Embargo ransomware: Rock’n’Rust

By Jan HolmanTomáš Zvara

Archived: 2026-04-05 15:26:21 UTC

ESET researchers have discovered new Rust-based tooling leading to the deployment of Embargo ransomware. Embargo is a relatively new player in the ransomware scene, first observed by ESET in June 2024. The new toolkit consists of a loader and an EDR killer, named MDeployer and MS4Killer respectively by ESET. MS4Killer is particularly noteworthy as it is custom compiled for each victim’s environment, targeting only selected security solutions. Both tools are written in Rust, the Embargo group’s language of choice for developing its ransomware.

> **Key points of this blogpost:**
>
> - Embargo is developing and testing new Rust-based tooling.
> - Differences in deployed versions, bugs, and leftover artifacts suggest that these tools are under active development.
> - The threat actor abuses Safe Mode to disable security solutions.
> - Embargo tailors its tools to each victim.

Overview

In July 2024, we observed ransomware incidents targeting US companies, where the threat actor utilized its new tooling. The versions of MDeployer and MS4Killer observed in each intrusion differ slightly, suggesting that the tools are actively developed. Interestingly, we spotted two different versions of MDeployer in a single intrusion, probably tweaked after a first, failed attempt.

This blogpost focuses on the analysis of MDeployer and MS4Killer and activity preceding the execution of the Embargo ransomware. MDeployer is a malicious loader used for deployment of MS4Killer and Embargo ransomware. MS4Killer is an EDR killer that abuses a vulnerable driver to disable the security products running on the victim’s machine.

Embargo

Embargo, observed for the first time in ESET telemetry in June 2024, made its public appearance in May 2024. Apart from successfully breaching high-profile targets, the group attracted attention because of its choice of programming language for ransomware payload. Embargo chose Rust, a cross-platform programming language, allowing development of more versatile ransomware targeting both Windows and Linux. Coming after BlackCat and Hive, Embargo is yet another group developing ransomware payloads in Rust.

Based on its modus operandi, Embargo seems to be a well-resourced group. It sets up its own infrastructure to communicate with victims (Figure 1), but also allows for communication via Tox. The group pressures victims

into paying by using double extortion and publishes the stolen data on its leak site. In an interview with an alleged group member, the group representative mentions a basic payout scheme for affiliates, suggesting that the group is providing RaaS (ransomware as a service). Recent law enforcement disruptions, affecting notorious groups like BlackCat and LockBit, triggered some reorganization in the RaaS space. These changes in global RaaS environment support the emergence of a sophisticated new actor. Given the group's sophistication, the existence of a typical leak site, and the group's claims, we assume that Embargo indeed operates as RaaS provider.

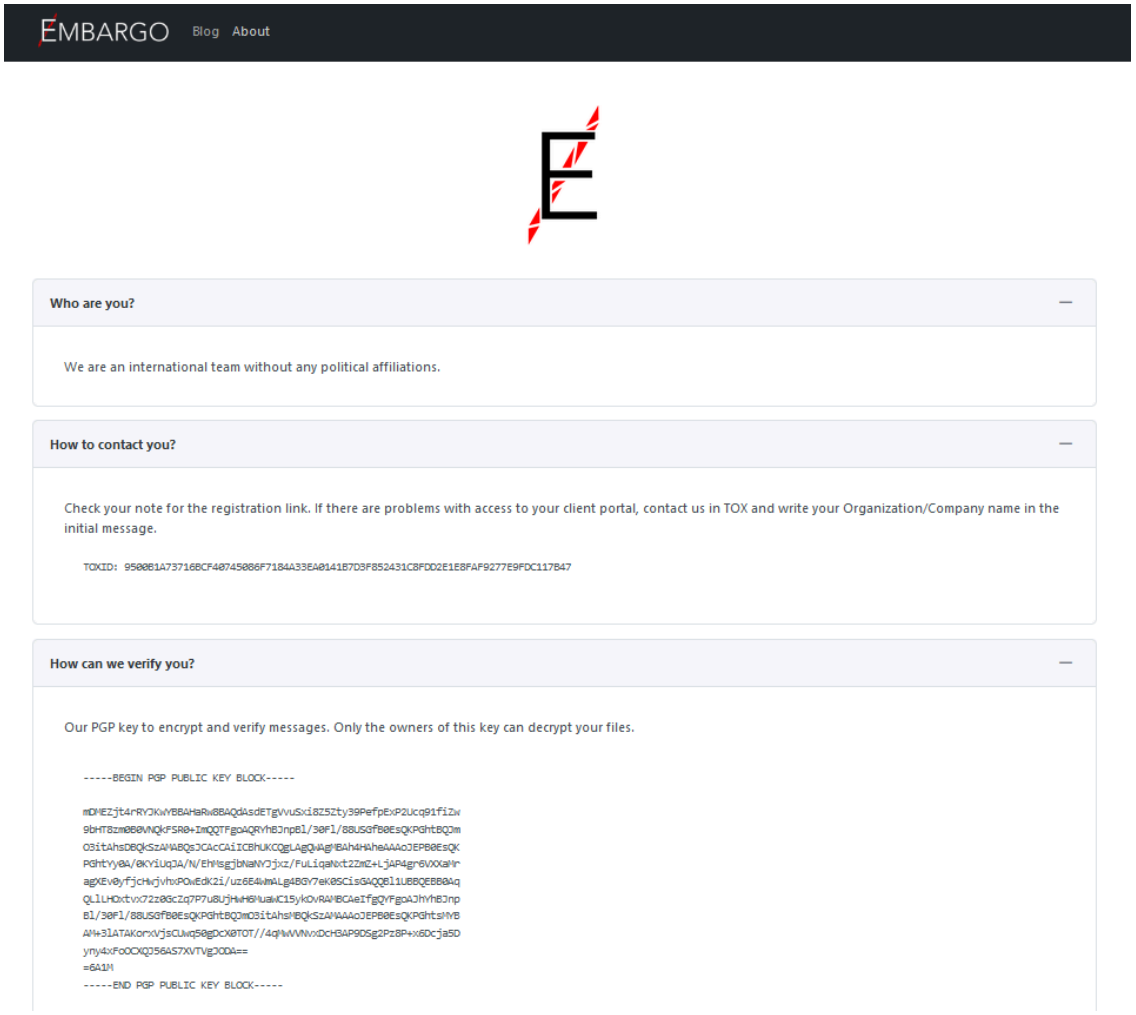
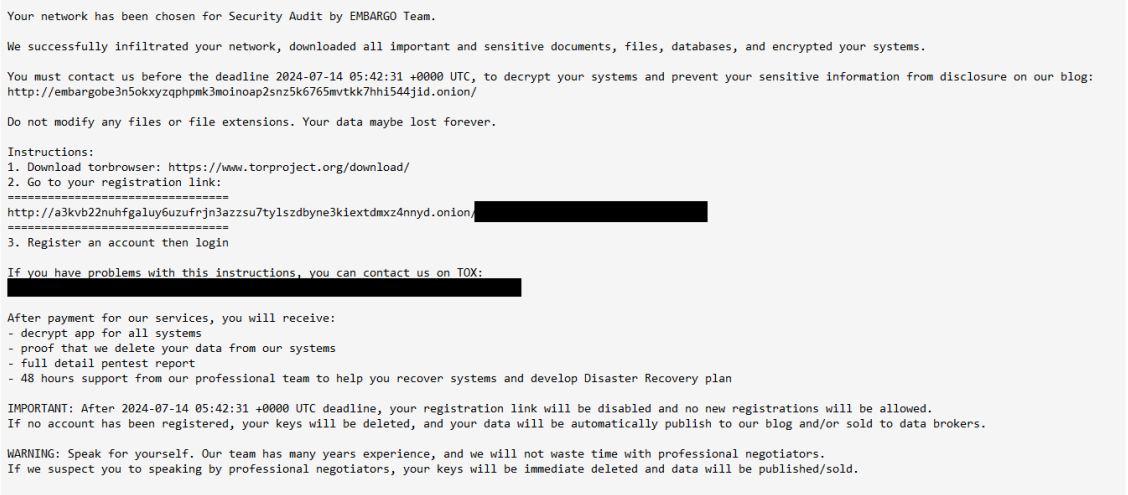


