Software Engineering Institute
Carnegie Mellon University

# The CERT® Guide to Coordinated Vulnerability Disclosure

Allen D. Householder
Garret Wassermann
Art Manion
Chris King

**August 2017**

**SPECIAL REPORT**
CMU/SEI-2017-SR-022

**CERT Division**

Distribution Statement A: Approved for Public Release; Distribution is Unlimited

http://www.sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

This report was prepared for the SEI Administrative Agent AFLCMC/AZS 5 Eglin Street Hanscom AFB, MA 01731-2100

DM17-0508

# Table of Contents

# List of Figures

# List of Tables

# Preface

Software and software-based products have vulnerabilities. Left unaddressed, those vulnerabilities expose to risk the systems on which they are deployed and the people who depend on them. In order for vulnerable systems to be fixed, those vulnerabilities must first be found. Once found, the vulnerable code must be patched or configurations must be modified. Patches must be distributed and deployed. Coordinated Vulnerability Disclosure (CVD) is a process intended to ensure that these steps occur in a way that minimizes the harm to society posed by vulnerable products. This guide provides an introduction to the key concepts, principles, and roles necessary to establish a successful CVD process. It also provides insights into how CVD can go awry and how to respond when it does so.

In a nutshell, CVD can be thought of as an iterative process that begins with someone finding a vulnerability, then repeatedly asking “what should I do with this information?” and “who else should I tell?” until the answers are “nothing,” and “no one.” But different parties have different perspectives and opinions on how those questions should be answered. These differences are what led us to write this guide.

The CERT Coordination Center has been coordinating the disclosure of vulnerability reports since its inception in 1988. Although both our organization and the Internet have grown and changed in the intervening decades, many of the charges of our initial charter remain central to our mission: to facilitate communication among experts working to solve security problems; to serve as a central point for identifying and correcting vulnerabilities in computer systems; to maintain close ties with research activities and conduct research to improve the security of existing systems; and to serve as a model for other incident response organizations.

If we have learned anything in nearly three decades of coordinating vulnerability disclosures at the CERT/CC, it is that there is no single right answer to many of the questions and controversies surrounding the disclosure of information about software and system vulnerabilities. In the traditional computing arena, most vendors and researchers have settled into a reasonable rhythm of allowing the vendor some time to fix vulnerabilities prior to publishing a vulnerability report more widely. Software as a service (SAAS) and software distributed through app stores can often fix and deploy patches to most customers quickly. On the opposite end of the spectrum, we find many Internet of Things (IoT) and embedded device vendors for whom fixing a vulnerability might require a firmware upgrade or even physical replacement of affected devices, neither of which can be expected to happen quickly (if at all). This diversity of requirements forces vendors and researchers alike to reconsider their expectations with respect to the timing and level of detail provided in vulnerability reports. Coupled with the proliferation of vendors who are relative novices at internet-enabled devices and are just becoming exposed to the world of vulnerability research and disclosure, the shift toward IoT can be expected to reinvigorate numerous disclosure debates as the various stakeholders work out their newfound positions.

Here’s just one example: in 2004, it was considered controversial [1] when the CERT/CC advised users to “use a different browser” in response to a vulnerability in the most popular browser of the day (VU#713878) [2]. However, consider the implications today if we were to issue similar advice: “use a different phone,” “drive a different car,” or “use a different bank.” If those phrases give you pause (as they do us), you have recognized how the importance of this issue has grown.

We often find that vendors of software-centric products are not prepared to receive and handle vulnerability reports from outside parties, such as the security research community. Many also lack the ability to perform their own vulnerability discovery within their development lifecycles. These difficulties tend to arise from one of two causes: (a) the vendor is comparatively small or new and has yet to form a product security incident response capability or (b) the vendor has deep engineering experience in its traditional product domain but has not fully incorporated the effect of network enabling its products into its engineering quality assurance practice. Typically, vendors in the latter group may have very strong skills in safety engineering or regulatory compliance, yet their internet security capability is lacking.

Our experience is that many novice vendors are surprised by the vulnerability disclosure process. We frequently find ourselves having conversations that rehash decades of vulnerability coordination and disclosure conversations with vendors who appear to experience something similar to the Kübler-Ross stages of grief (denial, anger, bargaining, depression, and acceptance) during the process.

Furthermore, we have observed that overly optimistic threat models are de rigueur among IoT products. Many IoT products are developed with what can only be described as naïve threat models that drastically underestimate the hostility of the environments into which the product will be deployed.

Even in cases where developers are security-knowledgeable, often they are composing systems out of components or libraries that may not have been developed with the same degree of security consideration. This weakness is especially pernicious in power- or bandwidth-constrained products and services where the goal of providing lightweight implementations can supersede the need to provide a minimum level of security. We believe this is a false economy that only defers a much larger cost when the product or service has been deployed, vulnerabilities are discovered, and remediation is difficult.

We anticipate that many of the current gaps in security analysis knowledge and tools surrounding the emergence of IoT devices will begin to close over the next few years. However, it may be some time before we can fully understand how the products already available today, let alone tomorrow, will impact the security of the networks onto which they are placed. The scope of the problem does not appear to contract any time soon.

We already live in a world where mobile devices outnumber traditional computers, and IoT stands to dwarf mobile computing in terms of the sheer number of devices within the next few years. As vulnerability discovery tools and techniques evolve into this space, so must our tools and processes for coordination and disclosure. Assumptions built into many vulnerability handling processes about disclosure timing, coordination channels, development cycles, scanning, patching, and so forth will need to be reevaluated in the light of hardware-based systems that are likely to dominate the future internet.

## About This Report

This is not a technical document. You will not learn anything new about fuzzing, debugging, ROP gadgets, exploit mitigations, heap spraying, exception handling, or anything about how computers work by reading this report. What you will learn is what happens to that knowledge and how its dissemination is affected by the human processes of communications and social behavior in the context of remediating security vulnerabilities.

This is not a history. We won't spend much time at all on the history of disclosure debates, or the fine details of whether collecting or dropping zero-days is always good or always bad. We will touch on these ideas only insofar as they intersect with the current topic of coordinated vulnerability disclosure.

This is not an indictment. We are not seeking to place blame on one party or another for the success or failure of any given vulnerability disclosure process. We've seen enough disclosure cases to know that people make choices based on their own values coupled with their assessment of a situation, and that even in cases where everyone agrees on what should happen, mistakes and unforeseeable events sometimes alter the trajectory from the plan.

This is not a standard. We assert no authority to bless the information here as "the way things ought to be done." In cases where standards exist, we refer to them, and this report is informed by them. In fact, we've been involved in creating some of them. But the recommendations made in this report should not be construed as "proper," "correct," or "ideal" in any way. As we'll show, disclosing vulnerabilities presents a number of difficult challenges, with long-reaching effects. The recommendations found here do, however, reflect our observation over the past few decades of what works (and what doesn't) in the pursuit of reducing the vulnerability of software and related products.

This is a summary of what we know about a complex social process that surrounds humans trying to make the software and systems they use more secure. It's about what to do (and what not to) when you find a vulnerability, or when you find out about a vulnerability. It's written for vulnerability analysts, security researchers, developers, and deployers; it's for both technical staff and their management alike. While we discuss a variety of roles that play a part in the process, we intentionally chose not to focus on any one role; instead we wrote for any party that might find itself engaged in coordinating a vulnerability disclosure.

We wrote it in an informal tone to make the content more approachable, since many readers' interest in this document may have been prompted by their first encounter with a vulnerability in a product they created or care about. The informality of our writing should not be construed as a lack of seriousness about the topic, however.

In a sense, this report is a travel guide for what might seem a foreign territory. Maybe you've passed through once or twice. Maybe you've only heard about the bad parts. You may be uncertain of what to do next, nervous about making a mistake, or even fearful of what might befall you. If you count yourself as one of those individuals, we want to reassure you that you are not alone; you are not the first to experience events like these or even your reaction to them. We're locals. We've been doing this for a while. Here's what we know.

# Acknowledgments

The material in the CERT® Guide to Coordinated Vulnerability Disclosure inherits from 29 years of analyzing vulnerabilities and navigating vulnerability disclosure issues at the CERT Coordination Center (CERT/CC). While a few of us may be the proximate authors of the words you are reading, many of the ideas these words represent have been bouncing around at CERT for years in one brain or another. We'd like to acknowledge those who contributed their part to this endeavor, whether knowingly or not:

Jared Allar, Jeff Carpenter, Cory Cohen, Roman Danyliw, Will Dormann, Chad Dougherty, James T. Ellis, Ian Finlay, Bill Fithen, Jonathan Foote, Jeff Gennari, Ryan Giobbi, Jeff Havrilla, Shawn Hernan, Allen Householder, Chris King, Dan Klinedinst, Joel Land, Jeff Lanza, Todd Lewellen, Navika Mahal, Art Manion, Joji Montelibano, Trent Novelly, Michael Orlando, Rich Pethia, Jeff Pruzynski, Robert Seacord, Stacey Stewart, David Warren, and Garret Wassermann.

# Executive Summary

Software-based products and services have vulnerabilities—conditions or behaviors that allow the violation of an explicit or implicit security policy. This should come as no surprise to those familiar with software. What many find surprising nowadays is just how many products and services should be considered software based. The devices we depend on to communicate and coordinate our lives, transport us from place to place, and keep us healthy have in recent years become more and more connected both to each other and to the world at large. As a result, society has developed an increasing dependence on software-based products and services along with a commensurate need to address the vulnerabilities that inevitably accompany them.

Adversaries take advantage of vulnerabilities to achieve goals at odds with the developers, deployers, users, and other stakeholders of the systems we depend on. Notifying the public that a problem exists without offering a specific course of action to remediate it can result in giving an adversary the advantage while the remediation gap persists. Yet there is no optimal formula for minimizing the potential for harm to be done short of avoiding the introduction of vulnerabilities in the first place. In short, vulnerability disclosure appears to be a wicked problem.[1]

Coordinated Vulnerability Disclosure (CVD) is a process for reducing adversary advantage while an information security vulnerability is being mitigated. CVD is a process, not an event. Releasing a patch or publishing a document are important events within the process, but do not define it.

CVD participants can be thought of as repeatedly asking these questions: What actions should I take in response to knowledge of this vulnerability in this product? Who else needs to know what, and when do they need to know it? The CVD process for a vulnerability ends when the answers to these questions are *nothing*, and *no one*.

CVD should not be confused with Vulnerability Management (VM). VM encompasses the process downstream of CVD, once the vulnerability has been disclosed and deployers must take action to respond. Section 1 introduces the CVD process and provides notes on relevant terminology.

## Principles of CVD

Section 2 covers principles of CVD, including the following:

- **Reduce Harm** –Decrease the potential for damage by publishing vulnerability information; using exploit mitigation technologies; reducing days of risk; releasing high-quality patches; and automating vulnerable host identification and patch deployment.
- **Presume Benevolence** – Assume that any individual who has taken the time and effort to reach out to a vendor or a coordinator to report an issue is likely benevolent and sincerely wishes to reduce the harm of the vulnerability.

[1] The definition of a *wicked problem* based on an article by Rittel and Webber [41] is given in Section 2.7.

- **Avoid Surprise** – Surprise tends to increase the risk of a negative outcome from the disclosure of a vulnerability and should be avoided.
- **Incentivize Desired Behavior** – It's usually better to reward good behavior than try to punish bad behavior. Incentives are important as they increase the likelihood of future cooperation between security researchers and organizations.
- **Ethical Considerations** – A number of ethical guidelines from both technical and journalistic professional societies can find application in the CVD process.
- **Process Improvement** – Participants in the CVD process should learn from their experience and improve their process accordingly. CVD can also provide important feedback to an organization's Software Development Lifecycle (SDL).
- **CVD as a Wicked Problem** – As we've already mentioned, vulnerability disclosure is a multifaceted problem for which there appear to be no "right" answers, only "better" or "worse" solutions in a given context.

## Roles in CVD

CVD begins with finding vulnerabilities and ends with the deployment of patches or mitigations. As a result, several distinct roles and stakeholders are involved in the CVD process. These include the following:

- **Finder (Discoverer)** – the individual or organization that identifies the vulnerability
- **Reporter** – the individual or organization that notifies the vendor of the vulnerability
- **Vendor** – the individual or organization that created or maintains the product that is vulnerable
- **Deployer** – the individual or organization that must deploy a patch or take other remediation action
- **Coordinator** – an individual or organization that facilitates the coordinated response process

It is possible and often the case that individuals and organizations play multiple roles. For example, a cloud service provider might act as both vendor and deployer, while a researcher might act as both finder and reporter. A vendor may also be both a deployer and a coordinator.

Reasons to engage a coordinator include reporter inexperience, reporter capacity, multiparty coordination cases, disputes among CVD participants, and vulnerabilities having significant infrastructure impacts.

Users, integrators, cloud and application service providers, Internet of Things (IoT) and mobile vendors, and governments are also stakeholders in the CVD process. We cover these roles and stakeholders in more detail in Section 3.

## Phases of CVD

The CVD process can be broadly defined as a set of phases, as described in Section 4. Although these phases may sometimes occur out of order, or even recur within the handling of a single vulnerability case (for example, each recipient of a case may need to independently validate a report), they often happen in the following order:

- **Discovery** – Someone discovers a vulnerability in a product.
- **Reporting** – The product's vendor or a third-party coordinator receives a vulnerability report.
- **Validation and Triage** – The receiver of a report validates it to ensure accuracy before prioritizing it for further action.
- **Remediation** – A remediation plan (ideally a software patch, but could also be other mechanisms) is developed and tested.
- **Public Awareness** – The vulnerability and its remediation plan is disclosed to the public.
- **Deployment** – The remediation is applied to deployed systems.

## CVD Process Variation

As an endeavor of human coordination at both the individual and organization levels, the CVD process can vary from participant to participant, over time, and in varying contexts. Some points of variation include those below:

- **Choosing a disclosure policy** – Disclosure policies may need to be adapted for different organizations, industries, and even products due to variations in business needs such as patch distribution or safety risks.
- **Coordinating among multiple parties** – Coordination between a single finder and a single vendor is relatively straightforward, but cases involving multiple finders, or complex supply chains often require extra care.
- **Pacing and synchronization** – Different organizations work at different operational tempos, which can increase the difficulty of synchronizing release of vulnerability information along with fixes.
- **Coordination Scope** – CVD participants must decide how far to go with the coordination process. For example, it may be preferable to coordinate the disclosure of critical infrastructure vulnerabilities all the way out to the system deployers, while for a mobile application it may be sufficient to notify the developer and simply allow the automatic update process take it from there.

Variation points in the CVD process are covered in Section 5.

## Troubleshooting CVD

CVD does not always go the way it's supposed to. We have encountered a number of obstacles along the way, which we describe in Section 6. These are among the things that can go wrong:

- **No vendor contact available** – This can occur because a contact could not be found, or the contact is unresponsive.
- **Participants stop responding** – Participants in CVD might have other priorities that draw their attention away from completing a CVD process in progress.
- **Information leaks** – Whether intentional or not, information that was intended for a private audience can find its way to others not involved in the CVD process.
- **Independent discovery** – Any vulnerability that can be found by one individual can be found by another, and not all of them will tell you about it.

- **Active exploitation** – Evidence that a vulnerability is being actively exploited by adversaries often implies a need to accelerate the CVD process to reduce users' exposure to risk.
- **Relationships go awry** – CVD is a process of coordinating human activities. As such, its success depends on building relationships among the participants.
- **Hype, marketing, and unwanted attention** – The reasons for reporting and disclosing vulnerabilities are many, but in some cases they can be used as a tool for marketing. This is not always conducive to the smooth flow of the CVD process.

When things do go askew in the course of the CVD process, it's often best to remain calm, avoid legal entanglements, and recognize that the parties involved are usually trying to do the right thing. In some cases, it may help to consider publishing earlier than originally planned or to engage a third-party coordinator to assist with mediating disputes. Regardless of the resulting action, CVD participants should learn from the experience.

## Operational Considerations

Participation in the CVD process can be improved with the support of tools and operational processes such as secure communications (e.g., encrypted email or https-enabled web portals), contact management, case tracking systems, code and system inventories, and test environments such as virtualized labs.

Operational security should also be considered. CVD participants will need to address key management for whatever communications encryption they decide to use. Policy guidelines for handling sensitive data should be clearly articulated within organizations. Furthermore, recipients of vulnerability reports (e.g., vendors and coordinators) should be wary of credulous action in response to reports. Things are often not what they originally seem. Reporters may have misinterpreted the impact of a vulnerability to be more or less severe than it actually is. Adversaries may be probing an organization's vulnerability response process to gain information or to distract from other events.

As happens in many security operations roles, staff burnout is a concern for managers of the CVD process. Job rotations and a sustained focus on CVD process improvement can help.

Further discussion of operational considerations can be found in Section 7.

## Open Problems in CVD

Organizations like the CERT Coordination Center have been coordinating vulnerability disclosures for decades, but some issues remain to be addressed. The emergence of a wider diversity of software-based systems in recent years has led to a need to revisit topics once thought nearly resolved. Vulnerability identity has become a resurgent issue in the past few years as the need to identify vulnerabilities for purposes of CVD and vulnerability management has spread far beyond the arena of traditional computing. A number of efforts are currently underway to improve the way forward.

More broadly, the rising prevalence of IoT products and their corresponding reliance on embedded systems with constrained hardware, power, bandwidth, and processing capabilities has led to a need to rethink CVD in light of assumptions that are no longer valid. Patching may be comparatively easy on a Windows system deployed on an enterprise network. Patching the firmware of a home router deployed to all the customers of a regional ISP is decidedly not so simple. The desktop system the doctor uses to write her notes might be patched long before the MRI machine that collected the data she's analyzing. Fixing a vulnerable networked device atop a pipeline in a remote forest might mean sending a human out to touch it. Each of these scenarios comes with an associated cost not usually factored into the CVD process for more traditional systems.

The way industries, governments, and society at large will address these issues remains to be seen. We offer Section 8 in the hope that it sheds some light on what is already known about these problems.

## Conclusion and Appendices

Vulnerability disclosure practices no longer affect only the computer users among us. Smart phones, ATMs, MRI machines, security cameras, cars, airplanes, and the like have become network-enabled software-dependent systems, making it nearly impossible to avoid participating in the world without the potential to be affected by security vulnerabilities. CVD is not a perfect solution, but it stands as the best we've found so far. We've compiled this guide to help spread the practice as widely as possible.

Five appendices are provided containing background on IoT vulnerability analysis, Traffic Light Protocol, examples of vulnerability report forms and disclosure templates, and pointers to five publicly available disclosure policy templates. An extensive bibliography is also included.

# Abstract

Security vulnerabilities remain a problem for vendors and deployers of software-based systems alike. Vendors play a key role by providing fixes for vulnerabilities, but they have no monopoly on the ability to discover vulnerabilities in their products and services. Knowledge of those vulnerabilities can increase adversarial advantage if deployers are left without recourse to remediate the risks they pose. Coordinated Vulnerability Disclosure (CVD) is the process of gathering information from vulnerability finders, coordinating the sharing of that information between relevant stakeholders, and disclosing the existence of software vulnerabilities and their mitigations to various stakeholders including the public. The CERT Coordination Center has been coordinating the disclosure of software vulnerabilities since its inception in 1988. This document is intended to serve as a guide to those who want to initiate, develop, or improve their own CVD capability. In it, the reader will find an overview of key principles underlying the CVD process, a survey of CVD stakeholders and their roles, and a description of CVD process phases, as well as advice concerning operational considerations and problems that may arise in the provision of CVD and related services.

# 1 Introduction

Imagine you’ve found a vulnerability in a product. What do you do with this knowledge?

Maybe nothing. There are many situations in which it’s perfectly reasonable to decide to go on about your business and let somebody else deal with it. You’re busy. You have more important things to do. It’s not your code, so it’s not your problem. Or maybe it is your code, but you wrote it a long time ago, and it will take a lot of effort to even try to fix it. Maybe the product has already reached end-of-life. Or perhaps fixing this bug will delay the launch of your new product that will supersede this version anyway. There are plenty of good reasons that it might not be worth the hassle of fixing it or reporting it. Although this is what a mathematician would refer to as a degenerate case of the disclosure process in which no disclosure occurs, it’s important to recognize that even starting the process of disclosure is a choice one must consider carefully.

Often, though, you will likely feel a need to take some action in response to this newfound knowledge of a vulnerability. If it’s your product, you might immediately set off to understand the root cause of the problem and fix it. Once fixed, you probably will want to draw attention to the fix so your product’s users can protect themselves.

Or perhaps the product is other people’s responsibility to fix, and you want to inform them of the existence of this vulnerability.

This is where the process of Coordinated Vulnerability Disclosure (CVD) begins.

## 1.1 Coordinated Vulnerability Disclosure is a Process, Not an Event

A process is “a series of actions or steps taken in order to achieve a particular end” [3]. Publishing a document is an action. Releasing a fix is an action. And while both of these are common events within the CVD process, they do not define it.

Perhaps the simplest description of the CVD process is that it starts with at least one individual becoming aware of a vulnerability in a product. This discovery event immediately divides the world into two sets of people: those who know about the vulnerability, and those who don’t. From that point on, those belonging to the set that knows about the vulnerability iterate on two questions:

1. What actions should I take in response to this knowledge?
2. Who else needs to know what, and when?

The CVD process continues until the answers to these questions are “nothing,” and “nobody.”

Simple enough? Hardly. If it were, this document would be considerably shorter. But with this simple iterator in mind, we’ll be better able to frame our discussion.

Ideally, product and service vulnerabilities would be either discovered by the vendor (developer) of the software product or service itself or reported to the vendor by a third party (finder, reporter). Informing vendors enables them to take action to address and correct vulnerabilities. In most cases, the vendor is the party best suited to correct the vulnerability at its origin. Vendors typically remediate vulnerabilities by developing and releasing an update to the product, also known as a patch. However, often the vendor issuing an update is just the first step towards remediation of the installed base of vulnerable systems. Deployers must still ensure that patches are deployed in a timely manner to the systems they need to protect. A more detailed discussion of roles in CVD can be found in Section 3.

## 1.2 CVD Context and Terminology Notes

Before we proceed to place CVD in context, we start with a few definitions.

### 1.2.1 Vulnerability

A *vulnerability* is a set of conditions or behaviors that allows the violation of an explicit or implicit security policy. Vulnerabilities can be caused by software defects, configuration or design decisions, unexpected interactions between systems, or environmental changes. Successful exploitation of a vulnerability has technical and risk impacts. Vulnerabilities can arise in information processing systems as early as the design phase and as late as system deployment.

NIST offers the following definitions of vulnerability [4]:

1. "Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source"
2. "A weakness in a system, application, or network that is subject to exploitation or misuse"
3. "Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited by a threat source"

Those familiar with the CERT Resiliency Management Model (RMM) may be accustomed to the more general definition of vulnerability in the Vulnerability Analysis and Resolution (VAR) practice: "A vulnerability is the susceptibility of an asset and associated service to disruption" [5]. A summary of the VAR process area of the CERT RMM can be found in Section 1.2.6.1.

While vulnerabilities can be found in many assets belonging to an organization—people, information, technology, and facilities—in this document we primarily focus on vulnerabilities in software or software-centric products and to a lesser degree services built on software-dependent products

While precisely defining vulnerability can be difficult, for our purpose a vulnerability may be thought of as an undesirable, exploitable, and likely unintended feature of software or hardware components that allows an attacker to perform actions that wouldn't otherwise be available to them. The impact of such vulnerabilities can vary greatly, from being able to access someone's private data, to taking control of a computer, to causing physical damage and bodily injury.

### 1.2.2 Exploits, Malware, and Incidents

We also need to get a few quick technical terms out of the way so they don't cloud the remaining discussion. An *exploit* is software that uses a vulnerability to achieve some effect. Sometimes the effect is as simple as demonstrating the existence of the vulnerability. Other times it plays a role in enabling adversaries to attack systems. *Malware* is software used by adversaries to compromise the security of a system or systems. But not all malware involves exploits. Finally, an *incident* is a violation or an attempted violation of a security policy, and may involve malware, exploits, or vulnerabilities (or none of these!)

### 1.2.3 Vulnerability Response (VR)

Vulnerability Response (VR) is the overall set of processes and practices that deal with the existence of vulnerabilities in systems. VR encompasses everything from reducing the introduction of vulnerabilities as part of a Secure Development Lifecycle (SDL) through the remediation of deployed vulnerabilities via patch deployment.

Vulnerability response in the design and development phases often takes the form of practices such as threat modeling [6] [7], secure coding [8] [9] [10], and architecture risk analysis [11] [12]. However, such practices seem unlikely to ever completely eliminate vulnerabilities from being introduced into released software and deployed systems. For those vulnerabilities that do escape detection by these early lifecycle practices, it is necessary to plan for their eventual discovery and disclosure.

The goals of vulnerability response include the following:

- Limit attacker advantage over defenders.
- Reduce the population of vulnerable product instances as quickly as possible.
- Reduce the impact of attacks against vulnerable systems.

### 1.2.4 Vulnerability Discovery

Vulnerability discovery can take many forms, from specifically targeted software testing to simple use of a system by a security-aware individual who notices some feature that seems out of place. In order for that discovery to be relevant to our discussion, it must result in a vulnerability report. Most discussions about vulnerability disclosure are referring to the handling of reports of newly discovered vulnerabilities in products for which no patch exists (for a more nuanced discussion regarding why we’re eschewing the term *zero-day vulnerability* here, see [13]). We further distinguish *vulnerability discovery* from *vulnerability scanning* in Section 1.2.7.

### 1.2.5 Coordinated Vulnerability Disclosure

Coordinated Vulnerability Disclosure is the process of gathering information from vulnerability finders, coordinating the sharing of that information between relevant stakeholders, and disclosing the existence of software vulnerabilities and their mitigations to various stakeholders, including the public. CVD is an important aspect of any successful VR process. CVD inputs are vulnerability reports arising from vulnerability discovery practices. CVD outputs for product vulnerabilities (software or hardware) usually include patches as well as vulnerability report documents or vulnerability database records, typically with some formal identifier (e.g., CVE [14], VU# [15], and BID [16]). Many operational vulnerabilities such as router misconfigurations, website vulnerabilities, or cloud service problems can be fixed in situ by the operator, but often do not result in a public disclosure.

ISO/IEC 29147 [17] defines Vulnerability Disclosure as follows:

> *Vulnerability disclosure is a process through which vendors and vulnerability finders may work cooperatively in finding solutions that reduce the risks associated with a vulnerability.*

> *It encompasses actions such as reporting, coordinating, and publishing information about a vulnerability and its resolution.*
>
> *The goals of vulnerability disclosure include the following: a) ensuring that identified vulnerabilities are addressed; b) minimizing the risk from vulnerabilities; c) providing users with sufficient information to evaluate risks from vulnerabilities to their systems;*

The stakeholders—in other words, the people who care about the existence of a vulnerability—vary on a case by case basis, but typically include those below:

- the reporter or finder of the vulnerability (often an independent security researcher)
- the vendor (developer) of the component that contains the vulnerability ("originating vendor")
- vendors that utilize the component containing the vulnerability in their own products ("downstream vendors")
- coordinators, vulnerability databases, or other organizations that specialize in incident response and vulnerability handling
- the general public / consumers who purchase and use products containing the vulnerable component

Disclosure, in turn, is the process by which information about a vulnerability (ideally with advice for mitigating or fixing it) is released to consumers of the product, and more generally, the public at large.

There is no single "right" way to do this. Sometimes, vulnerability information is disclosed in a blog post by the finder of the vulnerability, or emailed to a security mailing list. Sometimes the vendor issues a security advisory to its customers or to the public. At the CERT/CC, we publish Vulnerability Notes on our website, often in parallel with other parties (i.e., the finder of the vulnerability and/or the vendor of the vulnerable product).

Furthermore, there persists a lack of agreement within the security community on whether, and under what conditions, vulnerability information should be disclosed to vendors, other stakeholders, and the public. Different people sometimes hold strongly differing opinions about the disclosure of software vulnerabilities. These differences tend to center on the timing of a vulnerability report's release, the type and degree of details included, and the audience to whom the report is provided.

As a result, the character of information in a vulnerability report can vary greatly. Some reports only warn of a general vulnerability in a specific product. Others are more detailed and provide actual examples of how to attack the flaw (these examples are called "proof of concept code," often shortened to "PoC").

It is worth reiterating that disclosure is not a singular event even for a single vulnerability. For more on the different phases of the process, see Section 3.

