

PowerBuilder 的安全性議題

這一篇來談談 PowerBuilder 程式的安全性問題。資訊系統安全性一直是很重要的議題，任何資訊專案的系統建置，一項不可遺漏的需求就是安全性要求。才今年二月的新聞，全球最安全晶片 TPM(Trusted Platform Module)遭人破解(請參考 <http://tw.news.yahoo.com/article/url/d/a/100210/78/20con.html>)，因此往往有人說所有的資安都不見效，怎樣做都會有人破解!由此可見，安全性是一個棘手的問題。

安全性的其中一環，就是程式碼本身的安全性。例如，你的程式碼是否容易為它人所破解，關鍵技術因此遭到破解?或是你的程式是否遭受竄改，加入不當程式碼，以剽竊或是破壞企業資訊?你如何得知你所購買的元件，裡面有沒有不當程式碼，這尤其在你沒有該元件原始程式碼的時候?

諸如以上總總，對於 PowerBuilder 的開發人員，面對這樣的問題，該如何面對?

壹、 PowerBuilder 安全性的考量

既然所有的安全措施都不保證沒有問題，那該如何處理?其實解決安全性問題應該可以用另一個想法來考量，就是成本和可行性。假如要破解一個安全性高的程式，需花費過度的時間或成本，那破解的目的就沒有意義了。與殺雞焉用牛刀、殺雞取卵有相同的道理。讓破解的一方失去破解的動力，目的已然達到。

我們應該要知道 PowerBuilder 在安全性上的機制，配合一些做法或工具，築起破解的高牆，就可增加 PowerBuilder 程式碼的安全。首先，PowerBuilder 在安全性上的考量，我們先看看(Code Access Security)(簡稱 CAS)的機制。

- Code Access Security

CAS 機制主要來自於.NET 的技術，有非常多的內容，符合.NET 程式在各方面安全的要求。筆者針對 PowerBuilder 在 CAS 安全性的支援上，簡化成兩個角度來看：第一個是程式外部的角度，第二個當然就是程式內部的角度。

- 外部的角度，就是把程式碼當作黑箱，你站在外部的位置上。例如，你呼叫了一個 API DLL 程式，你只需要呼叫函數名稱，並傳入參數即可。又例如你購買了其他開發商所開發的 PowerBuilder 程式框架，你只是加入到 Library path，以便你的程式可以

