Defeating the iPhone Passcode

Author:

Brad Antoniewicz

Table of Contents

Introduction

Apple's iPhone has swept the mobile device market since its July, 2007 release. No matter the demographic - tweens, corporate executives, grandmothers - they all *must* have an iPhone. When a single device becomes so widely used, it's only a matter of time before it comes into the workplace. As the bridge between consumer and corporate is crossed, the threats to corporate data increases, and security must become a serious concern. In my role as a security consultant, I have had the opportunity to review one of the iPhone's most important security features: the "Passcode Lock" mechanism. This whitepaper includes a basic overview of the locking mechanism, information on where the passcode is stored, and instruction on how to overwrite that location to ultimately defeat the mechanism.

Note: After discovering the method described below and writing this whitepaper, I came across the "iPhone Forensices" book and Jonathan Zdziarki's webinar (http://www.oreillynet.com/pub/e/1093). These resources provided a way to perform a similar operation using MacOSX to achieve the same goal of bypassing the iPhone's passcode. Jonathan's method is in many ways better than this method, especially because relocated, rather than overwrote, the keychain file. Jonathan's method also works in previous firmware versions by using a similar process but by modifying a different file. Nonetheless I felt it was still beneficial to publish this paper as it demonstrates the serious security concerns raised when an organization deploys the iPhone to its users. This whitepaper also describes how to perform the operation in Windows, and demonstrates other tasks which can be beneficial for anyone performing iPhone hacks.

Acknowledgements

Big ups to Jeremy Allen, Alex Smolen, Dean Saxe, Rudolph Araujo, and Jerry Pierce. Special thanks to the hackint0sh.org forums as well. Finally if you haven't already, please be sure to watch the iPhone Forensics webinar by Jonathan Zdziarski at: http://www.oreillynet.com/pub/e/1093.

Background Information

Before diving into the technical side of things, let's review the "Passcode Lock" mechanism and go over some terminology.

The "Passcode Lock" Mechanism

As with most handheld devices, the iPhone has a locking mechanism to protect the device's data. This locking mechanism can be manually invoked, but is more frequently invoked by an "Auto-Lock" feature. The "Auto-Lock" feature will wait for the phone to be idle for a user defined number of minutes before automatically invoking the locking mechanism. When the device is in the locked and powered on state, the data on it cannot be accessed via iTunes or any other means.

Once the iPhone is in a locked state, the user must enter a passcode to access the device. This prevents against data theft in a scenario where the device is lost or stolen. The passcode is normally 4 digits long, which may be susceptible to brute force attacks given enough time and patience. There is also a delay after a certain number of incorrect attempts. Finally, the device can wipe its memory if the lockout threshold is reached which would probably end up deterring most attackers. The locking mechanism can also be set to use a more complex password containing any combination of letters, numbers or symbols. This can be done using the iPhone Configuration Utility freely available from Apple (http://www.apple.com/support/iphone/enterprise/). Passwords are expected to be more common than passcodes in corporate environments where security configurations can be mandated. The iPhone Configuration Utility provides a range of policies that can be specified to require users to choose appropriately complex passwords to protect the device.

The Keychain

The keychain is a SQLite database stored in the `/private/var/Keychains/keychain-2.db` file on the iPhone. It stores the passwords used on the device, including any passwords used for email accounts. As of firmware version 2.2(?), the device passcode is also stored within this keychain file.

DFU Mode

Device Firmware Update (DFU or sometimes called Device Failsafe Utility) Mode is an iPhone state where the device's operating system can be overwritten. This is what is commonly used to jail break the phone and install a custom firmware.

QuickPwn

QuickPwn (http://www.quickpwn.com/) is a closed-source, Windows-based tool for jail breaking the iPhone released by the iPhone Development team and poorlad. What makes QuickPwn unique is the method it uses to write data to the iPhone. Normally, when an iPhone is reflashed using iTunes, all data on the device is lost. QuickPwn uses a different method which preserves all data except the data it is explicitly told to overwrite.

QuickPwn was written so iPhone owners can easily jail break their iPhone with limited technical savvy. Because of this, QuickPwn performs a number of internal checks to minimize user interaction. For our purposes, these checks will actually prove to be a nuisance, so we'll have to modify it to make it work.

Bypassing the Passcode

Let’s start to put all of this information together. We know that the passcode is stored within the keychain. If we can overwrite the keychain with one that doesn’t contain a passcode, we may be allowed to bypass the passcode protecting the device (because it doesn’t exist after we overwrite it!). We also know that if we put the phone in DFU mode, we can overwrite data on the phone, and if we use QuickPwn, we can selectively overwrite only particular data. So, using a customized QuickPwn package, we can overwrite the keychain and effectively bypass the passcode.

