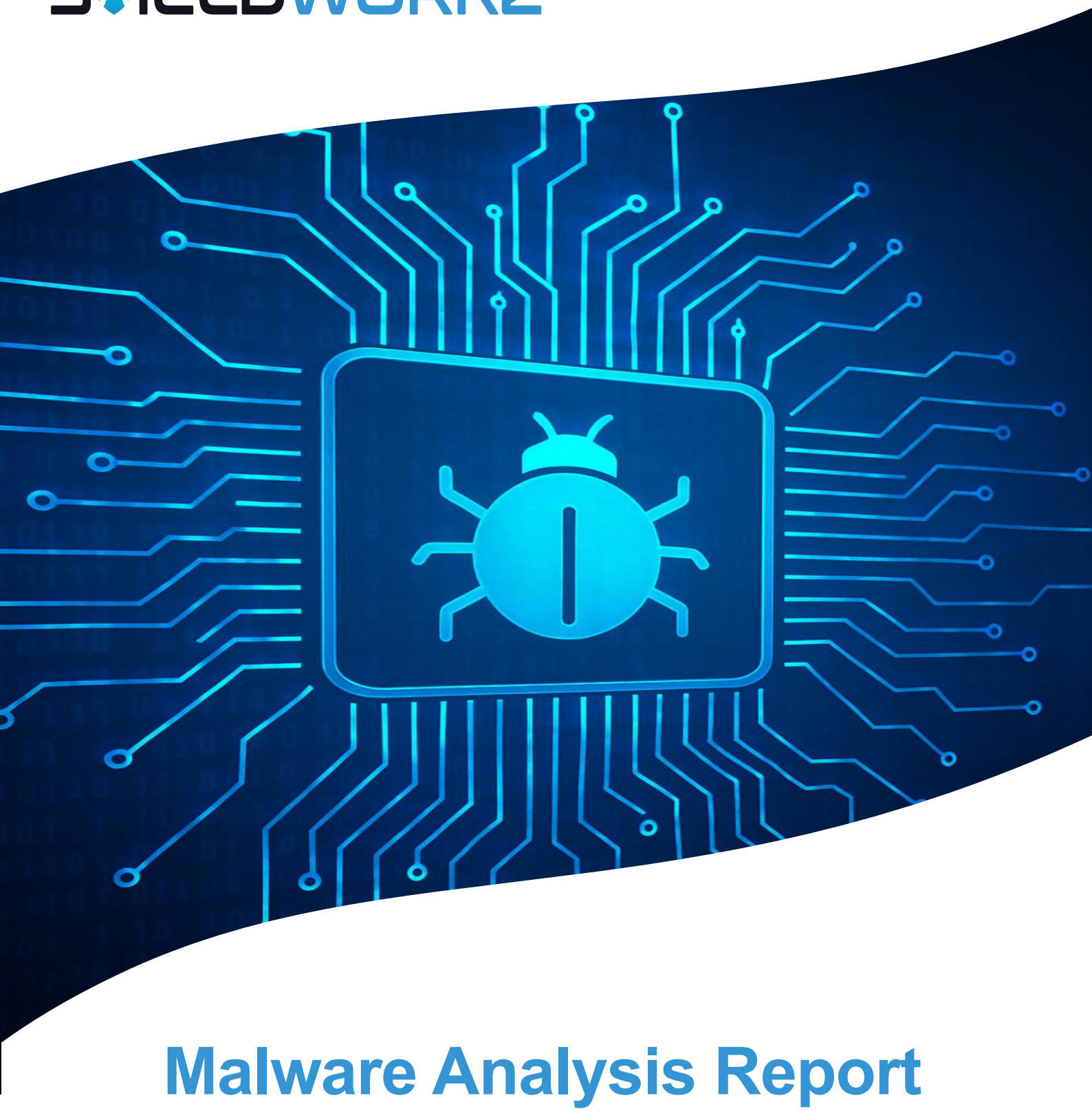
SHIELDWORKZ



Malware Analysis Report
FROSTYGOOP

Report Date: August 2025

Document Properties

Document Title	Malware Analysis Report – FROSTYGOOP			
Document Version	1.0			
Author	Uday M, Shan Chokshi			
Reviewed	Supreet A Joshi			
Published	August 2025			
No of Pages	21			
Classification	☒Public	☐Internal Only	☐Confidential	☐Restricted
Document Type	☐Draft		☒Release	

Contents

Figures

Tables

1. Introduction

This report presents a detailed technical investigation into **FrostyGoop**, an emerging malware strain designed to target industrial control systems (ICS) through direct interaction with the **Modbus TCP protocol**. This malware was initially detected in late 2024 and observed continuing into early 2025. Despite early identification and public reporting, malware remains active in the wild.

This demonstrates an evolution in threat actor capability, focusing not on IT-centric disruptions but on operational technology (OT) environments that remain vulnerable due to legacy communication protocols and inadequate segmentation. By manipulating protocol-level behavior rather than relying on traditional malware installation across critical assets, FrostyGoop exemplifies a new wave of ICS-specific attacks that blend simplicity, effectiveness, and stealth.

2. Report Scope

This analysis encompasses multiple layers to provide complete visibility into FrostyGoop's attack lifecycle:

1. **Static Analysis** – Disassembling and inspecting the Go-compiled binary and its embedded JSON-based task configuration.
2. **Dynamic/Behavioral Analysis** – Executing the malware in a sandboxed OT network simulation and tracking process-level actions.
3. **Network/Protocol Analysis** – Capturing Modbus TCP traffic on port 502 to identify register manipulations and command sequences.
4. **Reverse Engineering** – Parsing encryption routines and internal logic flows using disassemblers like Binary Ninja.

3. Executive Summary

A newly identified malware variant, internally referenced as FrostyGoop, is a targeted threat to operational technology (OT) environments. The malware is specifically designed to exploit the unprotected nature of the Modbus TCP protocol, allowing it to directly interact with industrial control systems (ICS) and manipulate device states at the register level.

FrostyGoop, also identified as BUSTLEBERM, is a notable piece of malware that came to public attention following a cyberattack on industrial control systems (ICS). Discovered in early 2024, it is significant because it is one of the few known malware variants specifically engineered to disrupt operational technology (OT) by exploiting the widely used Modbus TCP communication protocol.

Technical Architecture and Functions

ICS/OT Targeting: FrostyGoop is a binary file for Windows, written in the Go programming language (Golang). Its primary function is to interact with ICS devices that communicate using the Modbus TCP protocol on port 502.

- Modbus Manipulation: The malware can both read from and write to the holding registers on targeted devices. By altering the values in these registers, attackers can change system configurations, monitor process statuses, and ultimately interfere with the physical operations managed by the equipment.
- Execution Methods: The malware can be run from the command line with specific parameters or by referencing a JSON configuration file. This flexibility allows attackers to specify target IP addresses and the exact Modbus commands they wish to execute.
- Simplicity and Impact: Despite not being a highly complex piece of code, FrostyGoop's true danger lies in its direct capability to cause physical damage to critical infrastructure. The straightforward nature of its design, which leverages readily available open-source libraries, serves as a clear example of how basic tools can be used to cause major disruption.

Real-World Consequences

FrostyGoop was reportedly used in an attack that led to a two-day heating failure for over 600 residential buildings in Ukraine during severe winter conditions. This incident underscores the serious, real-world consequences of cyberattacks on critical infrastructure. The attack is believed to have started with an initial breach of a vulnerable router, followed by lateral movement within the network to gain control over the ICS environment.

3.1 Key Findings

The following outlines the key findings of the report:

- **Type:** ICS disruptive malware compiled in Go, targeting Modbus-enabled control devices.
- **Infection Vector:** Likely introduced via vulnerable perimeter hardware or internet-exposed OT endpoints.
- **Risk Level**: **High** - Particularly in environments reliant on legacy industrial protocols without authentication or encryption.

