GCVE-BCP-02 - Practical Guide to Vulnerability Handling and Disclosure

Contents

0.1 Practical Guide to Vulnerability Handling and Disclosure

GCVE.eu

- **Version**: 1.2
- **Status**: Draft (for Public Review)
- **Date**: 2025-07-28
- **Authors**: GCVE Working Group
- **BCP ID**: BCP-02

0.1.1 Introduction

Vulnerabilities in software can pose serious risks to users and organizations. A clear and effective process for handling and disclosing security vulnerabilities maintains user trust and protects systems.

This guide provides actionable recommendations for GCVE GNA, software developers, open source project maintainers, vendors, and organizations to manage vulnerability reports from discovery to resolution and public disclosure. It is organized into key stages of a vulnerability's life-cycle, including:

- **preparation and receipt of a report,**
- **investigation and remediation,**
- **communication and coordinated disclosure.**

Overall, the guide establishes a transparent process that encourages responsible reporting and safeguards users.

0.1.2 Definitions

Vulnerability: A flaw or weakness in a software system that could be exploited to compromise the system's security or functionality.

Vulnerability Report: Information provided about a potential vulnerability in a product or service. A report typically includes details needed to reproduce the issue, such as affected components, steps to trigger the bug, and expected vs. actual behavior.

Coordinated Vulnerability Disclosure (CVD): A practice where the reporter and the vendor collaborate privately to resolve a vulnerability before public disclosure. The vulnerability details are kept confidential until a fix or mitigation is available, at which point an advisory is published. This coordination helps protect users by ensuring vulnerabilities are not disclosed without a remediation in place.

Advisory: A public notice that a vulnerability has been identified and fixed. Advisories are typically published by the vendor to inform users about the issue's impact, affected versions, and how to obtain the patch or mitigation.

0.1.3 Roles and Responsibilities

Effective vulnerability handling involves several stakeholders with distinct roles. Defining responsibilities for each role helps ensure accountability and clarity during the process:

- **Submitter or Reporter (Security Researchers or Users):** The person or team who discovers a potential vulnerability and reports it to the appropriate party (usually the vendor or maintainer). This could be an independent security researcher, a user, a member of the development team, or a third-party analyst. Submitters or reporters are expected to investigate and disclose vulnerabilities in good faith.
- **Submitter/Reporter and Vendor Interaction:** The submitter or reporter should follow the vendor's reporting guidelines, avoid publicizing the issue before giving the vendor a chance to fix it, and provide sufficient detail to reproduce the problem. Ethical reporters respect privacy and legal boundaries during testing, and they do not exploit the vulnerability for personal gain. When participating in a bug bounty program, they adhere to its scope and rules.
- **Vendors / Maintainers**: Vendors and open source project maintainers act on vulnerability reports promptly and professionally. They have a clear intake mechanism (e.g. a security contact email or form), acknowledge and maintain open communication with reporters, prioritize, analyze and fix issues based on risk, and communicate updates about confirmed vulnerabilities and their fixes to all users. Vendors must also avoid hostile responses like threatening legal action against those who report issues in good faith.
- **Product Security Team (PSIRT)**: An organization's PSIRT or designated security response team, when present, coordinates the entire handling of the vulnerability response process. They receive and triage incoming reports, involve relevant development teams, track remediation progress, and ensure communication flows to reporters and management. The PSIRT also works on preparing advisories and post-remediation follow-ups. In an open source context,

the PSIRT may simply consist of the core maintainers or a subset of project contributors who handle security issues.

- **Developers and QA Engineers**: Once a report is validated, developers are responsible for developing the code fixes or mitigations. QA or testing teams (or in smaller projects, the developers themselves) must test the fixes to confirm the vulnerability is resolved and that no new issues are introduced. They also verify that the fix doesn't negatively impact functionality. Development and QA staff should treat security fixes with high priority and follow secure coding practices to prevent re-introduction of similar flaws.
- **Secure Coding Practices**: Secure coding is the practice of developing computer software in such a way that guards against the accidental introduction of security vulnerabilities. Defects, bugs and logic flaws are consistently the primary cause of commonly exploited software vulnerabilities.
- **Coordinators (Third-Party Facilitators)**: In some cases, a coordinator such as a Computer Emergency Response team (CERT) or a vulnerability broker can ensure responsible information sharing and awareness among multiple vendors impacted by a vulnerability. They facilitate information sharing when the same issue affects multiple vendors or when a reporter and vendor require a neutral intermediary. Coordinators help synchronize remediation timelines among all affected parties and can assist in broader communications, such as publishing a joint advisory.
- **Users/Customers**: While users are the beneficiaries of the process rather than active participants in handling, they are ultimately responsible for applying updates and mitigations that vendors release. Vendors should simplify the process for users to learn about security updates and take action. Users, on their part, should stay informed via the channels the vendor provides (mailing lists, security pages, etc.) and promptly install security patches to protect their systems.

