Exploring The Gap in Web Application Security Courses

Zach Steele
PUBP6727
07/20/2025

Georgia Tech

Introduction

- Work experience
  - Penetration tested over 150 applications.
  - Explain reproduction methodologies.
  - Teach remediation guidance.
  - Point developers to pre-existing secure code.
  - 4 years as a developer.
- The research
  - Developers do not see application security as a top priority.[1]
  - Only 69% of critical vulnerabilities receive patches.[2]
  - 43% increase in CVEs reported in 2023 compared to 2021.[3]
  - AI code assistants are increasing the amount of insecure code in applications.[9]

The Problem

Web application vulnerabilities continue to rise each year due to the gap in training courses and tooling options available to developers.

- Courses are often too high level, or require massive time commitments.
  - 68% of developers only do secure code training because of a compliance need or recent exploit.[5]
  - 88% of respondents were not aware of a good course or cited time constraints as their primary reason for not taking secure coding courses.[6]
- Programmers cannot utilize penetration testing tools due to corporate policies and time constraints.[7]
  - 68% of companies do not do annual or bi-annual penetration tests.[4]

My Solution

My goal is to teach developers how to efficiently identify and remediate vulnerabilities in their applications, without using penetration testing tools, to reduce the likelihood of deploying insecure code to production.

- Web application penetration testing course for developers.
  - Vulnerability checklist.
  - Payload lists and malicious files for testing.
  - Third party references for further education.
  - Secure coding practices template.
  - Video series teaching penetration testing concepts.
- How is this unique?
  - No penetration testing tools required.
  - Focused on discovering vulnerabilities.
  - Show real world examples.
  - Reduced overhead.

Code Analysis Resources

One of the first videos in the series teaches code review basics and provides links to third party learning resources. The goal of this content is to encourage developers to start focusing on their code as they are building new functionality. I highlight the OWASP[8] checklist and code review guide as the framework for creating strong coding standards within a development team. Lastly, I provide real examples of vulnerable code alongside the secure corrections.

Don't allow untrusted user input in the following functions.

```
eval()
innerHTML
```

Additional resources

https://www.iothreat.com/blog/dangerous-js-functions https://cheatsheetseries.owasp.org/cheatsheets/Nodejs_Security_Cheat_Sheet.html

Regex Filtering

If you would like to see specific examples of attacker payloads, visit the following link. This may help determine what characters should be allowed in your specific context.

https://github.com/swisskyrepo/PayloadsAllTheThings

The following regex filters may cause a denial of service.

Source: https://pentesterlab.com/badges/codereview

```
/^dev-(\w+)+\d+\.website\.com$/
(a+)+
([a-zA-Z]+)*
(a|aa)+
(a|a?)+
(.*a){x} for x \\> 10
```

Allowed regex: `^[a-zA-Z0-9 ]*$`

Georgia Tech

	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/
Session Management				
Session Fixation Not Possible		Medium	Issue fresh session cookies after login	https://owasp.org/www-community/controls/Session_Fixation_Protection

Malicious Files

Multiple test cases require malicious files with different payloads and file extensions. I walk through each file and their intended behavior while stressing that we cannot test with files found online due to unknown security risks they may present. Since most organizations do not allow employees to download specific file types or content, I provide the file contents for the user to copy paste. In the event that company policies prevent employees from using our files, I also teach the viewer to rely on non-malicious files that are already on their computer by default.

We will not be hosting files because most people cannot download vulnerable files off github from company networks. Please understand how these files work before utilizing them.

Clickjacking

Put the link to your site into the iframe section at the bottom of each file, then save as an html file. Log in to your application, then double click on the following html file so that it opens in the same browser. If you can see your website in the iframe, this proves your application is vulnerable to clickjacking.

