cybereason

maliciouslife
BY CYBEREASON

Search | Subscribe

Rundll32: The Infamous Proxy for Executing Malicious Code

rundll32.exe, which allows you to load and execute code. It is often used by adversaries during their offensive operations to execute malicious code through a process which we will explain in detail.

WHAT IS RUNDLL32.EXE?

Rundll32.exe is a Microsoft-signed binary used to load dynamic link libraries (DLLs) in Windows. It is native to Windows and present in both 32 and 64 bit versions, respectively present in these places:

```
C:\Windows\System32\rundll32.exe
C:\Windows\SysWOW64\rundll32.exe
```

Here are the signing details:

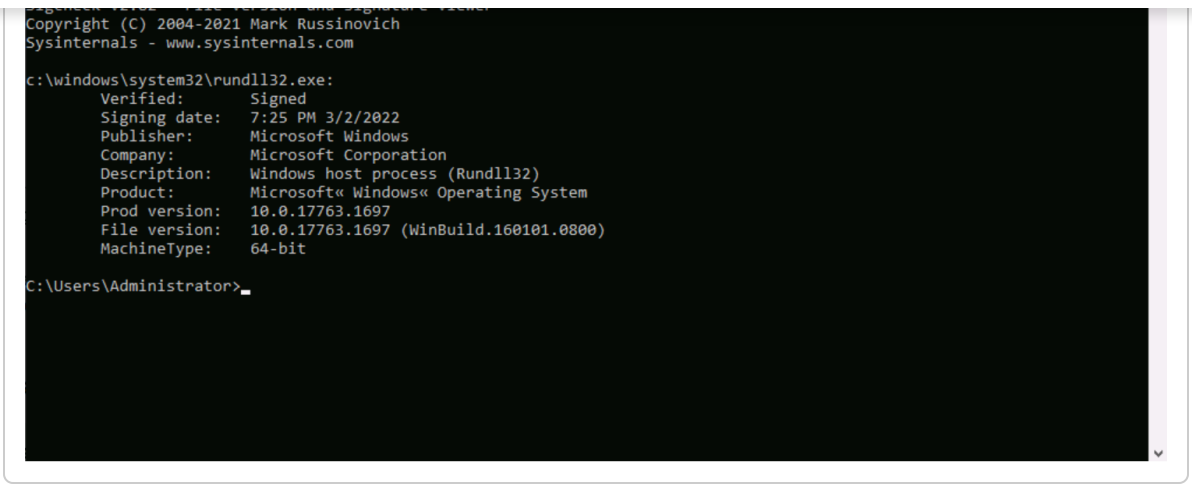

Figure 1 - Rundll32.exe signature overview

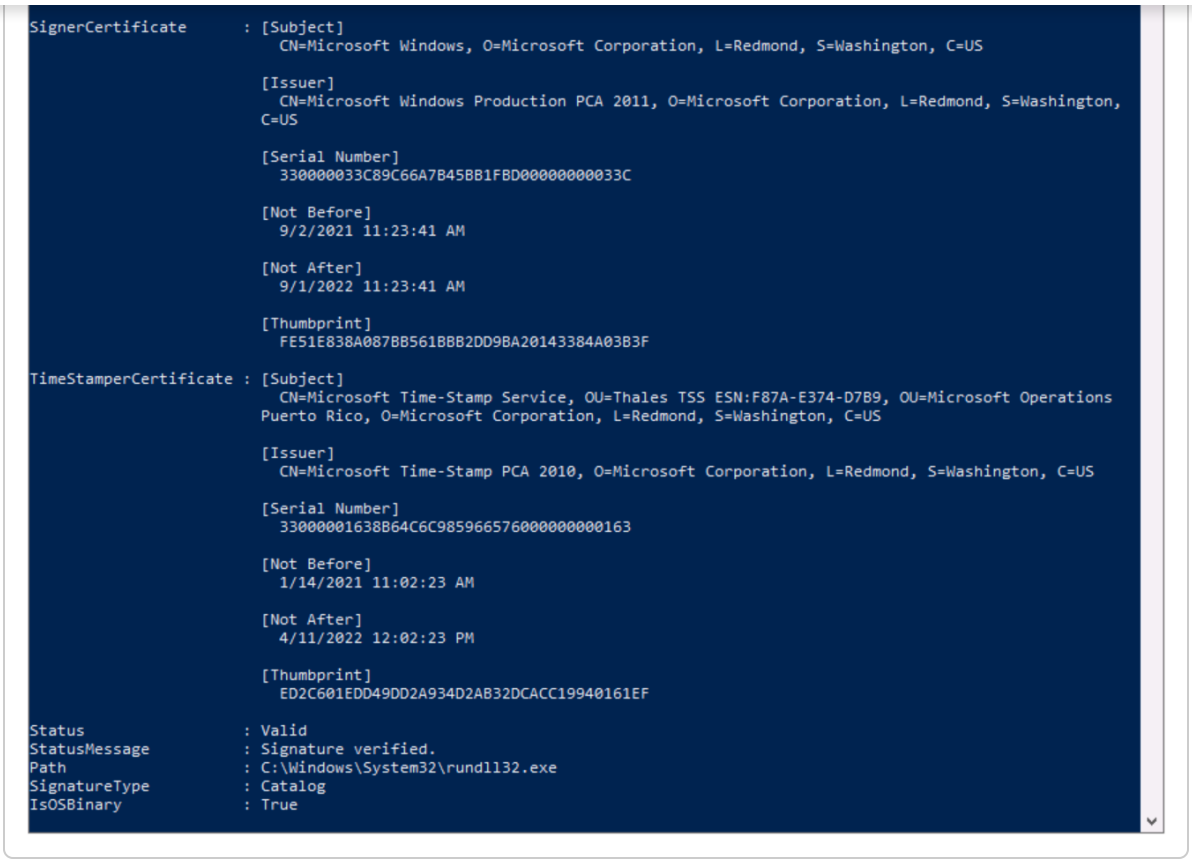

Figure 2 - Rundll32.exe signature details

On the one hand, rundll32.exe is an executable signed by Microsoft which is natively present on all Windows systems; on the other hand, it is also very flexible and efficient for loading code into memory, acting as a proxy for this purpose. Moreover, because rundll32.exe benefits from a certain degree of trust, it could be a possible AppLocker and Software Restriction Policies (SRP) bypass.

Last but not least, rundll32.exe is also able to help to dump the memory of processes, such as the LSASS (*Local Security Authority*

For these reasons, it is a very interesting and frequently used tool by adversaries to proxy execution of arbitrary malicious code and dump LSASS memory. This technique is also mapped and described in the MITRE ATT&CK™ Enterprise framework, rundll32.exe has its own sub-category, as below:

Tactic	Defense Evasion
Technique ID	T1218 .011
Technique Name	System Binary Proxy Execution
Sub-technique Name	System Binary Proxy Execution: Rundll32
