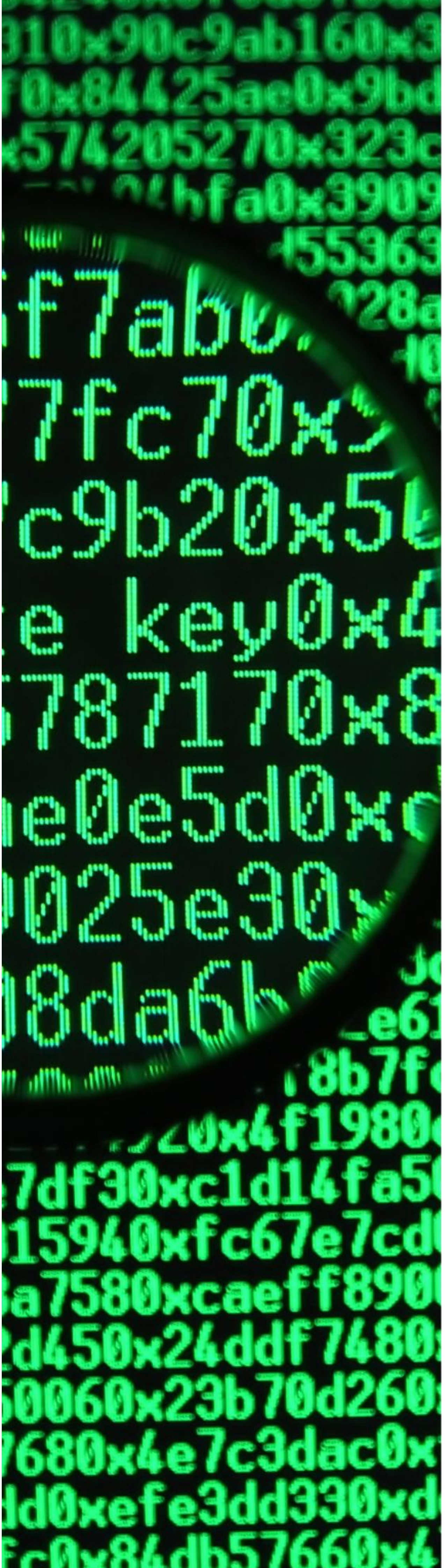

Software Bills of Materials for IoT and OT devices

An introduction to the use of SBOMs in the procurement and maintenance of connected devices

An IoTSF Whitepaper

Release: 1.1.0

Acknowledgements

This paper is an output of the IoTSF Supply Chain Working Group:

Editor and chair

Amyas Phillips, Ambotec Consulting

Working group members

Prof. Carsten Maple, University of Warwick
Florian Lukavsky, IoT Inspector
Ian Pearson, Microchip
Michael Richardson, Sandelman Software Works
Nigel Hanson, Trimble
Prof. Paul Kearney, Birmingham City University
Robert Dobson, Device Authority

Contributors

Adrian Tomasset, Doulos
Alex Suditu, Enevo Group
Allan Friedman, CISA DHS
Andy Culmer, IT Dev
Brendan Moran, Arm
Eliot Lear, CISCO
Isaac Dangana, Red Alert Labs
John Moor, IoTSF
Nick Allott, NquiringMinds
Raymond Wong, SecureThingz
Richard Marshall, Xitex
Shadi Razak, Angoka
Trevor Hall, Synaptics

About the IoT Security Foundation

The Internet of Things Security Foundation (IoTSF) is a not-for-profit, global membership organisation. Established in 2015, we are an international response to the complex challenges posed by cybersecurity in the expansive hyper-connected IoT world.

Working collaboratively, IoTSF is the natural destination for IoT technology producers and users which includes cybersecurity professionals, IoT hardware and software product vendors, network operators, system specifiers, integrators, distributors, retailers, insurers, local authorities, and government agencies.

Thank you for downloading this paper – we hope it is useful and helps you with your security journey. We also invite you to look at our website for more informative guidelines, reports, conference talks, blogs and more.

We would also like to invite you to join our growing membership base – whilst our publications are a good source of knowledge, being a member offers superior value for knowledge exchange and brand status.

See our website for more:

IoTSecurityFoundation.org

BUILD SECURE – BUY SECURE – BE SECURE

Table of Contents

1 Introduction

Regulators in many domains have begun to look seriously at software vendors' and operators' management of supply chain risks. Events have woken them up to the fact that modern software supply chains leave connected systems highly vulnerable to attack, so they are making new rules. Vendors and end users in these supply chains, including those for IoT/OT, need to act now or risk exclusion from markets and even liability for negligence [kitchen-liability] [ftc-log4j].

These regulations span consumer IoT (UK Product Security and Telecommunications Infrastructure Act, EU Cyber Resilience Act) to national critical infrastructure (EU NIS 2). Generating the most action so far is USA President Biden's Executive Order 14028, requiring suppliers of federal agencies to implement rigorous software supply chain risk management practices by September 2023 or face being replaced.

Software Bills of Materials (SBOMs) are mentioned explicitly in several of these regulations and as highly visible artefacts and enablers of many of the key processes they have attracted a lot of attention.

This paper aims to explain for **developers**, **buyers** and **operators** of IoT/OT why they should care about SBOMs and what they need to do about them.

It has been prepared by a working group of the IoT Security Foundation's Supply Chain Integrity Project, drawing on the experiences and insights of IoTSF members and contributors representing all parts of the IoT ecosystem. By documenting, advancing and sharing the current state of the art the IoTSF aims to advance IoT security, ultimately enabling wider deployment of this beneficial technology.

2 Background

2.1 Solarwinds and Log4Shell

In September 2019 hackers working for the Russian foreign intelligence service infiltrated the development environment of Solarwinds Corporation's Orion IT management software and inserted a back door of their own design. This backdoored software was distributed to some 18,000 Orion customers enabling the attackers to compromise dozens of high-value targets including national security organisations before the attack was detected and the back door removed. The incredible scale and success of this attack propelled supply chains to the top of cyber- and national-security agendas.

