How We Seized 15 Active Ransomware Campaigns Targeting Linux File Storage Servers

Introduction

It is rare to see ransomware being used to target the Linux operating system. However, cyber criminals seem to adapt to this emerging environment and use a variety of creative methods to gain profits from this landscape.

We at Intezer have **detected** and **temporarily DoS'd the operation** of a ransomware targeting Linux-based file storage systems (NAS servers).

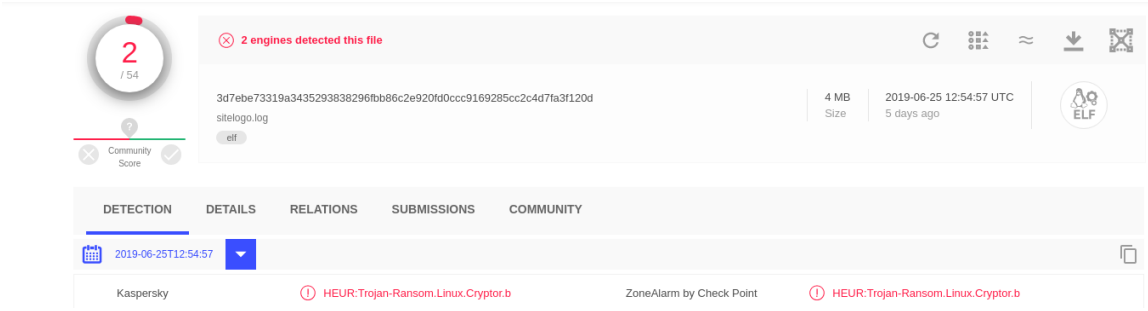
We have named the ransomware **QNAPCrypt**, as this is the name the authors have appeared to label the malware. QNAP is a well-known vendor for selling NAS servers, which the malware was intended to infect and encrypt the containing files for ransom. NAS servers normally store large amounts of important data and files, which make them a valuable target for attackers and especially a viable target for ransomware campaigns.

This malware currently has very low detection rates in all major security solutions.

The first two sections of this blog post will explain in brief how QNAPCrypt operates and how we were able to take advantage of two design flaws in the ransomware infrastructure in order to temporarily stop the campaign—preventing the malware from infecting additional victims and forcing the authors behind this malware to deploy new instances. Lastly, we will present a detailed technical analysis of the malware and the investigation of the entire campaign.

For reference, here is the genetic analysis of the QNAPCrypt malware:

- **ARM variant**
- **x86 variant**



How the Ransomware Works

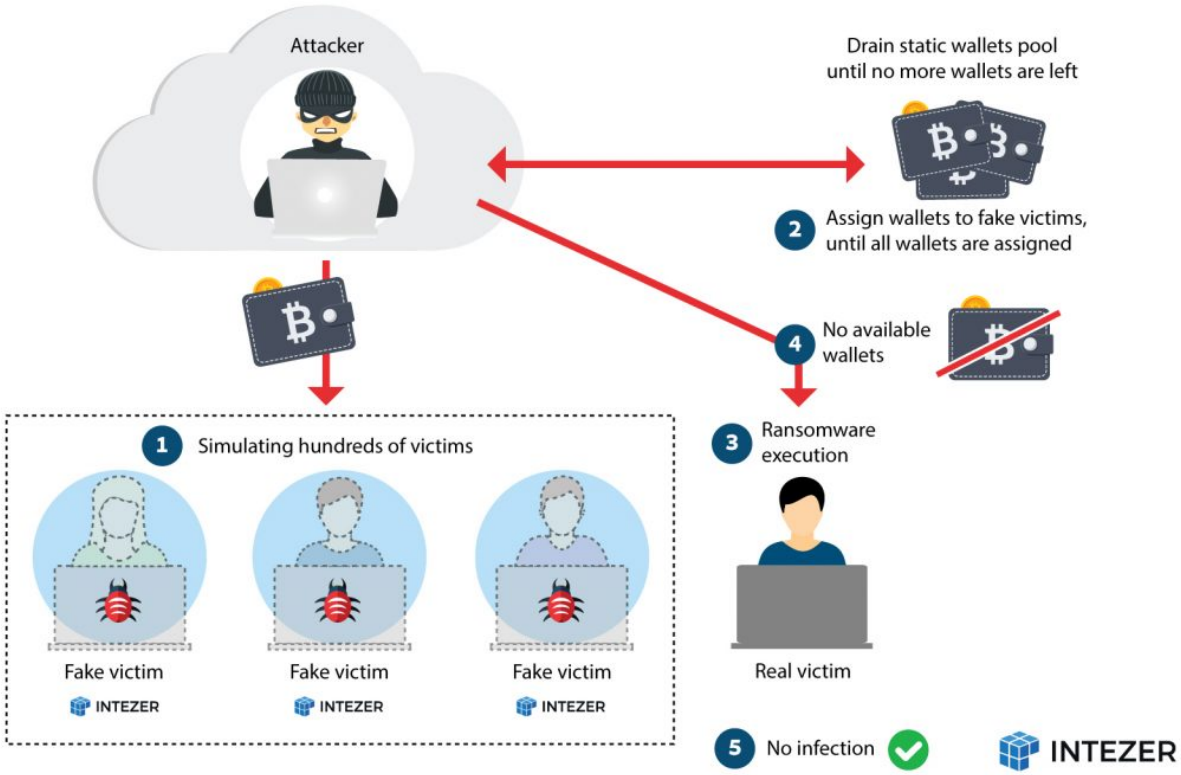
The QNAPCrypt ransomware works similarly to other ransomware, including encrypting all files and delivering a ransom note. However, there are several important differences:

1. The ransom note was included solely as a text file, without any message on the screen—naturally, because it is a server and not an endpoint.

2. Every victim is provided with a different, unique Bitcoin wallet—this could help the attackers avoid being traced.

3. Once a victim is compromised, the malware requests a wallet address and a public RSA key from the command and control server (C&C) before file encryption.

How We Seized the Campaign



In order to further research the malware and its operation, we wrote a script to simulate infections on a wide scale to see how the wallet generation mechanism worked in the attackers' back end.

After simulating the infections of hundreds of virtual "victims", we discovered two major design flaws in the ransomware infrastructure which led us to seize the operation:

1. The list of bitcoin wallets was created in advance and it was static. Therefore, it does not create a new wallet for each new victim in real time, but rather it pulls a wallet address from a fixed, predetermined list.

2. Once all of the wallets are allocated (or sent), the ransomware would not be able to continue its malicious operation in the victim's machine.

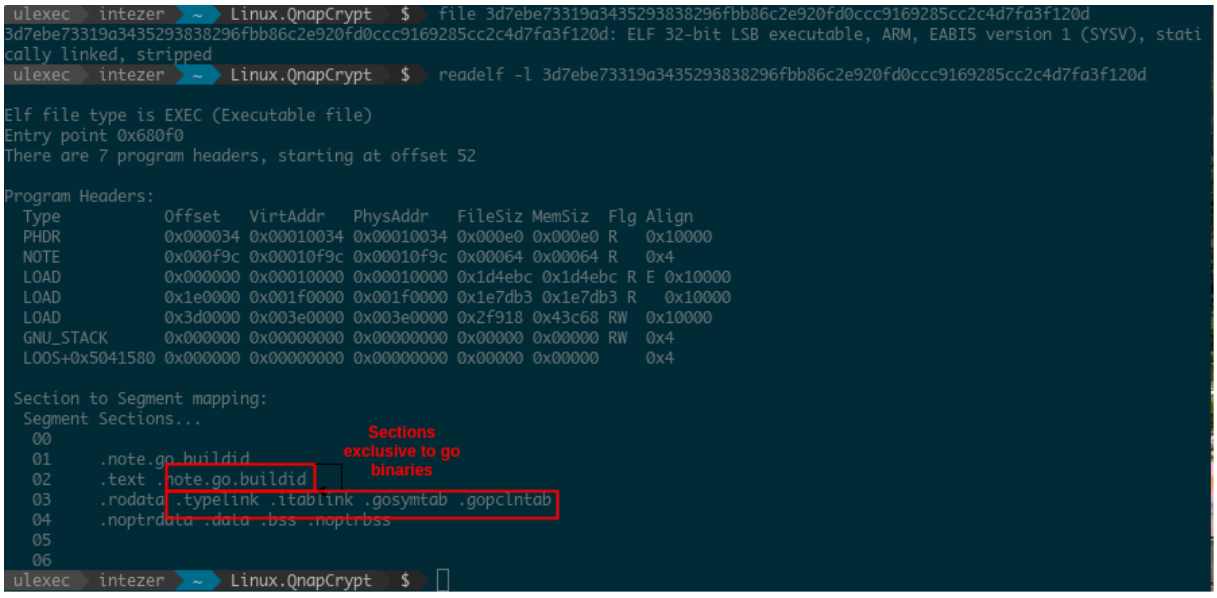
After simulating the infection of more than 1,091 victims from 15 different campaigns, we encountered that the attackers ran out of unique Bitcoin wallets to supply to their victims. As a result, any future infection will be unsuccessful and the authors behind this malware were forced to update their implants in order to circumvent this design flaw in their infrastructure to continue with their malicious operations.

