AIS 2023
2023 사이버위협 대응 인공지능 보안 컨퍼런스

AI-based Digital Evidence Enhancement Technology
for Threat Intelligence Analysis

DBP Deep Binary Profiler

SANDS Lab

Hyunjong Lee hjlee@sandslab.io
ChangGyun Kim cgkim@ksign.com


Who We Are

Team Members

SANDS Lab

Research

- DBP(Deep Binary Profiler) : 코드 유사도 기반 위협 정보 자동 분석 기술 연구
- DDP(Deep Document Profiler) : 다차원 메타데이터 정보 기반의 위협 정보 자동 분석 기술 연구
- DRP(Deep Rule Profiler) : 어텐션 기반의 위협 코드 식별을 위한 룰 자동 생성 기술 연구

Achievement

- NET 인증 “바이너리 역공학 기반 공격자 프로파일링 기술”
- NET 인증 “다차원 메타 데이터 추출 분석 기반 비실행형 악성코드 프로파일링 및 탐지 기술”

Conference

- RSA Conference 2023 참가
- VB Conference 2023 CTA TIPS, DBP 발표
- AVAR Conference 2023, DBP 기술 발표 예정

Our Research Goal

Threat Analytics
To draw insights from data and make our data valuable

AI-based Threat Profiling
To analyze a large quantities of variant threats and assist code analysts

Objective Threat Evidence
To provide objective evidence and explain why it is malicious

Outline

North Korea Cyber Attack?

’13 3·20 Cyber Attack

Malware implanted in the management server;
possibility of North Korean involvement

Some media reports have said that computers failed to boot up properly, and displayed an image of three skulls alongside a message claiming that the systems had been “hacked by Whois Team”.

However, in Sophos's testing so far we have not been able to replicate this payload.

According to a Reuters report, LG U+, the company which provides internet services to at least some of the companies named above, says that it believes its network was hacked.

The malware, detected proactively by Sophos products as Mal/EncPk-ACE, has been dubbed “DarkSeoul” by experts analysing its code at SophosLabs.

What's curious is that the malware is not particularly sophisticated. Sophos products have been able to detect the malware for nearly a year, and the various commands embedded in the malicious code have not been obfuscated.

For this reason, it's hard to jump to the immediate conclusion that this was necessarily evidence of a “cyberwarfare” attack coming from North Korea.

’14 Sony Pictures Hacking

Film production company behind
Kim Jong-un assassination movie hacked;
investigating ties to 'North Korean involvement'.

Sony hack: White House views attack as security issue

19 December 2014

The interview stars Seth Rogen and James Franco as journalists enlisted to kill Kim Jong-un

A cyber attack on Sony Pictures that forced the cancellation of a major film release is being seen as a serious national security matter, the US says.

A White House spokesman said the US believed the hacking was the work of a "sophisticated actor" - but refused to confirm if North Korea was responsible.

Sony withdrew The Interview, a new comedy film about North Korea's leader, after threats from hackers.

Hackers have already released sensitive information stored on Sony computers.

They later issued a warning to members of the public planning to see The Interview.

Referring to the 11 September 2001 terror attacks, they said "the world will be full of fear" if the film was screened.

’16 Interpark data leak

Interpark personal data breach linked
to North Korea;
attackers exploited weak security measures.

Breaking | N. Korea behind Interpark's massive customer data leak: police

South Korean police said Thursday that North Korea was behind the latest hacking of a leading online shopping mall, which led to the leak of personal information of some 10 million customers.

The remark came after police conducted a detailed probe into the server of Interpark Corp., after an unidentified entity broke into it and stole customer-related information in May.

Police said the Internet Protocol addresses used by the hackers were identical to those used by North Korean hackers in previous cases. The malicious codes used were also similar to past examples of North Korean foul play, it added.

Experts said North Korea's hacking attempts apparently followed the latest economic sanctions on the communist's regime, which induced it to find other ways to raise foreign currency.

Interpark's 10.3 million customer information leaked

2016-07-26 18:26 | Companies

After the hackers obtained the data from Interpark in May, they also sent e-mails to the company's executives, asking for 3 billion won ($2.66 million) in bitcoins, a virtual currency exchangeable online.

One of the Korean-language e-mails, included vocabulary used only in the North, a tell-tale sign indicating that Pyongyang was behind the cyber attacks. (Yonhap)

’17 WannaCry Ransomware

Global ransomware shock, variants spreading...
South Korea issues 'Monday alert'.

Massive ransomware infection hits computers in 99 countries

13 May 2017

The ransomware has been identified as WannaCry - here shown in a safe environment on a security researcher's computer

A massive cyber-attack using tools believed to have been stolen from the US National Security Agency (NSA) has struck organisations around the world.

Cyber-security firm Avast said it had seen 75,000 cases of the ransomware - known as WannaCry and variants of that name - around the world.

There are reports of infections in 99 countries, including Russia and China.

Among the worst hit was the National Health Service (NHS) in England and Scotland.

The BBC understands **about 40 NHS organisations and some medical practices** were hit, with operations and appointments cancelled.

"Who, How, Why? We need answers!"

Outline

AI-based digital evidence enhancement technology
for Cybersecurity Threat Response

1 · **Background**

2 · **Deep Binary Profiler**

3 · **Case Study**

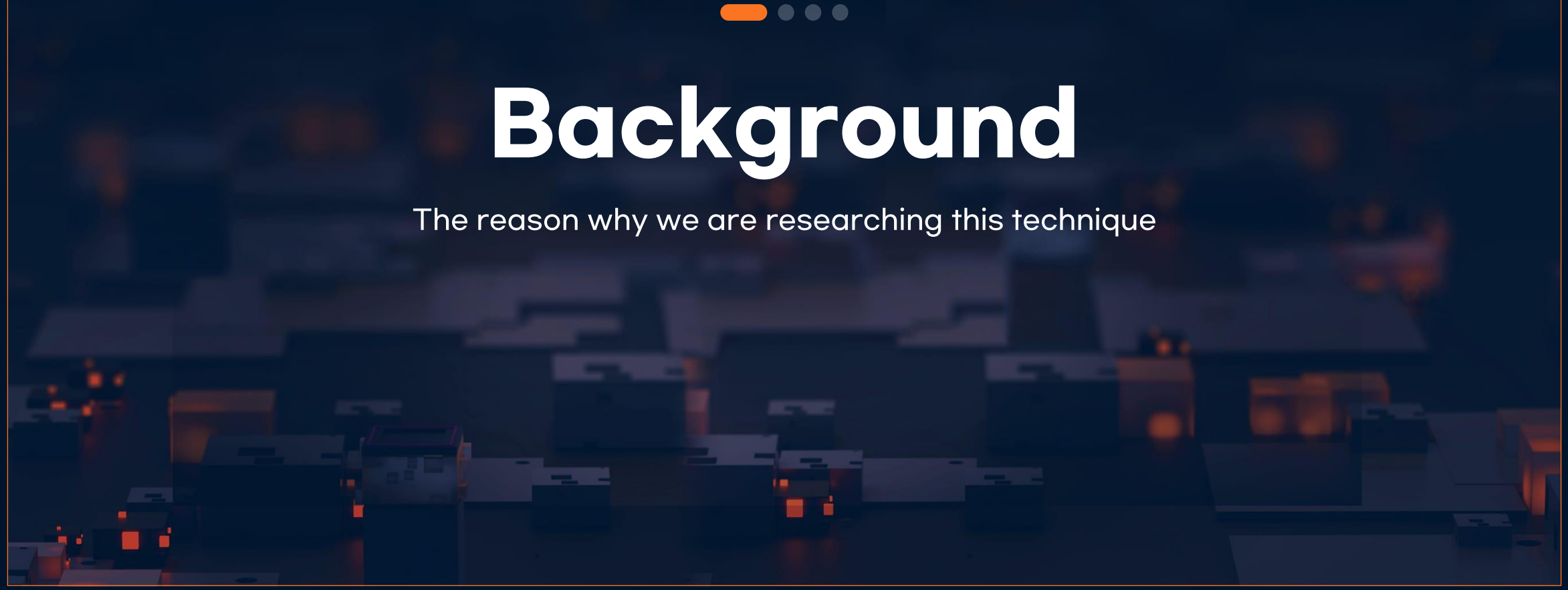
4 · **Summary**

SANDS Lab

1

Background

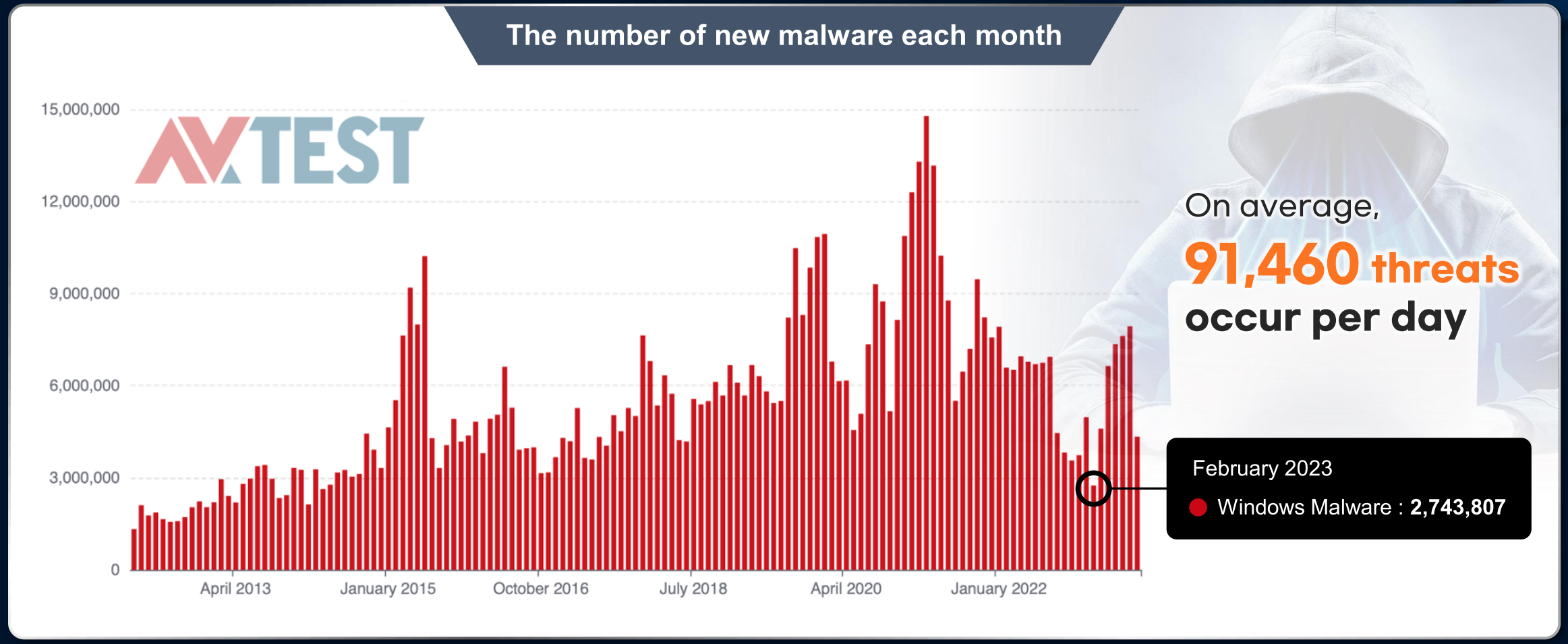
The reason why we are researching this technique



