Chapter 34
FPVA Step 5, Dissemination of Results

Revision 1.1, October 2025.

Objectives

- Learn about the role of the assessment team once a vulnerability has been found.
- Learn about the role of the software development team once a vulnerability has been found.

34.1 You Have Found a Vulnerability, So Now What?!

Congratulations! If you have made it this far, you have successfully worked through Steps 1 to 4 and found one or more vulnerabilities in the system that you are assessing.

Or, perhaps, condolences! Your software development team has employed an assessment team and that team has delivered their first vulnerability reports to you.

So, now what? In this chapter, we will walk through the roles of the assessment and software development teams considering the point at which one or more vulnerabilities have been found. For many security analysts, what happens after the vulnerability is found is much less exciting than the hunt for the vulnerability. However, what you do at this stage can have a dramatic effect on your software security. For the software development team, this event can be a transformational moment.

> The process of dealing with a new vulnerability is a partnership between the assessment team and the software development team. Each partner has an important role and only by working together can this process be effective.

34.2 The Role of the Assessment Team

Once you have found a vulnerability and produced a demonstration that the vulnerability is exploitable, you will need to write a report that describes all the details of what you have found, perhaps including suggestions on how to remediate the vulnerability. In general, you will produce a separate report for each vulnerability that you found.

After you have produced a report, you will work with the software development team to help them through the process of remediating the vulnerability.

34.2.1 Producing Vulnerability Reports

There are several approaches to reporting a vulnerability that include the FPVA vulnerability format and the standard Common Vulnerabilities and Exposures (CVE) format[1]. These formats have enough in common that you can initially base your report on whatever is the most expedient for your project. We will start with the FPVA format as it is a straightforward process to produce a CVE from that information.

Other formats, such as the Open Source Vulnerability (OSV) schema[2], exist with many similarities to the FPVA and CVE formats. If you are familiar with any of the vulnerability report formats, the others will be easy to learn and understand.

34.2.2 The FPVA Report

The FVPA report format provides enough information for developers to understand the vulnerability that was found, reproduce it, and potentially provide guidance as to how to remediate it. The reports are designed so that the entire report is useful to the software development team, but with a few sections removed, the redacted report will still be informative to users of the software without revealing too much information to allow someone to easily recreate the attack.

The Public Parts of the Report

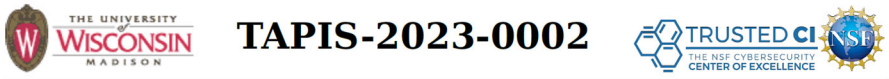
The report starts with a vulnerability ID number, which includes the system name, year, and report number. In the case of Figure *1*, we see "TAPIS-2023-0002".

The ID is followed by a short – typically one sentence – summary of the vulnerability. This summary should be crafted to communicate the most important aspects of the vulnerability. For example, in Figure *1* we see "Any local Tapis user", which provides information about from where the threat is coming, and "impersonate other users", which describes the malicious action that could occur.

Next is a block of summary characteristics that help the reader understand the scope and impact of the vulnerability. Here we explain the most important fields in this block.

---

[1] `https://www.cve.org/`
[2] `https://google.github.io/osv.dev/`



Summary:

Any local Tapis user can decode their respective user JSON Web Tokens (JWTs) and encode them with malicious information to impersonate other users and services either within the same tenant or other tenants.

Component	Vulnerable Versions	Platform	Availability	Fix Available
N/A	v3 1.3.8	all	Publicly Available	Yes
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified	Any Tapis User	None	Low	High
Fixed Date	**Credit**			
11/20/2023	Sai Krishna Chaparala Gia-Minh Nguyen Elisa Heymann			

Access Required: Any Tapis User

This vulnerability requires the attacker to have a valid user account with Tapis

Effort Required: Low

The attacker should belong to a particular tenant, be able to retrieve their respective Tapis user JWTs, and be able to access tools that allow the modification of the aforementioned JWTs

Impact/Consequences: High

By modifying their existing user JWTs, any Tapis user can impersonate other users within the same tenant, submit jobs as other users, and grant themselves **ADMIN** privileges over a Tapis tenant.

Figure 1: Sample Vulnerability Report for the Tapis System, Public Elements Shown

Component describes the particular part of the software system that contains the vulnerability.

Vulnerable Versions is important as it describes which versions of the software are subject to the vulnerability. It should never be omitted.

Platform describes on what operating system and hardware the software runs. It is not unusual for a vulnerability to only occur on certain operating systems, such as only in Windows and not on Linux. Or only on certain processors, such as only Pentium processors and not on ARM. Or even only on when compiled by a specific compiler, such as only with clang and not by gcc. There are also cases where the vulnerability might only occur with certain versions of a compiler or operating system. It is important to be as specific as possible here.

