Reversemode

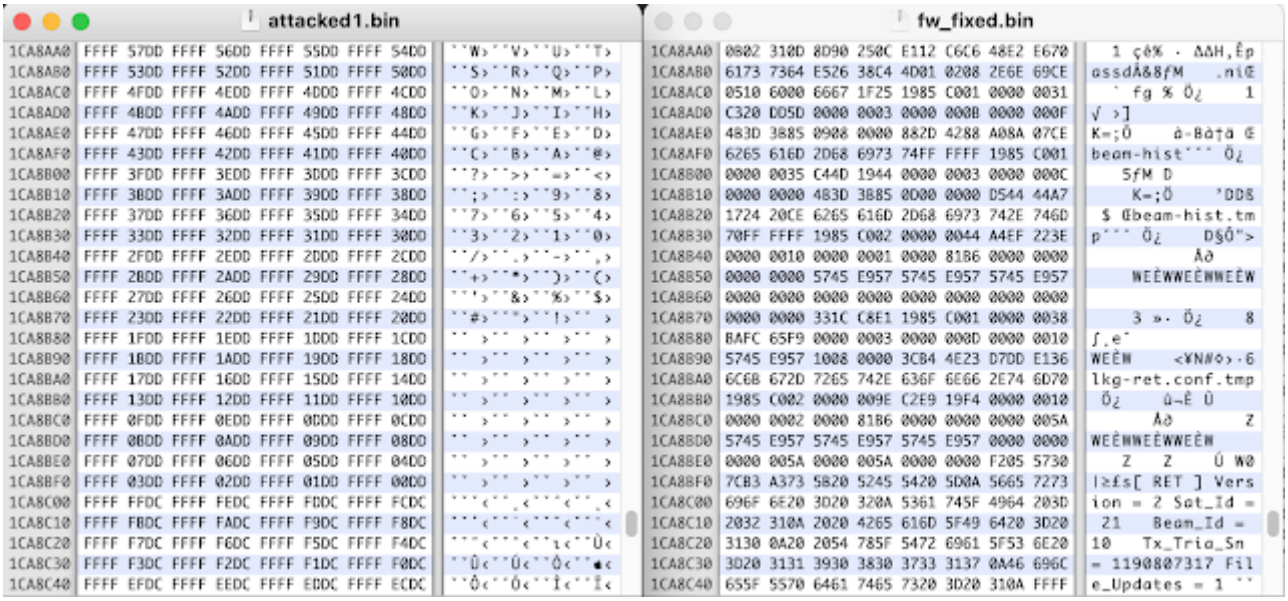
VIASAT incident: from speculation to technical details.

34 days after the incident, yesterday Viasat published a statement providing some technical details about the attack that affected tens of thousands of its SATCOM terminals. Also yesterday, I eventually had access to two Surfbeam2 modems: one was targeted during the attack and the other was in a working condition. Thank you so much to the person who disinterestedly donated the attacked modem.

I've been closely covering this issue since the beginning, providing a plausible theory based on the information that was available at that time, and my experience in this field. Actually, it seems that this theory was pretty close to what really happened.

> Subsequent investigation and forensic analysis identified a ground-based network intrusion by an attacker exploiting a misconfiguration in a VPN appliance to gain remote access to the trusted management segment of the KA-SAT network. The attacker moved laterally through this trusted management network to a specific network segment used to manage and operate the network, and then used this network access to execute legitimate, targeted management commands on a large number of residential modems simultaneously. Specifically, these destructive commands overwrote key data in flash memory on the modems, rendering the modems unable to access the network, but not permanently unusable.

Fortunately, now we can move from just pure speculation into something more tangible, so I dumped the flash memory for both modems (Spansion S29GL256P90TFCR2) and the differences were pretty clear. In the following picture you can see 'attacked1.bin', which belongs to the targeted modem and 'fw_fixed.bin', coming from the modem in working conditions.



A destructive pattern, that corrupted the flash memory rendering the SATCOM modems inoperable, can be observed on the left, confirming what Viasat stated yesterday.

After verifying the destructive attack, I'm now statically analyzing the firmware extracted from the 'clean' modem. Firmware version is 3.7.3.10.9.