#### 1.2.5.1 Who is Responsible Here?

You may be familiar with the term *responsible disclosure* [18] and wonder how it's different from CVD. The history of *responsible disclosure* makes for a long story best told over adult beverages at a hotel bar during a security conference, so we won’t go into it here. Without belaboring the topic, the sticking point comes down to the fact that what constitutes “responsible” behavior is a matter of opinion that is always framed within the values of whoever is using the term. The vendors cry, “Disclosing a vulnerability without an available patch is not responsible!” “Not fixing this vulnerability quicker is not responsible!” the finders retort. Meanwhile, the deployer asks, “Who’s responsible for fixing this?” while knowing the answer all too well.

Because of the inherent value judgement and lack of agreement on its definition, the CERT/CC, along with numerous other organizations, advocates for the use of the term *Coordinated Vulnerability Disclosure (CVD)* [19] [20] to reduce misunderstanding and promote cooperation.

### 1.2.6 Vulnerability Management (VM)

Vulnerability Management (VM) is the common term for tasks such as vulnerability scanning, patch testing, and deployment. VM practices nearly always deal with the output of CVD practices, not the inputs. VM practices focus on the positive action of identifying specific systems affected by known (post-disclosure) vulnerabilities and reducing the risks they pose through the application of mitigations or remediation such as patches or configuration changes. NIST Special Publication 800-40 provides a Guide to Enterprise Patch Management Technologies [21]. VM practices also appear within the Vulnerability Analysis and Resolution operational process of the CERT RMM [5].

#### 1.2.6.1 Vulnerability Analysis and Resolution (VAR)

Vulnerability Analysis and Resolution (VAR) is an operational process described within the CERT RMM that closely overlaps with the concept of Vulnerability Management. Although the RMM is designed with a focus on operational resilience for organizations, there is sufficient overlap with our topic that it’s worth highlighting here. Within the RMM’s VAR process area, a number of goals and practices are identified:

- Prepare for Vulnerability Analysis and Resolution.
  - Establish Scope – The assets and operational environments that must be examined for vulnerabilities are identified.
  - Establish a Vulnerability Analysis and Resolution Strategy.
- Establish and maintain a process for identifying and analyzing vulnerabilities.
  - Identify Sources of Vulnerability Information.
  - Discover Vulnerabilities.
  - Analyze Vulnerabilities to determine whether they need to be reduced or eliminated.
- Manage Exposure to Vulnerabilities – Strategies are developed and implemented to manage exposure to identified vulnerabilities.
- Identify Root Causes – The root causes of vulnerabilities are examined to improve vulnerability analysis and resolution and reduce organizational exposure. Perform review of identified vulnerabilities to determine and address underlying causes.

### 1.2.7 Products and Instances

In talking about things that have vulnerabilities, we try to maintain a clear distinction between a *product* being vulnerable, and an *instance of a product* being vulnerable. For example, Windows 10 (the product) might be vulnerable to a specific flaw, but that is a separate situation from a server running Windows 10 (an instance) being vulnerable. Vulnerabilities affecting products may not always affect every instance of a product; for example, a vulnerability may require a special configuration or setup to be exploited, so any instance not in that configuration state would actually be unaffected by the vulnerability, despite the product at-large being vulnerable.

This distinction becomes important when one is talking about the practices associated with Vulnerability Management (VM)—namely *vulnerability scanning*—in contrast to CVD and *vulnerability discovery*. VM entails the identification of *instances* of a *product* on which action must be taken to remediate known vulnerabilities in the product. VM is concerned with the eradication of the instances of known vulnerabilities in deployed systems, whereas CVD is concerned with the repair of vulnerabilities at the product level.

### 1.2.8 Incident vs. Vulnerability Response

Sometimes the term “Incident Response” is used synonymously with Vulnerability Response. These two concepts are related, but different; Vulnerability Response specifically indicates responding to reports of product vulnerabilities, usually via the CVD process, whereas Incident Response is more general and can also include other security events such as network intrusions. We will generally stick to the Vulnerability Response terminology since this work is specifically about CVD.

## 1.3 Why Coordinate Vulnerability Disclosures?

Vulnerability disclosures fall between two extremes:

1. Disclose everything you know about a vulnerability to everyone as soon as you know it.
2. Never disclose anything you know about a vulnerability to anyone.

Prior research into vulnerability disclosure practices [22] has shown that neither approach is socially optimal. Thus, we are given to hope that we can improve on these extremes by striking a balance in between. But doing so requires several questions to be answered: how much information should be released? To whom? And when? Do you wait for a patch to be deployed before announcing the vulnerability’s existence? Do you wait for the patch to be available but not yet deployed? Is it okay to acknowledge that you know of a vulnerability in a product without providing any other details?

It’s also important to consider that not all factors are within control of the parties involved in the disclosure process. Adversaries can discover vulnerabilities and use them to exploit vulnerable systems regardless of your participation in a well-coordinated disclosure process. And yet many vulnerabilities might never be exploited in attacks. So how should we approach the question of potential harm and the questions surrounding risk and reward of vulnerability disclosure?

The CERT/CC believes the Coordinated Vulnerability Disclosure (CVD) process provides a reasonable balance of these competing interests. The public and especially users of vulnerable products deserve to be informed about issues with those products and how the vendor handles those issues. At the same time, disclosing such information without review and mitigation only opens the public up to exploitation. The ideal scenario occurs when everyone coordinates and cooperates to protect the public. This coordination may also be turned into a public relations win for the vendor by quickly addressing the issue, thereby avoiding bad press for being unprepared.

Some vendors express concern about the negative attention brought by having a long list of publicly disclosed vulnerabilities in their products. In our opinion, the number of vulnerabilities found in a vendor's products is less valuable as an indicator of the vendor's security stance than the consistency of its response to vulnerabilities in a comprehensive and timely manner. In the end, the goal of CVD is to help users make more informed decisions about actions they can take to secure their systems.

The Forum of Incident Response and Security Teams (FIRST) [23], which consists of many public and private organizations and companies involved in vulnerability and security incident handling, has established a Vulnerability Coordination Special Interest Group to develop some common CVD best practices and guidelines [24]. While the existence of individual vulnerabilities may be unexpected and surprising, these common practices should help lead to fewer surprises for all stakeholders in the CVD process itself.

Governments and international organizations also recognize the need for coordinated vulnerability disclosure practices. In 2015, the Department of Commerce's National Telecommunications and Information Administration initiated a Multistakeholder Process for Cybersecurity Vulnerabilities [25] to

> *develop a broad, shared understanding of the overlapping interests between security researchers and the vendors and owners of products discovered to be vulnerable, and to establish a consensus about voluntary principles to promote better collaboration. The question of how vulnerabilities can and should be disclosed will be a critical part of the discussion, as will how vendors receive and respond to this information. However, disclosure is only one aspect of successful collaboration.*

## 1.4 Previewing the Remainder of this Document

We explore a number of principles of Coordinated Vulnerability Disclosure in Section 2. Section 3 describes the various roles involved in CVD. Common phases of the CVD process are covered in Section 4. The CVD process can vary depending on multiple factors, which we discuss in Section 5. But things do not always go smoothly, so Section 6 offers advice for troubleshooting the CVD process. Section 7 highlights operational considerations surrounding implementation of a CVD capability. In Section 8, we discuss a few open issues in the CVD space. Our conclusion can be found in Section 9, followed by a bibliography and multiple appendices. The appendices contain additional information about CVD issues specific to the Internet of Things, sample forms used in CVD processes, as well as references to disclosure policies, practices, and related information.

# 2 Principles of Coordinated Vulnerability Disclosure

*Change your opinions, keep to your principles; change your leaves, keep intact your roots.*

*– Victor Hugo*

Over the years, the CERT/CC has identified a number of principles that guide our efforts in coordinating vulnerability disclosures and which seem to be present in many successful CVD programs. These principles include the following:

- Reduce Harm
- Presume Benevolence
- Avoid Surprise
- Incentivize Desired Behavior
- Ethical Considerations
- Process Improvement
- CVD as a Wicked Problem

We cover each of these in more detail below.

## 2.1 Reduce Harm

Harm reduction is a term borrowed from the public health community. In that context, it is used to describe efforts intended to reduce the harm caused by drug use and unsafe health practices, rather than on the eradication of the problem. For example, one of the tenets of harm reduction is that there will never be a drug-free society, and so preparations must be made to reduce the harm of drugs that currently exist since we will never be completely free of them [26] [27].

This concept applies to software vulnerabilities as well: that it may be possible to reduce the potential for harm even if vulnerabilities cannot be fully eliminated. At its core, harm reduction with respect to vulnerable software is about balancing the ability for system defenders to take action while avoiding an increase in attacker advantage.

Experience has shown that nearly all software-centric products contain vulnerabilities, and this will likely remain true, especially as code complexity continues to increase. In fact, the potential for vulnerabilities will likely never go away since a previously secure system can become vulnerable when deployed into a new context, or simply due to environmental changes or the development of novel attack techniques. Systems tend to outlive their threat models. The Flatiron Building in New York City stands as an example of this phenomenon in the physical world. Built prior to the Wright brothers' flight at Kitty Hawk, NC, today it is vulnerable to attack using an airliner as a weapon. It's difficult to argue that the designers should have "built security in" for attacks that would have been considered science fiction at the time of deployment [28].

Since vulnerabilities are likely to persist despite our best efforts, CVD works best when it focuses on reducing the harm vulnerabilities can cause. Some approaches to reducing the harm caused by vulnerable software and systems include the following:

- Publishing vulnerability information. Providing high-quality, timely, targeted, automated dissemination of vulnerability information enables defenders to make informed decisions and take action quickly.
- Encouraging the adoption and widespread use of exploit mitigation techniques on all platforms.
- Reducing days of risk. Selecting reasonable disclosure deadlines is one way of achieving the goal of minimizing the time between a vulnerability's discovery and the remediation of its last deployed instance [22]. Another way is to shorten the time between vulnerability disclosure and patch deployment by automating patch distribution using secure update mechanisms that make use of cryptographically signed updates or other technologies.
- Releasing high-quality patches. Increasing defenders' trust that patches won't break things or have undesirable side effects reduces lag in patch deployment by reducing the defenders' testing burden.
- When possible, automated patch deployment can improve patch deployment rates too.

## 2.2 Presume Benevolence

Benevolence refers to the morally valuable character trait or virtue of being inclined to act to benefit others. In terms of the CVD process, we have found that it is usually best to assume that any individual who has taken the time and effort to reach out to a vendor or a coordinator to report an issue is likely benevolent and sincerely wishes to reduce the risk posed by the vulnerability. While each reporter may have secondary motives (such as those listed in Table 1 below), and may even be difficult to work with at times, allowing negative associations about a CVD participants' motives to accumulate can color your language and discussions with them.

This isn't to say you should maintain your belief that researcher is acting in good faith when presented with evidence to the contrary. Rather, one should keep in mind that participants are working toward a common goal: reducing the harm caused by deployed insecure systems.

I Am the Cavalry describes Finder/Reporter motivations thus [29]:

*Table 1: I Am the Cavalry's Finder / Reporter Motivations*

| Finder / Reporter Motivation | Description |
|---|---|
| **Protect** | make the world a safer place. These researchers are drawn to problems where they feel they can make a difference. |
| **Puzzle** | tinker out of curiosity. This type of researcher is typically a hobbyist and is driven to understand how things work. |

| | |
|---|---|
| **Prestige** | seek pride and notability. These researchers often want to be the best, or very well known for their work. |
| **Profit** | to earn money. These researchers trade on their skills as a primary or secondary income. |
| **Politics** | ideological and principled. These researchers, whether patriots or protestors, strongly support or oppose causes. |

The Awareness and Adoption Group within the NTIA Multistakeholder Process for Cybersecurity Vulnerabilities [25] surveyed security researchers and vendors, finding that [30]

- 92% of researchers participate in some form of CVD.
- 70% of researchers expected regular communication from the vendor about their report. Frustrated expectations were often cited as the reason for abandoning the CVD process
- 60% of researchers cited threat of legal action as a reason they might not work with a vendor to disclose
- 15% of researchers expected a bounty in return for their disclosure

## 2.3 Avoid Surprise

As with most situations in which multiple parties are engaged in a potentially stressful and contentious negotiation, surprise tends to increase the risk of a negative outcome. The importance of clearly communicating expectations across all parties involved in a CVD process cannot be overemphasized.

If we expect cooperation between all parties and stakeholders, we should do our best to match their expectations of being “in the loop” and minimize their surprise. Publicly disclosing a vulnerability without coordinating first can result in panic and an aversion to future cooperation from vendors and finders alike. CVD promotes continued cooperation and increases the likelihood that future vulnerabilities will also be addressed and remedied.

## 2.4 Incentivize Desired Behavior

A degree of community outreach is an important part of any CVD process. Not everyone shares the same values, concerns, perspectives, or even ethical foundations, so it’s not reasonable to expect everyone to play by your rules. Keeping that in mind, we’ve found that it’s usually better to reward good behavior than try to punish bad behavior. Such incentives are important as they increase the likelihood of continued cooperation between CVD participants.

Incentives can take many forms:

- **Recognition** – Public recognition is often used as a reward for “playing by the rules” in CVD.
- **Gifts** – Small gifts (or “swag”) such as T-shirts, stickers, and so forth give researchers a good feeling about the organization.

- **Money** – Bug bounties can turn CVD into piece work.
- **Employment** – We have observed cases where organizations choose to hire the researchers who report vulnerabilities to them, either on a temporary (contract) or full-time basis. This is of course neither required nor expected, but having a reputation of doing so can be an effective way for a vendor to encourage positive interactions.

## 2.5 Ethical Considerations

At present, there is no generally accepted set of ethical guidelines for CVD. In the security response arena, work toward defining ethical guidelines is ongoing. The Forum of Incident Response and Security Teams (FIRST) has established a special interest group to develop a code of ethics for its member teams and liaisons [31]. However, that does not imply that there is a complete absence of relevant guidance in the matter. Here we highlight some ethics advice from related sources.

### 2.5.1 Ethics in Related Professional Societies

Various computing-related professional societies have established their own codes of ethics. Each of these has application to CVD.

The Association for Computing Machinery (ACM) Code of Ethics and Professional Conduct [32] includes the following general imperatives:

- Contribute to society and human well-being.
- Avoid harm to others.
- Be honest and trustworthy.
- Be fair and take action not to discriminate.
- Honor property rights including copyrights and patent.
- Give proper credit for intellectual property.
- Respect the privacy of others.
- Honor confidentiality.

The Usenix' System Administrators' Code of Ethics [33] includes an ethical responsibility "to make decisions consistent with the safety, privacy, and well-being of my community and the public, and to disclose promptly factors that might pose unexamined risks or dangers."

### 2.5.2 Journalism Ethics

In many ways, disclosing a vulnerability can be thought of as a form of journalistic reporting, in that

> *The purpose of journalism is … to provide citizens with the information they need to make the best possible decisions about their lives, their communities, their societies, and their governments [34].*

By analogy, vulnerability disclosure provides individuals and organizations with the information they need to make the best possible decisions about their products, their computing systems and networks, and the security of their information.

We find the four major principles offered by The Society of Professional Journalists Code of Ethics to be relevant to CVD as well [35]:

- **Seek truth and report it** – Ethical journalism should be accurate and fair. Journalists should be honest and courageous in gathering, reporting and interpreting information.
- **Minimize harm** – Ethical journalism treats sources, subjects, colleagues and members of the public as human beings deserving of respect.
- **Act independently** – The highest and primary obligation of ethical journalism is to serve the public.
- **Be accountable and transparent** – Ethical journalism means taking responsibility for one's work and explaining one's decisions to the public.

## 2.6 Process Improvement

In reviewing their experience in the CVD process, participants should capture ideas that worked well and note failures. This feedback can be used to improve both the Software Development Lifecycle and the CVD process itself.

The CVD process can create a pipeline for regular patching cycles and may reveal blocking issues that prevent a more efficient software patch deployment mechanism. A successful program provides the vendor with a degree of crowdsourcing for security research and testing of its products. However, CVD should be considered complementary to a vendor's internal research and testing as part of the Software Development Lifecycle, not as a wholesale replacement for internally driven security testing.

### 2.6.1 CVD and the Security Feedback Loop

A successful CVD program feeds vulnerability information back into the vendor's Software Development Lifecycle. This information can result in more secure development processes, helping to prevent the introduction of vulnerabilities in the first place.

Yet the reality of today's software is that much of its legacy code was not originally produced within a secure development process. Andy Ozment and Stuart Schechter studied the impact of legacy code on the security of modern software and how large code changes might introduce vulnerabilities [36]. The positive news is that *foundational vulnerabilities*—ones that existed in the very first release and carried through the most recent version of the software—decay over time. We can find them, fix them, and make the code base stronger overall. However, the bad news is that as the low-hanging fruit of foundational vulnerabilities are fixed, the remaining foundational vulnerabilities tend to be more subtle or complex, making them increasingly difficult to discover.

Furthermore, ongoing development and code changes can introduce new vulnerabilities, making it unlikely for the security process to ever be "finished." Even with modern architecture development and secure coding practices, software bugs (and in particular security vulnerabilities) remain a likely result as new features are added or code is refactored. This can happen for many reasons, not all of them technical. A recent article highlighted the difficulty of getting teams of people to work together, resulting in poor software architecture [37]. While the authors were primarily concerned with maintainability and performance, bugs (and particularly security vulnerability bugs) are an important side effect of inadequate architecture and teamwork process.

Another possibility is that, even with good internal processes and teamwork, no software model or specification can comprehensively account for the variety of environments the software may operate in [38]. If we cannot predict the environment, we cannot predict all the ways that things may go wrong. In fact, research has shown that it appears impossible to model or predict the number of vulnerabilities that may be found through tools like fuzzing—and, by extension, the number of vulnerabilities that exist in a product [39] [40]. The best advice seems to be to assume that vulnerabilities will be found indefinitely into the future and work to ensure that any remaining vulnerabilities cause minimal harm to users and systems.

A successful CVD process helps encourage the search for and reporting of vulnerabilities while minimizing harm to users. Developers supporting a successful CVD process can expect to see the overall security of their code improve over time as vulnerabilities are found and removed.

### 2.6.2 Improving the CVD Process Itself

Feeding lessons learned back into the development process, CVD can

- reduce creation of new vulnerabilities
- increase pre-release testing to find vulnerabilities

Participation in CVD may allow discussions between your developers and security researchers on new tools or methods for vulnerability discovery such as static analysis or fuzzing. These tools and methods can then be evaluated for inclusion in ongoing development processes if they succeed in finding bugs and vulnerabilities in your product. Essentially, CVD can facilitate field testing of new analysis methods for finding bugs.

## 2.7 CVD as a Wicked Problem

Horst W.J. Rittel and Melvin M. Webber, in their 1973 article "Dilemmas in a General Theory of Planning," describe the following characteristics of *wicked problems* [41]:

1. "**There is no definitive formulation of a wicked problem**" – solving the problem is analogous to understanding it.
2. "**Wicked problems have no stopping rule**" – the problem has no intrinsic criteria to indicate that a solution is sufficient; solutions depend rather on the planner deciding to stop planning.
3. "**Solutions to wicked problems are not true-or-false, but good-or-bad**" – the judgement of a solution's fitness by parties involved will be filtered through their values and their predisposed ideology. These judgements are usually "expressed as 'good' or 'bad' or, more likely, as 'better or worse' or 'satisfying' or 'good enough.'"
4. "**There is no immediate and no ultimate test of a solution to a wicked problem**" – solutions can have far-reaching and not always clear effects in both time and scope. Likewise, the desirability of the outcomes may not become clear until much later.
5. "**Every solution to a wicked problem is a 'one-shot operation'; because there is no opportunity to learn by trial-and-error, every attempt counts significantly**" – actions taken in response to the problem affect the options available to future solutions.

6. "**Wicked problems do not have an enumerable (or an exhaustively describable) set of potential solutions, nor is there a well-described set of permissible operations that may be incorporated into the plan**" – it is not possible to demonstrate that all possible solutions have been considered or even identified.
7. "**Every wicked problem is essentially unique**" – The meaning of *essentially unique* is given as "despite long lists of similarities between a current problem and a previous one, there always might be an additional distinguishing property that is of overriding importance."
8. "**Every wicked problem can be considered to be a symptom of another problem**" – every identified cause leads to a "higher level" problem of which the current problem is a symptom. As a result, incremental approaches or marginal improvements may have little or no impact on the problem.
9. "**The existence of a discrepancy representing a wicked problem can be explained in numerous ways. The choice of explanation determines the nature of the problem**'s **resolution**" – The way a problem is described influences the solutions proposed.
10. "**The planner has no right to be wrong**" – The goal of a solution is not to find an ultimate truth about the world, rather it is to improve conditions for those who inhabit it.

We assert that vulnerability disclosure can be thought of as a wicked problem, offering this document as evidence to that effect.

# 3 Roles in CVD

*What people say, what people do, and what they say they do are entirely different things.*

*– Margaret Mead*

Certain roles are critical to the Coordinated Vulnerability Disclosure process, as described below:

- **Finder (Discoverer)** – the individual or organization that identifies the vulnerability
- **Reporter** – the individual or organization that notifies the vendor of the vulnerability
- **Vendor** – the individual or organization that created or maintains the product that is vulnerable
- **Deployer** – the individual or organization that must deploy a patch or take other remediation action
- **Coordinator** – an individual or organization that facilitates the coordinated response process

Although a more detailed description of the CVD process is provided in Section 4, a simple sketch of the relationships between these roles is shown in Figure 1.

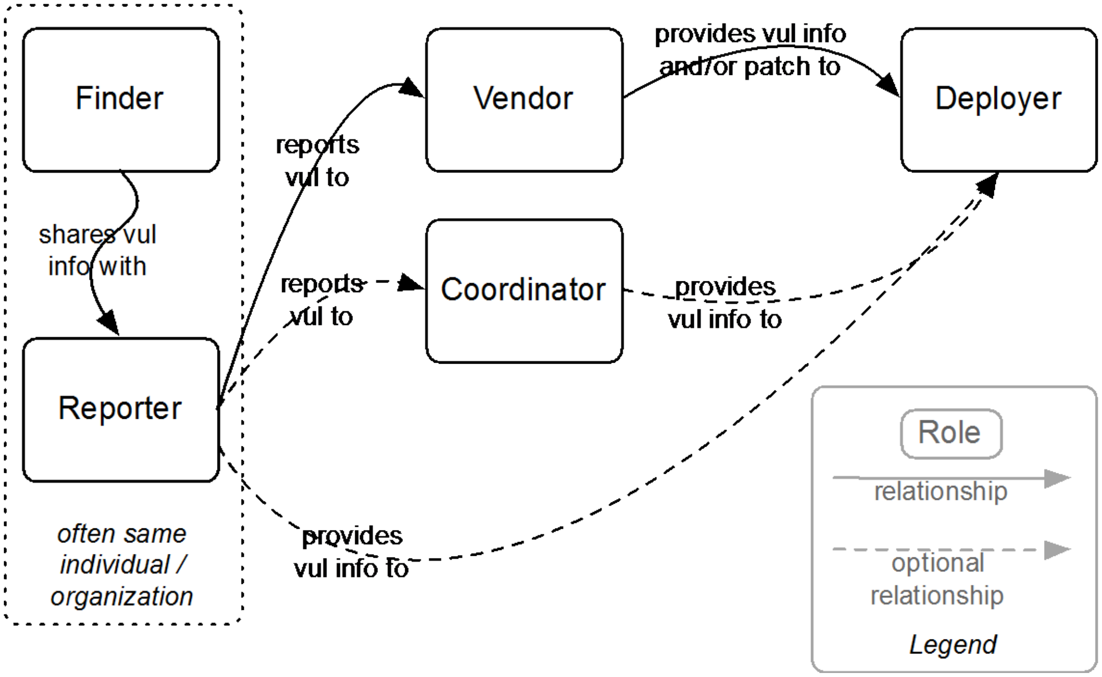

*Figure 1: CVD Role Relationships*

It is possible and often the case that individuals and organizations play multiple roles. For example, a cloud service provider might act as both vendor and deployer, while a researcher might act as both finder and reporter. A vendor may also be both a deployer and a coordinator. In fact, the CERT/CC has played all five roles over time, although not usually simultaneously.

## 3.1 Finder

ISO/IEC 29147 [17] defines a *finder* as an “individual or organization that identifies a potential vulnerability in a product or online service,” noting that “finders can be researchers, security companies, users, governments, or coordinators.” In the interest of consistency, we will use this definition of finder, although in other documentation we’ve used the term discoverer for this same role. We do, however, distinguish between the role of finder and the role of reporter, as seen in this section and the next.

Vulnerabilities can be found by just about anyone. All it takes is for someone to notice an unexpected or surprising behavior of a system. Although it is common for independent security researchers to hunt vulnerabilities as either a hobby or profession, finders need not self-identify as security researchers or hackers. Vulnerabilities have been found by people of many backgrounds:

- students and professional academics studying novel ways to exploit systems or protocols
- open source developers who notice that a software bug has security implications
- system administrators who recognize a vulnerability during the course of troubleshooting a system error
- professional security analysts who observe a previously unknown product vulnerability while testing an organization’s infrastructure during a penetration test engagement
- people using software or web services who mistyped some input or simply clicked on the wrong thing
- children who like to press buttons. Kristoffer Von Hassel, a five-year-old from San Diego discovered a vulnerability in Microsoft’s Xbox Live service just by holding down the space bar and was able to log in to his father’s account without the password [42].

There are also organizations that look for vulnerabilities. Some of them work under contract to vendors directly. Some work for the vendors’ customers who deploy the software. And some have independent motivation to find vulnerabilities, perhaps to demonstrate their competence in finding vulnerabilities in the interest of their security consulting practice’s business development.

Furthermore, vendors may choose to look for vulnerabilities in their own products—a practice that we strongly encourage. This can be done via (a) in-house expertise and testing, (b) contracted security testing, or (c) solicited on a per-vulnerability basis using a bug bounty program. Many vendors integrate testing for vulnerabilities into their development process. Microsoft, for example, includes static, dynamic, and fuzz testing for vulnerabilities in its phases of the Security Development Lifecycle [43]. The BSIMM model suggests that many vendors in various industries already employ techniques in architecture analysis, code review, and security testing to find vulnerabilities as part of their development cycle [44].

Regardless of who finds a vulnerability, there are a few common events that follow the discovery:

1. The finder composes a vulnerability report, as discussed in Section 4.2.4.
2. The finder (or reporter, if these are distinct individuals) provides that report to someone. Often the vulnerability report would be provided to the vendor, but that’s not always the case. Sometimes the report might be sent to a coordinator. If the vulnerability is discovered internally to a vendor, then the report may simply be forwarded to the responsible team within the

organization—for example, filed as a security-related bug report. We cover the coordinator role in Section 3.5. A discussion of the reporting process can be found in Section 4.2.

3. (Optional) Finders, reporters, vendors, or coordinators might prepare a document to publish. The finder often wants to draw attention to his or her discovery and subsequent analysis by publishing a document, blog post, or conference presentation, to share the findings with a larger audience. Vendors typically want to publish a document as well to inform their users that action has been taken to resolve the problem, and to prompt their users to take any required remediation actions. Publishing of vulnerability information is covered in Section 4.5.

It is of course possible for a finder to find a vulnerability and tell no one. However, in that case there is no disclosure involved so we do not address that scenario further in this document.

## 3.2 Reporter

The defining characteristic of vulnerability *reporters* is that they originate the message that informs a vendor or coordinator of a vulnerability. In most cases, the *reporter* is also the *finder* of the vulnerability. However, this is not always the case. For example, the finder might be an employee at an organization that also has in-house vulnerability coordinators who act as the communications liaison with the affected vendor(s).

Alternatively, it could be that someone analyzing a piece of malware realized that it exploited a previously undisclosed vulnerability. In both cases, the party communicating the vulnerability information to the vendor is not the original finder. That said, whether or not the reporter is the original finder is often not as relevant as whether the newly provided information is sufficient to determine the existence and impact of the problem reported.

## 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 5.4.2. 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 [30]:

- 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.

### 3.3.1 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.

### 3.3.2 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 [17]. ISO/IEC 30111 addresses the internal processes associated with vendor vulnerability response [45].

#### 3.3.2.1 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.

### 3.3.3 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.

#### 3.3.3.1 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) [46] and Cisco PSIRT [47] . Many vendor PSIRTs are active in the Forum of Incident Response and Security Teams (FIRST) [48].

#### 3.3.3.2 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.

#### 3.3.3.3 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 5.4.2.

#### 3.3.3.4 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.

## 3.4 Deployer

The *deployer* role refers to the individual or organization responsible for the fielded systems that use or otherwise depend on products with vulnerabilities. Deployers include the following:

- network and cloud infrastructure providers
- <anything>-as-a-service providers
- outsourced IT operations
- in-house IT operations
- individual users

Deployers typically must take some action in response to a vulnerability in a product they've deployed. Most often this means deploying a patch, but it can also involve the application of security controls, such as reconfiguring defensive systems, adding monitoring or detection rules, or applying mitigations.