Availability describes whether the vulnerability is known to the public.

Fix Available describes if a fix for the vulnerability is available and, if so, the **Fixed Date** describes when the fix became available.

Next are a fields that describe the effort required by an attacker and consequences of such an attack. These fields are then expanded below with more details.

Access Required describes the type of access required to exploit this vulnerability. For example, the report in Figure *1*says that this vulnerability can be caused by "Any Tapis User", i.e., any valid user of the Tapis system who is logged in. It might also be for a specific kind of user, such as "Any Tapis user with EXECUTE permissions". It might be anyone on the Internet, someone with an account on a host inside the organization, or someone with an account on the specific host.

Effort Required summarizes how difficult it is for an attacker to exploit the vulnerability. At exploit that requires expert skills would be "High" and an exploit that requires no special security skills would be "Low". Not that a complex exploit might be described as "Low" effort if a tool or script that automated the attack was publicly available.

Impact/Consequences summarizes the effect of the vulnerability. If the vulnerability would cause some inconvenience but no serious loss of confidentiality, integrity, or availability then it is likely "Low" impact. A vulnerability that could expose sensitive information or disrupt operations of the system is "High" impact.

The (Initially) Private Part of the Report

The software development will want to understand exactly how the vulnerability was exploited and how to fix it. So, the vulnerability report will continue with additional fields that are initially only available to the development team.

Once the vulnerability has been fixed, a release with the fix has been made available, and sufficient time has past for users to update their installation, we recommend that the full report is released publicly.

The sections in the private part of the report are:

Full Details are step-by-step instructions on how to exploit the vulnerability. These instructions should be sufficiently detailed so that an experienced programmer or analyst could reproduce the exploit. They should include the details of commands used, data formats, and expected outputs including screen images. This section of the report is typically the longest and most detailed.

Cause is a description of the root cause of the vulnerability. It is a description of the programming or design practice that allowed the vulnerability to happen.

Proposed Fix is the initial suggestion from the assessment team as to how to fix (remediate) the vulnerability. After the report is released, the assessment and development teams will work together to identify the most

effective way to fix the vulnerability. The goal is to find an effective fix that will provide long-term security and not expose any new issues.

Actual Fix is the method ultimately used by the development team to fix the vulnerability. It might be exactly what was proposed by the assessment team or some variation developed based on development team's broader understanding of the software, considering their processes and standards.

34.2.3 The CVE Ecosystem

The idea of a standard vulnerability report was proposed in January 1999 by Mann and Christy from MITRE Corporation[3]. The Common Vulnerabilities and Exposures (CVE) program was officially launched in December of that year, with 29 organization immediately participating. In 2002, the U.S. National Institutes of Standards and Technology (NIST) recommended that software development organizations report their vulnerabilities in the CVE format.

34.2.3.1 Vulnerability Databases

The first widely used database of vulnerabilities was created by NIST in 1999, initially called the Internet Category of Attack Toolkit (ICAT)[4]. It was rebranded in 2005 at the National Vulnerability Database (NVD). Note that the NVD serves more than just the U.S.; it is a global resource. As of September 2025, the NVD contained over 300,000 vulnerability reports.

Other vulnerability databases that are based on the CVE format include the U.S. Department of Homeland Security's Cybersecurity and Infrastructure Security Agency (CISA)'s Known Exploited Vulnerabilities (KEV) catalog[5] and the GitHub Advisory Database[6].

34.2.3.2 CVE Report

The CVE report contains fields with similar purpose to the ones used in the FPVA format. Note that several of the fields, such as those that describe the underlying weakness for this vulnerability (CWE) or the severity of the vulnerability (CVSS) are standards in themselves. These related standards were explained in Chapter 17, where we describe dependency analysis tools.

Below, we describe some of the key fields in a CVE.

CVE Identifier is the standard name for the vulnerability report. An example is CVE-2014-0160, which reported the "Heartbleed" vulnerability

---

[3] `https://www.cve.org/about/history`
[4] `https://nvd.nist.gov/general/brief-history`
[5] `https://www.cisa.gov/known-exploited-vulnerabilities-catalog`
[6] `https://github.com/advisories`

in the OpenSSL secure socket communication library. “2014” is the year of the report and “0160” is the unique sequence number in that year.

Severity Score – common Vulnerability Scoring Systems (CVSS) – describes the impact of this vulnerability. More details on the CVSS are presented in Chapter 17.

