http://www.nologin.org

ActiveX - Active Exploitation

01/2008

warlord
warlord@nologin.org
Share what I know, learn what I don't

Contents

Chapter 1

Foreword

First of all, I'd like to explain what this paper is all about, and especially, what it is not. A few months ago I got into the technical details of ActiveX for the first time. Prior to this point I only had some vague ideas and a general understanding of what it is and how it works. What I did first is probably quite obvious: I googled. To my surprise though, I could not find a single paper discussing ActiveX and how to exploit it. My next step was to contact some generally smart and knowledgable friends to harvest the required information from them. I was even more surprised to find that some of the most skilled people out there lacked the same knowledge that I did. Perhaps it's our common background, coming from the Unix/Linux world, but whatever the reason, I had to work to collect the information I now possess. But still, I feel like I'm the one-eyed man explaining what the world looks like to the blind.

The fact that there are tons of ActiveX exploits on Milw0rm which would suggest that the knowledge is out there by now. I wonder why no one took the time to write it all up so the less knowledgable may get into this theater as well. It's the intention of this paper to fill this gap. If you already know everything about ActiveX, if you've found your own 0day and exploited it successfully, I probably can't teach you any new tricks. Everyone else I invite to read on.

Chapter 2

Introduction

ActiveX[1] is a Microsoft technology introduced in 1996 and based on the Component Object Model (COM) and Object Linking and Embedding (OLE) technologies. The intention of COM has been to create easily reusable pieces of code by creating objects that offer interfaces which can be called by other COM objects or programs. This technology is widely used for what Microsoft calls ActiveX[2] which represents the integration of COM into Internet Explorer. This integration offers the ability to interface with Windows as well as third-party applications with the MS browser. This allows for the easy extension of functionality in the Internet Explorer by giving software developers the ability to create complex applications which can interface with websites through the browser.

There are various ways for an ActiveX control to end up on any given machine. Besides all the controls which are part of IE or the operating system, programs may install and register ActiveX controls of their own to offer a diverse set of functions in IE. Another way of installing a new control is through web sites themselves. Depending on Internet Explorer security settings, a website may try to instantiate a control, for example Shockwave Flash, and failing to do so may prompt the user to install the Shockwave Flash ActiveX control.

Security issues seems to be a constant problem with ActiveX controls. In fact, it seems most vulnerabilities in Windows nowadays are actually due to poorly-written third-party controls which allow malicious websites to exploit buffer overflows or abuse command injection vulnerabilities. Quite often these controls make the impression of their authors not having realized their code can be instantiated from a remote website.

The following chapters will describe methods to find, analyze, and exploit bugs in ActiveX controls will be presented to the reader.

Chapter 3

Control and functionality enumeration

Any given Windows installation is likely to have a significant number of registered COM objects. For the purpose of this paper, however, we are only interested in controls which may be instantiated from a website. Quite a number of the following details are taken out of the excellent *The Art of Software Security Assessment*[3], a book I strongly recommend to anyone interested in application security.

ActiveX controls are usually, but not always, instantiated by passing their CLSID to CoCreateInstance. The respective class identifier (CLSID) is used as a unique value which is associated with each control in order to distinguish it from its peers. A list of all the existing CLSIDs on a given Windows installation can be found in the registry in `HKEY_CLASSES_ROOT\CLSID`, which actually is nothing but an alias to `HKEY_LOCAL_MACHINE\Software\Classes\CLSID`.

Within the CLSID key there are thousands of different class identifiers, all of them specifying ActiveX controls. However, only a subset of those can be instantiated by a website. Controls marked as *safe for scripting* are granted this ability. To determine whether a certain control has this ability, it has to be part of the respective category. Specifically, the category can be found in the registry in the form: `HKEY_CLASSES_ROOT\CLSID\<control clsid>\Implemented Categories`. If a control is safe for scripting it may indicate this by having a subkey with the GUID `7DD95801-9882-11CF-9FA9-00AA06C42C4`. Similarly, the 'safe for initialization' category is listed in the same location, but with a slightly different GUID. Its value is `7DD95802-9882-11CF-9FA9-00AA06C42C4`.

In the end though, not being part of these categories doesn't necessarily mean that a control cannot be called from IE. The component may dynamically report itself as being safe for scripting when it is instantiated through IE. The only surefire way is to try and instantiate a control and see if it can be used. Axman[5] is an ActiveX fuzzer written by HD Moore which can automate this check for all of the different CLSIDs on a system. Another tool to enumerate the controls in question is iDefense's ComRaider[4], another ActiveX fuzzer, which has the ability to build a database of controls that IE should be able to instantiate.

3.1 ProgIDs

Besides the long and rather hard to memorize CLSID there is often a second way of instantiating a certain control. This can be accomplished through the use of a control's program ID (progID). Quite similar to IP addresses and the domain name system(DNS), progIDs can be looked up to determine the matching CLSID. Once the right one has been determined, Internet Explorer goes on as if the CLSID had been provided in the first place.

