Vulnerability Detection in ActiveX Controls through Automated Fuzz Testing

Will Dormann and Dan Plakosh

CERT® Coordination Center
Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612
http://www.cert.org

Abstract. Vulnerabilities in ActiveX controls are frequently used by attackers to compromise systems using the Microsoft Internet Explorer web browser. A programming or design flaw in an ActiveX control can allow arbitrary code execution as the result of viewing a specially-crafted web page. In this paper, we examine effective techniques for fuzz testing ActiveX controls, using the Dranzer tool developed at CERT. By testing a large number of ActiveX controls, we are able to provide some insight into the current state of ActiveX security.

1 Introduction

We live in a world where software vulnerabilities are pervasive. One important aspect of a vulnerability is how it can be reached, or what are its attack vectors. In the early days of the internet, server-side vulnerabilities were targeted the most. For example, the Morris Worm of 1988 worked by exploiting vulnerabilities in sendmail and fingerd [1]. Even as late as 2001, vulnerabilities in high-profile network server software were widely exploited [2]. As the internet landscape has changed, there has been a shift in focus to client-side vulnerabilities [4]. With most software vulnerabilities, the attack vector is to cause the vulnerable application to process specially-crafted data. With server-side applications, such as sendmail or fingerd, an attacker may connect to the vulnerable service as a client and send data that was crafted in a way that causes the service to crash in an exploitable manner. As time passed, popular services have become robust through thorough testing, and firewalls have become more commonplace, which restrict the number of server-side applications an attacker may be able to access. But at the same time, the number of client systems connected to the internet and the functionality of client software has increased dramatically.

Web browsers are of high interest when it comes to client-side vulnerabilities. The normal operation of a web browser is to parse, interpret, and render untrusted content that is provided by a web server. In the early days of the web, a web browser's primary functions were text layout, image rendering, and hyperlink navigation. Even with this limited functionality, viewing a malicious web page could cause the execution of arbitrary code. For example, the image

rendering library used by a web browser may crash when processing a crafted JPEG image [3]. Depending on the details of the crash, it may be exploitable to run code of the attacker's choice. The introduction of Netscape 2.0 in March of 1996 brought with it the support of a scripting language called JavaScript. While JavaScript revolutionized user interactivity with web pages, it also turned out to be a technology that is frequently leveraged by attackers.

With the release of Internet Explorer 3.0 in 1996, Microsoft introduced support for ActiveX, which originated as the Component Object Model, or COM [5]. COM allows developers to make reusable objects that can be used by other applications. COM objects can be written in a variety of programming languages. The minimum requirement is that the object implements the *IUnknown* interface [6]. A COM object that has been designed for use in the Internet Explorer web browser is commonly referred to as an ActiveX control [7]. With its support for ActiveX controls, Internet Explorer allowed for the creation of web pages that had never-before seen levels of functionality. ActiveX controls are not limited by a sandbox like Java applets [8], and any Windows developer could easily make their code available for use in the Internet Explorer web browser. Internet Explorer's support for both ActiveX and scripting languages gives the browser a large attack surface [9] and a high level of control, which makes it a primary target for attackers.

Because of the ubiquity of Internet Explorer and ActiveX, CERT developed a fuzz tester [10] for detecting vulnerabilities in ActiveX controls. The remainder of this paper is structured as follows. Section 2 looks at some historical events and also other rationale for investigating ActiveX security. Section 3 describes the attack surfaces of ActiveX controls. Section 4 discusses how we test these attack surfaces of ActiveX controls, and also describes the techniques used to automate the test processes. As the result of testing a large number of ActiveX controls, we have been able to compile the results to obtain the statistics listed in section 5.