Background : Keywords

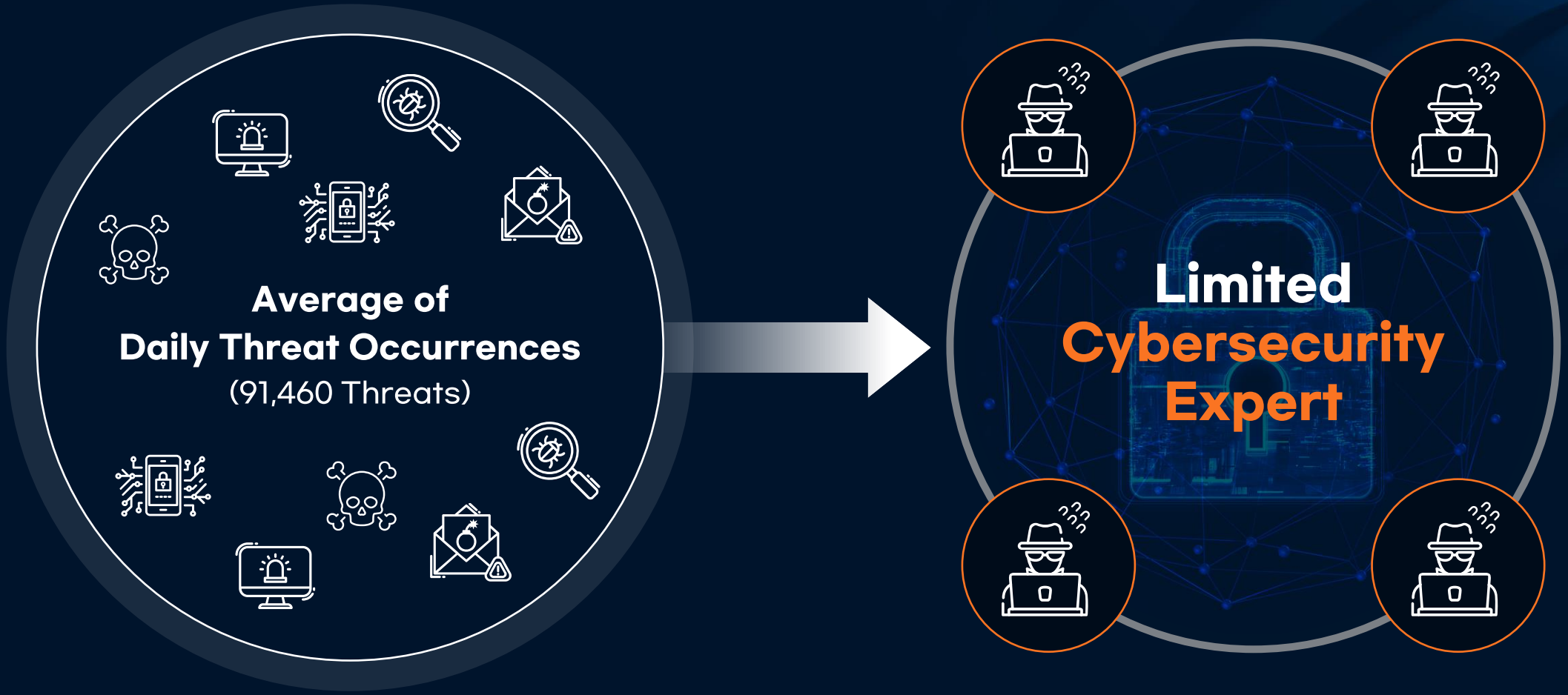
Background : Limitations of Manual Analysis

Every month, a significant number of threats are occurring,
and the volume of new malware is steadily increasing.



Background : Limitations of Manual Analysis

Analyzing the daily surge of threats manually with limited cybersecurity experts is impractical.



Background : High-level Requirements

Analyzing the intent and functionality of threats at the assembly code level through malware reverse engineering requires advanced skills and can only be carried out by a limited number of experts.

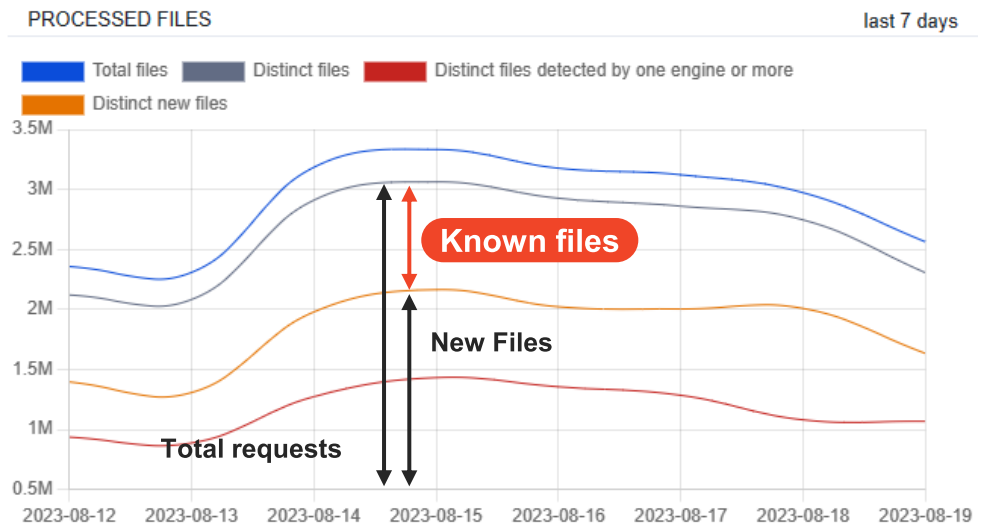
Technical Requirements for Threat Profiling

- **Threat Analysis** (Origin, Distribution, Potential Target)
- **Malware Development Knowledge** (Attack Techniques)
- Memory Analysis
- Behavioral Analysis
- Static and Dynamic Analysis
- Assembly Language
- Reverse Engineering
- Networking Protocols
- Computer Fundamentals
- Programming Basics

Background : Recurrence of Variant Threats

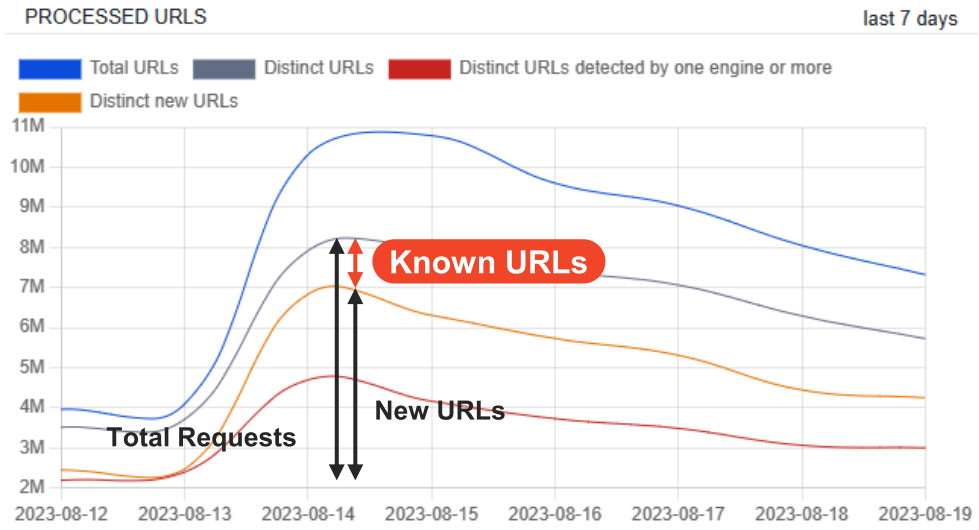
Based on the VirusTotal statistics as of August 19, 2023.

Percentage of Known Files



29.32% | **931,134 Files**

Percentage of Known URLs



13.70% | **1,085,249 URLs**

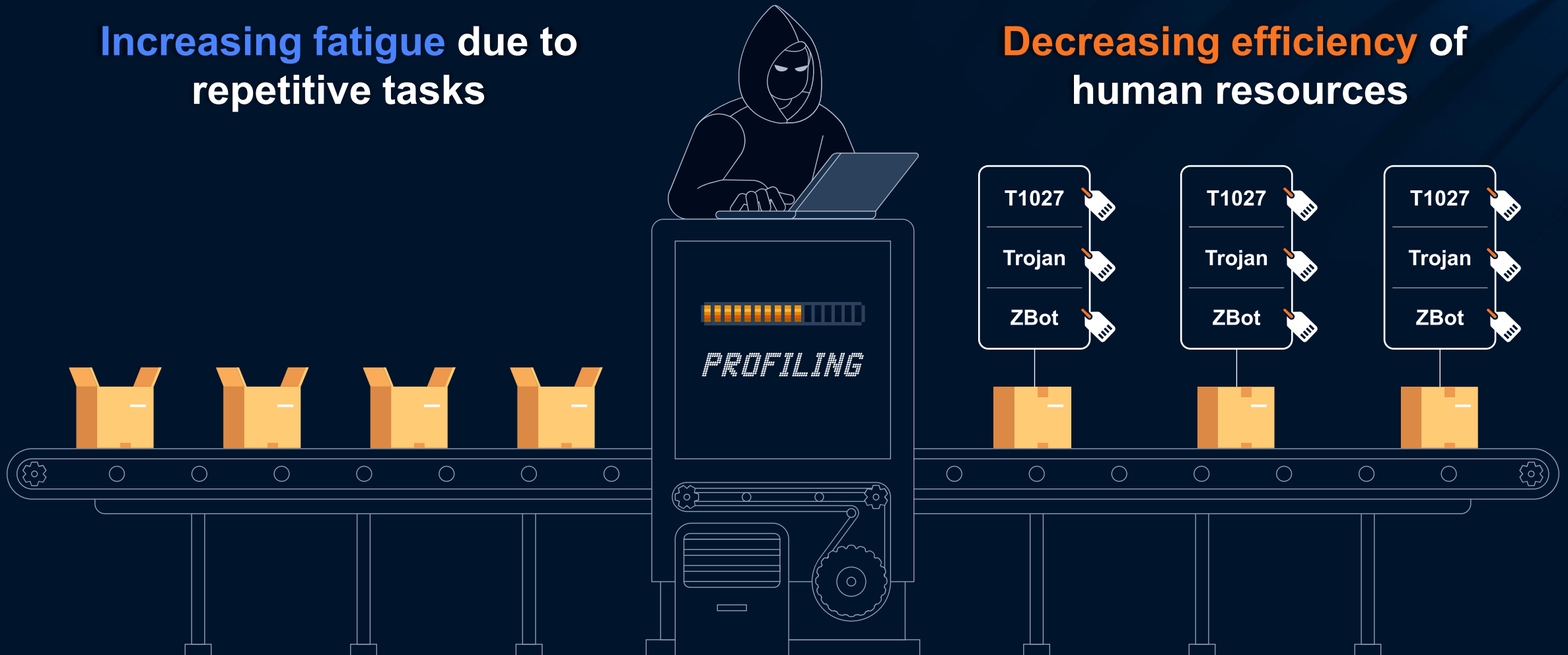
While the percentages look low, the actual numbers represent a substantial volume.

Background : Recurrence of Variant Threats

Repeated threats with the same or nearly identical patterns make profiling tasks into mere manual labor.

Increasing fatigue due to repetitive tasks

Decreasing efficiency of human resources



Background : Subjectivity in Analysis

Threat profiling often relies on the personal experiential knowledge of analysts, making it challenging to ensure objectivity.

Analysts A

This threat is a **false positive**. It may appear to be a cyber attack, but it's not actually one.

Analysts B

This is malware created by **Lazarus Threat Actor in North Korea**, with a function implemented using T1082 technique, and it appears to be a **Trojan** malware.

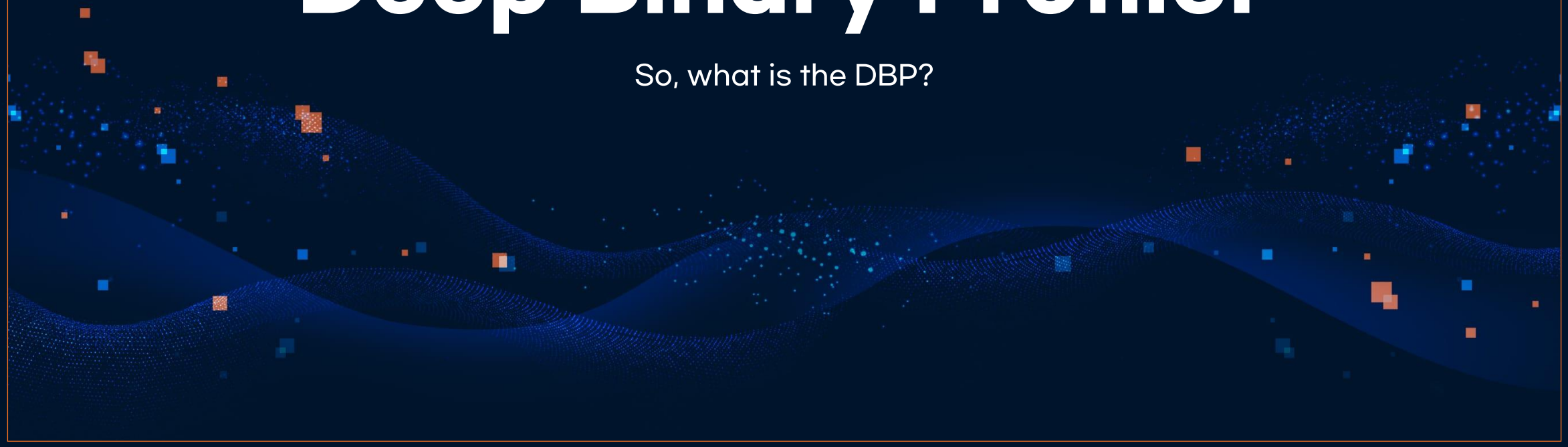
Analysts C

Attributed to APT17, there is a significant presence of functions implementing **the Discovery Tactic internally.** Furthermore, with its use of **T1497 for virtual environment detection**, it appears to be an **advanced Trojan malware** likely designed to evade dynamic analysis

