

# ActiveX

A security testing methodology

Casaba Security, LLC  
[www.casabasecurity.com](http://www.casabasecurity.com)

**casaba**

# Table of Contents

---

<b>1</b>	<b>INTRODUCTION .....</b>	<b>3</b>
<b>2</b>	<b>TESTING ACTIVEX CONTROLS.....</b>	<b>3</b>
2.1	ACTIVEX CONTROLS ARE NOT ALONE .....	4
<b>3</b>	<b>IDENTIFY ACTIVEX CONTROLS INITIALIZATION AND SCRIPTING SAFETY (SFI AND SFS) .....</b>	<b>4</b>
3.1	CLSID'S AND PROGID'S .....	5
3.2	KILL-BITS AND PHOENIX-BITS .....	6
<b>4</b>	<b>ENUMERATING THE ACTIVEX CONTROL - INTERROGATION TOOLS.....</b>	<b>6</b>
<b>5</b>	<b>UNDERSTANDING SAME-ORIGIN-POLICY AND SITELOCK .....</b>	<b>8</b>
5.1	<i>SAME-ORIGIN-POLICY</i> .....	8
5.2	<i>SITELOCK</i> .....	11
<b>6</b>	<b>FUZZING AND INPUT TESTING .....</b>	<b>12</b>
<b>7</b>	<b>REPURPOSING THE ACTIVEX CONTROL'S FUNCTIONALITY.....</b>	<b>15</b>
7.1	CATEGORY: OBJECT CREATION LEADS TO REMOTE CODE EXECUTION .....	15
7.2	CATEGORY: UPLOADING, DOWNLOADING, CREATING, AND EXECUTING ARBITRARY FILES.....	15
7.3	CATEGORY: VIOLATING OR BYPASSING SECURITY ZONES AND SAME-ORIGIN-POLICY .....	16
<b>8</b>	<b>TARGETTING CODE REVIEWS.....</b>	<b>16</b>
<b>9</b>	<b>AUTHOR BIOGRAPHY .....</b>	<b>16</b>

## 1 Introduction

ActiveX controls, like any other browser plugin, provide a ripe attack surface for the malicious. Finding an exploitable flaw in a popular control gets MSRC attention at Microsoft, and similar attention at other large companies.

You'll sometimes hear ActiveX and COM used interchangeably, that's because at its core an ActiveX control is a COM object exposed through scriptable interfaces.

In this short testing article we'll be covering the fundamentals of security testing an ActiveX control. The process goes something like this:

- Identifying whether a control is safe for scripting/safe for initialization
- Same-origin-policy
- Figuring out how the control works
- ProgId and CLSID
- Kill bits
- Methods and parameters
- Input testing and fuzzing
- Repurposing or logical testing

Tools to interrogate and understand an ActiveX control - ActiveX Safety Detailer, OLEView.

Tools for fuzzing input parameters of an ActiveX control – COMRaider and AxMan.

We'll be using some tools from [Hunting Security Bugs](#), which is recommended reading to further your research in this area. You can download companion content:

<http://www.microsoft.com/mspress/companion/0-7356-2187-X/>

## 2 Testing ActiveX Controls

ActiveX controls are typically native code (e.g. C++) compiled binaries registered with the Windows operating system. Through a registration process the ActiveX control is considered scriptable, meaning that Internet Explorer can load the control and HTML or javascript can interact with it. Because ActiveX controls run native code in the browser, they can serve as an extension to the browser. This can lead to numerous security threats not the least of which being that the control can bypass Internet Explorer's most precious security defenses like 'security zones' and 'same-origin-policy.'

As someone testing ActiveX controls you'll want to try these things:

- Identify if the control is SFI/SFS
- Enumerate the control's methods, properties, events and parameters

- Understanding same-origin-policy and sitelock
- Test the parameters through input fuzzing or other means
- Try to repurpose the methods
- Utilize the code (assuming you have access) for targeted code reviews to assist testing

Prioritization is a common theme when testing software and is no different here. If you only have a short time to dedicate, then set clear goals for yourself. Before going too much further, I highly recommend you grab the book [Hunting Security Bugs](#). It was written by the Microsoft Office security test team and includes the most comprehensive coverage of software security testing and ActiveX controls.

## 2.1 ActiveX controls are not alone

I don't fall victim to the common FUD that ActiveX controls are bad, and everyone should switch browsers because they're so insecure and damaging. It's a common reaction for some people to jump at the chance to slander Internet Explorer, and use the history of ActiveX control vulnerabilities as fuel for the fire. Maybe there's some justification there, but the fact is that almost all modern web browsers allow third-party native code to run inside them. Yes it's true; in reality ActiveX controls aren't much different than what other browsers call plug-ins, extensions, or add-ons. Don't let the marketing fool you in these cases.