2 History and Rationale

On August 22 and 23, 2000, the CERT Coordination Center held a workshop in Pittsburgh, Pennsylvania to discuss security issues related to ActiveX controls [11]. The result of this workshop was a paper that thoroughly documents ActiveX, its risks, and the actions that end-users and developers can use to more safely use ActiveX. In this paper, several ActiveX vulnerabilities are described, including the Microsoft scriptlet.typelib and Eyedog vulnerabilities [12]. The scriptlet.typelib ActiveX control contained unsafe methods, yet it was marked as "Safe for Scripting" in Internet Explorer. As a result, this control was widely exploited by the KAK virus and the Bubbleboy exploit [11]. The Eyedog ActiveX control contained unsafe methods and was also vulnerable to buffer overflow [12].

On November 2, 2004, Internet Exploiter 0.1 was published by Berend-Jan Wever to the Full-Disclosure mailing list [13]. This segment of JavaScript code prepares heap memory on client systems to facilitate exploitation of bugs that

cause memory access violations. In this particular case, it was used to exploit the Internet Explorer IFRAME element buffer overflow [14].

On March 1, 2005, Shane Hird posted a message to the Bugtraq mailing list titled "IObjectSafety and Internet Explorer" [15]. This post describes the process that Internet Explorer uses to determine if an ActiveX control is safe for scripting or initialization. When Internet Explorer checks if a control implements the IObjectSafety interface, it requires the COM server process to start. In certain cases, this can cause Internet Explorer to crash in a way that may be exploitable to execute arbitrary code [16]. Hird described this process as a "design flaw." A tool called axfuzz [17] was released, which can check if a control implements IObjectSafety and can perform basic fuzz tests.

The combination of the publicly-available Internet Exploiter JavaScript code and the publicly-available axfuzz testing tool made it easier for people to discover ActiveX vulnerabilities [18]. The first vulnerable COM object that CERT/CC discovered that crashes in manner described by Hird is the BlnMgr Proxy COM object [19], which comes with Microsoft Office. This vulnerability was discovered using a modified version of axfuzz along with a PHP web page that attempts to sequentially instantiate COM objects that are specified in an input list. Any controls that caused a crash were later investigated with a debugger to determine if the crash appeared to be exploitable.

Several characteristics about ActiveX served as motivation to spend the effort to develop a fuzz testing tool. First is ubiquity. Microsoft Windows is the most popular desktop operating system, and Internet Explorer is the default web browser in Windows. The default configuration of Internet Explorer has ActiveX enabled. Second is uniformity. ActiveX provides a standard interface, which simplifies the ability to fuzz test whatever code may lie behind that interface. The final characteristic is coverage. Microsoft is not the only vendor that uses ActiveX. By writing an ActiveX fuzz testing tool, we are able to test code that is written by multiple vendors in any language from any part of the world. For these reasons, we developed the *"Dranzer"* ActiveX fuzz tester.

3 ActiveX Attack Surfaces

ActiveX controls can have several types of flaws that may be leveraged by attackers. We refer to these types of flaws as "attack surfaces."

3.1 Crash on Instantiation

As explained in the Bugtraq post by Hird, some COM objects cause Internet Explorer to crash when it checks for the IObjectSafety interface of the control. Such controls were likely never designed to be used in the Internet Explorer web browser. Simply opening a web page that refers to one such control can cause the browser to crash. Depending on the nature of the crash, it may be exploitable to run arbitrary code through use of heap-spraying techniques like Internet Exploiter. A crash that results from a null pointer dereference is less likely to be

exploitable than one that references a memory location that is reachable via heap spraying.

3.2 Input Validation

An ActiveX control may be exploitable if it fails to properly validate input. For example, if an ActiveX control accepts a string from an untrusted source, it may be vulnerable to a buffer overflow if it does not properly validate that string length.

Methods and Properties One of the characteristics of an ActiveX control is whether it is "Safe for Scripting" or not. A control can indicate that it is Safe for Scripting either through the IObjectSafety interface or the appropriate registry key [20]. When an ActiveX control asserts that it is Safe for Scripting, it means that Internet Explorer can call its methods and get or set its properties through use of scripting, such as JavaScript or VBScript. If the control fails to properly validate input to its methods, it may cause a crash as the result of receiving malformed data to its methods.

