase. This first draft is intended to be published to a set of industry experts for comment and feedback and it is anticipated that the scheme itself would be updated and enhanced over time based on feedback and adoption.

The **'Other Security Issues That Need Investment and Help'** are recommendations for big ticket items that are either causing current significant pain across the Internet or have the potential to raise the bar on Internet security as a whole.

Overview

If you open the news on any given day and read about the latest data breach, you are reminded that software security is hard. When you take a step back and think about the volume of emerging technology and think about industry trends such as increasing the velocity of software releases and the reuse of code and services, you could be forgiven for holding your hands up and concluding that things are trending in the wrong direction for us to ever have secure software.

But there is hope. The problem of insecure software is not a new one and there is significant prior art for how to achieve it. Commercial companies like Microsoft have made radical changes going from industry pariahs to relatively shiny examples while some new tech companies now bake security into their DNA from the outset. Security of course doesn't come without a cost across the team but after decades of examples we have now learned what works and what doesn't work, and can match techniques and tools to team culture and minimize the impact; in fact in most cases creating a net positive effect on overall development.

This document is arranged into three main sections, **'Eight Best Practices'**, a **'Certification scheme'** and **'Other Security Issues That Need Investment and Help'**. Core sections and / or sub-sections start with text describing the intent and goal of the section and then describes specific practices in a table format. Where possible each practice has been written in a manner so that it can easily be verified for example "The project team publishes the security policy, visibly linked from the main project page and at /security". Using this structure teams implementing the TSI can quickly see what is required and anyone verifying implementation are not required to make subjective decisions. The open source development model itself is of course as varied as the type of projects that embrace it and it is not realistic to develop a "one size fits all" scheme. The scheme as written is designed to be able to be taken and implemented "as is" but also is designed to allow teams to meet the goals with valid alternative solutions or compensating controls so as to allow teams to embed security into their process without having to change their process for the sake of security.

The Eight Best Practices describes a set of "activities" that teams producing secure software should do. This section balances guidance that is meaningful and relatively easy to implement without being overly prescriptive or rigid. They are:

1. Roles and Responsibilities
2. Security Policy
3. Know Your Contributors
4. The Software Supply Chain
5. Technical Security Guidance
6. Security Playbooks
7. Security Testing
8. Secure Releases and Updates

The final section of the document describes a **Certification Scheme** that is designed to enable open-source projects to self-certify, and for commercial open-source companies to provide higher levels of independent third party certification through

a network of Linux Foundation certified security consultants. Throughout the document we allow for levels of security maturity and ease of getting started by describing varying depth of specific practices as **Basic, Standard** and **Advanced.**

Basic practices are considered things that everyone should do, regardless of their project type and maturity. They are generally easy to implement and have a low overhead to the team while providing a basic level of assurance. Knowing that a team applies all of the Basic practices allows consumers to quickly appreciate that all the basics have been thought about and are being implemented.

Standard practices provide a higher level of assurance but usually require a higher degree of overhead therefore are suited to more mature projects and teams. Standard practices require some thought and come with some over-head but are appropriate to software teams producing applications that run in production. Knowing that a team applies all of the Standard practices allows consumers to quickly appreciate that security is important to the project.

Advanced practices go further than Standard and are designed for teams producing mission critical software or for teams wishing to use and or demonstrate security as a differentiator. Advanced practices usually require careful implementation and come with a cost. Knowing that a team applies all of the Advanced practices allows consumers to quickly appreciate that security is of utmost importance to the project.

The Eight Best Practices

Roles and Responsibilities

This sections goal is to define the 'who' of a teams security program. In this section we describe how each organization should assign responsibility for policy and technical security to individuals and make sure that everyone is aware of their responsibilities across the organization. While assigning ownership feels formal, bureaucratic and even old school; a lack of clear roles and responsibilities is one of the biggest root causes leading to security issues and therefore one of the highest value practices any team can undertake. Care has been taken to consider roles and responsibilities in the context of open-source projects. Please note that one individual may fill multiple roles, especially in smaller projects.