2

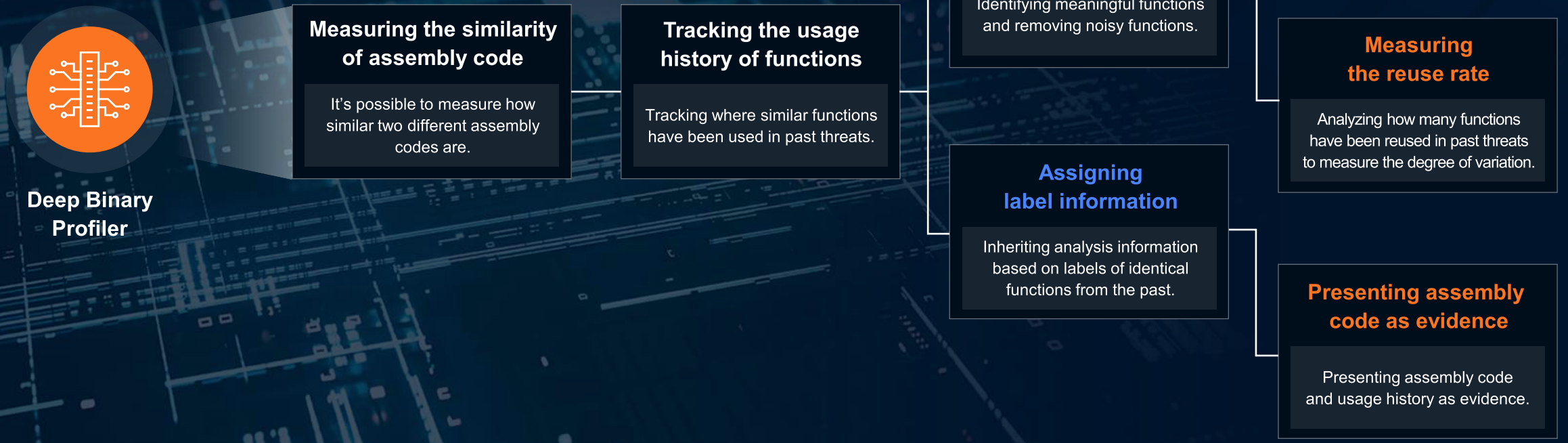
Deep Binary Profiler

So, what is the DBP?



Deep Binary Profiler : Technical Tree

Deep Binary Profiler(DBP) is a technology that separates a threats into multiple functions and discovers Function reuse from past threats.



Deep Binary Profiler : Embedding Model

To calculate the similarity between two different assembly codes numerically, they should be represented as numerical vectors rather than text.

```
; =============== S U B R O U T I N E =======================================

; Attributes: bp-based frame

sub_32047EE0    proc near               ; DATA XREF: .rdata:320495F8↓o

var_C           = dword ptr -0Ch
var_4           = dword ptr -4
arg_4           = dword ptr  0Ch

; FUNCTION CHUNK AT .text:32047680 SIZE 00000025 BYTES

; __unwind { // SEH_32047EE0
                push    ebp
                mov     ebp, esp
                push    0FFFFFFFFh
                push    offset SEH_32047EE0
                mov     eax, large fs:0
                push    eax
                mov     eax, dword_3205B1C0
                xor     eax, ebp
                push    eax
                lea     eax, [ebp+var_C]
                mov     large fs:0, eax
                mov     dword_32061A68, 0
                mov     dword_32061A6C, 0
                mov     dword_32061A70, 0
;   try {
                mov     [ebp+var_4], 0
                push    offset stru_32061A74 ; lpCriticalSection
                call    ds:InitializeCriticalSection
;   } // starts at 32047F20
                mov     [ebp+var_4], 0FFFFFFFFh
                push    offset sub_320481F0 ; void (__cdecl *)()
                call    _atexit
                add     esp, 4
                mov     ecx, [ebp+var_C]
                mov     large fs:0, ecx
                pop     ecx
                mov     esp, ebp
                pop     ebp
                retn
; } // starts at 32047EE0
sub_32047EE0    endp
```

×

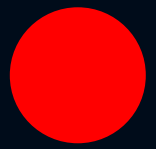
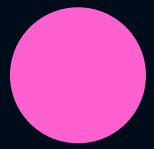
```
; =============== S U B R O U T I N E =======================================

; Attributes: bp-based frame

; void __cdecl sub_32048120()
sub_32048120    proc near               ; DATA XREF: sub_32047E10+59↑o

var_C           = dword ptr -0Ch
var_4           = dword ptr -4
arg_4           = dword ptr  0Ch

; FUNCTION CHUNK AT .text:32047810 SIZE 00000025 BYTES

; __unwind { // SEH_32048120
                push    ebp
                mov     ebp, esp
                push    0FFFFFFFFh
                push    offset SEH_32048120
                mov     eax, large fs:0
                push    eax
                sub     esp, 0Ch
                mov     eax, dword_3205B1C0
                xor     eax, ebp
                push    eax
                lea     eax, [ebp+var_C]
                mov     large fs:0, eax
;   try {
                mov     [ebp+var_4], 0
                push    offset stru_32061A9C ; lpCriticalSection
                call    ds:DeleteCriticalSection
;   } // starts at 32048145
                mov     [ebp+var_4], 0FFFFFFFFh
                mov     ecx, offset dword_32061A90
                call    sub_3201F200
                mov     ecx, [ebp+var_C]
                mov     large fs:0, ecx
                pop     ecx
                mov     esp, ebp
                pop     ebp
                retn
; } // starts at 32048120
sub_32048120    endp
```

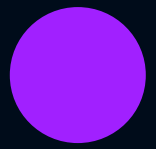
It’s not numerical, so similarity calculation is not possible

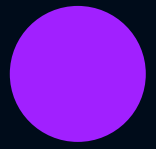
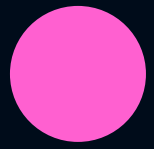
Deep Binary Profiler : Embedding Model

In the case of color data, since the combinations of colors for red, green, and blue can be represented as a three-dimensional vector, similarity calculation is possible.


Black (0, 0, 0) × 
White (255, 255, 255) → **6% similarity**


Red (255, 0, 0) × 
Pink (255, 96, 208) → **57% similarity**

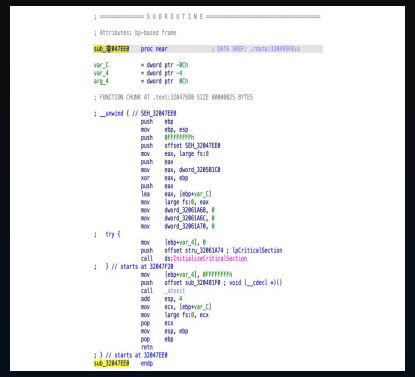

Purple (160, 32, 255) × 
Sky blue (80, 208, 255) → **50% similarity**


Purple (160, 32, 255) × 
Pink (255, 96, 208) → **63% similarity**

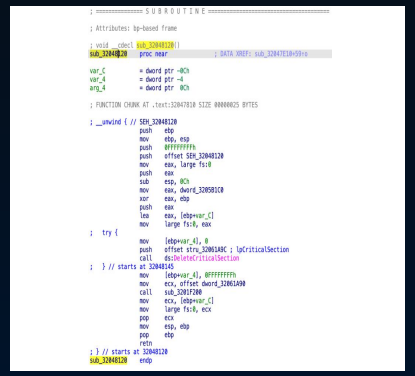
Deep Binary Profiler : Embedding Model

To obtain numerical vectors, including the meaning of assembly code, embedding AI models are used, enabling similarity calculations

Source



Assembly Code A



Assembly Code B

Vectorization

Embedding model

Similarity Calculation

Vector A

0.5 1.2 0.3 … 1.5 1.0

×

Vector B

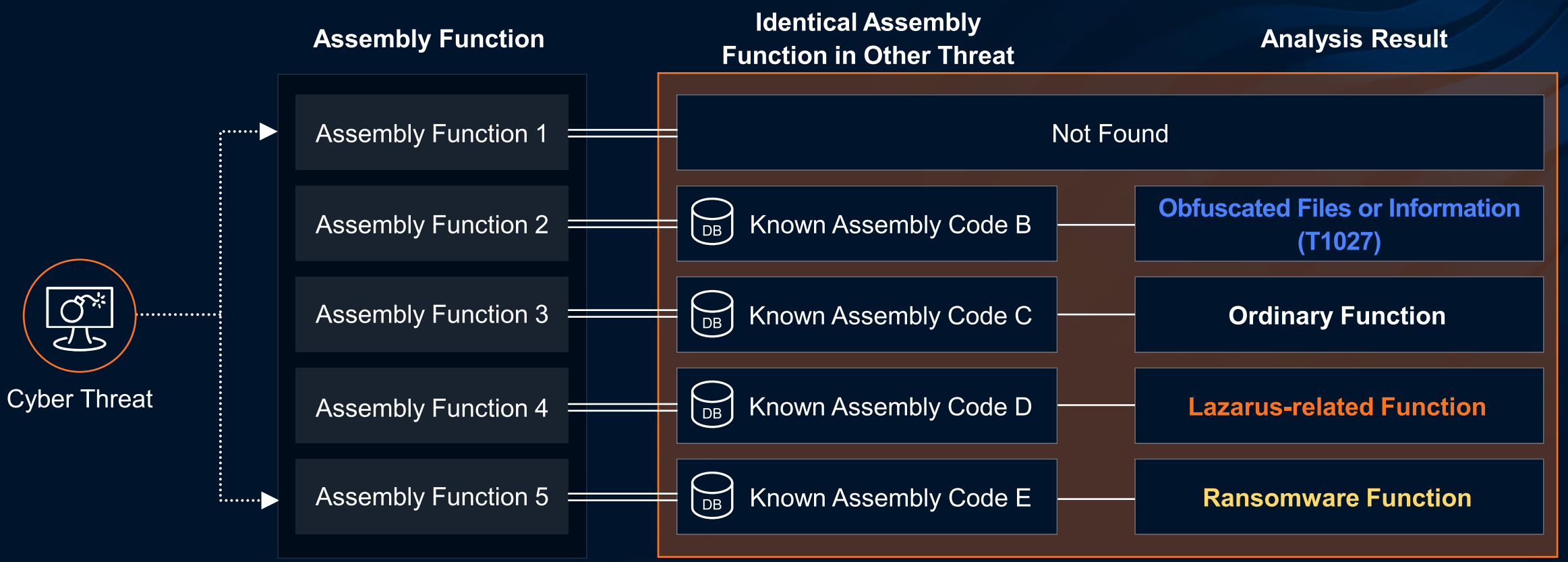
0.5 1.2 0.3 … 1.5 1.0

= 99% Similarity

= Identical Function

Similar assembly codes are transformed into similar vector representations, resulting in high similarity measurements.

Deep Binary Profiler : The Flow of Profiling

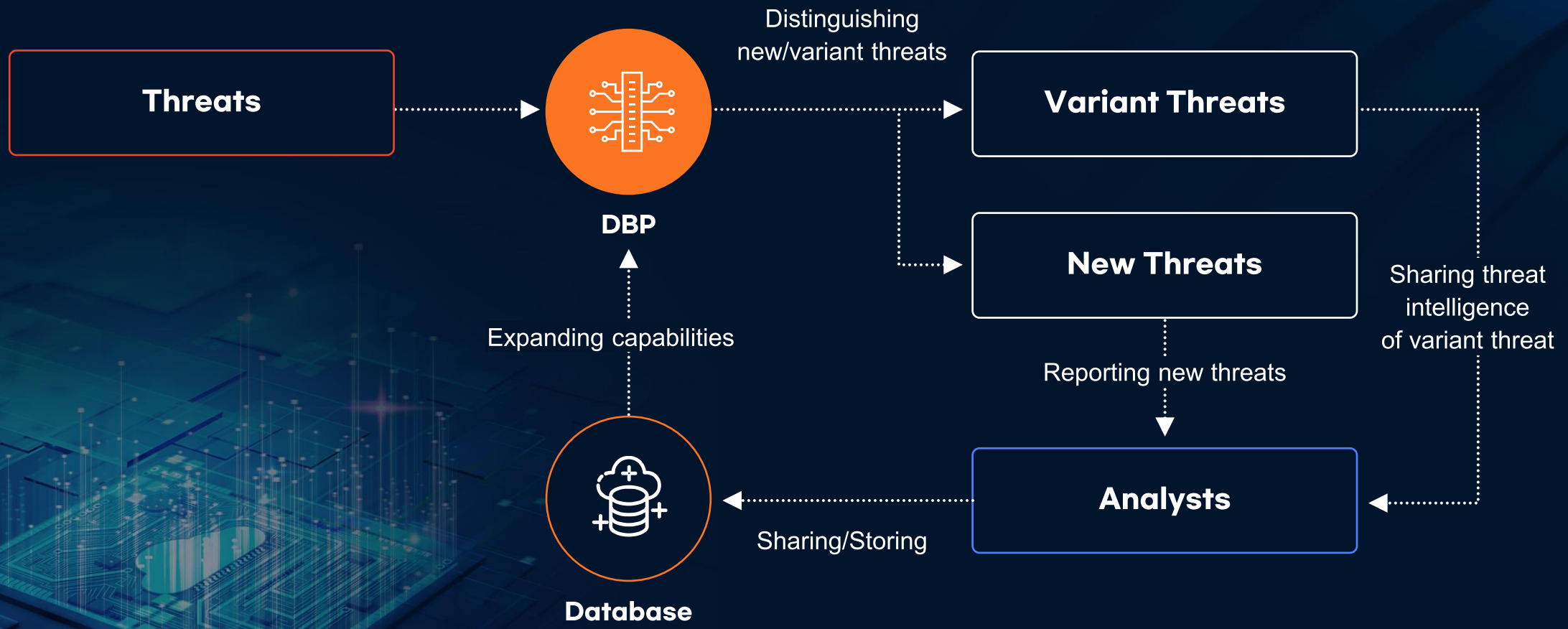


Deep Binary Profiler

This threat may belong to the Lazarus Group as it incorporates a function associated with T1027. Furthermore, it reuses a function commonly employed by Ransomware. As a result, this threat shows a 92% similarity with a prior threat known as Threat A

Deep Binary Profiler : Extending the Capabilities

Through the sharing of threat intelligence, we can establish a virtuous cycle that reduces the workload of analysts while storing meaningful analysis results in a database, thereby expanding DBP's analytical capabilities



Deep Binary Profiler : DBP Output

Once the DBP analysis is completed, the following analysis results are generated by function, and these function analysis results are aggregated to create summary information.