4. Potential Impact

By issuing unauthorized Modbus read and write instructions over TCP port 502, the malware can alter critical device parameters, such as sensor values and actuator commands. This capability poses a direct threat to system stability and physical infrastructure, potentially causing operational degradation, safety concerns, or loss of visibility during sensitive operational periods.

4.1 Detection & Mitigation Recommendations

- Implement strong network segmentation and eliminate direct external access to control systems.
- Utilize intrusion detection mechanisms that support industrial protocol analysis to flag abnormal traffic patterns.
- Enforce strict access policies for engineering workstations and gateway devices.
- Harden perimeter assets by disabling unnecessary services, applying security patches and securing remote access configurations.
- Adopt OT-specific monitoring strategies to detect protocol-level deviations in real time.

5. Tools and Environment

To emulate a realistic OT environment and derive actionable insights, the hybrid toolset was used:

1. Binary Ninja, Cutter, DiE – for unpacking, decompiling, and evaluating Go-compiled code.
2. Wireshark – to monitor and verify unauthorized Modbus TCP write requests targeting ENCO controllers.
3. Process Monitor and sandbox environments – to trace file and console output behaviour.
4. Custom Python scripts – to extract AES keys and validate encrypted JSON tasks.

6. Context & Real-World Impact

FrostyGoop exemplifies a form of **"exploit-less"** attack targeting OT networks by directly abusing native industrial protocols. Rather than installing malware on critical endpoints, the attacker issues unauthorized Modbus instructions from an already compromised edge asset or proxy host—avoiding detection by conventional IT defence layers.

In one documented 2025 incident, this behaviour resulted in operational failure across a district energy management system, where unauthorized Modbus commands disrupted setpoints and schedules responsible for environmental control. The attack lasted multiple hours, impacting service continuity and prompting emergency manual overrides.

Initial access was likely gained through a misconfigured or outdated perimeter gateway device. Once inside the flat OT network, the attacker issued register-level commands to critical control components using default or unauthenticated access paths.

7. Malware Overview

7.1 Malware Family & Name

FrostyGoop is a custom-developed ICS-targeted malware designed to exploit the **Modbus TCP protocol**. Compiled in Go, the malware communicates directly with Modbus-enabled field devices over **TCP port 502**. Its design focuses on precision task execution rather than persistence or infection spread. The tool's modular configuration—controlled via encrypted JSON task files—enables remote operators to issue register-level commands that can directly impact industrial operations. FrostyGoop operates without the need to exploit system vulnerabilities or inject itself into persistent runtime environments.

7.2 Technical Description

The malware is built using the Go programming language and integrates community-sourced libraries such as rolfl/modbus and hsblhsn/queues to construct and transmit valid Modbus commands. Supported operations include:

- 0x03 – Read Holding Registers
- 0x06 – Write Single Holding Register
- 0x10 – Write Multiple Holding Registers

FrostyGoop is configured using a JSON task file or CLI arguments, defining execution flow, target device IPs, register addresses, and operational intervals. Upon launch, the malware establishes socket connections to listed devices and executes the specified Modbus commands in sequence. It logs all interactions, including timestamps, command status, and Modbus exception codes. Once complete, the process self-terminates, leaving no resident payload or installation artifacts.

To reduce traceability, FrostyGoop features anti-debugging routines by inspecting the BeingDebugged flag at runtime. Configuration files may be optionally encrypted using a standalone utility (go-encrypt.exe) to obscure task content and bypass basic signature detection or plaintext inspection.

7.3 First Detection

FrostyGoop was identified in mid-2025 during incident response activities involving unexpected protocol-level behavior in an industrial control environment. The initial sample was discovered on a compromised engineering workstation issuing unscheduled Modbus write instructions to critical process devices. Subsequent analysis confirmed it to be part of a new malware family capable of executing remote register manipulation without requiring lateral movement or elevated privileges.

7.4 Classification

FrostyGoop is better described as an **"OT attack tool"** rather than conventional malware such as ransomware or keyloggers. It is designed for operational impact, not financial gain.

- Category: ICS-targeted malware (protocol-level manipulation).
- Tactics: Control disruption through automated Modbus register writes.
- Persistence: None — the malware performs one-shot execution and exits.
- Footprint: Minimal — no registry edits, scheduled tasks, or lateral movement components.

7.5 Attribution & Threat Actors

Although no widely recognized attribution to a specific threat group exists, public analysis links the Lviv incident to cyber operations aligned with Russian interests, suggesting it was part of a broader strategy of conflict-driven sabotage. Independent assessments indicated that while attribution to known actors was absent, the activity likely occurred within the geopolitical landscape of Russia's ongoing campaign targeting Ukrainian critical infrastructure.

8. Malware Sample Information

SHA256 Hash	File Size	File Type	File Description
5d2e4fd08f81e3b2eb2f3eaae16eb32ae02e760afc36fa17f4649322f6da53fb	3.7 MB (3,699,200 bytes)	PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows	Windows executable file for FrostyGoop malware
a63ba88ad869085f1625729708ba65e87f5b37d7be9153b3db1a1b0e3fed309c	2.4 MB (2,439,680 bytes)	PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows	Windows executable file for FrostyGoop malware
2fd9cb69ef30c0d00a61851b2d96350a9be68c7f1f25a31f896082cfbf39559a	3.4 MB (3,359,232 bytes)	PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows	Windows executable file for FrostyGoop malware
c64b67c116044708e282d0d1a8caea2360270a7fc679befa5e28d1ca15f6714c	2.0 MB (1,951,232 bytes)	PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows	Windows executable file for FrostyGoop malware
91062ed8cc5d92a3235936fb93c1e9181b901ce6fb9d4100cc01167cdc08745f	2.5 MB (2,516,480 bytes)	PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows	Windows executable file for FrostyGoop malware
a25f91b6133cb4eb3ecb3e0598bbab16b80baa40059e623e387a6b1082d6f575	2.5 MB (2,515,968 bytes)	PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows	Windows executable file for FrostyGoop malware
9cf30d82a86a9485f7bbd0786a5de207cf4902691a3efcfc966248cb1e87d5b7	1.8 MB (1,773,568 bytes)	PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows	Executable for go-encrypt.exe, likely used during FrostyGoop configuration stages.
06919e6651820eb7f783cea8f5bc78184f3d437bc9c6cde9bfbe1e38e5c73160	0.4 KB (379 bytes)	JSON text data	task-test.json — likely used to test encrypted task scheduling in FrostyGoop

Table 1: Malware Samples

9. Infection Vector

The FrostyGoop malware campaign employed a multi-phase intrusion methodology that focused less on software vulnerabilities and more on systemic weaknesses in ICS network architecture. Rather than relying on conventional malware exploits or code injections, the operators abused protocol trust and segmentation gaps to establish control within OT environments.

- **Initial Access via Perimeter Exposure:** The attackers gained entry through an externally accessible network device with outdated firmware and weak authentication controls. Once access was achieved, a lightweight web-accessible shell was deployed, enabling persistent command execution on an IT/OT boundary system. This provided a long-term foothold in the environment.

- **Credential Harvesting & Persistence:** Upon initial access, the attackers collected system artifacts such as local password hashes and registry hives to extract valid credentials. These credentials were used to authenticate against additional internal systems and maintain stealthy access over extended periods.

- **Lateral Movement via Encrypted Tunneling:** With valid credentials in hand, the threat actors established a remote encrypted tunnel (e.g., VPN over L2TP/IPSec or similar protocols) to maintain access to the internal network. This tunneling facilitated covert command and control without triggering basic anomaly detection. The attackers moved laterally into adjacent ICS segments with minimal resistance due to weak segmentation and a lack of active network monitoring.

- **Direct ICS Protocol Abuse:** Unlike typical malware that exploits software vulnerabilities (CVEs), FrostyGoop required no exploit to execute its objectives. Instead, it communicated directly with Modbus-enabled devices using unauthenticated function codes, including:
    - 0x03 – Read Holding Registers
    - 0x06 – Write Single Holding Register
    - 0x10 – Write Multiple Holding Registers

These commands were transmitted over TCP port 502, allowing real-time manipulation of industrial processes, such as altering control setpoints or disrupting operational timing—all without deploying code on the control devices themselves.