Reference	Task and Description	Basic	Standard	Advanced
PO-1	The Organization has assigned an individual to act as the Chief Security Officer. It is essential that there is an individual who is ultimately responsible for security. This individual should be responsible for setting the security policy and making risk based decisions and defining the organization's **security release criteria** (see section 8).	X	X	X
PO-2	The organization has assigned an individual to act as the Lead Security Engineer. The technical security lead works with the CSO to make the technical judgements about vulnerabilities and incidents and is the owner of the organization's security guidance.		X	X

CONT >

Reference	Task and Description	Basic	Standard	Advanced
PO-3	The Organization has assigned an individual to act as the Lead Security Architect. The lead security architect makes and adjudicates security architecture decisions in products produced by the organization. This role owns the Security Architecture guide.		X	X
PO-4	Everyone in the organization is made aware of their responsibilities for security. Security is everyone's responsibility and by ensuring everyone is aware of the organizations policy and key roles and responsibilities such as that of the CSO, the collective power of the team can be harnessed.	X	X	X
PO-5	Everyone in the org attends annual training which explains the policy, people's roles and responsibilities, key processes and covers the top eight best practices. Periodic group training is widely thought of as an immunity booster. Attaching relevant training to company meetings or events is often an effective approach.		X	X
PO-6	The organization has a dedicated full-time security team. By having a set of dedicated security members who only work on security related projects, the organization is not forced to inevitably trade priorities.			X

Security Policy

This sections goal is to define the 'what' of a teams security program. In this section we describe how each organization should publish a policy that describes, at a high level how the organization thinks about and intends to implement security. While the term 'policy' feels formal, bureaucratic and even old school; they can be written in a human voice with clear and concise text that provides a clear north star for all members of the organization. Care has been taken to consider policies in the context of open-source projects.

Reference	Description	Basic	Standard	Advanced
SP-1	Publish an organizational security policy. The organization should create and publish their organization security policy linked from their main page and available at/security.	X	X	X
SP-2	Publish project level security readme files. Each project should publish the security policy along with any project specific overrides in a security readme file that can be found in the root of each projects git repo.		X	X
SP-3	Read and Acknowledge the Policy Everyone in the organization is required to read and acknowledge the security policy. This provides awareness and responsibility to the broad organization.		X	X
SP-4	Attend Annual Security Policy Training Everyone in the organization attends annual training on the security policy to enhance or refresh their knowledge.		X	X
SP-5	Contractors should read and acknowledge the security policy. All contractors and third parties are required to read and acknowledge the security policy. This provides awareness and responsibility to third parties.			X

Know Your Contributors

This section's goal is to define a set of practices so that organizations can trust those contributing to it and so that consumers can trust that the software was produced by well intentioned people. In the current cyber security climate we have seen malicious backdoors and malware become commonplace in open-source such as NPM packages and such offensive techniques of poisoning upstream code is a known offensive playbook of cyber warfare. Knowing your contributors so you and your projects consumers can make risk based decisions about what to trust just makes for common sense. As with other sections care has been taken to consider policies in the context of open-source projects.

Reference	Description	Basic	Standard	Advanced
KYC-1	Verify The Identity of All Contributors Knowing who is contributing to designs and implementation allows you to place trust in those individuals. Identity verification can range from personal verification with known individuals to checking government issued identification and online systems like those that use personal and financial information.	X	X	X
KYC-2	Require Strong Authentication Using strong authentication such as multiple factors means a higher level of trust and more confidence in maintaining the identity chain.	X	X	X
KYC-3	Roles Based Access and Principle of Least Privilege Only grant users the permissions to do their job and no more. Use roles to place users into common groups and assign permissions to the group rather than to individuals.		X	X

CONT >

Reference	Description	Basic	Standard	Advanced
KYC-4	Implement a Contributor License Agreement Requiring that all code commits require the completion of legally enforceable contributor license agreement, raises the bar on the quality of contributions. Adding security provisions into the CLA usually leads to a higher level of assurance.		X	X
KYC-5	Publish list of contributors and their contributions By publishing a complete list of all contributions to the software, consumers are able to make their own decisions about what to trust by correlating contributors and their work. For instance caution can be taken if an unknown developer contributed complex data handling code.			X

The Software Supply Chain

Attacks on the software supply chain have become commonplace with adversaries clearly understanding that they can have a bigger and more effective impact with less effort than targeting individual systems or indeed individuals. This section describes how to lock-down the toolchain and verify things that flow across it and as with all other sections care has been taken to consider policies in the context of open-source projects. Like other sections there are a myriad of potential additional controls that could be suggested here but in the spirit of making the TSI something that could be relatively easily adopted with as little friction as is necessary for organizations we have chosen what we consider to the ones that offer the biggest bang for the buck.

