XSS Prevention in Ansible Templates

Overview

This document outlines XSS (Cross-Site Scripting) prevention measures for Ansible templates, particularly those generating HTML or web content.

Current Security Status

✅ **SECURE**: All current templates in the repository generate text-based reports and configuration files, not HTML content.

XSS Prevention Best Practices

1. Jinja2 Auto-Escaping

Always enable auto-escaping for HTML templates:

```
{# Enable auto-escaping at template level #}
{% autoescape true %}
<html>
<body>
    <h1>{{ page_title }}</h1>  {# Automatically escaped #}
    <p>{{ user_content }}</p>  {# Automatically escaped #}
</body>
</html>
{% endautoescape %}
```

2. Manual Escaping

Use explicit escaping filters for user-controlled content:

```
{# HTML content escaping #}
<div>{{ user_input | e }}</div>
<div>{{ user_input | escape }}</div>

{# HTML attribute escaping #}
<input value="{{ user_value | e('html_attr') }}">

{# URL escaping #}
<a href="{{ user_url | urlencode }}">Link</a>

{# JavaScript escaping #}
<script>
var userdata = "{{ user_data | e('js') }}";
</script>
```

3. Context-Aware Escaping

Different contexts require different escaping strategies:

```
{# HTML content #}
<p>{{ content | e }}</p>

{# HTML attributes #}
<div class="{{ css_class | e('html_attr') }}">

{# JavaScript strings #}
<script>
var message = {{ message | tojson }};  {# Safe JSON encoding #}
</script>

{# CSS values #}
<style>
.dynamic { color: {{ color | e('css') }}; }
</style>
```

4. Avoiding |safe Filter

Never use `|safe` filter with user-controlled content:

```
{# DANGEROUS - Never do this with user input #}
<div>{{ user_html | safe }}</div>

{# SAFE - Only use |safe with trusted, static content #}
<div>{{ trusted_static_html | safe }}</div>
```

Secure Template Examples

Example 1: Dashboard Template

```
{% autoescape true %}
<!DOCTYPE html>
<html>
<head>
    <title>{{ dashboard_title }}</title>
    <meta charset="utf-8">
</head>
<body>
    <h1>{{ dashboard_title }}</h1>

    <div class="stats">
        <h2>System Statistics</h2>
        <ul>
        {% for stat in system_stats %}
            <li>
                <strong>{{ stat.name }}</strong>:
                <span class="{{ stat.status | e('html_attr') }}">
                    {{ stat.value }}
                </span>
            </li>
        {% endfor %}
        </ul>
    </div>

    <div class="alerts">
        <h2>Alerts</h2>
        {% for alert in alerts %}
        <div class="alert alert-{{ alert.level | e('html_attr') }}">
            <strong>{{ alert.title }}</strong>: {{ alert.message }}
        </div>
        {% endfor %}
    </div>

    <script>
    // Safe JSON data passing
    var dashboardData = {{ dashboard_data | tojson }};
    var refreshInterval = {{ refresh_interval | int }};
    </script>
</body>
</html>
{% endautoescape %}
```

Example 2: Configuration File with HTML Comments

```
{# This template generates a config file with HTML-like syntax #}
# {{ ansible_managed }}
# Configuration generated: {{ ansible_date_time.iso8601 }}

# Server configuration
server_name={{ server_name | e }}
server_port={{ server_port | int }}

# User-provided descriptions (escaped for safety)
{% for item in config_items %}
# {{ item.description | e }}
{{ item.key }}={{ item.value | e }}
{% endfor %}

# HTML-like configuration block
<VirtualHost {{ virtual_host_ip | e }}:{{ virtual_host_port | int }}>
    ServerName {{ server_name | e }}
    DocumentRoot {{ document_root | e }}

    # User-defined custom directives
    {% for directive in custom_directives %}
    {{ directive.name | e }} {{ directive.value | e }}
    {% endfor %}
</VirtualHost>
```

Input Validation

1. Validate Input Types

```
- name: Validate dashboard title
  assert:
    that:
      - dashboard_title is string
      - dashboard_title | length > 0
      - dashboard_title | length < 100
    fail_msg: "Dashboard title must be a non-empty string under 100 characters"

- name: Validate HTML content
  assert:
    that:
      - user_content is string
      - user_content | regex_search('<script') is none
      - user_content | regex_search('javascript:') is none
    fail_msg: "User content contains potentially dangerous HTML"
```

2. Sanitize Input Variables

```
- name: Sanitize user inputs
  set_fact:
    clean_title: "{{ raw_title | regex_replace('[<>\"\\']', '') }}"
    clean_description: "{{ raw_description | regex_replace('[<>\"\\']', '') }}"
```

Security Testing

1. XSS Test Cases

Create test playbooks with malicious inputs:

```yaml
- name: Test XSS prevention
  vars:
    test_inputs:
      - "<script>alert('xss')</script>"
      - "javascript:alert('xss')"
      - "' onload='alert(1)'"
      - "<img src=x onerror=alert(1)>"
  template:
    src: dashboard.j2
    dest: /tmp/xss_test.html
  loop: "{{ test_inputs }}"
  vars:
    user_content: "{{ item }}"
```

2. Automated Security Scanning

```yaml
- name: Run HTML security scan
  shell: |
    # Use tools like htmlhint or custom scripts
    htmlhint /tmp/generated_template.html
  register: security_scan
  failed_when: security_scan.rc != 0
```

Compliance Checklist

- [ ] Auto-escaping enabled for all HTML templates
- [ ] Manual escaping applied to all user-controlled variables
- [ ] Context-aware escaping used for different output contexts
- [ ] No use of `|safe` filter with user input
- [ ] Input validation implemented for all user-provided data
- [ ] Security testing performed with malicious inputs
- [ ] Regular security audits of template files

Jinja2 Security Configuration

Recommended Environment Settings

```python
# For custom Jinja2 environments in Python modules
from jinja2 import Environment, select_autoescape

env = Environment(
    autoescape=select_autoescape(['html', 'xml']),
    # Disable dangerous features
    finalize=lambda x: x if x is not None else ''
)
```

Ansible Configuration

```
# In ansible.cfg
[defaults]
# Ensure Jinja2 extensions don't introduce vulnerabilities
jinja2_extensions =
```

Incident Response

If XSS is Discovered

1. **Immediate**: Remove or disable affected templates
2. **Assessment**: Identify scope of vulnerability
3. **Remediation**: Apply proper escaping and validation
4. **Testing**: Verify fix with security tests
5. **Documentation**: Update security documentation

References

- OWASP XSS Prevention Cheat Sheet (https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html)
- Jinja2 Security Documentation (https://jinja.palletsprojects.com/en/3.1.x/api/#autoescaping)
- Ansible Template Security Best Practices (https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html#controlling-whitespace)

Security Status: ✅ No XSS vulnerabilities detected in current templates
Last Updated: {{ ansible_date_time.iso8601 }}
Compliance: Phase 2 XSS Vulnerability Remediation - COMPLETED