The following files come from PortSwigger: https://portswigger.net/web-security/clickjacking

Basic: test1.html

```
<style>
    iframe {
        position:relative;
        width:$width_value;
        height: $height_value;
        opacity: $opacity;
        z-index: 2;
    }
    div {
        position:absolute;
        top:$top_value;
        left:$side_value;
        z-index: 1;
    }
</style>
<div>Test me</div>
<iframe src="https://www.yourSiteHere.com/dashboard"></iframe>
```

Georgia Tech

Payload Lists

Viewers can choose between the short payload list and the long payload list for all checklist items. The long payload list includes more payloads for greater coverage, while the short payload list focuses on the highest success rate payloads for various situations. The payloads were chosen based on my experience as a penetration tester, where I have built out my own flowcharts and cheat sheets to test more efficiently.

SQL Injection on Login Pages

Username Field:

```
'
''
admin'--
admin')--
admin' or 1=1--
admin') or 1=1-- -
admin' AND 1=1--
admin'||1=1#
```

Password Field:

```
'
''
' or 1=1--
') or 1=1-- -
```

Directory Traversal

Please use a file/file path that exists on the web server. For example, etc/passwd if its a linux server and windows/win.ini if it is a windows server. Extend the ../ as needed.

```
../../../etc
../../../etc/passwd
....//....//....//etc/passwd
..%252f..%252f..%252fetc/passwd
../../../windows/win.ini
```

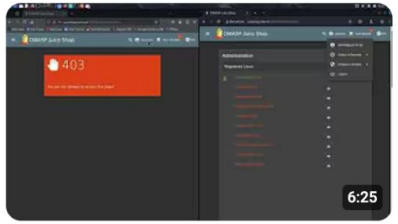
Georgia Tech

Video Series

The YouTube video series walks the viewers through the purpose of this course, how to use the material, and how to safely add new test cases even if the viewer does not fully understand how to exploit vulnerabilities in that category. Each video contains a timeline, allowing the user to quickly skip around to relevant test cases or sections of each checklist item. Lastly, the video descriptions contain links to the material shown in the demo and any relevant third party resources to learn more about the vulnerabilities, exploits, and remediation.

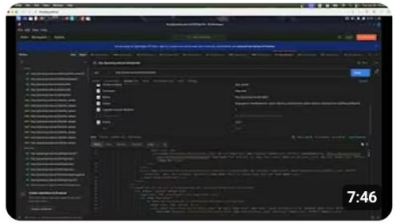
Optional: How To Expand The Checklist

13 views • 3 days ago



Vertical Priv Esc

13 views • 4 days ago

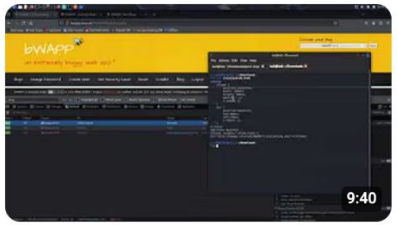


Horizontal Priv Esc

16 views • 5 days ago

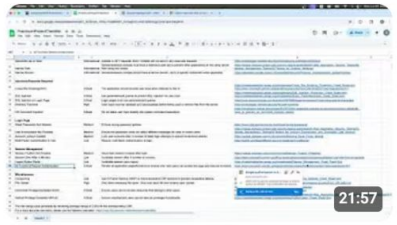
File Upload Vulnerabilities

29 views • 6 days ago



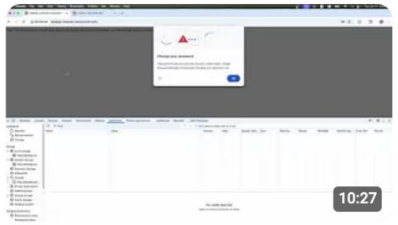
Clickjacking

29 views • 6 days ago



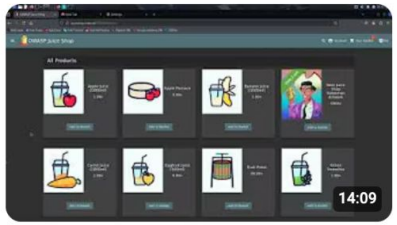
Session Management

27 views • 7 days ago



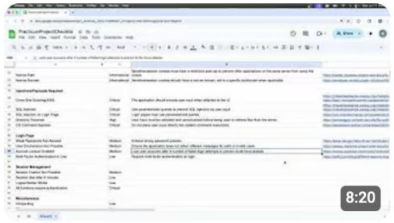
Optional: Comprehensive Testing Without Any Tools

