

307 – Bitcoin wallet

Team Information

Team Name: DogeCoin

Team Member: Dongbin Oh, Donghyun Kim, Donghyun Kim, Yeongwoong Kim

Email Address: dfc-dogecoin@naver.com

Instructions

Description In order to verify transactions on the Bitcoin network, investigators are sometimes forced to identify Bitcoin addresses on suspects' PCs

Questions

1. Present a methodology for finding Bitcoin addresses in a Windows memory dump extracted from the system the following Bitcoin wallet services were used or being used. (100 points)

Armory	Bitcoin Core	Bither
BitPay	Electrum	

- You need to find out which wallet service the address was used in.
 - You can get points by checking only the addresses generated by the user in the wallet service. We don't want you to create a transaction on the mainnet to solve this problem!
2. Implement the methodology proposed in Q1. (200 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:	Volatility	Publisher:	Volatility Foundation
Version:	3.1.1		
URL:	https://github.com/volatilityfoundation/volatility3		

Name:	Yara	Publisher:	VirusTotal
Version:	4.1.2		
URL:	https://github.com/VirusTotal/yara		

Name:	Gimp	Publisher:	The GIMP Team
Version:	2.10.28		
URL:	https://www.gimp.org/		

Step-by-step methodology:

1. Present a methodology for finding Bitcoin addresses in a Windows memory dump extracted from the system the following Bitcoin wallet services were used or being used. (100 points)

Windows 메모리 덤프에서 Bitcoin 지갑 서비스들에서 사용하는 주소를 찾기 위해서는 그 특성과 정보에 대해 먼저 파악할 필요가 있다. 아래는 제시된 서비스들에 대한 정보 및 특징을 조사하여 나열한 것이다.

Armory	Bitcoin Core	Bither
BitPay	Electrum	

[그림 1] 제시된 Bitcoin wallet Service

■ Bitcoin Service 별 특징

[표 1] Armory 정보 및 특징

프로그램	Armory
암호화 지원	암호화 필수
지갑 주소 생성	Default 지갑 주소 100 개, Receive 버튼 클릭시 1 개 추가
주소 형식	Default 100 개: P2PKH 추가 주소: 선택가능(p2pkh, p2sh-p2pk, p2sh-p2wpkh)
지갑 파일 위치	%appdata%\Roaming\Armory\armory_{wallet_name}.wallet

[표 2] Bitcoin Core 정보 및 특징

프로그램	Bitcoin Core
암호화 지원	암호화 선택 가능
지갑 주소 생성	Receive 시마다 1 개씩 생성
주소 형식	Default: Bech32 Non-bech32(p2sh) 선택 가능
지갑 파일 위치	%appdata%\Roaming\Bitcoin\wallets\{wallet_name}\wallet.dat

[표 3] Electrum 정보 및 특징

프로그램	Electrum
암호화 지원	암호화 선택 가능
지갑 주소 생성	Default 지갑 주소 Receiving Address 20 개, Change Address 10 개
주소 형식	Bech32 형식
지갑 파일 위치	%appdata%\Roaming\Electrum\wallets\default_wallet

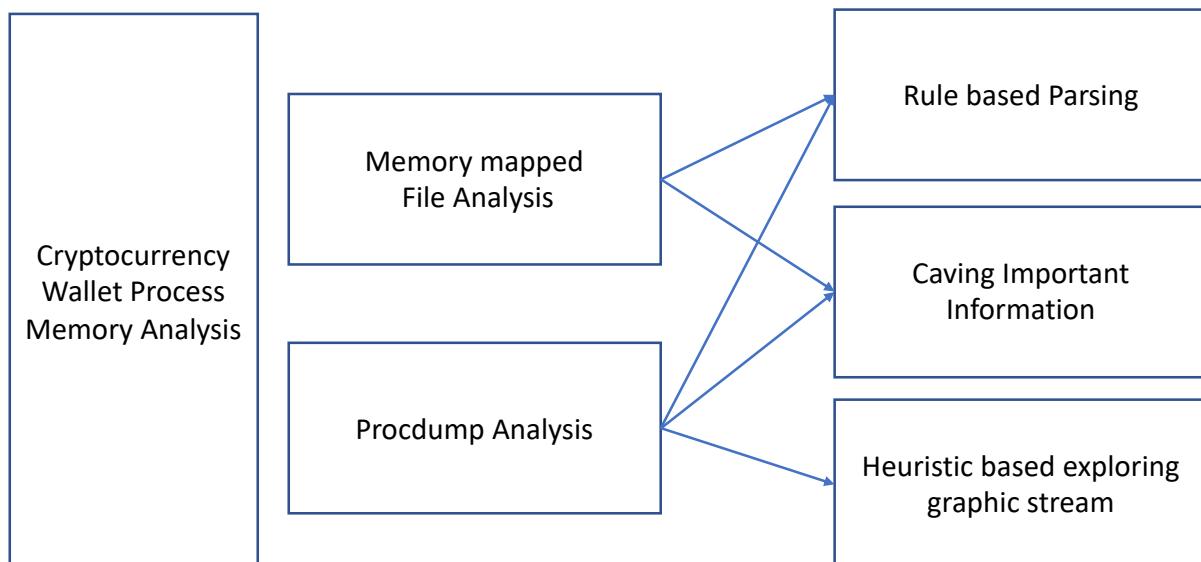
[표 4] Bitpay 정보 및 특징

프로그램	Bitpay
암호화 지원	암호화 선택 가능
지갑 주소 생성	Receive 시마다 1개씩 생성
주소 형식	Bech32 형식
지갑 파일 위치	지갑 파일 없음, c:\users\user\.bitpay\app 폴더에 관련 파일 저장 (chrome 과 유사한 구조) Users\{user}\.bitpay\ app\Local Storage\leveldb\00000n.log 에 일부 주소 목록 저장되는 것 확인

[표 5] Bither 정보 및 특징

프로그램	Bither
암호화 지원	암호화 필수
지갑 주소 생성	1개씩 생성
주소 형식	P2PKH
지갑 파일 위치	C:\Users\IEUser\AppData\Roaming\Bither\wallet

본 문항을 해결하기 위해 아래의 방법론을 제안하였다.



[그림 2] 제안 Methodology 도식

방법론의 각 요소에 대한 설명은 아래와 같다.

■ Memory Mapped File Analysis

본 방법은 volatility3의 Filescan을 이용한 방법이다. Filescan의 경우 pool 태그 스캐닝을 사용하여 물리적 메모리에서 FILE_OBJECT 들을 찾아내는 방법이다. 프로세스에서 사용하고 있는 파일을 목록을 알 수 있으며 구체적으로는 FILE_OBJECT 의 물리적 오프셋, 파일이름, 오브젝트에 연결된 포인터 수, 오브젝트의 핸들 수, 오브젝트의 실질적 권한들이 보여질 것이다.

이후 관련된 파일을 덤프 하여 구조 분석을 통해 wallet의 Address를 획득할 수 있다.

Volatility 3 기준 Filescan을 사용하는 방법은 다음과 같다. Wallet과 관련 있는 파일을 습득하기 위해서는 키워드 기반의 grep을 이용할 수 있다.

[표 6] wallet 파일 탐색

```
vol.py -f "/path/to/file" windows.filescan | grep wallet.dat
```

이후 해당 파일을 dump 하기 위해서는 아래의 방법을 사용하여 한다.

[표 7] wallet 파일 덤프

```
vol.py -f "/path/to/file" -o "/path/to/dir" windows.dumpfiles --virtaddr <offset>
```

■ Procdump Analysis

[昱 8] Process Dump

```
vol.py -f "/path/to/file" -o "/path/to/dir" windows.memmap --dump --pid <PID>
```

■ Rule based Parsing & Carving Important Information

30C20980	52 52 38 69 4A 38 37 69 6F 6F 71 68 72 54 33 33	RR8iJ87hooghrT33
30C20990	44 43 00 00 00 00 00 00 63 17 A4 05 00 07 00 8E 31 39 4C 31 6D 69 36 4E 43 70 6B 71 79 66 68 70	DC.....c.H....ž
30C209A0	31 39 4C 31 6D 69 36 4E 43 70 6B 71 79 66 68 70	15L1ml6NCpkqyfhp
30C209B0	6D 35 75 54 4D 7A 56 46 68 48 56 51 5A 4A 46 72	mSuTMzVFhHVQZUFr
30C209C0	6B 42 00 00 00 00 00 00 66 17 A1 05 00 08 00 8E	kB.....f.i....ž
30C209D0	31 46 4C 77 63 31 52 71 56 39 42 5A 4A 41 66 77	1FLwclRqV9BZJafw
30C209E0	4C 65 51 52 4A 31 31 36 4D 70 33 52 57 79 67 39	LeQRJ1lMg3RNWyg9
30C209F0	58 72 00 00 00 00 00 00 65 17 A2 05 00 09 00 8E	Xr.....e.c....ž
30C20A00	31 42 67 41 72 42 48 33 72 34 63 73 6E 34 76	1BgArBH3rs4csn4v
30C20A10	6D 73 38 6A 67 57 7A 31 6E 46 74 36 48 34 66 38	ms8jgWzlnFt6H4f8
30C20A20	78 37 00 00 00 00 00 00 58 17 9F 05 00 0A 00 80	x7.....X.Y....€
30C20A30	63 35 63 34 34 32 62 66 35 65 30 38 31 31 32 32	c5c442bf5e081122
30C20A40	61 39 30 62 00 45 69 57 63 6F 4E 50 7A 58 4A 31	a90b.E1WcoNpzXJ1
30C20A50	77 6E 00 00 00 00 00 00 5F 17 98 05 00 0B 00 8E	wn....._....ž
30C20A60	31 4E 32 68 6B 67 50 38 38 45 41 59 54 38 37 52	1N2hkpgP8EAYT87R
30C20A70	58 57 62 59 57 34 57 62 56 6A 72 74 37 35 69 4D	XNbYW4WBVjrt75iM
30C20A80	48 4C 00 00 00 00 00 00 52 17 95 05 00 0C 00 90	HL.....R.*....
30C20A90	62 35 62 34 34 32 62 66 35 65 30 38 31 31 32 32	-f-1420-45-00120

[그림 3] Proces Dump 를 통해 확인한 Wallet Address 매핑 구조

메모리에 Mapping된 파일이나 Procdump를 통해 수집한 Process dump, 또는 full memory dump에 대해서는 정규표현식을 이용하여 wallet address라고 추정되는 것을 선별할 수 있다. 현재까지 사용되는 주소로는 3가지로 P2PKH, P2SH, Bech32가 있으며¹ 각각의 포맷을 이용해 String Search 기반으로 우선 선별하였다. 뿐만 아니라 실험 중 생성한 지갑 주소의 list를 사전에 보관하여, 정규표현식에 match 되는 결과와 비교 후 아래와 같이 False Positive를 줄이기 위해 별도의 Rule을 마련하였다.

OC49E0EO	22 3A 20 7B 0D 0A 20 20 20 20 20 20 20 20 20 20 22 62	" : { .. }
OC49EOF0	63 31 71 35 37 78 6D 6B 67 6A 6A 70 73 36 73 63	f1q57xmk gjjps6s
OC49E100	66 33 61 32 35 35 78 61 61 68 75 37 66 37 37 67	f3a2u5xaahu7f77g
OC49E110	78 63 38 32 39 75 34 35 75 22 3A 20 7B 0D 0A 20	kc829u45t : { ..
OC49E120	20 20 20 20 20 20 20 20 20 20 22 61 6D 6F 75	"amou
OC49E130	6E 74 5F 73 61 74 22 3A 20 30 2C 0D 0A 20 20 20	nt_sat": 0, ..
OC49E140	20 20 20 20 20 20 20 20 20 22 62 69 70 37 30 22	"bip70"
OC49E150	3A 20 6E 75 6C 6C 2C 0D 0A 20 20 20 20 20 20 20	: null, ..
OC49E160	20 20 20 20 22 65 78 70 22 3A 20 38 36 34 30	"exp": 8640
OC49E170	30 2C 0D 0A 20 20 20 20 20 20 20 20 20 20 20 20	0, ..
OC49E180	22 68 65 69 67 68 74 22 3A 20 37 30 30 38 32 31	"height": 700821
OC49E190	2C 0D 0A 20 20 20 20 20 20 20 20 20 20 20 20 22	, .. "
OC49E1A0	69 64 22 3A 20 22 31 66 35 64 63 35 63 61 31 38	id": "1f5dc5ca18
OC49E1B0	22 2C 0D 0A 20 20 20 20 20 20 20 20 20 20 20 20	"..,
OC49E1C0	22 6D 65 73 73 61 67 65 22 3A 20 22 22 2C 0D 0A	"message": "", ..
OC49E1D0	20 20 20 20 20 20 20 20 20 20 20 22 6F 75 74	"out
OC49E1E0	20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20	"out..", ..

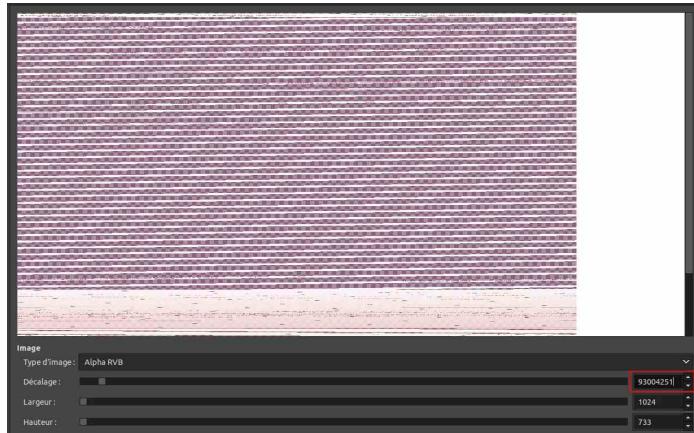
[그림 4] Hex Pattern 탐색

Rule을 구성하기 위해서는 XML과 JSON과 같이 데이터 구조 내에 존재하는 규칙성을 발견하는 것이 일반적이고 추가적으로 지갑 주소 앞 뒤로 존재하는 Hex Pattern을 일반화하였다.