Automation of the deployment process increases the efficiency of the deployer's response at the same time it decreases the duration of the risk posed by vulnerable systems.

Although the deployer role is primarily concerned with Vulnerability Management practices that sit downstream of CVD, it's worth spending a few moments to understand how it fits in with CVD.

### 3.4.1 Deployer Vulnerability Response Process

A deployer’s vulnerability response process usually involves the following sequence of stages:

- Become aware of vulnerability, mitigation, and/or fix.
- Prioritize the mitigation or fix into existing workload (triage).
- Test the mitigation or fix.
  - Confirm that the fix addresses the problem.
  - Avoid undesirable side effects.
- Identify affected systems and plan the deployment:
  - staged or all-at-once
  - automated or manual
  - scheduled update window or out-of-band
- Deploy the mitigation or fix to affected systems.

We cover each of these in more detail below.

#### 3.4.1.1 Become Aware

In order to take action, a deployer must know about the vulnerability and have sufficient information to act on. Most often this information originates from the product vendor. However, since not all vulnerability reports are coordinated with the vendor for disclosure, vulnerability information can arrive from other sources as well.

Deployers should be on the lookout for and pay attention to

- vendor security notices
- vendor customer support notices (not all vendors provide separate security notices, nor are all vulnerabilities always explicitly called out in update notes)
- vulnerability and threat intelligence services
- security discussions online including social media
- mass media coverage of vulnerabilities

#### 3.4.1.2 Prioritize Response

Deployers have many responsibilities beyond deploying patches. As a result, they need to prioritize their work and integrate patch deployment into their normal operations cycle. That might mean testing, scheduling out-of-band fixes, or planning for scheduled maintenance windows. Just as vendors need to triage reports in order to prioritize patch development appropriately, deployers must decide which patches and mitigations to deploy and when to deploy them. The deployer’s workload often makes it difficult to patch all the things as quickly as they would like.

#### 3.4.1.3 Test the Solution

Testing prior to deployment is important if either of the following conditions is true:

- The system’s availability and performance are critical.

- Reverting a patch deployment gone bad is difficult.

In environments with efficient automated deployment and rollback capabilities, it may not be as necessary to test as heavily. But that’s often an ideal scenario that few deployers find themselves in. Staged deployments or rollouts can be a significant help here—where some portion of the affected systems are updated to confirm the fix prior to wider rollout—allowing deployers to balance patch deployment with the risk of negative side effects.

#### 3.4.1.4 Plan the Deployment

Deployers have many options when it comes to planning to deploy a patch or mitigation. Highly automated environments can dramatically shorten the time required to complete these stages, but the functions described here will usually still occur regardless of the deployer’s automated patching capability.

Planning for a patch deployment requires two major steps:

1. Identify and enumerate system instances affected by the vulnerability. Vulnerability management tools can be used to scan for affected systems and prioritize patch deployment. Information about affected hosts helps to define the scale of the patching effort required.
2. Set the deployment schedule. If there are relatively few systems under management and vulnerabilities are fairly rare, a first-in-first-out process might suffice. Larger enterprises often have scheduled maintenance windows during which they can deploy most patches. Alternatively, an organization might choose to push out a patch outside of a scheduled maintenance window, especially in cases where a vulnerability is being actively exploited or significant harm is expected should the vulnerability remain unpatched until the next maintenance window. Essentially the question boils down to deploy now or defer to later?

#### 3.4.1.5 Execute the Plan

Obviously, it is important to actually carry out the deployment of the mitigation or fix. Automated patch deployment tools can make this process quite efficient. Regardless of the degree of automation of patch deployment, recurring or continuous monitoring for vulnerabilities can help measure the success of the deployment effort.

## 3.5 Coordinator

Complicated or complex CVD cases can often benefit from the help of a coordinator. A coordinator acts as a relay or information broker between other stakeholders. Several types of coordinators with slightly different roles and domains exist. We list a few here.

### 3.5.1 Computer Security Incident Response Team (CSIRT)

A Computer Security Incident Response Team (CSIRT) is a service organization that is responsible for receiving, reviewing, and responding to computer security incident reports and activity. Their services are usually performed for a defined constituency that could be a parent entity such as a corporate, governmental, or educational organization; a region or country; a research network; or a paid client. A CSIRT can be a formalized team or an ad-hoc team. A formalized team performs incident response work as its major job function. An ad-hoc team is called together during an ongoing computer security incident or to respond to an incident when the need arises [49].

### 3.5.2 CSIRT with National Responsibility

CSIRTs with National Responsibility, also known as *National CSIRTs*, are designated by a country or economy to have specific responsibilities in cyber protection for the country or economy. A National CSIRT can be inside or outside of government, but must be specifically recognized by the government as having responsibility in the country or economy [50]. In addition to functioning as a clearing house for incident response across government departments and agencies, CSIRTs with National Responsibility often have some degree of responsibility or oversight for coordinating vulnerability response across their nation's critical infrastructure. US-CERT, part of the Department of Homeland Security, has been designated as the national CSIRT for the United States. We maintain a list of National CSIRTS on the CERT website [51].

### 3.5.3 Product Security Incident Response Team (PSIRT)

Over time, Product Security Incident Response Teams (PSIRTs) have emerged as a specialized form of CSIRT, allowing vendors to focus their response to product security issues. Although not all vendors have dedicated PSIRTs, vulnerability response is sufficiently different from security incident response that larger vendor organizations can usually justify having a distinct function to deal with it. PSIRTs usually provide an interface to the outside world to receive vulnerability reports as well as serving as a central coordinator between internal departments for the organization's vulnerability response for its products. When reporting a vulnerability to a vendor, the reporter will usually be communicating with the vendor's PSIRT. For example, Cisco, Oracle, Intel, Microsoft, Apple, Adobe, and others have established internal PSIRTs. Many PSIRTs participate in the Forum for Incident Response and Security Teams [48].

### 3.5.4 Security Research Organizations

Organizations that perform security research on other vendors' products in the course of their own business sometimes establish their own coordination capability in order to handle the disclosure process for the vulnerabilities they find. A wide variety of organizations perform this kind of security research, whether for profit or for non-commercial reasons. Some examples include managed security service providers, government agencies, and academic research teams. Some of these organizations are vendors of products or services themselves, and combine their PSIRT's vulnerability response capability with their externally facing coordination capability.

Furthermore, organizations that provide vulnerability management and scanning tools and services are often well-positioned to act as a disclosure coordinator for the vulnerabilities their products detect. This applies especially when those vulnerabilities have not already been disclosed to either the vendor or the public. Alternatively, organizations such as these may choose to partner with another coordinating organization in order to promote transparency and reduce the perception of bias in their vulnerability disclosure process.

### 3.5.5 Bug Bounties and Commercial Brokers

In recent years, a new class of coordinator has emerged in the form of commercial bug bounty program providers. Many individual vendors have established programs to compensate security researchers for their efforts in discovering vulnerabilities in the vendor's products. Creation of a bug bounty program has been noted as an indicator of maturity in vendors' vulnerability response efforts. In some cases, vendor bug bounty programs are enabled by other companies that provide tools and services to facilitate vulnerability coordination. Companies such as BugCrowd [52], HackerOne [53], Synack [54], and Cobalt [55] offer turnkey solutions for vendors who want to bootstrap their own vulnerability response program.

While bug bounty programs help address the vulnerability coordination needs of individual vendors, there still are vulnerabilities that require larger scale coordination. In particular, multivendor coordination remains a challenge for many organizations. As individual vendors have become more mature in their handling of vulnerabilities in their products, the role of multivendor coordination has increased in importance for more traditional vulnerability coordinators such as the CERT/CC [56], NCSC-NL [57], NCSC-FI [58], and JPCERT/CC [59].

### 3.5.6 Information Sharing and Analysis Organizations (ISAOs) and Centers (ISACs)

Information Sharing and Analysis Organizations (ISAOs) and Centers (ISACs) are non-government entities that serve various roles in gathering, analyzing, and disseminating critical infrastructure cybersecurity information across private sector organizations of various sizes and capabilities [60] [61]. These organizations have only begun to emerge in earnest within the past few years, but they are already actively involved in the coordination and deployment of vulnerability mitigations. Furthermore, it seems likely that some number of critical infrastructure sectors will need to become involved further in the coordination of the vulnerability discovery, disclosure, and remediation processes.

### 3.5.7 Reasons to Engage a Coordinator

There are a number of reasons that a finder, reporter, or vendor may wish to engage a third-party coordinator to assist with the CVD process.

#### 3.5.7.1 Reporter Inexperience

Novice reporters sometimes request assistance from coordinators to increase the chances of a successful resolution to the vulnerability they have found. Working with a coordinator for the first few cases can help develop a reporter’s knowledge of the CVD process. From the coordinator’s perspective, working with novice reporters serves to transfer knowledge of CVD to the security research community, thereby improving vulnerability response overall. We have found that novice reporters usually learn quickly and are willing to do most of the coordination effort themselves, but just need occasional advice on how the process should work.

#### 3.5.7.2 Reporter Capacity

Seeing a CVD case through to resolution can at times be a protracted process. Not all reporters have the time or resources to follow up on vulnerabilities they’ve reported. In such situations, a coordinator can help by offloading some of the effort. However, coordinators are often limited in their capacity as well, and must accordingly prioritize the cases they choose to take on. As a result, coordinators and reporters alike should take care to set clear expectations with each other as to what roles they expect to play in any given coordination case.

#### 3.5.7.3 Multiple Vendors Involved

At its most effective, CVD follows the supply chain affected by the vulnerability. As a mental model, it can be useful to think of the supply chain as horizontal or vertical. A horizontal supply chain implies that many vendors need to independently make changes to their products in order to fix a vulnerability. A vertical supply chain implies that one vendor might originate the fix, but many other vendors may need to update their products after the original fix is available. Software libraries tend to have vertical supply chains. Protocol implementations often have horizontal supply chains.

We discuss horizontal and vertical supply chains in Section 5.4.2 below.

#### 3.5.7.4 CVD Disputes

Occasionally vendors and reporters have difficulty arriving at a mutually acceptable response to the existence of a vulnerability. Disputes can arise for many reasons, including the following:

- whether the behavior described in the report is reproducible
- whether the behavior described in the report has security implications
- the impact of the vulnerability to deployed systems
- whether to publicly disclose the vulnerability
- how much detail to include in a public disclosure
- the timing of public disclosure
- whether extensions should be made to deadlines set by one party or another, whether or not they have been mutually agreed to previously

In these situations, and many others, reporters and/or vendors may find it useful to engage the services of a third-party coordinator to assist with conflict resolution. Drawing on the experience and relative neutrality of a third-party coordinator can often dissipate some of the potential animosity that can arise in contentious cases.

#### 3.5.7.5 Major Infrastructure Impacts

In situations where a vulnerability has the potential for major impact to critical infrastructure, it may be necessary to coordinate not only with vendors to fix the vulnerable products, but also with major deployers. The primary concern in these cases is to ensure that internet and other critical infrastructure remains available so that deployers and other network defenders can acquire and deploy the necessary information and patches.

Luckily this scenario is rare, but we have seen it come up in cases affecting internet routing, the Domain Name System (DNS), internet protocols, and the like. Vulnerabilities that affect basic Internet services such as DNS (which also serves as an example of a horizontal supply chain) affect a massive number of vendors; a coordinator can help contact and disseminate information to vendors, service providers, and other critical organizations for quick remediation.

## 3.6 Other Roles and Variations

There can be other roles in the CVD process too, but they tend to be subordinate to the ones already described. We discuss a few of them here.

### 3.6.1 Users

Individual users of vulnerable products overlap with deployers as described above. In the case where the user must trigger an update or install a patch, the user is playing the role of a deployer. In other cases, the user depends on another deployer (e.g., the user’s IT support staff or an app store’s automatic update capability). In these latter cases the user does not play as active a role in the vulnerability response process.

### 3.6.2 Integrator

System integrators most often can be considered as playing the deployer role; however, depending on their contractual responsibilities and business relationships, they may also play roles as vendors or even coordinators in some cases.

### 3.6.3 Cloud and Application Service Providers

Insofar as cloud-based services are built on traditional computing platforms, cloud service providers can be considered deployers as we’ve described above. However, as cloud-based services (e.g., software, platform, and infrastructure as a service) have risen to prominence, they have also distinguished themselves from traditional software vendors in that their development, deployment, and delivery processes for security fixes tend to be much more direct.

For many cloud providers, the number of distinct instances of their software is quite limited, and control is centralized, so there are fewer independent decision makers in the path from vulnerability report to patch deployment.

Furthermore, the prevalence of DevOps practices among such providers means that the time from code commit to last vulnerable system patched can sometimes be measured in minutes. To be sure, development and delivery processes in traditional software environments have accelerated considerably as well, but the fact that cloud service providers have direct control over the vulnerable systems makes a significant difference in their ability to mitigate vulnerabilities across all their users in short order.

### 3.6.4 Internet of Things

Another class of vendors are the purveyors of Internet of Things (IoT) products. The physicality of IoT products and services often places them on the opposite end of the deployment spectrum from cloud-based services.

Unlike most other devices (laptops, PCs, smartphones, tablets), many of today’s IoT products are either non-updateable or require significant effort to update. Whether we’re talking about cars, televisions, medical devices, airplanes, sensors, home automation, or industrial control systems, too often today the patch deployment process involves going out and physically touching the thing that must be updated. Systems that cannot be updated become less secure over time as new vulnerabilities are found and novel attack techniques emerge. Because vulnerabilities are often discovered long after a system has been delivered, systems that lack facilities for secure updates once deployed present a long-term risk to the networks in which they reside. This design flaw is perhaps the most significant one already found in many IoT products, and if not corrected across the board, could lead to years if not decades of increasingly insecure devices acting as reservoirs of infection or as platforms for lateral movement by adversaries of all types. Patch deployment will likely improve as more connected things get over-the-air (OTA) update capabilities, but there is already a large installed base of systems lacking such features.

Furthermore, systems at the lower end of the price range might have "fire and forget" assumptions built into their pricing model, meaning that there is neither the technical means to deliver updates nor the support capability in place to even develop them in the first place. In the long run, regulatory intervention may influence IoT vendors to improve their vulnerability response capabilities, but the gap today is large and will likely be difficult to close entirely unless market incentives shift toward more holistic and improved security posture.

Another issue with IoT devices is their supply chain, whereby the vendor of the final product actually has very little to do with the hardware, firmware, or software development of the product it sells. We frequently observe pervasive use of third-party libraries in integrated products with neither recognition of nor adequate planning for how to fix or mitigate the vulnerabilities they inevitably contain. When developers embed a library into their product, that product often inherits vulnerabilities subsequently found in the incorporated code. Although the third-party library problem is equally pervasive in the traditional computing, cloud, and mobile worlds, it is even more concerning in contexts where many libraries wind up as binary blobs and are simply included in the firmware as such. Lacking the ability to analyze this black box code either in manual source code reviews or using most code analysis tools, IoT vendors may find it difficult to examine and improve the code's security.

### 3.6.5 Mobile Platforms and Applications

Mobile devices present yet another class of stakeholders that has grown distinct in recent years. The device vendors themselves are most akin to IoT vendors, but app developers can be quite a diverse bunch, ranging from very large traditional software companies, to cloud service providers, to novices with a good idea and a few hours of coding. Perhaps the most significant outstanding issue is that many mobile devices have multi-stage, vertical supply chains, each step of which can stand in the way of security updates reaching their intended beneficiaries (i.e., the users) [62]. In both the mobile and IoT spaces, high-viscosity supply chains are bad for end-user security.

### 3.6.6 Governments

Governments are multifaceted stakeholders in regards to cybersecurity vulnerabilities and their disclosure. While they have always had a role as owners and operators of vulnerable networks and systems, issues surrounding vulnerability discovery, coordination, disclosure, and mitigation have become increasingly important to governments worldwide.

As the industries they regulate move toward increasing connectivity, agencies with oversight responsibilities will likely see an increased demand to extend their safety monitoring to include security issues (especially for security issues that directly impact safety). To that end, changes are happening rapidly on multiple fronts. For example, in the United States recent developments include the following: The FDA Medical Device Reporting process enables oversight and detection of potential device-related safety issues [63]. The National Highway Transportation and Safety Commission (NHTSA) collects reports of vehicle safety issues, which helps to drive its investigation and recall processes [64]. The FAA offers a number of safety reporting capabilities as well [65].

Beyond just documenting observed issues, some government agencies take an active learning approach when broader engineering failures occur. The aforementioned FDA and NHTSA reporting programs serve this purpose, but other programs exist as well. For example, the National Transportation Safety Board is explicitly tasked with investigating transportation accidents, and NASA collects lessons learned in a public database [66]. This kind of continuous improvement process has demonstrated its effectiveness in a variety of environments and seems to provide a good model for cybersecurity vulnerabilities in both the private and public sectors.

The United States is not alone in realizing that vulnerability discovery, disclosure, and remediation is important to national interests. These cybersecurity issues have been global for quite some time. The EU Parliament recently held hearings on modernizing export controls and the trade in zero-day vulnerabilities [67]. Meanwhile, a quick glance at the vulnerability database catalog being developed by the FIRST gives a good indication of the international interest in this problem space [68].

# 4 Phases of CVD

*You go through phases. You have to reinvent reasons for playing, and one year's answer might not do for another.*

*-Yo-Yo Ma*

There are a number of proposed models of the CVD process that have slightly varying phases [17] [18] [45] [69]. Below, we adapt a version of the ISO/IEC 30111 [45] process with more phases to better describe what we have seen at the CERT/CC.

- **Discovery** – A researcher (not necessarily an academic one) discovers a vulnerability by using one of numerous tools and processes.
- **Reporting** – A researcher submits a vulnerability report to a software or product vendor, or a third-party coordinator if necessary.
- **Validation and Triage** – The analyst validates the report to ensure accuracy before action can be taken and prioritizes reports relative to others.
- **Remediation** – A remediation plan (ideally a software patch, but could also be other mechanisms) is developed and tested.
- **Public Awareness** – The vulnerability and its remediation plan is disclosed to the public.
- **Deployment** – The remediation is applied to deployed systems.

A mapping of CVD phases to CVD roles is provided in Table 2.

*Table 2: Mapping CVD Roles to Phases*

| Roles → Phases | Finder | Reporter | Vendor | Coordinator | Deployer |
|---|---|---|---|---|---|
| **Discovery** | Finds vulnerabilities | | | | |
| **Reporting** | Prepares report | Reports vuls to vendor(s) and/or coordinators | Receives reports | Receives reports<br>Acts as reporter proxy | |
| **Validation and Triage** | | | Validates reports received<br>Prioritizes report for response | Validates reports received<br>Prioritizes report for response | |
| **Remediation** | | Confirms fix | Prepares patches<br>Develops advice, workarounds | Coordinates multiparty response<br>Develops advice, workarounds | |
| **Public Awareness** | Publishes report | Publishes report | Publishes report | Publishes report | Receives report |
| **Deployment** | | | | | Deploys fix or mitigation |

We will next discuss each of these phases in more detail.

## 4.1 Discovery

> *Reports that say that something hasn't happened are always interesting to me, because as we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there*

*are also unknown unknowns – the ones we don’t know we don’t know.*
*– Donald Rumsfeld*

Aside from the simplest applications, software development is difficult, complex, and prone to error. As a result, the likelihood that any given software-based product or component is free of vulnerabilities is extremely low. For vendors, this implies the need to create a response capability to handle vulnerability reports, whether those reports come from sources internal or external to the vendor.

### 4.1.1 Why Look for Vulnerabilities?

Ultimately, we can’t fix vulnerabilities we don’t know about. While software engineering best-practices, code audits, testing (including fuzzing), and application security testing are important parts of the development lifecycle, security research is important for rooting out hidden vulnerabilities. Some organizations may have in-house expertise to find and identify security vulnerabilities, but most will not. For some vendors, encouraging independent vulnerability finders may be the only way to stay on top of the latest trends in vulnerability research and exploitation techniques.

Many organizations hire application security testers or code auditors to look for vulnerabilities. While such testing is certainly important and commendable, it is important to understand that absence of evidence is not always evidence of absence. Rumsfeld’s point about unknown unknowns applies here. A clean audit or pen test report should not be taken as evidence that the software is free of vulnerabilities. All software-based systems have problems we’re not even aware of and so we don’t even know to look for them. Because such vulnerabilities may exist and can be exploited without warning, vendors and deployers should establish their VR capability in preparation for this eventuality.

### 4.1.2 Avoid Unnecessary Risk in Finding Vulnerabilities

Finders should exercise an appropriate degree of care when performing vulnerability research. This will help to alleviate legal concerns and limit the potential for damage to others.

Vulnerability research should of course be performed on equipment that the finder is authorized to use for the purpose. If the research is performed on behalf of an organization such as a private security firm or university, permission should be obtained before attempting research on organization-owned equipment.

Likewise, organizations should make the rules and process for obtaining permission very clear and easy to find. For example, a form or email address provided on an intranet page might be sufficient. Employees hired specifically to find vulnerabilities should be briefed on necessary rules and provided with concrete permission as part of the on-boarding process. Failure to adequately document permissible scope and authority for vulnerability testing can lead to frustration and other negative consequences with various legal ramifications.

#### 4.1.2.1 Operational Risk

In general, the software or devices tested should not be production systems that support or have access to real data or users. When possible, dedicated, controlled testing environments should be established. Such a testing environment often consists of virtual machines (VMs) in a virtual network firewalled off from any production network. Even as a finder in a controlled testing scenario, you should keep in mind the potential for unintended consequences (i.e., the unknown unknowns). Always try to limit the potential for unintended negative impact of testing, even within your controlled environment. If the impact cannot be constrained to a controlled environment with relatively known consequences, do not attempt to test your exploit and instead report your findings directly to the vendor.

#### 4.1.2.2 Safety Risk

Safety-critical systems have been defined as "systems whose failure could result in loss of life, significant property damage, or damage to the environment [70]." A high degree of caution is both appropriate and necessary when testing the security of safety-critical systems, such as medical devices, industrial equipment, or vehicles. A proof of concept exploit to demonstrate a vulnerability on a traditional computer might cause a calculator to pop up on the screen. A proof of concept exploit on a car might cause it to behave erratically, potentially leading to injury or death. Testing or demonstrating safety-critical systems outside a controlled environment, or when there is any chance of harming unwitting bystanders is unacceptable under any circumstances.

#### 4.1.2.3 Legal Risk

Depending on the circumstances, finders may be subject to a non-disclosure agreement (NDA) regarding any vulnerabilities found. This is often the case when vulnerability testing is performed on behalf the vendor whether directly as an employee, or under contract as part of a consulting firm or as a freelance consultant. Finders should be aware of this possibility and consider the legal implications of any relevant NDAs before reporting a vulnerability to any third party.

That said, vendors are strongly encouraged to avoid requiring NDAs of reporters if at all possible. Many finders prefer to avoid the legal entanglements that NDAs entail and will be discouraged from reporting vulnerabilities when an NDA is involved.  This can leave vendors unaware of potential threats to their products and services and in turn, their users.

Additionally, in some environments, such as medical devices, healthcare, education, or financial information systems, there may be legal consequences to accessing real data (under HIPAA [71], FERPA [72], COPPA [73], and similar laws, industry standards such as PCI DSS [74], etc.), so we again reiterate the need to perform research only in controlled test environments, preferably with fake data.

For more information on the legal implications of vulnerability disclosure, we refer you to the EFF's *Coders' Rights Project Vulnerability Reporting FAQ* [75].

## 4.2 Reporting

> *No matter who you are, most of the smartest people work for someone else.*
>
> *– Bill Joy, Sun Microsystems*

Vendors need a mechanism to receive vulnerability reports from others. This reporting mechanism should be easy enough to use that it encourages rather than discourages reports. It can be as simple as a dedicated email address for reporting security issues, a secure web form, or a bug bounty program.

### 4.2.1 Create Secure Channels for Reporting

Whether you are a vendor or a coordinator, you need to have open channels for communication with vulnerability finders and reporters. In our experience, the most common means of communication is email. For this reason, the CERT/CC recommends that vendors establish a specific and well-publicized email alias such as <security@example.com> solely for receipt of vulnerability reports.

However, since email is an insecure communications channel by default, many vendors, reporters, and coordinators prefer to use encrypted mail instead. Although x.509 encrypted mail exists, we have found PGP-compatible tools such as GnuPG to be more widely used by CVD participants. Vendors are encouraged to create and publish a PGP key affiliated with the security email alias to allow the confidentiality of sensitive reports to be maintained in transit.

Alternatively, some vendors choose to offer a web form specifically for receiving reports of security-related issues. Such forms can then deliver the report directly to your security or engineering team. The CERT/CC discourages reliance on general “Contact Us” web forms that pass through an organization’s communications or customer support teams. Many finders will balk at having to get past these nontechnical interfaces into the vendor. In addition, security messages often must be triaged and processed differently than other incoming contacts.

Another possibility is to make use of a third-party bug bounty or coordination platform. For more information on common CVD tools, see Section 7.

### 4.2.2 Encourage Reporting

Anyone who becomes aware of a vulnerability that does not appear to have been remediated should report the vulnerability to the vendor. One should not assume that a vendor is aware of a specific vulnerability unless it has already been publicly reported, whether by the vendor or elsewhere. The easier it is to report vulnerabilities to a vendor, the less likely that the vendor will be surprised by vulnerability reports disclosed directly to the public.

Aside from the technical aspects of encouraging reporting, vendors can also provide reporters with other incentives, as discussed in Section 2.4.

### 4.2.3 Reduce Friction in the Reporting Process

As a vendor, it is important to not treat reporters with suspicion or hostility. It’s likely they have important information about your product, and they want to share it with you.

Furthermore, vendors should take care not to frustrate reporters’ attempts to make contact by building too many assumptions into their CVD process. In the course of our own vulnerability coordination efforts, we have observed all of the following erroneous assumptions built into vendor’s CVD process:

- *The reporter is always a customer or has a customer ID.* – At the CERT/CC, we have hit walls in our communication attempts when a vendor’s technical support function refuses to help us without a customer ID number. Be sure your tech support staff understand how to forward vulnerability reports to the appropriate individuals or groups within the organization. Vulnerability reports can arrive from anyone.

- *The reporter is willing and/or able to fill out a web form.* – Some reporters prefer to use anonymous email; be sure you have more communication lines open than just a web form.
- *The reporter is a human.* – Sometimes reports can be auto-generated by tools. Include a clearly defined reporting format for tools if at all possible.
- *The reporter can send or receive encrypted mail.* – The CERT/CC encourages encrypted mail when possible, but it is not appropriate to presume all reporters can or must use encrypted mail. If the reporter declines to use encrypted mail, offer other options. These may include encrypted zip files or a company upload service such as FTP.
- *The reporter has an account on your private portal.* – The reporter may not be a customer with a portal account; furthermore, the reporter may wish to remain anonymous and will be unwilling to register for a portal account. Again, be sure it is easy for reporters to find more than one communication channel.
- *The reporter will wait indefinitely for your reply before communicating with others about what they know.* – Technology sometimes fails, and we wonder if a message was received. It is helpful to let the reporter know as soon as possible that the report was received. Give regular updates on the process so that the reporter is involved and there is mutual understanding. If reporters are kept out of the loop, they may seek out a third-party coordinator or even publish their report without notice.
- *The reporter will keep your correspondence private.* – Lack of response or rudeness on the part of a vendor may result in the reporter choosing to post the correspondence publicly. In addition to the negative attention this draws to the vendor and reporter alike, such negative experiences may discourage finders and reporters from reporting vulnerabilities to the vendor in the future.

### 4.2.4 Providing Useful Information

Reporting a vulnerability requires that the vulnerability is well-documented. This typically means providing the following information:

- the exact product version(s) affected
- a description of how the vulnerability was discovered (including what tools were used) or what you were doing when you encountered the vulnerability
- proof of concept (PoC) code or reproduction instructions demonstrating how the vulnerability might be exploited
- ideally, a suggested patch or remediation action if the reporter is aware of how to fix the vulnerability
- description of the impact of the vulnerability and attack scenario (Kymberlee Price discusses the importance of providing a clear attack scenario in her article [76].)
- any time constraints (for example, give a date of publication or presentation at a conference if you know)

Reporters that do not provide enough information to a vendor or coordinator may find their reports delayed or even rejected. Using CWE [77] or CAPEC [78] as a reference might be helpful to describe the type of vulnerability you have found and common ways to fix it the problem

An example of a template for a vulnerability report, based on the CERT/CC's own Vulnerability Reporting Form (VRF) [79], is provided in Appendix D.

Vendors that require additional information to validate reports should clearly document their specific requirements in their vulnerability disclosure policy, reporting form, or process description.

