Chapter 29
Introduction to First Principles Vulnerability Analysis

Revision 4.0, March 2025.

Objectives

- Learn to think like an analyst.
- Review the secure software development lifecycle
- Understand the need for in-depth vulnerability assessment.
- Understand when to do such an assessment and by whom.
- Get an overview of the five steps of the First Principles Vulnerability Assessment methodology.

29.1 Motivation

We start by mentioning a well-known fact: All software has vulnerabilities. If you are not hearing about vulnerabilities in a specific piece of software, it means that the developers are either not telling or, even worse, not looking. We also know that modern software is complex and large, which means that vulnerabilities are more than likely to be present and ready to be exploited. We are concerned about both vulnerabilities that can be exploited by authorized users and by outsiders.

> Our primary goal is to find the vulnerabilities before the attacker finds and exploits them.

In addition to the fact that software has vulnerabilities, there is an unpleasant asymmetry, as the attacker chooses when, where, and how to break into a system, while we, the defenders, need to protect against all possible attacks. That means that we should be defending against currently known attacks, as well as new attacks yet to be discovered.

We should be thinking about security at every step in the Software Development Life Cycle, from the first moment you conceive of the software, up until the moment that you release it (and perhaps continue to evaluate the software even after release). As we previously discussed, secure software starts with an understanding of secure design principles (Chapter 4), continues the modeling of threats that a system might encounter (Chapters 5 through 8), and is supported by a strong understanding of the programming errors that can lead to vulnerabilities and the techniques needed to avoid them (Section 3). The programming process can be augmented by the use of testing (Section 7) and static and dynamic tools (Section 6).

The frequent use of automated assessment tools can help to find weaknesses in our code. We should do that from the moment we write the first line of code, and continue to do it through the implementation, for example before every commit. Using tools is essential, but we still should be aware of their limitations. The tools will help us find some basic errors, but they can miss complex ones, and some of them may be important. That is the *false negatives* problem. At the same time, tools can produce voluminous reports, and several of the issues they report may not be issues at all. That is the *false positives* problem.

To cap off this process, and push our software security to the next level, we can perform an in-depth vulnerability assessment of the software. The goal of such an assessment is to look beyond a list of previously known vulnerabilities that might be found by a variety of tools. Otherwise, we would be fighting the wars of the past, and not the ones of the future; approaches based on known vulnerabilities will not find new types and variations of attacks. We want to identify the most important components of the system (the *high value assets*) and then find the ways that those components are vulnerable. We need to focus on the high value assets as we want to find the most critical vulnerabilities in the system. Otherwise, we risk consuming the resources allocated to the in-depth vulnerability assessment effort before finding serious vulnerabilities.

You can think of in depth-assessment that is done when the software is complete as the equivalent of what a designer does with Threat Modeling (described in Chapters 5 and 6) at design time. Looking at the Microsoft Security Development Lifecycle (Figure 1), we can see Threat Modeling early in the Design stage and in-depth software vulnerability assessment happening in the later (post implementation) Verification stage. If such an assessment is done by an outside independent organization, it might even happen after the software is released.

An in-depth vulnerability of assessment is:

Analyst centric: It takes a person to perform it. While there are tools to help with code assessment, they are currently no substitute for a careful assessment by a human analyst.

White box: We assume that we have the source code, documentation, and access to the development team to answer questions.

Time consuming: While we can support this effort with a variety of tools, there will still be an extensive human component to the work. For a real system, such an assessment can take up to four to eight months.

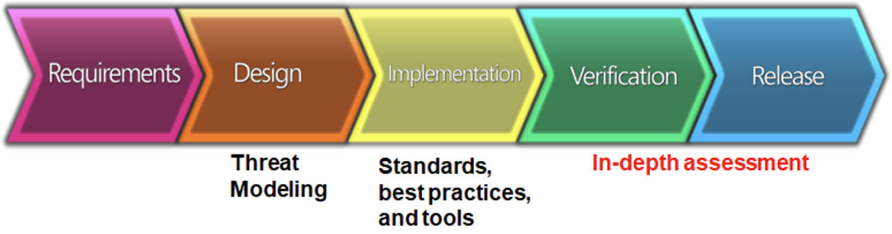

Figure 1: The Microsoft Security Development Lifecycle

29.2 In-Depth Vulnerability Assessment

In this chapter, we introduce you to a technique for performing in-depth vulnerability assessment of a software system. This kind of technique can help to increase your confidence in the security of your code. And learning such a technique will help you to start thinking like a security analyst. Note that we will describe just one approach to in-depth vulnerability assessment. However, if you master this one, it will be easy for you to learn others.