In November 2021 Chen Zhao Jun, an Alibaba cloud services security analyst, reported to the Apache Foundation a severe vulnerability in its extremely widely-used Java logging library Log4j. The vulnerability was dubbed Log4Shell (CVE-2021-44228) and precipitated a worldwide crisis. Despite fixes being published within days of Apache's public announcement Log4Shell was quickly and massively exploited. IT organisations found it next to impossible to determine for themselves whether their suppliers had used Log4j in their products, hampering mitigation efforts. Software vendors had to field a tidal wave of enquiries while looking for Log4j in their own products and rushing out versions incorporating successive fixes. In many cases they had to ask *their* suppliers and wait for fixes from upstream themselves. Meanwhile millions of organisations remained oblivious to the danger and did nothing.

Out of this chaos two lessons emerged: that accidental vulnerabilities can be just as damaging as supply chain attacks, and that modern extended software supply chains make it difficult to effectively manage exposure to both. It also threw up the uncomfortable possibility that had Chen Zhao Jun made his disclosure exclusively to his government's authorities, as required by a recently-enacted Chinese law, Log4Shell might have quietly been racked into an arsenal of cyber weapons.

These two incidents did more than anything else to bring software supply chains to regulators' attention.

2.2 Characteristics of modern software development

Modern software is rarely created from scratch but rather integrates a relatively small amount of new code with tens, hundreds or even thousands of pre-existing components, comprised of proprietary and open-source libraries and services. The irresistible attraction of re-using existing software to improve developer productivity has driven decades of constant innovation in abstracting, sharing and consuming software components.

The IoT revolution in particular has been enabled by the availability of highly-reusable embedded and cloud software platforms integrating solutions to multiple design challenges at low cost. Practically all connected embedded systems use such an "IoT OS", often in the form of reference code from the microcontroller vendor, plus additional imported code in the form of peripheral drivers, board support packages and application-specific libraries. This imported code can easily represent 90-99% of the codebase. Looking at this another way, the

IoT ecosystem has succeeded in taking an incredible 90-99% of costs out of developing for IoT/OT devices. This is incontrovertibly a huge success.

Importing software components is now so commonplace and easy that many projects don't give it a second thought, even though imported components routinely come with dozens of their own dependencies. Anything goes, as long as it helps advance the project.

As this approach has spread, three problems have emerged.

The first is that although an organisation may use cutting-edge quality techniques in its own projects, what its suppliers do is entirely up to them. Beyond their immediate suppliers they may not even know who is in their supply chain. By importing externally-developed components on the basis only of "does it work?" organisations give up control over their own software's quality and security [1].

The second is that keeping up to date with issues and security releases across a diverse collection of externally-maintained components is notoriously difficult. Software vendors need to assess and act on security notifications on externally-maintained components promptly because any one of them might expose customers to attack, and the longer it takes the more likely an issue is to be exploited in customer deployments. Few organisations currently succeed in doing this well, and even they are reliant on their suppliers to do the same.

The third is that when using externally-developed software supplied under multiple different copyright licences it is all too easy to lose track of their cumulative terms and become non-compliant with them. While it might not be considered as falling in the domain of cybersecurity, non-compliance does introduce a risk of disruption due to litigation by the copyright holders. It is an issue that has attracted the attention of corporate legal departments.

2.3 SBOMs to the rescue

SBOMs have recently received a lot of attention, so much so that the impression may be given that they are a silver bullet for supply chain risks.

The first thing to understand about SBOMs is that they do not solve these problems on their own. What they do is enable solutions. Cybersecurity and license management tools, processes and policies become more effective, with fewer blind spots, when applied systematically against a diligently-maintained list of software components. The goal should be to implement the solutions, not simply SBOMs.

The second thing to appreciate is that generating a list of software components to support these solutions is more of a challenge than it appears. A usable SBOM needs to identify components universally and unambiguously, supplemented with additional information such as dependency relationships, maintainers, and licences, all in a widely-shareable and

[1] In software, security is a subset of quality.

therefore standardised form. It needs to be as complete as possible across what may be many kinds of components, and kept up to date with the software it documents.

The third thing to know is that SBOMs are not new. Many maintainers of IoT/OT libraries and devices already use some kind of software inventory to solve exactly these problems, or simply to build their applications. Many more already have the ability to automatically generate SBOMs of reasonable quality, but aren't using it. What's new is the regulatorily-driven interest in levelling-up supply chain security, including by standardising and communicating SBOMs through supply chains, and by raising the bar for using them effectively.

In the following sections we will outline what should go into SBOMs and the main SBOM standards, how IoT/OT vendors should generate and share SBOMs, and how everyone in the IoT/OT supply chain should use SBOMs to effectively reduce cyber risks for IoT/OT operators.

