Stairwell

Maui ransomware

Threat report

Silas Cutler, Principal Reverse Engineer

06/07/2022

Table of contents

As ransomware has grown to epidemic proportions, the ecosystems of Ransomware-as-a-Service (RaaS) gangs such as Conti, LockBit, and BlackCat have become broadly recognizable. Outside of that ecosystem, there are other ransomware families that often receive less attention, yet are important to study because they can help broaden our understanding of the ways threat actors may conduct extortion operations.

In June 2022, the Stairwell research team investigated one of these lesser-known families, the Maui ransomware. Maui stood out to us because of a lack of several key features we commonly see with tooling from RaaS providers, such as an embedded ransom note to provide recovery instructions or automated means of transmitting encryption keys to attackers. Instead, we believe that Maui is manually operated, in which operators will specify which files to encrypt when executing it and then exfiltrate the resulting runtime artifacts.

There are many aspects to Maui ransomware that are unknown, including usage context. The following report will provide a technical overview of the Maui ransomware; our goal with the publication of our findings is to raise awareness of this ransomware and provide a starting point for other researchers.

Technical overview

The earliest identified copy of Maui (SHA256 hash: `5b7ecf7e9d0715f1122baf4ce745c5fcd769dee48150616753fec4d6da16e99e`) was first collected by Stairwell's inception platform on 3 April 2022. All identified copies of Maui (as of this report) have shared a compilation timestamp of 15 April 2021 04:36:00 UTC. Based on overlapping compilation timestamps and error messages, it is believed that Maui is configured using an unidentified external builder.

Maui is believed to be designed for manual execution by attackers. When executed at the command line without any arguments, Maui prints *usage* information, detailing supported command-line parameters (shown in Figure 1). The only required argument is a folder path, which Maui will parse and encrypt identified files.

```
Usage: maui [-ptx] [PATH]
Options:
-p dir: Set Log Directory (Default: Current Directory)
-t n:           Set Thread Count (Default: 1)
-x:             Self Melt (Default: No)
```

Figure 1: Maui command line usage details

Embedded usage instructions and the assessed use of a builder is common when there is an operational separation between developers and users of a malware family. The Stairwell research team has not identified any public offerings for Maui and assesses that it is likely privately developed.

Encryption

Instead of relying upon external infrastructure to receive encryption keys, Maui creates three files in the same directory it was executed from (unless a custom log directory is passed using the `-p` command line argument) containing the results of its execution. These files are likely exfiltrated by Maui operators and processed by private tooling to generate associated decryption tooling. A description of each of these files is provided below:

File Name	Description
`maui.evd`	RSA private key generated at runtime, encrypted using hard-coded public key
`maui.key`	RSA public key generated at runtime, encoded using XOR key generated from hard drive information
`maui.log`	Log file containing output console output from execution

Table 1: Files generated by Maui

The strategy used in Maui for encrypting files can be logically divided into three layers, similar to Conti (2021)[1] and ShiOne (2022).[2]

At the inner layer, files are encrypted using Advanced Encryption Standard (AES) with a unique 16-byte key for each file. Corresponding AES keys are RSA (Rivest–Shamir–Adleman) encrypted using a keypair generated the first time Maui is run (referred to hereafter as the *runtime RSA keys*). This key pair represents the second layer of encryption and, unless Maui is run under different conditions, will be unique to each system. At the final layer, runtime RSA keys are encrypted, using a different, hard-coded RSA public key (stored at the end of the Maui executable).

From the limited number of observed samples, it is unclear if this hard-coded public key is unique to campaigns, targeted networks, or individual operators.