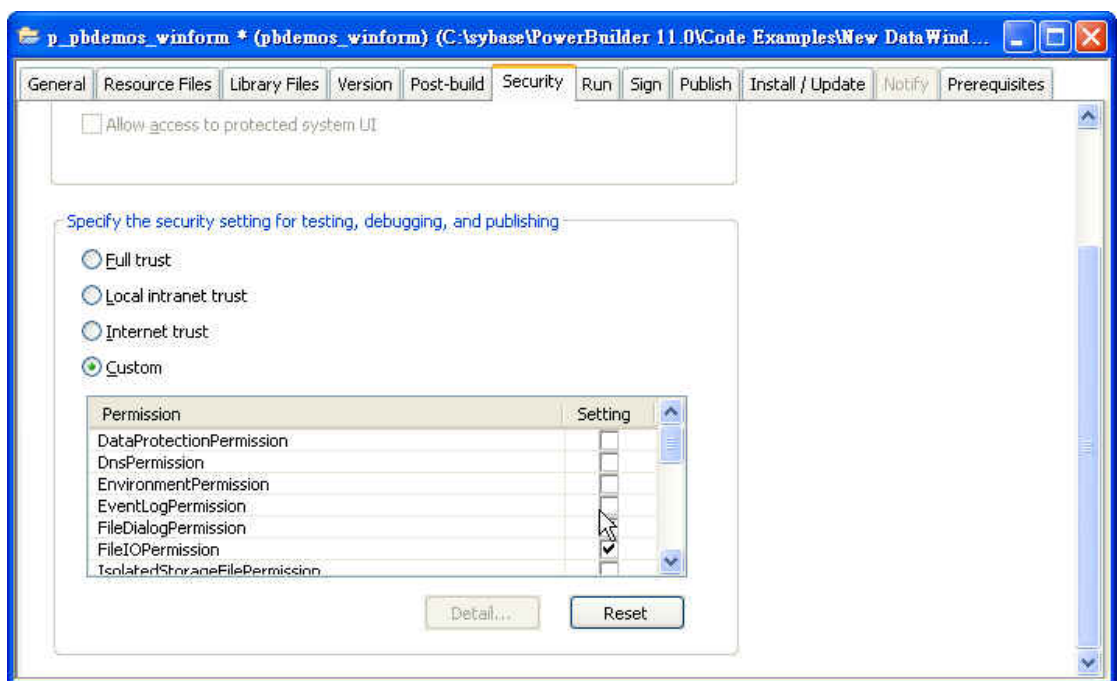
March 10 M-Power eNew

本篇文章版權為倍力資訊股份有限公司所有，未經書面同意，嚴禁複製、轉載

呼叫，可以繼承。以上，我們完全不用了解該 DLL 檔案，或是框架是怎麼寫的，我們只要知道如何使用它，如何呼叫他即可，他只是一個黑箱。

外部的角度，其問題就是黑箱有無執行不當的程式碼？例如，當你登入系統，進入程式後，此黑箱會不會將你登入的帳號密碼記錄下來，並偷偷記發 email 出來？

要解決這樣的問題，就是將此黑箱外部，再加入重重的枷鎖或限制，避免黑箱做太多不該做的事情。CAS 就是做枷鎖的事情，他可以做的限制諸如檔案的輸入/輸出、環境變數的讀取、資料庫連線...等等，各位可以參考圖一的範例：



圖一：PowerBuilder CAS 設定畫面

圖一的設定是在 Project 物件內，所以其安全性的設定範圍，是針對整個 PowerBuilder Application 為主，而非針對小範圍的程式。假如你要針對特定範圍的話，那就要以內部的角度來看了。

- 內部的角度，就是利用 PowerBuilder 和 .NET 程式的互通性(interoperability) PowerBuilder 從 11 的版本開始，可以呼叫 .NET 的程式碼，這個特性我們稱之為互通性。例如，你可以呼叫 .NET Assembly，或是呼叫 .NET Framework 既有的功能。呼叫的結果可以傳給 PowerBuilder，因此就達到串通的目的，資料可在 PB 和 .NET 遊走。此時，.NET Framework 頓時成為你 PowerBuilder 的程式庫，可依據需要呼叫

之。

內部的角度，就是以 PowerBuilder 呼叫 .NET Framework 提供 CAS 機制的 Assembly(類似元件的意思)，用寫程式的方式限制某一段程式碼是否允許執行你要的功能。例如，你要限制用來驗證帳號密碼的函數，不可執行讀寫外部檔案的功能，也不可送出 smtp(用以寄出 email)的訊息出去。以這個範例，你要撰寫程式，呼叫 CAS 機制的 .NET 程式，在呼叫這個函數前和呼叫後的這個範圍內，禁止執行上述動作，就可以避免此函數有不當動作執行了。

內部的角度最大的不同，就是依你的需要來限制 CAS 的範圍，而非外部的角度，一次針對整個 PB 應用程式。這具備很大的彈性，但是因為不是 PB 既有提供的功能，所以尚需要自行撰寫，複雜度高些。

CAS 機制來自於 .NET 的技術，所以 PowerBuilder 對於 CAS 的機制，只有在 PB .NET WebForm、Windows Form、Assembly 以及 Web Services 這四種程式類型上才有支援。各位若對於 CAS 有更深的興趣，可以參考倍力技術專欄：
http://www.mpinfo.com.tw/TechnologyColumnFiles/PB_T_200811.pdf，討論更多有關 CAS 的技術細節。