By clearly identifying these roles and expectations, an organization or project can ensure well-managed contributions towards swiftly resolving vulnerabilities when they arise.

0.1.4 Preparing for Vulnerability Handling

Preparation is critical **before** any vulnerability reports come in. Having a plan and resources in place will make the handling process far more efficient and effective. Key preparation steps include:

- **Establish a Vulnerability Disclosure Policy**: Create a clearly written document that outlines how people should report vulnerabilities to you and what they can expect in return. This policy should be easy to find and may often be a publicly accessible policy on a website or a `SECURITY.md` file in open source projects. It should specify the preferred contact method and provide links to a dedicated security email address or web form. The policy should detail what information to include in a report and outline your commitment to researchers. These commitments

should capture any expectations for communication, acknowledgement, and not pursuing legal action for good-faith research. If you have a bug bounty program or reward offering, mention how it works and the scope of vulnerabilities covered. Also include any safe harbor statement reassuring researchers that if they follow the guidelines, they will not be penalized.

- **Designate a Security Response Team**: Assign specific individuals from your organization or project team to the defined roles and responsibilities for handling vulnerability reports. These may be members of the security or engineering team in a company or maintainers in an open source project. These roles should provide for receiving, assessing, and prioritizing reports, fixing the vulnerability, and communicating with stakeholders including the reporter, vendors, and users. Make sure these individuals know their roles and have the authority to coordinate fixes across development teams.
- **Define an Internal Workflow**: Establish what to do when a report arrives:
  - Communicate issue Routing: Everyone in the organization should know how to escalate a security issue to the response team. For example, even customer support or developers who accidentally receive a report should know where to forward it.
  - Set Up Secure Communication Channels: Provide secure and easy-to-use reporting channels. Commonly, a security contact email (e.g. security@example.com) should be combined with a public PGP/GPG key so that sensitive vulnerability details can be encrypted in-transit. Web-based submission forms or ticket systems that restrict reports from public view and use encryption (HTTPS) can also provide a secure channel. If using an issue tracker like those in GitHub or GitLab, instruct reporters to not post vulnerabilities in public tickets; instead provide a private reporting mechanism or use the platform's private disclosure feature if available.
  - Develop Internal Procedures and Guidelines: Document how your team will handle vulnerability reports step by step. This internal guide should include at a minimum the following items:
  - Document triage criteria. Provide an internal guide for how to prioritize reports. Severity rating systems like CVSS can help categorize issues.
  - Publish target timelines. Each step in handling process should have a commitment for resolution, such as acknowledging report receipt within two business days and providing a plan for resolution within one week. Critically, a commitment should be posted about resolving vulnerabilities within a specific timeframe.
  - Establish the technical process: Identify how you will analyze vulnerability reports, develop and test fixes, and deploy patches.
  - Maintain accountability: Set up a controlled and auditable system to track vulnerability cases from report to resolution – this could be a special project in your issue tracker or

a simple internal spreadsheet/ticket system. Tracking helps ensure nothing falls through the cracks and allows management to review metrics like response times.

- **Practice and Training**: Treat vulnerability handling as a process that benefits from practice. Run simulations or tabletop exercises that reinforce what workflows to follow when someone reports a critical software bug. Ensure the response team is familiar with the procedures and tools, as many items like encrypted emails and coordinating CVE requests are not frequently used. Train developers on secure coding and common vulnerability types so that they can respond more effectively and ideally prevent issues. Commit to continuous improvement through updating your plan periodically based on these dry runs or any real incidents.
- **Resource Allocation**: Management should allocate adequate resources for vulnerability handling. This means having people with time to investigate and fix security issues on short notice and possibly budgeting for external help if needed. For instance, payments for penetration testing or bug bounties may require financing. Without proper resourcing and clear management support, even a well-documented process can falter. Make sure everyone prioritizes security bug fixes as a part of the development lifecycle rather than as an afterthought.

