Attackers target Ukraine using GoMet backdoor

Executive summary

Since the Russian invasion of Ukraine began, Ukrainians have been under a nearly constant barrage of cyber attacks. Working jointly with Ukrainian organizations, Cisco Talos has discovered a fairly uncommon piece of malware targeting Ukraine — this time aimed at a large software development company whose software is used in various state organizations within Ukraine. We believe that this campaign is likely sourced by Russian state-sponsored actors or those acting in their interests. As this firm is involved in software development, we cannot ignore the possibility that the perpetrating threat actor's intent was to gain access to source a supply chain-style attack, though at this time we do not have any evidence that they were successful. Cisco Talos confirmed that the malware is a slightly modified version of the open-source backdoor named "GoMet." The malware was first observed on March 28, 2022.

GoMet backdoor

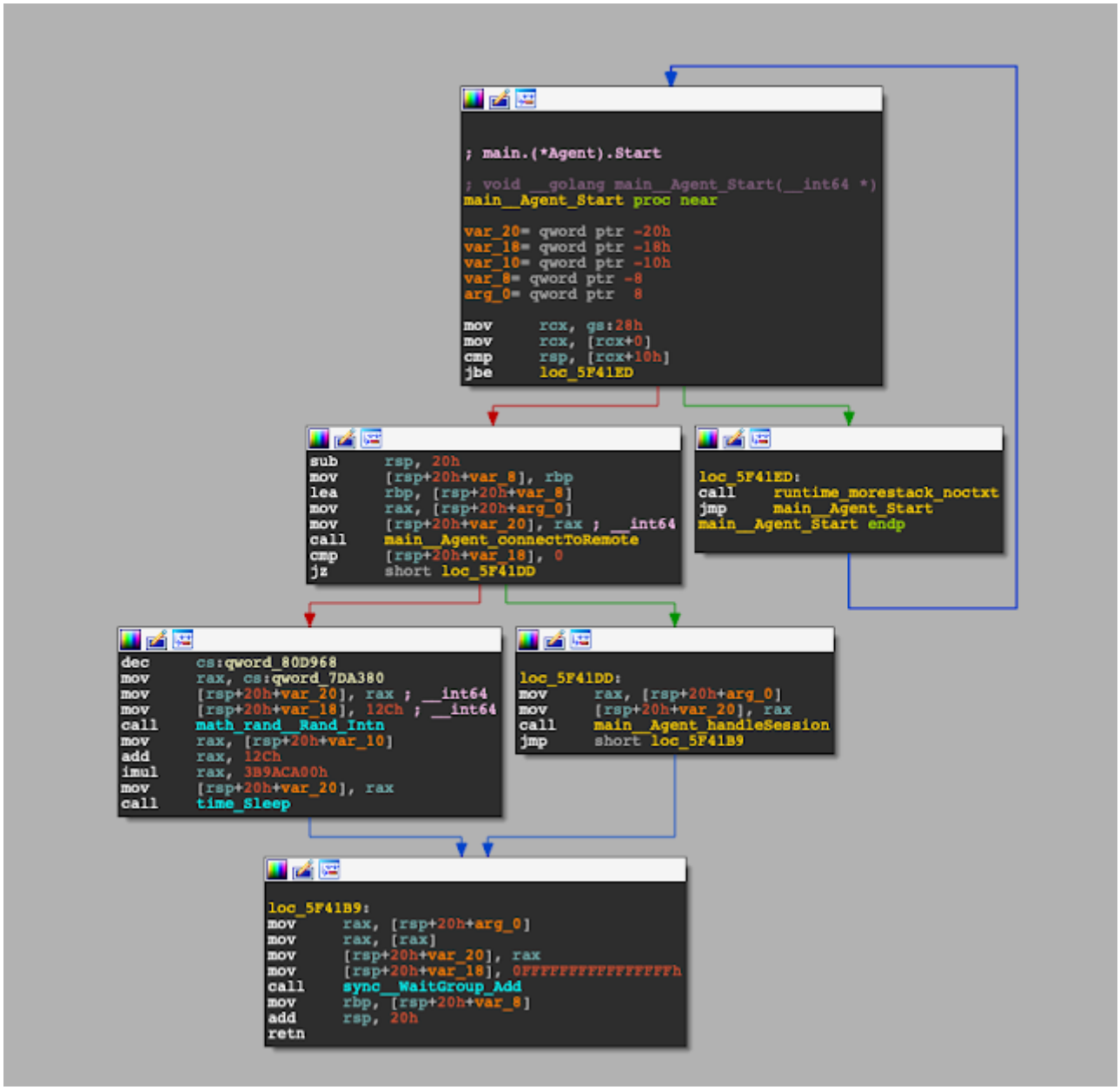
The story of this backdoor is rather curious — there are two documented cases of its usage by sophisticated threat actors. First, in 2020, attackers were deploying this malware after the successful exploitation of CVE-2020-5902, a vulnerability in F5 BIG-IP so severe that USCYBERCOM posted a tweet urging all users to patch the application. The second is more recent and involved the successful exploitation of CVE-2022-1040, a remote code execution vulnerability in Sophos Firewall.