¹ https://en.bitcoin.it/wiki/Invoice_address

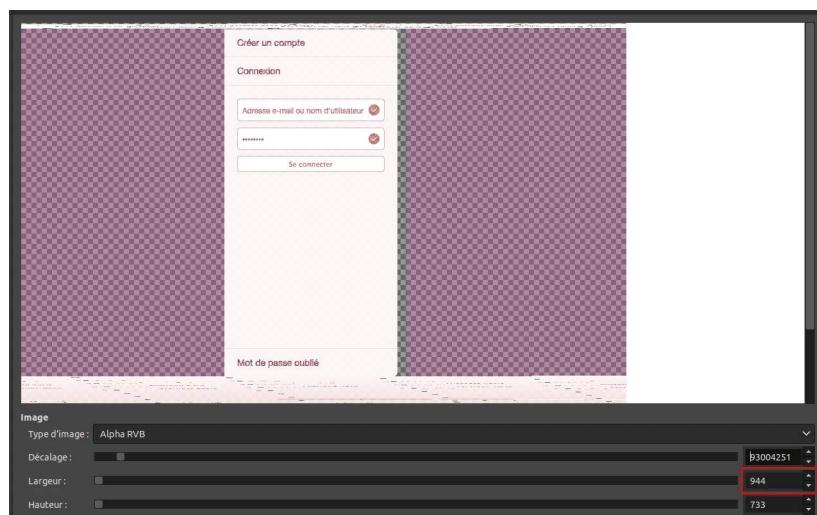
■ Heuristic based exploring graphic stream

본 방법은 CTF에서 주로 사용되는 방법²으로 앞서 수집한 Process memory dump를 활용한다. Procdump를 통해 추출된 Process memory dump 내의 Graphic 관련 데이터를 이용한 방법으로, 제한적인 환경에서 잔존하는 사용자의 화면 관련 데이터를 나타낸다. GIMP라는 도구를 이용하여 Process memory dump를 raw input 상태로 여는 것이 필요하다. 이후 Image Type은 RGB Alpha, width와 height는 적의하도록 조정한다.



[그림 5] 이미지 조정

이후 Displacement Bar를 조정하여 원하는 그래픽이 나올 것 같은 부분을 capture 한 뒤 추가적으로 width와 height를 추가 조정하여 아래와 같이 원하는 화면을 얻을 수 있다.



[그림 6] 복구된 화면

위와 같은 방법은 의도치 않은 유저의 시스템 정보³ 등을 찾을 수 있다는 점이 이점이지만, 그 원리가 정확하지 않고, 자동화가 불가능하다는 한계가 있다.

² <https://developpaper.com/ctf-realizing-windows-memory-forensics-with-volatility-and-gimp/>

³ <https://w00tsec.blogspot.com/2015/02/extracting-raw-pictures-from-memory.html>

2. Implement the methodology proposed in Q1. (200 points)

본 항목에서는 앞서 서술한 방법론들을 기반으로 각 Bitcoin Wallet Service가 사용되거나 종료된 형태의 메모리 덤프에서 어떻게 활용이 가능한지 서술한다. 또한 자동화가 가능한 영역에 대해서는 Tool의 형태로 구현함에 사용된 알고리즘에 대해 서술한다.

■ Bitcoin Core

[표 9] Bitcoin Core 비교 정보

프로그램	Bitcoin_core
종료 덤프	<ul style="list-style-type: none">■ 주소 스트링 Search 일부 성공■ 지갑 파일 Dump 가능
실행 덤프	<ul style="list-style-type: none">■ 주소 스트링 Search 성공■ 지갑 파일 Dump 불가■ Gimp 시도 시, 화면 획득 가능하나 지갑 주소 노출이 필요■ 아래 첨부된 형식 유지되며 주소 값 기록됨
비고	<ul style="list-style-type: none">■ 암호화와 무관

(1) Filescan & Dump

메모리 상 Mapping된 파일을 이용하여 분석하기 위해 Filescan 플러그인을 이용하여 어떠한 파일을 dump할 수 있는지 선별하였다. 앞서 지갑 주소가 포함된 파일을 선별하였기 때문에 해당 파일이 메모리에 존재하는지 확인한 후 덤프하여 지갑 주소가 존재하는지 여부를 파악하면 된다.

Bitcoin core의 경우 지갑 파일의 주소가 AppData\Roaming\Bitcoin\wallets 안에 존재하므로, 해당 경로 중 “wallet” 키워드를 이용하여 필터링을 진행하였다. 해당 결과는 아래와 같다.

■ 프로세스가 Live 상태인 Memory Dump

```
(volatility3_env) heero@ubuntu:~/DFC-2021-501$ python vol.py -f bitcoin/image/bitcoin_core_enc.vmem filescan | grep -i "wallet"
0xc70faf9a87c0.0\Users\Dogecoin\AppData\Roaming\Bitcoin\wallets\wallet_name_test_encrypted\db.log      216
0xc70fb13b2360  \Users\Dogecoin\AppData\Roaming\Bitcoin\wallets\wallet_name_test_encrypted\database\log.000000
0003    216
0xc70fb13b2680  \Users\Dogecoin\AppData\Roaming\Bitcoin\wallets\wallet_name_test_encrypted\database\log.000000
0003    216
(volatility3_env) heero@ubuntu:~/DFC-2021-501$
```

[그림 7] Live Process 에서의 Wallet 파일 덤프

■ 프로세스가 Dead 상태인 Memory Dump

```
(volatility3_env) heero@ubuntu:~/DFC-2021-501$ python vol.py -f bitcoin/image/bitcoin_core_60sec_wallet_dat.vmem windows.filesca
n | grep "wallet"
0xc70fb0509bc0.0\Users\Dogecoin\AppData\Roaming\Bitcoin\wallets\wallet_name_test_encrypted\wallet.dat   216
```

[그림 8] Dead Process 에서의 Wallet 파일 덤프

Process 상태가 Dead일 때 Bitcoin Core의 address 주소를 담고 있는 wallet.dat 파일이 덤프 가능한 것을 확인할 수 있다. 해당 파일을 덤프하여 아래와 같이 정규표현식을 통해 중요 정보를 Carving하여 지갑 파일을 복구할 수 있다.

```
(base) sin90@odongbin-ui-MacBookPro vol13 % python vol.py -f bitcoin_core_unenc_30sec.vmem windows.filescan | grep wallet
0xc70fb09df9f0.0\Users\Dogecoin\AppData\Roaming\Bitcoin\wallets\wallet_name_test\wallet.dat      216
(base) sin90@odongbin-ui-MacBookPro vol13 % python vol.py -o . -f bitcoin_core_unenc_30sec.vmem windows.dumpfiles --virtaddr=0xc70fb09df9f0
Volatility 3 Framework 1.1.1
Progress: 100.00          PDB scanning finished
Cache   FileObject     FileName        Result
DataSectionObject 0xc70fb09df9f0_wallet.dat      file.0xc70fb09df9f0.0xc70fab472590.DataSectionObject.wallet.dat.dat
(base) sin90@odongbin-ui-MacBookPro vol13 %
```

[그림 9] Dead Process에서 Wallet Dump 성공 화면

(2) Process dump

메모리 상 Mapping된 파일을 이용하여 분석하기 위해 Filescan 플러그인을 이용하여 어떠한 파일을 dump할 수 있는지 선별하였다. 앞서 지갑 주소가 포함된 파일을 선별하였기 때문에 해당 파일이 메모리에 존재하는지 확인한 후 덤프하여 지갑 주소가 존재하는지 여부를 파악하면 된다.

■ 프로세스가 Live 상태인 Memory Dump

000CE6D0	3A FA 57 20 B6 78 0E CD 47 B4 CA 21 6E 00 01 6D	:úW ¶x.ÍG'Ê!n..m
000CE6E0	01 00 00 00 09 00 00 00 00 00 00 00 E1 52 45 61áREa
000CE6F0	01 00 00 00 22 33 46 37 47 47 56 41 4E 78 36 38"3F7GGVANx68
000CE700	33 4E 63 77 72 63 52 4C 47 31 71 38 46 74 59 5A	3NcwrcRLG1q8FtYZ
000CE710	59 73 4D 36 73 54 32 14 6C 61 62 65 6C 5F 74 65	YsM6sT2.label_te
000CE720	73 74 5F 6E 6F 6E 42 65 63 68 5F 33 00 E0 D2 9F	st_nonBech_3.àÖÝ
000CE730	80 0E 00 00 16 6D 65 73 73 61 67 65 5F 74 65 73	€...message_tes
000CE740	74 5F 6E 6F 6E 42 65 63 68 5F 33 00 00 02 00 00	t_nonBech_3.....
000CE750	30 00 01 08 64 65 73 74 64 61 74 61 22 33 46 37	0...destdata"3F7
000CE760	47 47 56 41 4E 78 36 38 33 4E 63 77 72 63 52 4C	GGVANx683NcwrcRL
000CE770	47 31 71 38 46 74 59 5A 59 73 4D 36 73 54 32 03	G1q8FtYZYsM6sT2.
000CE780	72 72 39 B4 08 00 01 07 72 65 63 65 69 76 65 4E	rr9'....receiveN
000CE790	2B 00 01 07 70 75 72 70 6F 73 65 22 33 46 37 47	+...purpose"3F7G
000CE7A0	47 56 41 4E 78 36 38 33 4E 63 77 72 63 52 4C 47	GVANx683NcwrcRLG
000CE7B0	31 71 38 46 74 59 5A 59 73 4D 36 73 54 32 00 00	lq8FtYZYsM6sT2..
000CE7C0	6E 00 01 6D 01 00 00 00 08 00 00 00 00 00 00 00	n..m.....
000CE7D0	CB 52 45 61 01 00 00 00 22 33 36 4C 68 63 50 62	ÉREa...."36LhcPb
000CE7E0	64 6E 33 44 72 75 58 39 41 52 65 31 66 70 52 52	dn3DruX9ARelfpRR
000CE7F0	59 4E 4D 48 32 73 63 41 67 47 45 14 6C 61 62 65	YNMH2scAgGE.labe
000CE800	6C 5F 74 65 73 74 5F 6E 6F 6E 42 65 63 68 5F 32	l_test_nonBech_2
000CE810	00 E0 D2 9F 80 0E 00 00 16 6D 65 73 73 61 67 65	.àÖÝ€...message
000CE820	5F 74 65 73 74 5F 6E 6F 6E 42 65 63 68 5F 32 00	_test_nonBech_2.

[그림 10] Live Process 상의 Wallet 데이터

Dump 내에 실험에 지갑 주소로 확인되는 문자열들을 발견하였고 아래와 같은 규칙성을 발견할 수 있었다.

26C6F0F0	01 00 00 00 09 00 00 00 00 00 00 01 E1 52 45 61áREa
26C6F100	01 00 00 00 22 33 46 37 47 47 56 41 4E 78 36 38"3F7GGVANx68
26C6F110	33 4E 63 77 72 63 52 4C 47 31 71 38 46 74 59 5A	3NcwrcRLG1q8FtYZ
26C6F120	59 73 4D 36 73 54 32 14 6C 61 62 65 6C 5F 74 65	YsM6sT2.label_te
26C6F130	73 74 5F 6E 6F 6E 42 65 63 68 5F 33 00 E0 D2 9F	st_nonBech_3.àÖÝ
26C6F140	80 0E 00 00 16 6D 65 73 73 61 67 65 5F 74 65 73	€...message_tes
26C6F150	74 5F 6E 6F 6E 42 65 63 68 5F 33 00 00 00 19 88	t_nonBech_3....^

[그림 11] 특정 구조체의 형태를 지니는 Wallet 데이터

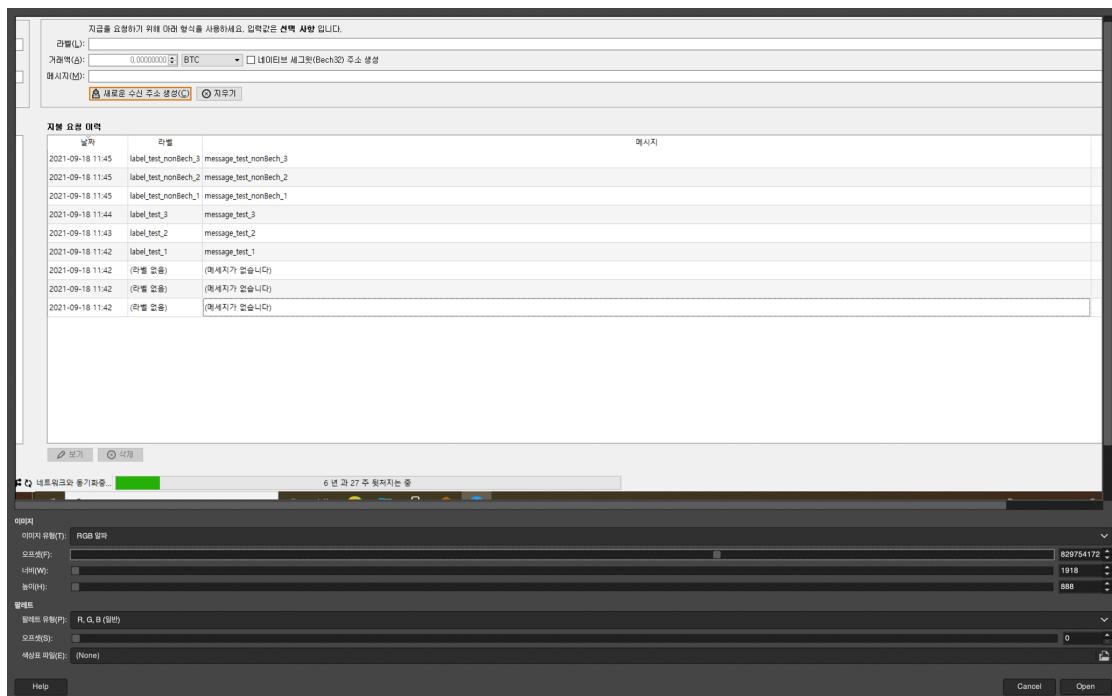
Memory dump 내에서 발견되는 지갑 주소의 값 특징을 정리하면 정규 표현식으로 탐지된 offset 의 이전 21바이트가 각각 차례 순으로 [01 00 00 00] [index 관련 8 바이트][Timestamp 4 바이트][01 00 00 00 22 or 2a (주소 플래그, 22 : non bech 형식, 2a : bech 형식] 으로 되어 있음을 확인할 수 있다. 이후에는 지갑 주소가 ASCII 문자열로 적혀있으며 label 데이터와 메세지 데이터가 뒤따르는 것을 확인할 수 있다.