12 views • 8 days ago



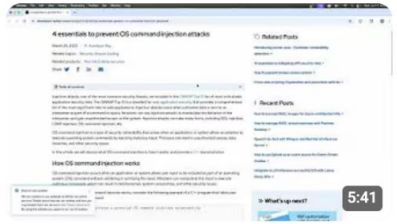
Optional: Business Logic Vulnerabilities

22 views • 12 days ago



Login Page Vulnerabilities

21 views • 12 days ago

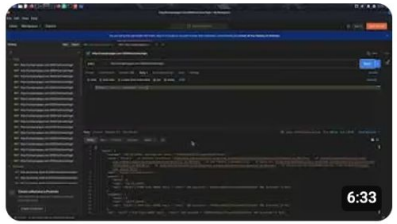


OS Command Injection

28 views • 2 weeks ago

Directory Traversal

27 views • 2 weeks ago



SQL Injection on Login Pages

22 views • 2 weeks ago

Georgia Tech

Evaluation

- Surveyed managers, application owners, penetration testers, and developers.
- Pre-training survey
  - Work Experience.
  - Experience with penetration testing/secure coding courses.
  - Course improvement opportunities.
  - Confidence identifying vulnerabilities.
- Post-training survey
  - Test an app with the course material.
  - Provide results and feedback.
  - Confidence identifying vulnerabilities.
  - Thoughts on testing frequency and audience.
- Interviews
  - Extract more information from the viewers.

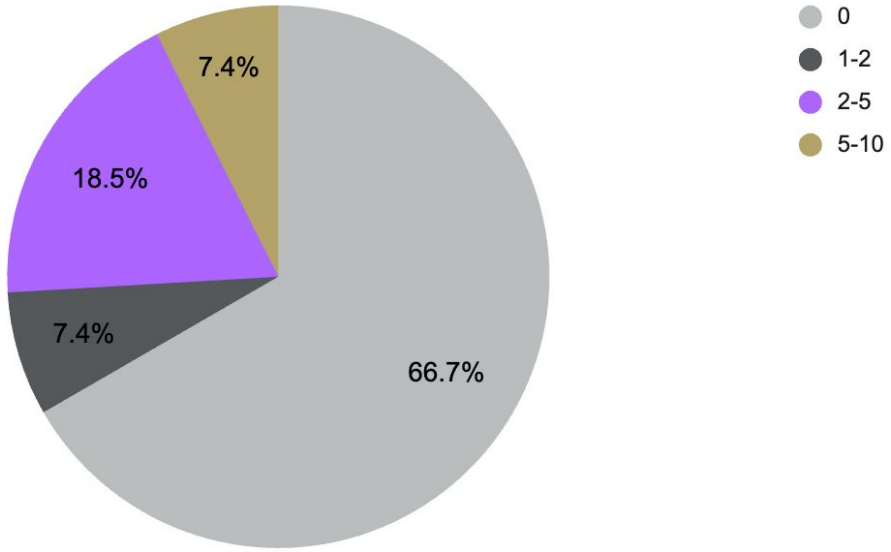
Survey Limitations

- Personal Bias
  - Friends may feel pressure to give favorable responses.
- Selection Bias
  - Penetration testers will be less likely to learn from my material, but more likely to see potential value.
  - Developers were more likely to increase confidence in testing after viewing the course.
- Non-Responsive Bias
  - Less than half of the recipients completed the pre-training survey.
  - Half of that group completed the post-training survey.
- Response Bias
  - Making unbiased survey questions can be extremely difficult.