Result

Once we bypass the passcode, all locally stored data (photos, emails, contacts, notes, etc...) is accessible.

Caveats

Remember, the keychain also stores the passwords for other applications. This means that if we overwrite the keychain file, we will have to re-input all passwords for applications which use the keychain. During testing, I was able to access all locally stored emails (including exchange accounts, etc...), but could not send or receive any new emails because I did not have the password stored in my keychain. As I discovered later on, it is possible to simply move the keychain then access it at a later date. This process was detailed on http://www.zdziarski.com/:

> To bypass the passcode in v2.2, all one needs to do is move the keychain out of the way, then reboot.
>
> ```
> mv /private/var/Keychains/keychain-2.db /
> ```
>
> This preserves the suspect's keychain, resets the passcode, and also temporarily disables any account passwords from the device so that the suspect's accounts won't be accessed by the iPhone, further preserving the file system. To restore all accounts, move the old keychain file back and reboot. You can manually remove the passcode lock from the keychain by deleting its record with sqlite3:
>
> ```
> delete from genp where acct = "DeviceLockPassword";
> ```

The process described below will only work on iPhone 3G devices. However, the process can be modified to work with other models.

No testing was performed on firmware versions besides 2.2.1. Since firmware 2.2 was first to store the passcode in the keychain, this exact sequence will not work on previous versions. However, the previous versions can be easily changed by modifying the settings file which enables the lock functionality as described in Jonathan Zdziarski's webinar (http://www.oreillynet.com/pub/e/1093).

Procedure

Test Environment

Testing was performed using a jail broken iPhone 3G 8GB (Phone A - which is used as our testing phone) and a stock, non-jail broken iPhone 3G 16GB (Phone B - which is our target phone). Phone B was set up with an Exchange email account, and various personal data (notes, pictures, etc.). Additionally, Phone B was configured with a passcode. The goal of the testing was to retrieve all of this data from Phone B without knowledge of the passcode.

The other hardware used was a laptop running Windows XP with iTunes 8, and an Ubuntu Linux VM.

Copying the Keychain

Phone A (test phone, previously jail broken using QuickPwn) was configured with a passcode then accessed remotely via SSH. The keychain was compressed with permissions intact:

```
iphoneA # tar -pcvf Keychain.tar /private/var/Keychain/keychain-2.db \
/private/var/Keychain/TrustStore.sqlite3
```

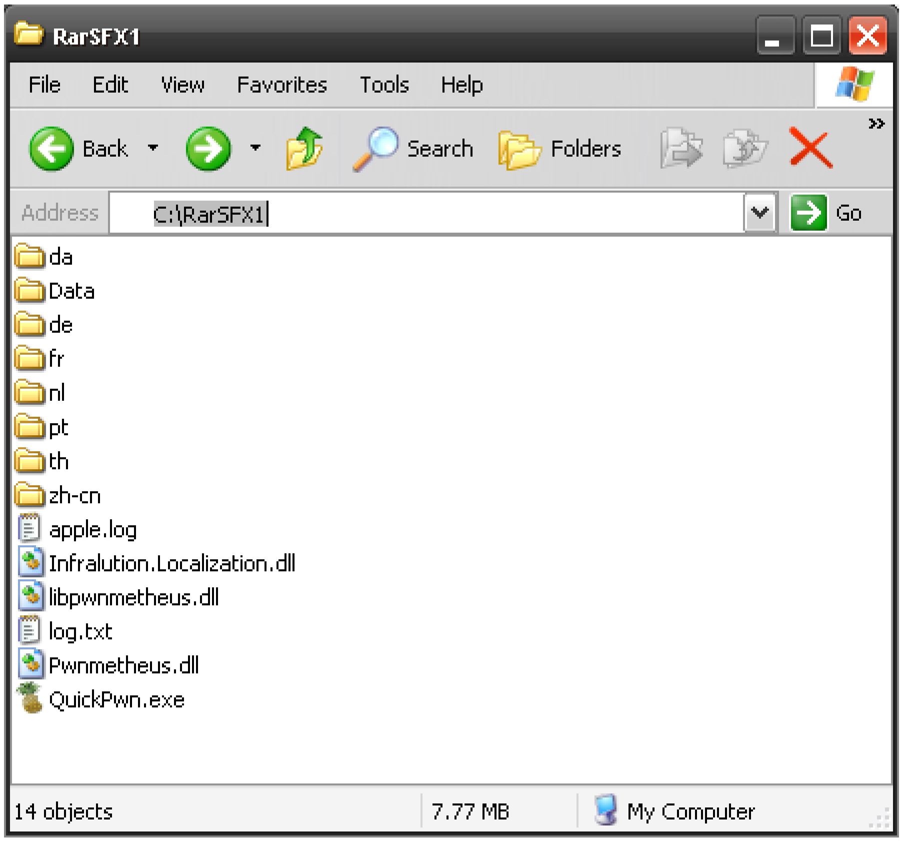
It was then copied off of the device using WinSCP

