Keygenning a Palm Crackme

By Potsmoke

First of all you will need the following programs:

SouthDebugger	http://www.jensbruhn.de
Java Runtime Environment v1.4 (JRE)	Found Here
Palm OS Emulator (POSE) v3.5 or older	Found Here
Palm OS ROM file v3.0 or newer	Found Here
X-Master.prc	Included in this tutorial .zip

I don't use any disassemblers in this tutorial, but if you want to use one I can recommend PalmdeMon (http://palmdemon.cjb.net), PrcEdit (http://prcedit.tk) and maybe IDA Pro.

INTRODUCTION

Today we will delve into the world of Palm cracking, which, i might say, is quite exciting. Let's start with some basic information. This text will only center upon Palm4 and older versions, Palm5 will not be mentioned. The reason for this is because Palm5 use a different CPU, ARM. Those of us who come from the world of windows cracking know that on x86 architectures we have registers like EAX, EBX, ECX etc. There are of course registers on Palm as well, but there are more of them and they have different names.

There are 8 data registers, these are: D0, D1, D2, D3, D4, D5, D6 and D7. We also have 8 address registers: A0, A1, A2, A3, A4, A5, A6 and A7, where A7 (USP/SSP) works as a stack pointer.

All the registers are 32bit. In addition to these registers we also have USP (User Stack Pointer), PC (Program Counter) and SR (Status Register). The PC register is equivalent to the EIP register from the x86 architecture.

Palm uses a DragonBall CPU from Motorola (MC68SZ328). That means we will not be dealing with x86 assembly (intel), but with 68k assembly, which you probably figured out when you saw the registers. 68k assembly has a different instruction set, and the syntax is opposite of what we windows crackers are used to. It is very much akin to AT&T syntax, which is used a lot on linux.

We can see this from these examples:

MOV EAX, 5 (Intel) corresponds to MOV 5, EAX (AT&T)
MOV EAX, EBX (Intel) corresponds to MOV EBX, EAX (AT&T)
MOV destination, source (Intel) corresponds to MOV source, destination (AT&T)

68k is akin to AT&T in this respect. This not only applies to MOV but also all the other instructions. People cracking on windows will be used to code like this: MOV AL, BYTE PTR [EBX+02]

On Palm you will see code like this: MOVE.B #$ (A2,D4), D0
As you can see it is a bit different, but understanding the code you are debugging shouldn't be too difficult.

Another thing you should be aware of is that on Palm the code "branches", while on windows it "jumps". The only real difference is the name of the instructions. If we compare some of the jump/branch commands they look like this:

BEQ (Branch if EQual) <----------> JE (Jump if Equal)
BNE (Branch if Not Equal) <----------> JNE (Jump if Not Equal)
BGE (Branch if Greater or Equal) <--------> JGE (Jump if Greater or Equal)
BLE (Branch if Less or Equal) <----------> JLE (Jump of Less or Equal)
BGT (Branch if Greater Than) <----------> JG (Jump if Greater)
BLT (Branch if Less Than) <----------> JL (Jump if Less)
BRA (Branch) <----------> JMP (Jump)
JMP (Jump) <----------> JMP (Jump)
JSR (Jump to SubRoutine) <----------> CALL (Call)

Now we've had a peek at the basics, and some of the differences between Palm and Windows, so let's start with the crackme.

CRACKING

Palm has something called HotSyncID, it is in a way the name of your Palm. All Palms have an ID like this. On Palm about 80% of all the programs has a registration routine where the serial is generated from your HotSyncID, so HotSyncID is kind of like a username.

That means a serial working for your HotSyncID won't work on other Palms than yours, that's usually why we have to keygen Palm programs (or patch, but that's no fun, right?) By the way, the opcode for NOP is 4E71 ;)

Start the emulator, load a ROM file (the operating system).
When you have done this you can install keygenme.prc and X-master.prc which you will find in the files directory. Drag and drop a file on the emulator window.

Right-click somewhere in the emulator and choose settings -> properties, in the bottom you will have to enter a HotSyncID, i chose Potsmoke.
When this is done we can start the debugger (run the south.bat file) and push the connect button.

Run the debugger in fullscreen, go to the windows meny and choose:

new breakpoint window
new register window
new memory-based disassembler window
new memory dump window

OK, we're ready to begin. Push the application button on the emulator to get the program listing (where all the programs are listed), choose X-Master and the emulator will halt. Now move on to the debugger to set a breakpoint. Since this crackme gets its username from the HotSyncID we can set a breakpoint on this API. Go to the breakpoint window in the debugger, push the trap breakpoint tab and enter: "sysTrapDlkGetSyncInfo"

The red light in the emulator means that the debugger is not in "contact" with the emulator, while the green one means we have "contact". As long as the green light is on we can't do anything in the emulator, it is halted so we can, for instance, set breakpoints in the debugger.

We've set a breakpoint on sysTrapDlkGetSyncInfo, so let's push F5 in the debugger to resume the emulator. Back to the emulator. Push the application button in the emulator to remove the X-master message. (we're now back in the program listing)

Now the fun starts. Choose keygenme, enter a serial and push the register button,
I used 112288. The emulator halts and we can start to debug, the green light is on.

