Threat Intelligence on Steroids

Exploits, Ransomware And Other Threats At Scale

About Me

Sarthak Misraa

Background

- Staff Threat Intelligence Researcher @ SentinelOne
- Reverse Engineer And Software Developer
- {twitter, discord, github}:= n0tduck1e

Flow Of The Presentation

- ➔ Identify the problem
- ➔ Establish a framework for performing technical threat intelligence at scale.
- ➔ Harbinger: Showcase the automation developed to enable large-scale threat analysis
- ➔ Accurately attribute techniques by analyzing techniques and obfuscation methods
- ➔ Leverage this intelligence to identify and track emerging trends in threats at scale

Motivation

- Monitor the evolution of techniques used in malware samples on a large scale.
- Avoid the need to reverse-engineer every sample that appears.
- Hunt for both packed and unpacked samples using the same rules.
- Focus on attributing techniques rather than just the artifacts created by the samples.

Harbinger: Threat Intelligence At Scale

- Harbinger is a suite of tools and automations developed to efficiently attribute and hunt on large corpses of files.
- This suite facilitates in-depth analysis and collection of execution artifacts, allowing researchers to search through the results, identify emerging trends and common technique patterns.
- By streamlining the process of file analysis and attribution, Harbinger enhances the abilities to perform threat intelligence at larger scale.

Key Components

Dr Plague: Sandbox	Rule Engine	Artifact Library

Dr Plague: Collecting Execution Artifacts

Dr Plague is simple generic unpacker and in-memory scanner. It provides the following features

- Dumps Syscalls And WinAPI
- Unpacks and dump memory regions
- Dump process injection payloads
- List of loaded modules (static and dynamic)
- Some anti-anti analysis tricks

- Forked Version CapeMon
    - Previously Used Intel Pin**
    - Before that Dynamio Rio**

Rule Engine

```
const char* device_handle_fail(const syscall_t* syscall_list,
                               size_t syscall_list_size) {
    for(size_t iter=0x0; iter<syscall_list_size; iter++) {
        if (syscall_list[iter].syscall_number == sys_NtDeviceIoControlFile
                && syscall_list[iter].return_value != 0x0)
        {
            return g_file_handle_list[sprintf(a_allocated_string_mem, "%d",
syscall_list[iter].args[0])]
        }
    }
    return "";
}
```

Rule Set Groups: Hunting, Attribution, Detection

Develop rules that are not affected by packers.

Unpacked memory dumps and injected payloads.

Yara In Post Processing And Hunting

C Rules In runtime

Sample Ingestion Life Cycle

Continuous Sample Ingestion	Dentonation	Rule Engine Processing	Searchable Artifacts
Integration With OSINT Feeds, LiveHunts or Other Samples	In-house tools such Dr Plague, to label and collect information from sample dynamically	Running Rules On Pre And Post Processed Artifacts	Searchable Library Of Collected Artifacts

Demo 1

Evolution Of CLFS Based Exploits

Use Case 1: Evolution Of CLFS Exploits

Since 2019, there have been ten vulnerabilities related to the Common Log File System (CLFS) that are actively being exploited in the wild by threat actors and ransomware groups according to CISA catalogue.

Let's Consider the triple combo of vulnerabilities patched in 2025:

CVE	Exploit Type	Threat Cluster/Group	Precursor to Ransomware	POC
CVE-2023-28252	Elevation Of Privilege	Nokoyawa	✅	✅
CVE-2022-37969	Elevation Of Privilege	Bian Lian	✅	✅

Great Blog:
https://securelist.com/windows-clfs-exploits-ransomware/111560/

CLFS Exploit: General Rule For CLFS

```
rule clfs_cve_general {
    meta:
        tags="clfs,cve_2023_28252"
        verdict="clfs_exploit"
    strings:
        $p0 = /\[NtOpenFile\(processhandle=0x...,desiredaccess=0x100021,objectattributes=.?.?.?\\C.?\
\Windows\\System32\\drivers\\CLFS\.SYS/ nocase
        $p1 = /\[NtCreateFile\(FileHandle=0x...,DesiredAccess=0x[^,]+,ObjectAttributes=(\\GLOBAL\?\?\
\LOG:\\\?\?\\|\\\?\?\\)C:\\Users\\Public/ nocase
    condition:
        $p0 and #p1 > 10
}
```

