Section: PUBP

Web Application Penetration Testing Course For Developers

Zach Steele

Problem Statement: In this section, new text is blue for the professor's convenience.

There is a large difference in priorities and skillset between penetration testers and code developers, leading to a troubling amount of insecure web applications and difficulty communicating vulnerabilities to application teams. While developers are constantly told to follow secure coding practices, oftentimes even utilizing security focused development frameworks, they rarely prioritize secure coding practices. (Secure Code Warrior, 2022) Perry et al. (2023) prove that insecure code is going to become even more common as AI code assistants become more popular because developers are over confident in its ability to write secure code. This problem will continue to grow if developers do not start taking cyber security and secure coding practices more seriously. In one study, 68% of participants stated that they only did secure code training because of a compliance need or recent exploit. (Security Journey, 2024)

Personally, I believe that developers do not take more cyber security training programs largely because of the gap in cyber security courses. These courses and programs fall into two categories based on my experiences as a penetration tester responsible for teaching application teams how to test and remediate vulnerabilities. First, developer focused resources are too high level to be helpful. These courses may teach the basics of specific misconfigurations or exploits, but they do not provide many opportunities for developers to walk away with knowledge that can be applied to their current applications. In one study, developers stated that training is not hands on, and/or it is taught in a vacuum. (Secure Code Warrior, 2022)

The second category of resources are targeted towards penetration testers, meaning the course is extremely in depth, requires specialized tools, and a massive time investment from the audience. As an example, PortSwigger's Cross-Site Scripting section includes 30 labs with each solution video ranging from 50 seconds to over 20 minutes. If a developer wanted to learn the risks, remediation, and how to exploit XSS at a basic level, they would have to parse through multiple pages of documentation and choose between 50 second videos with zero context/explanations or 20+ minute videos explaining testing methodologies and edge cases per lab. Lastly, these walkthrough videos do not use tools that are available to developers in the average corporate environment. (PortSwigger, 2025)

Most developers are limited to tools focused on the development lifecycle due to the risks associated with scanning your application. (Young, 2025) We can see evidence of this in the Fortra (2024) survey, stating that 24% of cyber security professionals don't use penetration testing tools and an additional 33% only use open source tools. This makes sense as cost was the second most important factor when considering penetration testing software. While this survey was aimed at cyber security professionals, we can assume that developers would have the same or lower buy-in to cyber security tools. An additional issue with utilizing new tools is the time commitment. Velez (2020) reports on the many struggles that junior developers face when attempting to install programming tools. This issue compounds the current time limitations that developers deal with in modern frameworks. Rajapakse et al. (2021) supports this point by proving that DevOps workflows do not allow for the time commitment of SAST and DAST tools.

From my own experience researching this topic, I often find the second type of courses do not cover a wide range of topics. Generally, these courses are aimed at teaching exploitation, but do not cover policy

based findings or misconfigurations, such as response header infractions. Please see the figure below showing a summary of my own opinions and analysis after completing other courses that are publicly available. Please note that this is a high level view of these courses and is not intended to be an in-depth review of each course. The purpose is not to disparage other courses, but to instead point out gaps or audience/tool differences that I hope to fill with my solution.

Course Name	Length	Tools Utilized	Pros	Cons
DIY Web App Pentesting Guide	Medium Article	Kali, Recon-ng, whatweb, nmap, Nikto, SQLMap	Showed how to catch low hanging fruit with scanners	Most developers cannot openly scan their applications or install tools like nmap. Only checked a few items in the application.
OWASP Web Security Testing Guide	100+ Pages	30+ tools and scripts used throughout the site.	The most comprehensive penetration guide online.	Information overload for the average developer. Does not show application behavior outside of full compromise.
Web App Testing	11 Hours	Burp	Covers each vulnerability in great depth.	Does not cover a wide range of vulnerabilities. Did not explain risks and remediation outside of test cases.
Ethical Hacking 101	2.75 Hours	Burp, Zap.	Used multiple test applications to show the risks of each vulnerability	Did not explain application behavior in partial exploitation situations or remediation.
Penetration Testing Course for Beginners	5 Hours	Burp, SQLMap, Nikto	Showed real world testing for vulnerabilities.	Only showed one or two examples per vulnerability covered, leaving the assessors to their own devices for further payload development. Did not cover full scope of risks associated with each vulnerability.
PortSwigger	270+ Labs	Burp, SQLMap, python scripts.	Smaller scope than the OWASP training, but with hands on training and better explanations. The number one resource for web application penetration testers to learn beginner to advanced exploitation.	Targeted towards web application penetration testers, meaning the material gets very detailed and covers many edge cases.
How to Analyze Code for Vulnerabilities	1.3 Hours	IDE	Podcast format, allowing for digestable explanations and discussions.	Almost entirely high level powerpoint bullets without many additional resources for devs to review.
Secure Coding – Best Practices (also for non developers!)	57 Minutes	Provides tool examples, but not required.	Covers a healthy range of topics with short exploit examples and blanket mock code snippets. Provides further training resources at the end.	Lacks context and true examples of application code, resulting in infromation that is hard to convert to our current applications.
Web Application Penetration Testing Tutorial	4.5 Hours	Burp, Fiddler, nmap, NetSparker	Shows a wide range of misconfigurations and high priority security vulnerabilities. Even opens the video with what the coures lacks(SQLi and Session Hijacking). The closest course to what I am trying to acheive with this project.	Does not target dev teams, meaning the course does not encourage a checklist and frequent test of your own applications. Additionally, missing a handful of misconfigurations that are extremely common in modern web apps.
Programming Foundations: Secure Coding	2 Hours	None	Great high level overview of secure practices throughout the entire lifecycle of a project.	Requires monthly membership. Entirely explained using animated content, does not show a single code snippet or exploit example.
Developing Secure Software	1.5 Hours	None	High level overview of many concepts ranging from coding practices to security testing. Easily digestable powerpoint format.	Almost entirely powerpoint bullet points, no real information for an app team to action on without doing their own third party research on each topic.

Figure 1: Course Comparison (Steele, 2025)

I believe this pushes away most developers that were willing to commit a small amount of time to upskilling their cyber security knowledge. According to one study, 44% of respondents stated the reason for not taking secure coding courses is because they were not aware of a good course and another 44% blamed time constraints. (Robinson & Russo, 2024) It is important to note that the solution described below may not solve the issues covered in this 88% of respondents because time constraints and course quality may be subjective. Ultimately, I believe that this gap in training material is the reason we continue to see an increase in vulnerabilities every year, even though most developers receive annual penetration tests on their applications and secure coding training. (Tribbey & Winterfeld, 2023) If application teams were capable of testing their environments and implemented security checks on a regular basis, we would most likely see a decrease in the 31% of critical vulnerabilities and 52% of lower priority vulnerabilities that go unremediated. (Lamar, 2025)

Solution Statement: In this section, new text is blue for the professor's convenience.

This project will teach developers to test their own applications for vulnerabilities via a training course dedicated to quickly learning penetration testing tactics for common vulnerabilities in order to increase the security of modern web applications.

Specifically, I plan on teaching developers how to test their applications from a similar set of material that penetration testers use to conduct assessments. Most web application penetration testers utilize a checklist with a set of vulnerabilities to test, payload lists, and an intercepting proxy tool to bypass client side controls. My goal is to use a similar framework that excludes complex attack types and unnecessary overhead. My target audience is developers who are looking for a way to reproduce findings or test new code for vulnerabilities before pushing to production. However, I will be providing third party links and suggestions on how to learn more as we go through the checklist for the audience members that would like to take penetration testing to the next level.

Lastly, I would like to explain my methodology for evaluating the effectiveness of this solution. It is easy to express the need for this course when reviewing the widespread security flaws across the internet. As an example, Pacheco (2020) found that only 2% of the top 1000 visited sites have a properly implemented Content-Security-Policy (CSP). In a few minutes, we could teach developers how to build a strong CSP to prevent exploitation of cross-site scripting and framing attacks. However, measuring the full impact and objective results of this course would require a set of developers who plan to complete the whole course and a much larger time window to analyze actions taken based on the material. Most firms would not allow developers to share vulnerabilities with external parties due to reputational risk, causing some limitations in our ability to test the effectiveness of this course on true corporate applications. Since we do not have enough time in this class to both complete the penetration testing course and test its effectiveness from start to finish, I will ask developers with varying ranges of security knowledge to review the material and provide feedback. If they have time, I will encourage them to test the material on their own applications or test applications. Additionally, they can complete specific, non-malicious, passive checks on public applications. Participants will document any vulnerabilities found using the course material, allowing us to prove the effectiveness on our sample audience. This feedback will include how effective they think the course will be once completed, and their level of confidence checking these items in their own applications. Additionally, since part of this course is based on changing the behavior of application teams, I will be asking the individuals what action items they plan on implementing into their own teams based on the course. Application owners can provide feedback explaining the value this course adds compared to current solutions in their own environments, such as company required training and code review checklists. This will give us perspective from the full range between security professionals and developers with no security background. Hopefully, this will result in a clear picture of the potential impact the complete course will have on a true audience. The figure below shows multiple survey questions in a bulleted format. Please (**See Appendix B)** for the updated pre and post training surveys.

-Which videos did you watch?
-Did the videos teach you about new vulnerabilities?(Yes/No)
-Did the videos improve your understanding of risks and remediation guidance for the vulnerabilities shown?
-Did you use the methodologies shown in the videos to find vulnerabilities in an application? If not, skip down to the next section of questions.
-If Yes: How was your experience attempting to follow the videos? Please provide feedback on the pros and cons of the material, explanations, and format.
-If Yes: Were you able to identify any vulnerabilities or misconfigurations in the application you tested?(Yes/No) If so, what were they?
-On a scale of 1-10, how confident are you that you could test your enterprise application for the common web application vulnerabilities covered in the videos that you watched?
-Do you believe that an application team could reasonably follow this entire series and identify vulnerabilities in their environment related to the checklist items covered in the videos?(Yes/No)
-Would you recommend app teams implement this testing on a regular basis?(Yes/No) If so, how often should app teams conduct these tests?
-Would it be beneficial for entire dev teams to watch this series?(Yes/No) Why or why not? Please note that this question is focused on long term outcomes.

Figure 2: Evaluation Survey, abridged format (Steele, 2025)

Completed Tasks (Last 2 Week):

- Completed the checklist items and GitHub documentation for the payloads required section and login page checks. Please visit the repo to view all work papers: https://github.gatech.edu/zsteele3/ZS6727Summer2025
- Next, I continued to add notes and outlines for the video series to prepare content throughout the videos.
- Afterwards, I prepared, filmed, and edited the entire payloads required section and login page checks. This resulted in video training material covering 9 checklist items with a few optional course videos, 70+ minutes of new content. To see the current course video series with detailed video descriptions and timelines, visit: https://www.youtube.com/watch?v=oNDEU_uNtzI&list=PLqPCUirqsN_x_seY51DiivfV-CjgJDKUp
- Updated the survey for developers, security professionals, and app team managers to based on feedback received from peers about the survey questions.
- Sent out a pre training and post training survey to roughly 25 individuals and encouraged them to share with their teams. (**See Appendix B**) I have started interviewing participants once they complete the second survey.

Tasks for the Next Project Report:

Over the next two weeks, I will continue to build out the payloads in GitHub and complete the checklist items from Session management to vertical privilege escalation. Additionally, I will film videos from the session management section through the outro. Lastly, I will review survey results and follow up with individuals to learn more. This feedback will be turned into action items for future course improvement.

Questions I have or Issues I'm running into:

None.

Methodology Paragraph Summary:

During the first few weeks, I built the structure of all required documents and created the video order to ensure I have a pathway to follow with a clear understanding of the final outcome. Afterwards, I spent a few hours preparing and creating the documentation required for each video in order. For example, when it was time to film the SQL injection video, I built out the payload list, checklist, and found the exact labs and tools that will best demonstrate how to test for SQL injection. This methodology provides a clear path forward after every action item and makes it easy to see the work completed each week along with what is coming in the following weeks. It is important to note that some videos and deliverables may need to be completed out of order due to required resources or time to complete. For example, the introductory video cannot be completed before building out the course materials because that video will display the completed checklist and GitHub to give the viewer an idea of what we will be doing for the remainder of the course. As the course gets built out, I will be sending out a survey to developers, penetration testers, and other technology professionals to see how they have utilized parts of the course. More information will be provided in the evaluation section.