Figure 1. Embargo About page on leak site

Embargo ransomware payloads that we observed during the July 2024 incidents share these attributes:

- Embargo ransomware drops its ransom note (Figure 2) named HOW_TO_RECOVER_FILES.txt in each encrypted directory.
- Encrypted files obtain a random six-letter extension consisting of hexadecimal characters, e.g., .b58eeb or .3d828a.
- Payloads create the mutex IntoTheFloodAgainSameOldTrip.

In a previous analysis from Cyble researchers, payloads created the mutex LoadUpOnGunsBringYourFriends. Noticeably, both mutex names are based on the lyrics of popular rock songs. Our analysis is consistent with that found in the Cyble article.



```
Your network has been chosen for Security Audit by EMBARGO Team.

We successfully infiltrated your network, downloaded all important and sensitive documents, files, databases, and encrypted your systems.

You must contact us before the deadline 2024-07-14 05:42:31 +0000 UTC, to decrypt your systems and prevent your sensitive information from disclosure on our blog:
http://embargobe3n5okxyzqphpmk3moinoap2snz5k6765mvtkk7hhi544jid.onion/

Do not modify any files or file extensions. Your data maybe lost forever.

Instructions:
1. Download torbrowser: https://www.torproject.org/download/
2. Go to your registration link:
================================
http://a3kvb22nuhfgaluy6uzufrjn3azzsu7ty1szdbyne3kiextdmxz4nnyd.onion/[redacted]
================================
3. Register an account then login

If you have problems with this instructions, you can contact us on TOX:
[redacted]

After payment for our services, you will receive:
- decrypt app for all systems
- proof that we delete your data from our systems
- full detail pentest report
- 48 hours support from our professional team to help you recover systems and develop Disaster Recovery plan

IMPORTANT: After 2024-07-14 05:42:31 +0000 UTC deadline, your registration link will be disabled and no new registrations will be allowed.
If no account has been registered, your keys will be deleted, and your data will be automatically publish to our blog and/or sold to data brokers.

WARNING: Speak for yourself. Our team has many years experience, and we will not waste time with professional negotiators.
If we suspect you to speaking by professional negotiators, your keys will be immediate deleted and data will be published/sold.
```

Figure 2. Embargo ransom note

MDeployer

MDeployer is the main malicious loader Embargo tries to deploy onto machines in the compromised network – it facilitates the rest of the attack, resulting in ransomware execution and file encryption.

Based on the name field in the IMAGE_EXPORT_DIRECTORY section of its PE header, we can tell that Embargo calls this tool Deployer. Thus, we decided to refer to it as MDeployer – EMbargo Deployer.

Its main purpose is to decrypt two encrypted files a.cache and b.cache (dropped by an unknown previous stage) and execute two payloads: MS4Killer and Embargo ransomware.

- It first attempts to decrypt the MS4Killer payload from the file b.cache, drops the decrypted file into praxisbackup.exe, and executes it.
- Next, it does the same for the ransomware payload, which is decrypted from a.cache, saved as pay.exe, and executed.
- When the ransomware finishes encrypting the system, MDeployer terminates the MS4Killer process, deletes the decrypted payloads and a driver file dropped by MS4Killer, and finally reboots the system.

MS4Killer is expected to run indefinitely, and MDeployer verifies this by calling the API function WaitForSingleObject, expecting the return value WAIT_TIMEOUT. If it is not running as it should be, MDeployer logs the message sysmon exited early and exits without executing the second payload. We discuss logging later in this blogpost.

In all MDeployer versions we've seen, both payloads were decrypted using the same hardcoded RC4 key – wlQYLoPCil3niI7x8CvR9EtNtL/aeaHrZ23LP3fAsJogVTIzdnZ5Pi09ZVeHFkiB.

During its execution, MDeployer interacts with multiple files. To ease understanding, Figure 3 demonstrates the relationship between the files.