CLFS Exploit: Specific To CVE-2022-37969

```
rule clfs_cve_2022_37969 {
    meta:
        tags="cve_2022_37969"
        verdict="clfs_exploit"
    strings:
        $p0 = /\[NtOpenFile\(processhandle=0x...,desiredaccess=0x100021,objectattributes=.?.?.?\\C.?\
\Windows\\System32\\drivers\\CLFS\.SYS/ nocase
        $p1 = /\[NtOpenFile\(processhandle=0x...,desiredaccess=0x[^,]+,objectattributes=\\Device\
\NamedPipe\\/
        $p2 = /\[NtFsControlFile[^\n]+\n/
        $p3 = /\[NtQuerySystemInformation\(SystemInformationClass=0x42[^\n]+\n/
        $p4 = /\[NtAllocateVirtualMemory\([^\n]+\n/
        $p5 = /\[NtQuerySystemInformation\(SystemInformationClass=0x42[^\n]+\n/
    condition:
        $p0
        and
        $p1
        and
        for any i in (1..#p2) : (
            $p3 at (@p2[i] + !p2[i])
            and
            for any j in (1..#p3) : (
                $p4 at (@p3[j] + !p3[j])
                and
                for any k in (1..#p4) : (
                    $p5 at (@p4[k] + !p4[k])
                )
            )
        )
}
```

exploit explained:
https://www.coresecurity.com/core-labs/articles/understanding-cve-2022-37969-windows-clfs-lpe

Exploit: Specific CVE-2022-37969 But in C

```
bool cve_2022_37969(const syscall_t* syscall_list, size_t syscall_list_size){
    for(size_t iter=0x0; iter<syscall_list_size; iter++){
        if (syscall_list[iter].syscall_number == sys_NtOpenFile && strcmp(syscall_list[iter].args[2],"\
\Device\\NamedPipe\\") && iter+8 < syscall_list_size ) {
            if (syscall_list[iter].syscall_number == sys_NtQuerySystemInformation &&
syscall_list[iter].args[0] == 0x42
                && syscall_list[iter+3].syscall_number == sys_NtAllocateVirtualMemory
                && syscall_list[iter+4].syscall_number == sys_NtQuerySystemInformation &&
syscall_list[iter].args[0] == 0x42
                && syscall_list[iter+5].syscall_number == sys_NtWriteFile
                ){
                return true;
            }
        }
    }
    return false;
}
```

CLFS Exploit: Hunting CVE-2023-28252

```
rule clfs_cve_2023 {
    meta:
        tags="clfs,cve_2023_28252"
        verdict="clfs_exploit"
    strings:
        $p0 = /\[NtOpenFile\(processhandle=0x...,desiredaccess=0x100021,objectattributes=.?.?.?\\C.?\
\Windows\\System32\\drivers\\CLFS\.SYS/ nocase
        $p1 = /\[NtCreateFile\(FileHandle=0x...,DesiredAccess=0xc0100080,ObjectAttributes=[^.,]+\.blf,/
    condition:
        $p0
        and
        #p1 > 10
}
```

exploit poc: https://www.coresecurity.com/core-labs/articles/analysis-cve-2023-28252-clfs-vulnerability

Demo 2

Medusa Ransomware And Its Tooling

Ransomware: Medusa Ransomware And Its Tooling

The Medusa ransomware is reportedly operated as a ransomware-as-a-service (RaaS).

Similar to the majority of ransomware operators, It does double extortion.

Medusa affiliates are creative and employ a myriad of different tooling.

Attackers using Medusa often use the Bring Your Own Vulnerable Driver (BYOVD) technique in attacks.

Medusa : Encryptor

Various methods are employed to deploy the locker.

Different versions of the locker feature slightly different code variations.