Besides talking about a 'management network' and 'legitimate management commands', Viasat did not provide any specific details about this. In my previous blog post I introduced the theory that probably 'TR069' was the involved management protocol.

Obviously, I can't completely confirm this scenario but I'll try to elaborate my reasoning.

Attacking via a management protocol

I think there are two main options: either the attackers abused a MAC management protocol or an application layer one.

For the MAC case ('ut_mac' binary), in general terms, the attackers would have required an even more privileged access to either the NOC or the Ground Stations, probably in a persistent way via malware. I guess that this kind of privileged access would have been enough to limit the attack to Ukraine, instead of knocking out half Europe. As a result, I'm inclined to think this was not the case.

On the other hand, a 'misconfigured VPN' that enabled the attackers to reach the 'management segment' and execute 'commands' seems to be more related to an application layer management protocol: SNMP or TR069.

SNMP

```
trapcommunity public
rocommunity public  default -V xperf
rwcommunity private default -V xperf

engineIDType 3
engineIDNic eth0

createUser viasat SHA "75@t1133" AES

rwuser viasat

dlmod vsatSb2Ut /usr/local/share/snmp/dlmod/vsatSb2Ut.so
```

An initial analysis of 'vsatSb2Ut.so' shows that the implemented MIB does not seem to provide the required functionality to perform this kind of attack.

Function name	Segment	Start
handle_vsatSb2UtCspConnected	.text	00010E28
handle_vsatSb2UtCspDegradedReason	.text	00010EE0
handle_vsatSb2UtCspDisconnectReason	.text	00010C00
handle_vsatSb2UtCspDisconnectTime	.text	00010CB8
handle_vsatSb2UtCspDosEventDetected	.text	00010868
handle_vsatSb2UtCspLastDosEvent	.text	000107B0
handle_vsatSb2UtCspLastFlPktsLost	.text	000109D8
handle_vsatSb2UtCspLastRlPktsLost	.text	00010920
handle_vsatSb2UtCspLastWebpageLoadDuration	.text	00010A90
handle_vsatSb2UtCspLastWebpageLoadTime	.text	00010B48
handle_vsatSb2UtCspOnlineTime	.text	00011038
handle_vsatSb2UtCspProcessRunning	.text	00010F98
handle_vsatSb2UtCspRetransReceivePkts	.text	000104D0
handle_vsatSb2UtCspRetransSendPkts	.text	00010588
handle_vsatSb2UtCspStartTime	.text	00010640
handle_vsatSb2UtMacConfAaaName	.text	0000A840
handle_vsatSb2UtMacConfDumpBB	.text	0000AAF0

I would initially discard this option.

TR069

As suggested in the previous blog post, the Surfbeam2 modems are deployed with the Axiros' AXACT client. The nature of the operations performed by TR069 clients makes them very convenient for an attack of this type.

```
<obj><n>Device</n><a></a>
<par><n>RootDataModelVersion</n><v>2.6</v><o>0</o><a>0</a></par>
<obj><n>ManagementServer</n><a></a>
<par><n>EnableCWMP</n><v>1</v><o>0</o><a>0</a></par>
<par><n>URL</n><v>http://10.88.0.157:9675/live/CPEManager/CPEs/genericTR69</v><o>0</o><a>0</a></par>
<par><n>Username</n><v>admin</v><o>0</o><a>0</a></par>
<par><n>Password</n><v>admin</v><o>0</o><a>0</a></par>
<par><n>PeriodicInformEnable</n><v>0</v><o>0</o><a>0</a></par>
<par><n>PeriodicInformInterval</n><v>3600</v><o>0</o><a>0</a></par>
<par><n>ConnectionRequestURL</n><v>http://:8089</v><o>2</o><a>0</a></par>
<par><n>ConnectionRequestUsername</n><v>admin</v><o>0</o><a>0</a></par>
<par><n>ConnectionRequestPassword</n><v>admin</v><o>0</o><a>0</a></par>
</obj>
```

cwmpdefault.xml

By reverse engineering the 'cwmpclient' binary it is possible to recover the Viasat's TR069 data model, analyze how it has been implemented as well as how it communicates with other components to perform the required actions (via IPC queues).