■ 프로세스가 Dead 상태인 Memory Dump

프로세스를 종료 30초 지난 후 dump를 수행한 것으로 실험을 진행하였는데 volatility를 이용한 프로세스 식별 및 Memory dump 내에서도 지갑 주소가 발견되지 않음을 확인하였다.

(3) Graphic Analysis

항목 2를 통해 획득한 Process Memory Dump를 GIMP를 통해 Open 하였다. Vol3 환경에서 Process memory dump를 얻는 방법은 live process의 메모리 덤프 밖에 없으므로 Live Process에서만 진행하였다. 실험에서 사용한 해상도인 1918*888과 이미지 유형 RGB Alpha를 세팅해주고, 오프셋을 조절하여 적절한 이미지가 나올 때 까지 조정하였다. 그 결과 아래와 같은 이미지를 얻을 수 있었다.



[그림 12] Memory Dump로부터 복원한 Bitcoin Core Graphic File

Bitcoin Core의 경우 지갑 주소가 노출 될 때는 received 기능을 사용할 때인데, 덤프 당시 시점에서는 해당 기능을 사용하고 있지 않으면 위의 그림과 같이 주소를 복구할 수 없다는 한계가 있다.

■ Bitpay

[표 10] Bitpay 비교 정보

프로그램	Bitpay
종료 덤프	<ul style="list-style-type: none">■ 주소 스트링 Search 일부 성공■ 지갑 파일 덤프 불가하나 주소 기록된 파일 덤프 가능■ Users\{user}\.bitpay\ app\Local Storage\leveldb\00000n.log■ 종료 직후에만 가능, 종료 후 수 초 이상 경과 및 프로세스까지 소멸하면 덤프 및 주소 스트링 Search 불가
실행 덤프	<ul style="list-style-type: none">■ 주소 스트링 Search 성공■ 지갑 파일 덤프 불가■ Gimp 가능
비교	<ul style="list-style-type: none">■ 2 가지 형태의 json 정보 발견■ 주소 발급 후 시간이 경과됨에 따라 소멸되는 경우도 있는 것으로 보임

(1) Filescan & dump

■ 프로세스가 live 상태인 memory dump

메모리 상 Mapping된 파일을 이용하여 분석하기 위해 Filescan 플러그인을 이용하여 어떠한 파일을 dump할 수 있는지 선별하였다. 앞서 지갑 주소가 포함된 파일을 선별하였기 때문에 해당 파일이 메모리에 존재하는지 확인한 후 덤프하여 지갑 주소가 존재하는지 여부를 파악하면 된다. Bitpay의 경우 wallet.dat 와 같은 별도의 지갑파일이 존재하지 않으며 c:\users\user\.bitpay\app 폴더에 관련 파일 저장하고 있다. 특히 Users\{user}\.bitpay\ app\Local Storage\leveldb\00000n.log에 일부 주소 목록 저장되는 것 확인하였다. 이에 Volatility의 filescan 플러그인의 필터링 키워드를 “bitpay”로 설정하고 지갑 주소를 확인할 수 있는 \leveldb\00000n.log 파일이 Mapping 되어있는지 확인하였다. 결과는 다음 그림과 같다.

■ 프로세스가 Live 상태인 Memory Dump

```
(volatile)llyt3_evm) heero@ubuntu:~/DFC-2021-501$ python vol.py -f bitcoin/image/bltpay_enc.vmem windows.filescan | grep -i "bitpay"
0xcb0d409e21250_0 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\resources\app.asar 216
0xcb0d409e226a0 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\resources\app.asar 216
0xcb0d40ab08b60 [Users\dogeCoin.\bitpay]\app\Local_Storage\[LevelDB]\CURRENT 216
0xcb0d40ab0e440 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\BitPay.exe 216
0xcb0d40ab0c4840 [Users\dogeCoin.\bitpay]\app\Local_Storage\[LevelDB]\MANIFEST-000001 216
0xcb0d40ac02810 [Users\dogeCoin.\bitpay]\app\GPUCache\data_3 216
0xcb0d40a7026e0 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\resources\app.asar 216
0xcb0d40a7037C0 [Users\dogeCoin.\bitpay]\app\Cache\Index 216
0xcb0d40a7040a0 [Users\dogeCoin.\bitpay]\app\lockfile 216
0xcb0d40a70e210 [Users\dogeCoin.\bitpay]\app\Local_Storage\[LevelDB]\MANIFEST-000001 216
0xcb0d40a711730 [Users\dogeCoin.\bitpay]\app\Code_Cache\js\Index 216
0xcb0d40a715bf0 [Users\dogeCoin.\bitpay]\app\Code_Cache\wasn't\Index 216
0xcb0d40a71aba0 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\icudtl.dat 216
0xcb0d40a71caee0 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\v8_context_snapshot.bin 216
0xcb0d40a71ea20 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\chrome_100_percent.pak 216
0xcb0d40a71f1f0 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\chrome_200_percent.pak 216
0xcb0d40a71f5b0 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\resources.pak 216
0xcb0d40a720196 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\locales\ko.pak 216
0xcb0d40a7247e0 [Users\dogeCoin.\bitpay]\app\Cache\index 216
0xcb0d40a727906 [Users\dogeCoin.\bitpay]\app\Code\data_2 216
0xcb0d40a728c0 [Users\dogeCoin.\bitpay]\app\Cache\Index 216
0xcb0d40a729920 [Users\dogeCoin.\bitpay]\app\Code\data_0 216
0xcb0d40a729a60 [Users\dogeCoin.\bitpay]\app\Code\data_1 216
0xcb0d40a729dd6 [Users\dogeCoin.\bitpay]\app\Code\data_1 216
0xcb0d40a72a0f0 [Users\dogeCoin.\bitpay]\app\Code\data_0 216
0xcb0d40a72a286 [Users\dogeCoin.\bitpay]\app\Code\data_2 216
0xcb0d40a72a736 [Users\dogeCoin.\bitpay]\app\GPUCache\data_0 216
0xcb0d40a72b30 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\resources\app.asar 216
0xcb0d40a72c350 [Users\dogeCoin.\bitpay]\app\Code\data_3 216
0xcb0d40a72c670 [Users\dogeCoin.\bitpay]\app\Code\data_3 216
0xcb0d40a72dc50 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\resources\app.asar 216
0xcb0d40a73fd66 [Users\dogeCoin.\bitpay]\app\GPUCache\index 216
0xcb0d40a749e96 [Users\dogeCoin.\bitpay]\app\GPUCache\Index 216
0xcb0d40a7491986 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\vulkan-1.dll 216
0xcb0d40a729986 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\BitPay.exe 216
0xcb0d40a8a5d586 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\resources\app.asar 216
0xcb0d40a8a9986 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\vulkan-1.dll 216
0xcb0d40a8a9b0e0 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\d3dcompiler_47.dll 216
0xcb0d40a8a9e66 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\chrome_200_percent.pak 216
0xcb0d40a8a9f4e0 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\icudtl.dat 216
0xcb0d40a8a9f9786 [Program Files\Windows Apps\18C76590.\BitPayForWindows_12.8.6.0_x64_tq51jcq72nbzw\app\chrome_100_percent.pak 216
0xcb0d40a99d60 [Users\dogeCoin.\bitpay]\app\Local_Storage\[LevelDB]\000003.lbd 216
0xcb0d40a991220 [Users\dogeCoin.\bitpay]\app\Local_Storage\[LevelDB]\000004.lbd 216
0xcb0d40a9915b0 [Users\dogeCoin.\bitpay]\app\Local_Storage\[LevelDB]\000004.log 216
```

[그림 13] Live 상태에서 로그 파일 탐색

■ 프로세스가 Dead 상태인 Memory Dump

```
(volatility_environ) [root@heero:~/]# ./FC-2021-501s python vol.py -f bitcoin/image/btipay_enc_30sec.vmem windows.filescan | grep -i "btipay"
0xcb040a0926a0 [Program Files\WindowsApps\18C7659D.BtPayforWindows_12.8.6.0_x64_tq51jcq72mbzw\app\resources\app.asar 216
0xcb040a0b68d0 [Users\dogeCoin\btipay\app\Local Storage\leveldb\CURRENT 216
0xcb040a0c4840 [Users\dogeCoin\btipay\app\Local Storage\leveldb\MANIFEST-000001 216
0xcb040a073c70 [Users\dogeCoin\btipay\app\Cache\index 216
0xcb040a071730 [Users\dogeCoin\btipay\app\Cache\index 216
0xcb040a0715b6f0 [Users\dogeCoin\btipay\app\Cache\wasm\index 216
0xcb040a0729a6b0 [Users\dogeCoin\btipay\app\Cache\data_1 216
0xcb040a072a0f60 [Users\dogeCoin\btipay\app\Cache\data_0 216
0xcb040a072a28f0 [Users\dogeCoin\btipay\app\Cache\data_2 216
0xcb040a072a73f0 [Users\dogeCoin\btipay\app\GPUCache\data_0 216
0xcb040a072c35f0 [Users\dogeCoin\btipay\app\Cache\data_3 216
0xcb040a072980 [Program Files\WindowsApps\18C7659D.BtPayforWindows_12.8.6.0_x64_tq51jcq72mbzw\app\BitPay.exe 216
0xcb040a0aa9b89 [Program Files\WindowsApps\18C7659D.BtPayforWindows_12.8.6.0_x64_tq51jcq72mbzw\app\vulkan-1.dll 216
0xcb040a0aa9b90 [Program Files\WindowsApps\18C7659D.BtPayforWindows_12.8.6.0_x64_tq51jcq72mbzw\app\d3dcompiler_47.dll 216
0xcb040a0aae060 [Program Files\WindowsApps\18C7659D.BtPayforWindows_12.8.6.0_x64_tq51jcq72mbzw\app\chrome_200_percent.pak 216
0xcb040a0aa970a0 [Program Files\WindowsApps\18C7659D.BtPayforWindows_12.8.6.0_x64_tq51jcq72mbzw\app\chrome_100_percent.pak 216
0xrhN40a01420 [Users\dogeCoin\btipay\app\binaries\vn-0.3.0.hml 216
0xcb040a0a915b20 [Users\dogeCoin\btipay\app\Local Storage\leveldb\000004.log 216
0xcb040a0a97cc0 [Users\dogeCoin\btipay\app\GPUCache\data_1 216
0xcb040a0a987c0 [Program Files\WindowsApps\18C7659D.BtPayforWindows_12.8.6.0_x64_tq51jcq72mbzw\app\libEGL.dll 216
0xcb040a0b5b30 [Users\dogeCoin\btipay\app\Cache\wasm\index\dир\the-real-index 216
0xcb040a0bae970 [Program Files\WindowsApps\18C7659D.BtPayforWindows_12.8.6.0_x64_tq51jcq72mbzw\assets\Square44x44Logo.png 216
0xcb040a0d1310 [Users\dogeCoin\btipay\app\Cache\j\index\dир\the-real-index 216
0xcb040a0d92c0 [Users\dogeCoin\btipay\app\GPUCache\data_2 216
0xcb040a0ca5960 [ProgramData\Packages\18C7659D.BtPayforWindows_tq51jcq72mbw\S-1-5-21-3404489634-1916728933-3810222922-1001\SystemAppData\Helium\Cache\b45bc826a091778c.dat 216
0xcb040a0ca8970 [Users\dogeCoin\AppData\Local\Packages\18C7659D.BtPayforWindows_tq51jcq72mbw\SystemAppData\Helium\UserClasses.dat 216
0xcb040a0ac1940 [Users\dogeCoin\AppData\Local\Packages\18C7659D.BtPayforWindows_tq51jcq72mbw\SystemAppData\Helium\User.dat 216
0xcb040a0ca0dc0 [ProgramData\Packages\18C7659D.BtPayforWindows_tq51jcq72mbw\S-1-5-21-3404489634-1916728933-3810222922-1001\SystemAppData\Helium\Cache\b45bc826a091778c\CM15.dat 216
0xcb040a0f713b0 [Program Files\WindowsApps\18C7659D.BtPayforWindows_12.8.6.0_x64_tq51jcq72mbzw\app\xManifest.xml 216
0xcb040a0b672a10 [Users\dogeCoin\btipay\app\Cache\f_000002 216
0xcb040a0b673c90 [Users\dogeCoin\btipay\app\Preferences 216
```

