버퍼 오버플로우 취약점 증명을 위한 프레임워크

권오훈*, 민병길*, 김 종*, 김수용**, 한광택**

포항공과대학교 컴퓨터공학과*, 국가보안기술연구소**

A Framework for Illustration of Buffer Overflow Vulnerability

O-Hoon Kwon*, Byung-Gil Min*, Jong Kim*, Su Yong Kim**, Kwang-Taek Han**

*Dept. of Computer Science & Engineering,
Pohang Univ. of Science and Technology

**National Security Research Institute

요 약

특정 프로그램의 취약점 여부를 판단하기 위해 프로그램의 버전이나 작동여부를 점검하여 검증하는 방식이 많이 사용되었다. 하지만, 이 방식은 취약점 증명에 영향을 미치는 많은 요소들을 모두 고려할 수 없기 때문에 점검 결과가 부정확하다는 단점이 있다. 최근에는 이를 보완하기 위해 해당 프로그램에 실제로 공격을 수행하고 그 성공 여부를 통해 취약점 여부를 검증하는 방식이 사용되고 있다. 이렇게 실제 공격코드를 이용하는 방법은 정확한 결과를 얻을 수 있다는 장점이 있지만, 공격코드를 구현하기 어렵다는 단점이 있다. 본 논문에서는 공격코드들의 구조를 분석하여 공통된 부분들을 모듈화하고 이를 재사용함으로써 다양한 취약점을 정확하게 점검할 수 있고, 새로운 취약점에 대해서도 쉽게 확장할 수 있는 취약점 증명을 위한 프레임워크를 제시한다.

I. 서론

정보 시스템이 폭 넓게 사용되고 그 중요성 또한 커지면서, 시스템 관리자들은 자신들의 정보 시스템에 대한 보안 취약점을 미리 점검하여 이에 대한 적절한 대응책을 마련하고 있다.

특정 프로그램의 취약점 여부를 판단하기 위한 방법은 주로 두 가지로 - 해당 프로그램의 버전이나 작동여부를 점검하는 방식과 실제로 공격을 수행하여 성공 여부를 검증하는 방식 - 구분할 수 있다. 기존에는 주로 전자에 의해 취약점 여부를 판단하기 위한 취약점 점검 도구들이 많았다. 이 는 구현이 쉽고, 시스템에 특별한 해를 끼치지 않으며, 점검 속도도 매우 빠르다는 장점 때문이다. 하지만, 취약점 증명에 영향을 미치는 다양한 요소들을 고려할 수 없기 때문에 점검 결과가 부정확하다는 단점이 있다. 후자의 방법은 정확한 결과를 얻을 수 있고, 취약점이 시스템에 끼치는 영향을 보여줄 수 있다는 장점이 있지만, 구현이 난해하며, 시스템에 다소 해를 끼칠 수 있다는 단점이 있다. 따라서, 두 가지 방식은 하나가 절대 우위에 있다기보다 서로 상호 보완적인 관계에 있다[1].

특히 90년대 이후 폭 넓게 이뤄지고 있으며 취약점을 지닌 프로그램에 대한 많은 보고가 지속적으로 보고 되고 있는, 버퍼 오버플로우 공격은 단순한 프로그램의 버전, 작동여부를 점검하는 방식으로는 적절한 취약점을 증명할 수가 없다. 때문에 실제로 공격을 수행해 볼 수 있는 공격코드들을 사용하여 취약점을 검증하고 있다. 하지만 다수의 취약점에 대해서 각각 공격코드들을 작성하여 사용하는 것은 차후 공격코드를 이해하기 어렵고, 개별적으로 작성된 이들 코드들을 통합하여 하나로 사용하기 어렵기 때문에 효율성이 떨어진다. 또, 새로운 취약점에 대해서 쉽게 그 기능을 확장할 수 없는 문제점을 지니고 있다.

