Video Series

PLEASE NOTE: This is my own personal notes for working through the video series. I saved this in GitHub to show the work going on behind the scenes, this is NOT part of the official training course material.

Links:

https://github.com/swisskyrepo/PayloadsAllTheThings

to start machine: open postman, burp suite, start juice shop, open PS, start apache2 and mysql services.

Video 1: Introduction to the video series – what we are doing and why. (5-10 mins)

In this video show the list of videos and explain that more material will be linked throughout if you want to go learn and try more. Show the checklist and payload list.

Mention that this testing methodology is not a full pentest meaning your app is secure. Our goal is to do a quick and dirty sanity check on our app to prevent exploitation from an attacker and in a lot of cases ensure that the pentesting team isn't going to find a lot of low hanging fruit in your app. The best thing to do is get a full pentest or run a Burp scan over your app. For the sake of this entire video series, we are assuming that we are testing lower lane environments and we do not have access to pentesting tools because most dev teams do not. This would be a great task to give to a junior developer or someone on your team that wants to transfer into security. A junior would learn a lot about writing secure code. Make sure to mention that if you can't test your app, you can do the payload free sections and the SCA section. —also show portswigger as the best alternative for a full course because we are not here to do perfect pentesting across all subjects.

Video 2: Intro to the tools – Show the tools that can be used to complete this series (10 mins)

open curl, postman, insomnia, burp. Show how we can get a request from the browser into each, then show sending a random payload in each so people have an idea of the flow.

-explain why we need a tool using encoding lab and change password in juice shop.

-show how to import a request into postman

-show how to use the browser with fetch

Optional: why we don't use other browsers video bc encoding on send. (5 mins)

Show how the different browsers encode payloads, so you have to be careful with your testing tool.

Video 3: Best coding practices and remediation overview. (10 mins)

This is not a secure coding practices course, it is a pentesting course designed for developers to find vulnerabilities in their own apps. However, I have set up sort of a template page for the app teams to fill out themselves with language specific links. We wouldnt have a lot of these vulns if the app teams coded securely, so this will try to nip our issues in the bud.

First, i recommend the entire dev team read the high level coding links.

Second, your code leads should all read the deeper links.

Then, your app team should work together to take the important pieces to build your own secure code development guidelines based on your companies risk appetite and security policies.

Again, this is more of a template with some examples, but here is how it could be used.

link to good coding practices and discuss blanket remediation like input validation, output encoding, fixing response headers and rate limiting will severely help. We can also discuss the basics of SCA and review. Show off the in progress github page.

https://owasp.org/www-project-secure-coding-practices-quick-reference-guide/assets/docs/OWASP_SCP_Quick_Reference_Guide_v21.pdf

https://owasp.org/www-project-code-review-guide/assets/OWASP_Code_Review_Guide_v2.pdf

Testing section

Video 4: Picking endpoints to test and covering continuous review.(25 mins)

what endpoints are important to us and what can we see before sending payloads. We can mention cookies and response headers here, but really we are focused on the continuous review section of the checklist. Those might be priority targets for our

payloads section. Can also cover what we can ignore, like js files and non sensitive endpoints.

Navigate around the app pointing out important endpoints and mention that we will use this method to select what we will use in future videos along with random one off labs.

If you know that user input is making its way into operating system commands, being reflected back on the page, or being included in SQL queries make sure to include these in your testing.

Aim: bwapp xml/xpath lab: click any button and see fatal error. This is bad from a user experience perspective, but also this helps an attacker figure out that they need to test out this endpoint. simplexml_load_file() tells us as the attacker what we should be trying here. Verbose error messages often lead to XSS as well.

Next show Juice shop login page, show the response itself to see the full error in inspect element network tab. The response shows an sqlite error. We can see exactly where our input is put into the SQL query.

Auto complete disabled and sensitive information in the URL. Show when we register it saves our inputs. If you logged in from the library someone could come behind you and see your answers from the registration page.Search for autocomplete=off

So if you ever get the google pop up to save credentials, that is a key indicator that autocomplete is not set to off.

Show that the post request to login is POST request meaning its encrypted over HTTPS and no machine in the middle can see it. However on the password change page we can see it in the URL meaning even over HTTPS anyone can see the new password

Video 5: Response headers. (15 mins)

Make sure to explain why at the start of each item

Ensuring secure response headers will help reduce impact of some attacks and even prevent execution of certain payloads.