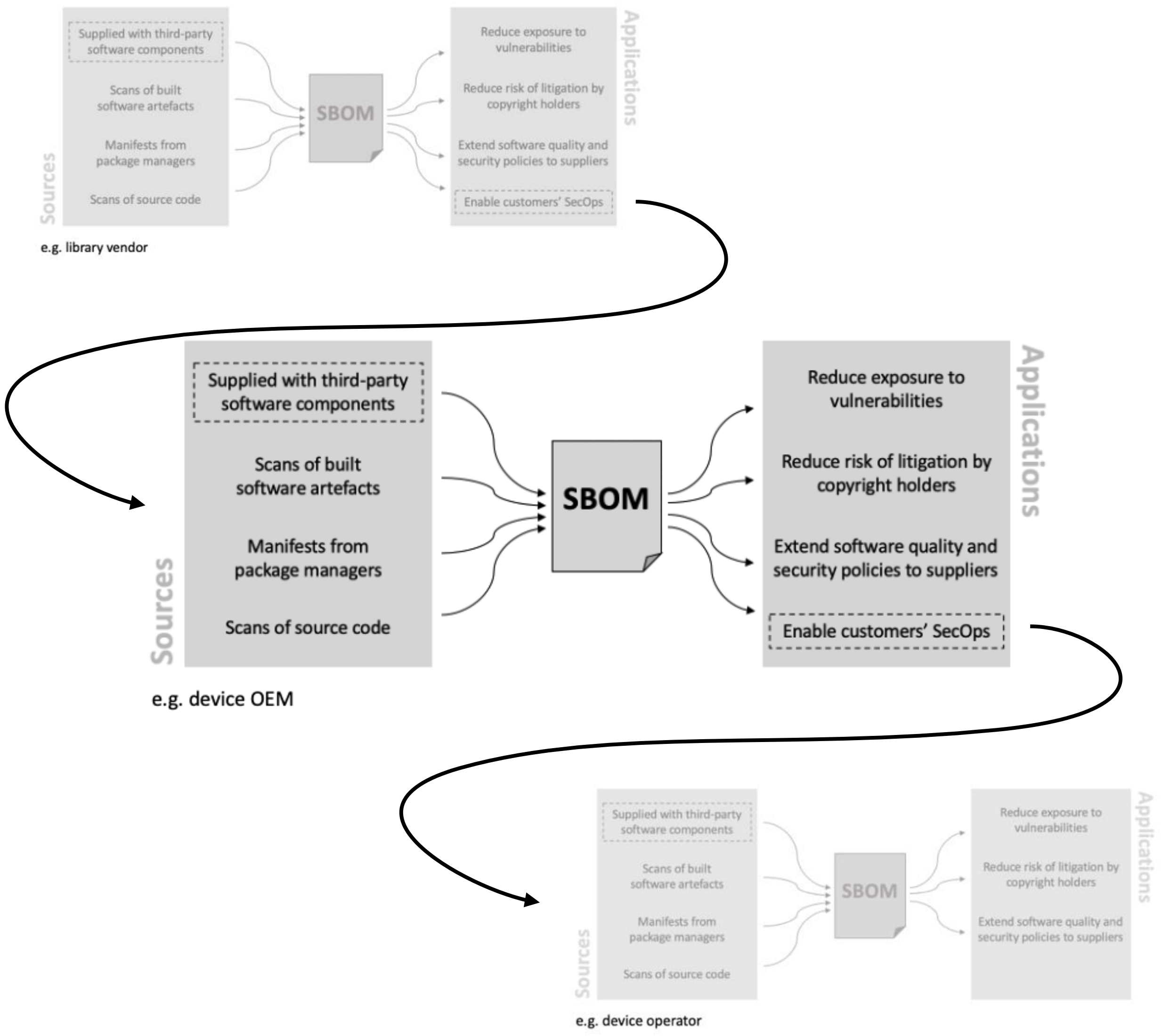
Sources
Supplied with third-party software components
Scans of built software artefacts
Manifests from package managers
Scans of source code
SBOM
Applications
Reduce exposure to vulnerabilities
Reduce risk of litigation by copyright holders
Extend software quality and security policies to suppliers
Enable customers' SecOps
e.g. library vendor
e.g. device OEM
e.g. device operator

Figure 1: Sharing SBOMs down the supply chain improves IoT/OT device security by i) enabling systematic review of eternally-developed components and ii) speeding up communication of vulnerabilities

3 Generating SBOMs

3.1 What's in an SBOM?

The United States' National Telecommunications and Information Administration (NTIA) has done seminal work on SBOMs. Defining the primary purpose of an SBOM as being to uniquely and unambiguously identify components and their relationships to one another [ntia-sbom-framing], it has determined that the minimum necessary data elements for this purpose are supplier, component name, version of the component, other unique identifiers, dependency relationship, author of SBOM data, and timestamp [ntia-sbom-minimum] [ntia-sbom-howto]. Additional data elements such as SPDX licence short ID and/or licence text may be populated as desired.

Because a main use case for SBOMs is to search vulnerability databases it is useful for an SBOM to include identifiers by which components are known in those databases. In the case of the United States' National Vulnerability Database (NVD) those are Common Platform Enumerations (CPEs) [nist-cpe]. Historically these have been assigned to components at the time a vulnerability is first reported in them, making it difficult to set up automatic searches before then. One solution is for library and device maintainers to register a CPE in advance [dangana-sboms]. However NVD is expected to deprecate CPE, possibly in favour of SoftWare IDentification tags (SWID) [iso-19770-1]. A belt and braces approach would be to locally generate a SWID and from that derive a CPE [nistir-8085], including both as unique identifiers in SBOMs shared with customers. The CPE name, SWID, or both can then be registered in the NVD or any other vulnerability database at need. However, the NTIA have taken the view that a global naming scheme is unrealistic and that it is probably sufficient for software and device maintainers to define and consistently use unambiguous supplier, component and version names, ideally supplemented with hashes, UUIDs or other unique identifiers.

If the original supplier of a component has not provided their preferred component names in an SBOM, downstream integrators are free to make an educated guess, noting the fact using the Author of SBOM Data element.

3.2 Automating SBOMs in CI/CD pipelines

A new SBOM must be generated for each successive software release, reflecting the incremented version number and any different or updated imported components. In a modern continuous integration / continuous deployment (CI/CD) pipeline this can and should [2] be automated. There are four potential sources of information, each with its own tools and advantages.

The go-to approach is to list dependencies from the relevant **package managers**. These might include language-specific source tools such as Python's pip, Java's Maven, JavaScript's npm and Rust's Cargo. In embedded C/C++ environments package management is not all that common, however many projects do use build systems such as Cmake or utilities such as the Zephyr project's West that can be leveraged to generate an SBOM. In virtual machines and

[2] EO 14028 explicitly requires automation, to ensure that SBOMs remain current.

containers application-level package managers such as apt, apk, dpkg, and opkg can be used to list installed modules and applications. Projects using containers can also look in the layer definitions or use tools like syft or Docker's sbom command. Yocto projects can have bitbake output a recipe-derived SBOM during build. In fact, SBOM tooling is available in practically all environments. There are even tools such as Microsoft's sbom-tool which integrate scanners for multiple package managers and build systems. Any combination of these can be invoked as part of an automated CI/CD pipeline.

The limitation of this approach is that visibility is limited to software components that have been added via package managers, which in any case may not be available to downstream users of the software.

Source code is generally considered to be a very reliable and easy source from which to generate an SBOM. Dependencies, linked libraries and embedded code can all be identified efficiently using a mixture of annotation [3] and scripting or commercial source code analysis (SCA) tools [4], which can additionally detect copy-pasted snippets. These methods tend to require a significant initial set-up effort followed by steady maintenance to prune false positives and add required information, but they can be highly effective and are easily automated in CI/CD pipelines.

The limitation of source code scanning for IoT/OT projects is that even device OEMs may not have access to the source code for all imported components, for example pre-built proprietary SDK components. Thus, this approach alone also often lacks completeness.

Compiled binaries are often the only resource available to downstream participants in the supply-chain, including device operators. Equipped with a database of hashes of known binary objects it may be possible to identify a particular binary. This can provide some insight but it reveals nothing about components included in the binary. Reverse engineering a binary is not impossible, but it is difficult, not automatable and unlikely to fully succeed. For users of pre-programmed embedded devices or modules even binaries may not be available.