Function Analysis Result

The ratio of each label

Function name

Identical function in other malware

```
{
  "threat_category": {
    "trojan": 0.75,
    "dropper": 0.25
  },
  "actor": {
    "lazarus": 1
  },
  "tid": {
    "T1543_003": 1
  },
  "verdict": {
    "malware": 1
  },
  "name": "sub_403690",
  "length": 100,
  "tag": [
    "trojan",
    "dropper",
    "lazarus",
    "T1543_003"
  ],
  "n_identical_function": 3,
  "identical_with": [
    {
      "sha256": "E2ECEC43DA974DB02F624ECADC94BAF1D21FD1A5C4990C15863BB9929F781A0A",
      "func": "sub_402BF0"
    },
    {
      "sha256": "0753F8A7AE38FDB830484D0D737F975884499B9335E70B7D22B7D4AB149C01B5",
      "func": "sub_403660"
    },
    {
      "sha256": "201A9C5FE6A8AE0D1C4312D07EF2066E5991B1462B68F102154BB9CB25BF59F9",
      "func": "sub_403270"
    }
  ]
},
```

Summary Information

Counting by labels within functions

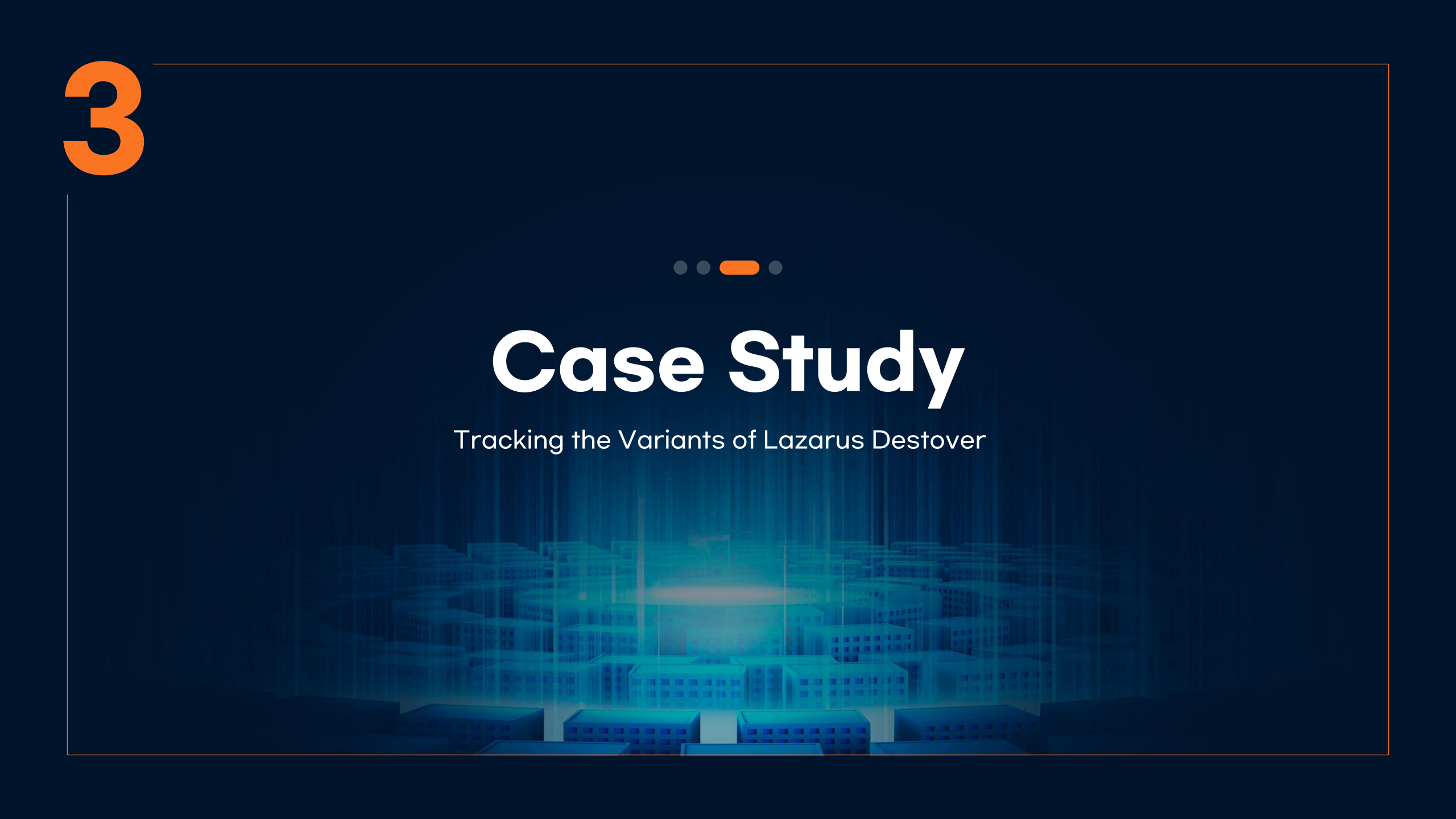
Predecessor Malware and the ratio of function reuse

```
"sha256": "4D4B17DDBCF4CE397F76CF0A2E230C9D513B23065F746A5EE2DE74F447BE39B9",
"summary": {
  "n_function": 47,
  "verdict": {
    "malware": 47
  },
  "threat_category": {
    "trojan": 46,
    "dropper": 7,
    "downloader": 1
  },
  "actor": {
    "lazarus": 29,
    "apt17": 16
  },
  "tid": {
    "T1543_003": 2,
    "T1083": 2,
    "T1070_006": 1,
    "T1016": 2,
    "T1134": 1,
    "T1027": 1
  },
  "n_variant": 6,
  "variant_from": [
    {
      "sha256": "201A9C5FE6A8AE0D1C4312D07EF2066E5991B1462B68F102154BB9CB25BF59F9",
      "count": 46,
      "ratio": 0.9787234042553191
    },
    {
      "sha256": "8A4FC5007FAF85E07710DCA705108DF9FD6252FE3D57DFADE314120D72F6D83F",
      "count": 31,
      "ratio": 0.6595744680851063
    },
    {
      "sha256": "6CE54331E126FD18C94E854A5E7FE3650A125CC83604F1A27A28F383E5193C07",
      "count": 31,
      "ratio": 0.6595744680851063
    },
```

