동적 버퍼 오버플로우 탐지기의 설계 및 구현

박주희, 허 신
한 양 대 학 교 전 자 계 산 학 과
{parkjh, shinheu}@cse.hanyang.ac.kr

The Design and Implementation of Dynamic Buffer Overflow Scanner

Ju-Hee Park, Shin Heu
Dept. of Computer Science & Engineering, Hanyang University

요 약

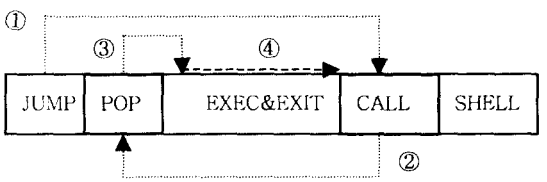
운영체제와 컴퓨터 언어상의 문제점인 버퍼 오버플로우가 보안상의 위협으로 등장하고 있다. 이를 이용한 공격은 한 호스트의 모든 제어권을 가질 수 있기 때문에 보안상의 위협 중 심각한 부류에 속한다.본 논문에서는 버퍼 오버플로우의 원리 및 관련연구를 설명하고 기존에 프로그램 각각에 따라 오버플로우 코드가 존재하던 것에 비해 일반적으로 적용 가능한 탐지 시스템을 제안한다.

1. 서론

운영체제와 컴퓨터 언어상의 문제점인 버퍼 오버플로우가 보안상의 위협으로 등장하고 있다. 1999년 CERT 권고사항(advisory)의 반 이상을 차지하는 이를 이용한 공격은 한 호스트의 모든 제어권을 가질 수 있기 때문에 보안상의 위협 중 심각한 부류에 속한다[1]. 본 논문에서는 문제 해결을 위하여 버퍼 오버플로우의 원리와 관련 연구를 서술하며 이를 바탕으로 본 논문이 제안하는 탐지기를 설명하고 구현과 결론으로 끝을 맺는다.

2. 버퍼 오버플로우

버퍼 오버플로우란 입력의 경계(boundary)를 점검하지 않는 버퍼에 할당된 버퍼의 크기 보다 큰 입력을 가하는 것이며 더 나아가 버퍼를 사용한 프로그램의 제어 흐름을 바꾸는 것을 말한다. 이는 C와 같이 배열이나 포인터 참조의 바운드를 자동으로 점검하지 않는 언어에는 본질적으로 타고난 문제점이다.[1]
1960년대 일부 해커들에 의해 사용되었던 이 방법은 1988년 fingerd의 버퍼 오버플로우 취약점 (vulnerability)을 이용한 Internet Worm 사건으로부터 일반인들에게 알려졌다[2].버퍼 오버플로우는 모든 운영체제에서 발생 가능하지만 본 논문에서는 UNIX를 예로 설명한다.
오버플로우 대상 프로그램에 주입되는 코드는 최종적으로 임의의 프로그램을 실행하지만 주로 쉘을 실행하므로 쉘코드라 불린다. 쉘코드는 <그림1> 같은 일반적인 구조와 실행 순서를 가지는 머신 코드의 집합이다[3].

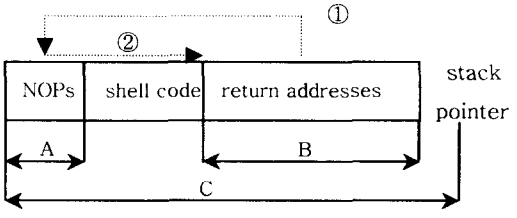


<그림1> 쉘 코드

1.프로그램의 제어권이 JUMP위치에 도달하면 CALL로 점프
2.POP위치로 호출을 수행
3.POP에서 실행시킬 프로그램의 경로와 이름이 담긴 문자열(SHELL)의 주소를 저장
4.오버플로우 성공 시에 실행될 프로그램의 코드와 실패 시의 탈출 코드

버퍼 오버플로우는 오버플로우 되는 메모리의 위치에 따라 스택 기반 오버플로우, 힙(heap) 기반 오버플로우로 나뉘며,방법에 따라 인자(argument) 오버플로우, 변수 오버플로우, 데이터 오버플로우 등이 있다[4].

2.1위치에 따른 분류
2.1.1 스택 기반 오버플로우
스택은 함수 호출 시 activation record를 저장하며 이는 호출 함수의 인자, 복귀 주소(return address),지역 변수를 포함한다.
쉘코드가 실행될 수 있도록 컨트롤을 옮겨주며 버퍼 오버플로우 시 주입되는 코드가 EGG코드이다. <그림2>는 쉘코드가 실행될 수 있도록 컨트롤을 옮겨주며 버퍼 오버플로우 시 주입되는 EGG코드의 일반적인 구조와 실행 순서를 나타낸다.