Advance preparations create a foundation that makes the vulnerability handling workflow run smoothly. It signals to researchers and users alike that your organization takes security seriously and can readily respond.

0.1.5 Receiving and Handling Vulnerability Reports

Vulnerability reports require effective stewardship and communication regarding their resolution to maintain trust and accountability with the research and user communities. Key management practices include:

- **Logging and Tracking**: As part of handling incoming reports, log each report in your internal tracking system. Record details such as the date received, reporter name/contact, affected product, a short description of the issue, its status (e.g. “under investigation”, “fix in progress”), and any relevant deadlines. For instance, if a reporter has indicated they intend to publish after 90 days, note that date. Tracking helps you manage multiple reports at once and ensures accountability. It also creates a record useful for post-incident review and improving your process over time.
- **Acknowledge Receipt Quickly**: The **intake** and **initial response** sets the tone for the entire handling process. Upon receiving a vulnerability report, assign a tracking ID for future correspondence and following the report throughout verification and resolution. Acknowledge the report as soon as possible – ideally within 24-48 hours. A simple receipt confirmation that indicates a planned review assures the reporter that you value and will act upon their effort. This can prevent a frustrated researcher from going public prematurely.

- **Ensure Confidentiality**: Treat all vulnerability reports as sensitive information. Restrict knowledge of the details to the people who need to know (your security team, relevant developers, management as appropriate). If a report comes in through a public channel by mistake (e.g. a public bug tracker or social media), move the conversation to a private channel promptly and ask the reporter to take down public details if possible. This limits exposure of the vulnerability until a fix is ready and reduces the window of opportunity for malicious actors.
- **Initial Triage (Severity and Scope Assessment)**: Your security response team should quickly triage the report. Determine what product the vulnerability affects and whether it involves a core component or optional module. Identify the potential vulnerability type and its severity. A defined severity scoring system like CVSS can isolate the severity of specific vulnerabilities separately from their risk (e.g. does it allow remote code execution, information disclosure, denial of service, etc.). Also assess the scope considering the number of users likely affected. During this phase, it's fine to ask the reporter for clarification or additional details if needed to understand the issue. Based on this initial assessment, you can prioritize the issue. For example, critical issues trigger an emergency response whereas low-risk issues go into the normal development queue.
- **Verify the Report**: Before immediately developing a fix, reproduce and verify the issue to confirm that it is a genuine vulnerability. This may involve your engineers replicating the steps provided by the reporter or creating a proof-of-concept exploit in a safe test environment. Verification should also document which versions/configurations are affected and what the exact impact is (e.g. can an attacker actually steal data, or is it a minor glitch?). If the report cannot be verified or appears to be incorrect, involve the reporter – let them know if additional evidence is needed or explain why it might not be considered a security issue. In some cases, what the reporter found might be a known issue or expected behavior; communicate that respectfully if so.
- **Duplicate or Out-of-Scope Issues**: If you discover that a reported issue duplicates something already reported or affects a component outside your responsibility (for example, a bug in a third-party library), explain the situation to the reporter. For duplicates, thank them and inform them of the issue's previous resolution. If possible, indicate the tracking number or status of the existing issue. For third-party issues, you might need to act as a coordinator and relay the report to the third-party vendor or to an appropriate coordinating center. Similarly, keep the reporter advised of this step. If the issue is in an older end-of-life product version that you no longer support, let the reporter know the product is not supported. However, if the vulnerability is critical, you might still consider informing users or updating documentation about the risk.
- **Ongoing Communication**: Throughout the handling of the report, maintain communication with the reporter. After the initial acknowledgement, update them when verification is done to confirm results from the verification or to ask for more information to reproduce the issue. Even if you don't have significant progress, a periodic update (e.g. "We are still working on a fix,

thank you for your patience") can reassure reporters. Communication provides transparency and communicates your commitment towards resolving vulnerabilities. This collaborative approach keeps the process coordinated and reduces the likelihood of surprise disclosures.

Receiving and handling reports properly builds trust with researchers. If they see you are responsive and professional, they are more likely to continue reporting to you rather than announcing issues publicly out of frustration. It also signals to your user community that you can deal with security issues seriously and efficiently.