For this technique to work for a given control, two requirements must be met. First, a control must have a ProgID subkey under their CLSID key in the register. ProgIDs are usually in the form Program.Component.Version such as SafeWia.Script.1. Second, as there is no point for Windows to walk through up to 2700 CLSIDs(in my example) to find the specified ProgID, the program ID itself must have a key in `HKEY_CLASSES_ROOT` with a subkey named `CLSID` which describes makes the association.

3.2 The Kill Bit

In some cases it is desirable to restrict a control from ever being instantiated in IE. This can be accomplished through the use of a *kill bit*. The kill bit can be defined by setting the 0x00000400 bit in the `CompatibilityFlags` DWORD associated with a given CLSID:

```
HKLM\SOFTWARE\Microsoft\Internet Explorer\ActiveX Compatibility\<CLSID>
```

3.3 User Specific Controls

With Windows XP, Microsoft introduced support for user-specific ActiveX controls. These do not require Administrator-level access to install because the controls are specific to a certain user, as the name already implies. These controls can be found under `HKEY_CURRENT_USER\Software\Classes`. While this functionality exists, most ActiveX controls are installed globally.

3.4 Determining Exported Functions

ActiveX controls implement various COM interfaces in the same manner as any other COM object. COM interfaces are well-defined definitions of what functions and properties a COM class must implement and support. COM provides the ability to dynamically query a COM class at runtime using QueryInterface to see what interfaces it implements. This is how IE determines if a control supports the *safe for scripting* interface (which is called `IObjectSafety`).

Chapter 4

Examples

4.1 MW6 Technologies QRCode ActiveX 3.0

In this section the previously provided information will be demonstrated with the help of a recent public ActiveX vulnerability and exploit. The vulnerable control is from a company called WM6 and comes with their "QRCode ActiveX" version 3.0. When I downloaded the software in January 2008, several months after the exploit was posted on Milw0rm in September, the vulnerable control was still part of the package.

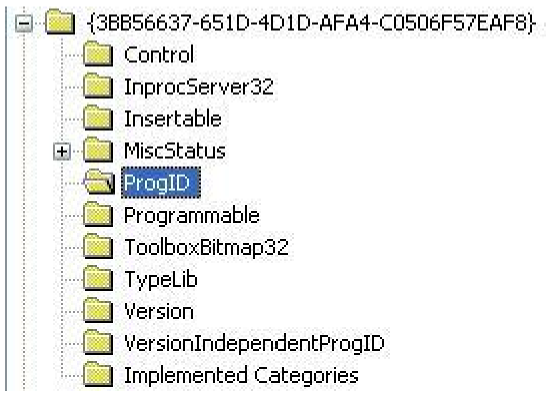
The control itself has a CLSID of 3BB56637-651D-4D1D-AFA4-C0506F57EAF8. After the installation of the software, it can be found in the registry in:

`HKEY_CLASSES_ROOT\CLSID\{3BB56637-651D-4D1D-AFA4-C0506F57EAF8}`.

The DLL that implements this control can be found on the harddrive in the file that is specified in the "InprocServer32" key. In this example it is:

`C:\WINDOWS\system32\MW6QRC~1.DLL`

A screenshot of what the entire registry entry for this control looks like:



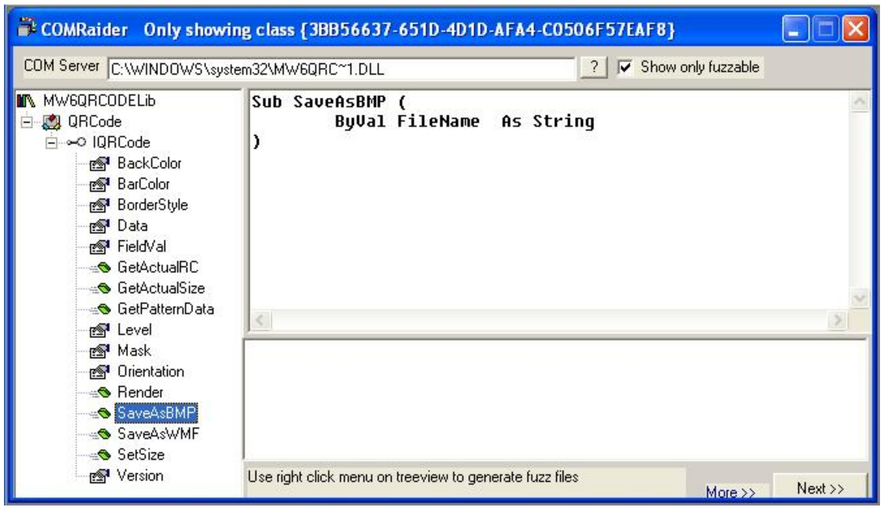
There are two interesting things to note here. For one, the ProgID key has a default value of MW6QRCode.QRCode.1. At the ProgID's corresponding location in the registry, namely `HKCR\MW6QRCode.QRCode.1`, the CLSID subkey contains the CLSID of that control. This tells us that this control can be instantiated using both its CLSID and ProgID. Another point of interest in the screenshot is the absence of the "Implemented Categories" key. This means that this control is neither part of the "safe for scripting" nor the "safe for initialization" category. However, it appears that the control must implement IObjectSafety since it is still possible to instantiate the control from IE. The following simple HTML code tries to instantiate the control.

```
<body>
<object classid='clsid:3BB56637-651D-4D1D-AFA4-C0506F57EAF8' id='test'>
</object>
</body>
```

The result of this snippet of code is the appearance of a little picture in IE. As this works just fine without Internet Explorer complaining about being unable to load the control, the next examination step is in order.

