selfhack*

Report

Selfhack AI* Penetration Testing Report

SH010333
example.com

WWW.SELFHACK.FI

CONTENTS

1 Executive Summary

The following section summarizes the scope of the security assessment, the assessment results, and sets out the countermeasures recommended by **SelfHack AI**.

1.1 Scope

During the security assessment conducted for the example.comapplication.

SelfHack AI, https://www.example.com/ evaluated using the black box approach in web application **25-03-2025 | 27-03-2025** date.

.....

.....

No customers were harmed or interrupted during the tests.

1.2 Target

The goal of the assessment was to identify any possible vulnerabilities in **https://www.example.com/** and uncover common configuration issues.

In this process, attention was paid to the following components and questions:

All kinds of security problems that may be **client-based and server-based** on the application side were tested. These tests were performed on globally accepted security frameworks such as **OWASP Top Ten, NIST etc.**. The vulnerabilities tested include the following titles:

- IDOR (Insecure Direct Object Reference) vulnerabilities
- Business logic vulnerabilities
- SQL injection
- XSS (Cross-Site Scripting)
- CSRF (Cross-Site Request Forgery)
- Weak authentication mechanisms
- Breakable session management
- File upload vulnerabilities
- Insecure APIs
- Privilege escalation vulnerabilities
- Insecure data storage
- Insecure data transmission (HTTP usage, SSL/TLS configuration issues)
- Open Redirect
- Local file inclusion

- File Inclusion (RFI)
- XML External Entity (XXE) vulnerability
- JSON injection
- Clickjacking attacks
- Denial of service (DoS) and distributed denial of service (DDoS) attacks
- Weak password policies and crackable passwords
- Misconfigured security headers
- CORS (Cross-Origin Resource Sharing) configuration vulnerabilities
- Cache vulnerabilities
- Patch deficiencies and use of outdated software components
- Command Injection
- LDAP injection
- Insecure file upload
- Security practices incompatible with desktop or mobile platforms
- Content security policy (CSP) deficiencies
- Content manipulation and data integrity issues
- Unspecified vulnerabilities that can be discovered with fuzzing tests
- Timing attacks
- Side-channel attacks

The scope of this security assessment is not to determine or verify the legal requirements and necessary information obligations in case the software is used by a real corporate structure or individuals. This **assessment is purely a pre-test to prevent hacking by malicious attackers in field operations**.

1.3 Executive Summary of the AI Powered Penetration Test Report for C-Level Management

The SelfHack AI-powered penetration test conducted on www.example.com uncovered multiple security weaknesses across application layers, exposing the platform to operational, reputational, and regulatory risks. These findings indicate a need for immediate remediation, enhanced input validation, and a shift toward secure-by-design principles.

🔢 **1. Parameter Type Mismatch Causes Oracle Leak High (CVSS 8.3)**
Submitting string input (e.g., 20-SELFHACKAI) to an integer parameter crashes the backend and exposes raw Oracle exception stack traces.

💥 Impact:

- System Instability: Improper inputs can disrupt normal operation.
- Sensitive Logic Disclosure: Reveals database package paths and error lines.
- Development Oversight: Breaks secure coding standards under ISO 27001 A.14.2.1.

🛡️ Mitigation: Implement strict input validation per parameter type. Gracefully handle type errors and never expose raw exceptions to the frontend.

🧠 2. IP Ban Bypass via X-Forwarded-For Header High (CVSS 8.0)

Attackers can add the header X-Forwarded-For: 127.0.0.1 to spoof their IP address and bypass IP-based access restrictions.

💥 Impact:

- Access Control Bypass: Banned users regain access.
- Security Trust Erosion: Perimeter rules become ineffective.
- Compliance Breach: Violates ISO 27001 A.9.1.2 (Access restrictions).

🛡️ Mitigation: Only trust IP headers from known reverse proxies. Strip or validate X-Forwarded-For headers at the edge (e.g., Cloudflare, NGINX).

🔓 3. CAPTCHA Bypass via Client-Side Only Control Medium (CVSS 6.8)

The CAPTCHA on the tariff form is validated only in the browser (client-side). Requests without CAPTCHA can still be processed successfully by the server.

💥 Impact:

- Bot Abuse Risk: Attackers can automate tariff calculations.
- Resource Consumption: Leads to potential denial of service or system overload.
- Compliance Gap: Violates secure processing under GDPR (Article 32).

🛡️ Mitigation: Implement server-side CAPTCHA verification using Google reCAPTCHA or similar. Reject any request without a valid token.

4. .NET Version Disclosure via 404 Error Page Low (CVSS 5.1)

Accessing a non-existent page returns a 404 with verbose information, including .NET Framework Version: 4.0.30319 and ASP.NET Version: 4.8.4770.0.

Impact:

- Technology Fingerprinting: Attackers can look up vulnerabilities specific to disclosed versions.
- Increased Risk Surface: Helps target zero-days or known CVEs.
- Security Misconfiguration: Violates OWASP Top 10 A06 (Vulnerable Components).

Mitigation: Disable detailed error messages in production. Replace with generic error pages. Remove technology stack banners from responses.