After several days of continuously DoS'ing their infrastructure, we have observed a newer variant in the wild that shares a significant amount of code with previous QNAPCrypt instances and **Linux.Rex**. This time, the newer variant uses an embedded static wallet and RSA public key in contrast to previous instances.

Technical Analysis

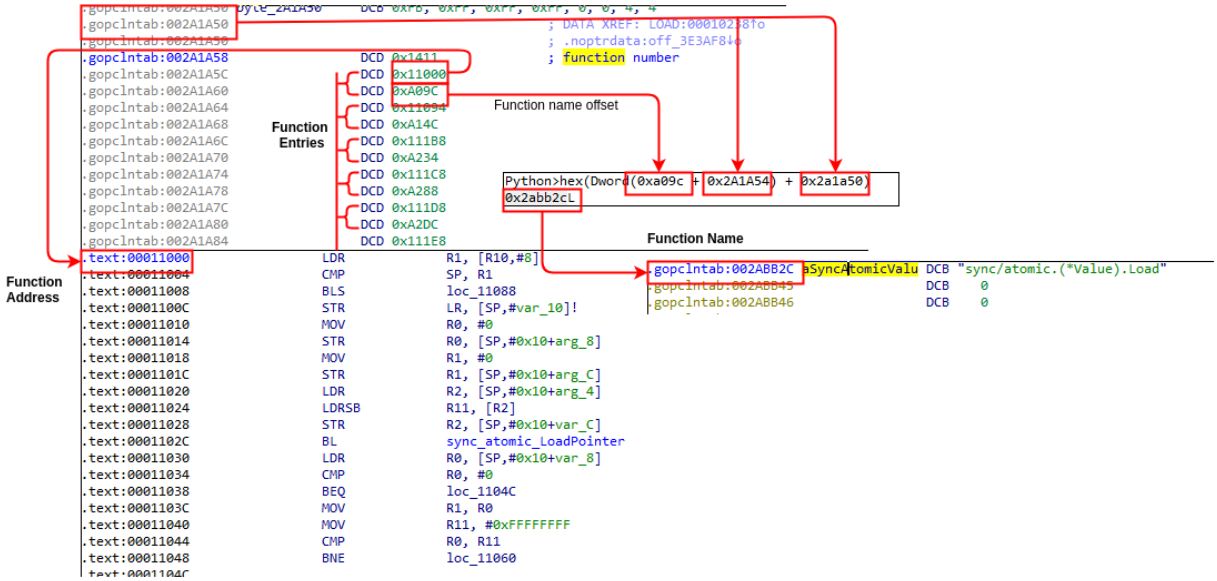
The initial implant we found came in the form of a statically linked Golang binary built with the Go linker for ARM architecture. Throughout our research, we were able to confirm that other variants exist for additional architectures such as x86 / x64.

Go binaries may seem difficult to analyze when they come stripped, since trying to make sense of stripped statically linked binaries is usually a more difficult task than analyzing stripped dynamically linked binaries.



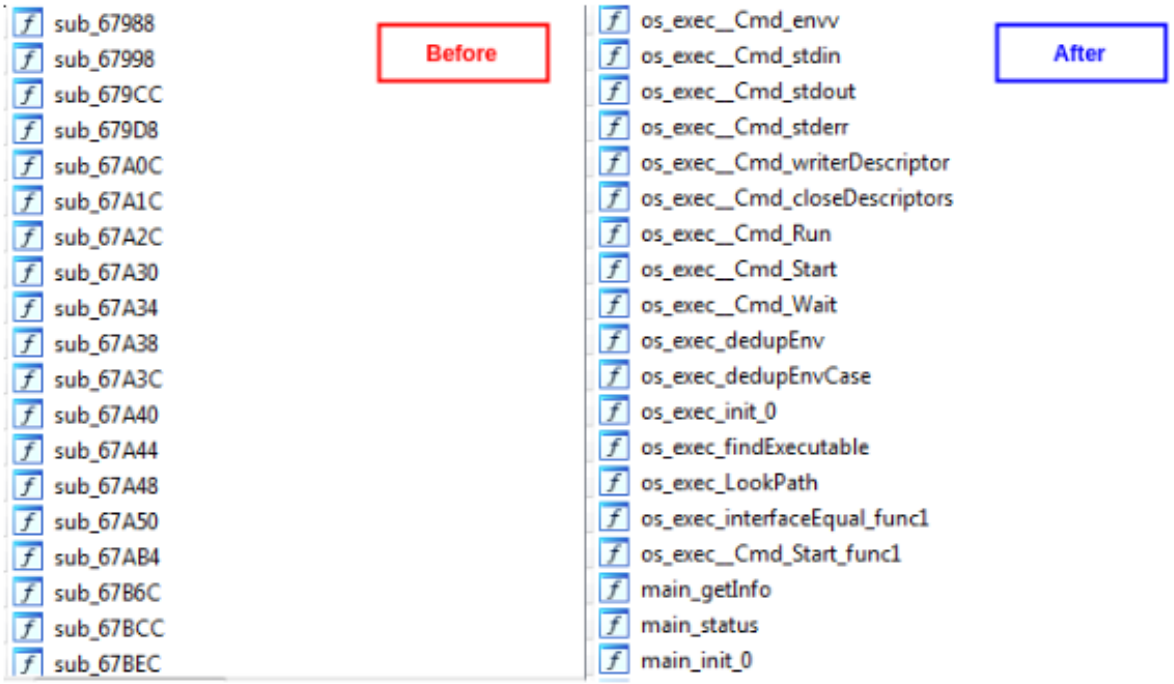
We can observe that this binary is indeed a Go executable by looking at the section names in its section header table.

If we know the location of these sections, in particular the *.gopclntab* section, we will be able to reconstruct symbol names and offsets. This methodology is illustrated in the following diagram:

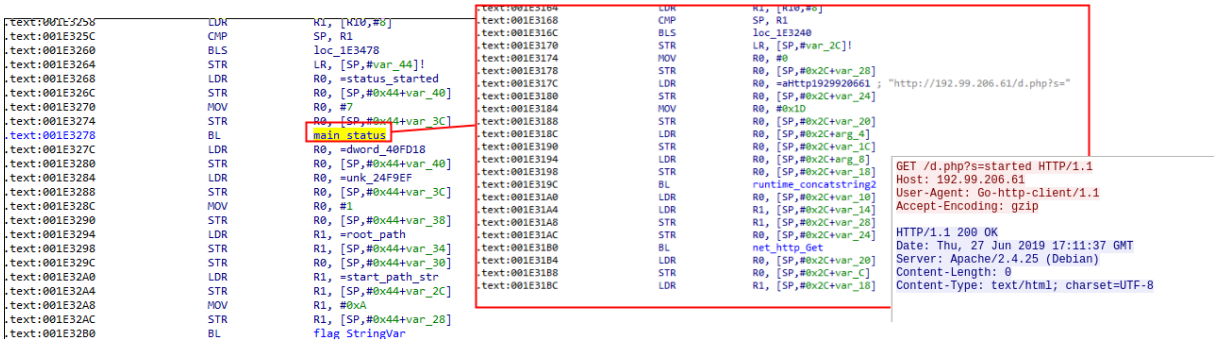


For further insights into populating function names in Go binaries we highly recommend to view Tim Strazzere's presentation and scripts in GitHub which document this technique.

After retrieving Go function names, analyzing the binary becomes much less complex since we can highlight the relevant functions of the application. Let's not forget that the binary is 4MB in size.