[그림 14] Dead 상태에서 로그 파일 탐색

프로세스가 Dead 상태인 경우는 종료 직후 수집을 원칙으로 하였다. 그러나 프로세스 종료 이후 30초 이상 경과 시 아래와 같이 Mapping된 파일이 소멸하는 것을 확인할 수 있었다.

```
(volatility3_env) heero@ubuntu:~/DFC-2021-501$ python vol.py -f bitcoin/image/bitpay_enc_60sec.vmem windows.filescan | grep -i "bitpay"  
0xcb0409e318d0.0\Program Files\WindowsApps\18C7659D.BitPayforWindows_12.8.6.0_x64_tq51jcz7mbzw\AppxManifest.xml 216  
0xcb0409e318d0\ProgramData\Microsoft\Windows\AppRepository\Packages\18C7659D.BitPayforWindows_12.8.6.0_x64_tq51jcz7mbzw\S-1-5-21-340448  
9634-916728933-381022292-1001.pckdpe 216
```

[그림 15] Dead 30 초 이후의 덤프에서 로그 파일 소멸 확인

우선은 프로세스 종료 직후 덤프와 프로세스 실행 중 덤프 모두 Filescan 결과 00000n.log 파일이 Mapping 된 것을 확인하였으므로 해당 파일 덤프를 통해 분석을 진행하였다.

```
(base) sin90@odongbin-ui-MacBookPro vol3 % python vol.py -f bitpay_unencrypted.vmem windows.filescan | grep 000004.log
0xd906e1928150.0\Users\Dogecoin\AppData\Local\Microsoft\Edge\User Data\Default\Local Storage\leveldb\000004.log 216
(base) sin90@odongbin-ui-MacBookPro vol3 % python vol.py -o . -f bitpay_unencrypted.vmem windows.dumpfiles --virtaddr=0xd906e1928150
Volatility 3 Framework 1.1.1
Progress: 100.00          PDB scanning finished
Cache   FileObject      FileName        Result

(base) sin90@odongbin-ui-MacBookPro vol3 % python vol.py -f bitpay_enc_30sec.vmem windows.filescan | grep 000004.log
0xcb040aa915b0.0\Users\Dogecoin\.bitpay\app\Local Storage\leveldb\000004.log    216
0xcb040b4a9260 \Users\Dogecoin\AppData\Local\Microsoft\Edge\User Data\Default\Local Storage\leveldb\000004.log 216
(base) sin90@odongbin-ui-MacBookPro vol3 % python vol.py -o . -f bitpay_enc_30sec.vmem windows.dumpfiles --virtaddr=0xcb040aa915b0
Volatility 3 Framework 1.1.1
Progress: 100.00          PDB scanning finished
Cache   FileObject      FileName        Result

DataSectionObject 0xcb040aa915b0 000004.log      file.0xcb040aa915b0.0xcb040c0052f0.DataSectionObject.000004.log.dat
```

[그림 16] 종료 직후와 실행 덤프로부터 로그 파일 덤프

프로세스가 Dead된 즉시 덤프 한 Memory는 파일 추출에 성공하였고, 프로세스가 Live한 상태의 Memory는 파일 추출에 실패한 것을 확인할 수 있었다.

528:	39	37	37	36	37	2C	22	62	61	6C	61	6E	63	65	22	3A	97767,"balance":
544:	22	30	2E	30	30	20	42	54	43	22	7D	01	3A	5F	66	69	"0.00 BTC").:_fi
560:	6C	65	3A	2F	2F	00	01	6C	61	73	74	41	64	64	72	65	le://..lastAddre
576:	73	73	2D	64	39	35	39	37	64	32	61	2D	63	35	62	34	s-d9597d2a-c5b4
592:	2D	34	34	35	64	2D	61	33	34	32	2D	34	37	32	63	34	-445d-a342-472c4
608:	61	65	65	36	32	35	66	2B	01	62	63	31	71	30	67	79	aee625f+.bclq0gy
624:	71	61	71	72	65	67	68	34	79	34	6E	78	73	6C	61	6D	qaqregh4y4nxslam
640:	74	36	6A	32	71	79	33	78	6C	6A	33	30	39	76	70	77	t6j2qy3x1lj309vpw
656:	76	61	64	AA	5E	31	7D	94	00	01	34	00	00	00	00	00	valid^1}"..4.....
672:	00	00	02	00	00	00	01	0C	4D	45	54	41	3A	66	69	6CMETA:fil
688:	65	3A	2F	2F	0D	08	BE	99	AD	E9	A1	D9	CA	17	10	86	e://..%P-é;ÜÉ..t

[그림 17] 확보한 로그 파일 내부의 Wallet Address

해당 파일 내에는 JSON 구조가 발견되면 lastAddress 이후에 "bc1"으로 시작되는 주소를 확인할 수 있다.

(2) Process Dump

■ 프로세스가 Live 상태인 memory dump

[그림 18] Live Process 내부의 Wallet 데이터

Bitpay의 경우 2가지 JSON 형식을 메모리 덤프 내에서 발견 가능함을 확인하였다. Json의 구체적인 내용

은 아래와 같다. 정규표현식으로 검색하였을 때 이외에도 많은 지갑 주소 후보가 검색된 바 있으나 있으나 padding과 같은 데이터로 인해 주소의 끝을 특정하기 어렵고 Validation 하기 어려워 아래의 JSON 형식을 주소 추출 대상으로 하였다.

[표 11] 추출 JSON 데이터

```
{"version":"1.0.0","createdOn":1631900172,"address":"bc1q4q0g4rzqcvhtqdardt99ep2ehsfnjpfshевd8","walletId":"efee2bb3-3685-4801-a6d4-8af317631b27","isChange":false,"path":"m/0/2","publicKeys":["02592e8233072c157fadd82a04c42c41ce84236542f70c34a7492d561eff045789"],"coin":"btc","network":"livenet","type":"P2WPKH","beRegistered":null}
```

BWS Event: NewAddress:
{ "version": "1.0.0", "createdOn": 1631898918, "id": "016318989184120000", "type": "NewAddress", "data": { "address": "bc1qj0kwsktlapkgz9lp23wxyqd4asrql0v8a0n6k6" }, "walletId": "efee2bb3-3685-4801-a6d4-8af317631b27", "creatorId": "46ab0a1f6d2fa633677d6117b83c4570d557c74693b542ba6453d344aee71999", "isCreator": null } }

JSON 형식 내에는 address와 같은 단어가 JSON의 Key를 구성하고 있으며 해당 value를 파싱하여 address를 확정할 수 있었다.

■ 프로세스가 Dead 상태인 memory dump

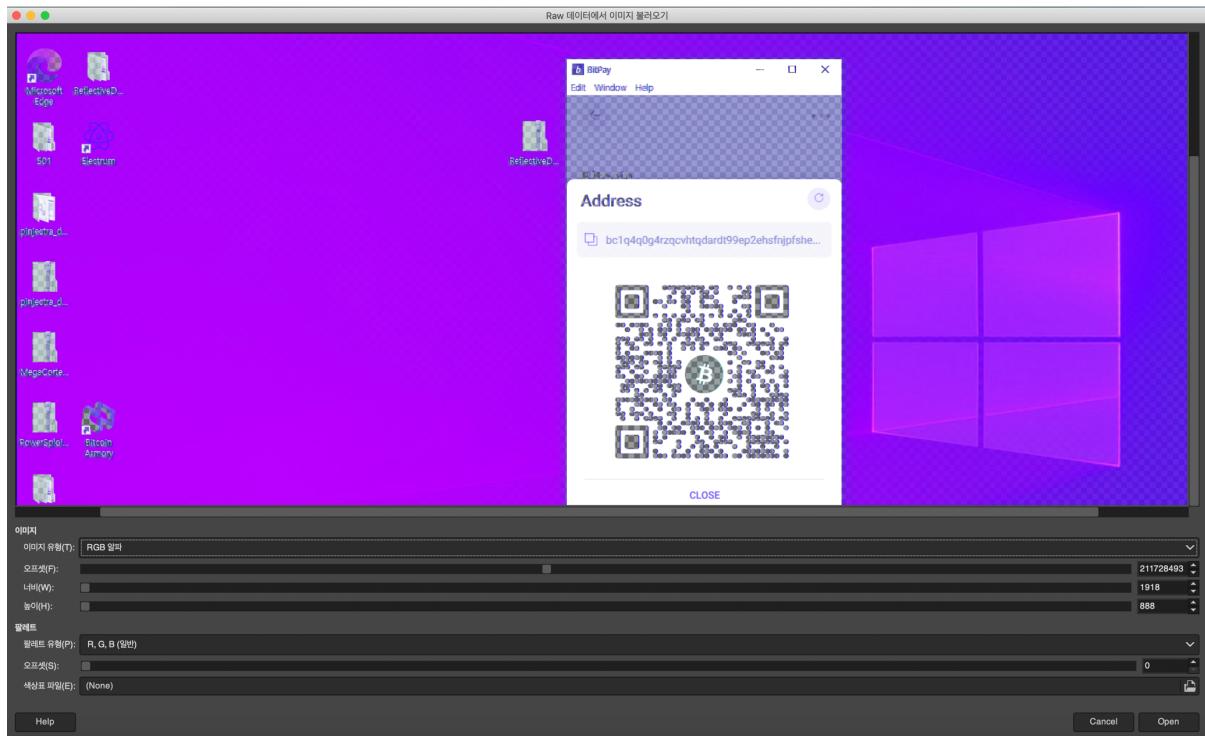
```
file://♦ 0x01balanceCache-15b4bad-5077-4d1b-bcc5-03c3b5017d14/<0x01>["updatedOn":1631897749,"balance":"0.00
DOGE"]<0x01>;file://♦ 0x01balanceCache-19abfe8b-793c-41c1-ab57-0a760260e81.<0x01>["updatedOn":1631897749,"balance":"0.00
LTC"]<0x01>;file://♦ 0x01balanceCache-d9597d2a-c5b4-445d-a342-472c4aee625f.<0x01>["updatedOn":1631897749,"balance":"0.00
BTC"]<0x01>;file://♦ 0x01balanceCache-d9597d2a-c5b4-445d-a342-472c4aee625f.<0x01>["updatedOn":1631897767,"balance":"0.00
<0x00>x8!jU<0x17><0x10>y<0x04><0x01>;file://♦ 0x01balanceCache-d9597d2a-c5b4-445d-a342-472c4aee625f.<0x01>["updatedOn":1631897767,"balance":"0.00
BTC"]<0x01>;file://♦ 0x01lastAddress-d9597d2a-c5b4-445d-a342-472c4aee625f+<0x01>b1qbygaapqmg4mxnslam6!2y023yl309pvpad=0<0x02>♦♦♦♦♦.0x01<0x0c>META:file://
<0x00>x6!jU<0x17><0x10>y<0x04><0x01>;file://♦ 0x01balanceCache-d9597d2a-c5b4-445d-a342-472c4aee625f+<0x01>["updatedOn":1631897848,"balance":"0.00
BTC"]<0x01>+<0x01><0x01>6♦♦♦♦♦.0x02<0x01>16♦♦♦♦♦.0x02<0x01>♦♦♦.0x01<0x0c>META:file://
<0x00>,0$0!jU<0x17><0x10>y<0x04><0x01>;file://♦ 0x01lastAddress-d9597d2a-c5b4-445d-a342-472c4aee625f[<0L><0x00><0x01><0x02>♦♦♦♦♦.0x01<0x0c>META:file://
<0x00>,0#n<0x17><0x10>y<0x04><0x01>;file://♦ 0x01lastAddress-d9597d2a-c5b4-445d-a342-472c4aee625f+<0x01>b1clw5ayg7utcksqv4d2q3h9jvq8jg3ngg5xm2H&0<0x13>d><0x01>:&♦♦♦♦♦.0x02<0x01><0x01><0x0c>META:file://
//:
<0x00>,0$00!jU<0x17><0x10>y<0x04>♦:file://♦ 0x01lastAddress-d9597d2a-c5b4-445d-a342-472c4aee625f@<0x00>:r<0x01><0x02>♦♦♦♦♦.0x04>♦♦♦♦♦.0x01<0x0c>META:file://
<0x00>,0#n!jU<0x17><0x10>y<0x04><0x01>;file://♦ 0x01balanceCache-15b4bad-5077-4d1b-bcc5-03c3b5017d14/<0x01>["updatedOn":1632295755,"balance":"0.00
DOGE"]<0x01>;file://♦ 0x01balanceCache-19abfe8b-793c-41c1-ab57-0a760260e81.<0x01>["updatedOn":1632295755,"balance":"0.00
LTC"]<0x01>;file://♦ 0x01balanceCache-d9597d2a-c5b4-445d-a342-472c4aee625f.<0x01>["updatedOn":1632295755,"balance":"0.00
BTC"]<0x01>;file://♦ 0x01balanceCache-d9597d2a-c5b4-445d-a342-472c4aee625f.<0x01>["updatedOn":1632295763,"balance":"0.00
BTC"]<0x01>]
```

[그림 19] Dead 이후의 Memory Dump에서 확인된 거래 관련 데이터

거래와 관련된 데이터를 일부 발견할 수 있었으나 핵심이 되는 지갑 주소와 관련된 데이터는 획득이 불가하였다.