## 3 Identify ActiveX Controls initialization and scripting safety (SFI and SFS)

ActiveX controls are COM objects registered with Windows like any other. However, they're called 'ActiveX controls' when they're marked safe for initialization and scripting and when they implement the IDispatch or IDispatchEx interfaces. All COM objects can be instantiated in Internet Explorer, and IE will query some of their interfaces to see if they're SFI/SFS. You can't interact with them through HTML and script unless they're marked SFI and SFS. This process is described more in [INFO: How Internet Explorer Determines If ActiveX Controls Are Safe](#).

Once marked SFI/SFS, the control can be manipulated through HTML or javascript in Internet Explorer.

The programmatic details for determining if an object is marked Safe for Initialization are described in the KB article above. You can also check the registry, where the GUID value 0C3C509F-111A-4E6C-B270-1D64BCFD26F9 below corresponds to the CLSID of the control.

```
[HKEY_CLASSES_ROOT\CLSID\  
    {0C3C509F-111A-4E6C-B270-1D64BCFD26F9}\  
        Implemented Categories\  
            {7DD95802-9882-11CF-9FA9-00AA006C42C4}]
```

In this case, {0C3C509F-111A-4E6C-B270-1D64BCFD26F9} is the CLSID of the COM object/ActiveX control. The Implemented Categories\{7DD95802-9882-11CF-9FA9-00AA006C42C4} is a static value that can be applied to any COM object to mark it SFI.

Determining safe for scripting (SFS) can be done programmatically if the control implements IObjectSafety along with IDispatch or IDispatchEx. You can query this programmatically as described in the KB article above. Like SFI, a registry key value can be set as well:

```
[HKEY_CLASSES_ROOT\CLSID
  0C3C509F-111A-4E6C-B270-1D64BCFD26F9]\
  Implemented Categories\
    {7DD95801-9882-11CF-9FA9-00AA006C42C4}]
```

Again, in this case, {0C3C509F-111A-4E6C-B270-1D64BCFD26F9} is the CLSID of the COM object/ActiveX control. The Implemented Categories\{7DD95801-9882-11CF-9FA9-00AA006C42C4} is a static value that can be applied to any COM object to mark it SFS.

AxDetail from [Hunting Security Bugs](#) shows the SFS/SFI settings, and other relevant information, just pass it the CLSID of the control you want to interrogate:

```
>AXDetail.exe {0C3C509F-111A-4E6C-B270-1D64BCFD26F9}
ActiveX Safety Detailer Version 1.0. Copyright (c) 2006 Microsoft
Corporation. All Rights Reserved.
CLSID: {0C3C509F-111A-4E6C-B270-1D64BCFD26F9}
Module: C:\tools\Sprocket.ocx

IObjectSafety call returned E_NOINTERFACE.

Actually Calling interfaces to see what is available.

IPersistPropertyBag is implemented.
IPersistPropertyBag2 returns E_NOINTERFACE.
IPersistStorage is implemented.
IPersistStream returns E_NOINTERFACE.
IPersistStreamInit is implemented.
IPersistMemory is implemented.
IPersistFile returns E_NOINTERFACE.
IPersistMoniker returns E_NOINTERFACE.
IDispatchEx returns E_NOINTERFACE.
IDispatch is implemented.

Registry SFS/SFI Component Categorization Analysis:
Component is marked SFI in registry.
Component is marked SFS in registry.

ActiveX Compatibility Flags Analysis:
There are no ActiveX Compatibility flags set.
```

### 3.1 CLSID's and ProgID's

That's all great, but at this point you might be wondering – **how the heck can I find the CLSID of the control?** Great question - here are a few tips:

- Monitor the registry with regmon.exe, from Sysinternals, (or some other installation monitoring software) while you're installing the ActiveX control. Then go back and look for registry writes to CLSID which write a GUID key and subkeys.
- Look at the HTML and javascript source of the webpage which invokes the control – it will either call it by ProgID or CLSID.

The human-readable equivalent of the CLSID, which is in GUID format, is the ProgID. Compare the two below, both of which represent the same ActiveX object:

- CLSID: {0C3C509F-111A-4E6C-B270-1D64BCFD26F9}
- ProgId: SPROCKET.Control.1

### 3.2 Kill-bits and Phoenix-bits

I'll close off this section by mentioning two ways that ActiveX controls are effectively disabled. You can refer to the Microsoft KB article: [How to stop an ActiveX control from running in Internet Explorer](#) for the gritty details.

Basically, a kill-bit is a registry key setting that tells Internet Explorer not to load the control, period. If this registry value exists, the control is useless in IE or any other kill-bit aware application.