Compliance Mapping Summary

Regulation / Standard	Affected Control	Risk Level
GDPR Article 32	Data processing without adequate control	High
ISO 27001 A.12.6.2	Error handling and information leakage	High
ISO 27001 A.14.2.1	Secure development practices	High
OWASP Top 10 A06, A09	Usage of outdated components, monitoring	Medium

✅ Recommended Next Steps

🛑 Disable Technical Error Messages in Production

1. Prevent the application from exposing detailed backend errors (e.g., database packages, line numbers) to users.
2. C-Level Relevance: Revealing internal system details publicly increases the risk of targeted cyberattacks and data breaches.

🧱 Enforce Server-Side CAPTCHA Validation

1. Ensure CAPTCHA checks are enforced on the backend, not just in the browser.
2. C-Level Relevance: Without proper validation, bots can bypass protection and overload business-critical services.

🛡️ Strengthen IP Access Controls

1. Reject spoofed IP addresses sent via headers like X-Forwarded-For.
2. C-Level Relevance: Malicious users can bypass bans and gain unauthorized access, undermining network defenses.

🧪 Implement Input Validation & Type Safety

1. Validate all incoming parameters to ensure they are of the correct type and format before processing.
2. C-Level Relevance: Prevents attackers from crashing or manipulating backend logic, reducing operational risk.

🚫 Hide Technology Stack & Framework Versions

- Remove version information (e.g., .NET Framework, ASP.NET) from error pages.
- C-Level Relevance: Publicly disclosing software versions helps attackers identify vulnerabilities to exploit.

🔍 Launch a Full Secure Code Review

- Conduct a comprehensive review of application code with a security focus.
- C-Level Relevance: A small investment in code review can prevent high-cost data breaches and regulatory fines.

📚 Align with GDPR & ISO 27001 Compliance

- Ensure all actions taken support GDPR Article 32 and ISO 27001 controls (A.12.6.2, A.14.2.1).
- C-Level Relevance: Strengthening compliance reduces legal exposure and demonstrates good governance.

⏱️ Recommended Timeline

Timeline	Actions
Immediate (0–3 days)	Mask error messages, enforce CAPTCHA, block spoofed IPs
Short Term (within 2 weeks)	Add input validation, deploy secure error pages
Mid Term (within 30 days)	Perform secure code review, document compliance status

🚨1.3.1 Worst Case Scenario:

💣 1. Targeted Cyberattack Leading to Data Breach

Exposed error messages and backend version leaks provide attackers with a blueprint of the system. With this knowledge, they could exploit known vulnerabilities in the .NET Framework or Oracle database, leading to unauthorized access to internal data or systems.

Business Impact: Compromise of sensitive operational data, legal liabilities under GDPR, and potential regulatory sanctions.

🤖 2. Automated Abuse & Service Disruption

Due to the missing server-side CAPTCHA validation, bots can flood key application endpoints such as the tariff calculator. Combined with IP spoofing bypass, attackers could generate massive automated traffic, resulting in downtime or degraded service for legitimate users.

Operational Risk: Unavailable services, damage to partner trust, and customer dissatisfaction.

🕵️ 3. Escalation to Internal Infrastructure

Attackers using verbose error outputs (e.g., ORA-01722, stack traces) may discover business logic flaws and elevate access beyond public interfaces, potentially compromising internal systems or manipulating backend operations.

Security Impact: Privilege escalation, loss of system integrity, and reputational damage.

📉 4. Regulatory Penalties & Reputational Loss

Failure to comply with GDPR Article 32 or ISO 27001 due to these misconfigurations and information leaks could trigger external audits, fines, and public trust issues.

Financial Risk: GDPR penalties up to €20M or 4% of global turnover, in addition to remediation costs and PR crisis management.

⚠ Executive Takeaway

The current weaknesses are not just technical; they open doors to business disruption, legal exposure, and long-term brand damage. In today's threat landscape, attackers don't need to break in—they just wait for mistakes to expose themselves.

📊 1.3.2 Business Impact

🏛️ 1. Operational Disruption

Critical endpoints like the tariff calculator can be abused by automated bots due to CAPTCHA weaknesses. Combined with IP spoofing, this could result in service slowdowns or outages, particularly during peak business periods.

Impact: Loss of service availability, disrupted logistics workflows, and partner dissatisfaction.

📉 2. Reputational Damage

Detailed error messages, version leaks, and missing controls create the perception of a poorly protected digital infrastructure. In the event of public discovery or breach, customer and partner trust may be significantly eroded.

Impact: Damaged credibility, negative media exposure, and increased scrutiny from stakeholders.

⚖️ 3. Regulatory & Legal Exposure

Current misconfigurations (e.g., verbose error handling, lack of access controls) may violate GDPR Article 32 and ISO 27001:2013 clauses related to secure development and information leakage.

Impact: Potential investigations, fines, and legal consequences tied to inadequate security controls.

