Software Supply Chain Risk Management

Problem & Opportunity

Lack of visibility and positive control of risks in third party software, including vendor products and contractor deliverables. Code developed outside enterprise boundaries is subject to opaque security criteria, and there are dangerous discontinuities between the emergence of risk in the software supply chain, the customer's awareness of those vulnerabilities and supplier provision of remediated updates.

Software inventory and third party risk management is today where manufacturing inventory management was in the 1980's:

1. Very little positive control of what flows in from vendors to customers.
2. Very little visibility into their own (or suppliers) internally built software – customers don't know what's in the software that suppliers have delivered, so there's no way to manage the risk of compromised dependencies.
3. No actionable time metrics associated with notification and remediation of compromised deliverables. Lacking visibility, customers don't know when and how their suppliers' products went bad. They don't know how long it took suppliers to detect or respond. And they don't know when a compromised deliverable was updated to a non-compromised deliverable. Without a stopwatch on these supply chain events, it's not feasible for customers to build and enforce contractual guarantees and remedies, to effectively manage third party risk, or to raise the bar for their suppliers based on time to remediation.

Even customers with sophisticated in-house cyber capabilities have not implemented effective risk management for software suppliers, which includes vendors, contract or outsourced software developers. Cyber certification regimes like DoD's CMMC[1] for defense suppliers are a point-in-time compliance drill – a snapshot – that gives suppliers a gold star for three years, with no assurance that any given product will be remediated and updated within hours, days, weeks, months or never[2].

Enterprises must define and enforce actionable supply chain criteria for software running on their infrastructure – vendor, contractor and internally developed code – and make sure that these criteria are continuously and consistently enforced.

Transparency: Software Inventories and Bills of Materials

The first tactical step is relatively simple: know what you have.[3] Given the volume and velocity of software dependency attacks – compromise of third-party software components of supplier code – it is table stakes to require a Software Bill of Material (SBOM) from software vendors, contract software developers and internal developer teams. This data should be structured, machine-readable and must contain all direct and transitive dependencies (components of components) in a software product or build. In forward-leaning industries like financial services, this is already

---

[1] Cybersecurity Maturity Model Certification, https://www.acq.osd.mil/cmmc/

[2] SAP had a critical vulnerability listed and unpatched for over 7 years: https://www.zdnet.com/article/50000-enterprise-firms-running-sap-software-vulnerable-to-attack/

[3] Sun Tzu Quote: "If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle"

standard operating procedure. Large banks maintain prohibited components lists to firewall vulnerable products from procurement, or demand discounts of up to 40% from software providers who cannot product an SBOM[4].

The Department of Commerce's Software Component Transparency initiative[5], comprised of stakeholders in healthcare, finance, government and technology has released a concrete set of definitions with working models of what it means for customers to demand, and for suppliers to provide, SBOMs for initial procurement and ongoing vulnerability management. Notwithstanding resistance from software vendors whose standard practice is to keep customers in the dark for the months it takes to push a patch, the consensus among regulated enterprises is that refusal to provide a third-party ingredients list is an unacceptable position for a software supplier to take. Modern engineering methods (software version control, continuous integration) automate the export of software manifests from development pipelines. Refusal to provide an SBOM means a supplier is either not using modern engineering methods (a huge operational risk), is unreasonably protective of product information which is not the company's "special sauce," or doesn't want to be held accountable for lack of maintenance.

Continuous Software Inventory Monitoring: Restoring Continuity of Assurance

Simply having an inventory list doesn't equal risk management: software inventory lists and SBOMS must be resolved and continuously mapped to live-state data as risks emerge. Third party SBOMs and manifests should be monitored ***out-of-band***, in parallel to a supplier's security process, to ensure that the customer's situational awareness is as timely as the supplier's, and that governance and remediation are triggered on the customer's initiative.

In some circles "SecDevOps" has become totemic buzzword for security. It connotes high levels of skill, agility and technical acumen. Hoodies. Tattoos. Startup attitude. SecDevOps is a twenty-teens code word for The Right Stuff. Cachet aside, what SecDevOps means is that software is security-checked as it's built and is pushed into production (operational deployment) as it's incrementally developed and tested. This is a *first party relationship*: the same company assembles code[6], tests and deploys that code on its own infrastructure. A first party SecDevOps workflow is continuous: the lag between new development and deployed updates is minimal.

For third-party software delivered from a supplier to a customer, SecDevOps is not a meaningful construct. Even if the supplier is automating security checks as they build their product *in-band*, there are discontinuities that leave customers in the blind:

1) Situational Awareness: Lags between the emergence of a vulnerability, supplier knowledge of the risk (if they're not building their software every day, pipeline security automation doesn't give a current read) and customer awareness of the vulnerability.
2) Remediation: Lag between supplier risk awareness and the existence of remediation
3) Software Logistics: Lag between the existence of a remediated update and the delivery of a remediated update to the customer

If the "Sec" is only going on in the supplier's pipeline, a customer has no situational awareness or ability to implement compensating controls until the supplier informs them of a vulnerability, which may not happen even when contract provisions require it. To restore continuity of assurance and

---

[4] FireEye Internal Supply Chain interview: https://www.fireeye.com/content/dam/fireeye-www/global/en/current-threats/pdfs/rpt-best-practices-in-cyber-supply-chain-risk-management.pdf
[5] NTIA WG: https://www.ntia.doc.gov/SoftwareTransparency
[6] Software products and builds are assemblies of open source software, 3rd party binaries and raw code

put customers in the position to take action, it is necessary to “trust but verify” on a continuous basis.

