Starting a Security Program on a Shoestring

Brian G. Myers

brian@safetylight.dev

Abstract

You are part of a small development team building a web application, and now someone tells you your product must be secure. Maybe the requirement comes from an auditor, or maybe from a prospective client. No one on your team was hired for security expertise. What should you do? How can you make meaningful progress with minimal knowledge? Where should you start?

You could hire experts, get training, identify risks and threats, prioritize your findings, design solutions, and implement them. That is the right way to go, but it makes for a slow start with a lot of heavy lifting up front. For people who do not have that option I propose a much easier way to get quick results and long-term improvements with minimal initial investment: just incorporate a vulnerability scanner—I will suggest a free one—into your software development process.

This paper explains how setting up a simple vulnerability management program will give immediate results, make your application more secure, improve your secure development lifecycle, help your team develop security expertise immediately relevant to their project, and meet some likely compliance requirements.

Along the way I will provide tips for choosing a vulnerability scanner. I'll show how to understand what the scanner finds, explain simple steps to turn use of a scanner into a vulnerability management program, and point out resources to help with questions that may arise.

Biography

Brian Myers (Ph.D, CISSP) has been working in security for five years and in software development for—well, long enough for his resume to include time at Borland and Netscape. He has written three books on Windows programming and taught introductory computer courses at the University of California, Berkeley. In his current job as Director of Information Security at WebMD Health Services Brian guides the work of multiple teams building a web platform in a HIPAA-regulated environment. He also serves on the CS/IS Industry Advisory Board at Western Oregon University.

1 Introduction

This paper suggests an approach to starting a security program in a company that does not yet have one. It focuses specifically on how QA can take a leading role in initiating a key part of any security program: vulnerability management. It assumes you belong to a smallish company with no security program, no budget for security, little experience in security, and minimal access to experts or formal training.

The impulse to improve your company's security posture might arise for a number of reasons. Maybe a potential customer is asking for details about how you protect their data. Maybe a security audit is headed your way. Or maybe you want to learn about security because it's fun and security experience is in demand.

Setting up a complete security program is a considerable effort. A complete program would include, for example, physical facility security, hiring practices, vendor management, training, risk management, and other areas for which you'd normally hire an information security professional. But even without all that expertise you can still make a reasonable start at driving security awareness and knowledge into your teams by starting small. The other parts can come later when the business discovers it needs them.

The problem of course is where to start without having to study up on large and unfamiliar areas. One of the main advantages of the approach I suggest is that it points you very quickly to particular bits of knowledge that are immediately relevant to your particular project. You do have to be willing to learn, but you can take it one small piece at a time.

Furthermore, not much technical knowledge is needed to follow the process this paper describes. All it takes is the ability to read HTML and HTTP traffic. For the purposes of this paper, I will assume my readers can make some sense out of this:

```
POST http://10.133.1.4/mutillidae/index.php?page=login.php HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: https://10.133.1.4/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 88
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cookie: uid=24; showhints=1; PHPSESSID=rpiq3oo77e7rssp2gbjv6gfus2;
username=brian
Host: 10.133.1.4

username=brian&password=qwerty%27+AND+%271%27%3D%271%27+--+&login-php-
submit-button=Login
```

In particular, it will help if you can see that in the last line someone is posting to the server a very odd value for a password. (The odd value is underlined.) That is a scanner-generated value probing for a possible vulnerability, and understanding that will help you reproduce the problem. That is all the technical expertise needed to get started with a web vulnerability scanner.

2 What are Vulnerabilities?

First, what is a vulnerability? What are we trying to find?

A vulnerability is weakness in a system that can be exploited by a malicious actor to perform unauthorized actions. Scanners look for actions that might leak data, destroy data, or make the data

inaccessible. For a standard introduction to common types of vulnerabilities, consult the OWASP Top Ten Project (see Resources section.) OWASP is an international non-profit whose purpose is to help companies create secure software, and one of their projects is a periodically updated list of the most common vulnerabilities that software development teams are likely to encounter.

Common vulnerability types include SQL injection, cross-site scripting, cross-site request forgery, and information disclosure. If you don't know what those are yet that's fine. Using a scanner is a good way to start learning about them.

3 Tools for Finding Vulnerabilities

A Google search for *vulnerability scanners* turns up scores of them. *Scanners* is a very broad category, however. Knowing what kind of vulnerabilities you want to find will narrow down the choices considerably.