0.1.6 Investigating and Resolving Vulnerabilities

Once a vulnerability report has been validated, begin investigating its root cause and implementing a resolution. During this phase development and security teams work together to eliminate the weakness. Important steps in investigation and resolution include:

- **In-Depth Analysis**: The team should thoroughly investigate the vulnerability. Root cause evaluations determine which part of the code or design is at fault. Understanding root cause is vital not only to fixing the current bug but also to identifing any related issues. Investigate if similar components might have the same flaw and whether the vulnerability has resided throughout other versions. Also, analyze the impact achieved by an attacker exploiting the vulnerability under different conditions. This helps in assessing severity and priority for the fix . For instance, a vulnerability allowing full system compromise is critical, whereas one causing a minor information leak might be lower priority). Formal severity scoring, like CVSS, can be useful to quantify impact and guide prioritization.
- **Developing a Fix or Mitigation**: Task the appropriate development team to create a remediation for the issue. This often means writing a patch to the software's code. If a full fix might take too much time, consider if a mitigation or workaround could reduce the interim risk. For example, instructing users to disable a vulnerable feature or providing a configuration change that blocks an exploit might be a temporary measure. Balance speed and quality when developing fixes – urgent fixes should be expedited, but even a quick fix needs at least some testing to ensure it resolves the problem without causing regressions. In extreme cases involving an actively exploited critical vulnerability, a vendor might release a temporary fix or even temporarily take a service offline until a proper patch is ready. These emergency measures underscore the importance of having a robust fix as soon as possible.
- **Testing the Fix**: Thorough testing verifies that the fix indeed closes the vulnerability and does not introduce new bugs or break functionality. Testing on all relevant platforms or versions of the software ensures consistency. Where practical, involving the original reporter in testing a patch or a secure test build can provide valuable confirmation that the vulnerability is resolved from an attacker's perspective. This also further engages the reporter in the process. If the re-

porter is not available for testing or the disclosure is sensitive, rigorous internal testing should leverage initial report descriptions, root causes and conditions identified during verification and development, and edge cases that could be related to the vulnerability.

- **Collateral Cleanup**: Investigating a vulnerability might reveal other issues or similar vulnerable code elsewhere. Take this opportunity to clean up related problems. For example, if the issue was a result of an insecure library or dependency, update that dependency across your project. If the root cause was a specific coding pattern, a quick scan of the codebase can reveal instances of that pattern. This proactive approach can prevent future vulnerabilities. It's also a good practice to check if any telemetry, incident reports, or logs provide evidence that the vulnerability was exploited. If so, this necessitates incident response.
- **Decision on Public Release Timing**: Plan when and how the fix will be released. Typically, you will coordinate the release of the patched software with the publication of a security advisory (see the sections on Communication and Publishing Advisories). If you have a regular release schedule, decide if this warrants an out-of-cycle emergency release or if it can wait for the next planned update. Security patches for critical issues often justify quicker releases. If the vulnerability is not very risky, bundling it in a scheduled release might be acceptable. In all cases, plan to release the fix before or at the same time as disclosing the vulnerability details to the public, to minimize user exposure.
- **Documenting the Fix**: When releasing the fix, prepare internal documentation of what was done. This includes updating any internal security knowledge base about the nature of the issue and how it was fixed for future reference. If a CVE (Common Vulnerabilities and Exposures) or GCVE ID is needed (and not already assigned by a reporter or coordinator), you might request one at this stage. Many open source projects can get CVEs via CNA coordination or GCVE via GNA or through portals like GitHub Security Advisories. One or more ID will be referenced in the public advisory to track the issue in vulnerability databases.

Diligent investigation and efficient remediation eliminate a vulnerability's threat. It's important to keep momentum throughout resolution – lengthy delays in fixing known vulnerabilities leave users at risk. Aim for a balance among scope, severity, and risk: address the issue as fast as possible, but also correctly and safely. Once a fix is ready and verified, begin preparations to communicate both the fix and disclose the vulnerability.

0.1.7 Communicating with Submitters and Users

Communication connects the entire vulnerability handling process. Its management focuses on the vulnerability reporter and the users or customers of the affected product.