SBOM documentation, generated from original metadata, sources and upstream SBOM documentation by each participant in a device's supply chain and shared downstream may be expected to provide the most complete and correct device SBOM. It would overcome the limitations that today make complete visibility of software content unachievable from any single position in a device's supply-chain. In practice this is still an emerging norm and few projects can expect to receive SBOMs with all of their imported components. Suppliers may be willing to rectify this but, in the meantime, developers must document their own code and fill in the details for imported components as best they can.

Automating generation of an SBOM from these sources and its sharing downstream unquestionably requires some investment of time and effort. This has fallen considerably with the wide availability of tooling and, equally importantly, mostly scales with the project. A

[3] For example, addition of SPDX licence short identifiers in the root folders of imported components

[4] Historically the focus of these tools has been to detect open-source software (OSS) components in source repositories by comparing code 'fingerprints' against proprietary databases of OSS software and other public codebases. Many now offer additional features such as detection of known vulnerabilities.

small IoT library developer might be able to create an SBOM by adding a command in a single package manager to their release script, and perhaps a second to sign the output. A complex and mature embedded project might require the one-time investment of several engineer-months to do the necessary detective work, pull together the various sources of information, transform it into a standard format and automate all of it [5]. Given the strong positive benefits either effort is likely easy to justify.

3.3 Sharing SBOMs

Sharing SBOMs downstream can be accomplished in many ways [ntia-sbom-sharing]. It should be automated, to keep downstream users up to date, and ideally it should be done consistently in a small number of expected ways, to simplify consumption. Table 1 lists recommended methods for different types of components.

Component type	Recommended methods of providing SBOM documentation
Source code libraries	SBOM is included in top-level directory
Binaries (for linking into downstream projects)	SBOM is included in an archive with the binary
Packaged binaries (to be run in OS environments)	SBOM is included in the package SBOM is posted as a web resource and its URL included in the package metadata
Device images (for installation on IoT/OT devices by operators)	SBOM is included in an archive with the device image
Device images (installed by manufacturer during production or via remote update mechanism)	SBOM is posted as a web resource AND Where devices connect to a central management service: Devices report to their central management service the URL of the SBOM Devices report to their central management service their software version number. Manufacturer publishes a web page or resource listing SBOM URLs against software version numbers for each model of device [6]. Where no central management service is used: Devices serve the SBOM URL at .well-known/sbom [rfc-8615] [draft-ietf-opsawg-sbom-access-13] Devices serve the SBOM via an extended Manufacturer Usage Description (MUD) [rfc-8520] [draft-lear-opsawg-mud-sbom-00]

[5] This doesn't include fixing any issues that might be exposed by the systematic vulnerability and licence compliance checking that should be implemented to take advantage of the SBOM. Established projects can expect to find several such. Fixing them might feel like an overhead but should be seen as one of the benefits.

[6] SBOMs for devices are associated with firmware versions, not with devices directly.

Component type	Recommended methods of providing SBOM documentation
Device images (of fully-managed devices)	Not necessary. Consuming SBOMs is a task for the operator of the managed devices.

Table 1: Recommended methods for providing SBOM documentation downstream

A question that often arises is whether SBOMs, particularly for binaries and device images, should be considered security-sensitive. Could access to a device SBOM help an attacker generate a list of imported vulnerabilities? Yes. Might that help them find one that is exploitable? Possibly. Could access to a device SBOM help an attacker conceive a supply chain attack that they wouldn't have figured out otherwise? Possibly. However, the benefits to defenders are much greater. It is often said that whereas attackers only need to find one vulnerability, defenders have to find them all, and the fact is that attackers don't need an SBOM to find one vulnerability. Far more likely, they already know about one and are looking for targets where it is unpatched. For defenders, on the other hand, using an SBOM is the acme of systematic defence against software supply chain risks. Although some organisations have decided to distribute SBOMs only to their customers, even they do not regard disclosure as a disaster and don't attempt to further secure them [7].

3.4 Standards

SBOM standards are necessary to facilitate sharing and to encourage the development of interoperable tools and services. Adopting one of these standards makes it easier and cheaper to generate, use and share SBOMs. The NTIA has published a useful review of current standards [ntia-standards-survey].

The leading standards are CycloneDX, originating in the OWASP information security community, and the Software Product Data Exchange (SPDX), originating in the OSS community. Although SWID tags have also been in the frame their core use case is as unique, unambiguous identifiers for software components.

The original use case of SPDX is licence compliance. It is maintained by a Linux Foundation working group. The first draft appeared in 2010 and the current specification is a freely-available ISO/IEC standard, ISO/IEC 5962:2021 [iso-5962]. Further development can be followed via the group's public GitHub repository.

The original use cases of CycloneDX were vulnerability identification, license compliance, and outdated component analysis. It has an open governance model. The first draft specification appeared in 2017 and the standard is now on version 1.4.

[7] What should be considered confidential is Information about exploitable vulnerabilities. See 4.2 Reducing exposure to vulnerabilities, below

Both standards have been matured through extensive use, have active communities and are under continuing development. Both aspire to support all SBOM use cases. Both are widely supported and maintain lists of compatible tools. Many tools support both. Neither offers special advantages for embedded projects.

3.5 Completeness, and the lack of it

The more accurate, detailed and complete an SBOM, the more useful it is. However, it has to be said that not only are SBOMs rarely any of those things, it can be very difficult to measure their quality objectively.

An internal SBOM generation effort may be able to deliver some metrics, for example fraction of code artefacts attributed, number of identified components for which no licence or an incompatible licence is detected, but these can overlook major sets of components, “unknown unknowns” that nobody’s thought of yet.

Even non-embedded software projects face challenges identifying and recording software components of different types and layers, for example binaries, source, package-managed, non package-managed, containerised, static and dynamic libraries.

IoT devices have all these challenges plus the whole stack from bare metal boot and quite likely additional deeply-embedded firmware in NICs, secure elements and other ICs. A deliberate and careful inventory of opaque and ‘hidden’ software components is the only way of making these into “known unknowns”.

Where an upstream supplier provides an SBOM downstream, those consuming it have no easy way of assuring themselves that it meets their standards. No standard or certification scheme exists, at least not yet. Some generate and analyse SBOMs for themselves from supplied code and built artefacts, using their own SCA tools. This can certainly encourage suppliers to step up to the mark but users are still reliant on suppliers for interpretation. For most, this amount of effort will not be justifiable.

