

OFFSEC

OSWE — Web Expert: Advanced Web Application Attacks Preparation Training

Delivered from Hacky Security's own curriculum, by instructors who also work our security operations centre. Classroom, online live, or at your office.

EXPERT

At a glance

COURSE	OSWE — Web Expert: Advanced Web Application Attacks	WHAT IS INCLUDED ✓ Instructor-led delivery by practitioners who also work our SOC ✓ Hands-on lab access for the duration of the course ✓ Course materials and lab guides ✓ Hacky Security certificate of completion ✓ Post-course support by email
TRACK	OffSec	
DOMAIN	Cyber Security	
LEVEL	Expert	
DURATION	On request	
DELIVERY	Classroom · Online live · On-site at your office	
LANGUAGE	English	
CLASS SIZE	Small cohorts — everyone gets lab time	

Who this is for

This is a expert course. It suits people already working in or moving into Cyber Security who want structured, hands-on preparation rather than a video library. If you are unsure whether the level fits your team, send us the job titles and we will tell you honestly — including if a different course would serve you better.

Course outline

About this course map

Twenty-four topics, mapped one-for-one to the OffSec WEB-300 (Advanced Web Attacks and Exploitation) syllabus and taught in the same order, so nothing the exam draws on is left uncovered. WEB-300 is the course that leads to the OSWE; this is Hacky Security's own preparation for it, not OffSec courseware and not OffSec lab access.

WEB-300 is a white-box course. Unlike the network-focused OSCP, almost everything here starts from the application's source code: you read a real codebase, find the flaw by following the data, and then write the exploit that proves it. The published topic list covers client-side flaws such as prototype pollution and DOM-based XSS, server-side classes such as SSRF, SQL injection, deserialisation and template injection, and the filter and comparison tricks — type juggling, magic hashes, regex and character bypasses — that turn a half-working payload into a working one. We teach each in the same sequence OffSec lists them.

Most of the time is spent reading code and writing code. The exam is not a matter of running a scanner and picking a CVE; it is failed by candidates who can spot a vulnerability but cannot turn it into a single script that exploits it end to end without a human in the loop. So the emphasis throughout is on source-code review and on building clean, self-contained proof-of-concept exploits, because that is exactly what the exam scores.

Module 01 — JavaScript Prototype Pollution

A modern client- and server-side flaw class where polluting a base object changes behaviour somewhere else entirely.

- How JavaScript's prototype chain works and why it is mutable at runtime
- Recognising the merge, clone and assignment patterns that let attacker keys reach a prototype
- Tracing a polluted property from its source to the sink where it causes harm
- Escalating pollution into denial of service, property injection or code execution
- Finding these sinks by reading the application and its dependencies rather than fuzzing blind
- Writing a repeatable exploit that sets up the pollution and triggers the payoff

Module 02 — Advanced Server-Side Request Forgery (SSRF)

Making the server issue the requests, then bending where and how those requests land.

- The mechanics of SSRF and why server-side clients trust their own network
- Reaching internal services, metadata endpoints and loopback-only interfaces
- Bypassing allow-lists and naive URL parsers with encoding, redirects and DNS tricks
- Turning a blind SSRF into a useful primitive through timing and out-of-band channels
- Chaining SSRF with a second flaw to reach code execution
- Building the exploit so it works against the target's actual filtering, not a textbook example

Module 03 — Web Security Tools and Methodologies

The working method that the rest of the course sits on: how to approach an unfamiliar codebase and drive the tools around it.

- A repeatable approach to reviewing an application you have never seen before
- Setting up an intercepting proxy and debugger to watch requests and code together
- Recovering and reading source code, and mapping routes to the handlers behind them

- Static and dynamic analysis, and knowing when to switch between them
- Scripting in Python to automate discovery and drive an exploit
- Keeping notes and evidence in a form the exam report will need

Module 04 — Source Code Analysis

The core skill of the certification: finding the bug by reading the code, not by poking the running app.

- Following user-controlled input from entry point to dangerous sink
- Reading authentication, session and access-control code for the gap between intent and behaviour
- Spotting insecure defaults, dangerous functions and unsafe library calls in context
- Working through a large codebase without getting lost in it
- Confirming a suspected flaw against the running application
- Turning a code-level finding into an exploitation plan

Module 05 — Persistent Cross-Site Scripting

Stored XSS as an attack on other users and on the application's own privileged interfaces.

- Where stored input is later rendered without adequate encoding
- Getting a payload past server-side sanitisation and context-specific filtering
- Targeting administrative and internal views that a normal user cannot reach directly
- Using stored XSS to drive actions in a higher-privileged session
- Reading the template and output layer to find where escaping is missing
- Delivering a payload that survives storage and fires reliably on render

Module 06 — Session Hijacking

Taking over an authenticated session by attacking how the session is issued, carried or predicted.

- How session tokens are generated, transmitted and validated
- Stealing or leaking a token through a companion flaw such as XSS
- Exploiting weaknesses in token handling and cookie configuration
- Reusing a captured session to act as the victim
- Chaining a hijack into escalation towards an administrative account
- Scripting the capture-and-reuse sequence for the exploit

Module 07 — .NET Deserialization

Untrusted serialized data turned into code execution on a .NET application.