Defense Bypassed	Anti-virus, Application control, Digital Certificate Validation

WHO IS CURRENTLY ABUSING RUNDLL32.EXE?

Although rundll32.exe has frequent and undeniable legitimate use, it is also taken advantage of by many attackers, ranging from state-affiliated groups (APTs) to cybercriminal groups to proxy execution of malicious code.

Some of these threat actors are among the most sophisticated attackers around and still rely on rundll32.exe during their operation.

operating out of China that targets entities in the US across a number of industry sectors, including infectious disease researchers, law firms, higher education institutions, defense contractors, policy think tanks, and NGOs.

- **APT29** (aka Cozy Bear): A threat group that has been attributed to Russia's Foreign Intelligence Service (SVR) that often targets government networks in Europe and NATO member countries, research institutes, and think tanks. APT29 reportedly compromised the Democratic National Committee starting in the summer of 2015 and is reportedly responsible for the SolarWinds breach and the resulting supply-chain attack in 2020, where victims of this campaign included government, consulting, technology, telecom, and other organizations in North America, Europe, Asia, and the Middle East.
- **Carbanak:** An international cybercriminal group that targets financial institutions since at least 2013, they install VNC server software that executes through rundll32.

We could also note that tools such as Cobalt Strike can use rundll32.exe to load DLL from the command line. This list could be much longer but the idea is to briefly summarize the importance, dangerousness and diversity of these groups that rely on rundll32.exe, so it is important to understand its mechanism to detect it.

HOW DOES RUNDLL32.EXE WORK?

