Vulnerability Assessment and Penetration Testing (VAPT)

Executive Summary

A web application security assessment was conducted on an authorized lab environment. During testing, a Server-Side Template Injection (SSTI) vulnerability was identified in the Form Tools application. The vulnerability allowed user-controlled input to be evaluated server-side, leading to Remote Code Execution (RCE).
The issue was validated through controlled testing and resulted in successful shell access as the web application user (www-data). This represents a high-severity security risk, as an attacker could execute arbitrary commands, access system files, and compromise the server.

Scope and Authorization

- Target Host: ssti.thm
- Target Application: Form Tools Web Application
- Scope: Web application logic, input handling, server-side processing
- Testing Type: Black-box assessment

All testing was performed within a controlled environment.

Methodology

The assessment followed a structured VAPT workflow:

1. Application reconnaissance
2. Version identification and vulnerability research
3. Vulnerability validation
4. Exploitation
5. Impact confirmation
6. Post-exploitation validation

The objective was to demonstrate understanding of the vulnerability lifecycle.

Findings Summary

Finding	Severity	Impact
Server-Side Template Injection (SSTI)	High	Remote Code Execution
Unvalidated User Input	Medium	Root cause enabling injection
Command Execution via Template	Critical	Full application compromise

Technical Findings

Server-Side Template Injection (SSTI)

During application testing, a user-controlled input field (Group Name) was identified where input was passed directly to backend processing. Client-side inspection confirmed that the input was not properly validated.

```
view_ns.create_new_group = function () {
  ft.dialog_activity_icon($("#add_group_popup"), "show");
  $.ajax({
    url: g.root_url + "/global/code/actions.php",
    type: "POST",
    dataType: "json",
    data: { group_name: $("#add_group_popup").find(".new_group_name").val(), actic
    success: view_ns.create_new_group_response,
    error: function (xhr, text_status, error_thrown) {
      ft.dialog_activity_icon($("#add_group_popup"), "hide");
      $("#add_group_popup").dialog("close");
      ft.error_handler(xhr, text_status, error_thrown);
```

Figure 1: *JavaScript logic showing user-controlled input being passed directly to backend processing.*

CVE Correlation

The application version was identified as **Form Tools 3.1.1**.

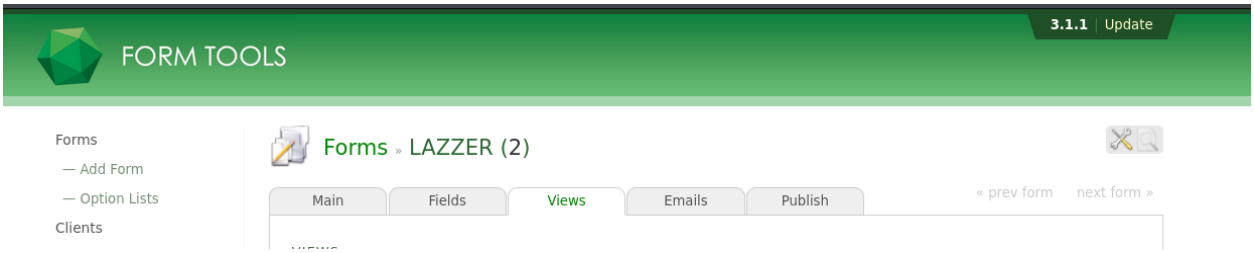

Figure 2: *Application version disclosure confirming Form Tools version 3.1.1.*

Public vulnerability research revealed a known vulnerability:

- CVE: **CVE-2024-22722**
- Vulnerability Type: Server-Side Template Injection
- Affected Parameter: Group Name
- Impact: Remote Code Execution