<그림2> EGG 코드

NOPs: 머신 코드 상의 no operation 코드의 집합으로 부정확한 복귀 주소를 보정(생략 가능)
shell code: 실행시킬 쉘코드
return address: NOPs영역 중 하나에 해당하는 메모리 번지의 집합

오버플로우 가능 여부는 A,B,C의 길이와 쉘코드의 기능에 의해 결정된다.

2.1.2 힙 기반 오버플로우
힙이란 어플리케이션에 의해 동적으로 할당되는 메모리 영역으로 힙 영역의 함수 포인터 등이 원하는 코드(쉘코드)를 실행하도록 오버플로우 시킨다. 주로 힙의 하위 버퍼를 오버플로우시켜 상위 힙의 동적 변수에 원하는 데이터를 덮어쓰는 기법을 사용한다[5].

2.2 방법에 따른 분류
2.2.1 인자 오버플로우
어플리케이션의 커맨드 라인 상의 입력 인자를 위하여 예약된 버퍼의 양보다 큰 임의의 코드를 주입할 때 발생한다.
2.2.2 변수 오버플로우
환경변수에 임의의 코드를 주입하여 프로그램이 해당 환경 변수를 엑세스할 때 발생한다.

2.2.3 데이터 오버플로우
프로그램에서 사용하는 데이터 자체를 오버플로우시켜 이를 이용하는 프로그램이 데이터를 처리할 때 발생하며 해당 데이터의 구조에 의존적이다.

입력의 경계를 점검하지 않는 C 언어의 함수로는 scanf, strcpy, strcat, sprintf, gets등이 있다. 그리고 비교적 안전하다고 알려진 C언어의 API도 여전히 위험요소를 지니고 있으며 많은 시스템에서 사용가능하지도 않다[1][4].
현재 알려진 사용자들에 대한 버퍼 오버플로우에 대한 대책은 취약점을 가지고 있는 프로그램에 해당 벤더에서 제공하는 패치를 가하는 것이다. 이는 취약점이 발표된 후의 보완으로써 보안상 문제점을 야기한다.

3. 관련연구

버퍼 오버플로우 탐지에 관한 연구는 정적 분석과 동적 분석으로 나눌 수 있다.
정적 분석은 탐지 대상 프로그램의 소스에 대해 검사를 수행하는 것으로 프로그램의 배포 전(proactive) 분석에 사용되어 라이브러리와 함수에 따른 오버플로우 가능성을 탐지한다. 분석 결과가 비교적 정확하다는 장점이 있으나 탐지 대상이 되는 소스 코드가 있어야 한다는 점과 취약점(false negatives)뿐만 아니라 경고성 결함(false positives)도 분석되어 결과를 실제로 응용하기 어려운 단점을 가지고 있다[1]. 이에 비해 동적 분석은 확장성과 응용가능성을 중요시하는 접근방법이다. 실행 파일만으로 분석이 가능하며 탐지기의 설계에 따라 성능이 결정되는 특징을 가진다.

4. 제안 모델

본 논문에서는 기존에 하나의 프로그램에 대한 플랫폼, 운영체제,버전별 탐지 코드(exploit)를 적용하던 것에 비해 일반적인 적용이 가능하며 실행 파일에 대한 탐지가 가능한 버퍼 오버플로우 탐지기를 제안한다. 이를 위하여 제안된 기능은 다음과 같다.

•원격 호스트의 정보 수집 - 접속시의 배너를 이용하거나 원격 호스트상의 TCP/IP스택의 특성을 이용하여 운영체제를 탐지한다[6].

•원격 데몬 탐지 - 포트 스캔을 실시하여 열린 포트에 대해 입출력을 실시하여 탐지를 실시

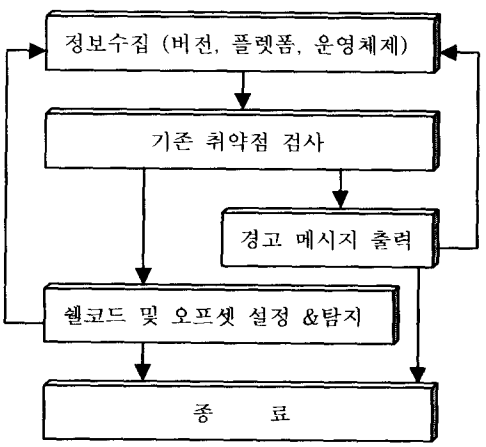
•탐지 대상 파일 설정 - 임의의 파일이나 suid가 root인 파일 탐지: setuserid 가 root인 파일들은 실행 중 임의로 컨트롤의 흐름이 바뀌면 root권한이 주어지므로 버퍼 오버플로우의 주요 목표가 된다.