Unfortunately, it does not look good. So far, I would highlight two issues:

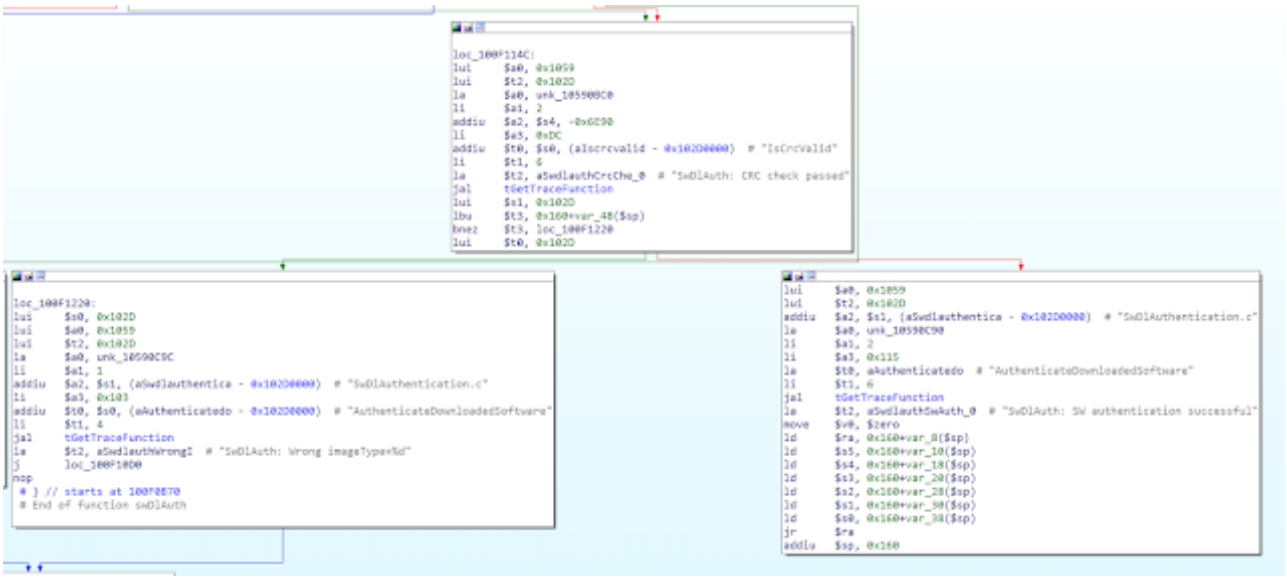
1. New firmware is not properly validated after being downloaded by the TR069 client. This means that the new firmware does not contain a cryptographically secure signature.

```
#!/bin/sh
# TR_069_SW_INSTALL.SH
# TR_0069 software upgrade script
# Authenticates software file, and installs into flash.
#
# Called from CLI/TR_069 client SW DL
#
SW_DOWNLOAD_DIR="/tmp"
SYSLOG_TAG="TR69_SW_INSTALL"
MIMIF="/root/mimIf"

usage()
{
    name=`basename $0`
    echo ""
    echo "****************NOTE: ut_mac must be running.****************"
    echo ""
    echo "usage: filename "
    echo "       filename: full filename (including path) to the software image. "
    echo "                 Must be a local file."
}
```

...

```
# filename arg must not include path
FILENAME=`basename $1`

# SW_FILE includes path. Scripts expect SW to be in /tmp
SW_FILE=$SW_DOWNLOAD_DIR/$FILENAME

mv $local_file $SW_FILE

sw_unwrap.sh $SW_FILE 1
if [ $? -ne 0 ]; then
    logMsg error "Error unwrapping software"
    exit 1
fi

logMsg debug "Validating SW"
swValidate $FILENAME > /dev/null 2>&1
RESULT=$?

if [ $RESULT -eq 1 ]
then
    logMsg notice "SW validated. Installing."
    /sbin/sw_install.sh $FILENAME -i -tr
```

'swValidate' is implemented in 'ut_mac' binary, which merely validates a CRC.



