

# Information Disclosure Vulnerability Notes

Eyad Islam El-Taher

November 17, 2025

## Introduction

**Information disclosure** (also known as information leakage) occurs when a website unintentionally reveals sensitive information to its users. This can include user data, business information, or technical details about the infrastructure. While some leaks may seem harmless, they can provide attackers with the missing pieces needed to construct serious attacks.

## Information Disclosure Vulnerability Happen When

- **Failure to remove internal content** from public content (e.g., developer comments in production)
- **Insecure configuration** of websites and related technologies (debug features enabled, default settings)
- **Flawed application design** that reveals information through behavioral differences
- **Verbose error messages** that disclose technical details or application logic
- **Backup files** are accessible in production environments
- **Version control data** is exposed publicly

## Information Disclosure Vulnerability Happen Where

- **Error messages and debug pages**
- **Developer comments** in HTML source code
- **Configuration files** (robots.txt, sitemap.xml, .git directories)
- **Backup and temporary files** (.bak, .tmp, files)
- **User account pages** and profile information
- **API responses** and headers
- **Directory listings** when no index file is present
- **Version control history** (.git, .svn directories)
- **Debugging endpoints** and diagnostic features

## Impact of Information Disclosure Vulnerability

- **Direct impact:**
  - Exposure of sensitive user data (PII, financial information)
  - Leakage of business-critical information
  - Compliance violations and legal consequences
  - Reputational damage to the organization
- **Indirect impact:**
  - Provides reconnaissance data for further attacks

- Reveals system architecture and technologies used
- Exposes API keys, credentials, or encryption keys
- Helps attackers understand application logic for exploitation
- **Severity factors:**
  - Sensitivity of the disclosed information
  - How easily the information can be used in further attacks
  - Whether the information is publicly accessible
  - The context and purpose of the application

## Examples of Information Disclosure

### 1- Verbose Error Messages

- **Description:** Applications that display detailed error messages revealing technical information
- **Examples:**

```
# Database errors revealing table structure
Error: Unknown column 'user' in 'field list'

# Stack traces revealing framework details
at com.example.Application.main(Application.java:42)

# File system paths
File not found: /var/www/html/admin/config.php
```

- **Practical Example:**

```
# Send unexpected data type to trigger error
GET /product?productId="example"

# Response reveals technology stack:
Apache Struts 2 2.3.31 - Stack Trace:
at org.apache.struts2.interceptor.
DebuggingInterceptor.intercept()
```

- **Impact:** Reveals application structure, database schema, file paths, and technology stack

### 2- Debugging Information Exposure

- **Description:** Debug features left enabled in production environments
- **Examples:**

```
# Debug parameters in URLs
https://site.com/admin?debug=true

# Stack traces with variable values
Variable 'api_key' = 'AKIAIOSFODNN7EXAMPLE'

# Session dumps
Session: {user_id: 123, role: 'admin', token: 'secret'}
```

- **Practical Example:**

```
# HTML comment reveals debug endpoint
<!-- Debug: /cgi-bin/phpinfo.php -->

# Accessing debug endpoint reveals:
SECRET_KEY = "p7f8x9r5c3v2b6n0m1l4k8j2h5g3f9d0"
Database credentials, API keys, environment variables
```

- **Impact:** Exposes sensitive runtime data, credentials, and application state

### 3- Backup File Access

- **Description:** Temporary or backup files accessible in web root
- **Examples:**

```
# Common backup file patterns
/sitemap.xml
/index.php.bak
/config.php.old
/database.sql~
/web.config.backup

# Source code disclosure
https://site.com/.git/config
https://site.com/.env.backup
```

- **Practical Example:**

```
# robots.txt reveals hidden directory
User-agent: *
Disallow: /backup/

# Access backup directory to find:
/backup/ProductTemplate.java.bak

# Source code contains hard-coded credentials:
connectionBuilder.password("p0stgr3s_db_p@ssw0rd")
```

- **Impact:** Source code exposure, credential leakage, application logic revelation

### 4- Directory Listings

- **Description:** Web servers configured to list directory contents
- **Examples:**

```
# Directory listing reveals sensitive files
[PARENTDIR] Parent Directory
[ ] backup.zip                2024-01-15 12:30 15M
[ ] database_dump.sql         2024-01-15 12:25 12M
[ ] admin_notes.txt           2024-01-15 12:20 1K
```

- **Practical Example:**

```
# Use Burp's content discovery
Engagement tools > Discover content

# Finds hidden directories:
/backup/ - contains source code backups
/logs/ - contains application logs
/tmp/ - contains temporary files
```

- **Impact:** Exposes file structure, backup files, configuration files, and sensitive documents