The [phoenix-bit](#) (aka AlternateCLSID), is a pointer to a new CLSID for the control. This enables developers to install an updated control with a safe, new CLSID, and have the control still be usable. IE will see the AlternateCLSID value when the old control is called, and map requests for the old CLSID to the new one.

#### Typical bugs or flaws we see around SFI/SFS violations:

- Controls get marked SFI/SFS that were never intended to be. Maybe they were marked SFS for internal test passes, but later the SFS settings were never removed before release.
- COM objects that were never marked SFI/SFS still get instantiated in Internet Explorer as part of the process to query their interfaces for SFI/SFS information! This means a faulty COM object that was never intended to be created in IE could actually crash IE. Note that this behavior has changed slightly with IE7's opt-in policy. More information at: [Microsoft Internet Explorer can use any COM object](#)

## 4 Enumerating the ActiveX control - Interrogation Tools

Once you know the CLSID or ProgID of your control, you'll want to see what it's made of by checking out its properties, methods, and parameters. A good place to start is with the HTML that invokes the control and makes it do stuff. If you know the webpage that hosts the control, you can view the HTML and javascript source for strings like:

## ActiveX – security testing methodology

- object
- CLSID
- Classid
- ActiveXObject

For example, the control could be called through script like:

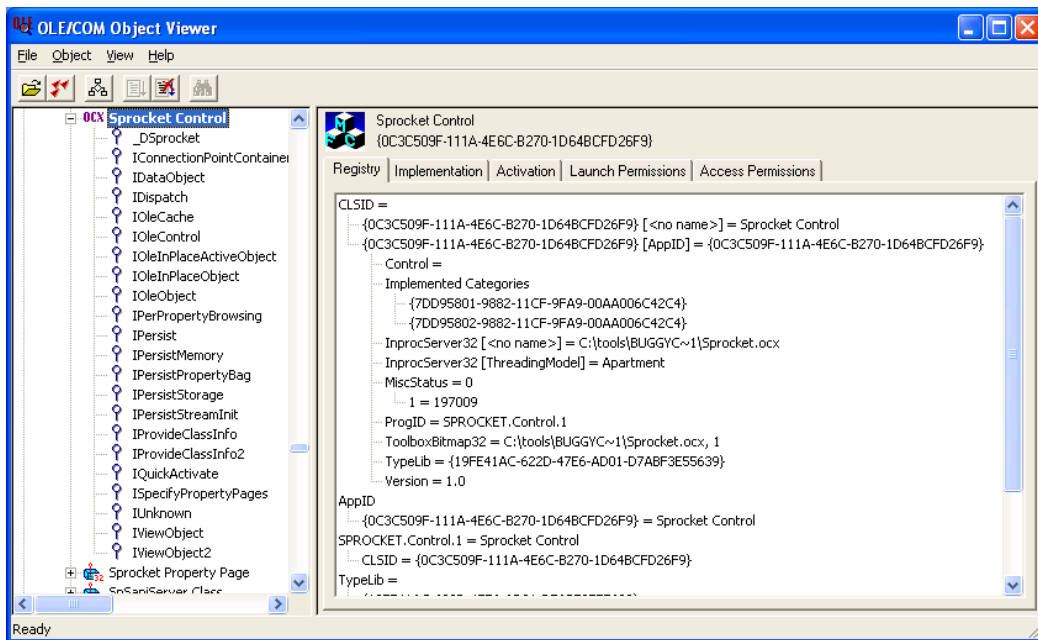
```
<script>
fso = new ActiveXObject("Scripting.FileSystemObject");
</script>
```

Or it could be embedded in the HTML and called with the <object> element like:

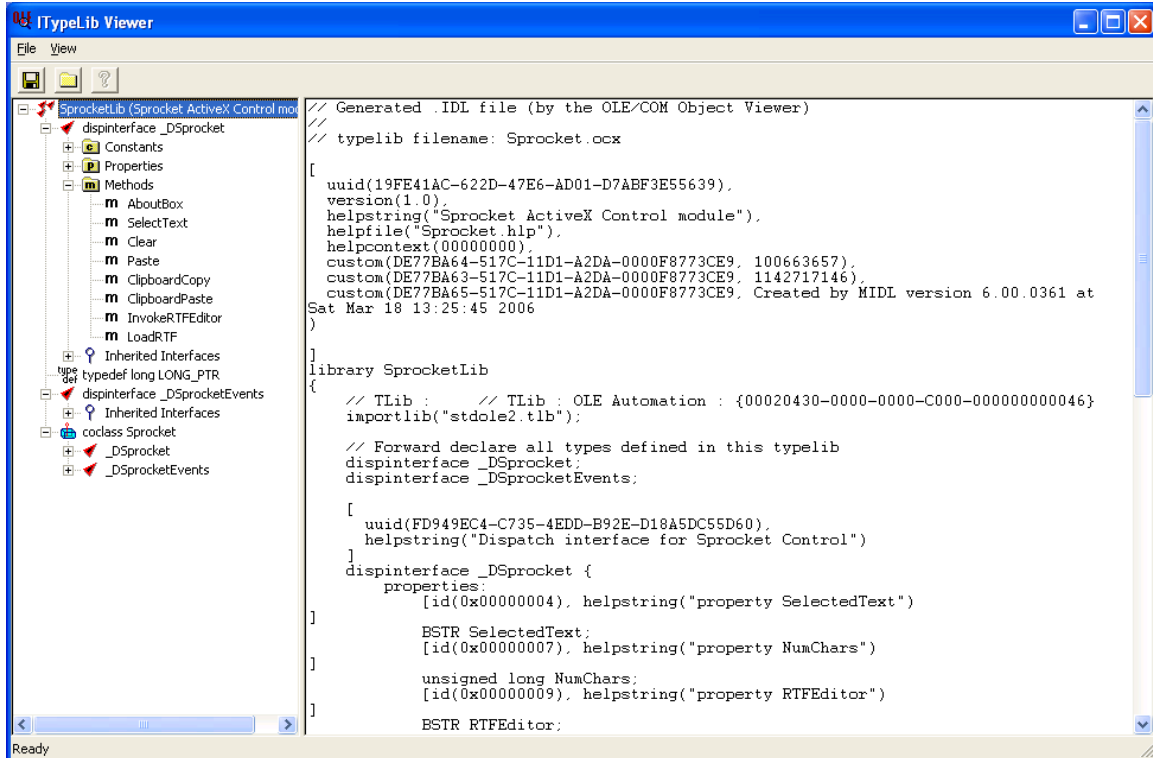
```
<OBJECT id="Editor" classid="clsid:0C3C509F-111A-4E6C-B270-
1D64BCFD26F9" height=300 width=300>
</OBJECT>
```

Aside from interacting with the control through Internet Explorer, you can use some other tools which act as similar COM containers. Grab Oleview.exe which installs with [Visual Studio](#), and also with the [Windows Server 2003 resource kit](#).

Oleview.exe queries the interfaces and expands the control to show you the exposed methods, properties and parameters you're interested in for testing.



It also has an integrated type library viewer which generates an Interface Definition Language (IDL) file. This is useful from viewing the functionality of the control in one place, and for building programmatic test harnesses if you want.



Remember, these tools are just to aid you in reviewing the functionality of the control. Two of your main goals during testing are to a) test the inputs and b) attempt to repurpose the control to perform malicious actions.

## 5 Understanding Same-origin-policy and Sitelock

Let's get something straight, ActiveX controls can mostly do what they want, meaning they can bypass the same-origin-policy (SOP) and concept of 'security zones'. Don't let SOP and sitelock confuse you, they're different concepts, but both are a form of 'domain restriction' making it appropriate to discuss them together.