Out-of-band monitoring of software inventory and SBOMs restores continuity of risk management and assurance because:

1) When a vulnerability emerges in a vendor product or contractor deliverable, the customer knows within hours or minutes, regardless of whether the supplier chooses to inform them. Compliance with contract provisions for vulnerability disclosure should also be automated.
2) Supply chain assurance automation can be used to trigger security workflows at the customer enterprise, including the implementation of compensating controls. Procurement and vendor management workflows, including the enforcement of security SLAs and provisions, can be triggered in parallel.

The simple existence of out-of-band monitoring shifts supplier security awareness and responsiveness. If a supplier knows that their customer will know about a newly published vulnerability in one of their software dependencies, whether they tell the customer about it or not, then the existence and remediation of that vulnerable dependency has more business relevance than if their customer was in the blind.

Out-of-band monitoring creates a Heisenberg effect: ***Observation changes what it observes*** One photon of cyber-vigilance gives an enterprise a handle on third party risk management.

In many cases, vendors’ product architecture or modifications of third-party dependencies have already remediated the vulnerabilities that monitoring of non-vendored third party dependencies will detect. In that case, it is the responsibility of the vendor to provide information about those remediations, and it is the responsibility of the customer to make an acceptance decision that doesn’t generate new alarms for remediated products on an ongoing basis. The biggest vendor critique of software transparency, that customers don’t care about documented remediation measures – they just want “clean scans” with no CVEs – is legitimate.

Time: The Metric That Matters

Time to remediation – which can be measured longitudinally as Mean Time to Remediation (MTTR) – is a proxy for almost every relevant process in information assurance. When a vulnerability emerges in a contractor’s build or a vendor’s product, how long does it take them to fix it and to push a remediated update? Hours? Days? Weeks? Months? With continuous out-of-band supply chain monitoring, MTTR can quantify cyber maturity. Suppliers who remediate and update within hours have operational capacity for rapid detection and remediation of security issues. A supplier who takes months to push an update, even when they know the product is being monitored out of band, demonstrates a lack of commitment to customer security and should be risk-rated accordingly.

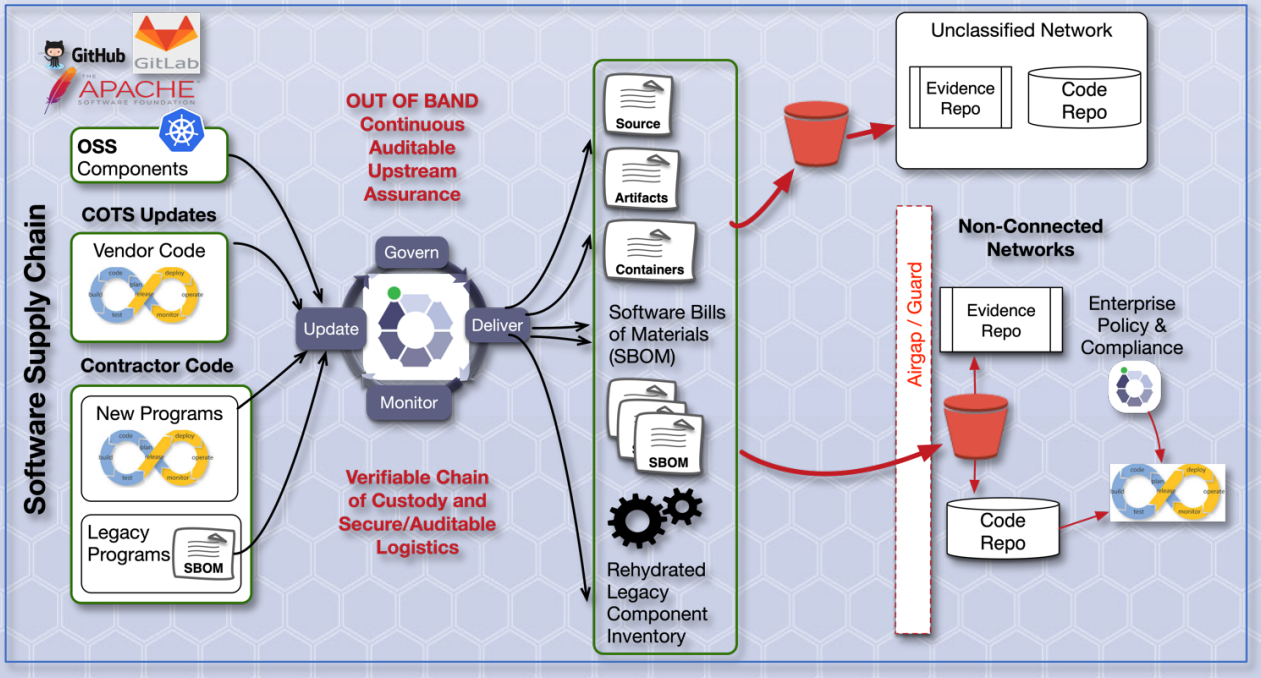
Continuity of assurance – including time-stamped ongoing analysis of previously delivered capabilities and updates – makes it possible to impose and enforce Service Level Agreements for remediation of vulnerabilities, and to tilt procurement in favor of suppliers who are measurably more responsive to security issues. Auditable time-to-remediation allows risk managers to bake security into contract provisions and financial flows, which is what suppliers care about. This can take the form of MTTR as a proposal qualification, as a weighting factor or as a set of financial incentives or financial remedies for measured thresholds of responsiveness to shifts in pass/fail on pre-defined criteria.

