RobinHood Ransomware “CoolMaker” Functions Not So Cool

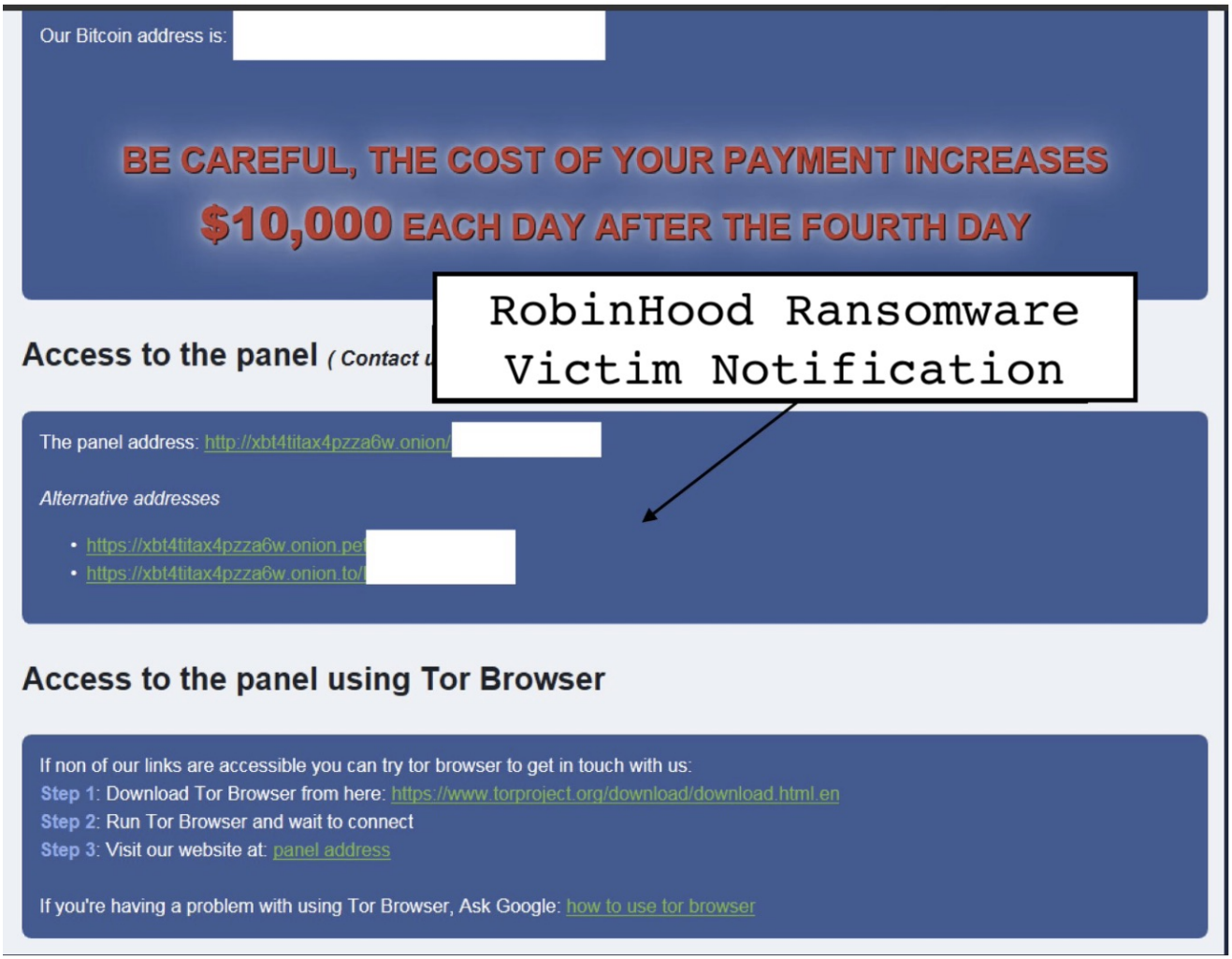
sentinelone.com/blog/robinhood-ransomware-coolmaker-function-not-cool/

Vitali Kremez

RobinHood ransomware is one of the more interesting Golang ransomware variants to have appeared on the ransomware landscape recently. The ransomware was previously used in the high-profile infection encrypting computers in the City of Greenville and most recently in the City of Baltimore. It was originally coded in the Go programming language and compiled to a 32-bit executable. In this technical analysis, we will explore the `main_CoolMaker` functions meant to disable the machine and interrupt backup and other PC vital services.

