FLARE-ON CHALLENGE 9 SOLUTION

BY BLAINE STANCILL (@MALWAREMECHANIC)

Challenge 2: PixelPoker

Challenge Prompt

I said you wouldn't win that last one. I lied. The last challenge was basically a captcha. Now the real work begins. Shall we play another game?

7-zip password: flare

Solution

PixelPoker.exe is a 32-bit executable game designed to challenge the user to click a specific pixel (no biggie, there's only 741x641 possible pixels). Executing the program displays a window with an image reminiscent of television static (*Figure 1*). Moving the cursor around the window updates the (X, Y) coordinates located in the window's title bar. Clicking within the window increments the click counter in the title bar and once 10 clicks have been reached, all further clicks generate a popup message indicating the game is over (*Figure 2*).

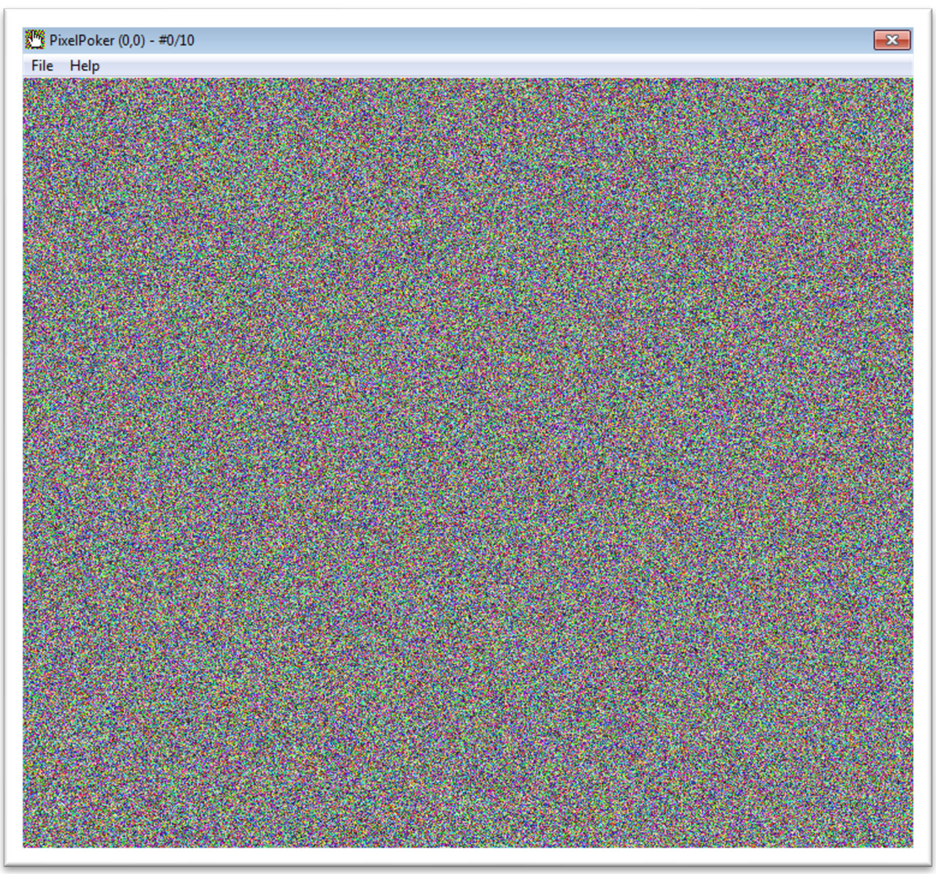

Figure 1: Initial window for PixelPoker.exe

Figure 2: Failure message after 10 clicks

Opening PixelPoker.exe in a dissembler of choice and navigating to WinMain (0x4016F0) shows a short function with the following overview:

- Load a resource image (0x40170C)
- Call a function to register a window class (sub_401120)
- Call a function to create an instance of the above window class (sub_401040)
- Enter window message dispatch loop

PixelPoker.exe loads a resource bitmap image named 129 and uses the image to define the window's dimensions and paint the window. Browsing the executable's resources using a tool such as Resource Hacker reveals two tv-static Bitmap resource images named 129 and 133. Resource 133 is left as an exercise to the user.