- **Configuration-Driven Attack Logic:** FrostyGoop's execution was governed by external JSON configuration files, which defined key task parameters, including:
    - Target IP addresses
    - Register types (holding, input, etc.)
    - Operation intervals
    - Task duration and repeat cycles

This modular design enabled precision attacks on multiple devices in parallel, with minimal interaction required from the attacker post-deployment.

- **OT-Specific Security Concerns**
    1. This malware bypasses traditional exploit-based detection by abusing protocol-level trust, especially in systems that use legacy ICS protocols like Modbus TCP without authentication.
    2. Monitor for encrypted tunneling activity, especially outbound VPN sessions from non-standard endpoints or geographic anomalies.
    3. Block direct internet access to ICS ports such as 502 (Modbus) and 44818 (ENIP) through robust segmentation and firewall policies.

4. Deploy deep packet inspection (DPI) and ICS-aware intrusion detection systems (IDS/IPS) to detect suspicious or repetitive Modbus function code usage.
5. Eliminate unnecessary remote access paths and harden perimeter infrastructure by applying firmware updates and disabling legacy management interfaces.

10. Technical Analysis

10.1 Static Analysis

The following details are based on static analysis of the Frostygoop malware:

Figure 1 shows Tasks and IP list under the register for main::main.TaskList___runtime.structtype_fields.

2643	00125e45	00526e45	Section(1)...	0a	A	*main.Task
2798	00126589	00527589	Section(1)...	0a	A	taskWorker
2996	00126f89	00527f89	Section(1)...	0c	A	*[]main.Task
3220	00127c2d	00528c2d	Section(1)...	0d	A	getTaskIpList
3253	00128072	00529072	Section(1)...	0e	A	*[][]main.Task
3266	00128142	00529142	Section(1)...	0e	A	*main.TaskList
3673	00129d8f	0052ad8f	Section(1)...	05	A	Tasks
3674	00129d95	0052ad95	Section(1)...	0c	A	json:"Tasks"
4176	00148e2b	00549e2b	Section(1)...	a5	A	*struct { F uintptr; q *queues.Q; status *main.StatusConf...
4211	0015fca7	00560ca7	Section(1)...	e8	A	dllexitThreadfloat32nanfloat64nangetsockoptgoroutine ...
4278	0016358c	0056458c	Section(1)...	0100	A	cycle info=[FILE.json]duplicated defer entryfreeIndex is...
8369	001b16f9	005b26f9	Section(1)...	1b	A	main.TaskList.getTaskIpList
8370	001b1715	005b2715	Section(1)...	17	A	main.TaskList.getIpList
8371	001b172d	005b272d	Section(1)...	14	A	main.Task.taskWorker
8377	001b17a5	005b27a5	Section(1)...	1c	A	main.TaskList.executeCommand
8388	001b1856	005b2856	Section(1)...	13	A	main.TaskList.getIp
19766	0037fad2	007dfad2	Section(12)...	1b	A	main.TaskList.getTaskIpList
19767	0037faee	007dfaee	Section(12)...	17	A	main.TaskList.getIpList
19768	0037fb06	007dfb06	Section(12)...	14	A	main.Task.taskWorker
19773	0037fb6a	007dfb6a	Section(12)...	1c	A	main.TaskList.executeCommand

Figure 1: DiE showing frostyGoop operations and register usage.

The figure below shows an example of an operation for Code, Address, Count, Value and State under the register for main::main.TaskList___runtime.structtype_fields through which it tries to modify the censor values accordingly.

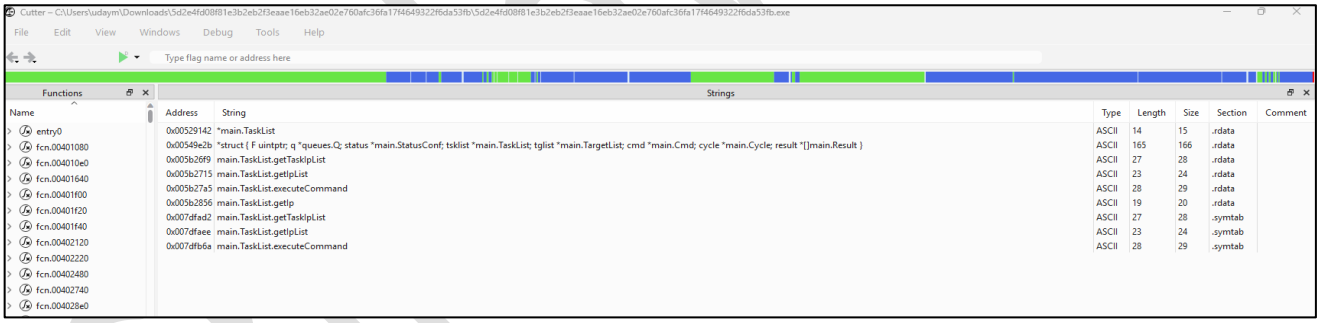


Figure 2: Cutter showing FrostyGoop operations for Code, Address, Count, Value and State.

Figure 3 shows the timing configuration for main.Cycle.getCycleConfig.

```
0x005b2664          .string "main.Cycle.getCycleConfig" ; len=26
;-- str.io_ioutil.ReadFile:
0x005b267e          .string "io/ioutil.ReadFile" ; len=19
;-- str.fmt.Println:
0x005b2691          .string "fmt.Println" ; len=12
;-- str.time.Time.Month:
0x005b269d          .string "time.Time.Month" ; len=16
;-- str.time.Time.Day:
0x005b26ad          .string "time.Time.Day" ; len=14
;-- str.time.Time.Hour:
0x005b26bb          .string "time.Time.Hour" ; len=15
;-- str.time.Time.Minute:
0x005b26ca          .string "time.Time.Minute" ; len=17
;-- str.time.Time.Second:
0x005b26db          .string "time.Time.Second" ; len=17
;-- str.main.subTime:
0x005b26ec          .string "main.subTime" ; len=13
0x005b26f9          .string "main.TaskList.getTaskIpList" ; len=28
0x005b2715          .string "main.TaskList.getIpList" ; len=24
0x005b272d          .string "main.Task.taskWorker" ; len=21
;-- str.main.makeRegsResult:
0x005b2742          .string "main.makeRegsResult" ; len=20
0x005b2756          .string "main.TargetList.getTargetIpList" ; len=32
0x005b2776          .string "main.defMarshal" ; len=16
0x005b2786          .string "main.defUnmarshal" ; len=18
0x005b2798          .string "main.process" ; len=13
0x005b27a5          .string "main.TaskList.executeCommand" ; len=29
0x005b27c2          .string "main.routine" ; len=13
0x005b27cf          .string "main.routine.func1" ; len=19
0x005b27e2          .string "main.main" ; len=10
```

Figure 3: Cutter showing FrostyGoop timing configuration in the registry entry

FrostyGoop utilizes Goccy's go-json, an optimized JSON encoding/decoding library designed as a high-performance alternative to Go's standard encoding/json package. It also integrates a lesser-known open-source task manager called queues, which orchestrates execution flow. Due to the limited visibility of this library in common malware, its presence can act as a distinguishing marker for identifying FrostyGoop activity.

Figure 4 shows our analysis of a Windows executable file for FrostyGoop within the tool DiE. This analysis reveals URLs from open-source libraries for Modbus, go-json and queues.