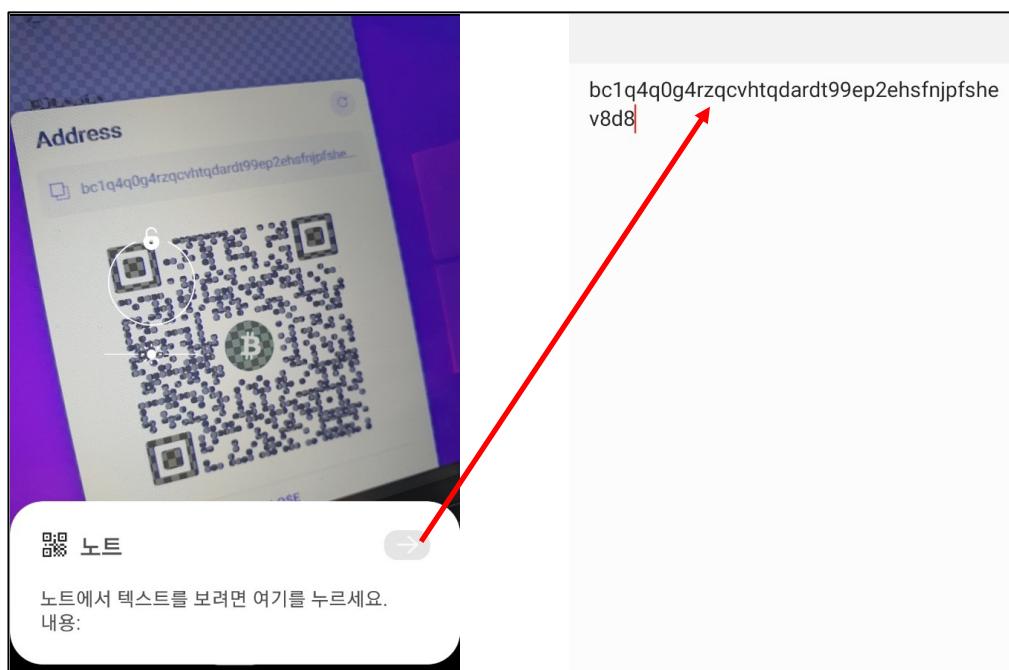
(3) Graphic Analysis

항목 2를 통해 획득한 Process Memory dump를 GIMP를 통해 open 하였다. Vol3 환경에서 Process memory dump를 얻는 방법은 live process의 메모리 덤프 밖에 없으므로 live process에서만 진행하였다. 실험에서 사용한 해상도인 1918*888과 이미지 유형 RGB alpha를 세팅해주고, 오프셋을 조절하여 적절한 이미지가 나올 때 까지 조정하였다. 그 결과 아래와 같은 이미지를 얻을 수 있었다.



[그림 20] Memory Dump로부터 복구한 Graphic File

Bitpay의 경우 지갑 주소가 위의 사진과 같이 일부 잘려서 보일 수 있으나, 아래의 QR 코드가 완전 하기 때문에 해당 QR Code를 해석하면 지갑 주소를 복구할 수 있다.



[그림 21] Slice 된 Address를 대체하여 QR로 복구한 화면

■ Electrum

[표 12] Electrum 비교 정보

프로그램	Electrum
종료 덤프	<ul style="list-style-type: none">■ 주소 스트링 Search 성공■ 지갑 파일 덤프 불가
실행 덤프	<ul style="list-style-type: none">■ 주소 스트링 Search 성공■ 지갑 파일 덤프 불가■ Gimp 가능
비고	<ul style="list-style-type: none">■ JSON 정보 발견■ 주소 전후로 규칙적인 패턴 발견

(1) Filescan & dump

```
(volatility3_env) heero@ubuntu:~/DFC-2021-501$ python vol.py -f bitcoin/image/electrum_enc_modified.vmem windows.filescan | grep -i "wallet"
(volatility3_env) heero@ubuntu:~/DFC-2021-501$ python vol.py -f bitcoin/image/electrum_enc_modified_30sec.vmem windows.filescan | grep -i "wallet"
```

[그림 22] Live 및 Dead 덤프에 대한 Wallet 파일 탐색

해당 wallet 어플리케이션의 경우 프로세스가 live 상태인 memory dump 프로세스가 dead 상태인 memory dump 모두에서 지갑 관련된 파일이 발견되지 않았다.

(2) Process dump

■ 프로세스가 Live 상태인 memory dump

669323696:	20	20	20	20	20	20	20	20	20	20	20	22	03	01	22	3A	20	1d
669323712:	22	32	36	32	31	36	36	34	37	37	63	22	2C	0A	20	20	"262166477c",..	
669323728:	20	20	20	20	20	20	20	20	20	20	22	6D	65	73	73	61	"messag	
669323744:	67	65	22	3A	20	22	22	2C	0A	20	20	20	20	20	20	20	ge": "",..	
669323760:	20	20	20	20	20	22	6F	75	74	70	75	74	73	22	3A	20	"outputs":	
669323776:	5B	0A	20	20	20	20	20	20	20	20	20	20	20	20	20	20	[.	
669323792:	20	20	5B	0A	20	20	20	20	20	20	20	20	20	20	20	20	[.	
669323808:	20	20	20	20	20	20	20	20	30	2C	0A	20	20	20	20	20	0,..	
669323824:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	22	"	
669323840:	62	63	31	71	6A	34	35	39	71	66	37	36	68	79	67	37	bclqj459qf76hyg7	
669323856:	38	7A	34	76	36	37	37	6E	61	38	30	70	36	35	76	73	8z4v877na80p65vs	
669323872:	71	6A	66	65	72	35	79	6C	77	6C	22	2C	0A	20	20	20	qjfer5ylwl",..	
669323888:	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20		
669323904:	20	30	0A	20	20	20	20	20	20	20	20	20	20	20	20	20	0.	
669323920:	20	20	20	5D	0A	20	20	20	20	20	20	20	20	20	20	20].	
669323936:	20	5D	2C	0A	20	20	20	20	20	20	20	20	20	20	20	20],..	
669323952:	22	72	65	71	75	65	73	74	6F	72	22	3A	20	6E	75	6C	"requestor": nul	
669323968:	6C	2C	0A	20	20	20	20	20	20	20	20	20	20	20	22	1,..		
669323984:	74	69	6D	65	22	3A	20	31	36	33	31	38	31	30	31	34	time": 163181014	

[그림 23] Live 상태에서의 Wallet Address

■ 프로세스가 Dead 상태인 memory dump

[그림 24] Dead 상태에서의 Wallet Address

dump 내에서는 process의 live 여부와 관계없이 json의 형태가 존재하며 주소 값 output이라는 key의 구조체 안에 지갑의 주소 데이터가 저장되고 있는 것을 확인하였다. 관련 내용은 아래와 같다.

[표 13] Electrum의 주소 데이터 구조체

```
"payment_requests": {
    "bc1q57xmkgjjps6scf3a2u5xaahu7f77gxc829u45t": {
        "amount_sat": 0,
        "bip70": null,
        "exp": 86400,
        "height": 700821,
        "id": "1f5dc5ca18",
        "message": "",
        "outputs": [
            [
                0,
                "bc1q57xmkgjjps6scf3a2u5xaahu7f77gxc829u45t",
                0
            ]
        ],
        "requestor": null,
        "time": 1631810147,
        "type": 0
    },
    "bc1q85fqtwmhzaep6hs0ne6c458rgjnfhvzaqw7zch": {
        "amount_sat": 0,
        "bip70": null,
        "exp": 604800,
        "height": 701539,
        "id": "02802a3667",
        "message": "",
        "outputs": [
            [
                0,
                "bc1q85fqtwmhzaep6hs0ne6c458rgjnfhvzaqw7zch",
                0
            ]
        ]
    }
}
```

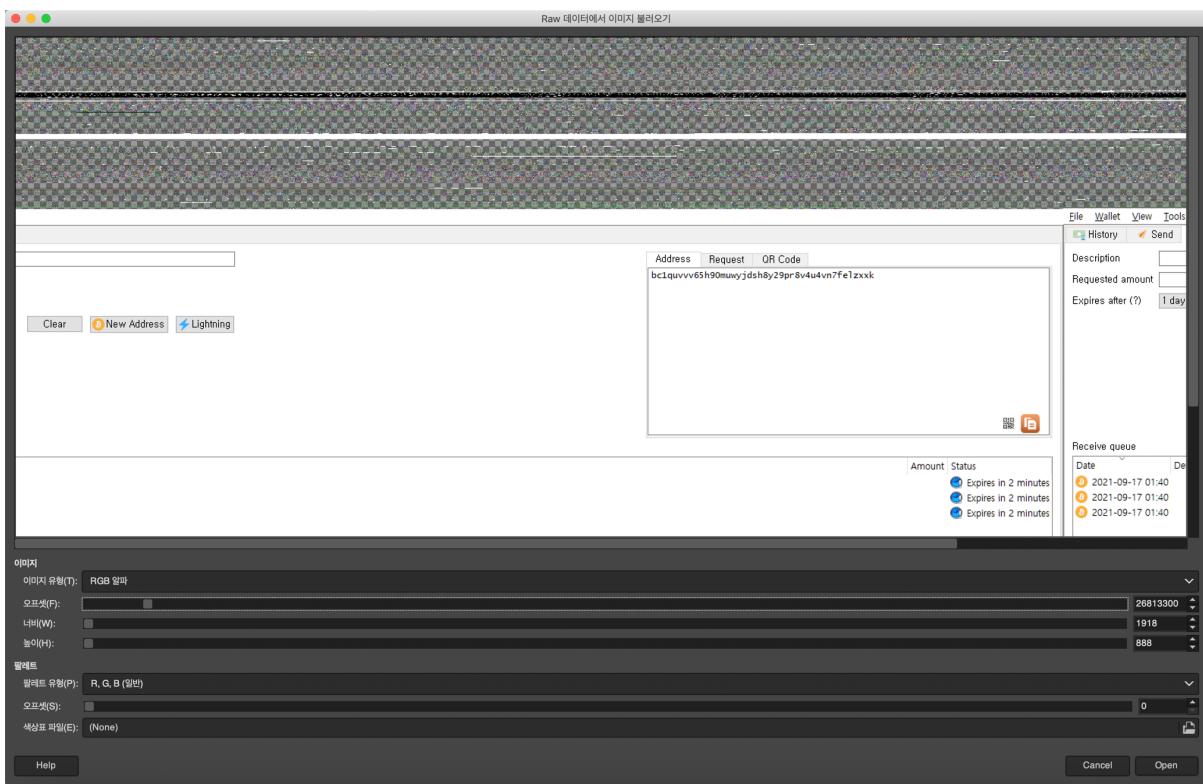
```

    0
]
],
“requestor”: null,
“time”: 1632214148,
“type”: 0
}

```

(3) Graphic Analysis

항목 2를 통해 획득한 Process Memory dump를 GIMP를 통해 open 하였다. Vol3 환경에서 Process memory dump를 얻는 방법은 live process의 메모리 덤프 밖에 없으므로 live process에서만 진행하였다. 실험에서 사용한 해상도인 1918*888과 이미지 유형 RGB alpha를 세팅해주고, 오프셋을 조절하여 적절한 이미지가 나올때 까지 조정하였다. 그 결과 아래와 같은 이미지를 얻을 수 있었다.



[그림 25] Memory Dump로부터 복구한 Graphic File

■ Armory

[표 14] Armory 비교 정보

프로그램	Armory
종료 덤프	<ul style="list-style-type: none"> 주소 스트링 Search 실패 지갑 파일 Filescan 결과 흔적 없음 및 dump 도 불가
실행 덤프	<ul style="list-style-type: none"> 주소 스트링 Search 성공 지갑 파일 Filescan 결과 흔적 없음 및 dump 도 불가 Gimp 가능
비교	<ul style="list-style-type: none"> 주소 값 전후로 규칙적 패턴 발견

(1) Filescan & dump

■ 프로세스가 Live 상태인 memory dump

```
(volatility3_env) heero@ubuntu:~/DFC-2021-501$ python vol.py -f bitcoin/image/armory_address_added.vmem filescan | grep -i "wallet"
(volatility3_env) heero@ubuntu:~/DFC-2021-501$ nished
(volatility3_env) heero@ubuntu:~/DFC-2021-501$
```

[그림 26] Live 상태에서 Wallet 파일 탐색

■ 프로세스가 Dead 상태인 memory dump

```
(volatility3_env) heero@ubuntu:~/DFC-2021-501$ python vol.py -f bitcoin/image/armory_address_added_30sec.vmem filescan | grep -i "wallet"
0xd0a03445da0.0\Users\Dogecoin\AppData\Roaming\Armory\armory_2g92h84g7_wallet.lmdb      216
(volatility3_env) heero@ubuntu:~/DFC-2021-501$
```

[그림 27] Dead 상태에서 Wallet 파일 탐색

프로세스가 live한 상태에서는 memory에 mapping된 지갑 관련 파일들이 존재하지 않았으나, 프로세스가 dead된 상태에서는 wallet 파일이 발견되었다. 그러나 해당 파일만으로 지갑 주소 값을 단순 String만 확인하기에는 역부족이며, 추후 역공학과 같은 방법으로 Armory wallet service에 대해 추가적인 연구가 필요해 보인다.