Both cases are very similar. They both start with the exploitation of a public vulnerability on appliances where the malicious actors then dropped GoMet as a backdoor. As of publishing time, Cisco Talos has no reason to believe these cases are related to the usage of this backdoor in Ukraine.

The original GoMet author posted the code on GitHub on March 31, 2019 and had commits until April 2, 2019. The commits didn't add any features but did fix some code convention aesthetics. The backdoor itself is a rather simple piece of software written in the Go programming language. It contains nearly all the usual functions an attacker might want in a remotely controlled agent. Agents can be deployed on a variety of operating systems (OS) or architectures (amd64, arm, etc.). GoMet supports job scheduling (via Cron or task scheduler depending on the OS), single command execution, file download, file upload or opening a shell. An additional notable feature of GoMet lies in its ability to daisy chain — whereby the attackers gain access to a network or machine and then use that same information to gain access to multiple networks and computers — connections from one implanted host to another. Such a feature could allow for communication out to the internet from otherwise completely "isolated" hosts.

This version was changed by malicious actors, in the original code, the cronjob is configured to be executed once every hour on the hour. In our samples, the cronjob is configured to run every two seconds. This change makes the sample slightly more noisy since it executes every two seconds, but also prevents an hour-long sleep if the connection fails which would allow for more aggressive reconnection to the C2.

The objective of the cron job defined in the main part of the malware is to check if it's connected to the C2, if not it will start the agent component again and connect to the C2. The picture below shows the execution flow of the C2 setup routine Agent.Start.



This flow reveals another change to the GitHub versions. If the C2 is unreachable, the sample will sleep for a random amount of time between five and 10 minutes. GO's sleep implementation uses nanoseconds. The Pseudo Code would look like the following: time_Sleep(1000000000 * (rnd_val + 300)).

The 'WaitGroup_Add' call in the disassembly screenshot can also be confusing. The trick is, the Go compiler is changing the source code WaitGroup.Done() to WaitGroup.Add(-1).

After the Agent.start routine is done, the next cron job triggered the execution of the serve() routine and tried to start another instance of the Agent.

The simplified source code of the GitHub version looks like this:

```go
func main() {

    var wg sync.WaitGroup
    wg.Add(1)

    go serve()

    c := cron.New()
    c.AddFunc("0 * * * * *", serve)
    c.Start()

    wg.Wait()
}

func serve() {
    if connected {                          // return if an agent instance is already running
        return
    }
    connected = true

    var wg sync.WaitGroup
    wg.Add(1)

    a := NewAgent(&wg)                      // start a new agent instance and try
    a.Start()

    wg.Wait()                               // Wait for Agent start

    connected = false
}

func (a *Agent) Start() {
    err := a.connectToRemote()              // try to connect to the C2 server
    if err == nil {
        a.handleSession()
    }
    a.wg.Done()                             // stop waiting
}
```