[1] https://assets.sentinelone.com/sentinellabs/conti-ransomware-unpacked
[2] https://blog.malwarebytes.com/threat-analysis/2018/02/encryption-101-shione-ransomware-case-study/

Hard-coded public key

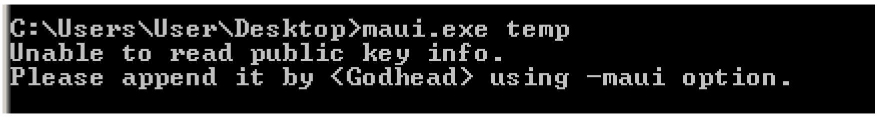
At the start of execution, Maui will load an RSA public key stored at the end of itself on disk. This key is stored in a format (shown in Figure 2) designed to allow for safe programmatic retrieval.

```
000bedf0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000bee00: 3081 9f30 0d06 092a 8648 86f7 0d01 0101  0..0...*.H......
000bee10: 0500 0381 8d00 3081 8902 8181 00b9 0893  ......0.........
000bee20: 47b1 444e 7caa 2627 6d01 dd0a b82d 91b0  G.DN|.&'m....-..
000bee30: e980 04e2 2a45 163c 555a 5dff 6761 74fd  ....*E.<UZ].gat.
000bee40: dc86 978a fa81 bf73 3ce1 7d38 d540 c615  .......s<.}8.@..
000bee50: 63f3 a5e1 eaa1 cf3c ab43 bbef ed7c 569e  c......<.C...|V.
000bee60: 9992 3b46 0959 843e 9ae5 a8e4 5e8c a01f  ..;F.Y.>....^...
000bee70: 5c64 6e68 209d 7d74 8d7e 59e0 ac4b 618f  \dnh .}t.~Y..Ka.
000bee80: 8c7d a141 e5ed 5427 1512 e7fd b307 56a6  .}.A..T'......V.
000bee90: 9031 d05a 81fc 1a80 2bb6 bacc 9502 0301  .1.Z....+.......
000beea0: 0001 4b42 5550 0100 0000 a200 0000       ..KBUP........

Key:
RSA Public Key     - 3081 .... 0001
PUBK Sentinel      - 4b42 5550
Key version        - 0100
```

Figure 2: Public key stored in the last bytes of Maui on disk

Using `fseek()`, Maui will read the last twelve bytes of itself from disk, verify the resulting first 4 bytes contain the static value of `PUBK`, followed by a number one, denoting the key version. If both of these checks pass, the 162-bytes preceding the `PUBK` sentinel are read, containing the public key. If these checks fail, a corresponding error message will be presented; the error message in Figure 3 corresponds to when the `PUBK` sentinel is not present.



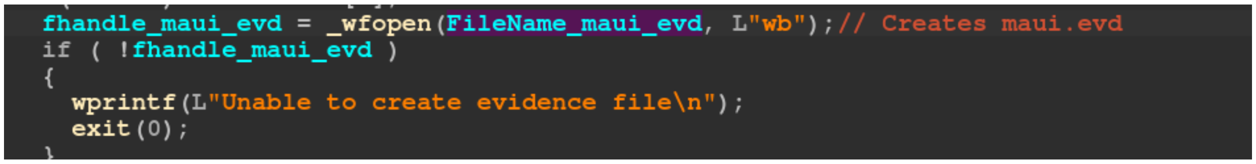
```
C:\Users\User\Desktop>maui.exe temp
Unable to read public key info.
Please append it by <Godhead> using -maui option.
```

Figure 3: Error message presented if Maui’s embedded key is not found

A proof-of-concept (POC) Python implementation of this key loading process is included in the Appendix of this report. This tool may also be useful for researchers for tracking public keys used in copies of Maui.

Runtime keys

Following extraction of the hard-coded RSA public key, Maui generates a new keypair using `RSA_generate_key()`. The resulting private key is then encrypted using the hard-coded public key and written to a file named `maui.evd`. Based on debug messages (shown in figure 4), the developers describe `maui.evd` as an evidence file.



```
fhandle_maui_evd = _wfopen(FileName_maui_evd, L"wb");// Creates maui.evd
if ( !fhandle_maui_evd )
{
  wprintf(L"Unable to create evidence file\n");
  exit(0);
```

Figure 4: Error messages referring to `maui.evd` as an evidence file

The corresponding public key is encoded using a 16-byte XOR key, generated using information about `\\.\PhysicalDrive0`, and written to a file named `maui.key`. XOR encoding was likely chosen for this file instead of RSA encryption to support key reuse if Maui is run multiple times on a target system.

