A quick guide to creating patch loaders.

The essence of the patch is that instead of the command 11 handler (which normally displays the useless poweroff not supported inscription), we introduce our handler into the loader code, which allows us to run arbitrary code (no more than 2K in size) in the bootloader's address space. The patch consists of 3 components:

- the actual code of the command handler
- Chipset identification unit directly behind the processor
- A chipset identification unit that is added to the end of the bootloader.

The handler code is a short program in machine code, the source code of which can be found in the loaders/pexec_*.asm file. There is a version of the program for translation to ARM or THUMB-2 code, depending on which mode is used in the code of the command handlers of the loader we are interested in.

The first identification block is used to determine the chipset type by all utilities of the complex (except qdload) interacting with the loader. This eliminates the need for the -k switch on the command line. The block has the following format:

```
  .word  0xdeadbeef  ; Signature for Block Programmatic Search
  .byte  id          ; A 1-byte chipset code assigned to it in the chipset.cfg config file.
```

The second identification block is used only by qdload. It allows the program to determine the boot parameters and chipset type directly from the bootloader file, which eliminates the need to specify the -k and -a switches.

So, the patching process begins by defining the download addresses and the command handler 11. To do this, use the qblinfo program. Run it by specifying the name of the loader file in the command line, and you get something like this:

```
$ .. /qblinfo NPRG9x35p.bin
** NPRG9x35p.bin: 95704 bytes

 Download address: 02a00000
 Code start address: 02a00028
 CMD 01 = 02a00c41
 CMD 03 = 02a00d95
 CMD 05 = 02A00e67
 CMD 07 = 02a00e75
 CMD 09 = 02a00ef5
 CMD 0b = 02A00e23
 CMD 11 = 02a00f2d
 CMD 13 = 02a00f71
 CMD 15 = 02a00fc1
 CMD 17 = 02a011fd
 CMD 19 = 02a0122b
 CMD 1b = 02A012A1
 CMD 1d = 02A00f47
 CMD 20 = 02a01399
 CMD 30 = 02a0132d
 CMD offset table: 16A38
 Invalid CMD handler: 02a00c33

 Unsigned code or no HW_ID value in Subject field
```

Out of all this magnificence, we are interested in 3 addresses: the download address (02a00000), the code start address (02a00028), and the command handler address 11 (02a00f2d). Now upload our file to the IDA from the address 2a000000, not forgetting to specify the ARM Little endian processor type. If the auto-analysis does not start, start it manually, specifying that the code starts at the address of the beginning of the code (02a00028).
Now go to the address of the command handler 11 (02a00f2d). Here's an example of what you can see around the address you're interested in:

```
CODE_ALL:02A00EF4 ; =============== S U B R O U T I N E =======================================
CODE_ALL:02A00EF4
CODE_ALL:02A00EF4
CODE_ALL:02A00EF4                 EXPORT handle_sync
CODE_ALL:02A00EF4 cmd_07_sync ; XREF DATE: APP_RAM:02A16A60#o
CODE_ALL:02A00EF4 PUSH {R4,LR}
CODE_ALL:02A00EF6                 ADDS            R1, R0, #6
CODE_ALL:02A00EF8 LDRH R0, [R0,#4]
CODE_ALL:02A00EFA SUBS R0, R0, #1
CODE_ALL:02A00EFC UXTH R4, R0
```

```
CODE_ALL:02A00EFE LDR R0, =reply_buffer
CODE_ALL:02A00F00 MOVS R2, #0xA
CODE_ALL:02A00F02 STRB R2, [R0]
CODE_ALL:02A00F04 MOVW R2, #0x3FD
CODE_ALL:02A00F08 CMP R4, R2
CODE_ALL:02A00F0A BLS loc_2A00F10
CODE_ALL:02A00F0C BL buffer_overflow_error
CODE_ALL:02A00F10 ; ---------------------------------------------------------------------------
CODE_ALL:02A00F10
CODE_ALL:02A00F10 loc_2A00F10; KÓÐI XREF: handle_sync+16#j
CODE_ALL: 02A00F10 BÆTIR R1, R1, #1
CODE_ALL: 02A00F12 BÆTIR R0, R0, #1
CODE_ALL: 02A00F14 MOV R2, R4
CODE_ALL:02A00F16 BL hostdl_memcpy
CODE_ALL:02A00F1A LDR R1, =escape_state
CODE_ALL:02A00F1C                 ADDS            R0, R4, #1
CODE_ALL:02A00F1E STRH R0, [R1,#8]
CODE_ALL:02A00F20 BL compute_reply_crc
CODE_ALL:02A00F24 BL force_xmit_reply
CODE_ALL:02A00F28 MOVS R0, #0
CODE_ALL:02A00F2A POP {R4,PC}
CODE_ALL:02A00F2A ; End of function handle_sync
CODE_ALL:02A00F2A
CODE_ALL:02A00F2C
CODE_ALL:02A00F2C ; =============== S U B R O U T I N E =======================================
CODE_ALL:02A00F2C
CODE_ALL:02A00F2C
CODE_ALL:02A00F2C                 EXPORT handle_power_off
CODE_ALL:02A00F2C cmd_11_power_off ; DATA XREF: APP_RAM:02A16A80#o
CODE_ALL:02A00F2C PUSH {R4,LR}
CODE_ALL:02A00F2E MOVS R0, #0x12
CODE_ALL:02A00F30 LDR R1, =reply_buffer
CODE_ALL:02A00F32 STRB R0, [R1]
CODE_ALL: 02A00F34 MOVS R0, #1
CODE_ALL:02A00F36 LDR R1, =escape_state
CODE_ALL:02A00F38 STRH R0, [R1,#8]
CODE_ALL:02A00F3A MOVS R0, #0x15
CODE_ALL:02A00F3C                 ADR             R1, aPowerOffNotSup ; "Power off not supported"
CODE_ALL:02A00F3E BL transmit_error
CODE_ALL: 02A00F42 MOVS R0, #0
CODE_ALL:02A00F44 POP {R4,PC}
CODE_ALL:02A00F44 ; End of function handle_power_off
```