2. *** Updated ***

A deeper look at the 'ut_app_execute_operation' function revealed that it is implementing a functionality that enables the ACS to install (upload and run) arbitrary binaries on the modem, without requiring either a signature verification or a complete firmware upgrade.

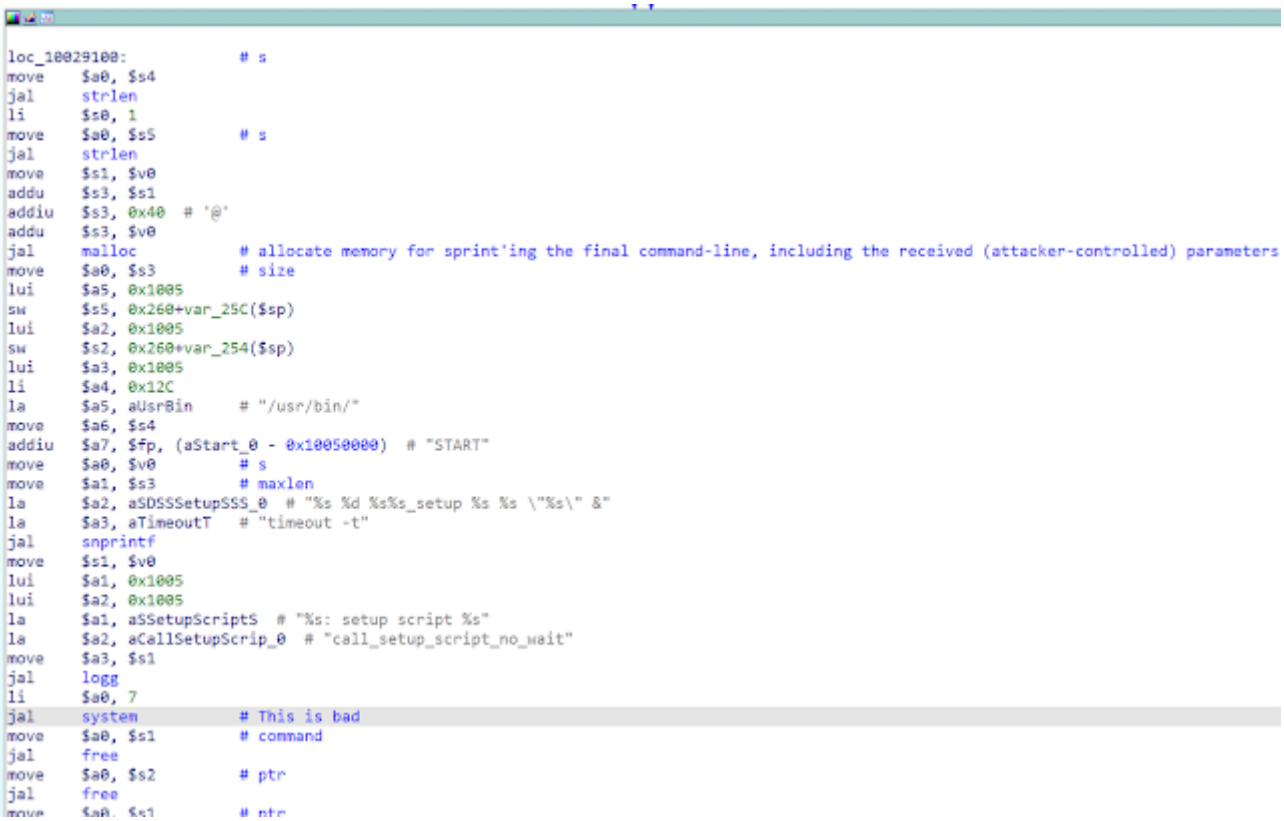
This functionality seems to match both the Viasat statement as well as the approach to deploy the 'AcidRain' wiper described by SentinelOne.

```
/*
    Binary:        '/usr/local/sbin/cwmpclient
    Function name: 'ut_app_execute_operation'
    Description:   'Axiros AXACT TR069 Client'

    TR069 Data Model: X-VIASAT_COM_app

    - Intended Functionality -

    It enables the ACS to upload and run custom binaries into the modem,
    without requiring a firmware upgrade.

    Related script: '/usr/bin/app_img_dwnid'

    - Potential Impact -

    Malicious actors may have abused this legitimate functionality
    to massively deploy the 'AcidRain' wiper to the Viasat Modems.

*/

lVar12 = strcmp(pcVar3,"INSTALL");
if (lVar12 == 0) {
  pcVar3 = (char *)dmos_getObjectParameterValue(param_1,"ImageID");
  if ((pcVar3 != (char *)0x0) && (*pcVar3 != '\0')) {
    pcVar5 = (char *)dmos_getObjectParameterValue(param_1,"ImageURL");
    if ((pcVar5 != (char *)0x0) && (*pcVar5 != '\0')) {
LAB_10029250:
      create_config_file(uVar1,pcVar3,pcVar5,pcVar9); // symbol edited
      uVar1 = execute_app_img_dwnld(auStack592); // symbol edited
      return uVar1;
    }
```