Reference	Description	Basic	Standard	Advanced
SC-1	Accept PR's Only Git Pull Requests or PR's should be the only way to get changes onto master branches. PR's should implement a peer review which includes security.	X		
SC-2	Use Protected branches Protected branches should be configured for all projects to prevent developers making changes without a PR and to avoid the commit history from being modified.		X	X
SC-3	Use Digitally Signed Commits Git systems can digitally sign commits which provide for a stronger authentication to verify the committer and which provides an integrity check on the commit itself.		X	X
SC-4	Assign Security Issues Immediately Security issues should be investigated immediately triaged and assigned a risk level and an owner before being acted on in accordance with the risk.	X	X	X

CONT >

Reference	Description	Basic	Standard	Advanced
SC-5	Use private issues for security issues. Use private issues to users to submit security issues. Security issues should only be seen by the projects security team until they are ready to be disclosed in accordance with your disclosure policy.		X	X
SC-6	Control the build servers access to infrastructure, code and packages. Package managers and build tools and very powerful and if coerced into malicious behavior can be dangerous. Build servers need to be setup so they cant damage infrastructure (including their own hosting), modify code they are building (beyond optimizations) and only access trusted packages.			X
SC-7	Use GRAFEAS and SPIFFE to authenticate your build pipeline ? NEED RESEARCH			X
SC-8	Use a secrets management system Never allow secrets to be stored on code and use a secrets management solution like Hashicorp Vault. Scan commits to prevent secrets from being committed.		X	X
SC-9	Don’t use vulnerable code in libraries. Block the use of vulnerable packages where you call vulnerable code in the build process. Implement software composition analysis (SCA).	X	X	X

CONT >

Reference	Description	Basic	Standard	Advanced
SC-10	Use a private package repo Use a private package repository which contains vetted and secure libraries. Ensure that the build servers can only connect to this system. Cache all binaries and code into this repo to protect from potential upstream availability issues.		X	X
SC-11	Use only valid signed packages Digitally signed packages provide a higher level of assurance of who created the software. All major package systems support signing although to date few packages sign. Please note PGP as implemented by maven central is not a good system and should not satisfy this requirement.			X
SC-12	Verify security of Open Source Verify the security of others open-source before incorporating it into your projects. Open source code should be scanned with SAST, manually review it if appropriate and look at the projects issue list and history.			X

Technical Security Guidance

Much like the value in having a security policy that serves as a north star on how you want people to do security, technical security guidance is needed to narrow potential solutions down to the ones an organization feels are appropriate and that provide the desired level of security. Technical security guidance has always suffered from two major afflictions. The first is that the surface area is vast and rapidly changing meaning documentation is rarely complete and very often out of date. The second is that when dealing with software there are many ways to accomplish the same things and two alternative security solutions may offer the same level of assurance. In many ways both fundamental issues are 'scale' problems that I think the Linux Foundation could solve by maintaining centralized core technical guidance and allowing organizations to extend and customize it. This would make an ideal 'value add" for a subscription based membership organization. Of course just like other sections care has been taken to consider policies in the context of open-source projects.

Reference	Description	Basic	Standard	Advanced
MTSG-1	Have an AppSec guide Have an application security that provides prescriptive guidance to your developers on how your organization wants them to avoid common security issues like those described in the OWASP Top Ten.		X	X
MTSG-2	Have an OS security configuration guide Have a technical guide that provides prescriptive guidance on how your organization requires OS's to be configured, including security latches, accounts and network configuration.		X	X
MTSG-3	Have a cloud security configuration guide Have a technical guide that provides prescriptive guidance on how your organization requires your cloud environment to be configured.		X	X

CONT >

Reference	Description	Basic	Standard	Advanced
MTSG-4	Have language specific security guidelines for all languages that you use Provide language specific guidance to avoid security issues inherent to languages, how to use security features of languages and provide reusable solutions to common problems in code.			X
MTSG-5	Have a security architecture best practices Have a technical guide that provides prescriptive guidance on how your organization requires your applications to be designed and architected. Provide reusable solutions to common problems in code.		X	X
MTSG-6	Have a cryptography guide Have a technical guide that provides prescriptive guidance on how your organization requires your applications to use cryptoga[hy. Include crypto libs, algorithms, key management and key lengths.		X	X

Security Playbooks

This sections goal is to define 'how' to do specific security processes, specifically incident response and vulnerability management processes. Like creating roles and responsibilities or publishing security policies this may feel formal, antiquated and old school but having pre-defined playbooks means that teams can focus on shipping software and not learning how to do security, especially at what is usually the time that is least convenient and most stressful. Like the technical security guidance I think the Linux Foundation could develop and maintain a centralized set of playbooks and allowing organizations to extend and customize them. This would again make an ideal 'value add' for a subscription based membership organization. And yes as with other sections care has been taken to consider the need to document security processes in the context of open-source projects.