## 4.3 Validation and Triage

*Extraordinary claims require extraordinary evidence.*

*– Carl Sagan*

When a vendor or coordinator receives a vulnerability report, it's usually necessary to prioritize it along with other vulnerability reports already in progress, new feature development, and possibly other non-security bug fixes. As a result, there are a few considerations to be made in dealing with incoming reports.

### 4.3.1 Validating Reports

Vulnerability reports received from potentially unknown sources may hold inaccurate information. One of the first tasks for the receiver of a report is to analyze the report's validity. A vulnerability report is basically an assertion that some set of conditions exists that permits an adversary to take some action in support of a goal. But just because it was reported doesn't make it true. Replication of the salient claims made in the report is an important step in the case handling process.

#### 4.3.1.1 Recognizing High-Quality Reports

Not all reports are actionable. Some reports may under-specify the problem, making it difficult or impossible to reproduce. Some may contain irrelevant details. Some will be well written and concise, but many will not. Some reports could describe problems that are already known or for which a fix is already in the pipeline.

In easy cases, a simple description of the vulnerability, a screenshot, or a copy/pasted snippet of code is all that is necessary to validate that a report is likely accurate. In more complex scenarios, stronger evidence and/or more effort might be required to confirm the vulnerability. Responsive vendors should ensure analysts have access to appropriate resources to test and validate bugs, such as virtual machines (VMs), a testing network, and debuggers.

It may be that reproducing the vulnerability is beyond the capability or time available by the first-tier recipient at the vendor. Most often this occurs when the conditions required to exploit the vulnerability are difficult to reproduce in a test environment. In this case, the triager can weigh the reputation of the reporter against the claims being made in the report and the impact if they were to be true. You don't want to dismiss a report of a serious vulnerability just because it is unexpected. A reporter with a high reputation might give weight to an otherwise low-quality report (although in our experience finders and reporters with a high reputation tend to have earned that reputation by submitting high-quality reports).

If there is difficulty in reproducing the vulnerability, follow up with the reporter promptly and courteously; be sure to be specific about what you tried so that the reporter can provide effective advice. In some cases, a report might be placed in a holding pattern while additional information is requested from the reporter.

The possibility also exists that someone could be sending you reports to waste your time, or erroneously believes the report is much more serious than your analysis suggests. Not all reports you receive warrant your attention. It is usually reasonable to decline reports if you provide the reporter with a summary of your analysis and the ability to appeal (presumably by providing the needed clarifying information).

Reporters should review Section 4.2 to ensure the report contains enough details for the recipient to verify and reproduce a vulnerability. Be as specific as you can. Vendors that follow up with questions are doing the right thing, and attempting to validate your report; be friendly and courteous and attempt to provide as much detail and help as you can.

### 4.3.2 Triage Heuristics

Even for the reports a vendor accepts as legitimate and worthwhile, it is likely that the development team does not have time to address every report at the moment it arrives. Thus, if a report is found to be valid, the next question is how to allocate resources to the report. Most often this requires some measure of how severe the vulnerability is. In some scenarios, the vulnerability may be a critical flaw that requires immediate action, while other cases might indicate a very rare and hard-to-exploit vulnerability that should be given a low priority.

There are a number of heuristics for evaluating the severity of vulnerabilities. Perhaps the most commonly known of these is the Common Vulnerability Scoring System (CVSS) [80]. This system allows a short standard description of the impact of a vulnerability and can be mapped to a score between 1.0 and 10.0 to help prioritization. A related but different metric is the Common Weakness Scoring System (CWSS) [81]. Whereas CVSS addresses the detailed impact of a specific vulnerability, CWSS can be used to evaluate the impact of a class of weaknesses. While scoring systems like CVSS and CWSS can be useful at establishing relative severity among reports, care must be taken in their use since scores do not always map well onto a vendor's or deployer's priorities.

Vendors should ensure their analysts are trained in the chosen heuristic and understand its strengths and weaknesses so that its result can be overridden when necessary. We do not, for example, recommend blind adherence to hard cutoffs such as "We only bother with reports that have a CVSS score greater than 7.0." No vulnerability scoring system is so precise. Ideally, whatever prioritization scheme is used should also be made transparent to reporters so that the process is understood by all stakeholders. Transparency in this part of the process can help prevent frustration and confusion when reporter and vendor disagree on severity of a vulnerability.

## 4.4 Remediation

A vulnerability can be remediated by providing a fix. Alternatively, mitigation instructions can often be developed as an interim solution.

Once the scope of the vulnerability has been adequately ascertained, it's time to prepare and test the fix (patch). The sequence of tasks tends to include identifying and isolating the vulnerability in the code; changing the code to eliminate the vulnerability; testing the changes, packaging the changes for distribution; and distributing the updated product. While the details of how to do this are often specific to the product as well as the vendor organization and its development process and are thus outside the scope of this document, we review each step in this section.

### 4.4.1 Isolating the Problem

Once a vulnerability report has been received and validated, it must be added into the work queue for the development team to isolate the underlying problem. This often requires input from developers knowledgeable in the code in order to precisely define the problem and understand its impacts.

Often a report will describe a single path to exploit a vulnerability, yet there may be other ways for an attacker to reach the same code and exploit it a different way. This makes it necessary for the developers and analysts responsible for fixing the code to explore the potential for other avenues of exploitation before zeroing in on the specific conditions found in the original report.

There is also the risk that the vulnerable code or component has been reused elsewhere. We have encountered vulnerabilities in multiple codebases that arose because a single developer worked on each project, copying the same vulnerable code into each of them.

### 4.4.2 Fix the Problem

Reporters and finders should recognize that developing, testing, and preparation of patches for deployment often requires some time. A vendor acting in good faith to ferret out the vulnerability and fix it thoroughly should usually be granted some leeway. A well-tested patch that is issued later is preferable to a prematurely released patch that creates further problems. We encourage finders, reporters, and vendors to communicate expectations *early* and *often* with respect to the status of the fix creation process as long as a vendor is responsive.

### 4.4.3 Mitigate What You Cannot Fix

In most cases, knowledge of a vulnerability leads the vendor to create and issue a fix to correct it. As a stopgap in scenarios where it may not possible to develop a timely fix, a vendor or third party will sometimes provide advice on actions that can be taken to mitigate the effects of the vulnerability. Sometimes this advice is as simple as instructions for disabling the affected features of the system. In other cases, mitigation advice might include detailed configuration changes to be made by deployers. However, in nearly all cases a full fix for the vulnerability is preferable to mitigation advice, which should at best be treated as a temporary solution to the problem.

## 4.5 Gaining Public Awareness

For defenders, deploying patches requires effort and is often avoided unless there is sufficient justification. Therefore it is important to provide at least a brief description of the vulnerability in the release notes for the updated code. Knowledge of the existence of a vulnerability is often the key driver causing patches to be deployed. A *silent fix*[2] is far less likely to be widely deployed than one that is clearly described.

Even within the CVD process, there are still many decisions to be made about the disclosure process. The audience, timing, and amount of information released can vary because of a number of factors. These choices might even vary from report to report, depending on the impact and consequences of a particular vulnerability.

Generally, finders, reporters, vendors, and coordinators should consider the following questions in establishing their policies and practices:

- *Should you disclose at all?* – Generally, the answer will be yes, but there may be factors that influence this decision. For example, some vulnerabilities, if exploited, could place lives in danger or cause severe socioeconomic harm. As a result, it may be prudent for reports of such vulnerabilities to be held indefinitely until the population of vulnerable systems has been reduced through patching or obsolescence. However, any decision to defer disclosure should be considered provisional or contingent on the continued absence of evidence of exploitation or adversarial knowledge of the vulnerability.
- *What information will you disclose?* – For example, will you publish all information about the vulnerability, including proof of concept code, or will you only release a brief description of the problem and a remediation? Generally speaking, there is a minimum amount of information required in order for a vulnerability report to be useful. Recall that the point of disclosure is to provoke some action, most often by deployers or any downstream vendors who were not already involved in the coordination process. If the details provided to the recipient are insufficient to cause that action to be taken, the disclosure process will not succeed.
- *To whom will you disclose?* – In most cases, the disclosure should be made publicly. However, in some scenarios the disclosure may be to a specific limited group. For example, if the pool of users is small and the vendor reaches out to every impacted user, a public disclosure may be unnecessary.
- *Via what channel(s) will you disclose?* – Will the vulnerability information be published on the vendor’s website? The reporter’s blog? BugTraq [82], Full Disclosure [83], or other mailing lists? Will you draw attention to it on social media? There are pros and cons to most of these options that must be weighed. When available, an organization’s communications or public relations groups should be involved in planning for the disclosure process. While it is usually neither possible nor practical to have every CVD case flow through them, leveraging their expertise in planning and developing the CVD capability can improve the process considerably.

[2] Where the vendor knows about and fixes the vulnerability but fails to mention the vulnerability’s existence in the subsequent release notes accompanying the new version.

- *What is your expectation of others in disclosing further (or not)?* – Be sure to discuss your expectations with all stakeholders and be prepared to negotiate.

Vendors in particular will need to address three main questions in providing vulnerability and fix information to defenders:

- What information should be provided about the vulnerability?
- Where should this information be provided?
- What, if any, additional measures should be taken to draw attention to the existence of the vulnerability or the availability of its fix?

### 4.5.1 Prepare and Circulate a Draft

Prior to public announcement of a vulnerability document, we find it helpful to circulate a draft document describing the vulnerability to give CVD participants an opportunity for discussion and commentary.

At a minimum, a draft advisory should be shared between the reporter and vendor to reduce the likelihood of either party being taken by surprise. Ideally, both parties should use this document to coordinate how much information is being released, and what future expectations might exist. Both parties also have the opportunity to correct erroneous information, as well as verify that credit for discovering or reporting the vulnerability is given to the correct person or organization. If multiple vendors are affected, or there are affected downstream vendors making use of the vulnerable software, it can be useful to share a draft with some or all of the affected vendors for even more feedback. Be sure to also discuss what channels to use for publication and disclosure.

The Traffic Light Protocol (TLP) may be useful when sharing draft information. We discuss applications of TLP to CVD in Section 7.2.2.1.

An example of a template for a vulnerability disclosure document is provided in the appendices.

### 4.5.2 Publishing

Once the draft circulation phase is complete, the next step is publishing the vulnerability document to whatever channels have been identified during previous phases.

Some vendors have a specific website that lists all their security advisories to date. Others might email the disclosure to a user mailing list or a security mailing list such as Full Disclosure [83] or BugTraq [82]. Reporters themselves may also chose to disclose by posting the advisory to a mailing list or including it in a personal or company blog. A common goal for reporters in the CVD process is to synchronize their publication with the vendor's response. As a result, near-simultaneous publication occurs quite often.

It is generally courteous for the vendor and reporter to contact each other after disclosure to inform one another that the disclosure went through as planned and provide URLs to the published documents.

### 4.5.3 Vulnerability Identifiers Improve Response

Many vulnerability reports can be similar, and sometimes a vendor or coordinator might receive multiple reports of similar vulnerabilities at the same time. Sometimes this is due to independent discovery, which we discuss in Section 6.5. Other times it reflects a report traversing multiple paths to arrive at its destination within the CVD process. This is fairly common in cases where a vulnerability affects products from multiple vendors. Using a common identifier improves coordination as it ensures that all coordinating parties can keep track of the issue.

The most common identifier in use today is the CVE ID [14], which is meant as a globally unique identifier for a public vulnerability report. CVE IDs can be obtained from the CVE Project at MITRE or one of several CVE Numbering Authorities (CNAs) established by MITRE—typically the vendors of common software products themselves [84]. Both reporters and vendors can request a CVE ID, but reporters should first check if the vendor they are coordinating with is already a CNA. This identifier should be included in any pre-disclosure shared drafts, so that all parties are aware of the common identifier.

### 4.5.4 Where to Publish

Publicly disclosing the existence of a vulnerability and the availability of its fix is usually considered to be the primary goal of the CVD process. Vendors will often provide vulnerability information

- on the vendor’s public website. Many vendors offer a security-focused section within the support section of their online offerings.
- to a public mailing list or a vendor-specific list

Vendors of bespoke software or products with highly focused customer bases are sometimes reasonably confident that they can reach their affected users directly. These vendors often publish vulnerability and fix information

- on a customer-only site
- via a customer support mailing list
- by individually notifying customers, for example through the technical sales channel

However, even with a well-organized customer contact database, it can be difficult for a vendor to be certain that all relevant decision makers are reached in a timely manner. Hence, we recommend that vendors publish at least basic vulnerability and fix announcements to their public website in addition to whatever direct customer contact communications they provide.

## 4.6 Promote Deployment

Although we tend to think of the CVD process as ending with the disclosure of a vulnerability, if the fix is not deployed the rest of the exercise is futile. A patch that is quietly posted to a website and not well advertised is almost useless in protecting users from vulnerabilities.

Deploying patches typically implies getting users, customers, and deployers to take positive action. Many software products are used by non-technical users. These users are often unaware of how to take remediative action for a vulnerability. A vendor's disclosure plan should consider how to reach the widest audience.

Products with secure automatic updates provide a good way to get a patch deployed quickly to a wide audience. However, not all users are able or willing to use automatic updates, so it is still important for vendors to draw attention to their fixes. Vendors should strive to implement easy and secure update methods in their products. In situations where this is not possible, the vendor's disclosure plan should be specific about how to spread the word of a new patch as quickly as possible.

### 4.6.1 Amplify the Message

Sometimes it is necessary to draw more attention to a problem or fix. Critical vulnerabilities, including those that are already being exploited or are highly likely to be exploited, may warrant attracting attention beyond merely publishing a document on the vendor's support site. In such cases, additional measures should be taken to draw attention to the existence of the vulnerability or the availability of its fix. Vendors should consider using

- announcements via social media. Many defenders use services like Twitter or Reddit as part of their daily situation awareness process, routinely sharing useful links and references with each other.
- mass media such as press releases, press conferences, and media interviews
- working with a coordinator or government agency to draw attention to a vulnerability or its fix. In particular, National CSIRTs can often provide advice or assistance with publicity on important issues.

### 4.6.2 Post-Publication Monitoring

Once a vulnerability and/or its fix has been disclosed, both vendors and reporters should look for feedback concerning any problems with either the documentation or the fix. In some cases, this can take the form of technical monitoring (e.g., monitoring download logs from the vendor's update service, checking inventories of deployed system versions, or even scanning) to ascertain the rate of defender deployments. Even if such technical monitoring is not possible, not permitted, risky, costly, or otherwise impractical, it is usually possible to monitor for user feedback via support requests, online discussions, and so forth.

In the event of slow uptake of the fix, additional effort might be warranted to call attention the vulnerability (for example, using social media), as discussed in Section 4.6.1.

It is also possible that the remediation advice is incorrect, or may not apply to all scenarios. Therefore the vendor and reporter should monitor for public discussion or reports of problems, so that the disclosure advisory and remediation information can be updated as necessary. Remember, the goal for remediation is to fix vulnerable product instances or at least reduce the impact of the vulnerability. Consequently, if a significant portion of the vulnerable product instances have not been remediated, that goal has not been achieved.

# 5 Process Variation Points

*All evolutionary biologists know that variation itself is nature's only irreducible essence…I had to place myself amidst the variation.*

*– Stephen Jay Gould*

While the previous section described the phases common to most CVD process implementations, there is considerable room for variation in the CVD process. In this section we explore a few of the variations we most often encounter.

## 5.1 Choosing a Disclosure Policy

For those responsible for implementing the CVD process, defining a disclosure policy is an important first step. A well-defined policy makes it clear what other participants in the CVD process can expect when they engage with you and establishes good relationships between finders, reporters, vendors, coordinators, and other stakeholders.

A disclosure policy typically describes what CVD stakeholders (finders, reporters, vendors, coordinators) can expect in terms of these factors:

- **Scope** – A description of the scope of issues to which the policy applies. This scope should be as explicit as possible, especially when there are specific boundaries of concern to the organization. If a bounty is to be paid for some classes of vulnerability reports, the scope definition should clearly delineate which kinds of reports will be eligible for the bounty.
- **Exceptions** – Any exceptional conditions that may alter the typical flow of the process
- **Safe Harbor** – Should your organization choose to explicitly disavow legal retribution against reporters who otherwise follow the policy, that fact should be clearly laid out in the policy document.
- **Report quality requirements** – It's okay to require reports to meet a certain level of quality before committing to taking action on them. However, it's also useful to judiciously apply the principle of robustness here: "In general, an implementation should be conservative in its sending behavior, and liberal in its receiving behavior" [85].
- **Preferred Communication Language(s)** – If the organization has preferences for specific (human) languages for reports, the policy should specify this. That said, English is usually acceptable as a default.
- **Contact Information** – How should reports be submitted? How can you be reached?
- **Timing** – Setting expectations for response timelines of the various milestones in a vulnerability report case can be helpful too. Most important are expected time to acknowledge receipt of a report and a default disclosure timeframe if one has been defined. An acknowledgement timeframe of 24-48 hours is common for vendors and coordinators, while 45-90 days seems to be the normal range for disclosures these days. That said, we recommend that both vendors and reporters treat policy-declared disclosure timeframes as the starting point of a negotiation process rather than a hard deadline.

A few examples of vulnerability disclosure policies can be found in Appendix E.

RFC 2350 provides recommendations on how to publish information about your CSIRT and disclosure policy and procedures [86].

## 5.2 Disclosure Choices

As we have mentioned previously, participants in Coordinated Vulnerability Disclosure iterate over the following questions:

1. What actions should I take in response to this knowledge?
2. Who else should I tell about it?
   a. What should I tell them?

Let’s take a moment to explore questions 2 and 2a in a few scenarios. Each of these disclosure options have advantages and disadvantages. In this section, we adapt and expand some terminology from Shepherd [87]:

- **No Disclosure** – All information about the vulnerability is kept private. Sometimes this is enforced by non-disclosure agreements (NDAs). Vendors sometimes prefer this scenario to protect trade secrets or to lessen the impact of perceived negative press. Some finders prefer not to disclose vulnerabilities to anyone, in the hope that malicious actors will not find out about the vulnerability if it is not disclosed. Data on the outcomes of a non-disclosure policy are difficult to come by, as these vulnerabilities are by definition hidden from public view.
- **Private Disclosure** – When a product’s vendor is aware of a vulnerability, the vendor may take action to address it but will only notify its own customer base of the vulnerability and its mitigation privately. Many of the same motives as the No Disclosure policy are also in play here; the hope is that malicious actors are much less likely to find out about and exploit a vulnerability if very few people are made aware of the issue. Avoiding negative press is also cited as a reason for this approach. Some vulnerability finders are satisfied by this method if all known customers can be reached, so that everyone using the software may be protected. However, this approach is often not practical for widely deployed or open source software.
- **Limited (Partial) Disclosure** – When a vulnerability is found, only some information about the vulnerability is disclosed to the public. The goal is typically to slow down reverse engineering and exploit development long enough for a fix to be developed and deployed. This is done by withholding proof of concept code or other technical details of the vulnerability while still providing enough information that users of the product may take action to mitigate the issue.
- **Full Disclosure** – When a vulnerability is found, all information about the vulnerability is disclosed to the public. Typically, this scenario results in the release of proof of concept exploit code along with a report describing the vulnerability. In some cases, finders following a full disclosure approach may not attempt to notify the vendor at all in advance of the public release of the vulnerability report. In other cases, they may contact the vendor simultaneously or shortly before issuing a public report. The belief is that this approach serves the greater good by allowing consumers to be aware of the full impact of issues in their products and demand action from vendors, as well as have information available to take appropriate defensive action and make more informed purchasing decisions. Another perceived benefit is

that a full disclosure allows other researchers and organizations to reproduce and confirm the vulnerability, whereas a more limited disclosure may not provide enough information to do so. Alternately, this type of disclosure may also be performed by the vendors themselves: many open source projects, for example, handle security issues in the open in order to maximize review of the vulnerability and testing of the proposed solution.

## 5.3 Two-Party CVD

The simplest instance of CVD is when there are only two parties involved: the finder of the vulnerability and the vendor who can fix the software. In this case, many of the complexities that arise in multiparty situations do not come into play. That is not to say that two-party CVD is always straightforward or easy. It can still be difficult for the finder of a vulnerability to make contact with the vendor. It can sometimes be difficult for the vendor to work with the finder toward a resolution. Personalities, attitudes, expectations, assumptions, and egos all play a part in the success or failure of even two-party CVD.

## 5.4 Multiparty CVD

Most of the interesting cases in CVD involve more than two parties, as these are the cases where the most care must be taken. Automation of the process can help somewhat, but the impact technology can have on the problem is limited by the inherent complexities involved in trying to get numerous organizations to synchronize their development, testing, and release processes in order to reduce the risk to users. From a coordinator’s perspective, it can be difficult to be fair, as you’re almost guaranteed to either miss some downstream vendor or wind up with one or more vendors ready to release while everyone is waiting for the other vendors to catch up. We discuss this conundrum further in Section 6 below.

The FIRST Vulnerability Coordination SIG [24] has published its “Guidelines and Practices for Multi-Party Vulnerability Coordination and Disclosure” [88] which we strongly recommend reading.

Success at multiparty coordination has more to do with understanding human communication and organization phenomena than with the technical details of the vulnerability. The hard parts are nearly always about coordinating the behavior of multiple humans with diverse values, motives, constraints, beliefs, feelings, and available energy and time. The vulnerability details may dictate the “what” of the response, but to a large degree human social behaviors decide the “how.”

In the next few subsections we discuss a number of issues that we have observed in performing multiparty CVD over the years.

### 5.4.1 Multiple Finders / Reporters

If one person can find a vulnerability, another person can too. While documented instances of independent discovery are relatively rare, independent discovery of vulnerabilities can and does happen. Perhaps the best-known example of multiple finders is Heartbleed (CVE-2014-0160). In part because of the complexity of the coordinated disclosure process, a second CVE identifier was assigned to the same vulnerability (CVE-2014-0346) [89]. We discuss independent discovery further in Section 6.5.

### 5.4.2 Complicated Supply Chains

Many products today are not developed by a single organization. Instead, they are assembled from components sourced from other organizations. For example, software libraries are often licensed for inclusion into other products. When a vulnerability is discovered in a library component, it is very likely that not only does the originating vendor of the library component need to take action, but all the downstream vendors whose products use it need to take action as well. Complex supply chains can increase confusion regarding who is responsible for coordinating, communicating, and ultimately fixing vulnerabilities, leading to delays and systems exposed to unnecessary risk.

Historically, the Android ecosystem provides a clear example of this phenomenon. A vulnerability in a library component used in the Android operating system might have to be fixed by the library developer, then incorporated into the Android project by Google, followed by the device manufacturer updating its custom build of Android, and by the network carrier performing its customizations and testing before finally reaching the consumer's device. Each additional step between the party responsible for fixing the code and the system owner (the device user, in this case) reduces the probability that the fix will be deployed in a timely manner, if at all [62].

At the CERT/CC, we often find it useful to distinguish between vertical and horizontal supply chains. While the vertical supply chain is most common, we do occasionally need to navigate horizontal supply chains in the course of the CVD process.

#### 5.4.2.1 Vertical Supply Chain

In a vertical supply chain, a vulnerability exists in multiple products because they all share a dependency on a vulnerable library or component. One vendor originates the fix. Many vendors then have to incorporate the originating vendor's fix into their products. Many vendors have to publish documents, distribute patches, and cause deployers to take action.

One example of a CVD process following a vertical supply chain is as follows: a vulnerability might be initially identified in product X, but is then isolated to a library that product X includes as a dependency. In this case, the library developer must be engaged as another party to the coordination process in the role of patch originator. The complexity does not end there though. Once the library vendor has completed its patch, not only does the vendor of product X have to integrate the fix, but all the other vendors that include the library need to update their products as well. We have done this kind of coordination in the past with vulnerabilities affecting MS-SQL [90], Oracle Outside In [91], and so on. The cascading effects of library vulnerabilities often result in significant subsets of users left vulnerable while they await their product vendor's updates.

#### 5.4.2.2 Horizontal Supply Chain

Even more complex in terms of coordination are cases where multiple products implement the same vulnerability, which is the primary characteristic of a horizontal supply chain. Examples include vulnerabilities arising from underspecified protocols, design flaws, and the like. Luckily these kinds of vulnerabilities are rare. CVD can become quite complicated when they occur, because multiple vendors must originate fixes for their own implementations. Many such cases combine with each originating vendor's downstream vertical supply chain as well, which only serves to increase the complexity. Many vendors have to publish docs and distribute patches, leading to deployers needing to take multiple actions.

Multiple implementation vulnerabilities can sometimes result from widespread copying of vulnerable code examples from books or websites or from developer tutorials that ignore or intentionally disable security features in order to simplify the learning process. While we cannot place the entirety of the blame for widespread Android SSL Man-in-the-Middle vulnerabilities such as VU#582497 [92] on any specific phenomenon, our spot checks of some of the vulnerable apps made it clear that parallel implementation of the same errors was a contributing factor in many of the affected apps. In that case, we identified more than 23,000 distinct apps [93], and coordinated with thousands of vendors.

A more pernicious example of multiple implementation is a vulnerability whose root cause lies in the specification or reference implementation of a network protocol. Because most vendor's products will specifically test for compatibility with these reference artifacts, such cases usually imply that every product supporting that feature will need to be fixed. Multi-originator cases can be very complex to coordinate. The SNMP vulnerabilities found in 2002 via the OUSPG PROTOS Test Suite c06-snmpv1 [94] [95] [96] [97] represented just such a case, and stand to this day as the most complex disclosure case the CERT/CC has ever coordinated.

### 5.4.3 Mass Notifications for Multiparty CVD

In their Usenix Security 2016 paper, Stock and colleagues [98] examined issues surrounding large-scale vulnerability notification campaigns. In this work, which focused on notifying website operators of vulnerabilities in their sites, they highlight significant difficulty in establishing a direct communication channel with vulnerable sites. The following is from their conclusion:

> *How do we inform affected parties about vulnerabilities on large scale? Identifying contact points remains the main challenge that has to be addressed by the Internet society, including network providers, CERTs, and registrars. We imagine that this problem could, for example, be tackled by centralized contact databases, more efficient dissemination strategies within hosters/CERTs, or even a new notification channel or trusted party responsible for such notifications. Until we find solutions to the reachability problem, the effects of large-scale notifications are likely to remain low in the future.*

## 5.5 Response Pacing and Synchronization

Problems can arise when the multiple parties involved in CVD function at different operational tempos. In both the vertical and horizontal supply chain cases discussed above, synchronized timing of disclosure to the public can be difficult to coordinate. The originating vendor(s) will usually want to release a patch announcement to the public as soon as it is ready. This can, however, put users of downstream products at increased risk. As a result, coordinators sometimes find it necessary to make the difficult choice to withhold notification from a vendor in a complicated multiparty disclosure case if that vendor cannot be trusted to cooperate with the coordination effort.

### 5.5.1 When One Party Wants to Release Early

In a multiparty coordination scenario, some vendors may want to release early to protect their customers. They have a good point: should Vendor A's customers be kept vulnerable just because Vendor B is taking longer to prepare its response? Yet an equally strong counterargument can be made: should customers of Vendor B be exposed to additional risk because Vendor A was faster at its vulnerability response process? There is no single right answer to this dilemma. The best you can do is keep the communication channels open and try to reduce the amount of surprise among participants. Planning for contingencies can be a useful exercise too—the focus of such a contingency should be how to respond if information about the vulnerability got out before you were ready for it.

### 5.5.2 Communication Topology

The complexity of coordination problems increases rapidly as more parties are involved in the coordination effort. Graph theory tells us the number of participant pairs increases as $N(N-1)/2$ for N participants. As a result, multiparty coordination using point-to-point communications do not scale well. Borrowing from communication network concepts, multiparty coordination involving more than a few participants can be improved with a shift to either a hub-and-spoke or shared-bus topology in lieu of a full mesh or collection of point-to-point communications (see Figure 2).