'/usr/bin/app_img_dwnid'

```
438  mainloop()
439  {
440      local status
441      local retry=0
442
443      # Validate the config file
444      validate_config
445
446      # Set default thread stack size
447      ulimit -s 1024
448
449      while :
450      do
451          # Keep trying to get an image until the link is found
452          get_image
453      status=$?
454
455      if [ "$status" = "2" ]
456      then
457          logger -s -t app_img_dwnld "Integrity check failed - don't retry"
458        # failure was an integrity check - stop trying to re-download
459          return 1
460      fi
461
462          if [ ! -f $INSTALL_DIR/$IMAGE_ID ]
463          then
464              logger -s -t app_img_dwnld "Failed to find image birthmark, retrying in 30 seconds"
465          if [ $retry -lt $INSTALL_RETRY_MAX ]
466          then
467          retry=`expr $retry + 1`
468          sleep 30
469              else
470          logger -s -t app_img_dwnld "Installing $APP_NAME failed after $INSTALL_RETRY_MAX times"
471          return 1
472          fi
473          else
474              break
475          fi
476      done
477
478      # Birthmark is verified with IMAGE_ID. Good image.
479      touch $IMG_INSTALLED
480
481      logger -s -t app_img_dwnld "Executing $APP_NAME install command: $INSTALL_CMD"
482      $INSTALL_CMD
```

Additionally, there are multiple command injection vulnerabilities that can be trivially exploited from a malicious ACS (or someone with the same privileged position in the network), i.e 'ut_app_execute_operation' for the custom 'X_VIASAT-COM_app' object ('cwmpclient')



```
loc_10029100:           # s
move    $a0, $s4
jal     strlen
li      $s0, 1
move    $a0, $s5        # s
jal     strlen
move    $s1, $v0
addu    $s3, $s1
addiu   $s3, 0x40  # '@'
addu    $s3, $v0
jal     malloc          # allocate memory for sprint'ing the final command-line, including the received (attacker-controlled) parameters
move    $a0, $s3        # size
lui     $a5, 0x1005
sw      $s5, 0x260+var_25C($sp)
lui     $a2, 0x1005
sw      $s2, 0x260+var_254($sp)
lui     $a3, 0x1005
li      $a4, 0x12C
la      $a5, aUsrBin    # "/usr/bin/"
move    $a6, $s4
addiu   $a7, $fp, (aStart_0 - 0x10050000)  # "START"
move    $a0, $v0        # s
move    $a1, $s3        # maxlen
la      $a2, aSDSSSetupSSS_0  # "%s %d %s%s_setup %s %s \"%s\" &"
la      $a3, aTimeoutT  # "timeout -t"
jal     snprintf
move    $s1, $v0
lui     $a1, 0x1005
lui     $a2, 0x1005
la      $a1, aSSetupScriptS  # "%s: setup script %s"
la      $a2, aCallSetupScrip_0  # "call_setup_script_no_wait"
move    $a3, $s1
jal     logg
li      $a0, 7
jal     system          # This is bad
move    $a0, $s1        # command
jal     free
move    $a0, $s2        # ptr
jal     free
move    $a0, $s1        # ptr
```