Survey Participants

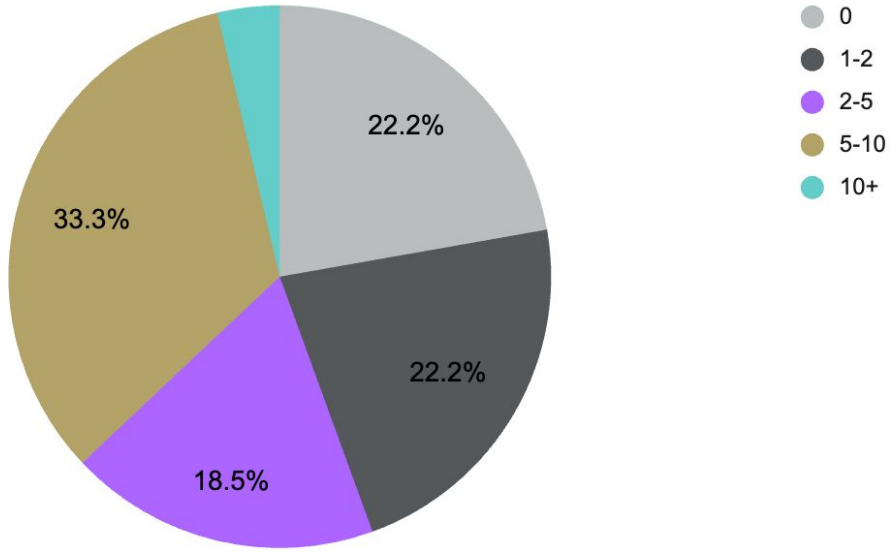
Web App Penetration Testing Experience

27 responses



Development Experience

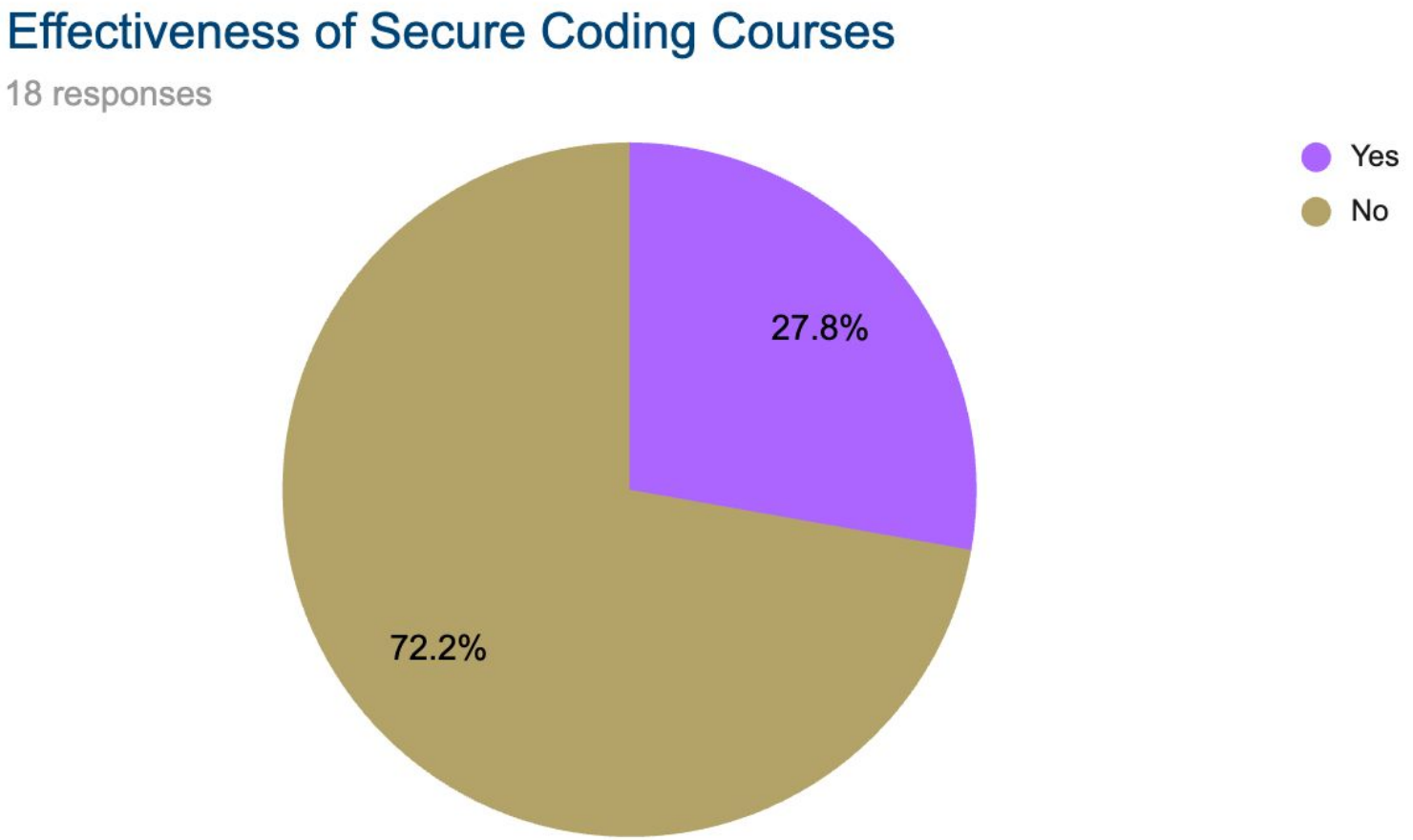
27 responses



Georgia Tech

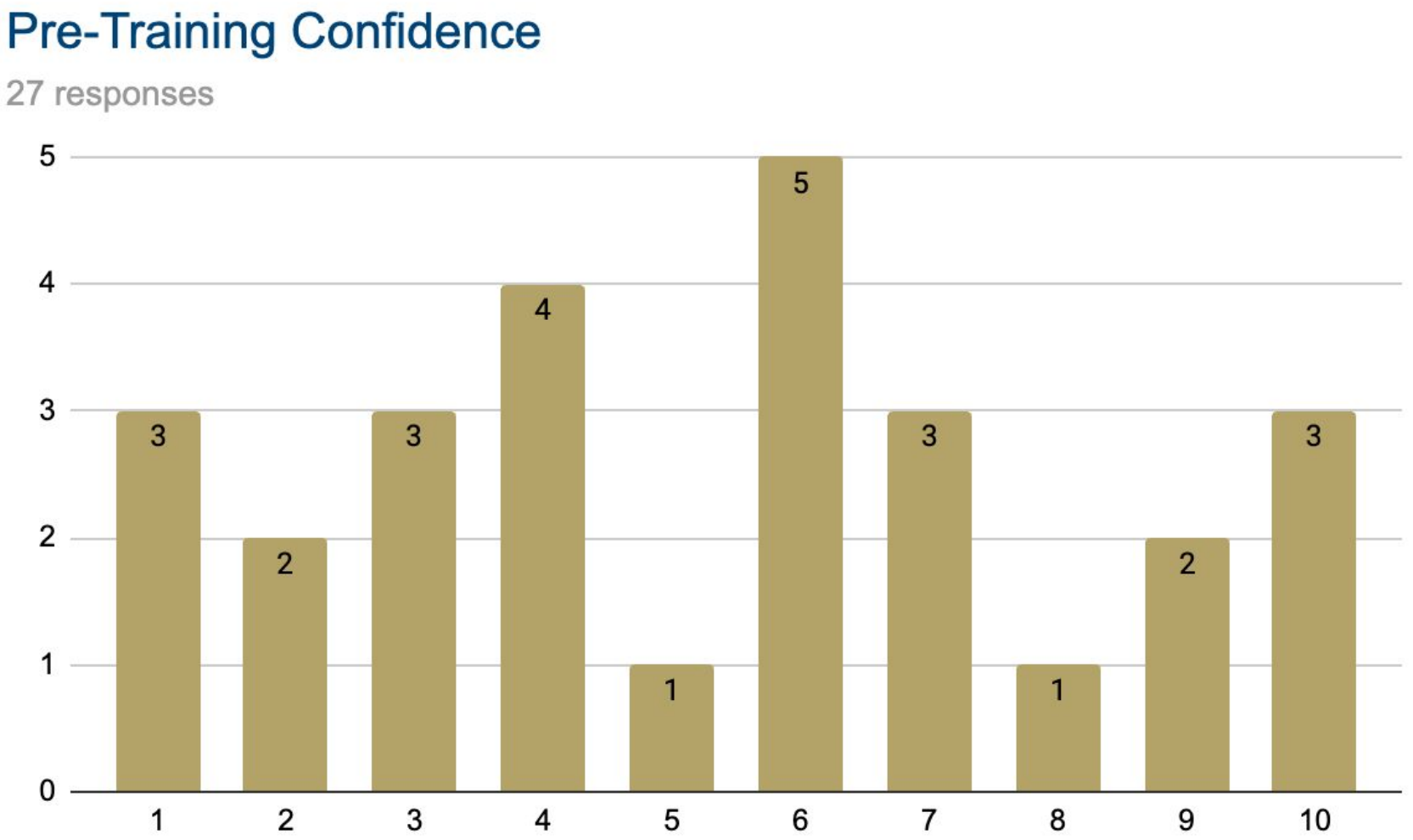
Past Course Feedback

72% of respondents stated that past secure coding courses and trainings did not improve their ability to write secure code or identify vulnerabilities within their code base. The group explained the biggest issues with secure coding courses are the lack of real world examples, reliance on powerpoint training, and a lack of material to continue learning after the course is over.



Georgia Tech

Pre-Training Survey Results Cont.