Initialization Parameters Just as a control may assert Safe for Scripting, it may also indicate that it is "Safe for Initialization" with persistent data. Rather than passing malformed data to a control's methods, it may be possible to pass malformed data as a parameter to the control when it is initialized.

3.3 Unsafe Methods

It is up to an ActiveX control's developer to mark the control as Safe for Scripting if it has methods that are designed to be used in Internet Explorer. Microsoft provides guidelines [21] for how to determine if a control should be marked as safe or not, however in many cases a control is incorrectly marked Safe for Scripting when it contains methods than can be abused by attackers.

4 Test Methodology

4.1 Crash on Instantiation

For COM objects that cause Internet Explorer to crash when a web page references the control, Dranzer generates HTML test case files and opens them with Internet Explorer. Dranzer sets the Alternate Data Stream Zone.Identifier to have Internet Explorer treat the test case as if it were loaded from the Internet zone. The reason for this is that the Local Machine zone has different security properties than the Internet zone, and consequently ActiveX controls may behave differently based on their zone. When Internet Explorer is launched to open the test case, a debugger is attached to the web browser. Once loaded, the page is reloaded, the ActiveX control is clicked, and then the browser is closed. If a crash occurs at any point, Dranzer logs the details for the COM object and the crash.

4.2 Input Validation

Methods and Properties Dranzer tests an ActiveX control's methods without using the Internet Explorer web browser or Windows Scripting Host. The ActiveX control is created in the process space of the testing tool, which dramatically increases test performance. Dranzer will emulate the routines that Internet Explorer uses to determine if a control is Safe for Scripting or not, and whether the control has a kill bit [22] set. For controls that can be scripted by Internet Explorer [23], Dranzer gets and sets its methods, and calls its methods using malformed data. For example, if a method accepts a string parameter, Dranzer tests the method by calling it with a 10k long string of lowercase 'x' characters. This test can find buffer overflow vulnerabilities. Integer parameters receive a value of '-1' to check for integer overflow vulnerabilities. Dranzer establishes an exception handler to capture access violation details and continue when possible. When an access violation is encountered by calling a method or property, Dranzer records a complete backtrace for the ActiveX control test. This is done because in many cases, the last method called when the access violation occurs is not the one that actually contained the fault. Consider the following sequence:

```
Method1(BSTR param1)
Method2()
```

It is possible that Dranzer reports an access violation in Method2, but in actuality it is likely that Method1 is where the overflow occurred, and calling Method2 triggered the access violation caused by the overflow. We refer to this type of flaw as a "second-order" vulnerability [24].

Initialization Parameters Unlike the testing of methods and properties, Dranzer uses Internet Explorer to test initialization parameters. This is done so that the passing of parameters from the HTML page to the ActiveX control is reproduced exactly as it occurs in the web browser. Additionally unlike the testing process for methods and properties, there is not a well-defined routine for obtaining the initialization parameters that an ActiveX control may accept. Dranzer employs two techniques for determining initialization parameter candidates. One technique uses the IPropertyBag interface of the ActiveX control. The other technique searches for text in sections of the COM server binary. The combination of these two techniques is effective in finding initialization parameters in an automated fashion. Once Dranzer compiles a list of initialization parameter candidates, it generates an HTML test case file and loads it into Internet Explorer, similar to the crash on instantiation test. The OBJECT section of the test case has a PARAM element that contains a malformed value. For example, a 10k string of lowercase 'x' characters is used for string parameters. If Internet Explorer crashes when opening the test case file, the file is copied to the Dranzer directory and the crash details are logged. The test case can be used to reproduce the crash. Further investigation is required to determine which of the initialization parameters caused the crash.