Extracting QuickPwn

QuickPwn version 2.2.5-2 was downloaded from http://www.quickpwn.com and extracted. `QuickPwn.exe` is just a packed version of the actual QuickPwn application, so in order to access the actually QuickPwn executable and data follow these steps:

1. Double click `QuickPwn.exe`
2. While QuickPwn.exe is running, go to Start -> Run and type `%TEMP%`
3. Look for the `RarSFX`*N* (where N is a number, this value changes per computer) directory or the last directory created . A look into this directory should reveal that it contains `QuickPwn.exe` and a number of additional files and directories. This is the QucikPwn application.
4. Copy this entire directory to a new location. In the example below, I have copied the directory to `C:\RarSFX1`.
5. Close the QuickPwn application.

Your directory should look similar to the screenshot below:



da
Data
de
fr
nl
pt
th
zh-cn
apple.log
Infralution.Localization.dll
libpwnmetheus.dll
log.txt
Pwnmetheus.dll
QuickPwn.exe

14 objects 7.77 MB My Computer

Modifying Cydia

Cydia is a package manager for applications written to run on jail broken iPhones. It's normally installed when you use QuickPwn to jail break a phone. In order to bypass the passcode on Phone B, we'll add the Keychain copied from Phone A and integrate it into the Cydia package. This ensures the Keychain is installed on Phone B during the jail break process with QuickPwn.

Using our example, copy `C:\RarSFX1\Data\Cydia.tar.gz` and our `Keychain.tar` to the Ubuntu Linux VM via SCP. Decompress `Cydia.tar.gz` then `Keychain.tar`. Then recompress the Cydia and the Keychain directory structures together into a single zipped tarball named Cydia.tar.gz. Ensure you preserve all permissions and run all commands as root.

```
LinuxBoxen # tar -pxvf Cydia.tar.gz
LinuxBoxen # tar -pvf Keychain.tar
LinuxBoxen # rm Cydia.tar.gz Keychain.tar
LinuxBoxen # ls -tla private/var/Keychain/keychain-2.db
-rw------- 1 64 root 28672 2009-02-04 12:51 private/var/Keychains/keychain-2.db
LinuxBoxen # tar -pcvf Cydia.tar .
LinuxBoxen # gzip Cyida.tar
```

The command on line 4 is to verify that `Keychain.tar` decompressed properly and the `keychain-2.db` file exists with proper permissions.

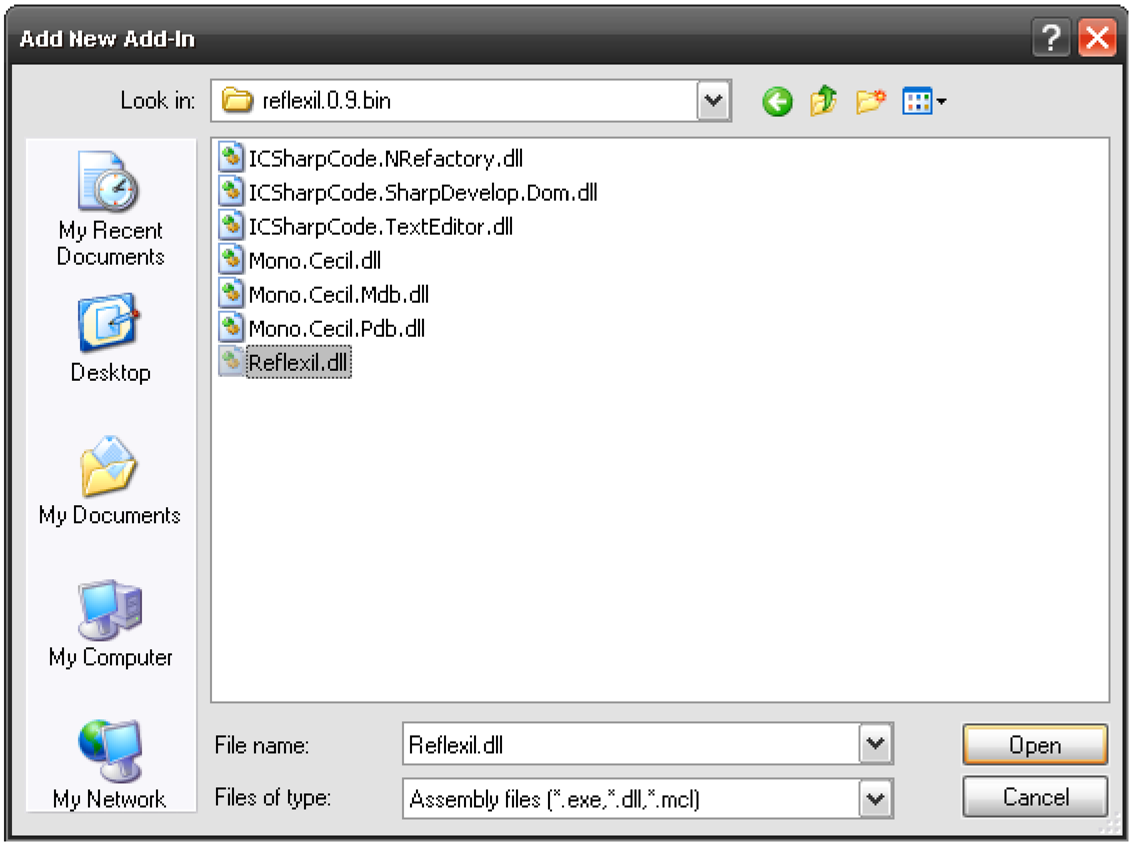
Now use WinSCP to copy your newly created `Cydia.tar.gz` from the Linux VM to the Windows system and replace the `C:\RarSFX1\Data\Cydia.tar.gz` file with it.