Timeline:

Week #	**Description of Task**	**Status**
W1	Installed VM, required tools and test apps.	Completed
W1	Created checklist template and first section.	Completed
W1/W2	Created GitHub and payloads page outline. https://github.gatech.edu/zsteele3/ZS6727Summer2025	Completed
W2	Created the outline for the video series.	Completed
W2	Created YouTube channel and uploaded placeholder video.	Completed
W3/W4	Built out the response headers and cookies checklist and GitHub pages.	Completed
W3	Recorded video for continuous review, response headers, and cookie testing.	Completed
W4	Conducted research on secure coding practices and filmed Video 3 - Best coding practices/remediation.	Completed
W4/W5	Created course evaluation survey and excel sheet showing pros/cons of current solutions.	Completed
W5/W6	Complete GitHub pages and checklist for all payload based exploits covered in the checklist.	Completed
W5/W6	Film videos 7(XSS) through 12(Login pages)	Completed
W5/W6	Reach out to developers for evaluation of material up to this point.	In Progress
W5/W6	Re-evaluate checklist items based on studies of most common vulnerabilities.	Completed
W7/W8	Complete videos 13(Session Management) to Video 16(Outro).	Not Started
W7/W8	Make changes on past material based on developer feedback.	Not Started
W7/W8	Film optional videos if time permits	Not Started
W9	Review entire project for places that may need improvement to make the series more cohesive.	Not Started

Evaluation: This section was completely re-written since the last progress report.

The solution section of my final paper details the reasoning behind every checklist item and how they were chosen to ensure maximum coverage given the limited time a developer will spend testing. As a brief summary, the items in the checklist were picked based on the percentage of vulnerabilities identified in those categories based on recent research papers and the time it takes to test the items. For example, Cross-Site Scripting is one of the more time consuming findings, but it was chosen due to the severity of exploitation and commonality reported by a penetration testing firm. (Freeman, 2023) An example of a check that is less time consuming is reviewing the Content-Security-Policy(CSP) response header. This is further discussed in both the progress report solution section and final report solution section.

After proving the checklist items are common vulnerabilities and high priority risks, we then have to evaluate the effectiveness of the payloads utilized in the course. The payload lists came from years of professional experience using my own flowcharts and payload lists to manually penetration test over 150 applications. I was unable to find any research explaining the success rate of specific payloads. However, the checklist items, course content, and payload lists are only effective if the viewers test their applications using the methods shown in the videos. For this reason, we came up with the two survey system explained below.

I plan to evaluate the success and effectiveness of this course by surveying multiple developers, application owners, and penetration testers with varying levels of development and cyber security knowledge. First, the participants will fill out a pre-training survey that sets a baseline for their current skillset and understanding of web application penetration testing. Additionally, there are questions asking about their opinions on current penetration testing and secure coding courses. This survey will be sent out before any participants have seen the course content. (**See Appendix B**)

After completing the first survey, the participants will receive the second survey along with links to all course material. The purpose of this survey will be to gather feedback to decide on future additional content, issues with past content and confidence in the material. Additionally, we hope to understand the action items that they plan on taking in their own teams, and encourage hands-on practice with the course material to identify vulnerabilities within test environments. Most developers would need approval before testing their code for some of the vulnerabilities covered in the course, and most likely could not share the results with us. However, all developers are empowered to test the material on training applications or complete non-malicious passive checks on public facing applications. This allows us to document the vulnerabilities that our audience was able to gather after completing sections of the course, measuring the effectiveness of the payload lists and testing methodologies. Our sample audience includes many participants that have never searched for web application vulnerabilities, proving that anyone can follow along with the material even if they do not have any prior pentesting or code development experience. Lastly, I will speak with all participants to extract any information that was not provided in the survey. The results of the survey should display a clear improvement in the individuals' understanding of the vulnerabilities and ability to test their own applications based on the videos that they watched. See (**Appendix B**) for the full survey.

This approach does have multiple limitations. First, it is very difficult to create an unbiased survey that fully encompasses the information that the researcher hopes to gain. Due to time constraints, the survey may not be perfectly unbiased and the questions may have room for improvement to better extract information from the participants to help achieve the goals mentioned previously. Next, participants do not have any incentive to complete the survey, leaving us with a low completion rate. However, it is important to note that the survey has only been in participants hands for a short period of time so this may

change. Lastly, the participants may be biased because of their relationships with me. I was able to mitigate this by giving the survey to multiple people with instructions to pass the survey around to their entire teams. Since the other team members do not know who I am or the purpose of these surveys, hopefully we will see less bias results.

Report Outline:

Please see (**Appendix C**) for the full report.

References:

Specific penetration testing payloads and remediation guidance will be linked throughout the GitHub repo and checklist resources section. The references below are all utilized in this document.

Secure code warrior survey finds 86% of developers do not view application security as a top priority. Secure Code Warrior. (Apr 05, 2022). https://www.securecodewarrior.com/press-releases/secure-code-warrior-survey-finds-86-of-developers-do-not-view-application-security-as-a-top-priority

A Study on Secure Coding Training. Security Journey. (2024, January). https://www.securityjourney.com/hubfs/SJ_StudyonSecureCodingTraining24_011624.pdf

Tribbey, B., & Winterfeld, S. (2023, April 18). *Slipping through the security gaps: The rise of application and API attacks* | Akamai. Akamai. https://www.akamai.com/blog/security-research/the-rise-of-application-and-api-attacks

Pacheco, P. (2020, September 3). *Content security policy limits dangerous activity... so why isn't everyone doing it?* Bitsight. https://www.bitsight.com/blog/content-security-policy-limits-dangerous-activity-so-why-isnt-everyone-doing-it

Robinson, C., & Russo, D. (2024, June). Secure Software Development Education 2024 Survey. https://www.linuxfoundation.org/hubfs/LF%20Research/Secure_Software_Development_Education_2024_Survey.pdf?hsLang=en

Lamar, J. (2025, April 14). *Key takeaways from the State of Pentesting Report 2025*. Cobalt. https://www.cobalt.io/blog/key-takeaways-state-of-pentesting-report-2025

Steele, Z. (2025, June). ZS6727Summer2025. GitHub. https://github.gatech.edu/zsteele3/ZS6727Summer2025

Perry, N., Srivastava, M., Kumar, D., & Boneh, D. (2023, December 18). *Do users write more insecure code with ai assistants?*. arXiv.org. https://arxiv.org/abs/2211.03622

Why secure code training doesn't stack up (and what you can do about it) - blog. Secure Code Warrior. (2021, April 8). https://www.securecodewarrior.com/article/why-secure-code-training-doesnt-stack-up-and-what-you-can-do-about-it

What is cross-site scripting (XSS) and how to prevent it?: Web security academy. What is cross-site scripting (XSS) and how to prevent it? | Web Security Academy. (n.d.). https://portswigger.net/web-security/cross-site-scripting

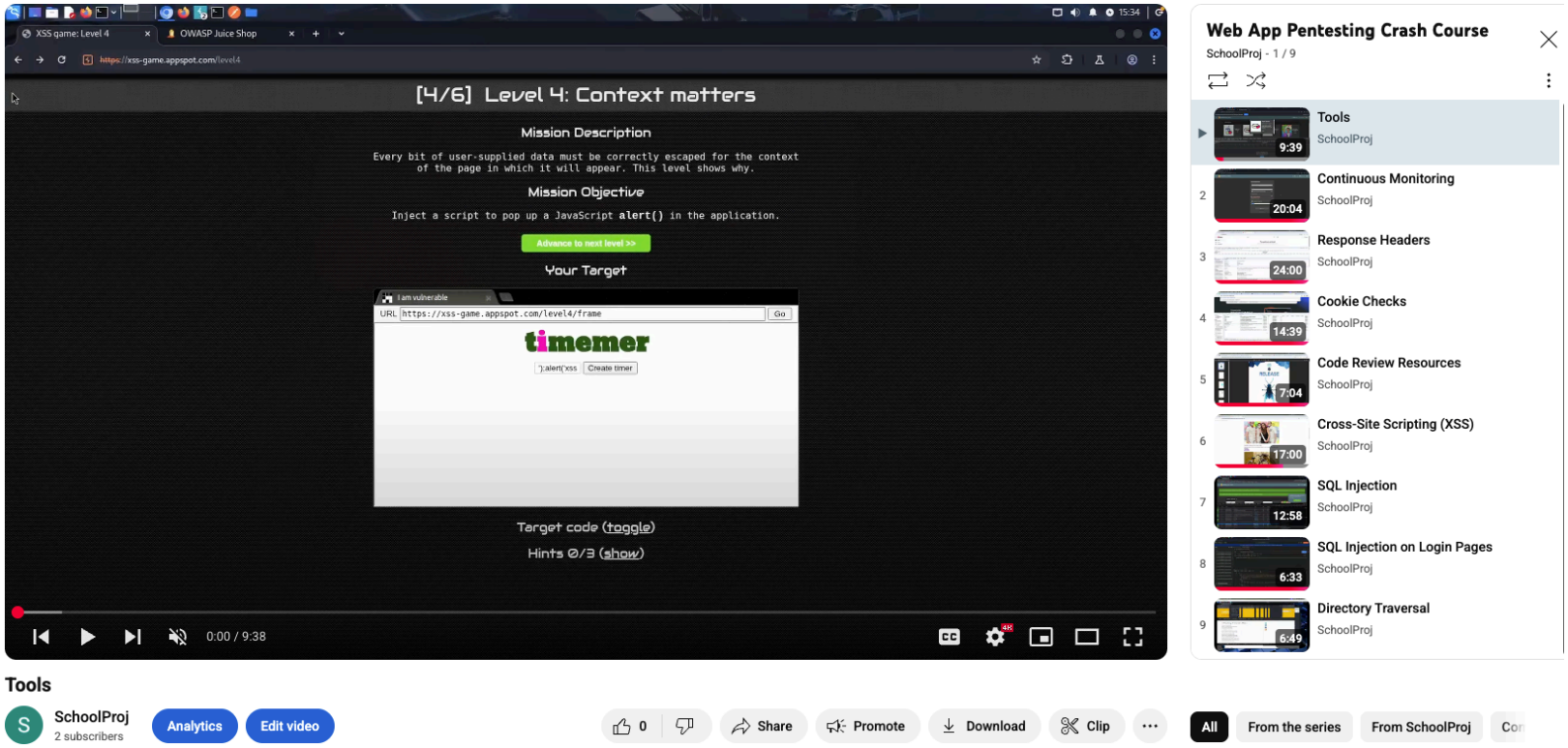
Young, L. (2025, May 19). *Penetration testing risks: Understanding the potential dangers of vulnerability assessments*. VaultMatrix.com. https://www.vaultmatrix.com/penetration-testing-risks-understanding-the-potential-dangers-of-vulnerability-assessments/

Velez, M., Yen, M., Le, M., Su, Z., & Alipour, M. A. (2020, March 14). *Student Adoption and Perceptions of a Web Integrated Development Environment*. ACM Digital Library. https://dl.acm.org/doi/pdf/10.1145/3328778.3366817

Rajapakse, R. N., Zahedi, M., & Babar, M. A. (2021, July 19). *An Empirical Analysis of Practitioners' Perspectives on Security Tool Integration into DevOps*. arxiv - Cornell University. https://arxiv.org/pdf/2107.02096

Freeman, C. (2023, November 14). *Understanding XSS: Why cross-site scripting still matters: Black duck blog*. Understanding XSS: Why Cross-Site Scripting Still Matters | Black Duck Blog. https://www.blackduck.com/blog/why-cross-site-scripting-still-matters.html

Appendix A: Web Application Penetration Testing Course for Developers (Steele, 2025)