4.3 Unsafe Methods

Dranzer is unable to programmatically determine if an ActiveX control contains unsafe methods. This is because the name of a method may not accurately represent what the method actually does. For example, DownloadAndExecute(BSTR url) may be perfectly safe and S(BSTR p1) may be dangerous. Also, some controls may contain dangerous methods but are tied to specific domains. This can be accomplished through the SiteLock template, for example [25]. An ActiveX control that is locked to a specific domain generally cannot be abused by attackers unless the host name lookup methodology of a client system is subverted [26]. For these reasons, Dranzer can only report the methods that are exposed by a control that is marked as Safe for Scripting. Manual testing is required to determine if a control contains unsafe methods that could be used by an attacker. We developed a set of web pages called the ActiveX Workspace for this purpose. Through use of the ActiveX Workspace, a control's properties can be retrieved (using get) or set, and methods can be called, all while examining the behavior of the system. For example, Process Monitor [27] can be used to monitor registry, file, and thread details, and Wireshark [28] can be used to monitor network traffic.

4.4 General ActiveX Fuzzing Design

Dranzer was designed to minimize or eliminate any required human interaction. This allows a large number of controls to be tested effectively with a minimal amount of effort.

Button Clicker Many ActiveX controls present new windows or dialogs when the control is initialized or certain methods are called. If these windows are not dismissed, the testing process can hang. Dranzer installs a global hook during the testing process to address this problem. Dranzer monitors all open windows and when a new window is activated, Dranzer determines if a window contains any buttons by looking for the class name "button" within the child windows. If the child window contains a button, the display name is obtained by using the GetWindowText() API. If a button contains a display name such as "OK," "No," or "Cancel," Dranzer clicks the button based on its priority as determined by its order in the "act upon" list. Dranzer will also close new windows, regardless of whether they have buttons or not. This design feature allows Dranzer to test most controls without reporting a hang due to a window that is opened.

Master / Slave Architecture The Dranzer ActiveX testing tool is implemented with a master/slave architecture. The Dranzer process is the master, which spawns the TestAndReport process. TestAndReport is the process that hosts the ActiveX control and performs the fuzz testing. In certain cases, an ActiveX control may cause the process that hosts the control to hang. If the TestAndReport process hangs, the master Dranzer process detects this, logs the

hang, and proceeds to test the next object. Some ActiveX flaws may cause the slave test process to experience an unhandled exception. For example, a stack buffer overflow in an ActiveX control may overwrite the Structured Exception Handler (SEH) [29] of the process that is hosting the control. In such a case, the test process will terminate without being able to capture the details of the access violation. In the cases of both hangs and unhandled exceptions, the master/slave architecture of Dranzer allows the tests to continue where other testing tools may fail.

Process Monitor Certain ActiveX controls may cause other processes to be spawned when the control is initialized or tested. When testing large numbers of controls, these extra processes may slow the system down and may interfere with test results. When an object is tested, Dranzer will take a snapshot of all processes running on the system. After the test completes, any extra processes are terminated. In tests that require Internet Explorer, Dranzer checks if Internet Explorer is running before performing the test. If it is running, Dranzer will terminate the process before proceeding. In our testing, several ActiveX controls behave differently if another Internet Explorer window is open. By eliminating this extra variable, we are able to get more accurate results.

Command-Line Driven While perhaps daunting to the new user, Dranzer was designed to be a command-line tool. Command-line tools are easily scriptable, while GUI tools can be difficult to automate.

ActiveX Test Target Selection Dranzer was designed to be flexible in the selection of ActiveX test targets. For example, one may wish to target a specific control. In such a case, the "include list" feature of Dranzer would be used. In this list, the user includes the CLSID [30], or unique identifier, for the ActiveX control. When run with the appropriate command line flags, Dranzer will test only those ActiveX controls that are specified.

Another useful way of running Dranzer is with the "exclude list" feature. Consider the case where it is not known which ActiveX controls an application may install. One can run Dranzer to generate a baseline of ActiveX controls installed at that time, then install the application, and then run Dranzer again, excluding those ActiveX controls in the baseline. For example, Ahead Nero 8 [31], which is a popular CD and DVD burning program for Windows, installs 1053 COM objects in our testing. Without the ability to use exclude lists, it would not be practical to test each of the controls that an application installs.