Modifying QuickPwn

Next, QuickPwn must be modified so it won't run its checks to ensure the device is connected properly when it first launches. We do this because Phone B is locked so it will not register properly with QuickPwn.

First, download .NET Reflector from http://www.red-gate.com/products/reflector/ and Reflexil from http://sebastien.lebreton.free.fr/reflexil/. Decompress them both, then load reflector and the Reflexil Add-On:

1. Launch Reflector, select C# from the drop down menu in the middle, then go to View -> Add-Ins
2. Click Add and select Reflexil.dll from the folder where you placed the decompressed Reflexil zip.



reflexil.0.9.bin

ICSharpCode.NRefactory.dll
ICSharpCode.SharpDevelop.Dom.dll
ICSharpCode.TextEditor.dll
Mono.Cecil.dll
Mono.Cecil.Mdb.dll
Mono.Cecil.Pdb.dll
Reflexil.dll

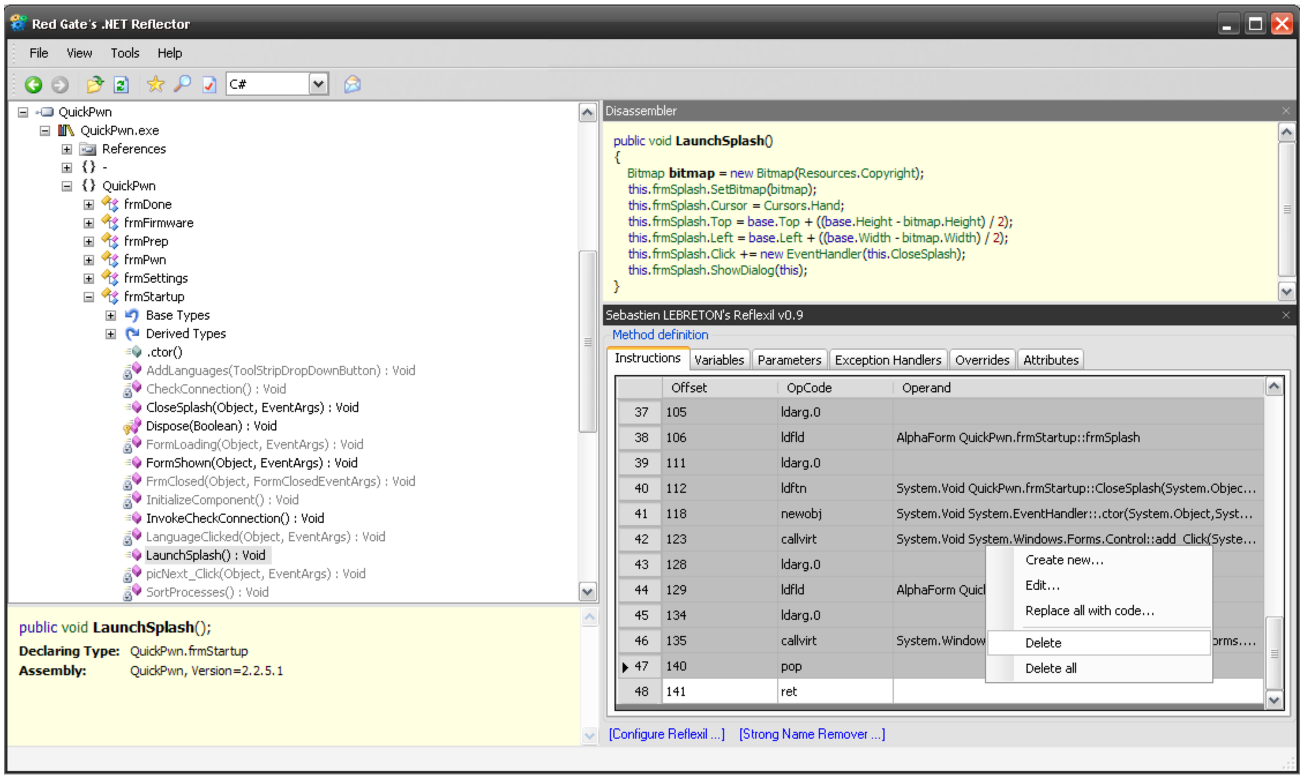
File name: Reflexil.dll
Files of type: Assembly files (*.exe,*.dll,*.mcl)