3.1 Types of scanners

Scanners fall into at least four categories that differ in where they look and what they find.

Scanner Type	Scanner Target Area	Findings	Popular Examples
Network	Server and network infrastructure	Infrastructure configuration and patching problems	OpenVAS Qualys Nessus
Static Application Security Testing (SAST)	Application source code	Suspect program logic (missing input validation; buffer overflow risks…)	Coverity AttackFlow SonarQube
Composition Analysis	Third-party libraries	Third-party libraries with known vulnerabilities	OWASP Dependency Check WhiteSource BlackDuck Sonatype
Dynamic Application Security Testing (DAST)	Running application	Weaknesses in web pages (XSS, injection…)	WhiteHat AppSpider Netsparker

Table 1 - Types of vulnerability scanners

All four types matter, and software development companies generally need all of them. But you have to start somewhere, and for the scenario I have imagined—a small company with a web application—the best starting point is likely dynamic testing (DAST.) A DAST scanner doesn't look at source code or at build artifacts. It visits the web pages in a running application and interacts with each one to see if it can find vulnerabilities a malicious actor might exploit.

3.2 What Exactly Does a DAST Scanner Do?

Operating a DAST scanner generally involves four basic steps.

1. *Configure the target.*
   First you tell the scanner what website to scan. This usually involves providing a URL and instructions for logging in to the site. Some scanners allow configuring secondary settings such

as what vulnerabilities to look for in the scan, how much bandwidth the scanner is allowed to use, and whether to include AJAX calls in the scan.

2. *Attack the site.*
   Next the scanner starts sending HTTP requests to the site in order to find problems. This involves three different operations. Some scanners execute all three steps at once while others ask you to initiate each as a distinct operation.

   a. Find the pages (crawl the site)
      The scanner starts with the target URL, look for links, and follows them recursively to discover as many pages as it can in the web site.

   b. Passive scan (discover pages and read them)
      The scanner examines the contents of each GET request and flags certain potential problems such as missing HTTP headers, insecure cookies, or exposed system information.

   c. Active scan (feed pages unexpected input)
      The scanner passes malicious values to the web server on query strings and forms. It sends the server values that are designed to turn up problems. It may fill in forms with extremely long input values, unexpected characters, semi-random “fuzzed” values, SQL commands, or any other hostile input that might produce unwanted behavior.

3. *Create a report.*
   When the scan is done the scanner makes the results available in reports. Most scanners produce executive summary reports for managers as well as details report for development teams. The scanner reports are your source of information for reporting defects.

The first two steps of a scanner attack, crawling and passive scanning, are read-only interactions and do not change anything in the site. An active attack, however, is likely to add, modify, and delete database records. An aggressive scan could also conceivably make the server so busy that other concurrent users suffer. In general, do not run a vulnerability scanner against a production environment.

3.3 How to Choose a Scanner

NIST and InfoSec have both issued papers on selecting scanners (see the *Resources* section below.) Table 2 presents a consolidated list of desirable features compiled from multiple sources.

Scanning Functions	
Transmission Protocol Support	HTTP/S, HTTP compression, XML…
Breadth	Types of vulnerabilities it can detect
Accuracy	False positive rate
Authentication Mechanisms	Form-based, NTLM, Kerberos, SSO…
Session Management	Detect login, logout, session expiration…
Configurable Aggressiveness	Rate throttling, vulnerability type selection…
CI/CD Integration	Automation support
Reporting	
Executive Summary	Overview of results with breakdowns and subtotals
Information Per Finding	Name and description Location, inputs, context Severity CVE or CWE Remediation guidance
Changes and Trends	What has changed since the last run Are we getting better over time
Machine-Readable Output	XML (for example)
Support	
Documentation	Manual, reference, guide, wiki…
Training	Tutorials, videos, consulting…
Community	Active forums
Vendor Track Record	Reliability, expertise, release history
Commercial Considerations	
Price	Licensing + professional services
Location	SaaS or On Premise
Consulting	Training, integration…

Table 2 - Criteria for evaluating vulnerability scanners

Given the scenario we are imagining—a team with little expertise and no budget—I limited my selection of candidates to scanners that:

- Are free to use
- Have been recently maintained
- Detect a good range of vulnerabilities
- Provide informative reports of findings
- Can be integrated with a build process

In practice the first two criteria eliminated most of the contenders. A large number of the commonly mentioned free scanners are not actively maintained. Of those that are, some are special-purpose and so fail the third criteria (range of vulnerabilities.)