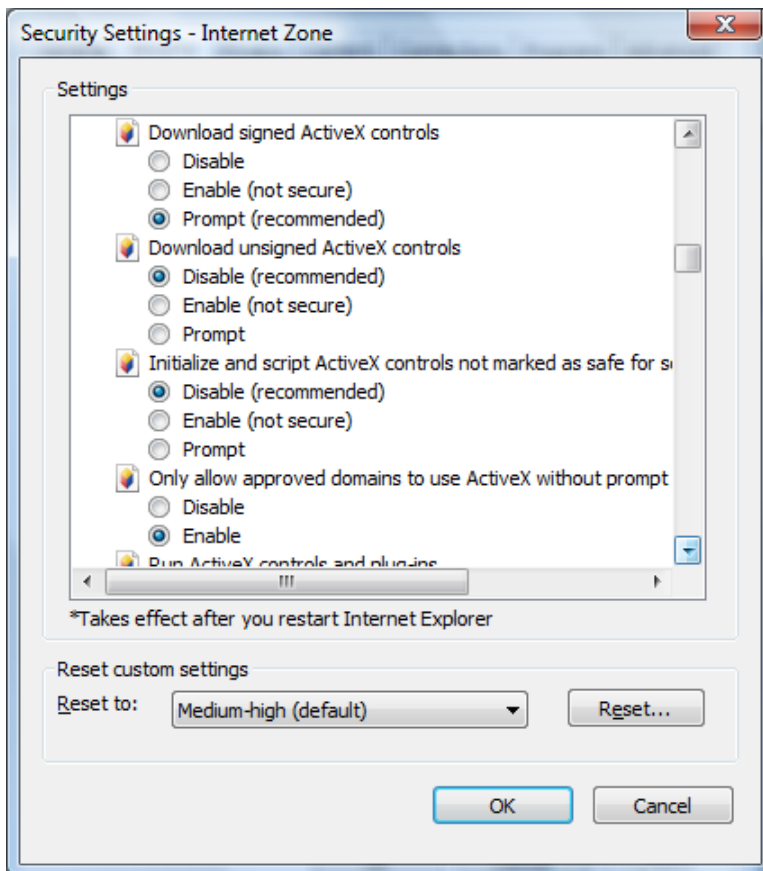
### 5.1 Same-origin-policy

Same-origin-policy is a web-browser concept originating with Netscape years ago. It's become a major security feature of all modern web-browsers. SOP enforces the rule that 'script originating from one domain should not be able to read, write, or infer content originating from another domain.' This is an incredibly important rule, if it didn't exist, we'd have zero security protections on the wild wild web. Just think about it, if your browser allowed <http://nottrusted.com> to manipulate the content originating from <http://yourbank.foo> then you probably wouldn't want to do business on the Web. Describing this in detail could take up its own article, so I'll just give a few more examples of the same-origin-policy restrictions set forth

in modern browsers. So if `http://microsoft.com` is the domain of origin, what would be considered the same domain, and what different?

- `http://a.microsoft.com` would be different (different Full Qualified Domain Name (FQDN) of origin)
- however there's an easy way around this - the browser will allow `*.microsoft.com` without hassle (because `document.domain` can be set to level 2 domain.tld - aka acceptable client-side mashup).
- `https://microsoft` would be different (protocols are different)
- `http://microsoft.com:81` would be different (ports are different)
- `http://live.com` would be different (domains are different)

Note this is not the same as Internet Explorer's concept of 'security zones'. Security zones have long been an important part of IE's security model. They provide boundaries for categorizing domains into four zones – Restricted, Internet, Intranet, and Trusted. Each zone has a default and customizable set of granular configuration options. For example, script is allowed to run in the Internet zone by default, but not in the Restricted zone. As you can imagine, the rules around ActiveX controls are a little more strict in the Internet zone than they are in the Trusted zone. The following image shows a few of these ActiveX control settings from the Internet zone.



I wanted to discuss some general scenarios where SOP comes into action in Internet Explorer, and others where you'd think it does, but it (somewhat confusingly) doesn't. To make matters more confusing, each browser can implement SOP differently.

When are cross-domain actions allowed by the browser? I can't cover everything in this short article, but a couple of interesting points to note are:

- Cross-domain form POSTs are actually allowed, contrary to popular belief.
- This means I can host a form on <http://nottrusted.com> that POSTs data to <http://microsoft.com> or some other domain. This starts to look like a cross-site request forgery (CSRF or one-click) attack but also intuitively seems like an SOP violation, although it's not.
- Including cross-domain script files via `<script src="file.js">` is allowed. So if the browser loads HTML from <http://microsoft.com> and there's a `<script src=http://nottrusted.com/evil.js>` reference, the browser allows loading the evil javascript into the DOM

#### When `<script src="somefile.js">` becomes an issue:

- If you search the web for JSON Hijacking you'll find examples of techniques people used to exploit JSON response data. It sort of works like a CSRF attack but is quite different. Because SOP allows access to remote script files in other domains, those script files have access to the originating DOM – you probably understand that already. What happens when the response from `somefile.js` is a JSON serialized object? The answer is nasty things.

For example, you visit <http://nottrusted.com> which includes `<script src=http://acmebank.foo/ajaxpage?action=getPersonalInfo>`. Your browser attempts to fetch the script src, and since it's an AJAX page with a JSON response, the browser consumes the personal info and <http://nottrusted.com> has access to it.

#### Typical bugs or flaws we see around same-origin-policy violations:

- The control performs HTTP requests at a low-level, subverting IE's SOP policy
- The control performs HTTP or other network operations (file uploads or downloads typically) to user-controllable domains (e.g. domain values get set through a parameter or property). An attacker can repurpose the control to upload to or download from arbitrary servers.

- The control does the above things, and also includes the user’s cookie from the originating domain. In other words, the control will get launched from one domain, and send an HTTP request to another domain, including the cookie from the originating domain.

## 5.2 Sitelock

Sitelock provides a way to restrict which domains can manipulate a control’s methods, properties, events and parameters. You can download the template for it here:

<http://www.microsoft.com/downloads/details.aspx?FamilyID=43CD7E1E-5719-45C0-88D9-EC9EA7FEFBCB&displaylang=en>.