Appendix A Figure 1: Course Playlist

	Pass	Rating	Description	Resources
				We encourage app teams to update the checklist and Payload lists with language specific links and company policies to ensure future developers follow the companies requirements.
Continuous Review				
Error Messages are Properly Handled		Low	Error messages should be short and generic. Do not show stack traces or informative messages to the user.	https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html
Autocomplete Disabled on Sensitive Input Fields		Low	Disable autocomplete on fields such as email, password, and security questions by setting autocomplete="off"	https://www.w3schools.com/howto/howto_html_autocomplete_off.asp
Sensitive Information is Masked		Medium	Sensitive Information should be masked to mitigate shoulder surfing attacks.	https://labex.io/tutorials/javascript-mask-a-value-28489
Sensitive Information Not in URL		Medium	Do not send sensitive information in GET request parameters.	https://cwe.mitre.org/data/definitions/598.html
Unneccessary Methods are Disabled		Low	Do not allow unsafe methods such as Trace or Connect on any endpoint. Disable DELETE, POST, etc. if not needed on the endpoint.	https://cyberwhite.co.uk/http-verbs-and-their-security-risks/
HTTP Disabled		High	Enforce HTTPS for all connections.	https://blog.matrixpost.net/redirect-from-http-to-https-using-the-iis-url-rewrite-module/
Response Headers				
Content-Security-Policy		Medium	Does not include unsafe directives, wildcarded directives, or wildcards in subdomains.	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Security-Policy https://csp-evaluator.withgoogle.com/
HSTS Response Header		Low	Use Strict-Transport-Security: max-age=63072000; includeSubDomains; preload. Only use preload if required.	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Strict-Transport-Security
Caching Headers		Low	Use Cache-Control: no-store. Less strict controls: Expires: 0, Cache-Control: no-cache	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Cache-Control https://www.rfc-editor.org/rfc/rfc9111.html
Information Leak Headers		Informational	Remove unneccessary response headers, such as: Server, X-Powered-By, X-AspNet-Version.	https://support.waters.com/KB_Inf/Other/WKB202501_How_to_disable_the_Server_HTTP_header_in_Microsoft_IIS
CORS Headers		Low	Do not set Access-Control-Allow-Origin to wildcard or allow reflected arbitrary origins/subdomains.	https://www.freecodecamp.org/news/exploiting-cors-guide-to-pentesting/#heading-exploitable-cors-cases
Cookies				
HttpOnly		Low	Prevents JavaScript from accessing the cookies, mitigating some XSS risks.	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Set-Cookie
Secure Flag		Low	Ensures cookies are only sent over HTTPS.	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Set-Cookie
SameSite Lax or Strict		Informational	SameSite cookies control cross-site request behavior to prevent CSRF. Lax: Cross-site requests will only send cookies in GET requests. Strict: Cookies will not send in any cross-site requests.	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Set-Cookie https://portswigger.net/web-security/csrf/bypassing-samesite-restrictions
Narrow Path		Informational	Sensitive/session cookies must have a restrictive path set to prevent other applications on the same server from using this cookie.	https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes
Narrow Domain		Informational	Sensitive/session cookies should have a narrow domain, set to a specific subdomain when applicable.	https://developer.mozilla.org/en-US/docs/Web/Security/Practical_implementation_guides/Cookies
Injections/Payloads Required				
Cross-Site Scripting(XSS)		Critical	The application should encode user input when reflected to the UI.	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html https://learn.microsoft.com/en-us/aspnet/core/security/cross-site-scripting?view=aspnetcore-9.0
SQL Injection		Critical	Use parameterized queries to prevent SQL injection via user input.	https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html
SQL Injection on Login Page		Critical	Login pages must use parameterized queries.	https://unix.stackexchange.com/questions/391866/regex-for-password-restricting-special-characters
Directory Traversal		High	User Input must be validated and canonicalized before being used to retrieve files from the server.	https://portswigger.net/web-security/file-path-traversal
OS Command Injection		Critical	Do not place user input directly into system command executions.	https://developers.redhat.com/articles/2023/03/29/4-essentials-prevent-os-command-injection-attacks#4_ways_to_prevent_os_command_injection_attacks
Login Page				
Weak Passwords Not Allowed		Medium	Enforce strong password policies.	https://www.cisa.gov/secure-our-world/use-strong-passwords
User Enumeration Not Possible		Medium	Ensure the application does not reflect different messages for valid or invalid users.	https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/04-Testing_for_Account_Enumeration_and_Guessable_User_Account
Account Lockout Enabled		Medium	Lock user accounts after X number of failed login attempts to prevent brute-force attacks.	https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks
Multi-Factor Authentication in Use		Low	Require multi-factor authentication at login.	https://auth0.com/blog/different-ways-to-implement-multifactor/

Appendix A Figure 2: Course Checklist

Cross-Site Scripting (XSS):

https://portswigger.net/web-security/cross-site-scripting

Enter the following payloads into user input fields and see how they are reflected in the application. If user input is reflected back unencoded and unfiltered, especially in javascript, then XSS is likely.

Alert() can be replaced with print() if your application is filtering out alert().

```
<b>test</b>
<script>alert(1)</script>

<img src=1 onerror=alert(1)>

javascript:alert(document.cookie)

"onload="alert(1)

${alert(1)}

alert`1`

'-alert()-'
```

SQL Injection

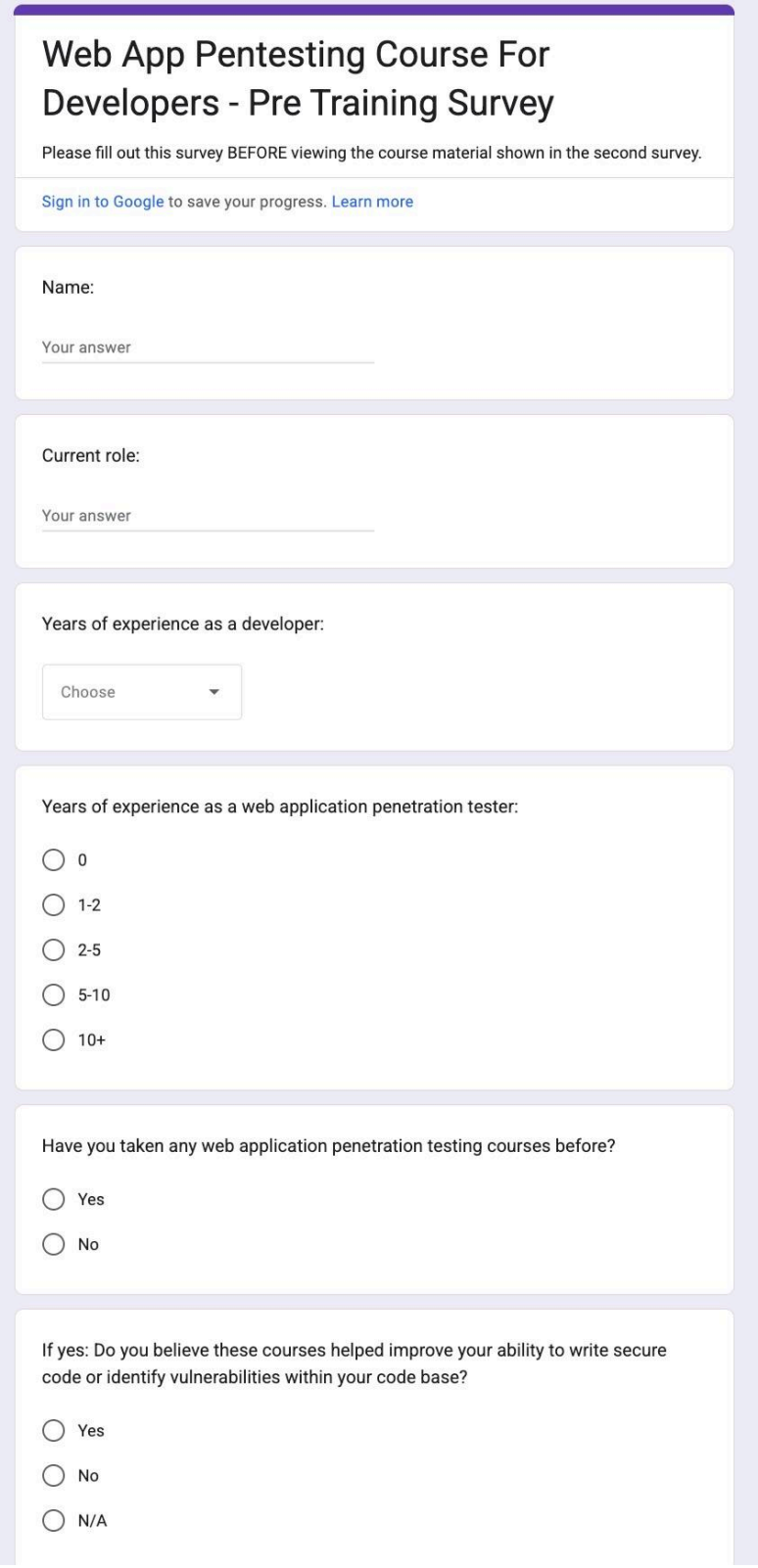
Review the following site for a cheat sheet showing specific requirements and payloads for the most common databases. This will help tailor the following payloads to your environment. For example, if your application uses MySQL, replace the `--` in the commands below with `#`.

https://portswigger.net/web-security/sql-injection/cheat-sheet

```
' followed by: ''

Oracle Concat: foo'||'bar
Microsoft Concat: foo'+'bar
PostgreSQL Concat: foo'||'bar
MySQL Concat: foo' 'bar

' OR 1=1 -- followed by: ' OR 1=2 --
' order by 1 -- followed by: ' order by 100 --
(SELECT 1) followed by: (SELECT 2)

'||pg_sleep(10)--
' OR sleep(5) OR '
'||pg_sleep(5)||'

SLEEP(1) /*' or SLEEP(1) or '" or SLEEP(1) or "*/
```

Appendix A Figure 3: Short Payload List XSS and SQLi Sections

Appendix B: Pre and Post Treatment Surveys



Web App Pentesting Course For Developers - Pre Training Survey

Please fill out this survey BEFORE viewing the course material shown in the second survey.

Sign in to Google to save your progress. Learn more

Name:

Your answer

Current role:

Your answer

Years of experience as a developer:

Choose

Years of experience as a web application penetration tester:

- 0
- 1-2
- 2-5
- 5-10
- 10+

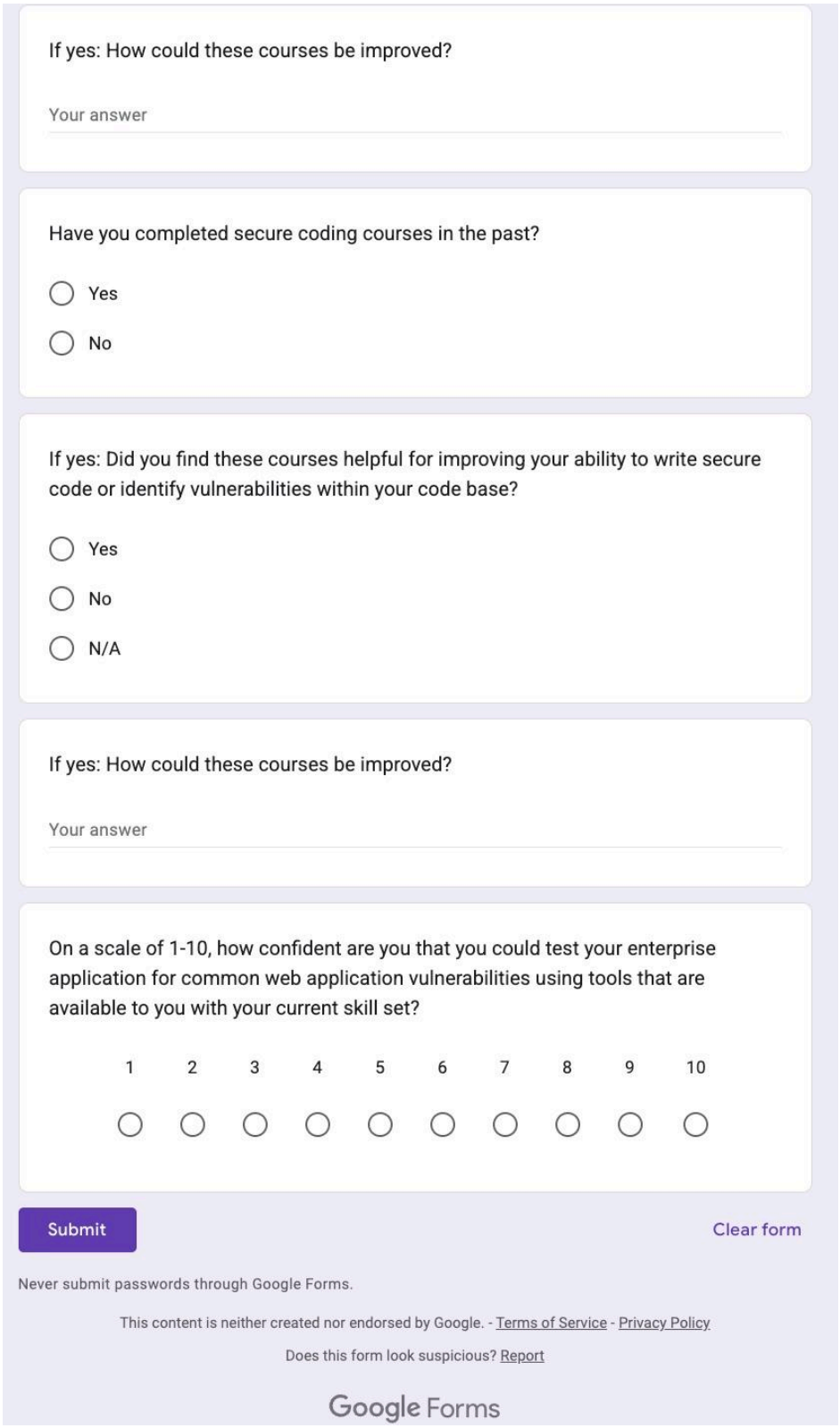
Have you taken any web application penetration testing courses before?