이 논문에서는 취약점 증명을 위한 버퍼 오버플로우 공격코드의 구조를 분석하고, 다양한 취약점을 점검할 수 있도록 코드의 재사용성을 높이고, 새로운 취약점 점검을 위해 쉽게 확장될 수 있는 프레임 워크를 제시하고 있다.

본문의 1절에서는 버퍼 오버플로우 공격의 개념과 그 원리에 대해서 기술하고 있으며, 2절에서는 이를 바탕으로 공격코드들의 구조를 분석하고 있다. 3절에서는 다양한 취약점 점검을 지원하고

※ 본 연구는 한국전자통신연구원 부설 국가보안기술연구소 '취약점 증명을 위한 프레임 워크'의 연구 결과로 국가보안기술연구소의 지원을 받음.

새로운 취약점에 대해서 공격코드를 쉽게 확장할 수 있는 구조를 제시하고 있다.

II. 본문

1.버퍼 오버플로우 공격의 원리

버퍼 오버플로우 공격이란 말 그대로 프로그램 내의 버퍼가 그 보다 큰 데이터를 받게 되어 넘치는 취약점을 이용하여, 버퍼를 오버플로우 시킴으로써 시스템의 관리자 권한을 얻거나 원하는 작업을 수행시키는 공격 방법이다. 이는 버퍼를 프로그램이나 함수의 리턴주소가 위치한 부분까지 넘치게 하여, 주소를 조작함으로서 프로그램의 흐름을 바꾸어 원하는 작업을 수행하도록 하기 때문에 가능하다[2].

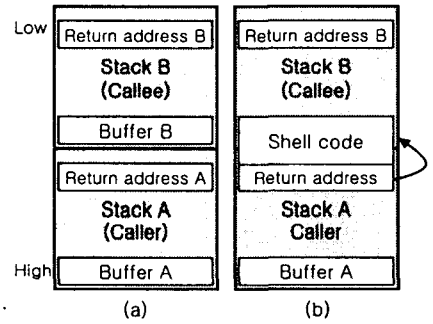

그림 1 : 스팍 솔라리스에서의 버퍼 오버플로우

서버 시스템으로 많이 이용되는 스팍 솔라리스(SPARC Solaris)의 경우는 리눅스와 달리 스택 영역에서 리턴주소가 지역변수보다 앞에 위치하게 된다. 따라서 자신의 스택영역에 기록된 리턴주소를 조작할 수가 없다. 그림 1의 (a)는 스팍 솔라리스에서의 스택의 구조를 보여주고 있다.

스택 A는 스택 B를 호출한 함수로 B보다 높은 주소(아래)에 위치하게 되며, 스택 B의 버퍼 B를 오버 플로우 시켜서 스택 A의 리턴주소 A를 조작할 수 있게 된다. 그림 1의 (b)는 이렇게 버퍼가 오버플로우된 메모리의 상태를 보여주고 있다. 따라서, 임의로 작성된 실행코드(shellcode)가 실행되는 것은 B를 호출한 함수 A가 종료되는 시점에서 리턴주소 A를 읽을 때가 된다. 이런 구조 때문에 리눅스보다 공격을 수행하기가 더 어려운 특성을 가지고 있다.

본 논문에서는 주요 기관과 기업에서 서버로 사용되어 보안 점검의 중요성이 큰, 스팍 솔라리스에서의 버퍼 오버플로우에 대한 연구를 바탕으로 하여 기술되었다.

2. 버퍼 오버플로우 공격코드의 구조

본 연구를 통하여 분석한 다양한 공격코드들의 구조는 표1과 같이 다양한 형태를 가졌다. 가장 간단한 경우의 내부공격은 쉘코드와 버퍼를 조작하는 부분만으로 이루어져 있고, 솔라리스 tclnetd에 대한 공격코드의 경우는 쉘코드, 원격접속, 버퍼조작, 루트쉘 접속, 자동화된 반복문으로 이루어져 있다. 솔라리스의 cachefsd에 대한 공격코드는 공격의 성공여부를 검증하기 위한 부분이 있었으며, snmpXdmid의 공격코드는 반복문을 사용하는 대신 매우 큰 NOP를 사용하여 공격을 수행하고 있다.

