# 208 – Find the cryptocurrency wallet password

## Team Information

**Team Name : LuckyVicky**

**Team Member : Eungchang Lee, Hyun Yi, Juho Heo, Dongkyu Lee**

**Email Address : dfc_luckyvicky@googlegroups.com**

## Instructions

**Description** The police received information that the suspect, a program developer, installed a wallet program on his personal computer and used cryptocurrency. During the search and seizure process, a cryptocurrency wallet program was found, but the password had been set. Even if you ask the suspect, he just says he doesn't know the password. Find the password for the cryptocurrency wallet used by the suspect.

| Target | Hash (MD5) |
|---|---|
| system.ad1 | 9bed2570df00e735ac065c393b11756d |

### Questions

# Please solve all problems based on UTC+9 time zone.

1. What is the name of the cryptocurrency wallet program installed on the computer? (20 points)
2. Where is the wallet password hidden? (60 points)
3. What is the wallet password? (120 points)

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and

results.

- Specify all tools used in deriving the conclusion(s).

## Tools used:

| | | | |
|---|---|---|---|
| Name: | Hashtab | Publisher: | Implbits Software |
| Version: | V6.0.0 | | |
| URL: | http://implbits.com | | |

| | | | |
|---|---|---|---|
| Name: | Visual Studio Code | Publisher: | Microsoft |
| Version: | 1.93.1 | | |
| URL: | https://code.visualstudio.com/ | | |

| | | | |
|---|---|---|---|
| Name: | WinMerge | Publisher: | Dean P.Grimm |
| Version: | 2.16.40 | | |
| URL: | https://winmerge.org/ | | |

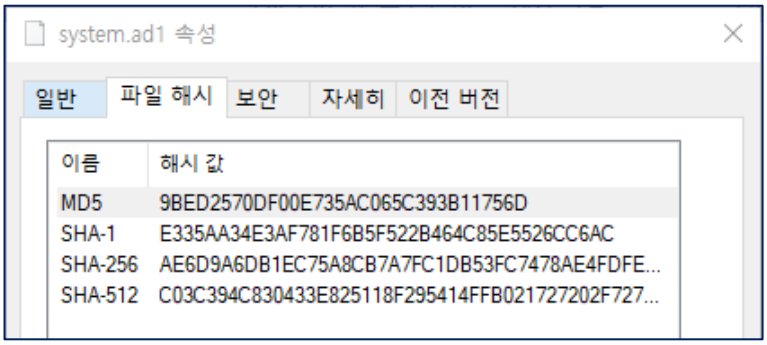
| | | | |
|---|---|---|---|
| Name: | asar | Publisher: | Electron |
| Version: | 3.2.0 | | |
| URL: | https://github.com/electron/asar | | |

| | | | |
|---|---|---|---|
| Name: | FTK Imager | Publisher: | AccessData |
| Version: | 4.7.1.2 | | |
| URL: | https://www.exterro.com/ | | |

| | | | |
|---|---|---|---|
| Name: | DB Browser for SQLite | Publisher: | Mauricio Piacentini |
| Version: | 5.15.2 | | |
| URL: | https://sqlitebrowser.org/ | | |

| | | | |
|---|---|---|---|
| Name: | Python | Publisher: | Python Software Foundation |
| Version: | 3.12.6 | | |
| URL: | https://www.python.org/ | | |

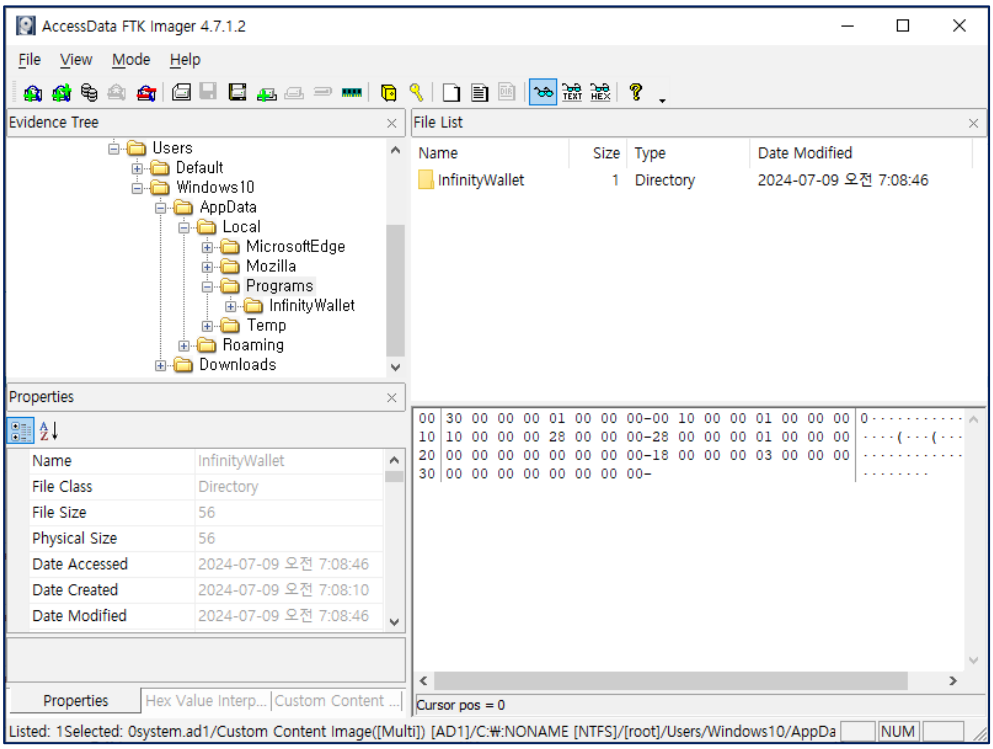
## Step-by-step methodology:




**[그림 1] system.ad1 파일의 해시 값**

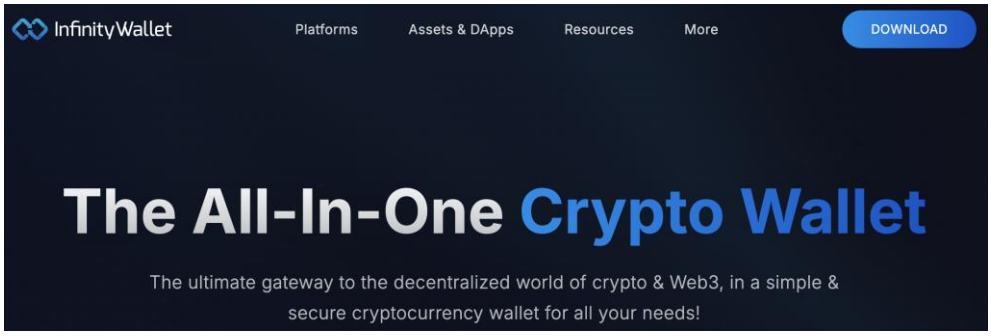
분석 대상 파일에 대한 MD5 해시 값이 일치함을 확인하였습니다.

**1. What is the name of the cryptocurrency wallet program installed on the computer? (20 points)**




**[그림 2] InfinityWallet 프로그램 설치 경로**

Windows 10 사용자의 %localappdata% 경로 내 Programs 폴더에서 InfinityWallet 폴더를 발견했습니다. 해당 폴더의 MAC Time을 분석한 결과, 2024년 7월 9일 오후 4시 8분(UTC+9)에 설치된 것으로 추정됩니다.




**[그림 3] InfinityWallet.io 메인 페이지**

InfinityWallet 공식 홈페이지에서는 이 프로그램이 암호화폐, Web3, 및 Dapp을 관리하는 기능을 가진 암호화폐 지갑이라고 설명하고 있습니다.

## 2. Where is the wallet password hidden? (60 points)

아래에서는 Firefox에서 발견된 문자열 정보와 InfinityWallet에서 발견된 문자열 정보에 대해 설명합니다.

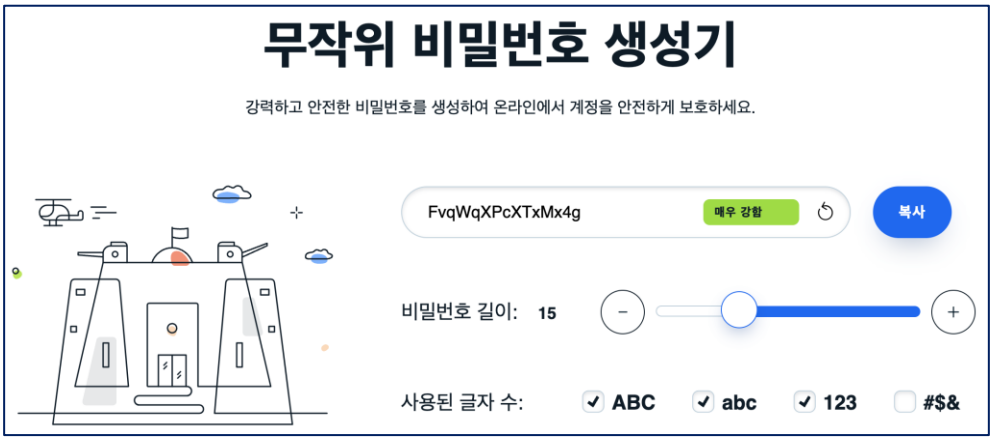
| | |
|---|---|
| Name | acc.Pocket1 |
| File Class | Directory |
| File Size | 56 |
| Physical Size | 56 |
| Date Accessed | 2024-07-11 오전 1:37:12 |
| Date Created | 2024-07-09 오전 7:30:48 |
| Date Modified | 2024-07-11 오전 1:37:12 |
| Encrypted | False |
| Compressed | False |
| Actual File | True |
| Alternate Data Stream Count | 1 |

**[그림 4] acc.Pocket1의 MAC Time**

InfinityWallet 프로그램에서 Pocket을 생성할 때, Roaming 폴더의 InfinityWallet 폴더에 "acc.Pocket{숫자}" 형식의 폴더가 생성됩니다. 제공된 이미지에서는 acc.Pocket1 폴더가 2024년 7월 9일 오후 4시 30분(UTC+9)에 생성된 것으로 나타납니다.

