Configuration and Deployment Management Testing

Test Network/Infrastructure Configuration (OTG-CONFIG-001)

Test Objective

The objective of this test is to perform a network scan on the target host to identify open TCP ports, the services running on those ports, and their respective versions. Discovering unnecessary open ports or outdated services can provide an attacker with valuable information and potential entry points into the system. This test uses the **nmap** utility, a standard tool for network exploration and security auditing.

Target Endpoint

The test is performed against the host machine running the application.

- **Host**: `localhost` (127.0.0.1)

Methodology

The testing script utilizes the `otg_001()` function to execute a network scan. It invokes `nmap` with specific flags to ensure a thorough and efficient scan. The key python snippet responsible for this is:

```
def otg_001():
    """OTG-CONFIG-001 - Network/Infrastructure Configuration"""
    banner("OTG-CONFIG-001 - Network/Infrastructure
Configuration")
    result = run(["nmap", "-sS", "-sV", "--top-ports", "1000",
"localhost"])
    log_run(result)
```

```
    return result
```

The command `nmap -sS -sV --top-ports 1000 localhost` performs the following actions:

- `-sS`: A TCP SYN scan (or "half-open" scan) which is stealthier and faster than a full TCP connect scan.
- `-sV`: Enables service and version detection to determine what software is running on the open ports.
- `--top-ports 1000`: Scans the 1,000 most common TCP ports.
- `localhost`: The target of the scan.

Step to Reproduce

1. Ensure that the `nmap` tool is installed on the testing machine.
2. Execute the command `nmap -sS -sV --top-ports 1000 localhost` from the command line.
3. The test output will list all open ports found within the top 1000. In this case, ports 135 (msrpc), 445 (microsoft-ds), 902 (vmware-auth), 912 (vmware-auth), and 8080 (http) were identified as open.
4. The presence of services like Windows RPC and VMware Auth on a web server could indicate an unnecessary exposure of services, increasing the attack surface.

Log Evidence


```
{
  "command": "nmap -sS -sV --top-ports 1000 localhost",
  "stdout": "Starting Nmap 7.95 ( https://nmap.org ) at 2025-
07-28 23:32 SE Asia Standard Time\nNmap scan report for
localhost (127.0.0.1)\nHost is up (0.00066s latency).\nOther
addresses for localhost (not scanned): ::1\nrDNS record for
127.0.0.1: frontend.test\nNot shown: 995 closed tcp ports
(reset)\nPORT     STATE SERVICE         VERSION\n135/tcp  open
msrpc           Microsoft Windows RPC\n445/tcp  open
microsoft-ds?\n902/tcp  open  ssl/vmware-auth VMware
Authentication Daemon 1.10 (Uses VNC, SOAP)\n912/tcp  open
vmware-auth     VMware Authentication Daemon 1.0 (Uses VNC,
```

```
SOAP)\n8080/tcp open  http            Apache httpd 2.4.25
((Debian))\nService Info: OS: Windows; CPE:
cpe:/o:microsoft:windows\n\nService detection performed. Please
report any incorrect results at https://nmap.org/submit/
.\nNmap done: 1 IP address (1 host up) scanned in 15.24
seconds",
  "stderr": "",
  "returncode": 0,
  "timeout": false
}
```


Test Application Platform Configuration (OTG-CONFIG-002)

Test Objective

The objective of this test is to gather detailed configuration information from the application's underlying platform, specifically the PHP interpreter and the Apache web server. Leaked configuration details, such as software versions, enabled modules, and internal paths, can provide an attacker with a roadmap for exploiting known vulnerabilities.

Target Endpoint

This test targets the application's runtime environment directly, using Docker to execute commands inside the running container.