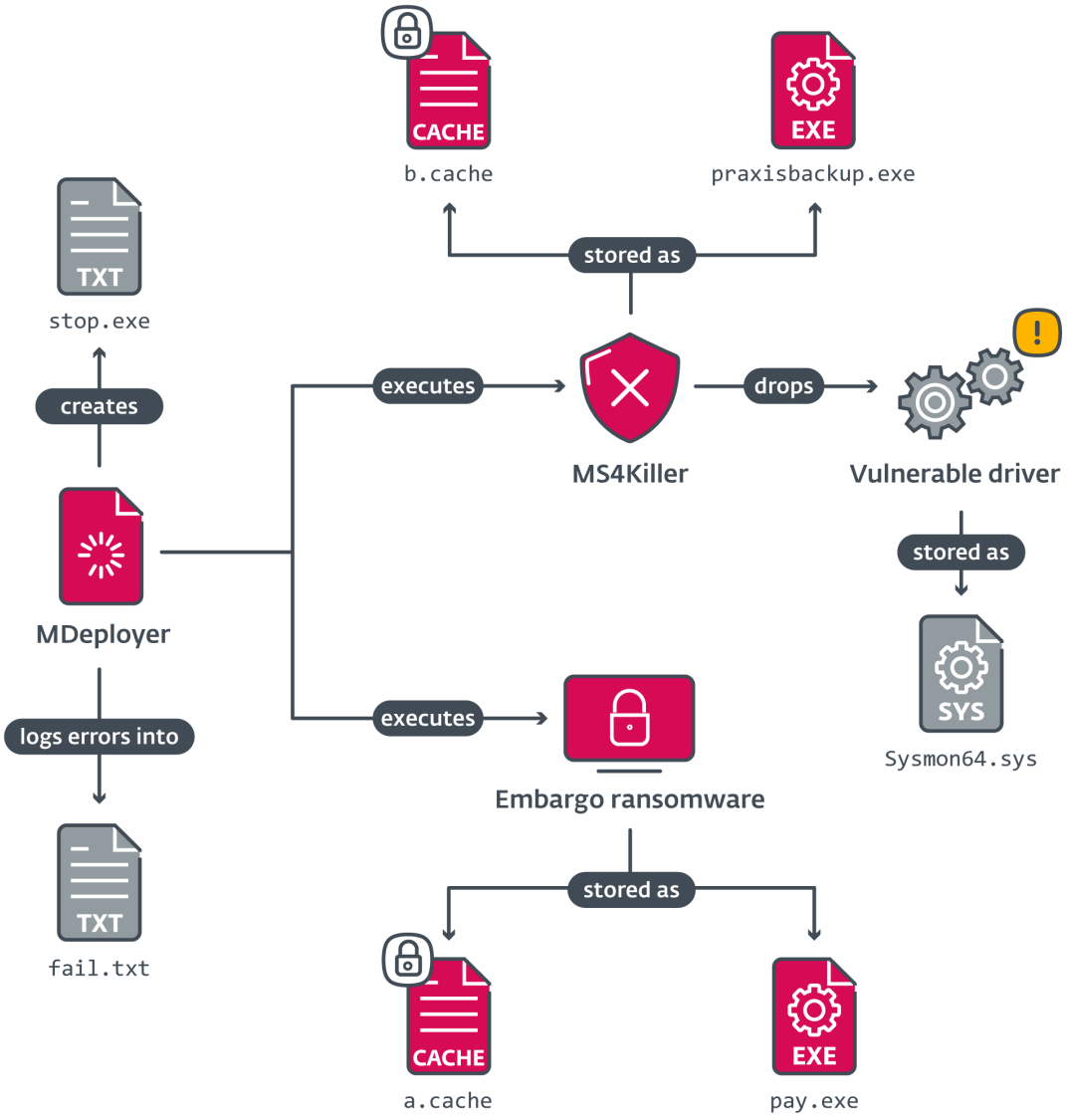


Figure 3. MDeployer execution diagram

Table 1 lists their purposes.

Table 1. Files manipulated by MDeployer

Path	Description
C:\Windows\Debug\b.cache	RC4-encrypted MS4Killer.
C:\Windows\Debug\a.cache	RC4-encrypted Embargo ransomware.
C:\Windows\praxisbackup.exe	Decrypted MS4Killer.
C:\Windows\Debug\pay.exe	Decrypted Embargo ransomware.
C:\Windows\Debug\fail.txt	Log file.
C:\Windows\Debug\stop.exe	Dummy file used for control flow.

Path	Description
C:\Windows\Sysmon64.sys	Legitimate vulnerable driver dropped by MS4Killer.

Safe Mode abuse

With only one exception among the incidents we investigated, where we saw it deployed as a DLL, MDeployer was compiled as an EXE file. The DLL variant contains the additional capability to disable security solutions.

For an overview of the DLL execution flow, refer to Figure 4.

