Thought Process

1) High level concepts of what is honeypot in cypersecurity
    a. Google: what is honeypot in cyber security
    b. https://www.crowdstrike.com/en-us/cybersecurity-101/exposure-management/honeypots/
    c. https://www.youtube.com/watch?v=6F5RYn0mB0A
    d. Ask chatgpt: what is honey pot in cypersecurity
    e.

It seems like a popular decoy techniques to attack cyber attackers and study their techniques: seems like legitimate network but in reality monitor systems. A honeypot can be modeled after any digital asset, including software applications, servers or the network itself

Too high level. How is this related to the task of the requirement:

Context
Web applications are often targeted by attacks that involve loading a webpage and automating or replicating HTML form behavior to abuse backend APIs. One defensive technique against such actors is the use of a "**honeypot" or "honeytrap"**—a mechanism designed to detect and deter automated abuse.
Your task is to research this concept and design a solution that meets the following criteria:
Requirements
1. **Client-Side Detection**
   Detection should occur entirely in the browser (no changes to the backend).
2. **Reusable Detection Logic**
   Implement detection logic as a **standalone, reusable script** that is separate from the HTML flow.

- It can be generic or configurable but **should not be hardcoded to a specific page**.

3. **Reporting**
   Detection results should be sent to a different, dedicated backend endpoint (you may mock this for the exercise).

google: honeypot technique of blocking spam in webform and client side

Found some very useful video:

1. https://www.youtube.com/shorts/4aVgn9px0iI
2. https://www.youtube.com/watch?v=vYc6jN579V4

and Article:

https://kdesineedi.medium.com/honeypots-in-web-forms-smart-spam-protection-without-compromising-ux-2f62cf0f6f41

https://www.getvero.com/resources/add-a-honeypot-to-website-forms-to-reduce-spam/

https://ivyforms.com/blog/what-is-a-honeypot/

Interpretation of requirement:

Client side implementation only

1. Provide a js library that can include in any webpage, that when the form is submit, detect if its was a bot signal
    a. Fill out the form on the hidden fields or
    b. Filled out the form too fast.
    c. Fill out the form without moving the mouse or keyboard
2. Report suspicious activity (this is signal only, not a true indicator)

Ok, now that I have an better of understand of what honeypot means, and how its used on the client side.

Seems like there are two primary ways to implements honeypot fields

```
.honeypot-field {

  display: none;

}
```

Pros:

Invisible to all users
Low risk of browser autofill

cons

Trivial for bots to detect - only fill visible fields

Ineffective against even moderately sophisticated bots

And

`position: absolute; left: -9999px;`

`pros`

Harder for naive bots to detect
Better at catching form replayers
Stronger signal than `display: none`

Cons:

Accessibility risks if misconfigured
Screen readers may announce the field
Autofill edge cases

Some browsers or password managers may autofill off-screen fields

Still detectable by advanced bots

Init prompt to chatgpt:

Context Web applications are often targeted by attacks that involve loading a webpage and automating or replicating HTML form behavior to abuse backend APIs. One defensive technique against such actors is the use of a "honeypot" or "honeytrap"—a mechanism designed to detect and deter automated abuse. Your task is to research this concept and design a solution that meets the following criteria: Requirements 1. Client-Side Detection Detection should occur entirely in the browser (no changes to the backend). 2. Reusable Detection Logic Implement detection logic as a standalone, reusable script that is separate from the HTML flow.

Output a basic:

```
(function (global) {
  class HoneypotDetector {
    constructor(config) {
```

```
    this.config = Object.assign({
      honeypotSelector: '[data-honeypot]',
      reportUrl: '/honeypot/report',
      minHumanTimeMs: 1500
    }, config);

    this.startTime = Date.now();
    this.interactions = {
      mouse: false,
      keyboard: false,
      focus: false
    };
  }

  analyzeForm(form) {
    const reasons = [];

    // Honeypot field check
    const honeypots = form.querySelectorAll(this.config.honeypotSelector);
    honeypots.forEach(input => {
      if (input.value && input.value.trim() !== '') {
        reasons.push('HONEYPOT_FIELD_FILLED');
      }
    });

    return {
      suspicious: reasons.length > 0,
      reasons,
      elapsed,
      interactions: this.interactions
    };
  }

  report(result) {
    if (!result.suspicious) return;

    navigator.sendBeacon(
      this.config.reportUrl,
      JSON.stringify({
        timestamp: Date.now(),
        userAgent: navigator.userAgent,
        result
      })
    );
  }
}
```

```
  global.HoneypotDetector = HoneypotDetector;
})(window);
```

```
<input type="text" name="company_fax" autocomplete="off" tabindex="-1"
style="display:none">
```

I like this approach because I have did something similar to this before:

https://help.goacoustic.com/hc/en-us/articles/360043311973-Web-tracking

it's a Immediately Invoked Function Expression (IIFE). Avoid polluting the global scope. Preferably namespace the it on the window object so people don't override it.

```
global.vhoang = window.vhoang || {};
global.vhoang.HoneypotDetector = HoneypotDetector;
```

to use in any js script:

```
<script>
    const detector = new vhoang.HoneypotDetector({
      reportUrl: 'http://localhost:8080/api/honeypot/report'
    });

    document
      .getElementById('registrationForm')
      .addEventListener('submit', (event) => {
        const result = detector.analyzeForm(event.target);
        detector.report(result);
      });
</script>
```

Incrementally, from reading on the internet and chatgpt, adding other dimensions such as how fast the form as submit and if there was any interaction with the keyboard and mouse when the form was submitted:

```
_bindEvents() {
  document.addEventListener('mousemove', () => this.interactions.mouse = true, { once: true });
  document.addEventListener('keydown', () => this.interactions.keyboard = true, { once: true });
  document.addEventListener('focusin', () => this.interactions.focus = true, { once: true });
}
```

```
const elapsed = Date.now() - this.startTime;
if (elapsed < this.config.minHumanTimeMs) {
  reasons.push(FORM_SUBMITTED_TOO_FAST);
}
```

```
if (!this.interactions.mouse && !this.interactions.keyboard) {
  reasons.push(NO_HUMAN_INTERACTION);
}
```

Aslo, IP is a weak signal to use because when client run on the cloud, the IP is really mask. But essentially we can use that to compute like a score as part of the score > threshold, then we can report the activity.

Also, I also ask chatgpt to generate a hashIdenty when we report a signal and on the server, we compute the hash again to make sure it has not been tempered with.

Also, these are bot signal, they are not 100% accurate, and advanced bot can bypass this. This often use as signal. On the serverside, we can use kafka stream or redis to do windowing or counting of numbers of clicks happen within a certain time window. If we suspect a bot, we should incorporate re-captcha and blocking functionality to prevent bot attacks.