💰 4. Financial Risk

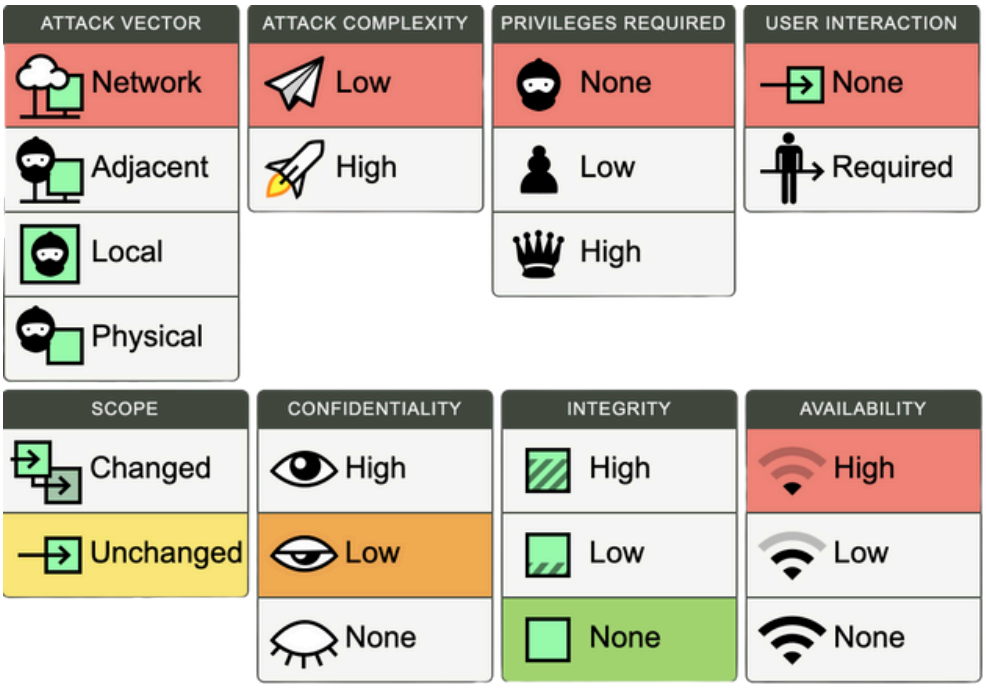
A successful exploit or sustained bot abuse could lead to system downtime, emergency remediation costs, legal counsel involvement, and possible fines — all of which translate into unplanned financial losses.

Impact: Business continuity costs, contract penalties, regulatory fines, and IT recovery spending.

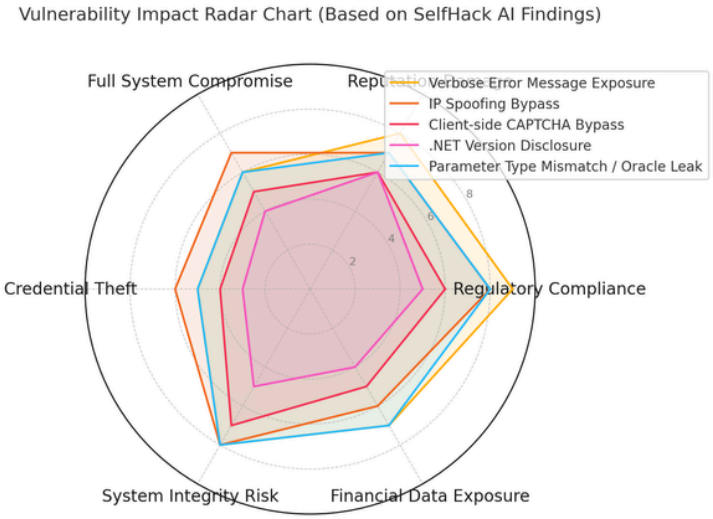
🧠 Executive Perspective

These are not theoretical risks. Each vulnerability represents a possible entry point into the organization's digital core — one that malicious actors can exploit with minimal effort, but maximum consequence. Addressing them is not only a security issue, it's a business resilience priority.

1.3.3 Technical Risk Assessment



The CVSS scores in the finding cards showing the vulnerabilities were calculated according to this matrix.



Above, you can see a radar chart of findings based on CVSS scores. The chart allows you to visually compare the risk level and CVSS score of each vulnerability. This allows you to more easily analyze which vulnerabilities stand out the most and which areas need to be prioritized for improvement.

1.3.4 Solution Proposal for the Manager

As a result of our work, a number of important technical solution suggestions are presented based on the findings revealed by this penetration test. These suggestions are prepared in a way that administrators can evaluate more directly and understandably. Detailed findings and technical suggestions will be presented in the form of finding cards on the following pages.

- **Secure Code Development Training:** The software team should be given comprehensive training on secure code development. Thanks to this training, more secure designs will be created in all future updates, thus preventing potential material and moral damages. After the training, the application should not be released without performing security tests for each feature developed. This approach will create a culture that keeps security at the center of the development process.

- **Creating a Security Life Cycle (SDLC):** A software development life cycle (SDLC) should be created. Each development process should be subjected to security tests in line with DevSecOps principles. Thus, each new feature and update should not be released without being evaluated and tested in terms of security.

- **Use of Web Security Products:** Security products such as a Web Application Firewall (WAF) should be positioned on the web application. Such solutions can provide protection against some assumed vulnerabilities. However, it may be insufficient against complex vulnerabilities such as business logic. However, thanks to its logging capabilities, it will be possible to detect attackers even if an attack occurs.

2 Approach

The following section outlines **Self Hack AI's security assessment** approach.

2.1 Testing Method

SelfHack AI performs security assessments to check the security of an application or all application components. The tools, methods, and techniques used by SelfHack AI fall into three categories:

- Widely known in the computer security and **"hacker"** communities.
- Internal tools developed to bypass the limitations of the conventional hacker toolkit.
- Expert knowledge; Security consultants look for vulnerabilities that may not be discovered using automated tools.

2.2 Scope and Timeline

The security assessment was conducted from **25-03-2025 | 27-03-2025** The purpose of this test is to test https://www.example.com/ applications for any vulnerabilities and common configuration issues.