Reference	Description	Basic	Standard	Advanced
SP-1	Incident Response Playbook An incident response playbook should be published that documents important ways in which an incident should be handled in your organization including; What is an incidents (including levels), Roles and Responsibilities, Service Level Agreement, and Communication Protocols.		X	X
SP-2	Vulnerability Management Playbook A vulnerability management playbook should be published that documents how the organization manages vulnerabilities including; Vulnerability types and severities, Roles and responsibilities, Service Level Agreement, and Communication Protocols.		X	X

Security Testing

This section describes various techniques and levels of security testing. In this section is a strong preference for automated testing which scales better, has less friction and less cost to the teams and aligns well to modern Continuous Delivery. The section does also cover some levels of manual testing, either for areas where currently automation is not available or practical and or to provide additional or higher levels of assurance under specific circumstances. Like all sections care has been taken to consider the need to document security processes in the context of open-source projects and it shouldn be worth noting that elsewhere in the document I recommend the Linux Foundation invests in building free open-source versions of some of these tools. High quality free versions do not exist today and this is an area that if they were available and widely used the security quality of software they were used on would likely significantly rise.

Reference	Description	Basic	Standard	Advanced
PST-1	Implement security linting on all checkins "Linting" code can prevent simple security issues such as developers checking in secret keys or using vulnerable deprecated functions. It is a low effort, high value activity that should be done by everyone.	X	X	X
PST-2	Perform a final manual security review on all major releases. While "automation is king" the current state of security tools is poor. Having a skilled human check a release including confirming that items in this list have been complete and using domain knowledge for each major release serves as a valuable additional control gate.	X	X	X

CONT >

Reference	Description	Basic	Standard	Advanced
PST-3	Do manual security reviews on all minor releases, having a skilled human check a release including confirming that items in this list have been complete and using domain knowledge serves as a valuable additional control gate. Perform a final manual security view on all minor release. Do manual security reviews on all minor releases, having a skilled human check a release including confirming that items in this list have been complete and using domain knowledge serves as a valuable additional control gate.		X	X
PST-4	Perform threat modelling on new projects during design Threat modelling is a process of determining the potential threats to a system and identifying countermeasures. Performing threat modeling on new systems during design helps avoid mistakes typically found in downstream testing.		X	X
PST-5	Perform threat model on all major architectural changes Performing threat modeling when changes are proposed to major system architecture or components, helps avoid mistakes typically found in downstream testing.		X	X
PST-6	Use control flow based SAST (Static Analysis Security Testing) on all major releases Control flow analysis balances speed and completeness and can be useful in identifying potential security issues.	X	X	X
PST-7	Use control flow based SAST on all merges to master Control flow analysis balances speed and completeness and can be useful in identifying potential security issues.		X	X

CONT >

Reference	Description	Basic	Standard	Advanced
PST-8	Use data flow based SAST on all releases Data flow analysis is usually slow but favors completeness and can be useful in identifying potential vulnerabilities.	X	X	X
PST-9	Use data flow based SAST on all merges to master Data flow analysis is usually slow but favors completeness and can be useful in identifying potential vulnerabilities.			X
PST-10	Use SCA tools Use Software Composition Analysis (SCA) tools and block the use of all components where a vulnerable method is being used and potentially exploitable	X	X	X

Note: With SAST there is control flow analysis that generates a control flow graph and iterates over it to find potential security issues. It is faster than data flow analysis but not as complete. Data flow analysis creates a control flow graph and a data flow graph. This is slower but more complete.

Note: More than 90% of the time vulnerable open source components are being used, the vulnerable parts of the code are not being called and therefore there is not an immediate vulnerability. Only when the vulnerable method of a vulnerable library is used should there be considered a vulnerability.

