IBM Global Services

VB2007

Novel code obfuscation with COM

Robert Freeman
Team Lead
Protection Technologies Group

IBM Internet Security Systems
Ahead of the threat.™

Agenda

- **What is code obfuscation?**
- **About this talk.**
- **COM Proxy to *Windows* API Technique.**
- **Demos.**
- **Detection opportunities and challenges.**
- **Candidacy as a future threat.**

What is code obfuscation?

- **Any time the original code intent is made difficult to ascertain.**
- **Goals of obfuscation vary: delay analysis, hope analysts miss something, bypass detection technology, etc.**
- **Scripts and compiled applications can utilize similar and dissimilar code obfuscation techniques.**
  - Dissimilar techniques highlight feasibility and relevance issues.

About this talk

- **Malware commonly uses code obfuscation whether it is a binary executable or script.**
- **Little discussion on combining the two (binary+script) may surprise us later on.**
- **Talk assumes some existing understanding of the Component Object Model (COM).**
- **Using COM, we can add named items to ActiveScript libraries such as VBScript and JavaScript to proxy calls to WinAPI from script code.**
- **Will discuss detection opportunities and challenges.**

COM Proxy Technique

- **At its core, it is like DOM creation:**
  - CoCreateInstance() with IID_IActiveScript.
  - QueryInterface() returned with IID_IActiveScriptParse.
  - SetScriptSite() with returned interface and self-created IActiveScriptSite.
  - InitNew() on IActiveScriptParse interface.
  - *AddNamedItem() to IActiveScript with your object exposing methods and properties.*

VB2007 COM Proxy Specifics

- **Many ways to implement.**
  - This will be reiterated later on.
- **This sample proxy exposes the following methods to the ActiveScript a la COM:**
  - LoadLibrary(BSTR libname, …)
  - RegisterCallback(IDispatch* callbackfunc, …)
  - BSTRtoLONG(BSTR inbstr, …)
  - Alert(BSTR alerttext)
- **Proxy in this case has zero knowledge of *Windows* libraries/APIs.**

VB2007 COM Proxy Specifics

How does it actually proxy between JS and Win32/64?

- **The IActiveScript engine will query OLE's GetIDsOfNames() for an ID to pass to the invoke() method for IDispatch objects.**
- **We can write our own IDispatch methods instead of relying on the default provided.**
- **We obtain the WinAPI call pointer during GetIDsOfNames() for OLE to pass to invoke() as the DispatchID (DISPID).**
- **invoke() accepts an arbitrary number of parameters from OLE as an array.**
- **Thus, a custom object returned by our LoadLibrary() will allow you to make calls to the *Windows* library as if the APIs were exposed by the object.**

Demonstration Placeholder

- **Scenarios to watch out for:**
  - Stand-alone EXE
  - DLL injected into other process
  - DLL used as browser plug-in
    - May need to instantiate JS/VBS libraries but not limited to one browser.

COM Proxy Specifics Reminder

- **Not all proxies need to be implemented like the sample discussed to work.**
  - GetIDsOfNames() doesn't need to be the device used to obtain an API address.
  - invoke() doesn't need to call the desired WinAPI.
- **Could do something esoteric like copying WinAPI parameters into a BSTR for further obfuscation.**
- **The type library information can be loaded either from a file/moniker or executable resource through OLE's LoadTypeLib().**
  - DIY alternative with OLE's CreateDispTypeInfo() and CreateStdDispatch().

Detecting COM Proxies to WinAPI

- **Look inside GetIDsOfNames() method:**
  - GetProcAddress() call
  - FS:[] references (incl. FS:[0])
  - References to static addresses outside of code/data pages
- **Look inside invoke() method:**
  - Any x86 LOOP instructions or equivalent?
  - x86 CALL to the DISPID sent to invoke()
  - PUSHA/POPA blocks

Detecting COM Proxies to WinAPI

- **Look inside exposed methods:**
  - Any calls to suspicious APIs using parameters passed from method input.
    - Don’t count on this!
  - Any x86 CALL to a BSTR input parameter.
    - A problem though with VARIANT type.
- **Suggestions do not cover all possibilities.**
  - Consider combining heuristics with knowledge of DOM-like creation.

COM Proxy to WinAPI Detection Challenges

- **Multiple ways to obtain the address of WinAPIs.**
- **Easy to obfuscate storage of WinAPI addresses in both proxy utility and script.**
- **Different ways to write the translation routine to place values on the stack for WinAPI calls.**
- **Stack translation need not occur in same code branch as WinAPI call.**
- **Stack translation could occur entirely within script, though it will require a helper method to write to arbitrary offsets.**
- **Proxy can have more "knowledge" than the VB2007 sample and hence reduce the arbitrary nature of the proxy implementation.**

COM Proxy to WinAPI Detection Challenges

- **The script language could be any installed ActiveScript library. The only changes needed to go from JS to VBS support is the CLASSID for CoCreateInstance()—and rewriting the script code for that language.**
- **The script (JS/VBS/etc) itself can be heavily obfuscated and simultaneously compressed/encrypted.**
  - Think of today's webfuscations if they could call Win32 APIs!
- **The script could be made to look innocuous by careful proxy design.**
- **The script component could exist remotely:**
  - WSOCK2 / WININET APIs
  - XMLHTTP ActiveX

COM Proxies:
Is this madness or is this Sparta?

- Madness:
  - Latency issues with API calls and callbacks.
  - Most proxies will be easy to detect with smart static signatures or behavioral analysis when processing malicious script.
  - Restricting availability of ActiveScript libraries thwarts attack.
- Sparta:
  - Few remaining frontiers?
  - Proxying may defeat some generic detection techniques.
  - Serial-variant lightweight proxies could be nasty.

Questions?

Thanks, the end.