3

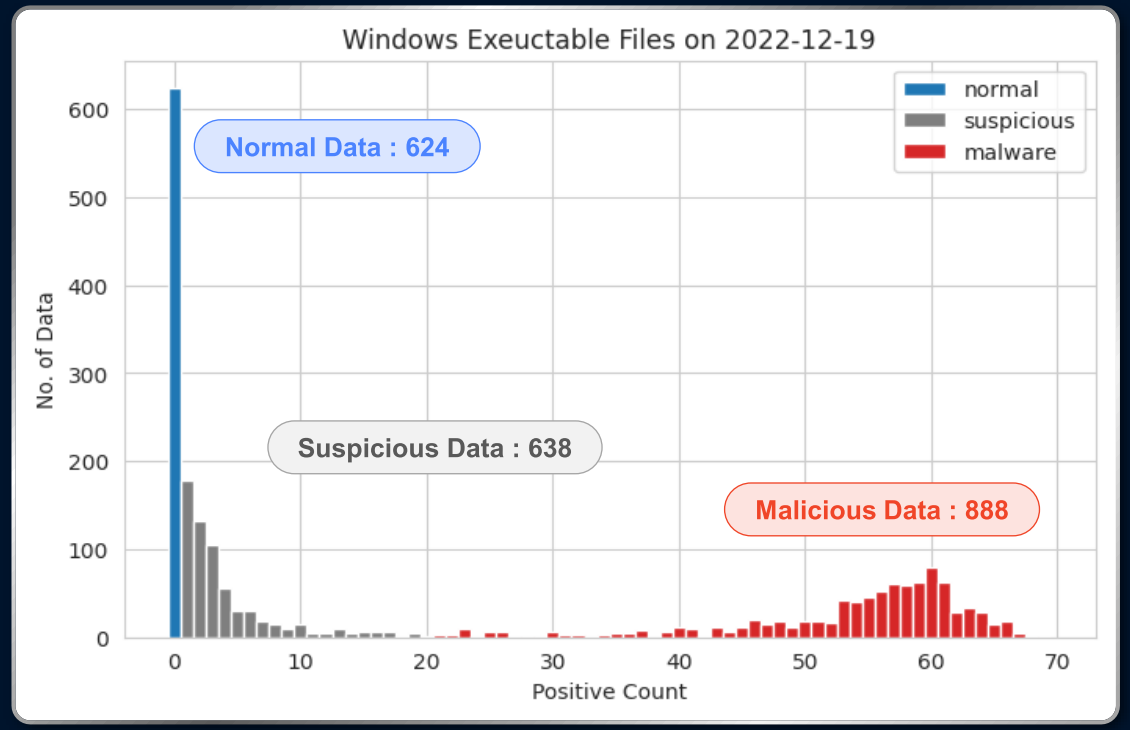
Case Study

Tracking the Variants of Lazarus Destover

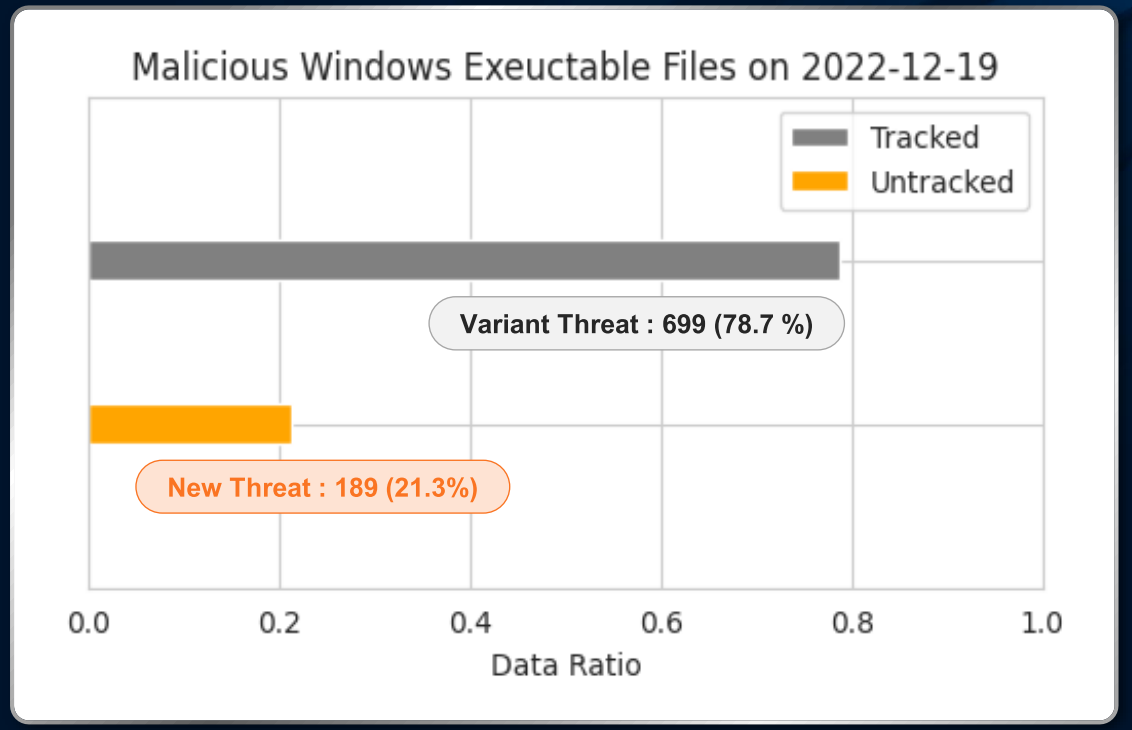


Case Study : Distinguishing New/Variant Threats

Analysis of 2,150 Data Samples Collected on December 19, 2022



Data Distribution Based on Antivirus Detection Count

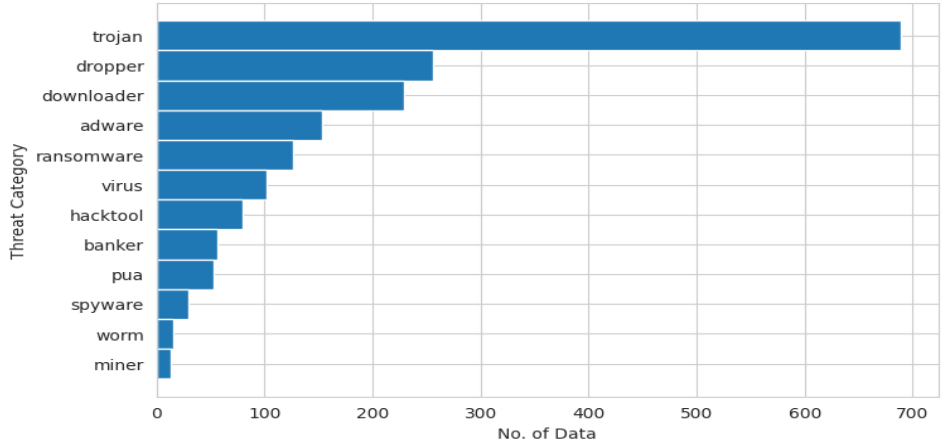


Ratio of New/Variant Threats in DBP Analysis Results

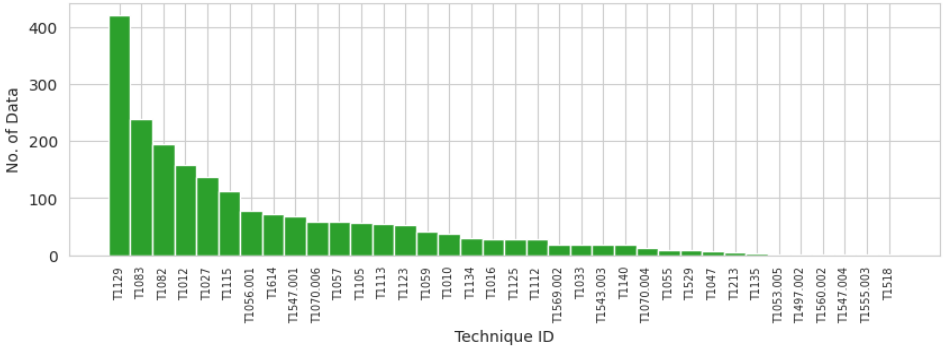
Case Study : Analyzing the Output of DBP

Basic statistical analysis for 699 threats associated with past threats.

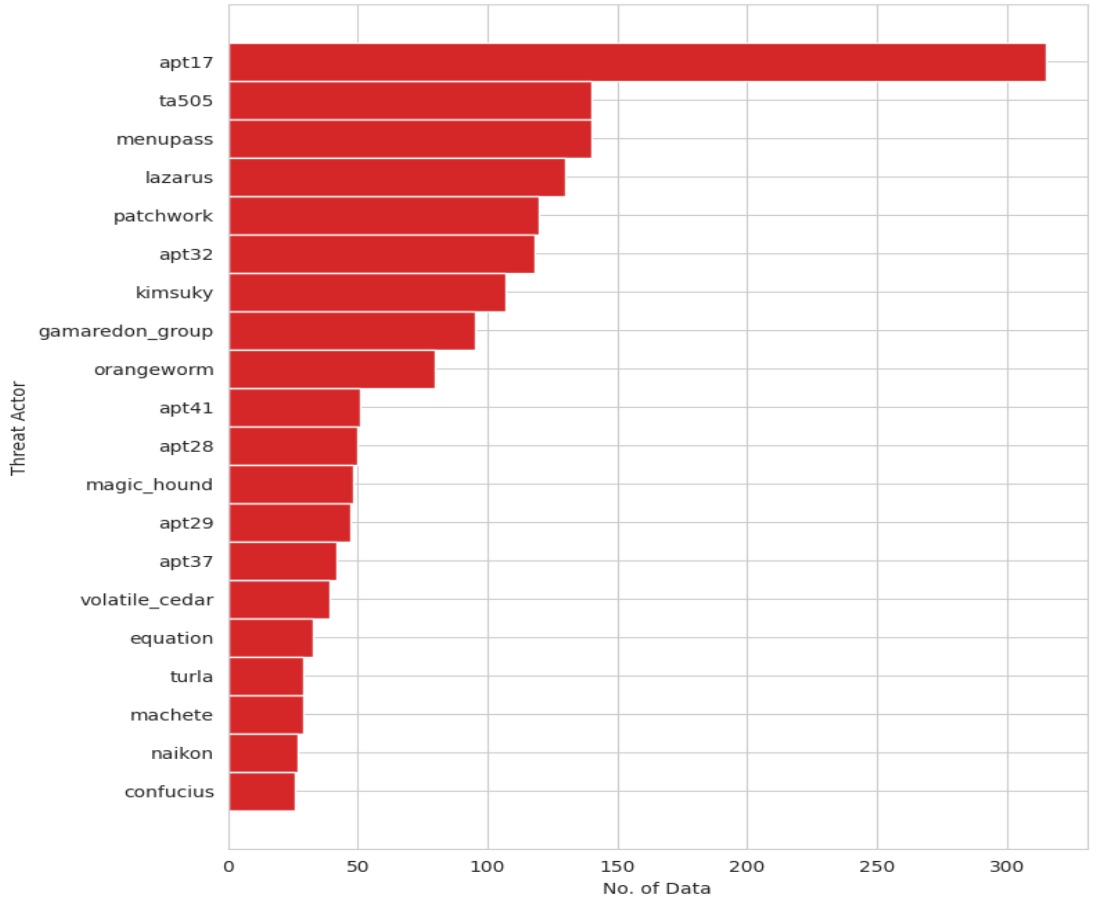
Threat Category of Tracked Malware



Technique ID of Tracked Malware



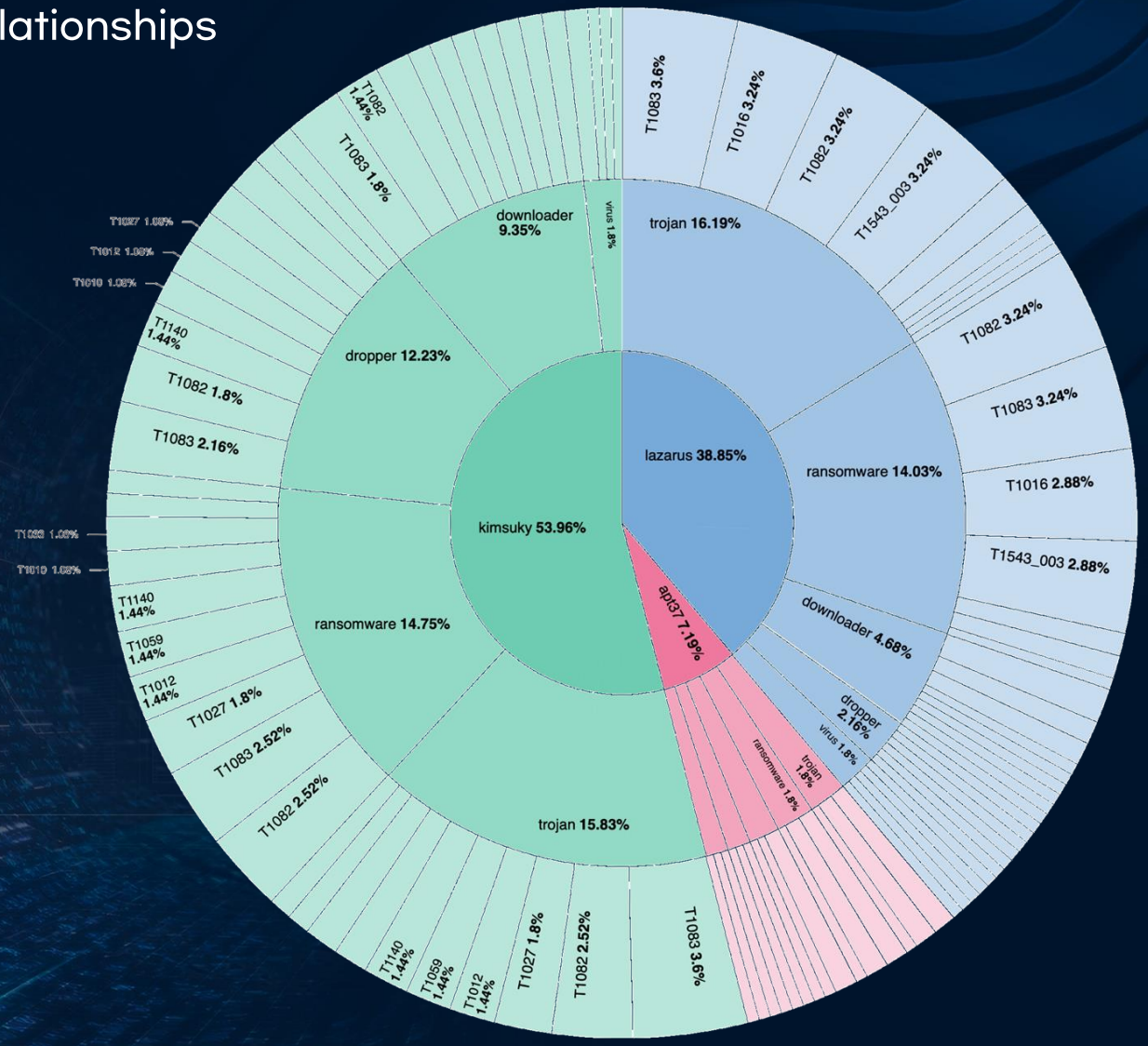
Top 20 Threat Actor of Tracked Malware



Case Study : Analyzing the Output of DBP

Furthermore, it is possible to analyze the interrelationships among Threat Actors, Threat Types, and TIDs