Here is the code not only for the operator of the command 11, but also for the handler of the command 09 (sync) that comes before it. In this handler, we need to find 2 important addresses:
- reply_buffer. An image of the packet is written here, which is sent to the host at the command processing window
- escape_state. This is the response packet parameter area, from which we only need a field in which the length of the response packet is recorded. So, here it is:

```
CODE_ALL:02A00EFE LDR R0, =reply_buffer
CODE_ALL:02A00F00 MOVS R2, #0xA
CODE_ALL:02A00F02 STRB R2, [R0]
```

We take the address reply_buffer. And here it is:

```
CODE_ALL:02A00F1A LDR R1, =escape_state
CODE_ALL:02A00F1C                 ADDS            R0, R4, #1
CODE_ALL:02A00F1E STRH R0, [R1,#8]
```

Take the address escape_state, as well as the offset at which the length of the packet is recorded (in this case, 8). But we don't need addresses in their pure form. We need to find the block in memory that these literals point to and find out its address. Go to the address reply_buffer, and look at all the links to this address. Some of the links will be of the form DCD reply_buffer. This is the DCD that is in the address space behind the 07 and 11 command handlers. It looks like this:

```
CODE_ALL:02A01098 off_2A01098 DCD escape_state ; DATE XREF: handle_hello:loc_2A00CA0#r
CODE_ALL:02A01098 ; handle_simple_read:loc_2A00E12#r ...
CODE_ALL:02A0109C off_2A0109C DCD reply_buffer; DATA XREF: handle_hello+6E#r
CODE_ALL:02A0109C ; handle_simple_read:loc_2A00DDA#r ...
```

This is where the consecutive addresses of escape_state and reply_buffer are located.
And finally, we need to define an address to exit the handler. To do this, we use the code for the command handler 07 that lies above the command handler 11, namely this code:

```
leavecmd:
CODE_ALL:02A00F20 BL compute_reply_crc
CODE_ALL:02A00F24 BL force_xmit_reply
```

```
CODE_ALL:02A00F28 MOVS R0, #0
CODE_ALL:02A00F2A POP {R4,PC}
```

For clarity, let's label the address of this code with the leavecmd label.
We are now ready to receive the binary image of our patch. Open the pexec_arm,asm or pexec_thumb.asm file in the editor, depending on the mode in which the processor is running in the command handlers of our loader. Next, add our addresses to the file:

```
pkt_data_len_off=8                      ; Here we enter the offset to the length of the package
                .ORG            0x02a00f20    ; And here is the address leavecmd
leavecmd:
                .ORG            0x02a00f2c  ; This is the address of the command handler 11
```

Now enter the chipset code into the identification block

```
        .byte       8       ; Chipset code
```

And finally, the address where our escape_state and reply_buf literals are located:

```
                .ORG 0x02a01098
reply_buf_ptr:    .word  0
escape_state_ptr: .word  0
```

That's it. Now let's translate our assembler source using any version of the AS assembler for arm:

```
$ arm-none-androideabi-as -o /dev/null -f=pexec_thumb.lst pexec_thumb.asm
```

Open the listing pexec_thumb.asm, and insert all the code of our command handler 11 on top of the existing one directly into the IDA database (edit — patch program — change byte). Now export the database to the resulting output file (file — produce file — create exe file). To distinguish a patched bootloader from an unpatched one, add the letter p to it at the end of the file name, for example, NPRG9x35p.bin.
The only thing left to do is to enter the second identification block into the loader:

```
$ ./setident NPRG9x35p.bin 8 02a00000
```

Here we specify the chipset code (8) assigned to it in the chipset.cfg file and the download address (02a00000). That's it, this completes the process of creating a patched bootloader.