USENIX Security '25 Artifact Appendix: Are CAPTCHAs Still Bot-hard? Generalized Visual CAPTCHA Solving with Agentic Vision Language Model

Xiwen Teoh[1,2], Yun Lin[1], Siqi Li[2], Ruofan Liu[2], Avi Sollomoni[3], Yehuda Afek[3], Yaniv Harel[3], Jin Song Dong[2]

Shanghai Jiao Tong University[1], *National University of Singapore*[2], *Tel Aviv University*[3]

A Artifact Appendix

A.1 Abstract

Visual CAPTCHAs present users with interactive puzzles that must be solved to access protected online content. Their effectiveness relies on the assumption that these challenges are difficult for bots but easy for humans. However, the rise of general-purpose AI models (e.g., ChatGPT) calls this assumption into question, potentially undermining the reliability of current CAPTCHA systems.

To evaluate the security of visual CAPTCHAs in this new landscape, we implement our main artifacts: (1) Halligan, a generalized visual language model (VLM)-based CAPTCHA solver, and (2) an interactive offline benchmark containing 26 diverse types of visual CAPTCHAs. Halligan functions as a VLM agent equipped with tools to abstract the CAPTCHA layout, explore possible options, and reason about or compare individual choices. It automatically formulates each visual CAPTCHA as a search optimization problem and generates an executable Python script to solve it. This script is reusable across all instances of the same CAPTCHA type. For functionality, we present Halligan's script generation workflow through execution traces and conduct attacks using the generated scripts on selected samples from the benchmark.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Halligan could be misused to attack CAPTCHAs on real-world websites. Malicious actors might exploit it for purposes such as promotion spam, registration abuse, data scraping, or bot-driven raiding. Such misuse can undermine trust in online communities, degrade user experiences, and disrupt the normal operations of websites. However, in this AE, we restricted all attacks to our offline benchmark. The benchmark is self-contained and does not interact with any live websites or CAPTCHA service providers. As such, AE reviewers do not have associated risks.

A.2.2 How to access

Both Halligan and the benchmark are available in our Zenodo repository (DOI: 10.5281/zenodo.15580922). Please use the latest version.

A.2.3 Hardware dependencies

We tested the functionality on a desktop computer with 16GB of GPU VRAM (Optional), 8GB of system RAM, and 16GB of available disk space.

A.2.4 Software dependencies

We recommend using Linux. Our setup was tested on Ubuntu 20.04.3 LTS with Pixi 0.47.0, CUDA 12.1 (Optional), Docker 24.0.7, and Docker Compose 2.21.0.

A.2.5 Benchmarks

The benchmark for AE is provided in the artifact.

A.3 Set-up

Before proceeding with the installation, please make sure that the following prerequisite software is installed.

1. Install Docker Desktop[1]
2. Install Pixi[2]

A.3.1 Installation

Setup Benchmark

1. In a new terminal, download and unzip `benchmark.zip` from Zenodo.
2. Change into the `benchmark` directory:

[1] https://docs.docker.com/compose/install/
[2] https://pixi.sh/dev/installation/

3. Create and start containers.

   ```
   docker compose up
   ```

   This takes a few minutes. It begins by building the `benchmark` Docker image using the provided Dockerfile, which should have a hash ID `1a27e353ef13`. Once the build is complete, it will launch two containers: the benchmark server and a browser. These containers expose ports 3000 and 5000 on the host, respectively.

Setup Halligan

1. In a new terminal, download and unzip `halligan.zip` from Zenodo.
2. Change into the `halligan` directory.
3. Setup Pixi environment.

   ```
   pixi install
   ```

   This may take a few minutes for Pixi to resolve all installation dependencies, update the lockfile and install the environment.

4. Make a copy of `.env.example` with the filename `.env`.

   ```
   cp .env.example .env
   ```

5. In `.env`, replace the value of `OPENAI_API_KEY` with our provided key (no line breaks):

   ```
   sed -i "s/^OPENAI_API_KEY=.*/OPENAI_API_KEY=
   sk-proj-0G5YTPRtbreaTJSxjzbDtvvvgIPcZm4Ag3-
   WwuVusHBp1Wp4Z-Ws7mSSP44hYT7fTLdKY07M3DT3Bl
   bkFJZEnH8-BeqmYNOtxP47to2IMuhfxwagw3ZkUpy4fLF1
   pttROQiYfKCJ0ry0L4QRGaFClblcK74A/" .env
   ```

A.3.2 Basic Test

1. In a new terminal, navigate to `/halligan`
2. Run the script `basic_test.py`, which checks that the benchmark, browser, and the key components of Halligan are functioning correctly.