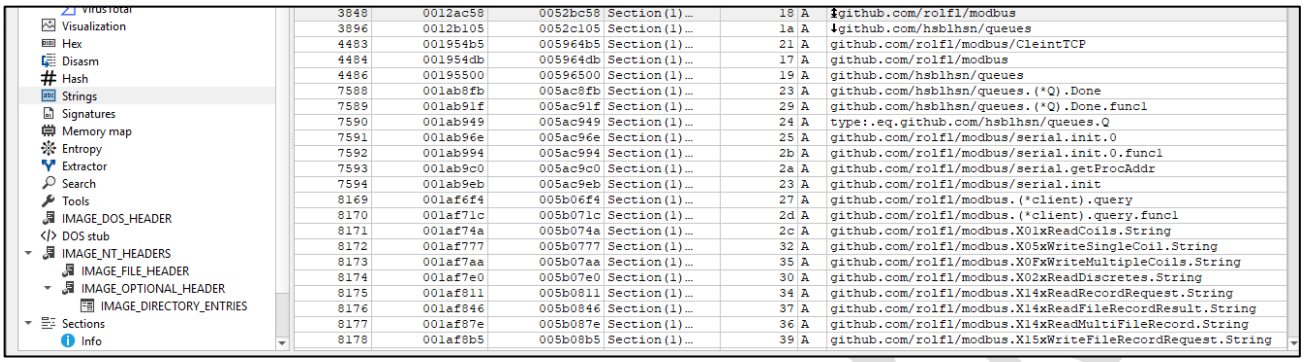


Figure 4: Open-source libraries: Modbus, go-json and queues.

- Static string data reveals embedded timezone references and error messages — indicators of reconnaissance or evasion logic.
- Found in .text section: possible staging logic involving malformed Go runtime strings (bucket.mpchan, nil channel) to disrupt targets.

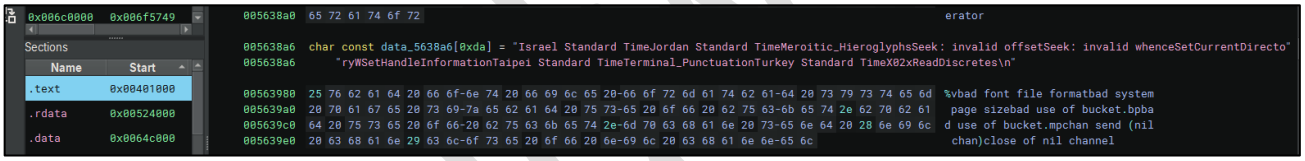


Figure 5: Disassembled code from a FrostyGoop sample showing a malinformed Go runtime strings.

Figure 6 shows the content of an encrypted JSON file generated by go-encrypt.exe.

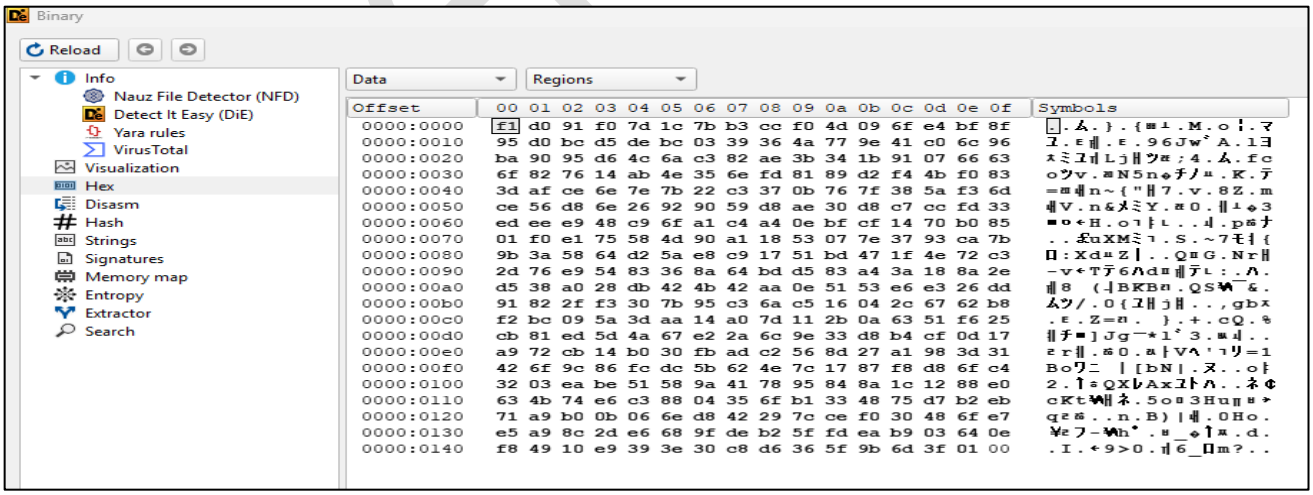


Figure 6: An encrypted JSON file viewed in a hex editor.

10.2 Dynamic Analysis (Behavioral)

The following details are based on the Dynamic analysis of the Frostygoop malware:

10.2.1 Go-encrypt.exe Sample Analysis

The investigation revealed a Windows executable sample named go-encrypt.exe written in Go that was not FrostyGoop, but it originally appeared on the same approximate date that other indicators of FrostyGoop were reported. Command-line options for this software reveal that the file is used to encrypt and decrypt JSON files, as illustrated in **Figure 7**.

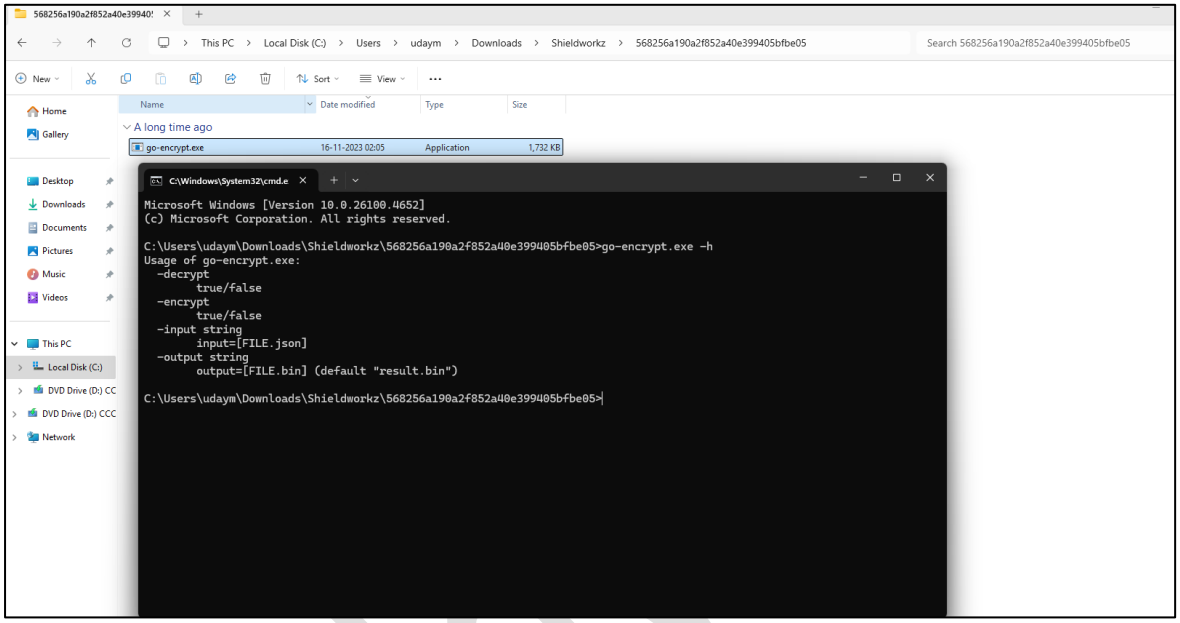


Figure 7: Command-line options for go-encrypt.exe.