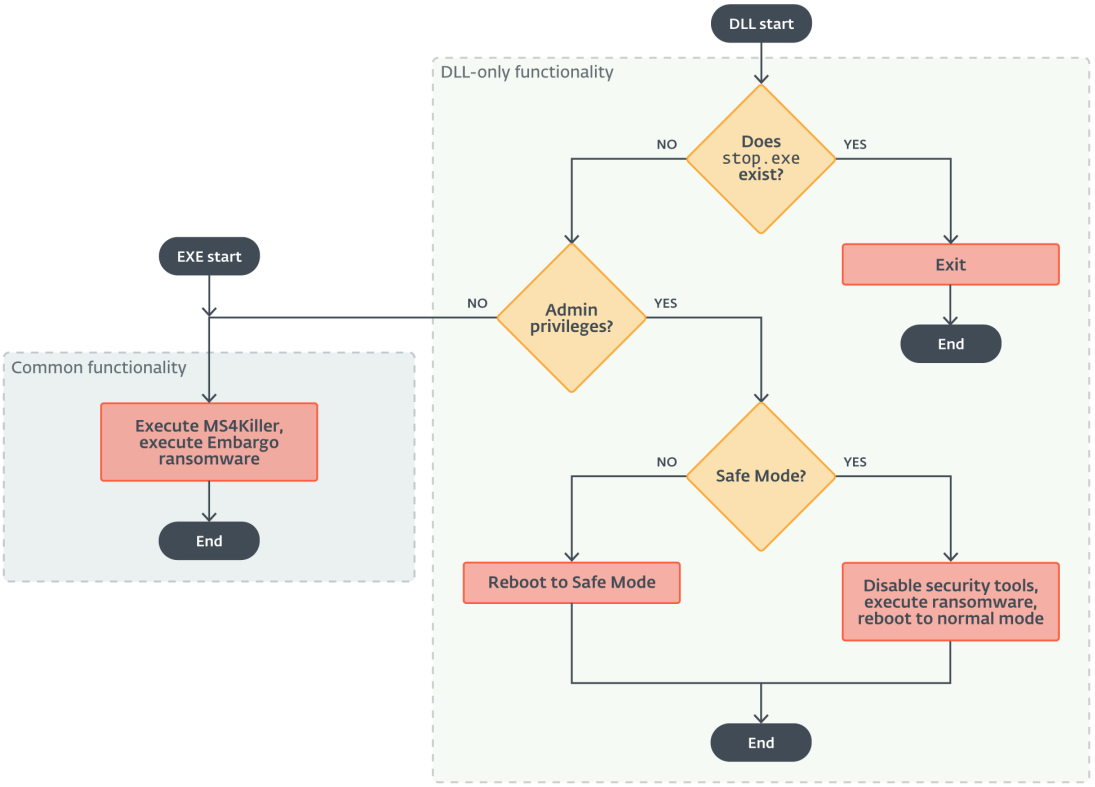


Figure 4. DLL MDeployer execution flow

The first difference happens right at the beginning of the DLL execution – this version actually checks whether the file stop.exe exists. The existence of this file means that MDeployer was already executed in the past and either it successfully deployed the ransomware payload or it exited with an error. So, if the file is found, the loader only does its cleanup routine and exits. Note that the EXE versions create the stop.exe file, but never check for its existence.

The next thing the DLL version of MDeployer does is to check whether it was executed with admin privileges. If it wasn't, it goes on exactly like the EXE version. In fact, the EXE versions were likely compiled using the source code of this single execution branch.

However, if it was executed with admin privileges, the loader attempts to reboot the victim's system into Safe Mode in order to disable selected security solutions.

Safe Mode, a diagnostic mode of the Windows OS, runs the system with only minimal functionality. Because of this, most cybersecurity measures and protections are not in effect in Safe Mode, which provides an opportunity for threat actors to exploit it to avoid detection. This technique is known among mature ransomware groups and has been abused in the past, as reported by Forbes in 2022.

The security-disabling functionality happens in two steps.

Step 1

The purpose of the first step is to reboot the system into Safe Mode. The loader achieves this using a combination of Windows command line tools bcdedit, sc, and reg to:

- set Safe Mode as the default boot mode,
- disable Windows Defender in Safe Mode,
- create a service, irnagentd, that executes the loader after the system is rebooted into Safe Mode, and
- restart the system.

Refer to the *Commands used by MDeployer* section for the full list of commands executed by the loader.

Step 2

Once in Safe Mode, the loader disables selected security tools by renaming their installation directories, then executes the Embargo ransomware payload.

After that, it does a "Safe Mode cleanup" – it deletes the decrypted ransomware file pay.exe, creates the control flow file stop.exe to prevent double encryption, deletes the persistence service irnagentd, and reboots the system back into normal mode.

BAT disabler

In one of the incidents, we also saw the extra functionality of the DLL loader implemented as a BAT script. This script targets a single security solution – a theme you will encounter again, later in this article. It used the same technique of rebooting into Safe Mode with the help of a persistence service, irnagentd, and then renaming the installed security software's installation directory. It even used the same stop.exe file for control flow and logged error messages into fail.exe (fail.txt in MDeployer).

This again shows that Embargo modifies its tools to suit each victim's environment.

Logging

In case MDeployer encounters any errors, it logs error messages into the file fail.txt and then creates the file stop.exe.

There are four stages that the attacker distinguishes in their log messages – they use a different prefix for logging errors in each of them:

- [dec] – payload decryption,
- [exec] – ransomware execution,
- [execk] – MS4Killer execution, and
- [kler] – MS4Killer run (this prefix is used when MS4Killer exits unexpectedly).

In the DLL version there are additional log message prefixes compared to the EXE versions:

- [sc], [sc delete] – creating or deleting the service irnagentd,
- [reg], [reg-del] – modifying Windows registry, and
- [setsb] – using the bcdedit.exe command line tool to set Safe Mode on next restart.

Cleanup

MDeployer has several variants of a cleanup routine launched at different occasions. This happens after the loader successfully executes the ransomware payload, and also if any errors are encountered during loader execution.

During cleanup, the loader terminates the MS4Killer process, deletes the decrypted payloads and the vulnerable driver dropped by MS4Killer, and creates the flow control file stop.exe.

In case the cleanup routine was prompted by the existence of stop.exe, MDeployer also deletes its own PE file.

Finally, it reboots the system by calling shutdown -r -f -t 00.

Execution

In all of the observed cases, the persistence of the loader was achieved by a scheduled task, Perf_sys (Figure 5), created by an already elevated system user BITCH\Administrator.

```
<Task version="1.3" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Author>BITCH\Administrator</Author>
    <URI>\Perf_sys</URI>
  </RegistrationInfo>
  <Triggers>…
  </Triggers>
  <Principals>
    <Principal id="Author">
      <RunLevel>HighestAvailable</RunLevel>
      <UserId>NT AUTHORITY\System</UserId>
      <LogonType>S4U</LogonType>
    </Principal>
  </Principals>
  <Settings>…
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>C:\Windows\fdasvc.exe</Command>
    </Exec>
  </Actions>
</Task>
```

Figure 5. Scheduled task Perf_sys by BITCH\Administrator e*xecuting the loader*

In one of the cases, we also collected a PowerShell script leading to the execution of MDeployer. The script was notably similar to the one used by WinRM-fs, so we assume with medium confidence that Embargo used that or a similar tool to deliver the loader from an unprotected machine.

Active development

There are several inconsistencies and examples of "messy control flow" in the loader samples we've seen so far that suggest the group's tools are still in active development and not "production ready".