F5 = Go (run/exit)
F6 = Out (exit JSR)
F7 = Next (Trace one instruction, don't enter JSR or follow branches)
F8 = Step (Enter JSR, follow branches)

If anyone was wondering the debugger is really slow. It takes a while to start and it's slow when tracing, but we'll survive.
Trace with F7, if you want to follow JSR/branches use F8.
If the disassembly window "lags" (the highlight cursor disappears) you can close the disassembly window and open a new one. This can happen when you are following branches/jsr, or when you just started debugging.
If you think it too slow to trace one instruction at a time, you can right-click somewhere in the disassembly window on the debugger and choose "go until" to go to the line you want.

The algo is not going straight down, please look at the branches/jumps and follow them to the right location.

```
SysTrap DlkGetSyncInfo                  We end up here (address 53D0A) in the debugger.
        tst.w     d0
        lea.l     $20(a7), a7
        bne       loc_2B2
        move.w    #$03E8, -(a7)
        SysTrap FrmGetFormPtr
        move.l    a0, d7
        pea.l     $29
        SysTrap MemHandleNew
        move.l    a0, d5
        move.l    d5, -(a7)
        SysTrap MemHandleLock
        move.l    a0, a2
        move.w    #$0029, -(a7)
        move.l    a4, -(a7)
        move.l    a2, -(a7)
        SysTrap StrNCopy
        move.l    a2, -(a7)
        SysTrap StrLen                  Get length of username.
        cmp.w     #$0008, d0            Is username 8 chars or more? If so, continue on.
        lea.l     $18(a7), a7           If your HotSyncID is less than 8 chars, change it and restart.
        bge.s     loc_1B4
```

You will never find this kind of check in normal Palm programs, people with HotSyncID as short as one char will always be able to register, at first this crackme did not use HotSyncID, thats why the check was added, and

someone forgot to remove it (hi Carp).

```
              move.l    d5, -(a7)
             SysTrap MemHandleUnlock
             move.l    a4, -(a7)
             SysTrap MemPtrUnlock
             move.l    #$0, d0
             add.w     #$8, a7
             bra       loc_2C4

loc_1B4:  move.l    #-$3CDEC324, d3
                                        Put C3213CDC in D3 (00000000-
3CDEC324=C3213CDC).
             move.l    #$0, d4          Clear counter.
             bra.s     loc_1D2          Continue on to loc_1D2.

loc_1BE:  move.b    $0(a2, d4.w), d0    Put first char of username in D0
(50).
             eor.l     d0, d3           EOR the first char with C3213CDC
(50^C3213CDC=C3213C8C).
             add.l     d3, d3           ADD D3 with itself
(C3213C8C+C3213C8C=86427918).
             eor.l     #$2258D32D, d3   EOR 2258D32D with the result
(2258D32D^86427918=A41AAA35).
             add.l     #$1, d4          Incrementing the counter by one
(algo part 1).

loc_1D2:        move.l    a2, -(a7)
             SysTrap StrLen             Get length of username again.
             cmp.l     d0, d4           Have we gone through all the
chars in username yet?
             add.w     #$4, a7
             blt.s     loc_1BE          Nope, we have more chars left, go
back to algo.
             move.l    #$1, d4
             bra.s     loc_212
```

Yep, all chars done, continue on with the rest. After we're done with all chars the result will be 8BA0635.

```
loc_1E4:        lsl.l     #$4, d3
```