5 Dranzer Test Results

From a fuzz-testing perspective, only certain types of COM objects are of interest. We are concerned primarily in ActiveX controls that are exploitable using Internet Explorer as an attack vector. Excepting COM objects that cause a

crash upon instantiation, the controls that are either marked Safe for Scripting (SFS) or Safe for Initialization (SFI) are the primary targets because they can be forced to process data from an untrusted source. If an ActiveX control contains a buffer overflow in a method but is not marked SFS, Internet Explorer cannot be used to call that method. If one must run a specially-written application to exploit that control, that doesn't give the attacker any advantage, as the victim is already running an arbitrary program at that point.

In our testing, we found that a Windows XP SP2 system contains 2701 COM objects by default. Out of these controls, 375, or 13.88 percent, are marked as either SFS or SFI. The percentage of ActiveX controls that are either SFS or SFI varies depending on the type of software used. For example, out of the 1053 COM objects installed by Ahead Nero 8, only four are either SFS or SFI. While testing stand-alone applications can be useful in checking their security, the return on investment (ROI) is not ideal. Each application must be selected manually, installed manually, and then tested. And out of the installed COM objects, a relatively low percentage of those will be marked SFS or SFI.

Downloaded ActiveX controls have multiple advantages over those installed by stand-alone applications. If a web page wants to use an ActiveX control, it can specify the CODEBASE or download location for that control [32]. Because downloaded ActiveX controls are generally all designed to be used in a web browser, a higher percentage of them are typically marked SFS or SFI. Windows keeps track of all ActiveX controls that it has downloaded and the associated download location for the control in the Windows registry. The final advantage of downloaded ActiveX controls comes in the method that can be used to determine test targets. A popular Windows program that is used to help browser users troubleshoot their web browsing experience is HijackThis [33]. This application takes an inventory of the various aspects of a Windows system that can affect web browsing. These reports are commonly posted on forums where analysts indicate which items need to be removed because they are spyware, are malicious, or contain other unnecessary components. One of the sections of a HijackThis report is a list of downloaded ActiveX controls, along with the location from which the control was downloaded.

By scraping HijackThis reports from various web forums, we have been able to compile a list of downloadable ActiveX controls, while archiving the controls at the same time. Not only does this give us a wide variety of software from different vendors in different countries to test, many downloadable ActiveX controls can be installed in an automated fashion. For the test results section of this paper, we focus on a set of 5956 ActiveX controls that were discovered in HijackThis reports and downloaded from the Internet. Because of the unknown nature of the controls being installed, the test environment consisted of a virtual machine that was on a network with no internet connectivity. For this reason, ActiveX controls that functioned as "downloader" controls to obtain additional software were not able to successfully download their payload.

Table 1 contains a breakdown of the properties of the downloaded ActiveX controls. Over half of ActiveX controls installed from the web are either SFS or

SFI. If a control is either SFS or SFI, it is most likely that it is both SFS and SFI. In some cases, the control is marked as SFS-only, and in a small number of cases, the control is marked as SFI-only. Figure 1 is a Venn diagram that demonstrates this distribution.

Table 1. Downloaded control safety distribution

Type	Count	Percent
SFS Only	971	16.30%
SFI Only	29	0.49%
SFS and SFI	2097	35.21%
SFS or SFI	3097	52.00%
Not Safe	2859	48.00%
Total	5956	100.00%