Scope Url: Https://www.example.com/

The test accounts used in the application were created so that they could be registered and created by security consultants when necessary.

No checks were made to intentionally compromise the availability of the services.

2.3 Test Classes

Systems in scope have been tested against the following test classes, where applicable:

- **Tested:** This attack vector has been tested by SelfHack as part of this security assessment.

- **Exploitable:** This attack vector has been successfully exploited during this security assessment.

Main Attack Pattern	Tested	Potentially Exploitable
Listing Server Contents	YES	YES
Abusing Default Accounts	YES	NO
Listing User Accounts	YES	NO
Exploiting Dangerous Protocol Methods	YES	NO
Misusing Improper Access Permissions	YES	YES
Abusing Unprotected Functions	YES	NO
Collecting Internal Information	YES	NO
Password Spraying	YES	NO
Reading Unencrypted Sensitive Data	YES	YES
Exploiting Known Application Vulnerabilities	YES	NO
Bypassing Authentication	YES	NO
Accessing Protected Functions	YES	NO
Accessing Protected Resources	YES	NO
Abusing File Extension Processing	YES	NO
Collecting Information from Code Comments	YES	NO
Collecting Information from System and Error Messages	YES	YES
Reading Old, Backup, and Unreferenced Files	YES	NO
Cross-Site Request Forgery (XSRF/CSRF)	YES	NO
HTML Injection / Cross-Site Scripting (XSS)	YES	NO
HTTP Response Splitting / Header Injection	YES	NO
Clickjacking	YES	NO
Session Fixation	YES	NO
Accessing the File System	YES	NO
Code Injection	YES	NO
Command Injection	YES	NO
Format String Injection	YES	NO
IMAP/SMTP Injection	YES	NO
LDAP Injection	YES	NO
ORM Injection	YES	NO
Buffer Overflow	YES	NO
Path Traversal	YES	NO
SQL Injection	YES	NO
SSI Injection	YES	NO
Server-Side Request Forgery (SSRF)	YES	NO
XML Injection	YES	NO
XPath Injection	YES	NO
Listing Session Identifiers	YES	NO
Abusing Session State Issues	YES	NO
Manipulating Data within Client-Side Application	YES	YES
Reading Sensitive Data within Client-Side Application	YES	YES
Abusing Sample Applications	YES	NO
Uploading Malicious Files	YES	NO
Password Cracking	YES	NO
Misusing Weak Random Number Generator (RNG)	YES	NO
Abusing Improper Resource Allocation	YES	YES
Locking Customer Accounts	YES	NO
Unknown Attack Pattern	YES	YES
Business Logic	YES	NO

2.4 Disclaimer ⚠

This report has been generated by SelfHack AI as part of a professional, AI-assisted penetration testing engagement. It is intended solely for the internal use of the authorized stakeholders of the target organization.
All findings, analyses, and recommendations presented in this document are based on information available at the time of testing and within the agreed testing scope. While reasonable care has been taken to identify vulnerabilities, no security assessment can guarantee the complete absence of risk.

SelfHack AI and its affiliates disclaim any liability for damages arising directly or indirectly from the use or misuse of this report. This includes but is not limited to business interruption, data loss, or third-party compromise.

The information herein may contain sensitive technical and security-related content. Unauthorized disclosure, reproduction, or distribution of this report, in whole or in part, is strictly prohibited.

All tests were conducted under controlled conditions, with no intent to cause service disruption or data manipulation. It is the responsibility of the organization to validate, prioritize, and implement the recommended mitigations in accordance with their internal risk management policies and compliance requirements (e.g., ISO 27001, GDPR).

3 Summary of Vulnerability Risks and Number

This section contains all the vulnerabilities identified in **https://www.example.com/** application.

Risk assessment	Vulnerabilities
Critical	0
High	2
Medium	1
Low	1
Total	4

Risks Icons




4 Detailed Analysis and Findings Cards

This section provides a detailed summary of the attacks and the vulnerabilities detected.

4.1 Verbose Error Exposure on Parameter Type Mismatch

High
CVSS 8.3

Finding Description:

The endpoint *https://www.example.com/mip/TariffCalc* throws verbose Oracle database error messages when an unexpected data type is passed into the ard[size] parameter. This parameter expects an integer. However, when a string such as 20-SELFHACKAI is submitted, the application returns detailed backend errors, exposing sensitive internal mechanisms.

The error response includes:

```
ORA-01722: invalid number
ORA-06512: at "MOBILEAPP.XXMIP_MOBILE_OPERATION_WS_PKG", line 399
ORA-06512: at line 1
```

This level of detail reveals the backend package name, exact code location (line number), and confirms the usage of Oracle DB—providing attackers valuable insight into the application's internals.

Affected URL:
https://www.example.com/mip/TariffCalc

Analysis of Vulnerability:

- Verbose Error Disclosure: The system leaks full Oracle errors to the client.
- Missing Input Type Validation: The server does not validate the input type before using it in SQL queries.
- Exposure of Internal Logic: Package names, function calls, and line numbers are disclosed.
- Precursor to SQL Injection or Reconnaissance: Attackers could map out backend logic for further exploitation.