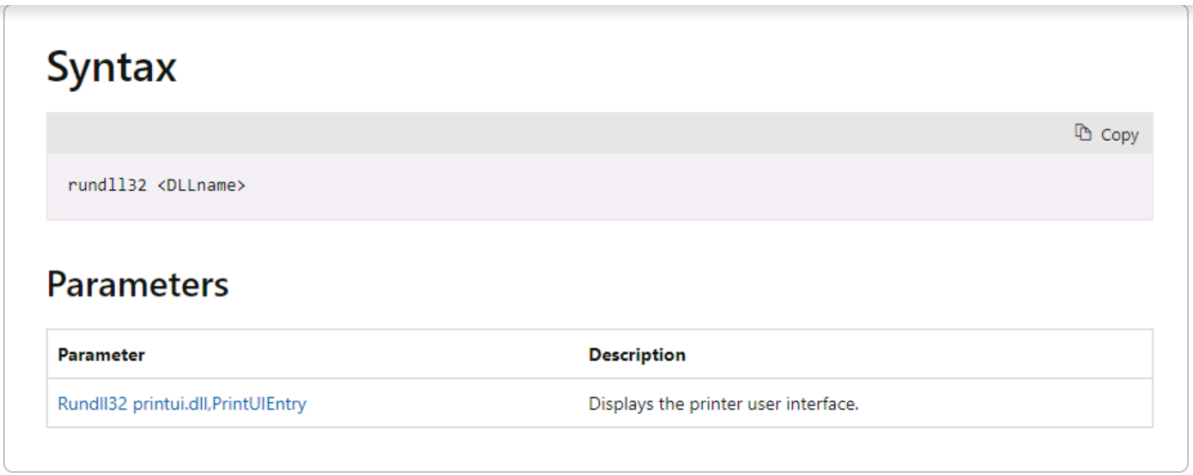

Figure 3 - Microsoft documentation for rundll32.exe

The syntax is not perfectly correct, you have to specify an entry point into the dynamic-link library (DLL) as below (otherwise nothing will happen):

```
rundll32.exe <DLL PATH>,<DLL Entry Point>
```

Note that <DLL PATH> could be both local and remote when the DLL is hosted on an accessible share, using UNC (Universal naming convention) paths in the second scenario.

Even if the entry point does not exist, the system will calls the DllMain function with the DLL_PROCESS_ATTACH value first and the relevant code will be executed followed by an error message because of the missing entry point.

below:

Figure 4 - Rundll32.exe relies on LoadLibraryExW function

Analyzing the stack, we notice a LoadLibraryExW call with:

- ESI register containing the DLL Path (C:\\Code\\RUNDLL32_TEST.dll)
- EDI register containing the entry point (BADENTRY)
- We also pushed values 8 and 0

To better understand these values, let's have a look at LoadLibraryExW function's (from libloaderapi.h) syntax:

```
HMODULE LoadLibraryExW(       // From this example:
  [in] LPCWSTR lpLibFileName, //
L"C:\\Code\\RUNDLL32_TEST.dll"
       HANDLE  hFile,         // 0
  [in] DWORD   dwFlags        // 8
);
```

The following parameters are used:

- **hFile:** This parameter is set to NULL (it is reserved for future use)
- **dwFlags:** The search strategy is set to 8 which correspond to LOAD_WITH_ALTERED_SEARCH_PATH

Know that LoadLibraryExW is theoretically able to load a DLL module without calling the DllMain function of the DLL but here, rundll32.exe is using dwFlags set to 8 which isn't inducing this behavior. The code is equivalent to:

```
hLibModule = LoadLibraryExW(<DLL PATH>,(HANDLE)0x0,8);
  if (hLibModule == (HMODULE)0x0) {
    res = GetLastError();
    if (res == 0xc1) {
     // ...
    }
    else {
      Arguments = (LPCWSTR *)0x0;
      dwFlags = 0x1200;
     // ...
    }
// ...
```

Note: As per Microsoft API documentation, when rundll32.exe calls the DllMain function with any specific entry point, (i.e. a value other than DLL_PROCESS_ATTACH), the return value is ignored. If the return value is FALSE when DllMain is called during process

COULD WE CREATE A DLL TO TEST IT?