Ultimately it is worth remembering that even a partial SBOM is better than no SBOM. Although use of SBOMs can greatly reduce exposure to compliance issues and known vulnerabilities, they may never be eliminated completely.

4 Using SBOMs

4.1 Complying with OSS licences

Complex but essentially standard functions can easily make up 90% of an IoT device's embedded software. The cost of developing and maintaining such a stack is far beyond what most embedded products can support, so OEMs acquire it from library and OS vendors and/or OSS projects. Nearly all IoT devices therefore contain a large amount of third-party software, the copyright in which remains with its creators. Users and operators of IoT devices have no rights to use it except those granted by the copyright holders via licence agreements. Licences can stipulate any terms the copyright holder chooses, although to simplify compliance most open-source software projects employ one of a handful of standard forms.

Non-compliance with licence terms exposes users and operators of IoT devices to the risk of litigation by copyright holders. Most OSS-related cases are settled amicably but where commercial software is involved, where there is a breakdown of goodwill, or where there is a copyright trolling situation [ipkat-trolls] there could be significant financial consequences, operational disruption, or both.

Alerted by a handful of high-profile litigations, the driving force behind SBOMs at many organisations has not been the CTO or CISO but the General Counsel, seeking to reduce licence compliance risk. **Operators** started asking suppliers to show their products are in full licence compliance during procurement. **Suppliers** partially achieved this by providing an SBOM listing software components and licences. Operators have no guarantee that such an SBOM is complete and accurate, so they may also require indemnification against copyright claims. Indemnification is not a concept any General Counsel is comfortable with, but over time legal teams at software vendors have developed an accommodation with engineering teams in which, provided the inventory of OSS software is complete and well maintained, and that engineers know to bring unusual licences for review, they can be comfortable with any amount of OSS components.

IoT/OT vendors who need an OSS licence compliance programme of their own should look at implementing the OpenChain standard, ISO/IEC 5230 [iso-5230], which has been developed as a service to the community by pioneers of such programmes. Like SPDX it is a freely available ISO standard. It is straightforward and proven to work for engineering departments, corporate legal teams and the OSS community alike.

4.2 Reducing exposure to vulnerabilities

IoT/OT is under increasing attack, and unpatched software vulnerabilities are certainly a leading risk. In the wider realm of IT, unpatched vulnerabilities are reckoned to be responsible for one in three information security breaches [verizon-2019] [ibm-2022]. This is why closing the window between vulnerabilities becoming known and mitigations being put in place has become a key goal for the entire IoT/OT supply chain.

Although tools and accepted practices are emerging, implementing an effective vulnerability management programme is still considered a challenge. Although nearly all vulnerabilities are announced alongside patches or other mitigations, estimates of the mean time IT organisations take to patch them range from 60 to 200 days. While lack of situational awareness is a major contributor, allowing vulnerabilities to be overlooked, better situational awareness often throws up so many vulnerabilities that SecOps teams are overwhelmed.

Nevertheless, **IoT/OT suppliers** committed to securing their products will not like to be caught out by something as simple as an unpatched vulnerability. Table 1 describes the capabilities they need to put in place.

Step	Description	Notes
1	Build an SBOM for own project	Implement the approaches described in 3.2 Automating SBOMs in CI/CD pipelines to generate and automatically update the SBOM. Share the SBOM downstream with every software release, using the approaches in Table 1.
2	Search public vulnerability databases	Resolve components in vulnerability databases and regularly query them for vulnerabilities in those components. As well as the NVD these databases include OSV, Sonatype OSS Index and VulDB. Some SCA tool vendors also maintain their own vulnerability databases. Each has its own specialities. Tools such as Grype, which accepts SPDX and CycloneDX SBOMs as input, and Yocto's cve-check can help with this. In the IoT/OT domain it is not unusual for security advisories on devices or SDKs to be notified via an email list, a web page, even the release notes for a patch. Although a third party might capture these into a vulnerability database, this cannot be relied upon. Suppliers should therefore be asked, or required, to make vulnerability disclosures via a specific database. If that is not possible then instead of a vulnerability database it will be necessary to monitor whatever channels they do use. One way or another all immediate suppliers should be monitored at least daily, and ideally all suppliers. When a vulnerability is found it should be stored for subsequent action, e.g., in a ticketing system.
3	Filter and triage vulnerabilities	A competent group should assess the exploitability of each imported one to filter out the false positives. Anecdotally, it is common to find that 95% of vulnerabilities in included components are not exploitable. A justification should be recorded for each ignored vulnerability. For the remaining vulnerabilities their severity should be assessed [8] and mitigating actions suggested for downstream users, to protect themselves with until a patch can be released. The list of current vulnerabilities and their statuses should be maintained automatically for each currently supported software release, being updated each time a new vulnerability is detected or a vulnerability's status changes. Ideally, upstream suppliers of imported components should provide their own assessments of vulnerabilities present in their components. Symmetrically, developers of IoT/OT devices and software components should share their vulnerability status lists automatically

[8] Using the Common Vulnerability Scoring System (CVSS) [first-cvss], for example.

Step	Description	Notes
		to downstream users, saving them time and reducing support calls. It is advisable to limit distribution of these lists to customers, where possible.
4	Deploy fixes	Fixes should be implemented, or incorporated, in priority order and released in new software versions. To speed up the propagation of patches through the supply chain it is important to assess and incorporate new component releases promptly. In development environments it is often possible to ensure this by generating alerts from package managers, or by going a step further and having CI/CD pipelines automatically pull in the latest component versions on each build.

Table 2: Four steps for IoT/OT suppliers implementing a vulnerability management programme. This may form part of a wider as product security incident response (PSIRT) capability.

For the purposes of sharing assessments of vulnerability exploitability downstream a standard format, Vulnerability Exploitability eXchange (VEX), is being developed by the NTIA [ntia-vex]. VEX is currently available in two formats: a profile of the Common Security Advisory Framework (CSAF) [oasis-csaf], a standard for machine readable security advisories developed by OASIS Open, and CycloneDX. SPDX is expected to add support in its upcoming 3.0 release.