	Local	Telnetd	Cachefsd	SnmpXdmid
Shellcode	O	O	O	O
Connection		O	O	O
Buffer Fabrication	O	O	O	O
Verification			O	O
Shell Connection		O	O	O
Brute Force		O	O	

표 1 : 주요 공격코드의 구조

이외에도 여러 버퍼 중에서 취약점을 가지는 버퍼로 조작된 버퍼를 넘겨주기 위해서 전송 시점을 결정하기 위한 부분이 있으며, 원격 공격의 경우는 서비스 요청을 처리하는 과정에서 버퍼 오버플로우가 일어나는 cachefsd와 같이 전송 시점을 결정하기 위한 부분이 별도로 구성되어 있지 않을 수도 있다.

앞서의 여러 공격코드들은 동일한 대상 시스템에 대해서도 다양한 쉘코드들을 사용하고 있으며, 버퍼의 조작, 원격접속 등의 부분들이 서로 종속적으로 코드가 작성되어 있다. 또 동일한 기능을 수행하는 부분들이 서로 다른 방식과 변수, 인터페이스를 가지고 작성되어 있었다. 따라서 다양한 취약점에 대한 검증이 가능하도록 하나로 통합하기가 어려웠으며, 새로운 취약점에 대해서 기존의 코드를 확장하여 사용할 수가 없는 단점을 지니고 있다.

3. 재사용과 확장성을 고려한 프레임워크

1) 정형화된 범용적 공격코드의 구조

앞서의 공격원리의 분석과 기존의 공격코드들의 분석을 통하여 범용적으로 적용이 가능한 정형화된 구조를 제시할 수 있었다. 버퍼 오버플로우의 다양한 경우를 고려한 정형화된 구조는 그림 2와 같다.

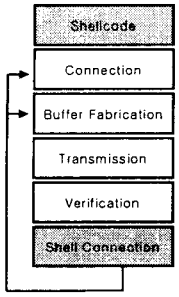

그림 2 : 정형화된 공격코드의 구조

쉘코드(Shellcode)부분은 공격이 성공하였을 경우에 실제로 실행되는 이진코드를 기술한 것이다. 쉘코드는 동일한 공격대상 시스템에 대해서는 완전하게 재사용이 가능하다. 즉 리눅스용으로 작성된 하나의 쉘코드는 리눅스를 대상으로 한 모든 버퍼 오버플로어 공격에서 이용되어 질 수 있다. 때문에 가장 크게 재사용과 단일화가 가능한 부분이다.

접속(Connection)을 수행하는 부분은 크게 소켓 프로그래밍을 이용한 방식과 원격 접속 절차(RPC)를 이용한 방식이 있으며, 각각의 데몬에 따라서 서로 다른 데이터 구조를 사용하고 있다. 접속을 수행하는 부분이 다른 부분에 대해서 독립적으로 수행되도록 보장하고 다른 모듈과의 인터페이스를 표준화함으로써 재사용과 확장성을 지원할 수 있다.

버퍼조작(Buffer Fabrication)부분은 취약점을 지닌 버퍼를 오버플로우 시킬 수 있는 적절한 크기의 버퍼를 구성하고, 여기에 쉘코드를 삽입하고 조작된 리턴주소를 기입하는 과정이다. 버퍼를 조작하는 방법은 동일하고 그 크기나 주소값들이 다르기 때문에 다양한 크기와 주소값을 입력받아서 수행하도록 단일화가 가능한 부분이다.