The fact that MDeployer deletes the vulnerable driver dropped by MS4Killer is particularly interesting because it shows that the two tools are being developed together. And yet there is a partial overlap in functionality – both MS4Killer and the DLL version of MDeployer attempt to disable security solutions.

It is not uncommon to see the loader delete the payload files only to attempt to execute one of them immediately after. See Figure 6, where MDeployer calls the cleanup function, during which pay.exe is deleted, but then tries to execute that very same file.

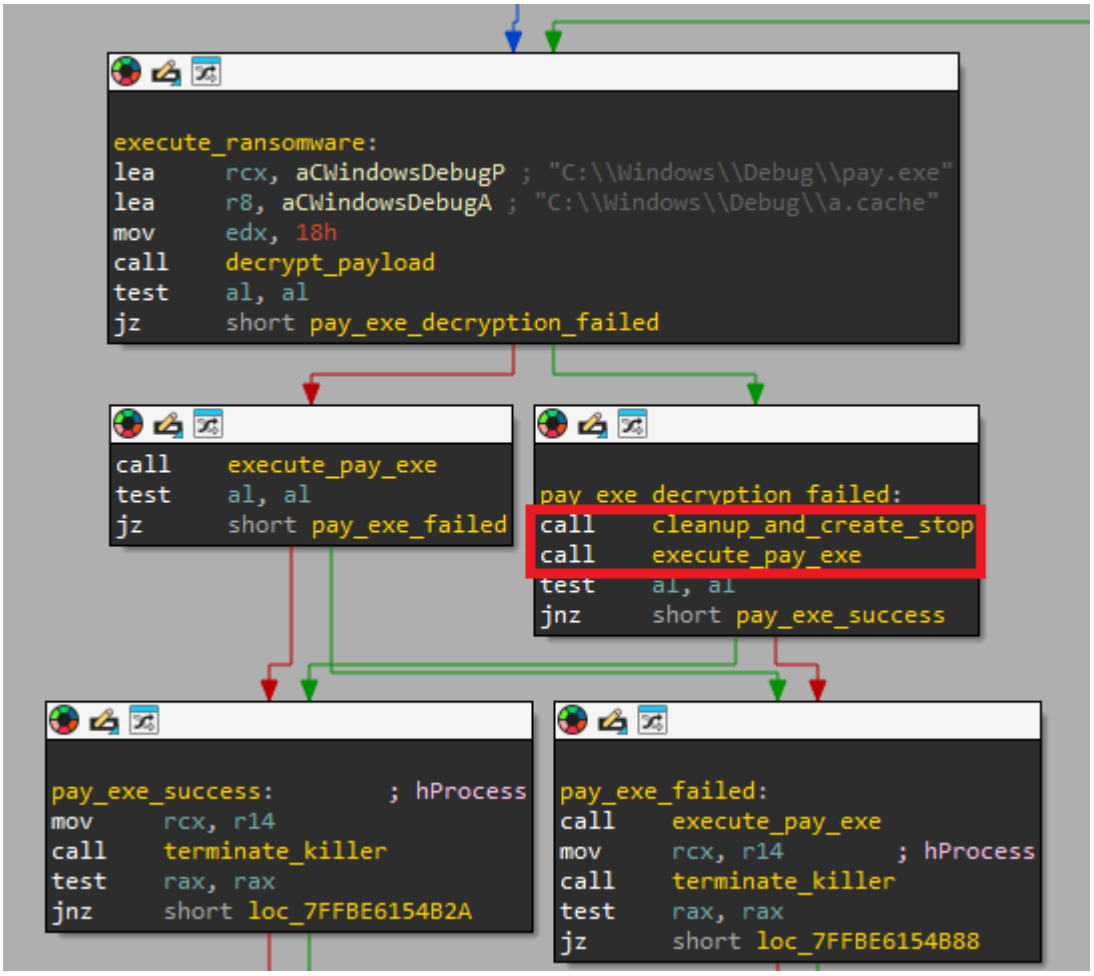


Figure 6. MDeployer in IDA Pro – the highlighted section shows attempted execution after deletion

In fact, the DLL version of the loader we've seen contains several bugs that prevent it from working altogether. This could explain why we've seen multiple versions of the loader being used in a single incident – the threat actor likely finds out about these problems as they go and then has to adapt on the fly.

MS4Killer

MS4Killer is a typical defense evasion tool that terminates security product processes using the technique known as Bring Your Own Vulnerable Driver (BYOVD). It is written, similar to the loader, in Rust. We believe that MS4Killer was heavily inspired by s4killer, a proof of concept (POC) published on GitHub, conveniently also written in Rust. Due to the resemblance with this existing POC, we refer to this tool as MS4Killer – short for EMbargo s4killer.

Extending the functionality

s4killer is designed to select a running process and terminate it from the kernel. It does so by installing and abusing a vulnerable driver that is stored in a global variable (.rdata section in the compiled code). The PID of the process to terminate is passed to s4killer as a program argument. The termination is performed via FilterConnectCommunicationPort and FilterSendMessage from the minifilter API.

Embargo extended the POC functionality with the following features:

- MS4Killer runs in an endless loop, constantly scanning for running processes.
- The list of process names to kill is hardcoded in the binary.
- The embedded driver blob is encrypted using RC4.
- Binary strings are encrypted using simple XOR, namely log messages, process names, and the RC4 key used for driver decryption.
- During the process termination phase, MS4Killer spawns itself as a child process, passing the PID of the process to kill as an argument.
- Process scanning and process termination are split into multiple threads by utilizing Rayon, a data parallelism library for Rust.

BYOVD

Bring your own vulnerable driver is a well-known technique where a threat actor abuses signed, vulnerable kernel drivers to gain kernel-level code execution. Ransomware affiliates often incorporate BYOVD tooling in their compromise chain to tamper with security solutions protecting the infrastructure being attacked. After disabling the security tooling, affiliates can run the ransomware payload without worrying whether their payload gets detected.

In this particular case, MS4Killer abuses an older, vulnerable minifilter driver: probmon.sys, version 3.0.0.4 (Figure 7), signed by an already revoked certificate from *ITM System Co.,LTD*. The driver is embedded in the MS4Killer binary as an RC4-encrypted blob. We reported the ITW misuse of this driver to Microsoft.