Product	Free	Last Release	Active	General Purpose
Arachni	Yes	2017	No	Yes
Brakeman	Yes	2019	Yes	No
Grabber	Yes	2013	No	Yes
Grendel-Scan	Yes	2012	No	Yes
IronWasp	Yes	2015	No	Yes
OWASP ZAP	Yes	2019	Yes	Yes
RatProxy	Yes	2009	No	Yes
Scan My Server	Freemium	2019	Yes	Yes
Skipfish	Yes	2012	No	Yes
SQLMap	Yes	2019	Yes	No
Vega	Yes	2016	No	Yes
W3af	Yes	2019	Yes	Yes
Wapiti	Yes	2019	Yes	Yes
Watcher	Yes	2017	No	Yes
WATOBO	Yes	2017	No	Yes
WebScarab	Yes	2011	No	No
Wfuzz	Yes	2019	Yes	No

Table 3 - Application vulnerability scanners I found mentioned in reviews

Eliminating those that have not been recently maintained immediately produces a much more manageable list. I allowed one inactive product—Arachni—into the list of finalists because knowledgeable reviewers praised it highly.

Product	Last Release	Notes
Arachni	2017	Out of date library dependencies
OWASP ZAP	2019	Thriving community
Scan My Server	2019	Limited to scanning one domain once a week
W3af	2019	Out of date library dependencies
Wapiti	2019	Limited set of vulnerabilities

Table 4 - Application vulnerability scanners that met my initial criteria

3.4 Recommendation

I tested most of these by attempting to install them and running them against two well-known test websites with known vulnerabilities (*Mutillidae* and *Juice Shop*; see Resources.) Two of the scanners quickly dropped from consideration. W3af has dependencies on long-out-of-date libraries that prevented me from installing it even on Ubuntu, the system where the creator says W3af is tested. (Someone more persistent than I might be able to make it work.) Wapiti installed and ran, but it missed some findings that other tools reported. It works, but there are better options. That left three contenders: Scan My Server, Arachni, and OWASP Zed Attack Proxy (ZAP.)

Scan My Server is the easiest of the three to use, but the free license allows scanning only one domain once a week. Also, since the product is SaaS, the test site must be accessible on the public internet. The free version of Scan My Server could work for monitoring an existing site but is clearly not meant to support ongoing software development.

Arachni was in many ways the scanner I wanted: a command-line tool you can point at a site, leave running, and then get a clear report with all the right info. Unfortunately, like W3af, Arachni depends on long out-of-date libraries. Installing it required finding a workaround, and even then a broken dependency interfered with plugins needed for automating authentication. The scanner can't log in to a site. That's a deal-breaker. The creator has acknowledged the problem on the support forum but the code base has been inactive for two years and the problem remains.

OWASP ZAP wins my recommendation here because it is actively maintained, easy to install, detects a good range of vulnerabilities, and is supported by a community of developers producing plugins, documentation, video tutorials, code updates, and regular releases. It is not, however, as easy to use as some, and the reporting function is weak.

The OWASP ZAP scanner is designed in part as a tool for penetration tests. Much of its design is intended to support interactive exploratory testing. It produces reports, but overall the scanner is built to support analyzing results directly from the user interface. ZAP does include automatic attacks designed to uncover a range of possible vulnerabilities. As you learn more about vulnerabilities some of the interactive features may interest you as well. It's a tool you can grow into.

4 Understanding the Results

Each potential problem that the scanner reports come with two sorts of details. Some of the details explain generally what kind of vulnerability was found and what the standard defenses for such a vulnerability are. Other details explain exactly what the scanner saw on your web page that aroused suspicion.

4.1 Information About the Vulnerability

Table 5 lists generic details the ZAP scanner provides about each vulnerability. The scanner will show the same values for these fields every time it finds the same kind of vulnerability.

Item	Description
Alert Name	The name of a potential risk that may exist in the application.
Description	Generic information about the vulnerability to explain what the scanner thinks it found.
Risk Level	The degree of risk the vulnerability may create.
Confidence	How likely the alert is to represent a genuine vulnerability.
Solution	Generic advice for mitigating vulnerabilities of this type.
CWE ID	A Common Weakness Enumeration identifier from the catalog of flaws maintained at cwe.mitre.org. The CWE ID is useful for researching details about the specific vulnerability.
Reference	URL(s) pointing to more detailed information about the vulnerability. ZAP may point to a Microsoft blog, an OWASP reference page, or some other reputable source.