3. Select Open then close the Add-In window
4. Next, go to File -> Open and select `c:\RarSFX1\QuickPwn.exe`

5. To remove the Splash Screen: expand `QuickPwn` -> `QuickPwn.exe` -> `{ } QuickPwn` -> `frmStartup` and double click "`LaunchSplash(): Void`".

6. Go to Tools -> Reflexil v0.9 to load the Add-In. You should see the Reflexil pane open in the right of the Reflector window

7. In the Reflexil pane on the bottom right, highlight all instructions except the last (`opcode: ret`). Right click and delete all the lines thus removing the splash screen display code.



```
QuickPwn
  QuickPwn.exe
    References
    { } -
    { } QuickPwn
      frmDone
      frmFirmware
      frmPrep
      frmPwn
      frmSettings
      frmStartup
        Base Types
        Derived Types
        .ctor()
        AddLanguages(ToolStripDropDownButton) : Void
        CheckConnection() : Void
        CloseSplash(Object, EventArgs) : Void
        Dispose(Boolean) : Void
        FormLoading(Object, EventArgs) : Void
        FormShown(Object, EventArgs) : Void
        FrmClosed(Object, FormClosedEventArgs) : Void
        InitializeComponent() : Void
        InvokeCheckConnection() : Void
        LanguageClicked(Object, EventArgs) : Void
        LaunchSplash() : Void
        picNext_Click(Object, EventArgs) : Void
        SortProcesses() : Void
```

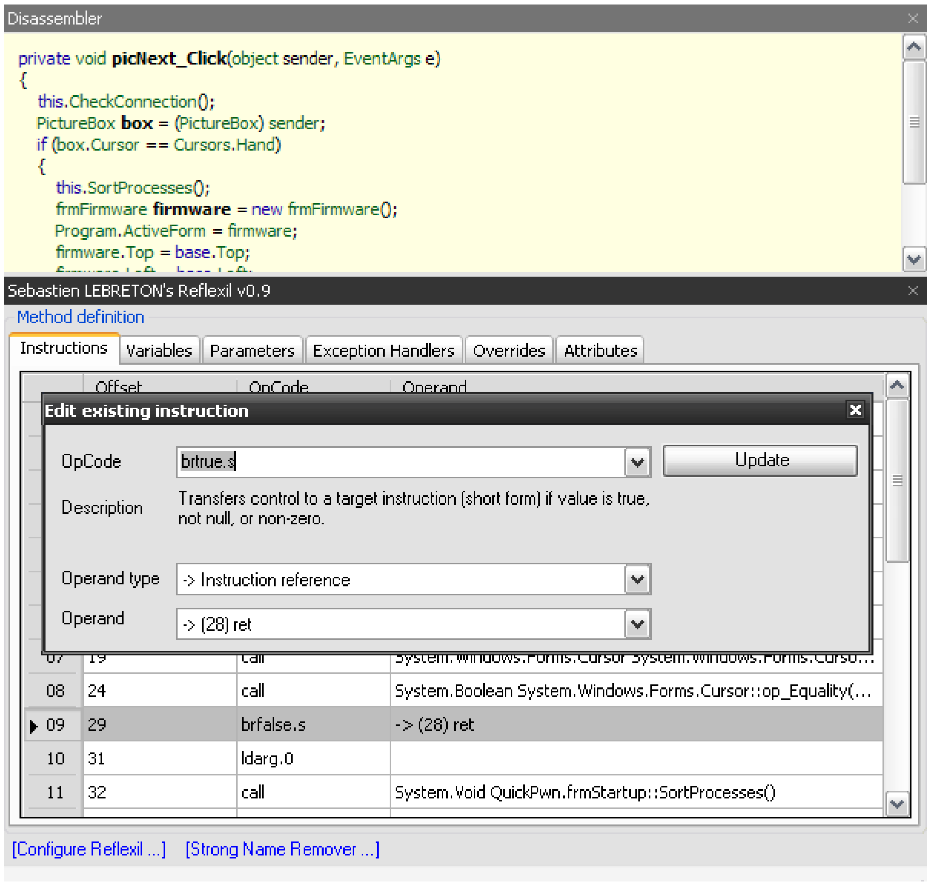
```
public void LaunchSplash();
Declaring Type:  QuickPwn.frmStartup
Assembly:        QuickPwn, Version=2.2.5.1
```

```
public void LaunchSplash()
{
   Bitmap bitmap = new Bitmap(Resources.Copyright);
   this.frmSplash.SetBitmap(bitmap);
   this.frmSplash.Cursor = Cursors.Hand;
   this.frmSplash.Top = base.Top + ((base.Height - bitmap.Height) / 2);
   this.frmSplash.Left = base.Left + ((base.Width - bitmap.Width) / 2);
   this.frmSplash.Click += new EventHandler(this.CloseSplash);
   this.frmSplash.ShowDialog(this);
}
```