Utilizing time as a metric enables other benefits. A supplier who can rapidly remediate a new critical vulnerability in an open source component with a version update is unlikely to be so saddled with technical debt that open source version updates will break their application. Conversely, long remediation times can be an indicator that the underlying product is brittle and has to be refactored – ongoing maintenance has been deferred, which is an operational risk that can be revealed by out-of-band monitoring of a supplier's dependencies: how many are massively out of date, or have no version specifications at all? Are these components abandoned? Have they changed control? When?

How Ion Channel Tackles the Challenge

Ion Channel is a software logistics and supply chain assurance platform geared for continuous assurance of third-party software, from open source libraries to vendor and contractor deliverables. Ion Channel provides software logistics – secure chain of custody, tamper-proofing and auditable delivery of software and software updates to locked-down customer-designated endpoints – and ongoing out-of-band assurance of payloads that have been delivered in the past. Payload types include source code, assured binaries and software containers, and delivery methods include auditable one-way transfers to disconnected networks.

For ongoing assurance, Ion Channel ingests SBOMs, software manifests and spreadsheet inventories of open source components. Assurance can also be automated as software is built on public or private git repositories or in a CI/CD pipeline. The platform is designed to provide transparency for the customer while protecting the supplier's intellectual property: software integrity, provenance, vulnerabilities, licenses, cyber hygiene and ecosystem risks are analyzed and test coverage is integrated into the payload's body of evidence, but supplier source code is not revealed to the customer.



Ion Channel: Supply Chain Risk Management Monitoring & Data Flow

Governance: Basic, Intermediate, Advanced and Continuous

Ion Channel can enforce business rules to gate software ingress to an enterprise. Pushing out the perimeter for ingress mitigates enterprise risk for malware that triggers at install-time vs. run-time: by the time that software is incorporated into a build or installed on a system, the damage is already done. It's better to stop compromised payloads at the loading dock.

Criteria for ongoing governance and compliance range from basic security criteria to fine-grain ecosystem risk metrics for high assurance and mission critical applications. Rules can be layered and tiered to differentiate compliance lanes for elite suppliers and critical or high-exposure systems or preferential procurement workflows. Deliverables and suppliers can be analyzed using different rule sets, allowing customers to understand the compliance cost of raising the bar: if the next level of vigilance is required, how many suppliers shift from green to red? Business lines and risk officers can use these what-if scenarios to make data-driven decisions about what to require and how to incentivize higher security posture across the supplier base.

Risk Vectors (from basic hygiene to high assurance) include:

- Virus detection
- License checks
- Hash checks and change detection
- Known vulnerabilities in the payload and payload dependencies
- Non-specified dependencies, which means the most recent version can roll into any update, increasing risk of dependency attacks.
- Outdated dependencies: technical debt makes the software harder to update
- Ecosystem risks: typo-squatting, risky patterns of maintenance
- Committer counts: how well-supported are the software dependencies? Is this supplier dependent on a component being maintained by one guy as a hobby?
- Change of control in underlying dependencies: have a supplier's dependencies been transferred by one entity to another

Ion Channel data can also be used to detect and enforce blacklists, included prohibited component lists and prohibited committer lists based on non-public or proprietary data. Ion Channel maintains large-scale data assets for deeper analysis into patterns of open source software maintenance, update and control, implicit end-of-life and counterfeiting (e.g. typo squatting). All logistics and assurance data are fully auditable, machine-readable and portable. There is no seat-licensing limit on access, and no restrictions to sharing the data other than commercial resale. Pricing is metered per analysis, which can be event-driven (per commit) or scheduled (e.g. daily).

POCs:

https://ionchannel.io

John Scott, President
John.Scott@ionchanel.io

JC Herz, COO
JC.Herz@ionchannel.io

sales@ionchannel.io