Overview of RobinHood Ransomware

RobinHood is a malware that encrypts the victim’s hard drive with the RSA+AES cryptographical combination and instructs the victim to reach out to them via Onion Tor website. The RobinHood ransomware drops the victim notification file on the desktop detailing the demands and how to make contact.



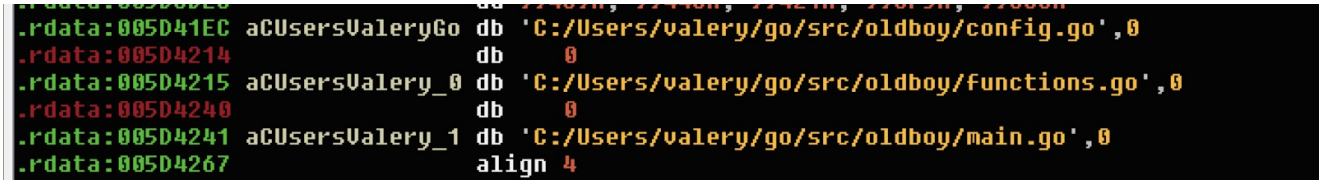
Once contact is made, the attackers claim they will make a decryption tool available, thereby allowing the victim to recover their precious files, in return for payments made in bitcoin.

Currently, it is unclear what the initial infection vector is. There is only one confirmed RobinHood Golang ransomware that we know of so far. It is also notable that the ransomware does not spread within the network; quite the opposite, it drops all Windows shares via “cmd.exe /c net use * /DELETE /Y”. That likely means that the ransomware is pushed on each machine individually after the initial network breach via the `psexec` and/or the domain controller.

> ***Update (July 26):*** Since this analysis, others have claimed that Robinhood was leveraging EternalBlue as a means to propagate. Those claims are incorrect, and it has now been confirmed by the City of Baltimore that Robinhood ransomware was **not** exploiting #EternalBlue/#BlueKeep vulnerabilities (CVE-2019-0708).
>
> **13. Resources indicate you didn't update the SMB patch Microsoft released in 2017 which could have prevented the ransomware attack from happening. Is this true? Why didn't you install the patch?**
>
> The SMB vulnerability was not a factor in the Baltimore City RobbinHood ransomware attack.

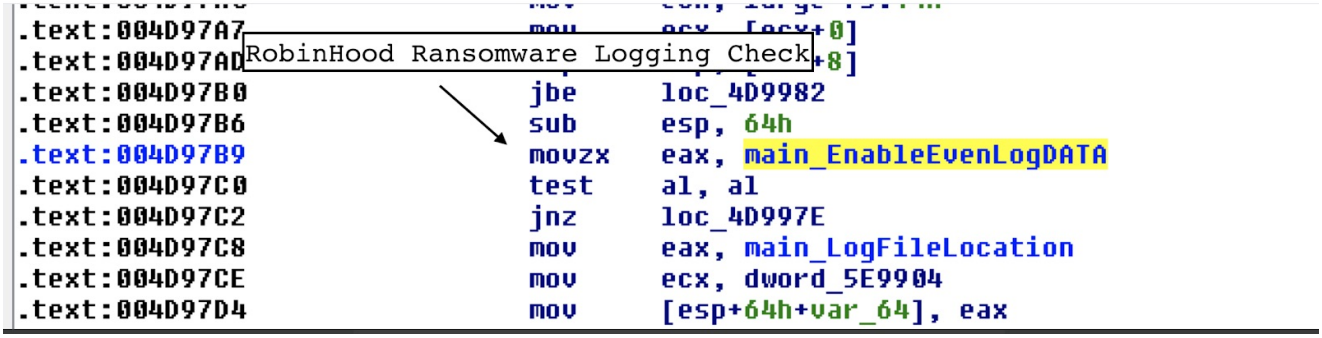
The ransomware expects to read "C:windowstemppub.key", and if the file is not found, the sample terminates. This suggests a possible antidote of creating and saving a "pub.key" file in "C:windowstemp" with no read or write privileges, which would cause the ransomware to abort its initial execution in its current known setup.