An in-depth assessment is our last significant chance to find vulnerabilities, and ideally it should be performed at testing time. That is after the implementation is completed but before the system is released, and therefore accessible to users and attackers. Of course, you also need to have an organized response to the finding of a vulnerability. As soon as a vulnerability is found the development team needs to fix it.

> The reality is that most of the time there are little or no resources, or even worse, no realization of the need to perform a vulnerability assessment, so it never happens.

Organizations often have the pressure to release the software by a deadline. To meet that deadline, frequent testing is sacrificed. So, you can imagine what happens to security-related activities.

The next issue to address is *who* will perform the in-depth vulnerability assessment?

> The primary rule is that we should have an independent assessment.

Software engineers have long known that testing groups must be independent of development groups. You cannot assess your own code, as you are biased. The design and implementation team is usually so familiar with the code that they find it difficult to think about the code in different ways. This is one area where you cannot take shortcuts. Even if the development team is outstanding at testing, they cannot do an effective assessment of their own code.

This independent assessment is performed by a security analyst, someone whose job is to specifically evaluate the security of the code, that is an external person or group of persons not involved in the development of your software. The analyst(s) may come from within your organization or from an organization that specializes in this task.

Assessment and remediation of vulnerabilities should be integrated into your software development process, as you must be prepared to respond to the vulnerabilities you find.

29.3 First Principles Vulnerability Assessment (FPVA)

The technique that we describe is called *First Principles Vulnerability Assessment*[1], or FPVA for short. This module is divided into six chapters, with this chapter introducing FPVA. Each of the remaining five parts addresses a different step of the FPVA process. The FPVA process proceeds in five distinct steps:

Step 1: Architectural Analysis

Step 2: Resource Identification

Step 3: Trust & Privilege Analysis

Step 4: Component Evaluation

Step 5: Dissemination of Results

The first three steps are aimed at getting an understanding of the big picture of the system to focus our search for vulnerabilities. Once we have that big picture, we understand the structure of the system and the areas that are most security sensitive. These three steps provide information necessary to focus the analyst's attention on *high value assets,* the parts of the system that would provide the biggest benefit to the attacker. This picture provides guidance to the analyst as to where to look in the code for problems.

[1] James A. Kupsch, Barton P. Miller, Eduardo César, and Elisa Heymann, "First Principles Vulnerability Assessment", *2010 ACM Cloud Computing Security Workshop (CCSW)*, Chicago, IL, October 2010.
`https://www.cs.wisc.edu/mist/papers/ccsw12sp-kupsch.pdf`

For example, the process (server) that authenticates users would be a high value component and a configuration file (such as a start-up file) that specifies which other programs to run would be a high value resource.

In a real system, it is essentially impossible to inspect every line of code for security problems. The first three steps of FPVA are essential to focusing the analyst's attention on the parts of the system that might create the biggest benefit for the attacker.

Based on the information in the first three steps, in Step 4, we then do a focused deep dive into the code. Step 5 is about reporting on the vulnerabilities found.

FPVA is not based on known vulnerabilities, but that does not mean that it will neglect already existing vulnerabilities. It means that FPVA can go beyond those and find new vulnerabilities, or vulnerabilities specific to the system with which you are dealing. FPVA structures the assessment activity so that the analyst focuses the search on the most serious vulnerabilities.

Remember from the introductory module that a vulnerability is a flaw in your system that can be exploited. In FPVA, if the analyst is not able to build an exploit for a potential vulnerability, then they do not report that problem.

After identifying a vulnerability, the analyst suggests an effective remediation and interacts with the development team as they fix the problem. It is also a good practice to save some of your security budget to reassess the code with the mitigated vulnerability, to make sure that not only was the vulnerability fixed, but also to check that no new vulnerabilities were introduced.

In each of the following chapters, we will describe the details of each step of FPVA.

29.4 Summary

FPVA is a methodology that allows an analyst to find vulnerabilities affecting the high value assets in a software system. It is a human-centric methodology consisting of five steps. This chapter introduced FPVA as motivation for the following chapters that will provide details of each step.

29.5 Exercises

1. Why should a vulnerability assessment be done by an outside team, a group other than the design and implementation team?

2. Why is a methodology such as FPVA needed when resources such as static analysis tools and dependency tools are available?
3. For a software project with which you have been involved, go through each step of the Software Development Life Cycle (requirement, design, implementation, verification, and release) and describe what your software team did at each step.