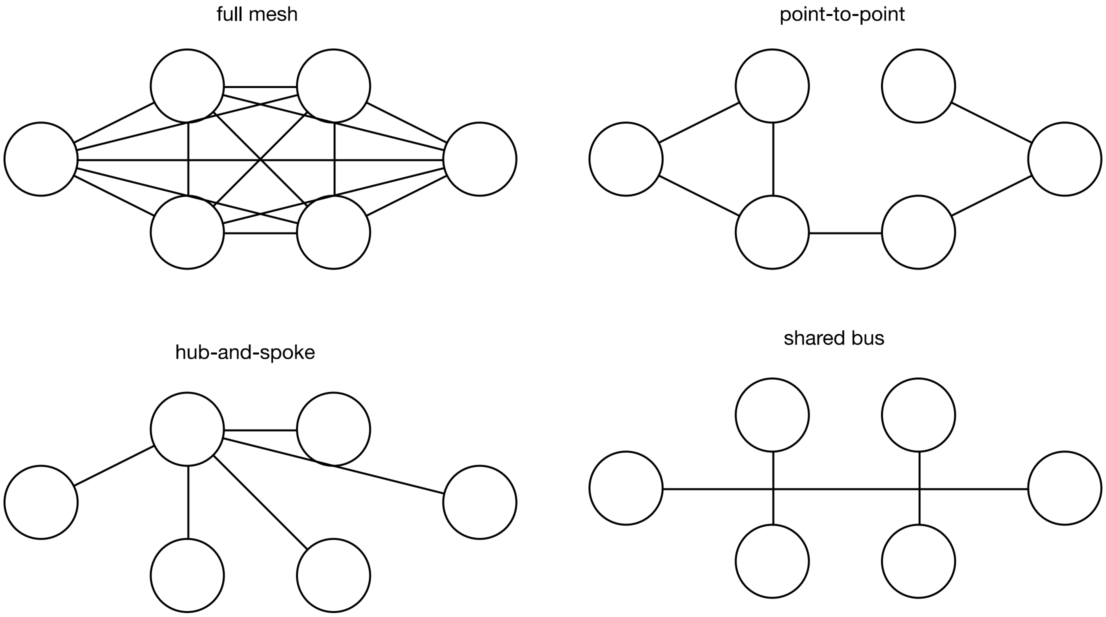

*Figure 2: Coordination Communication Topologies*

The CERT/CC has historically applied a hub-and-spoke approach to coordination for cases where it was feasible to maintain separate conversations with the affected parties. Maintaining a hub-and-spoke coordination topology for each distinct case requires some forethought into tools and practices, though—you can't just carbon-copy everybody, and without good tracking tools, keeping tabs on who knows what can be difficult. A hub-and-spoke topology allows a coordinator to maintain operational security since each conversation has only two participants: the coordinator and the other party. The tradeoff is that the coordinator (hub) can easily become a bottleneck during especially active coordination situations.

Some of the larger coordination efforts we have encountered have required more of a shared-bus approach through the use of conference calls, group meetings, and private mailing lists. This approach puts the CVD participants in direct contact with each other rather than having a coordinator acting as a proxy for all communications while minimizing the communication overhead. A shared-bus approach can increase the efficiency of communications, but can on occasion make it harder to reach agreement on what is to be done.

### 5.5.3 Motivating Synchronized Release

CVD process discussions tend to focus on the handling of individual vulnerability cases rather than the social fabric surrounding vulnerability coordination we construct over time. Shifting away from the individual units of work to the social structure can suggest a way out of some of the more contentious points in any given case.

We previously described the multiparty delay problem. Game theory provides us with the prisoners' dilemma as model for thinking about this concern. The main takeaway from research into the prisoners' dilemma is that by shifting one's perspective to considering a repeated game, it's possible to find better solutions than would be possible in a one-shot game with no history. The recognition that it's a repeated game leads to improved cooperation among players who would otherwise be motivated to act solely in their own self-interest in each round [99].

One approach we've found to work is to remind the parties involved that this will likely not be the last multiparty vulnerability coordination effort in which they find themselves. A vendor that repeatedly releases early will likely get left out of future coordination efforts. Because of this, the quicker vendors might be motivated to delay so they get the vulnerability information the next time. Perhaps most important for those wanting to release early is to remember that this is a repeated game; you might be first one ready to publish this time but that may not always be the case. Consideration for the other parties involved in any given case can yield better outcomes in the long run.

In the end, everyone benefits from vendors improving their vulnerability response processes, so helping the laggards become more efficient can sometimes become a secondary goal of the coordination process.

## 5.6 Maintaining Pre-Disclosure Secrecy

> *Three can keep a secret, if two of them are dead.*
>
> – *Benjamin Franklin*

The more people who know a secret, the more likely it is to leak. Simple probability theory tells us that even if the probability of any given party leaking is very low, the cumulative probability of a leak increases exponentially with the number of parties involved [100].

Returning to our simple model, and the "Who needs to know what, when?" question, multiparty disclosure highlights the need to balance need-to-know with need-to-share. There are varying degrees of need-to-know. Not everyone needs to know the same thing at the same time. Patch originators are usually notified early in the process, since their answer to "What do I need to do in response to this knowledge?" (i.e., create a patch) is often on the critical path for any downstream parties to be able to take action. Downstream vendors (patch consumers) and deployers can be notified later.

### 5.6.1 Coordinating Further Downstream

Vulnerabilities having the potential for significant impact can lead to coordination efforts beyond the traditional product vendor space. Infrastructure and service providers are sometimes brought in early, if there are mitigations that can be deployed in advance of the availability of a fix. This can be especially helpful in cases where the vulnerability may affect the infrastructure necessary to distribute the patch in the first place.

### 5.6.2 Do You Include Deployers?

Be careful to consider fairness though: By what criteria should you notify service provider X but not service provider Y? At some point, the complexity of who knows what gets high enough that the likelihood of a leak goes to 1, and you might as well go public.

### 5.6.3 Complex Communications Reduce Trust

It's also important to be aware that not all participants along the chain of disclosure will be equally trustworthy. That's not to say they are actively malicious, just that they may have incompatible values or priorities that lead them to disclose the existence of the vulnerability to others earlier than you'd prefer.

## 5.7 Disclosure Timing

> *When you say it's gonna happen now,*
>
> *When exactly do you mean?*
>
> *See I've already waited too long*
>
> *And all my hope is gone*
>
> *– The Smiths, "How Soon is Now?"*

How long is "long enough" to respond to a vulnerability? Is 45 days long enough? Is 90 days too short? Is 217 days unreasonable? Three years? Talk among yourselves. We can wait.

As with so many questions that arise in the CVD process, there is no single right answer. So rather than trying to solve an underspecified set of inequalities, let's have a look at some of the factors that tend to play into timing choices. This will give us an opportunity to see where some of the variability comes from.

### 5.7.1 Conference Schedules and Disclosure Timing

Conference schedules often drive researcher timelines. This is a big one. There is a rhythmic cycle to the vulnerability disclosure calendar. Black Hat [101] and DEF CON [102] happen in early August every year. Usenix Security [103] is usually right after that. The RSA Conference [104] is in the late winter or early spring. CanSecWest [105] is in the spring. Smaller conferences are scattered in between. Many of these conferences rely on presenters describing novel attack methods in varying degrees of detail. However, in order for researchers to analyze, develop, and demonstrate those techniques, vulnerabilities are often uncovered in extant products. That means that coordinating the disclosure of the vulnerabilities they've found is a common part of the conference preparation process for presenters. The CERT/CC often observes an increased rate of vulnerability reports a few months in advance of these conferences. Vendors would do well to be aware of these schedules and be prepared to respond quickly and appropriately to seemingly inflexible deadlines for disclosure.

### 5.7.2 Vendor Reputation and Willingness to Cooperate

Vendors that are perceived to treat vulnerability reporters poorly or that are perceived to be slow or unresponsive may find themselves being left to discover reports of vulnerabilities in their products at the same time as the public becomes aware of them. CVD is a social process, remember? And the game is played over and over, by players who share knowledge between rounds.

### 5.7.3 Declarative Disclosure Policies Reduce Uncertainty

Avoiding surprise was one of the principles in Section 2. To that end, explicitly declared policies (from both researchers and vendors) are a good thing. Expected disclosure timing is an important question to ask whenever a report is received. Sometimes the reporter or coordinator acting on the reporter's behalf has a standing policy of X days with no exceptions. Other reporters may be more flexible. If in doubt, ask.

### 5.7.4 Diverting from the Plan

> *Je n'ai jamais eu un plan d'opérations.*
>
> *– Napoleon Bonaparte*

Plans are one thing, but reality sometimes disagrees with our assessment of it. Breaking a previous disclosure timeline agreement is sometimes necessary when events warrant. Below we cover a few reasons to release earlier or later than planned.

Reasons to release early include

- evidence of active exploitation
- vendor fails to respond, is not acting in good faith, or denies the existence of a vulnerability
- vulnerability is known to be discovered by adversaries, so the race to defend vulnerable systems is more focused
- all known users have been notified and patched (usually via private channels)

Reasons to hold back release include

- vendor not ready with fix, but continuing to make progress and is acting in good faith
- vulnerabilities with severe impact, especially those affecting safety-critical or critical infrastructure
- cases where new information is found late in the process, for example that there are important but previously unrecognized dependencies that alter the impact of the vulnerability or patch deployability

In cases that divert from the planned disclosure date, it sometimes helps to seek the opinion of a neutral third party for advice on how to proceed. Finders, reporters, and vendors can each have valid yet conflicting perspectives on what the best course of action might be. Coordinator organizations are often able to help resolve conflicts by taking a neutral approach to the situation and advising one or more parties in light of their prior experience.

### 5.7.5 Releasing Partial Information Can Help Adversaries

When considering what information to release about a vulnerability, our advice is “Don’t tease.” Our experience shows that the mere knowledge of a vulnerability’s existence in a feature of some product is sufficient for a skillful person to discover it for themselves. Rumor of a vulnerability draws attention from knowledgeable people with vulnerability finding skills—and there’s no guarantee that all those people will have users’ best interests in mind. Thus, teasing the existence of a vulnerability in a product can sometimes provide an adversarial advantage that increases risk to end users.

# 6 Troubleshooting CVD

> *Good process serves you so you can serve customers. But if you're not watchful, the process can become the thing. This can happen very easily in large organizations. The process becomes the proxy for the result you want. You stop looking at outcomes and just make sure you're doing the process right. Gulp. It's not that rare to hear a junior leader defend a bad outcome with something like, "Well, we followed the process." A more experienced leader will use it as an opportunity to investigate and improve the process. The process is not the thing. It's always worth asking, do we own the process or does the process own us?*
>
> *– Jeff Bezos, 2016 Letter to Amazon Shareholders*
>
> *I came here to make the world a better place, but I think I broke it.*
>
> *– Judy Hopps, Zootopia*

As we've mentioned throughout this document, CVD can occasionally be a complex process. In this section, we'll first cover some of the common ways things can go wrong and finish with some suggestions on what to do if and when they do.

## 6.1 Unable to Find Vendor Contact

Any number of factors can lead to difficulty in making the initial connection between a would-be reporter and the party or parties that can do something about the vulnerability they want to report. Sometimes products outlive vendors. This can even happen in open source projects where the code is still out there but the team that built it has scattered to the winds. Companies go bankrupt. People change jobs. Maybe Vendor A included a library from Vendor B, who licensed it from C, but they only got a binary executable and didn't get the source code; and Vendor C is a spinoff of a conglomerate going through bankruptcy proceedings in a different country where they don't speak your language. Things can get complicated.

## 6.2 Unresponsive Vendor

Furthermore, even when you can find the vendor, not all vendors have established processes for receiving vulnerability reports. Again, potential reasons abound:

- They haven't thought about it, even though they should have.
- They don't realize they need it, even though they do.
- They think their software process is already good enough, even if it's not.
- They assume anyone reporting a problem is an evil hacker, even though they're wrong.

The U.S. Federal Trade Commission has brought legal action against vendors for not having sufficient vulnerability response capabilities. In their complaint against ASUS [106], they cite the company's failure to

> *maintain an adequate process for receiving and addressing security vulnerability reports from third parties such as security researchers and academics; … perform sufficient analysis*

> *of reported vulnerabilities in order to correct or mitigate all reasonably detectable instances of a reported vulnerability, such as those elsewhere in the software or in future releases; and … provide adequate notice to consumers regarding (i) known vulnerabilities or security risks, (ii) steps that consumers could take to mitigate such vulnerabilities or risks, and (iii) the availability of software updates that would correct or mitigate the vulnerabilities or risks.*

Similar complaints have been included in FTC filings against HTC America [107] and Fandango [108].

## 6.3 Somebody Stops Replying

Sometimes one of the parties involved in a CVD effort will stop responding. Often, this is simply a reflection of priorities and attention shifting elsewhere rather than intentional behavior. It's usually best to give the benefit of the doubt and keep trying to reestablish contact if one of the CVD participants goes unresponsive.

Even in cases where the vendor has stopped responding in the midst of a coordination effort, the CERT/CC recommends that reporters send the vendor a "heads up" message with some lead time before publishing, optionally including a draft of the document about to be published. This helps the vendor prepare its communication plan if necessary, and sometimes helps to identify any lingering misunderstandings on the technical aspects of the vulnerability. Ammar Askar's blog post about a Minecraft vulnerability serves as an example where a quick heads up to the vendor could have avoided some confusion [109].

## 6.4 Intentional or Accidental Leaks

Sometimes information leaks out of the CVD process. Perhaps an email gets CC'ed to someone who didn't need to know. Somebody might talk too much at a conference. Somebody could tweet that they just found a vulnerability in product X, providing no other details. Somebody might intentionally disclose the information to someone not involved in the supply chain for the fix.

Unfortunately, not everyone plays by the same rules. You might find information you thought was shared in confidence showing up in some non-confidential location. It might be a simple misunderstanding, mismatched expectations, or in rare cases, a malicious act. Regardless of how it leaked, there are three major questions to ask:

1. What information leaked?
2. How did the information leak?
3. How will you respond?

As we noted in Section 5.7.5, mere knowledge that a vulnerability exists in a certain component can sometimes be enough to enable a determined individual or organization to find it. While a partial leak of information isn't necessarily as serious as a detailed leak, it should at least trigger additional consideration of accelerating the disclosure timeline.

## 6.5 Independent Discovery

If one person can find a vulnerability, somebody else can, too. Andy Ozment [110] showed that "vulnerability rediscovery occurs 'in the wild' and that it is not particularly uncommon." Finifter and colleagues, reviewing a dataset of Chrome vulnerabilities, identified 15 out of 668 (2.25%) that had been independently discovered by multiple parties [111]. They go on to mention similar rates for Firefox vulnerabilities. Ablon and Bogart [112] studied a stockpile of zero day vulnerabilities, estimating that "after a year approximately 5.7 percent have been discovered and disclosed by others." Herr and Schneier [113] find browser vulnerabilities having rediscovery rates between 11% and 20% annually for the years 2013-2015. For Android vulnerabilities during the 2015-2016 timeframe, they found an annual rediscovery rate of 22%.

What is to be done when the CVD process is underway for a vulnerability, and a seemingly independent report of the same vulnerability arrives? One approach is to accelerate the disclosure timeline, possibly disclosing immediately. This approach assumes that if a vulnerability has been found and reported by multiple individuals acting independently, then it must be an easy vulnerability to find. This in turn implies that others who haven't reported it may also be aware of its existence, thereby increasing the likelihood of its availability to adversaries.

While we find this to be a reasonable conclusion, CVD participants should be wary of duplicate reports that are not independent. Truly independent discovery does yield some indication of the difficulty of finding a vulnerability. But vulnerability finders and security researchers talk to each other, and they sometimes hunt in the same places. An announcement of interesting vulnerabilities in a product can spur others to turn their attention and tools to that product. Even a casual "I've been looking at product X and found some interesting things" can put someone else on the hunt for vulnerabilities in product X. Any judgement of independence should consider the degree to which there is community interest in a product. As the popularity of products wax and wane through their lifespan, so too will security researcher attention.

An example of a coordination failure occurred during the vulnerability disclosure of Heartbleed. Two organizations, Codenomicon and Google, both discovered the vulnerability around the same time. When the vulnerability was reported a second time to the OpenSSL team, the team assumed a possible leak and the vulnerability was quickly disclosed publicly [114]. A more coordinated response may have allowed further remediation to be available immediately at disclosure time.

Even more insidious is a phenomenon we've observed in bug bounty scenarios. Because they pay for reports, bug bounties can unintentionally provide incentives for finders to share their reports with others prior to reporting, allowing multiple individuals to report the same bug, and potentially share in a larger payout. CVD is a social game: as such, its incentives affect participants' behavior.

Rather than prescribing a single rule that independent discovery should immediately trigger release of the vulnerability information, we suggest that CVD participants discuss the implications of rediscovery on a case-by-case basis in order to decide the best course of action for the particular case.

## 6.6 Active Exploitation

If evidence comes to light that a vulnerability is being exploited in the wild, that is usually a strong indication to accelerate the disclosure timeline. Active exploitation is indicative of either independent discovery or an information leak from the CVD process (whether intentional or accidental), with the added concern that not only does an adversary know about the vulnerability but is already using it. Hence, in the case of known exploitation, it's usually best to consider disclosing what is known about the vulnerability—hopefully with some mitigation instructions—as soon as possible even if a patch is not yet available. From the vendor's standpoint, acknowledging that you're already aware of the vulnerability and are working on a fix can help restore users' confidence in your product and the process that produced it.

## 6.7 Relationships that Go Sideways

We'll repeat it again here because it's so important: the participants in CVD are humans. They have feelings, and those feelings can get hurt. People get frustrated, angry, and sometimes just have bad days. The first thing to do when things appear to be going awry in a CVD case is to give people some slack to make mistakes.

The more transparent your process is—and the closer it is to what other folks are doing—the better you will be able to avoid problems. Good documentation is a start, but documenting a byzantine process isn't as useful as simplifying the process and then documenting that!

## 6.8 Hype, Marketing, and Unwanted Attention

In the past few years we've witnessed the rise of branded vulnerabilities: Heartbleed [89], Badlock [115], Shell Shock [116], and GHOST [117]. Does having a marketing department behind a vulnerability disclosure make that vulnerability worse than others without the marketing push? Not in any technical sense, no. Instead, what it does is draw additional attention to the vulnerability—so vendors can be forced to adjust the priority of the vulnerability cases they're working on and allocate resources toward addressing whatever vulnerability is getting the hype. Are branded vulnerabilities good or bad for internet security? The only good answer is the lesson of the Taoist parable of the farmer and the horse: "Maybe." [118].

### 6.8.1 The Streisand Effect

Attempts to squash true information once it's been revealed tends not only to spread the information more widely, but also to backfire on whoever is trying to conceal it. The name comes from a case involving the removal of online photos of a famous celebrity's house [119]. The attempt to suppress the photos only drew attention to them resulting in many more people seeing them than would have otherwise.

This scenario comes up from time to time in CVD cases. Often it takes the form of a vendor trying to suppress the publication of a report about a vulnerability in its product, with some threat of legal action if the information is released. As we've discussed previously, the knowledge that a vulnerability exists in some feature of a product can be sufficient for a knowledgeable individual to rediscover the vulnerability. The legal threats usually serve to amplify the discussion of the case within the security community, which draws more attention to the vendor and its products at the same time it demotivates reporters' willingness to participate in the CVD process. Even more problematic is that when such attention comes to focus on the vendors' products, it is very likely that additional vulnerabilities will be found—while simultaneously less likely that anyone will bother to report them to the vendor before disclosing them publicly. Vendors should not underestimate spite as a motivation for vulnerability discovery.

## 6.9 What to Do When Things Go Wrong

While we can't tell you what to do in every possible combination of contingencies that may arise in the CVD process, we can suggest the following guidelines to help you navigate the complexity.

### 6.9.1 Keep Calm and Carry On

Although problems with the disclosure process can be stressful, it's better to keep emotions in check while resolving issues. Recall from Section 2.2 that a presumption of benevolence is helpful when navigating the CVD process. As we have described thus far in Section 6, multiple things can go wrong in the disclosure process, but often these problems do not arise as a result of intentional acts of malice. So even if something has gone wrong, it's still good to give the benefit of the doubt to the good intentions of the involved stakeholders.

### 6.9.2 Avoid Legal Entanglements

Whatever the issue is in the context of a vulnerability disclosure, lawyers alone are rarely the right answer. Cease-and-desist letters tend to backfire as described in Section 6.8.1. Responding with legal threats can have negative public relations effects in the long term for vendors as well:

- It gives the appearance that the vendor is more concerned about protecting its image than users' security.
- It can give the impression that the organization is bullying an individual.
- It can drive future researchers away from reporting the vulnerabilities they find.

### 6.9.3 Recognize the Helpers

For vendors: A person who shows up at your door to tell you about a vulnerability in your product is not the enemy. That person is your friend.

For researchers: A vendor who is responsive is doing better than many.

For all parties involved in CVD: Give credit where it's due. Many participants in CVD are there because they care about making things better (see Table 1:  I Am the Cavalry's Finder / Reporter Motivations). Recognizing them for their good work keeps them engaged and helps everybody in the long run.

### 6.9.4 Consider Publishing Early

Recall that the goal of CVD is to help users make more informed decisions about actions they can take to secure their systems. Sometimes it becomes obvious that the coordination of a disclosure has failed. In these cases, it may make more sense to publish earlier than expected than to continue to withhold information from those who could use it to defend their systems.

See also Sections 6.4, 6.5, and 6.6.

### 6.9.5 Engage a Third-Party Coordinator

We have outlined a variety of ways in which the CVD process might not go as smoothly as you'd like, whether you are a finder, reporter, vendor, coordinator, or deployer. When problems arise that you're not prepared to handle, or even if you just need a quick opinion on what to do next, there are a number of coordinating organizations available to help. These include the following:

- CERT/CC
- national CSIRTs that handle CVD cases
  - JPCERT/CC
  - NCSC-FI
  - NCSC-NL
- larger vendors (Google, Microsoft, etc.)
- bug bounty operators (BugCrowd, HackerOne, etc.)

### 6.9.6 Learn from the Experience

Any process worth doing more than once is one worth improving. To that end, we recommend that participants in CVD take good notes. Hold a retrospective to identify things that went well, things that didn't, and explore changes you can make to your process for next time. This very document is in large part the result of notes taken during "lessons learned" sessions with vulnerability coordinators at the CERT/CC.

As an example of questions to begin a retrospective discussion, consider this list derived from the Scrum Alliance [120]:

- What went well?
- What went wrong?
- What could we do differently to improve?

# 7 Operational Considerations

*There was a time when nails were high-tech. There was a time when people had to be told how to use a telephone. Technology is just a tool. People use tools to improve their lives.*

*– Tom Clancy*

Participating in a CVD process over time requires a set of tools and practices in order to be successful. In this chapter, we outline a few tools of the trade, and consider common operational security and personnel management issues.

## 7.1 Tools of the Trade

An organization's capabilities are supported by people, process, and tools, and CVD is no different. This section covers some commonly used tools.

### 7.1.1 Secure Communication Channels

Secure communications can be nuanced to maintain in operation, so we first turn our attention to establishing and maintaining this capability.

#### 7.1.1.1 Email

Email is a simple, tried-and-true method of communication over the Internet. Simply because everyone has access to it, email is likely to remain a common way of receiving vulnerability reports and communications.

Because of the potential for organizational changes like employee promotions and turnover, having an individual's email address as a security point of contact is generally discouraged. A better solution is to establish an email account specifically for vulnerability reports and CVD activity. Many vendors choose to reserve an email address like <security@example.com> or <psirt@example.com> for this purpose. RFC2350 [86] suggests you include contact information as part of your overall CSIRT disclosure policy and process publication. Additionally, we recommend that this information be listed clearly under the Contact Us page or a similar page on your organization's website. Ideally, an Internet search for "*<vendor> report vulnerability*" should lead to that contact information.

The security email account should be an alias that is forwarded to or shared by one or more people. This way, multiple team members can check the incoming mail and cover for each other when team members are out of the office. Even if the security team is only one person, having an alias makes it easier to adapt should that person leave the organization; to the outside world, no contact information needs to change.

Although receiving information via email is convenient, it is not a very good mechanism for tracking multiple cases at once. Rather than sending security emails to individual mailboxes, vendors, coordinators, and even large-scale reporters should consider setting up a web-based case tracking system instead. That way, received emails can automatically generate new cases or augment existing ones, which can then be assigned to team members and tracked as necessary. More on this topic can be found in Section 7.1.4.

#### 7.1.1.2 Secure Email

Email by itself is not a secure medium. We recommend that emails containing sensitive information be encrypted [121]. This includes emails containing information about unpatched or otherwise sensitive vulnerabilities. The most common tools for email encryption are Pretty Good Privacy (PGP) [122] or Gnu Privacy Guard (GPG) [123], and Secure/Multipurpose Internet Mail Extensions (S/MIME) using X.509 certificates [124]. By far, PGP/GPG is the most commonly used by CVD participants. It has a learning curve but is relatively easy to use and maintain once set up. A number of tools exist to make working with PGP/GPG easier. In practice, very few individuals and organizations make use of S/MIME & X.509 for their CVD process; while you may offer S/MIME & X.509 as an option, PGP/GPG is recommended as the default.

The greatest difficulty with PGP/GPG (and really, any email encryption scheme) is encryption key management. Key management will be discussed in Section 7.2.1.

#### 7.1.1.3 Web Forms and Portals

Most vulnerability reports have a similar structure, making a web form a preferable method for receiving vulnerability reports for many organizations. To secure your web form, you will need to enable HTTPS and TLS, and obtain a TLS certificate from a Certificate Authority (CA). A free option is Let’s Encrypt, maintained by the Internet Security Research Group (ISRG) [125].

TLS ensures that the transmission is secure, but does not authenticate who sent the report. If this is a requirement, you may also need to implement login accounts on your web form with adequate authentication mechanisms such as two-factor or X.509 certificates. However, this setup increases the complexity for reporters to provide information to you, and as a result is rare in practice and likely unnecessary. Once set up, an HTTPS web form provides an easy-to-use way for reporters to contact you with vulnerability reports and other information.

However, a problem does come up: how do you acknowledge receipt of the report? Possibilities include the following:

- *Echoing back the message in a confirmation email.* The reporter knows you received the text, but now the text was sent in plain text email for everyone to read. It is unlikely you want to do this.
- *Send a brief thank you message, but without details.* The reporter can now be assured that the report was received. But what if you have a follow up question for the reporter? It’s likely you will need to send an email, which will require encryption to keep the details of the question and answer secure.
- *Send a brief thank you message, and ask the reporter to log in to a portal to view the message conversation.* This resolves the issue of two-way secure communications, but now you need to establish a public-facing portal and manage portal credentials. This raises additional operational considerations, for example those below:
  - user account management and password changes

  - two-factor authentication or X.509 as possible requirement to ensure user identity

  Portal credential maintenance introduces more complications, and for many vendors is likely not worth the effort. Another disadvantage of using a portal is that vendors and organizations may be unwilling to create accounts on another vendor's portal. This may discourage multi-party communication and coordination, effectively preventing a vendor or reporter from participating in multi-party CVD.

In the CERT/CC's experience, PGP/GPG encrypted email is the most common solution for a primary contact channel; HTTPS websites or third-party platforms can complement but may not always be sufficient to replace email for your organization's CVD needs.

### 7.1.2 Contact Management

For most reporters, the contact management process simply consists of maintaining a vendor's email address and PGP/GPG key in compatible mail client software. Contact management becomes vitally important to multiparty CVD, and is a particular concern for third-party coordinators. A common choice is Thunderbird with Enigmail [126], but other open source solutions such as Outlook with gpg4win [127], or KMail with KGpg/Kleopatra [128] and proprietary solutions such as Outlook with Symantec Encryption Desktop [122] also exist.

Finding vendor contacts can be difficult. Not all vendors include contact information in an easily searchable page, such as a Contact Us page linked from the vendor's homepage. Some alternatives include searching old mailing lists, using social media, or even sending physical letters to a business address [129].

In order to protect privacy during the disclosure process, mailing lists or simply carbon-copying all recipients to a single message is likely not an acceptable action in most scenarios. Vendors in many cases would like to keep their vulnerability information private except for what is specifically intended to be shared. At the CERT/CC, we have developed some in-house mailing scripts that auto-generate individual encrypted emails, one for each vendor we attempt to reach. In this way, we can maintain privacy up front, but can introduce two vendors should there be mutual interest in collaboration. Our current tools were written with our internal systems and network policies in mind. Other coordinators may look into similar efforts. We covered communication topologies for CVD in Section 5.5.2.

### 7.1.3 Bug Bounty Platforms

A number of third-party CVD platforms now exist to facilitate communication between vendors and reporters [52] [53] [54] [55]. Although they are often referred to as bug bounty platforms, often the "bounty" aspect is in fact optional—vendors can use bug bounty platforms to receive reports without needing to compensate reporters unless they choose to do so.

CVD platforms provide a secure communications channel (HTTPS) for reporters to communicate with vendors. These platforms generally allow two-way communications, making it easy for ongoing discussion between vendor and reporter. This channel is usually hosted by a third party in a software-as-a-service model, which may be important to some organizations that are not able to maintain their own infrastructure due to resource constraints. Of course, having vulnerability information hosted on third-party infrastructure may also present a data privacy risk to some organizations, so it is important to consult internal policies before determining if a CVD platform fits your organization’s needs and requirements.

An important note regarding these platforms is that the CVD platform by its nature requires a login. As explained in our discussion in the last section, requiring an account may discourage some reporters or other organizations from joining the platform, locking them out of discussion. Organizations should consider whether the benefits of using a CVD service outweigh this concern.

