Open Source Software Vulnerability Response: A Guide

Von Welch (von@vonwelch.com)

July 5th, 2010

A Overview

Software security is an important aspect of any mature software project and spans all aspects of software development from design through support. One important aspect to software security is responding to discovered vulnerabilities in the software after that software has been released and is in use by users and/or other software projects. A vulnerability can lead to exploits, means for malicious parties to take advantage of the vulnerability to corrupt systems on which it is deployed. Since exploits have strong negative consequences on software users, it is an imperative that software projects respond to vulnerabilities with remediations. Development of an exploit is particularly possible if the vulnerability becomes public, since information about the vulnerability simplifies development of an exploit. Ideally, a remediation is disseminated before the vulnerability becomes public knowledge (much less an exploit is developed).

The goal of this document is to capture current best practices for responding to security vulnerabilities for open source software projects, which have some unique challenges due to the normally open nature of these projects. The document was written based on the author's experience establishing and participating in a vulnerability response process for the Globus Project [3] and researching other vulnerability response processes found in the references.

B Goal and Objectives of Vulnerability Response

The ultimate goal of a vulnerability response process is to minimize the impact of vulnerabilities on users and other consumers of the software. The most disruptive consequence of a vulnerability is a malicious party developing an exploit that takes advantage of the vulnerability to compromise software in use by users. Once a vulnerability becomes publicly known, development of such an exploit is greatly eased since that knowledge gives an exploit developer specific information on the vulnerable software.

When the project privately learns of a vulnerability before it goes public, the project has a decision to make in terms of how long they keep knowledge of the vulnerability private. At one extreme the project could keep all vulnerabilities private until they become public through processes beyond the project's control. At

the other extreme, a project could make all vulnerabilities public as soon as they are aware of them, following that with development of a remediation.

Both of these extreme approaches have their downside. The first approach, private until public through other processes, means users will be vulnerable when vulnerabilities become public or are exploited since there is no advance dissemination of a remediation. The second approach, making vulnerabilities public immediately, means users will be vulnerable to possible exploit from that point in time until a remediation is disseminated.

This document describes a response process commonly in use, which is a balance between these two extremes. The goal of the process being to encourage initial private reporting of vulnerabilities to the project and then a subsequent public release of vulnerability information once a remediation has been developed and is ready for coincident dissemination.

The primary goal of this vulnerability response approach is to minimize the time between the vulnerability becoming publicly known and a remediation being available. Ideally, this response is such that the remediation is available when, or even before, the vulnerability becomes public, or, failing that, before an exploit is found to take advantage of the vulnerability. Encouraging private reporting of vulnerabilities to the project is important because doing so maximizes the chance the project is aware of vulnerabilities before they become public.

To achieve this goal, there are a number of objectives:

- Providing the user community a means of privately reporting vulnerabilities and encouraging the use of that mechanism in order to keep new vulnerabilities private until remediation is ready.
- Having a predictable process for responding to vulnerabilities so that the user community knows what to expect.
- Communicating as much as possible, as soon as possible, about a vulnerability without increasing the risk of exploit of the vulnerability.
- Making sure users can easily find information about past vulnerabilities that have remediation available.

C Challenges for Open Source Projects

Open source projects differ from closed source projects in that not only is the source code of the project public, but typically the project's development processes are public as well. These processes include bug reports, developer communication (e.g. email lists) and commits to the projects source code repository, along with any comments for the commits.

This presents a challenge for open source projects in vulnerability response in there is a tension between these open processes and a desire for privacy in vulnerability response. Ideally, as discussed in the prior section, a project will learn of a

vulnerability privately and develop a remediation before the vulnerability is known of outside the project. Maintaining that privacy is desirable in vulnerability response in order to minimize risk of an exploit being developed before a remediation for the vulnerability is developed, disseminated and applied.

There is also the sociological issue involved in that keeping aspects of development private is contrary to both internal culture of open source projects and the expectations of their user communities.

The Background section of the "Handling Mozilla Security Bugs" web page [4] has a very good discussion of this matter.

Additionally, open source software is likely to rely on or include other open source software packages. Vulnerabilities in these packages can cause vulnerabilities in the project's software distribution. As discussed subsequently, this brings about a need for open source projects to monitor the projects they rely on for vulnerabilities.