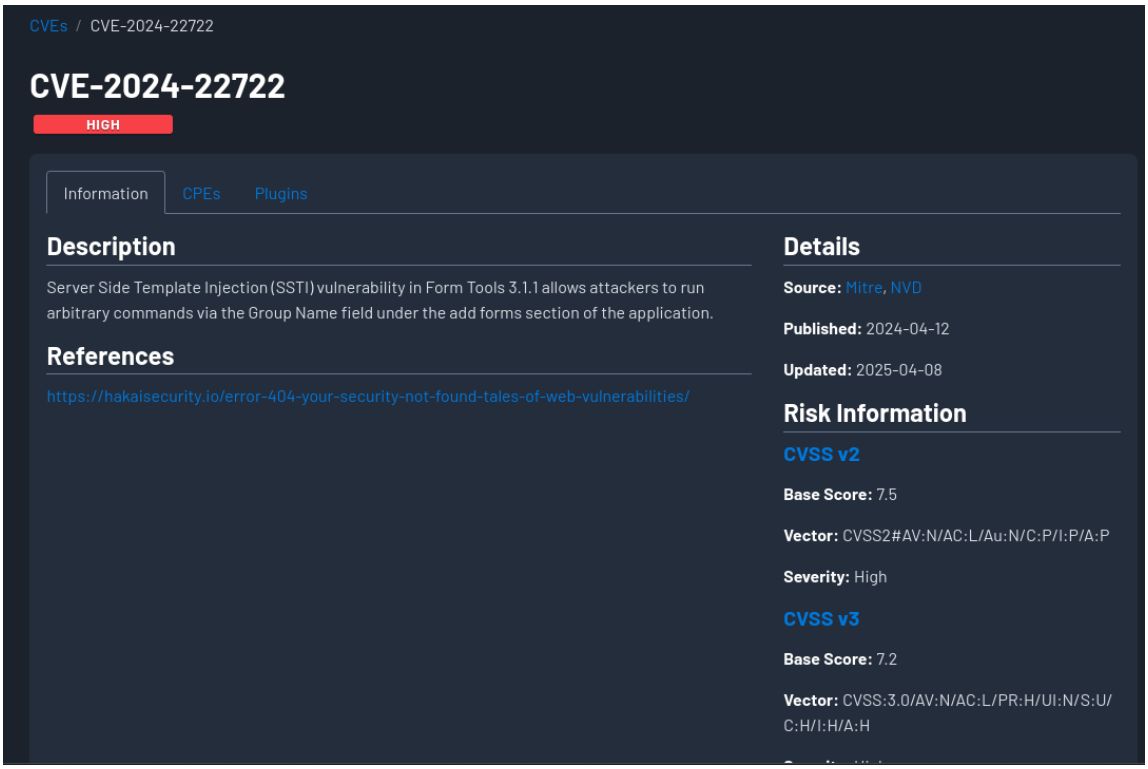

Figure 3: *Public CVE documentation confirming SSTI vulnerability affecting the Group Name parameter.*

Vulnerability Validation

A non-malicious test payload was used to confirm whether server-side evaluation was occurring. The application evaluated the input rather than displaying it as plain text, confirming SSTI behavior.

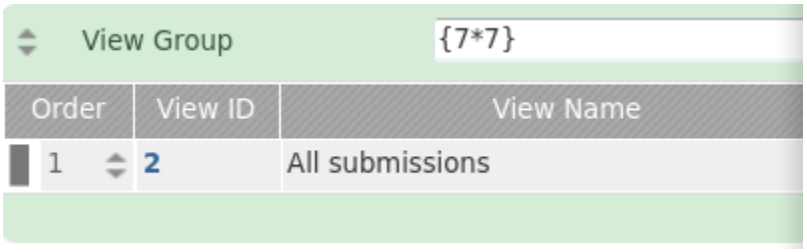

Figure 4: *Initial non-malicious SSTI test payload submitted to validate template processing behavior.*

Figure 5: *Application response confirming server-side evaluation of user input.*
This confirmed that user input was being processed by a template engine.

Exploitation and Impact Validation

After confirming SSTI behavior, further testing demonstrated that the vulnerability allowed command execution on the server. Execution was confirmed under the www-data context.

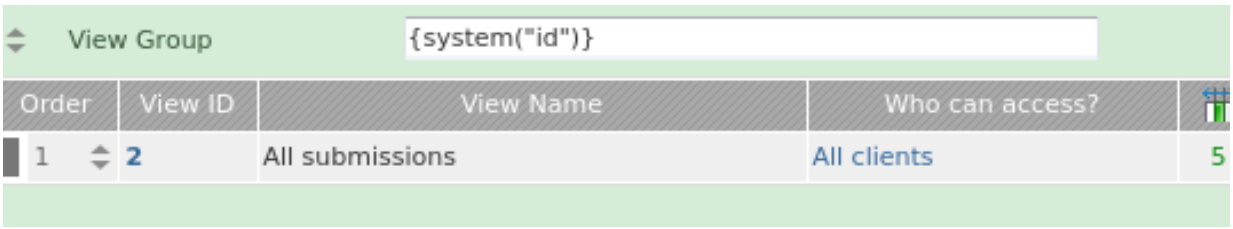

Figure 6: *Payload execution confirming Smarty-like template behavior allowing command execution.*

A controlled command execution test using the id command confirmed that attacker-supplied input was executed on the server, revealing the execution context as the www-data user.