### 7.1.4 Case and Bug Tracking

Case tracking systems such as bug trackers or trouble ticket systems are often used by vendors, coordinators, and reporters for tracking vulnerability reports. Such systems can centralize the vulnerability response process and provide the ability to track individual cases. Case tracking systems also provide a means of collecting data about recurring security issues and the performance of the response process itself.

We have found it important to distinguish that vulnerability reports are not always bug reports. A single vulnerability report might contain information on more than one bug; alternately, it may describe a configuration issue that would not typically be considered a bug in the first place. In general, vulnerability reports and bug reports should be thought of as having a many-to-many relationship, allowing one or more vulnerability reports to be linked to one or more bug reports.

That said, bug tracking systems can still be useful for vulnerability report tracking if this distinction is kept in mind, and the bug tracking system has the appropriate capabilities. At the CERT/CC, we’ve found that more complicated CVD cases—for example the multiparty cases described in Section 5.4—become more like projects than tickets.

### 7.1.5 Code and System Inventories

As we discussed in Section 5.4.2, software-based products are typically assembled from components rather than written from scratch. Economies of scale apply to code, and most organizations would consider it wasteful to write a feature from scratch when that feature is available at a reasonable licensing cost for inclusion into their own products. As a result, each product is not merely the result of a single vendor’s development process, but of an entire supply chain. Libraries get included in other libraries, which form subcomponents in the composition of larger and more complex products.

For product vendors, an important part of the vulnerability response process is knowing what weaknesses your products might have. You can address this by clearly identifying any third-party libraries or utilities that are included with your products, and being alert and responsive to vulnerability disclosures in any third-party products that may affect your own products.

In recent years, a class of Software Composition Analysis tools (such as those offered by WhiteSource Software [130], Black Duck Software [131], Sonatype [132], Synopsys [133], Flexera Software [134], and the like) have come into use to identify potential licensing conflicts in commercial and open source products. Many of these tools are potentially useful to vendors looking to create an inventory of libraries and third-party components used in their products for security analysis purposes as well.

As a vendor, identifying your products' third-party dependencies is only the first step. After that, you should take steps to ensure that your vulnerability response capability maintains awareness of any security-related announcements about those upstream products on which your product depends. Some component vendors offer special communication channels (e.g., a mailing list) for their licensees. If you've worked with a coordinator like the CERT/CC in the past, ask if you can be placed in a special notification list for a particular library or product.

Furthermore, since it is likely that your products will in turn be used as components in some other vendors' solution, it can be a good practice to provide an inventory of components along with your product. A number of data formats and specifications have emerged in the software supply chain management space and are in use by product vendors already. These include the following:

- Software Identification (SWID) Tags [135]
- Common Platform Enumeration (CPE) [136]
- The Software Package Data Exchange (SPDX) [137]

### 7.1.6 Test Bench and Virtualization

Having an internal testing infrastructure is vital to proper triage and resolution of vulnerability reports as we discussed in Section 4.3.1. Not only is testing useful for confirming reports, or reproducing and isolating bugs; it can also serve as a platform for an organization to develop its own vulnerability discovery capability. The analysts responsible for confirming incoming reports will over time develop a familiarity with the ways in which a product is vulnerable, and, given appropriate training and support, can begin to apply this knowledge directly to the product without having to wait for vulnerability reports to arrive from elsewhere. We have followed this exact path at the CERT/CC: tools such as Dranzer [138] and BFF [139] came directly out of vulnerability analysts applying broad knowledge of vulnerabilities gained from detailed analysis of reports toward the development of automated discovery processes.

By far the easiest way to build a vulnerability testing infrastructure is the use of virtualization technologies. Many different virtual machine environments can be built for receivers of vulnerability reports to verify or clarify the reports they receive. At the CERT/CC, we maintain a firewalled testing network in which virtual machines can be placed for testing. We also maintain a few permanent pre-configured servers on this network (HTTP web servers, etc.) to allow easy testing of certain classes of vulnerabilities, such as "drive-by" browser downloads. Much of this infrastructure could be replicated entirely in a virtual network within a single machine, or in a cloud-based environment if desired.

Be sure your analysts have proper access to any necessary software needed for testing. This includes maintaining appropriate software licenses for proprietary software, although in many cases free and/or open source alternatives are available. Toolkits often include the following:

- virtualization platform, often a need to support multiple operating systems
- debuggers

- source code analysis tools
- binary analysis tools (decompilers, etc.)
- network sniffing tools
- hex editors
- text editors
- visualization (often built into other tools)

Vendors with more hardware-centric products may need to additionally maintain more physical gear or specialized test bench equipment in order to have sufficient capacity to confirm reports.

## 7.2 Operational Security

Operational security, often shortened to “opsec,” is an important part of CVD operations. Opsec includes your ability to maintain security and confidentiality for information associated with vulnerability reports prior to disclosure.

### 7.2.1 PGP/GPG Key Management

Separate from the issue of maintaining encryption keys for your contacts, you must also maintain your own individual or organizational encryption key.

PGP/GPG is a form of asymmetric encryption that makes use of two different encryption keys called your *public key* and your *private key*. The public key is intended to be shared; you advertise your public key, and individuals or organizations wishing to contact you use your public key to encrypt a message. Messages encrypted to your public key can only be decrypted by the private key; therefore, it is important that your private key stays private, and that no one outside of your team or organization has access to this key.

A general discussion on encryption algorithms is beyond the scope of this report, but at the time of this writing, it is recommended to generate RSA keys with a length of at least 4096 bits to ensure security into the near-to-moderate-term future.

#### 7.2.1.1 Use a Passphrase and Control Access

We recommend setting a strong passphrase on any key. Without a passphrase, anyone who obtains the key file can immediately use the key to sign messages or decrypt messages; if the private key is leaked to unauthorized users but a passphrase was applied, then the user would need to also know the passphrase before any damage could be done.

Of course, a persistent attacker could attempt to brute force the phrase, so ideally the private key should be kept somewhere safe, out of the hands of any unauthorized users. In other words, only members of the CVD team should know the passphrase or even have access to the key file. Ideally, if your CVD capability includes a dedicated communications team, restrict knowledge of the passphrase and access to the key material only to those members directly involved with communications. At the CERT/CC, we have implemented this by having dedicated systems for CVD communications work; private keys are only accessible from these systems. In this setup, users must specifically request access and can be allowed or denied based on need.

#### 7.2.1.2 Use Revocation Certificates and Key Rotation

A concern is that the private key may land into unauthorized hands. This might occur in the event of a network breach, but another possibility is a disgruntled former employee retaining a copy of the key. Because of these possibilities, organizations using shared key material should generate a revocation certificate for their PGP/GPG key and store it somewhere safe. Obviously, it is important to restrict access to the passphrase and revocation certificate; typically, it should only be accessible by management. Should any emergency or security event occur, you can publish the revocation certificate to notify the public to not trust this key anymore. When a user imports your revocation certificate, it marks the associated PGP/GPG key as unusable and untrustworthy.

To further mitigate these concerns, we recommended that you rotate encryption keys (i.e., generate a new one) regularly. Some teams choose to use the same key for two, three, or more years without change. This recommendation arises to address concerns regarding the threat of network breaches and the potential impact of losing control of the private key. At the CERT/CC, we generate a new PGP/GPG key yearly.

Another reason to rotate keys is to revoke access to future encrypted email when analysts leave the CVD team. The best way to do this is to generate a new key whenever there are personnel changes; in this way, future messages will be encrypted to the new key with a new passphrase, and any former team members will be unable to access these messages even if they still have access to the old key.

Regardless of how often you generate a new PGP key, your latest PGP public key needs to be available to individuals and organizations so that they may contact you. Be sure your key generation process includes necessary steps to put your PGP public key in the public’s hands. This can consist of posting your PGP public key directly to your organization website, or pushing your key to one of many PGP public key servers. You can use these same mechanisms to distribute your revocation certificate should the need arise.

#### 7.2.1.3 Practical Tips for Key Management

We wrap up this discussion with a review of recommended practices for PGP/GPG key management:

- Generate an RSA key of at least 4096 bits.
- Restrict access to the private key material.
- Use a strong passphrase on the key.
- Restrict knowledge of the key passphrase to only those members of the CVD team involved in communications.
- Generate a revocation certificate for each key.
- Store the key passphrase and a revocation certificate in a safe location, such as a locked cabinet or safe in a secured area of the organization.
- Generate a new key whenever a member leaves the CVD team, and revoke the old key.
- Generate a new key periodically, regardless of other factors.
- Make your latest public key available in a known location to ensure recipients always have access to the latest key.

### 7.2.2 Handling Sensitive Data

Some of the information that passes through a CVD process may include information on an organization’s internal processes, trade secrets, or even national security interests in some scenarios. Proper precautions need to be established. It is recommended that recipients treat all received data as private unless explicitly given permission to share.

Of course, there is some ambiguity when you say *private*: does that mean the information is for the whole organization? Just the CVD team? Possibly even a single analyst? In general, vulnerability information should be shared with the fewest number of people possible to effectively coordinate and remediate a vulnerability prior to disclosure. Clearly declaring the data’s sensitivity can help to make that determination.

#### 7.2.2.1 Traffic Light Protocol (TLP)

The Traffic Light Protocol (TLP) has been adopted for a standards-track by FIRST [140]. By marking a document with a TLP level—Red, Amber, Green, or White—a sender can easily communicate the sensitivity of vulnerability information and expectations about sharing it further.

In the context of CVD, the following applies:

- TLP:GREEN and TLP:AMBER are best suited for information shared between reporters, vendors, and coordinators during phases prior to public announcement of a vulnerability.
- If pre-publication announcements are made to deployers or other stakeholders, TLP:RED or TLP:AMBER could be a good fit.
- TLP:WHITE is most useful for public disclosures.

See Appendix B for more on TLP.

### 7.2.3 Don’t Automatically Trust Reports

There are two reasons that organizations receiving vulnerability reports should maintain a degree of wariness regarding the reports they receive. The first is intentional misdirection of your CVD capability, which we already discussed in Section 4.3.1.1. The second is subtler, in that the technical infrastructure you deploy to manage CVD cases can potentially be affected by the vulnerabilities you are coordinating.

Vulnerability reports may contain hostile attachments—not necessarily as an attack, but simply a reporter sending a proof-of-concept for your review—so vendors and coordinators should design their report tracking systems and process accordingly. Be sure attachments to vulnerability reports are not opened automatically anywhere along the process. You might also institute a policy that such attachments are only to be opened within an isolated testing environment, not on production systems.

CVD participants should keep in mind that their case tracking and email systems themselves present attack surface and may be affected by the very vulnerabilities they are designed to coordinate. We have witnessed reports containing examples of image parsing vulnerabilities causing problems for both webmail and ticketing systems that automatically generate thumbnail previews of image attachments. Vendors and coordinators concerned about such risks should consider the degree to which their CVD support infrastructure is integrated with normal business operations systems. In some scenarios, maintaining parallel infrastructure may be preferable.

## 7.3 CVD Staffing Considerations

Vulnerability analysis and response may require networking and forensics skills for certain classes of vulnerabilities, but often also requires some mix of the following skills:

- programming skills, especially in common languages (C, C++, Python, Java)
- reverse engineering and debugging
- knowledge of low-level operating system features for Windows, Mac and/or Linux
- hardware architecture and basic electrical engineering
- software security testing
- virtualization and some infrastructure automation
- written communications
- customer-service mindset

In most organizations, these skills will likely be dispersed among a team of people rather than expecting a single person to be fluent with all of these topics.

### 7.3.1 Beware Analyst Burnout

Some organizations may have a small enough flow of incoming vulnerability reports that all the CVD-related roles can be fulfilled by a single team, or even a single person. Other organizations might choose to split the technical analysis roles apart from the more human-oriented communication and coordination roles. No matter the arrangements, it is important that vendors and coordinators establishing a CVD capability mitigate the potential for analyst burnout.

Burnout of security analysts is well-documented phenomenon [141] [142] [143]. Analysts working full-time in a CVD process are at risk of this too. A vendor’s CVD capability may receive a large amount of incoming reports each week, especially at larger vendors. This can result in CVD staff becoming stressed and having low job satisfaction, leading to lower quality of work and ultimately employee attrition. The costs of lower quality work (e.g., missing an important report), employee turnover (e.g., hiring and training a new analyst), and associated damage to the vendor’s reputation suggest that this problem should be addressed ahead of time with reasonable precautions.

At the CERT/CC, we have attempted to mitigate this issue with reasonable success by implementing the suggestions below. Research has shown that many of these are effective responses to commonly-held morale problems [143].

- *Staying well-staffed and rotating responsibility.* Organizations may choose to have several team members, trained in the CVD process and tools, who can temporarily assist should a regular CVD analyst be unavailable for any reason, even if these additional team members do not typically do CVD day-to-day. Of course, handing off reports between temporary and full-time analysts leads to other operational concerns as previously discussed, so this must be done carefully. Organizations must also take care that these temporary team members are not pulled away from their own work so often that they themselves experience burnout.

A related possibility shared with us by a vendor is the possibility of *work rotation*, whereby team members are rotated in and out of CVD roles; rather than temporary, the rotation is permanent among a larger group of team members. An example would be an analyst spending one week in a CVD role, followed by two to three weeks on a different project or role. The same concerns in our above discussion would apply; organizations must be careful to balance time in and out of CVD roles in order to maximize the effectiveness of the rotation.

- *Allowing analyst independence.* Generally, you should trust your analysts to make good decisions during the report triage process, and empower them to make CVD decisions. Allowing analyst autonomy with management's specific blessing provides relief to analysts attempting to prioritize reports. Many reports will be incomplete, inaccurate, unimportant, or not actionable; allowing analysts to make the judgment on which reports deserve priority and which should be closed may help reduce work-related stress.
- *Allotting professional development time*. Analysts schedule some time each week to focus on professional development or projects. During these times, the analyst is not expected to perform CVD duties. Providing this time in whole-day chunks is preferable to spreading it out across the week. It is also important that during these days the analyst not be disturbed; urgent tasks should be handled by other on-duty analysts as much as possible. This suggestion may be combined with work rotation to allow for regular project work outside of the scope of CVD.
- *Seeking out automation*. We have encouraged analysts to document procedures and processes that need updating or could even be automated. Prototypes can be implemented and rolled out to decrease the cognitive workload of analysts. Processes and tools should be reviewed regularly to ensure they are aiding the analyst, rather than fighting the analyst.

Due to the possibility of burnout and the associated costs, the CERT/CC recommends that CVD capability be established within a well-resourced team or teams specifically created for this task, rather than concentrating the responsibilities to a small team, or even a single person. Our suggestions above may be helpful to combat analyst burnout, but do not form an exhaustive list of possible actions.

# 8 Open Problems in CVD

*Sometimes it seems as if there are more solutions than problems. On closer scrutiny, it turns out that many of today's problems are a result of yesterday's solutions.*

*-Thomas Sowell*

Although we have attempted to describe CVD as thoroughly as we can, there are a number of open problems in CVD. In this chapter, we cover two major topic areas that the CERT/CC has been working on in recent years. First, we discuss the increasing difficulty in identifying vulnerabilities. Second, we'll cover the changes that IoT implies to CVD now and for the foreseeable future.

## 8.1 Vulnerability IDs and DBs

The units of work in CVD are vulnerability reports or cases. However, a single case may actually address multiple vulnerabilities. Teasing out how many problems are involved in a report can be tricky at times. The implications of this in terms of the CVD process and the compilation of vulnerability databases is significant. This section is adapted from a CERT/CC blog post by Householder [144].

### 8.1.1 On the Complexities of Vulnerability Identity

Vulnerability identifiers can serve multiple purposes. They may be used to identify the following:

- a vulnerability report or case
- a document or database entry that describes a vulnerability (e.g., CERT Vulnerability Notes [15])
- the specific flaw that such a document or report describes [14]

Now this isn't really a problem as long as one case describes one vulnerability and that case results in the creation of one document. But that's not always the case, for a number of reasons, including those below:

- Different processes use different abstractions to define what "unit vulnerability" is. For example, CVE has specific guidance on counting rules [84].
- It's rare for vendors to release single-issue patches. More often they prefer to roll up multiple fixes into a single release, and then publish a document about the release [145].
- In the case of independent discovery, or at least duplicate reporting, multiple cases may be opened describing the same vulnerability. In some instances, this fact may not become obvious until considerable effort has been put into isolating the bugs in each report. For example, a single vulnerability can manifest in different ways depending on how it's triggered. The connection might only be discovered on root cause analysis.
- Automated testing such as fuzzing can lead to rapid discovery of very large numbers of unique failure cases that are difficult to resolve into specific bugs.

- Automated testing can also identify so many individual vulnerabilities that human-oriented case handling processes cannot scale to treat each one individually. Here's an extreme example of this phenomenon: although the CERT/CC published only a single Vulnerability Note for Android apps that failed to validate SSL certificates, in the end it covered 23,667 vulnerable apps [92] [93]. Should each get its own identifier? Yes, and we did assign individual VU# identifiers to each vulnerable app. But this highlights the distinction between the vulnerability and the document that describes it.

As of this writing, work is underway within the Vulnerability Report Data Exchange special interest group (VRDX-SIG) within FIRST [146] on a vulnerability report cross-reference data model that will allow for the expression of relationships between vulnerability reports. The current work in progress can be found at https://github.com/FIRSTdotorg/vrdx-sig-vxref-wip.

In order to make it easier to relate vulnerability reports and records to each other, the VRDX work represents the following concepts: "possibly related," "related," "not equal," "equal," "superset," "subset," and "overlap."

### 8.1.2 What CVE Isn't

Because of the prevalence and popular use of CVE IDs in the vulnerability response space, many people assume that vulnerability identity is synonymous with Common Vulnerabilities and Exposures (CVE) [14]. However, let's briefly look at some ways in which that assumption is inaccurate:

- CVE has limited scope of coverage.
- Not all known vulnerabilities are assigned a CVE ID.
- Not all vulnerabilities assigned a CVE ID have a corresponding record in the CVE database.

### 8.1.3 Every Vulnerability Database Makes Choices

As the CERT/CC's vulnerability analysis efforts have expanded into vulnerability coordination for non-traditional computing products (mobile, vehicles, medical devices, IoT, etc.) [147], we've also begun to hit up against another set of issues affecting vulnerability identities and compatibility across vulnerability databases (VDBs): namely, bias.

Steve Christey Coley and Brian Martin mention a number of biases that affect all VDBs in their BlackHat 2013 talk [148]:

- **Selection bias**. Not all products receive equal scrutiny. Not all vul reports are included in VDBs.
- **Publication bias**. Not all results get published. Some vuls are found but never reported to anyone.
- **Abstraction bias**. This bias is an artifact of the process that VDBs use to assign identifiers to vulnerabilities. (Is it 1 vul or 3? 23,667 or 1?)
- **Measurement bias**. This bias encompasses errors in how a vulnerability is analyzed, verified, and catalogued.

In an ideal scientific world, bias can be factored into analytical results based on the data collected. But VDBs don't exist solely in the service of scientific purity. Every vulnerability database or catalog makes choices driven by the business requirements and organizational environments in which those VDBs operate. These choices include the following:

1. **Sources of vulnerability information monitored.** Monitoring all the potential sources of vulnerability information is unrealistic for resource-constrained VDBs; to date we have found none that are not so constrained. This choice is one source of selection bias.
2. **Inclusion and exclusion criteria.** Rules that define what subset of records from the sources monitored will be included (or not) in the VDB must be decided. What kind of vulnerabilities does the VDB track? Is it platform specific? Is it just a single vendor collecting reports in its own products? Is it focused on a particular business sector? This choice is another source of selection bias.
3. **Content detail.** How much (and what kind of) detail goes into each record in a VDB is something that must be decided: for example, whether to include exploit information, workarounds, detection criteria, and so forth.
4. **Abstraction.** What is a "unit" vulnerability? Does this report represent one vul or many? That choice depends on what purpose the VDB serves. Christey and Martin cover this issue in their list of biases, describing it as "the most prevalent source of problems for analysis." CVE has made their abstraction content decision guidance available [149].
5. **Uncertainty tolerance.** How certain is the information included in the record? Is the goal of the VDB to be authoritative on first publication? Or can it tolerate being wrong sometimes in favor of getting things out more quickly?
6. **Latency tolerance.** How quickly do new records need to be placed in the VDB following the initial disclosure? This choice is a distinct tradeoff with uncertainty: consider the differences between breaking news coverage and a history book.
7. **Capacity constraints.** For a VDB, incoming vulnerability report volume is unconstrained while the capacity to consume and process those reports into database records is decidedly not (especially with humans in the loop, and as of this writing they still are).
8. **Users and consumers of the data.** Ultimately, a VDB must serve some useful purpose to some audience in order for it to continue to exist. There is a wide variety of uses for the information contained in VDBs (vulnerability scanning, vulnerability management systems, long-term trend analysis, academic research, quality improvement efforts, supporting acquisition or purchasing decisions, evaluating vendor process effectiveness, etc.), so it shouldn't be surprising that user requirements can drive many of the other choices the VDB operators have to make.

It's important to note that even if two vulnerability databases agree on the first four items in the list above (sources to watch, inclusion criteria, content detail, and abstraction), over time it's easy to wind up with completely distinct data sets due to the latter items (uncertainty tolerance, latency tolerance, capacity constraints, and user needs).

### 8.1.4 Where We Are vs. Where We Need to Be

The vulnerability databases you are probably most familiar with, such as the National Vulnerability Database (NVD) [150], Common Vulnerabilities and Exposures (CVE) [14], and the CERT

Vulnerability Notes Database [15] have historically focused on vulnerabilities affecting traditional computing platforms (Windows, Linux, OS X, and other Unix-derived operating systems) with only a smattering of coverage for vulnerabilities in other platforms like mobile or embedded systems, websites, and cloud services.

In the case of websites and cloud services this gap may be acceptable since most such services are effectively single instances of a large-scale distributed system and therefore only the service provider needs to apply a fix. In those cases, there might not be a need for a common identifier since nobody is trying to coordinate efforts across responsible parties. But in the mobile and embedded spaces, we definitely see the need for identifiers to serve the needs of both disclosure coordination and patch deployment.

Furthermore, there is a strong English language and English-speaking country bias in the major U.S.-based VDBs (hopefully this isn't terribly surprising). China has not one but two major VDBs: China National Vulnerability Database of Information Security (CNNVD) [151] and China National Vulnerability Database (CNVD) [152]. We have been working with CSIRTs around the world (e.g., JPCERT/CC [59] and NCSC-FI [58]) to coordinate vulnerability response for years and realize the importance of international cooperation and interoperability in vulnerability response.

Given all the above, and in the context of the surging prevalence of bug bounty programs, it seems likely that in the coming years there will be more, not fewer VDBs around the world than there are today. We anticipate those VDBs will cover more products, sectors, languages, countries, and platforms than VDBs have in the past.

Coordinating vulnerability response at local, national, and global scales requires that we have the means to relate vulnerability reports to each other, regardless of the process that originated them. Furthermore, whether they are driven by national, commercial, or sector-specific interests, there will be a need for interoperability across all those coordination processes and the VDBs into which they feed.

### 8.1.5 Vulnerability IDs, Fast and Slow

Over time, it has become clear that the days of the "One Vulnerability ID to Rule Them All" are coming to a close and we need to start planning for that change. As we've covered above, one of the key observations we've made has been the growing need for multiple vulnerability identifiers and databases that serve different audiences, support diverse business practices, and operate at different characteristic rates.

In his book *Thinking, Fast and Slow*, Daniel Kahneman describes human thought processes in terms of two distinct systems [153]:

- System 1: Fast, automatic, frequent, emotional, stereotypic, subconscious
- System 2: Slow, effortful, infrequent, logical, calculating, conscious

Making the analogy to CVD processes, notice that historically there has been a need for slower, consistently high-quality, authoritative vulnerability records, trading off higher latency for lower noise. Deconfliction of duplicate records happens before an ID record (e.g., a CVE record) is issued, and reconciliation of errors can be difficult. To date, this practice is the ideal for which many VDBs have strived. Those VDBs remain a valuable resource in the defense of systems and networks around the globe.

Yet there is a different ideal, just as valid: one in which vulnerability IDs are assigned quickly, possibly non-authoritatively, and based on reports of variable quality. This process looks more like “issue, then deconflict.” For this new process to work well, post-hoc reconciliation needs to become easier.

If you’re familiar with the gitflow process [154] in software development, you might recognize this distinction as analogous to the one between the *develop* and *master* branches of a software project. The bulk of the work happens in and around the *develop* branch, and only when things have settled out does the *master* branch get updated (and merge conflicts are as inevitable as death and taxes).

### 8.1.6 A Path Toward VDB Interoperability

As mentioned in Section 8.1.1 above, the FIRST VRDX-SIG is working on a vulnerability cross-reference scheme that would allow for widely distributed vulnerability ID assignments and VDBs to run at whatever rate is necessary, while enabling the ability to reconcile them later once the dust clears:

- When necessary, the CVD process could operate in System 1 for quick response, and clean up any resulting confusion afterwards.
- A tactical response-focused VDB might be able to tolerate more uncertainty in trade for lower latency.
- A VDB with more academic leanings could do a deep-dive analysis on root causes in exchange for having fewer records and higher latency.

The main idea was that VDB records can be related to each other in one of the following ways:

- equality and inequality (two records describe the same vulnerability or vulnerabilities, or they refer to different ones)
- superset and subset (one record is more abstract than the other)
- overlap (related but not fully contained)

This work builds on both prior work at the CERT/CC and Harold Booth and Karen Scarfone’s October 2013 IETF Draft Vulnerability Data Model [155]. However, while it would be great if we could get to a unified data model like the IETF draft for vulnerability information exchange eventually, for now the simplest thing that could possibly work seemed to be coming up with a way to relate records within or between vulnerability databases that explicitly addresses the choices and biases described above. The unified data model might be a longer way off, and we were anticipating the need to reconcile VDBs much sooner.

### 8.1.7 Looking Ahead

Everything we have discussed in this section is work in progress, and some things are changing rapidly on a number of related fronts. Nevertheless, while it’s hard to say how we’ll get there, it seems inevitable that we'll eventually reach a point where vulnerability IDs can be issued (and deconflicted) at the speed necessary to improve coordinated global vulnerability response while maintaining our ability to have high-quality, trusted sources of vulnerability information.

Here in the CERT/CC Vulnerability Analysis team, we recognize the need for slower, "correct-then-issue" vulnerability IDs as well as faster moving "issue-then-correct" IDs. We believe that there is room for both (and in fact many points in between). Our goal in participating in the VRDX-SIG is to enable interoperability between any willing VDBs. We intend to continue our efforts to build a better way forward that suits everyone who shares our interest in seeing that vulnerabilities get coordinated and disclosed, and that patches are created and deployed, all with an eye toward minimizing societal harm in the process.

## 8.2 IoT and CVD

Next we turn our attention to the implications that the Internet of Things brings to the CVD discussion.

"Smart things" are expected to outnumber traditional computers in the near future and will likely surpass mobile phones not long thereafter. IoT will have implications to the protection of privacy, opportunities for fraud and abuse, and ensuring safety. Every vulnerable thing becomes a potential point of leverage for an attacker to persist or maneuver laterally through a network. Immature security on IoT devices can leak information that could allow an attacker to gain a foothold.

Because many such systems and devices are expected to remain operationally useful for years or even decades with minimal intervention, it is especially important that their security be thoroughly understood prior to deployment. This section collects a number of issues we observed in the course of recent work done by the CERT Vulnerability Analysis team, and is adapted from a CERT/CC blog post by Householder [156].

### 8.2.1 Black Boxes

We identified issues such as the inclusion of networked appliances in a larger system where the appliances provided networked services based on sensor data. Enterprise security policy treated the device as a black box rather than a general-purpose computer with regard to patch levels, included software, and so forth. The attack vector posed by the sensor data interface had not been considered either.

### 8.2.2 Unrecognized Subcomponents

In a number of projects, we observed that while many systems were composed of highly specified off-the-shelf and custom components, the vendors providing those systems often could not identify the third-party subcomponents present in the delivered codebase. The problem can be as simple as not identifying statically linked libraries or as complicated as dealing with complex supply chains for code components.

### 8.2.3 Long-Lived and Hard-to-Patch

We observed various devices with wireless data capabilities embedded within a larger system yet little or no ability to patch the fielded systems except within very sparse service windows. Instances where physical contact with the device is required in order to update it can be especially problematic once vulnerabilities are discovered. (See Dan Geer’s talk at the Security of Things Forum for more on the “long-lived and not reachable” problem [157].)