Table 5 - Details to explain the type of vulnerability found

4.2 Information About the Attack

The next table lists details the ZAP scanner provides to explain why it thinks your specific page might have a problem. These details tell what the scanner sent to the web server and what it saw in the response.

Item	Description
URL	The page where the risk was found.
Method	The HTTP verb that produced the suspect result (typically GET or POST.)
Parameter	A specific place in the HTTP message that ZAP manipulated to produce the vulnerability. It is the name part of a name/value pair in the query string or in a form. To understand what happened you'll want to find the parameter with this name and look at the value that ZAP assigned to it.
Attack	An excerpt from the HTTP message that the scanner sent to the application. It is the piece of the message that the scanner crafted as an attack attempting to uncover a vulnerability.
Evidence	An excerpt from the HTTP message that ZAP received from the server. The excerpt is the part of the message where ZAP sees a problem.
Request Header & Body Response Header & Body	The complete HTTP traffic from which the scanner concluded you may have a vulnerability. Often the Parameter, Attack, and Evidence values are enough, but sometimes you need to see the entire interaction for context.

Table 6 - Details to help you reproduce the problem in your application

Understanding what the attack did is critical. You must understand the attack in order to reproduce the problem. People new to vulnerabilities may find this step difficult at first, but it is a valuable learning opportunity.

The next step is key: how easy is it to understand what the scanner tells you? Can you reproduce the problem? Would you be able to describe it to a developer? Making sense of the findings is not as hard as it may initially seem. I will demonstrate with three examples of problems ZAP found in the Mutillidae site.

4.3 First Example: Path Traversal

Table 7 lists the most useful details from a ZAP report of a path traversal problem. *Path traversal* occurs when a malicious person invents URLs to request files that should not be accessible. An improperly configured server may return files that are not meant to be part of the web site's content.

Item	Description
Alert Name	Path Traversal
URL	http://10.133.1.4/mutillidae/index.php?page=%2Fetc%2Fpasswd
Risk	High
Parameter	Page
Attack	/etc/passwd
Evidence	root:x:0:0

Table 7 - Details ZAP provided about a path traversal vulnerability

In this particular case the scanner says it manipulated a parameter called *Page*, crafted an attack value of */etc/passwd*, and found in the response evidence that the attack had succeeded. The evidence is the string *root:x:0:0*. Readers familiar with Linux may find that report perfectly clear. In case you do not, here is what happened.

The browser encountered a URL with a query string value named *page*. This was the original URL:

https://10.133.1.4/mutillidae/index.php?page=login.php

The scanner tried replacing *login.php* with the name of a different file. It requested a well-known Linux operating system file that has nothing to do with any web server:

http://10.133.1.4/mutillidae/index.php?page=/etc/passwd

And of course in HTTP special characters such as slashes must be HTML-encoded, so the actual attack the scanner sent to the server looked like this:

http://10.133.1.4/mutillidae/index.php?page=%2Fetc%2Fpasswd

To reproduce the problem, paste the attack URL into the browser. Instead of rendering the login page, the browser renders the contents of the /etc/passwd file where Linux stores information about system users (see Figure 1.) Obviously the scanner has indeed found a real vulnerability in the Mutillidae site. It also provided enough information to reproduce the problem and create a defect report.

