Sri Lanka Institute of Information Technology

Path Traversal

IE2062 - Web Security

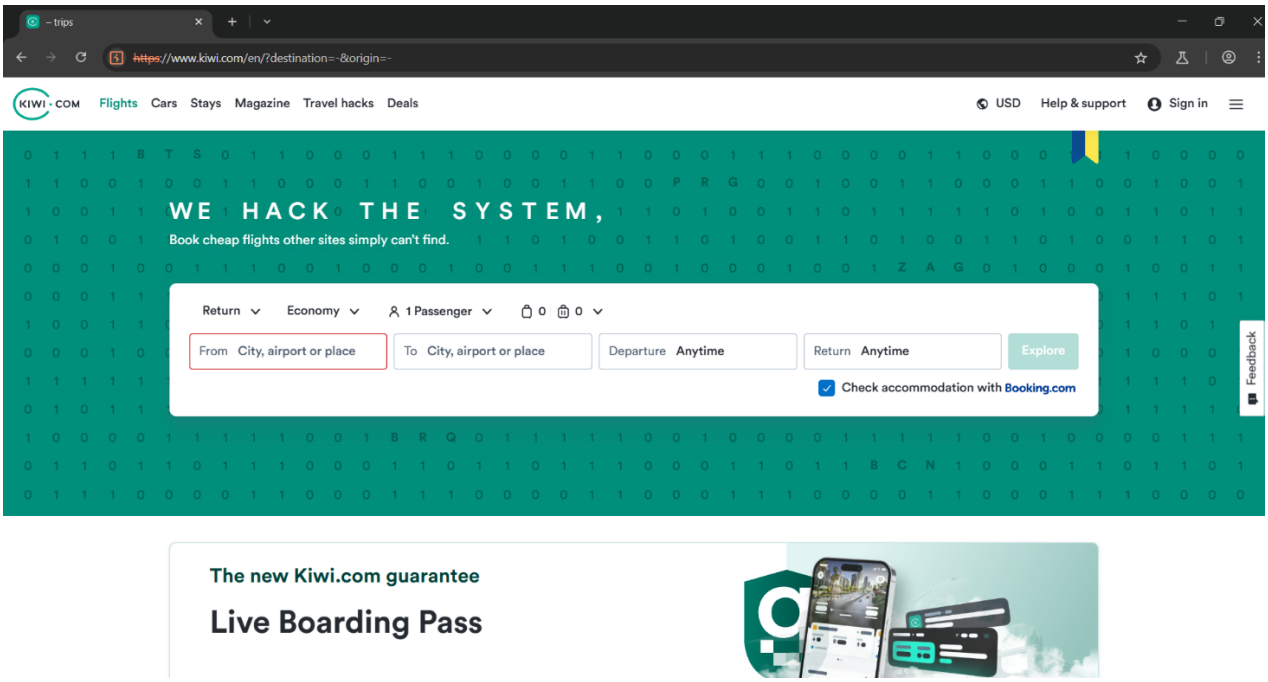
IT23269484 - T. H. Ranasinghe

Path Traversal Vulnerability in Kiwi.com

1. Vulnerability Title

Path Traversal Vulnerability in Kiwi.com Web Application

2. Vulnerability Description



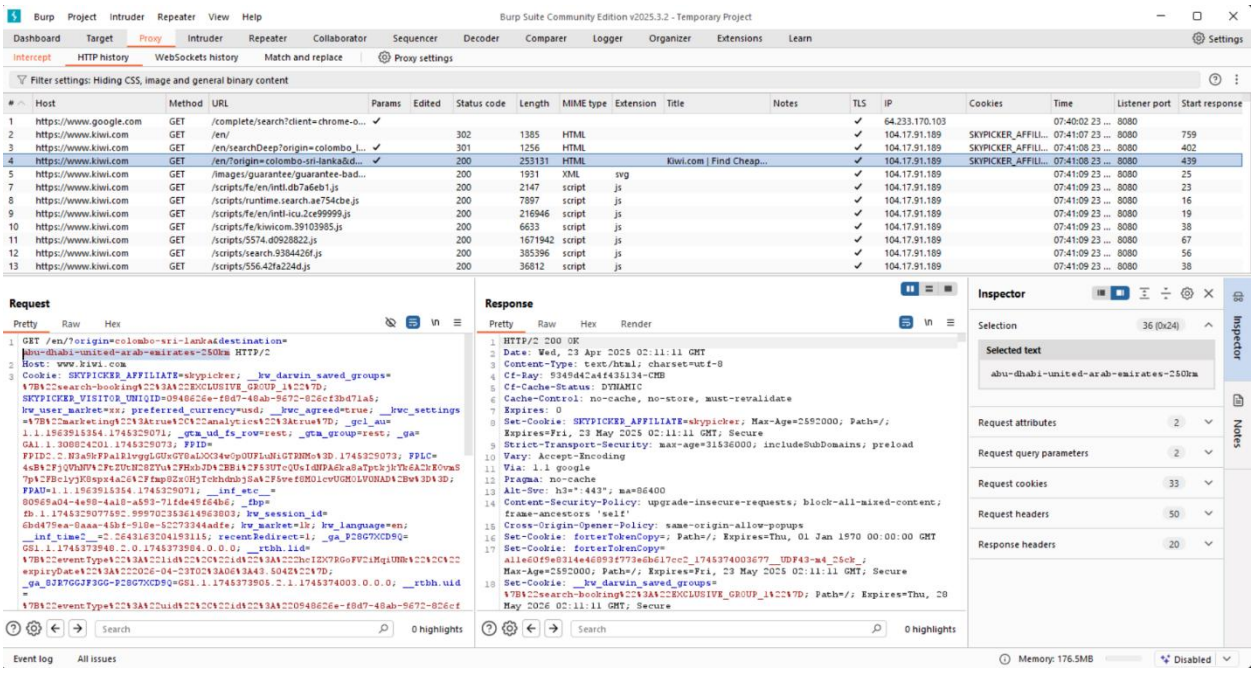
What is Path Traversal

Path Traversal, also called Directory Traversal, is a web security vulnerability that lets attackers access files outside a web application's intended folder. Web apps often use user input to decide which files to load, like images or pages. If this input isn't properly checked, attackers can use sequences like ../ (dot-dot-slash) to move up directory levels. For example, changing a URL from /images/photo.jpg to /images/../../etc/passwd might let an