- **Container Name**: `dvwa`

Methodology

The `otg_002()` function in the script runs commands inside the `dvwa` Docker container. This requires the **Docker** client to be installed and running. The function executes two commands to retrieve configuration data:

```
def otg_002():
```

```
    """OTG-CONFIG-002 - Application Platform Configuration"""
    banner("OTG-CONFIG-002 - Application Platform
Configuration")
    results = {}
    for cmd in [["docker", "exec", "dvwa", "php", "-i"],
["docker", "exec", "dvwa", "apache2ctl", "-S"]]:
        result = run(cmd)
        key = cmd[-2] + "_" + cmd[-1]
        results[key] = result
        log_run(result)
    return results
```

- `docker exec dvwa php -i`: This command executes the PHP command-line interpreter with the `-i` flag, which outputs the complete PHP configuration information (the equivalent of a `phpinfo()` page).
- `docker exec dvwa apache2ctl -S`: This command queries the Apache control interface to dump the parsed virtual host configuration, including the server root, document root, and log file locations.

Step to Reproduce

1. Ensure the DVWA application is running in a Docker container named `dvwa`.
2. From the host machine's terminal, run the command `docker exec dvwa php -i`.
3. Analyze the output for sensitive information. The log evidence reveals the exact PHP version (`7.0.30-0+deb9u1`), system information (`Linux ... WSL2`), and risky PHP directives like `expose_php = On` and `allow_url_fopen = On`.
4. Next, run `docker exec dvwa apache2ctl -S`.
5. Review the output for Apache configuration details, which confirms the user and group the server runs as (`www-data`) and the location of configuration files.

Log Evidence

(Note: Due to its length, the full phpinfo output is truncated for brevity)


```
{
  "php_-i": {
```

```
    "command": "docker exec dvwa php -i",
    "stdout": "phpinfo()\nPHP Version => 7.0.30-
0+deb9u1\n\nSystem => Linux d8d87a9dda55 5.15.153.1-microsoft-
standard-WSL2 #1 SMP Fri Mar 29 23:14:13 UTC 2024 x86_64\n...",
    "returncode": 0
  },
  "apache2ctl_-S": {
    "command": "docker exec dvwa apache2ctl -S",
    "stdout": "VirtualHost configuration:\n*:80
172.17.0.2 (/etc/apache2/sites-enabled/000-
default.conf:1)\nServerRoot: \"/etc/apache2\"\nMain
DocumentRoot: \"/var/www/html\"\nMain ErrorLog:
\"/var/log/apache2/error.log\"\nMutex watchdog-callback:
using_defaults\nMutex default: dir=\"/var/run/apache2/\"
mechanism=default \nMutex mpm-accept: using_defaults\nPidFile:
\"/var/run/apache2/apache2.pid\"\nDefine: DUMP_VHOSTS\nDefine:
DUMP_RUN_CFG\nUser: name=\"www-data\" id=33\nGroup: name=\"www-
data\" id=33",
    "stderr": "AH00558: apache2: Could not reliably determine
the server's fully qualified domain name, using 172.17.0.2. Set
the 'ServerName' directive globally to suppress this message",
    "returncode": 0
  }
}
```


Test File Extensions Handling for Sensitive Information (OTG-CONFIG-003)

Test Objective

This test aims to discover if the web server handles different file extensions in a way that might leak sensitive information. For example, a request for `config.php.bak` might be served as a plain text file, disclosing its source code. The test relies on the tool **ffuf** for fuzzing file extensions.

Target Endpoint

- **URL**: `http://localhost:8080/`

Methodology

The `otg_003()` function is designed to automate the discovery of misconfigured file handlers. It first attempts to download a list of common web extensions from the SecLists repository. It then uses the `ffuf` tool to request files with these extensions on the target server, looking for responses that indicate success (e.g., status code 200). However, the test was unable to run.

The script checks for the presence of `ffuf` using `shutil.which("ffuf")`. If the tool is not found in the system's PATH, the test is skipped.

Step to Reproduce

1. The script attempts to locate the `ffuf` executable on the testing machine.
2. In this case, the tool was not found.
3. The test immediately aborted and reported that `ffuf` was missing. No network requests were made to the target.

