NERC
NORTH AMERICAN ELECTRIC RELIABILITY CORPORATION

Security Guideline

Risk Considerations for Open Source Software

December 6, 2022

RELIABILITY | RESILIENCE | SECURITY

3353 Peachtree Road NE
Suite 600, North Tower
Atlanta, GA 30326
404-446-2560 | www.nerc.com

Table of Contents

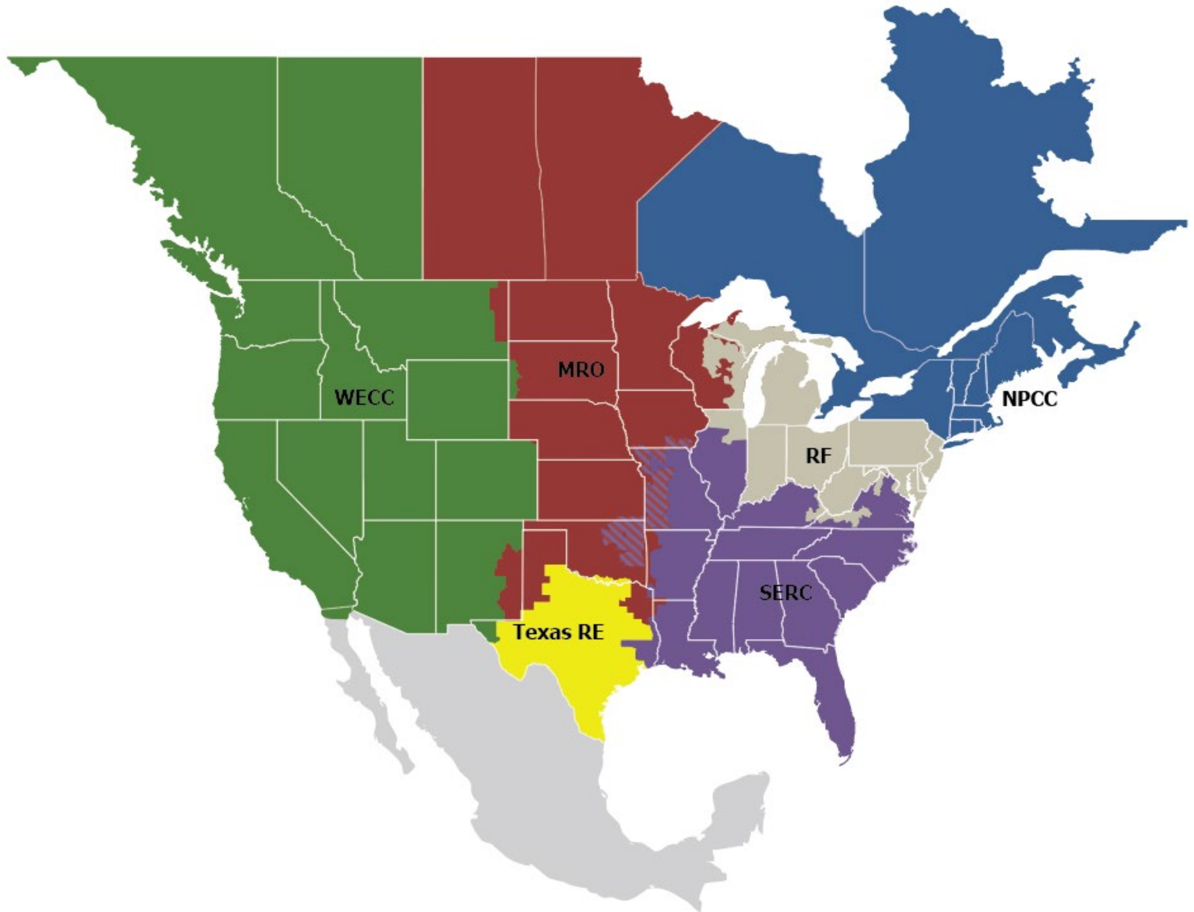
Preface

Electricity is a key component of the fabric of modern society and the Electric Reliability Organization (ERO) Enterprise serves to strengthen that fabric. The vision for the ERO Enterprise, which is comprised of the North American Electric Reliability Corporation (NERC) and the six Regional Entities, is a highly reliable and secure North American bulk power system (BPS). Our mission is to assure the effective and efficient reduction of risks to the reliability and security of the grid.