### 8.2.4 New Interfaces Bring New Threats

We also encountered smart grid devices built out of a traditional electrical component coupled to an embedded Linux system to provide networked services. In a deployment context, the device was treated as an appliance. However, the impact of potential vulnerabilities in the general-purpose operating system embedded in the device had not been fully addressed.

### 8.2.5 Summarizing the IoT’s Impact on CVD

We anticipate that many of the current gaps in security analysis tools and knowledge will begin to close over the next few years. However, it may be some time before we can fully understand how the systems already available today, let alone tomorrow, will impact the security of the networks onto which they are placed. The scope of the problem does not appear to contract any time soon.

We already live in a world where mobile devices outnumber traditional computers, and IoT stand to dwarf mobile computing in terms of the sheer number of devices within the next few years. As vulnerability discovery tools and techniques evolve into this space, so must our tools and processes for coordination and disclosure. Assumptions built into the CVD process about disclosure timing, coordination channels, development cycles, scanning, patching, and so on, will need to be reevaluated in the light of hardware-based systems that are likely to dominate the future internet.

# 9 Conclusion

*When I was a boy and I would see scary things in the news, my mother would say to me, 'Look for the helpers. You will always find people who are helping.'*

*– Mister Rogers*

The scope of the citizenry affected by cybersecurity vulnerabilities has widened considerably in recent years. In the past, one might have argued that only computer users were affected by vulnerabilities and their disclosure: this is no longer the case. Affected users now include those who have smartphones, watch smart TVs, use credit cards or ATMs for banking and/or shopping, drive cars, fly in airplanes, go to the hospital for diagnostic imaging or intravenous medicine, live in houses with smart meters, and so forth. The list goes on to include nearly everyone, and "opting out" is not a viable position for most people to take.

In an ideal world, software would do exactly what we expect it to do, and nothing we don't want it to do.

In an ideal world, vendors would be receptive to finding out about vulnerabilities in their products, and would recognize the service provided to them by those who find and report problems. They would be motivated to place user safety, privacy, and security at the top of their priorities.

In an ideal world, human communications would be clear to all parties involved. Well-meaning parties would never misunderstand or misinterpret each other's words or intentions. People would always be polite, patient, humble, calm, without guile, and willing to put aside their own interests for those of others.

We do not live in an ideal world.

In the world we find ourselves occupying, software-based systems exhibit complex behaviors, increasingly exceeding the limits of human comprehension [158]. As a society, we have become capable of building things we don't fully understand. The difference between what a thing does and what you expect it to do can lead to uncertainty, confusion, fear, and vulnerability.

But it's not just the technology that falls short of our ideals. It should come as no surprise that humans have diverse emotions and motives. Values differ. Feelings get hurt, people get frustrated. Words are misinterpreted. Incentives promote individual choices that conflict with each other. What's good for the individual is sometimes bad for the collective, and vice-versa.

And so, we're left to muddle through. To confront each day as an opportunity to learn, another chance to improve, and make tomorrow start a little better than yesterday ended. We scan the horizon to reduce surprise. We test for flaws, we probe for weaknesses, and we identify recurring patterns and themes that lead to undesired outcomes. We fix what we can, mitigate what we can't fix, and remain vigilant over what we can't mitigate. We coordinate vulnerability disclosure because we realize we're all in this together.

Thanks for reading.

# Appendix A – On the Internet of Things and Vulnerability Analysis

This appendix is adapted from two CERT/CC Blog Posts [159] [156].

## IoT Vulnerability Discovery

In 2014 CERT performed a study of vulnerability discovery techniques for IoT systems. As we reviewed the literature, we found a number of techniques in common use. Here they are, ranked in approximately descending order of popularity in the research we surveyed:

1. **Reading documentation**: This includes product data sheets, protocol specifications, Internet Drafts and RFCs, manufacturer documentation and specs, patents, hardware documentation, support sites, bug trackers, discussion forums, FCC filings, developer documentation, and related information.
2. **Reverse engineering**: In most cases, this consists of reverse engineering (RE) binary firmware or other software to understand its function. However, there are instances in which merely understanding a proprietary file format is sufficient to direct further analyses. Hardware RE appears in some research, but has not been as prevalent as RE of software or file formats. As security researchers develop more hardware knowledge and skills (or as individuals with those skills become security researchers) we expect the prevalence of hardware RE to increase in the security literature.
3. **Protocol analysis**: Understanding the communication protocols used by a system is vital to identifying remotely exploitable vulnerabilities. This technique can take the form of simply sniffing traffic to find mistrusted input or channels, or reverse engineering a proprietary protocol enough to build a fuzzer for it. Decoding both the syntax and semantics can be important. In wireless systems, this technique can also take the form of using a software defined radio (SDR) to perform signal analysis, which for this purpose is essentially protocol analysis at a lower level of the stack.
4. **Modeling and simulation**: Threat modeling from the attacker perspective was mentioned in a handful of papers, as was modeling and simulation of either the system or its protocols for further analysis using mathematical techniques such as game or graph theory.
5. **Fuzzing**: Generating randomized input is a common way to test how a system deals with arbitrary input. Fuzzing of network protocols is a common method cited in a number of reports.
6. **Input or traffic generation and spoofing**: Unlike fuzzing, spoofing usually consists of constructing otherwise valid input to a system to cause it to exhibit unexpected behavior. Constructing bogus input from a valid or trusted source also falls into this category.
7. **Scanning**: Because most IoT are composed of multiple components, each of which may have its own architecture and code base, it is often the case that a researcher can find known vulnerabilities in systems simply by using available vulnerability scanning tools such as Nessus or Metasploit.

8. **Hardware hacking**: This technique involves interfacing directly with the electronics at the circuit level. It is a form of physical-level reverse engineering and can include mapping circuits and connecting with JTAG to dump memory state or firmware.
9. **Debugging**: This technique uses software-based or hardware-based debuggers. JTAG is a common hardware debugging interface mentioned in many reports.
10. **Writing code**: This technique involves developing custom tools to assist with extracting, characterizing, and analyzing data to identify vulnerabilities.
11. **Application of specialized knowledge and skills**: In some cases, just knowing how a system works and approaching it with a security mindset is sufficient to find vulnerabilities. Examples include RFID and ModBus.

Many of the techniques listed above are common to vulnerability discovery in the traditional computing and mobile world. However, the low-hanging fruit appears to hang much lower in the IoT than in traditional computing. From a security perspective, even mobile systems have a head start, although they are not as far along as traditional computing platforms. The fact is that many of the vulnerabilities found thus far in IoT would be considered trivial—and rightly so—in the more mature market of servers and desktop computing. Yet the relative scale of the IoT market makes even trivial vulnerabilities potentially risky in aggregate.

## IoT Vulnerability Analysis

In our review of recent security research that focused on vulnerability discovery in the Internet of Things, we identified several key differences between IoT and traditional computing and mobile platforms, including

1. **Limited instrumentation**: The vulnerability analyst's ability to instrument the system in order to test its security can be limited. Many of the systems comprise embedded devices that are effectively black boxes at the network level. On the surface, this limitation might appear to be beneficial to the security of the system; if it's hard to create an analysis environment, it might be difficult to find vulnerabilities in the system. However, the problem is that while a determined and/or well-resourced attacker can overcome such obstacles and get on with finding vulnerabilities, a lack of instrumentation can make it difficult even for the vendor to adequately test the security of its own products.
2. **Less familiar system architectures**: IoT architectures are often different from those most often encountered by the typical vulnerability analyst. In short, ARM is neither x86 nor IA64, and some embedded systems are neither. Although this limitation is trivially obvious at a technical level, many vulnerability researchers and analysts will have to overcome this skill gap if they are to remain effective at finding and remediating vulnerabilities in IoT.
3. **Limited user interfaces**: User interfaces on the devices themselves are extremely limited—a few LEDs, maybe some switches or buttons, and that's about it. Thus, significant effort can be required just to provide input or get the feedback needed to perform security analysis work.
4. **Proprietary protocols**: The network protocols used above the transport layer are often proprietary. Although the spread of HTTP/HTTPS continues in this space as it has in the traditional and mobile spaces, there are many extant protocols that are poorly documented or wholly undocumented. The effort required to identify and understand higher level protocols,

given sometimes scant information about them, can be daunting. Techniques and tools for network protocol inference and reverse engineering can be effective tactics. However, if vendors were more open with their protocol specifications, much of the need for that effort would be obviated.

5. **Lack of updatability**: Unlike most other devices (laptops, PCs, smartphones, tablets), many IoT are either non-updateable or require significant effort to update. Systems that cannot be updated become less secure over time as new vulnerabilities are found and novel attack techniques emerge. Because vulnerabilities are often discovered long after a system has been delivered, systems that lack facilities for secure updates once deployed present a long-term risk to the networks in which they reside. This design flaw is perhaps the most significant one already found in many IoT, and if not corrected across the board, could lead to years if not decades of increasingly insecure devices acting as reservoirs of infection or as platforms for lateral movement by attackers of all types.
6. **Lack of security tools**: Security tools used for prevention, detection, analysis, and remediation in traditional computing systems have evolved and matured significantly over a period of decades. And while in many cases similar concepts apply to IoT, the practitioner will observe a distinct gap in available tools when attempting to secure or even observe such a system in detail. Packet capture and decoding, traffic analysis, reverse engineering and binary analysis, and the like are all transferable as concepts if not directly as tools, yet the tooling is far weaker when you get outside of the realm of Windows and Unix-based (including OSX) operating systems running on x86/IA64 architectures.
7. **Vulnerability scanning tool and database bias**: Vulnerability scanning tools largely look for known vulnerabilities. They, in turn, depend on vulnerability databases for their source material. However, databases of known vulnerabilities—CVE [14], the National Vulnerability Database (NVD) [150], Japan Vulnerability Notes (JVN) [160] and the CERT Vulnerability Notes Database [15] to name a few—are heavily biased by their history of tracking vulnerabilities in traditional computing systems (e.g., Windows, Linux, OSX, Unix and variants). Recent conversations with these and other vulnerability database operators indicate that the need to expand coverage into IoT is either a topic of active investigation and discussion or a work already in progress. However, we can expect the existing gap to remain for some time as these capabilities ramp up.
8. **Inadequate threat models**: Overly optimistic threat models are de rigueur among IoT. Many IoT are developed with what can only be described as naive threat models that drastically underestimate the hostility of the environments into which the system will be deployed. (Undocumented threat models are still threat models, even if they only exist in the assumptions made by the developer.) Even in cases where the developer of the main system is security-knowledgeable, he or she often is composing systems out of components or libraries that may not have been developed with the same degree of security consideration. This weakness is especially pernicious in power- or bandwidth-constrained systems where the goal of providing lightweight implementations supersedes the need to provide a minimum level of security. We believe this is a false economy that only defers a much larger cost when the system has been deployed, vulnerabilities are discovered, and remediation is difficult.
9. **Third-party library vulnerabilities**: We observe pervasive use of third-party libraries with neither recognition of nor adequate planning for how to fix or mitigate the vulnerabilities they inevitably contain. When a developer embeds a library into a system, that system can

inherit vulnerabilities subsequently found in the incorporated code. Although this is true in the traditional computing world, it is even more concerning in contexts where many libraries wind up as binary blobs and are simply included in the firmware as such. Lacking the ability to analyze this black box code either in manual source code reviews or using most code analysis tools, vendors may find it difficult to examine the code's security.

10. **Unprepared vendors**: Often we find that IoT vendors are not prepared to receive and handle vulnerability reports from outside parties, such as the security researcher community. Many also lack the ability to perform their own vulnerability discovery within their development lifecycle. These difficulties tend to arise from one of two causes:

    a. The vendor is comparatively small or new and has yet to form a product security incident response capability.

    b. The vendor has deep engineering experience in its domain but has not fully incorporated the effect of network-enabling its devices into its engineering quality assurance (this is related to the inadequate threat model point above).

       Typically, vendors in the latter group may have very strong skills in safety engineering or regulatory compliance, yet their internet security capability is lacking. Our experience is that many IoT vendors are surprised by the vulnerability disclosure process. We frequently find ourselves having conversations that rehash two decades of vulnerability coordination and disclosure debates with vendors who appear to experience something similar to the Kübler-Ross stages of grief[3] during the process.

11. **Unresolved vulnerability disclosure debates**: If we have learned anything in decades of CVD at the CERT/CC, it is that there is no single right answer to most vulnerability disclosure questions. However, in the traditional computing arena, most vendors and researchers have settled into a reasonable rhythm of allowing the vendor some time to fix vulnerabilities prior to publishing a vulnerability report more widely. Software as a service (SAAS) and software distributed through app stores can often fix and deploy patches to most customers quickly. On the opposite end of the spectrum, we find many IoT and embedded device vendors for whom fixing a vulnerability might require a firmware upgrade or even physical replacement of affected devices. This diversity of requirements forces vendors and researchers alike to reconsider their expectations with respect to the timing and level of detail provided in vulnerability reports based on the systems affected. Coupled with the proliferation of IoT vendors who are relative novices at internet-enabled devices and just becoming exposed to the world of vulnerability research and disclosure, the shift toward IoT can be expected to reinvigorate numerous disclosure debates as the various stakeholders work out their newfound positions.

## IoT Parting Thoughts

Although vulnerability analysis for IoT has much in common with security research in traditional computing and mobile environments, there are a number of important distinctions outlined in this

[3] The Kübler-Ross stages of grief are denial, anger, bargaining, depression, and acceptance. See http://www.ekr-foundation.org/

appendix. The threats posed by these systems given their current proliferation trajectory are concerning.

Even as they become more common, it can be difficult to identify the threats posed to a network by IoT either alone or in aggregate. In the simplest sense one might think of it as a “hidden Linux” problem: How many devices can you find in your immediate vicinity containing some form of Linux? Do you know what their patch status is? Do you know how you’d deal with a critical vulnerability affecting them? Furthermore, while the hidden Linux problem isn’t going away any time soon, we believe the third-party library problem will long outlast it. How many vulnerable image parsers with a network-accessible attack vector share your home with you? How would you patch them?

Dan Geer [157] puts it thus:

> *[A]n advanced persistent threat, one that is difficult to discover, difficult to remove, and difficult to attribute, is easier in a low-end monoculture, easier in an environment where much of the computing is done by devices that are deaf and mute once installed or where those devices operate at the very bottom of the software stack, where those devices bring no relevant societal risk by their onesies and twosies, but do bring relevant societal risk at today’s extant scales much less the scales coming soon.*

We agree.

# Appendix B – Traffic Light Protocol

This appendix is reproduced from https://www.first.org/tlp [140].

## FIRST Standards Definitions and Usage Guidance — Version 1.0

### 1. Introduction

a. The Traffic Light Protocol (TLP) was created in order to facilitate greater sharing of information. TLP is a set of designations used to ensure that sensitive information is shared with the appropriate audience. It employs four colors to indicate expected sharing boundaries to be applied by the recipient(s). TLP only has four colors; any designations not listed in this standard are not considered valid by FIRST.

b. TLP provides a simple and intuitive schema for indicating when and how sensitive information can be shared, facilitating more frequent and effective collaboration. TLP is not a "control marking" or classification scheme. TLP was not designed to handle licensing terms, handling and encryption rules, and restrictions on action or instrumentation of information. TLP labels and their definitions are not intended to have any effect on freedom of information or "sunshine" laws in any jurisdiction.

c. TLP is optimized for ease of adoption, human readability and person-to-person sharing; it may be used in automated sharing exchanges, but is not optimized for that use.

d. TLP is distinct from the Chatham House Rule (when a meeting, or part thereof, is held under the Chatham House Rule, participants are free to use the information received, but neither the identity nor the affiliation of the speaker(s), nor that of any other participant, may be revealed), but may be used in conjunction if it is deemed appropriate by participants in an information exchange.

e. The source is responsible for ensuring that recipients of TLP information understand and can follow TLP sharing guidance.

f. If a recipient needs to share the information more widely than indicated by the original TLP designation, they must obtain explicit permission from the original source.

### 2. Usage

a. How to use TLP in email: TLP-designated email correspondence should indicate the TLP color of the information in the Subject line and in the body of the email, prior to the designated information itself. The TLP color must be in capital letters: TLP:RED, TLP:AMBER, TLP:GREEN, or TLP:WHITE.

b. How to use TLP in documents: TLP-designated documents should indicate the TLP color of the information in the header and footer of each page. To avoid confusion with existing control marking schemes, it is advisable to right-justify TLP designations. The TLP color should appear in capital letters and in 12-point type or greater.

RGB: TLP:RED : R=255, G=0, B=51, background: R=0, G=0, B=0 TLP:AMBER : R=255, G=192, B=0, background: R=0, G=0, B=0 TLP:GREEN : R=51, G=255, B=0, background: R=0, G=0, B=0 TLP:WHITE : R=255, G=255, B=255, background: R=0, G=0, B=0

CMYK: TLP:RED : C=0, M=100, Y=79, K=0, background: C=0, M=0, Y=0,
K=100 TLP:AMBER : C=0, M=25, Y=100, K=0, background: C=0, M=0, Y=0,
K=100 TLP:GREEN : C=79, M=0, Y=100, K=0, background: C=0, M=0, Y=0,
K=100 TLP:WHITE : C=0, M=0, Y=0, K=0, background: C=0, M=0, Y=0, K=100

### 3. TLP definitions

a. TLP:RED = Not for disclosure, restricted to participants only.

Sources may use TLP:RED when information cannot be effectively acted upon by additional parties and could lead to impacts on a party's privacy, reputation, or operations if misused. Recipients may not share TLP:RED information with any parties outside of the specific exchange, meeting, or conversation in which it was originally disclosed. In the context of a meeting, for example, TLP:RED information is limited to those present at the meeting. In most circumstances, TLP:RED should be exchanged verbally or in person.

b. TLP:AMBER = Limited disclosure, restricted to participants' organizations.

Sources may use TLP:AMBER when information requires support to be effectively acted upon, yet carries risks to privacy, reputation, or operations if shared outside of the organizations involved. Recipients may only share TLP:AMBER information with members of their own organization, and with clients or customers who need to know the information to protect themselves or prevent further harm. Sources are at liberty to specify additional intended limits of the sharing; these must be adhered to.

c. TLP:GREEN = Limited disclosure, restricted to the community.

Sources may use TLP:GREEN when information is useful for the awareness of all participating organizations as well as with peers within the broader community or sector. Recipients may share TLP:GREEN information with peers and partner organizations within their sector or community, but not via publicly accessible channels. Information in this category can be circulated widely within a particular community. TLP:GREEN information may not be released outside of the community.

d. TLP:WHITE = Disclosure is not limited.

Sources may use TLP:WHITE when information carries minimal or no foreseeable risk of misuse, in accordance with applicable rules and procedures for public release. Subject to standard copyright rules, TLP:WHITE information may be distributed without restriction.

### Notes:

1. This document uses “should” and “must” as defined by RFC-2119.

2. Comments or suggestions on this document can be sent to tlp-sig@first.org.

# Appendix C – Sample Vulnerability Report Form

This is a vulnerability report, typically sent from a reporter to a vendor. These reports may also be shared among other third parties, by the reporter, the vendor, or a coordinator.

This is a report example based on the CERT/CC's Vulnerability Reporting Form [79], and is not meant to be exhaustive of all possibilities. Please modify the sections and format as necessary to better suit your needs.

## Vulnerability Report

The information below should be handled as (choose one):

TLP:RED / TLP:AMBER / TLP:GREEN / TLP: WHITE

### Vulnerability

- Software/Product(s) containing the vulnerability:
- Vulnerability Description:
- How may an attacker exploit this vulnerability? (Proof of Concept):
- What is the impact of exploiting this vulnerability? (What does an attacker gain that the attacker didn't have before?)
- How did you find the vulnerability? (Be specific about tools and versions you used.)
- When did you find the vulnerability?

### Disclosure Plans

- I have already reported this vulnerability to the following vendors and organizations:
- Is this vulnerability being publicly discussed? YES/NO, if yes then provide URL.
- Is there evidence that this vulnerability is being actively exploited? YES/NO, if yes, then provide URL/evidence.
- I plan to publicly disclose this vulnerability...
  - on this date: (Please include your time zone.)
  - at this URL:

### Reporter

- Name:
- Organization:
- Email:
- PGP Public Key (ASCII Armored or a URL):
- Telephone:
- May we provide your contact information to third parties? YES/NO
- Do you want to be publicly acknowledged in a disclosure? YES/NO

## Additional Information

- Vendor Tracking ID, CERT Tracking ID, or CVE ID if known:
- Additional Comments:

# Appendix D – Sample Vulnerability Disclosure Document

The vulnerability disclosure document is also often referred to as a "security advisory," particularly if published by the vendor.

This is an example of a vulnerability disclosure document based on CERT/CC's Vulnerability Notes [15] format. It is not meant to be exhaustive of all scenarios. Please modify the sections and format as necessary to better suit your needs.

## Vulnerability Disclosure Document

### Overview

- Brief Vulnerability Description: (try to keep it to 1-2 sentences)

### Vulnerability ID

- CVE ID for this Vulnerability [14]:
- Any other IDs (vendor tracking ID, bug tracker ID, CERT ID, etc.):

### Description

- Software/Product(s) containing the vulnerability:
- Version number of vulnerable software/product:
- Product Vendor:
- Type of Vulnerability, if known: (see MITRE's CWE page [77] for list of common types of vulnerabilities)
- Vulnerability Description:
- How may an attacker exploit this vulnerability? (Proof of Concept):

### Impact

- What is the impact of exploiting this vulnerability? (What does an attacker gain that the attacker didn't have before?)

### CVSS Score

- CVSS:3.0/AV:?/AC:?/PR:?/UI:?/S:?/C:?/I:?/A:? – 0.0 (LOW/MEDIUM/HIGH/CRITICAL)
- (Provide the full CVSS vector, not only the score. If possible, provide guidance on the temporal and environmental metrics, not only the base metrics [80].)

### Resolution

- Version containing the fix:
- URL or contact information to obtain the fix:
- Alternately, if no fix is available, list workaround or mitigation advice below:

### Reporter

This vulnerability was reported/discovered by _____________.

### Author and/or Contact Info

For more information or questions, please contact:

- Name:
- Organization:
- Email:
- PGP Public Key (ASCII Armored or a URL):

### Disclosure Timeline

- Date of First Vendor Contact Attempt:
- Date of Vendor Response:
- Date of Patch Release:
- Disclosure Date:

(List more dates here as necessary to document your communication attempts.)

### References

(List reference URLs here: for example, vendor advisory, other disclosures, and links to advice on mitigating problems.)

# Appendix E – Disclosure Policy Templates

## NTIA Early Stage Template

The NTIA Early Stage Template focuses on vulnerability disclosure policy development in safety-critical industries, in which the potential for harm directly impacts public safety or causes physical damage (e.g., automobiles or medical devices), but the lessons are easily adaptable by any organization that builds or maintains its own software or systems. A discussion of issues and template policy is included.

https://www.ntia.doc.gov/files/ntia/publications/ntia_vuln_disclosure_early_stage_template.pdf

## Open Source Vulnerability Disclosure Framework

BugCrowd and CipherLaw created the Open Source Vulnerability Disclosure Framework, offered under a Creative Commons Attribution 4.0 International License. The framework "is designed to quickly and smoothly prepare your organization to work with the independent security researcher community while reducing the legal risks to researchers and companies." In addition to a policy template "written with both simplicity and legal completeness in mind," a guidance document is provided for setting up a vulnerability disclosure program.

https://github.com/bugcrowd/disclosure-policy

## U.S. GSA Vulnerability Disclosure Policy

The United States General Services Administration (GSA)'s Technology Transformation Service (TTS) provides its vulnerability disclosure policy as a public domain resource.

https://github.com/18F/vulnerability-disclosure-policy

## ENISA Good Practice Guide on Vulnerability Disclosure

The Good Practice Guide on Vulnerability Disclosure from European Union Agency for Network and Information Security (ENISA) includes an annotated vulnerability disclosure policy template as an Annex.

https://www.enisa.europa.eu/publications/vulnerability-disclosure/at_download/fullReport

## U.S. Department of Justice Framework for a Vulnerability Disclosure Program for Online Systems

The United States Department of Justice (DoJ) has published a white paper containing guidance aimed at developing vulnerability disclosure programs for online systems and services. This report makes a point to distinguish online systems and services from "third-party vulnerability disclosure and hands-on—rather than remote—examination of software, devices, or hardware" because of potentially distinct legal issues that may arise.

https://www.justice.gov/criminal-ccips/page/file/983996/download

The aforementioned report is one of many related white papers provided by the DoJ’s Computer Crime and Intellectual Property section.

https://www.justice.gov/criminal-ccips/ccips-documents-and-reports

## Where to Look for More

Numerous organizations have already posted their vulnerability disclosure policies. A wide variety of these policies can be found by searching the web for “vulnerability disclosure policy,” or “vulnerability disclosure program,” or by browsing third-party vulnerability disclosure (e.g., bug bounty) service providers’ hosted programs.

# Bibliography

*URLs are valid as of the publication date of this document.*

[1] B. Cancilla, "Return of the Browser Wars," August 2004. [Online]. Available: http://www.ibmsystemsmag.com/ibmi/trends/whatsnew/Return-of-the-Browser-Wars/. [Accessed 17 May 2017].

[2] A. Manion, "Vulnerability Note VU#713878 Microsoft Internet Explorer does not properly validate source of redirected frame," CERT/CC, 9 June 2004. [Online]. Available: https://www.kb.cert.org/vuls/id/713878. [Accessed 17 May 2017].

[3] Oxford Living Dictionaries (English), "process," [Online]. Available: https://en.oxforddictionaries.com/definition/process. [Accessed 17 May 2017].

[4] Kissel, Richard (Editor), "NISTIR 7298 Revision 2 Glossary of Key Information Security Terms," U.S. Department of Commerce, 2013.

[5] R. Caralli, J. H. Allen and D. W. White, CERT Resilience Management Model: A Maturity Model for Managing Operational Resilience, Addison-Wesley Professional, 2010.

[6] A. Shostack, Threat modeling: Designing for Security, John Wiley & Sons, 2014.

[7] F. Swiderski and W. Snyder, Threat Modeling, Microsoft Press, 2004.

[8] R. C. Seacord, The CERT C Secure Coding Standard, Pearson Education, 2008.

[9] F. Long, D. Mohindra, R. C. Seacord and D. a. S. D. Sutherland, The CERT Oracle Secure Coding Standard for Java, Addison-Wesley Professional, 2011.

[10] G. McGraw, Software Security: Building Security In, Addison-Wesley Professional, 2006.

[11] G. Peterson, P. Hope and S. Lavenhar, "Architectural Risk Analysis," 2 July 2013. [Online]. Available: https://www.us-cert.gov/bsi/articles/best-practices/architectural-risk-analysis/architectural-risk-analysis. [Accessed 23 May 2017].

[12] J. Ryoo, R. Kazman and P. Anand, "Architectural Analysis for Security," *IEEE Security & Privacy*, vol. 13, no. 6, pp. 52-59, 2015.

[13] A. Householder, "Like Nailing Jelly to the Wall: Difficulties in Defining "Zero-Day Exploit," CERT, 7 July 2015. [Online]. Available: https://insights.sei.cmu.edu/cert/2015/07/like-nailing-jelly-to-the-wall-difficulties-in-defining-zero-day-exploit.html. [Accessed 23 May 2017].

[14] MITRE, "Common Vulnerabilities and Exposures," [Online]. Available: https://cve.mitre.org/. [Accessed 16 May 2017].

[15] CERT/CC, "Vulnerability Notes Database," [Online]. Available: https://www.kb.cert.org/vuls. [Accessed 16 May 2017].

[16] SecurityFocus, "Vulnerabilities," [Online]. Available: http://www.securityfocus.com/bid. [Accessed 23 May 2017].

[17] ISO/IEC, "ISO/IEC 29147:2014 Information technology—Security techniques—Vulnerability disclosure," 2014.

[18] S. Christey and C. Wysopal, "Responsible Vulnerability Disclosure Process draft-christey-wysopal-vuln-disclosure-00.txt," February 2002. [Online]. Available: https://tools.ietf.org/html/draft-christey-wysopal-vuln-disclosure-00. [Accessed 17 May 2017].

[19] MSRC Ecosystem Strategy Team, "Coordinated Vulnerability Disclosure: Bringing Balance to the Force," 22 July 2010. [Online]. Available: https://blogs.technet.microsoft.com/ecostrat/2010/07/22/coordinated-vulnerability-disclosure-bringing-balance-to-the-force/. [Accessed 23 May 2017].

[20] Microsoft Security Response Center, "Coordinated Vulnerability Disclosure," Microsoft, [Online]. Available: https://technet.microsoft.com/en-us/security/dn467923.aspx. [Accessed 23 May 2017].

[21] M. Souppaya and K. Scarfone, "NIST Special Publication 800-40 Revision 3 Guide to Enterprise Patch Management Technologies," U.S. Department of Commerce, 2013.