•기존 취약점(vulnerability) 검사 - 취약점이 알려진 파일이면 경고 메시지를 출력한다. 이를 위하여 프로그램의 이름 및 버전,플랫폼,운영체제 각각에 대한 정보를 모듈화 하여 필요 시 조합한다.

•다양한 쉘 코드 지원 - 수집된 정보를 가지고 쉘코드를 선택하며 프로그램의 기능에 따라 다음을 포함하는 다양한 수정이 필요하다[7].
-필터링: 프로그램에서 특정 문자를 변환하거 필터링할 때 쉘코드에서 있어서는 안될 코드를 필터링한다.
-소켓 열기: 데몬을 오버플로우 시킨 후 crash 방지를 위하여 소켓을 열어 원격 탐지를 가능하게 한다.

•실행 프로그램에 대한 정보 - 라이브러리 호출 추적기를 이용하여 실행중인 프로그램을 디스어셈블하여 머신 코드에서 특정 코드(어셈블리 코드 혹은 C 함수에 대한 머신 코드)와의 패턴 매칭을 시도한다.

• 스택 포인터에 대한 쉘코드의 오프셋 결정
-garbage filling: 오버플로우 시킬 때 지정하는 값으로 탐지 시 지문(fingerprint)으로 사용된다. 출력 가능한 (printable) ASCII 문자이어야 하며 머신 코드로 해석되었을 때 탐지에 영향을 주지 않아야 한다.
예) INC EAX (0x41=' A')
-코드 주입에 대한 출력(segmentation fault 혹은 illegal command)확인: 예를 들어 core가 생성되면 디버거와 함께 실행하여 segmentation fault 발생시의 컨트롤 변화(레지스터의 값)를 조사하여 garbage의 삽입 유무에 따라 오프셋을 정할 수 있다.
위의 조건에 따른 탐지기의 동작을 <그림 3>에서 나타내었다.



<그림 3> 탐지기의 동작 방식

5. 구현

버퍼 오버플로우는 플랫폼과 운영체제에 따라 각각 다른 쉘 코드와 탐색 코드를 갖는다. 본 논문에서 구현된 플랫폼은 Intel Pentium이며 운영체제는 Linux (Redhat 6.1)이다. C 컴파일러는 gcc, 디버거는 gdb를 사용하였으며 라이브러리 호출 추적기로는 ltrace와 objdump를 이용하였다.
이를 사용하여 구현한 결과 CERT의 bugtraq 에서 발표된 취약점을 포함한 qpoper, sendmail, man, named 등 다수의 프로그램에서 취약점을 발견할 수 있었다.

5. 결론 및 향후 연구 과제

본 논문에서는 플랫폼과 운영체제가 주어진 환경에서의 실행 파일에 대한 일반적인 버퍼 오버플로우 탐지기를 제안하였다. 이를 이용하여 임의의 파일에 대한 버퍼 오버플로우를 탐지함으로써 신뢰도를 높일 수 있을 것이며 다른 플랫폼으로의 이식에 관한 연구가 진행중이다.

6. 참고 문헌

[1]. David Wagner, Jeffrey S. Foster, Eric A. Brewer, Alexander Aiken. " A First Step Toward Automated Detection of Buffer Overrun Vulnerabilities" , In Proceedings of the Year 2000 Network and Distributed System Security Symposium, 2000.

[2]. D. Seeley. "A Tour of the Worm". Proceedings of the 1989 Winter USENIX Technical Conference pp.287-314. January 1989.

[3]. Aleph One. " Smashing The Stack For Fun And Profit" . Phrack Magazine issue49 vol.7 , Nov. 1996.

[4]. Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. "Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade". DARPA Information Survivability Conference and Expo. January 2000.

[5]. Matt Conover. " w00w00 on Heap Overflows" . http://www.w00w00.org/articles

[6]. Fyodor. " Remote OS detection via TCP/IP Stack Fingerprinting" . Phrack Magazine issue54 vol.8, Dec. 1998.

[7]. Taeho Oh. " Advanced buffer overflow exploit" http://www.securityfocus.com