00006E10	22 31 03 22 30 6E 6E 77 19 60 2A 00 13 22 33 21
00006F00	9C 2F DB 35 9D 1E 2C C8 C7 26 E0 72 28 21 75 CD æ/ð5..,Èç&är(!uf
00006F10	AC CE 5F 00 68 00 00 00 00 05 00 AA 40 00 00 ~í..h.....*ø..
00006F20	00 67 01 42 80 04 D1 90 14 78 10 F3 8A 58 5D D0 .g.BE.Ñ..x.öŠXjÐ
00006F30	7D 21 3D AF 66 74 65 DE 99 86 0D 9C 5F F6 AB 71 !)-ft...Pmt.æ_ø<q
00006F40	C6 2C 80 D3 58 75 06 E3 18 8F F7 02 D6 F6 18 3C È,éÓKu.ä..-,Öö.<
00006F50	D7 FA 00 E9 3E C0 88 7D CD BF FE F5 D0 6E 28 F0 xú.é~À^)íçþðñ(ð
00006F60	2F 5F D6 BB 09 6B 22 81 03 D1 90 14 78 10 F3 8A / _Öw.K"..Ñ..x.öŠ
00006F70	58 5D D0 7D 21 3D AF 66 00 00 30 00 00 02 00 X}þ)!-f..0.....
00006F80	09 00 32 6A 4E 6D 4E 43 56 4B 75 00 00 00 00 ..2jNmNCVKu.....
00006F90	00 02 00 01 00 00 00 00 00 00 04 00 00 00
00006FA0	00 00 00 00 00 00 00 00 00 00 00 6B 00 00 00
00006FB0	00 00 00 0A 00 00 00 00 00 00 00 0A 00 00 00
00006FC0	00 00 04 00 A1 00 00 00 09 32 6A 4E 6D 4E 43 56i....2jNmNCV
00006FD0	4B 75 05 00 00 00 00 00 0A 00 B0 32 6A 4E 6D 4E Ku....."2jNmN
00006FF0	43 56 4B 75 04 00 00 00 00 0A 00 00 00 00 00 00 CVRü.....
00006FF0	04 00 A0 00 00 00 09 32 67 39 32 68 38 34 67 372g92h84g7
00007000	07 00 00 00 00 00 00 00 00 02 00 14 00 7A 0Fz.

[그림 28] 덤프한 Wallet 파일의 내부 데이터

(2) Process dump

■ 프로세스가 Live 상태인 memory dump

63	68	00	00	00	00	00	BC	9C	36	04	00	81	00	8E	ch.....¾œ6....ž
33	43	6E	56	73	78	47	32	32	44	39	6F	50	69	6A	59
67	5A	48	53	77	75	44	5A	43	4C	57	37	74	4A	33	65
57	6B	00	00	00	00	00	B3	9C	35	04	00	82	00	90	Wk.....¾œ5...,..
63	35	63	34	34	32	62	66	35	65	30	34	31	31	32	32

[그림 29] Live 프로세스 상의 Armory 주소 저장

Memory dump 내에서는 다음과 같은 패턴이 발견되었다. 우선 64바이트의 구조체를 사용하며 구조체는 크게 fore_bytes, 주소, back_bytes 이렇게 3부분으로 구분할 수 있다. 그 중 fore_bytes와 back_bytes는 아래의 조건을 만족하도록 되어있는 것을 추가로 확인하였다. 정규표현식으로 확인한 지갑 주소를 기준으로 아래의 조건을 만족시킨다면 armory의 사용 흔적이라고 추측 가능하다.

[표 15] Live 상태에서 Armory의 주소 구조체

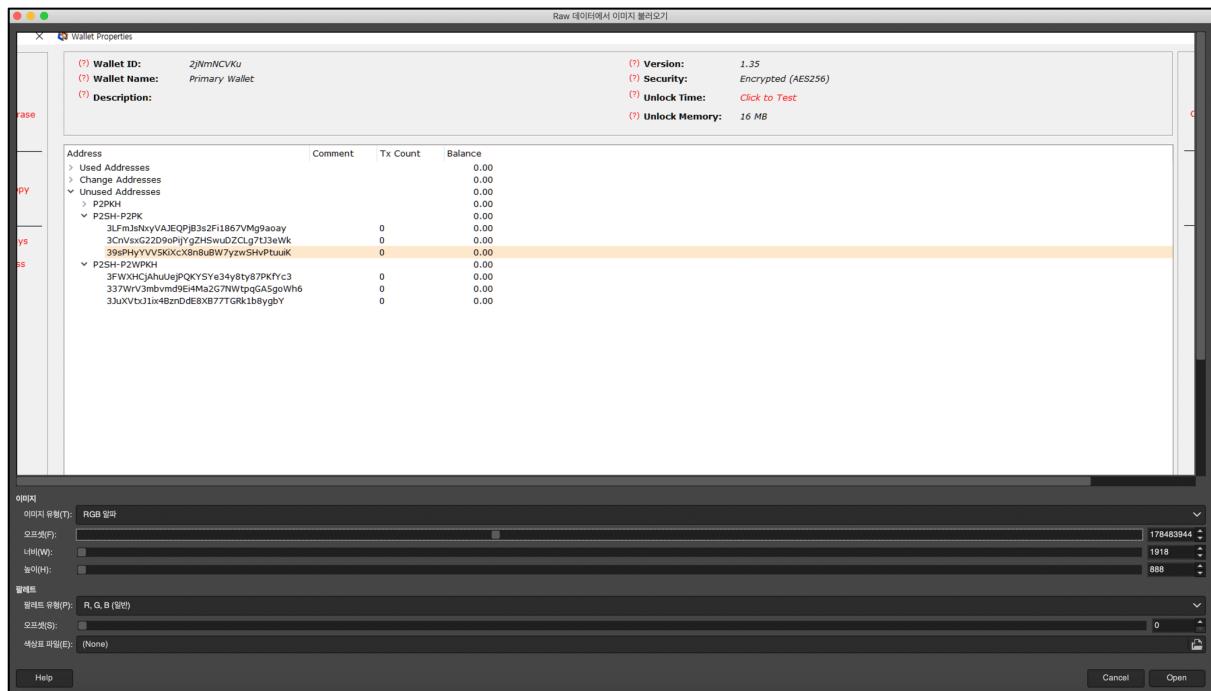
```
condition1 = fore_bytes[-7] == back_bytes[-7]
condition2 = fore_bytes[-6] != back_bytes[-6]
condition3 = fore_bytes[-5] == back_bytes[-5]
condition4 = fore_bytes[-4] == back_bytes[-4] == 0
condition5 = fore_bytes[-3] + 1 == back_bytes[-3]
```

■ 프로세스가 Dead 상태인 memory dump

정규 표현식 검색 결과 dead 상태의 process의 memory dump에서는 지갑 주소 matching이 되지 않았다.

(3) Graphic Analysis

항목 2를 통해 획득한 Process Memory dump를 GIMP를 통해 open 하였다. Vol3 환경에서 Process memory dump를 얻는 방법은 live process의 메모리 덤프 밖에 없으므로 live process에서만 진행하였다. 실험에서 사용한 해상도인 1918*888과 이미지 유형 RGB alpha를 세팅해주고, 오프셋을 조절하여 적절한 이미지가 나올때 까지 조정하였다. 그 결과 아래와 같은 이미지를 얻을 수 있었다.



[그림 30] Memory Dump로부터 복원한 Graphic File

■ Bither

[표 16] Bither 비교 정보

프로그램	Bither
종료 덤프	<ul style="list-style-type: none">■ 주소 스트링 서치 성공■ 지갑 파일 Filescan 결과 흔적 존재하지 않음
실행 덤프	<ul style="list-style-type: none">■ 주소 스트링 서치 성공■ 지갑 파일 Filescan 결과 흔적 존재하지 않음
비교	<ul style="list-style-type: none">■ 주소값 전후로 규칙적 패턴 발견

(1) Filescan & dump

■ 프로세스가 live 상태인 memory dump

```
(base) sin90@odongbin-ui-MacBookPro vol3 % python vol.py -f bither_on.vmem windows.filescan | grep bither
0x7cc1aa0 100.0\Users\IEUser\AppData\Roaming\Bither\bither.db-journal 216
0x7cfeaf20 \Users\IEUser\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\2V5LZE9A\logo_bither[1].jpg 216
0xd12af20 \Users\IEUser\AppData\Roaming\Bither\bither.properties 216
0xd2c8e20 \Users\IEUser\AppData\Roaming\Bither\log\bither.log 216
0xd46f610 \Users\IEUser\AppData\Roaming\JWrapper-Bither\JWrapper-Bither-00075020282-complete\bither-desktop-1.4.8.jar 216
0xd59a4d0 \Users\IEUser\AppData\Roaming\Bither\bither.db 216
```

[그림 31] Live 상태에서 Wallet 파일 탐색

■ 프로세스가 dead 상태인 memory dump

```
(base) sin90@odongbin-ui-MacBookPro vol3 % python vol.py -f bither_off.vmem windows.filescan | grep bither
0x7cfeaf20 100.0\Users\IEUser\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\2V5LZE9A\logo_bither[1].jpg 216
0xd12af20 \Users\IEUser\AppData\Roaming\Bither\bither.properties 216
0xd2c8e20 \Users\IEUser\AppData\Roaming\Bither\log\bither.log 216
0xd46f610 \Users\IEUser\AppData\Roaming\JWrapper-Bither\JWrapper-Bither-00075020282-complete\bither-desktop-1.4.8.jar 216
0xd59a4d0 \Users\IEUser\AppData\Roaming\Bither\bither.db 216
```

[그림 32] Dead 상태에서 Wallet 파일 탐색

그러나 Bither의 경우 address.db로 지갑주소를 관리하는데, live 상태와 dead상태 모두 해당 파일이 발견되지 않았다.

(2) Process dump

■ 프로세스가 Live 상태인 memory dump

2가지 종류의 Pattern 이 발견되었다. 첫째는 지갑 주소가 나열된 형식이다. 아래와 같이 구분자로 2X 03 51 01 이라는 Hex 값을 구분자로 사용할 수 있다.

[그림 33] Live 상태에서 Wallet의 저작 패턴 (1)

그러나 이 패턴의 경우 실제 지갑의 주소도 가지고 있지만 지갑 주소와 관련 없는 정규 표현식 match 결과도 포함하여 우리가 원하는 지갑 주소만 Parsing 하기 힘들다는 단점이 있다.

이 경우에는 단순히 지갑 주소의 나열이 아니라 지갑 주소 이후에 추가적인 데이터가 포함되고, 이후 다시 구분자와 지갑 주소와 데이터가 나열된 형식을 나타낸다

[구분자][지갑주소, 34바이트] / [96바이트 0-9A-F] / [32바이트 0-9A-F] 데이터로 구성되어 있다. Bither의 경우 지갑 주소가 34바이트로 고정되는 것을 확인하여 패턴을 구성하였다.

65	09	08	09	05	31	41	66	48	34	55	78	7A	4A	4A	56	e....1AfH4UxzJJV
53	4D	65	76	32	5A	66	57	4C	70	52	75	78	4C	47	59	SMev2ZfWLpRuxLGY
7A	63	72	32	42	65	4C	30	30	30	30	33	38	33	30	35	zcr2BeL000038305
37	36	43	46	36	39	38	31	33	36	32	35	33	31	31	42	76CF69813625311B
39	36	31	30	35	37	36	33	32	41	42	33	31	43	41	41	961057632AB31CAA
32	34	32	39	35	39	38	32	37	34	31	30	41	39	36	36	242959827410A966
34	43	42	45	31	30	39	31	42	38	31	37	44	37	34	44	4CBE1091B817D74D
38	37	42	46	39	44	39	30	36	33	39	36	41	37	30	42	87BF9D906396A70B
43	42	35	37	30	38	41	2F	37	31	45	38	32	35	38	42	CB5708A/71E8258B
36	36	35	31	39	33	38	30	35	33	44	43	46	44	33	31	6651938053DCFD31
36	39	44	44	34	32	34	46	2F	38	36	46	41	38	35	37	69DD424F/86FA857
36	46	38	33	42	36	44	42	44	69	53	31	52	31	66	74	6F83B6DBDiS1R1ft
41	4C	79	79	75	41	57	72	78	4E	72	4C	4A	6F	75	66	ALyyuAWrxNrLJouf
73	4E	48	46	76	44	42	78	42	53	36	37	5A	36	44	51	sNHFvDbxBs6726DQ
66	34	58	76	6F	01	7C	0C	88	72	21	81	6F	02	09	51	f4Xvo. .^r!.o..Q
82	31	65	09	08	09	05	31	43	48	58	32	41	6E	33	48	,le....1CHX2An3H
4A	41	5A	6A	64	4B	79	52	34	58	56	51	69	68	67	48	JAZjdKyR4XVQihgH

[그림 34] Live 상태에서 Wallet의 저장 패턴 (2)

추가적으로 Bither의 경우 HD account 라고 하는 1개만 생성할 수 있는 지갑도 존재하는데 해당 지갑 주소의 경우 아래와 같은 패턴을 보인다

784212704:	43	37	43	30	43	43	31	4C	52	6D	48	72	33	46	5A	47		C7C0CC11LRmHr3FZG
784212720:	41	78	73	33	6F	55	43	58	38	79	62	69	38	75	54	48	Axs3oUCX8ybi8uTH	
784212736:	71	69	71	62	72	4D	77	62	48	57	36	59	43	34	47	66	qiqbrMwbHW6YC4Gf	
784212752:	70	33	36	54	66	68	44	69	61	38	36	57	65	53	75	38	p36TfhDia96WeSu8	
784212768:	4D	6D	69	52	38	76	73	39	6F	7A	62	59	4B	69	35	6A	MmiR8vs9ozbYKi5j	
784212784:	66	6E	65	54	78	44	75	71	47	4D	5A	64	67	73	72	64	fneTxDuqGMZdgsrd	
784212800:	47	46	48	65	52	41	41	69	68	43	57	58	45	56	43	71	GFHeRAAiCWXEVcQ	
784212816:	69	33	70	51	79	47	74	55	5A	67	6D	6D	56	31	76	58	i3pQyGtUZgmmVlvX	
784212832:	4C	6D	32	51	43	6D	63	37	36	34	46	64	54	63	32	31	Lm2QCmc764FdTc21	

[그림 35] Bither 의 특이 Wallet 에 대한 Pattern

[0-9A-F 데이터] +지갑 주소 + “HW” 문자열을 통해 원하는 데이터를 parsing 할 수 있다.