Threat Intelligence related to North Korea

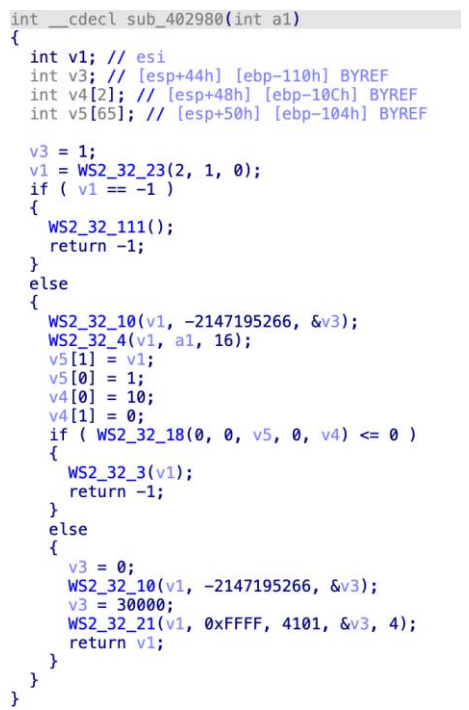


Case Study : Tracking Related Threats by Function Reuse

Utilizing DBP allows for tracking the reuse of these specific functions implemented by the Lazarus Group.

```
{
  HANDLE FileA; // eax
  void *v3; // esi
  HANDLE v5; // esi
  struct _FILETIME LastWriteTime; // [esp+Ch] [ebp-18h] BYREF
  struct _FILETIME LastAccessTime; // [esp+14h] [ebp-10h] BYREF
  struct _FILETIME CreationTime; // [esp+1Ch] [ebp-8h] BYREF

  if ( a1 )
    FileA = CreateFileA(a1, 0x80000000, 1u, 0, 3u, 0x2000000u, 0);
  else
    FileA = CreateFileA(FileName, 0x80000000, 1u, 0, 3u, 0x2000000u, 0);
  v3 = FileA;
  if ( FileA == (HANDLE)-1 )
    return 0;
  if ( GetFileTime(FileA, &CreationTime, &LastAccessTime, &LastWriteTime) )
  {
    CloseHandle(v3);
    v5 = CreateFileA(lpFileName, 0xC0000000, 3u, 0, 3u, 0x2000000u, 0);
    if ( v5 == (HANDLE)-1 )
    {
      return 0;
    }
    else
    {
      SetFileTime(v5, &CreationTime, &LastAccessTime, &LastWriteTime);
      CloseHandle(v5);
      return 1;
    }
  }
  else
  {
    CloseHandle(v3);
    return 0;
  }
}
```

```
result = OpenSCManagerA(0, 0, 2u);
v2 = result;
if ( result )
{
  ServiceA = CreateServiceA(
               result,
               ServiceName,
               DisplayName,
               0xF01FFu,
               0x110u,
               2u,
               1u,
               lpBinaryPathName,
               0,
               0,
               0,
               0,
               0);
  v4 = ServiceA;
  if ( ServiceA )
  {
    lpBinaryPathName = DisplayName;
    ChangeServiceConfig2A(ServiceA, 1u, &lpBinaryPathName);
    v8[1] = 60000;
    v8[3] = 60000;
    v8[0] = 1;
    v8[5] = 0;
    Info[1] = 0;
    Info[2] = 0;
    v8[2] = 1;
    v8[4] = 1;
    Info[0] = 86400;
    Info[3] = 3;
    Info[4] = (int)v8;
    ChangeServiceConfig2A(v4, 2u, Info);
    CloseServiceHandle(v4);
    v5 = OpenServiceA(v2, ServiceName, 0xF01FFu);
    v6 = v5;
    if ( v5 )
    {
      StartServiceA(v5, 0, 0);
      CloseServiceHandle(v6);
      CloseServiceHandle(v2);
      return 0;
    }
    else
    {
      CloseServiceHandle(v2);
      return (SC_HANDLE)1;
    }
  }
  else
  {
    CloseServiceHandle(v2);
    return 0;
  }
}
return result;
}
```

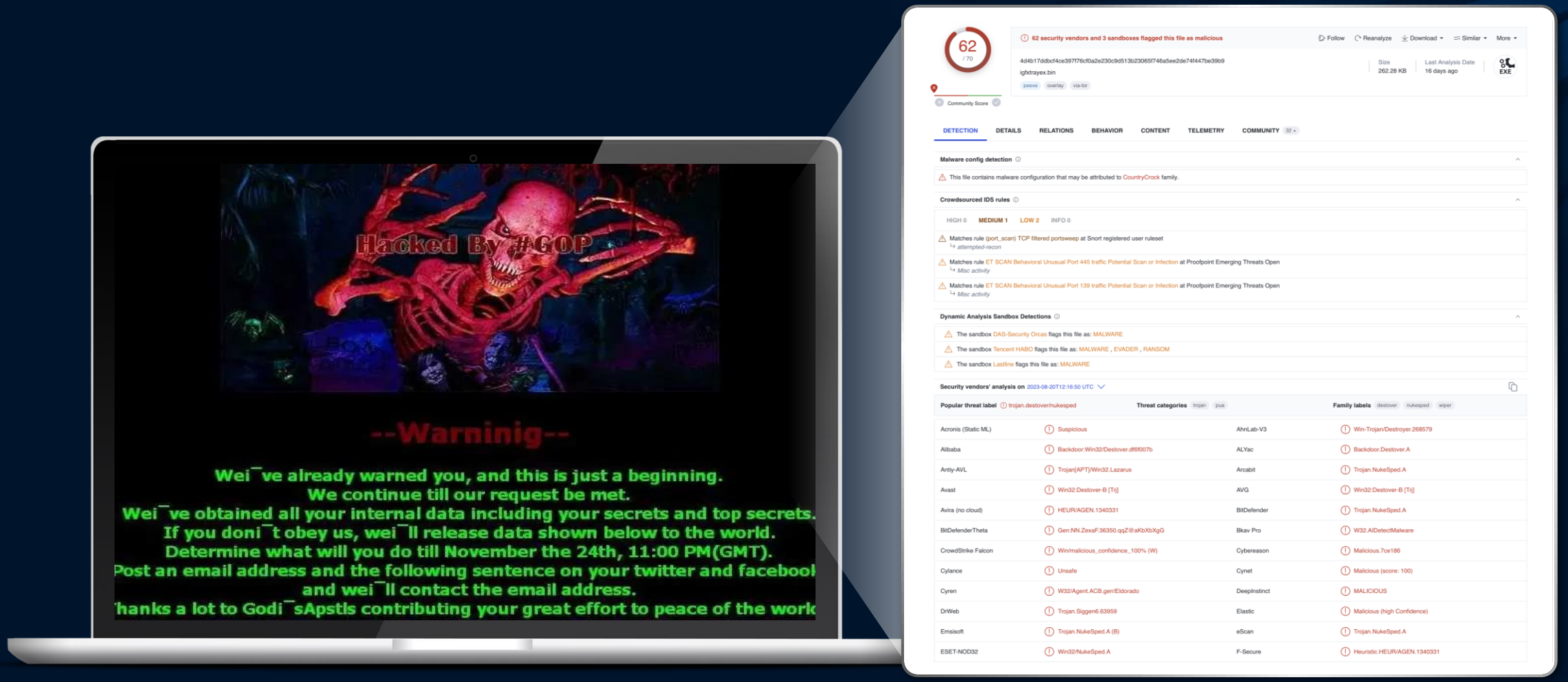


```
int __cdecl sub_402980(int a1)
{
  int v1; // esi
  int v3; // [esp+44h] [ebp-110h] BYREF
  int v4[2]; // [esp+48h] [ebp-10Ch] BYREF
  int v5[65]; // [esp+50h] [ebp-104h] BYREF

  v3 = 1;
  v1 = WS2_32_23(2, 1, 0);
  if ( v1 == -1 )
  {
    WS2_32_111();
    return -1;
  }
  else
  {
    WS2_32_10(v1, -2147195266, &v3);
    WS2_32_4(v1, a1, 16);
    v5[1] = v1;
    v5[0] = 1;
    v4[0] = 10;
    v4[1] = 0;
    if ( WS2_32_18(0, 0, v5, 0, v4) <= 0 )
    {
      WS2_32_3(v1);
      return -1;
    }
    else
    {
      v3 = 0;
      WS2_32_10(v1, -2147195266, &v3);
      v3 = 30000;
      WS2_32_21(v1, 0xFFFF, 4101, &v3, 4);
      return v1;
    }
  }
}
```

sub_4016E0	sub_403690	sub_402980
Lazarus	Lazarus	Lazarus
Dropper	Dropper	Dropper, Trojan
T1070.006 (Indicator Removal : Timestomp)	T1543.003 (Create or Modify System Process : Windows Service)	-

Case Study : Tracking Related Threats by Function Reuse

We discovered the Destover malware used in the Sony Pictures hack that occurred on November 24, 2014, from the December 2022 feed data



Case Study : Tracking Related Threats by Function Reuse

Announcing North Korea's Lazarus group attack on Sony Pictures Entertainment using Destover malware, based on the logic, encryption methods, and utilized IoC information of Previous North Korean malware

Nov 2014

Sony Pictures Entertainment attacked by a hacker group with the name GOP(Guardians of Peace)

Leakage of files related to latest and unreleased films, private conversations among employees

Dec 2014

The US FBI announced that the Sony Hacking was attributed to North Korea

Analyzing encryption methods, data deletion, and network communication related to North Korea

Dec 2014

Kaspersky, “Sony/Destover: mystery North Korean actor’s destructive and past network activity”

Analyzing the similarities in the Shamoon, DarkSeoul and Sony hacks

Dec 2016

Kaspersky Daily, “What is known about the Lazarus Group: Sony hack, military espionage, attacks on Korean banks and other crimes”

The attackers reused segments of different malware to implement other malicious code