The chain associated with the locker is often preceded by BrC4, BYVOD, and/or general tricks for disablement of antivirus

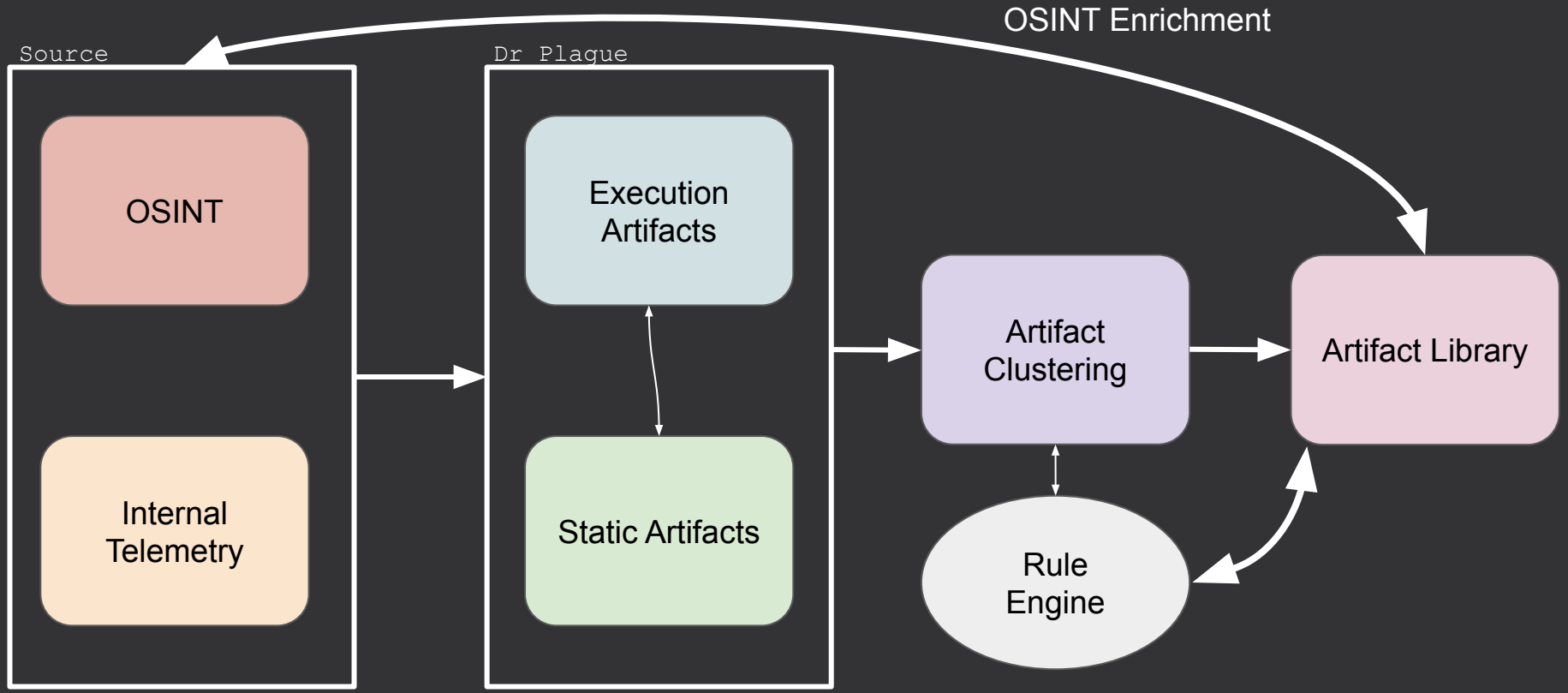
```
Usage:
        ThreadRansomExecuter.exe[-i in_path][-l locker_path][-d directory_depth][-t thread_count]
Options :
        - l locker_path       Locker file path
        - i in_path           Encrypt <in_path> folder.
        - d directory_depth   Directory depth limit for threads, default = 0
        - t thread_count      Thread Count, default = 50
        - f skip sys folders Skip System default Folders , default = false
PS C:\Users\Public\Documents>
```

Tag Rule: Failed Device Handle

```
const char* device_handle_fail(const syscall_t* syscall_list, size_t syscall_list_size){
    for(size_t iter=0x0; iter<syscall_list_size; iter++){
        if (syscall_list[iter].syscall_number == sys_NtDeviceIoControlFile &&
syscall_list[iter].return_value != 0x0) {
            return g_file_handle_list[sprintf(a_allocated_string_mem, "%d",
syscall_list[iter].args[0])]
        }
    }
    return false;
}
```

Sounds Good, But Does it Scale ?