File encryption

Files are encrypted by Maui using Advanced Encryption Standard (AES) in CBC mode using a 32-byte key generated per file. Keys are prefixed by the hard-coded string `dogd`, followed by 28 bytes generated using `RAND_bytes()`.[3]

Each file encrypted by Maui contains a custom header, allowing the malware to programmatically identify already encrypted files. This header includes the file's original path and an encrypted copy of the AES key (encrypted using the runtime RSA public key). An example of this header is defined in the figure below.

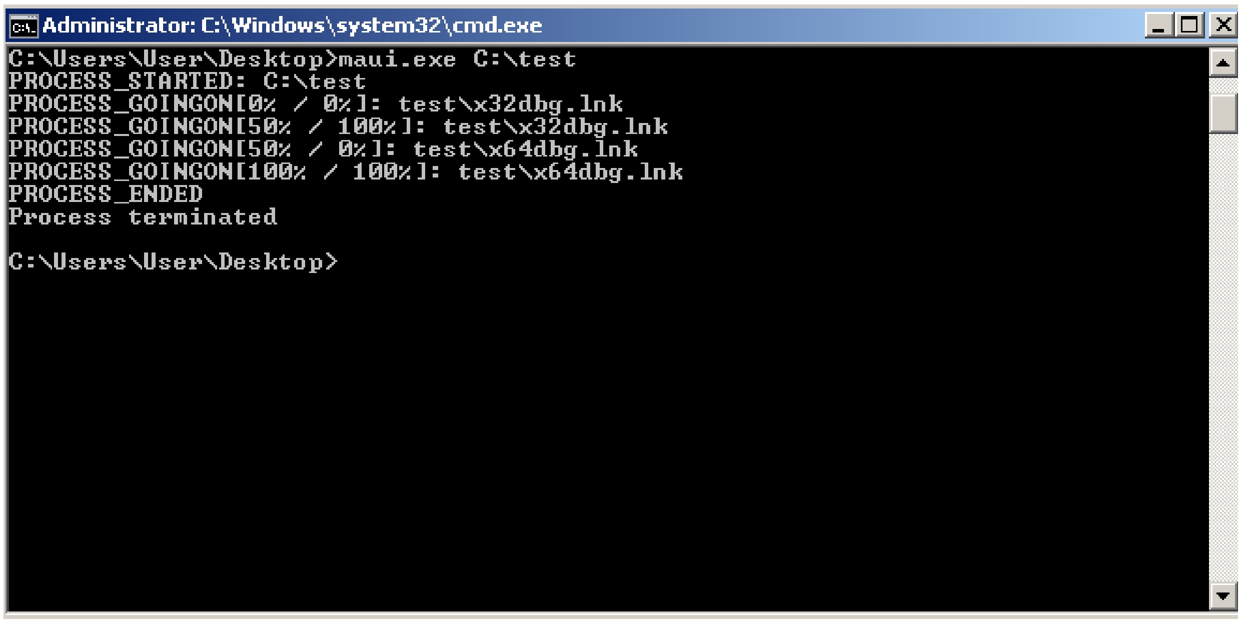
```
00000000: 5450 5243 0100 0000 5500 7300 6500 7200  TPRC....U.s.e.r.
00000010: 5c00 4400 6500 7300 6b00 7400 6f00 7000  \.D.e.s.k.t.o.p.
00000020: 5c00 4900 4400 4100 2000 5000 7200 6f00  \.I.D.A. .P.r.o.
00000030: 2000 2800 3300 3200 2d00 6200 6900 7400   .(.3.2.-.b.i.t.
00000040: 2900 2e00 6c00 6e00 6b00 0000 a303 0000  )...l.n.k.......
00000050: 0000 0000 47d9 7e5f 1552 56c4 f743 28d5  ....G.~_.RV..C(.
00000060: 79e0 ccb8 fc9b e021 48b5 cd67 5223 b027  y......!H..gR#.'
...
000000c0: fa9e a110 096f 26f9 ecc1 5f70 71fd 13da  .....o&..._pq...
000000d0: fa88 1894 fa96 ca88 2f41 771f 7570 a4d1  ......../Aw.up..
000000e0: 5587 6440 83f6 222d 3a38 f42b d1d6 2d4c  U.d@.."-:8.+..-L
...
```