- Yes
- No

If yes: Do you believe these courses helped improve your ability to write secure code or identify vulnerabilities within your code base?

- Yes
- No
- N/A

Appendix B Figure 1: Pre-Training Survey



Appendix B Figure 2: Pre-Training Survey Cont.

Web App Pentesting Course For Developers - Post Training Survey

Please complete the pre training survey first. After, view the material below and fill out this survey.

Video Playlist: https://www.youtube.com/watch?v=oNDEU_uNtzI&list=PLqPCUirqsN_x_seY51DiivfV-CjgJDKUp

Checklist: https://github.com/SchoolProjZS2025/SteeleCapstone2025/blob/main/PracticumProjectChecklist.xlsx

Payload list: https://github.com/SchoolProjZS2025/SteeleCapstone2025/blob/main/ShortPayloadList.md

Sign in to Google to save your progress. Learn more

Name:

Your answer

Years of experience as a developer:

- ◯ 0
- ◯ 1-2
- ◯ 2-5
- ◯ 5-10
- ◯ 10+

Years of experience as a web application penetration tester:

- ◯ 0
- ◯ 1-2
- ◯ 2-5
- ◯ 5-10
- ◯ 10+

Please watch at least two videos in the series before answering the questions below. If possible, follow along using your own application, a test app, or public sites(non-malicious passive checks only).

Please note that the tools video is only showing tools that can be used for testing and the code analysis video does not show any vulnerability testing.

- ◯ Ack

Appendix B Figure 3: Post-Training Survey

Which videos did you watch?

- [ ] Tools
- [ ] Continuous Monitoring
- [ ] Response Headers
- [ ] Cookie Checks
- [ ] Code Review Resources
- [ ] Cross-Site Scripting
- [ ] SQL Injection
- [ ] SQL Injection on Login Pages
- [ ] Directory Traversal
- [ ] OS Command Injection
- [ ] Login Page Checks
- [ ] Session Management
- [ ] Clickjacking
- [ ] File Upload
- [ ] Horizontal Priv Esc
- [ ] Vertical Priv Esc

Did the videos teach you about new vulnerabilities?(Yes/No) Feel free to elaborate.

Your answer

Did the videos improve your understanding of risks and remediation guidance for the vulnerabilities shown?

- ◯ Yes
- ◯ No

Did you use the methodologies shown in the videos to find vulnerabilities in an application? If not, skip down to the next section of questions.

- ◯ Yes
- ◯ No

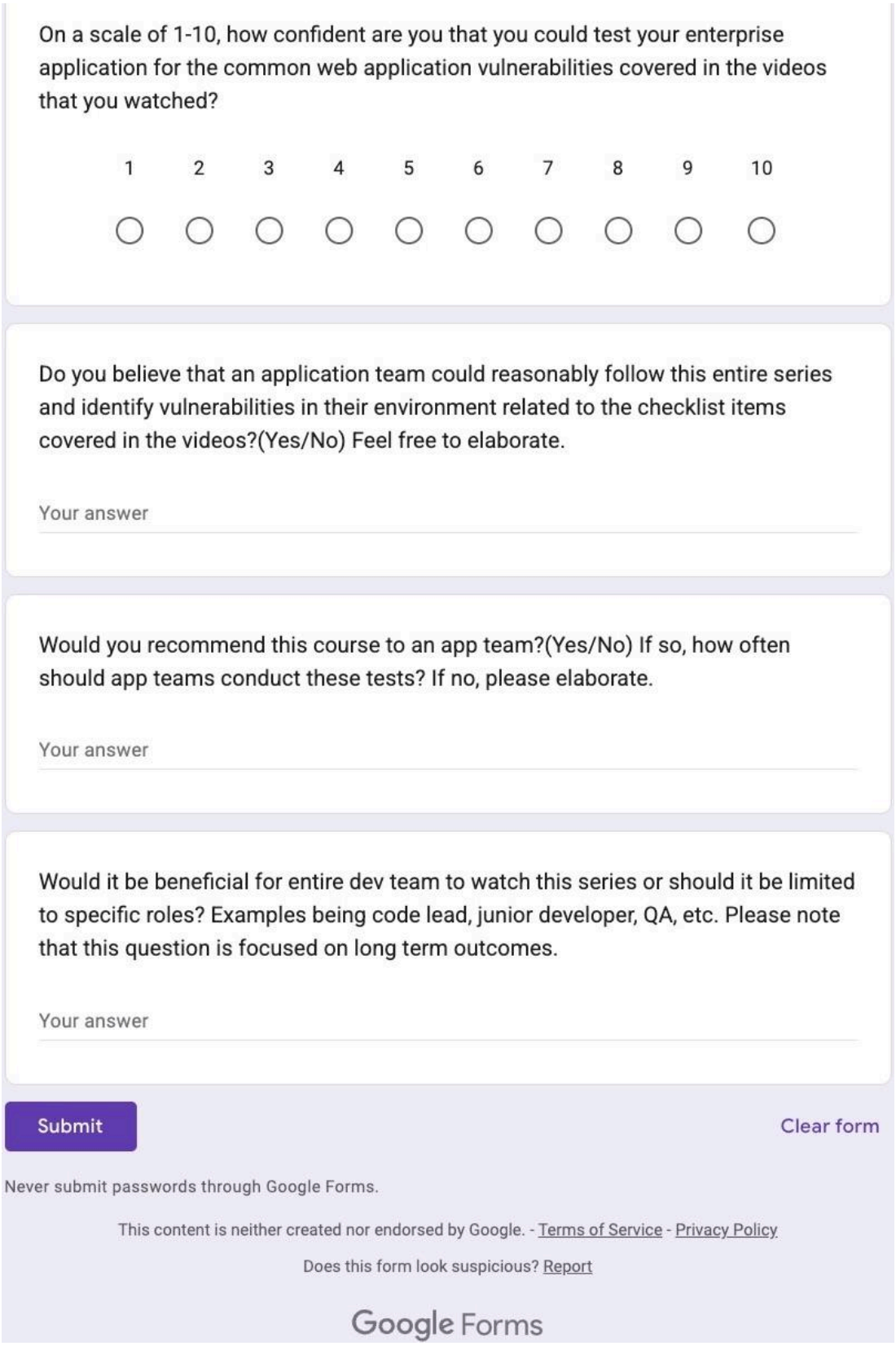
If Yes: How was your experience attempting to follow the videos? Please provide feedback on the pros and cons of the material, explanations, and format.

Your answer

If Yes: Were you able to identify any vulnerabilities or misconfigurations in the application you tested?(Yes/No) If so, what were they?

Your answer

Appendix B Figure 3: Post-Training Survey Cont.



Appendix B Figure 4: Post-Training Survey Cont.

Appendix C: Current Final Paper Progress

Introduction

The technology industry has rapidly grown over the last two decades, creating a massive digital attack surface for threat actors. Companies across the globe have attempted to combat these potential threats by providing their development teams with security training and introducing secure coding methodologies, such as the Secure Development Lifecycle (SDL). Additionally, cyber security departments have continued to introduce new roles and mitigating tools, but a large amount of these controls only serve as a bandaid to protect against bad coding practices and architectural design issues.

Developers and security professionals may feel as though their organizations are continuing to introduce redundant controls and training courses, but it is not fixing the overarching problem. Trend data shows that the number of CVEs found each year continues to rise. (Armstrong et al., 2024) To compound this issue, only 69% of critical vulnerabilities are remediated while an astonishing 48% of lower priority vulnerabilities receive patches. (Lamar, 2025) This tells us that application teams are continuing to see new vulnerabilities while struggling to remediate previously identified issues. Companies should be focused on reducing these vulnerabilities at the source, as data breaches are now averaging almost $5M per incident.(IBM, 2024) Even when companies do not experience a data breach, they could be paying hundreds of thousands of dollars for vulnerable code.(Security Journey, 2024)

I believe that giving developers hands-on experience with identifying web application vulnerabilities in their own applications will reduce the amount of insecure code that gets deployed to production and will help the app teams understand the importance of secure coding practices. The purpose of this project is to research the current penetration testing courses available online to identify potential reasons why developers do not take these courses more often. Then, create a solution that fills the gap in current training courses. Specifically, by creating a new penetration testing course designed for developers.

While this course is not going to be explicitly teaching secure coding practices, it will be teaching application teams how to identify common vulnerabilities in their code and the risks associated with these vulnerabilities. This will also assist with the team's understanding of the “why” component as they continue researching how to thwart attacks and write secure code. While this course is not a complete replacement for a true penetration test, it will undoubtedly help uncover vulnerabilities for the 68% of companies that do not do annual or bi-annual penetration tests. (Palatty, 2025) Additionally, we will be providing language specific remediation advice in some cases, but the course will mostly focus on language agnostic remediation advice because studies show that 79% of developers preferred the idea of a language agnostic secure coding course. (Robinson & Russo, 2024, p.20)

Problem Statement Please note this section is identical to the problem statement in the top of this progress report.

There is a large difference in priorities and skillset between penetration testers and code developers, leading to a troubling amount of insecure web applications and difficulty communicating vulnerabilities to application teams. While developers are constantly told to follow secure coding practices, oftentimes even utilizing security focused development frameworks, they rarely prioritize secure coding practices. (Secure Code Warrior, 2022) Perry et al. (2023) prove that insecure code is going to become even more common as AI code assistants become more popular because developers are over confident in its ability

to write secure code. This problem will continue to grow if developers do not start taking cyber security and secure coding practices more seriously. In one study, 68% of participants stated that they only did secure code training because of a compliance need or recent exploit. (Security Journey, 2024)

Personally, I believe that developers do not take more cyber security training programs largely because of the gap in cyber security courses. These courses and programs fall into two categories based on my experiences as a penetration tester responsible for teaching application teams how to test and remediate vulnerabilities. First, developer focused resources are too high level to be helpful. These courses may teach the basics of specific misconfigurations or exploits, but they do not provide many opportunities for developers to walk away with knowledge that can be applied to their current applications. In one study, developers stated that training is not hands on, and/or it is taught in a vacuum. (Secure Code Warrior, 2022)

The second category of resources are targeted towards penetration testers, meaning the course is extremely in depth, requires specialized tools, and a massive time investment from the audience. As an example, PortSwigger's Cross-Site Scripting section includes 30 labs with each solution video ranging from 50 seconds to over 20 minutes. If a developer wanted to learn the risks, remediation, and how to exploit XSS at a basic level, they would have to parse through multiple pages of documentation and choose between 50 second videos with zero context/explanations or 20+ minute videos explaining testing methodologies and edge cases per lab. Lastly, these walkthrough videos do not use tools that are available to developers in the average corporate environment. (PortSwigger, 2025)

Most developers are limited to tools focused on the development lifecycle due to the risks associated with scanning your application. (Young, 2025) We can see evidence of this in the Fortra (2024) survey, stating that 24% of cyber security professionals don't use penetration testing tools and an additional 33% only use open source tools. This makes sense as cost was the second most important factor when considering penetration testing software. While this survey was aimed at cyber security professionals, we can assume that developers would have the same or lower buy-in to cyber security tools. An additional issue with utilizing new tools is the time commitment. Velez (2020) reports on the many struggles that junior developers face when attempting to install programming tools. This issue compounds the current time limitations that developers deal with in modern frameworks. Rajapakse et al. (2021) supports this point by proving that devops workflows do not allow for the time commitment of SAST and DAST tools.