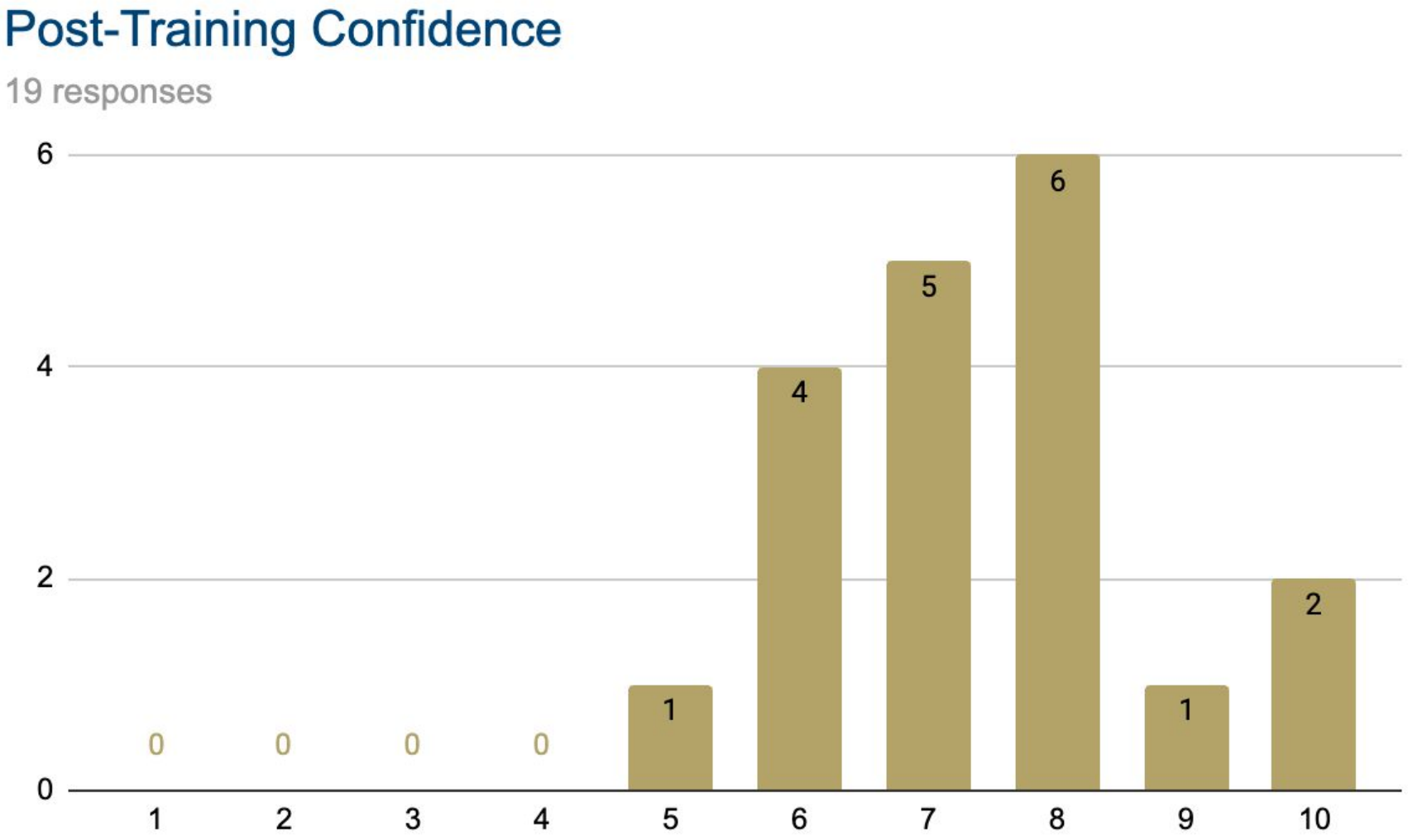


Post-Training Survey Results

- Survey Complications
  - Two participants lowered their confidence after watching the videos due to overconfidence prior to seeing the exploits.
  - During interviews, respondents mentioned their hesitance towards edge ratings.
- Video Series Learning Opportunities
  - 100% of participants with less than 2 years of pentesting experience learned from the series.
  - 94.7% stated the course improved their understanding of risk and remediation.
  - 13 individuals used the material to find vulnerabilities within an application.
  - Most respondents recommended the training for all developers.
  - Participants recommended dev teams test their application using my course before major code releases or on an annual basis.

Georgia Tech

Post-Training Survey Results Cont.



Solution Limitations

- Edge Case Exploits and Payloads
  - We chose accessibility over exhaustive payload lists.
  - There is not enough research on specific payload success rates.
- Lack of Topics
  - Focused on common vulnerabilities, leaving multiple less common categories untested.
  - Ex: WebSockets are only utilized in 7.3% of applications.[15]
- Executive Buy-In
  - Team managers have to buy-in and give their devs the time to learn and test.
  - This could also assist with better metric tracking.

Future Work

- Compile Feedback
  - Update course content to fill any gaps.
  - Re-film videos to better explain any lacking items.
- Add new vulnerabilities to the course
  - Conduct research into new vulnerabilities.
  - Review survey feedback for desired new checklist items.
- Update the survey to remove bias
  - Ensure participants are taking the survey seriously.
  - Fix the non-responsive bias.
- Engage management teams for increased buy-in
  - Discuss the benefits of the course.
  - Attempt to restructure time allotment for the developers.

Conclusion

- The Research
  - Developers do not have the time or tools to utilize penetration testing courses.