   ```
   pixi run pytest basic_test.py --verbose
   ```

   Once completed, verify that the terminal displays 29 lines of `PASSED` test messages:

   ```
   basic_test.py::test_browser PASSED
   basic_test.py::test_benchmark PASSED
   basic_test.py::test_captchas[...] PASSED
   ...
   basic_test.py::test_halligan PASSED
   ===== 29 passed in 31.45s =====
   ```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *Halligan automatically generates an executable Python script for solving challenges of a given visual CAPTCHA type. This capability is demonstrated by Experiment (E1), whose three-stage process is detailed in Section 3 and illustrated in Figure 2.*

(C2): *Using the generated scripts, Halligan successfully solves at least one challenge from each of the 26 visual CAPTCHA types included in the benchmark. This is proven by Experiment (E2), with full details provided in Table 4.*

A.4.2 Experiments

(E1): *[10 human-minutes + 20 compute-minutes]: The first experiment demonstrates Halligan's script generation process.*

Preparation: It is assumed that you have followed the set-up instructions (§A.3.1)

Execution: Run the experiment as follows:

1. Open a new terminal session and navigate to `/halligan`.
2. Run the generation script:

   ```
   pixi run python generate.py
   ```

Results: Verify that a log file matching the pattern agent-*.log is created. In the log file, confirm that Halligan has generated scripts for 26 out of 26 CAPTCHAs, each with a unique name, and that each entry is followed by `[Stage 1] ... [Stage 3]`, resulting in a total of 104 lines. A sample output is shown below for reference:

```
Generating script (1 out of 26): lemin
    [Stage 1] Objective Identification
    [Stage 2] Structure Abstraction
    [Stage 3] Solution Composition
Generating script (2 out of 26): geetest/slide
    ...
Generating script (26 out of 26): amazon
    [Stage 1] Objective Identification
    [Stage 2] Structure Abstraction
    [Stage 3] Solution Composition
```

Next, verify that a new directory `/results/generation`, relative to the generation script, is created. This directory should contain 26 `.ipynb` generation trace files, each with a unique name. These files show how Halligan generates the solution script for each CAPTCHA.

```
/lemin.ipynb
/geetest_slide.ipynb
...
/amazon.ipynb
```

Examine one of the `.ipynb` trace files, verify that there is a `PROMPT =` and `RESPONSE =` code block under the headings "Objective Identification", "Structure Abstraction", and "Solution Composition". For example:

```
PROMPT = ...
```

```
RESPONSE = ...
TIME = 7.902754155918956
FINGERPRINT = fp_5bf33c1ec4
TOTAL_TOKENS = 866
PROMPT_TOKENS = 707
COMPLETION_TOKENS = 159
```

(E2): *[5 human-minutes + 20 compute-minutes]: The second experiment uses scripts generated by Halligan to solve 26 different types of CAPTCHAs in the benchmark.*
Preparation: It is assumed that you have followed the set-up instructions (§A.3.1)
Execution: Run the experiment as follows:

1. Open a new terminal session and navigate to `/halligan`.
2. Run the execution script:
   `pixi run python execute.py`

Results: Verify that a log file matching the pattern `agent-*.log` is created. In the log file, confirm that Halligan has tested 26 out of 26 CAPTCHAs, each with a unique name, and that each entry is followed by `Solved: True`, resulting in a total of 52 lines. A sample output is shown below for reference:

```
Testing CAPTCHA (1 out of 26): lemin
Solved: True
Testing CAPTCHA (2 out of 26): geetest/slide
Solved: True
...
Testing CAPTCHA (26 out of 26): amazon
Solved: True
```

Next, verify that a new directory `/results/execution`, relative to the execution script, is created. This directory should contain 26 `.ipynb` execution trace files, each with a unique name. These files demonstrate how Halligan's script derives the solution for each CAPTCHA.

```
/lemin.ipynb
/geetest_slide.ipynb
...
/amazon.ipynb
```

A.5 Notes on Reusability

Each CAPTCHA type in the benchmark is implemented as a Flask Blueprint and can be accessed or extended via routes following the pattern `.../{CAPTCHA_NAME}/{CHALLENGE_ID}`. For more information on Flask Blueprints, see this tutorial.

Halligan can be applied to other visual CAPTCHAs outside of the benchmark, without requiring additional configuration. For more details, see Section 4.4. Halligan also supports two example practical use cases: (1) it can aid in anti-phishing and security crawling by revealing CAPTCHA-cloaked phishing websites, and (2) it can assist webmasters and security practitioners in designing more robust anti-bot mechanisms by testing the effectiveness of new CAPTCHA challenges.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.