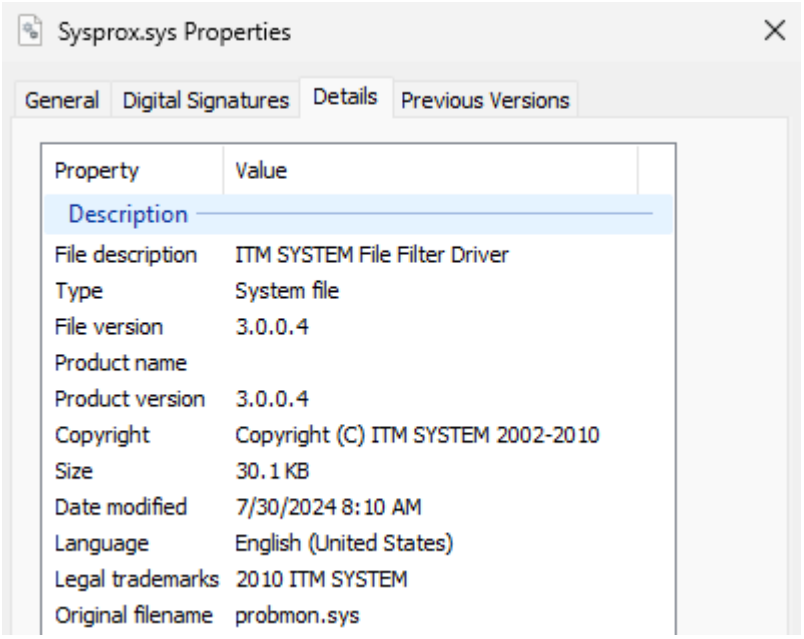


Figure 7. Attributes of the exploited driver probmon.sys

String decryption

MS4Killer uses encryption to hide embedded strings in the binary from plain sight: specifically, it XORs log message strings, the RC4 key used to decrypt the embedded driver, and the list of process names to terminate. Figure 8 shows an example of log message decryption, where the Windows OpenProcessToken API is called. If the function fails, a user-defined function (renamed to xor_str in Figure 8) decrypts the XORed string and stores the result, [-] OpenProcessToken, into its first argument passed by reference. The decrypted string, appended with error information, is then written to standard out.

```
681         TokenHandle.alloc_size = 0LL;
682         CurrentProcess = GetCurrentProcess();
683         if ( !OpenProcessToken(CurrentProcess, 0x30u, &TokenHandle) )
684         {
685           LastError = w_GetLastError();
686           xor_str(&decrypted, encrypted_log_str, 0x14uLL);// [-] OpenProcessToken
687           key_stream.alloc_size = 0LL;
688           LODWORD(key_stream.buff) = LastError;
689           log_error(decrypted._string.buff, decrypted._string.size, &key_stream, &off_7FF6D1C1E8B0, &off_7FF6D1C1EA50);
690         }
```

Figure 8. Decryption of log message after OpenProcessToken *API call*

Loading probmon.sys

As mentioned previously, the legitimate vulnerable driver is embedded as an RC4-encrypted blob (using the key FGFOUDa87c21Vg+cxrr71boU6EG+QC1mwViTciNaTUBuW4gQbcKboN9THK4K35sL), which is also XOR encrypted, in the MS4Killer binary. We have observed two different file paths where MS4Killer drops the vulnerable driver:

- C:\Windows\System32\drivers\Sysprox.sys (Figure 9)
- C:\Windows\Sysmon64.sys

```
688     qmemcpy(driver_path, "C:\\Windows\\System32\\Drivers\\Sysprox.sys", 39);
689     xor_str(&rc4_key, xored_rc4_key, 0x40uLL);
690     v50 = rc4_key.buff;
691     if...                                   // check if size is 0x40
692     v51 = _mm_loadu_si128(rc4_key.buff);    // move rc4 key to Buffer
693     v52 = *(rc4_key.buff + 1);
694     v53 = _mm_loadu_si128(rc4_key.buff + 2);
695     v481 = *(rc4_key.buff + 3);
696     v480 = v53;
697     v479 = v52;
698     *lpMem = v51;
699     *&rc4_key_buff[48] = v481;
700     *&rc4_key_buff[32] = v53;
701     *&rc4_key_buff[16] = v52;
702     *rc4_key_buff = v51;
703     rc4_init(&key_stream, rc4_key_buff);
704     BYTE2(v524) = 0;
705     memmove(rc4_key_buff, &obf_driver, 0x7890uLL);
706     v54 = v524;
707     v55 = BYTE1(v524);
708     for...                                  // rc4 enc loop
709     LOBYTE(v524) = v54;
710     BYTE1(v524) = v55;
711     BYTE3(v524) = 0;
712     if ( !w_CreateFile(driver_path, 39LL) )
```

Figure 9. Decryption and dropping of vulnerable driver probmon.sys

Driver loading is consistent with s4killer:

1. enabling the SeLoadDriverPrivilege necessary for loading and unloading device drivers,
2. creating a service via CreateServiceW,
3. creating additional registry keys, required for filter loading, in HKLM\SYSTEM\ControlSet001\services\<service_name>, and
4. loading a minifilter driver into the system via FilterLoad.

We have observed MS4Killer use three different service names so far: Sysprox, Proxmon, and Sysmon64.

