

This content represents the latest contributions to the Web Security Testing Guide, and may frequently change.

Store

OWASP®

PROJECTS

CHAPTERS

EVENTS

ABOUT

Donate

Join

Search OWASP

Watch

271

Star

4,998

Store

Donate

WSTG - Latest

[Home](#) > [Latest](#) > [4-Web Application Security Testing](#) > [07-Input Validation Testing](#)

Testing for IMAP SMTP Injection

ID
WSTG-INPV-10

Summary

This threat affects all applications that communicate with mail servers (IMAP/SMTP), generally webmail applications. The aim of this test is to verify the capacity to inject arbitrary IMAP/SMTP commands into the mail servers, due to input data not being properly sanitized.

The IMAP/SMTP Injection technique is more effective if the mail server is not directly accessible from Internet. Where full communication with the backend mail server is possible, it is recommended to conduct direct testing.

An IMAP/SMTP Injection makes it possible to access a mail server which otherwise would not be

The OWASP® Foundation works to improve the security of software through its community-led open source software projects, hundreds of chapters worldwide, tens of thousands of members, and by hosting local and global conferences.

WSTG Contents

- 0. Foreword by Eoin Keary
- 1. Frontispiece
- 2. Introduction
- 2.1 The OWASP Testing Project
- 2.2 Principles of Testing
- 2.3 Testing Techniques Explained
- 2.4 Manual Inspections and Reviews
- 2.5 Threat Modeling
- 2.6 Source Code Review
- 2.7 Penetration Testing
- 2.8 The Need for a Balanced Approach
- 2.9 Deriving Security Test Requirements
- 2.10 Security Tests Integrated in Development and Testing

access a mail server which otherwise would not be directly accessible from the Internet. In some cases, these internal systems do not have the same level of infrastructure security and hardening that is applied to the front-end web servers. Therefore, mail server results may be more vulnerable to

attacks by end users (see the scheme presented in Figure 1).

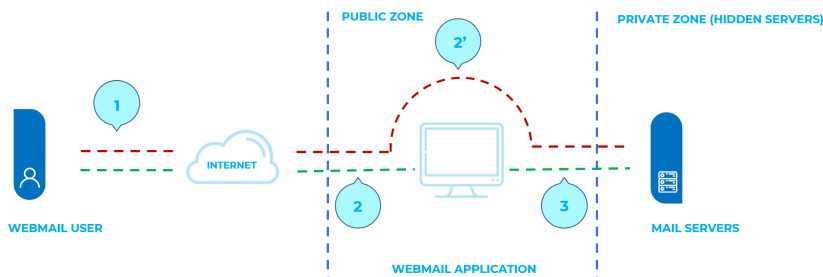


Figure 4.7.10-1: Communication with the mail servers using the IMAP/SMTP Injection technique

Figure 1 depicts the flow of traffic generally seen when using webmail technologies. Step 1 and 2 is the user interacting with the webmail client, whereas step 2 is the tester bypassing the webmail client and interacting with the back-end mail servers directly.

This technique allows a wide variety of actions and attacks. The possibilities depend on the type and scope of injection and the mail server technology being tested.

Some examples of attacks using the IMAP/SMTP Injection technique are:

- Exploitation of vulnerabilities in the IMAP/SMTP protocol
- Application restrictions evasion
- Anti-automation process evasion
- Information leaks
- Relay/SPAM

Development and testing

Workflows

2.11 Security Test Data Analysis and Reporting

3. The OWASP Testing Framework

3.1 The Web Security Testing Framework

3.2 Phase 1 Before Development Begins

3.3 Phase 2 During Definition and Design

3.4 Phase 3 During Development

3.5 Phase 4 During Deployment

3.6 Phase 5 During Maintenance and Operations

3.7 A Typical SDLC Testing Workflow

3.8 Penetration Testing Methodologies

4. Web Application Security Testing

4.0 Introduction and Objectives

4.1 Information Gathering

4.1.1 Conduct Search Engine Discovery Reconnaissance for Information Leakage

4.1.2 Fingerprint Web Server

4.1.3 Review Webserver Metafiles for Information Leakage

4.1.4 Enumerate Applications on Webserver

4.1.5 Review Webpage Content for Information Leakage

4.1.6 Identify Application Entry Points

4.1.7 Map Execution Paths

Test Objectives

- Identify IMAP/SMTP injection points.
- Understand the data flow and deployment structure of the system.
- Assess the injection impacts.

How to Test

Identifying Vulnerable Parameters

In order to detect vulnerable parameters, the tester has to analyze the application’s ability in handling input. Input validation testing requires the tester to send bogus, or malicious, requests to the server and analyse the response. In a secure application, the response should be an error with some corresponding action telling the client that something has gone wrong. In a vulnerable application, the malicious request may be processed by the back-end application that will answer with a HTTP 200 OK response message.