After executing go-encrypt.exe using the -encrypt argument, it creates two files:

- An encrypted json file
- A 32-byte file containing a decryption key named key

Figure 8 shows the encryption, decryption, and the generated key.

```
C:\Users\udaym\Downloads\Shieldworkz\568256a190a2f852a40e399405bfbe05>go-encrypt.exe -input task_test.json -output x.bin -encrypt true
2025/07/27 15:35:16 [runtime.main:proc.go:250][INFO] Key: %x
 [101 88 56 79 104 117 97 68 71 121 50 116 57 78 87 103 110 109 88 90 82 77 109 86 87 117 117 72 55 81 110 103]

C:\Users\udaym\Downloads\Shieldworkz\568256a190a2f852a40e399405bfbe05>go-encrypt.exe -input x.bin -output clean.json -decrypt true
2025/07/27 15:35:26 [runtime.main:proc.go:250][INFO] Key: %x
 [101 88 56 79 104 117 97 68 71 121 50 116 57 78 87 103 110 109 88 90 82 77 109 86 87 117 117 72 55 81 110 103]
Decrypted file was created with file permissions 0777

C:\Users\udaym\Downloads\Shieldworkz\568256a190a2f852a40e399405bfbe05>dir
 Volume in drive C has no label.
 Volume Serial Number is 9E7D-0980

 Directory of C:\Users\udaym\Downloads\Shieldworkz\568256a190a2f852a40e399405bfbe05

27-07-2025  15:35    <DIR>          .
27-07-2025  11:00    <DIR>          ..
22-07-2025  10:07         3,699,200 5d2e4fd08f81e3b2eb2f3eaae16eb32ae02e760afc36fa17f4649322f6da53fb.exe
27-07-2025  15:20         3,699,200 5d2e4fd08f81e3b2eb2f3eaae16eb32ae02e760afc36fa17f4649322f6da53fb.exe.00_00387200.exe
27-07-2025  15:35               240 clean.json
16-11-2023  02:05         1,773,568 go-encrypt.exe
27-07-2025  15:35                32 key
27-07-2025  15:20               240 task_test.json
27-07-2025  15:35               336 x.bin
               7 File(s)      9,172,816 bytes
               2 Dir(s)  37,447,303,168 bytes free

C:\Users\udaym\Downloads\Shieldworkz\568256a190a2f852a40e399405bfbe05>
```

Figure 8: Using go-encrypt.exe to encrypt and decrypt a JSON file.

Figure 9 shows a filtered list of processes generated by go-encrypt.exe in Process Monitor. We have highlighted when go-encrypt.exe created the decryption file named key and the 32-character content of this key file.

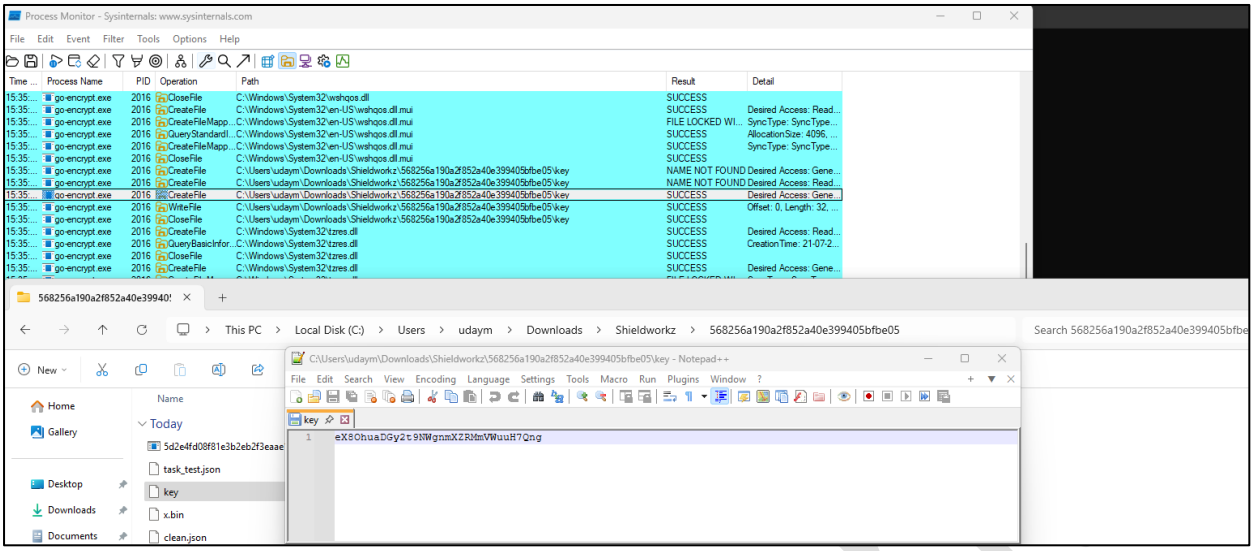


Figure 9: Process Monitor showing go-encrypt.exe generating the key file.

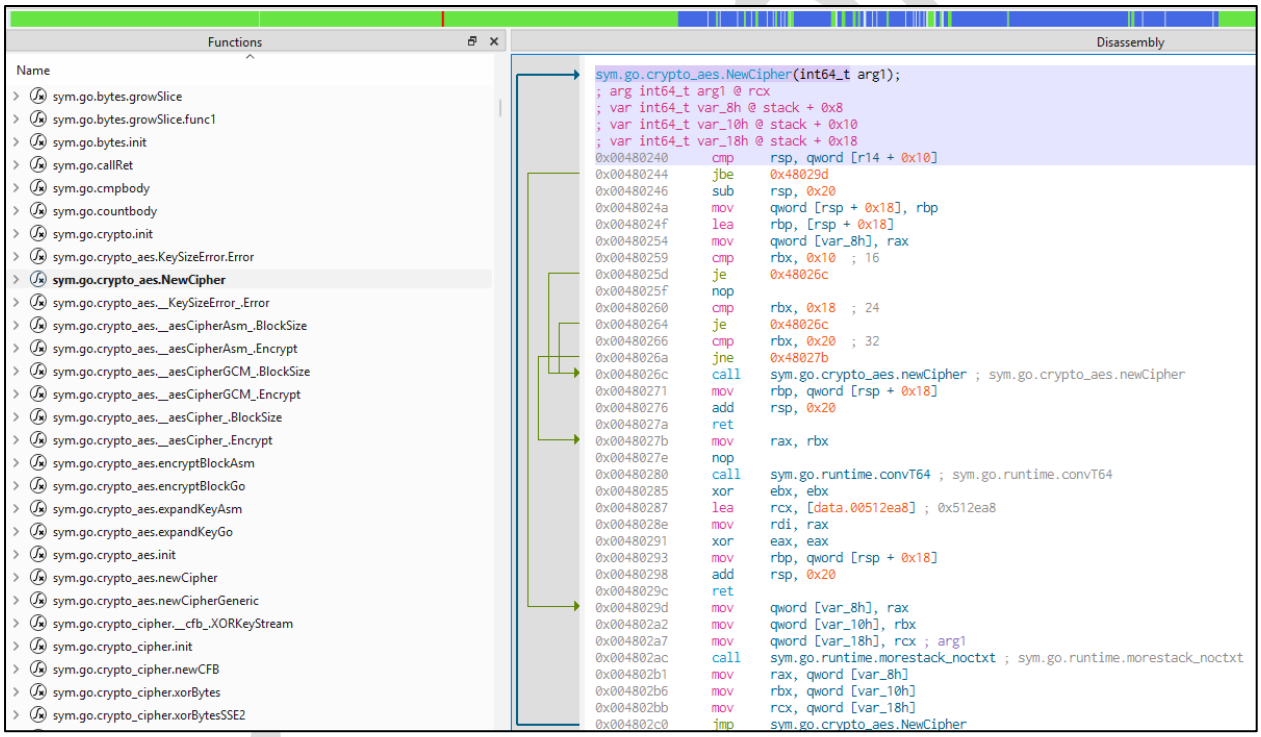


Figure 10: Decompiled code of go-encrypt.exe showing its AES main encryption routine.

Decompiling go-encrypt.exe uses the AES encryption algorithm to create the encryption/decryption key in the key file, as shown in **Figure 10**(above) and **Figure 11**(below).