- How .NET serialization works and why deserializing attacker data is dangerous
- Recognising vulnerable formatters and the sinks that call them
- Building deserialization gadget chains, including with ysoserial.net
- Getting a crafted object accepted by the application's input path
- Confirming execution and turning it into a stable primitive
- Packaging the whole chain into a single working exploit

Module 08 — Remote Code Execution

Reaching arbitrary execution on the host, usually by chaining what the earlier modules found.

- The routes that lead from a web flaw to running commands on the server
- Combining a foothold vulnerability with a second flaw to reach execution
- Working within the constraints of the application's language and platform
- Establishing a reliable shell or command channel
- Verifying execution cleanly for the report
- Automating the chain so it fires without manual steps

Module 09 — Blind SQL Injection

Extracting data from an injection point that returns no visible output.

- Detecting injection where the response gives nothing back directly
- Boolean-based and time-based inference techniques
- Building an efficient extraction routine rather than a character-at-a-time crawl
- Handling the target database engine's particular syntax and quirks
- Reading the query-building code to understand exactly what you can inject into
- Writing a Python exploit that pulls the data automatically

Module 10 — Data Exfiltration

Getting extracted data out efficiently and reliably once an injection or read primitive exists.

- Turning a read primitive into a repeatable, automated exfiltration routine
- Choosing an in-band or out-of-band channel to suit the target's behaviour
- Encoding and chunking data to survive filters and length limits
- Managing rate and error handling so a long extraction actually completes
- Confirming the integrity of what you pulled back
- Folding exfiltration into the wider exploit script

Module 11 — Bypassing File Upload Restrictions and File Extension Filters

Turning a file upload into code execution when the application tries to stop exactly that.

- How upload handlers try to constrain type, extension and content
- Defeating extension allow-lists, MIME checks and content inspection
- Placing an uploaded file where the server will execute it
- Combining an upload with path or configuration quirks to reach execution
- Reading the upload-handling code to find the gap in its checks
- Scripting the upload-and-trigger sequence

Module 12 — PHP Type Juggling with Loose Comparisons

Abusing PHP's loose equality so that values which are not equal compare as if they were.

- How PHP coerces types across the loose comparison operators
- Where loose comparisons appear in authentication and token-checking code
- Crafting inputs that satisfy a check they should fail

- The interaction of type juggling with hashes and encoded values
- Finding these comparisons by reading the code, not by guessing
- Building an exploit that walks straight past the flawed check

Module 13 — PostgreSQL Extension and User-Defined Functions

Using PostgreSQL's own extensibility to move from database access towards the host.

- How PostgreSQL extensions and user-defined functions load and run
- The privileges and configuration that make this possible
- Loading a function that reaches outside the database
- Working within the version and platform constraints of the target
- Recognising the opportunity from the database's setup and permissions
- Automating the setup so the exploit is reproducible

Module 14 — Bypassing REGEX Restrictions

Getting a payload past input validation that relies on a regular expression.

- How anchoring, multiline flags and greedy matching create gaps in a regex filter
- Crafting input that the pattern fails to catch
- Exploiting the difference between what a regex checks and what the sink accepts
- Newline and encoding tricks that defeat naive patterns
- Reading the validation code to see exactly what the pattern does and does not cover
- Turning the bypass into a working payload

Module 15 — Magic Hashes

Exploiting hash values that a loose comparison treats as a number, so different inputs collide.

- What magic hashes are and why they matter under loose comparison
- The interaction with PHP type juggling from the earlier module
- Finding password or token checks that compare hashes loosely
- Producing an input whose hash satisfies the flawed check
- Recognising the pattern in authentication code
- Scripting the search for and use of a qualifying value

Module 16 — Bypassing Character Restrictions

Building a working payload when the characters you need are filtered or forbidden.

- Constructing payloads without the characters the filter blocks
- Alternative syntax, encoding and language features that stand in for banned characters
- Getting past both denylists and length limits
- Applying this within SQL, command and template contexts
- Reading the filter to know precisely what is off-limits
- Assembling a restricted-character payload that still executes

Module 17 – UDF Reverse Shells

Turning database user-defined functions into a shell on the underlying host.

- Using a user-defined function to run operating-system commands
- Building or loading the function the target's engine and platform require
- Establishing a reverse shell from inside the database
- Handling the privilege and configuration prerequisites
- Confirming host access cleanly
- Automating the deployment as part of the exploit

Module 18 – PostgreSQL Large Objects

Abusing PostgreSQL's large-object facility to read from and write to the file system.

- How the large-object interface stores and retrieves binary data
- Reading server-side files through large objects
- Writing files to disk to reach code execution
- The privileges and settings that gate this technique
- Combining it with the other PostgreSQL modules for a full chain
- Scripting the read or write into the exploit

Module 19 – DOM-Based Cross-Site Scripting (Black Box)

Finding and exploiting client-side XSS without the source, working only from the running app.