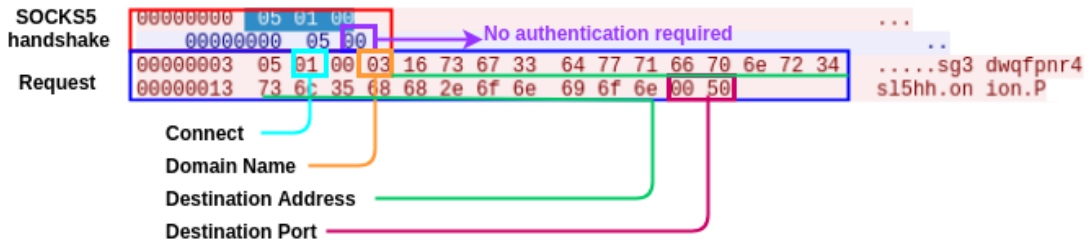


After several cryptography algorithm initializations and parsing of arguments for directory whitelisting and alike functionalities, the malware will send a GET request to the CNC as a means to communicate that a new victim has been compromised and that system locking is taking place:



After sending this GET request, the malware will attempt to retrieve victim keys configuration using a client for the SOCKS proxy protocol version 5.

This proxy will request to connect to an onion domain name. The following represents the relevant packets for this connection:



After successful connection through the proxy to the onion domain, an additional GET request to the ransomware REST API is completed in order to retrieve the RSA public key that will be used to encrypt the file system—a unique Bitcoin wallet and the

ransom note specific to the victim. All of these artifacts seem to be retrieved based on a specific campaign ID.

```
loc_1E2D84                              ; CODE XREF: main_getInfo+52C↓j
                LDR             R0, =aGet ; "GET"
                STR             R0, [SP,#0x90+var_8C]
                MOV             R0, #3
                STR             R0, [SP,#0x90+var_88]
                LDR             R0, [SP,#0x90+arg_4]
                STR             R0, [SP,#0x90+var_84]
                LDR             R0, [SP,#0x90+arg_8]
                STR             R0, [SP,#0x90+var_80]
                MOV             R0, #0
                STR             R0, [SP,#0x90+var_7C]
                MOV             R1, #0
                STR             R1, [SP,#0x90+var_78]
                BL              net_http_NewRequest
                LDR             R0, [SP,#0x90+var_74]
                LDR             R1, [SP,#0x90+var_70]
                LDR             R2, [SP,#0x90+var_6C]
                CMP             R1, #0
                BEQ             loc_1E2E5C
```

```
00000020  47 45 54 20 2f 61 70 69  2f 47 65 74 41 76 61 69   GET /api /GetAvai
00000030  6c 4b 65 79 73 42 79 43  61 6d 70 49 64 2f 31 30   lKeysByC ampId/10
00000040  20 48 54 54 50 2f 31 2e  31 0d 0a 48 6f 73 74 3a    HTTP/1. 1..Host:
00000050  20 73 67 33 64 77 71 66  70 6e 72 34 73 6c 35 68    sg3dwqf pnr4sl5h
00000060  68 2e 6f 6e 69 6f 6e 0d  0a 55 73 65 72 2d 41 67   h.onion. .User-Ag
00000070  65 6e 74 3a 20 47 6f 2d  68 74 74 70 2d 63 6c 69   ent: Go- http-cli
00000080  65 6e 74 2f 31 2e 31 0d  0a 41 63 63 65 70 74 2d   ent/1.1. .Accept-
00000090  45 6e 63 6f 64 69 6e 67  3a 20 67 7a 69 70 0d 0a   Encoding : gzip..
000000A0  0d 0a                                              ..
```

The response from the server is the following:

```
0000000C  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d   HTTP/1.1  200 OK.
0000001C  0a 43 6f 6e 74 65 6e 74  2d 54 79 70 65 3a 20 61   .Content -Type: a
0000002C  70 70 6c 69 63 61 74 69  6f 6e 2f 6a 73 6f 6e 0d   pplicati on/json.
0000003C  0a 44 61 74 65 3a 20 54  68 75 2c 20 32 37 20 4a   .Date: T hu, 27 J
0000004C  75 6e 20 32 30 31 39 20  31 37 3a 31 31 3a 35 31   un 2019  17:11:51
0000005C  20 47 4d 54 0d 0a 43 6f  6e 74 65 6e 74 2d 4c 65    GMT..Co ntent-Le
0000006C  6e 67 74 68 3a 20 35 36  32 0d 0a 0d 0a 7b 22 52   ngth: 56 2....{"R
0000007C  73 61 50 75 62 6c 69 63  4b 65 79 22 3a 22 2d 2d   saPublic Key":"--
0000008C  2d 2d 2d 42 45 47 49 4e  20 52 53 41 20 50 55 42   ---BEGIN  RSA PUB
0000009C  4c 49 43 20 4b 45 59 2d  2d 2d 2d 2d 5c 72 5c 6e   LIC KEY- ----\r\n
000000AC  4d 46 77 77 44 51 59 4a  4b 6f 5a 49 68 76 63 4e   MFwwDQYJ KoZIhvcN
000000BC  41 51 45 42 42 51 41 44  53 77 41 77 53 41 4a 42   AQEBBQAD SwAwSAJB
000000CC  41 4e 4c 74 4e 4d 54 70  75 2f 5a 77 39 79 6e 6c   ANLtNMTp u/Zw9ynl
000000DC  68 46 4d 43 37 35 35 45  68 37 7a 4b 38 33 52 76   hFMC755E h7zK83Rv
000000EC  37 67 31 45 35 61 37 4b  77 67 44 2f 75 36 53 45   7g1E5a7K wgD/u6SE
000000FC  67 76 37 6c 31 43 6a 6f  6c 67 43 41 4c 52 68 33   gv7l1Cjo lgCALRh3
0000010C  47 79 30 72 35 61 59 62  6d 51 50 48 6c 39 69 6f   Gy0r5aYb mQPHl9io
0000011C  38 45 48 56 38 75 38 43  41 77 45 41 41 51 3d 3d   8EHV8u8C AwEAAQ==
0000012C  5c 72 5c 6e 2d 2d 2d 2d  2d 45 4e 44 20 52 53 41   \r\n---- -END RSA
0000013C  20 50 55 42 4c 49 43 20  4b 45 59 2d 2d 2d 2d 2d    PUBLIC  KEY-----
0000014C  5c 72 5c 6e 22 2c 22 42  74 63 50 75 62 6c 69 63   \r\n","B tcPublic
0000015C  4b 65 79 22 3a 22 31 37  4d 6e 48 41 48 76 59 75   Key":"17 MnHAHvYu
0000016C  71 54 6d 59 43 59 79 6a  68 45 41 62 34 36 44 68   qTmYCYyj hEAb46Dh
0000017C  39 69 77 31 74 44 76 51  22 2c 22 52 65 61 64 6d   9iw1tDvQ ","Readm
0000018C  65 22 3a 22 41 6c 6c 20  79 6f 75 72 20 64 61 74   e":"All  your dat
0000019C  61 20 68 61 73 20 62 65  65 6e 20 6c 6f 63 6b 65   a has be en locke
000001AC  64 28 63 72 79 70 74 65  64 29 2e 5c 72 5c 6e 48   d(crypte d).\r\nH
000001BC  6f 77 20 74 6f 20 75 6e  63 6c 6f 63 6b 28 64 65   ow to un clock(de
000001CC  63 72 79 70 74 29 20 69  6e 73 74 72 75 63 74 69   crypt) i nstructi
000001DC  6f 6e 20 6c 6f 63 61 74  65 64 20 69 6e 20 74 68   on locat ed in th
000001EC  69 73 20 54 4f 52 20 77  65 62 73 69 74 65 3a 20   is TOR w ebsite:
000001FC  68 74 74 70 3a 2f 2f 73  67 33 64 77 71 66 70 6e   http://s g3dwqfpn
0000020C  72 34 73 6c 35 68 68 2e  6f 6e 69 6f 6e 2f 6f 72   r4sl5hh. onion/or
0000021C  64 65 72 2f 31 37 4d 6e  48 41 48 76 59 75 71 54   der/17Mn HAHvYuqT
0000022C  6d 59 43 59 79 6a 68 45  41 62 34 36 44 68 39 69   mYCYyjhE Ab46Dh9i
0000023C  77 31 74 44 76 51 5c 72  5c 6e 55 73 65 20 54 4f   w1tDvQ\r \nUse TO
0000024C  52 20 62 72 6f 77 73 65  72 20 66 6f 72 20 61 63   R browse r for ac
0000025C  63 65 73 73 20 2e 6f 6e  69 6f 6e 20 77 65 62 73   cess .on ion webs
0000026C  69 74 65 73 2e 5c 72 5c  6e 68 74 74 70 73 3a 2f   ites.\r\ nhttps:/
0000027C  2f 64 75 63 6b 64 75 63  6b 67 6f 2e 63 6f 6d 2f   /duckduc kgo.com/
0000028C  68 74 6d 6c 3f 71 3d 74  6f 72 2b 62 72 6f 77 73   html?q=t or+brows
0000029C  65 72 2b 68 6f 77 2b 74  6f 5c 72 5c 6e 22 7d      er+how+t o\r\n"}
```