Reliability | Resilience | Security

Because nearly 400 million citizens in North America are counting on us

The North American BPS is made up of six Regional Entity boundaries as shown in the map and corresponding table below. The multicolored area denotes overlap as some load-serving entities participate in one Regional Entity while associated Transmission Owners/Operators participate in another.



MRO	Midwest Reliability Organization
NPCC	Northeast Power Coordinating Council
RF	ReliabilityFirst
SERC	SERC Reliability Corporation
Texas RE	Texas Reliability Entity
WECC	WECC

Preamble

The NERC Reliability and Security Technical Committee (RSTC), through its subcommittees and working groups, develops and triennially reviews reliability and security guidelines in accordance with the procedures set forth in the RSTC Charter. Reliability and security guidelines include the collective experience, expertise, and judgment of the industry on matters that impact BPS operations, planning, and security. Reliability and security guidelines provide key practices, guidance, and information on specific issues critical to promote and maintain a highly reliable and secure BPS.

Each entity registered in the NERC compliance registry is responsible and accountable for maintaining reliability and compliance with applicable mandatory Reliability Standards. Reliability and security guidelines are not binding norms or parameters nor are they Reliability Standards; however, NERC encourages entities to review, validate, adjust, and/or develop a program with the practices set forth in this guideline. Entities should review this guideline in detail and in conjunction with evaluations of their internal processes and procedures; these reviews could highlight that appropriate changes are needed, and these changes should be done with consideration of system design, configuration, and business practices.

Executive Summary

"Open-source software is software with source code that anyone can inspect, modify, and enhance."[1] It may or may not also be available for download in binary form (ready to use). Common examples of open-source software are the Linux Operating System, Mozilla Firefox, and Google Chromium (an open-source browser project that forms the basis of Google Chrome). Users of open-source software may merely download and use it as is, may download source code and modify or build it for internal or product use, and may participate as members of the development community. These usage patterns differ significantly in the degree of expertise and familiarity that the user has with the code; thus, it also affects the risk organizations incur when using it.

Users of open-source software have to assess the risks of using open-source software with what they can learn about its trustworthiness.

The following are some key observations in this guideline:

- The average software product today includes over 135 components; some products include thousands of components, and about 90% of those components are open source.[2] In other words, the greatest part of the code and risk in almost any modern software product comes from open-source components.
- A useful way to analyze risk and set priorities for reducing risk is to create a threat model for the system being considered. A threat model helps an organization to determine the type and extent of compensating security controls that might be needed to reduce risk to an acceptable level.
- Risks of software can be roughly divided into two categories: risk that the software might not be authentic (a malicious look-alike) and risk that the software has defects that can be exploited as security vulnerabilities or that might cause misoperations.
- Because membership in a group of developers for open-source projects is dynamic and without borders, there is a risk that a developer may join the group who has or develops malicious intent.
- It may make sense to remain a version or two behind a released current version unless a security vulnerability forces an upgrade to the current version. In any case, installing the update first on a test system is important because it provides an opportunity to test that critical functionality remains intact and undesirable behaviors are not observed.
- If software becomes unsupported, usually when the development group announces that one or more versions will no longer be supported, the software poses an unknown risk because any security issues discovered in the software will not be fixed, and an organization may not be made aware of them.
- Both software products and software components can be created with names designed to take advantage of typos or inattention on the part of developers or users of software.
- The Internet provides many resources that will provide evidence bearing on the activity, reputation, and trustworthiness both of a software product and of its development community.
- Studying some past cyber security issues in the software supply chain helps to understand the following risks and illustrate how defensive principles can be applied.
- Because of the difficulty of inspecting software (especially in the case where a contractual relationship does not exist with the supplier), it may be necessary to take steps to mitigate the risks of operating it in a specific environment.

[1] https://opensource.com/resources/what-open-source
[2] Ibid

Introduction

Purpose

This security guideline provides information about factors that contribute to the risk that open-source software might pose to an organization's risk assessment methods. Besides discussing how to assess risks, it also has suggestions for mitigation techniques for addressing risk, practices for obtaining evidence of trustworthiness, and options for controlling risk once the decision has been made to use open-source software.