Fig. 1. Downloaded control safety distribution diagram

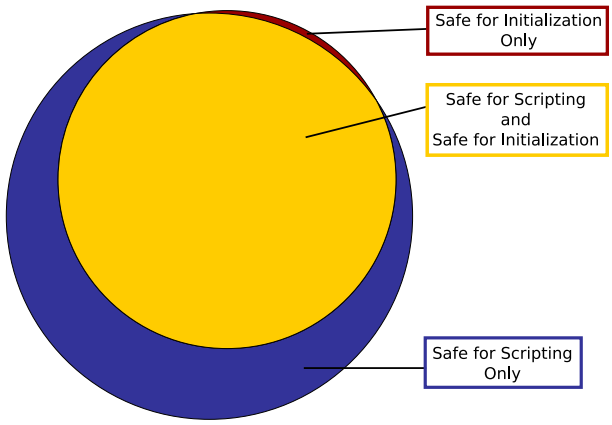

Table 2 shows the test results for those downloaded ActiveX controls that are either SFS or SFI. All Dranzer tests were performed on the controls that were either SFS or SFI

Approximately 23 percent of downloaded ActiveX controls that are either SFS or SFI fail at least one Dranzer test, resulting in a crash. Certain crashes are more likely to be exploitable than others. Table 3 shows which crashes look to be exploitable based on the Dranzer reports. For our report, an *interesting* crash is one that has one of three characteristics: First, the access violation occurs

Table 2. Safe for Scripting or Safe for Initialization control test results

Result	Count	Percent
Pass	1779	57.44%
Hang	613	19.79%
Crash	705	22.76%
Total	3097	100.00%

when attempting to execute code at a certain address. If this address is reachable with heap spraying, then it is usually trivial to create an exploit web page for the vulnerability. Second, the control caused an unhandled exception. In these cases, the Structured Exception Handler (SEH) is often overwritten as the result of a stack buffer overflow, which are also usually exploitable. Third, the access violation occurred at an address under control of the attacker. For example, a buffer overflow with a string of lowercase 'x' characters that results in an access violation trying to read memory location `0x78787878`. In our testing, all three of these types of crashes have a high chance of exploitability. Table 3 also has a breakdown of the *interesting* crashes. Note that the percentages for the three types of interesting crashes adds up to more than 100 percent. This because a single control may fail multiple Dranzer tests with different crash characteristics.

Table 3. Crash characteristics for Safe for Scripting or Safe for Initialization objects that failed Dranzer tests

Crash Type	Count	Percent
Less interesting	421	59.72%
Interesting	284	40.28%
Total	705	100.00%

Interesting crashes	Count	Percent
Access violation executing	63	22.18%
Unhandled exception	173	60.92%
Attacker-controlled address	105	36.97%
Total	284	100.00%

6 Related Work

CERT Dranzer was developed because of limitations with the other publicly-available ActiveX fuzz testing tools. Below is a brief description of the other ActiveX tools. Table 4 compares the tools using a known vulnerable ActiveX control, the AOL YGP Pic Downloader Plugin [34].

6.1 Axfuzz

Axfuzz [17] provided the initial model for creating the Dranzer tool. Evaluating the software on an entire system requires a combination of axenum (used to

enumerate the COM objects on a system) and axfuzz tools. In cases where the testing tools crash, the tools must be restarted manually at a point after the crash. Crash details are not included in the reports.

6.2 COMRaider

COMRaider [35] is a graphical tool for fuzz testing a single COM object. Crash details are included, which can aid in the determination of which COM flaws may be exploitable. Due to the program design, a high level of user interaction is required, and brute force testing of multiple COM objects is not easy.

6.3 AxMan

AxMan [36] is a web-based ActiveX fuzzing tool. Because it is web-based, it requires a web server and a high amount of user interaction to work. An advantage AxMan holds over a tool like COMRaider is that it can batch process multiple COM objects at a time. A debugger must be attached manually to Internet Explorer to retrieve crash results. Once a crash is encountered, the test process must be restarted manually after the crash occurs.

6.4 COMbust

COMbust [37] is a COM auditing tool presented at a BlackHat Briefing in July 2003, by Frederic Bret-Mounet of @stake (now Symantec). COMbust is a command-line tool that includes multiple test cases for each parameter. However, because the testing occurs in a single process, it is difficult to determine vulnerable methods beyond the first one.