NOTE: It might be a good idea to suggest your development team implement sitelock or some form of domain restrictions if appropriate.

Sitelock provides a new interface called `IObjectSafetySiteLock` which a control can implement to enforce a whitelist of domains allowed to use the control. I won’t go into the details too much because you can download the code and the documentation to see that.

The package also comes with a useful tool called `sitelist.exe` which will query the `IObjectSafetySiteLock` interface to retrieve the list of allowed domains. This is important to do for sanity checks. Here’s an example usage:

```
>sitelist.exe {0C3C509F-111A-4E6C-B270-1D64BCFD26F9}
SiteList: Utility to dump domain list from a site-locked ActiveX
control.

IObjectSafetySiteLock not implemented.
```

In this case sitelock was ‘not implemented’ but if it were you’d see a list of allowed domains returned.

### Typical bugs or flaws we see here:

- Test domains slip through into release builds
- The INTRANET domain slips into release builds (this means any site in the Intranet zone can use the control)
- Wildcards are misplaced (e.g. \*.microsoft.com might be okay but \*microsoft.com would not be!)
- The list grows over time and doesn’t get purged or cleaned out

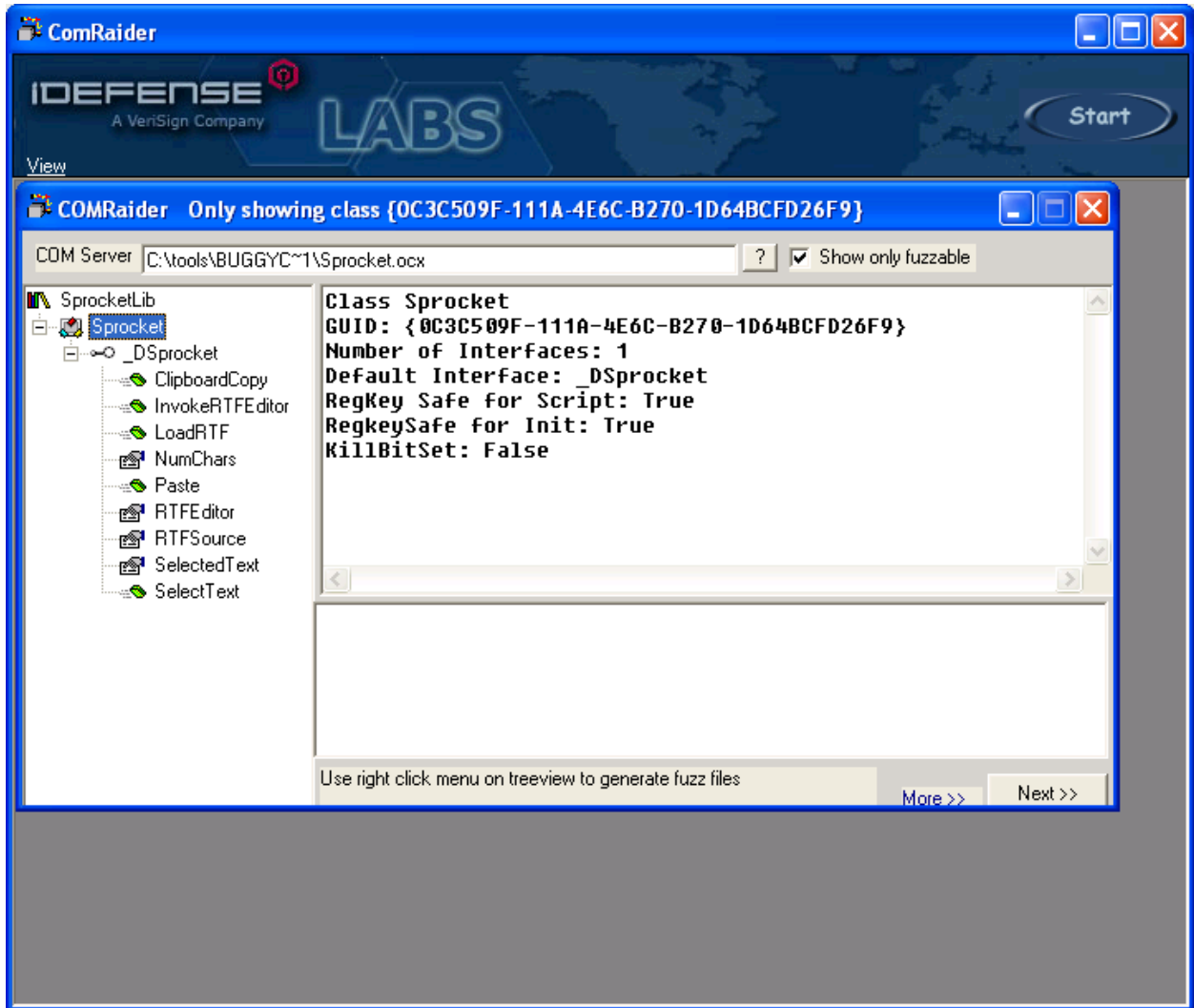
## 6 Fuzzing and Input Testing

You can create HTML test files yourself that call the methods, properties, parameters, and events of the control, and that's probably a good idea. But if you want to get started quickly there are some tools you can kick off while you're preparing your test harness.