There are nuances with apis, but we will mostly ignore that here.

Cache controls - talk about back button capability. And using a library computer may store things locally that an attacker could see afterwards.

CORS - if you had vulnerable cors on your app, facebook could send a request to your app then read the info in the response. Allowing them to enumerate sensitive information or even see if their attack is working depending on the setup of their exploit and what other vulnerabilities were chained with this.

Video 6: Cookies (10 mins)

Make sure to explain why along the way

Video 7: Cross-Site Scripting (XSS): 15 mins

In this video we will be covering the first item in our payloads required section of the checklist, but before we jump into testing I want to cover a few disclaimers. First, as mentioned throughout the documentation of this course and in our intro video, this course should not be treated as a complete and comprehensive penetration test. We are introducing the concepts and teaching the basics, but there are much better resources out there if you want to learn tons of edge cases and become self sufficient with your learning and pentesting abilities.

If you want a full pentest go learn from those resources, pay for a full manual pentest, or at the very least run some commercial scanners over the app.

Next, our payload lists are small because we want to cover a lot of ground quickly, we dont want to get caught up sending a bunch of low percentage payloads. While this does reduce the risk of settings off alerts or getting blocked by WAFs, I do want to point it out as a possibility since most companies will have alerts in place for these types of payloads. The last thing I want to note is that we use non-malicious payloads, do not start throwing random payloads offline into your application without any understanding of what that payload does. With that said, the best resource for massive payload lists and edge case situations is payload all the things.

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS%20Injection

Disclaimer - PortSwigger and OWASP are great at teaching the entirety of a penetration test and covering a bunch of edge cases. We are not here to teach the audience how to be top tier pentesters, there are entire companies dedicated to that(like portswigger and owasp). We are here to teach common situations with common payloads. If you want a full pentest go learn from those resources, pay for a full manual pentest, or at the very least run some commercial scanners over the app.

Start with what can be done with XSS and if you want to learn more you can go to XYZ places. Go into Payloads, then jump into remediation.

What is XSS? Cross-site scripting is a vulnerability caused by an application incorrectly interpreting user input as valid HTML or javascript. where the application that allows an attacker

What is the risk of XSS? This allows an attacker to input some type of malicious script into the application that will be executed by the victim users browser. We will do some basic testing in a moment, but some examples of full exploitation include stealing a victim users cookie, having that user take some action in the application like altering our attacker users account to change the role to admin or even access some sensitive data that the victim user has access to in the app.

There are multiple ways to deliver this payload to the victim. For example, we could send them a link to our attacker website, have them click on a link to the victim application, or in some cases the payload might be stored in the application somewhere and it would execute when the victim stumbles across that page. We could spend 20 minutes going down the rabbit holes of risk and delivery mechanism nuances, but that is a quick overview.

First example, reflected xss into html context with nothing encoded. I am going to do this lab using burp suite because I think it makes the demonstration more clear, but afterwards we will switch back to our postman project.

First we use bold tags to see that our input is being reflected back, lets look at the response to see it. Unless you have a specific need to allow users to use bold tags or anything similar, we could go ahead stop here and jump down to remediation. But for the sake of the demonstration, lets move down to the next payload in our list.

Lets try a second lab SWITCH OVER TO POSTMAN, this time we will look at the behavior of the app as we try these payloads. Reflected xss into attribute with angle brackets html-encoded.

Our last example is a common situation you might find yourself in if you have a content-security policy in place. Reflected XSS into protected by csp, with csp bypass. We will use the developer tools to see why this isnt executing. Keep this in mind as you are trying these payloads, but again, remember that we dont need execution to prove that we need to remediate the endpoint.

LASTLY, lets cover remediation. First, all user input should be encoded on output, this will prevent the browser from misinterpreting user input as javascript.

Second, and this is a blanket piece of remediation advice we will see throughout this video series, validate user input with an allowlist. If the parameter does not expect users to use special characters, go ahead and deny that users request entirely. You can try to sanitize user input, but that comes with its own challenges. In the checklist we have included a couple resources for remediation guidance, but feel free to change this out for something that is more specific to your application.

