3.3. Vendor

The vendor is the party responsible for updating the product containing the vulnerability. Most often a vendor is a company or other organization, but an individual can also be a vendor. For example, a student who developed an app and placed it in a mobile app store for free download meets this definition of vendor, as does a large multinational company with thousands of developers across the globe. Many open source libraries are maintained by a single person or a small independent team; we still refer to these individuals and groups as vendors.

As software-centric systems find their way into various industries, more and more vendors of traditional products find themselves becoming software vendors.

Moving beyond traditional software companies, recent years have seen the rise in networked products and services from a variety of industries, including those below:

- consumer products, such as home automation and the internet of things (IoT)
- internet service providers (ISPs) and the makers of devices that access ISP services: internet modems, routers, access points, and the like
- mobile phone manufacturers and service providers
- industrial control systems, building automation, HVAC manufacturers
- infrastructure suppliers and increasingly "smart" utility services including water and sewer services and the energy industry
- transportation services, including the airline and automotive industries
- medical devices and health-related device manufacturers

Furthermore, since many modern products are in fact composed of software and hardware components from multiple vendors, the CVD process increasingly involves multiple tiers of vendors, as we discuss in Section 5.4. For example, the CVD process for a vulnerability in a software library component may need to include the originating author of the vulnerable component as well as all the downstream vendors who incorporated that component into their products. Each of these vendors in turn will need to update their products in order for the fix to be deployed to all vulnerable systems.

The NTIA Awareness and Adoption Working Group survey (previously mentioned in Section 2.2) found the following [1]:

- 60-80% of the more mature vendors followed CVD practices
- 76% of those mature vendors developed their vulnerability handling procedures in-house.
- Vendors' perceived need for a vulnerability disclosure policy was driven by a sense of corporate responsibility or customer demand.
- Only a third of responding companies considered and/or required suppliers to have their own vulnerability handling procedures.

Vendor as the Introducer of Vulnerabilities

The vendor often plays an important but less discussed role as well, as the creator of the software or system that introduces the vulnerability. While good practices like code reviews, continuous testing and integration, well-trained developers, mentoring, architectural choices, and so forth can reduce the rate of introduction of new vulnerabilities, these practices thus far have not eliminated them completely. Thus, a well-established CVD capability is also essential to the development process.

Vendor Vulnerability Response Process

In order to effectively mitigate the impact of vulnerabilities in their products and services, vendors must be able to perform the following specific tasks:

- receive reports
- triage, analyze, and test claims made in reports received
- fix bugs
- distribute patch(es)
- (recommended) publish a document
- (recommended) improve internal development process

The ISO/IEC standards 29147 _Vulnerability disclosure_ and 30111 _Vulnerability handling processes_ offer specific models for external- and internal-facing vendor vulnerability response practices. Readers are encouraged to review and apply those standards to their operational vulnerability response practice. ISO/IEC 29147 describes an outward-facing CVD process [2]. ISO/IEC 30111 addresses the internal processes associated with vendor vulnerability response [3].

Evaluating the Vendor Security Response Process

It is a mistake to evaluate a product favorably based solely on its having a low number of publicly known vulnerabilities. In fact, the known vulnerability count in a product is usually not indicative of the quality of a product. There are many reasons a product may have few public vulnerability reports: these include (1) the vendor might lack proper CVD capabilities or have a history of threatening legal action against finders and reporters if they publish vulnerability reports, or (2) the product's prevalence or niche may be too small to warrant finder attention.

Instead, we have found that a vendor's CVD capability and vulnerability response process maturity is often a more important indicator of its commitment to quality than its vulnerability counts alone. Development practices, as human processes, inevitably fail. Vendors that acknowledge this fact and create a good CVD practice are well positioned to compensate for this inevitability.

Vendor Sub-Roles
There are various sub-roles one might find within a vendor organization. In small organizations, an individual might play all the sub-roles at once. Larger organizations often have teams that correspond to the sub-roles identified here. Each of these sub-roles has a part to play in the vendor's vulnerability response practice.

**PSIRT**

A vendor might choose to establish a Product Security Incident Response Team (PSIRT). This is similar to a Computer Security Incident Response Team (CSIRT), but is engaged for product security "incidents" (e.g., vulnerability reports and reports of exploitation of the company's products). The PSIRT acts as an interface between the public and the developers. Examples include the Microsoft Security Response Center (MSRC) [4] and Cisco PSIRT [5]. Many vendor PSIRTs are active in the Forum of Incident Response and Security Teams (FIRST) [7].

**Developers**

For vendors of sufficient size to have a dedicated PSIRT, the vulnerability response and development processes are likely found in different parts of the organization.

The development role usually has the responsibility to:

- identify what to fix and how to fix it
- create the patch
- integrate the patch into releasable products

The PSIRT should be in close contact with the developers in order to coordinated fixes.

**Patch Originator vs. Downstream Vendor**

Although a single vendor is usually the originator of a patch for a given vulnerability, this is not always the case. Some vendors will have products affected by a vulnerability while they are not the originator of the initial fix. Ideally the CVD process should cover not just the patch originator but also the downstream vendors. The complexity of the software supply chain can make this difficult to coordinate as we discuss in Section 5.4.

**Process Improvement**

Having a mechanism to receive and track the disposition of vulnerability reports is an important first step in establishing a vendor's vulnerability response capability. But it should not stop there; vendors should strive for continuous improvement of their software development process. Improving the development process can reduce the number of vulnerabilities in future products. Vendors can establish a feedback loop by performing a root cause analysis of vulnerabilities reported. Lessons learned can then inform modifications to the development process. Some of the ways vulnerability response can feed back into the development lifecycle include the following:

- **Root cause analysis** – to identify common causes and learn how to reduce future introduction of similar vulnerabilities. Questions to ask include the following: How did this vulnerability make it into the released product without being detected? How could it have been found and fixed earlier, before release? How might the vulnerability have been avoided entirely?
- **Automated testing** – to find vulnerabilities sooner, ideally before release. Continuous integration (CI) systems and DevOps practices provide excellent opportunities to incorporate automated security testing. For example, a CI server could initiate a fuzzing campaign on each nightly build of a product. An automated release process might require that code pass all static analysis tests with no significant findings before proceeding.
- **Threat modeling** – to identify high-risk portions of a product earlier in the development process so potential vulnerabilities can be found and addressed at design time, before they are even implemented.

**References**