Background

"Open-source software is software with source code that anyone can inspect, modify, and enhance."[3] It may or may not also be available for download in binary form (ready to use). Common examples of open-source software are the Linux Operating System, Mozilla Firefox, and Google Chromium (an open-source browser project that forms the basis of Google Chrome).

Open-source software is updated periodically and updated versions incorporate submissions from a community of contributing developers to implement new features, address security vulnerabilities, and fixes bugs/flaws/glitches. Users obtain the software—as well as updates when new versions are released—from a distribution point maintained by the community.

User trust in open-source software involves a combination of trust in the distribution point for the software, confidence in the supporting community of developers and supporters, and observations from inspecting the code; each of these vary by degrees depending upon the user's technical capability. Most users are "downloaders" and entirely dependent on the development and user communities to maintain the security of the software.

Developers of open-source software differ widely in their assumptions regarding the technical capabilities of the users who will download the software. The thoroughness of documentation and the level of support to realistically expect from the developer community can also vary greatly. Because there is no buyer-seller relationship or contract to establish requirements and expectations, users must evaluate their ability to support their use of the product themselves, including proper installation, configuration, and ongoing monitoring of the distribution point for news of security vulnerabilities and updates.

[3] https://opensource.com/resources/what-open-source

Chapter 1: Risk and Open-Source Software

Risks Due to Open-Source Software Components in Proprietary Software

The average software product today includes over 135 components;[4] some products include thousands of components, and typically, about 90% of them are open source.[5] In other words, the greatest part of the code and risk in almost any modern software product comes from open-source components.

Software users may be able to directly download patches for vulnerabilities in open-source components if the component is installed alongside the application, but it is more common for those components to be built in, meaning that users rely on the supplier of the application to provide an official patch. Regardless, knowing "the software in the software" (i.e., the components included in the software) that a user organization runs is the key to mitigating risk. An up-to-date "Software Bill of Materials" (SBOM)[6] from the software supplier contains a list of those components.

Having an SBOM for a software product will enable a user organization to know when they might be affected by vulnerabilities in a product's components (both open source and proprietary). Organizations can coordinate with the software supplier to understand and mitigate the risk posed by those vulnerabilities. In most cases, the mitigation will simply be applying a patch provided by the supplier; however, other mitigation measures may be required in some cases.

Obtaining an SBOM for an open-source software product may be difficult. Suppliers of proprietary software may not offer it, and obtaining an SBOM for an open-source product may require that an organization use software development tools to obtain the information. In these cases, the best that can be done is to reduce risk in a general way.

Analyzing Operational Risk: What Can Go Wrong?

A useful way to analyze risk and set priorities for reducing risk is to create a *threat model*[7] for the system being considered. A threat model describes the things that affect the system, the things the system can affect, and the criticality of those things. It also describes what can go wrong, whether accidentally or deliberately induced by an attacker, and how likely those are as well as ways those threats can be reduced. The following is a list of common things to consider when analyzing operational risks for software:

- What systems can be connected to or are reachable from the system and how critical are they?
- What would be the impact (criticality) if the system failed (went off-line or was disabled)?
- What would be the impact (criticality) if the system misoperated (had inappropriate outputs)?
- Where do the system's inputs come from and how trustworthy are those inputs?
- How directly can the system be reached by an external attacker?
- Does the software require an Internet connection?
- Can the system be reached by an internal network user in a different department?
- Does the software require or run with administrative privilege?
- How trustworthy is the software?

[4] https://www.sonatype.com/resources/white-paper-state-of-the-software-supply-chain-2020
[5] Ibid
[6] https://www.ntia.doc.gov/SBOM
[7] https://www.upguard.com/blog/what-is-threat-modelling

A threat model helps an organization to determine the type and extent of compensating security controls that might be needed to reduce risk to an acceptable level.[8]

Evaluating Trustworthiness of Software

Risks from software can be roughly divided into two categories: risk that the software might not be authentic (a malicious look-alike) and risk that the software has defects that can be exploited as security vulnerabilities or that might cause misoperation. There is also some possibility that the software might contain deliberately introduced malicious functionality; this should also be considered as a possible defect.