Continuous Monitoring WorkFlow



Continuous Monitoring Of Threat Evolution

- Continuously collect Samples from Open Source Intelligence (OSINT) sources and detonate.

- Analyzing a large volume of samples efficiently and thoroughly to :-
  - Cluster similar artifacts to streamline the triage process.
  - Identify emerging trends in malware behavior and characteristics.

- Searchable library for the collected artifacts to facilitate easy access and analysis.
  - Postgresql For Syscalls, Winapi, Process Trees and Hunt Results
  - FileSystem blobs For Other Things

Benefits Of Using this Approach

Internal Telemetry And OSINT Sources

Source	Rich Execution Artifacts	Monitoring Rule Logic Flexibility	Clustering/ Attribution	Proactivity	Ease Of Monitoring
Internal Telemetry	✅ *	✅	✅	➖	✅
OSINT Sources	❌	➖	❌	✅	➖
Harbinger	✅	✅	✅	➖	✅

Legend

✅ : Supported
❌ : Limited/No Support
➖ : Partial Support

Limitations And Future Of The Project

- Add interface for CAPA
- UI leaves a lot to be desired (._.')
- Enhance network capture capabilities and implement similar hunting rules for network fingerprinting.
- Revise the codebase and initiate the process of open-sourcing key components.
- Host a limited-access version online to facilitate sample submissions.
- Upgrade the API interface for better usability and functionality.

Conclusions

Harbinger Rules can reveal interesting insights when done right

Writing attribution rules in c opens a new vector for monitoring evolution of techniques

Threat Intelligence with greater emphasis on the technique employed rather than solely on the epiphenomenon

A sample may be not work, yet it can still provide valuable insights from an intelligence perspective.

Thanks For Attending The Talk

QnA

The Harbinger

Reports Hunt Running Jobs

Detonation Reports

Date	Sample UUID	Sha1 Hash	Status	Verdict	Tags
Sun May 18 17:49:56 IST 2025	0df83b3e-29fb-412d-adba-98fdf4606570	2d40eceb6cc5d26f100ae68de91b7f1ae1e9f1df	Completed	byovd	byovd packed IoControlCode:0x9876c004|0x9876c094 Device:Blackout
Sun May 18 11:55:13 IST 2025	32e4f6ce-3d96-40ae-9dd3-4a724c1432cb	ee4575cf9818636781677d63236d3dc65652deab	Completed	ransomware	ransomware medusa packed
Sun May 18 17:26:54 IST 2025	3da3b6d2-7d55-457e-bfab-99a133a363ed	f4df2125a3453e7f996a8d3870864d02ad751dbd	Completed	byovd	byovd IoControlCode:0x9876c004|0x9876c094 Device:Blackout
Sun May 18 18:03:12 IST 2025	490007bf-c90d-4d8b-bd99-999ceb22ef8b	f2b6ee379ad2e9b9fd772419c9b5f4cb3d7ee542	Completed	unknown	byovd packed FailedNtCreateFile Device:DifferentBYOVD
Sun May 18 16:33:53 IST 2025	d776612a-eed3-4489-a5ad-0115e83c034a	0823d067541de16325e5454a91b57262365a0705	Completed	ransomware	ransomware medusa packed
Sun May 18 14:21:20 IST 2025	d98f6929-d5fe-4b2e-8626-5bb950a1e82f	7fa9b73ef32259e80950d3625bdf949b7ff3370f	Completed	byovd	byovd packed IoControlCode:0x80002010|0x80002048 Device:ZemanaAntiMalware

Back to Top

Execution Summary

General Information

Report ID: 32e4f6ce-3d96-40ae-9dd3-4a724c1432cb

Execution Date: Sun May 18 11:55:13 IST 2025

Process ID: 7176

Children Processes: 5612, 7132, 2700, 2868, 5280, 3900, 6732, 4776, 1660, 3520, 3732, 5060, 4636, 264, 4156, 3740, 7772, 4420, 4468, 1916, 2452, 7576, 4036, 6344, 7200, 6980, 7572, 392, 4176, 5528, 1792, 7208, 3532, 7568, 5728, 2748, 820, 6996, 3760, 6176, 3484, 1904, 5700, 1456, 616, 2072, 6316, 2204, 1460, 3156, 7560, 7536, 6284, 7600, 1500, 6092, 2340, 1368, 5692, 7084, 4668, 2380, 6472, 4692, 256, 524, 7684, 6340, 1124, 2684, 916, 7224, 2212,

