오버플로우 공격에 대한 방어 기술 연구

신동휘*, 김승주, 원동호**

*성균관대학교 정보통신공학부 정보보호연구소

The Research of Defense Technique of Overflow Attack

Dong-Hwi Shin*, Seungjoo Kim, Dongho Won**

*Information Security Group, School of Information and Commuication Engineering, SungKyunKwan University

요 약

Buffer Overflow 취약점은 오래전에 발표된 기술이지만 현재까지도 보안 취약점에서 많은 부분을 차지하는 것 중에 하나이다. CVE Vulnerability Databasc에서 Buffer Overflow를 검색해 보면 알 수 있듯이 지금까지 수많은 Buffer Overflow 취약점들의 발견되었고 이를 바탕으로 하는 공격들이 많이 일어나고 있다. 또한 공격에 대처하기 위하여 각각의 공격방법에 대한 대처 방법을 제시하고 커널과 컴파일러의 패치에 이루어지고 있지만 해커들은 그에 대응하는 공격 방법들을 찾아내곤 한다. 본 논문에는 지금까지 발표된 많은 Buffer Overflow 공격 기술에 대해 주제별로 확인하고 보다 근본적인 해결책을 만들기 위한 방향을 제시하고자 한다.

I. 서론

Buffer Overflow는 프로그램이 실행하는 과정에서 Buffer에 할당된 크기보다 더 많은 양의 데이터를 저장시키면서 발생되는 문제를 말한다. 이러한 문제가 일어나는 이유는 임의의 함수가 Buffer에 데이터를 저장하는 과정에서 입력되는 데이터의 크기를 확인하지 않는 즉, Boundary Check를 하지 않기 때문이다.

많이 알려진 Stack Buffer Overflow 공격은 할당된 Buffer보다 큰 데이터를 입력하고 Heap Overflow는 선언된 파일명이나 함수 포인터들을 수정하여 공격자 실행시키기 원하는 코드를 실행하도록 하는 것이다. 물론 Data, Text 등의 다른 영역을 활용하는 많은 공격기법들도 있다.

위와 같은 Buffer Overflow 공격을 방지하기 위해서 가장 근본적인 해결책으로는 프로그램의 개발 단계에서 Buffer Overflow에 취약한 함수들을 사용하지 않고 Secure Coding 기법을 적용하는 것이다. 하지만 Secure Coding 적용이 현실적으로 어렵기 때문에 다른 방어 방법들이 제시되고 있다. 방어 방법들로서 Stack 영역에 Return Address가 수정되었는지 확인하기 위한 Stackguard, 취약함 함수를 보완한 libsafe, Stack의 실행 권한을 제거한 NX, Random한 데이터 영역을 나타내는 ASLR, 컴파일러의 Dummy 추가 등의 기법이 적용되었고 Return Address의 저장과 비교, H/W를 사용하는 방법, Pointerguard등의 방법들이 제시되고 있다.

하지만 여전히 Overflow 취약점은 발표되고 있으며 완벽한 해결책은 제시되지 않았다. Overflow의 문제는 모두 실행 가능한 파일에서 일어나므로 본 논문에서는 Overflow 방어기술 개발에 기여하고자 Linux상에서 실행파일의 형

* 주저자, **교신저자

※ 본 연구는 정보통신부 및 정보 통신 연구 진흥원의 대학 IT 연구센터 육성 · 지원사업의 연구 결과로서 수행되었음.

식인 ELF를 사용하는 방법을 제안하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 Overflow 공격에 대한 간략히 설명한다. 3장에서는 현재 사용되고 있거나 논문에서 제시되고 있는 Overflow 방어 기술과 문제점에 대해서 설명한다. 4장에서는 또 다른 방법의 Overflow 공격 방어기술을 제안한다. 마지막으로 5장에서는 논문의 결론에 대하여 기술한다.

