COMP.SEC.300-2024-2025-1 Secure Programming

Nea Parjanen

AbuseIPDB + Spur

IP Checker project

Table of contents

AbuseIPDB+Spur Mozilla IP Checker tool extension

Since I work in Security Operations Center, where suspicious IP addresses are checked very frequently, I had an idea to create a tool that would have multiple IP checker tools in one tool. This project is the starting point for that kind of tool which has already two different websites used to check the IP. In the real world this project should include more technologies and things to check with the IP. But the idea is there.

The Mozilla extension IP checker project works when setup as a temporary extension. Then API key is set in the options.html. There is a video showcasing this in my github (DEMO VID).

After the setup user can highlight any text on their browser and use right click to open the right click shortcut menu and view the “AbuseIPDB+Spur” lookup extension with my icon. The extension can use both IPv4 and IPv6 addresses. The idea is that the user sees at the same time if the IP address is a VPN and from which country it is from.

SOC (Security Operations Center) related to this extension

To explain more deeply why this tool is needed in SOC is that security monitoring is very hectic and everything needs to be done very efficiently. Having tools to give out information about suspicious IP addresses is very important and this is why this tool is valuable. The analysts can draw quick conclusions of events seeing the IP address information very extensively.

Also, this extension is working locally on the user's computer so no data about the IP addresses used is actually logged anywhere, which is also great.

Why this version should not be used in work

This version is just a starting point of how many other tools I could add to it. Also, there are a lot of security measures that should be taken before potentially exposing the used IP addresses to third parties. So even though this code is secure there are security risks such as whether using the API saves the checked IP addresses and AbuseIPDB could make a user profile about which kind of IP addresses I am checking. This could be used as a way to get information about the clients our company has. That would break the NDA rules.

Also, for the same reasons I will not publish this with the Mozilla marketplace even though it could have been a nice learning experience how to do it. I feel like this tool

could be made even better and then used internally in work as a tool and not a general tool people can all use.

User Experience and Interface The popup window is designed to be:

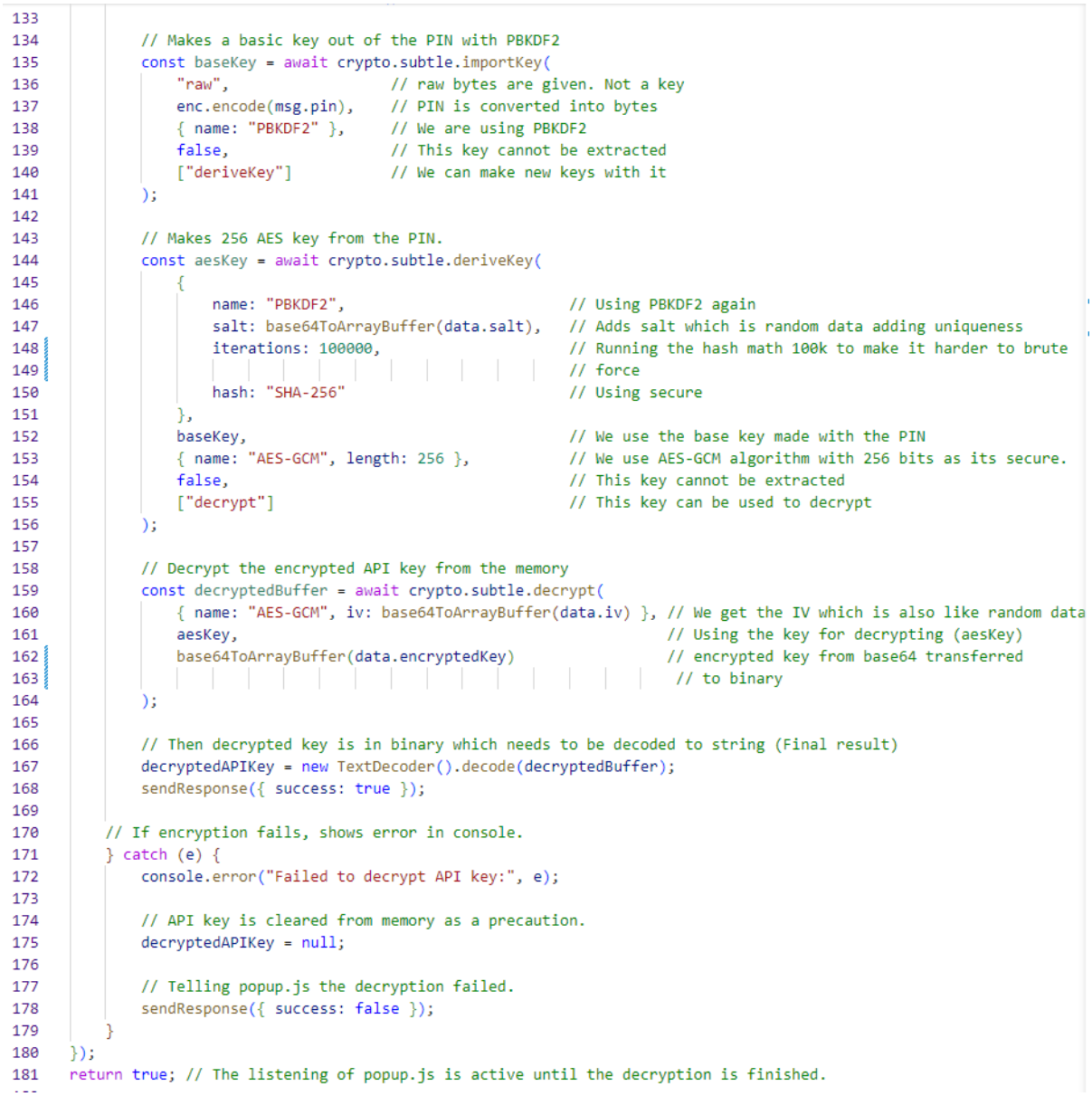
- Dynamic: This extension fetches the window information and adds the right amount of padding. It works on different monitors.