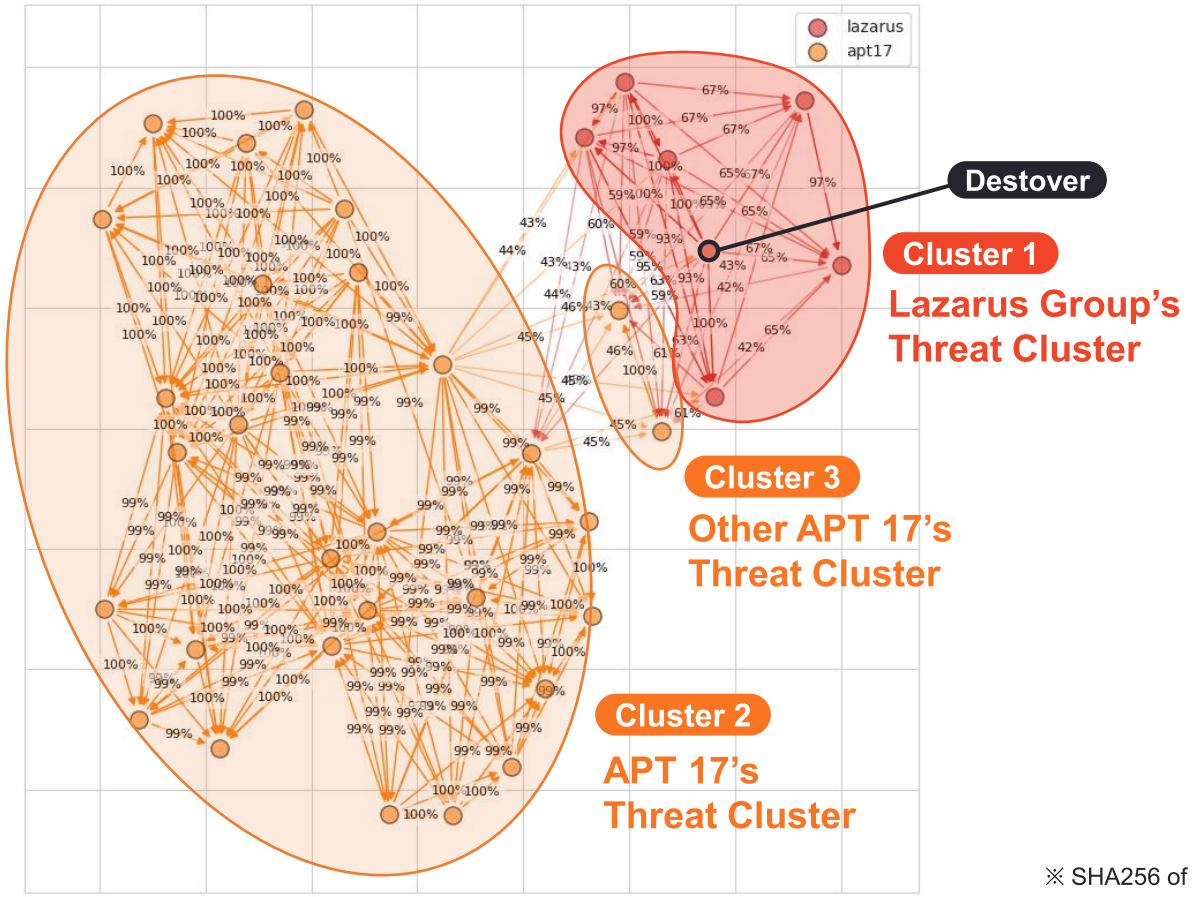
Sep 2018

The US government prosecutes North Korean Lazarus Group hacker, Jin-Hyuk Park

Case Study : Tracking Related Threats by Function Reuse

Tracking variant threats based on the Destover malware

The Relation Graph of Variant Sample



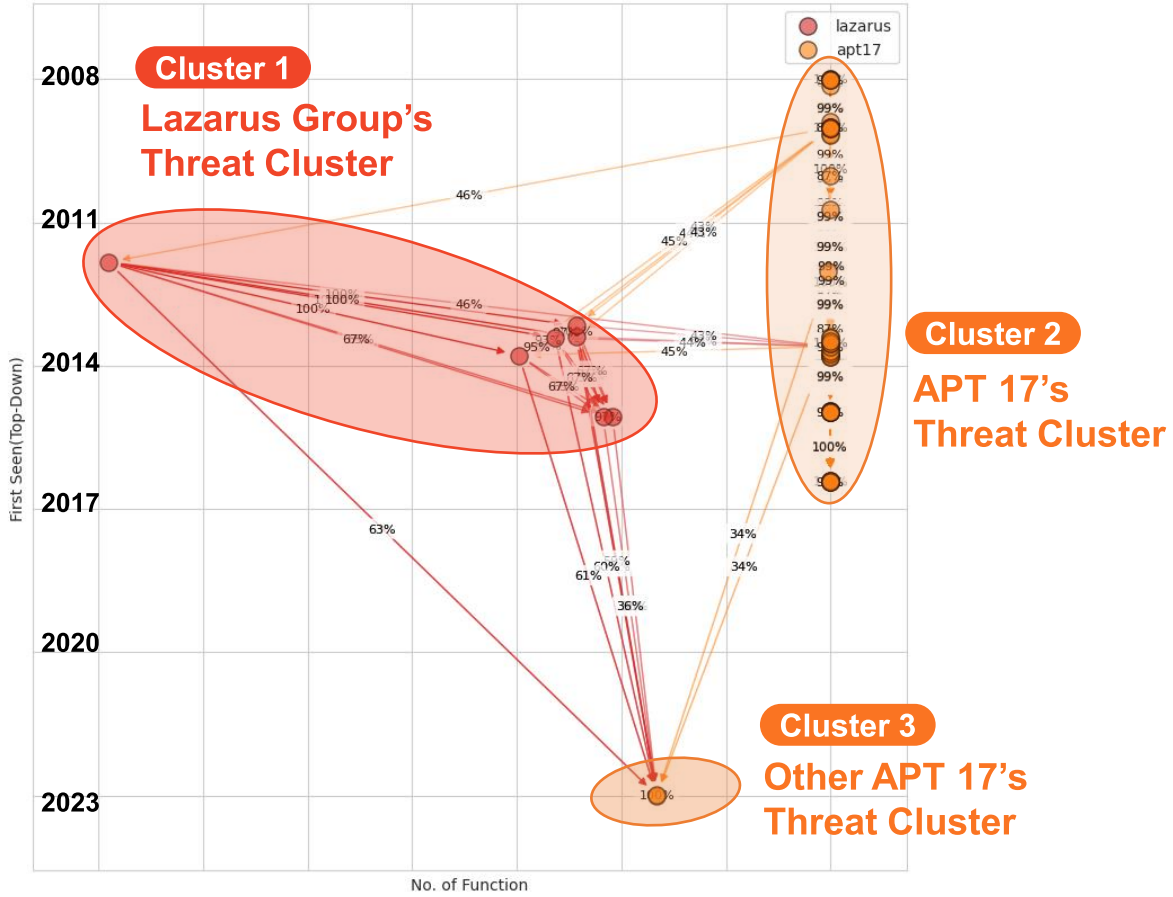
Continuously tracking malware with over 40% function reuse from each predecessor, starting with the base sample, the Destover malware.

※ SHA256 of Destover sample : 4d4b17ddbcf4ce397f76cf0a2e230c9d513b23065f746a5ee2de74f447be39b9

Case Study : Tracking Related Threats by Function Reuse

Organizing nodes based on the first-seen date and the quantity of functions in the malware

The Relation Graph of Variant Sample



Cluster 1 **Lazarus Group's Threat Cluster (Destover Malware)**

4d4b17ddbcf4ce397f76cf0a2e230c9d513b23065f746a5ee2de74f447be39b9

201a9c5fe6a8ae0d1c4312d07ef2066e5991b1462b68f102154bb9cb25bf59f9

✕

Cluster 2 **APT17's Threat Cluster (zxproxy Malware)**

0df665f53136ffabf905ee9cda0f33296c8fa12ef85d56624bb37af152c61775

and 30 Others

=

Cluster 3 **Other APT 17's Threat Cluster (Trojan Malware)**

50b5d3c56af17568ef22e5c97ec52b257b41fd040f4ee8d24a334835ac09e45c

468b395bd9f97eaebdb8f07ce114c1b19a9d583d4bfd5458bd9bcd4325cb850f

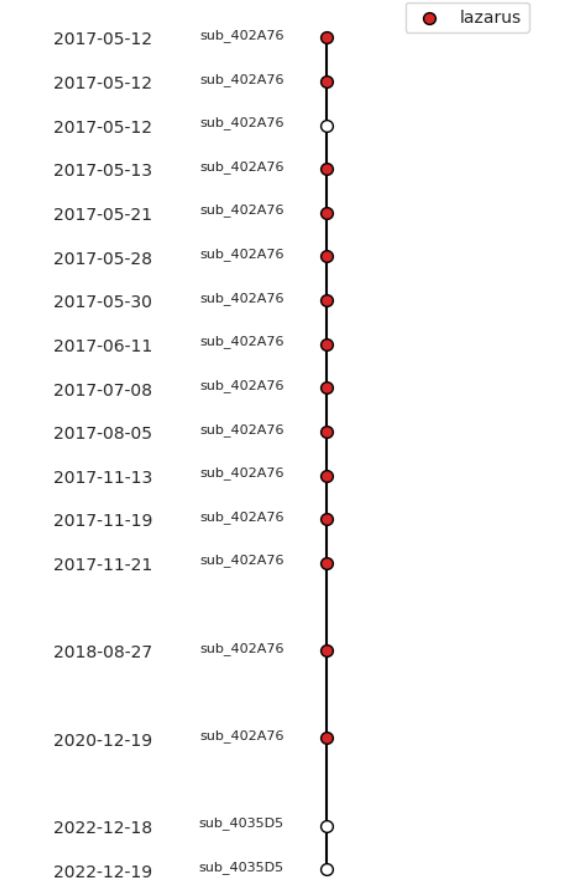
Which function is reused?

Received 12 functions from Lazarus and 22 functions from APT17

Case Study : Tracking Related Threats by Function Reuse

A sample in Cluster 3 reused a function commonly employed by Lazarus Group (Cluster 1)

Function Lifetime



sub_402A76 in Cluster 1

```
if ( !a2 )
{
  Src = (int)&unk_40F57C;
  exception::exception((exception *)pExceptionObject, (const char *const *)&Src);
  CxxThrowException(pExceptionObject, (_ThrowInfo *)&pThrowInfo);
}
v6 = a4;
if ( a4 != 16 && a4 != 24 && a4 != 32 )
{
  Src = (int)&unk_40F57C;
  exception::exception((exception *)pExceptionObject, (const char *const *)&Src);
  CxxThrowException(pExceptionObject, (_ThrowInfo *)&pThrowInfo);
}
if ( Size != 16 && Size != 24 && Size != 32 )
{
  Src = (int)&unk_40F57C;
  exception::exception((exception *)pExceptionObject, (const char *const *)&Src);
  CxxThrowException(pExceptionObject, (_ThrowInfo *)&pThrowInfo);
}
v45 = Size;
*(_DWORD *)(this + 972) = Size;
v44 = (const void *)Src;
*(_DWORD *)(this + 968) = v6;
memcpy((void *)(this + 976), v44, v45);
memcpy((void *)(this + 1008), (const void *)Src, *(_DWORD *)(this + 972));
v7 = *(_DWORD *)(this + 968);
if ( v7 == 16 )
{
  v10 = *(_DWORD *)(this + 972);
  if ( v10 == 16 )
    v9 = 10;
  else
    v9 = v10 != 24 ? 14 : 12;
}
else
{
  if ( v7 != 24 )
  {
    *(_DWORD *)(this + 1040) = 14;
    goto LABEL_19;
  }
  v8 = (*(_DWORD *)(this + 972) == 32) - 1;
  LOBYTE(v8) = v8 & 0xFE;
  v9 = v8 + 14;
}
*(_DWORD *)(this + 1040) = v9;
LABEL_19:
v11 = 0;
v12 = *(int *)(this + 1040) < 0;
a4 = *(_DWORD *)(this + 972) / 4;
if ( !v12 )
{
  v13 = (char *)(this + 8);
  do
  {
    if ( a4 > 0 )
      memset(v13, 0, 4 * a4);
    ++v11;
    v13 += 32;
  }
  while ( v11 <= *(_DWORD *)(this + 1040) );
}
v14 = 0;
if ( *(int *)(this + 1040) >= 0 )
{
  v15 = (char *)(this + 488);
  do
  {
    if ( a4 > 0 )
      memset(v15, 0, 4 * a4);
    ++v14;
    v15 += 32;
  }
  while ( v14 <= *(_DWORD *)(this + 1040) );
}
```

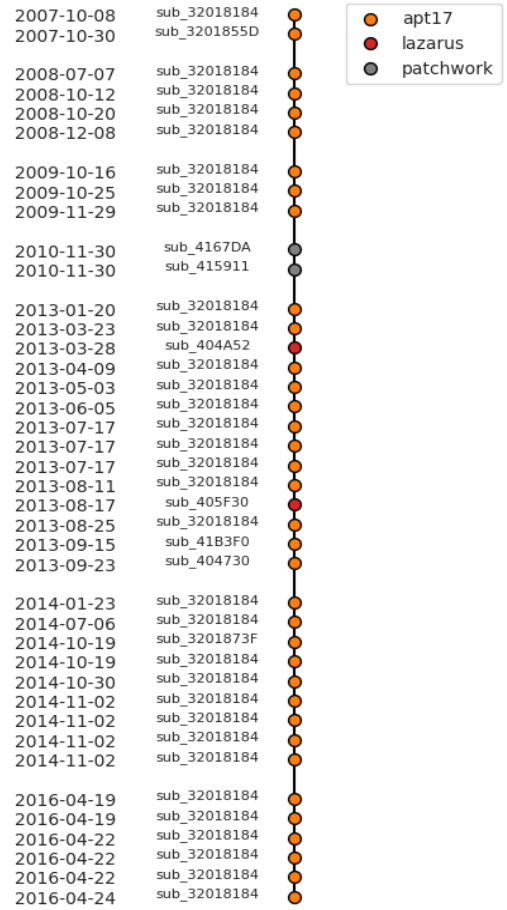
sub_4035D5 in Cluster 3

```
if ( !a2 )
{
  Src = aEmptyKey;
  exception::exception((exception *)pExceptionObject, (const char *const *)&Src);
  _CxxThrowException(pExceptionObject, (_ThrowInfo *)&pThrowInfo);
}
v6 = a4;
if ( a4 != 16 && a4 != 24 && a4 != 32 )
{
  Src = aIncorrectKeyLe;
  exception::exception((exception *)pExceptionObject, (const char *const *)&Src);
  _CxxThrowException(pExceptionObject, (_ThrowInfo *)&pThrowInfo);
}
if ( Size != 16 && Size != 24 && Size != 32 )
{
  Src = aIncorrectBlock;
  exception::exception((exception *)pExceptionObject, (const char *const *)&Src);
  _CxxThrowException(pExceptionObject, (_ThrowInfo *)&pThrowInfo);
}
v45 = Size;
*(_DWORD *)(this + 972) = Size;
v44 = Src;
*(_DWORD *)(this + 968) = v6;
memcpy((void *)(this + 980), v44, v45);
memcpy((void *)(this + 1012), Src, *(_DWORD *)(this + 972));
v7 = *(_DWORD *)(this + 968);
if ( v7 == 16 )
{
  v10 = *(_DWORD *)(this + 972);
  if ( v10 == 16 )
    v9 = 10;
  else
    v9 = v10 != 24 ? 14 : 12;
}
else
{
  if ( v7 != 24 )
  {
    *(_DWORD *)(this + 976) = 14;
    goto LABEL_19;
  }
  v8 = (*(_DWORD *)(this + 972) == 32) - 1;
  LOBYTE(v8) = v8 & 0xFE;
  v9 = v8 + 14;
}
*(_DWORD *)(this + 976) = v9;
LABEL_19:
v11 = 0;
v12 = *(int *)(this + 976) < 0;
a4 = *(_DWORD *)(this + 972) / 4;
if ( !v12 )
{
  v13 = (char *)(this + 8);
  do
  {
    if ( a4 > 0 )
      memset(v13, 0, 4 * a4);
    ++v11;
    v13 += 32;
  }
  while ( v11 <= *(_DWORD *)(this + 976) );
}
v14 = 0;
if ( *(int *)(this + 976) >= 0 )
{
  v15 = (char *)(this + 488);
  do
  {
    if ( a4 > 0 )
      memset(v15, 0, 4 * a4);
    ++v14;
    v15 += 32;
  }
  while ( v14 <= *(_DWORD *)(this + 976) );
}
```

Case Study : Tracking Related Threats by Function Reuse

This sample reused a function commonly employed by APT17 (Cluster 2)