Conclusion

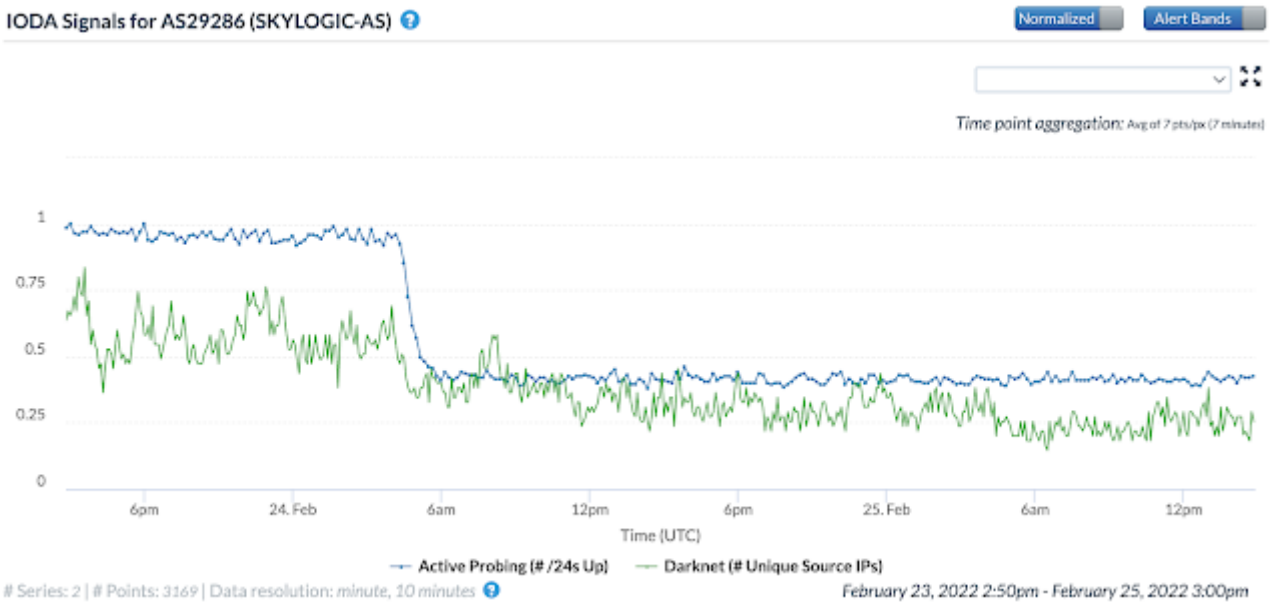
There are similarities between these issues and the approach followed by the attackers in the Viasat incident, but I am not implying that these vulnerabilities were actually abused by the attackers. However, the security posture of the Surfbeam2 firmware does not look good.

Hopefully these vulnerabilities are no longer present in the newest Viasat firmware, otherwise that would be a problem...

There are several unknowns yet to be resolved.

1. How the initial compromise of the VPN appliance worked. Did the attackers have valid credentials (maybe stolen from either Skylogic or its partners) or they exploited a known vulnerability (assuming an 0day doesn't match a 'misconfigured VPN appliance' explanation)?

2. How exactly the attack propagated to other countries, lasting for several hours. One of the affected persons I talked to got his modem knocked out around 9:00 am (GMT+1), several hours after the initial attack.



3. Before the destructive payload was executed, there was any other kind of malicious code running in the modems for a short period of time? Sentinelone published a very interesting research on 'AcidRain', a wiper that is able to generate the same destructive pattern observed in the modem's flash memory.

```
data_to_overwrite = allocated_region;
if (allocated_region < puVar1) {
  value_to_write = 0xffffffff;
  do {
    *allocated_region = value_to_write;
    allocated_region = allocated_region + 1;
    value_to_write = value_to_write - 1;
  } while (allocated_region < puVar1);
```

Coincidentally, this wiper also has similarities with 'VPNfilter' malware.

4. Did the compromise of the management segment involve additional attacks besides the VPN issue?

Unfortunately these technical questions can only be answered by people with an insider knowledge. Let's see if Viasat is willing to provide further details on this case.