The risk of software being inauthentic or having been tampered with can be reduced by always obtaining software from its original point of distribution; normally that would be from a web site maintained by its creators. The original distribution point can usually be found by using an Internet search engine or perhaps in blog postings by its users. The distribution site should be protected by transport layer security (an HTTPS URL) that allows an organization to check the web site certificate to ensure that it is valid and that the certificate was issued to a “subject” or “subject alternate name” that is consistent with expectations for the software.[9]

The risk from defects is determined by a combination of the probability that defects will exist, offset by the existence of an active community of maintainers, the competence of their development and testing processes, and their responsiveness in addressing defects.

The Problem of Malicious Changes

Because membership in a group of developers for open-source projects is dynamic and without borders, there is a risk that a developer may join the group who has or develops malicious intent; thus, a once-trustworthy piece of software may become a threat.[10]

The following links provide examples of malicious software changes:

- “Malicious code that stole authentication credentials”: https://arstechnica.com/gadgets/2021/04/backdoored-developer-tool-that-stole-credentials-escaped-notice-for-3-months/ -
- “Set of packages steadily grew… more than 200 packages … the malicious payloads of these packages were PII stealers”: https://jfrog.com/blog/large-scale-npm-attack-targets-azure-developers-with-malicious-packages/
- “Would fetch malicious code from pastebin.com and collect … sensitive information from the … host”: https://www.helpnetsecurity.com/2019/08/21/backdoored-ruby-gems/ -
- “Purposefully corrupted … open-source libraries … that thousands of users depend on”: https://www.theverge.com/2022/1/9/22874949/developer-corrupts-open-source-libraries-projects-affected -
- “Discovery of three malicious npm cryptomining packages … impersonating a popular, legitimate library”: https://blog.sonatype.com/npm-project-used-by-millions-hijacked-in-supply-chain-attack

Organizations should pay attention to shifts in ownership or in who appears to be leading the project. Malicious changes or unwanted functionality can occur if a project changes ownership, which is especially likely if a project is

[8] https://en.wikipedia.org/wiki/Threat_model
[9] https://www.globalsign.com/en/ssl-information-center/what-is-an-ssl-certificate/
[10] https://www.darkreading.com/application-security/how-hackers-infiltrate-open-source-projects

abandoned. There have been instances of a malicious maintainer adopting a project from an owner who no longer wishes to maintain it. The new owner then took advantage of an established user base to gain backdoor access to victim computer systems.

Trust and the Problem of Updating

Related to the problem of malicious changes to the software is the issue of how to approach updates from the supplier. Depending on the level of trustworthiness, it may make sense to remain a version or two behind a released current version unless a security vulnerability forces an upgrade to the current version to resolve it. Keeping a prior step/version provides an opportunity for defects to be discovered and reported by the user community.

In any case, installing the update first on a test system is important because it provides an opportunity to test that the critical functionality remains intact and undesirable behaviors are not observed.

What If Software Becomes End-of-Life or Unsupported?

If software becomes unsupported, usually when the development group announces that one or more versions will no longer be supported, the software poses an unknown risk because any security issues discovered in the software will not be fixed, and an organization may not be made aware of them even if vulnerabilities become known. An open-source project can also be simply abandoned with no announcement at all.

Unsupported software must be regarded as untrusted. If used, compensating security measures should be applied (see **Mitigating Risk**).

A Special Trust Problem: Unintended Software Because of Name or Namespace Confusion

Both software products and components can be created with names designed to take advantage of typos or inattention on the part of developers or users of software. Users can be tricked into downloading software from an unintended repository,[11] and/or not realizing that the software has a name that is "almost" correct[12] or a lookalike (e.g., missing or repeated characters; similar-looking characters from a different character set). Other common ploys to mislead users include adding a prefix or suffix to a name or using a different top-level domain (e.g. .biz instead of .com).

The same thing can happen to a developer who adds a dependency to software that they are building. If a software dependency is from a private repository, the developer should avoid namespace confusion by carefully specifying its location. Developers and users alike also should avoid obtaining software or components using links from an untrusted source. Following a known trustworthy link stored in a FAVORITES location or by typing the software name or URL manually and checking carefully before pressing ENTER are the best defenses.