■ 프로세스가 Dead 상태인 memory dump

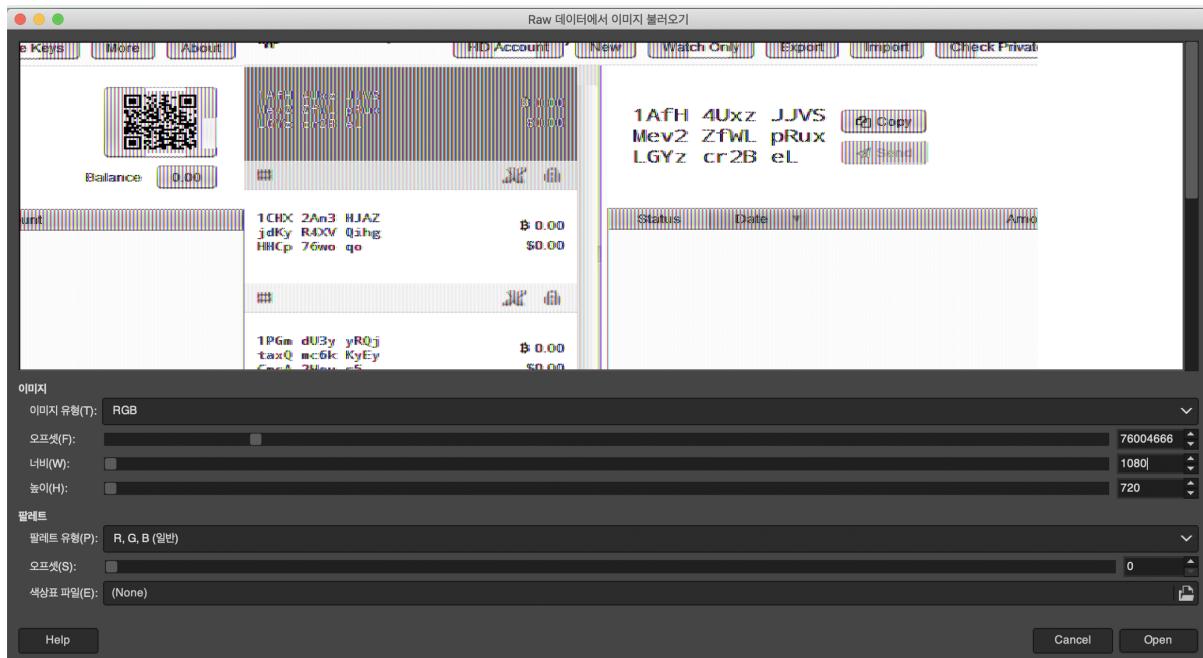
878324592:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
878324608:	00	00	00	00	00	00	00	00	00	00	00	00	26	03	51	01& Q.
878324624:	31	41	66	48	34	55	78	7A	4A	4A	56	53	4D	65	76	32	1AfH4UxzJJVSMev2
878324640:	5A	66	57	4C	70	52	75	78	4C	47	59	7A	63	72	32	42	ZfWLpRuxLGYzcr2B
878324656:	65	4C	03	26	03	51	01	31	43	48	58	32	41	6E	33	48	eL.&.Q.1CHX2An3H
878324672:	4A	41	5A	6A	64	4B	79	52	34	58	56	51	69	68	67	48	JAZjdKyR4XVQihgH
878324688:	48	43	70	37	36	77	6F	71	6F	02	25	03	51	09	31	50	HCp76woqo.%Q.1P
878324704:	47	6D	64	55	33	79	79	52	51	6A	74	61	78	51	6D	63	GmdU3yyRQjtaxQmc
878324720:	36	6B	4B	79	45	79	43	6D	63	41	32	48	6F	75	73	35	6kKyEyCmcA2Hous5

[그림 36] Dead 상태에서 Wallet 의 저장 패턴

앞선 Live 상태에서 확인한 첫번째 패턴이 Dead 상태의 memory dump에서 확인되었다. 마찬가지로 실제 사용 지갑 주소 구분이 어려워 사용할 수 없다는 한계가 존재한다.

(3) Graphic Analysis

항목 2를 통해 획득한 Process Memory dump를 GIMP를 통해 open 하였다. Vol3 환경에서 Process memory dump를 얻는 방법은 live process의 메모리 덤프 밖에 없으므로 live process에서만 진행하였다. 실험에서 사용한 해상도인 1080*720과 이미지 유형 RGB alpha를 세팅해주고, 오프셋을 조절하여 적절한 이미지가 나올 때 까지 조정하였다. 그 결과 아래와 같은 이미지를 얻을 수 있었다.

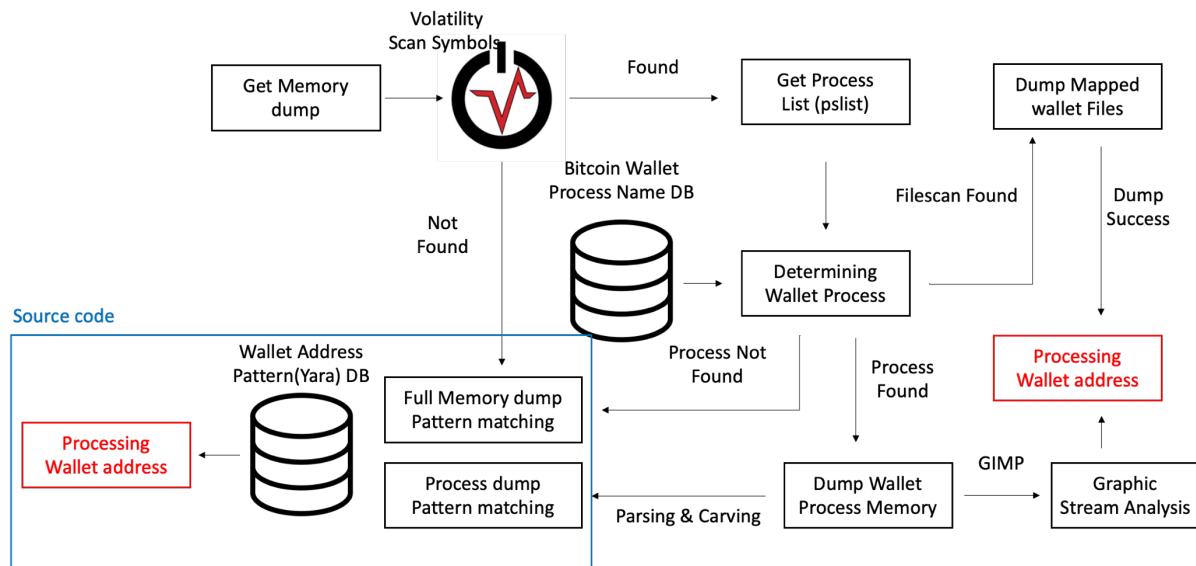


[그림 37] Memory Dump로부터 복원한 Graphic File

화면에 존재하는 주소 값에 대해서는 확실히 확인할 수 있었지만 모든 주소 값을 확인할 수는 없었다.

■ Wrap up

위의 특징들을 기반으로 하여, 특히 메모리 덤프 내에서 사용할 수 있는 각 지갑 어플리케이션 별 Yara Rule과 Source code를 제안하였다. 뿐만 아니라 위의 특징들을 종합하여 기 제안한 방법론에서 보다 체계적인 분석 방법론을 제시하였다.



[그림 38] 제안한 방법론의 전체 구성도 및 소스코드 제공 범위

1) Bitcoin Core

Bitcoin Core는 이전의 실험 및 분석 결과에 의하면 특정 구조체에 기반하여 (NonBech, Bech에 따라 달라지는 플래그 값, 이후 구성되는 문자열) 주소 값이 적재되는 것을 확인할 수 있었다. 구조체 형태에 기반한 Yara Rule을 작성하여 탐지하되, 메모리 페이징 등의 이유로 구조체가 온전하지 않은 형태는 정규표현식을 통해 탐지하도록 하였다. 과정은 아래와 같다.

- (1) Bench/NonBench 값을 Header로 삼고, 플래그 값 이후로 “label” 데이터가 존재하는 구조체의 형태를 가지는 Yara Rule을 정의한다.
- (2) 해당 Yara Rule을 컴파일 한 뒤, 패턴 매칭을 수행한다.
- (3) 구조체가 일부 변경되었을 경우를 고려하여 정의한 구조체의 Minimum한 정규식을 정의해 탐지한다.
- (4) 정규식과 Yara Rule에 의해 매칭된 데이터는 구조체가 적용된 데이터로 실제 주소 데이터만 획득하기 위해서는 이전에 정의한 Header/Tails 데이터 사이즈만큼 Slice를 수행한다.
- (5) 중복 탐지되는 주소는 병합 수행

[표 17] 메모리 상의 Bitcoin Core Wallet Address YARA Rule

```
rule BitcoinCoreRule {
    strings:
        $struct_1 = { 22 [34] 14 6C 61 62 65 6C }
        $struct_2 = { 2A [34] 14 6C 61 62 65 6C }
        $struct_3 = { 22 [42] 0C 6C 61 62 65 6C }
        $struct_4 = { 2A [42] 0C 6C 61 62 65 6C }
        $struct_5 = { 64 65 73 74 64 61 74 61 2A [42] 03 72 72 }
    condition:
        any of them
}
```

2) Bitpay

Bitpay는 ":{"address":"bc1qj0kwsktlapkgz9lp23wxyqd4asrql0v8a0n6k6"}," 형태의 JSON 데이터가 발견되거나 lastAddress라는 문자열 이후 수십 바이트 이내에 주소 값이 발견된다는 점에 착안하여 YARA Rule 및 패이썬을 이용하여 주소 값을 찾아낸다. 과정은 아래와 같다.

- (1) 불임(table 1)과 같은 문자열 offset을 yara로 탐색
- (2) 탐색된 offset부터 256byte를 읽어들여 임시 데이터 파일로 저장함
- (3) 저장된 임시파일에 strings 실행, 문자열만 추출
- (4) 추출된 문자열에 대하여 비트코인 주소 정규표현식 "bc1[a-kl-zA-HJ-NP-Z0-9]{25,39}" 검색
 - A. bitpay에서 생성되는 주소값은 bech32형식으로, bc1으로 시작함,
 - B. 아래의 yara에도 이러한 규칙 적용함
- (5) 매칭된 문자열을 주소로 판정

[표 18] 메모리 상의 Bitpay Wallet Address YARA Rule

```
rule bitpay
{
    strings:
        $address_json = /address": "bc1[a-kl-zA-HJ-NP-Z0-9]{25,39}\\"/>
        $last_address = "lastAddress"
    condition:
        any of them
}
```

3) Electrum

사용하거나 사용중인 주소 값에 대하여 Json 데이터가 발견되며, "outputs":[~~"주소값"~~] 의 패턴이 발견된다는 점에 착안하여 다음과 같이 주소 값을 탐지한다.

- (1) 불임(table2)의 yara rule로 “outputs”: 문자열 탐색
- (2) 탐색된 문자열의 offset으로부터 256 byte를 읽어들여 임시파일로 저장
- (3) 저장된 임시파일에 대하여 strings 실행, 문자열 추출
- (4) 추출된 문자열에 대하여 비트코인 주소 정규표현식 “[13]bc1[a-km-zA-HJ-NP-ZO-9]{25,39}” 검색
- (5) 매칭된 문자열을 주소로 판정

[표 19] 메모리 상의 Electrum Wallet Address YARA Rule

```
rule electrum
{
    strings:
        $address_json = "\"outputs\":"
    condition:
        all of them
}
```

4) Armory

Armory는 상술한 Bitpay, Electrum 등과 다르게 주소 값을 확실하게 식별할 수 있는 식별자가 발견되지 않는다. 따라서 정규표현식을 기반으로 주소 값을 탐색하되, 오탐을 줄이기 위하여 필터링 과정이 필요하다. 과정은 다음과 같다.

- (1) 덤프파일에 대하여 strings 실행, 문자열 추출
- (2) 추출된 문자열에 대하여 비트코인 주소 정규표현식 탐색 - 오탐 다수 발생
- (3) 정렬 및 단일화(sort, uniq 실행)
- (4) 단일화된 문자열을 모두 yara string으로 작성, condition은 any of them으로 작성(그림1 참고)
- (5) 메모리 덤프 파일에 작성한 yara rule 실행, 각 문자열 offset 탐색
- (6) 탐지된 offset에 대하여 조건 비교
 - A. 탐지된 offset - 16 offset부터 64바이트의 구조체에 대하여 조건 비교
 - B. 구조체는 크게 fore_bytes 구조체[0:16], 주소값 구조체[16:??], back_bytes 구조체 [?:64] 이렇게 3부분으로 구분
 - C. fore_bytes와 back_bytes는 아래의 조건을 만족
 - i. condition1 = fore_bytes[-7] == back_bytes[-7]
 - ii. condition2 = fore_bytes[-6] != back_bytes[-6]

- iii. condition3 = fore_bytes[-5] == back_bytes[-5]
- iv. condition4 = fore_bytes[-4] == back_bytes[-4] == 0

(7) 조건 일치하면 해당 문자열은 주소로 판정

[표 20] 메모리 상의 Armory Wallet Address YARA Rule

```
rule armory
{
    strings:
        $s1="30D11DAA3F01F5AF9F28573A4521B927"
        $s2="1e29ceb5901ccb39c465c5d8df6e7fa7"
        $s3="1006998991164425862847190493"
        $s4="11c06bb1b62479ca2248df0abcd05"
    (중략)
```

5) Bither

Bither에서 수집해야하는 주소는 2 가지로 위에서 탐색한 주소 특징을 활용하여 Python coding 을 진행하였다. 우선 정규표현식을 이용해 [구분자][지갑 주소, 34 바이트] / [96 바이트 0-9A-F] / [32 바이트 0-9A-F] 를 파싱하였다. 이후 String Slicing 을 통하여 우리가 원하는 지갑 주소 34 바이트만 얻고, 중복을 제거하였다.

```
patterns = re.compile(b"[1][a-km-zA-HJ-NP-Z0-9]{33}[A-F0-9]{96}\|[A-F0-9]{32}\|")
with open(self.file, "rb") as f:
    data = f.read()
#f_results = [m for m in re.finditer(patterns, data)]
test = re.findall(patterns, data)
results = []
for i in test :
    results.append(i[:34].decode())
results=list(set(results))
```

[그림 39] 정규 표현식을 통한 Wallet 데이터 탐색

HD Account 역시 자체적인 규칙인 [0-9A-F 데이터] +지갑 주소 + “HW”를 기반으로 정규표현식을 작성하였고 String Slicing 을 통하여 우리가 원하는 지갑 주소 34 바이트만 추출한 뒤 중복을 제거하였다.

```
patterns2 = re.compile(b"[A-F0-9]{6}[1][a-km-zA-HJ-NP-Z0-9]{33}HW") # Hd account
test2 = re.findall(patterns2, data)
results2 = []
for i in test2 :
    results2.append(i[-36:-2].decode())
results2=list(set(results2))
```

[그림 40] HW Account 패턴에 대한 정규식

도구 사용

다음은 제안한 방법과 Yara Rule을 기반으로 메모리 덤프에서 Wallet Address를 탐색할 수 있는 도구이다.