至於原生的 PowerBuilder 功能，倒沒有支援 CAS，此時，就需要另外的工具來達到安全性上的需求了。

貳、 PBProtect 工具

目前市面上很多工具，都有破解包括 PowerBuilder、Java 或是 .NET 程式語言的工具，可以將這些程式的執行檔，轉回原本的原始碼。這樣的工具，其實就是在做反組譯(Decompiler)的事情。

以 PowerBuilder 而言，我們寫好的程式叫做 source code，它會先轉換成 p-code，p-code 會在 PowerBuilder Virtual Machine 上 compiler 成 byte-code，然後執行。這是 PowerBuilder 從 source code 到執行的一連串順序。一般市面上的反組譯工具，都是將 PowerBuilder 的 byte-code，一直回到 source code，以供你的需要。一般而言，反組譯並不違法，通常是因為你的原始碼已經遺失了，才會用來還原你的程式。但是假如你把反組譯拿來做惡意、違法的事情，這才是違法的。

為何會提到反組譯的機制呢？因為反組譯提供作惡意、違法事情的機會，所以我們假如讓被反組譯出來的程式，顯示一堆亂碼，是否就可以避免不好的事情發生？答案是肯定的，而目前市面上能做到這個功能的，就是：PBProtect 工具。

March 10 M-Power eNew

本篇文章版權為倍力資訊股份有限公司所有，未經書面同意，嚴禁複製、轉載

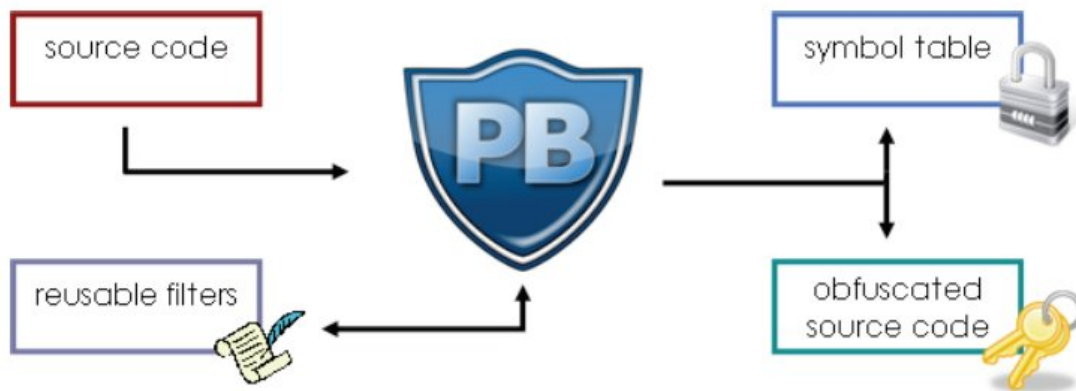
PBProtect 可以將反組譯出來的程式碼，變成亂碼，它主要利用的機制，就是在 **compiler** 之前，先將 **source code** 編成亂碼，讓這些亂碼去 **compiler**，自然反組譯回來也是亂碼，所以無法有進一步的應用。

■ **Obfuscator(混淆)**

PBProtect 將原始程式碼變成亂碼，主要是利用混淆程式碼的動作，稱之為 **Obfuscator**。他不是將所有程式碼變成亂碼(全部程式變成亂碼，就無法 **compiler** 成功了)，而是只將程式內所有你宣告的各種變數、函數以及物件名稱，全部重新命名成一個長字串，讓讀取程式並瞭解程式變成一件不可能達到的事情，並且保持 **PowerBuilder** 程式可 **compiler**。當這些轉換成功後再去 **compiler**，**compiler** 成功的 **byte-code** 程式利用反組譯再反轉回來，自然也是不容易辨識的原始程式了。

當然，各位可能會問，假如經過重新命名的程式，要如何返回正常命名的原始碼？基本上，PBProtect 會將原始的名稱和新的名稱做個對照表，稱之為 **symbol table**，這是一個 **xml** 檔案，也是還原時參考用的。你不要將此檔案外漏出去，這會增加被破解的機率。當然，以實務上而言，假如一開始的程式你已經備份，然後交由 PBProtect 製作亂碼，那這個 **symbol table** 已經不重要，你可以在 PBProtect 執行完後，立刻將之刪除，避免外漏的危險。