| | | | |
|---|---|---|---|
| 31 | 31 | https://www.avast.com/ko-kr/random-password-generator | 무작위 비밀번호 생성기 \| 12345 … |
| 32 | 32 | https://www.avast.com/ko-kr/random-password-generator#pc | 무작위 비밀번호 생성기 \| 12345 … |

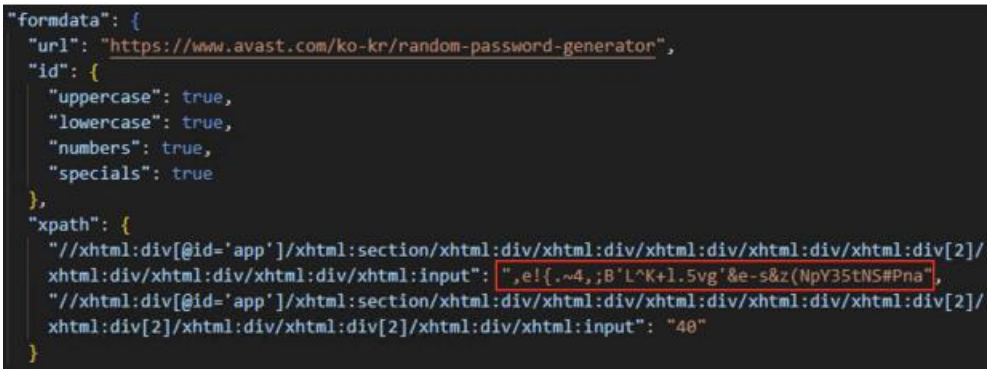
**[그림 5] Firefox 브라우저를 사용하여 방문한 페이지 기록**




**[그림 6] Avast에서 제공하는 무작위 비밀번호 생성기**

Pocket1 생성 시간 이전에 사용자의 Firefox 브라우저 활동 기록을 분석한 결과, Avast의 무작위 비밀번호 생성기 사이트에 접근한 것을 확인할 수 있었습니다. 해당 사이트에서는 1자리에서 50자리까지의 대소문자, 숫자, 특수문자를 조합하여 무작위 비밀번호를 생성하는 서비스를 제공하고 있습니다.

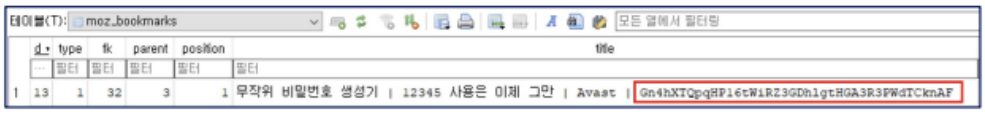

**[그림 7] recovery.jsonlz4 파일에서 발견된 비밀번호 생성 기록 1**




**[그림 8] recovery.baklz4 파일에서 발견된 비밀번호 생성 기록 2**

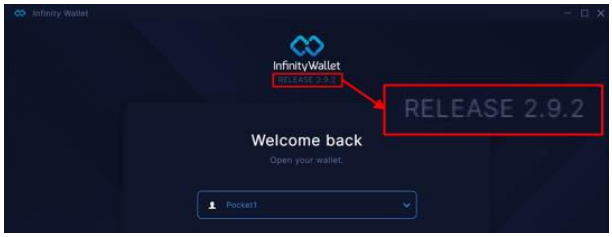
Firefox는 세션 복원 기능이 있어 Firefox 프로필 폴더 내에 세션 데이터를 기록하고 있습니다. 사용자의 Firefox 세션 데이터 중 recovery.jsonlz4 파일과 recovery.backlz4 파일에서 위의 그림과 같은 비밀번호 생성 이력을 볼 수 있습니다.


**[그림 9] targeting.snapshot.json 파일에서 발견된 문자열**




**[그림 10] places.sqlite 파일에서 발견된 문자열**

Firefox의 북마크 기록에서도 비밀번호 성향을 띄는 문자열이 발견되었습니다. 발견된 세 문자열 모두 40자리로 구성되어 있으며, 대소문자, 숫자, 특수문자 등의 조합으로 이루어져 있습니다. 그러나 북마크에 기록된 문자열은 기본적으로 저장된 이름(title) 정보 뒤에 사용자가 임의로 추가한 문자열로 보입니다.



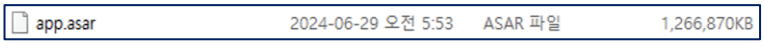

**[그림 11] 사용자의 컴퓨터에 설치된 InfinityWallet의 실행 화면**


**[그림 12] 사용자 폴더 내 InfinityWallet.exe 파일의 버전**

사용자의 컴퓨터에 설치된 InfinityWallet 폴더를 추출하여 실행한 결과, 위 그림과 같이 2.9.2 버전임을 확인할 수 있습니다. 그러나 InfinityWallet.exe 실행 파일의 파일 버전 정보에 따르면, 위 그림과 같이 2.9.5 버전임을 알 수 있습니다. 이로 인해 실행 파일과 구동에 사용되는 파일 간의 버전 불일치가 확인되었고, 이에 대한 분석을 진행하였습니다.

| | |
|---|---|
| Name | app.asar |
| File Class | Regular File |
| File Size | 1,460,252,092 |
| Physical Size | 1,460,252,672 |
| Start Cluster | 111,149 |
| Date Accessed | 2024-07-11 오전 1:11:04 |
| Date Created | 2024-07-02 오전 4:59:18 |
| Date Modified | 2024-07-10 오전 8:11:33 |
| Encrypted | False |