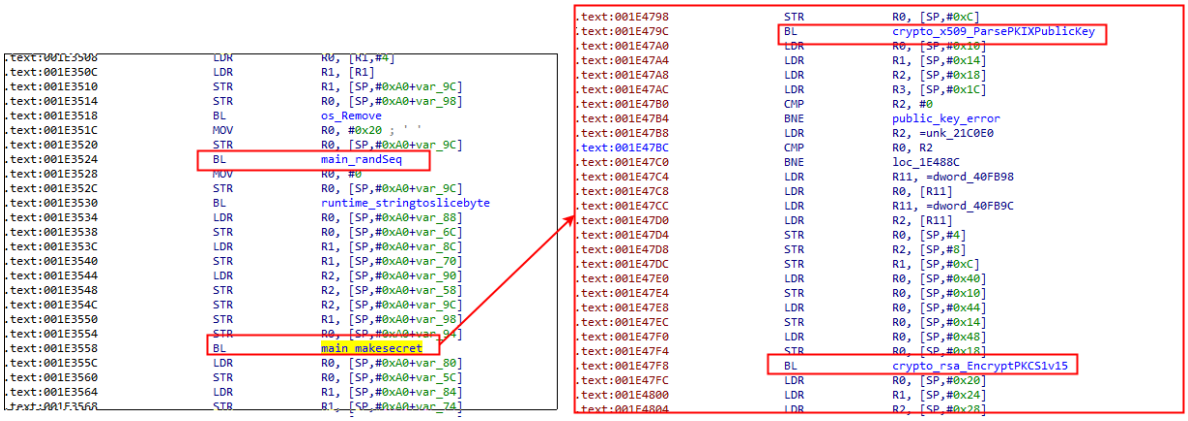
HTTP Response

RSA Public Key

Bitcoin Wallet

Ransom note

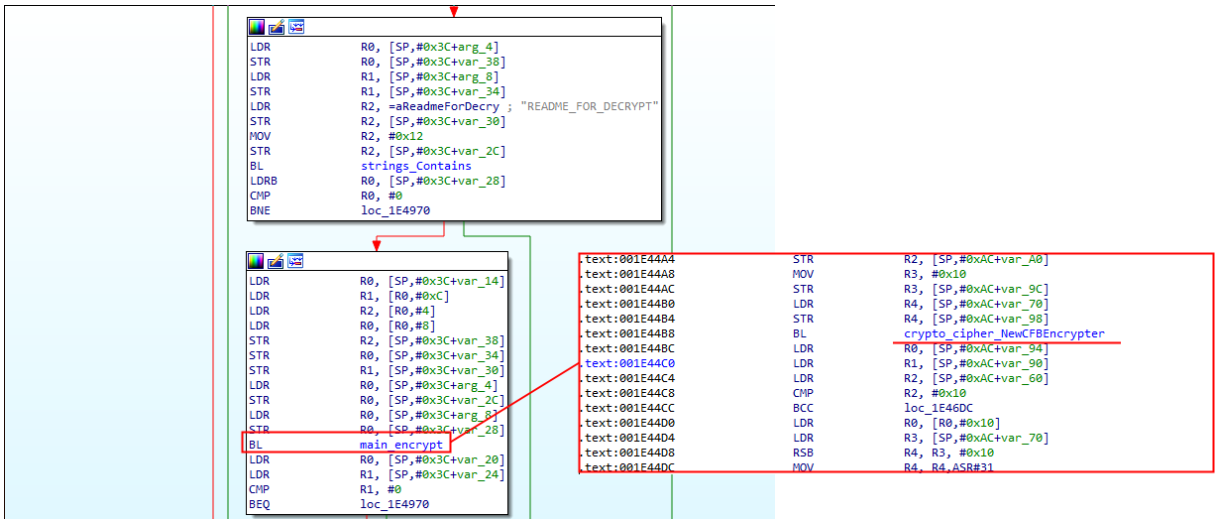
After victim configuration has been retrieved, the malware will proceed to remove itself and then it will parse the retrieved RSA public key.



This RSA public key will be used to encrypt a random sequence of bytes that would be used to encrypt the file system later on. This encrypted key will be base64 encoded and it will be written at the end of the ransom note file called README_FOR_DECRYPT.txt. We also noted that the ransomware distributes a different Bitcoin wallet per each compromised system:

```
All your data has been locked(crypted).
How to unclock(decrypt) instruction located in this TOR website: http://sg3dwqfpnr4sl5hh.onion/order/1F5vvweNaFZwz1ABjRYaJtbHARdjYpHvMM
Use TOR browser for access .onion websites.
https://duckduckgo.com/html?q=tor+browser+how+to

Do NOT remove this file and NOT remove last line in this file!
ndPY1xFVLKpqfMXpq/a31HPf0nOxvGQRSvuA8O/siV1QBo+VVJCs7IxDbRV3dpKxzGt9Cru2Hisk21bq2m6j+g==

All your data has been locked(crypted).
How to unclock(decrypt) instruction located in this TOR website: http://sg3dwqfpnr4sl5hh.onion/order/1KQrAcppntzUuvZ25QWto34G1A7wNTQU8h
Use TOR browser for access .onion websites.
https://duckduckgo.com/html?q=tor+browser+how+to

Do NOT remove this file and NOT remove last line in this file!
gsx9I05YjZ6j3SkoAJ0WRIfMO1nyh3IFRikmKOIyQKRCRBhFGpcNdTPzch7lT6vnoyEu9CnA4bWTaPINIzBFRQ==
```

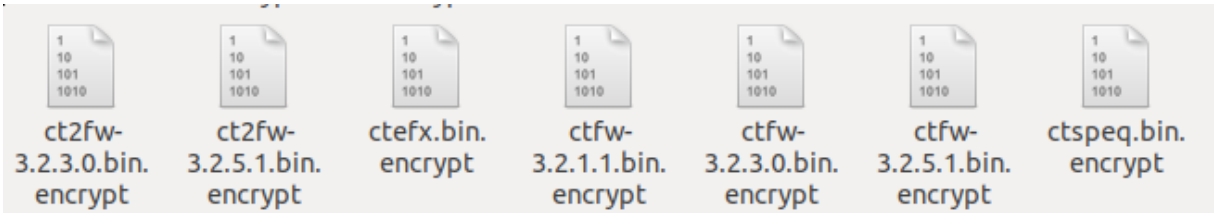
After this file is created, the malware will proceed to execute the locking mechanism by walking the file system encrypting files using AES CFB with the derived encrypted key, avoiding to encrypt the ransom note just created:



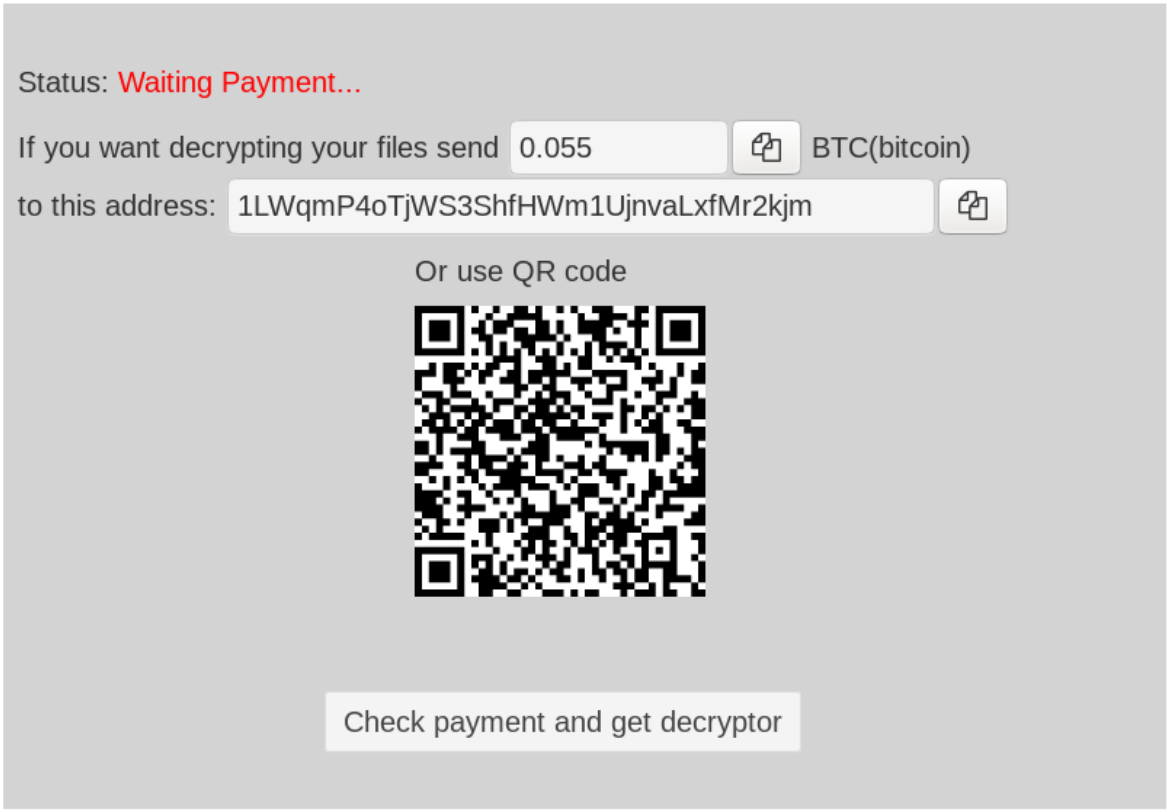
The malware will target files with the following extensions:

```
.rodata:0024FC9E                DCB "aim.alx.ans.apk.apt.arj.aro.arw.asa.asc.ase.asp.asr.att.aty.avi."
.rodata:0024FC9E                DCB "awm.awp.awt.aww.axd.bar.bat.bay.bc6.bc7.big.bik.bin.bit.bkf.bkp."
.rodata:0024FC9E                DCB "bml.bok.bpw.bsa.bwp.bz2.c++.cab.cas.cat.cdf.cdr.cer.cfg.cfm.cfr."
.rodata:0024FC9E                DCB "cha.chm.cms.con.cpg.cpp.cr2.crl.crp.crt.crw.csp.csr.css.csv.cxx."
.rodata:0024FC9E                DCB "dap.das.dat.db0.dba.dbf.dbm.dbx.dcr.der.dll.dml.dmp.dng.doc.dot."
.rodata:0024FC9E                DCB "dwg.dwk.dwt.dxf.dxg.ece.eml.epk.eps.erf.esm.ewp.far.fdb.fit.flv."
.rodata:0024FC9E                DCB "fmp.fos.fpk.fsh.fwp.gdb.gho.gif.gne.gpg.gsp.gxk.hdm.hkx.htc.htm."
.rodata:0024FC9E                DCB "htx.hxs.idc.idx.ifx.iqy.iso.itl.itm.iwd.iwi.jcz.jpe.jpg.jsp.jss."
.rodata:0024FC9E                DCB "jst.jvs.jws.kdb.kdc.key.kit.ksd.lbc.lbf.lrf.ltx.lvl.lzh.m3u.m4a."
.rodata:0024FC9E                DCB "map.max.mdb.mdf.mef.mht.mjs.mlx.mov.moz.mp3.mpd.mpp.mvc.mvr.myo."
.rodata:0024FC9E                DCB "nba.nbf.ncf.ngc.nod.nrw.nsf.ntl.nv2.nxg.nzb.oam.odb.odc.odm.odp."
.rodata:0024FC9E                DCB "ods.odt.ofx.olp.orf.oth.p12.p7b.p7c.pac.pak.pdb.pdd.pdf.pef.pem."
.rodata:0024FC9E                DCB "pfx.pgp.php.png.pot.ppj.pps.ppt.prf.pro.psd.psk.psp.pst.psw.ptw."
.rodata:0024FC9E                DCB "ptx.pub.qba.qbb.qbo.qbw.qbx.qdf.qfx.qic.qif.qrm.r3d.raf.rar.raw."
.rodata:0024FC9E                DCB "re4.rim.rjs.rsn.rss.rtf.rw2.rw3.rwl.rwp.saj.sav.sdb.sdc.sdf.sht."
.rodata:0024FC9E                DCB "sid.sie.sis.sko.slm.snx.spc.sql.sr2.src.srf.srw.ssp.stc.stl.stm."
.rodata:0024FC9E                DCB "stp.sum.svc.svg.svr.swz.sxc.t12.t13.tar.tax.tbl.tbz.tcl.tgz.tib."
.rodata:0024FC9E                DCB "tor.tpl.txt.ucf.upk.url.vbd.vbo.vcf.vdf.vdi.vdw.vlp.vmx.vpk.vrt."
.rodata:0024FC9E                DCB "vtf.w3x.wav.wb2.wbs.wdb.web.wgp.wgt.wma.wml.wmo.wmv.woa.wpd.wpp."
.rodata:0024FC9E                DCB "wps.wpx.wrf.x3f.x_t.xbl.xbm.xht.xla.xlk.xll.xlm.xls.xlt.xlw.xml."
.rodata:0024FC9E                DCB "xpd.xpm.xps.xss.xul.xwd.xws.xxx.zfo.zip.zul.zvz"
.rodata:0025020D                DCB 0x30 ; 0            ; DATA XREF: net_http_http2FrameHeader_writeDebug+168↑o
```

After encryption, the malware will rename the affected files so that they will be prefixed with '.encrypt':



In order for system decryption to take place the base64 encoded random sequence encrypted with the RSA public key will be needed to be sent to the ransomware operator via the onion domain site after paying the demanded ransom:



After system locking has taken place, the ransomware will communicate that it has finished with the victim once again to the CNC:

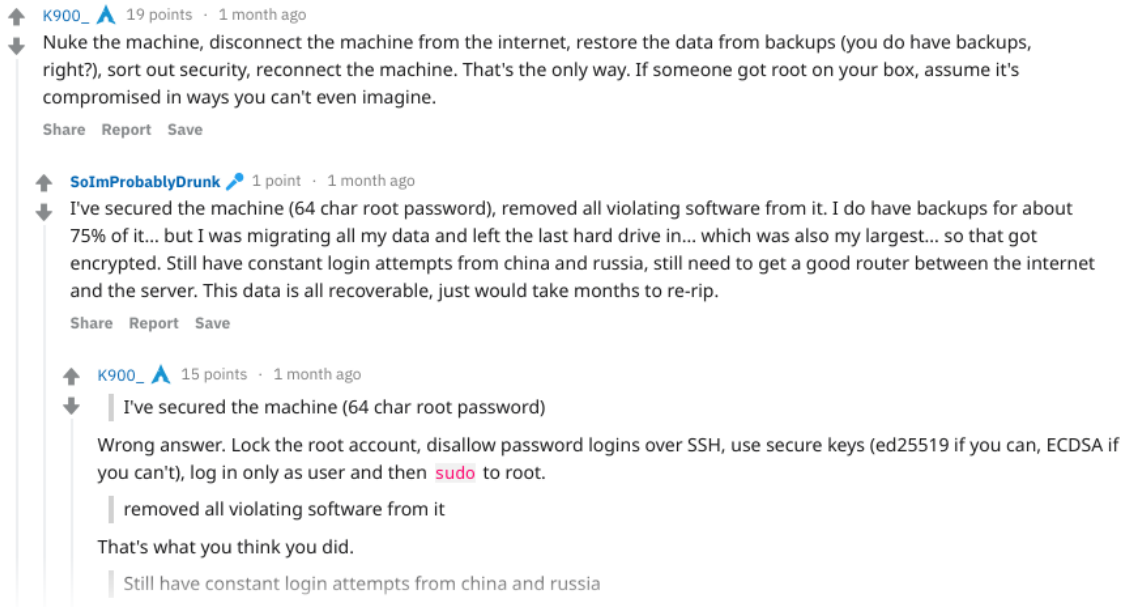
```
.text:001E38CC loc_1E38CC                              ; CODE XREF: main_main+3B4↑j
.text:001E38CC                 LDR             R0, =aDone ; "done"
.text:001E38D0                 STR             R0, [SP,#0xA0+encrypted_rand_sequence]
.text:001E38D4                 MOV             R0, #4
.text:001E38D8                 STR             R0, [SP,#0xA0+var_98]
.text:001E38DC                 BL              main_status
.text:001E38E0                 LDR             PC, [SP+0xA0+var_A0],#0xA0
```

```
GET /d.php?s=done HTTP/1.1
Host: 192.99.206.61
User-Agent: Go-http-client/1.1
Accept-Encoding: gzip
```