🧨 Proof of Concept (PoC):

- An attacker sends a malformed input, e.g. ard[size]=20-SELFHACKAI.
- The backend attempts to parse the value as an integer and fails.
- The Oracle engine throws an ORA-01722 exception.
- This is returned to the frontend verbatim, leaking:
  - DBMS type (Oracle)
  - Package and procedure name
  - Code line number
  - Internal structure layout (good for reconnaissance)

💥Potential Consequences of the Vulnerability:

- Information Disclosure:
  - Exposes internals such as DB packages, line numbers, and platform architecture.
- Attack Surface Expansion:
  - Allows attackers to test more injection vectors or backend logic pathways.
- Security Misconfiguration:
  - Reflects poor error sanitization and absence of centralized exception handling.
- Compliance Violations:
  - Violates GDPR (Article 32), ISO 27001 (A.14.2.1 – Secure development practices).

Vulnerability Screenshots:

4.1 Verbose Error Exposure on Parameter Type Mismatch

Solution Proposal:

✅ Code-Level Mitigation:

🛠️Type Validation:

```
try {
    int size = Integer.parseInt(request.getParameter("ard[size]"));
} catch (NumberFormatException e) {
    return ResponseEntity.badRequest().body("Invalid input format.");
}
```

🛠️Sanitized Exception Handling:

```
catch (SQLException ex) {
    logger.error("Database error occurred.", ex); // Internal log only
    return ResponseEntity.status(500).body("Internal server error.");
}
```

🛠️Avoid exposing database stack traces in user-facing responses.

🛡️ Server-Level Mitigation:

- Global Error Handling Middleware:

Implement global exception handlers in frameworks like Spring Boot, .NET, or Node.js to mask technical errors from end users.

- Web Application Firewall (WAF):

Configure the WAF to detect and block verbose database errors (e.g., patterns like ORA- or SQLSTATE in HTTP responses).

- Custom Error Pages:

Return consistent and non-descriptive error messages such as:

```
"Something went wrong. Please try again later."
```

📅 Remediation Timeline:

Priority Level	Action Items	Deadline	Responsible Party
🔴 **Critical (Verbose DB Error Exposure)**	- Implement input type validation - Add centralized exception handling - Mask database error responses - Configure WAF rules to detect/leak patterns	**Within 3 Days**	Backend Developers, DevSecOps
🟠 **High** (Security Header & Logging Enhancements)	- Introduce generic error pages - Configure structured logging - Alert for "ORA-" patterns in logs	**Within 7 Days**	DevOps, Security Team
🟡 **Medium** (Compliance Validation & Regression Testing)	- Perform code review and regression testing - Validate fixes against GDPR/ISO 27001 standards	**Within 14 Days**	QA Team, Compliance Team
🟢 **Ongoing (Hardening & Awareness)**	- Introduce secure coding guidelines - Conduct developer security awareness training	**Monthly Review**	Security Leadership

4.2 IP Block Bypass via X-Forwarded-For Header Spoofing

High
CVSS 8.0

Finding Description:

The endpoint /tr on the domain *https://www.example.com implements IP-based blocking.* However, this mechanism can be bypassed by injecting a forged IP address using the X-Forwarded-For HTTP header. When this header is set to 127.0.0.1, the server returns a 200 OK response. If the header is removed, the server returns a 302 Found status and redirects the user to /Site/BlockedIPPage.html.

This reveals a critical misconfiguration where the backend relies on unsanitized X-Forwarded-For values to determine IP trust.

Affected URL:
https://www.example.com/tr

Analysis of Vulnerability:

- Trusting Untrusted Headers: The backend trusts the X-Forwarded-For value without verifying if the request originates from a legitimate proxy.
- IP-Based Logic Bypass: The IP restriction mechanism is ineffective due to lack of header validation or source IP verification.
- Abuse of Localhost Trust: Setting the value to 127.0.0.1 (loopback) exploits internal trust configurations, allowing blocked users full access.
- Security Misconfiguration: Reflects improper use of HTTP headers in access control mechanisms.

🧨 Proof of Concept (PoC):

- Attacker is blocked by IP and redirected to /Site/BlockedIPPage.html.
- They resend the same request with:

```
X-Forwarded-For: 127.0.0.1
```

- The server interprets this as a trusted internal request and returns full access (HTTP 200 OK).
- The attacker regains access despite being blacklisted.

💥Potential Consequences of the Vulnerability:

- Access Control Bypass:

Users who are banned or geo-blocked can still access protected areas.

- Abuse by Bots or Attackers:

Automated tools can spoof headers to circumvent IP-based rate limits or bans.