From my own experience researching this topic, I often find the second type of courses do not cover a wide range of topics. Generally, these courses are aimed at teaching exploitation, but do not cover policy based findings or misconfigurations, such as response header infractions. Please see the figure below showing a summary of my own opinions and analysis after completing other courses that are publicly available. Please note that this is a high level view of these courses and is not intended to be an in-depth review of each course. The purpose is not to disparage other courses, but to instead point out gaps or audience/tool differences that I hope to fill with my solution.

Course Name	Length	Tools Utilized	Pros	Cons
DIY Web App Pentesting Guide	Medium Article	Kali, Recon-ng, whatweb, nmap, Nikto, SQLMap	Showed how to catch low hanging fruit with scanners	Most developers cannot openly scan their applications or install tools like nmap. Only checked a few items in the application.
OWASP Web Security Testing Guide	100+ Pages	30+ tools and scripts used throughout the site.	The most comprehensive penetration guide online.	Information overload for the average developer. Does not show application behavior outside of full compromise.
Web App Testing	11 Hours	Burp	Covers each vulnerability in great depth.	Does not cover a wide range of vulnerabilities. Did not explain risks and remediation outside of test cases.
Ethical Hacking 101	2.75 Hours	Burp, Zap.	Used multiple test applications to show the risks of each vulnerability	Did not explain application behavior in partial exploitation situations or remediation.
Penetration Testing Course for Beginners	5 Hours	Burp, SQLMap, Nikto	Showed real world testing for vulnerabilities.	Only showed one or two examples per vulnerability covered, leaving the assessors to their own devices for further payload development. Did not cover full scope of risks associated with each vulnerability.
PortSwigger	270+ Labs	Burp, SQLMap, python scripts.	Smaller scope than the OWASP training, but with hands on training and better explanations. The number one resource for web application penetration testers to learn beginner to advanced exploitation.	Targeted towards web application penetration testers, meaning the material gets very detailed and covers many edge cases.
How to Analyze Code for Vulnerabilities	1.3 Hours	IDE	Podcast format, allowing for digestable explanations and discussions.	Almost entirely high level powerpoint bullets without many additional resources for devs to review.
Secure Coding – Best Practices (also for non developers!)	57 Minutes	Provides tool examples, but not required.	Covers a healthy range of topics with short exploit examples and blanket mock code snippets. Provides further training resources at the end.	Lacks context and true examples of application code, resulting in infromation that is hard to convert to our current applications.
Web Application Penetration Testing Tutorial	4.5 Hours	Burp, Fiddler, nmap, NetSparker	Shows a wide range of misconfigurations and high priority security vulnerabilities. Even opens the video with what the coures lacks(SQLi and Session Hijacking). The closest course to what I am trying to acheive with this project.	Does not target dev teams, meaning the course does not encourage a checklist and frequent test of your own applications. Additionally, missing a handful of misconfigurations that are extremely common in modern web apps.
Programming Foundations: Secure Coding	2 Hours	None	Great high level overview of secure practices throughout the entire lifecycle of a project.	Requires monthly membership. Entirely explained using animated content, does not show a single code snippet or exploit example.
Developing Secure Software	1.5 Hours	None	High level overview of many concepts ranging from coding practices to security testing. Easily digestable powerpoint format.	Almost entirely powerpoint bullet points, no real information for an app team to action on without doing their own third party research on each topic.

Figure 1: Course Comparison (Steele, 2025)

I believe this pushes away most developers that were willing to commit a small amount of time to upskilling their cyber security knowledge. According to one study, 44% of respondents stated the reason for not taking secure coding courses is because they were not aware of a good course and another 44% blamed time constraints. (Robinson & Russo, 2024) It is important to note that the solution described below may not solve the issues covered in this 88% of respondents because time constraints and course quality may be subjective. Ultimately, I believe that this gap in training material is the reason we continue to see an increase in vulnerabilities every year, even though most developers receive annual penetration tests on their applications and secure coding training. (Tribbey & Winterfeld, 2023) If application teams were capable of testing their environments and implemented security checks on a regular basis, we would most likely see a decrease in the 31% of critical vulnerabilities and 52% of lower priority vulnerabilities that go unremediated. (Lamar, 2025)

My Solution

This project will teach developers to test their own applications for vulnerabilities via a training course dedicated to quickly learning penetration testing tactics for common vulnerabilities in order to increase the security of modern web applications.Achieving this goal requires creating a penetration testing course that fills the gap between in-depth hacking courses and high level overviews of web application vulnerabilities. Unlike most available resources, we will target an audience of developers who are looking to give their application a light penetration test and configuration sanity check. The goal of this solution is not to make developers top level penetration testers or experts in secure coding practices. The goal is to

create a course that developers will actually utilize to reduce vulnerabilities within their own applications. However, this course will also provide third party resources in the event that the viewer wants to dive further into specific topics.

Specifically, this project includes a checklist with the most common attack types and misconfigurations, payloads for the injection attacks, and a video series showing developers how to utilize the material in their own environments with multiple different tool alternatives. The course material works together to provide the developers with enough information to quickly jump to different vulnerability types, then learn each testing methodology as needed. Additionally, we provide a page dedicated to resources for secure coding practices with the goal of giving the application teams a starting point for building out their own requirements based on the context and language of their application. The figure below shows the regex filtering portion of this page. As shown, we point the application in the right direction, but expect them to provide their development teams with guidance that better fits their needs and company policies. As previously stated, this course was not made to teach secure coding practices, but we included this segment because it is important that an application team fixes their vulnerabilities at the root of the problem. Additionally, this material can benefit the application teams even if they do not continue into the penetration testing segment. Ideally, the code leads would agree on the guidance in this section and require all developers familiarize themselves with the requirements.

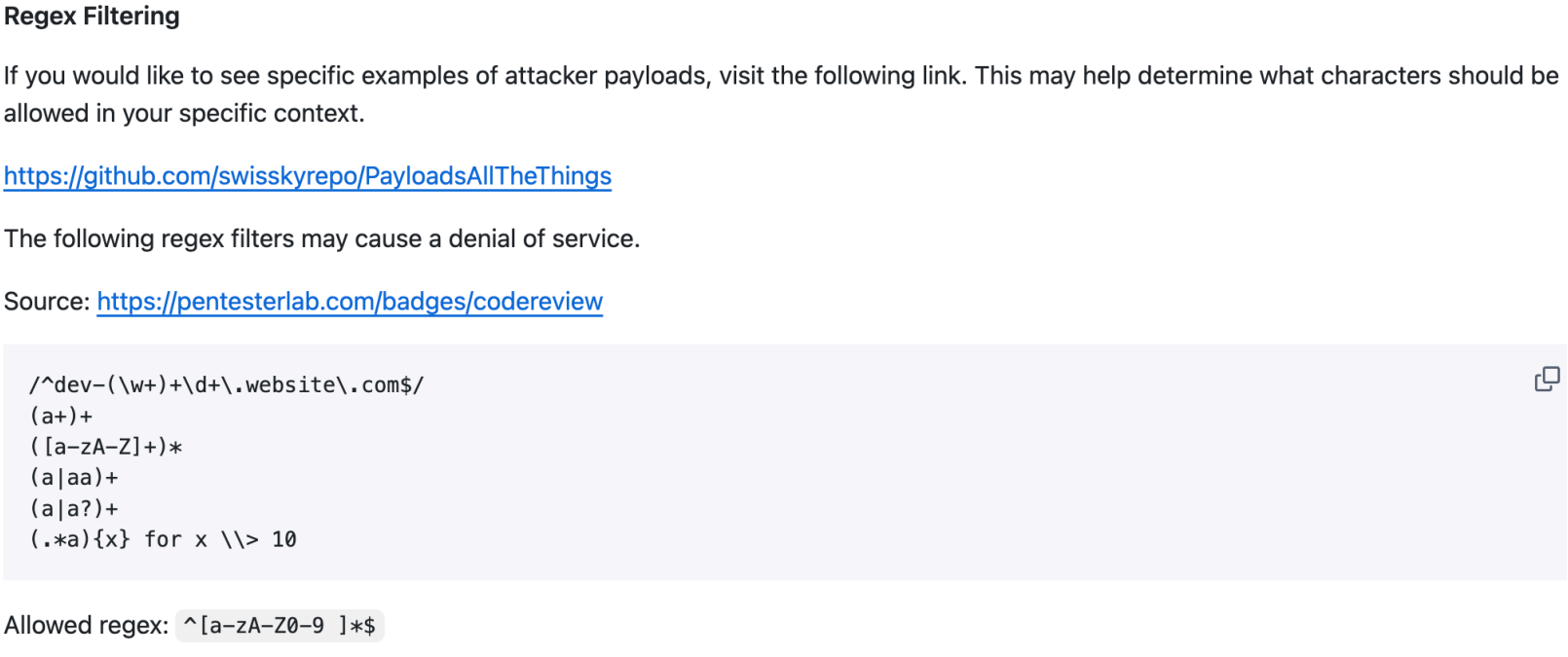


Figure 2: Regex filtering section of secure coding practices page

To make this course as effective as possible, we identified the best set of vulnerabilities to cover in a reasonable amount of time. The material for our course was selected by evaluating a combination of time to conduct the test, difficulty of testing, impact, and how common the vulnerability is found in modern applications. It would not be a great use of the developer's time to learn a difficult attack type that is rarely found in applications, especially if this vulnerability does not carry severe risks. The items covered in this course fall into a few different categories, creating a checklist that has great coverage and, in some

cases, allows the developer to learn one testing technique that can be applied across multiple items.

First, the checklist was populated with vulnerabilities and misconfigurations that are easy to find, requiring a small amount of time to test, regardless of likelihood and impact. This includes reviewing cookie attributes, response headers, and looking out for incorrect application behavior throughout the testing window. This was the first set of test cases to enter the scope of this project because these simple checks can heavily reduce the impact of high priority findings. As an example, a Content-Security Policy(CSP) can be quickly reviewed manually, or using online tools, to determine if the CSP is complete and safe. This powerful response header is responsible for preventing cross-site scripting(XSS), clickjacking, and other framing attacks. However, according to Pacheco (2020), it is only implemented correctly on 2% of the top 1000 most visited sites. Reviewing this checklist item will take far less time than attempting cross-site scripting across the application, and it can heavily reduce the risk of XSS in some cases. The checklist also includes lower impact findings, such as information leak response headers because the CSP and information leak response headers can be checked at the same time.

	Pass	Rating	Description	Resources
				We encourage app teams to update the checklist
Continuous Review				
Error Messages are Properly Handled		Low	Error messages should be short and generic. Do not show stack traces or informative messages to the user.	https://cheatsheetseries.owasp.org/cheatshe
Autocomplete Disabled on Sensitive Input Fields		Low	Disable autocomplete on fields such as email, password, and security questions by setting autocomplete="off"	https://www.w3schools.com/howto/howto_ht
Sensitive Information is Masked		Medium	Sensitive Information should be masked to mitigate shoulder surfing attacks.	https://labex.io/tutorials/javascript-mask-a-va
Sensitive Information Not in URL		Medium	Do not send sensitive information in GET request parameters.	https://cwe.mitre.org/data/definitions/598.htm
Unneccessary Methods are Disabled		Low	Do not allow unsafe methods such as Trace or Connect on any endpoint. Disable DELETE, POST, etc. if not needed on the endpoint.	https://cyberwhite.co.uk/http-verbs-and-their-
HTTP Disabled		High	Enforce HTTPS for all connections.	https://blog.matrixpost.net/redirect-from-http-
Response Headers				
Content-Security-Policy		Medium	Does not include unsafe directives, wildcarded directives, or wildcards in subdomains.	https://developer.mozilla.org/en-US/docs/We https://csp-evaluator.withgoogle.com/
HSTS Response Header		Low	Use Strict-Transport-Security: max-age=63072000; includeSubDomains; preload. Only use preload if required.	https://developer.mozilla.org/en-US/docs/We
Caching Headers		Low	Use Cache-Control: no-store. Less strict controls: Expires: 0, Cache-Control: no-cache	https://developer.mozilla.org/en-US/docs/We https://www.rfc-editor.org/rfc/rfc9111.html
Information Leak Headers		Informational	Remove unneccessary response headers, such as: Server, X-Powered-By, X-AspNet-Version.	https://support.waters.com/KB_Inf/Other/WK
CORS Headers		Low	Do not set Access-Control-Allow-Origin to wildcard or allow reflected arbitrary origins/subdomains.	https://www.freecodecamp.org/news/exploitir
Cookies				
HttpOnly		Low	Prevents JavaScript from accessing the cookies, mitigating some XSS risks.	https://developer.mozilla.org/en-US/docs/We
Secure Flag		Low	Ensures cookies are only sent over HTTPS.	https://developer.mozilla.org/en-US/docs/We
SameSite Lax or Strict		Informational	SameSite cookies control cross-site request behavior to prevent CSRF. Lax: Cross-site requests will only send cookies in GET requests. Strict: Cookies will not send in any cross-site requests.	https://developer.mozilla.org/en-US/docs/We https://portswigger.net/web-security/csrf/bypa
Narrow Path		Informational	Sensitive/session cookies must have a restrictive path set to prevent other applications on the same server from using this cookie.	https://owasp.org/www-project-web-security-
Narrow Domain		Informational	Sensitive/session cookies should have a narrow domain, set to a specific subdomain when applicable.	https://developer.mozilla.org/en-US/docs/We