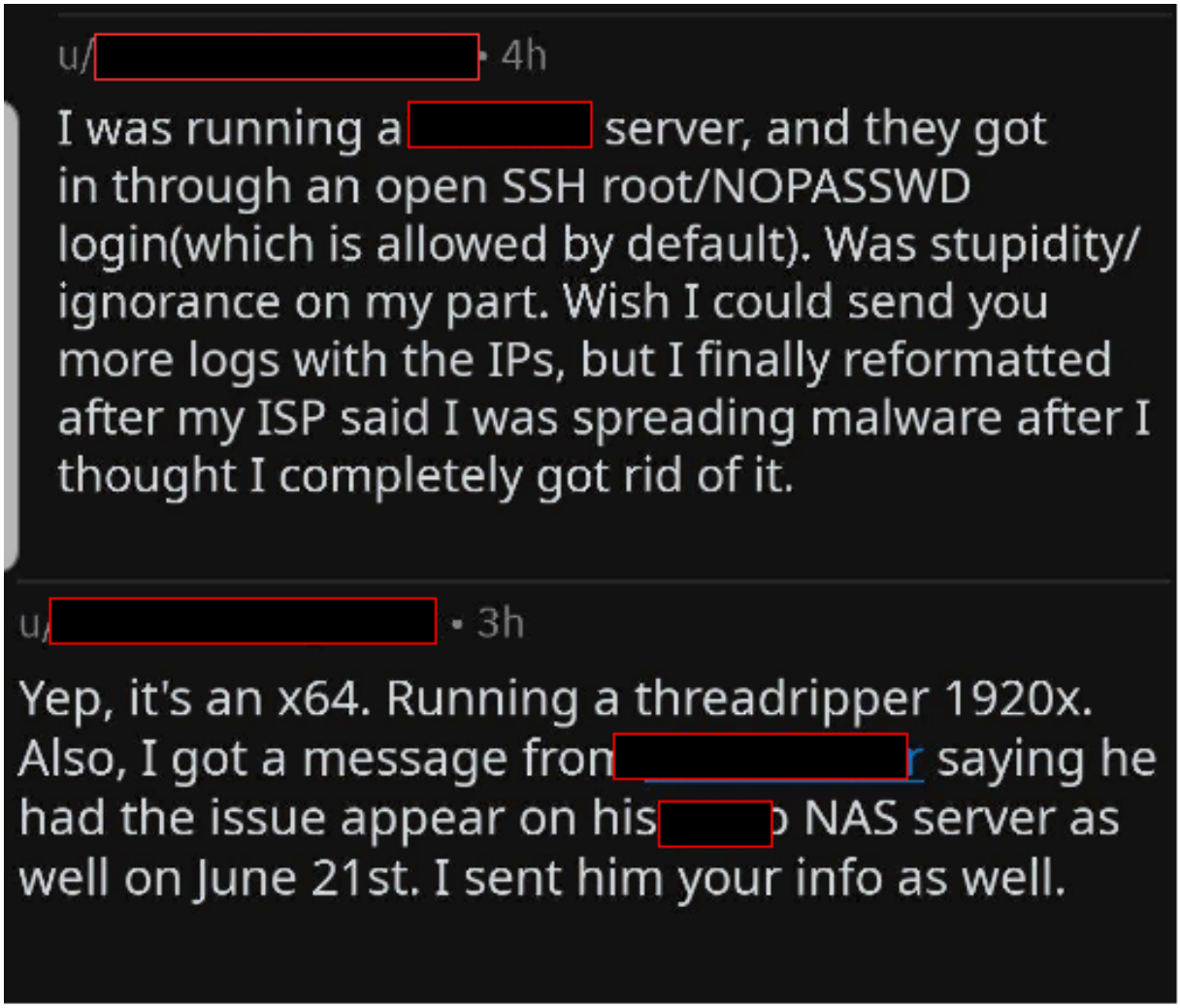
Looking Outside of the Binary

One of our intended goals that we wanted to achieve when analyzing QNAPCrypt was to assess the scale of victims the ransomware was dealing with.

We were able to find a Reddit thread in which we contacted some of the affected victims:

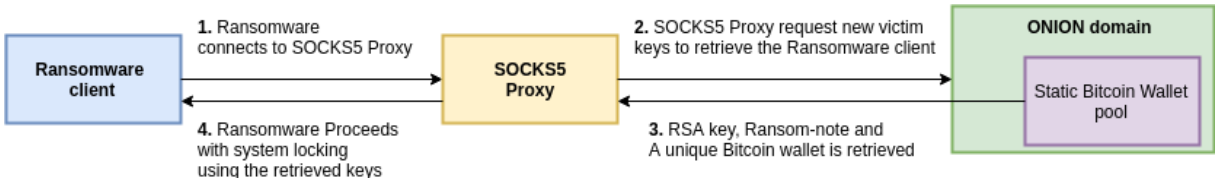


While talking to some of the victims related to the various campaigns of this malware, we were able to identify the initial attack vector as SSH brute force attacks and that they were targeting mainly NAS server providers, which corresponds to how the attacker has chosen to label this malware:



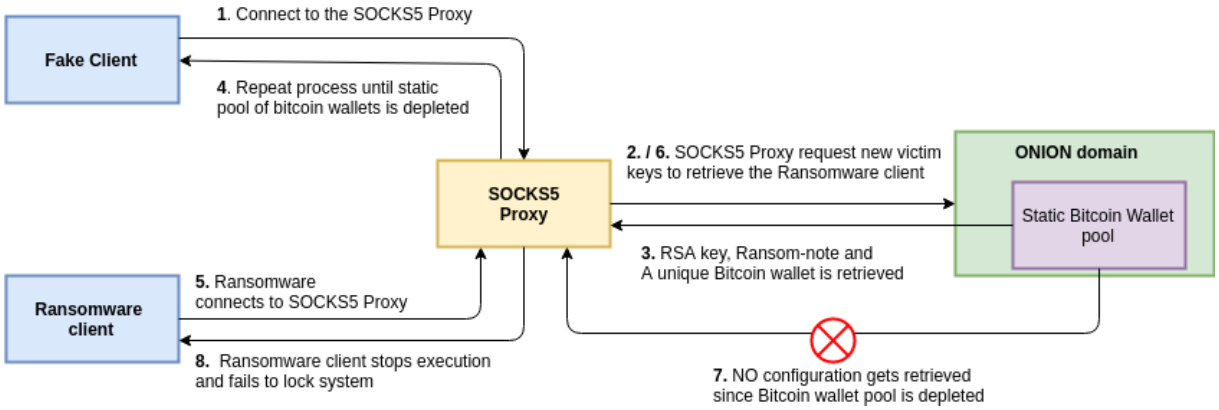
After making these findings we studied their infrastructure to determine if there was anything we could do to interact with this threat actor's operations.

While researching the ARM instance of the malware, we observed that there was a request through their REST API in order to retrieve new victim configuration keys as previously discussed. The following diagram is a high level overview of the ransomware operation:



The connection to the SOCKS5 proxy is completed without any authentication enforced, and anyone would have the capability to connect to it.

Therefore, we decided to interact with the ransomware infrastructure in order to retrieve configuration keys and potentially temporarily shut down the operation of the ransomware to prevent infection of future victims that were compromised by instances of the ransomware that followed the previous design architecture:



This idea simply abuses the fact that no authentication is enforced to connect to the SOCKS5 proxy as previously mentioned. Since the authors behind this ransomware were delivering one Bitcoin wallet per victim from a static pool of already generated wallets, we could replicate the infection packets to retrieve all of the wallets until they had no further wallets under their control. Therefore, when a genuine infection would occur, the ransom client would not be able to retrieve configuration artifacts.

We wrote the following script in order to implement the methodology described above:

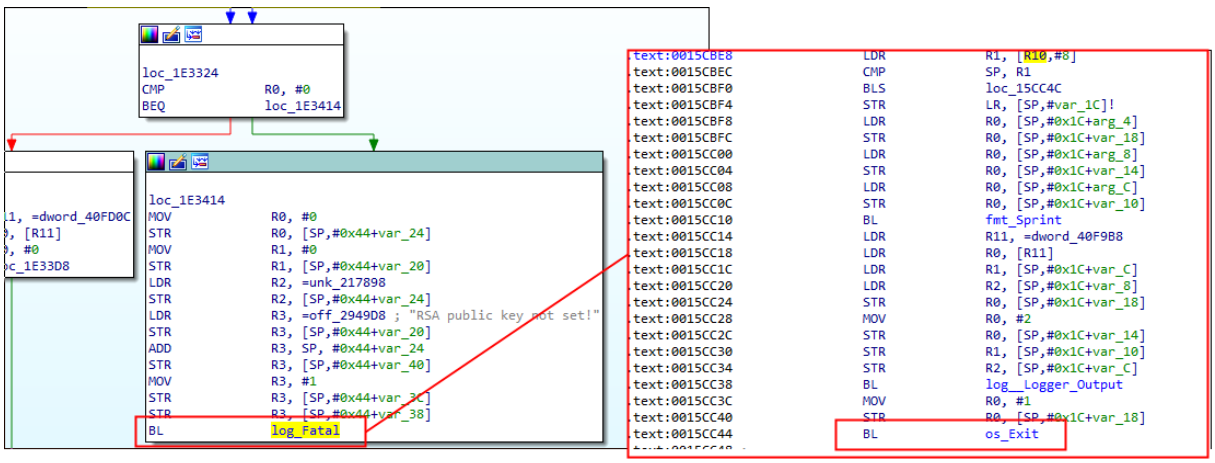
```
import socket
import hexdump
import json
import sys

HOST = '192.99.206.61'
PORT = 65000

for i in range(15):
    BTC_WALLETS = list()
    while True:
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.connect((HOST, PORT))

        s.send(b'\x05\x01\x00')
        data = s.recv(1024)
        hexdump.hexdump(data)

        s.send(b'\x05\x01\x00\x03\x16' + b'sg3dwqfpnr4sl5hh.onion\x00' +
b'\x50')
        data = s.recv(1024)
```

```
hexdump.hexdump(data)

s.send(b'GET /api/GetAvailKeysByCampId/%.2d HTTP/1.1\x0d\x0a' % i +
        b'Host: sg3dwqfpnr4sl5hh.onion\x0d\x0a' +
        b'User-Agent: http/2\x0d\x0a' +
        b'Accept-Encoding: gzip\x0a\x0d\x0a')
data = s.recv(1024)
print '[+] Campaign id %.2d' % i
hexdump.hexdump(data)

try:
    data = json.loads(data[data.find('{'):])
    print data['BtcPublicKey']
    s.close()

    if data['BtcPublicKey'] not in BTC_WALLETS:
        BTC_WALLETS.append(data['BtcPublicKey'])
    else:
        sys.exit()

except ValueError as e:
    print "[+] CAMPAIN HAS NO WALLETS LEFT"
    with open("wallets_%0d.txt" % i, 'w+') as fd:
        for wallet in BTC_WALLETS:
            fd.write(wallet+'\n')
    break
```

We were able to collect a total of 1,091 unique wallets meant to be delivered to new victims distributed among 15 different campaigns.

```
ulexec intezer ~ Linux.QnapCrypt $ cat wallets* | wc -l
1091
ulexec intezer ~ Linux.QnapCrypt $
```

Furthermore, by depleting the attacker's stored Bitcoin wallets we were able to stop this malware from infecting new victims temporarily, since if there is a failure to parse the RSA public key the client will just exit:



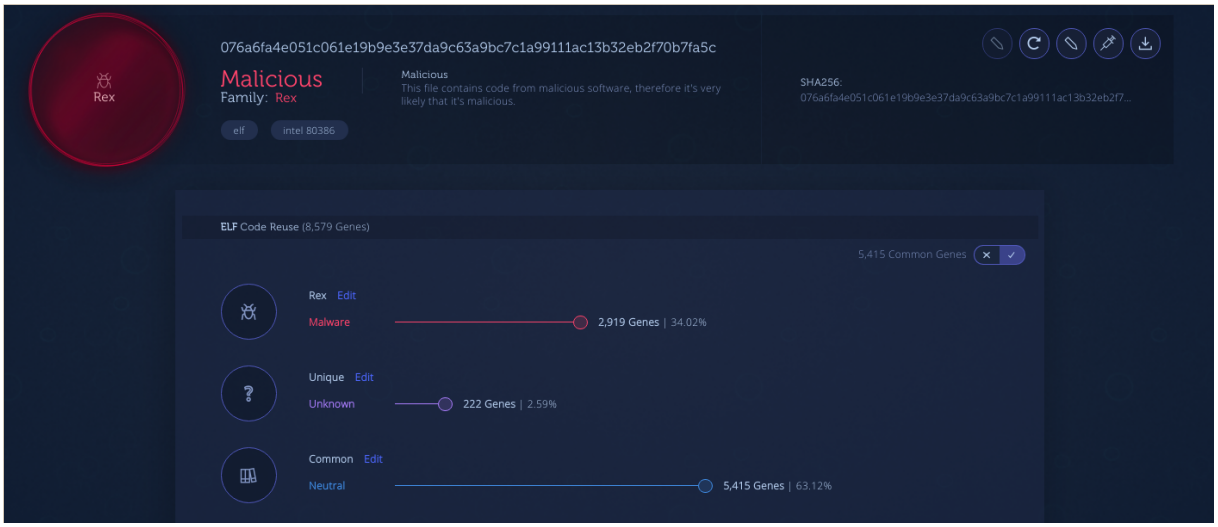
The following screenshot shows the packets that the onion domain will retrieve after the entire static Bitcoin wallet pool was depleted:

```
00000000: 05 00                                           ..
00000000: 05 00 00 01 00 00 00 00  00 00                  ..........
00000000: 48 54 54 50 2F 31 2E 31  20 32 30 30 20 4F 4B 0D  HTTP/1.1 200 OK.
00000010: 0A 43 6F 6E 74 65 6E 74  2D 54 79 70 65 3A 20 61  .Content-Type: a
00000020: 70 70 6C 69 63 61 74 69  6F 6E 2F 6A 73 6F 6E 0D  pplication/json.
00000030: 0A 44 61 74 65 3A 20 4D  6F 6E 2C 20 30 31 20 4A  .Date: Mon, 01 J
00000040: 75 6C 20 32 30 31 39 20  31 33 3A 35 38 3A 30 37  ul 2019 13:58:07
00000050: 20 47 4D 54 0D 0A 43 6F  6E 74 65 6E 74 2D 4C 65   GMT..Content-Le
00000060: 6E 67 74 68 3A 20 30 0D  0A 0D 0A                 ngth: 0....
[+] CAMPAIN HAS NO WALLETS LEFT
```

The HTTP request returns a 200 but with a content length of 0, therefore failing to retrieve configuration, and thus the ransomware client stops execution. This implies that we were able to identify an easy method to prevent further infections of this ransomware by constantly depleting its static bitcoin wallet pool.

Attribution and Attackers Reaction

After several days of continuously DoS'ing QNAPCrypt clients, we encountered another QNAPCrypt sample—but this time targeting x86 systems.



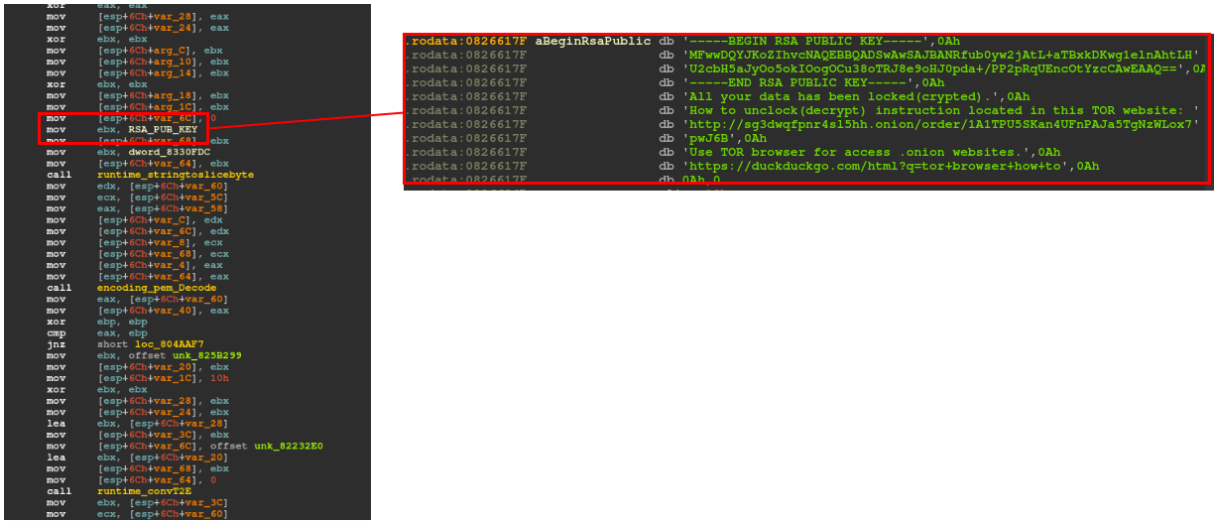
Based on Genetic Malware Analysis, we observed that this specific implant reused a large portion of code with old instances of x86 Linux.Rex builds. Linux.Rex is known

for deploying exploits against Drupal servers in 2016, in order to conduct ransomware and DDoS operations.