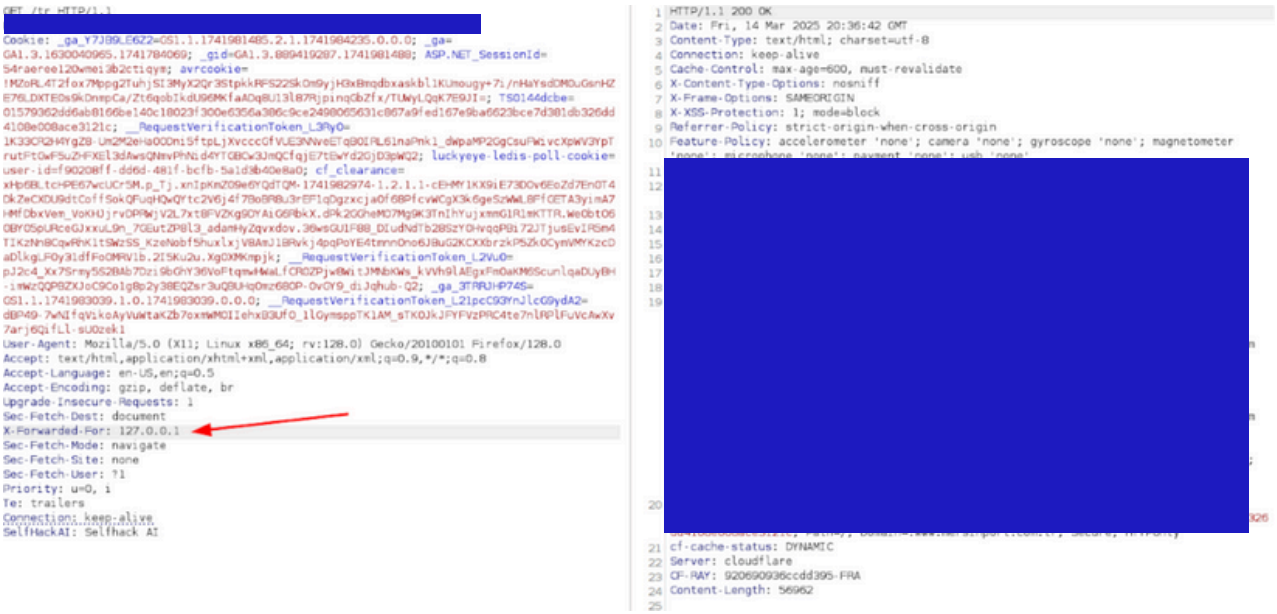
- Reputation Risk:

Security filters meant to deter bad actors are rendered useless.

- Compliance Violation:

Failing to enforce proper IP restrictions may breach GDPR/ISO 27001 access control obligations (A.9.1.2).

Vulnerability Screenshots:



4.2 IP Block Bypass via X-Forwarded-For Header Spoofing

Solution Proposal:

✅ Code-Level Mitigation:

🛠️ Sanitize X-Forwarded-For Header:

- Only honor this header when the request comes through trusted reverse proxies (e.g., Cloudflare, NGINX, AWS ALB).
- Use middleware or frameworks with strict IP source chaining policies.

🛠️ Use Real Client IPs from Trusted Proxies:

```
# Flask example using werkzeug's trusted proxy support
app.wsgi_app = ProxyFix(app.wsgi_app, x_for=1, trusted_proxies=["Cloudflare_IP_Ranges"])
```

🛠️ Reject Known Malicious Patterns:

- If X-Forwarded-For contains 127.0.0.1, localhost, or private IP ranges, reject the request.

🛡️ Server-Level Mitigation:

🛠️ Web Server / Reverse Proxy Config:

- Configure NGINX or Apache to strip X-Forwarded-For from incoming requests unless set by known upstreams.

Example (NGINX):

```
location / {
    if ($http_x_forwarded_for ~* "127\.0\.0\.1|localhost") {
        return 403;
    }
}
```

🛠️ Use Real IP Modules:

- In NGINX: Use ngx_http_realip_module to extract client IP only if the request comes from Cloudflare or another proxy.

🛠️ Cloudflare Configuration:

- Enable "True Client IP" or restore original visitor IP via Cloudflare headers (e.g., CF-Connecting-IP).

📅 Remediation Timeline:

Priority	Action	Deadline	Owner
🔴 Critical	Validate IPs from trusted proxies only	Within 3 Days	Backend / DevOps
🟠 High	Strip `X-Forwarded-For` if not from trusted source	Within 7 Days	Web Server Admin
🟡 Medium	Log and alert on suspicious IP spoofing attempts	Within 14 Days	Security Monitoring

4.3 CAPTCHA Bypass via Client-Side Only Validation

Medium
CVSS 6.8

Finding Description:

The endpoint *https://www.example.com/mip/TariffCalc/Home/Hesapla?Length=4* was tested and found to be vulnerable to CAPTCHA bypass. Although a CAPTCHA is visibly present on the form interface, the validation is only enforced on the client-side, and the backend does not verify CAPTCHA tokens before processing requests.

As a result, attackers can submit automated POST requests directly to the endpoint without solving the CAPTCHA, and still receive a valid response.

This undermines the core purpose of CAPTCHA, which is to prevent bots from abusing the form or system.

Affected URL:
https://www.example.com/mip/TariffCalc/Home/Hesapla

Analysis of Vulnerability:

- Missing Server-Side Validation: CAPTCHA token is never validated on the server.
- Bot Mitigation Failure: Malicious scripts and bots can flood the endpoint.
- Business Logic Flaw: Assumes CAPTCHA presence equates to protection, but the server-side lacks enforcement.