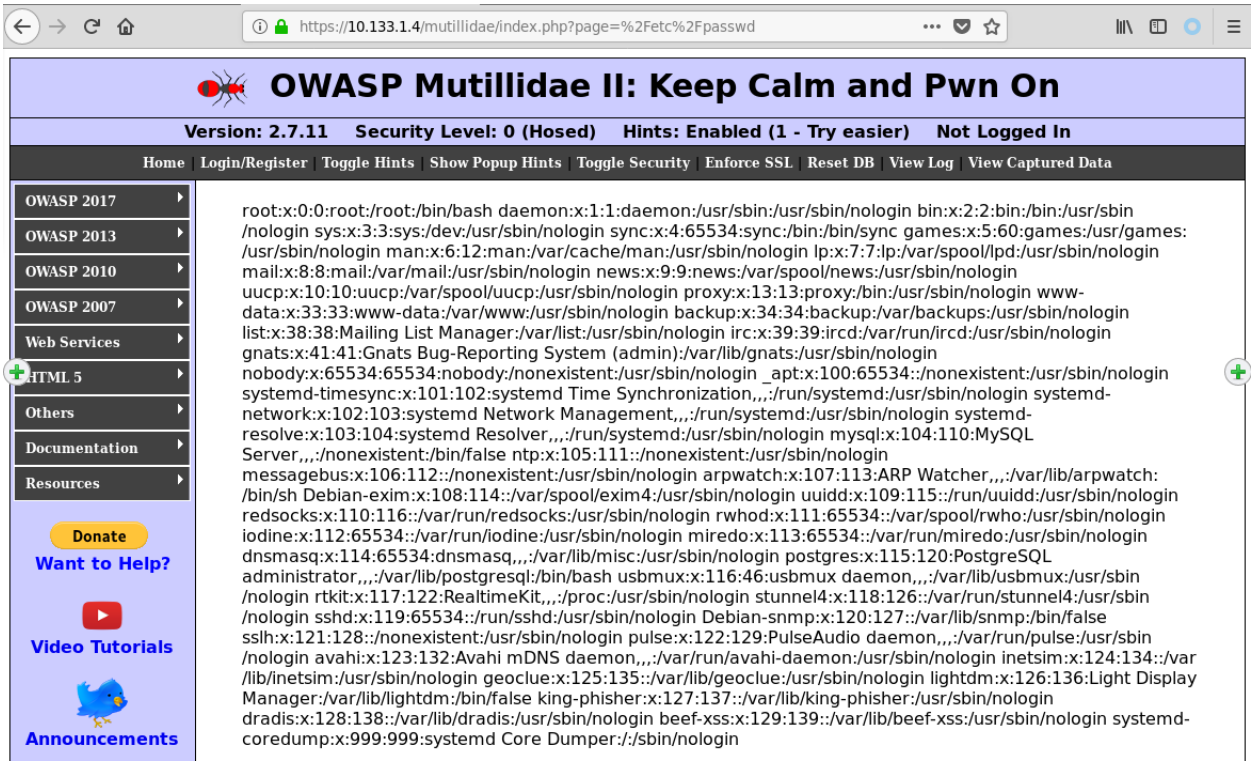

Figure 1 - The result of a path traversal attack in a vulnerable website

4.4 Example Two: A False Positive

Most scanners are not 100% accurate. They sometimes draw your attention to things that are not really problems.

Item	Description
Alert Name	Session ID Cookie Accessible to JavaScript
URL	http://10.133.1.4/mutillidae/index.php?page=register.php
Risk	Medium
Parameter	showhints
Attack	cookie field: [showhints]
Other Info	session identifier cookie field [showhints], value [1] may be accessed using JavaScript in the web browser The url on which the issue was discovered was flagged as a logon page.

Table 8 - Details ZAP provided about an allegedly insecure cookie

The scanner noticed in the HTTP REQUEST a cookie value that lacked an httponly attribute:

Cookie: showhints=1; PHPSESSID=rpiq3oo77e7rssp2gbjv6gfus2

There are in fact two cookies on this line—showhints and PHPSESSID—and neither one has the httponly attribute. The alert for the showhints cookie is wrong: showhints is not a session identifier. It is a flag governing a minor behavior in the user interface. It is true that the showhints cookie could be accessible to malicious JavaScript, but the cookie setting does nothing sensitive: it just turns on or off the display of hints in the user interface. You can safely ignore this alert.

(The ZAP scanner also generated the same warning for the other cookie, PHPSESSID. *That* alert is valid. PHPSESSID is indeed a session identifier and should have the httponly attribute.)

4.5 Third Example: SQL Injection

In this last example ZAP is reporting something more complicated: a SQL injection vulnerability. (If you are not already familiar with SQL Injection you can easily find documentation on the web[1].)

Item	Description
Alert Name	SQL Injection
URL	http://10.133.1.4/mutillidae/index.php?page=login.php
Risk	High
Parameter	username
Attack	brian' AND '1'='1' --
Other Info	The page results were successfully manipulated using the boolean conditions [brian' AND '1'='1' --] and [brian' AND '1'='2' --]

Table 9 - Details ZAP provided about a SQL injection vulnerability

The URL is the site's login page, and the attack involves a parameter called *username*. ZAP has attacked by submitting unusual values for the user name when logging in. The *Other Info* section says that the attack actually involved *two* attempts to log in, not just one:

Figure 2 - The first part of ZAP's attempt to find a SQL injection vulnerability

[1] For example, the Hacksplaining site (sponsored by NetSparker) has good visual explanations for a number of common vulnerability types including SQL injection: https://www.hacksplaining.com/lessons.