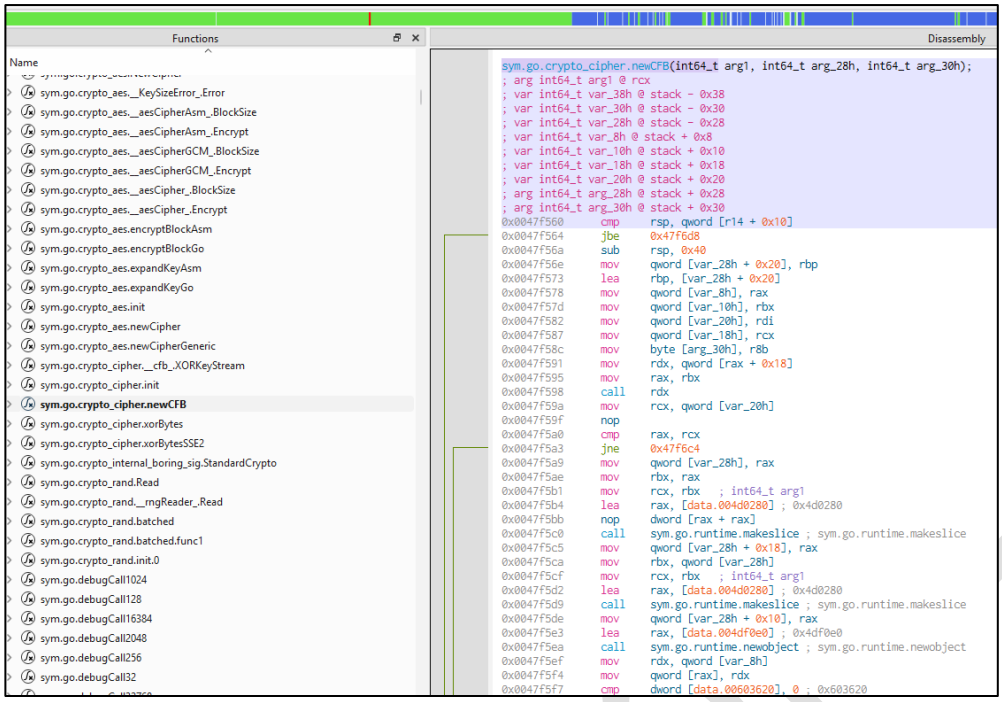


Figure 11: Decompiled code of go-encrypt.exe showing AES algorithm usage for the encryption.

As shown previously for the key generated in Figure 7, the key value is in decimal format. The decimal value of the key from **Figure 7** is:

```
101 88 56 79 104 117 97 68 71 121 50 116 57 78 87 103 110 109 88 90 82 77 109 86 87 117 117 72 55 81 110 103
```

The 32-byte value of the key in hexadecimal is:

```
65 58 38 4F 68 75 61 44 47 79 32 74 39 4E 57 67 6E 6D 58 5A 52 4D 6D 56 57 75 75 48 37 51 6E 67
```

We can retrieve binary data or ASCII data to or from binary data. Similarly, confirmed that the 32-byte hexadecimal value of the key from our example matches the ASCII value in the key file using CyberChef, as shown below in **Figure 12**.

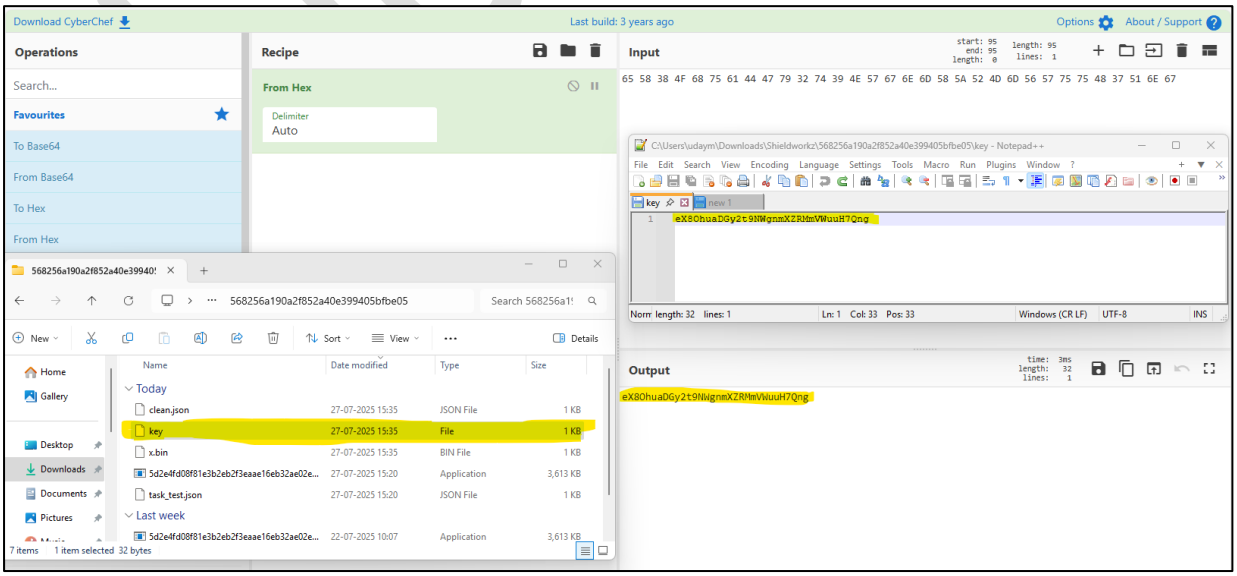


Figure 12: Using CyberChef to verify the hex value matches the ASCII value of the key file.

10.3 Network Analysis

To analyze FrostyGoop traffic, have tested two samples, The two FrostyGoop samples have the following SHA256 hashes:

```
5d2e4fd08f81e3b2eb2f3eaae16eb32ae02e760afc36fa17f4649322f6da53fb

a63ba88ad869085f1625729708ba65e87f5b37d7be9153b3db1a1b0e3fed309c
```

The task_test.json configuration file specifies only function code 3, which corresponds to the Modbus **"Read Holding Registers"** operation. As a result, all observed FrostyGoop samples exclusively issued read requests targeting holding registers on devices accessible via TCP port 502.

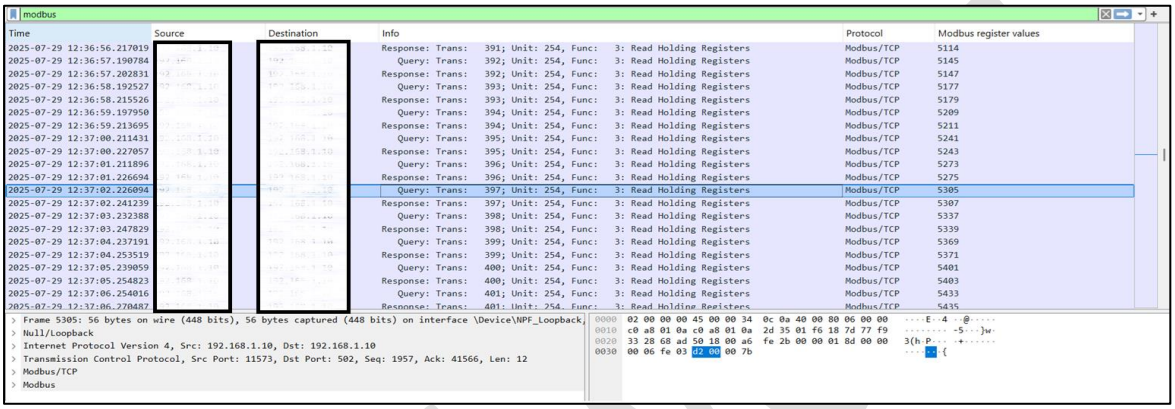


Figure 13: Example of Modbus traffic from our FrostyGoop sample test filtered in Wireshark.

Figure 13 shows an example of the Modbus traffic generated during our test of the FrostyGoop samples, filtered in Wireshark with a customized column display. It reveals Modbus traffic to over TCP port 502, as well as the four register values specified in the task_test.json configuration file:

5275	5401	5339	5403

Figure 14 shows an example of a Modbus function code 3 request to read values from the holding registers of the ENCO device, starting with the register number 53760 for the next 123 registers.

The device responded with values from registry 5275-5403. These registry entries hold UINT16 values for unsigned integers that can range from 0-65535.