# Advanced Information Disclosure Scenarios

## 1- Version Control Exposure

- **Description:** Exposed .git, .svn, or other VCS directories
- **Exploitation:**

```
# Download entire .git directory
wget -r https://YOUR-LAB-ID.web-security-academy.net/.git/

# Extract commit history and source code
git log --oneline
git diff commit1 commit2

# Find sensitive data in history
git grep "password\\|api_key\\|secret"
```

- **Practical Example:**

```
# Browse to exposed .git directory
https://site.com/.git/

# Find commit with sensitive changes:
git log --oneline
"Remove admin password from config"

# View diff to see removed password:
- password: "super_secret_admin_123"
+ password: os.environ.get("ADMIN_PASSWORD")
```

- **Impact:** Full source code access, commit history, sensitive data in previous versions

## 2- Configuration File Leaks

- **Description:** Sensitive configuration files accessible via web
- **Examples:**

```
# Common configuration files
/.env
/config.json
/web.config
/application.properties
/config/database.yml

# Contents often include:
DB_PASSWORD=SuperSecret123!
API_KEY=AKIAI44QH8DHBEXAMPLE
SECRET_KEY=base64-encoded-secret
```

- **Practical Example:**

```
# robots.txt disclosure
User-agent: *
Disallow: /admin/
Disallow: /backup/
Disallow: /config/

# Leads to discovery of:
/config/database.properties
db.password=jdbc:postgresql://localhost:5432/mydb
```

- **Impact:** Database credentials, API keys, encryption keys, and system configuration

### 3- HTTP Method and Header Manipulation

- **Description:** Information disclosed through HTTP methods and headers
- **Examples:**

```
# TRACE method reveals internal headers
TRACE /admin HTTP/1.1

# Response shows added headers:
X-Custom-IP-Authorization: 123.45.67.89
X-Internal-Auth: Bearer secret-token

# OPTIONS reveals available methods
Allow: GET, POST, PUT, DELETE, DEBUG
```

- **Practical Example:**

```
# Admin panel restricted to local IPs
GET /admin --> "Admin panel only accessible locally"

# TRACE reveals IP authorization header:
TRACE /admin
X-Custom-IP-Authorization: 123.45.67.89

# Bypass with header manipulation:
X-Custom-IP-Authorization: 127.0.0.1
GET /admin --> Full admin access granted
```

- **Impact:** Technology fingerprinting, internal header disclosure, access control bypass

### 4- Application Logic Information Leakage

- **Description:** Information leaked through application behavior differences
- **Examples:**

```
# Different error messages
Valid user: "Invalid password"
Invalid user: "User not found"

# Timing differences
Valid API key: 1500ms response (database query)
Invalid API key: 50ms response (immediate rejection)
```

- **Practical Example:**

```
# Registration endpoint leaks existence
POST /register
{"email": "existing@site.com"}
--> "Email already registered"

POST /register
{"email": "new@site.com"}
--> "Registration email sent"

# Allows user enumeration and reconnaissance
```

- **Impact:** User enumeration, resource discovery, application logic mapping

# Testing Methodology

- **Manual Testing Approaches:**

- Use Burp's "Engagement tools" > "Find comments"
- Check for common backup file extensions (.bak, .old, )
- Test unexpected input types to trigger errors
- Examine all HTTP headers and responses carefully
- Use TRACE and OPTIONS HTTP methods

- **Automated Testing:**

- Burp Scanner for automatic detection
- Content discovery with Burp Intruder
- Custom wordlists for backup files
- Git repository scanning tools

- **Common Testing Steps:**

1. Check robots.txt and sitemap.xml
2. Search for developer comments in source
3. Test for verbose error messages
4. Look for exposed .git/.svn directories
5. Check for backup files
6. Examine HTTP headers and methods
7. Use content discovery tools

# Remediation and Prevention

- **Generic Error Messages:**

- Use generic error messages in production
- Avoid revealing stack traces or technical details
- Implement custom error pages
- Log detailed errors server-side only

- **Secure Configuration:**

- Disable debugging and diagnostic features in production
- Restrict HTTP methods (disable TRACE, DEBUG, etc.)
- Configure web servers to disable directory listings
- Remove default files and samples
- Use security headers (X-Content-Type-Options: nosniff)

- **Code and File Management:**

- Strip developer comments from production code
- Implement build processes to remove backup files
- Use .gitignore to exclude sensitive files from version control
- Regularly audit for exposed backup files
- Never deploy .git directories to production

- **Access Control:**

- Implement proper access controls for sensitive information
- Restrict access to configuration files and directories
- Use authentication and authorization for user data