attacker read the /etc/passwd file on a Linux server, which contains user account details. This happens because the app doesn't validate the input, and weak server permissions allow access to sensitive files.

Path Traversal on Kiwi.com

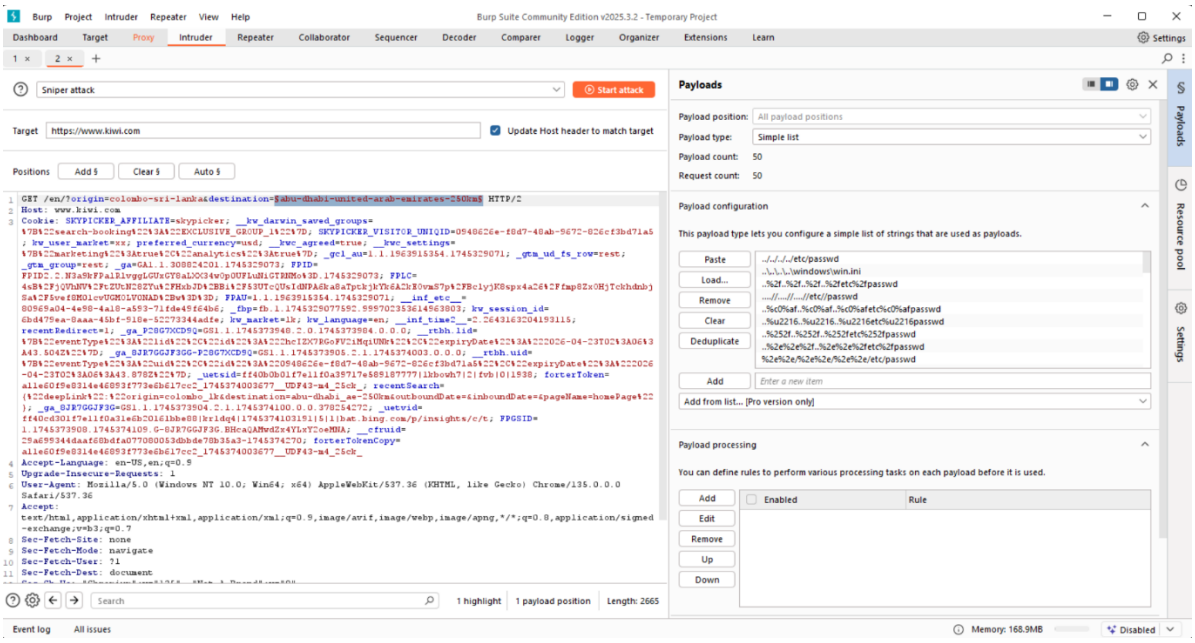
Path traversal testing was carried out on https://www.kiwi.com/en/ to determine whether the application properly handles user input in the URL path. The endpoint: **https://www.kiwi.com/en/origin-colombo-sri-lanka-destination-abu-dhabi-united-arab-emirates-250km** was tested using various path traversal payloads such as ../../etc/passwd, with tools like Burp Suite. However, none of the test payloads succeeded in accessing restricted files or revealing sensitive data. All responses appeared normal, indicating that the application handled the inputs safely.