II. Overflow 공격 기술

2.1 Stack Overflow

Stack이란 임의의 프로세스가 메모리에 적재되어 실행될 때 할당되는 영역중의 하나로서 함수에서 사용되는 지역변수를 동적으로 할당하고 함수에 인자를 건네주는 등의 역할을 수행하는 공간이다. 다시 말해 프로그램이 실행되는 동안에 데이터를 포함하고 있는 메모리의 연속적인 블록이다. 또한 Buffer란 같은 데이터 타입의 메모리의 연속된 블록이다. 그리고 Return Address는 어떤 함수가 호출되면 호출된 함수가 끝나고 돌아가서 실행될 곳의 주소를 저장하는 영역이다. 그렇다면 Stack Overflow는 Stack에 할당된 Buffer의 크기보다 많은 데이터 입력을 통하여 Buffer에 할당된 영역을 벗어나서, 임의의 영역에 데이터가 변경되는 것이다. 이러한 문제점을 이용하여 공격자는 실행되기 원하는 함수의 주소를 호출할 수 있도록 Stack의 데이터들을 수정하게 되는데 공격자가 수정하고자 영역은 Return Address이다. 공격자는 Boundary Check를 하지 않는 취약한 함수(ex. strcpy, sprintf, scanf 등)의 사용여부를 확인하고 Setuid/Setgid Bit(또는 Sticky Bit, 프로그램이 실행되는 동안에 일시적으로 Root의 권한으로 권한 변경이 일어남.)가 설정된 프로그램을 Debugging하여 Buffer의 크기와 Return Address의 위치를 확인한다. 이를 바탕으로 공격자가 실행하기 원하는 곳의 주소(Shellcode의 주소)를 Return Address에 저장하되도록, 할당된 Buffer의 크기보다 많은 입력값을 주게 되어 Root의 권한을 가지는 Shell을 띄우게 되는 방식으로 공격을 진행하게 된다.

그림 1은 Process 실행 시 Memory Structure이고 그림 2는 Stack Overflow를 위한 Payload를 나타내었다.

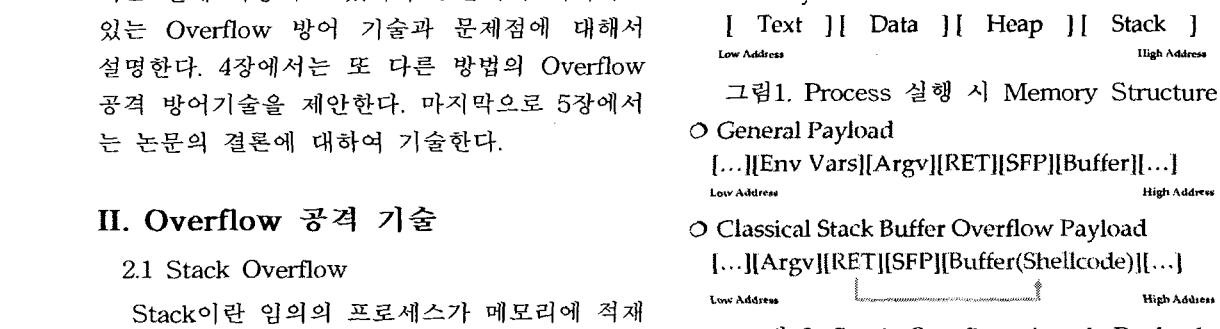

그림1. Process 실행 시 Memory Structure

그림 2. Stack Overflow Attack Payload

2.2 Heap Overflow

Heap이란 프로그램에 의해서 동적으로 할당되는 메모리 영역이다. Heap Overflow가 Stack Overflow보다는 널리 알려지지 않은데, 그 이유는 Stack Overflow를 사용한 공격보다 어렵다는 것과 특별한 테크닉에 의존하며 특히 선행조건을 만족해야 하는 등의 문제 때문이다. 하지만 Heap Overflow가 Stack Overflow에서 나타나는 문제점을 해결할 수 있는 등의 장점들이 있기 때문에 사용되고 있다. Stack Overflow에서는 공격자 변경하기 원하는 영역의 주소가 Return Address였다. 하지만 Heap Overflow에서는 변경시킬 포인터가 Buffer 다음에 위치하도록 하며 할당된 Buffer이다. 이 Buffer를 Overflow 시켜서 프로그램 내부에 선언된 임의의 파일명이나 함수 포인터를 공격자가 실행시키기 원하는 주소로 덮어써서 그것을 실행시켜서 Root Shell을 띄우게 되는 것이다.

III. Overflow 방어 기술

3.1 Stackguard[2]

Stackgurad는 Stack 영역의 보호를 위하여 Stack의 영역에 Return Address의 수정이 일어나는지 확인하기 위해서 "canary"라는 특별한 문자를 사용한다. 만약 "canary"라는 문자를 확인할 수 없다면 Return Address가 수정될 가능성이 매우 높기 때문에 프로그램의 실행을 막을 수 있으며, 공격자가 "canary"라는 문자를

미리 알아낼 가능성이 있기 때문에 Random/NULL/Terminator/XOR Canary(그림 3)등을 사용하여 공격자가 쉽게 공격할 수 없게 만든다. 하지만 Stackguard도 안전하다고 말할 수는 없다. Kil3r이라는 해커가 Phrack에 Stackguard를 우회할 수 있는 방법을 제시했기 때문이다[3].

○ Stackguard Payload

[...][Env Var][Argv][RET][Canary] [Buffer][...]