Figure 3 - The second part of ZAP's attempt to find a SQL injection vulnerability

Of course the site has no users with names like *brian' AND '1'='2'* -- . Both logins should fail for the same reason: no such user exists. The scanner tries both logins, compares them, and notices that in fact they do *not* produce the same result. The fact that the results differ strongly suggests one of those inputs altered the semantics of some SQL statement on the back end. No user should be able to do that!

Attempting to reproduce the scanner's SQL injection attack in the Mutillidae site reveals that the first post actually succeeded in logging in with no error message at all! (If you try this, be sure to include the final space after the two hyphens for each input. The presence of the space is not obvious in the screen shots, but you can see it clearly in the *Other Info* field where the space appears between the last hyphen and the right square bracket: "--]".)

4.6 Risk Ratings

The scanner did not understand that the first POST produced a successful login. The scanner reached its conclusion solely from noticing that the results of the two POSTs differed. It does not try to determine how a hacker might exploit a flaw, only that a flaw exists. Understanding how much damage a hacker might do with that flaw in your particular site would be very useful, but it is beyond the capacity of a scanner to determine. ZAP rated the SQL injection risk High because SQL injection often results in serious problems such as exposed data, not because on this particular page the attack achieved an unauthorized login.

Knowing the actual danger in your site would help in assessing risk and prioritizing work, but accurate assessments generally require knowledge of how hackers work. Experienced development teams may well be able to determine that the actual risk in their own site is lower (or higher, but more often lower) than the scanner's rating. Without that kind of experience, though, the best indicator is the risk level assigned by the scanner.

In this case the scanner is certainly right: this SQL injection is a high-risk defect.

4.7 Gaining Expertise

People new to application vulnerabilities may find their first scanner report daunting. It's full of phrases like *Path Traversal*, *SQL Injection*, *Information Disclosure*, and *Cross Site Scripting*. But as with many things in software development, hard tasks become much easier when you take small steps and iterate. The scanner helps immensely: it produces a prioritized list of security topics that would be worthwhile to learn because they are relevant to *your* web application. This is a great way to begin learning more about application security. Pick the scanner findings up one at a time in order. Take whatever time you need to learn the first one, and then go on to the next. Maybe each item takes a day, or maybe a week. Research each new vulnerability until you can make sense of the scanner's attack report and reproduce the problem for developers. Any speed, even a slow one, counts as progress. Your expertise increases item by item.

Lots of resources are available to explain vulnerabilities: web sites, forums, videos, meetups—I'll list some in the Resources section at the end of this paper.

4.8 Am I Invulnerable Now?

Suppose you have worked through all the scanner findings and fixed every reported issue. Does that mean your site has no vulnerabilities?

Unfortunately, no. It means only that the site has none of the vulnerabilities the scanner knows how to find. Not all vulnerabilities are easily discovered by a script. Some are too complex for easy scripting. Some involve non-deterministic behavior. Some are newly discovered or not well known. And some are in the infrastructure—the network and servers—not the application.

But addressing scanner findings certainly makes an application harder to hack. It means you have at least avoided a set of common, well-known problems. That is a worthwhile accomplishment. And along the way you may derive other benefits as well, as I'll suggest next.

5 Managing Vulnerabilities

I hope at this point you are convinced that with a manageable amount of effort you can download a scanner, get results for your application, and turn the results into actionable work items. If you continue to scan periodically, to understand the findings, and to work with your teams to fix them, you will be doing more than fixing vulnerabilities. You will also be:

- making yourself more knowledgeable about security—which was, I imagined at the start, one of the motivations that might draw you into an effort like this.

- spreading knowledge by drawing your team into conversations leading to fixes. A more informed team should produce fewer vulnerabilities to start with.

And with that effort you are beginning influence your team's approach to quality. But you can go further. With just a little more effort you can turn what you have already started into a *vulnerability management program*. The goals of a management program include:

- ensuring the vulnerabilities are found and fixed systematically
- monitoring progress to see how you are doing and where you can improve
- creating an audit trail so you can prove to customers and auditors that you are doing the right things

The center of a vulnerability management program is a document—perhaps a working agreement on a wiki—in listing actions a team will take in order to accomplish those goals. Be sure to say who is responsible for performing each action and when they will do it.