整個 PBProtect 的架構，可以如下圖所示：



圖二：PBProtect

左上角是原始程式碼，左下角有提到 **reusable filters**，顧名思義，就是用來設定過濾條件，以便你可以選擇要將哪些物件的名稱予以混淆。混淆後的程式碼在右下角，稱之為 **obfuscated source code**，將來反組譯程式的結果也就是這個，裡面一大堆看不懂的名稱。至於右上角就是對照表，用來對照原始物件名稱和改名後的名稱。

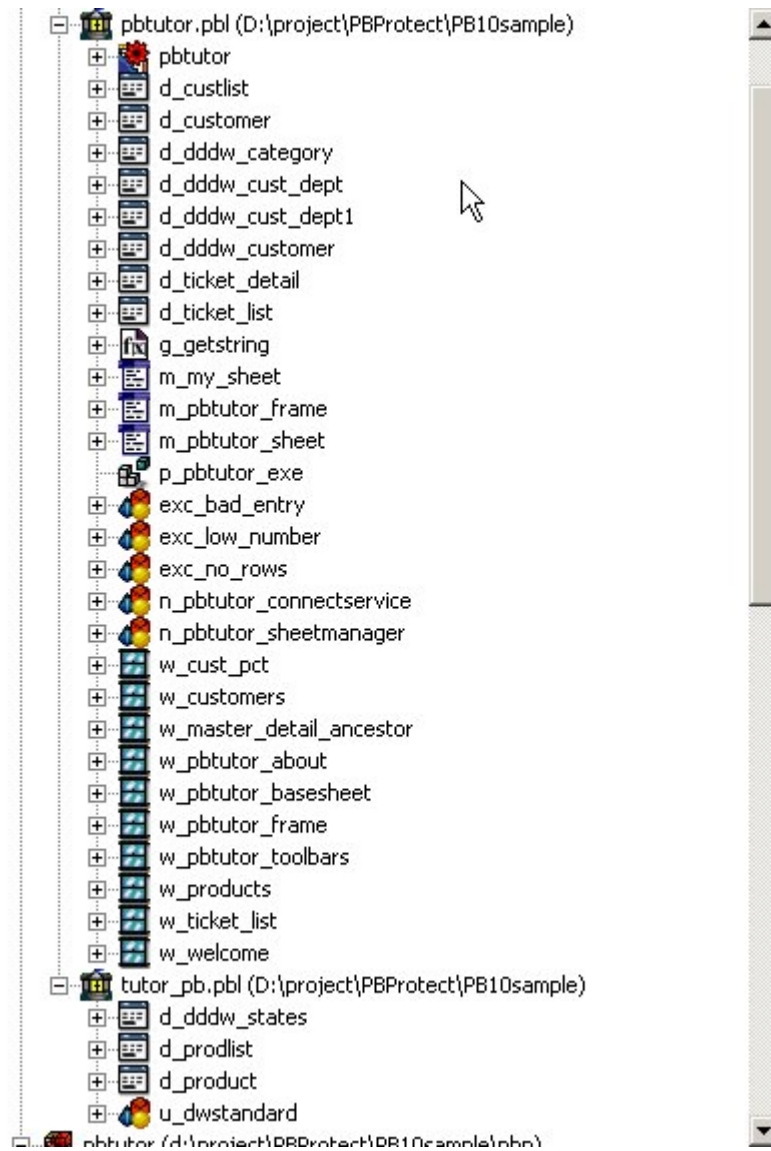
PBProtect 這樣的作法，可以避免幾個惡意的動作：

- 競爭者剽竊你的程式碼，以取得關鍵技術
- 將反組譯出來的程式碼，修改版權和畫面樣板後銷售
- 加入 Trojan 病毒或 bug，破壞企業形象
- 瞭解程式，更改程式，略過安全機制
- ...