[3] https://www.openssl.org/docs/man1.1.1/man3/RAND_bytes.html

```
Key:
Custom Header                  = 5450 5243
Static value                   = 0100 0000
Original File Path             = 5500 ...
Encrypted AES Key              = 47d9 - 1894
Encrypted file                 = fa96 - [EOF]
```

Figure 5: Mail Encrypted file header

While Maui is encrypting files, it outputs status information back to operators, as can be seen in Figure 6.



```
C:\Users\User\Desktop>maui.exe C:\test
PROCESS_STARTED: C:\test
PROCESS_GOINGON[0% / 0%]: test\x32dbg.lnk
PROCESS_GOINGON[50% / 100%]: test\x32dbg.lnk
PROCESS_GOINGON[50% / 0%]: test\x64dbg.lnk
PROCESS_GOINGON[100% / 100%]: test\x64dbg.lnk
PROCESS_ENDED
Process terminated

C:\Users\User\Desktop>
```

Figure 6: Command line output during execution

Appendix

PoC key extractor

The following Python script can be used to extract public RSA keys stored in copies of Maui.

```
#!/usr/bin/env python3
# Author: Silas Cutler (silas@stairwell.com)
# Desc: Maui public key extractor

import os
import sys
from Crypto.PublicKey import RSA

def parse_key(inkey):
    keyPub = RSA.importKey(inkey)
    print(keyPub.exportKey().decode('utf-8'))

def main():
    with open(sys.argv[1], 'rb') as fhandle:
        # Read first 12-bytes containing headers
        fhandle.seek(-12, os.SEEK_END)

        #Check config PUBK header
        if fhandle.read(4) == b'KBUP':
            print(" [D] Found PUBK sentinel")
       else:
            print(" [X] Missing PUBK sentinel")
            return False
        # Check pub key version
        if fhandle.read(1) == b'\x01':
            print(" [D] Found pub key version")
       else:
            print(" [X] Invalid pub key version")
            return False

        # Extract Pub Key
        fhandle.seek(-174, os.SEEK_END)
        rsakey = fhandle.read(162)
        parse_key(rsakey)

if __name__ == "__main__":
    main()
```

YARA rules

Stairwell's Inception platform users already have access to associated YARA rules automatically.

```
rule MauiRansomware
{
    meta:
        author= "Silas Cutler (Silas@Stairwell.com)"
        description = "Detection for Maui Ransomware"
        version = "0.1"
    strings:
        $ = "Unable to read public key info." wide
        $ = "it by <Godhead> using -maui option." wide
        $ = "Incompatible public key version." wide
        $ = "maui.key" wide
        $ = "maui.evd" wide
        $ = "Unable to encrypt private key" wide
        $ = "Unable to create evidence file" wide
        $ = "PROCESS_GOINGON[%d%% / %d%%]: %s" wide
        $ = "demigod.key" wide
        $ = "Usage: maui [-ptx] [PATH]" wide
        $ = "-p dir: Set Log Directory (Default: Current Directory)" wide
        $ = "-t n:          Set Thread Count (Default: 1)" wide
        $ = "-x:            Self Melt (Default: No)" wide

        // File header loading (x32-bit)
        $ = { 44 24 24 44 49 56 45 ?? 44 24 28 01 00 00 00 ?? 44 24 2C 10 00 00 00 }
        $ = { 44 4F 47 44 ?? ?? 04 01 00 00 00 }

    condition:
        3 of them or
        (
            uint32(filesize-8) == 0x00000001 and
            uint32(filesize-12) == 0x5055424B
        )
}
```

Files

File Name	File Type	Size	Sha256 Hash
proc.exe	Windows portable executable	764K	45d8ac1ac692d6bb0fe776620371fca0 2b60cac8db23c4cc7ab5df262da42b78
aui.exe	Windows portable executable	764K	5b7ecf7e9d0715f1122baf4ce745c5fc d769dee48150616753fec4d6da16e99e
Maui.exe	Windows portable executable	764K	830207029d83fd46a4a89cd623103ba2 321b866428aa04360376e6a390063570

For more information on the intelligence provided in this report, contact us at threatresearch@stairwell.com