Figure 3: Quick to verify section of the checklist

After implementing checks for the previously mentioned items, our next step was to include vulnerabilities that require payloads inserted by an attacker. These vulnerabilities are largely based on common technology stacks and functionality within modern applications. As an example, relational database models made up 4 of the top 5 database models in May 2025.(DB-Engines, 2025) Since database injections are time consuming and difficult to fully exploit, we chose to only include SQL injections for this course, excluding noSQL injection attacks. The other most common finding in this section of the checklist is XSS, making up 19% of Synopsys's high risk vulnerabilities from 2022 assessments. (Freeman, 2023)

Finally, the checklist was populated with common vulnerabilities and misconfigurations that do not require malicious payloads. The goal was to test any items that were not covered in the previous two

sections. We grouped checklist items together based on location in the application or attack vector commonalities. As an example, when reviewing the login page, we wanted to cover Multi-Factor authentication (MFA) as only 34% of medium sized firms have implemented MFA. (Worthington & Ozsahan, 2025) MFA implementation may be difficult depending on the architecture of the application, but it is a simple check that could prevent attackers from completely compromising a user's account. Another test case covered in this section is account lockout after X number of bad attempts. Developers can quickly test their applications account lockout controls without any tools. According to ?? survey, 35% of identity-related incidents were related to brute force attacks. (Longstreet, 2024) While account lockout is not a 100% solution to this issue, it will heavily reduce the chances of an attacker brute forcing a user's credentials. Using a similar testing methodology, we also check for user enumeration, a vulnerability found in over 93% of applications in one survey. (Maceiras et al., 2024) A combination of these 3 tests can severely reduce the likelihood of account takeover.

After establishing a set of vulnerabilities and misconfigurations to test, it was critical to build out the course in a way that would meet our goal of filling the gap between current solutions. To do this, we had to prioritize speed and conversational teaching over complete coverage and strict formatting. As an example, when covering response headers, it made more sense to explain testing methodology for all response headers up front since they would be tested the same way. Next, each item was tested individually by explaining risk, showing real examples, then providing remediation guidance. For comparison, Portswigger (2025) and HackTheBox (HTB Academy, n.d.) both choose to teach response header misconfigurations and remediation during the modules related to their respective exploits. The purpose of their structure is to teach the entire context and multiple edge cases related to an exploitable vulnerability. For example, CSPs are covered after teaching the basics of XSS in the HackTheBox course to show more advanced situations once the basics have been covered. However, my structure is easier to follow without having the underlying context that a penetration tester would have. If an application team tells a developer to test and fix their CSP, the developer would have to know that the CSP is most likely covered in the XSS section of HackTheBox because a CSP is most commonly used to thwart XSS attacks. With my course, the developer could jump to the exact moment we teach CSPs by looking at the checklist. Additionally, this same video introduces the developer to the other response headers that could be reviewed at the same time as the CSP. Lastly, instead of showing multiple examples of misconfigured CSP exploitation, we instead focus on speed by providing third party resources to assist with testing and remediating the issue if the app team wants to dig in further. Again, this course is aimed at developers trying to upskill and test their applications without much overhead.

As shown in figure?? below, we exchanged strict formatting and wide edge case coverage for a list of payloads that work in the most common situations and a few important comments to guide application teams along the way. This looser formatting was designed for a less technical audience with the additional goal of encouraging edits from the application team. While PayloadAllTheThings (Swisskyrepo, n.d.) is the most comprehensive payload list publically available, it would quickly overwhelm an application team because it includes many edge case payloads, oftentimes in an order that does not mirror success rate.

Efficacy/evaluation

The solution section of this paper covers more details about the likelihood and risks associated with the

vulnerabilities and misconfigurations covered in the course. We believe the combination of selected items would result in the highest reduction of risk and most coverage possible given the limited time frame of the practicum course and avoiding resource overload for the developers taking the course.

With that said, this course is only useful if audience retention is high and viewers take action in their own environments. Since we did not have the time or resources to survey the long term implications of this course, we decided to survey professional developers, penetration testers, and other technology focused professionals.
-This section needs a ton of work. Bring down the evaluation section from the progress report and make sure to cover limitations with this approach in depth, like participant bias. Make sure to explain that we sent this survey to a lot of people who do not know me.

Limitations
The biggest limitation with this course is the overall coverage of the material. Since the goal of this project was to fill the gap in training material, we chose accessibility over coverage. The idea behind this project was that if an application team implemented our checklist and light testing, they would be able to identify and remediate vulnerabilities without the time sink required to complete a more comprehensive course, such as PortSwigger (2025). We are very clear upfront that this course is not a comprehensive penetration test. To completely ensure adequate coverage, an application team must get a full web application penetration test from a professional pentester. On the other hand, if the company wanted their own in-house security team, they would have to utilize one of the previously mentioned resources because this course does not teach a comprehensive testing methodology for a wide range of situations or edge cases that rarely result in identifying a vulnerability. As an example, the cross-site scripting(XSS) section of our course did not cover CSP bypass or unicode character substitution. First, we did not cover CSP bypass because potential XSS can be identified without getting full exploitation. Second, using unicode characters to bypass filters has an extremely low success rate. If we spent the time to cover similar situations across the entire list of checklist items, we would have doubled or tripled the overall length of the course.

Additionally, this course does not cover the full set of topics that would be covered by an in-depth course and it does not have the complete list of checklist items that would be found on a true penetration test. These additional checklist items were not included for two reasons. First, we did not want to scare developers away by making the course too long. Second, some test cases require a full understanding of the application, vulnerability, and test case due to the “out of the box” thinking that comes with some situations. As an example, during a true penetration test, the tester is responsible for identifying potential business logic bypasses and attempting to exploit those situations using any relevant attacks. While these attacks are fairly common, it is very difficult to teach someone how to adequately test for these vulnerabilities because each situation has its own unique variables, making blanket advice difficult. Third, some of the vulnerabilities and test cases left off the checklist require uncommon technology stacks, or have a low success rate. In my opinion, that did not warrant the required time sink from the development team. One item in this category is web socket vulnerabilities. In this ?? study by ??, they found that WebSockets are only used in 7.3% of the top 1000 sites. (Murley et al., 2021) Their research reveals 74.4% of WebSockets do not verify the Origin header and 14.1% are accessible over unencrypted channels. This means out of 1000 websites, less than 55 are not validating the Origin header and less than

11 are accessible over unencrypted channels. Unfortunately, this research does not discuss the breakdown of WebSocket functionality, meaning we cannot estimate the number of applications that would see true impact from these misconfigurations. However, these statistics do prove that WebSocket usage is too uncommon for us to include it in our testing. We do encourage application teams to read the resources provided in the code analysis GitHub page because it could lead to additional test cases, like WebSockets, that the application team may have been missing out on.

Future Work

Organize the material in a better fashion, create flow charts for payloads, similar to the SSTI payload flow chart. Include other less common vulnerabilities that may only apply to teams in certain contexts. Find a way to get more app teams to implement this into their testing.

Conclusion

Empty.

References(final paper references)

Armstrong, K., Williams, K., Landfield, K., & Lawler, S. (2024, October). *25th Anniversary Report*. CVE. https://www.cve.org/Resources/Media/Cve25YearsAnniversaryReport.pdf

Lamar, J. (2025, April 14). *Key takeaways from the State of Pentesting Report 2025*. Cobalt. https://www.cobalt.io/blog/key-takeaways-state-of-pentesting-report-2025

Cost of a data breach 2024. IBM. (2024, July). https://www.ibm.com/reports/data-breach

How secure coding training can save your budget. Security Journey. (2024, May 1). https://www.securityjourney.com/post/how-secure-coding-training-can-save-your-budget

Palatty, N. J. (2025, February 5). *83 penetration testing statistics: Key facts and figures*. Astra Security. https://www.getastra.com/blog/security-audit/penetration-testing-statistics/

Robinson, C., & Russo, D. (2024, June). Secure Software Development Education 2024 Survey. https://www.linuxfoundation.org/hubfs/LF%20Research/Secure_Software_Development_Education_2024_Survey.pdf?hsLang=en

Tribbey, B., & Winterfeld, S. (2023, April 18). *Slipping through the security gaps: The rise of application and API attacks* | Akamai. Akamai. https://www.akamai.com/blog/security-research/the-rise-of-application-and-api-attacks

What is cross-site scripting (XSS) and how to prevent it?: Web security academy. What is cross-site scripting (XSS) and how to prevent it? | Web Security Academy. (n.d.). https://portswigger.net/web-security/cross-site-scripting

Young, L. (2025, May 19). *Penetration testing risks: Understanding the potential dangers of vulnerability assessments*. VaultMatrix.com.

https://www.vaultmatrix.com/penetration-testing-risks-understanding-the-potential-dangers-of-vulnerability-assessments/

2024 penetration testing report. Fortra. (2024). https://static.fortra.com/core-security/pdfs/guides/fta-cs-2024-pen-testing-report-gd.pdf

Velez, M., Yen, M., Le, M., Su, Z., & Alipour, M. A. (2020, March 14). *Student Adoption and Perceptions of a Web Integrated Development Environment*. ACM Digital Library. https://dl.acm.org/doi/pdf/10.1145/3328778.3366817

Rajapakse, R. N., Zahedi, M., & Babar, M. A. (2021, July 19). *An Empirical Analysis of Practitioners' Perspectives on Security Tool Integration into DevOps*. arxiv - Cornell University. https://arxiv.org/pdf/2107.02096

Secure code warrior survey finds 86% of developers do not view application security as a top priority. Secure Code Warrior. (Apr 05, 2022).

Pacheco, P. (2020, September 3). *Content security policy limits dangerous activity... so why isn't everyone doing it?* Bitsight. https://www.bitsight.com/blog/content-security-policy-limits-dangerous-activity-so-why-isnt-everyone-doing-it

DB-Engines ranking. DB-Engines. (2025, June). https://db-engines.com/en/ranking

Freeman, C. (2023, November 14). *Understanding XSS: Why cross-site scripting still matters: Black duck blog*. Understanding XSS: Why Cross-Site Scripting Still Matters | Black Duck Blog. https://www.blackduck.com/blog/why-cross-site-scripting-still-matters.html

Worthington, D., & Ozsahan, H. (2025, January 3). *2025 multi-factor authentication (MFA) Statistics & Trends to know*. JumpCloud. https://jumpcloud.com/blog/multi-factor-authentication-statistics

Longstreet, A. (2024, June 21). *The State of Identity Security for 2024: Identity-Based Threats, Breaches, & Security Best Practices*. BeyondTrust. https://www.beyondtrust.com/blog/entry/the-state-of-identity-security-identity-based-threats-breaches-security-best-practices

Maceiras, M., Niksirat, K. S., Bernard, G., Garbinato, B., Cherubini, M., Humbert, M., & Huguenin, K. (2024, June 17). *Know their customers: An empirical study of online account enumeration attacks*. ACM Digital Library. https://dl.acm.org/doi/10.1145/3664201