We will now create a simple 32-bits C++ DLL named RUNDLL32_TEST.dll which will execute calc.exe once the DllMain function is called with the DLL_PROCESS_ATTACH value and we will create a specific exported entry point to execute cmd.exe. Let's also provide debug information using MessageBox:

```
#include "pch.h"
#include <windows.h>

// Reference:
// https://docs.microsoft.com/en-
us/windows/win32/dlls/dllmain
// https://docs.microsoft.com/en-
us/windows/win32/dlls/dynamic-link-library-entry-
point-function

BOOL APIENTRY DllMain(
    HMODULE hModule, // handle to DLL module (same
as HINSTANCE)
    DWORD fdwReason, // reason for calling function
    LPVOID lpReserved // reserved
) {

    STARTUPINFOA si = {
      sizeof(STARTUPINFOA)
```

```
"C:\\Windows\\System32\\calc.exe";

    // Perform actions based on the reason for
calling
    switch (fdwReason) {

    case DLL_PROCESS_ATTACH:

        // A process loads the DLL (initialize once
for each new process)
        // Return FALSE to fail DLL load

        // Start a "calc.exe" child process
        if (!CreateProcessA(appCalc, NULL, NULL,
NULL, FALSE, 0, NULL, NULL, &si, &pi)) {
            MessageBox(NULL, TEXT("CreateProcessA()
failed\n") + GetLastError(), TEXT("Error"), MB_OK |
MB_ICONINFORMATION);
            return FALSE;
        }

        MessageBox(NULL, TEXT("Hello, DLL is
attached"), TEXT("Hi!"), MB_OK |
MB_ICONINFORMATION);
        return TRUE;

    case DLL_THREAD_ATTACH:
        // Do thread-specific initialization
```

```
        // Thread exits normally
        // Do thread-specific cleanup
        break;

    case DLL_PROCESS_DETACH:
        // A process unloads the DLL
        // Perform any necessary cleanup
        break;

    }
    return TRUE; // Successful DLL_PROCESS_ATTACH
}

// Export function
extern "C"
__declspec(dllexport) void SpecificEntryPoint() {

    MessageBox(NULL, TEXT("Hello from a DLL
exported function"), TEXT("Hi!"), MB_OK |
MB_ICONINFORMATION);

    STARTUPINFOA si = {
      sizeof(STARTUPINFOA)
    };
    PROCESS_INFORMATION pi;
    LPCSTR appCmd =
"C:\\Windows\\System32\\cmd.exe";
```

```
        MessageBox(NULL, TEXT("CreateProcessA()
failed\n") + GetLastError(), TEXT("Error"), MB_OK |
MB_ICONINFORMATION);
    }

}
```

It is important to note that even when targeting a specific entry point (DLL-exported function), the code within the DllMain function from DLL_PROCESS_ATTACH will still be executed.

Note: This behavior was previously discussed, it is related to the dwFlags set to LOAD_WITH_ALTERED_SEARCH_PATH when calling LoadLibraryExW from rundll32.exe (this flag is not under control).

With our DLL, we will have two applications executed (calc.exe and cmd.exe) and two message boxes in this scenario when using the SpecificEntryPoint.

Finally, please note that it is not a good practice to call CreateProcess within DllMain, as it could lead to improper synchronization and cause an application to deadlock as creating a process can load another DLL.

LET'S PLAY WITH OUR DLL!

```
rundll32.exe RUNDLL32_TEST.dll,ThisEntryDoesNotExists
```

This command, targeting a fictive entry point (non-existing DLL-exported function), will result in the execution of calc.exe because of the previously mentioned point.

Note: Without an entry point, even non-existing one, the DLL will not be loaded (despite what Microsoft's documentation describes).

Now, let's play with our DLL-exported function named SpecificEntryPoint to execute cmd.exe using the following command:

```
rundll32.exe RUNDLL32_TEST.dll,SpecificEntryPoint
```

We previously made an observation about LoadLibraryExW which was important as the specified module is loaded into the address space of the calling process (rundll32.exe), hence we could recover loaded modules inspecting the thread's stack as we can see below:

Figure 5 - Thread's stack overview (reaching a provided DLL's entry-point)

independent process:

Figure 6 - Process overview (rundll32 spawned a child with “unmapped” parent)

However, thanks to the Cybereason Defense Platform, we could examine the history, all loaded modules and all other relevant information and also visualize the processes tree to notice that rundll32.exe is the parent of cmd.exe:

Figure 7 - Cybereason’s investigation dashboard to hunt for rundll32.exe executions

Figure 8 - Visualization of the rundll32.exe Process Tree using Cybereason

This example isn’t malicious by itself therefore we wouldn't raise a MalOp™ for this execution, nevertheless, all relevant context and information are stored and accessible.

WHAT ELSE COULD RUNDLL32.EXE DO?

We have seen that rundll32.exe is a powerful asset for adversaries to proxy execution of arbitrary and malicious code. This binary has

Subsystem Service) process memory to retrieve credentials.