	Offset	OpCode	Operand
37	105	ldarg.0	
38	106	ldfld	AlphaForm QuickPwn.frmStartup::frmSplash
39	111	ldarg.0	
40	112	ldftn	System.Void QuickPwn.frmStartup::CloseSplash(System.Objec...
41	118	newobj	System.Void System.EventHandler::.ctor(System.Object,Syst...
42	123	callvirt	System.Void System.Windows.Forms.Control::add_Click(Syste...
43	128	ldarg.0	
44	129	ldfld	AlphaForm Quic
45	134	ldarg.0	
46	135	callvirt	System.Window ... orms....
▶ 47	140	pop	
48	141	ret	

8. Now for the good stuff! In the left hand pane of .Net Reflector expand `QuickPwn` -> `QuickPwn.exe` -> `{ } QuickPwn` -> `frmStartup` and double click "`picNext_Click(Object, EventArgs): Void`". This is the function for the startup form which activated the Next button.

9. In the Reflexil pane on the bottom right, go down to instruction 09, right click -> edit and change the opcode from "`brfalse.s`" to "`brtrue.s`"



```
private void picNext_Click(object sender, EventArgs e)
{
   this.CheckConnection();
   PictureBox box = (PictureBox) sender;
   if (box.Cursor == Cursors.Hand)
   {
      this.SortProcesses();
      frmFirmware firmware = new frmFirmware();
      Program.ActiveForm = firmware;
      firmware.Top = base.Top;
```

Sebastien LEBRETON's Reflexil v0.9

Method definition

Edit existing instruction

OpCode: brtrue.s

Description: Transfers control to a target instruction (short form) if value is true, not null, or non-zero.

Operand type: -> Instruction reference

Operand: -> (28) ret

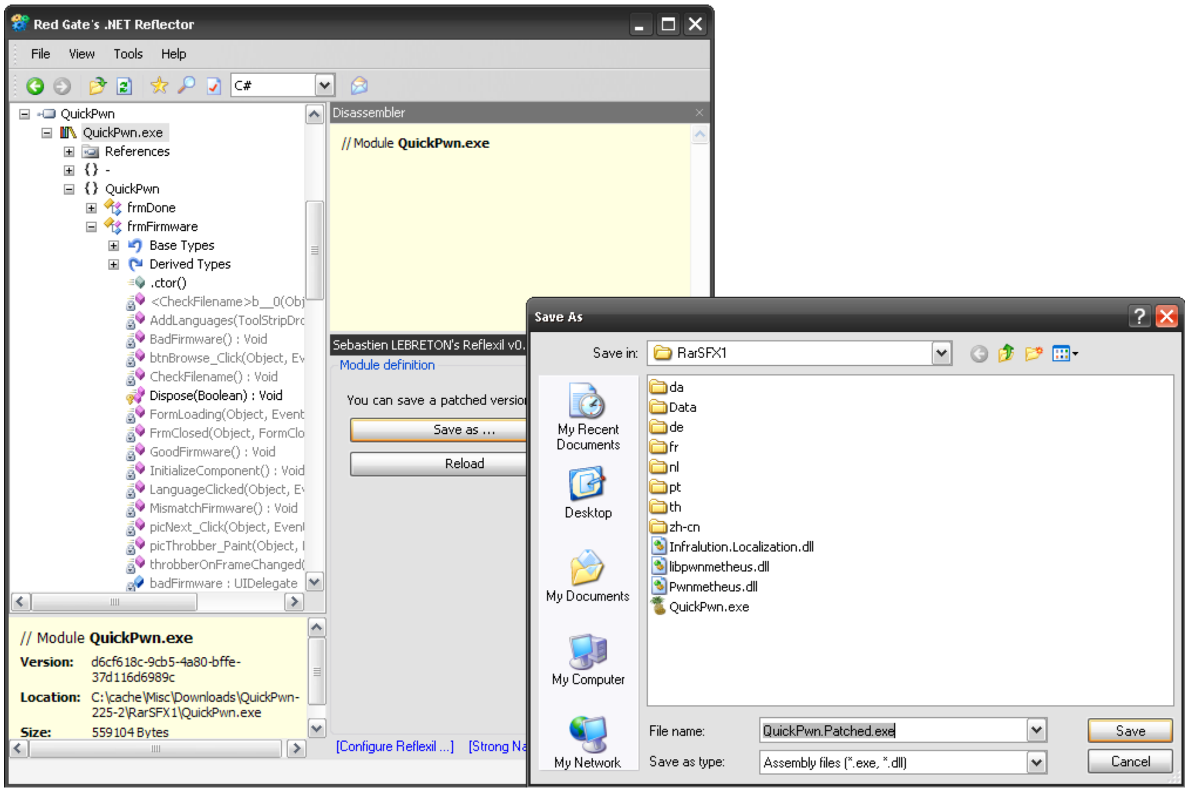
	Offset	OpCode	Operand
08	24	call	System.Boolean System.Windows.Forms.Cursor::op_Equality(...
09	29	brfalse.s	-> (28) ret
10	31	ldarg.0	
11	32	call	System.Void QuickPwn.frmStartup::SortProcesses()