Low Address High Address

그림3. Stackguard Stack

3.2 *libsafe[4]*

libsafe는 Buffer Overflow와 Format String 공격을 방어하기 위해서 보안 취약성을 가지고 있는 함수들을 보완하여 개발한 것이다. 하지만 Heap Overflow와 같은 방법으로 libsafe를 무력화가 가능하므로 안전하다고 말할 수 없다.

3.3 No eXecute(NX)[5]

NX는 No eXecute를 의미하는데, NX 속성을 가지도록 설계되어 있는 임의의 섹션은 단지 데이터로만 쓰일 수 있다. 그렇기 때문에 프로세서나 임의의 명령은 해당 섹션에 위치할 수 없다. 그러므로 NX 속성이 있는 섹션에 공격자가 실행시키고자 하는 임의의 코드를 주입하여 실행시키는 것은 불가능하게 된다. 이 기술은 OpenBSD에서는 W^X[6], Redhat의 Fedora Core에서는 Exec Shield[7], Gentoo등의 시스템에서는 PaX[8]에 적용되어 사용되고 있으며 윈도우에서는 XP 서비스 팩 2/2003서버 서비스 팩 1[9]에서 적용되어 사용되고 있다. NX 속성이 적용되기 이전의 Stack 영역에서는 실행이 가능하여 Stack Overflow 공격이 가능하였으나 이제는 Stack에 NX 속성을 부여하여 실행이 불가능하여 Stack을 통한 공격을 어려워졌다. 하지만 Solar Designer가 제시한 Return-into-libc 공격[10][11]을 통하여 NX를 우회하여 Overflow 공격이 가능하다.

3.4 Address Space Layout Randomization[12]

ASLR은 주요한 데이터 영역의 위치를 가상 주소(Virtual Address)에서 Random하게 하는 것이다. 이 기술은 Executable, Library, Heap, Stack에 적용된다. 이와 같은 ASLR은 NX에서 나타나는 Return-into-libc에 공격을 무력화 시킬 수 있고 NX와 동일하게 W^X, Exec-Shield, PaX에 적용되어 사용되고 있다.

3.5 Gcc Dummy

GCC는 Linux에서 사용하는 GNU 표준 Compiler의 모음이다. GCC는 2.96 이후 버전으로 Compile된 프로그램의 경우에 개발자가 작성한 Buffer의 크기와 다르게 Buffer의 구조가 변형되는데, 이는 정의된 Buffer들 사이에 임의의 Dummy가 붙어 공격자가 Buffer Overflow 공격을 하는데 어려움을 준다. 이 것은 GDB를 통하여 Disassemble하게 되면 Assembly Code에서 Buffer의 구조가 변형된 것을 알 수 있다. 하지만 Dummy가 완벽하게 Random하지 않고 규칙을 가지고 있기 때문에 공격자가 Dummy의 크기를 예측할 수 있다.

3.6 기타

위에서 설명한 방어 기술 외에 논문에서 제시되는 방법들로는 Return Address를 임의의 영역에 복사여 변경여부 확인하는 방법[13], H/W에서 암호화 기술을 사용하는 방법[14], Pointerguard[15], Pointer에 Watermark사용[16], System Call의 확인[17], Shadow Stack[18], RAD[19] 등이 있다 하지만 운영체제의 성능 문제 및 기타 문제로 인하여 활용도는 높지 않다.

IV. 새로운 Overflow 방어 기술 제시

앞장에서 제시한 Overflow에 대한 방어기술의 대부분은 하나의 공격 방법을 막기 위한 방법들이다. Overflow라는 문제점은 실행파일에 대해서 나타는 문제이기 때문에 본 논문에서는 Linux에서의 실행파일 형식인 ELF(Executable and Linking Format)[20]를 적용한 방법을 제시하고자 한다.

4.1 ELF

ELF란 32bit Intel Architecture 환경에서 동작하는 portable object file format이다. 이전의 a.out과 COFF가 있었지만 ELF가 보다 강력하고 유연성을 가지고 있어 표준으로 선택되어

사용되고 있다.