Hidden process list

MS4Killer constantly compares running processes against an embedded list of security software process names, which are also XOR-encrypted. Right after the driver loads, MS4Killer decrypts the list of process names (Figure 10).

```
1240    xor_str(v442, aCMKQ7yl, 0x11uLL);          // SentinelAgent.exe
1241    xor_str(v441, aCMKSbQhc, 0x17uLL);         // SentinelAgentWorker.exe
1242    xor_str(v440, byte_140055840, 0x17uLL);    // SentinelServiceHost.exe
1243    xor_str(v439, byte_140055857, 0x18uLL);    // SentinelStaticEngine.exe
1244    xor_str(v438, aA5gdDiWWFhc, 0x17uLL);      // LogProcessorService.exe
1245    xor_str(v437, aCMiQOnMU9xc9s, 0x1FuLL);    // SentinelStaticEngineScanner
1246    xor_str(v444, aCMrU0DJCkNh, 0x19uLL);      // SentinelHelperService.exe
1247    xor_str(v443, byte_1400558BE, 0x1DuLL);    // SentinelBrowserNativeHost.exe
1248    xor_str(phkResult, aA5tyDyWD, 0x10uLL);    // LogCollector.exe
1249    xor_str(hKey, aCMwJBhMTkNh, 0x19uLL);      // SentinelMemoryScanner.exe
1250    xor_str(&v513, aCMhB0Dq, 0x12uLL);         // SentinelRanger.exe
1251    xor_str(TokenHandle, aCMh13uMhc, 0x17uLL);// SentinelRemediation.exe
1252    xor_str(Luid, aCMhJIlOIH7, 0x1BuLL);       // SentinelRemoteShellHost.exe
1253    xor_str(lpMem, aCMiKLjM2cH9s, 0x1FuLL);    // SentinelScanFromContextMenu.exe
1254    xor_str(hHandle, aCylance, 0xEuLL);        // CylanceSvc.exe
1255    xor_str(&Buffer, byte_140055975, 8uLL);    // ekrn.exe
1256    xor_str(Luid, aZV8D, 8uLL);                // WRSA.exe
1257    xor_str(lpMem, aZOHIfLF, 0x13uLL);         // WRSkyClient.x64.exe
1258    xor_str(hHandle, aZXdDhF0q7ZfC, 0x15uLL); // WRCoreService.x64.exe
1259    xor_str(&Buffer, byte_1400559AD, 0xBuLL); // MsMpEng.exe
1260    xor_str(Luid, byte_1400559B8, 7uLL);       // dsa.exe
1261    xor_str(lpMem, byte_1400559BF, 0xEuLL);    // ds_monitor.exe
1262    xor_str(hHandle, byte_1400559CD, 0xCuLL); // Notifier.exe
1263    xor_str(&Buffer, byte_1400559D9, 0x14uLL);// coreServiceShell.exe
1264    xor_str(hHandle, byte_1400559ED, 0xBuLL); // firefox.exe
1265    xor_str(&Buffer, byte_1400559AD, 0xBuLL); // MsMpEng.exe
1266    xor_str(TokenHandle, byte_1400559F8, 0x16uLL);// EPProtectedService.exe
1267    xor_str(Luid, byte_140055A0E, 0x18uLL);    // EPIntegrationService.exe
1268    xor_str(lpMem, aORrOY0, 0xDuLL);           // bdredline.exe
1269    xor_str(hHandle, aHRuHnBlzfC, 0x15uLL);    // EPSecurityService.exe
1270    xor_str(&Buffer, byte_140055A48, 0x13uLL);// EPUpdateService.exe
1271    xor_str(hHandle, aHVqU4, 0xCuLL);          // ERAAgent.exe
1272    xor_str(&Buffer, byte_140055975, 8uLL);    // ekrn.exe
```

Figure 10. Example of the encrypted, embedded security software process names from one MS4Killer sample

These process names reference processes from multiple security products (see also *Appendix: Example of MS4Killer termination process list*). The code snippet in Figure 10 shows that there are duplicates in the process names (like ekrn.exe), some of the strings are decrypted to the same location (see the variables hHandle, Luid, and lpMem) and there is one dummy process name: firefox.exe. Furthermore, following the cross-references of decrypted string variables leads to comparison logic, where only a subset of process names is utilized. Figure 11 shows a code snippet, where, in that particular case, only process names ERAAgent.exe and ekrn.exe, which are from ESET products, are compared against the running processes. Close inspection of multiple MS4Killer samples shows that, in each intrusion, only processes of a particular security solution are monitored, despite the embedded process list always containing process names from multiple security products.

```
1974        proc_name = *(&v224[-19] + 43 * v233 - 1);// current running process name
1975        proc_len = v224[-19].m128i_i64[43 * v233];// current running process name length
1976        if ( (ERAAgent_len != proc_len || memcmp(ERAAgent_exe, *(&v224[-19] + 43 * v233 - 1), ERAAgent_len))// check ERAAgent.exe
1977          && (ekrn_exe_len != proc_len || memcmp(ekrn_exe, proc_name, ekrn_exe_len)) )// check ekrn.exe
1978        {
1979          goto no_match;
1980        }                                   // else: process name match
```

Figure 11. Decision logic determining which processes are terminated

We saw evidence suggesting that MS4Killer samples were compiled shortly before the actual attacks and targeted only the security solution protecting the victim's machine.

Conclusion

In this blogpost, we have provided an analysis of new Rust tools that we named MDeployer and MS4Killer, which are actively utilized by the new ransomware group – Embargo. Embargo is a new player in the ransomware space, with the ambition to rise to the level of the seasoned gangs. We have provided arguments for why we believe that the Embargo group offers RaaS.

The main purpose of the Embargo toolkit is to secure successful deployment of the ransomware payload by disabling the security solution in the victim's infrastructure. Embargo puts a lot of effort into that, replicating the same functionality at different stages of the attack (BAT script, MDeployer, and MS4Killer all contain security-solution-disabling functionality). We have also observed the attackers' ability to adjust their tools on the fly, during an active intrusion, for a particular security solution.

Both MDeployer and MS4Killer are written in Rust. The same is true for the ransomware payload, suggesting Rust is the go-to language for the group's developers. We have observed deployment of two different versions of MDeployer during one incident. The deployed loader also contained logical bugs that disrupted the proper functionality of the tool. Based on the way the tools are tweaked during intrusions and the closeness of the compilation timestamps to the times of intrusions, we assume that the attacker deploying the tools has the ability to quickly modify the source code and recompile their tools during an intrusion.