- How DOM XSS arises entirely in client-side JavaScript
- Tracing sources to sinks by reading the delivered scripts in the browser
- Detecting the flaw with no server-side code to review
- Crafting a payload that fires in the DOM context
- The black-box workflow when source is unavailable
- Building a reliable, repeatable trigger

Module 20 – Server-Side Template Injection

Getting code execution by injecting into a server-side template engine.

- How template engines evaluate expressions and where user input reaches them
- Identifying the specific engine from its behaviour and error output
- Building payloads that escalate from expression evaluation to code execution
- Working within the sandboxing and restrictions a given engine imposes
- Finding the injectable render call by reading the code
- Automating detection and exploitation for the target engine

Module 21 – Weak Random Token Generation

Predicting or forging tokens that were generated without proper randomness.

- Where applications use predictable values for security-sensitive tokens
- Recognising weak sources of randomness in the code
- Reconstructing or predicting a token from observed values

- Using a forged token to reset a password or hijack a flow
- Confirming the weakness against the running application
- Scripting the prediction and the attack that uses it

Module 22 — XML External Entity Injection

Abusing an XML parser that resolves external entities the attacker controls.

- How XXE arises from default external-entity resolution
- Reading local files and reaching internal services through entities
- Exploiting blind XXE with out-of-band techniques
- Getting a payload past parser configuration and input handling
- Finding the vulnerable parser call in the source
- Building an exploit that retrieves data or pivots onward

Module 23 — RCE via Database Functions

Reaching command execution through functionality the database itself provides.

- Database features and functions that can touch the operating system
- Moving from SQL access to host command execution
- Meeting the privilege and configuration prerequisites on the target
- Combining this route with an injection foothold
- Confirming and stabilising the resulting access
- Folding the escalation into the end-to-end exploit

Module 24 — OS Command Injection via WebSockets (Black Box)

Command injection reached over a WebSocket channel, worked without source access.

- How WebSocket messages carry data into server-side handlers
- Detecting command injection over a persistent WebSocket connection
- Crafting messages that inject into an OS command call
- The black-box approach when the handler code is not available
- Establishing command execution and a channel back
- Scripting the WebSocket interaction into a self-contained exploit

Who should take this

Experienced offensive practitioners: penetration testers who already handle web assessments, application security engineers, bug-bounty hunters, and developers who want to attack the kind of code they write. It is a specialist, senior-level course.

Be honest with yourself about the starting point. OSWE is hard, and it is hard in a specific way: it assumes you can already read source code in more than one language, follow data through an unfamiliar codebase, and write your own tooling in Python. You do not need to be a professional developer, but if reading code and scripting are not already comfortable, that gap will hurt more than any single technique on the list. OffSec recommends prior comfort with at least PEN-200-level material and general web fundamentals before starting. This is not an entry point into the field; it is a step up for people already in it.

The exam

The OSWE exam is set and awarded by OffSec. It is entirely practical: you are given remote access to target applications and have to find the flaws, exploit them, and prove it with working code.

- A proctored, hands-on assessment lasting 47 hours and 45 minutes against the target environment
- A further 24 hours afterwards to write and submit a professional report
- Scored out of 100 points across the exam's target machines, with 85 points required to pass
- For each target you must submit a functional proof-of-concept script that runs end to end without manual interaction
- The report must document the vulnerabilities, the exploitation steps and the exploit source code
- Automated exploitation tools, source-code scanners and mass vulnerability scanners are not permitted; a defined set of general tools is

The exam voucher is purchased separately from OffSec, along with any lab access, which is part of their product rather than ours. We prepare you for the exam; you book and sit it through OffSec directly.

How we deliver it

Classroom or online live, or at your office for a private group. Most of the time is spent reading real code and writing exploits against applications we have built or broken on purpose, so you practise the actual exam task — find the flaw in the source, then script the attack — rather than following a walkthrough to a known answer. Small cohorts, so everyone gets time reviewing code with an instructor. Exploit-scripting and report discipline are built in from the start, because the exam scores the script and the write-up, not just the finding.

Mapped to the OffSec WEB-300 syllabus and taught from Hacky Security's own curriculum and lab estate by instructors who do offensive web work for a living. This is preparation for the OSWE; the certification, voucher and any official lab access come from OffSec.

Sitting the official exam

Exam voucher sold separately. We buy it from the certification body on your behalf and send you the code — you book your own exam date. This course is Hacky Security's own preparation training. The certification itself is awarded by the certification body, not by us.

Investment

Priced per cohort. Tell us how many people and which delivery mode and we will send a quote the same working day.

Book this course

Tell us how many people, where, and when — we will come back with dates and a quote within one working day.

+91-8091900310 • info@hackysecurity.com

<https://hackysecurity.com/courses/oswe/>

CHANDIGARH • SHIMLA • DELHI • DUBAI • LONDON

Independent preparation training delivered from Hacky Security's own curriculum. Hacky Security Private Limited is not affiliated with or endorsed by the certification body named in this document. Certification names and codes are the trademarks of their respective owners and are used only to describe what this course prepares you for. Prices, durations and outlines are indicative and may change; the quotation we send you is the binding one. Issued 12 September 2026.