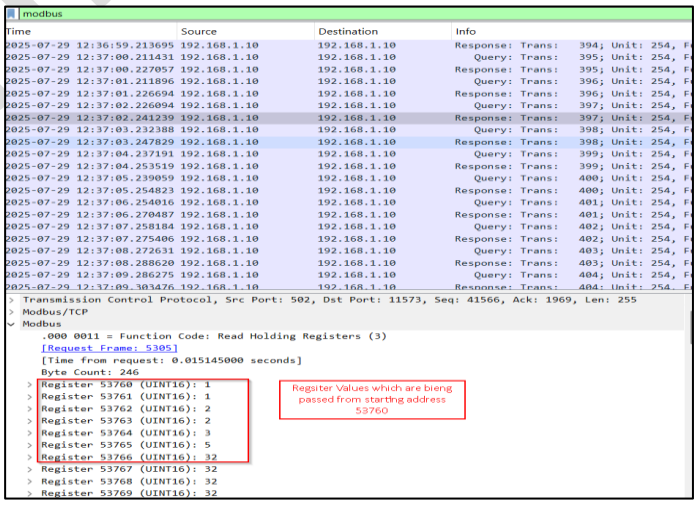


Figure 14: Modbus interaction with the hardware device.

If the JSON configuration file contains the number 1 as a word count value, only one register is returned. If it does not contain the number 1 as a word count value, more than one register is returned.

Figure 15 shows a code snippet from a **FrostyGoop** sample with the logic to select Modbus operations depending on the value provided:

- 3 for read holding registers.
- 6 for write a single holding register.
- 16 for writing multiple holding registers operation.

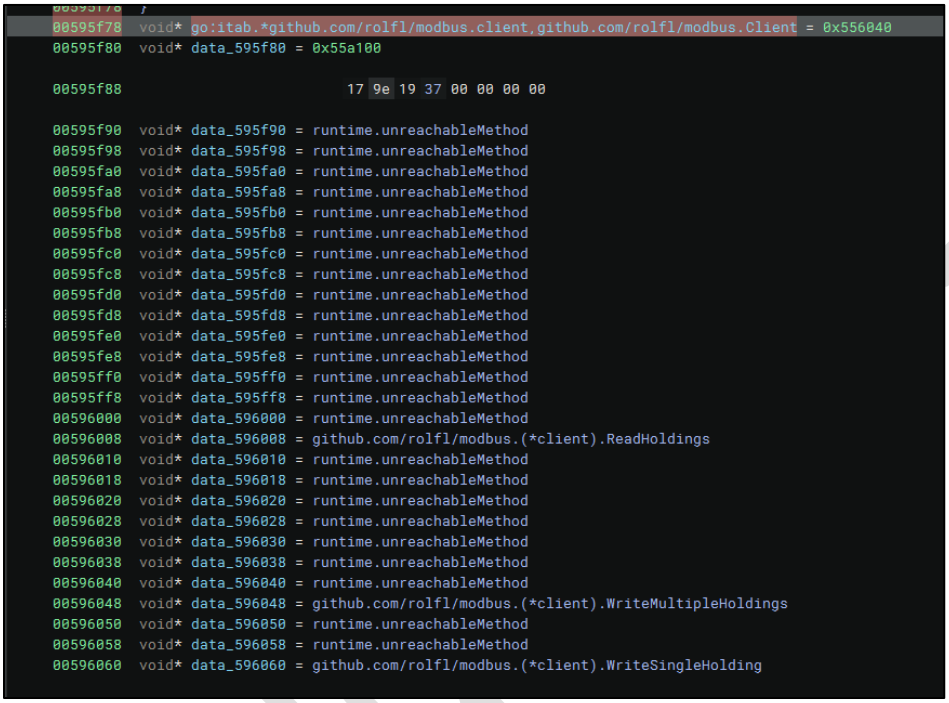


Figure 15: Code snippet from a FrostyGoop sample showing how it implements.

11. Indicator of Compromise (IOCs)

File Hashes (SHA-256):

- **5d2e4fd08f81e3b2eb2f3eaae16eb32ae02e760afc36fa17f4649322f6da53fb** – Windows Go-compiled FrostyGoop/BUSTLEBERM variant 1.
- **a63ba88ad869085f1625729708ba65e87f5b37d7be9153b3db1a1b0e3fed309c** – Variation with extra Go sections.

Configuration & Support Tool Filenames:

- **06919e6651820eb7f783cea8f5bc78184f3d437bc9c6cde9bfbe1e38e5c73160** -- task_test.json – JSON task file defining target lists, Modbus function codes, and scheduling.
- **9cf30d82a86a9485f7bbd0786a5de207cf4902691a3efcfc966248cb1e87d5b7** -- go-encrypt.exe – AES-CFB encryption utility for JSON files, likely used alongside FrostyGoop.

Behavioral and Network Artifacts:

- Modbus TCP traffic (port 502) with function codes 0x03 (read), 0x06 (write single), 0x10 (write multiple) targeting ENCO controllers.
- JSON patterns containing fields like Code, Count, TargetList, StartTime, IntervalTime, and WorkTime.

OT-Protocol Anomalies & Encrypted Task Activity:

- Unencrypted Modbus interactions if the internal environment is compromised, encrypted JSON execution when task files are externally staged.

C2 & Infrastructure Artefacts (Reported but not fully public):

- VPN tunnels (L2TP) from external servers—likely Moscow-hosted—to internal OT networks during January 2024.
- Command and control via exposed MikroTik routers using web shells like ReGeorg or custom scripts.

12. MITRE ATT&CK Mapping

FrostyGoop aligns with multiple MITRE ATT&CK techniques across both Enterprise and ICS frameworks. This mapping reflects its use of protocol abuse, lateral movement, and OT-specific disruption tactics.

Enterprise Techniques

ATT&CK Tactic	ATT&CK Technique	Description
Initial Access	T1190 – Exploit Public-Facing Application	Likely exploited a vulnerability in an externally facing MikroTik router to gain initial network access.
Credential Access	T1003.002 – OS Credential Dumping: SAM	Exfiltrated the Windows SAM registry hive to harvest credentials.
Defense Evasion	T1562.010 – Impair Defences: Firmware Downgrade	Implemented firmware rollback to disable device logging—likely aiding in activity concealment.
Persistence	T1505.003 – Server Software Component: Web Shell	Installed a ReGeorg web shell to maintain access via the compromised router.
Command and control	T1071 – Application Layer Protocol	Used L2TP VPN from external IPs (e.g., Moscow servers) for C2 connectivity.

Table 2: Mitre Att&ck mapping (Enterprise Techniques)

ICS-Specific Techniques

ATT&CK Tactic	ATT&CK Technique	Description
Impact	T0826 – Loss of Availability	Manipulated Modbus registers to disrupt heating systems, causing service outages.
Impact	T0829 – Loss of View	Downgraded firmware on OT devices to disable monitoring capability.
Impact	T0836 – Modify Parameter	Issued Modbus write commands to alter control setpoints and sensor outputs.
Discovery	T0807 – Command-Line Interface	Leveraged Go-based CLI to control Modbus functions.
Network Effects	T0885 – Commonly Used Port	Exploited unprotected Modbus on TCP port 502 to communicate with ICS devices.
Network Effects	T0869 – Standard Application Layer Protocol	Used the Modbus TCP protocol directly to command OT systems.

Table 3: Mitre Att&ck mapping (ICS-Specific Techniques)

Associated Campaign & Software Identifiers

- **Campaign C0041 – "FrostyGoop Incident"**
  Originally observed in January 2024, targeting a municipal heating provider in Ukraine via Modbus TCP register manipulation and unauthorized ICS protocol access.
  Note: While the campaign was first attributed in 2024, recent telemetry and honeypot activity in 2025 suggest potential reuse or mimicry of similar TTPs, indicating either a revival or evolution of the original tactics.
- **Software S1165 – FrostyGoop (aka BUSTLEBERM)**
  A Go-based executable identified for its ability to read and write ICS registers over Modbus TCP, exhibiting protocol-aware behaviour without traditional exploit payloads.