- Efficiency: Fetches the right tab index so user can close the popups with Ctrl + W and continue where they left.

- Fast copying: User can copy AbuseIPDB information such as the ASN, ISP information fast by double clicking and Ctrl + C. (The titles of the AbuseIPDB information cannot be clicked). This was set in the style.css in the “strong” label.

OWASP Top 10 Security Concerns & Fixes

- A02:2021 - Cryptographic Failures
    - Stores API key securely using Web Crypto API to make cryptographic key using the user input PIN and encrypts it using 256-bit AES-GCM symmetric encryption. We make salt and use PBKDF2 with 100 000 iterations and SHA-256 to make the PIN a secure key. Then we use random number IV to make the AES-GCM key. This means the key is very hard to know without knowing the PIN.
    - Considering that it is very rare that an attacker could get remote access or physical access to the device, this is a very secure method to store the API key encrypted in the memory. In the real world it might be unnecessary to use PIN. Accepting the risk. (background.js)



```
133
134         // Makes a basic key out of the PIN with PBKDF2
135         const baseKey = await crypto.subtle.importKey(
136             "raw",                  // raw bytes are given. Not a key
137             enc.encode(msg.pin),    // PIN is converted into bytes
138             { name: "PBKDF2" },     // We are using PBKDF2
139             false,                  // This key cannot be extracted
140             ["deriveKey"]           // We can make new keys with it
141         );
142
143         // Makes 256 AES key from the PIN.
144         const aesKey = await crypto.subtle.deriveKey(
145             {
146                 name: "PBKDF2",                         // Using PBKDF2 again
147                 salt: base64ToArrayBuffer(data.salt),   // Adds salt which is random data adding uniqueness
148                 iterations: 100000,                     // Running the hash math 100k to make it harder to brute
149                                                         // force
150                 hash: "SHA-256"                         // Using secure
151             },
152             baseKey,                                    // We use the base key made with the PIN
153             { name: "AES-GCM", length: 256 },           // We use AES-GCM algorithm with 256 bits as its secure.
154             false,                                      // This key cannot be extracted
155             ["decrypt"]                                 // This key can be used to decrypt
156         );
157
158         // Decrypt the encrypted API key from the memory
159         const decryptedBuffer = await crypto.subtle.decrypt(
160             { name: "AES-GCM", iv: base64ToArrayBuffer(data.iv) }, // We get the IV which is also like random data
161             aesKey,                                               // Using the key for decrypting (aesKey)
162             base64ToArrayBuffer(data.encryptedKey)                // encrypted key from base64 transferred
163                                                                    // to binary
164         );
165
166         // Then decrypted key is in binary which needs to be decoded to string (Final result)
167         decryptedAPIKey = new TextDecoder().decode(decryptedBuffer);
168         sendResponse({ success: true });
169
170     // If encryption fails, shows error in console.
171     } catch (e) {
172         console.error("Failed to decrypt API key:", e);
173
174         // API key is cleared from memory as a precaution.
175         decryptedAPIKey = null;
176
177         // Telling popup.js the decryption failed.
178         sendResponse({ success: false });
179     }
180 });
181 return true; // The listening of popup.js is active until the decryption is finished.
```

- A03:2021 – Injection
    - This code has input validation for IP addresses using regex. (IPv6 regex is long and can't fit the picture well. The code ignores if you try to put anything else highlighted than valid IP address. This has been tested in the test excel. (background.js)

```
// ChatGPT 4o created regex patterns for ipv4:
const ipv4Pattern = /^(25[0-5]|2[0-4]\d|[01]?\d?\d)(\.(25[0-5]|2[0-4]\d|[01]?\d?\d)){3}$/;
// ChatGPT 4o created regex patterns for ipv6:
const ipv6Pattern = /^(?:(?:[0-9A-Fa-f]{1,4}:){7}(?:[0-9A-Fa-f]{1,4}|:)|(?:[0-9A-Fa-f]{1,4}:){6}(?:(?:2