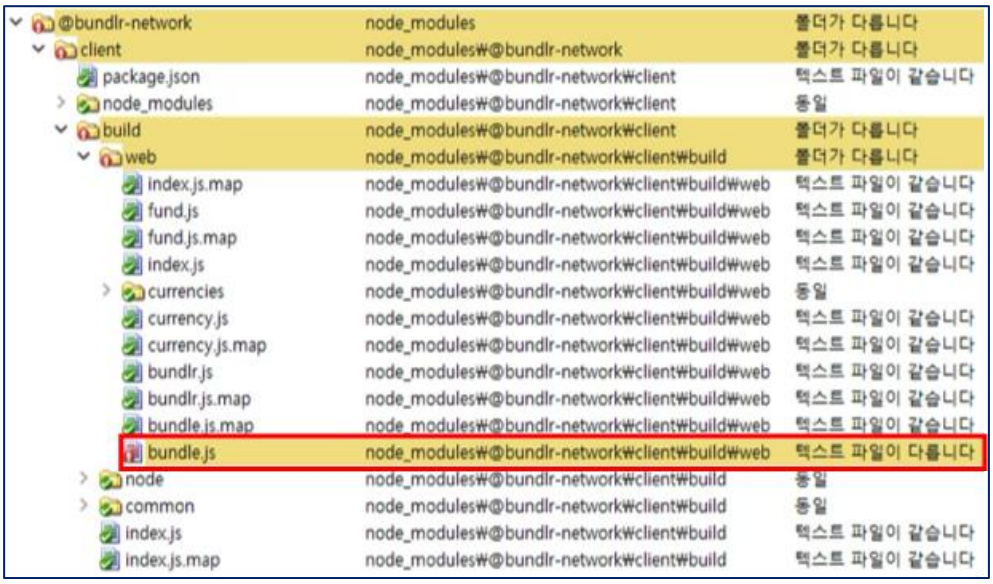
**[그림 13] 사용자 폴더 내 app.asar 파일의 MAC Time 정보**




**[그림 14] 정상적인 InfinityWallet 2.9.5 버전의 app.asar 파일**

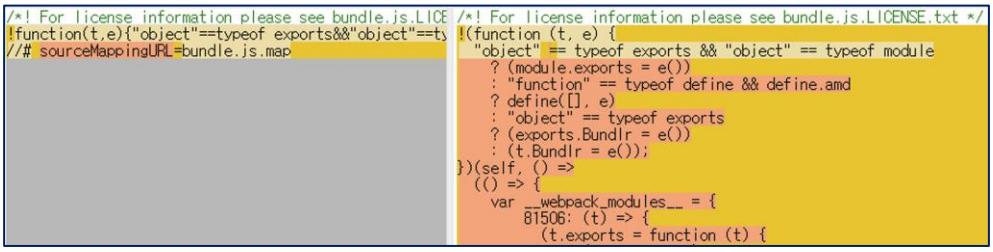
InfinityWallet은 Electron 프레임워크로 만들어진 애플리케이션이며, asar 아카이브를 기반으로 실행됩니다. 그러나 사용자 폴더 내에 존재하던 app.asar 파일과 정상적으로 InfinityWallet 2.9.5 버전을 설치했을 때의 파일 수정 시간이 다르며, 파일 크기도 차이가 나는 것을 확인할 수 있습니다. 즉, 사용자 폴더 내에 존재하는 app.asar 파일은 수정이 발생했음을 알 수 있습니다.

수정된 것으로 추정되는 app.asar 파일은 InfinityWallet 2.9.2를 기반으로 만들어진 것으로 보입니다. 분석을 위해 사용자 폴더 내의 app.asar 파일과 InfinityWallet 2.9.2 버전을 설치한 후 추출한 정상 app.asar 파일을 비교 분석하였습니다. 두 파일 모두 asar 패키지를 사용하여 추출하였습니다.




**[그림 15] 두 파일의 비교 분석**

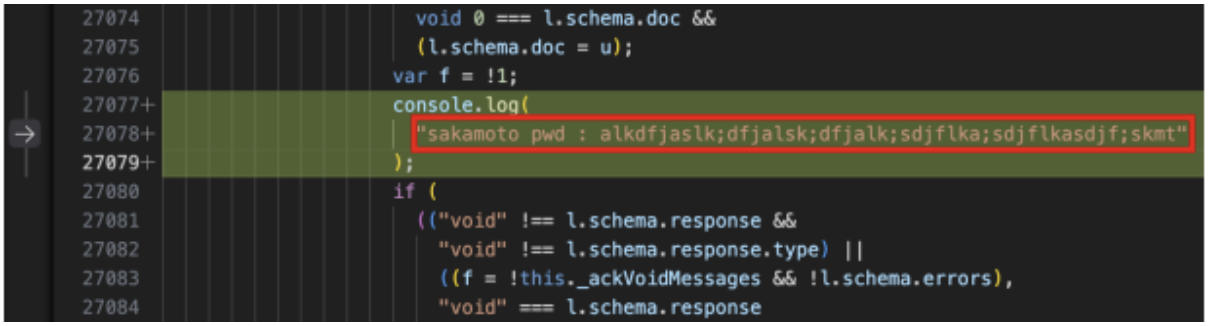
두 파일에서 추출한 폴더와 파일을 비교한 결과, 위 그림과 같이 "bundle.js" 파일 하나가 일치하지 않는 것을 확인할 수 있었습니다.