This suggests that, based on the findings in the report, the application has proper input validation and directory access controls in place to prevent path traversal vulnerabilities.



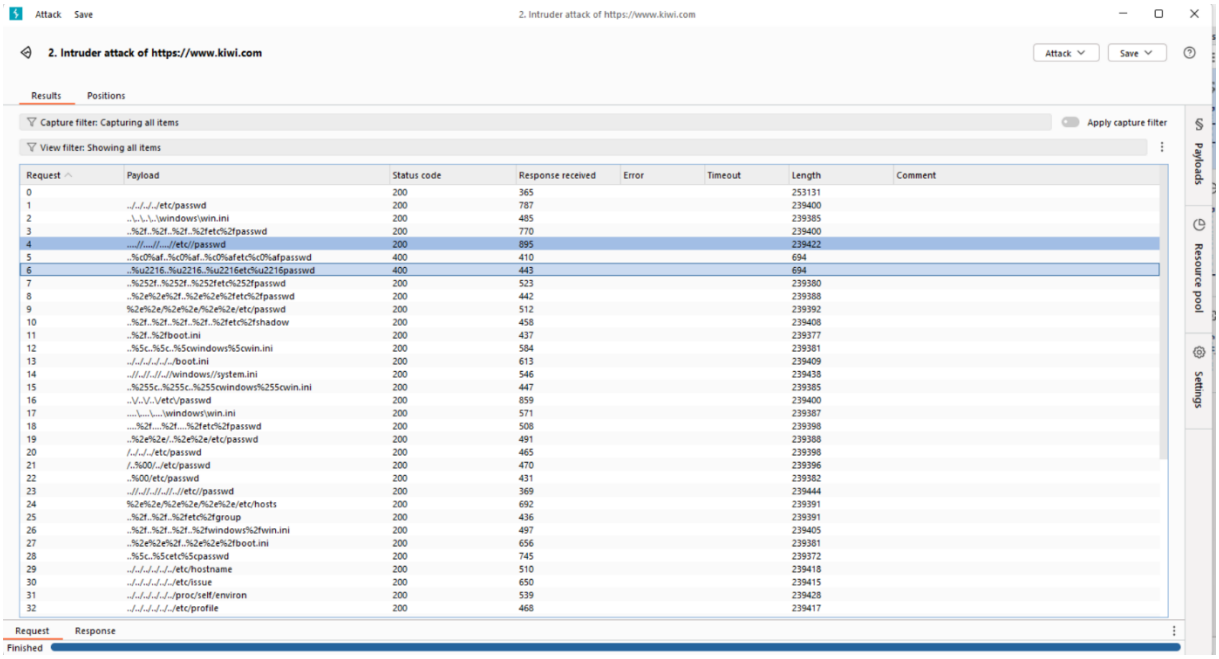
This shows the Intruder attack results on https://www.kiwi.com using path traversal payloads (e.g., ../../../../etc/passwd).