전송(Transmission)부분은 공격 대상 프로그램이 여러 번의 입력을 받을 경우, 어떤 버퍼로 조작된 버퍼를 넘겨줄 것인가를 결정하여 실제적으로 버퍼 오버플로우를 일으키도록 버퍼를 전송하는 부분이다. 대부분의 경우 취약점을 지닌 프로그램의 반환값이나 프롬프트를 문자열비교하여 이루어지게 된다. 이 부분은 어떤 문자열과 비교를 수행할 것인가를 외부로부터 입력받아서 수행하게 함으로써 코드를 재사용하고 다른 공격으로 확장하여 사용하게 할 수 있다.

검증(Verification)부분은 공격의 성공여부를 검증하는 역할을 수행하게 된다. 반복적으로 공격을 시도하는 경우에 성공여부를 검증하여 실패한 경우는 계속 공격을 수행하고, 성공하였을 경우에는 반복문을 종료시키게 된다. 각각의 공격 대상에 따라서 성패를 검증하는 방법은 다양하게 구성되지만, 그 결과 값은 성공여부이므로 다른 모듈과의 인터페이스를 표준화함으로써 다양한 검증방법을 유연하게 사용하고 확장할 수 있다.

쉘접속(Shell Connection)부분은 원격 접속의 경우에 사용되는 부분으로, 공격이 성공하였을 경우 원격으로 루트쉘을 사용할 수 있도록 해주는 부분이다. 모든 공격코드에서 이 부분은 동일하기 때문에 완벽한 통합과 재사용이 가능하다.

마지막으로 외부에 화살표로 표시된 부분은 자동화된 공격을 지원하기 위한 부분으로, 조작된 버퍼의 크기나 리턴주소를 지속적으로 바꾸면서 시도하여 적절한 크기와 리턴주소에서 공격의 성공이 이뤄지도록 하는 부분이다. 반복문의 횟수와 시작주소의 위치, 증감값을 입력받게 함으로서 다양한 공격코드를 지원할 수 있도록 할 수 있다.

2) 재사용과 확장이 가능한 프레임워크

실제의 공격코드들은 앞서 제시된 정형화된 모든 구조를 지니지 않고 일부분만을 지니고 있다. 가장 간단한 내부 공격의 경우는 단순히 쉘코드와 버퍼 조작부분만이 있으며, 원격 공격의 경우는 쉘코드, 데몬 접속부분, 버퍼조작과 쉘접속 부분을 기본적으로 가지지만 전송시점이나 검증, 외부의 반복문에 의한 자동화된 공격의 경우는 선택적으로 사용하게 된다. 또한 서로 다양한 구조를 가지고 각각의 부분들이 수행하는 작업과 인터페이스가 서로 다르게 작성되어 있거나 각각의 기능들을 수행하는 부분이 분리되지 않고 사용되어 코드의 재사용이 어렵게 되어 있다. 따라서 다른 환경이나 시스템으로 쉽게 코드를 변경하여 사용하기가 어렵고 새로운 취약점에 대해서도 코드를 쉽게 확장하여 사용하지 못하고 그에 따른 코드들을 다시 작성해 주어야 한다. 앞서 살펴본 정형화된 공격코드의 구조를 이용하여 재사용성과 확장성을 고려한 프레임 워크를 그림 3과 같이 제시할 수 있다.

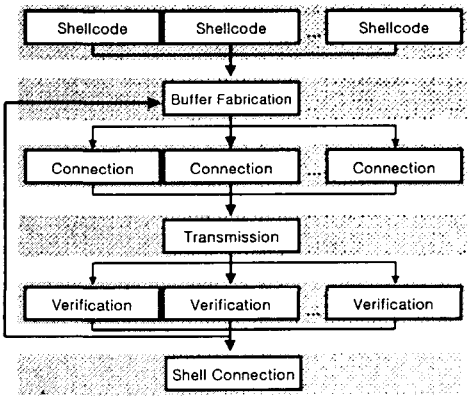

그림 3 : 재사용과 확장성을 고려한 프레임 워크

3) 프레임워크의 재사용 및 확장성 검증

이러한 프레임 워크를 따른 공격코드들이 정말로 쉽게 재사용되고 확장될 수 있는지를 검증하

기 위하여, 가상의 데몬을 작성하여 다양한 변화를 주면서 공격코드들이 어떻게 재사용되고 확장되어질 수 있는지를 실험하였다[3].

