分組名單（不足 5 個人空著就好）：

姓名	學號
陳孟楷	113550021
蔡昀呈	113550058

1. Name of the paper:

Password Managers: Attacks and Defenses David Silver, Suman Jana, and Dan Boneh, Stanford University; Eric Chen and Collin Jackson, Carnegie Mellon University.

2. Summary:

The paper presents an in-depth analysis of the security vulnerabilities in popular password managers, focusing on their autofill behaviors across different platforms. The authors investigate both browser-integrated and third-party password managers, evaluating how and when these tools decide to automatically fill in login credentials. Through a series of controlled experiments, the paper identifies several design flaws that allow attackers to extract passwords remotely without any user interaction.

A central attack scenario discussed in the paper involves a user connecting to a rogue WiFi hotspot controlled by an attacker. Once connected, the attacker serves a landing page that invisibly loads multiple iFrames referencing vulnerable sites the user has previously visited. If the password manager autofills login fields inside those iFrames, malicious JavaScript injected by the attacker can steal the credentials and transmit them to an external server. The authors classify these as "sweep attacks," capable of extracting up to 10 passwords per second.

The paper also introduces two key defenses: forcing user interaction before autofill, and secure filling, a mechanism that prevents JavaScript from accessing autofilled passwords and ensures credentials are only submitted to known domains. The authors implemented these defenses in Chromium to demonstrate feasibility.

3. Strength(s) of the paper:

One of the most significant strengths of the paper is its comprehensive empirical evaluation. The authors analyze ten password managers across various desktop and mobile platforms, offering a broad and practical overview of the real-world implications of autofill policies. This comparative approach highlights inconsistencies among implementations and helps identify systemic weaknesses.

Another strength is the introduction of concrete attack vectors that do not rely on user naivety or software exploits. The authors show that simply connecting to an untrusted network (e.g., a public café WiFi) is enough to compromise password confidentiality. This makes the findings highly relevant to everyday users.

Moreover, the proposed "secure filling" mechanism reflects thoughtful design. It strikes a balance between usability and security by allowing autofill under safer conditions while restricting unsafe behaviors. The idea of making the password unreadable by JavaScript, yet still usable in form submission, addresses a core vulnerability elegantly.

4. Weakness(es) of the paper:

While the paper excels in its technical rigor, there are some limitations. First, the scope of defenses is narrow, mainly focusing on the browser's or password manager’s behavior. The authors acknowledge that server-side defenses are limited, but a more detailed exploration of collaboration between browsers and websites (e.g., through headers or extended HTML5 attributes) would enhance the long-term applicability of their solution.

Second, although the secure filling solution is promising, its compatibility with AJAX-based login forms is problematic. Many modern websites use JavaScript to validate or submit forms asynchronously. The proposed workarounds, such as creating a separate login iFrame or using a new API (sendPassword()), are technically feasible but require non-trivial changes on the part of website developers. This limits short-term adoption.

Lastly, the research is now dated (published in 2014), and modern password managers and browsers may have evolved since then. Although the core ideas remain relevant, the paper lacks a discussion on how its recommendations fit into long-term security architecture trends like WebAuthn or hardware-backed credentials.

5. Your own reflection

A. What did I learn from this paper?

I learned that the autofill convenience provided by password managers can be a double-edged sword. Even trusted platforms like Chrome or Safari can inadvertently expose sensitive data through weak assumptions about the security of the surrounding environment. The idea that a network attacker could silently extract dozens of passwords without user interaction was both eye-opening and concerning.

B. How would I improve or extend the work?

- Formalizing autofill policies using a declarative model, possibly involving content security policy (CSP) enhancements that allow developers to whitelist trusted autofill behaviors.
- Developing a compatibility layer that helps websites adapt to secure filling through browser extensions or polyfills. This could ease the transition and foster adoption.

C. What are the unsolved questions?

- How can password managers integrate with biometric or hardware tokens while still maintaining secure autofill?
- Can browsers use machine learning to detect phishing pages before autofilling?
- What's the best way to enforce secure password submission in a world dominated by JavaScript-heavy SPAs (single-page applications)?

D. What are the broader impacts?

The broader impact of this paper lies in user trust and internet security hygiene. By exposing these flaws, the authors compel browser vendors and password manager developers to rethink usability-focused features that may carry hidden risks. From a policy standpoint, this research supports the argument that convenience features should be opt-in with visible warnings when security cannot be guaranteed.

If widely adopted, secure filling could reduce successful phishing attacks, decrease credential leaks, and ultimately contribute to a safer web ecosystem.

6. Realization:

This code simulates a Secure Autofill mechanism designed to prevent passwords stored in a password manager from being stolen by malicious JavaScript. It is a simplified implementation based on the protection strategy proposed in Section 5 of the paper.

The code only contains the JS part. To be implemented, the HTML part still need to be completed.

The code:

```
    <script>
    const savedEntry = {
      domain: window.location.hostname,
      formAction: "https://example.com/login",
      fieldName: "userpass",
      password: "mySuperSecret123",
    };

    const dummyValue = "********";
    let isAutofilled = false;
    let realPassword = "";

    function showWarning(msg) {
      document.getElementById("warning").textContent = msg;
    }

    function secureAutofill() {
      const form = document.getElementById("login-form");
      const passwordField =
form.querySelector(`input[name="${savedEntry.fieldName}"]`);

      const formAction = form.action;
      const domain = window.location.hostname;

    // 比對 domain, form.action, password field name 是否和密碼儲存一致，避
免被導向偽裝網頁
      if (
        domain === savedEntry.domain &&
        formAction === savedEntry.formAction &&
        passwordField.name === savedEntry.fieldName
      ) {
        // 填入假密碼，避免 Javascript 調用 input.value 時就看到密碼
        passwordField.value = dummyValue;
        passwordField.setAttribute("data-secure-fill", "true");
        // 將欄位設定為唯讀，不能直接修改
        passwordField.readOnly = true;
        isAutofilled = true;
        realPassword = savedEntry.password;
      } else {
        showWarning("Secure autofill denied: metadata mismatch.");
      }
    }

    document.getElementById("login-form").addEventListener("submit",
function (e) {
      const passwordField =
this.querySelector(`input[name="${savedEntry.fieldName}"]`);

      if (isAutofilled && passwordField.getAttribute("data-secure-fill")
=== "true") {
        // 在送出前，如果 action 或欄位名稱被更動，就阻止送出並發出警告
        if (
          this.action === savedEntry.formAction &&
          passwordField.name === savedEntry.fieldName
        ) {
          // 送出表單前，若所有條件接吻和，將假密碼換成真密碼送出
          passwordField.value = realPassword;
          passwordField.readOnly = false;
        } else {
          e.preventDefault();
          showWarning("Submission blocked: form modified after
autofill.");
        }
      }
    });

    secureAutofill();
  </script>
```