Log Evidence


```
{
  "status": "ffuf not found"
}
```


Review Old, Backup and Unreferenced Files for Sensitive Information (OTG-CONFIG-004)

Test Objective

The purpose of this test is to search for old, backup, and unreferenced files that may have been left on the server. Developers sometimes leave copies of files (e.g., `db_connect.php.old`, `archive.zip`) in the webroot, which can contain credentials,

configuration details, or source code. This test uses the **gobuster** tool for directory and file brute-forcing.

Target Endpoint

- **URL**: `http://localhost:8080`

Methodology

The `otg_004()` function attempts to find sensitive files by brute-forcing common filenames and backup extensions. It uses a wordlist from SecLists and specifies a list of extensions (`-x bak,old,orig,txt,swp,tmp`) for `gobuster` to try. Like the previous test, this one depended on a tool that was not available.

The script checks if `gobuster` is installed using `shutil.which("gobuster")`. Since the tool was not found, the test could not be executed.

Step to Reproduce

1. The script attempts to find the `gobuster` executable.
2. The tool was not present on the system.
3. The test was skipped, and the result indicates that the required tool was not found.

Log Evidence


```
{
  "status": "gobuster not found"
}
```


Enumerate Infrastructure and Application Admin Interfaces (OTG-CONFIG-005)

Test Objective

This test aims to identify administrative interfaces by inspecting the web application's pages for links that may lead to them. Discovering admin panels is a critical first step for an attacker trying to gain elevated privileges.

Target Endpoint

- **URL**: `http://localhost:8080/`

Methodology

The `otg_005()` function performs a simple but effective check. It sends an HTTP GET request to the application's root page using the Python **requests** library. It then uses a regular expression to parse the HTML response and extract all URLs found in `href` attributes. Finally, it filters this list to find any links containing the substring "admin".

```
def otg_005():
    """OTG-CONFIG-005 – Enumerate Admin Interfaces"""
    banner("OTG-CONFIG-005 – Enumerate Admin Interfaces")
    r = http_get("/")
    # ...
    links = re.findall(r'href=["\'](.*?)["\']', r.text, re.I)
    admin_like = [l for l in links if "admin" in l.lower()]
    print("Possible admin links:", admin_like)
    return admin_like
```

Step to Reproduce

1. Send an HTTP GET request to `http://localhost:8080/`.
2. Examine the HTML source code of the response.
3. Search for any anchor tags (`<a>`) whose `href` attribute contains the word "admin" (case-insensitive).
4. In this test, no such links were found on the homepage, resulting in an empty list.

Log Evidence

```
[]
```

```
================================================================
=======
  OTG-CONFIG-005 - Enumerate Admin Interfaces
================================================================
=======
Possible admin links: []
```

Test HTTP Methods (OTG-CONFIG-006)

Test Objective

The objective is to determine which HTTP methods (or verbs) are enabled on the web server. While `GET` and `POST` are standard, other methods like `PUT`, `DELETE`, `PATCH`, and `DEBUG` can pose security risks if enabled unnecessarily and could be used by an attacker to modify files on the server or gain information.

Target Endpoint

- **URL**: `http://localhost:8080`

Methodology

The `otg_006()` function iterates through a predefined list of HTTP methods. For each method, it uses the Python **requests** library to send a request to the target URL and records the server's response status code and content length. A status code of 200 (OK) indicates the method is likely enabled, whereas a 405 (Method Not Allowed) or 501 (Not Implemented) would indicate it is disabled.

```
def otg_006():
```

```
    """OTG-CONFIG-006 - Test HTTP Methods"""
    banner("OTG-CONFIG-006 - Test HTTP Methods")
    methods = ["GET", "POST", "PUT", "DELETE", "PATCH",
"TRACE", "TRACK", "CONNECT", "DEBUG"]
    results = {}
    for m in methods:
        resp = requests.request(m, DVWA_URL, timeout=5)
        results[m] = {"status": resp.status_code, "len":
len(resp.content)}
    return results
```

Step to Reproduce

1. Using a tool like `curl`, send a request for each HTTP method to the target URL. For example, to test the `PUT` method, run: `curl -X PUT -I http://localhost:8080`.
2. Observe the HTTP status code in the server's response.
3. The test results show that `PUT`, `DELETE`, `PATCH`, `TRACK`, and `DEBUG` all returned a status of `200 OK`. This indicates a misconfiguration, as these methods are enabled and should be disabled on a production server. The `TRACE` method correctly returned `405 Method Not Allowed`.