The ransomware contains the following debug artifacts:




```
C:/Users/valery/go/src/oldboy/config.go
C:/Users/valery/go/src/oldboy/functions.go
C:/Users/valery/go/src/oldboy/main.go
```

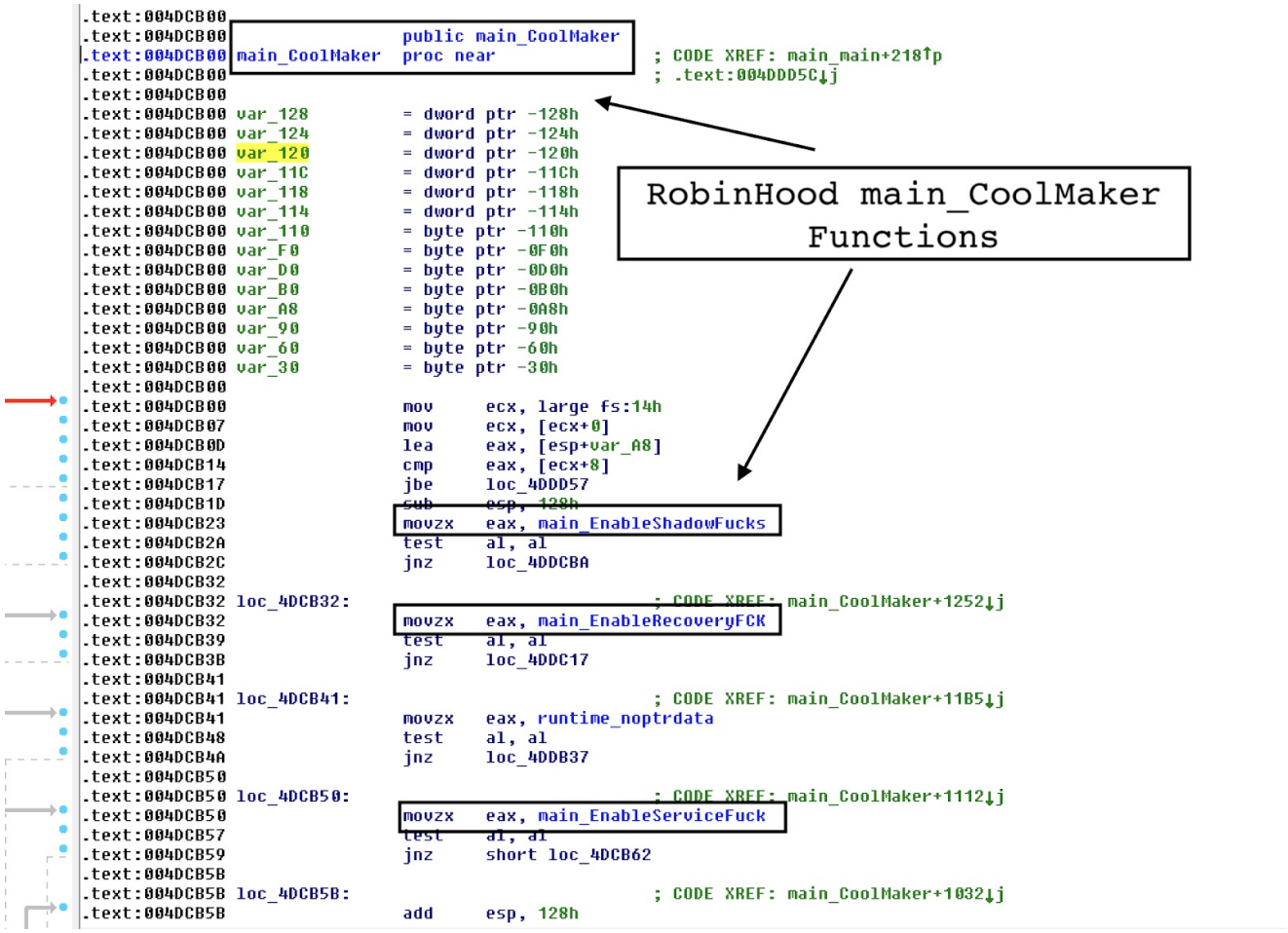
It is also notable that the ransomware contains full debugging capabilities to write logs to "C:windowstemprbf.log"; however, the ransomware was compiled with `main_EnableEventLogDATA` disabled, but it could be patched to retrieve and activate this feature.