- Status code 200 means the request succeeded, possibly indicating a vulnerability if sensitive files were accessed.
- Responses with different lengths or response codes (like 400) might indicate filtering or blocked payloads.
- The "Payload" column shows the different path traversal strings tested.
- "Status" column helps identify which payloads succeeded or were blocked.
- A successful ../../../../etc/passwd attack might return a longer response with UNIX user data.
- If response length is consistent across payloads, the app may return a generic error page could indicate partial mitigation.
- Look for differences in response time, error messages, or unusual content.



This shows the HTTP history tab, listing all requests made through the Burp Proxy.

- The highlighted request includes a path with suspicious parameters (abu-dhabi-united-arab-emirates-250km).
- The response (right pane) is 200 OK, showing that the server accepted the request and included headers like Strict-Transport-Security, Set-Cookie, etc.
- Shows full request and response—useful for analyzing server behavior.
- Useful headers found:
    - Strict-Transport-Security: Prevents man-in-the-middle attacks over HTTP.
    - Set-Cookie: Used for session handling; can be a target for XSS.
    - Content-Security-Policy: Not seen here—may indicate CSP misconfiguration.
- You can right-click → "Send to Repeater" for manual testing of suspicious parameters.
- The URL path might hint at open redirect or file inclusion vectors depending on server behavior.



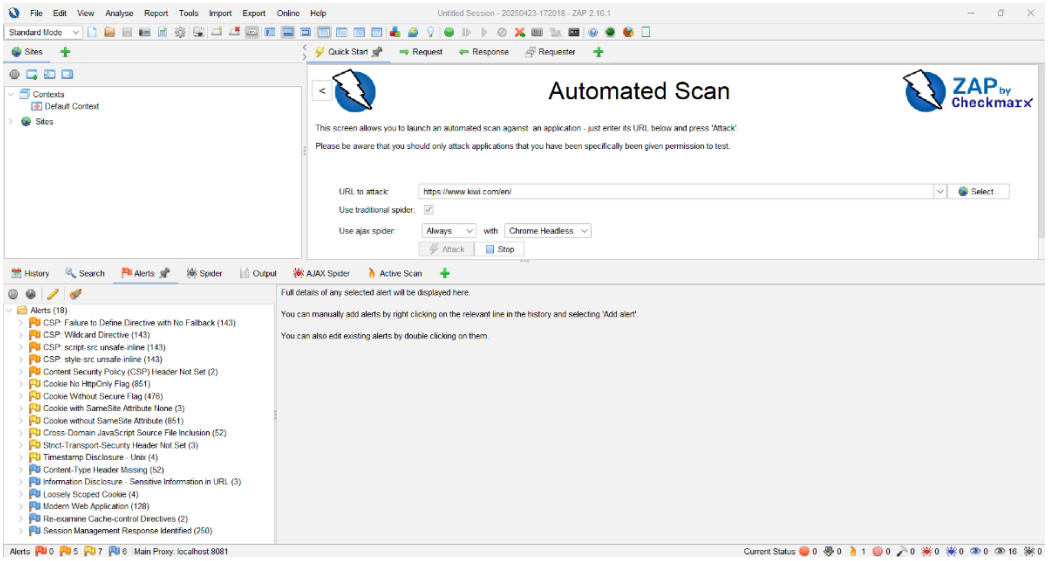
This shows the Intruder configuration screen:

- Attack type: Sniper
- Target: https://www.kiwi.com
- One injection point in the query parameter destination
- A list of path traversal payloads is set to test the vulnerability in the selected parameter.
- Sniper attack: Good for single-parameter testing each payload is tested one at a time.
- If multiple inputs need to be tested together (e.g., username and password), use Cluster Bomb.
- The destination parameter is likely user-controlled, making it a good injection point.
- You can add custom payloads or use Burp's built-in Payload Sets for traversal, XSS, LFI, etc.
- Consider enabling grep match for keywords like root:x, Warning, or server errors to flag interesting responses.