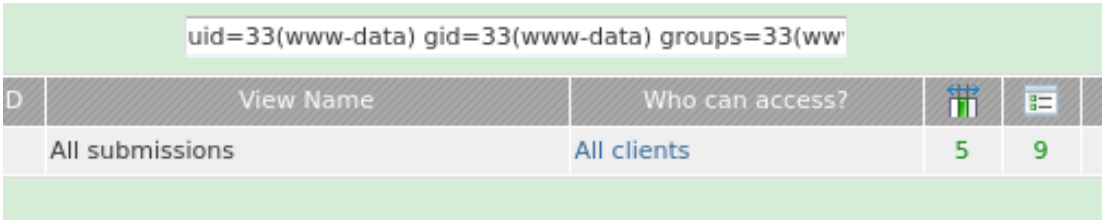

Figure 7: *Successful execution of the* `id` *command confirming remote code execution as the* `www-data` *user.*

This confirms successful Remote Code Execution and demonstrates that attacker-controlled input is executed by the server.

Reverse Shell Validation

To demonstrate real-world impact, controlled validation was performed by establishing a reverse shell connection.
The process included:

- Identifying available binaries on the target
- Creating an environment-appropriate reverse shell
- Delivering the payload using the vulnerable input
- Receiving a callback connection

Figure 8: *Identification of available binaries and shell environment on the target system.*

The file was retrieved using a standard HTTP request mechanism (e.g., curl) executed through the vulnerable input path.

Figure 9: *Target system retrieving the payload from the attacker-controlled HTTP server.*

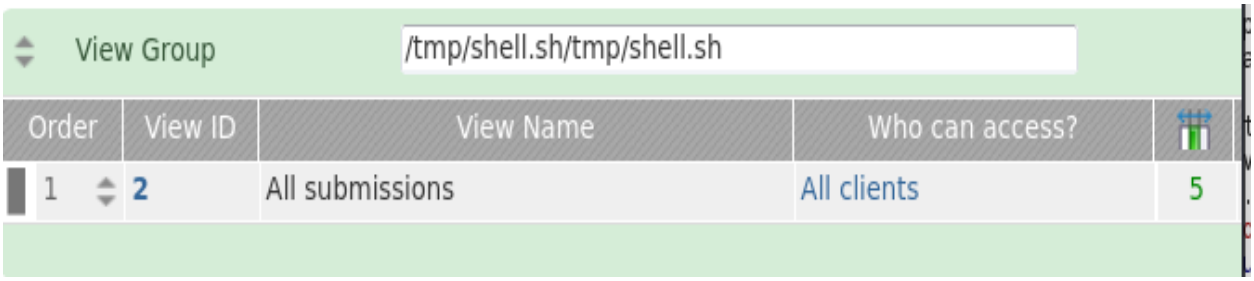

Figure 10: *Verification that the reverse shell script was successfully written to the target system.*

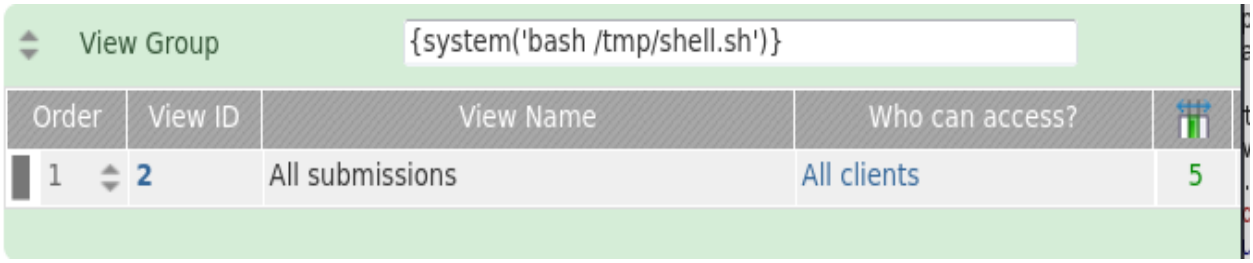

Figure 11: *Execution of the uploaded reverse shell payload via the vulnerable input flow.*

Figure 12: *Successful reverse shell connection established on the attacker system.*

Shell access confirmed:

- User: www-data

This demonstrates that an attacker could gain persistent interactive access to the system.