**[그림 16] 정상 bundle.js(좌), 수정된 bundle.js(우) 비교**

두 파일을 비교해보면 정상 bundle.js 파일은 최적화 및 난독화를 위해 압축(Minify)되어 있는 반면, 수정된 bundle.js 파일은 압축이 해제된 상태임을 확인할 수 있습니다.

정확한 분석을 위해 정상 bundle.js 파일의 코드 압축을 해제하여 분석을 진행하였습니다.




**[그림 17] 수정된 bundle.js 파일에 추가된 코드**

정상 bundle.js 파일의 코드 압축을 해제한 후, 두 파일의 텍스트를 비교한 결과, 위 그림과 같이 특정 구문에서 "sakamoto pwd : alkdfjaslk;dfjalsk;dfjalk;sdjflka;sdjflkasdjf;skmt"를 출력하는 코드가 추가된 것을 확인할 수 있었습니다.

결론적으로 정리하자면, 숨겨진 지갑 패스워드의 경로는 다음과 같습니다.

**Users/Windows10/AppData/Local/Programs/InfinityWallet/resources/app.asar**

**app.asar/node_modules/@bundlr-network/client/build/web/bundle.js**

## 3. What is the wallet password? (120 points)

아래에서는 실제 Pocket1의 비밀번호를 찾기 위한 접근 방법에 대해 설명합니다.

```
165         webPreferences: {
166             webSecurity:!isTest,
167             allowRunningInsecureContent:false,
168             webviewTag: true,
169             backgroundThrottling:false,
170             devTools: true,
171             transparent:false,
172             enableRemoteModule: true,
173             nodeIntegration:true,
174             contextIsolation: false,
175             spellcheck:false
176         },
```

**[그림 18] InfinityWallet 개발자 도구 활성화**

우선, InfinityWallet의 Pocket 생성 및 인증에 사용되는 로직을 분석하였습니다. InfinityWallet 애플리케이션을 디버깅하기 위해 app.asar 파일에서 개발자 도구를 활성화하여 분석을 진행하였습니다.

기본적으로 Pocket의 인증정보를 저장하는 파일은 Pocket 폴더 내에 존재하는 info.infinity 파일입니다. 따라서, 해당 파일이 생성되고 Pocket 인증 시 사용되는 과정에 대해 분석하였습니다.

첫 번째로, Pocket 생성 시 info.infinity가 생성되는 과정입니다.

```
async set(key, val, password) {  key = "account", val = "Pocket11", password = "qwer1234"
    var exist = await this.checkForFile();
    if (exist) {
        var res = await this.decrypt(this.path, password);  res = undefined, password = "qwer1234"
        var toJson = this.default;  toJson = undefined
        if (res.length > 0) {  res = undefined
            toJson = JSON.parse(res);  toJson = undefined
        }
        toJson[key] = val;  toJson = undefined, key = "account", val = "Pocket11"
        await this.encrypt(JSON.stringify(toJson), password, this.path);  password = "qwer1234"
    }
    else {
        var dc = {};  dc = {account: 'Pocket11'}
        dc[key] = val;  key = "account", val = "Pocket11"
        await this.encrypt(JSON.stringify(dc), password, this.path);
    }
```

**[그림 19] 16 바이트 크기의 IV 생성**

Pocket 비밀번호 검증에 사용될 평문 텍스트를 생성합니다. 이 텍스트는 JSON 형식으로 생성되며, 키는 "account", 값은 새로 생성되는 Pocket 이름으로 지정됩니다. 예를 들어, 생성되는 Pocket 이 Pocket11 일 경우, 생성되는 평문 텍스트는 {"account":"Pocket11"}가 됩니다.

```
encrypt(text, password, file) {
    return new Promise((resolve, reject) => {

        function bufferToStream(buffer) {
            let stream = new Duplex();
            stream.push(buffer);
            stream.push(null);
            return stream;
        }
        function getCipherKey(password) {
            return crypto.createHash('sha256').update(password).digest();
        }
        // Generate a secure, pseudo random initialization vector.
        const initVect = crypto.randomBytes(16);

        // Generate a cipher key from the password.
        const CIPHER_KEY = getCipherKey(password);
        var tbuffer = Buffer.from(text, "utf8");

        const readStream = bufferToStream(tbuffer);
        const gzip = zlib.createGzip();
        const cipher = crypto.createCipheriv('aes-256-cbc', CIPHER_KEY, initVect);
        const appendInitVect = new AppendInitVect(initVect);
        // Create a write stream with a different file extension.
        const writeStream = fs.createWriteStream(file);
        writeStream.on('close', function () {

            resolve("completed");
        });
        readStream
            .pipe(gzip)
            .pipe(cipher)
            .pipe(appendInitVect)
            .pipe(writeStream);

    });
}
```