This will be where we explain how to use the payload list to quickly scan over endpoints looking for weird interactions. Remember to point out the content type in the response for XSS or checking for CSP as helpers to see if its even possible to exploit.

Video 8: SQL Injection 15 mins

Lets take a look at sql injection and sql injection on login pages. These are separate items in our checklist because even though the risks and remediation will be very similar, the payloads and testing methodology will be slightly different. Since this is such a difficult vulnerability to test for, I encourage you to do targeted testing based on your knowledge of the code base. So, we will look at payloads for MySQL, Oracle, and a few others, but you should know what database is being used so you should go ahead and use the payloads follow the correct format. Additionally, SQL injection happens because user input finds its way into an SQL query, so if you can, start by testing the parameters that you know are being used in SQL queries.

Lets start off by discussing what SQL injection is and what the risks are.

SQL injection is a vulnerability that allows an attacker to manipulate sql queries by injecting malicious SQL into user input that gets placed into an SQL query on the back

end. They could use this to view sensitive data, modify or delete data, or even make changes to the database itself. Depending on the limitations of the situation, this may vary, but imagine giving an attacker full access to your database. They would run destructive commands, try to steal any information that may be valuable or even try to shut down the database itself. These attacks can get pretty in depth because of the power this vulnerabilities lends to an attacker.

Cover risk, do a few labs showing how to use the full list of basic payloads because behavior can be different. Explain remediation. Make sure to cover the fact this requires a connection to SQLi, even if it's an API to another tool/team's app.

Lab 1: PS SQL injection attack, querying the database type and version on oracle

Juice shop search

Lab 2: PS Blind SQL inj with time delays

Video 9: SQL Injection on a login page 15 mins

Explain how this is different from the other SQL injection section.

Juice shop login page

Lab 2: PS SQL inj vuln allowing login bypass

Video 10: Directory Traversal 15 mins

Explain how this can lead into OS command injection and what parameters would be vulnerable.

What is path traversal? Path traversal or directory traversal is a vulnerability that allows an attacker to access files and directories on the web server by manipulating the file path in a given parameter.

The risks of this vulnerability are having an attacker see any application code or system files. Additionally, I want to mention that endpoints that are vulnerable to path traversal are often vulnerable to command injection, which is our next video. While it may not be necessary to test for command injection if you find path traversal, I do want to make note of that.

Video 11: OS Command Injection 15 mins

Explain how this vulnerability arises and what parameters to test.

OS command injection, or just called command injection, is a vulnerability where an attacker is able to execute operating system commands on the web server. The risks associated with this include viewing sensitive data, pivoting to other systems, or taking distructive actions. This is a very powerful exploit, just imagine giving an attacker access to your web server, there are tons of different things they could do.

Remediation: first, dont put user input into OS commands in the first place, otherwise, allowlist input so users are only inputted the characters that are expected for that parameter.

Video 12: Login Page Issues 15 mins

Explain that account lockout can help prevent password brute forcing. Discuss time based username enumeration and explain that this requires tools showing response times.

Optional: testing process

This is my methodology, but there are other ways you could break up this work. Ultimately we need to check every checklist item, but you can blow through the checklist in a way that is more comprehensive and gives more context to what we are testing as we go.

Im always looking for the continuous review items as I go through this, so that kind of goes without saying.

First, I hit the first page of the app and check the response headers.

Next, I login to the app, see if my password autocompletes next time. Note that the app is missing mfa.

See if the cookie is new and check the cookie attributes while im here.

Test the logout button

Test the session inactivity time.

Look for a file upload page.

Test clickjacking.

Test a few requests from an unauthenticated state.

–This is by no means a complete pentest and its important to pentest every piece of your application. But this quick demo has given us pretty good coverage for systemic issues and test cases. Now its about gaining context and trying every single endpoint.

Video 13: Session Management 15 mins

Video 14: Clickjacking 5 mins

Provide 2 HTML files on github.

Video 15: Malicious File Upload 5-10 mins

Provide XXE, SVG, HTML files.

Video 16: Outro 5 mins

Optional: NoSQL Inj 10 mins

Optional: XXE 10 mins

Optional: SSRF 10 mins

Youtube video descriptions:

Links in every video: Our github