The simplified pseudo-code for the samples in the wild looks like this:

```
func main() {

    agent_counter = 3                       // Max. number of agent instances

    wg_main.Add(1)

    go serve()
    c = cron.New()
    c.AddFunc("*/2 * * * * *", serve)       // Execute 'serve' every 2 sec
    c.Start()

    wg_main.Wait()
}

func serve() {

    var wg sync.WaitGroup

    wg.Add(1)

    go agent_start(&wg)

    wg.Wait()

    if agent_counter < 0 {
        c.Stop()                            // Stop the scheduler (does not stop any jobs already running).
        wg_main.Done()
    }

}

func agent_start(wg *sync.WaitGroup) {

    agent_counter = agent_counter - 1

    err := a.connectToRemote()              // try to connect to the C2 server
    if err == nil {
        a.handleSession()
    }

    time.Sleep(2 * time.Second)             // 2 = random value in real backdoor

    wg.Done()
}
```

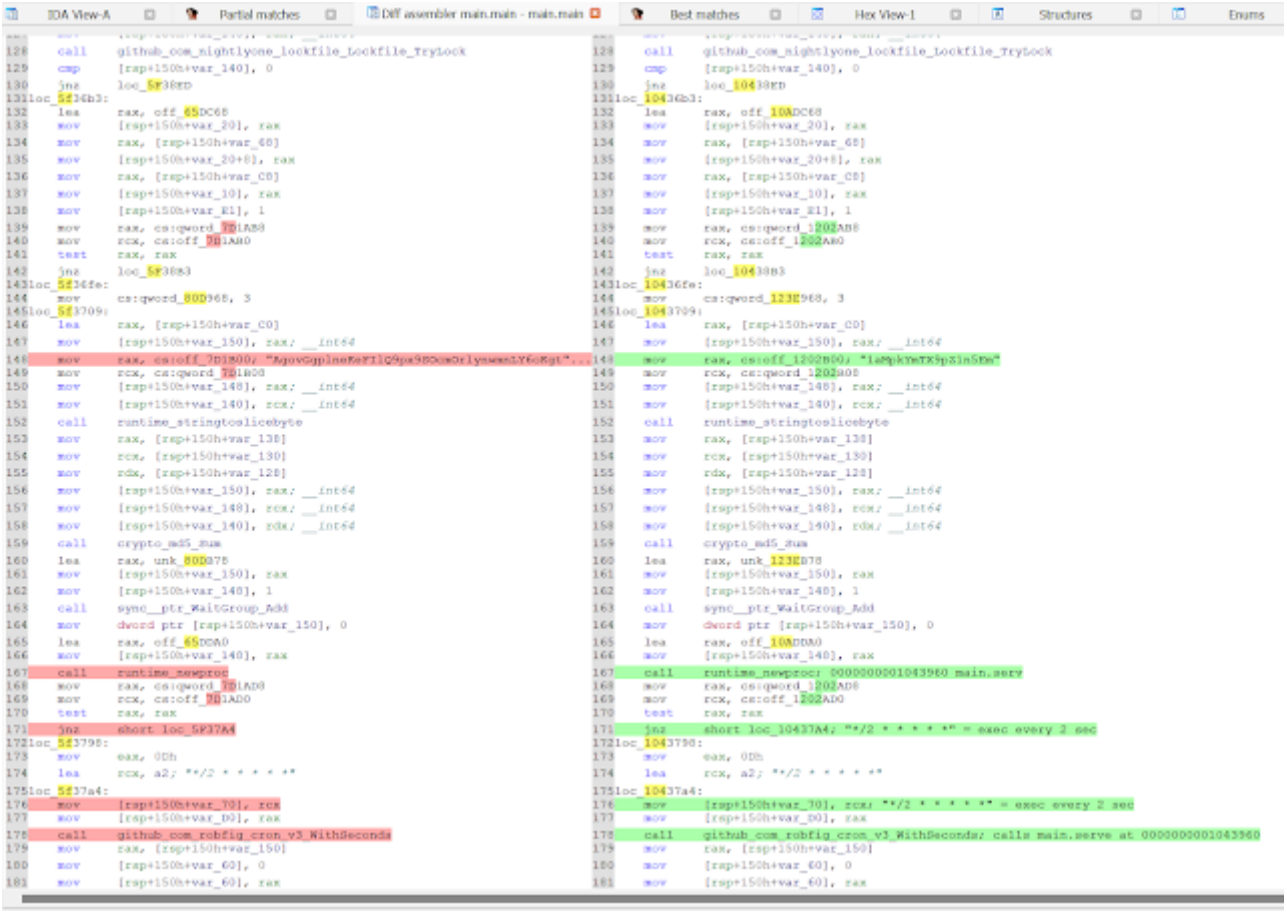
Talos found two samples of this version of the backdoor:

f24158c5132943fbdeee4de4cedd063541916175434f82047b6576f86897b1cb (FctSec.exe)

950ba2cc9b1dfaadf6919e05c854c2eaabbacb769b2ff684de11c3094a03ee88 (SQLocalM86.exe)

These samples have minor differences but are likely built from the same source code, just with a slightly different configuration.

If we look closely at the functions, they are not 100% equal, but we can see that the changes are mainly strings and similar victim or compiler-dependent data, along with researcher comments. Below is the Main.Main function as an example.



The malicious activity we detected included a fake Windows update scheduled tasks created by the GoMet dropper. Additionally, the malware used a somewhat novel approach to persistence. It enumerated the autorun values and, instead of creating a new one, replaced one of the existing goodware autorun executables with the malware. This potentially could avoid detection or hinder forensic analysis.

In one of the cases, about 60 seconds before the schtask query is executed, a blank CMD process is opened and then subsequently executes systeminfo and schtask queries rather than these queries being chain opened by svchost or services or another process. This execution looks like:

C:\WINDOWS\system32\cmd.exe 7)

systeminfo

schtasks /query /tn microsoft\windows\windowsupdate\scheduled

schtasks /query /tn microsoft\windows\windowsupdate\scheduled /v

Infrastructure

Both samples have the command and control (C2) IP address hardcoded, which is 111.90.139[.]122. Communication occurs via HTTPS on the default port.

The certificate on this server was issued on April 4, 2021 as a self-signed certificate, with the 9b5e112e683a3605c9481d8f565cfb3b7e2feab7 SHA-1 fingerprint. This indicates that this campaign preparation began as early as April 2021. At the moment, there are no known domains associated with this IP address and the last time there was a domain associated with it was on Jan. 23, 2021, which is outside the known attack time frame.

Conclusion

As the war in Ukraine rages on with little resolution in sight, we are reminded that attackers will try just about anything to gain additional leverage over their Ukrainian adversaries. Cisco Talos expects to see the continued deployment of a range of cyber weapons targeting the Ukrainian government and its counterparts. We remain vigilant and are committed to helping Ukraine defend its networks against such cyber attacks and working closely with our strategic allies in the region to gather and provide actionable threat intelligence.

In this instance, we saw a software company targeted with a backdoor designed for additional persistent access. We also observed the threat actor take active steps to prevent detection of their tooling by obfuscating samples and utilizing novel persistence techniques. This access could be leveraged in a variety of ways, including deeper access or launching additional attacks, including the potential for software supply chain compromise. It's a reminder that although the cyber activities haven't necessarily risen to the level many have expected, Ukraine is still facing a well-funded, determined adversary that can inflict damage in a variety of ways — this is just the latest example of those attempts.

We assess with moderate to high confidence that these actions are being conducted by Russian state-sponsored actors or those acting in their interests.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.

Indicators of Compromise

SHA-256 Hashes

f24158c5132943fbdeee4de4cedd063541916175434f82047b6576f86897b1cb
950ba2cc9b1dfaadf6919e05c854c2eaabbacb769b2ff684de11c3094a03ee88

IPs

111.90.139[.]122