> *For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.*
>
> *ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.*

IoCs

Files

SHA-1	Filename	Detection	Description
A1B98B1FBF69AF79E5A3F27AA6256417488CC117	dtest.dll	Win64/Agent.ECY	MDeployer - loader deploying MS4Killer and Embargo ransomware.
F0A25529B0D0AABCE9D72BA46AAF1C78C5B48C31	fxc.exe	Win64/Agent.ECY	MDeployer - loader deploying MS4Killer and Embargo ransomware.
2BA9BF8DD320990119F42F6F68846D8FB14194D6	fdasvc.exe	Win64/Agent.ECY	MDeployer - loader deploying MS4Killer and

SHA-1	Filename	Detection	Description
			Embargo ransomware.
888F27DD2269119CF952 4474A6A0B559D0D201A1	praxisbackup.exe	Win64/Agent.ECW	MS4Killer - Embargo EDR Killer.
BA14C43031411240A083 6BEDF8C8692B54698E05	praxisbackup.exe	Win64/Agent.ECW	MS4Killer - Embargo EDR Killer.
8A85C1399A0E404C8285 A723C4214942A45BBFF9	pay.exe	Win32/Filecoder.Embargo.A	Embargo ransomware.
612EC1D41B2AA2518363 B18381FD89C12315100F	win32.exe	Win32/Filecoder.Embargo.A	Embargo ransomware.
7310D6399683BA3EB2F6 95A2071E0E45891D743B	Sysmon64.sys	Win64/ITMSystem.A	Legitimate vulnerable driver, probmon.sys, dropped and used by MS4Killer.
7310D6399683BA3EB2F6 95A2071E0E45891D743B	Sysprox.sys	Win64/ITMSystem.A	Legitimate vulnerable driver, probmon.sys, dropped and used by MS4Killer.

Certificate

Serial number	010000000001306DE166BE
Thumbprint	A88758892ED21DD1704E5528AD2D8036FEE4102C
Subject CN	ITM System Co.,LTD
Subject O	ITM System Co.,LTD
Subject L	Guro-gu
Subject S	N/A
Subject C	KR
Valid from	2011-06-08 06:01:39

Valid to	2014-06-07 08:32:23

Additional MDeployer file paths

- C:\Windows\Debug\b.cache
- C:\Windows\Debug\a.cache
- C:\Windows\Debug\fail.txt
- C:\Windows\Debug\stop.exe

Commands used by MDeployer

- reg delete HKLM\SYSTEM\CurrentControlSet\Control\Safeboot\Network\WinDefend /f
- C:\Windows\System32\cmd.exe /c takeown /R /A /F "C:\ProgramData\[redacted]" /D Y
- C:\Windows\System32\cmd.exe /c takeown /R /A /F "C:\Program Files\[redacted]" /D Y
- sc create irnagentd binpath="C:\Windows\System32\cmd.exe /c start /B rundll32.exe C:\Windows\Debug\dtest.dll,Open" start=auto
- sc delete irnagentd
- reg add HKLM\SYSTEM\CurrentControlSet\Control\Safeboot\Network\irnagentd /t REG_SZ /d Service /f
- C:\Windows\System32\cmd.exe /c bcdedit /set {default} safeboot Minimal
- C:\Windows\System32\cmd.exe /c bcdedit /deletevalue {default} safeboot
- reg delete HKLM\SYSTEM\CurrentControlSet\Control\Safeboot\Network\WinDefend /f
- C:\Windows\System32\cmd.exe /c ping localhost -n 5 > nul & del C:\Windows\Debug\dtest.dll
- shutdown -r -f -t 00
- C:\Windows\praxisbackup.exe
- C:\Windows\Debug\pay.exe

MITRE ATT&CK techniques

This table was built using version 15 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1587.001	Develop Capabilities: Malware	Embargo group develops its custom toolkit – MDeployer, MS4Killer, and Embargo ransomware.
Execution	T1059.003	Command-Line Interface: Windows Command Shell	Embargo group executes a BAT script that disables security solutions.
	T1059.001	Command-Line Interface: PowerShell	Embargo group uses PowerShell to transfer MDeployer to victims' machines.

Tactic	ID	Name	Description
	T1053.005	Scheduled Task/Job: Scheduled Task	Embargo group uses scheduled tasks to run MDeployer on compromised endpoints.
	T1569.002	System Services: Service Execution	Embargo group uses a Windows service to execute MDeployer in Safe Mode.
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Embargo group modifies the Windows registry to start a custom service in Safe Mode.
	T1136.002	Create Account: Domain Account	Embargo group creates its own domain accounts.
Defense Evasion	T1562.001	Impair Defenses: Disable or Modify Tools	MDeployer, MS4Killer, and a BAT script disable security solutions.
	T1562.009	Impair Defenses: Safe Mode Boot	MDeployer and a BAT script reboot into Safe Mode.
	T1070.004	Indicator Removal: File Deletion	MDeployer deletes dropped files during cleanup.
	T1112	Modify Registry	MS4Killer modifies the registry to load a legitimate vulnerable driver.
	T1027.013	Obfuscated Files or Information: Encrypted/Encoded File	Payloads loaded by MDeployer are RC4 encrypted.
Discovery	T1135	Network Share Discovery	Embargo ransomware performs network share discovery.
	T1083	File and Directory Discovery	Embargo ransomware performs file and directory discovery.
Impact	T1490	Inhibit System Recovery	Embargo ransomware disables automatic Windows recovery.
	T1486	Data Encrypted for Impact	Embargo ransomware encrypts files on compromised machines.

Appendix: Example of MS4Killer termination process list (in alphabetical order)

SentinelAgent.exe SentinelAgentWorker.exe SentinelServiceHost.exe SentinelStaticEngine.exe LogProcessorService.exe SentinelStaticEngineScanner.exe SentinelHelperService.exe SentinelBrowserNativeHost.exe LogCollector.exe SentinelMemoryScanner.exe SentinelRanger.exe SentinelRemediation.exe SentinelRemoteShellHost.exe SentinelScanFromContextMenu.exe CylanceSvc.exe ekrn.exe WRSA.exe	WRSkyClient.x64.exe WRCoreService.x64.exe MsMpEng.exe dsa.exe ds_monitor.exe Notifier.exe coreServiceShell.exe firefox.exe MsMpEng.exe EPProtectedService.exe EPIntegrationService.exe bdredline.exe EPSecurityService.exe EPUpdateService.exe ERAAgent.exe ekrn.exe

Source: https://www.welivesecurity.com/en/eset-research/embargo-ransomware-rocknrust/