Here's an example:

Working Agreement for Vulnerability Management

Purpose

This procedure ensures that the software development team systematically eradicates risky vulnerabilities from their code.

Process

1. QA must run a vulnerability scan at least monthly and save scan reports in a shared folder.
2. QA must create stories for any items that in their judgement merit a High or Critical severity rating.
3. Developers must triage Critical items within one business day and High items within one week.
4. Teams will prioritize triaged vulnerabilities appropriately for the risk each represents.
5. QA will call a quarterly meeting to review vulnerability activity with management.
   - The meeting will evaluate progress and consider improvements.
   - The report and notes for each quarterly meeting will be saved in the shared folder.

Figure 4 - An example of a vulnerability management procedure

Adjust the frequency of activity and the severity thresholds to suit your situation. This agreement allows teams to override the scanner's decision about severity if their investigation determines that in their website the vulnerability introduces more or less risk that the scanner's generic estimate. And importantly, the agreement results in *artifacts of compliance*—evidence to show auditors. Artifacts include:

- the working agreement (auditors like written procedures)
- saved scan reports to show the scans were run with the expected frequency
- defect reports to prove that risky findings were processed and resolved
- quarterly reports to prove the team is monitoring progress and adjusting as necessary to get the best results

If your efforts haven't received attention from management yet, maybe they will when you start reporting on the quantity and severity of vulnerabilities found over time. You can look for patterns in the results. Are we more secure this month than we were last month? Are certain code areas more prone to vulnerabilities than others? Why? Do certain types of vulnerabilities occur more often? If so, perhaps it's time for a brown-bag presentation on how teams can avoid falling repeatedly into a particular trap.

6 Awareness and Training

As a team gains experience finding and fixing vulnerabilities they generally make choices about how to solve certain common problems within their own codebase. They may choose particular defenses for session fixation or cross-site request forgery. They may build a library of regular expressions for validating user input. They may make decisions about what error messages should or should not say. Start recording these decisions on your wiki, and you have a *Secure Coding Guidelines* document for the team to follow.

Share the *Secure Coding Guidelines* with new team members. And now that we are talking about onboarding, maybe new team members should also view YouTube videos about common vulnerabilities to ensure that everyone starts with at least a basic understanding of what vulnerabilities are and why they matter. Training and guidelines become part of your vulnerability management strategy.

By now we have gone much farther than you might have imagined when I first talked about running a scanner: you have a team of people trained to find vulnerabilities; practices agreed and documented for producing secure code; a vulnerability management program that produces quantifiable reports to identify trends and improve processes; and training to ensure that new team members are capable of participating in your security process.

7 Next Steps

We have accomplished much, but vulnerability management is only one piece of a complete security program. Security has a place in every phase of the product development lifecycle, as Table 10 suggests.

Lifecycle Phase	Security Activities
Initiation and Requirements	• Identify security requirements along with functional requirements.
Design	• Identify security risks and consider possible ways to reduce risk. (Look up *threat modeling* for help here.) • Identify use cases and abuse cases to drive requirements. • Evaluate third-party libraries for security and reliability.
Implementation	• Follow *Secure Coding Guidelines*. • Do code reviews. • Write test plans that include security requirements. • Write unit tests that include security requirements. • Run scanners (static, dynamic, third-party, and network) • Integrate automated tests with the CI/CD pipeline. • Ensure only approved changes are deployed to Production.
Operations/Maintenance	• Manage vulnerabilities. • Monitor site activity for potentially malicious behavior. • Patch out-of-date software.
Disposition	• Remove code that is no longer needed. Reduce the attack surface. • Periodically purge data that is no longer need. Hackers can't steal what isn't there.

Table 10 - Security activities appropriate to each phase of a software development lifecycle

8 Summary

At the beginning I promised to show how QA can take a leading role in initiating a key part of any security program: vulnerability management. Even with no budget and little security experience you can run a free vulnerability scanner to get a prioritized list of things to learn and fix. This is a great way to start improving both your product's security and your own professional expertise.

To review, these are the basic steps:

1. Talk to team members informally about exploring what a scanner uncovers. Be sure they're willing to participate.
2. Choose a scanner such as ZAP. Run it on your web application.
3. Pick the most important vulnerability in the report and learn about it. When you understand it well enough to create a defect report from the scanner results, move on to the next finding.