Weakness that Caused the Vulnerability – Common Weakness Enumeration (CWE) – describes the coding error that allowed the vulnerability to happen. Again, more details are presented in Chapter 17.

Versions describes which versions of the software known or be affected and which have unknown status.

Description is a textual description of how the vulnerability works.

References is a list of external resources that explain details of the vulnerability.

34.3 The Role of the Software Development Team

Of course, the process of undergoing a security assessment of any kind can make you nervous. When the first vulnerabilities start rolling in, the first instinct of the development team is to be quite upset. As the wonderful writer Doug Adams wrote on cover of his *Hitchhiker’s Guide to the Galaxy* books: “Don’t Panic!” You are an important partner in this process, and your goal is to be an active participant throughout it.

The software development team’s response to these first vulnerability reports can be similar to the five stages of grief

1. Denial: You just cannot believe that something so bad could be happening in your code. This discovery can be quite disorienting the first time that it happens.
2. Anger: This anger can be directed at either your management or programmers or the assessment team (or all of these). “We wouldn’t have started this process if we knew what a mess it would uncover.”
3. Bargaining: Something like: “I know that we said that we’d publicize our findings, but maybe we really shouldn’t?”
4. Depression: We start to hear comments like “We’re screwed. No one will trust our software and we’ll lose our funding or investors.”
5. Acceptance: The team starts to accept what has happened and moves forward. Then the good stuff starts to happen.

If this is the development team’s first experience with such an activity, they might need a bit of care. However, in the end, your software will be more resilient and your team stronger and more security aware, so it’s worth it.

Essential to dealing with vulnerabilities found at your behest, and those found by outsiders, is to have a plan. You will need to plan on how to schedule the fixes, who will do them and to what versions of your code the fixes will be applied.

34.3.1 Helping to Enable the Assessment Process

You can make this process go smoother and make the team that is assessing your code more effective. The key elements to enabling this process are good documentation, access to your code, and a designated contact person on your project.

The assessment team is going to try to understand your software in detail. Having good documentation can help to accelerate that process. Accurate and up to date user and installation documentation is the minimum you should be able provide. Hopefully your project already has such documentation available. In addition, documents that describe the design and structure of your software can be hugely valuable to the assessment team (though this type of documentation is, unfortunately, relatively uncommon).

The major reference that the assessment team will use is the source code. That code needs to be complete and up to date.

As part of understanding your system, the assessment team will need build, install, and run your software. For many projects, especially ones that run their software only on their own computers, this process is often fragile, with many local system dependencies. It is often a confusing, difficult, and lengthy process to get the software running elsewhere.

These challenges have led us to require that the software team provide us access to their software in one of two ways:

1. Provide a buildable and working system, ready to go in a container or virtual machine. Using a container or virtual machine allows your team to provide a complete build and run environment, including the source code, compilers, libraries, and other tools.
2. Provide access to your development system, similar to the access that your programmers have. Such an approach can be the easiest to set up, though there may be policy or security concerns about allowing outsiders to have access to your system. Such an approach can be made more complex by restrictions on remote login to your development systems or issuing of login credentials to non-staff.

And they need access to you, your software staff. Your goal is to help the assessment team move forward as efficiently as possible.

Key to this process is to assign a contact person within your organization who will respond promptly to questions and reports, and direct these to the right person in your organization.

You should expect that the assessment team will work independently, asking questions as needed but providing little feedback until the process is completed.

34.3.2 Receiving the Vulnerability Report and Responding to It

For many software teams, the most shocking moment of this process is when you receive your first vulnerability report. The designated a contact person receives the report and communicates it within the software development team. At that point, you will want to assign one or more people to understand what the report says, get any clarifications if necessary, decide on a remediation strategy, and communicate that information back to the assessment team. If the assessment has made any incorrect assumptions about how the system works or is deployed, this will be the time to communicate that information so that they can make adjustments to their processes.

A key part of evaluating a vulnerability is to prioritize your response to it. Certainly, if this vulnerability exposes your entire system to remote administrator access, responding to it would be quite urgent. However, the economics of software development will come into play here. For many vulnerabilities, you will have to rank their importance alongside other vulnerability reports, functionality reports, and new developments.

Essentially, you will be integrating these reports into your risk assessment process. Such risk assessment may involve your management, users, and other stakeholders.

34.3.3 Develop a Security Release Process

Once you have fixed the vulnerability, you will have to inject that fix into your release stream. Depending on the urgency of the vulnerability, you may decide to immediately issue a new release or wait until the next scheduled release.

Ideally, you want to have the security releases separate from functionality releases.

A **security release** is one that fixes only errors in a current version of the system, adding no new functionality.