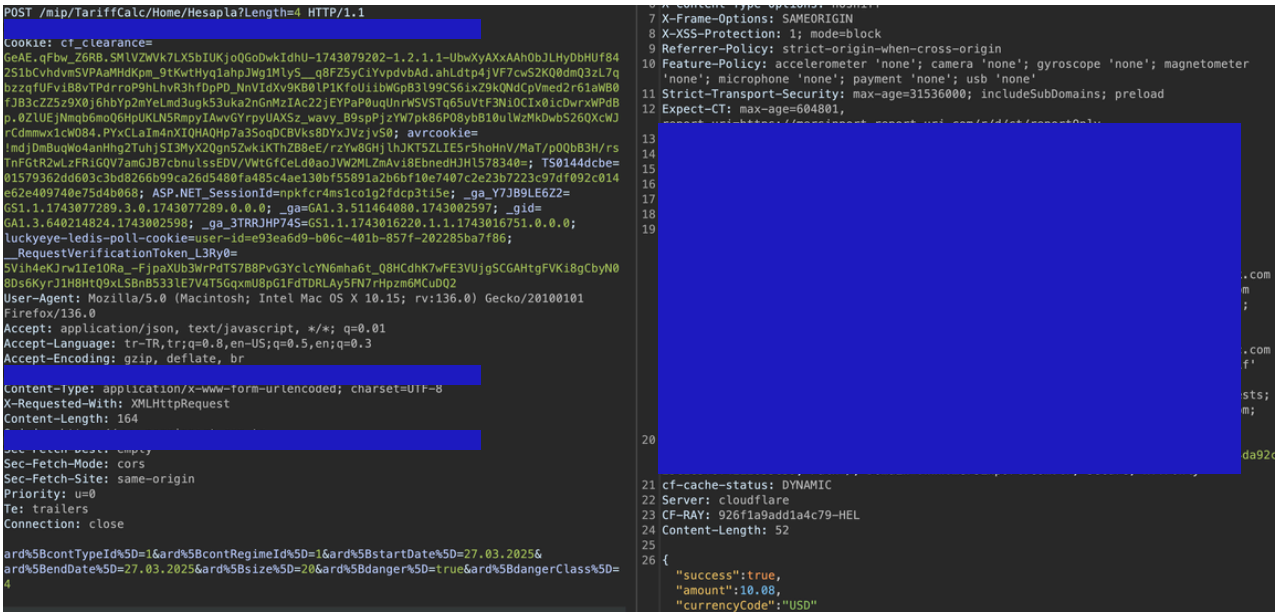
🧨 Proof of Concept (PoC):

- Attacker analyzes form submission in browser and extracts request structure.
- They replicate the request without the CAPTCHA payload using a tool like Burp Suite, curl, or a script.
- The request is accepted and processed by the server.
- No CAPTCHA check is performed; the attacker can now automate tariff calculations or spam the service.

💥Potential Consequences of the Vulnerability:

- Abuse of Business Logic:
  - Automated abuse of tariff calculations can lead to service degradation or data scraping.
- Denial of Service Vector:
  - Attackers can use bots to send high volumes of requests, causing slowdowns.
- Trust & Reputation Loss:
  - Users may lose trust in the service's integrity if automation goes uncontrolled.

Vulnerability Screenshots:



4.3 CAPTCHA Bypass via Client-Side Only Validation

Solution Proposal:

✅ Code-Level Mitigation:

🛠️ Enforce CAPTCHA Token Verification on Server:

- Backend should receive a CAPTCHA token (e.g., reCAPTCHA v2/v3) and verify it via Google/Cloudflare API:

```python
import requests

def verify_captcha(token):
    response = requests.post(
        "https://www.google.com/recaptcha/api/siteverify",
        data={'secret': 'your-secret-key', 'response': token}
    )
    return response.json().get('success', False)
```

🛠️ Token Expiry and Replay Protection:

- Tokens must be one-time use only, validated within a short TTL (Time-To-Live).

🛠️ Mandatory Token Presence:

- Reject all requests that do not include a valid CAPTCHA field in the payload.

🛡️ Server-Level Mitigation:

🛠️ Rate Limiting:

- Enforce IP-based throttling for endpoints such as /TariffCalc/Home/Hesapla.

🛠️ WAF Integration:

- Configure rules to detect rapid repeated POST requests lacking CAPTCHA tokens.
- Block behavior resembling automated scripts.

🛠️ Security Headers:

- Add HTTP security headers to prevent frontend CAPTCHA scripts from being bypassed or manipulated.

📅 Remediation Timeline:

Priority	Action	Deadline	Owner
🔴 Critical	Implement server-side CAPTCHA validation	Within 3 Days	Backend Developers
🟠 High	Add rate-limiting and CAPTCHA enforcement checks	Within 5 Days	DevSecOps Team
🟡 Medium	WAF & bot traffic monitoring configuration	Within 10 Days	Security Engineer

4.4 Technology Stack & Version Disclosure via Verbose Error Pages

Low
CVSS 5.1

Finding Description:

The URL *https://www.example.com/mip/wbreport/Home/SELFHACKAI* triggers an *HTTP 404 Not Found error*. However, instead of serving a generic or custom error page, the server leaks detailed version information about the application's underlying technology stack.

The following internal details are disclosed in the response:

```
Microsoft .NET Framework Version: 4.0.30319
ASP.NET Version: 4.8.4770.0
```

This type of *verbose error message provides attackers* with valuable reconnaissance data, helping them tailor specific exploit payloads based on known vulnerabilities for those software versions.

Affected URL:
https://www.example.com/mip/wbreport/Home/SELFHACKAI

Analysis of Vulnerability:

- Information Disclosure: Exact .NET Framework and ASP.NET versions are exposed.
- Reconnaissance Enabler: Attackers can correlate the versions with known CVEs (e.g., RCE, deserialization, padding oracle vulnerabilities).
- Missing Custom Error Handling: Default error page is returned, which reveals backend metadata.