- **Communication with the Reporter:** From the moment a report is received until after the issue is resolved, it's best practice to keep the reporter informed and engaged. Key communication

milestones with reporters include:

- **Acknowledgment**: As noted earlier, confirm to the reporter that you received their report and provide a reference ID for it. Thank them for alerting you.
- **Verification Results**: After you investigate, let the reporter know the outcome. If you verified the vulnerability, say so and that you've begun remediation efforts. If you could not reproduce the issue or determined it's not a vulnerability, provide that feedback. Be tactful and appreciative – even if a report turns out not to be a valid security issue, acknowledge the effort and explain your reasoning to avoid discouraging future reports.
- **Updates During Remediation**: While working on the fix, send periodic updates. Although these do not need to share full technical details, they should inform reporters of progress towards identifying root causes, developing patches, and testing fixes. If there are delays or unforeseen complications, be honest about them and your continued work towards resolution. Regular communication keeps the reporter on your side and maintains trust.
- **Coordination on Disclosure**: As the fix nears completion, coordinate the public disclosure with the reporter. If you plan to publish an advisory, let them know the expected timeline. If the reporter had initially set a disclosure deadline, update them if you need more time and negotiate if necessary. Most researchers are willing to extend time if they see progress and good faith from the vendor. Discuss whether the reporter would like to be credited in the advisory or if they prefer anonymity. Also, agree on the date and time of public disclosure. Ideally, release the advisory simultaneously with or shortly after releasing patched software so that users can immediately protect themselves.
- **Post-Resolution Follow-up**: After the fix is released and the advisory is public, it's courteous to follow up with the reporter. Thank them again for their responsible disclosure and perhaps share any lessons learned or improvements you plan because of their report. Keeping a positive relationship can lead to the reporter helping you again in the future or becoming an advocate for your project's security posture.

Throughout all communications with reporters, maintain a professional, collaborative, and appreciative tone. Remember that the broader security community will judge vendors by how they treat researchers – being responsive and fair will enhance your reputation.

- **Communication with Users and Stakeholders:** Users of your software need to know about vulnerabilities in order to protect themselves. However, user communication is typically done **after** or at the time a fix is available to avoid alerting attackers. Key points for user communication include:
- **Security Advisories**: As detailed in the next section, a security advisory is the primary vehicle to inform users. Ensure the advisory is easily accessible – for example, via a dedicated security page on your website, a mailing list announcement, release notes, or a blog post on your official

blog. In an enterprise setting, you might also directly email affected customers or issue a press release for very critical issues.

- **Urgency and Guidance**: Clearly communicate how urgent the issue is and what users should do. If it's a critical vulnerability, the advisory should encourage immediate updating. If there are mitigations or workarounds, spell them out so users who can't patch immediately can still reduce risk. Always prefer actionable guidance – for example, "Upgrade to version 4.2.1 or later, which contains the fix" or "Apply the patch linked here" or "As a temporary workaround, disable the XYZ feature until you can update."
- **Clarity and Honesty**: Avoid downplaying the issue or burying the information. Be transparent about what could happen if the vulnerability is exploited but also avoid excessive fear. Use a factual and helpful tone to describe the nature of the vulnerability and its impact in general terms. For example, note, "A buffer overflow in image processing library could allow code execution and allow an attacker to potentially take control of the application". Clarify the affected versions and which versions contain the fix. Explain if only certain configurations are vulnerable so users can assess their own risk.
- **Support Channels**: Provide users with a way to get help or ask questions about the issue. This could be your normal support channel or a forum where they can seek guidance if the update process is unclear. Enterprise customers, for example, might have account managers to contact. Open source projects might use their issue tracker or mailing list for follow-up questions. Monitor these channels after disclosure to clarify any confusion.
- **Internal Stakeholders**: Inform internal groups as needed. Customer support teams should be briefed on the issue as soon as or slightly before it's public so they can handle inquiries. Sales or account reps might need advance notice if they are dealing with customers who require prompt notification of security issues. In some cases, such as when the vulnerability must be reported to regulators under certain laws, legal or compliance teams should know details of the vulnerability and its fix. Having a prepared statement or FAQ for internal teams helps ensure a consistent message.

Effective communication promotes a responsive, transparent, and empathetic relationship with both reporters and users. It helps prevent misunderstandings from reporters feeling ignored and users lacking information about security issues. Good communication practices ultimately lead to a smoother coordinated disclosure and a safer environment for everyone.