RobinHood Ransomware's CoolMaker Function

RobinHood ransomware's `main_CoolMaker` function contains a plethora of subfunctions meant to disable and disrupt the victim's PC backups and services. Some of the most interesting Golang functions are stored here, with names riddled with expletives. These are responsible for actions such as deleting shadow copies via the impolitely named `ShadowFucks` function (vssadmin.exe delete shadows /all /quiet and WMIC shadowcopy delete), `RecoveryFCK` (Bcdedit.exe /set {default} recoveryenabled no, Bcdedit.exe /set {default} bootstatuspolicy ignoreallfailures), and `ServiceFuck` (cmd.exe /c sc.exe stop <list of services).

Aside from these, the somewhat more temperately named `wevtutil` (wevtutil.exe cl Application, wevtutil.exe cl Security, and wevtutil.exe cl System.exe) is also found here, which functions to clear logs.




```
.text:004DCB00
.text:004DCB00                  public main_CoolMaker
.text:004DCB00 main_CoolMaker   proc near               ; CODE XREF: main_main+218↑p
.text:004DCB00                                          ; .text:004DDD5C↓j
.text:004DCB00
.text:004DCB00
.text:004DCB00 var_128          = dword ptr -128h
.text:004DCB00 var_124          = dword ptr -124h
.text:004DCB00 var_120          = dword ptr -120h
.text:004DCB00 var_11C          = dword ptr -11Ch
.text:004DCB00 var_118          = dword ptr -118h
.text:004DCB00 var_114          = dword ptr -114h
.text:004DCB00 var_110          = byte ptr -110h
.text:004DCB00 var_F0           = byte ptr -0F0h
.text:004DCB00 var_D0           = byte ptr -0D0h
.text:004DCB00 var_B0           = byte ptr -0B0h
.text:004DCB00 var_A8           = byte ptr -0A8h
.text:004DCB00 var_90           = byte ptr -90h
.text:004DCB00 var_60           = byte ptr -60h
.text:004DCB00 var_30           = byte ptr -30h
.text:004DCB00
.text:004DCB00                  mov     ecx, large fs:14h
.text:004DCB07                  mov     ecx, [ecx+0]
.text:004DCB0D                  lea     eax, [esp+var_A8]
.text:004DCB14                  cmp     eax, [ecx+8]
.text:004DCB17                  jbe     loc_4DDD57
.text:004DCB1D                  sub     esp, 128h
.text:004DCB23                  movzx   eax, main_EnableShadowFucks
.text:004DCB2A                  test    al, al
.text:004DCB2C                  jnz     loc_4DDCBA
.text:004DCB32
.text:004DCB32 loc_4DCB32:                              ; CODE XREF: main_CoolMaker+1252↓j
.text:004DCB32                  movzx   eax, main_EnableRecoveryFCK
.text:004DCB39                  test    al, al
.text:004DCB3B                  jnz     loc_4DDC17
.text:004DCB41
.text:004DCB41 loc_4DCB41:                              ; CODE XREF: main_CoolMaker+11B5↓j
.text:004DCB41                  movzx   eax, runtime_noptrdata
.text:004DCB48                  test    al, al
.text:004DCB4A                  jnz     loc_4DDB37
.text:004DCB50
.text:004DCB50 loc_4DCB50:                              ; CODE XREF: main_CoolMaker+1112↓j
.text:004DCB50                  movzx   eax, main_EnableServiceFuck
.text:004DCB57                  test    al, al
.text:004DCB59                  jnz     short loc_4DCB62
.text:004DCB5B
.text:004DCB5B loc_4DCB5B:                              ; CODE XREF: main_CoolMaker+1032↓j
.text:004DCB5B                  add     esp, 128h
```

Closing Thoughts

While the RobinHood ransomware does not appear to be sophisticated, it does include higher-level Go programming language code, and its related network intrusions are more interesting as they targeted large government entities such as City of Greenville and City of Baltimore, a tactic reminiscent of previous SamSam ransomware attacks demanding high payouts with individual ransoms set per machine.

The group behind this ransomware and its attacks may prove to be more interesting than the ransomware itself due to the apparent well-planned and orchestrated network intrusions prior to the deployment of their new Go ransomware. It's reasonable to assume that we can expect to see more attacks from this threat actor on public institutions that fail to implement a ransomware-resistant security solution.

Read more about Cyber Security

- Ursnif – A Polymorphic Delivery Mechanism Explained
- Asus ShadowHammer Episode – A Custom Made Supply Chain Attack
- 7 Reasons To Move Away From Legacy AV
- How Malware Can Easily Defeat Apple's macOS Security

- What Is Windows PowerShell (And Could It Be Malicious)?