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. 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.

References