🧨 Proof of Concept (PoC):

- Attacker crafts a request to a non-existing endpoint (/wbreport/Home/SELFHACKAI).
- The server responds with a verbose 404 page containing full framework version details.
- Attacker uses this to search for public exploits targeting:

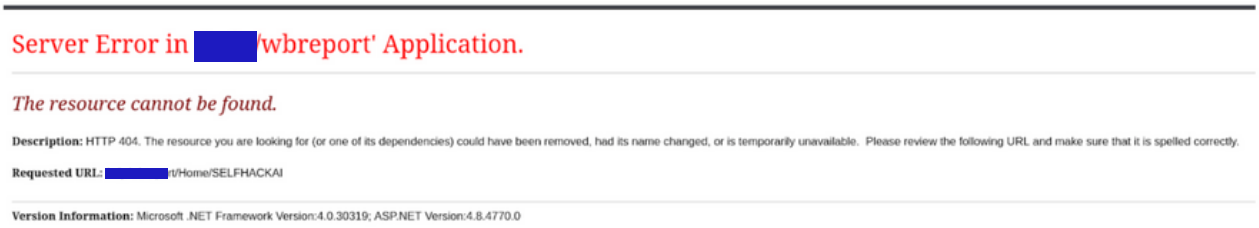
ASP.NET 4.8.4770.0
.NET Framework 4.0.30319

- Follow-up attacks may include deserialization, RCE, or DoS attempts aligned with the exposed software versions.

💥Potential Consequences of the Vulnerability:

- Exploit Development: Attackers can develop targeted payloads against specific .NET vulnerabilities.
- Increased Attack Surface: Future zero-days may target this specific version range.
- Compliance Violation: Violates best practices under OWASP Top 10 (A06:2021 – Vulnerable and Outdated Components).
- Reputation Risk: Internal information leakage is a sign of weak security hygiene.

Vulnerability Screenshots:



4.4 Technology Stack & Version Disclosure via Verbose Error Pages

Solution Proposal:

✅ Code-Level Mitigation:

🛠️ Custom Error Pages:

- Configure the web application to serve generic user-friendly error pages for all 404, 500, and 403 errors.
- ASP.NET (web.config):

```
<customErrors mode="On" defaultRedirect="~/Error/General">
  <error statusCode="404" redirect="~/Error/NotFound" />
  <error statusCode="500" redirect="~/Error/InternalServerError" />
</customErrors>
```

🛠️ Remove Stack and Version Info:

- Disable debug mode in production:

```
<compilation debug="false" />
```

🛡️ Server-Level Mitigation:

🛠️ IIS Configuration:

In web.config or IIS Manager, configure generic error pages and prevent detailed stack or version information from being returned.

🛠️ Header Stripping:

- Use a reverse proxy (e.g., NGINX, Cloudflare) to strip or suppress Server, X-AspNet-Version, and similar headers.

📅 Remediation Timeline:

Priority	Action	Deadline	Responsible Team
🔴 Critical	Suppress detailed error messages in production	Within 2 Days	DevOps / IT Ops
🟠 High	Add global error handling middleware/pages	Within 5 Days	Backend Development
🟡 Medium	Review all error-handling and routing logic	Within 10 Days	QA & Security Review

5 Version History

Version	Date	Status/Changes	Created By	Person in charge
1.0	27-03-2025	Last Version	Self Hack AI	Self Hack AI

Addition A.1 Risk Calculation

All discovered security risks were assessed with a risk score. The risk score is calculated from a risk matrix consisting of probability and severity. Probability describes the probability that an attacker will discover and exploit the vulnerability. Severity refers to the severity and impact of the vulnerability. Since severity affects risk more strongly than probability, it is included in the square of the equation. Probability and severity are multiplied to determine the risk score, which allows the assessment of the risks posed by a vulnerability.

To allow for a simple textual description of the risk, **the scores were classified into four main categories**:

Risk assessment	Risk Score
Critical	81-100
High	41-80
Medium	21-40
Low	1-10
Total	100

Addition A.2 Total Risk Score

To determine the total risk for a system, a network, or an entire company, the individual risks need to be summed up. However, simple addition is not valid because it does not match the actual behavior of individual vulnerabilities relative to each other. Two vulnerabilities with the same risk do not result in an overall risk that is twice as high. Therefore, the energy sum formula is used to calculate the total risk.

$$10 \lg (10^{R_1/10} + 10^{R_2/10} + \ldots + 10^{R_n/10}) = R_{total}$$

R ... Single Risk
R_{total} ... Total Risk

$$\begin{aligned} R_{\text{total}} &= 10 \cdot \log_{10}(10^{60/10} + 10^{55/10} + 10^{30/10} + 10^{10/10}) \\ &= 10 \cdot \log_{10}(10^6 + 10^{5.5} + 10^3 + 10^1) \\ &= 10 \cdot \log_{10}(1,000,000 + 316,228 + 1,000 + 10) \\ &= 10 \cdot \log_{10}(1,317,238) \\ &\approx 10 \cdot 6.12 = 61.2 \end{aligned}$$

According to the formula given and based on the specified vulnerability ratings, your total overall **risk score is approximately 61.2** This result reflects the combined effect of the risks and is considered a high score due to the weight of critical vulnerabilities.

However, it is risky to make a fractional estimate in penetration test reports. Therefore, the highest possible risk value is 100. If the total risk exceeds this value, it is fixed at 100 points. Your **approximate risk score is 61.2 , rounded to the risk value of 61.0 as stated** above.

selfhack*

WWW.SELFHACK.FI

info@selfhack.fi

Selfhack OY
Helsinki, Finland
Business ID: 3448051-6