LSL 4 with the result from the first algo part (4<<8BA0635=8BA06350).

```
                lea.l     $0(a2, d4.w), a3
                move.b    (a3), d0              Put the second char from
username in D0 (6F).
                muls.w    #$0086, d0
                                                MULS 86h with second char from
username (86h*6F=3A1A).

                add.l     d0, d3                ADD the result with 8BA06350
(3A1A+8BA06350=8BA09D6A).
                asr.l     #$2, d3               ASR the result with 2
(8BA09D6A>>2=E2E8275A).
                eor.l     #-$2A16BE44, d3       EOR that result with D5E941BC
(E2E8275A^D5E941BC=370166E6).   And again i used 00000000-
2A16BE44 to get the value D5E941BC.
                move.b    $1(a3), d0            Get the third char from username
(74), notice that 74 is in the lower D0.
                seq.s     d0                    Clears the byte we're suppose to
NEG :(
                neg.b     d0                    This is suppose to NEG the 74
value, but it's not working (due a mistake, right Carp?) so ignore both the
SEQ and the NEG instruction (remember it tries to NEG the last byte value in
D0).
                eor.w     #$006F, d0            EOR the result from the SEQ and
NEG with 6F (0^6F=6F) because of the error mentioned above the value of
D0 will always be zero, and the EOR will only work as a regular ADD
instead.
                or.l      d0, d3                OR the result with 370166E6
(6F||370166E6=370166EF).
                add.l     #$1, d4               Increment the counter by one (algo
part 2).

loc_212:        move.l    a2, -(a7)
                SysTrap StrLen                  Get username length once more.
                sub.w     #$1, d0               Decrement by one, the value is
used in the second algo part, namelength-1.
                cmp.l     d0, d4                Have we gone through all chars in
username on the second part of the algo?
                add.w     #$4, a7
                blt.s     loc_1E4               Return and go to the second part of
the algo if not all chars are done.
                pea.l     $29                   Our final result, which is the serial,
is CA18CC6F (for Potsmoke at least) when we finished the entire algo
(part1+part2).
                SysTrap MemHandleNew
                move.l    a0, a3
```

```
            move.l    a3, -(a7)
            SysTrap MemHandleLock
            move.l    a0, d4
            move.l    d3, -(a7)
            move.l    d4, -(a7)
            SysTrap StrIToH
            move.w    #$03EA, -(a7)          A0 now points to the correct serial.
            move.l    d7, -(a7)
            SysTrap FrmGetObjectIndex
            add.w     #$6, a7
            move.w    d0, -(a7)
            move.l    d7, -(a7)
            SysTrap FrmGetObjectPtr
            move.l    a0, d3
            move.l    d3, -(a7)
            SysTrap FldGetTextPtr          Get the serial number we typed in,
as you see the correct serial is already calculated, this is quite normal with
Palm.
            move.l    a0, a2               A0 now points to the serial we
typed in.
            move.l    a2, d0
            lea.l     $1A(a7), a7
            beq.s     loc_296
            move.l    d4, -(a7)
            SysTrap StrLen                 Get username length.
            move.l    d0, d3
            move.l    a2, -(a7)
            SysTrap StrLen                 Get the length of our serial.
            move.l    d0, d7
            cmp.l     d3, d7               Is username length same or less
than serial length?
            add.w     #$8, a7
            ble.s     loc_27E
            move.l    d7, d3

loc_27E:    move.l    d3, -(a7)
            move.l    d4, -(a7)
            move.l    a2, -(a7)
            SysTrap StrNCaselessCompare  Compare our serial with the
correct one.
            tst.w     d0                   Are they equal?
            lea.l     $C(a7), a7
            bne.s     loc_296
            move.b    #$7C, $1(a6)
            DC.W #FFFF
```

...

I skipped some code between here

```
loc_296:  jsr.l    loc_126
          tst.b    d0                  Was the two serials equal?
          bne.s    loc_336             Yep, jump to valid serial message.
          move.w   #$03E8, -(a7)       Put the resource ID (1000) for
invalid serial message on the stack.
          SysTrap FrmAlert             Show invalid serial message.
          add.w    #$2, a7
          bra.s    loc_340             Bye bye, the serial was invalid.

loc_336:      move.w   #$044C, -(a7)   Put the resource ID (1100) for valid
serial message on the stack.
          SysTrap FrmAlert             Show valid serial message.
          add.w    #$2, a7

loc_340:      move.l   #$1, d3         Set the registered "bit"
```

Now I have described the algorithm, and a keygen should be possible. Everyone who owns a Palm probably send their programs to the Palm using a PC with Windows, so why not make a windows keygen?

A working algo can look like this:

```
;D3=eax, D0=ebx, ecx=counter, edx=counter

          xor ecx,ecx
          xor edx,edx

          mov eax, 0C3213CDCh
main1:     movsx ebx, byte ptr [namebuffer+ecx]
          xor eax, ebx
          add eax, eax
          xor eax, 2258D32Dh
          inc ecx
          cmp ecx, [namelenght]
          jne main1

          mov ecx, 1
```

```
        mov edx, 2
main2:      shl eax, 4
        movsx ebx, byte ptr [namebuffer+ecx]
        imul ebx, ebx, 86h
        add eax, ebx
        sar eax, 2
        xor eax, 0D5E941BCh
        movsx ebx, byte ptr [namebuffer+edx]
        xor ebx, ebx
        xor ebx, 6Fh   ;we could have used ADD here, because of the
SEQ/NEG error
        or eax, ebx
        inc ecx
        inc edx
        cmp edx, [namelenght]
        jne main2
        ret
```

As I have already mentioned, the HotSyncID is used on 80% of all Palm programs. A Palm also have another kind of ID, the ROM ID. It is seldom used, but if we ever find one we will have trouble keygenning it. The Emulator has no ROM ID, it is blank instead of the 12 characters it should be. There is a solution to this problem, we can use a small program to set a "fake" ROM ID, but that's not part of this tutorial =)
Not all Palms (old ones?) have a ROM ID, that's probably the reason why it is not used very often. So remember, you need an extra program to keygen programs using the ROM ID.

GREETS/THANKS

Thanks to Carpathia for the crackme he made for me!
Thanks to Elessar for the english translation
Another thanks to Carpathia and Elessar for taking the time to read the tutorial looking for errors and bad explanations.
(Wonder how much it helped, Elessar was in a hurry and Carpathia was drunk)

I would also like to greet everybody in #cracking.no (EFnet) and all FHCF members. A greeting also goes out to RET and the old ID and DREAD members.

Norwegian version of this tutorial can be found at www.fhcf.net

Potsmoke@fhcf.net