Post-Exploitation Validation

To confirm the impact without performing destructive actions, controlled validation was performed by accessing non-destructive system files.
File accessed:

- /etc/passwd

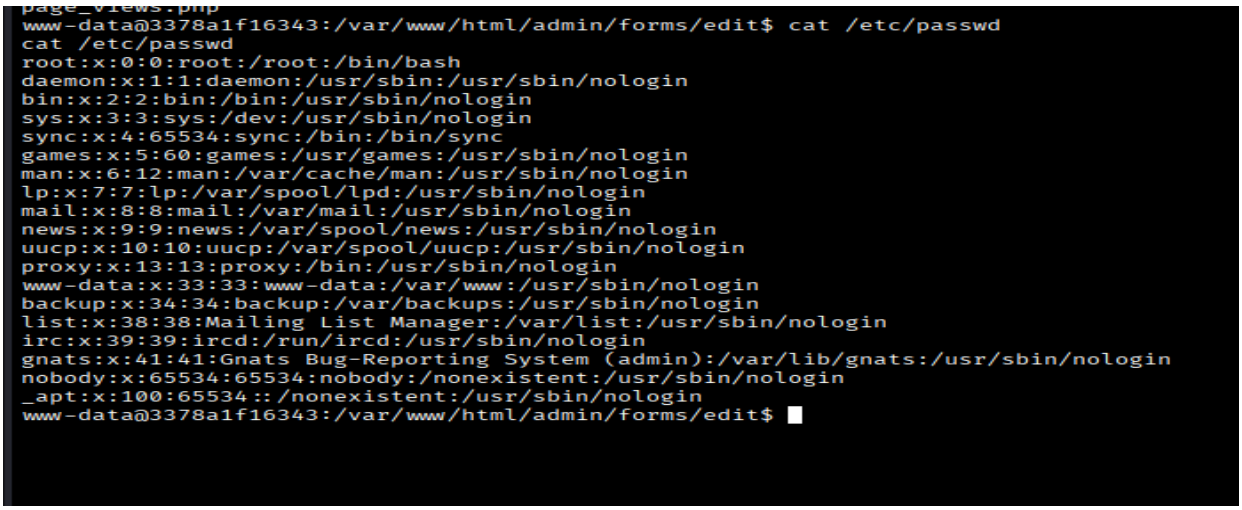

Figure 14: *Controlled access to /etc/passwd demonstrating post-exploitation system-level impact.*

Post-exploitation activities were intentionally limited to validation only. Privilege escalation was not pursued, as no immediate misconfigurations or escalation paths were identified during testing and the objective of the task was to validate impact rather than fully compromise the system. The assessment was therefore concluded after confirming successful access to system resources.

Risk Impact

If this vulnerability existed in a real-world system, an attacker could:

- Execute arbitrary commands on the server
- Upload or modify files
- Access sensitive configuration files
- Potentially escalate privileges
- Compromise the entire application and underlying system

This makes the issue **high risk and critical to address**.

Remediation Recommendations

- Sanitize and validate all user-controlled input
- Avoid passing raw user input into template engines
- Disable dangerous template functions such as system execution
- Patch the application to a version not affected by CVE-2024-22722
- Conduct regular secure code reviews and security testing

Conclusion

This assessment demonstrates how improper input validation can lead to severe security consequences. A single vulnerable input field escalated into full Remote Code Execution and shell-level access.
The task highlights the importance of:

- Secure input handling
- Secure template configuration
- Understanding application behavior
- Performing manual security testing, not just automated scans

References

1. OWASP Foundation. *OWASP Web Security Testing Guide (WSTG)* https://owasp.org/www-project-web-security-testing-guide/
2. MITRE. *CVE-2024-22722 – Form Tools Server-Side Template Injection Vulnerability* https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2024-22722
3. NIST. *Guide to Vulnerability Assessment (SP 800-115)* https://csrc.nist.gov/publications/detail/sp/800-115/final
4. TryHackMe. *Authorized Web Application Security Lab Environment* https://tryhackme.com/
5. PortSwigger. *Server-Side Template Injection (SSTI) Explanation* https://portswigger.net/web-security/server-side-template-injection
6. OWASP. *Input Validation Cheat Sheet* https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html