Being cognizant of their own responsibility to disclose vulnerabilities, developers of IoT/OT devices and software components should themselves implement a vulnerability disclosure programme and make disclosures via a public vulnerability database. The IoTSF's Vulnerability Disclosure Best Practice Guidelines [iotsf-vulns] describe a proven approach.

Operators of IoT/OT don't need to consider anyone further downstream. For them the first step in implementing rapid SBOM-based vulnerability management is to have IoT/OT device vendors supply up-to-date, complete and accurate SBOMs with every software update, including the identifiers that they will use when notifying device vulnerabilities to public vulnerability databases. Ideally, they should also commit to supplying VEX information.

The next step is to build an awareness of what is running where, automatically pulling device IDs and software versions from device management applications and retrieving SBOMs from wherever the relevant suppliers have made them available. This information needs to be stored in a vulnerability management system of some kind.

Checking for known vulnerabilities, followed by VEX-aided filtering and triage, can then proceed as described in Table 2 and the exploitable vulnerabilities addressed in order of severity, deploying mitigations until patched software is available.

Again, to close the window of vulnerability as much as possible it is important to assess and deploy new software releases promptly. Depending on how device updates are being managed it may be possible to configure alerts from device management platforms. If new versions are notified by email or web page announcements it will be necessary to monitor them. If devices are designed to update themselves automatically then SecOps teams can sit back and watch the vulnerability management system track progress.

4.3 Securing software supply chains

Many approaches to managing supply chain risks require that suppliers demonstrate compliance with specified policies. For example, compliance with rules of origin requires information about contributors and maintainers; compliance with secure software development standards requires information about the SDLC practices of suppliers; compliance with policies about maintenance commitments requires information about suppliers' maintenance horizon and response time commitments.

Ideally, software suppliers would self-report in their SBOMs their own performance on widely-accepted supply chain security compliance axes using widely-accepted metrics, optionally certified by independent third parties. For now, anyone attempting to implement such a supply chain risk management programme must do their own research and map the results into their own metrics, using an SBOM as a simple to-do list.

IoT/OT software suppliers aim to produce high quality software because that keeps customers happy, increases future sales, reduces support costs and frees up time for new product development. Outsourcing common problems to upstream developers is seen as a free ticket to higher quality because they can give the problem their dedicated attention, and lower costs because their solution serves a wider customer base. Unfortunately, although cost is easily visible, quality is not, so IoT/OT software suppliers are challenged to hold upstream developers consistently to any standard of quality and security. In fact, many do not have any formal requirements [Veracode-2021].

For IoT/OT software suppliers a good software supply chain risk management programme starts at home, by picking a set of quality and security practices covering the areas listed in Table 3, below:

Area	Aim	Suggested guidelines / standards
Code quality enhancement and secure-by-design practices	Reduce the incidence of vulnerabilities	NIST's Secure Software Development Framework (SSDF) [nist-sp800-161] should work for just about everyone. IoT device vendors should supplement this with a set of best practices for securing devices though their development, production and deployment, such as the IoTSF's Security Assurance Framework [iotsf-saf].
Security of the software development and production environment	Reduce the risk of sabotage	
Vulnerability reporting and disclosure	To establish channels of communication around vulnerabilities	IoTSF Vulnerability Disclosure Best Practice Guidelines [iotsf-vulns] The CERT Guide to Coordinated Vulnerability Disclosure [cert-vulns]

Area	Aim	Suggested guidelines / standards
		UK National Cyber Security Centre Vulnerability Disclosure Toolkit [ncsc-vulns] In addition to these guidelines, this area should include a commitment to publishing vulnerabilities to a public database.
Vulnerability monitoring and remediation	To close the window of exposure	See 4.2 Reducing exposure to vulnerabilities, above.
Security maintenance	Avoid reliance on unmaintained components	These areas covering reliability of supply do not yet benefit from community-developed best practice guidelines.
Supplier continuity risk management	Ensure that someone will be around in future to maintain each externally-developed component	

Table 3: Six areas to consider when implementing a supply chain risk management programme

After implementing these policies internally, ways should be found of bringing imported components up to the same level. Full compliance is unlikely to be achieved quickly. An SBOM can be used as to-do list, against which components' compliance plan and progress can be recorded.

For small OSS components it is advisable to approach compliance as a collaboration. For example, an OSS utility library with a single developer is unlikely to adhere to a specific secure SDLC, but it likely does implement multiple secure SDLC practices and it is still useful to downstream software developers. One option they have is to fork it and bring it up to compliance as an integral part of their product, instead of as an imported component. Alternatively, the original developer may be willing to accept contributions upstream, where they may benefit other users and possibly receive beneficial contributions in turn.

Informal OSS components will also struggle to give maintenance commitments. Instead of asking an informal project for a formal contract it may be sufficient to extrapolate from recent maintenance activity and track record of handling defects. Alternatively, the developers may be willing to take on a maintenance contract [9]. The option always also exists to fork the component and maintain it in-house.

[9] In fact, they may very much appreciate it.

Finding an accommodation between the world of OSS and software supply chain security is an emerging area of practice in which many practical details still need to be worked out.

IoT/OT device OEMs implementing supply chain security programmes need to consider device design, manufacturing and provisioning processes as well as the software supply chain (Figure 2). For a guide to the risks and best practices see the IoTSF's whitepaper Securing the Internet of Things Supply Chain [iotsf-supply-chain].

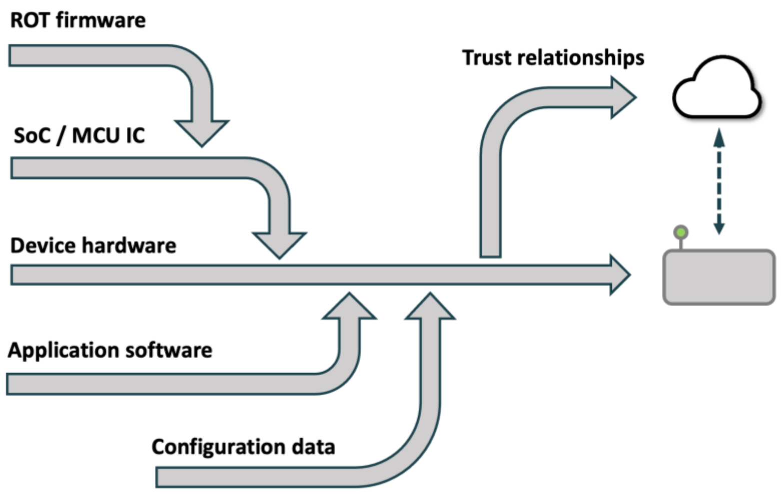

Figure 2: Major branches of a typical IoT/OT device supply chain

IoT/OT operators on their part need to be aware that using suppliers who are unable or unwilling to invest in quality and security exposes them to more vulnerabilities, both accidental and deliberately inserted by attackers, and those vulnerabilities will take longer to fix. The more critical the applications in which they are deploying IoT/OT devices, the more urgently they need to evaluate supply chain security as part of their procurement process.