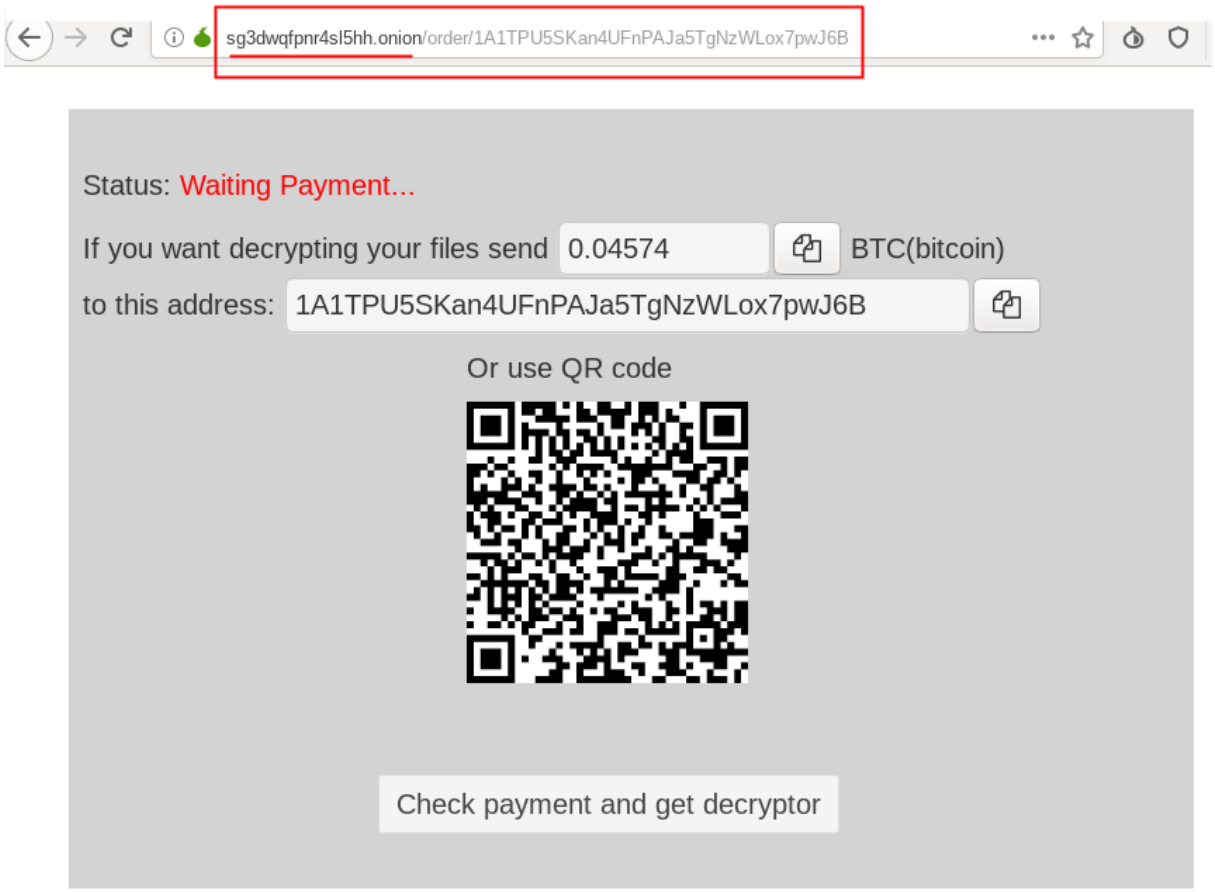
The following represents some of the code similarities between Linux.Rex and newer QNAPCrypt variants:

Although both implants implement different functionality, it is noticeable that both were written in a similar manner.

Furthermore, we can observe similarities with the ARM instance of QNAPCrypt but with a major difference—the RSA public key, Bitcoin wallet and ransom note are hardcoded in the binary:



We can also see that the hardcoded onion domain is exactly the same as in the ARM variant, and the site design to pay the ransom is also the same, although the demanded ransom in Bitcoin seems to be lower than in previous variants:



We interpret the discovery of these newer instances with hardcoded configuration to be a response from the threat actors behind this campaign to attempt to circumvent the DoS that their non connectionless instances were suffering. This implied that they were forced to change their implants and to centralize their bitcoin wallets, making the tracking of their income via their ransomware campaigns more convenient.

Conclusion

We have covered the operation of the QNAPCrypt ransomware, and how we were able to find design flaws to prevent the malware from running in newer victims' machines and forcing the attackers behind the malware to update their implants in order to circumvent these flaws.

Additionally, Golang malware seems to be on the rise, since it appears to be a very convenient language to create cross-platform malware.

Furthermore, we have discussed how Linux ransomware has slightly different targets than Windows ransomware, in this case targeting NAS servers rather than Linux endpoints.

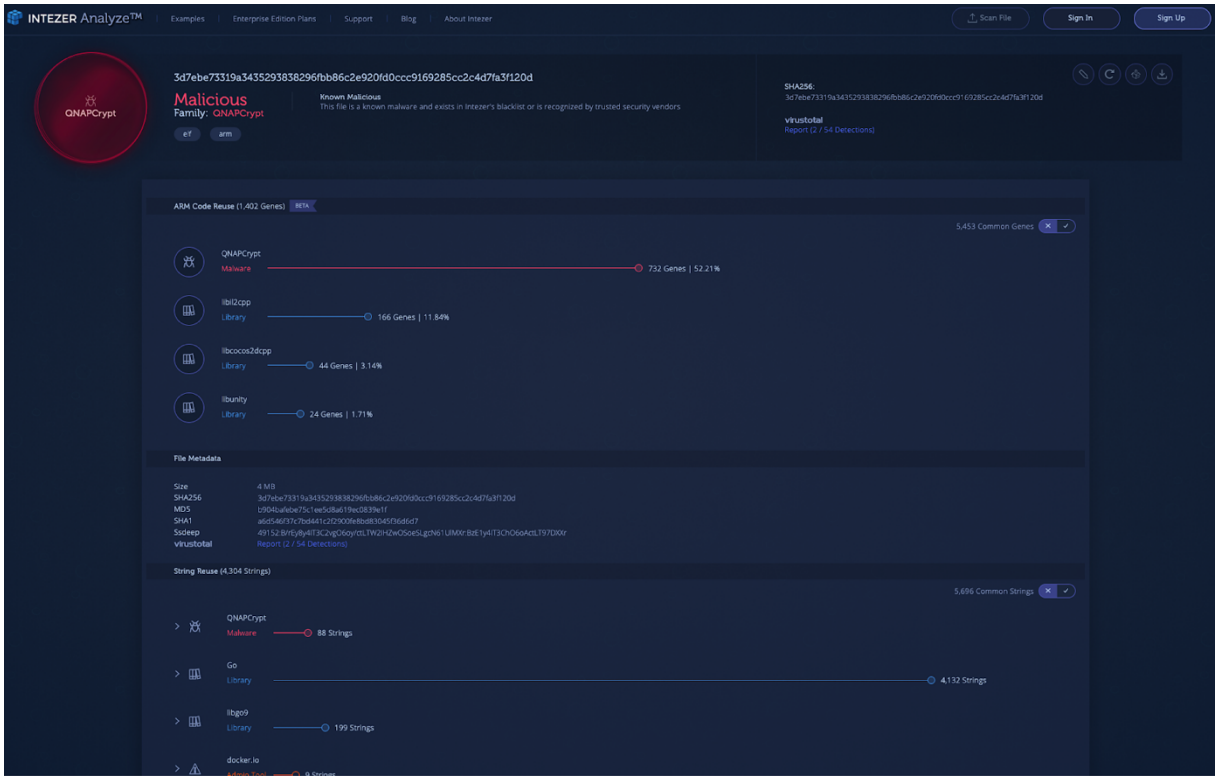
Unfortunately detection rates of QNAPCrypt are low, and the ransomware could create significant monetary losses and economic damage in comparison to other types of Linux threats.

We have created a custom YARA signature for detecting future variants of QNAPCrypt.

Genetic Analysis

The QNAPCrypt malware variants are now indexed in Intezer's genetic database. If you have a suspicious file that you suspect to be QNAPCrypt or other malware from the Rex group, you can upload it to Intezer Analyze to detect code reuse to this

threat family and many others. You are welcome to [try it for free in our community edition](#).



Genetic Analysis of the QNAPCrypt ARM variant

IOCs

sg3dwqfpnr4sl5hh[.]onion

192.99.206[.]61

3d7ebe73319a3435293838296fbb86c2e920fd0ccc9169285cc2c4d7fa3f120d

076a6fa4e051c061e19b9e3e37da9c63a9bc7c1a99111ac13b32eb2f70b7fa5c