10. Finally, expand `QuickPwn` -> `QuickPwn.exe` -> `{ } QuickPwn` -> `frmFirmware` and double click double click "`.ctor()`"

11. In the Reflexil pane on the bottom right go down to instruction 28 and change the opcode from "`brfalse`" to "`brtrue`"

12. Then, in the Reflexil pane on the bottom right go down to instruction 32 and change the opcode from "`brture.s`" to "`brfalse.s`"

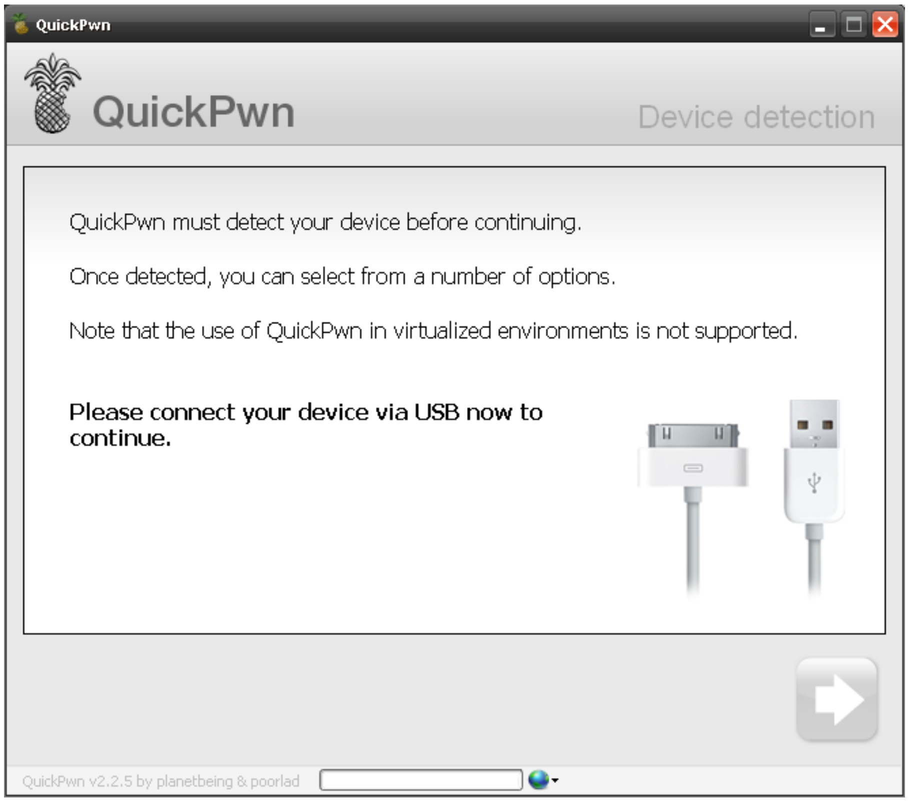
13. Finally, click QuickPwn.exe from the left pane tree, and click "Save As" from the Reflexil pane on the bottom right. Save it as `C:\RarSFX1\QuickPwn.Patched.exe`



// Module **QuickPwn.exe**
Version: d6cf618c-9cb5-4a80-bffe-37d116d6989c
Location: C:\cache\Misc\Downloads\QuickPwn-225-2\RarSFX1\QuickPwn.exe
Size: 559104 Bytes

Finishing up the Process

The last step is to jail break Phone B and overwrite its Keychain with the Keychain from Phone A. Launch the patched QuickPwn executable (`QuickPwn.Patched.exe`) while ensuring no phones are connected to the Windows computer. You'll notice that the splash screen doesn't start up, and the blue "Next" arrow is grayed out. Even though its grayed and looks disabled, just click on it to continue.