It is important to note that the requests being sent should match the technology being tested. Sending SQL injection strings for Microsoft SQL server when a MySQL server is being used will result in false positive responses. In this case, sending malicious IMAP commands is modus operandi since IMAP is the underlying protocol being tested.

IMAP special parameters that should be used are:

On the IMAP server	On the SMTP server
Authentication	Emissor email

Through Application

4.1.8 Fingerprint Web Application Framework

4.1.9 Fingerprint Web Application

4.1.10 Map Application Architecture

4.2 Configuration and Deployment Management Testing

4.2.1 Test Network Infrastructure Configuration

4.2.2 Test Application Platform Configuration

4.2.3 Test File Extensions Handling for Sensitive Information

4.2.4 Review Old Backup and Unreferenced Files for Sensitive Information

4.2.5 Enumerate Infrastructure and Application Admin Interfaces

4.2.6 Test HTTP Methods

4.2.7 Test HTTP Strict Transport Security

4.2.8 Test RIA Cross Domain Policy

4.2.9 Test File Permission

4.2.10 Test for Subdomain Takeover

4.2.11 Test Cloud Storage

4.2.12 Test for Content Security Policy

4.3 Identity Management Testing

4.3.1 Test Role Definitions

4.3.2 Test User Registration Process

4.3.3 Test Account Provisioning Process

operations with mail boxes (list, read, create, delete, rename)	On the IMAP server Destination email server

operations with messages (read, copy, move, delete)	Subject
Disconnection	Message body
	Attached files

In this example, the “mailbox” parameter is being tested by manipulating all requests with the parameter in:

http://<webmail>/src/read_body.php? mailbox=INBOX&passed_id=46106&startMessage=1

The following examples can be used.

- Assign a null value to the parameter:

http://<webmail>/src/read_body.php? mailbox=&passed_id=46106&startMessage=1

- Substitute the value with a random value:

http://<webmail>/src/read_body.php?

4.3.4 Testing for Account Enumeration and Guessable User Account
4.3.5 Testing for Weak or Unenforced Username Policy
4.4 Authentication Testing
4.4.1 Testing for Credentials Transported over an Encrypted Channel
4.4.2 Testing for Default Credentials
4.4.3 Testing for Weak Lock Out Mechanism
4.4.4 Testing for Bypassing Authentication Schema
4.4.5 Testing for Vulnerable Remember Password
4.4.6 Testing for Browser Cache Weaknesses
4.4.7 Testing for Weak Password Policy
4.4.8 Testing for Weak Security Question Answer
4.4.9 Testing for Weak Password Change or Reset Functionalities
4.4.10 Testing for Weaker Authentication in Alternative Channel
4.4.11 Testing Multi-Factor Authentication
4.5 Authorization Testing
4.5.1 Testing Directory Traversal File Include
4.5.2 Testing for Bypassing Authorization Schema
4.5.3 Testing for Privilege Escalation
4.5.4 Testing for Insecure Direct

`http://<webmail>/src/read_body.php?mailbox=NOTEXIST&passed_id=46106&startMessage=1`

- Add other values to the parameter:

`http://<webmail>/src/read_body.php?mailbox=INBOXPARAMETER2&passed_id=46106&startMessage=1`

- Add non-standard special characters (i.e.: \, ', ", @, #, !, |):

`http://<webmail>/src/read_body.php?mailbox=INBOX"&passed_id=46106&startMessage=1`

- Eliminate the parameter:

`http://<webmail>/src/read_body.php?passed_id=46106&startMessage=1`

The final result of the above testing gives the tester three possible situations: S1 - The application returns a error code/message S2 - The application does not return an error code/message, but it does not realize the requested operation S3 - The application does not return an error code/message and realizes the operation requested normally

Situations S1 and S2 represent successful IMAP/SMTP injection.

An attacker's aim is receiving the S1 response, as it is an indicator that the application is vulnerable to injection and further manipulation.

Let's suppose that a user retrieves the email headers using the following HTTP request:

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/10-Testing_for_IMAP_SMTP_... 5/11

Object References

4.5.5 Testing for OAuth

Weaknesses

4.5.5.1 Testing for OAuth

Authorization Server

Weaknesses

4.5.5.2 Testing for OAuth Client

Weaknesses

4.6 Session Management Testing

4.6.1 Testing for Session Management Schema

4.6.2 Testing for Cookies Attributes

4.6.3 Testing for Session Fixation

4.6.4 Testing for Exposed Session Variables

4.6.5 Testing for Cross Site Request Forgery

4.6.6 Testing for Logout Functionality

4.6.7 Testing Session Timeout

4.6.8 Testing for Session Puzzling

4.6.9 Testing for Session Hijacking

4.6.10 Testing JSON Web Tokens

4.7 Input Validation Testing

4.7.1 Testing for Reflected Cross Site Scripting

4.7.2 Testing for Stored Cross Site Scripting

4.7.3 Testing for HTTP Verb Tampering

4.7.4 Testing for HTTP Parameter Pollution

4.7.5 Testing for SQL Injection

...adding the following request:

```
http://<webmail>/src/view_header.php?
mailbox=INBOX&passed_id=46105&passed_e
nt_id=0
```