File Meta Information

File Size: 641024 Bytes

File Type: PE32 executable for MS Windows 6.00 (console)

MD5: 47386ee20a6a94830ee4fa38b419a6f7

SHA1: ee4575cf9818636781677d63236d3dc65652deab

SHA256: 736de79e0a2d08156bae608b2a3e63336829d59d38d61907642149a566ebd270

Malware Tags:

Dlls Loaded

Name: c:\dr_plague_pin\third_party\comms_dll\release\comms_dll.dll

Name: c:\dr_plague_pin\third_party\yara_dll\release\jansson.dll

Name: c:\dr_plague_pin\third_party\yara_dll\release\libcrypto-3.dll

Name: c:\dr_plague_pin\third_party\yara_dll\release\yara_scanner.dll

Name: c:\users\public\documents\ee4575cf9818636781677d63236d3dc65652deab
Section Name: .text
Section Permission: RX
Section MemoryMap: start_address=0xe21000,size=0x781f7
Section Name: .rdata
Section Permission: RX
Section MemoryMap: start_address=0xe9a000,size=0x19b04
Section Name: .data
Section Permission: RW
Section MemoryMap: start_address=0xeb4000,size=0xd838
Section Name: .rsrc
Section Permission: RX
Section MemoryMap: start_address=0xec2000,size=0x1e0
Section Name: .reloc
Section Permission: RX
Section MemoryMap: start_address=0xec3000,size=0x7374