**[그림 20] info.infinity 생성 중 주요 과정**

이후, Info.infinity 파일 생성을 위한 encrypt 함수가 실행됩니다. 앞서 생성한 평문 텍스트를 압축하고 암호화하는 과정을 담고 있습니다.

```
// Generate a secure, pseudo random initialization vector.
const initVect = crypto.randomBytes(16);
```

**[그림 21] 16바이트의 IV 생성**

1. 16바이트 크기의 랜덤 IV를 생성합니다.

```
function getCipherKey(password) {
    return crypto.createHash('sha256').update(password).digest();
}
```

**[그림 22] 평문 비밀번호를 SHA-256 해싱**

2. 사용자가 입력한 평문 비밀번호를 SHA-256 해시 알고리즘으로 해싱하여 암호화에 필요한 32바이트 길이의 키를 생성합니다.

```
var tbuffer = Buffer.from(text, "utf8");

const readStream = bufferToStream(tbuffer);
const gzip = zlib.createGzip();
```

**[그림 23] 평문 텍스트 변환 및 gzip 스트림 생성**

3. 평문 텍스트를 Buffer 객체로 변환하고, 다시 스트림으로 변환합니다.

4. zlib.createGzip() 함수를 사용하여 gzip 압축을 위한 스트림을 생성합니다.

```
const cipher = crypto.createCipheriv('aes-256-cbc', CIPHER_KEY, initVect);
```

**[그림 24] AES-256-CBC 모드 암호화 스트림 생성**

5. 앞서 생성한 CIPHER_KEY와 initVect를 통해 crypto.createCipheriv를 사용하여 AES-256-CBC 모드의 암호화 스트림을 생성합니다.

```
const appendInitVect = new AppendInitVect(initVect);
```

**[그림 25] 초기화 벡터 추가 스트림 생성**

6. AppendInitVect 클래스를 사용하여 암호화된 데이터에 초기화 벡터를 추가하는 스트림을 생성합니다.

```
readStream
    .pipe(gzip)
    .pipe(cipher)
    .pipe(appendInitVect)
    .pipe(writeStream);
```

**[그림 26] 스트림 파이프라인 설정**

7. 각 스트림을 파이프로 연결하여 데이터 흐름을 설정합니다.

해당 과정을 살펴보면, 데이터 처리 순서는 다음과 같습니다: ① 원본 텍스트 데이터를 읽음 {"account":"Pocket11"}, ② 해당 데이터를 Gzip을 통해 압축, ③ 압축된 데이터를 AES 알고리즘으로 암호화, ④ 암호화된 데이터 앞에 IV를 추가, ⑤ 최종 결과를 파일에 저장합니다.

```
// Create a write stream with a different file extension.
const writeStream = fs.createWriteStream(file);
writeStream.on('close', function () {

    resolve("completed");
});
```

**[그림 27] 작업 완료 처리**

8. writeStream의 'close' 이벤트를 리스닝하여 파일 쓰기가 완료되면 Promise를 해결합니다. 이 과정을 거치면 해당 Pocket 폴더에 인증 정보를 포함하는 info.infinity 파일이 생성됩니다.

다음은 사용자가 Pocket에 접근하기 위해 앞서 생성한 info.infinity 파일을 복호화하는 과정을 설명합니다.

```
decrypt(file, password) {

    // First, get the initialization vector from the file.
    function getCipherKey(password) {
        try {
            var createPassword = crypto.createHash('sha256').update(password).digest()
            return createPassword;
        }
        catch{
            return '';
        }

    }
    return new Promise((resolve) => {

        var readInitVect = fs.createReadStream(file, { end: 15 });
        let initVect;
        readInitVect
            .on('data', function (chunk) {
                initVect = chunk;
            })
            .on('close', () => {
                try {
                    const cipherKey = getCipherKey(password);
                    if (cipherKey == '') {
                        resolve('');
                        return;
                    }
                    const readStream = fs.createReadStream(file, { start: 16 });
                    readStream.on('error', function (err) {
                        resolve('')
                    });
                    try {
                        const decipher = crypto.createDecipheriv('aes-256-cbc', cipherKey, initVect);
                        const unzip = zlib.createUnzip();
                        readStream.pipe(decipher).on('error', function (err) {
                            resolve('')
                        }).pipe(unzip).on('error', function (err) {
                            resolve('')
                        }).pipe(concat2(function (error, buffer) {
                            if (error) resolve('');
                            else {

                                resolve(buffer.toString("utf8"));
                            }
                            })).on('error', function (err) {
                                resolve('')
                            });
                    }
                    catch (e) {
                        console.error(e)
                        resolve('');
                    }

                }
                catch (error) {
                    console.error(error)
                    resolve('');
                }

            })
            .on('error', (e) => {
              console.error(e);
              resolve('')
            });
    });
    // Once we've got the initialization vector, we can decrypt the file.
```

**[그림 28] info.infinity 복호화 과정**

복호화 과정은 앞서 설명한 암호화 과정을 반대로 진행합니다. 이 과정은 info.infinity 파일과 비밀번호를 매개변수로 받아 진행되며, 주요 부분은 다음과 같습니다

```
var readInitVect = fs.createReadStream(file, { end: 15 });
let initVect;
```