0.1.8 Coordinated Vulnerability Disclosure

Coordinated Vulnerability Disclosure (CVD) is the practice of working together with all involved parties to handle a vulnerability privately until a public disclosure is made at an appropriate time. Embracing CVD principles is highly recommended, as it strikes a balance between **security** (giving vendors

time to fix issues) and **transparency** (eventually informing the public). Here's how to approach coordinated disclosure:

- **Private Collaboration First**: When a vulnerability is reported, both the reporter and the vendor agree to address it confidentially before making details public. The vendor commits to investigate and fix the issue while the reporter agrees to withhold public disclosure for a reasonable period or until the fix is released. This cooperation ensures users are not put at undue risk by early disclosure of the bug.
- **Setting a Disclosure Timeline**: An essential aspect of CVD is agreeing on how long the vendor can take to produce a fix before the vulnerability information might be revealed. Many organizations follow an informal 90-day or 60-day guideline, which means that the researcher might publicly disclose the vulnerability anyway if a fix isn't ready after the elapsed time. However, timelines can be flexible if both parties communicate. As a vendor, try to estimate and propose a timeline to issue a patch based on initial risk assessments and complexity of the vulnerability. If you realize you need more time, inform the reporter as soon as possible and provide justification. Researchers often appreciate updates and can grant extensions if progress is evident. The key is to avoid leaving the reporter without guidance; lack of communication is a primary reason researchers go public out of frustration.
- **Involving a Coordinator**: Sometimes involving an impartial third party (coordinator) is useful or necessary. For example, if the reporter can't get a response from the vendor, they might reach out to a CERT or other coordinator to alert the vendor. Conversely, a vendor receiving a report about another vendor's product or a library with a flaw should pass that information to the right party and possibly involve a coordinator for multi-party issues. In multi-vendor situations, such as a vulnerability in an open source component that affects many downstream projects, coordination ensures that everyone gets critical information needed to fix their part and disclosure can happen jointly. Multi-party coordinators can establish a communication channel through mailing list or calls with all vendors and set ground rules for information sharing and the disclosure date. Organizations like CERT/CC or industry groups often help orchestrate this.
- **No-Fix Scenarios**: Ideally, every reported vulnerability is fixed before disclosure. However, a vendor may decide not to fix a reported issue after deeming it an acceptable risk or deciding they won't fix it due to architectural reasons. In coordinated disclosure, the vendor must communicate this decision to the reporter. They may still choose to disclose it publicly if they believe users should know. As a vendor, you should explain your reasoning in the advisory if this happens. Not fixing is generally discouraged for any significant security problem. Coordinated disclosure in such cases might break down, and the reporter could publish their findings. To maintain goodwill, these situations should be handled with transparency and respect for the researcher's perspective.
- **Handling Leaks or Early Disclosure**: Despite best efforts, sometimes vulnerability details leak

or a reporter publishes early. If vulnerability information becomes public before a fix is out, shift into incident response mode. Quickly assess the risk to users and consider releasing mitigation instructions or interim patches. Communicate openly with users about the situation – even if it's uncomfortable, it's better to acknowledge an early disclosure and provide guidance than to stay silent. Afterward, analyze what went wrong in the coordination and root causes for early disclosure or information leakage. Then, improve the process. These scenarios underscore why establishing trust and acting swiftly on reports is so important in CVD.

In essence, coordinated vulnerability disclosure is about **trust and timing**. Vendors must show reporters that they take issues seriously and will act, and reporters must give vendors the opportunity to resolve issues for the greater good of users. By coordinating, you ensure that when the world learns of a vulnerability, there is already a solution available – this greatly minimizes the potential harm from that vulnerability. CVD has become the industry standard approach and is a cornerstone of this guide's recommendations.

0.1.9 Publishing Advisories

Publishing a security advisory is the capstone of the vulnerability handling process. An advisory should explain what the issue is, who is affected, and how it can be fixed. These answers support a public record of the vulnerability and its fix, and it informs all users about what action to take. A well-crafted advisory considers the following in its timing and content.

- **Timing of Advisory Release**: Coordinate the advisory release with the availability of the fix. Ideally, the advisory should be published at the same time (or very shortly after) you release the patched software version. This way, users reading the advisory can immediately take action to secure their systems. Never significantly precede the fix with an advisory to avoid tipping off attackers before a fix is available. Conversely avoid delaying an advisory long after the fix, as users might not realize a security update is important without the context. In cases where a fix is rolled out silently (e.g. via auto-update), an advisory should still follow to document the issue.
- **Summary**: Provide aA brief description of the vulnerability and its impact. For example, "A buffer overflow in the image parsing library could allow an attacker to execute arbitrary code on versions 1.2 through 1.4 of the application."
- **Affected Products/Versions**: List the specific product names and versions that are vulnerable. Be as precise as possible (e.g. "Versions 1.0.0 to 1.4.2 are affected; version 1.4.3 and above contain the fix"). If older, unsupported versions are also affected, mention them as well but note if they are not patched.
- **Solution (Fixed Versions)**: State the version numbers or update that fixes the issue. If the fix is available as a patch, hotfix, or commit, provide links or instructions on how to obtain and apply it. In open source projects, this might be a reference to a specific commit or a new release tag.