D The Vulnerability Reporter

A factor in successful vulnerability response and encouraging private reporting of vulnerabilities is understanding the motivations of someone who takes the effort to privately report a vulnerability to the project. Typically such a reporter will either be a security researcher or an expert user; in either case it's important to ensure that reporter feels their report is being handled appropriately and, when the vulnerability is disseminated, provide them with appropriate credit. A reporter who feels the project is not giving a vulnerability appropriate attention may become frustrated and make the vulnerability public as an effort to motivate the project (the Mozilla process [4] explicitly empowers the reporter to make the vulnerability public at any time). A reporter who feels they were not given their due credit may not use the private reporting process in the future.

E Vulnerability Response Process

The vulnerability response process goes through a number of steps: discovery of the vulnerability, assessment of the vulnerability, documenting and tracking of the vulnerability, early communication to users, developing a remediation for the vulnerability, and disseminating the remediation. Additionally, a project should take a number of preparatory steps to ensure this process goes smoothly.

Ideally information about the vulnerability remains private from the time the vulnerability is discovered until a remediation is developed, tested, disseminated and enough time has passed to allow for its application by users. By maintaining this privacy, the risk of an exploit being developed before the vulnerability is addressed is greatly reduced. Having a remediation available with time for users to apply it before an exploit is developed is the biggest factor in reducing impact on the user community.

There are however times, for reasons beyond the control of the project, when the vulnerability becomes public before a remediation is available (these scenarios are discussed in Section E.2). In these cases, particularly if the exploit already exists or would be easy to concoct, reducing impact may mean speed to create and disseminate a remediation is of the essence and privacy may only be of importance to obscure clues that might make an exploit easier to create or increase its impact.

E.1 Preparation

There are a number of steps an open source software project should take in setting up a vulnerability response process:

- Have a public, documented process. Having the process public eases some of the tensions of private activities in an open source project as the community will understand what it is doing. It also makes the process more predictable for the users and makes sure the internal project team is in sync.
- Establish and document a policy on what software is supported in terms of security vulnerability response. E.g. What prior versions before the current version will be supported? Are "development releases" supported? Is there "contributed code" that is not supported? Having a clear policy on this will allow the project to focus resources on the supported software and allow the user community to have appropriate expectations.
- Establish a way for vulnerabilities reports to come in privately. Document this and make it easy to find from the project's web site. This is typically an email list that must be monitored by humans and accepts email from any address (yes, spam will be an issue). Having a PGP key published for privacy is a good idea.
- Establish a well-known method for disseminating information about vulnerabilities. This method, typically a vulnerability announcement email list, should be low-volume and focus solely on vulnerability-related information so that information on it is treated seriously.
- Establish an archive for listing all past vulnerabilities and their remediations. This will allow users to find any information about previous vulnerabilities and make sure their deployments are up-to-date.
- Have a good process for testing patches quickly (e.g. a set of regression tests).
- Have a defined process for upgrading deployed software, either through an auto-update mechanism, a means to manually apply patches, or, worst case, how to replace a deployed version with a newer version wholesale.
- Know what other software the project relies on and how they handle vulnerabilities. If the project repackages or depends on another package for deployment, vulnerabilities in that package could create vulnerabilities in the deployments of the project's software. At a minimum the project should know how those projects respond to vulnerabilities and monitor their means of dissemination. Ideally the project should integrate itself into their vulnerability response process to learn about vulnerabilities as soon as possible.

E.2 Vulnerability Discovery

The first stage of dealing with a vulnerability is becoming aware of that vulnerability, ideally before it becomes public knowledge. If a vulnerability becomes known to the project at the same time it becomes public, the project is already behind and response is reduced to the matter of producing a remediation in a reasonable time (the more severe and/or easily exploited, the shorter the time). There are a number of scenarios by which vulnerability discovery can occur, listed from best-case to worst-case:

- Internal Discovery: A member of the project team discovers the vulnerability either through a security review or chance. This gives the project full control over public disclosure of the vulnerability.
- Private Report: A member of the public discovers the vulnerability and reports it privately to the project. As discussed in Section D, this means the reporter also has ability to disclose the vulnerability in addition to the project. This scenario includes a vulnerability discovered in software the project depends on that is reported privately to the project by that software team.
- Vulnerability in underlying software: A vulnerability is discovered in software on which the project software relies and the project learns of the vulnerability before it goes public because of their relationship with the underlying software project. In this case the underlying software project has control of the vulnerability disclosure and the open source project has a head start to address the vulnerability in their own software.
- Public vulnerability: A vulnerability is discovered only when it goes public. This could be as a public vulnerability disclosed in an underlying software package, accidental disclosure by a well-meaning reporter (e.g. using the wrong email list), or on purpose by a reporter for some reason.
- Zero-day exploit: A vulnerability is discovered only when a exploit for that vulnerability is being actively used in the wild or an exploit is disclosed along with the vulnerability. Having a proven exploit increased time pressure on the project to respond quickly.