They can start by asking their OEM suppliers what their supply chain looks like, including (via an SBOM) their software supply chain, and how the integrity and quality of assets in it are assured. Unfortunately, no comprehensive standard exists to which IoT/OT OEMs can certify themselves, so it is up to operators to decide what constitutes an acceptable answer. At least the six areas in Table 3 should be addressed, plus the resilience to attack of deployed devices, the integrity of software assets in production environments, the confidentiality of device secrets, and controls over the signing of devices into certificate chains of trust.

Focusing back on the software supply chain, security requirements, like other requirements, are best expressed in terms of measurable outcomes. Although the most appropriate measure would be the number of exploitable vulnerabilities that is, unfortunately, unknowable. Instead, proxy measures must be used, such as code quality metrics, historical mean time to response, and compliance with a specified set of product and information security best practices. IoT/OT operators should pick a set, or as the US federal government has done with EO 14028, develop their own.

5 Next steps for the IoT community

IoT users increasingly recognise that the security of their operations depends not just on their own security posture but on that of their whole supply chain. Identifying and mitigating supply chain cyber risks is becoming a fundamental requirement for IoT security practitioners. If IoT suppliers haven't already begun to address this in their procurement, production and SecOps practices they will soon find customers asking "Why not?"

Improving supply chain security requires an understanding of what you've got and where it has come from. Standardised sharable SBOMs are a major advance towards that, bringing transparency to what are otherwise opaque software supply chains. Used effectively, an SBOM can significantly reduce exposures to known software vulnerabilities. It also grants its holder insights into supply chain risks, allowing action to be taken to mitigate them.

Vulnerabilities are inevitable. Secure IoT/OT software requires effort not only to minimise vulnerabilities, but also to handle them well. Every organisation developing and shipping software in the IoT supply chain should take steps to make SBOMs available and to use that information to reduce their exposure to vulnerabilities in components from upstream. They should also take steps to adopt best practices in vulnerability reporting, without which SBOMs' downstream usefulness in vulnerability management is diminished.

IoT supply chains are complex constructions that produce and integrate hardware, software and trust. Software is necessarily a huge part of all of them because IoT device economics depend on wide reuse of expensively-developed common technology, much of which consists of shared libraries and software platforms. Anyone looking to assure and maintain the quality of their IoT software likely needs to examine what their suppliers are doing as much as what they are doing themselves. An SBOM is a necessary first step towards this, documenting suppliers through the dependency tree. They can then be measured against policies designed to reduce the risk of attackers using them to launch cyberattacks, although actioning this information is going to be easier for new projects than established ones

Maintaining and using an SBOM even just for vulnerability management delivers real and significant improvements in cybersecurity. Sharing a standards-compliant SBOM downstream, as well as helping your users' implementation of supply chain security policies and vulnerability management operations, is a visible indicator of your commitment to supply chain security.

Set these security and productivity benefits against well-understood and easily sustainable costs. Many IoT developers already maintain SBOMs for their own use, even if only for licence compliance. For those that don't, the approaches for doing so are now established and uncontroversial. Emerging standards have reduced costs by driving development of a growing ecosystem of SBOM tooling, easily automated in CI/CD pipelines.

IOT software suppliers have a responsibility to their users to deliver quality, secure software. Users have a responsibility to themselves to choose their suppliers wisely, and to keep up to date with security advisories and patches. There are many suppliers and users in the IoT software supply chain and nothing should be stopping any of them from adopting SBOMs to help them shoulder these responsibilities better. The benefits are clear and present a strong

ROI. The technology is sufficiently mature and still improving. There is no need to wait for someone else to take the first step, or to let worries about reaching full coverage stand in the way of partial coverage because SBOMs are not an all or nothing proposition: they deliver progressively more benefits as their coverage increases, which can be expected as the practice of shipping SBOMs with products moves from rarity to commonplace.

Continuing dialogue between IoT/OT OEMs, software vendors, OSS projects, operators and regulators in forums such as the IoTSF will enable the community to rapidly build on and improve these best practices.

References

In order of appearance:

[kitchen-liability] *The Evolution of Legal Risks Pertaining to Patch Management and Vulnerability Management*; Kitchen et al, Vol. 59 No.2 Article 6 Duquesne Law Review (2021); Available at: https://dsc.duq.edu/dlr/vol59/iss2/6

[ftc-log4j] *FTC warns companies to remediate Log4j security vulnerability*; FTC (2022-01-04); accessed at https://www.ftc.gov/policy/advocacy-research/tech-at-ftc/2022/01/ftc-warns-companies-remediate-log4j-security-vulnerability, warns companies that "The FTC intends to use its full legal authority to pursue companies that fail to take reasonable steps to protect consumer data from exposure as a result of Log4j, or similar known vulnerabilities in the future."

[ntia-sbom-framing] *Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM)*; NTIA Multistakeholder Process on Software Component Transparency Framing Working Group (2019); accessed at https://www.ntia.gov/files/ntia/publications/framingsbom_20191112.pdf

[ntia-sbom-minimum] *The Minimum Elements For a Software Bill of Materials (SBOM)*; United States Department of Commerce (2021); accessed at https://www.ntia.doc.gov/files/ntia/publications/sbom_minimum_elements_report.pdf

[ntia-sbom-howto] *How-To Guide for SBOM Generation*; NTIA Software Transparency Healthcare POC (2021); accessed at https://www.ntia.gov/files/ntia/publications/howto_guide_for_sbom_generation_v1.pdf

[spdx-list] The SPDX Project maintains a list of open source software licenses and has assigned them unique short identifiers. The list can be found at https://spdx.org/licenses/

[nist-cpe] NIST maintain the CPE specifications as part of the Security Content Automation Protocol (SCAP). See overview and specifications at https://csrc.nist.gov/projects/security-content-automation-protocol/specifications/cpe