Function Lifetime



sub_32018184 in Cluster 2

```
v1 = *(_DWORD *)(a1 + 5812);
if ( v1 <= 13 )
{
  *(_WORD *)(a1 + 5808) |= 2 << v1;
  *(_DWORD *)(a1 + 5812) = v1 + 3;
}
else
{
  v2 = 2 << v1;
  v3 = *(_DWORD *)(a1 + 20);
  *(_WORD *)(a1 + 5808) |= v2;
  *(_BYTE *)(*(_DWORD *)(a1 + 8) + v3) = *(_BYTE *)(a1 + 5808);
  ++*(_DWORD *)(a1 + 20);
  *(_BYTE *)(*(_DWORD *)(a1 + 8) + *(_DWORD *)(a1 + 20)) = *(_BYTE *)(a1 + 5809);
  v4 = *(_DWORD *)(a1 + 5812);
  ++*(_DWORD *)(a1 + 20);
  *(_DWORD *)(a1 + 5812) = v4 - 13;
  *(_WORD *)(a1 + 5808) = 2u >> (16 - v4);
}
v5 = *(_DWORD *)(a1 + 5812);
if ( v5 <= 9 )
{
  *(_WORD *)(a1 + 5808) |= 458752 << v5;
  *(_DWORD *)(a1 + 5812) = v5 + 7;
}
else
{
  v6 = *(_DWORD *)(a1 + 8);
  *(_WORD *)(a1 + 5808) = *(_WORD *)(a1 + 5808);
  *(_BYTE *)(v6 + (*(_DWORD *)(a1 + 20))++) = *(_BYTE *)(a1 + 5808);
  *(_BYTE *)(*(_DWORD *)(a1 + 8) + *(_DWORD *)(a1 + 20)) = *(_BYTE *)(a1 + 5809);
  v7 = *(_DWORD *)(a1 + 5812);
  ++*(_DWORD *)(a1 + 20);
  *(_WORD *)(a1 + 5808) = 0;
  *(_DWORD *)(a1 + 5812) = v7 - 9;
}
sub_32019769(a1);
v8 = *(_DWORD *)(a1 + 5812);
result = *(_DWORD *)(a1 + 5804) - v8 + 11;
if ( result < 9 )
{
  if ( v8 <= 13 )
  {
    *(_WORD *)(a1 + 5808) |= 2 << v8;
    *(_DWORD *)(a1 + 5812) = v8 + 3;
  }
  else
  {
    v10 = 2 << v8;
    v11 = *(_DWORD *)(a1 + 20);
    *(_WORD *)(a1 + 5808) |= v10;
    *(_BYTE *)(*(_DWORD *)(a1 + 8) + v11) = *(_BYTE *)(a1 + 5808);
    ++*(_DWORD *)(a1 + 20);
    *(_BYTE *)(*(_DWORD *)(a1 + 8) + *(_DWORD *)(a1 + 20)) = *(_BYTE *)(a1 + 5809);
    v12 = *(_DWORD *)(a1 + 5812);
    ++*(_DWORD *)(a1 + 20);
    *(_DWORD *)(a1 + 5812) = v12 - 13;
    *(_WORD *)(a1 + 5808) = 2u >> (16 - v12);
  }
  v13 = *(_DWORD *)(a1 + 5812);
  if ( v13 <= 9 )
  {
    *(_WORD *)(a1 + 5808) |= 458752 << v13;
    *(_DWORD *)(a1 + 5812) = v13 + 7;
  }
  else
  {
    v14 = *(_DWORD *)(a1 + 8);
    *(_WORD *)(a1 + 5808) = *(_WORD *)(a1 + 5808);
    *(_BYTE *)(v14 + (*(_DWORD *)(a1 + 20))++) = *(_BYTE *)(a1 + 5808);
    *(_BYTE *)(*(_DWORD *)(a1 + 8) + *(_DWORD *)(a1 + 20)) = *(_BYTE *)(a1 + 5809);
    v15 = *(_DWORD *)(a1 + 5812);
    ++*(_DWORD *)(a1 + 20);
    *(_WORD *)(a1 + 5808) = 0;
    *(_DWORD *)(a1 + 5812) = v15 - 9;
  }
  result = sub_32019769(a1);
}
*(_DWORD *)(a1 + 5804) = 7;
return result;
```

=

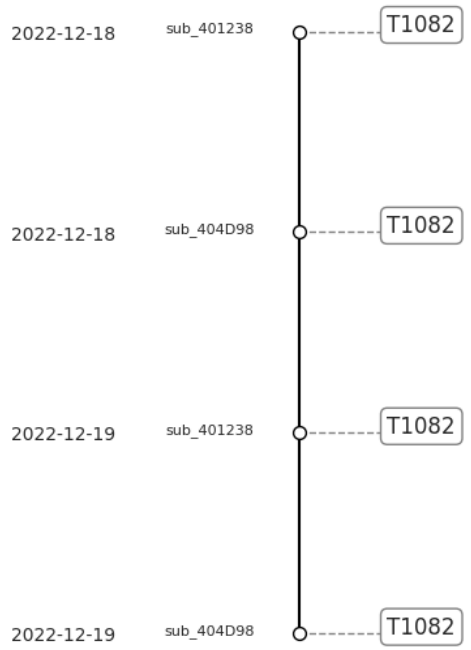
sub_40973C in Cluster 3

```
v1 = *(_DWORD *)(a1 + 5820);
if ( v1 <= 13 )
{
  *(_WORD *)(a1 + 5816) |= 2 << v1;
  *(_DWORD *)(a1 + 5820) = v1 + 3;
}
else
{
  v2 = 2 << v1;
  v3 = *(_DWORD *)(a1 + 20);
  *(_WORD *)(a1 + 5816) |= v2;
  *(_BYTE *)(*(_DWORD *)(a1 + 8) + v3) = *(_BYTE *)(a1 + 5816);
  ++*(_DWORD *)(a1 + 20);
  *(_BYTE *)(*(_DWORD *)(a1 + 8) + *(_DWORD *)(a1 + 20)) = *(_BYTE *)(a1 + 5817);
  v4 = *(_DWORD *)(a1 + 5820);
  ++*(_DWORD *)(a1 + 20);
  *(_DWORD *)(a1 + 5820) = v4 - 13;
  *(_WORD *)(a1 + 5816) = 2u >> (16 - v4);
}
v5 = *(_DWORD *)(a1 + 5820);
if ( v5 <= 9 )
{
  *(_WORD *)(a1 + 5816) |= 458752 << v5;
  *(_DWORD *)(a1 + 5820) = v5 + 7;
}
else
{
  v6 = *(_DWORD *)(a1 + 8);
  *(_WORD *)(a1 + 5816) = *(_WORD *)(a1 + 5816);
  *(_BYTE *)(v6 + (*(_DWORD *)(a1 + 20))++) = *(_BYTE *)(a1 + 5816);
  *(_BYTE *)(*(_DWORD *)(a1 + 8) + *(_DWORD *)(a1 + 20)) = *(_BYTE *)(a1 + 5817);
  v7 = *(_DWORD *)(a1 + 5820);
  ++*(_DWORD *)(a1 + 20);
  *(_WORD *)(a1 + 5816) = 0;
  *(_DWORD *)(a1 + 5820) = v7 - 9;
}
sub_40AD4D(a1);
v8 = *(_DWORD *)(a1 + 5820);
result = *(_DWORD *)(a1 + 5812) - v8 + 11;
if ( result < 9 )
{
  if ( v8 <= 13 )
  {
    *(_WORD *)(a1 + 5816) |= 2 << v8;
    *(_DWORD *)(a1 + 5820) = v8 + 3;
  }
  else
  {
    v10 = 2 << v8;
    v11 = *(_DWORD *)(a1 + 20);
    *(_WORD *)(a1 + 5816) |= v10;
    *(_BYTE *)(*(_DWORD *)(a1 + 8) + v11) = *(_BYTE *)(a1 + 5816);
    ++*(_DWORD *)(a1 + 20);
    *(_BYTE *)(*(_DWORD *)(a1 + 8) + *(_DWORD *)(a1 + 20)) = *(_BYTE *)(a1 + 5817);
    v12 = *(_DWORD *)(a1 + 5820);
    ++*(_DWORD *)(a1 + 20);
    *(_DWORD *)(a1 + 5820) = v12 - 13;
    *(_WORD *)(a1 + 5816) = 2u >> (16 - v12);
  }
  v13 = *(_DWORD *)(a1 + 5820);
  if ( v13 <= 9 )
  {
    *(_WORD *)(a1 + 5816) |= 458752 << v13;
    *(_DWORD *)(a1 + 5820) = v13 + 7;
  }
  else
  {
    v14 = *(_DWORD *)(a1 + 8);
    *(_WORD *)(a1 + 5816) = *(_WORD *)(a1 + 5816);
    *(_BYTE *)(v14 + (*(_DWORD *)(a1 + 20))++) = *(_BYTE *)(a1 + 5816);
    *(_BYTE *)(*(_DWORD *)(a1 + 8) + *(_DWORD *)(a1 + 20)) = *(_BYTE *)(a1 + 5817);
    v15 = *(_DWORD *)(a1 + 5820);
    ++*(_DWORD *)(a1 + 20);
    *(_WORD *)(a1 + 5816) = 0;
    *(_DWORD *)(a1 + 5820) = v15 - 9;
  }
  result = sub_40AD4D(a1);
}
*(_DWORD *)(a1 + 5812) = 7;
return result;
```

Case Study : Tracking Related Threats by Function Reuse

Additionally, this sample reused a function related with the T1082 attack technique, and this function is not related to any specific threat actor.

Function Lifetime



sub_402A76 in Cluster1

```
int __cdecl sub_404D98(wchar_t *Source, wchar_t *Destination, size_t Count)
{
  int v4; // esi
  const wchar_t *v5; // eax
  wchar_t *v6; // eax
  wchar_t *v7; // eax
  const WCHAR *v8; // edi
  wchar_t *v9; // ebx
  wchar_t v10[260]; // [esp+Ch] [ebp-618h] BYREF
  WCHAR Buffer[260]; // [esp+214h] [ebp-410h] BYREF
  wchar_t Format[260]; // [esp+41Ch] [ebp-208h] BYREF

  memset(Format, 0, sizeof(Format));
  memset(Buffer, 0, sizeof(Buffer));
  memset(v10, 0, sizeof(v10));
  if ( wcschr(Source, 0x25u) )
  {
    v4 = 0;
    v5 = Source;
    while ( 1 )
    {
      v6 = wcschr(v5, 0x25u);
      if ( !v6 )
        break;
      v5 = v6 + 1;
      ++v4;
    }
    if ( v4 % 2 != 1
      && (wcsncpy(Format, Source, 0x104u),
          v7 = wcschr(Format, 0x25u),
          v8 = v7 + 1,
          *v7 = 0,
          v9 = wcschr(v7 + 1, 0x25u),
          *v9 = 0,
          GetEnvironmentVariableW(v8, Buffer, 0x104u)) )
    {
      swprintf(v10, (const size_t)aSSS, Format, Buffer, v9 + 1);
      return sub_404D98(v10, Destination, Count);
    }
    else
    {
      return 0;
    }
  }
  else
  {
    wcsncpy(Destination, Source, Count);
    return 1;
  }
}
```

=

sub_4035D5 in Cluster3

```
int __cdecl sub_401238(wchar_t *Source, wchar_t *Destination, size_t Count)
{
  int v3; // esi
  const wchar_t *v5; // eax
  wchar_t *v6; // eax
  wchar_t *v7; // eax
  const WCHAR *v8; // edi
  wchar_t *v9; // ebx
  wchar_t v10[260]; // [esp+Ch] [ebp-618h] BYREF
  WCHAR Buffer[260]; // [esp+214h] [ebp-410h] BYREF
  wchar_t Format[260]; // [esp+41Ch] [ebp-208h] BYREF

  v3 = 0;
  memset(Format, 0, sizeof(Format));
  memset(Buffer, 0, sizeof(Buffer));
  memset(v10, 0, sizeof(v10));
  if ( !Source )
    return 0;
  if ( !wcschr(Source, 0x25u) )
  {
    wcsncpy(Destination, Source, Count);
    return 1;
  }
  v5 = Source;
  while ( 1 )
  {
    v6 = wcschr(v5, 0x25u);
    if ( !v6 )
      break;
    v5 = v6 + 1;
    ++v3;
  }
  if ( v3 % 2 == 1 )
    return 0;
  wcsncpy(Format, Source, 0x104u);
  v7 = wcschr(Format, 0x25u);
  *v7 = 0;
  v8 = v7 + 1;
  v9 = wcschr(v7 + 1, 0x25u);
  *v9 = 0;
  if ( !GetEnvironmentVariableW(v8, Buffer, 0x104u) )
    return 0;
  swprintf(v10, (const size_t)aSSS, Format, Buffer, v9 + 1);
  return sub_401238(v10, Destination, Count);
}
```

4

Summary

Summary

Limitations of Manual Analysis

High-level Requirements

Recurrence of Variant Threats

Subjectivity in Analysis

Automated TI Analysis

Analysis Assistance

Distinguishing New/Variant

Assembly Code as Evidence

Q&A

SANDS Lab

Hyunjong Lee hjlee@sandslab.io
ChangGyun Kim cgkim@ksign.com