4. As the team begins working on fixes together, invite them to think about how to avoid introducing similar vulnerabilities in the future. Record on a wiki page whatever defenses you agree on. This becomes the *Secure Coding Guidelines*.
5. Ask the team how often they want to run the scanner and what kinds of problems they will commit to fixing. Capture those decisions in a working agreement.
6. When new people join the team, be sure they know at least as much as you do about vulnerabilities. If they don't, train them.
7. Periodically, perhaps quarterly, review the scanner results and defect reports to identify progress and problems.
8. If your business is regulated (by HIPAA or PCI, for example), or if you have customers who sometimes want to audit your security practices, be sure the working agreement includes creating records of ongoing vulnerability management efforts—scanner results, bug reports, and notes from periodic reviews.

Talk to management and engage their support for automating parts of the security work and integrating it with the build pipeline. As you gain experience with multiple types of vulnerabilities, bring that knowledge into design discussions. One new testing tool can be the first step in creating a culture of quality and security.

Resources

Criteria for Evaluating Vulnerability Scanners

Black, Paul E.; Elizabeth Fong; Vadim Okun; and Romain Gaucher. 2007. *Software Assurance Tools: Web Application Scanner Functional Specification Version 1.0*. NIST Special Publication 500-269. National Institute of Standards and Technology (NIST).

Chen, Shay. 2012. “A Step-by-Step Guide for Choosing the Best Scanner.” InfoSec Island. http://www.infosecisland.com/blogview/21926-A-Step-by-Step-Guide-for-Choosing-the-Best-Scanner.html (accessed August 14, 2019.)

Filkins, Barbara. 2017. *Asking the Right Questions: A Buyer’s Guide to Dynamic Scanning to Secure Web Applications*. A SANS Whitepaper.

Web Application Security Consortium. 2009. “Web Application Security Scanner Evaluation Criteria.” http://projects.webappsec.org/w/page/13246986/Web%20Application%20Security%20Scanner%20Evaluation%20Criteria (accessed August 14, 2019.)

Comparisons of Vulnerability Scanners

Chen, Shay. 2017. “Evaluation of Web Application Vulnerability Scanners in Modern PenTest/SSDLC Usage Scenarios.” http://sectooladdict.blogspot.com/ (accessed August 14, 2019.)

InfoSec Institute. 2019. *14 Best Open Source Web Application Vulnerability Scanners*. https://resources.infosecinstitute.com/14-popular-web-application-vulnerability-scanners/ (accessed August 14, 2019.)

Upguard. 2019. “Arachni vs OWASP ZAP.” https://www.upguard.com/articles/arachni-vs-owasp-zap (accessed August 14, 2019.)

Vulnerability Scanners Discussed in This Paper

Arachni Web Application Security Scanner Framework
https://www.arachni-scanner.com/

OWASP Zed Attack Proxy Project (ZAP)
https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Scan My Server
https://scanmyserver.com

w3af Web Application Attack and Audit Framework
http://w3af.org/

Wapiti
http://wapiti.sourceforge.net

Vulnerable Websites Mentioned in This Paper

Mutillidae
http://www.irongeek.com/i.php?page=mutillidae/mutillidae-deliberately-vulnerable-php-owasp-top-10

OWASP Juice Shop Project
https://www.owasp.org/index.php/OWASP_Juice_Shop_Project

Help Understanding Vulnerabilities

DevCentral, a technology community sponsored by F5, has posted talks on YouTube describing common web application vulnerabilities.
https://www.youtube.com/user/devcentral
https://devcentral.f5.com/s/

Hacksplaining is a training project sponsored by Netsparker. The Hacksplaining site has free visually appealing animated online explanations of common vulnerability types.
https://www.hacksplaining.com/lessons

OWASP local chapters. The Portland chapter meets monthly for networking and presentations. Anyone is welcome to attend. Here you can find people to talk to about application security issues. Look for meeting announcements on Meetup.
https://www.owasp.org/index.php/Portland
https://www.meetup.com/OWASP-Portland-Chapter/

OWASP Top Ten – 2017: The Ten Most Critical Web Application Security Risks. 2017. This PDF from The Open Web Application Security Project has a page for each of the top ten vulnerability types. Each page has a short explanation of the vulnerability along with information about the risk it represents, how it can be exploited, and sources of further information.
https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Udemy has several inexpensive courses on web application security.
https://www.udemy.com/course/web-application-security/
https://www.udemy.com/course/web-application-security-for-absolute-beginners-no-coding/