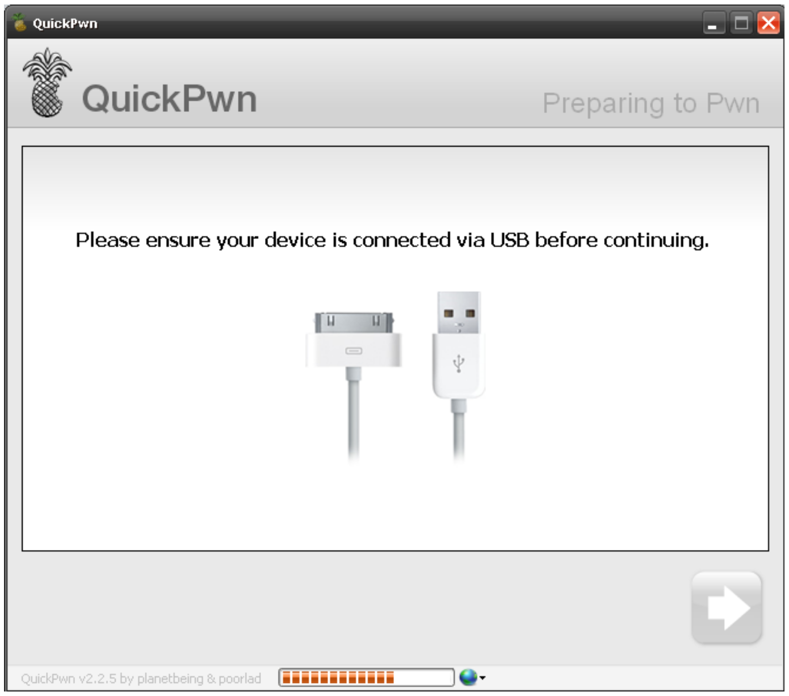
QuickPwn must detect your device before continuing.

Once detected, you can select from a number of options.

Note that the use of QuickPwn in virtualized environments is not supported.

Please connect your device via USB now to continue.

Follow the instructions by providing it the 2.2.1 firmware IPSW and once you get to the "Preparing to Pwn" screen (shown below) connect your passcode protected device. Again, the Next button may not turn blue, but you should be able to click it anyway. Now follow the on-screen directions to put your phone into DFU mode.



Please ensure your device is connected via USB before continuing.

Once the process completes and you see the Apple logo on the screen of the iPhone, disconnect the USB cable and your device should be no longer passcode protected.

I'm sure if you decided to write your own application specifically for this process you'd be able to do this a little more efficiently. However this method will work in the meantime.

Use responsibly and enjoy!

Miscellaneous Information

In the spirit of information sharing, here are some Miscellaneous tasks/information which may come in handy. All of the tasks require a jail broken iPhone.

Accessing the Command line

Connect the iPhone to the system via the USB cable, launch iTunes, then use one of the following tools to achieve command line access to the iPhone.

iTunnel

Available from (http://www.makkiaweb.net/itunnel/download.html), iTunnel will also require you to install OpenSSH via Cydia in order to connect. You can use something like "Toggle SSH" so you can enable/disable SSH via the iPhone interface.

The default installation provides a pretty GUI to automatically launch iTunnel for you. This is not really needed. All you need is `iTunnel.exe` and the companion `iTunesMobileDevice.dll`. To launch iTunnel just type:

```
C:\> iTunnel.exe 22 22
```

The first attribute is the local port and the second attribute is the remote port. You will have to change the local port if you have a local SSH daemon running. Now SSH to localhost specifying the port you defined when launching iTunnel.

iphoneinterface.exe

With a jailbroken iPhone, you can use `iphoneinterface.exe` to gain basic shell access to the system. It is much similar than the shell you'd get with iTunnel, but may come in handy for specific purposes (i.e scripting).

Important files

File	Description
`/var/mobile/Library/Mail`	Folder contain all the mail stored on the system.

`/var/mobile/Library/Notes/notes.db`	Notes
`/var/mobile/Library/Media/DCIM/100APPLE`	Photos stored in the “Camera Roll” of the iPhone. This is usually all pictures taken with the iPhone’s Camera
`/var/Mobile/Library/Photos/`	All other Photos or images stored on the iPhone
`/var/root/Library/AddressBook`	Contacts
`/var/root/Library/Calendar`	Calendar Info
`/var/root/Library/Preferences`	Device Preferences