**[그림 29] IV 추출**

1. info.infinity 파일의 첫 16바이트를 IV로 추출하여 initVect 변수에 저장합니다.

```
.on('close', () => {
    try {
        const cipherKey = getCipherKey(password);
        if (cipherKey == '') {
            resolve('');
            return;
        }
```

**[그림 30] cipherKey 생성**

```
// First, get the initialization vector from the file.
function getCipherKey(password) {
    try {
        var createPassword = crypto.createHash('sha256').update(password).digest()
        return createPassword;
    }
    catch{
        return '';
    }
```

**[그림 31] getCipherKey 함수**

2. 입력된 password를 SHA-256 해시 알고리즘으로 해시하여 32바이트의 암호화 키를 생성합니다.

```
const readStream = fs.createReadStream(file, { start: 16 });
```

**[그림 32] 암호화 데이터 추출**

3. info.infinity 파일에서 IV 영역 이후 17번째 바이트부터 끝까지 읽는 스트림을 생성합니다.

```
const decipher = crypto.createDecipheriv('aes-256-cbc', cipherKey, initVect);
const unzip = zlib.createUnzip();
```

**[그림 33] 복호화 및 압축 해제 스트림 생성**

4. crypto.createDecipheriv를 사용하여 AES-256-CBC 모드의 복호화 객체를 생성합니다.

5. zlib.createUnzip()을 사용하여 gzip 압축 해제를 위한 스트림을 생성합니다.

```
    readStream.pipe(decipher).on('error', function (err) {
        resolve('')
    }).pipe(unzip).on('error', function (err) {
        resolve('')
    }).pipe(concat2(function (error, buffer) {
        if (error) resolve('');
        else {

            resolve(buffer.toString("utf8"));
        }
        })).on('error', function (err) {
            resolve('')
        });
}
```

**[그림 34] 복호화 및 압축해제 파이프라인**

6. 각 스트림을 파이프로 연결하여 데이터 흐름을 설정합니다.

해당 과정을 살펴보면, 데이터 처리 순서는 다음과 같습니다:
① 초기화 벡터 이후의 데이터를 읽어 AES-256-CBC 모드로 복호화
② 복호화된 데이터를 gzip을 통해 압축 해제
③ 최종적으로 얻은 원본 텍스트를 Promise를 통해 반환

위 과정을 통해 정상적으로 복호화에 성공하면, 앞서 Pocket 생성 시 사용되었던 평문 텍스트 {"account":"Pocket11"}를 확인할 수 있습니다. 또한, 앞서 확인한 비밀번호들을 조합하여 info.infinity 파일에 대한 사전 공격을 진행할 수 있습니다.

아래는 사전 공격에 사용된 파이썬 코드입니다.

```
from itertools import permutations
from Cryptodome.Cipher import AES
from Cryptodome.Hash import SHA256
from Cryptodome.Util.Padding import unpad
import gzip
import io
import multiprocessing
from tqdm import tqdm

# 주어진 비밀번호들
passwords = [
    'YG3E31qu6vZG3oaTWiMqKxJlumvY3x6K0saNJ6L3',
    ",e!{.~4,;B'L^K+l.5vg'&e-s&z(NpY35tNS#Pna",
    'Gn4hXTQpqHP16tWiRZ3GDhlgtHGA3R3PWdTCknAF',
    'alkdfjaslk;dfjalsk;dfjalk;sdjflka;sdjflkasdjf;skmt'
]

# 마지막 비밀번호를 ';'로 분리하여 리스트로 만듦
last_passwords = passwords[-1].split(';')

# 모든 비밀번호 목록 업데이트
all_passwords = passwords[:-1] + last_passwords

# 암호화된 파일 경로
encrypted_file_path = 'info.infinity'  # 실제 파일 경로
```

```
# 복호화 함수 정의
def decrypt_file(args):
    password, encrypted_file_data = args
    try:
        # 비밀번호로부터 CIPHER_KEY 생성
        hash_obj = SHA256.new(data=password.encode('utf-8'))
        CIPHER_KEY_bytes = hash_obj.digest()

        # 암호화된 파일에서 초기화 벡터와 데이터 분리
        initVect_bytes = encrypted_file_data[:16]
        encrypted_data = encrypted_file_data[16:]

        # AES-256-CBC 모드로 암호 해독 객체 생성
        cipher = AES.new(CIPHER_KEY_bytes, AES.MODE_CBC, initVect_bytes)

        # 암호 해독 수행
        decrypted_padded_data = cipher.decrypt(encrypted_data)

        # PKCS7 패딩 제거
        decrypted_data = unpad(decrypted_padded_data, AES.block_size)

        # gzip 압축 해제
        with gzip.GzipFile(fileobj=io.BytesIO(decrypted_data)) as gzip_file:
            original_data = gzip_file.read()

        # 원본 텍스트로 디코딩 (UTF-8)
        original_text = original_data.decode('utf-8')

        return password, original_text

    except Exception:
        # 복호화 실패 시 None 반환
        return None

if __name__ == '__main__':
    import os
    from math import factorial

    # 암호화된 파일 내용을 메모리에 로드 (프로세스 간 공유를 위해)
```

```
with open(encrypted_file_path, 'rb') as f:
    encrypted_file_data = f.read()

# 총 시도할 비밀번호 수 계산
n = len(all_passwords)
total_permutations = sum(factorial(n) // factorial(n - r) for r in range(1, n+1))
print(f"[+] 총 시도할 비밀번호 수: {total_permutations}")

# 복호화 성공 여부를 추적하기 위한 변수
found = False

# 각 길이별로 순열 생성 및 복호화 시도
for r in range(1, n+1):
    permutations_r = permutations(all_passwords, r)
    total_r = factorial(n) // factorial(n - r)
    print(f"₩n[-] 길이 {r}인 비밀번호 순열 시도 중 ({total_r}개)...")

    # 멀티프로세싱 Pool을 각 길이마다 생성
    num_processes = multiprocessing.cpu_count()
    pool = multiprocessing.Pool(processes=num_processes)

    # 진행 상황 표시를 위한 tqdm 설정
    with tqdm(total=total_r, desc=f"길이 {r} 처리 중") as pbar:
        # 복호화 작업을 수행하는 제너레이터 함수 정의
        def generate_tasks():
            for pw_perm in permutations_r:
                password = ''.join(pw_perm)
                yield (password, encrypted_file_data)

        # 결과를 비동기적으로 처리
        for result in pool.imap_unordered(decrypt_file, generate_tasks(), chunksize=100):
            pbar.update()
            if result:
                password, original_text = result
                print("₩n₩n[+] 복호화 성공!")
                print(f"[+] 사용된 비밀번호: {password}")
                print(f"[+] 복호화된 텍스트: {original_text}")
                found = True
                break  # 복호화에 성공했으므로 루프 탈출
```

```
            pool.close()
            pool.join()  # 모든 프로세스가 종료될 때까지 대기

        if found:
            break  # 복호화에 성공했으므로 길이 루프 탈출

    if not found:
        print("₩n복호화에 성공하지 못했습니다.")
```