3. Additional Scans

OWASP ZAP Scan

An OWASP ZAP scan was performed to identify vulnerabilities:



ZAP found:

No path traversal vulnerabilities either.

Nikto Scan

```
┌──(kali㉿kali)-[~]
└─$ nikto -h https://www.kiwi.com/en/
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Multiple IPs found: 104.17.92.189, 104.17.91.189
+ Target IP:          104.17.92.189
+ Target Hostname:    www.kiwi.com
+ Target Port:        443
---------------------------------------------------------------------------
+ SSL Info:        Subject:  /CN=kiwi.com
                   Ciphers:  TLS_AES_256_GCM_SHA384
                   Issuer:   /C=US/O=Let's Encrypt/CN=E5
+ Start Time:         2025-04-23 18:18:10 (GMT5.5)
---------------------------------------------------------------------------
+ Server: cloudflare
+ /en/: Retrieved via header: 1.1 google.
+ /en/: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /en/: An alt-svc header was found which is advertising HTTP/3. The endpoint is: ':443'. Nikto cannot test HTTP/3 over QUIC. See: https://developer.mozi
lla.org/en-US/docs/Web/HTTP/Headers/alt-svc
+ /en/: Cookie SKYPICKER_AFFILIATE created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie ; Path created without the secure flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie ; Path created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie __kw_darwin_saved_groups created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie kw_session_id created without the secure flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie kw_session_id created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie SKYPICKER_VISITOR_UNIQID created without the secure flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie SKYPICKER_VISITOR_UNIQID created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie kw_user_market created without the secure flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie kw_user_market created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie kw_market created without the secure flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie kw_market created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie kw_language created without the secure flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ /en/: Cookie kw_language created without the httponly flag. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
+ ERROR: Error limit (20) reached for host, giving up. Last error: opening stream: can't connect: Connect failed: ; Connection timed out at /var/lib/nikt
o/plugins/LW2.pm line 5254.
: Connection timed out
+ Scan terminated: 8 error(s) and 17 item(s) reported on remote host
+ End Time:           2025-04-23 18:34:29 (GMT5.5) (979 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

- Scans for dangerous files, directories, and misconfigurations.
- Can show potential file disclosures or backup files (e.g., /etc/passwd, .bak, .old).
- Checks for HTTP methods like PUT/DELETE, which could aid exploitation.

If Nikto reports:

- "/etc/passwd" found → clear sign of path traversal.
- "Directory indexing enabled" → might allow manual traversal testing.
- Suspicious file extensions or backup files → exploitation may be possible.

Nikto identified:

- **No evidence of path traversal vulnerabilities.**

4. Mitigation Techniques

1. Validate input
    - Allow only expected characters (e.g., filenames, IDs).
    - Block ../, ..\\, %2e%2e%2f, etc.
2. Sanitize user input
    - Remove or escape dangerous sequences like ../.
3. Use safe functions
    - Use realpath() (PHP), os.path.abspath() (Python), getCanonicalPath() (Java).
4. Restrict to a base directory
    - Only allow access inside a specific folder (e.g., /uploads).
5. Avoid direct file access from user input
    - Map user input to filenames instead of using raw input.
6. Set proper file permissions
    - Prevent web apps from accessing sensitive system directories.
7. Keep software updated
    - Patch web frameworks and libraries regularly.
8. Use a Web Application Firewall (WAF)
    - Helps detect and block path traversal attempts

5. Conclusion

After testing the Kiwi.com website using Burp Suite and running automated scans with OWASP ZAP and Nikto, no Path Traversal vulnerability was found. Many test inputs (payloads) were used to try and access hidden or sensitive files, but none of them worked.

The website always responded in a normal way and didn't show any signs of being tricked or hacked. The scanners also didn't find any problems related to path traversal.

This means the website is handling user input well and is protecting its files from this type of attack. Still, it's a good idea to keep checking the website regularly, follow good security practices, and improve input checks to stay safe from future risks.