An attacker might modify the value of the parameter INBOX by injecting the character " (%22 using URL

encoding):

```
http://<webmail>/src/view_header.php?
mailbox=INBOX%22&passed_id=46105&passe
d_ent_id=0
```

In this case, the application answer may be:

```
ERROR: Bad or malformed request.
Query: SELECT "INBOX""
Server responded: Unexpected extra
arguments to Select
```

The situation S2 is harder to test successfully. The tester needs to use blind command injection in order to determine if the server is vulnerable.

On the other hand, the last situation (S3) is not relevant in this paragraph.

List of vulnerable parameters

- *Affected functionality*
- *Type of possible injection (IMAP/SMTP)*

Understanding the Data Flow and Deployment Structure of the Client

After identifying all vulnerable parameters (for example, passed_id), the tester needs to determine what level of injection is possible and then design a testing plan to further exploit the application

4.7.5.1 Testing for Oracle

4.7.5.2 Testing for MySQL

4.7.5.3 Testing for SQL Server

4.7.5.4 Testing PostgreSQL

4.7.5.5 Testing for MS Access

4.7.5.6 Testing for NoSQL Injection

4.7.5.7 Testing for ORM Injection

4.7.5.8 Testing for Client-side

4.7.6 Testing for LDAP Injection

4.7.7 Testing for XML Injection

4.7.8 Testing for SSI Injection

4.7.9 Testing for XPath Injection

4.7.10 Testing for IMAP SMTP Injection

4.7.11 Testing for Code Injection

4.7.11.1 Testing for File Inclusion

4.7.12 Testing for Command Injection

4.7.13 Testing for Format String Injection

4.7.14 Testing for Incubated Vulnerability

4.7.15 Testing for HTTP Splitting Smuggling

4.7.16 Testing for HTTP Incoming Requests

4.7.17 Testing for Host Header Injection

4.7.18 Testing for Server-side Template Injection

4.7.19 Testing for Server-Side Request Forgery

4.7.20 Testing for Mass Assignment

4.8 Testing for Error Handling

4.8.1 Testing for Improper Error Handling

4.8.2 Testing for Stack Traces

4.9 Testing for Weak

In this test case, we have detected that the application's `passed_id` parameter is vulnerable and is used in the following request:

```
http://<webmail>/src/read_body.php?
mailbox=INBOX&passed_id=46225&startMes
sage=1
```

Using the following test case (providing an alphabetical value when a numerical value is required):

```
http://<webmail>/src/read_body.php?
mailbox=INBOX&passed_id=test&startMess
age=1
```

will generate the following error message:

```
ERROR : Bad or malformed request.
Query: FETCH test:test BODY[HEADER]
Server responded: Error in IMAP command
received by server.
```

In this example, the error message returned the name of the executed command and the corresponding parameters.

In other situations, the error message (not controlled by the application) contains the name of the executed command, but reading the suitable [RFC](#) allows the tester to understand what other possible commands can be executed.

If the application does not return descriptive error messages, the tester needs to analyze the affected functionality to deduce all the possible commands (and parameters) associated with the above mentioned functionality. For example, if a vulnerable parameter has been detected in the create mailbox

Cryptography

4.9.1 Testing for Weak Transport Layer Security

4.9.2 Testing for Padding Oracle

4.9.3 Testing for Sensitive Information Sent via Unencrypted Channels

4.9.4 Testing for Weak Encryption

4.10 Business Logic Testing

4.10.0 Introduction to Business Logic

4.10.1 Test Business Logic Data Validation

4.10.2 Test Ability to Forge Requests

4.10.3 Test Integrity Checks

4.10.4 Test for Process Timing

4.10.5 Test Number of Times a Function Can Be Used Limits

4.10.6 Testing for the Circumvention of Work Flows

4.10.7 Test Defenses Against Application Misuse

4.10.8 Test Upload of Unexpected File Types

4.10.9 Test Upload of Malicious Files

4.10.10 Test Payment Functionality

4.11 Client-side Testing

4.11.1 Testing for DOM-Based Cross Site Scripting

4.11.1.1 Testing for Self DOM Based Cross-Site Scripting

4.11.2 Testing for JavaScript Execution

4.11.3 Testing for HTML Injection

4.11.4 Testing for Client-side

functionality, it is logical to assume that the affected IMAP command is CREATE. According to the RFC, the CREATE command accepts one parameter which specifies the name of the mailbox to create.