當然還有很多負面的影響，但是要強調的是這些惡意的動作，都是來自於反組譯成功的動作。PBProtect 讓反組譯後的程式無法解讀，自然都無法做其他程式改寫的動作了。

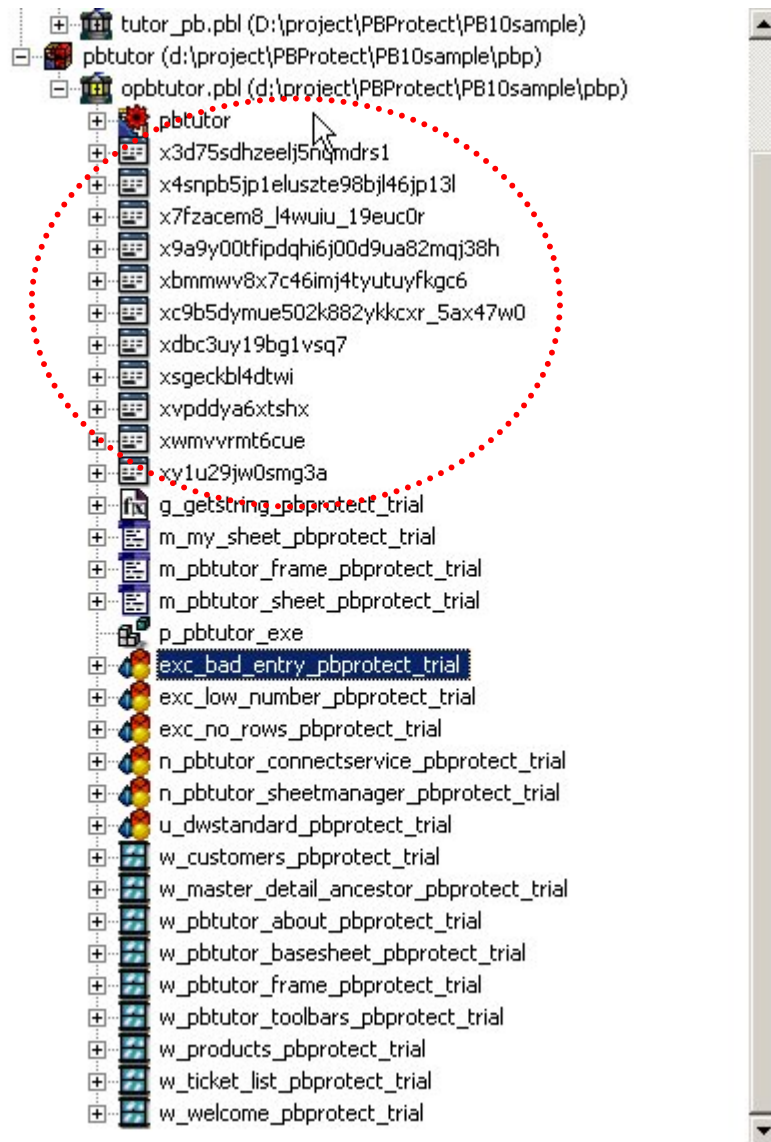
■ 範例對照

我們已下圖的範例來看，這是沒有經過混淆的原本狀況：



圖三：原本的程式

在經過混淆之後，筆者刻意只混淆 datawindow 物件，也就是以”d_”開頭的物件，其餘物件仍保留。以這樣的設定混淆後，其結果如下圖：



圖四：混淆後的程式碼

結論

PowerBuilder 程式的安全性，以往是一個不好處理的問題，因為客戶問起 PowerBuilder 程式碼的安全保證，往往缺乏有力的資料證明他安全無誤。現在 PowerBuilder 支援.NET 的作法，自然具備安全的機制。而對於 PBProtect 這個工具，藉由更改原始碼，讓程式碼即使被反組譯，也無法讀取，更提供所謂程式碼本身的安全機制。因此，對於 PowerBuilder 安全性的支援，也更為完整了。