```

    - The AbuseIPDB information escaped using escapehtml. (popup.js)

```
// This function escapes symbols from html which we fetch from AbuseIPDB
function escapeHtml(text) {
    return text.replace(/[&<>"']/g, match => {
        const escapeMap = {
            "&": "&amp;",
            "<": "&lt;",
            ">": "&gt;",
            "\"": "&quot;",
            "'": "&#039;"
        };
        return escapeMap[match] || match;
    });
```

```
if (info.isp) {
    outputHtml += `<p><strong>ISP: </strong>${escapeHtml(info.isp)}</p>`;
}
```

    - We use encodeURIComponent which means that we give the IP as a plaintext so it could not be executed as code. This is kind of unnecessary since the ip is always validated before sending but I liked to include it. (background.js)

```
// here we connect to the API using key. The verbous flag gives us the full country name.
fetch(`https://api.abuseipdb.com/api/v2/check?ipAddress=${encodeURIComponent(ip)}&maxAgeInDays=90&verbose=true`, {
    method: "GET",                    // GET message means we are asking something from the api
    headers: {
        "Accept": "application/json",   // Response is given in json format
        "Key": decryptedAPIKey          // Key is provided with the fetch
    }
```

- A05:2021 Security Misconfiguration
    - In manifest.json we make very secure permissions. The extension only allows using the objects needed like tabs, the shortcut menu, the storage and the two trusted urls used by the extension. Furthermore, the content_security_policy only accepts code from itself, style from itself, connects only to the API url and only accepts images from itself. This means the user cannot inject code from dev tools or other way.

- A07:2021 – Identification & Authentication failures
    - I use PIN to authenticate user every sesion. Credentials are saved using the PBKDF2 and AES-GCM and everything is stored in local memory.
- A08:2021 – Software and Data integrity failures
    - I am not using external libraris or remote code so they cannot modify the code to make it more insecure. All that comes from external sources like the API information escaped and blocked if not behaved correctly.
    - There is also error handling for various parts of the code. For example, when fetching data, we are checking whether the HTTP response is OK before continuing. Also, user errors like wrong PIN are notified to user.
- A10:2021 – Server-Side Request forgery
    - Since the urls for the API connection are hardcoded and permissions set correctly, the attacker cannot make it call any other server than the api that was set.
    - All user input is also validated and handled as plaintext so the attacker cannot modify any code with it.
- A06:2021 – Vulnerable and Outdated Components
    - As stated, before I have no third-party libraries. Also, we use web crypto API which is browsers built in library and not a third-party library. Web Crypto API is handled more reliable security experts, and it gets updates as required.

The other point in the OWASP I didn't find related to this project. The main thing I would make better to have the PIN be more secure password but in real world it would kill the usability of the whole extension. It was very interesting to make overall already a secure

code even more secure, and I would call this project to be a bit overkill on the security. It was very fun to learn the techniques though.

Future development ideas

- More IP lookup technologies

What I Learned

- How to make Mozilla browser extension and testing it
- How to use Web extension shorcut menu and local storage
- How to use the local storage securely using Web Crypto.
- How to use the API for AbuseIPDB (third-party)
- How to validate user input with regex
- How to make UI dynamic
- How to make everything that I did, more secure
- In the end everything was kind of new to me

Use of ChatGPT (GPT-4o)

- Helped emotionally to push through the project
    - For me programming is a heavy source of anxiousness, and I rarely challenge myself to do anything related to programming. For me programming is a trauma and hard topic to work on.

      ChatGPT helped me start this project by convincing me I could do it with its help and learn a lot.
- Made me the base of the extension (What kind of files to start with, how to test the extension in about:debugging page on Mozilla etc.)
- Generated regex for complex IPv4 and IPv6 formats.
- Helped with generating the logic of the code
- Made the whole implementation of AES-GCM and PBKDF2 cryptographic keys logic. It explained to me also how it works.
- All other logic used in the project is heavily the product of AI's logic that I put to work.
- I also made sure to use ChatGPT when making my presentation and documentation for brainstorming purposes. All documentation text is written by me.

Even though I did the majority of my project using ChatGPT 4o I was still learning and asking why everything is done in the way the AI think it should be done. I also saw that there were lot of repetitive code and some nonlogical solutions used which I corrected on my own. I felt very intelligent doing that. I think that I have gone a long way with this project, and I have gained more self-confidence in coding. I think ChatGPT is very good for this to make it more pleasant to work with new and scary things. After you get used to the new things you will start to need ChatGPT less and less. This happens for a simple reason that ChatGPT starts slowly producing worse output than you could do on your own when you become better in what you are doing. I think this project is a good example of that.

Conclusion

This project is a good example of how to make Mozilla extensions secure with API integrations. It is meant to be an IP checker tool in a SOC but by its own it should not be used in real world just yet.