Secure Releases and Updates

The final of the eight best practices is arguably the most important for end users and in many ways can be considered the one that will have the most visible impact. By defining a "secure release" and being transparent about what went into that definition, organizations can earn the trust of consumers and consumers can adopt open-source projects with a higher degree of confidence. The Certification scheme that is later in this document proposes that the Linux Foundation hosts a directory of certified secure releases and has a process for invalidating releases when issues are found or incidents occur. In many ways this is analogous to running a secure open-source package distribution system, something else I believe the foundation should consider.

Reference	Description	Basic	Standard	Advanced
SRU-1	Have a security release criteria Having a security release criteria that defines the security quality of releases means that engineering have a definition of what is acceptable and that auditors and users can validate security against this criteria. Criteria should include processes that have been followed and results such as no high risk vulnerabilities.	X	X	X
SRU-2	Digitally sign releases By digitally signing software releases, users and tools built around users can verify the identity of the developers and therefore make decisions about what to trust. Note: hashes on web sites and PGP doesn't count.		X	X
SRU-3	Ship a "Security Build Certificate" with with each release. See the Security Build Certificate section later in this document.		X	X

CONT >

SRU-4	Publish information about and deprecate insecure versions Armed with vulnerability information users will almost always chose safe versions. When vulnerabilities are found in versions, the organization should deprecate the version and publish information about the vulnerabilities.	X	X
SRU-5	Implement an automated secure update system As has been shown in operating systems, users will take security updates if the mechanism is easy. Free open source technology such as the Trusted Update Framework already exists to fulfill this need.		X

A Certification Scheme

There is a real opportunity to build and operate a "certification Scheme" that is based on the established model of the Cloud Security Alliance or CSA (**https://cloudsecurityalliance.org/**) STAR program (**https://cloudsecurityalliance.org/star/**). Here are some examples from **Atlassian**, **SalesForce** and **GitHub**.

At a high level the CSA publishes a set of criteria much like the eight best practices that proceed this section in the document. The CSA then allows software producers to register for free on their web site and self-certify with the results being published to a central directory. The CSA additionally certifies security consultants who can provide independent assurance.

The scheme works because it has incentives for the software producers, software consumers and the security consulting industry.

The advantage to software producers is that they can point potential customers to a directory location rather than the tiresome buredon of completing vendor security questionnaires.

The advantage to consumers is that they can quickly look in a single directory for answers avoiding procurement duplication.

The advantage to the security industry is that they can earn services revenue assessing and remediating security issues for companies.

The model has worked extremely well for cloud service providers and I believe a version of this would work extremely well for open-source software providers. The foundation could find ways to include levels of access to material that help an organization get certified such as security standards (see previous sections) and even discounts to certified assessors.

Other Security Issues That Need Investment & Help

This section describes a set of specific security issues that fall outside of the Trustworthy Security Initiative but represent significant security issues facing the industry and as such need to be addressed.

Security Build Certificate

One of the greatest challenges the industry faces is being able to know "how secure" a product is. Today there is rarely a transparent way to know what has been done and what the results are leaving consumers to be "in the dark" and vendors able to hide behind their own practices often disguised as subjective opinion. While analogies almost always have flaws, it's hard to imagine a car coming off a production line without a safety certificate yet acceptable that software including software that goes into the car has no such thing.

I propose that the foundation builds a specification and a set of tools that produce a certificate that can be attached to a software release that addresses this. The certificate would be able to be validated against a release and allow both a machine and or human to be able to parse a set of security claims and reproduce them if needed. For example a security vendor may claim that they have ran a security tools over the code base and remediated all high risk findings. A consumer should be able to replicate this test and verify that this was indeed the case. The types of verification can and should include the use of automated security tools like SAST, DAST and SCA as well as verification of security processes like the presence of security readmes in repos and that security response emails are valid.

I am very aware that there are many details that would need to be figured out and that there would be significant political alignment required from software vendors and from security tools vendors, but a scheme like this could have a significant and lasting effect on the security quality of open source software and the internet at large.

The Lack of Good Open Source Security Testing Tools

One of the greatest impediments to the widespread adoption of security testing tools is the lack of credible free open-source options and while its true that some companies like Coverity offer limited free scanning, commercial tools are cumbersome, not extensible and don't fit well with the ethos and culture of open-source projects. I believe that having high quality free open source security testing tools would have a significant impact on open-source projects willingness and ability to find security issues and I propose that the foundation invests in developing them.