Table 4. Comparison of ActiveX fuzz testing tools.

	Dranzer	**axfuzz**	**COMRaider**	**AxMan**	**COMbust**
Test time	**1 sec**	4 sec	140 sec	660 sec	2 sec
Exceptions found	**3**	2	3	1	1
Test instantiation crash	**Yes**	No	No	No	No
Test methods (SFS)	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Test initialization (SFI)	**Yes**	No	No	No	No
Test unsafe methods	**Yes**	No	No	No	**Yes**
Output	Text	Text	**Database**	None	Text
User interaction required	**None**	Medium	High	High	Medium
Test multiple objects	**Yes**	**Yes**	**Yes**	**Yes**	No
Test method sequences	**Yes**	**Yes**	No	**Yes**	**Yes**
Crash details reported	**Yes**	No	**Yes**	No	No

7 Conclusion

We have learned through testing thousands of ActiveX controls that using Internet Explorer with ActiveX enabled is a high-risk activity because Internet Explorer can be coerced to install malicious software or provide an intruder with a way to compromise your system. Automation is essential to be able to test a large amount of software. This was achieved through automation internal to the design of Dranzer itself and also through the automation of ActiveX control discovery, ActiveX control archiving, and ActiveX control installation on a test machine. For safe web browsing, the CERT/CC suggests disabling ActiveX for the Internet zone [38]. However because the default configuration for Internet Explorer is to enable ActiveX in the Internet zone, end-users are being put at risk. The tests that Dranzer currently performs are finding only the simplest buffer and integer overflows, and yet it has been effective in finding vulnerabilities.

8 Future Work

Dranzer is well-architected, as demonstrated by its ability to test a large volume of ActiveX controls with minimal user interaction. However, there are several areas where Dranzer can be improved, for instance by adding more test cases. Rather than just a simple 10k string of lowercase 'x' characters or a -1 integer, Dranzer could test for format string vulnerabilities, more integer edge cases, or special strings, such as those beginning with "http://." Another improvement we have contemplated is to test methods and parameters in different orders. Dranzer currently uses one order with properties first and then methods after that. Dranzer could find more vulnerabilities by simply randomizing the test order, or perhaps by using a more intelligent algorithm that tests methods with the most number of parameters first and the least number of parameters last. Dranzer could also combine initialization parameter tests with method and property tests for those controls that are both SFS and SFI.

Dranzer is already an effective tool, but it should prove to be even more useful with these improvements.

References

1. RFC 1135 The Helminthiasis of the Internet (1989), `http://tools.ietf.org/html/rfc1135`
2. CERT® Incident Note IN-2001-03, Exploitation of BIND Vulnerabilities (2001), `http://www.cert.org/incident_notes/IN-2001-03.html`
3. JPEG COM Marker Processing Vulnerability in Netscape Browsers (and Microsoft Products) (2000), `http://www.openwall.com/advisories/OW-002-netscape-jpeg/`
4. CORE Security, Client-side Exploits, `http://www.coresecurity.com/?module=ContentMod&action=item&id=519`