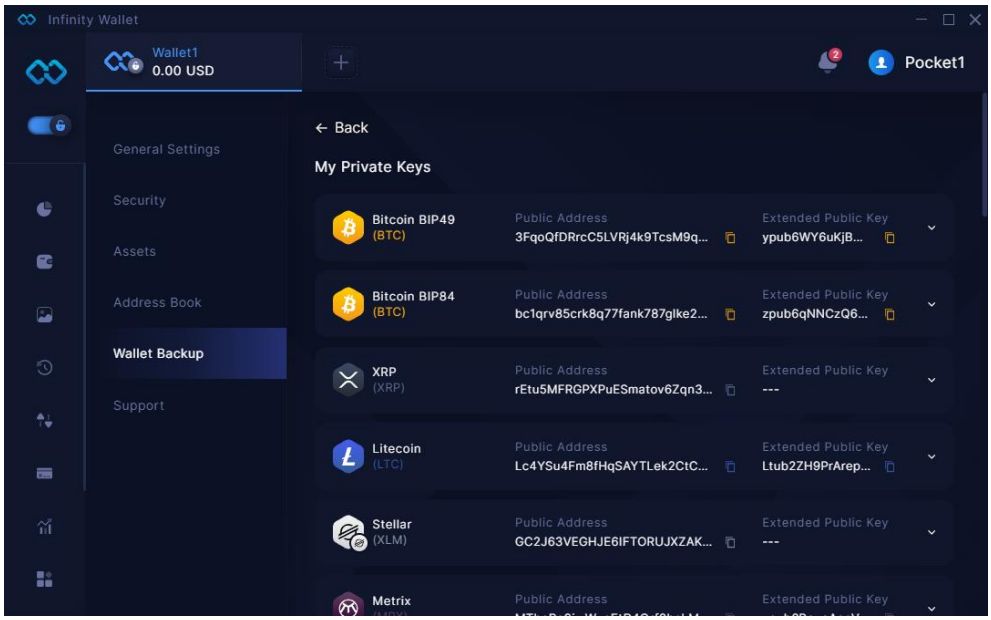
**[표 1] info.infinity 사전 공격 파이썬 스크립트**

4번째 비밀번호의 경우 ";"가 구분자로 사용되는 것으로 추정되어 문자열을 분리하였으며, 총 9개의 문자열을 사용하여 길이가 1인 순열부터 길이가 9인 순열까지 차례로 생성하여 복호화 가능한 키를 찾는 과정을 진행합니다. 생성되는 순열의 개수는 약 98만 개입니다.

```
[+] 복호화 성공!
[+] 사용된 비밀번호: Gn4hXTQpqHP16tWiRZ3GDhlgtHGA3R3PWdTCknAFskmt
[+] 복호화된 텍스트: {"account":"Pocket1","account_id":"8b10e6eb000dff5992ce96ee267c2de13e
a663aae28f4849b05598bcf10c0eeb"}
```

**[그림 35] 복호화 성공 메시지**

해당 파이썬 스크립트를 실행한 결과, 길이가 2인 순열에서 Pocket1의 비밀번호를 찾을 수 있었습니다.




**[그림 36] Pocket1 로그인 화면**

앞서 찾은 비밀번호 **Gn4hXTQpqHP16tWiRZ3GDhlgtHGA3R3PWdTCknAFskmt**를 사용하여 정상적으로 로그인이 가능하였습니다.