MITRE ATT&CK Mapping: This technique is named "OS Credential Dumping: LSASS Memory" and has the ID T1003.001.

The syntax is as below:

```
rundll32.exe C:\windows\System32\comsvcs.dll,
MiniDump <LSASS PID> <DUMP PATH> full
```

Where <LSASS PID> is the process ID of LSASS and <DUMP PATH> the output to be written. Note that this requires Local Administrator or SYSTEM privileges.

Example:

Figure 9 - Rundll32.exe using comsvcs.dll to dump LSASS process memory

We demonstrated that rundll32.exe, which is a signed Microsoft binary, helps to proxy execution of arbitrary and malicious code and dump credentials.

Anything else, please? Of course! This binary is also able to:

- Execute JavaScript

- Manipulate COM Objects
- Etc.

You can refer to the LOLBas projects for further information and examples.

HOW DOES CYBEREASON DETECT RUNDLL32.EXE?

Thanks to our advanced AI-powered technology, we offer the perfect alternative to manually triaging and investigating a flood of alerts full of false positives: the Cybereason MalOp—an interactive graphical display of the full malicious operation from root cause. Instantly see all elements of an attack correlated across every impacted asset with the option for automated or one-click remediation, reducing investigation periods by as much as 93% so Defenders can eliminate threats in a matter of minutes rather than days.

Indeed, on the one hand, we already noticed that Cybereason is able to avoid false positives about benign use of rundll32.exe, using our test DLL to spawn another Windows binary which is not causing any harm to the system:

Figure 10 - Rundll32.exe Process Tree as seen in the Cybereason Defense Platform (Non-Malicious)

Note: For the following tests, we configured the EDR with features and options in "detection" mode only to demonstrate detection capabilities without immediately blocking the attacks ("prevention" mode).

MALICIOUS EXAMPLE #1 - LSASS DUMP

However, Cybereason will – of course – detect malicious behaviors of rundll32.exe and trigger active MalOps when it is relevant. For example, we demonstrated a technique when adversaries abuse rundll32.exe and comsvcs.dll to dump LSASS memory.

The Defender can leverage the MalOp™ to quickly ascertain all relevant information about the attack:

Figure 11 - MalOp for Rundll32.exe dumping LSASS process memory as seen in the Cybereason Defense Platform

The Defender is then able to explore the process tree to hunt for suspicious activities:

Figure 12 - Rundll32.exe Process Tree as seen in the Cybereason Defense Platform (Malicious)

To continue our examples, we could now create a malicious DLL using msfvenom to initiate a reverse shell to an adversary-controlled system. We will select a staged Reverse TCP Meterpreter payload and name it sample.dll:

Figure 13 - Msfvenom - Meterpreter payload creation (DLL)

Let's now execute the DLL using rundll32.exe:

```
rundll32.exe C:\Payloads\sample.dll,XYZ
```

A MalOp is created upon execution of this DLL to notify the Defender:

Figure 14 - MalOp on "rundll32.exe" when loading a Meterpreter payload (DLL)

Note: The DLL is injecting a Meterpreter payload in Memory using Reflective DLL Injection technique, hence the reference to the "Module Injection" within the MalOp.

The Defender is well alerted and then able to explore the process tree to continue the hunt:

The Defender can proceed with additional adversary hunting from the process tree where we notice floating modules which are indicators of Reflective DLL Injection (Delivering Meterpreter):

Figure 16 - Meterpreter - Detection of floating modules

We notice a machine with IP 10.160.155.26 (victim) connecting to another machine with IP 10.160.201.220 on port TCP/8080. This corresponds to the reverse Meterpreter (TCP) payload we created (sample.dll) and executed using rundll32.exe:

Figure 17 - Meterpreter - Connection to the attacker

MALICIOUS EXAMPLE #3 - VBSCRIPT EXECUTION

Let's take a look at another technique to execute VBScript and spawn a PowerShell prompt:

```
rundll32.exe vbscript:"\..\mshtml,RunHTMLApplication
"+String(CreateObject("WScript.Shell").Run("powershell.
exe"),0)
```

indirect execution is still very unusual and suspicious because it could be used by adversaries as a roundabout execution means.