The window's dimensions are set in the function sub_401040 based on the two global variables dword_413280 and dword_413284 (*Figure 3*). At runtime these variables contain the values 741 and 641 respectively (i.e., the width and height of the loaded image). Let's rename them to g_img_width and g_img_height to make future code snippets easier to understand.

```
.text:00401092 loc_401092:
.text:00401092 mov     eax, dword_413280       ; g_img_width (741)
.text:00401097 mov     [ebp+Rect.right], eax   ; window width
.text:0040109A mov     eax, dword_413284       ; g_img_height (641)
.text:0040109F push    1
.text:004010A1 mov     [ebp+Rect.bottom], eax  ; window height
.text:004010A4 lea     eax, [ebp+Rect]
.text:004010A7 push    0CF0000h
.text:004010AC push    eax
.text:004010AD mov     [ebp+Rect.left], 0
.text:004010B4 mov     [ebp+Rect.top], 0
.text:004010BB call    ds:AdjustWindowRect
```

Figure 3: Setting window dimensions

When dealing with GUI windows the most important aspect to inspect is how the window class is defined and registered, see the documentation below:

- https://docs.microsoft.com/en-us/windows/win32/learnwin32/creating-a-window

Specifically, we're interested in the *window procedure* of the window class as this function is responsible for handling the window messages sent to our GUI window (e.g., WM_COMMAND, WM_LBUTTONDOWN, WM_DESTROY, etc...). Inspecting the function sub_401120 where the window class is defined, we see the window procedure value is populated with a pointer to the function sub_4012C0 (*Figure 4*).

```
.text:0040113A mov     [ebp+var_30.lpfnWndProc], offset sub_4012C0
```

Figure 4: Window procedure pointer

At first glance the function sub_4012C0 looks a bit intimidating, but after some careful inspection we see it's a large switch statement that handles different window messages sent to our window. The window message we're interested in is WM_LBUTTONDOWN that corresponds to a left-button mouse click when we click a pixel.

The window procedure function follows a standard function prototype outlined below (*Figure 5*).

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
```

Figure 5: Window procedure prototype

Following the dataflow of the msg variable and looking at the multiple comparisons, the comparison for WM_LBUTTONDOWN occurs at 0x40141C. Following the target of the conditional jump takes us to location loc_401436 (*Figure 6*) where the mouse click is processed.

```
.text:004012DE      mov     eax, [ebp+Msg]
.text:004012E1      add     esp, 0Ch
.text:004012E4      cmp     eax, 111h
     [...SNIP...]
.text:0040140D      mov     ecx, eax
.text:0040140F      sub     ecx, 200h         ; WM_MOUSEMOVE
.text:00401415      jz      loc_40157B
.text:0040141B      dec     ecx               ; WM_LBUTTONDOWN
.text:0040141C      jz      short loc_401436  ; Clicks take this jump
```

Figure 6: Left-click window message comparison

Starting at location loc_401436, the lParam variable is used to derive the X and Y coordinates of the clicked pixel. To further understand this, consult the documentation below and *Figure 7*:

- https://docs.microsoft.com/en-us/windows/win32/inputdev/wm-lbuttondown

```
.text:00401436 loc_401436:
.text:00401436 mov     eax, [ebp+lParam]
.text:00401439 movsx   edi, ax            ; edi = clicked X coordinate
```

```
.text:0040143C shr     eax, 10h
.text:0040143F push    ebx
.text:00401440 movsx   ebx, ax            ; ebx = clicked Y coordinate
```

Figure 7: Deriving X and Y coordinates from lParam

Following the above snippet is a series of comparisons. The first comparison at 0x40144B checks if we have reached the click limit of 10 clicks and displays a message box when met. The next two comparisons at 0x401486 and 0x40149D compare calculated coordinates to the clicked (X, Y) coordinates derived from the lParam value – these must represent the pixel's coordinates we need to click! The calculations for this pixel are displayed in *Figure 8*.

```
.text:0040146F inc   eax
.text:00401470 xor   edx, edx
.text:00401472 mov   g_click_counter, eax
.text:00401477 mov   eax, dword_412004
.text:0040147C mov   esi, g_img_width  ; g_img_width = 741
.text:00401482 div   esi               ; edx = dword_412004 % g_img_width
.text:00401484 cmp   edi, edx          ; edi = clicked X coordinate
.text:00401486 jnz   loc_401556
.text:0040148C mov   eax, dword_412008
.text:00401491 xor   edx, edx
.text:00401493 mov   ecx, g_img_height ; g_img_height = 641
.text:00401499 div   ecx               ; edx = dword_412008 % g_img_height
.text:0040149B cmp   ebx, edx          ; ebx = clicked Y coordinate
.text:0040149D jnz   loc_40154E
```

Figure 8: Comparing calculated and clicked coordinates

Based on *Figure 8* the clicked (X, Y) coordinates are being compared to coordinates calculated using the modulo of the image's width and height, and the global variables dword_412004 and dword_412008. The values of the global variables are shown below in *Figure 9*.