Log Evidence


```
{
  "GET": { "status": 200, "len": 1523 },
  "POST": { "status": 200, "len": 1523 },
  "PUT": { "status": 200, "len": 1523 },
  "DELETE": { "status": 200, "len": 1523 },
  "PATCH": { "status": 200, "len": 1523 },
  "TRACE": { "status": 405, "len": 300 },
  "TRACK": { "status": 200, "len": 1523 },
  "CONNECT": { "status": 400, "len": 302 },
  "DEBUG": { "status": 200, "len": 1523 }
```

```
}
```

Test HTTP Strict Transport Security (OTG-CONFIG-007)

Test Objective

This test checks for the presence of the `Strict-Transport-Security` (HSTS) HTTP response header. HSTS is a security mechanism that tells browsers to only communicate with the server using HTTPS, which helps prevent man-in-the-middle attacks such as protocol downgrades and cookie hijacking.

Target Endpoint

- **URL**: `http://localhost:8080/`

Methodology

The `otg_007()` function sends a standard HTTP GET request to the target URL. It then inspects the headers of the server's response to check for the existence of the "Strict-Transport-Security" header. The result indicates whether the header was found and, if so, its value.

```
def otg_007():
    """OTG-CONFIG-007 - HTTP Strict Transport Security"""
    banner("OTG-CONFIG-007 - HTTP Strict Transport Security")
    r = http_get("/")
    if r:
        hsts = r.headers.get("Strict-Transport-Security", None)
        print("HSTS header:", hsts)
        return {"present": hsts is not None, "value": hsts}
    # ...
```

Step to Reproduce

1. Use a tool like `curl` with the `-I` flag to view the response headers from the server: `curl -I http://localhost:8080`.
2. Scan the list of response headers for "Strict-Transport-Security".
3. The test results confirm that the header is not present ("HSTS header: None"), indicating that the application is not protected by HSTS.

Log Evidence

```
{
  "present": false,
  "value": null
}
```

```
================================================================
=======
  OTG-CONFIG-007 - HTTP Strict Transport Security
================================================================
=======
HSTS header: None
```

Test RIA cross domain policy (OTG-CONFIG-008)

Test Objective

The objective of this test is to check for the existence of cross-domain policy files used by Rich Internet Applications (RIAs) like Adobe Flash (`crossdomain.xml`) and Microsoft Silverlight (`clientaccesspolicy.xml`). A loosely configured policy file could allow malicious RIA applications from other domains to interact with the target application and exfiltrate data.

Target Endpoint

- **URL 1**: `http://localhost:8080/crossdomain.xml`
- **URL 2**: `http://localhost:8080/clientaccesspolicy.xml`

Methodology

The `otg_008()` function attempts to fetch the two policy files by making separate HTTP GET requests to their default locations. It reports the status of each request. If the files are not found (e.g., resulting in a 404 error), it is generally considered a secure state, as no policy is defined.

Step to Reproduce

1. Attempt to access `http://localhost:8080/crossdomain.xml` using `curl` or a web browser.
2. Attempt to access `http://localhost:8080/clientaccesspolicy.xml`.
3. The test shows that the requests for both files failed. This means the files are not present, which is the desired outcome if Flash or Silverlight applications are not intended to interact with the site.

Log Evidence


```
{
  "/crossdomain.xml": {
    "status": "error"
  },
  "/clientaccesspolicy.xml": {
    "status": "error"
  }
}
```


```
================================================================
=======
  OTG-CONFIG-008 - RIA Cross-Domain Policy
================================================================
=======
```

```
/crossdomain.xml: Request failed
/clientaccesspolicy.xml: Request failed
```