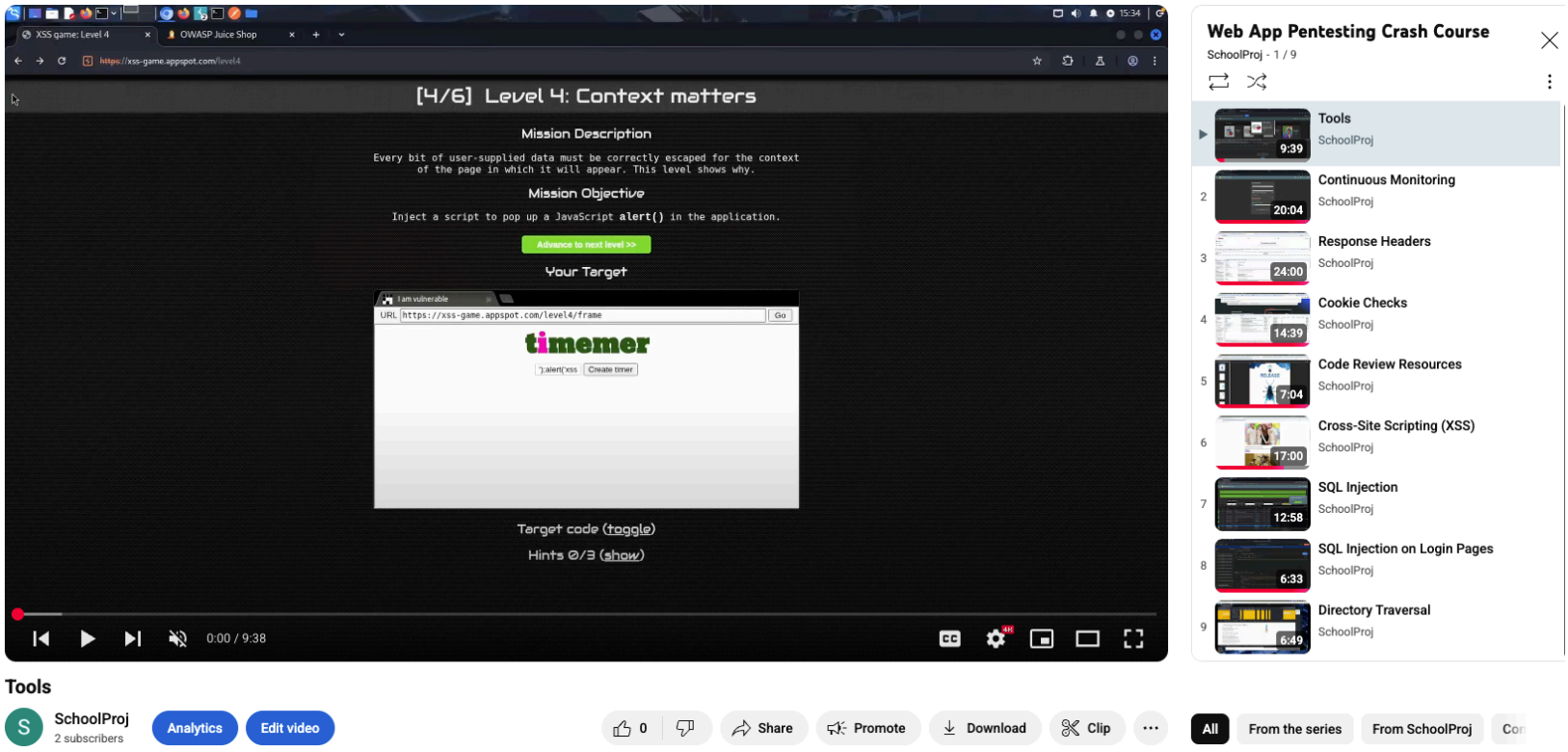
HTB Academy. (n.d.). *Senior web penetration tester job role path: HTB academy*. HTB Academy. https://academy.hackthebox.com/path/preview/senior-web-penetration-tester

Swisskyrepo. (n.d.). *Swisskyrepo/payloadsallthethings: A list of useful payloads and bypass for web application security and Pentest/CTF*. GitHub. https://github.com/swisskyrepo/PayloadsAllTheThings

Murley, P., Ma, Z., & Mason, J. (2021, April 23). *WebSocket Adoption and the Landscape of the Real-Time Web*. Georgia Tech. https://faculty.cc.gatech.edu/~mbailey/publications/www21_websocket.pdf

Appendix(this is the final paper appendix)

Appendix A: Web Application Penetration Testing Course for Developers (Steele, 2025)



Appendix A Figure 1: Course Playlist

	Pass	Rating	Description	Resources
				We encourage app teams to update the checklist and Payload lists with language specific links and company policies to ensure future developers follow the companies requirements.
Continuous Review				
Error Messages are Properly Handled		Low	Error messages should be short and generic. Do not show stack traces or informative messages to the user.	https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html
Autocomplete Disabled on Sensitive Input Fields		Low	Disable autocomplete on fields such as email, password, and security questions by setting autocomplete="off"	https://www.w3schools.com/howto/howto_html_autocomplete_off.asp
Sensitive Information is Masked		Medium	Sensitive Information should be masked to mitigate shoulder surfing attacks.	https://labex.io/tutorials/javascript-mask-a-value-28489
Sensitive Information Not in URL		Medium	Do not send sensitive information in GET request parameters.	https://cwe.mitre.org/data/definitions/598.html
Unneccessary Methods are Disabled		Low	Do not allow unsafe methods such as Trace or Connect on any endpoint. Disable DELETE, POST, etc. if not needed on the endpoint.	https://cyberwhite.co.uk/http-verbs-and-their-security-risks/
HTTP Disabled		High	Enforce HTTPS for all connections.	https://blog.matrixpost.net/redirect-from-http-to-https-using-the-iis-url-rewrite-module/
Response Headers				
Content-Security-Policy		Medium	Does not include unsafe directives, wildcarded directives, or wildcards in subdomains.	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Content-Security-Policy https://csp-evaluator.withgoogle.com/
HSTS Response Header		Low	Use Strict-Transport-Security: max-age=63072000; includeSubDomains; preload. Only use preload if required.	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Strict-Transport-Security
Caching Headers		Low	Use Cache-Control: no-store. Less strict controls: Expires: 0, Cache-Control: no-cache	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Cache-Control https://www.rfc-editor.org/rfc/rfc9111.html
Information Leak Headers		Informational	Remove unneccessary response headers, such as: Server, X-Powered-By, X-AspNet-Version.	https://support.waters.com/KB_Inf/Other/WKB202501_How_to_disable_the_Server_HTTP_header_in_Microsoft_IIS
CORS Headers		Low	Do not set Access-Control-Allow-Origin to wildcard or allow reflected arbitrary origins/subdomains.	https://www.freecodecamp.org/news/exploiting-cors-guide-to-pentesting/#heading-exploitable-cors-cases
Cookies				
HttpOnly		Low	Prevents JavaScript from accessing the cookies, mitigating some XSS risks.	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Set-Cookie
Secure Flag		Low	Ensures cookies are only sent over HTTPS.	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Set-Cookie
SameSite Lax or Strict		Informational	SameSite cookies control cross-site request behavior to prevent CSRF. Lax: Cross-site requests will only send cookies in GET requests. Strict: Cookies will not send in any cross-site requests.	https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Set-Cookie https://portswigger.net/web-security/csrf/bypassing-samesite-restrictions
Narrow Path		Informational	Sensitive/session cookies must have a restrictive path set to prevent other applications on the same server from using this cookie.	https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes
Narrow Domain		Informational	Sensitive/session cookies should have a narrow domain, set to a specific subdomain when applicable.	https://developer.mozilla.org/en-US/docs/Web/Security/Practical_implementation_guides/Cookies
Injections/Payloads Required				
Cross-Site Scripting(XSS)		Critical	The application should encode user input when reflected to the UI.	https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html https://learn.microsoft.com/en-us/aspnet/core/security/cross-site-scripting?view=aspnetcore-9.0
SQL Injection		Critical	Use parameterized queries to prevent SQL injection via user input.	https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html https://cheatsheetseries.owasp.org/cheatsheets/Query_Parameterization_Cheat_Sheet.html
SQL Injection on Login Page		Critical	Login pages must use parameterized queries.	https://unix.stackexchange.com/questions/391866/regex-for-password-restricting-special-characters
Directory Traversal		High	User Input must be validated and canonicalized before being used to retrieve files from the server.	https://portswigger.net/web-security/file-path-traversal
OS Command Injection		Critical	Do not place user input directly into system command executions.	https://developers.redhat.com/articles/2023/03/29/4-essentials-prevent-os-command-injection-attacks#4_ways_to_prevent_os_command_injection_attacks
Login Page				
Weak Passwords Not Allowed		Medium	Enforce strong password policies.	https://www.cisa.gov/secure-our-world/use-strong-passwords
User Enumeration Not Possible		Medium	Ensure the application does not reflect different messages for valid or invalid users.	https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/03-Identity_Management_Testing/04-Testing_for_Account_Enumeration_and_Guessable_User_Account
Account Lockout Enabled		Medium	Lock user accounts after X number of failed login attempts to prevent brute-force attacks.	https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks
Multi-Factor Authentication in Use		Low	Require multi-factor authentication at login.	https://auth0.com/blog/different-ways-to-implement-multifactor/

Appendix A Figure 2: Course Checklist

Cross-Site Scripting (XSS):

https://portswigger.net/web-security/cross-site-scripting

Enter the following payloads into user input fields and see how they are reflected in the application. If user input is reflected back unencoded and unfiltered, especially in javascript, then XSS is likely.

Alert() can be replaced with print() if your application is filtering out alert().

```
<b>test</b>
<script>alert(1)</script>

<img src=1 onerror=alert(1)>

javascript:alert(document.cookie)

"onload="alert(1)

${alert(1)}

alert`1`

'-alert()-'
```

SQL Injection

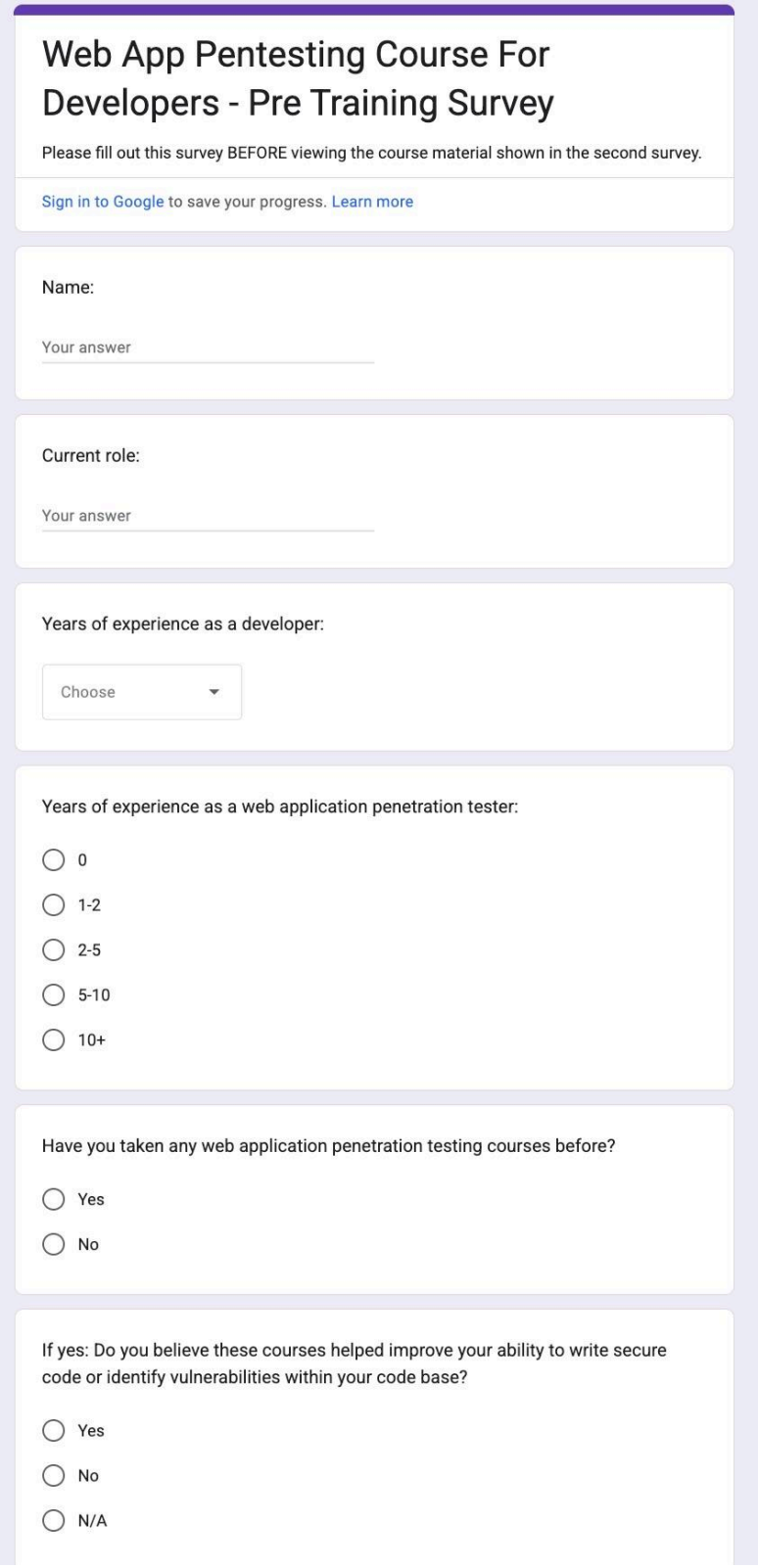
Review the following site for a cheat sheet showing specific requirements and payloads for the most common databases. This will help tailor the following payloads to your environment. For example, if your application uses MySQL, replace the `--` in the commands below with `#`.

https://portswigger.net/web-security/sql-injection/cheat-sheet

```
' followed by: ''

Oracle Concat: foo'||'bar
Microsoft Concat: foo'+'bar
PostgreSQL Concat: foo'||'bar
MySQL Concat: foo' 'bar

' OR 1=1 -- followed by: ' OR 1=2 --
' order by 1 -- followed by: ' order by 100 --
(SELECT 1) followed by: (SELECT 2)

'||pg_sleep(10)--
' OR sleep(5) OR '
'||pg_sleep(5)||'

SLEEP(1) /*' or SLEEP(1) or '" or SLEEP(1) or "*/
```

Appendix A Figure 3: Short Payload List XSS and SQLi Sections

Appendix B: Pre and Post Treatment Surveys



Web App Pentesting Course For Developers - Pre Training Survey

Please fill out this survey BEFORE viewing the course material shown in the second survey.