List of IMAP/SMTP commands affected

- *Type, value, and number of parameters expected by the affected IMAP/SMTP commands*

IMAP/SMTP Command Injection

Once the tester has identified vulnerable parameters and has analyzed the context in which they are executed, the next stage is exploiting the functionality.

This stage has two possible outcomes:

1. The injection is possible in an unauthenticated state: the affected functionality does not require the user to be authenticated. The injected (IMAP) commands available are limited to: CAPABILITY, NOOP, AUTHENTICATE, LOGIN, and LOGOUT.
2. The injection is only possible in an authenticated state: the successful exploitation requires the user to be fully authenticated before testing can continue.

In any case, the typical structure of an IMAP/SMTP Injection is as follows:

- Header: ending of the expected command;
- Body: injection of the new command;
- Footer: beginning of the expected command.

It is important to remember that, in order to execute an IMAP/SMTP command, the previous command

URL Redirect

4.11.5 Testing for CSS Injection

4.11.6 Testing for Client-side

Resource Manipulation

4.11.7 Testing Cross Origin

Resource Sharing

4.11.8 Testing for Cross Site Flashing

4.11.9 Testing for Clickjacking

4.11.10 Testing WebSockets

4.11.11 Testing Web Messaging

4.11.12 Testing Browser Storage

4.11.13 Testing for Cross Site Script Inclusion

4.11.14 Testing for Reverse Tabnabbing

4.12 API Testing

4.12.1 Testing GraphQL

5. Reporting

5.1 Reporting Structure

5.2 Naming Schemes

Appendix A. Testing Tools Resource

Appendix B. Suggested Reading

Appendix C. Fuzz Vectors

Appendix D. Encoded Injection

Appendix E. History

Appendix F. Leveraging Dev Tools

Upcoming OWASP Global Events

OWASP 2022 Global AppSec San Francisco

- November 14-18, 2022
Pacific Standard Time

which is terminated with the previous command.

must be terminated with the CRLF (%0d%0a) sequence.

Let's suppose that in the [Identifying vulnerable parameters](#) stage, the attacker detects that the

parameter message_id in the following request is vulnerable:

```
http://<webmail>/read_email.php?
message_id=4791
```

Let's suppose also that the outcome of the analysis performed in the stage 2 ("Understanding the data flow and deployment structure of the client") has identified the command and arguments associated with this parameter as:

```
FETCH 4791 BODY[HEADER]
```

In this scenario, the IMAP injection structure would be:

```
http://<webmail>/read_email.php?
message_id=4791 BODY[HEADER]%0d%0aV100
CAPABILITY%0d%0aV101 FETCH 4791
```

Which would generate the following commands:

```
???? FETCH 4791 BODY[HEADER]
V100 CAPABILITY
V101 FETCH 4791 BODY[HEADER]
```

where:

```
Header = 4791 BODY[HEADER]
Body = %0d%0aV100 CAPABILITY%0d%0a
Footer = V101 FETCH 4791
```

List of IMAP/SMTP commands affected

(PST)

[OWASP December Webinar](#)

- December 5-6, 2022
Eastern Standard Time
(EST)

[OWASP Global AppSec Dublin 2023](#)

- February 13-16, 2023

- *Arbitrary IMAP/SMTP command injection*

References

Whitepapers

- [RFC 0821 “Simple Mail Transfer Protocol”](#)
- [RFC 3501 “Internet Message Access Protocol - Version 4rev1”](#)
- [Vicente Aguilera Díaz: “MX Injection: Capturing and Exploiting Hidden Mail Servers”](#)

 [Edit on GitHub](#)

Spotlight: DevOps India Summit 2022



An Xellentro Event

DevOps India Summit is the largest DevOps and SRE Global Summit from India. While it started as a physical event in Bangalore, the pandemic has made it to shift to Virtual mode. 2022 is the 5th Year of the event. The theme for

Corporate Supporters





[Become a corporate supporter](#)

[HOME](#) [PROJECTS](#) [CHAPTERS](#) [EVENTS](#) [ABOUT](#) [PRIVACY](#)



[SITEMAP](#) [CONTACT](#)

OWASP, Open Web Application Security Project, and Global AppSec are registered trademarks and AppSec Days, AppSec California, AppSec Cali, SnowFROC, LASCON, and the OWASP logo are trademarks of the OWASP Foundation, Inc. Unless otherwise specified, all content on the site is Creative Commons Attribution-ShareAlike v4.0 and provided without warranty of service or accuracy. For more information, please refer to our [General Disclaimer](#). OWASP does not endorse or recommend commercial products or services, allowing our community to remain vendor neutral with the collective wisdom of the best minds in software security worldwide. Copyright 2022, OWASP Foundation, Inc.