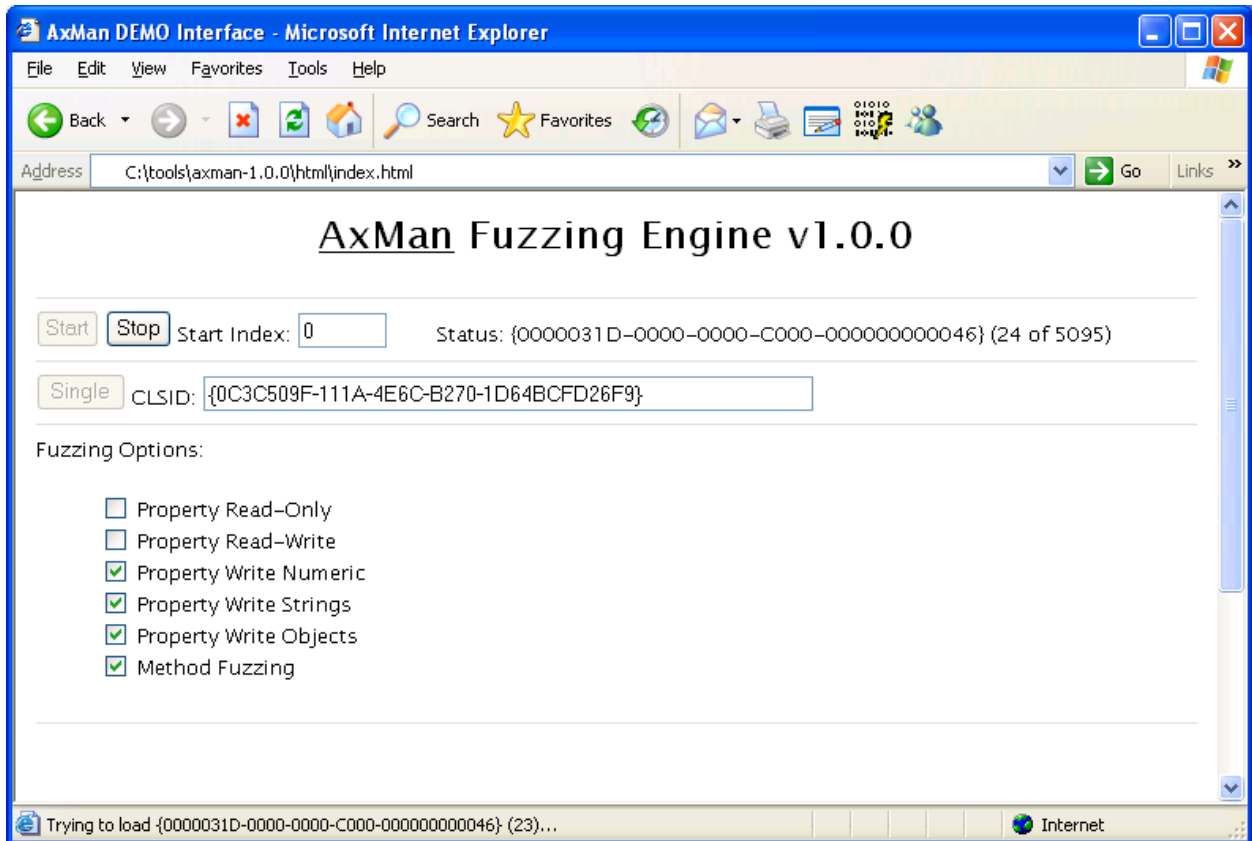
[IDefense's COMRaider](#) by David Zimmer provides a nice UI for interrogating controls and fuzzing them. You can in fact do some neat things with this:

- scan your system and enumerate all SFS objects
- load COM and ActiveX binaries directly
- use the integrated type library viewer to analyze the exposed methods and parameters
- automate the fuzzing process with integrated debugging for exception logging
- support for symbols

In addition to actually catching crashes and helping to analyze them with the integrated debugger, COMRaider will save all test cases as .vbs files which you can use to reproduce any crash.