5. The Component Object Model: A Technical Overview, http://msdn2.microsoft.com/en-us/library/ms809980.aspx
6. INFO: Difference Between OLE Controls and ActiveX Controls, http://support.microsoft.com/kb/159621
7. Security Vulnerability Research & Defense : The Kill-Bit FAQ: Part 2 of 3 http://blogs.technet.com/swi/archive/2008/02/07/The-Kill_2D00_Bit-FAQ_3A00_-Post-2-of-3.aspx
8. Java Security Architecture: The Original Sandbox Model, http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc1.html#18313
9. Howard, M., Pincus, J., and Wing, J.: Measuring Relative Attack Surfaces (2003), http://www.cs.cmu.edu/%7Ewing/publications/Howard-Wing03.pdf
10. Sawyer, J.: Tech Insight: The Buzz Around Fuzzing, Fuzzing tools can help identify vulnerabilities before the bad guys do http://www.darkreading.com/document.asp?doc_id=144773
11. CERT/CC: Results of the Security in ActiveX Workshop., http://www.cert.org/reports/activeX_report.pdf
12. Microsoft Security Program: Microsoft Security Bulletin (MS99-032) Patch Available for "scriptlet.typelib/Eyedog" Vulnerability, http://www.microsoft.com/technet/security/Bulletin/MS99-032.mspx
13. MSIE <IFRAME> and <FRAME> tag NAME property bufferoverflow PoC exploit (was: python does mangleme (with IE bugs!)), http://seclists.org/fulldisclosure/2004/Nov/0053.html
14. Vulnerability Note VU#842160 Microsoft Internet Explorer vulnerable to buffer overflow via FRAME and IFRAME elements, http://www.kb.cert.org/vuls/id/842160
15. IObjectSafety and Internet Explorer, http://www.securityfocus.com/archive/1/391803
16. Vulnerability Note VU#959049 Multiple COM objects cause memory corruption in Microsoft Internet Explorer, http://www.kb.cert.org/vuls/id/959049
17. SourceForge.net: axfuzz, http://sourceforge.net/projects/axfuzz/
18. MoAxB - Month of ActiveX Bug, http://moaxb.blogspot.com/
19. Vulnerability Note VU#898241 Microsoft BlnMgr Proxy (blnmgrps.dll) COM object fails to implement required methods, http://www.kb.cert.org/vuls/id/898241
20. Safe Initialization and Scripting for ActiveX Controls, http://msdn2.microsoft.com/en-us/library/aa751977.aspx
21. Designing Secure ActiveX Controls, http://msdn2.microsoft.com/en-us/library/aa752035.aspx
22. How to stop an ActiveX control from running in Internet Explorer, http://support.microsoft.com/kb/240797
23. Security Vulnerability Research & Defense : Not safe = not dangerous? How to tell if ActiveX vulnerabilities are exploitable in Internet Explorer, http://blogs.technet.com/swi/archive/2008/02/03/activex-controls.aspx
24. Ollmann, G.: Second-order Code Injection Attacks, HowtostopanActiveXcontrolfromrunninginInternetExplorer
25. SiteLock Template 1.04 for ActiveX Controls, http://msdn.microsoft.com/archive/en-us/samples/internet/components/sitelock/default.asp?frame=true
26. Vulnerability Note VU#400601 Symantec Automated Support Assistant ActiveX control buffer overflow, http://www.kb.cert.org/vuls/id/400601

27. Process Monitor v1.26 By Mark Russinovich and Bryce Cogswell, http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx
28. Wireshark: Go deep., http://www.wireshark.org/
29. Structured Exception Handling, http://msdn2.microsoft.com/en-us/library/ms680657(VS.85).aspx
30. CLSID Key, http://msdn2.microsoft.com/en-us/library/aa908849.aspx
31. Nero - Nero - Nero 8, http://www.nero.com/enu/nero8-introduction.html
32. CODEBASE Attribute — codeBase Property, http://msdn2.microsoft.com/en-us/library/ms533576(VS.85).aspx
33. TrendSecure — TrendMicro HijackThis Overview , http://www.trendsecure.com/portal/en-US/tools/security_tools/hijackthis
34. Vulnerability Note VU#661524 AOL YGP Pic Downloader Plugin ActiveX control buffer overflow, http://www.kb.cert.org/vuls/id/661524
35. Fuzzing Software Tools // iDefense Labs, http://labs.idefense.com/software/fuzzing.php#more_comraider
36. AxMan ActiveX Fuzzer, http://www.metasploit.com/users/hdm/tools/axman/
37. COMbust, http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-bretmounet-combust.zip
38. Dormann, W., Rafail, J.: Securing Your Web Browser, http://www.cert.org/tech_tips/securing_browser/