Static Analysis Security Testing commercial tools can be as much as a million dollars a year for making them out of the reach or used sparingly for those that can afford them. They are also generally produced by late stage companies that are focused on maximizing revenue and not innovating or keeping up with the pace of change of languages and development techniques like DevOps. No current commercial tools for instance would work well in a CNCF Cloud Native pipeline based in Spinnaker and the closed source also means that teams can't extend or customize tools to fit their use cases.

A good example is that in the Java community the FindBugs tool has seen little development for almost ten years, with frustrated users forking the core code periodically and that fork then inevitably dying.

There are several promising projects that could be used as the base for such an effort. All have significant holes and may well not work together well and so it's entirely possible that starting from scratch is a more sensible approach.

FindSecBugs - **https://find-sec-bugs.github.io/**

Zap - **https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project**

SRCLib - **https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project**

OWASP Dependency Checker - **https://www.owasp.org/index.php/OWASP_Dependency_Check**

OSS Fuzz and ClsuterFuzz - **https://github.com/google/oss-fuzz https://google.github.io/clusterfuzz/**

I would recommend trying to recruit Jacob West to lead this effort. Jacob is an amazing guy, a first class human, local to San Francisco, one of the core engineers at Fortify and co-author of the defacto static analysis book.

Open Source Package Distribution is a Risk to The Internet

Open source package distribution is plain broken. At the heart of the problem is the fact that commercial companies have inherited the sites that have become the defacto distribution systems for languages or frameworks such as Maven Central and NPMJS. Their commercial interests do not line up with those of the Internet at large resulting in a multitude of broken situations such as:

- you can not take a complete copy of all free open source java libraries to analyze their security without asking SonaType (who will say no).
- None of the package distribution sites offer / enforce strong authentication for publishers
- Sites provide weak signing such as Mavens PGP which only verifies the publisher owns an email address.
- None of the publishers deplecate vulnerable packages
- NPMJS offers a free audit service which is notoriously laden with false positives and appears to be targeting an upsell into a paid version.
- All registries have had upstream malware which is removed on a best efforts basis. No registries to my knowledge have invested in building technologies such as those from Google or Apple to protect they upstream appstores.

I propose the foundation develops and operates a central library distribution system for all supported languages and builds in the appropriate security features.

Vulnerability Disclosure is Broken

Knowing what vulnerabilities exist in open source code is critical in being able to determine if your code or code you are consuming is vulnerable. The Common Vulnerability and Exposure system that was developed and operated by Mitre has long stood as the defacto vulnerability disclosure system and central database of all vulnerabilities.

About a decade ago when development practices started to change CVE started to become irrelevant and has been unable to adapt. There are several underlying problems.

Format - The CVE format itself is a human readable format that is not intended to be parsed by tools and describe were in code and under what circumstances a vulnerability can occur. The CVE format was also created before dependency managers were prevalent and so have no real notion of the impact of dependent libraries. As a result CVE's today are simply pointers to potential issues and many that would appear to be correct on the surface are simply not when analyzed.

Disclosure Process - The generally followed and widely accepted coordinated disclosure process was born from an IETF draft authored by Chris Wysopal. At the time we lived in a world of predominantly waterfall created closed source software created by a relatively small number of vendors and so the draft aligned to giving vendors long periods of time behind closed doors to fix issues. This doesn't work in an open-source devops world where bad actors can research potential issues and hunt for in-flight fixes. We need to rethink disclosure for the era of devops and open-source and develop a new IETF style draft that the industry can support.

Scale - CVE was designed in an era when we had a handful of large software vendors and a handful of products. As such they would follow process and the flow of issues was manageable. Today we have millions of software producers releasing millions of products, many as open source libraries. When combined with the speeds of DevOps, most developers today fix issues inline, sometimes documenting the fix in a commit log or readme but rarely getting a CVE number assigned. I am aware of a study conducted for the US Intelligence community in which it was suggested that there are approximately 250,000 vulnerabilities in open source libraries compared to the approximateoy 10,000 disclosed through the CVE system.

Commercial Conflicts - Vulnerabilities that can be exploited have a very large value on the dark market. So called Zero day vulnerabilities in popular software that can give root or equivalent access can sell for as much as a million dollars a time. Software Composition Vendors have found that searching for what have become know as half-days (in plain sight in commit comments and similar if you know where to look) are a valuable differentiator and security researchers are able to monetize zero days into a profitable business. Malware gangs continue to exploit these issues and we are starting to see some uses in the creation of ransomware.

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**