  - Most secure coding courses are too high level to be helpful.[1]
- Bridging the gap
  - Created a penetration testing course designed for developers.
  - Short, high success rate, payload lists.
  - Easy to follow video series.
- The Results
  - Developers saw a 2.8/10 confidence increase in detecting web app vulnerabilities.
  - 13 participants used the course to find their first vulnerability.
- Future Work to reduce limitations
  - Gaining management buy-in.
  - Expanding course content.
  - Rebuilding the survey.

Georgia Tech

Citations

1. *Secure code warrior survey finds 86% of developers do not view application security as a top priority.* Secure Code Warrior. (Apr 05, 2022).
2. 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
3. Armstrong, K., Williams, K., Landfield, K., & Lawler, S. (2024, October). *25th Anniversary Report*. CVE. https://www.cve.org/Resources/Media/Cve25YearsAnniversaryReport.pdf
4. 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/
5. *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
6. 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
7. 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

Georgia Tech

Citations Cont.

8. OWASP Web Application Penetration Checklist. OWASP. (n.d.). https://owasp.org/www-project-web-security-testing-guide/assets/archive/OWASP_Web_Application_Penetration_Checklist_v1_1.pdf
9. 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
10. Steele, Z. (2025, July). Web Application Penetration Testing Crash Course For Developers. GitHub. https://github.com/SchoolProjZS2025/SteeleCapstone2025
11. 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
12. 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
13. *DB-Engines ranking*. DB-Engines. (2025, June). https://db-engines.com/en/ranking
14. Survey bias types that researchers need to know about. Qualtrics.(n.d). https://www.qualtrics.com/experience-management/research/survey-bias/
15. 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

Georgia Tech