[표 21] 도구 다운로드 링크

<https://drive.google.com/file/d/1-QGqB61xLQwqzn4AQuBpiKjZWCFCdFsn/view?usp=sharing>

위의 링크를 통해 도구를 다운로드하고, 압축을 해제하면 다음과 같은 구성을 확인할 수 있다.

이름	수정한 날짜	유형	크기
.idea	2021-09-30 오후 12:16	파일 폴더	
rules	2021-09-30 오후 1:27	파일 폴더	
final	2021-09-30 오후 2:45	Python File	7KB
strings	2021-06-22 오후 2:58	응용 프로그램	362KB
yara64	2021-08-23 오후 8:37	응용 프로그램	2,142KB

[그림 41] 압축 파일 해제 결과

내부에는 소스코드와 파이프라인의 대상이 되는 어플리케이션이 동봉되어 있음을 확인할 수 있다.

이름	수정한 날짜	유형	크기
armory.yar	2021-09-30 오후 2:47	YAR 파일	103KB
bitcoincore.yar	2021-09-30 오후 2:14	YAR 파일	1KB
bither.yar	2021-09-30 오후 12:32	YAR 파일	0KB
bitpay.yar	2021-09-30 오후 12:06	YAR 파일	1KB
electrum.yar	2021-09-30 오후 12:06	YAR 파일	1KB

[그림 42] rules 폴더 내부의 Yara Rule

또한 내부의 폴더 중 “rules” 폴더에서 저장된 Yara Rule을 확인할 수 있으며, 패턴이 변경될 시 해당 폴더 내부의 Yara Rule을 변경해야한다.

도구 설치 및 수행 환경

본 도구는 별도의 라이브러리 설치를 요구하지 않는다. 동봉된 어플리케이션 버전과 테스트 환경은 아래와 같다.

[표 22] 도구 테스트 운영체제 및 언어

OS	Windows 10 Pro 20H2 (19042.1237)
Python	3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)

[표 23] 연동 어플리케이션 버전

strings	2.5.4
yara64	4.1.2

도구 수행 예제

■ Armory

```
C:\Users\dhhyun\Documents\coin>python final.py --image_file armory.vmem --yara_dir rules
Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

[!] Address : 12LgTPtJToWKwYiQxb2V89xNDFWvCYP9q7
[!] Address : 15odW9lgvCH68cgsZco4B8hCVs1rnwdh7T
[!] Address : 18dr0e8fCCTn2J8fMsuU4Eqmg2LMQs5gnB
[!] Address : 14ABucVm0BjDUy/UqCHAqtEhZn8S3mb5d1
[!] Address : 1L4UNq3uD6AvUFydvBdjS6EszA2FVbY8xn
```

[그림 43] Armory Wallet Address 탐색 결과

■ Bitcoin Core

```
C:\Users\dhhyun\Documents\coin>python final.py --image_file bitcoincore.vmem --yara_dir rules
[+] Offset: 0x577774
[-] Address b'3kn3m5QcT3JgJbbau5pScodShBz2gFTob'
[+] Offset: 0x54b3f294
[-] Address b'39say4AKWkGEzVKwuE7jLkVJunM1kyd6xN'
[+] Offset: 0x680c6474
[-] Address b'3lw4gtYZ9j fake7yx5ae6o8Ahq3Ct19GR'
[+] Offset: 0x30cb80f4
[-] Address b'bc1qfy2bzuj xmueve6jyggy56d7hsqsw451c82tf2p5'
[+] Offset: 0x368632f4
[-] Address b'bc1qamikaka6192qup0q32k0jupkjyr0zr5etfgsze'
[+] Offset: 0x56641b74
```

[그림 44] Bitcoin Core Wallet Address 탐색 결과

■ Bither

```
C:\Users\dhhyun\Documents\coin>python final.py --image_file bither.vmem --yara_dir rules
[*] wallet address
['10HX2An3HJAZjdKyR4XVQihgHHOp76woq', '1AfH4UxzJJVSMeV2zfWLpRuxLGYzcr2BeL', '1PGndu3yyRQjtaxQmc6kKyEyCmcA2Hous5']
[*] HD account address
[]
```

[그림 45] Bither Wallet Address 탐색 결과

■ Bitpay

```
C:\Users\dhhyun\Documents\coin>python final.py --image_file bitpay.vmem --yara_dir rules
[!] Bitpay Wallet Address
[-] Address: bc1q4q0g4rzqcvhtqdardt99ep2ehsfnjpfshew8d8
[-] Address: bc1qwoenyzhstr3jcnepknanqmuveyedvrrtmtfdf
[-] Address: bc1qwoenyzhstr3jcnepknanqmuveyedvrrtmtfdf
[-] Address: bc1qwoenyzhstr3jcnepknanqmuveyedvrrtmtfdf
[-] Address: bc1q4q0g4rzqcvhtqdardt99ep2ehsfnjpfshew8d8
[-] Address: bc1q4q0g4rzqcvhtqdardt99ep2ehsfnjpfshew8d8
```

[그림 46] Bitpay Wallet Address 탐색 결과

■ Electrum

```
C:\Users\dhhyun\Documents\coin>python final.py --image_file electrum.vmem --yara_dir rules
[!] Electrum Wallet Address
[+] Offset: 0x27e511f5
[-] Address: bc1qj459qf76hyg78z4v677na80p65vsqjfer5ylwl
[+] Offset: 0x4e591378
[-] Address: bc1q57xmkgj jps6scf3a2u5xaahu7f77gxc829u45t
[+] Offset: 0x4e591589
[-] Address: bc1q85fqtwmhzaep6hs0ne6c458rgjnfhvzaqw7zch
[-] Offset: 0x4e59170e
```

[그림 47] Electrum Wallet Address 탐색 결과

작성한 Rule 간 False Positive가 존재하는지, Completeness를 확인하기 위해 생성한 샘플 간 교차 검증을 진행하였다. 결과는 아래와 같다.

■ Bitcoin core wallet address

```
C:\Users\dhhyun\Documents\coin>python final.py --image_file bitcoincore.vmem --yara_dir rules
[!] Armory Wallet Address

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

[!] Bitpay Wallet Address
[!] Electrum Wallet Address
[!] Bither Wallet Address
[*] wallet address
[*]
[*] HD account address
[*]
[!] Bitcoin Core Wallet Address
[+] Offset: 0x757774
[-] Address b'3Kn3m5qcT3JgJbbauu5pScodShBz2qFTob'
[+] Offset: 0x54b3f294
[-] Address b'39saY4AKWkGEzVKwuE7jLkVJunM1Kyd6xN'
[+] Offset: 0x680c474
[-] Address b'3Nw4qtYE79ifake7yx5aeGoR4h3Ct19GB'
```

[그림 48] Bitcoin core rule 검증

■ Bither wallet address

```
C:\Users\dhhyun\Documents\coin>python final.py --image_file bither.vmem --yara_dir rules
[!] Armory Wallet Address

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

warning: rule "armory" in rules\armory.yar(15699): rule is slowing down scanning
error: rule "armory" in rules\armory.yar(15699): too many strings in rule "armory" (limit: 10000)
[!] Bitpay Wallet Address
[!] Electrum Wallet Address
[!] Bither Wallet Address
[*] wallet address
[*] '1AfH4UxzJJVSMev2ZfWLpRuxLGYzcr2BeL', '1PGmdU3yyRQjtaxQmc6kKyEyCmcA2Hous5', '1CHX2An3HJAZjdKyR4XVQi hgHH0p76woqo'
[*] HD account address
[*]
[!] Bitcoin Core Wallet Address
```

[그림 49] Bither rule 검증

■ Bitpay wallet address

```
C:\Users\dhhyun\Documents\coin>python final.py --image_file bitpay.vmem --yara_dir rules
[!] Bitpay Wallet Address
[-] Address: bc1q4q0g4rzqcvhtqdardt99ep2ehsfnjpfshев8d8
[-] Address: bc1awpenyzhstr3jcnepknanqmuvyeyedvrrtmtfdf
[-] Address: bc1qj0kwsktlapkgz9lp23wxyqd4asrlq10v8a0n6k6
[!] Electrum Wallet Address
[!] Bither Wallet Address
```

[그림 50] Bitpay rule 검증

■ Electrum wallet address

```
C:\Users\dhhyun\Documents\coin>python final.py --image_file electrum.vmem --yara_dir rules
[!] Armory Wallet Address

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

[!] Bitpay Wallet Address
[!] Electrum Wallet Address
[+] Offset: 0x27e511f5
[-] Address: bc1qj459qf76hyg78z4v677na80p65vsqjer5ylwl
[+] Offset: 0x4e591378
[-] Address: bc1qg7xmkgj jps6scf3a2u5xaahu7f77gxc829u45t
[+] Offset: 0x4e591589
[-] Address: bc1q85fqtwmhzaep6hs0ne6c458rgjnfhvzaqw7zch
[+] Offset: 0x4e59179a
[-] Address: bc1q85fqtwmhzaep6hs0ne6c458rgjnfhvzaqw7zch
```

[그림 51] Electrum rule 검증

■ Armory wallet address

```
C:\Users\ldhyun\Documents\coin>python final.py --image_file electrum.vmem --yara_dir rules
[!] Armory Wallet Address

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

[!] Address : 36SomeUnparsedVersionsExist
[!] Address : 36SomeUnparsedVersionsExist6
[!] Address : 36SomeUnparsedVersionsExist7
[!] Address : 37SomeUnparsedVersionsExist
[!] Address : 11SomeUnparsedVersionsExist1
[!] Address : 35SomeUnparsedVersionsExistC
[!] Address : 13SomeUnparsedVersionsExist
[!] Address : 38SomeUnparsedVersionsExist0
[!] Address : 38SomeUnparsedVersionsExist
[!] Address : 38SomeUnparsedVersionsExistc
[!] Address : 38SomeUnparsedVersionsExist8
[!] Address : 38SomeUnparsedVersionsExistF
[!] Address : 35SomeUnparsedVersionsExist4
[!] Address : 1cSomeUnparsedVersionsExist1
[!] Address : 10SomeUnparsedVersionsExist1
[!] Address : 10SomeUnparsedVersionsExist
[!] Address : 10SomeUnparsedVersionsExistE
[!] Address : 10SomeUnparsedVersionsExistk
```

[그림 52] Armory rule 검증 (정교화 이전)

Armory의 경우 Electurm 관련 메모리 덤프에서 [그림 52]와 같이 일부 False Positive가 발생함을 확인할 수 있었다. 해당 False Postive의 경우 모두 같은 형태의 패턴(3XSomeUnparsedVesionExist~, 1XSomeUnparsedVesionExist~)로 일반적인 비트코인 지갑 주소에서는 발견되지 않는 형태(SomeUnparsedVesionExist)를 가지고 있다. 해당 형태를 Blacklist화 하여 Rule의 정교화를 추가하였다.

```
C:\Users\ldhyun\Documents\coin>python final.py --image_file electrum.vmem --yara_dir rules
[!] Armory Wallet Address

Strings v2.54 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2021 Mark Russinovich
Sysinternals - www.sysinternals.com

[!] Bitpay Wallet Address
[!] Electrum Wallet Address
+| Offset: 0x27e511f5
|-| Address: bc1qj459qf76hyg78z4v677na80p65vsqjfer5ylwl
+| Offset: 0x4e591378
|-| Address: bc1g57xmkgjjps6scf3a2u5xaahu7f77gxc829u45t
+| Offset: 0x4e591589
|-| Address: bc1q851qtwmhzaep6hs0ne6c458rgjnfhvzaqw7zch
+| Offset: 0x4e59179a
|-| Address: bc1q9tm9sed4z7mm84jh4ax9p7h68s6gmjt27nyuhk
+| Offset: 0x4e5919b5
|-| Address: bc1q9tm9sed4z7mm84jh4ax9p7h68s6gmjt27nyuhk
```

[그림 53] Armory rule 검증 (정교화 이후)

■ Final Performance

최종적으로 생성한 Rule과 이를 활용하는 Source code의 성능은 아래의 그림과 같이 False Positive나 True Negative 없이 터지하고 있음을 확인할 수 있다.

[표 24] 최종 성능 측정 결과표

	Bitcoin core	Bitpay	Electrum	Armory	Bither
Bitcoin core	O	X	X	X	X
Bitpay	X	O	X	X	X
Electrum	X	X	O	X	X
Armory	X	X	X	O	X
Bither	X	X	X	X	O