가장 간단한 경우로 내부에서 데몬을 사용할 수 있는 경우에서 가상 데몬을 공격할 수 있는 간단한 내부공격 코드는 제시된 프레임 워크에 맞추어 쉘코드와 버퍼조작 부분만으로 이루어졌다. 데몬이 원격에서만 공격이 가능한 경우로 변화를 주었을 때, 앞서 작성된 공격코드에 접속모듈과 전송모듈만을 추가하여 줌으로써 쉽게 새로운 공격코드로 확장할 수 있었다. 데몬의 버퍼크기와 위치에 변화를 주었을 경우에는, 공격코드는 반복문을 돌면서 버퍼의 크기와 주소값을 변화시키면서 시도하도록 변경이 되었는데, 이 경우에 있어도 버퍼조작 모듈에 크기, 주소값을 변화시키면서 넣이 줄 수 있도록 외부의 반복문만을 추가함으로써 쉽게 공격코드의 확장이 가능하였다. 또한, 데몬의 외부 접속방법을 소켓에서 원격호출절차(RPC)로 변경한 경우에는 접속부분과 전송부분만을 변경함으로써 공격코드를 쉽게 확장할 수 있었다. 덧붙여서 telnetd처럼 데몬이 여러 번의 입력을 받아들이고 그 중 일부 버퍼에 취약점이 있는 경우에도 버퍼 전송 시점을 결정하는 모듈을 이용하여 쉽게 확장할 수 있었다. 표2는 제시된 프레임 워크의 타당성을 검증하기 위하여 수행되었던, 공격에 영향을 미치는 각 요소들의 조

접속방법	취약점		자동화
Socket	Remote1[1]	Remote2[2]	Brute Force
RPC	Remote1	Remote2	Brute Force

표 2 : 프레임 워크 검증에 사용된 조합들

합을 정리한 것이다.

이와 같이 쉽게 공격코드를 재사용하고 확장이 가능했던 것은, 각각의 모듈들이 다른 모듈들의 코드에 종속되지 않고 독립적으로 자신의 기능들을 수행하면서 표준화된 인터페이스와 다양한 인수값을 지원하도록 작성되었기 때문이다. 데몬의 변화에 따라 새로운 공격방법으로 확장하는데 있어 기존의 코드를 그대로 사용하거나 인수값만을 변화시켜주면 되고, 새로운 코드를 추가하는 경우에도 다른 모듈의 수정없이 인터페이스만을 만족시켜주도록 작성함으로서 쉽게 기능 확장을 지원할 수 있다.

III. 결론

본 연구에서는 버퍼 오버플로우 공격과 실제 공격코드들을 분석하여, 다양한 취약점에 적용가능하고 새로운 취약점으로 확장이 용이한 프레임 워크를 제시하였다. 그리고, 제시한 구조에 맞춰서 공격코드를 작성하면, 하나의 코드가 다양한 취약점 검증에 사용될 수 있고, 코드의 조합과 변형이 가능하므로 쉽게 새로운 취약점을 증명하기 위한 공격코드로 확장이 가능함을 검증하였다.

1) 데몬 자체에 취약점이 있는 경우
2) 데몬에서 취약한 프로그램을 호출하는 경우

참고문헌

[1] Shostack, A., S. Blake., "Toward a Taxonomy of Network Security Assessment Techniques," Proceedings of the 1999 Black Hat Briefings, July 1999.

[2] Aleph One, "Smashing The Stack for Fun and Profit," Phrack Magazine, 49(14), 1998.

[3] PLUS(포항공대 유닉스 보안 연구회), "Security PLUS for UNIX", 영진출판사, 2000.