- **Workarounds/Mitigations**: If applicable, describe any temporary measures users can take if they cannot immediately apply the fix. For example: "Until you can update, you can disable the image parsing feature by doing X," or "Configure the firewall to block port Y to mitigate the issue." Not all advisories have mitigations, but include them if they exist.
- **Acknowledgment**: Credit the reporter or others involved in discovering the vulnerability if they consent to be named. This typically goes near the end: e.g. "We thank Jane Doe for reporting this issue responsibly."
- **CVE or GCVE or any Tracking ID**: If a CVE or GCVE ID has been assigned to this vulnerability, include it. CVEs or GCVE are useful for indexing the issue in global databases and for users who track vulnerabilities via scanning tools. If you don't have a CVE or GCVE ID, you should include an internal tracking number instead of nothing at all – CVEs or GCVE are not strictly required but are standard for notable vulnerabilities.
- **Advisory Content** – Additional Information: In addition to the basics, including more details can be helpful:
  - **Severity/Rating**: Indicate how severe the issue is (Critical High/Medium/Low) to help users prioritize the update. If you use CVSS, provide the score and vector string.
  - **Discovery Timeline**: It is a best practice to include a timeline of events and describe when the issue was reported, when it was fixed, and when the advisory is published. This transparency can demonstrate your adherence to a responsible process and give credit to the timeline of the reporter's cooperation.
  - **Technical Details**: Depending on your audience, you might add a section with more technical explanation of the vulnerability. This can help advanced users or peer reviewers understand the nature of the flaw. It's also useful for historical record and for other developers to learn. If the vulnerability is complex, you might summarize how it was found or how it works. You may omit deep technical details if you fear it might aid exploit development. However, sharing details of a vulnerability after a patch is issued is considered good for transparency and education.
  - **References**: Link to any relevant references. For example, if this issue was discussed publicly or if it's related to a known vulnerability class, you might reference external documents or prior advisories.
  - **Update Instructions**: If updating is non-trivial and requires configuration changes or a series of steps, outline the steps or link to upgrade documentation.
  - **Format and Distribution**: Present the advisory in a format accessible to your users. Many organizations use plain text or Markdown for advisory documents and sometimes publish advisories in HTML on websites. Use a consistent template so users know where to find information. Consider distributing the advisory through multiple channels:

  - Post it on your official website (preferably in a dedicated Security Advisories section). A machine-parsable format should be available to facilitate the discovery and processing of vulnerabilities. This can be achieved with open source tool such as vulnerability-lookup.
  - Send it to a mailing list dedicated to announcements or security updates.
  - Publish it via your project's blog or news section.
  - Share on forums or community channels where users gather.
  - For open source projects, you might also use your source repository's advisory features (e.g. GitHub Security Advisory) which can send alerts to users of the project.

  Ensure that once published, the advisory remains available indefinitely for reference. Don't take down old advisories; they serve as a historical security record.

- **Post-Publication Monitoring**: After publishing, monitor the reaction. Be ready to answer questions from users or the media. If any detail in the advisory is found to be incorrect or unclear, issue a correction or update the advisory. Sometimes after release, additional research may discover that the vulnerability affected additional versions or that the fix had a bug. These discoveries and their subsequent fixes require an update to the advisory. Monitor for any signs of public exploitation now that the vulnerability is published. If something arises, you might need to alert users or provide additional mitigation advice.

Publishing advisories is a fundamental duty to your user base. A well-handled advisory not only helps users protect themselves but also builds your reputation for transparency. Stakeholders from enterprise customers to open source users will appreciate clear and timely advisories. Remember, acknowledging security issues publicly does not mean your software is less secure. On the contrary, it shows you take security seriously and deal with issues head-on, which ultimately **increases** user trust.