4.1.1 Enumerating Exported Interfaces

By now it has been shown that the example control can be instantiated from IE just fine. The question now is what kind of interfaces the control provides to the caller. By submitting the specific CLSID of the control that is to be examined to ComRaider, the tool lists all of the controls implemented functions as well as the kind and number of expected parameters. An alternative to ComRaider is the so-called OLE-COM object viewer that comes with the platform SDK and Visual Studio.



```
Sub SaveAsBMP (
        ByVal FileName  As String
)
```

Use right click menu on treeview to generate fuzz files

4.1.2 Exploitation

After playing around with various functions, it soon becomes obvious that `SaveAsBMP` and `SaveAsWMF` happily accept any path provided to save the generated graphic in the specified location. This can make it possible to overwrite existing files with the picture if the user running IE has sufficient access. This is a perfect example of a program using untrusted data and operating on it without any kind of checks. It is likely that the control's author did not consider the security implications of what they were doing.

A sample exploit for this vulnerability, written by shinnai, can be found on Milw0rm: http://www.milw0rm.com/exploits/4420.

4.2 HP Info Center

On December 12th, 2007, a vulnerability in an ActiveX control which was shipped by default with multiple series of Hewlett Packard notebooks was disclosed. The issue itself was found in a piece of software called the HP Info Center. The vulnerability allowed remote read and write access to the registry as well as the execution of arbitrary commands. By instantiating this control in Internet Explorer and calling the vulnerable functions it was possible to run software with the same level of access as the user running IE. Porkythepig found and disclosed this serious threat and wrote a detailed report as well as a sample exploit covering three attack vectors.

The HP control with the CLSID 62DDEB79-15B2-41E3-8834-D3B80493887A was responsible for the listed vulnerabilities. By default it installs itself into `C:\Program Files\Hewlett-Packard\HP Info Center`. In his advisory, porky listed three potentially insecure methods as well as the expected parameters:

- VARIANT GetRegValue(String sHKey, String sectionName, String keyName);
- void SetRegValue(String sHKey, String sSectionName, String sKeyName, String sValue);
- void LaunchApp(String appPath, String params, int cmdShow);

While the first and second method allow for remote read and write access to the registry, the third function runs arbitrary programs. For example, an attacker could execute cmd.exe with arbitrary arguments.

In this example the vulnerable control provided remote access to the victims machine. Sample code to exploit all three functions can once again be found on Milw0rm: `http://www.milw0rm.com/exploits/4720`.

4.3 Vantage Linguistics AnswerWorks

The third and last example of various ActiveX vulnerabilities is in the Vantage Linguistics AnswerWorks. Advisories covering this vulnerability were released in December, 2007. The awApi4.AnswerWorks.1 control exports several functions which are prone to stack-based buffer overflows. The functions `GetHistory()`, `GetSeedQuery()` and `SetSeedQuery()` fail to properly handle long strings provided by a malicious website. The resulting stack-based buffer overflow allows for the execution of arbitrary code, as "e.b." demonstrates with a proof of concept that binds a shell to port 4444 when the exploit succeeds.

When the exploit is loaded from a webserver it instatiates the CLSID and links the created object to a variable named `obj`. It then calls the `GetHistory()` function with a carefully crafted string which consists of 214 A's to fill the buffer followed by a return address which overwrites the one saved on the stack. After those 4 bytes come 12 NOPs and then finally the shellcode. As one can easily see, this exploit is based on the same techniques that can be seen in many other stack-based exploits.

The exploit mentioned in this example can also be found on Milw0rm: `http://www.milw0rm.com/exploits/4825`.

Chapter 5

Summary

This paper has provided a brief introduction to ActiveX. The focus has been on discussing some of the underlying technology and security related issues that can manifest themselves. This was meant to equip the reader with enough background knowledge to examine ActiveX controls from a security point of view. The author hopes he managed to describe the big picture in enough detail to provide readers with enough information on the matter to base further research on the aquired knowledge.

5.1 Acknowledgements

wastedimage - For answering the first questions
deft - For providing lots of answers and examples
rjohnson - For filling in details deft forgot to mention
skape - For background knowledge on underlying functions
hdm - For knowing all the rest

Bibliography

[1] ActiveX Controls @ Wikipedia
http://en.wikipedia.org/wiki/ActiveX_control

[2] ActiveX Controls
http://msdn2.microsoft.com/en-us/library/aa751968.aspx

[3] The art of software security assessment
http://taossa.com

[4] ComRaider
http://labs.idefense.com/software/fuzzing.php#more_comraider

[5] Axman ActiveX Fuzzer
http://www.metasploit.com/users/hdm/tools/axman/