If you are currently supporting multiple release versions of your software, then you would develop a security release for each version.

Early in our work on FPVA, we had an experience with an assessment that demonstrated the benefit for separate security releases.

> We were assessing a job scheduling system for a distributed environment and found a serious vulnerability that could allow an ordinary user to easily get administrator privileges. This was serious enough of a vulnerability that the software development team felt it necessary to fix it and release the fixed version as soon as possible. The quickest way for them to do that was add the fix to a pending functionality release, which was coming out at the end of the week.
>
> Once the release was issued, along with a notification that it contained an urgent security fix, a major science facility went into a panic. This facility was in the middle of a major multi-week computation. When the release was issued, they had to decide between two difficult choices:
>
> 1. Update to the new release, which included major new functionality. As a result, their current computation might not be compatible with the new version, so they would have to start it over again, losing a couple of weeks of progress.
> 2. Do not update to the new release, knowing now that their system was at serious risk.

If the software development team had produced a security release of their system, then the science facility could have updated immediately, adding little risk to their computational progress while knowing that they had prevent attacks based on the privilege escalation vulnerability.

34.3.4 Develop a Vulnerability Notification Strategy

Once you have produced a release that fixes the vulnerability, your next decision if what to tell your user community about it and when.

We always advocate openness in such communications with your users. An organization that is silent about vulnerabilities is either not looking for them, which is bad as it indicates lack of a proper security program. Or not telling their users about what they have found, which is bad because such practices erode trust with your users.

When you produce a release with one or more security fixes, you should include a basic notification of the type of problem that you fixed. The FPVA vulnerability report format (Section 34.2.2) is designed to release the top portion of the report as a summary of what was found. Other report formats, such as the CVE, usually do not include enough details to allow an attacker to develop an exploit. If you want to publicize your vulnerability reports via CVEs, considering registering your organization as a CVE Number

Authority (CNA)[7] so that you can directly submit new CVEs to the NVD, receiving wider visibility of your reports.

Later, when sufficient time has passed for users to have updated their software, you can release the full details of the vulnerability and related exploit. We usually recommend six months as the waiting time for the full details to be released.

If a software development team has had little experience with vulnerability reporting they are often nervous about admitting that they had vulnerabilities in their code. Our experience over many years, in both the research and commercial worlds, is that users, investors, and funding agencies universally appreciate such openness as it indicates an active and effective security program. We have seen reports of such a vulnerability assessment process used by research groups to bolster their grant proposals and by companies to attract investment funding.

What about Open Source?

When your software project is based on open source, new releases that contain security fixes will also contain the details of how the code was fixed. Such information could give a potential attacker enough information to craft an attack on installations that have not yet installed the new release.

While open source software offers many benefits related to security, this aspect of open source clearly requires some consideration. Though note that attackers are quite skilled at reverse engineering of the binary code in a closed-source system, rapidly finding the parts of the code that changed and what was the functionality of the change. So, using open source is not a good vs. bad choice; it is merely one of degree of difficulty in crafting an attack.

If you have a well identified group of critical user sites, you might consider sending out a security notice to your contact at each site shortly before the release is made available so that they are prepared to update more quickly.

34.3.5 A Change of Culture within the Development Team:

When security becomes a first class task within your software development team, and when reports start arriving, security *awareness* is significantly increased.

This process will affect the way that your developers look at code and the way that they write code. There is a productive feedback cycle that the more your team is involved with handling vulnerabilities, the more they bring that knowledge to their coding activities. Their code will get better from a security point of view.

---

[7] `https://cveform.mitre.org/`

A major landmark in a software development team is when your developers start reporting vulnerabilities that they have found themselves. When this happens, you are seeing a tangible benefit of the vulnerability assessment process.

34.4 Summary

In this chapter, we have:

- Learn about the role of the assessment team once a vulnerability has been found.
- Learn about the role of the software development team once a vulnerability has been found.

34.4.1 Exercises

1. Compare the FPVA, CVE, and OSV vulnerability reporting formats.
    a. What common features do you find in all three of these formats?
    b. What are the advantages of each format?
    c. Which format (or formats) might be appropriate for a software development project in which you are participating?
2. What are the advantages and disadvantages of having security releases separate from functionality releases?
3. Why does the security assessment team not provide any updates to software development team until they have a completed vulnerability report?
4. What should the software development team provide to the assessment team to build and run the software that is being assessed?
5. Consider the vulnerability reporting process.
    a. What are the challenges that an open source development project faces in vulnerability reporting that are not faced by a closed source project?
    b. Are their similar issues faced by a closed source project?