HD Moore's AxMan is another tool to help you fuzz-test your ActiveX controls quickly. It loads up as an HTML page and calls an executable to drive the fuzzing. This tool has helped to identify numerous memory corruption vulnerabilities which led to MSRC cases.



Of course you can always test inputs yourself, either manually or by building your own tools. Start out with the HTML. If you know the control you want to test, then it's sometimes best to just use the HTML that you know invokes the control. You can get this from the developer usually, or if you don't know the developer you can get it from the webpage which initially loads the control (see earlier section). Having the HTML, or creating it on your own can be very helpful for input testing, and for repurposing testing.

With just the basic HTML, you can write a little javascript to automatically invoke the control and send it the types of inputs you want – nulls, long strings, ints, longs, floats, off-by-ones, and format strings like %n%n%n. Unicode characters are also good to try for inputs, and often help shake out bugs in character encodings and conversions.

If you want to really get involved with this, check out the [Peach Fuzzing Platform](#) which will provide you a solid framework from which to build your own fully customizable fuzzer. Experiment with some of these concepts yourself – there are plenty of good fuzzing references out there to assist.

## 7 Repurposing the ActiveX control's functionality

This section probably deserves the most attention, and likely requires the most creative thinking from the tester. For the most information on this subject I suggest chapter 18 of the [Hunting Security Bugs](#) book which walks you through a very technical and deep discussion of repurposing attacks.

Repurposing is a term meaning – how can you abuse the logic of the control to use it in ways that were unintended. The best way to understand this is to look at some recent examples. I'll group these into different categories since some are similar:

### 7.1 Category: Object creation leads to remote code execution

#### Vulnerable controls:

WMI Object Broker ActiveX Vulnerability ([CVE-2006-4704](#))

RDS.Dataspace ActiveX Vulnerability ([CVE-2006-0003](#))

**Description:** These two controls exposed methods which could be used to invoke other COM and ActiveX controls on the system. Why's this bad? Because it gave access to COM objects that weren't marked SFS, and it could bypass kill-bit detection as well. Imagine giving an attacker the ability to leverage one method to access any installed COM or ActiveX object on your system, the FileSystem object, Microsoft Office's controls, etc. That's giving away a lot of power.

### 7.2 Category: Uploading, downloading, creating, and executing arbitrary files

#### Vulnerable controls:

WeOnlyDo! SFTP ActiveX control ([CVE-2006-1175](#))

First4Internet CodeSupport ActiveX control ([CVE-2005-3650](#))

Microsoft Log Sink Class ActiveX control ([CAN-2005-0360](#))

CA-2000-12 HHCtrl ActiveX Control ([CVE-2000-0201](#))

**Description:** Too often it's the case where functionality exists that downloads and/or uploads arbitrary files from and to arbitrary servers. Sometimes you'll even see where an ActiveX control can create or even execute any file on the filesystem! Imagine you've just installed some new software which included an ActiveX control that was not sitelock protected. You visit a webpage that calls the control and uses it to download malware to your system. The site then uses the control to upload whatever files it wants from your system. It's important to recognize the impact of allowing any domain to control this behavior, and understand that sitelocking to a trusted domain would have significantly mitigated the attack.

### 7.3 Category: Violating or bypassing security zones and same-origin-policy

#### Vulnerable controls:

IE WebBrowser ActiveX control ([CVE-2004-0549](#))

IE WebBrowser ActiveX control ([CAN-2003-0814](#))

IE DHTML Editing control ([CAN-2004-1319](#))

Microsoft Windows HTML Help ActiveX control ([CAN-2004-1043](#))

**Description:** These are some of the most scary cases, when a control can be used to read and modify HTML/DOM content in arbitrary domains outside of the domain used to host the control. Also, you might see the case where the control can be used to execute script in an elevated security zone like Intranet, Trusted, or Local.

## 8 Targetting code reviews

To wrap things up I'll just mention the importance of code review for identifying vulnerabilities. If you have access to the code, you can review the exposed methods directly to get an idea of what they're capable of doing. I've found many flaws this way, including buffer overflows, repurposing, and same-origin-policy violations like I've described above. Take the time to get familiar with the source if you can, and get the developer to talk you through the design and spec so you can approach the security review with full-knowledge and find the important issues.

If you have any comments or questions please contact me using my information below.

## 9 Author Biography

Chris Weber is a founding member of Casaba Security, a [software security consulting](#) firm. At Casaba he specializes in business and security testing for global software vendors, social networking and instant messaging providers.

Chris has authored several security books, articles and presentations. He has identified hundreds of security vulnerabilities in many widely used software products. He was recently technical editor of [Hunting Security Bugs](#) from MS Press.