[11] https://blog.sonatype.com/why-namespacing-matters-in-public-open-source-repositories
[12] https://www.upguard.com/blog/typosquatting

Intelligence Sources

The Internet offers many resources that provide evidence that indicates the activity, reputation, and trustworthiness of a software product, its development community, and other relevant information. The following are such resources:

- Search for reported security vulnerabilities: https://nvd.nist.gov/vuln/search?execution=e2s1
- Perform virus scan of files and URLs. Submit a hash of the file to see the results. Hash values can be computed using the "-h" option of SigCheck.[13] The DETAILS and RELATIONS tabs can provide useful information: https://www.virustotal.com/en
- Use code analysis statistics for some open-source projects. Look for "Defect Density" as a rough guide: https://scan.coverity.com/projects
- Find activity statistics for some open-source projects along with a "Nutshell" summary: https://www.openhub.net
- For projects hosted on GitHub, review activity and dependency info: https://github.com

[13] https://docs.microsoft.com/en-us/sysinternals/downloads/sigcheck

Chapter 2: Case Studies

Studying some past cyber security issues in the software supply chain helps to understand the following risks and illustrate how defensive principles can be applied:

Havex[14]

- An attacker compromised legitimate distribution points for proprietary industrial control system software, leading to compromise of the suppliers' customers when these versions were installed as updates.
- *Risk:* An attacker can compromise a distribution point for software.
- *Countermeasure*: Confirm authenticity of software by verifying its digital signature or by verifying the correct hash value if it is unsigned.

Heartbleed[15]

- Heartbleed (CVE-2014-0160) was a serious implementation flaw in OpenSSL of the Secure Socket Layer (SSL) "heartbeat" feature. It allowed a malformed heartbeat request to obtain replies that contained confidential data, such as usernames and passwords that had been sent protected by transport layer security. It was originally discovered on April 3, 2014, and a patch was announced by the OpenSSL project on April 7, 2014.
- Heartbleed is notable because it illustrates that even active and well-supported projects can have serious issues.
- *Risk:* Serious defects can and do arise even in projects with an attentive developer community.
- *Countermeasure:* Ensure that there is an understanding of where security announcements and advisories will be published and that there is a process to receive and review updates when they are released.

Colors.js[16]

- A developer deliberately corrupted two JavaScript libraries imported by thousands of users (colors.js and faker.js). It was apparently done in protest to doing "free work" to maintain open-source code.
- *Risk:* A trustworthy project may become a threat.
- *Countermeasure*: Test and inspect updates in an isolated environment before deploying them.

Log4j[17]

- Log4j is a logging component that is used for indicating a need for intervention by the operator or for recording information about events or exceptional conditions so they can be analyzed later. It is widely used because it provides many convenient features. Log4j was vulnerable to attacks where logging of particular data caused the logging subsystem to connect to external network (Internet) hosts. There were a number of related vulnerabilities: CVE-2021-44228, CVE-2021-45046, CVE-2021-45105, and CVE-2021-44832.
- *Risk:* Because Log4j is an application infrastructure component, it is "submerged" in most software and not readily apparent. When the vulnerability(s) was announced, most of those affected did not know it.

[14] https://ics-cert.us-cert.gov/advisories/ICSA-14-178-01
[15] https://www.us-cert.gov/ncas/alerts/TA14-098A http://xkcd.com/1354/
[16] https://nvd.nist.gov/vuln/detail/CVE-2021-23567 ;
(see also: https://www.theverge.com/2022/1/9/22874949/developer-corrupts-open-source-libraries-projects-affected)
[17] https://www.cisa.gov/uscert/apache-log4j-vulnerability-guidance

- *Countermeasure:* Apply risk-containment measures as discussed in the **Mitigating Risk** section. Some risks are inherent to the way software is developed and extremely difficult for users to detect. Understand that such risks exist and take steps to limit damage potential and detect malicious activity.