```
.data:00412004 dword_412004 dd 52414C46h   ; 'RALF'
.data:00412008 dword_412008 dd 6E4F2D45h   ; 'nO-E'
```

Figure 9: Hidden correct coordinates

Performing the modulo in *Figure 8* yields the calculated (X, Y) coordinates of (95, 313) as shown below in *Figure 10*. Let's click this pixel!

```
>>> 0x52414C46 % 741
95
>>> 0x6E4F2D45 % 641
313
```

Figure 10: Computing the correct coordinates

Clicking pixel (95, 313) reveals the winning flag below as seen in *Figure 11*. Hope you enjoyed this challenge!

- w1nN3r_W!NneR_cHick3n_d1nNer@flare-on.com

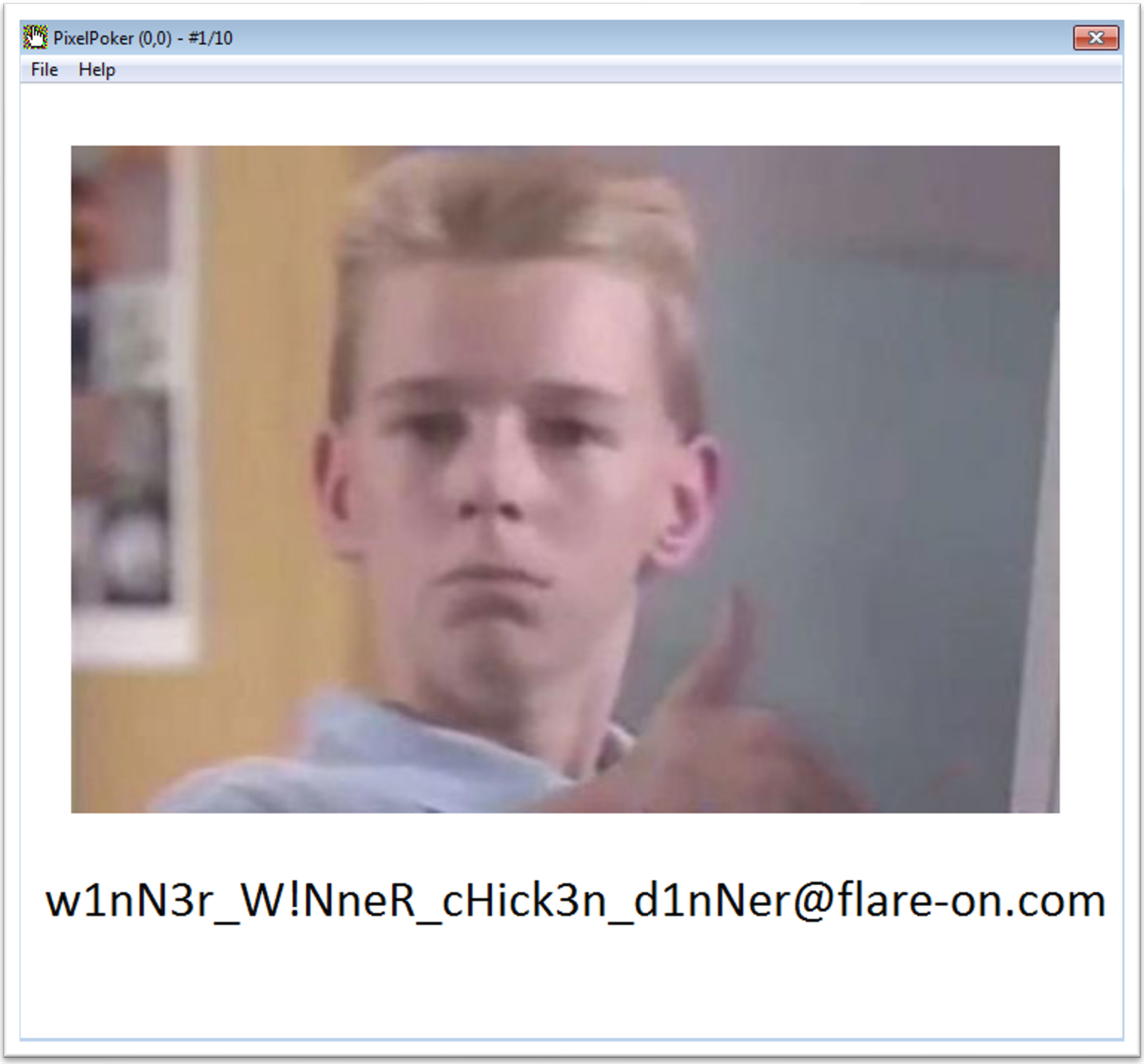

Figure 11: Win screen with flag