Sign in to Google to save your progress. Learn more

Name:

Your answer

Current role:

Your answer

Years of experience as a developer:

Choose

Years of experience as a web application penetration tester:

- 0
- 1-2
- 2-5
- 5-10
- 10+

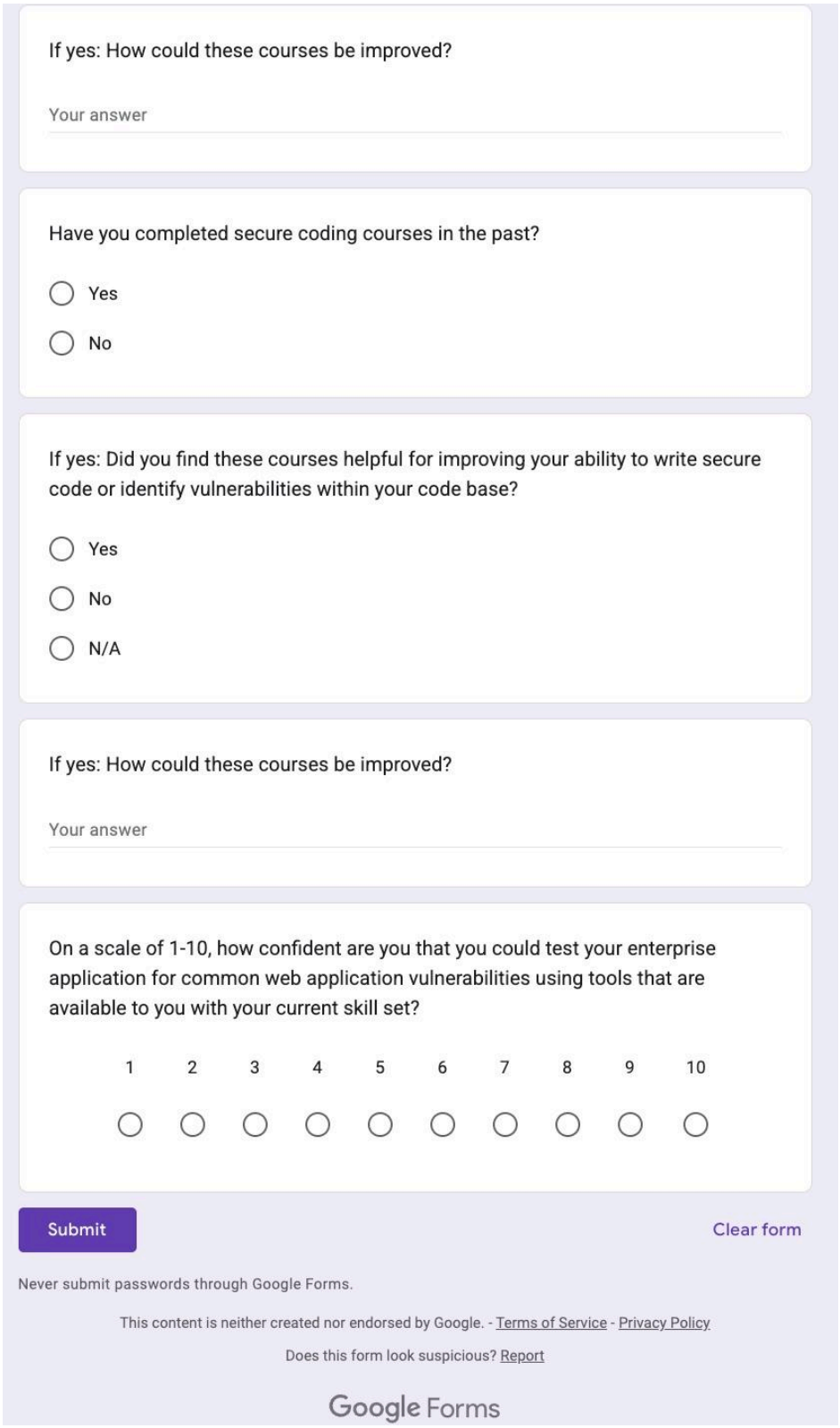
Have you taken any web application penetration testing courses before?

- Yes
- No

If yes: Do you believe these courses helped improve your ability to write secure code or identify vulnerabilities within your code base?

- Yes
- No
- N/A

Appendix B Figure 1: Pre-Training Survey



Appendix B Figure 2: Pre-Training Survey Cont.

Web App Pentesting Course For Developers - Post Training Survey

Please complete the pre training survey first. After, view the material below and fill out this survey.

Video Playlist: https://www.youtube.com/watch?v=oNDEU_uNtzI&list=PLqPCUirqsN_x_seY51DiivfV-CjgJDKUp

Checklist: https://github.com/SchoolProjZS2025/SteeleCapstone2025/blob/main/PracticumProjectChecklist.xlsx

Payload list: https://github.com/SchoolProjZS2025/SteeleCapstone2025/blob/main/ShortPayloadList.md

Sign in to Google to save your progress. Learn more

Name:

Your answer

Years of experience as a developer:

- ◯ 0
- ◯ 1-2
- ◯ 2-5
- ◯ 5-10
- ◯ 10+

Years of experience as a web application penetration tester:

- ◯ 0
- ◯ 1-2
- ◯ 2-5
- ◯ 5-10
- ◯ 10+

Please watch at least two videos in the series before answering the questions below. If possible, follow along using your own application, a test app, or public sites(non-malicious passive checks only).

Please note that the tools video is only showing tools that can be used for testing and the code analysis video does not show any vulnerability testing.

- ◯ Ack

Appendix B Figure 3: Post-Training Survey

Which videos did you watch?

- [ ] Tools
- [ ] Continuous Monitoring
- [ ] Response Headers
- [ ] Cookie Checks
- [ ] Code Review Resources
- [ ] Cross-Site Scripting
- [ ] SQL Injection
- [ ] SQL Injection on Login Pages
- [ ] Directory Traversal
- [ ] OS Command Injection
- [ ] Login Page Checks
- [ ] Session Management
- [ ] Clickjacking
- [ ] File Upload
- [ ] Horizontal Priv Esc
- [ ] Vertical Priv Esc

Did the videos teach you about new vulnerabilities?(Yes/No) Feel free to elaborate.

Your answer

Did the videos improve your understanding of risks and remediation guidance for the vulnerabilities shown?

- ◯ Yes
- ◯ No

Did you use the methodologies shown in the videos to find vulnerabilities in an application? If not, skip down to the next section of questions.

- ◯ Yes
- ◯ No

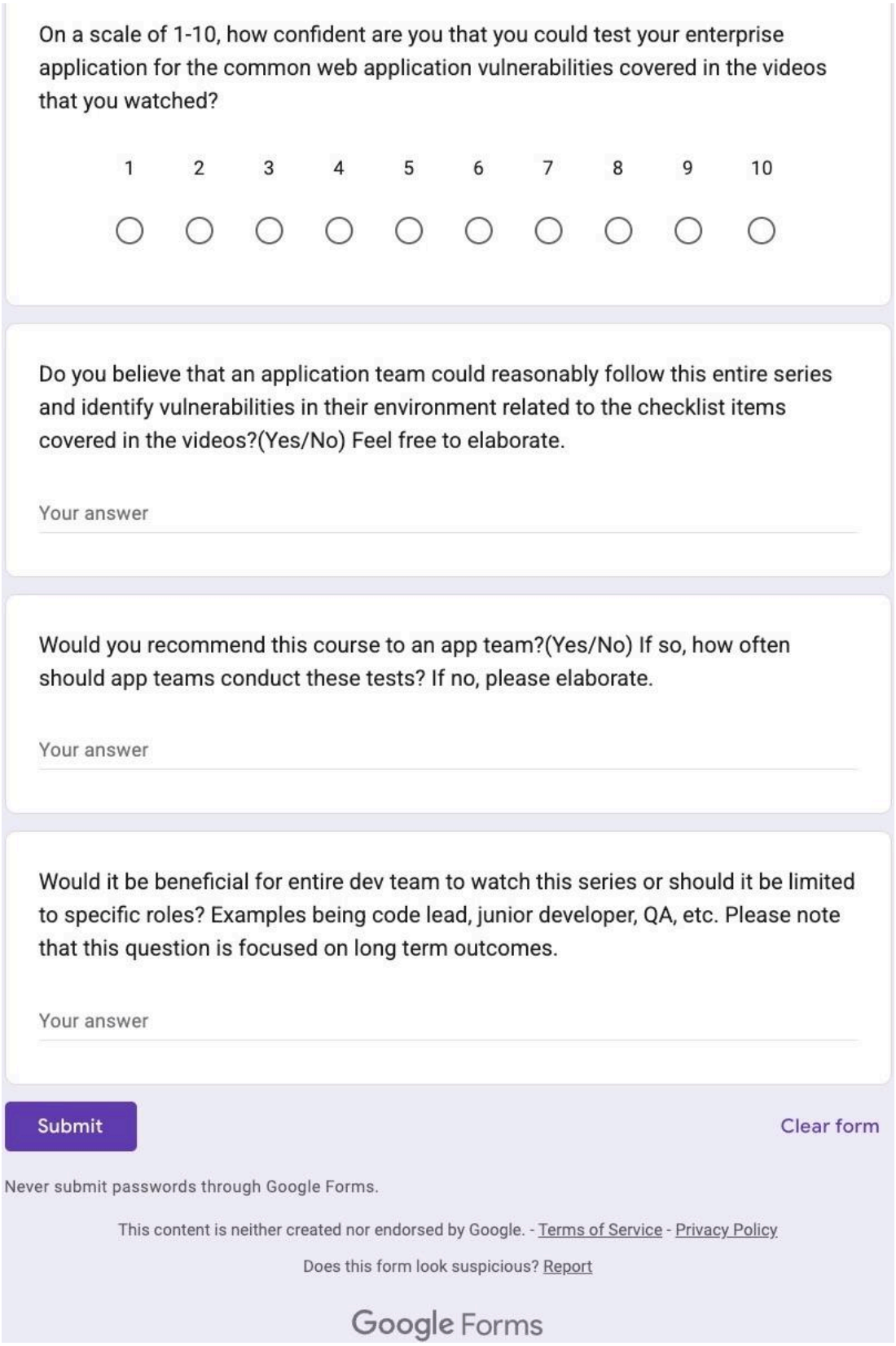
If Yes: How was your experience attempting to follow the videos? Please provide feedback on the pros and cons of the material, explanations, and format.

Your answer

If Yes: Were you able to identify any vulnerabilities or misconfigurations in the application you tested?(Yes/No) If so, what were they?

Your answer

Appendix B Figure 3: Post-Training Survey Cont.



Appendix B Figure 4: Post-Training Survey Cont.