[dangana-sboms] *Using SBOMs to Secure Industrial IoT Devices*; Dangana and Alrich, Red Alert Labs; Industry IoT Consortium Journal of Innovation (2022-07-27); accessed at https://www.iiconsortium.org/wp-content/uploads/sites/2/2022/07/6-JOI_20220727_Using_SBOMs_to_Secure_Industrial_IoT_Devices_Standalone.pdf

[owasp-vuln-naming] OWASP maintains a useful guide to software artefact naming schemes for use in vulnerability management. It can be found in OWASP's Web Security Testing Guide at https://owasp.org/www-project-web-security-testing-guide/latest/5-Reporting/02-Naming_Schemes

[iso-19770-2] *ISO/IEC 19770-2:2015 Information technology — IT asset management — Part 2: Software identification tag*; ISO/IEC (2015); available for a fee at https://www.iso.org/standard/65666.html

[nistir-8085] The procedure is defined in NISTIR 8085 Forming Common Platform Enumeration (CPE) Names from Software Identification (SWID) Tags https://csrc.nist.gov/publications/detail/nistir/8085/draft

[ntia-sbom-sharing] *Sharing and Exchanging SBOMs*; NTIA Multistakeholder Process on Software Component Transparency Framing Working Group (2021); accessed at https://www.ntia.gov/files/ntia/publications/ntia_sbom_sharing_exchanging_sboms-10feb2021.pdf

[rfc-8615] *Well-Known Uniform Resource Identifiers (URIs)*; Nottingham, IETF Operations and Management Area Working Group (2019); accessed at https://datatracker.ietf.org/doc/rfc8615/

[draft-ietf-opsawg-sbom-access-13] *Discovering and Retrieving Software Transparency and Vulnerability Information*; Lear et al, IETF Operations and Management Area Working Group (2023); accessed at https://datatracker.ietf.org/doc/draft-ietf-opsawg-sbom-access/

[rfc-8520] *Manufacturer Usage Description Specification*; Lear et al, IETF Operations and Management Area Working Group (2020); accessed at https://datatracker.ietf.org/doc/rfc8520/

[draft-lear-opsawg-mud-sbom-00] *SBOM Extension for MUD*; Lear et al, IETF Operations and Management Area Working Group (2020); accessed at https://datatracker.ietf.org/doc/draft-lear-opsawg-mud-sbom/

[ntia-standards-survey] *Survey of Existing SBOM Formats and Standards*; NTIA Software Transparency Working Group on Standards and Formats (2021); accessed at https://www.ntia.gov/files/ntia/publications/sbom_formats_survey-version-2021.pdf

[iso-5962] *ISO/IEC 5962:2021, Information technology — SPDX Specification V2.2.1*; ISO/IEC (2021); available free of charge at https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[ipkat-trolls] For an example see *Copyright Trolling: Abusive Litigation Based on a GPL Compliance*; Giedrimaite, The IPKat (2019); accessed at https://ipkitten.blogspot.com/2019/02/copyright-trolling-abusive-litigation.html

[iso-5230] *ISO/IEC 5230:2020 Information technology — OpenChain Specification*; ISO/IEC (2020); available free of charge at https://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[verizon-2019] *Data Breach Investigations Report*; Verizon Enterprise (2019); accessed at https://www.verizon.com/business/resources/reports/2019-data-breach-investigations-report.pdf

[ibm-2022] *X-Force Threat Intelligence Index*; IBM Security (2022); accessed at https://www.ibm.com/reports/threat-intelligence/

[first-cvss] *Common Vulnerability Scoring System v3.1*; Forum of Incident Response and Security Teams (FIRST) (2019). Documentation and training resources are available at https://www.first.org/cvss/

[ntia-vex] *Vulnerability-Exploitability eXchange (VEX) – An Overview*; NTIA (2021); accessed at https://www.ntia.gov/files/ntia/publications/vex_one-page_summary.pdf

[oasis-csaf] *Common Security Advisory Framework Version 2.0*; OASIS Open (2012); see section 4.5 Profile 5: VEX directly at https://docs.oasis-open.org/csaf/csaf/v2.0/csd01/csaf-v2.0-csd01.html#45-profile-5-vex

[iotsf-vulns] *Vulnerability Disclosure Best Practice Guidelines, Release 2.0*; IoTSF (2021); available at https://www.iotsecurityfoundation.org/wp-content/uploads/2021/09/IoTSF-Vulnerability-Disclosure-Best-Practice-Guidelines-Release-2.0.pdf

[cert-vulns] The CERT Guide to Coordinated Vulnerability Disclosure; Householder et al, Software Engineering Institute, Carnegie Mellon University (2017); accessed at https://resources.sei.cmu.edu/asset_files/SpecialReport/2017_003_001_503340.pdf

[ncsc-vulns] Vulnerability Disclosure Toolkit; UK National Cyber Security Centre (2020); accessed at https://www.ncsc.gov.uk/files/NCSC-Vulnerability-disclosure-Toolkit-v2.pdf

[Veracode-2021] *State of Software Security v11: Open Source Edition*; Veracode (2021); accessed at https://info.veracode.com/fy22-state-of-software-security-v11-open-source-edition.html?utm_source=press-release&utm_medium=pr&utm_campaign=VER012T000001JCLvAAO&utm_term=press-release&utm_content=state-of-software-security-v11-open-source-edition. This 2021 survey found that only half of companies had any kind of OSS acquisition policy, and that those primarily concerned allowable licenses.

[nist-sp800-161] *SP 800-161 Rev. 1 Cybersecurity Supply Chain Risk Management Practices for Systems and Organizations*; NIST (2022); available at https://csrc.nist.gov/publications/detail/sp/800-161/rev-1/final

[iotsf-saf] *IoT Security Assurance Framework, Release 3.0*; IoTSF (2021); https://www.iotsecurityfoundation.org/wp-content/uploads/2021/11/IoTSF-IoT-Security-Assurance-Framework-Release-3.0-Nov-2021-1.pdf

[iotsf-supply-chain] *Securing the Internet of things Supply Chain*; IoTSF (2022); available at https://www.iotsecurityfoundation.org/iot-supply-chain-cybersecurity-guidance/

Document End

IoTSF-SBOM V1.1.0 (2023-02-20)

An introduction to the use of SBOMs in the procurement and maintenance of connected devices