It is a good idea at the time of discovery to determine who in the project has responsibility for responding to the vulnerability. Usually it should either be a member of the project security team or a developer who has expertise in the software most relevant to the vulnerability. In the author's experience, it is best if the responsible party is well versed with the response process.

E.3 Assess

The next stage in vulnerability response is assessing the reported vulnerability. The goal of the assessment process is to categorize the vulnerability based on its severity. This section gives an overview of an assessment process; for a more detailed description of a process, see the Common Vulnerability Scoring System [1].

Factors to be considered for assessment are:

- The level of difficulty of implementing an exploit for the vulnerability, assuming such an exploit does not already exist;
- The impact of a implemented exploit;
- Whether the exploit effects supported software; and
- Whether there is a temporary work-around users can use to protect themselves from an exploit until remediation is available.

Based on these factors, the vulnerability should be categorized into one of the following categories:

- *Not a vulnerability.* If the vulnerability is determined to not actually be a vulnerability, the project should respond back to reporter (if applicable) explaining their decision and the rationale behind it. This ends the vulnerability response process, though perhaps a standard bug report might result.
- *Vulnerability in unsupported software*. The vulnerability is legitimate, but is only affects unsupported software (e.g. an older version). In this case the project may choose not to fix the vulnerability. Standard practice would be to disseminate information regarding the vulnerability to inform any users still using the old software and encourage their migration to supported versions.
- *Low-risk/impact vulnerability*. If the creation an exploit seems very remote or the impact negligible, the project could decide to treat the exploit as a normal software bug, make it public and then handle it with normal processes. This should be done only after careful consideration, as if it turns out the assessment is incorrect, the information on the vulnerability will have been made public, easing any exploit creation.
- *Moderate-risk/impact vulnerability*. Some vulnerabilities have a reasonable degree of risk, but are deemed not sufficient to require an immediate dissemination of remediation. In this case the remediation dissemination can wait until a regularly scheduled software release or patch cycle.
- *High-risk/impact vulnerability*. The risk and/or impact is deemed to be so great that an remediation is to be disseminated as soon as possible. This entails an out-of-cycle patch or software update.
- *Extremely high-risk/impact vulnerability*. The risk and/or impact is deemed to be so great that waiting for remediation is deemed too great a risk. In this case users will be advised to discontinue use of the software until a remediation is available. Details of the vulnerability should still be limited to prevent exploit development unless they are so obvious as to provide greater benefits to the users instead of malicious parties.

During assessment, estimating the level of effort and personnel required to implement a remediation for the vulnerability is useful for subsequent steps of the response process.

E.4 Tracking the Vulnerability

As with any other software bug, it's important to track the vulnerability so that it can be clearly identified, discussions regarding it captured for reference, and its ultimate resolution documented. If the project's bug tracking system supports access control, such that bugs can be kept private to the team responding to the vulnerability, this is the standard way of tracking vulnerabilities, with the access control being lifted after dissemination. If the project's bug tracking system doesn't support access control it will be necessary to use an ad hoc tracking system that provides privacy (e.g. a limited distribution email list) and then transcribe it into the bug tracking system after dissemination.

In addition, it's common to identify vulnerabilities with a unique identifier such as a Common Vulnerabilities and Exposure (CVE)[7] identifier. Using something well known, such as CVE, has advantages in that it is well recognized. However, a project-specific identifier works fine as well (e.g. the Globus project uses the year followed by a sequential number to identify vulnerabilities [3]).

E.5 Early Communication

With high-impact vulnerabilities, it is good practice to give users an early warning that such a vulnerability has been discovered and that remediation is forthcoming. This is particularly true if application of the remediation is expected to be particularly disruptive, if an exploit is already known to exist, or if a temporary work-around to remediate the vulnerability is available.

The communication needs to walk a fine line between giving enough detail to be useful and providing too much detail as to provide clues to exploiting the vulnerability. In general, the communication should specify when the remediation is expected to be available and if there is anything that users can do in the interim to mitigate their risk.