4.2 Overflow 방어 기술 방안

대부분의 프로그램은 Shell 위에서 프로세스로 동작하게 되는데 이때 Kernel은 ELF를 확인하고 각 섹션 별로 나누어서 적절한 메모리에 가져다 놓고 실행을 하게 된다. 그러므로 다른 영역을 활용한 Overflow 문제를 해결하기 위해서는 ELF를 사용한 방어 기술이 적절하다. 특히 Data, BSS, Text 섹션들의 정보가 ELF에 들어있다. 이 영역들은 많이 알려져 있는 Stack 영역에 대한 공격이 어려워지면서 Overflow 공격을 위해 사용되는 섹션이다. 그러므로 ELF에서 각 섹션의 정보를 활용한 방어기술의 개발이 가능하다. 예를 들면, ELF의 섹션에서 .init 그리고 .fini라는 부분에서 .init는 프로세스의 초기화에 관련된 부분이고 .fini는 프로세스의 종료에 관련된 부분인데 .init와 .fini를 사용하여 프로세스의 시작과 끝에서 프로세스의 이후 동작을 확신할 수 있는 알고리즘을 추가하여 보다 신뢰성 있는 프로세스의 진행이 이루질 수 있다. 또한 임의의 라이브러리 함수를 호출할 때 .plt와 .got를 사용하는데 공격자가 예측할 수 없도록 Random한 Offset를 사용하도록 하는 방법을 제안할 수 있다. 이를 통하여 공격자는 .plt와 .got의 내용을 예측하여 공격하는 것도 불가능하게 만든다. 또한 각 영역의 시작주소 그리고 크기 등이 저장되는 곳들의 주소를 절대주로로 바꾸는 과정에서 mapping table을 사용하는데, Offset를 예측하기 어렵게 만든다면 공격이 매우 어려워질 것이다. 또한 ELF와 비슷하게 윈도우 시스템에는 PE라는 것이 있다. 그러므로 Overflow 공격을 막기 위한 ELF 연구를 발전시켜 PE에도 적용시켜 ELF와 PE의 구분 없이 Overflow 공격을 방어할 수 있는 기술이 가능할 것이다.

V. 결론 및 발전 방향

본 논문에서는 Overflow 공격 중 Stack과 Heap에서 발생할 수 있는 공격 방법에 대해 간략히 설명하였다. 그리고 방어에 대한 간략한 소개에서는 시스템들에서 적용되고 있는 것들을 기술인 Stackguard, libsafe, NX, ASRL, Dummy 등을 주로 설명하였다. 이와 같이 Stack 영역에서 발생하는 Overflow에 대한 문제는 이제 거의 해결단계에 접어들었다. 하지만 지금도 많은 공격방법들이 만들어지고 공개되지 않은 공격방법들도 존재한다. 그러므로 보다 근본적인 방어 기술의 개발을 위하여 ELF를 활용하는 것을 제안하였다. 현재 알려졌거나 알려지지 않은 많은 공격방법들이 있으므로 앞으로 방어를 위한 연구는 계속되어야 할 것이고 ELF를 개선하여 방어하는 연구가 진행되어야 할 것이다.

[참고문헌]

[1] Alephl, Smashing The Stack For Fun and Profit Phrack 49-I4 Nov. 1996

[2] Stackguard(http://immunix.org/stackgurad.html)

[3] Bulba and Kil3r, Bypassing stackguard and stackshield Phrack 50-10 May. 2000

[4] libsafe(http://www.research.avayalabs.com/project/libsafe)

[5] NX(http://omniknow.com/common/wiki.php?in=en&term=NX_bit)

[6] W^X(http://www.openbsd.org/papers/ven05-deraadt/index.html)

[7] Exec-Shield(http://people.redhat.com/mingo/exec-shield/)

[8] PaX(http://pax.grsecurity.net/)

[9] XP ServicePack 2/2003Server ServicePack 1(http://www.microsoft.com)

[10] Solar Designer, Getting around non-executable stack (and fix) Aug. 1997

[11] Nergal, The advanced return-into-lib(c) exploits: Pax case study Phrack Dec. 2001

[12] ASLR(http://pax.grsecurity.net/docs/aslr.txt)

[13] Yong-Joon Park, Gyungho Lee, Repairing Return Address Stack for Buffer Overflow Protection, Apr. 2004

[14] Nathan Tuck Brad Calder George Varghese, Hardware and Binary Modification Support for Code Pointer Protection From Buffer Overflow, Dec. 2004

[15] Crispin Cowan, Steve Beattie, John Johansen, and Perry Wagle, "PointGuardTM: Protecting Pointers from Buffer Overflow Vulnerabilitie", 12th USENIX Security Symposium, pp.91-104

[16] Yongsu Park and Yookun Cho, An Efficient Pointer Protection Scheme to Defend Buffer Overflow Attacks, Jun. 2004

[17] Yang-Seo Choi, Dong-il Seo, and Sung-Won Sohn, A New Stack Buffer Overflow Hacking Defense Technique with Memory Address Confirmation Dec. 2001

[18] Marc L. Corliss E Christopher Lewis Amir Roth, Using DISE to Protect Return Addresses from Attack Mar. 2005

[19] Tzi-cker Chiueh Fu-Hau Hsu, RAD: A Compile-Time Solution to Buffer Overflow Attacks, Apr. 2001

[20] Tool Interface Standard (TIS) Executable and Linking Format Specification Version 1.2 May. 1995