Further Reading & Resources

- **MITRE Campaign C0041 – FrostyGoop Incident**: Official campaign profile detailing the heating plant compromise.
- **MITRE Software S1165 – FrostyGoop**: Technical breakdown of its ICS-specific TTPs.
- **MITRE Technique T1190**: Explains how a public-facing application exploits a function.

13. Threat Hunting Methodology – FrostyGoop

This methodology builds on the findings in the FrostyGoop analysis to enable proactive detection of similar ICS-targeted threats. It focuses on identifying protocol-level anomalies, tool usage, and infrastructure patterns unique to this malware family.

13.1 Hunting Scope & Data Sources

1. **In-scope environments:** ICS/OT networks where Modbus TCP (port 502) is in use, especially where ENCO controllers or other Modbus-enabled devices are deployed.
2. **Data sources:**
   - **Network telemetry:** Firewall, VPN, and perimeter device logs; full-packet captures (PCAP); NetFlow/IPFIX.
   - **Endpoint telemetry:** EDR/Sysmon process creation, registry access logs, file integrity monitoring.
   - **ICS-specific logs:** Engineering workstation command histories, PLC/HMI audit trails.
   - **Threat intel feeds:** Internal IOC lists (hashes, file names, config patterns) from confirmed FrostyGoop incidents.

13.2 IOC-Driven Hunting

1. **File and Tool Artifacts**

   - Search for the following SHA256 hashes in file inventories or EDR logs:
     5d2e4fd08f81e3b2eb2f3eaae16eb32ae02e760afc36fa17f4649322f6da53fb,
     a63ba88ad869085f1625729708ba65e87f5b37d7be9153b3db1a1b0e3fed309c,
     9cf30d82a86a9485f7bbd0786a5de207cf4902691a3efcfc966248cb1e87d5b7 (go-encrypt.exe).
   - Flag presence of JSON config files (task_test.json) containing FrostyGoop schema fields (Code, Count, TargetList, StartTime, IntervalTime, WorkTime).

2. **Network & Infrastructure Artifacts**

- Identify Modbus TCP traffic with function codes:
- 0x03 – Read Holding Registers
- 0x06 – Write Single Holding Register
- 0x10 – Write Multiple Holding Registers
- Correlate with register ranges seen in the report (e.g., 5275–5403) to detect known targeting patterns.
- Search for L2TP/IPSec VPN connections to/from suspicious geographies (notably Moscow-based endpoints) on ports 1701, 500, 4500.
- Check perimeter logs for web shell access patterns consistent with ReGeorg or custom HTTP tunneling scripts.

13.3 Behaviour-Based Hunting

1. **Modbus Protocol Abuse**

- Query ICS-aware DPI logs for:
  - Modbus writes (0x06 or 0x10) issued outside approved engineering schedules.
  - Read commands (0x03) requesting unusually large register counts.
  - Sequential register manipulation across multiple devices within short intervals.
- Compare command timing against known operational cycles to flag anomalies.

2. **Encrypted Task Deployment**

- Detect AES-encrypted JSON task files staged on engineering workstations or OT gateways.
- Monitor for execution of go-encrypt.exe or other Go-compiled utilities in ICS environments.

3. **Credential Access & Lateral Movement**

- Identify access to \Windows\System32\config\SAM registry hive or unusual reg.exe activity from OT endpoints.
- Search for new VPN tunnels initiated by engineering workstations or OT gateways.

13.4 Detection Engineering Integration

1. **SIEM rules:**
   - Trigger on Modbus writes from IT subnets to OT assets.
   - Alert on execution of unsigned Go binaries in OT zones.
   - Flag VPN connections overlapping with Modbus manipulation events.

2. **YARA/Sigma rules:**
   - Match Go build artifacts and embedded FrostyGoop config schema in binaries.
   - Detect JSON task structures matching observed field/value patterns.

3. **Correlation rules:**
   - Combine VPN connection, web shell activity, and Modbus manipulation into a single multi-stage alert.

13.5 Validation & Continuous Hunting

- Use FrostyGoop PCAP samples to validate detection rules in the lab.
- Deploy Modbus honeypots to capture unsolicited read/write attempts and compare payload structures with known FrostyGoop patterns.
- Schedule quarterly hunts focused on:
  - Modbus function code anomalies.
  - Go-compiled binary execution in OT.
  - Unauthorized VPN tunnels into ICS zones.

14. Evasion Techniques v/s Countermeasures

Evasion Technique	Description	Countermeasure Implemented
Encrypted Tasking (AES)	Task JSON files are encrypted with AES and separate 32-byte key files to evade signature and plaintext inspection.	Executed & instrumented go-encrypt.exe in sandbox, captured key via ProcMon, decrypted JSON with CyberChef, exposed full tasking data.
Anti-Debug Checks	Malware checks the BeingDebugged flag to hinder analysis.	Used Debuggers like debugger X86 and X64 in a sandbox that bypasses BeingDebugged, enabling full tracing.
Exploit-less Protocol Abuse	Direct Modbus/TCP communication from a compromised edge host avoids noisy IT malware behaviours.	Checked manually for anomalous Modbus function codes and register write changes in Windows on TCP 502.
Minimal Footprint / No Persistence	One-shot execution with no scheduled tasks, registry keys, or local persistence artefacts.	Shifted to network-based detection, hunting on Modbus patterns and JSON schema rather than host-based IOC reliance

Table 4: Evasion Techniques V/S Countermeasures used to Overcome Evasive Techniques

15. Detection & Mitigation

15.1 Detection Strategies

1. **Modbus TCP Traffic Monitoring**
   - Flag unusual traffic to TCP port 502 with atypical Modbus function codes (0x06, 0x10), especially outside engineering schedules.
   - Leverage DPI-capable IDS/IPS systems with support for behaviour-based OT rules.

2. **Alert on Go-Executable Artifacts**
   - Monitor execution of unrecognized Go-compiled binaries (e.g., with “Go build ID:” in metadata).
   - Detect presence or use of go-encrypt.exe or related JSON task files (e.g., task_test.json) via file integrity or EDR alerts.

3. **Network Tunnel & VPN Monitoring**
   - Watch for L2TP tunnel establishment from unusual geolocations—especially to/from Russia-based servers.
   - Analyze perimeter firewall logs for anomalies or unknown external connections on ports 1701, 500, 4500.

4. **Endpoint Forensics & Registry Monitoring**
   - Track registry access behaviors, such as SAM hive dumping tools used for credential harvesting.
   - Detect presence of web shells via file hashes or web-server log anomalies.

5. **YARA/Sigma Signatures**
   - Apply Sigma rules for ICS-specific TTPs, such as unexpected Modbus writes and external tunnel creation.

15.2 Mitigation & Hardening

1. **Effective Network Segmentation**
    - Enforce strict IT/OT zone separation, block direct Modbus TCP (port 502) access from outside networks.

2. **Access Control & Patching**
    - Disable unnecessary interfaces and services on perimeter devices (e.g., Telnet, unused router web GUIs).
    - Regularly update firmware on ICS infrastructure, especially internet-facing routers.

3. **Protocol Hardening**
    - Deploy Modbus over TLS (Modbus/TCP Secure) or shift to authenticated ICS protocols (e.g., CIP Secure, OPC UA Secure Channels).
    - Enforce Allow-list traffic policies, allowing only known ICS masters/HMI to communicate with PLCs/RTUs.

4. **Continuous OT Visibility**
    - Use specialized OT monitoring tools to generate alerts on activity spikes or unusual Modbus command patterns.

5. **Incident Response & Deception**
    - Prepare ICS-specific IR playbooks for Modbus abuse, define clear communication and system isolation steps.
    - Implement deception sensors (e.g., honeypots mimicking ICS devices) to detect reconnaissance or unauthorized command attempts.

6. **Testing & Validation**
    - Conduct periodic VAPT and table-top exercises simulating Modbus protocol manipulation.
    - Train OT staff on recognizing irregular ICS activity, for example, logs and alert flows in SOPs.

←----------**End of Document**----------→