Chapter 3: Mitigating Risk

Because of the difficulty of inspecting software (especially in the case where a contractual relationship does not exist with the supplier), it may be necessary to mitigate the risks of operating it in a specific environment. The following are general risk reduction suggestions:

- Reduce attack surface[18]
  - Remove or disable non-essential features
  - Reduce privilege level (software should not be run with administrative privilege)
  - Apply patches and updates (when possible) in a reasonable timeframe after the supplier releases them
- Control access
  - Restrict access to users cognizant of the risks and trained in secure operation procedures
- Isolate
  - Prevent access to or from the Internet and tightly control access to the internal network by using network segmentation and restrictive firewall rules
  - Consider operating on a hardened system reserved for that application
- Monitor
  - Use and monitor a host-based firewall
  - Enhance monitoring effectiveness by using log analysis tools
  - Consider application whitelisting

Additional topics and guidance for supply chain security can be found at the NERC Supply Chain Risk Mitigation Program page.[19]

[18] https://en.wikipedia.org/wiki/Attack_surface
[19] https://www.nerc.com/pa/comp/Pages/Supply-Chain-Risk-Mitigation-Program.aspx

Contributors

NERC gratefully acknowledges the contributions and assistance of the following industry experts in the preparation of this guideline.

Name	Entity
George Masters, CISSP	Schweitzer Engineering Laboratories, Inc.
Tom Alrich	Tom Alrich LLC
Tony Eddleman	Nebraska Public Power District
Pierre Janse van Rensburg, GCIH	BBA Inc.
David Sopata	ReliabilityFirst Corporation

Guideline Information and Revision History

Guideline Information	
Category/Topic: Supply Chain	**Reliability Guideline/Security Guideline/Hybrid:** Security Guideline
Identification Number: SG-SCH-1222-1	**Subgroup:** Supply Chain Working Group (SCWG)

Revision History

Version	Comments	Approval Date
Original	Security Guideline for the Electricity Sector – Supply Chain: Risk Considerations for Open-Source Software, 4 Pages Approved by Critical Infrastructure Protection Committee	9/17/2019
V1.0	Draft, 5/4/2020 • Format paper to conform to template from NERC, bringing in pro-forma elements • Add a conclusion • Add log4j to case studies • Add discussion of malicious changes by a contributor • Add discussion of open-source software incorporated into a product • Add discussion of namespace / name confusion as risks • Make discussion of unsupported software a subtopic • Update mentions of BPS to BES, and correct acronym/abbreviations	
V1.1	Edit date on cover page, edit footer	
V1.2	Correct reference to Google Chromium and add concept of namespace confusion	
V2.0	Approved by the Reliability and Security Technical Committee	12/06/2022

Metrics

Pursuant to the Commission's Order on January 19, 2021, *North American Electric Reliability Corporation*, 174 FERC ¶ 61,030 (2021), reliability and security guidelines shall now include metrics to support evaluation during triennial review consistent with the RSTC Charter.

Baseline Metrics

All NERC reliability and security guidelines include the following baseline metrics:

- BPS performance prior to and after a reliability or security guideline as reflected in NERC's State of Reliability Report and Long Term Reliability Assessments (e.g., Long Term Reliability Assessment and seasonal assessments)
- Use and effectiveness of a reliability or security guideline as reported by industry via survey
- Industry assessment of the extent to which a reliability or security guideline is addressing risk as reported via survey

Specific Metrics

The RSTC or any of its subcommittees can modify and propose metrics specific to the guideline in order to measure and evaluate its effectiveness, listed as follows:

- The Supply Chain Working Group (SCWG) will ask users to respond to survey questions regarding their use of this security guideline. Survey responses will be analyzed to determine whether the guideline's recommendations are helping users implement measures that mitigate the risks associated with open source software.
- SCWG Security Guidelines will be reviewed, updated as needed and sent out for industry comments every three years. Comments will be reviewed and addressed prior to requesting RSTC approval.

Effectiveness Survey

On January 19, 2021, FERC accepted the NERC proposed approach for evaluating Reliability and Security Guidelines. This evaluation process takes place under the leadership of the RSTC and includes:

- industry survey on effectiveness of Reliability and Security Guidelines;
- triennial review with a recommendation to NERC on the effectiveness of a Reliability or Security Guideline and/or whether risks warrant additional measures; and
- NERC's determination whether additional action might be appropriate to address potential risks to reliability in light of the RSTC's recommendation and all other data within NERC's possession pertaining to the relevant issue.

NERC is asking entities who are users of Reliability and Security Guidelines to respond to the short survey provided in the link below.

Guideline Effectiveness Survey

Errata

N/A