E.6 Remediate the Vulnerability

The next step in the process is to create a remediation for the vulnerability, usually in the form of a software patch. A typical problem here for open source projects is that when the remediation is committed to the project's version control system, those source code changes and accompanying developer comments are public. This means a malicious party could monitor commits, seeking to get a head start on developing and using an exploit for any vulnerability before the project disseminates a remediation.

One approach to addressing this problem would be to not commit any remediation until after dissemination. However, for many projects having source code changes outside their main version control system is difficult since they won't be included in testing and automated build processes.

A common compromise, as specified in the Apache Software Foundation response process [1] for example, is to commit the remediation but omit any comment making it obvious the commit is related to a vulnerability. Once the remediation is disseminated, the commit can be augmented with information about the vulnerability it is addressing. This compromise might not be wise if the source code changes are obviously addressing a vulnerability (e.g. a buffer overflow fix might be obvious).

Testing of the remediation is important to make sure it doesn't break anything. Ideally the full suite of tests would be run to insure the patch doesn't break any functionality, which would put the user community in a tough situation of having to choose between functionality and security.

E.7 Disseminate

Once the remediation is ready, it should be placed on the project's public security patches web page and then the vulnerability can be announced along with a pointer to the patch and/or directions for the project's upgrade process. The announcement should give credit to the vulnerability reporter as discussed in Section D, discuss ramifications of the vulnerability, and the status of an exploit (is one known to exist or how easy would one be to create).

After dissemination of the vulnerability, internal aspects of the response can be made pubic including any bug tracking mechanisms for the vulnerability and commit messages in the version control system can be augmented to identify the relevant vulnerability they are addressing.

F Other Security Considerations

Discussions in this document assume there is not a malicious third party attempting to infiltrate the vulnerability response mechanism. If that were to be the case, the project may need to use other tools to provide privacy for the response process, e.g. encrypted email to discuss the vulnerability and its remediation.

G Developing Trusted Key User Communities

An enhancement to the approach laid out in this document is to develop trust relationships with key user communities such that they can be brought into the vulnerability response process. This approach is only viable for projects that have a small number of easily identified key user communities. For example, this was the case for the Globus Project when it established its process since it had a small number of key user communities defined by its funding from federal agencies. However, this is an unusual circumstance for an open source project and hence this enhancement is not in broad enough use to be considered a best practice, but is something that projects in appropriate circumstances may want to consider.

This approach works by identifying one or two members of each key user community and including them on the project's vulnerability response email lists and discussions. Finding members of those communities who have both a cybersecurity background and are very familiar with how those communities use the software seemed to work well. Those communities and members were expected to maintain privacy of the vulnerability response process (in the case of the Globus Project, there was no legal or formal agreement in place).

The benefits of including those communities in the vulnerability response process were:

- They provided additional input to the assessment process since they brought additional points of view on how the software is deployed and used, contributing to the assessment process and providing feedback on the suitability of the remediation;
- By being part of the vulnerability response process, these key user communities had an early warning of the vulnerability and the forthcoming remediation, allowing those communities to protect themselves until the remediation was disseminated and better prepare to apply the remediation, reducing the impact of the vulnerability on those communities; and
- These communities could provide testing of the remediation before it went public.

H Acknowledgements

The author graciously acknowledges contributions to this document from Jim Barlow, Randy Butler, Mike Dopheide, Ben Laurie, and Adam Slagell.

I Copyright and License

J Latest Version of This Paper

The latest version of this paper can be found at:

http://papers.vonwelch.com/vulnerability-response

K References

1. ASF Project Security for Committers. http://www.apache.org/security/committers.html

2. ASF Security Team. http://www.apache.org/security/
3. Globus Alliance Security Vulnerability Handling. http://dev.globus.org/wiki/SecurityCommittee/Security_Vulnerability_Handling
4. Handling Mozilla Security Bugs. Version 1.1. February 2003. http://www.mozilla.org/projects/security/security-bugs-policy.html
5. Rick Ring. Open Source, Red Hat, and Security. http://www.redhat.com/f/pdf/RH_OsRHandSecurity.pdf
6. WebKit Security Policy. http://webkit.org/security/
7. Common Vulnerabilities and Exposures (CVS). http://www.cve.mitre.org/
8. Common Vulnerability Scoring System (CVSS). http://www.first.org/cvss/