[22] A. Arora, A. Nandkumar and R. Telang, "Does information security attack frequency increase with vulnerability disclosure? An empirical analysis," *Information Systems Frontiers*, vol. 8, no. 5, pp. 350-362, 2006.

[23] FIRST, "Forum for Incident Response and Security Teams," [Online]. Available: https://www.first.org/. [Accessed 17 May 2017].

[24] FIRST, "Vulnerability Coordination SIG," [Online]. Available: https://www.first.org/global/sigs/vulnerability-coordination. [Accessed 17 May 2017].

[25] National Telecommunications and Information Administration, "Multistakeholder Process: Cybersecurity Vulnerabilities," 15 December 2016. [Online]. Available: https://www.ntia.doc.gov/other-publication/2016/multistakeholder-process-cybersecurity-vulnerabilities. [Accessed 17 May 2017].

[26] Harm Reduction Coalition, "Principles of Harm Reduction," [Online]. Available: http://harmreduction.org/about-us/principles-of-harm-reduction/. [Accessed 23 May 2017].

[27] Harm Reduction Coalition, "What is harm reduction?" [Online]. Available: https://www.hri.global/what-is-harm-reduction. [Accessed 23 May 2017].

[28] A. Householder, "Systemic Vulnerabilities: An Allegorical Tale of SteampunkVulnerability to Aero-Physical Threats," August 2015. [Online]. Available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=442528. [Accessed 17 May 2017].

[29] I Am The Cavalry, "5 Motivations of Security Researchers," [Online]. Available: https://www.iamthecavalry.org/motivations/. [Accessed 17 May 2017].

[30] NTIA Awareness and Adoption Working Group, "Vulnerability Disclosure Attitudes and Actions: A Research Report from the NTIA Awareness and Adoption Group," 15 December 2016. [Online]. Available: https://www.ntia.doc.gov/files/ntia/publications/2016_ntia_a_a_vulnerability_disclosure_insights_report.pdf. [Accessed 6 June 2017].

[31] FIRST, "Ethics SIG," [Online]. Available: https://www.first.org/global/sigs/ethics. [Accessed 17 May 2017].

[32] Association for Computing Machinery, "ACM Code of Ethics and Professional Conduct," 16 October 1992. [Online]. Available: https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct. [Accessed 17 May 2017].

[33] USENIX, "System Administrators' Code of Ethics," 30 September 2003. [Online]. Available: https://www.usenix.org/system-administrators-code-ethics. [Accessed 17 May 2017].

[34] American Press Institute, "What is the purpose of journalism?" [Online]. Available: https://www.americanpressinstitute.org/journalism-essentials/what-is-journalism/purpose-journalism/. [Accessed 17 May 2017].

[35] Society of Professional Journalists, "SPJ Code of Ethics," 6 September 2014. [Online]. Available: https://www.spj.org/ethicscode.asp. [Accessed 17 May 2017].

[36] A. Ozment and S. E. Schechter, "Milk or wine: Does software security improve with age?" in *USENIX Security*, 2006.

[37] K. Matsudaira, "Bad Software Architecture Is a People Problem," *Communications of the ACM*, vol. 59, no. 9, pp. 42-43, September 2016.

[38] J. M. Wing, "A Symbiotic Relationship Between Formal Methods and Security," in *Proceedings of the Conference on Computer Security, Dependability and Assurance: From Needs to Solutions*, 1998.

[39] E. Bobukh, "Equation of a Fuzzing Curve — Part 1/2," 18 December 2014. [Online]. Available: https://blogs.msdn.microsoft.com/eugene_bobukh/2014/12/18/equation-of-a-fuzzing-curve-part-12/. [Accessed 23 May 2017].

[40] E. Bobukh, "Equation of a Fuzzing Curve — Part 2/2," 6 January 2015. [Online]. Available: https://blogs.msdn.microsoft.com/eugene_bobukh/2015/01/06/equation-of-a-fuzzing-curve-part-22/. [Accessed 23 May 2017].

[41] H. W. Rittel and M. M. Webber, "Dilemmas in a General Theory of Planning," *Policy Sciences,* vol. 4, no. 1973, pp. 155-169, June 1973.

[42] BBC, "Xbox password flaw exposed by five-year-old boy," 4 April 2014. [Online]. Available: http://www.bbc.com/news/technology-26879185. [Accessed 16 May 2017].

[43] Microsoft, "What is the Security Development Lifecycle?" [Online]. Available: https://www.microsoft.com/en-us/sdl/. [Accessed 16 May 2017].

[44] BSIMM, "BSIMM Framework," [Online]. Available: https://www.bsimm.com/framework/. [Accessed 16 May 2017].

[45] ISO/IEC, "ISO/IEC 30111:2013 Information technology—Security techniques—Vulnerability handling processes," 2013.

[46] Microsoft, "Microsoft Security Response Center," [Online]. Available: https://technet.microsoft.com/en-us/security/dn440717.aspx. [Accessed 23 May 2017].

[47] Cisco Systems, "Security Vulnerability Policy," [Online]. Available: https://www.cisco.com/c/en/us/about/security-center/security-vulnerability-policy.html. [Accessed 23 May 2017].

[48] FIRST, "FIRST Teams," [Online]. Available: https://www.first.org/members/teams. [Accessed 16 May 2017].

[49] CERT Division, "CSIRT Frequently Asked Questions (FAQ)," Software Engineering Institute, [Online]. Available: https://www.cert.org/incident-management/csirt-development/csirt-faq.cfm? [Accessed 16 May 2017].

[50] CERT Division, "Incident Management: Resources for National CSIRTs," Software Engineering Institute, [Online]. Available: https://www.cert.org/incident-management/national-csirts/index.cfm. [Accessed 16 May 2017].

[51] CERT, "List of National CSIRTs," [Online]. Available: https://www.cert.org/incident-management/national-csirts/national-csirts.cfm. [Accessed 23 May 2017].

[52] BugCrowd, "BugCrowd," [Online]. Available: https://bugcrowd.com/. [Accessed 23 May 2017].

[53] HackerOne, "HackerOne," [Online]. Available: https://www.hackerone.com. [Accessed 23 May 2017].

[54] SynAck, "SynAck," [Online]. Available: https://www.synack.com. [Accessed 23 May 2017].

[55] Cobalt Labs Inc., "Cobalt," [Online]. Available: https://cobalt.io/. [Accessed 23 May 2017].

[56] CERT, "Vulnerability Analysis," [Online]. Available: https://www.cert.org/vulnerability-analysis/. [Accessed 23 May 2017].

[57] National Cyber Security Centre Netherlands, "NCSC-NL," [Online]. Available: https://www.ncsc.nl/english. [Accessed 23 May 2017].

[58] NCSC-FI, "Finnish Communications Regulatory Authority / National Cyber Security Centre Finland," [Online]. Available: https://www.viestintavirasto.fi/en/cybersecurity.html.

[59] JPCERT/CC, "Japan Computer Emergency Response Team Coordination Center," [Online]. Available: https://www.jpcert.or.jp/english/. [Accessed 16 May 2017].

[60] U.S. Department of Homeland Security, "Information Sharing and Analysis Organizations (ISAOs)," [Online]. Available: https://www.dhs.gov/isao. [Accessed 23 May 2017].

[61] National Council of ISACs, "National Council of ISACs," [Online]. Available: https://www.nationalisacs.org/. [Accessed 23 May 2017].

[62] W. Dormann, "Supporting the Android Ecosystem," 19 October 2015. [Online]. Available: https://insights.sei.cmu.edu/cert/2015/10/supporting-the-android-ecosystem.html. [Accessed 23 May 2017].

[63] U.S. Food & Drug Administration, "Medical Device Reporting (MDR)," [Online]. Available: https://www.fda.gov/medicaldevices/safety/reportaproblem/. [Accessed 23 May 2017].

[64] National Highway Traffic Safety Administration, "File a Vehicle Safety Complaint," [Online]. Available: https://www-odi.nhtsa.dot.gov/VehicleComplaint/. [Accessed 23 May 2017].

[65] Federal Aviation Administration, "Report Safety Issues," [Online]. Available: https://www.faa.gov/aircraft/safety/report/. [Accessed 23 May 2017].

[66] NASA Office of the Chief Engineer, "NASA Lessons Learned," NASA Lessons Learned Steering Committee (LLSC), [Online]. Available: https://www.nasa.gov/offices/oce/functions/lessons/index.html. [Accessed 16 May 2017].

[67] European Commission, "Dual Use Controls: Commission proposes to modernise and strengthen controls on exports of dual-use items," 28 September 2016. [Online]. Available: http://europa.eu/rapid/press-release_IP-16-3190_en.htm. [Accessed 23 May 2017].

[68] FIRST, "Vulnerability Database Catalog," FIRST VRDX SIG, 17 March 2016. [Online]. Available: https://www.first.org/global/sigs/vrdx/vdb-catalog. [Accessed 16 May 2017].

[69] J. T. Chambers and J. W. Thompson, "National Infrastructure Advisory Council Vulnerability Disclosure Framework Final Report and Recommendations by the Council," 13 January 2004. [Online]. Available: https://www.dhs.gov/xlibrary/assets/vdwgreport.pdf. [Accessed 17 May 2017].

[70] J. C. Knight, "Safety critical systems: challenges and directions," in *ICSE '02 Proceedings of the 24th International Conference on Software Engineering*, Orlando, 2002.

[71] U.S. Department of Health & Human Services, "Health Information Privacy," [Online]. Available: https://www.hhs.gov/hipaa/. [Accessed 23 May 2017].

[72] U.S. Department of Education, "Family Educational Rights and Privacy Act (FERPA)," [Online]. Available: https://ed.gov/policy/gen/guid/fpco/ferpa/index.html. [Accessed 23 May 2017].

[73] Federal Trade Commission, "Children's Online Privacy Protection Rule ("COPPA")," [Online]. Available: https://www.ftc.gov/enforcement/rules/rulemaking-regulatory-reform-proceedings/childrens-online-privacy-protection-rule. [Accessed 23 May 2017].

[74] PCI Security Standards Council, "PCI Security," [Online]. Available: https://www.pcisecuritystandards.org/pci_security/. [Accessed 23 May 2017].

[75] Electronic Frontier Foundation, "Coders' Rights Project Vulnerability Reporting FAQ," [Online]. Available: https://www.eff.org/issues/coders/vulnerability-reporting-faq. [Accessed 17 May 2017].

[76] K. Price, "Writing a bug report - Attack Scenario and Impact are key!" 2 August 2015. [Online]. Available: https://forum.bugcrowd.com/t/writing-a-bug-report-attack-scenario-and-impact-are-key/640. [Accessed 17 May 2017].

[77] MITRE, "Common Weakness Enumeration (CWE)," [Online]. Available: https://cwe.mitre.org/. [Accessed 17 May 2017].

[78] MITRE, "Common Attack Pattern Enumeration and Classification," [Online]. Available: https://capec.mitre.org/. [Accessed 17 May 2017].

[79] CERT/CC, "Vulnerability Reporting Form," [Online]. Available: https://vulcoord.cert.org/VulReport/. [Accessed 17 May 2017].

[80] FIRST, "Common Vulnerability Scoring System," [Online]. Available: https://www.first.org/cvss. [Accessed 17 May 2017].

[81] MITRE, "Common Weakness Scoring System (CWSS) version 1.0.1," 5 September 2014. [Online]. Available: https://cwe.mitre.org/cwss/cwss_v1.0.1.html. [Accessed 17 May 2017].

[82] Security Focus, "BugTraq Archive," [Online]. Available: http://www.securityfocus.com/archive/1. [Accessed 23 May 2017].

[83] Seclists.org, "Full Disclosure Mailing List," [Online]. Available: http://seclists.org/fulldisclosure/. [Accessed 23 May 2017].

[84] MITRE, "Common Vulnerabilities and Exposures (CVE) Numbering Authority (CNA) Rules Version 1.1," 16 September 2016. [Online]. Available: https://cve.mitre.org/cve/cna/CNA_Rules_v1.1.pdf. [Accessed 16 May 2017].

[85] J. Postel, "Internet Protocol (RFC 760)," 1980.

[86] N. Brownlee and E. Guttman, "Expectations for Computer Security Incident Response," The Internet Society, 1998.

[87] S. Shepherd, "Vulnerability Disclosure: How Do We Define Responsible Disclosure?" SANS GIAC SEC Practical Repository, 2003.

[88] FIRST, "Multi-Party Coordination and Disclosure," [Online]. Available: https://www.first.org/global/sigs/vulnerability-coordination/multiparty. [Accessed 6 June 2017].

[89] Codenomicon, "The Heartbleed Bug," 29 April 2014. [Online]. Available: http://heartbleed.com/. [Accessed 16 May 2017].

[90] J. P. Lanza, "Vulnerability Note VU#484891 Microsoft SQL Server 2000 contains stack buffer overflow in SQL Server Resolution Service," 26 July 2002. [Online]. Available: https://www.kb.cert.org/vuls/id/484891. [Accessed 23 May 2017].

[91] W. Dormann, "Vulnerability Note VU#916896 Oracle Outside In 8.5.2 contains multiple stack buffer overflows," 20 January 2016. [Online]. Available: https://www.kb.cert.org/vuls/id/916896. [Accessed 23 May 2017].

[92] W. Dormann, "Vulnerability Note VU#582497 Multiple Android applications fail to properly validate SSL certificates," CERT/CC, 3 September 2014. [Online]. Available: https://www.kb.cert.org/vuls/id/582497. [Accessed 16 May 2017].

[93] W. Dormann, "Android apps that fail to validate SSL," 29 August 2014. [Online]. Available: https://docs.google.com/spreadsheets/d/1t5GXwjw82SyunALVJb2w0zi3FoLRIkfGPc7AMjRF0r4. [Accessed 16 May 2017].

[94] University of Oulu, "PROTOS Test-Suite: c06-snmpv1," 2002. [Online]. Available: https://www.ee.oulu.fi/research/ouspg/PROTOS_Test-Suite_c06-snmpv1. [Accessed 16 May 2017].

[95] I. A. Finlay, S. V. Hernan, J. A. Rafail, C. Dougherty, A. D. Householder, M. Lindner and A. Manion, "Multiple Vulnerabilities in Many Implementations of the Simple Network Management Protocol (SNMP),"

CERT/CC, 12 February 2002. [Online]. Available: https://www.cert.org/historical/advisories/CA-2002-03.cfm. [Accessed 16 May 2017].

[96] I. A. Finlay, "Vulnerability Note VU#854306 Multiple vulnerabilities in SNMPv1 request handling," CERT/CC, 12 February 2002. [Online]. Available: https://www.kb.cert.org/vuls/id/854306. [Accessed 16 May 2017].

[97] I. A. Finlay, "Vulnerability Note VU#107186 Multiple vulnerabilities in SNMPv1 trap handling," CERT/CC, 12 February 2002. [Online]. Available: https://www.kb.cert.org/vuls/id/107186. [Accessed 16 May 2017].

[98] B. Stock, G. Pellegrino and C. Rossow, "Hey, You Have a Problem: On the Feasibility of Large-Scale Web Vulnerability Notification," in *25th USENIX Security Symposium*, 2016.

[99] R. M. Axelrod, The Evolution of Cooperation, Revised ed., Basic books, 2006.

[100] D. R. Grimes, "On the Viability of Conspiratorial Beliefs," *PLOS One,* vol. 11, no. 1, p. e0147905, 26 January 2016.

[101] Black Hat, "Black Hat," [Online]. Available: https://www.blackhat.com/. [Accessed 23 May 2017].

[102] DEF CON, "DEF CON," [Online]. Available: https://www.defcon.org/. [Accessed 23 May 2017].

[103] USENIX, "USENIX Security Conferences," [Online]. Available: https://www.usenix.org/conferences/byname/108. [Accessed 23 May 2017].

[104] RSA, "RSA Conference," [Online]. Available: https://www.rsaconference.com/. [Accessed 23 May 2017].

[105] CanSecWest, "CanSecWest Vancouver 2018," [Online]. Available: https://cansecwest.com/. [Accessed 23 May 2017].

[106] Federal Trade Commission, "ASUSTeK Computer Inc., In the Matter of," 28 July 2016. [Online]. Available: https://www.ftc.gov/enforcement/cases-proceedings/142-3156/asustek-computer-inc-matter. [Accessed 16 May 2017].

[107] Federal Trade Commission, "HTC America Inc., In the Matter of," 2 July 2013. [Online]. Available: https://www.ftc.gov/enforcement/cases-proceedings/122-3049/htc-america-inc-matter. [Accessed 16 May 2017].

[108] Federal Trade Commission, "Fandango, LLC," 19 August 2014. [Online]. Available: https://www.ftc.gov/enforcement/cases-proceedings/132-3089/fandango-llc. [Accessed 16 May 2017].

[109] A. Askar, "Minecraft Vulnerability Advisory," 16 April 2015. [Online]. Available: http://blog.ammaraskar.com/minecraft-vulnerability-advisory/. [Accessed 23 May 2017].

[110] A. Ozment, "The Likelihood of Vulnerability Rediscovery and the Social Utility of Vulnerability Hunting," in *Workshop on Economics and Information Security*, 2005.

[111] M. Finifter, D. Akhawe and D. Wagner, "An Empirical Study of Vulnerability Rewards Programs," in *22nd USENIX Security Symposium*, 2013.

[112] L. Ablon and T. Bogart, "Zero Days, Thousands of Nights," RAND Corporation, 2017.

[113] T. Herr and B. Schneier, "Taking Stock: Estimating Vulnerability Rediscovery," 7 March 2017. [Online]. Available: https://ssrn.com/abstract=2928758. [Accessed 16 May 2017].

[114] B. Grubb, "Heartbleed disclosure timeline: who knew what and when," The Sydney Morning Herald, 15 April 2014. [Online]. Available: http://www.smh.com.au/it-pro/security-it/heartbleed-disclosure-timeline-who-knew-what-and-when-20140414-zqurk.html. [Accessed 23 May 2017].

[115] SerNet, "Badlock Bug," 12 April 2016. [Online]. Available: http://www.badlock.org/. [Accessed 23 May 2017].

[116] N. Perlroth, "Security Experts Expect 'Shellshock' Software Bug in Bash to Be Significant," 25 September 2014. [Online]. Available: https://www.nytimes.com/2014/09/26/technology/security-experts-expect-shellshock-software-bug-to-be-significant.html. [Accessed 23 May 2017].

[117] A. Sarwate, "The GHOST Vulnerability," 27 January 2015. [Online]. Available: https://blog.qualys.com/laws-of-vulnerabilities/2015/01/27/the-ghost-vulnerability. [Accessed 23 May 2017].

[118] A. Watts, C. Huang and L. Chih-chang. Tao: The Watercourse Way, Pantheon, 1975.

[119] M. Masnick, "For 10 Years Everyone's Been Using 'The Streisand Effect' Without Paying; Now I'm Going To Start Issuing Takedowns," 8 January 2015. [Online]. Available: https://www.techdirt.com/articles/20150107/13292829624/10-years-everyones-been-using-streisand-effect-without-paying-now-im-going-to-start-issuing-takedowns.shtml. [Accessed 23 May 2017].

[120] R. Devendra, "Key Elements of the Sprint Retrospective," 24 April 2014. [Online]. Available: https://www.scrumalliance.org/community/articles/2014/april/key-elements-of-sprint-retrospective. [Accessed 23 May 2017].

[121] CERT/CC, "Sending Sensitive Information," [Online]. Available: https://www.cert.org/contact/sensitive-information.cfm. [Accessed 24 May 2017].

[122] Symantec, "Symantec Desktop Email Encryption," [Online]. Available: https://www.symantec.com/products/information-protection/encryption/desktop-email-encryption. [Accessed 24 May 2017].

[123] The GnuPG Project, "GNU Privacy Guard," [Online]. Available: https://gnupg.org/. [Accessed 24 May 2017].

[124] B. Ramsdell and S. Turner, "RFC 5751 Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification," January 2010. [Online]. Available: https://tools.ietf.org/html/rfc5751. [Accessed 24 May 2017].

[125] Internet Security Research Group (ISRG), "Let's Encrypt," [Online]. Available: https://letsencrypt.org/. [Accessed 16 May 2017].

[126] The Enigmail Project, "Enigmail," [Online]. Available: https://www.enigmail.net/index.php/en/. [Accessed 24 May 2017].

[127] Gpg4win Initiative, "GNU Privacy Guard for Windows," [Online]. Available: https://www.gpg4win.org/. [Accessed 24 May 2017].

[128] "KGpg," [Online]. Available: https://utils.kde.org/projects/kgpg/. [Accessed 24 May 2017].

[129] G. Wassermann, "Reach Out and Mail Someone," 6 August 2015. [Online]. Available: https://insights.sei.cmu.edu/cert/2015/08/reach-out-and-mail-someone.html. [Accessed 24 May 2017].

[130] "White Source Software," [Online]. Available: https://www.whitesourcesoftware.com/. [Accessed 24 May 2017].

[131] "Black Duck Software," [Online]. Available: https://www.blackducksoftware.com. [Accessed 24 May 2017].

[132] "Sonatype," [Online]. Available: https://www.sonatype.com/. [Accessed 24 May 2017].

[133] "Synopsis," [Online]. Available: https://www.synopsys.com/. [Accessed 24 May 2017].

[134] "Flexera Software," [Online]. Available: https://www.flexerasoftware.com/. [Accessed 24 May 2017].

[135] TagVault.org, "SWID Tags," [Online]. Available: http://tagvault.org/swid-tags/. [Accessed 16 May 2017].

[136] National Institute of Standards and Technology, "Common Platform Enumeration (CPE)," [Online]. Available: https://scap.nist.gov/specifications/cpe/ [Accessed 16 May 2017].

[137] SPDX Workgroup, "Software Package Data Exchange," [Online]. Available: https://spdx.org/ . [Accessed 16 May 2017].

[138] CERT, "Dranzer," [Online]. Available: https://vuls.cert.org/confluence/display/tools/Dranzer. [Accessed 24 May 2017].

[139] CERT, "BFF - Basic Fuzzing Framework," [Online]. Available: https://vuls.cert.org/confluence/display/tools/CERT+BFF+-+Basic+Fuzzing+Framework. [Accessed 24 May 2017].

[140] FIRST, "TRAFFIC LIGHT PROTOCOL (TLP) FIRST Standards Definitions and Usage Guidance — Version 1.0," [Online]. Available: https://www.first.org/tlp. [Accessed 16 May 2017].

[141] B. Rothke, "Building a Security Operations Center (SOC)," 29 Feb 2012. [Online]. Available: https://www.rsaconference.com/events/us12/agenda/sessions/683/building-a-security-operations-center-soc. [Accessed 24 May 2017].

[142] S. Ragan, "Avoiding burnout: Ten tips for hackers working incident response," 30 April 2014. [Online]. Available: http://www.csoonline.com/article/2149900/infosec-careers/avoiding-burnout-ten-tips-for-hackers-working-incident-response.html. [Accessed 24 May 2017].

[143] S. C. Sundaramurthy, A. G. Bardas, J. Case, X. Ou, M. Wesch, J. McHugh and S. R. Rajagopalan, "A human capital model for mitigating security analyst burnout," in *Proceedings of the Eleventh Symposium on Usable Privacy and Security (SOUPS 2015)*, July 2015.

[144] A. Householder, "Vulnerability IDs, Fast and Slow," 11 March 2016. [Online]. Available: https://insights.sei.cmu.edu/cert/2016/03/vulnerability-ids-fast-and-slow.html. [Accessed 7 June 2017].

[145] N. Mercer, "Further simplifying servicing models for Windows 7 and Windows 8.1," 15 August 2016. [Online]. Available: https://blogs.technet.microsoft.com/windowsitpro/2016/08/15/further-simplifying-servicing-model-for-windows-7-and-windows-8-1/. [Accessed 24 May 2017].

[146] FIRST, "Vulnerability Reporting and Data eXchange SIG (VRDX-SIG)," [Online]. Available: https://www.first.org/global/sigs/vrdx. [Accessed 16 May 2017].

[147] D. Klinedinst, "Coordinating Vulnerabilities in IoT Devices," 27 January 2016. [Online]. Available: https://insights.sei.cmu.edu/cert/2016/01/coordinating-vulnerabilities-in-iot-devices.html. [Accessed 16 May 2017].

[148] S. Christey Coley and B. Martin, "Buying Into the Bias: Why Vulnerability Statistics Suck," in *BlackHat*, 2013.

[149] MITRE, "CVE Abstraction Content Decisions: Rationale and Application," 15 June 2005. [Online]. Available: https://cve.mitre.org/cve/editorial_policies/cd_abstraction.html. [Accessed 24 May 2017].

[150] National Institute of Standards and Technology, "National Vulnerability Database," [Online]. Available: https://nvd.nist.gov/. [Accessed 16 May 2017].

[151] CNNVD, "China National Vulnerability Database of Information Security," [Online]. Available: http://www.cnnvd.org.cn/. [Accessed 16 May 2017].

[152] CNVD, "China National Vulnerability Database," [Online]. Available: http://www.cnvd.org.cn/. [Accessed 16 May 2017].

[153] D. Kahneman, Thinking, Fast and Slow, Macmillan, 2011.

[154] V. Driessen, "A successful Git branching model," 5 January 2010. [Online]. Available: http://nvie.com/posts/a-successful-git-branching-model/. [Accessed 16 May 2017].

[155] H. Booth and K. Scarfone, "Vulnerability Data Model draft-booth-sacm-vuln-model-02," 25 April 2013. [Online]. Available: https://tools.ietf.org/html/draft-booth-sacm-vuln-model-02. [Accessed 16 May 2107].

[156] A. Householder, "Vulnerability Discovery for Emerging Networked Systems," 20 November 2014. [Online]. Available: https://insights.sei.cmu.edu/cert/2014/11/-vulnerability-discovery-for-emerging-networked-systems.html. [Accessed 16 May 2017].

[157] D. Geer, "Security of Things," 14 May 2014. [Online]. Available: http://geer.tinho.net/geer.secot.7v14.txt. [Accessed 16 May 2017].

[158] S. Arbesman, Overcomplicated: Technology at the Limits of Comprehension, Current, 2016.

[159] A. Householder, "What's Different About Vulnerability Analysis and Discovery in Emerging Networked Systems?" 6 January 2015. [Online]. Available: https://insights.sei.cmu.edu/cert/2015/01/-whats-different-about-vulnerability-analysis-and-discovery-in-emerging-networked-systems.html. [Accessed 16 May 2017].

[160] JPCERT/CC and IPA, "Japan Vulnerability Notes," [Online]. Available: https://jvn.jp/en/. [Accessed 16 May 2017].

[161] O. H. Alhazmi, Y. K. Malaiya and I. Ray, "Measuring, analyzing and predicting security vulnerabilities in software systems," *Computers & Security,* vol. 26, no. 3, pp. 219-228, 2007.

[162] Wikipedia, "Wicked problem," [Online]. Available: https://en.wikipedia.org/wiki/Wicked_problem. [Accessed 5 June 2017].

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY
(Leave Blank)

2. REPORT DATE
August 2017

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE
The CERT® Guide to
Coordinated Vulnerability Disclosure

5. FUNDING NUMBERS
FA8721-05-C-0003

6. AUTHOR(S)
Allen D. Householder
Garret Wassermann
Art Manion
Chris King

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER
CMU/SEI-2017-SR-022

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFLCMC/PZE/Hanscom
Enterprise Acquisition Division
20 Schilling Circle
Building 1305
Hanscom AFB, MA 01731-2116

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
n/a

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified/Unlimited, DTIC, NTIS

12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

Security vulnerabilities remain a problem for vendors and deployers of software-based systems alike. Vendors play a key role by providing fixes for vulnerabilities, but they have no monopoly on the ability to discover vulnerabilities in their products and services. Knowledge of those vulnerabilities can increase adversarial advantage if deployers are left without recourse to remediate the risks they pose. Coordinated Vulnerability Disclosure (CVD) is the process of gathering information from vulnerability finders, coordinating the sharing of that information between relevant stakeholders, and disclosing the existence of software vulnerabilities and their mitigations to various stakeholders including the public. The CERT Coordination Center has been coordinating the disclosure of software vulnerabilities since its inception in 1988. This document is intended to serve as a guide to those who want to initiate, develop, or improve their own CVD capability. In it, the reader will find an overview of key principles underlying the CVD process, a survey of CVD stakeholders and their roles, and a description of CVD process phases, as well as advice concerning operational considerations and problems that may arise in the provision of CVD and related services.

S14. SUBJECT TERMS
Coordinated Vulnerability Disclosure, CVD, vulnerability response process, vulnerability report, CERT-CC, CSIRT, PSIRT, software vulnerability, software security

15. NUMBER OF PAGES
121

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT
UL