Name: c:\windows\syswow64\advapi32.dll

Name: c:\windows\syswow64\bcrypt.dll

Syscalls

[NtSetInformationProcess(ProcessHandle=0xffffffff,ProcessInformationClass=0x22,ProcessInformation=0xfcfad4,ProcessInformationLength=0x4)->0xc0000022]
[NtQueryInformationThread(threadhandle=0xfffffffe,threadinformationclass=0x2a,threadinformation=0xfcf90c,threadinformationlength=0x4,returnlength=0x0)->0x
[0x162(0x74d1f37c0000006b,0x74d1f37c00fcf888,0x74d1f37c00000004,)->0x0]
[0xad(0x74d1f37c01263548,0x74d1f37c001f0003,0x74d1f37c00000000,)->0x0]
[0xcd(0x74d1f37c01263544,0x74d1f37c000f00ff,)->0x0]
[0x1a0(0x74d1f37c00000190,0x74d1f37c0000000d,)->0x0]
[0xc4(0x74d1f37c01263578,0x74d1f37c00000000,0x74d1f37c00000000,0x74d1f37c00000008,0x74d1f37c00100002,0x74d1f37c01263560,0x74d1f37c01263520,0x74d1f37c0000
[0xca(0x74d1f37c0126357c,0x74d1f37c00000001,0x74d1f37c00000000,0x74d1f37c01263560,0x74d1f37c01263520,)->0x0]
[0x90(0x74d1f37c00000198,0x74d1f37c0000018c,0x74d1f37c00000194,0x74d1f37c01263580,0x74d1f37c01263560,0x74d1f37c00000000,0x74d1f37c00000001,0x74d1f37c00fcf
[0xc4(0x74d1f37c012635c0,0x74d1f37c00000000,0x74d1f37c00000000,0x74d1f37c00000008,0x74d1f37c00100002,0x74d1f37c01263560,0x74d1f37c00000000,0x74d1f37c0000
[0xca(0x74d1f37c012635c4,0x74d1f37c00000001,0x74d1f37c00000000,0x74d1f37c01263560,0x74d1f37c00000000,)->0x0]
[0x90(0x74d1f37c000001a0,0x74d1f37c0000018c,0x74d1f37c0000019c,0x74d1f37c012635c8,0x74d1f37c01263560,0x74d1f37c00000000,0x74d1f37c00000000,0x74d1f37c00fcf
[0x1a0(0x74d1f37c00000190,0x74d1f37c00000002,)->0x0]
[0x1a0(0x74d1f37c00000190,0x74d1f37c00000005,)->0x0]
[NtProtectVirtualMemory(ProcessHandle=0xffffffff,BaseAddress=0xe9a000,NumberOfBytesToProtect=0x1000,NewAccessProtection=0x4,OldAccessProtection=0x2)->0x0]
[NtOpenSection(SectionHandle=0x1a4,ACCESS_MASK=0xd,ObjectAttributes=bcrypt.dll)->0x0]
[NtMapviewOfSection(SectionHandle=0x1a4,ProcessHandle=0xffffffff,BaseAddress=0x75e30000,Zerobits=0x0,CommitSize=0x0,SectionOffSet=0x0,ViewSize=0x19000,Inh
[0x1ea(0x74d1f37c0000001c,0x74d1f37c012740a8,0x74d1f37c778f077c,0x74d1f37c00fcf27c,0x74d1f37c012740a8,0x74d1f37c00000000,)->0x1]
[NtProtectVirtualMemory(ProcessHandle=0xffffffff,BaseAddress=0x75e46000,NumberOfBytesToProtect=0x1000,NewAccessProtection=0x2,OldAccessProtection=0x4)->0x
[NtProtectVirtualMemory(ProcessHandle=0xffffffff,BaseAddress=0x779c9000,NumberOfBytesToProtect=0x3000,NewAccessProtection=0x4,OldAccessProtection=0x2)->0x
[NtProtectVirtualMemory(ProcessHandle=0xffffffff,BaseAddress=0x779c9000,NumberOfBytesToProtect=0x3000,NewAccessProtection=0x2,OldAccessProtection=0x4)->0x
[NtProtectVirtualMemory(ProcessHandle=0xffffffff,BaseAddress=0x75e45000,NumberOfBytesToProtect=0x1000,NewAccessProtection=0x4,OldAccessProtection=0x2)->0x
[0x43(0x74d1f3f40c16f908,0x74d1f3f400000001,)->0x778d5970]
[NtProtectVirtualMemory(ProcessHandle=0xffffffff,BaseAddress=0xe9a000,NumberOfBytesToProtect=0x1000,NewAccessProtection=0x2,OldAccessProtection=0x4)->0x0]
[NtProtectVirtualMemory(ProcessHandle=0xffffffff,BaseAddress=0x75e45000,NumberOfBytesToProtect=0x1000,NewAccessProtection=0x2,OldAccessProtection=0x4)->0x
[0x4c()->0x0]
[0x1d4(0x74d1f3f400000190,)->0x0]
[0x4c()->0x0]
[0x19f(0x74d1f37cffffffff,0x74d1f37c00000004,0x74d1f37c00000001,0x74d1f37c00fcf9fc,)->0xc00000bb]
[0x19f(0x74d1f37cffffffff,0x74d1f37c00000004,0x74d1f37c00000001,0x74d1f37c00fcf8bc,)->0xc00000bb]
[0x1c4(0x74d1f37c0000000f,0x74d1f37c00fcf690,0x74d1f37c000000a0,0x74d1f37c00fcf690,)->0x0]
[0x1c4(0x74d1f37c0000000f,0x74d1f37c00fcf6d8,0x74d1f37c000000a0,0x74d1f37c00fcf6d8,)->0x0]
[NtQueryInformationProcess(processhandle=0xffffffff,processinformationclass=0x56,processinformation=0xfcf738,processinformationlength=0x0,returnlength=0xcf
[0x1c4(0x74d1f37c0000000f,0x74d1f37c00fcf6b8,0x74d1f37c000000a0,0x74d1f37c00fcf6d8,)->0x0]
[0x1c4(0x74d1f37c0000001e,0x74d1f37c00fcf788,0x74d1f37c00000018,0x74d1f37c00fcf710,)->0x0]
[0xc0(0x74d1f37c00fcf78c,0x74d1f37c00100003,0x74d1f37c00000000,0x74d1f37c00000000,)->0x0]