Thanks to our *Behavioral Execution Protection* feature, it is correctly marked as suspicious (unlike the cmd.exe and calc.exe executions we have previously seen, if you do remember):

Figure 18 - MalOp on rundll32.exe "Malicious process" (Behavioral execution protection) as seen in the Cybereason Defense Platform

HUNTING QUERIES

The queries provided in this section can be used to hunt for possible malicious rundll32.exe processes. First of all, the following query provide all instances of rundll32.exe (including non-malicious ones), to have an overview of activities:

```
/#/s/search?queryString=0<-
Process"elementDisplayName:)rundll32.exe"&viewDetails
=false
```

Result:

Then, it is possible to refine and limit the search to suspicious occurrences, you can filter on Has Suspicion is True and/or Has Malops is True with the query below or using the user interface:

```
/#/s/search?queryString=0<-
Process"elementDisplayName:)rundll32.exe
,hasSuspicions:true,hasMalops:true"&sorting=
elementDisplayName&sortingDirection=1&viewDetails=false
```

Figure 20 - [Hunt] List of all Rundll32.exe processes with Suspicions/MalOps

It is also possible to extend your query and hunt for outgoing connections with the query below (or using the user interface to graphically build the query):

```
/#/s/search?queryString=2<-
Process"elementDisplayName:)rundll32.exe,hasOutgoingCo
nnection:true"->connections-
>remoteAddress&sorting=elementDisplayName&sortingDirec
tion=1&viewDetails=false
```

Figure 21 - [Hunt] Query creation - Rundll32 with outgoing connections as seen in the Cybereason Defense Platform

Figure 22 - [Hunt] Results - Rundll32 with outgoing connections as seen in the Cybereason Defense Platform

We notice the remote IP 10.160.201.222 which is the adversary-controlled machine that received the Meterpreter reverse shell from the victim machine.

In the real world, this IP address could be used as a pivot (or "IoC" for indicator of Compromise) to pursue the Incident Response (IR) process and hunt for other malicious activities related to this specific address.

Cybereason is dedicated to teaming with Defenders to end attacks on the endpoint, across enterprise, to everywhere the battle is taking place. Learn more about the AI-driven Cybereason Defense Platform here or schedule a demo today to learn how your organization can benefit from an operation-centric approach to security.

ABOUT THE RESEARCHER:

certified). He has also worked as a security architect and more formerly as a network security engineer, system engineer and developer. When he is not working, he enjoys spending time with his family and going for walks in nature to recharge his batteries. Always looking for innovation and new challenges, he is delighted to work in Cybereason's Blue Team, at the forefront of Cybersecurity and continue to learn in this exciting and ever-changing world.

SHARE

ABOUT THE AUTHOR

Cybereason Blue Team

The Cybereason Blue Team is a global unit focused on mitigating advanced adversarial techniques leveraged by high level Threat Actors and Red Teams across the globe. The Blue Team is comprised of experts in Red Teaming, Penetration Testing, Digital Forensics and Incident Response, giving them a unique insight into both sides of the coin. Working alongside customers and third-party Red Teams, and leveraging the Cybereason platform, the Blue Team is able to push the boundaries of detection and response well beyond commonly known Tactics, Techniques and Procedures (TTPs), reversing the adversarial advantage long before new techniques are adopted by mainstream Threat Actors. As part of Cybereason Threat Intelligence, the Blue Team is able to

All Posts by Cybereason Blue Team →

Related Posts

THREAT ANALYSIS REPORT: From Shathak Emails to the Conti Ransomware

The ITG23 group is partnering with the TA551 (Shatak) threat group to distribute ITG23's TrickBot and BazarBackdoor malware which attackers use to deploy Conti ransomware on compromised systems...

THREAT ANALYSIS: Cobalt Strike - IcedID, Emotet and QBot

The Cybereason GSOC delivers details on three recently observed attack scenarios where fast-moving malicious actors used the malware loaders IcedID, QBot and Emotet to deploy the Cobalt Strike framework on the compromised systems...

cybereason

Unlocking the Potential of AI in Cybersecurity: Embracing the Future and Its Complexities

Malicious Life Podcast: Operation Snow White, Part 2

THREAT ANALYSIS: Beast Ransomware

CATEGORIES

Research
Podcasts
Webinars
Resources
Videos
News

All Posts

NEWSLETTER

Never miss a blog

Get the latest research, expert insights, and security industry news.

About

- who we are
- careers
- contact

Resources

- blog
- case studies
- webinars
- white papers

Platform

- overview
- endpoint protection
- edr
- mdr

Terms of Use | Privacy Notice | Do Not Sell | Security