0.1.10 Best Practices

The following best practices reinforce the guidance above and provide a quick checklist for building a robust process:

- **Encourage Responsible Reporting**: Make it easy for people to report vulnerabilities to you. Publish clear instructions and promise a supportive response. Consider using a security.txt file on your website or repository so automated tools and researchers can find your contact info. Provide a way to encrypt sensitive reports (PGP key). Prominently assure researchers that you welcome reports and will not take legal action for good-faith efforts.
- **Respond and Remediate Promptly**: Time is of the essence in security. Strive to acknowledge reports quickly and fix critical issues as fast as possible without sacrificing quality. Even if a bug is complex, make interim plans and keep all parties informed. A slow response can result in

public disclosure without a fix, which puts users at risk. Treat researchers as partners in security. Be polite, thankful, and honest in all interactions.

- **Protect Sensitive Information**: Handle vulnerability details on a need-to-know basis until disclosure. Use encrypted channels for communications. Limit public discussion of the issue until the advisory is released. If working with multiple organizations, use Non-Disclosure Agreements (NDA's) or trusted channels as needed to prevent leaks. Always error on the side of caution with sensitive information. Use a clear classification standard, while distributing the information or interacting with third-parties, such as the Traffic Light Protocol (TLP).
- **Coordinate and Collaborate**: Follow coordinated disclosure principles. Work with reporters on timelines. If multiple vendors are involved, take initiative in reaching out to coordinate a joint response. Share information with platform owners or other stakeholders if the vulnerability could extend to them. Security is a team effort; collaboration can be the difference between a minor contained issue and a major public incident.
- **Learn and Improve**: After each vulnerability case review process steps that went well or could be improved. Implement improvements to your systems and practices based on these observations. Each vulnerability is also a lesson for developers: share the root cause and teach the team how to avoid that class of bug in the future. Over time, your software should get more secure and your handling process more efficient.
- **Stay Informed on Security Practices**: The security landscape evolves. Keep your vulnerability handling aligned with current best practices including industry standards and new guidelines from organizations like CERT/CC, FIRST.org, or NIST. Participate in security communities or forums to learn from others' disclosure experiences. For open source maintainers, many organizations and groups provide resources and forums to discuss disclosure challenges unique to open source – leveraging these can strengthen your own procedures.
- **Legal and Regulatory Compliance**: Be aware of any legal requirements regarding vulnerability disclosure that apply to you. Certain industries or regions may have laws about reporting breaches or vulnerabilities to regulators. While this guide focuses on the process itself, always ensure your legal counsel is in the loop if a vulnerability could trigger regulatory obligations. Also, having a clear policy with safe harbor language can protect both you and researchers by setting mutual expectations.
- **Recognize and Reward**: When possible, acknowledge the contributions of those who help improve your security. Publicly credit researchers (with their permission) in advisories or on a Hall of Fame page to encourage others to report. If resources allow, consider offering rewards.

By adhering to these best practices, organizations and open source projects can create a robust ecosystem for vulnerability handling and disclosure. The result is a win-win: security researchers have confidence that reporting issues will lead to constructive action, and users benefit from timely fixes and open communication about the security of the products they rely on.

By following the guidance in this document, software maintainers and organizations can better protect their users and improve their products' security over time. Vulnerability handling and disclosure is an ongoing commitment – one that, when done right, significantly reduces risk and builds trust in the software. It transforms security vulnerabilities from potential disasters into opportunities for learning and strengthening the resilience of our systems.

0.2 References

- OWASP - Vulnerability Disclosure - OWASP Cheat Sheet
- CIRCL - Coordinated Vulnerability Disclosure (CVD) Policy
- NIST - Recommendations for Federal Vulnerability Disclosure Guidelines
- Secure Coding - Taylor, Art; Brian Buege; Randy Layman (2006). Hacking Exposed J2EE & Java. McGraw-Hill Primis. p. 426.
- IETF Internet-Draft - Responsible Vulnerability Disclosure Process draft-christey-wysopal-vuln-disclosure-00

0.3 Acknowledgments

0.3.1 BCP-02 Coordinators

- Alexandre Dulaunoy, CIRCL
- Sascha Rommelfangent, CIRCL

0.3.2 Contributions

The GCVE initiative gratefully acknowledges the substantial contributions from the following individuals:

- Cédric Bonhomme
- Xavier Claude
- Matthew J. Harmon
- Quentin Jerome