BBL단위 기계어 패턴을 이용한 버퍼 오버플로우 탐지법

한광희*, 정영길**, 진영택***
한밭대학교 컴퓨터공학과
e-mail:konai1238@gmail.com*
e-mail:younggil94@naver.com**
e-mail:ytjin@hanbat.ac.kr***

Buffer Overflow Detection Using Machine Pattern of BBL

Kwang-Hee Han, Young-Gil Joeng, Young-Taek Jin
Dept of Computer Engineering, HanBat University

요 약

버퍼 오버플로우 취약점은 자체적으로 외부코드를 실행시키거나 다른 취약점과 연계해 공격될 수 있는 취약점으로 이를 탐지하기 위해 많은 연구들이 진행되었다. 기존의 버퍼 오버플로우 취약점 탐지기법에 대한 연구는 취약함수를 검색하거나 동적오염분석으로 얻은 비 신뢰 데이터의 흐름을 감시하여 취약함수로 전파되는 걸 탐지하는 방식 이었다. 하지만 이러한 기법은 다양한 한계를 가진다. 본 논문은 이러한 문제들을 해결하기 위해 버퍼 오버플로우가 발생하는 기계어 패턴을 BBL단위로 감시하고, 비 신뢰 데이터의 흐름을 추적하여 기존의 탐지 사각지대를 없애는 방법을 제안한다.

1. 서론

버퍼오버플로우 공격은 자체적으로 외부코드를 실행시키는 사고로 이어질 수 있으며, 다른 취약점을 발동 시키는 용도로 사용될 수 있기 때문에 취약함수에 대한 운영체제 독립적, 종속적 방어기법에 많은 방법들이 소개되어 있다[1][2][5][6][7][8]. 버퍼가 가지는 고유 성질을 방어하는 기법 외에 컴파일 수준에서는 버퍼 오버플로우에 취약한 함수 사용을 경고하고 있으며, 여러 취약점 탐지기에서는 취약한 함수의 사용을 탐지하거나 동적 오염 분석기법을 적용하여 오염데이터의 흐름이 취약함수로 향하는지를 판단하는 기법들이 존재한다[1][5]. 이러한 버퍼 오버플로우에 취약한 함수를 탐지/감시 하는 기법들은 탐지 도구 관리에 효율적이지 못하며 문서화된 함수만을 감시하기 때문에 인라인 어셈블리로 만들어진 사용자 정의 함수, 컴파일러 최적화에 대한 함수 인라인, 버전별로 유동된 이름을 가지는 네이티브 함수를 대상으로 독립적이지 못하다.

본 논문에서는 버퍼 오버플로우가 가능한 취약한 기계어 패턴을 탐지하는 방식을 제안한다. 제안 방법은 Dynamic Binary Instrumentation Tool과 오염분석을 이용해 버퍼복사, 이동과 관련된 기계 명령어를 감시하고 비 신뢰 데이터의 흐름을 추적하여 기계명령의 특정 연산자/피연산자로 작용하는지 분석한다. 제안 방법을 이용해 버퍼오버플로우를 동적으로 탐지하는 BOFDetector을 구현하였으며, 상용화된 소프트웨어 및 취약점이 존재하는 소프트웨어에 적용하여 테스트를 진행하였다.

2. BOFDetector

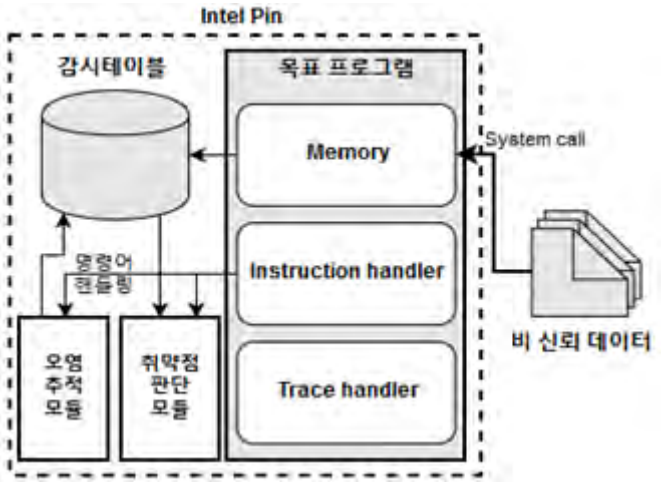
본 논문은 기존의 취약 함수를 감시하는 기법이 아닌, 버퍼 복사/이동에 관련된 기계어의 패턴을 찾고 오염된 데이터가 해당 명령셋의 연산자 작용을 감시하는 방법을 제안한다. 버퍼 오버플로우 취약점은 버퍼의 복사/이동에 의해 발생하게 된다. 이는 버퍼의 수용 가능한 용량을 벗어난 크기를 검증하지 않고 복사함수로 넘기거나, 크기 한계를 지정하지 않고 Null바이트에 의존하는 문자열 복사 등에 의해 발생하게 된다. x86 아키텍처의 레지스터는 32bit 사이즈로, 한 개의 레지스터만을 사용하여 버퍼를 복사하는 명령은 4바이트가 한계이기 때문에 이터레이터 연산이 필요하게 된다. 크기를 인자로 받는 함수는 이 이터레이터의 조건에 크기가 들어가게 되며 만약 크기에 대한 데이터가 오염된 비 신뢰성 데이터라면 오버플로우 취약점이 발생하게 된다. 문자열의 경우 목적지 버퍼의 할당 크기가 비 신뢰성 데이터라면 오버플로우 취약점으로 판단 한다. 이러한 함수들은 표1 과 같다.

<표 1> 버퍼오버플로우 취약함수

sprintf	snprintf	fprintf
strcpy	strcat	strncat
memcpy	sprintf_s	...

표1의 함수들의 기계 명령셋은 이터레이터 조건 판별, 점프 명령, 복사 명령이다. 이 때 이터레이터의 조건 판별

식과 복사 되는 버퍼가 오염 데이터와 관련 있다면 버퍼 오버플로우로 판단할 수 있다. 본 기법을 이용하기 위해선 Dynamic Binary Instrumentation Tool과 동적오염 분석 기법[5]이 필요하다.



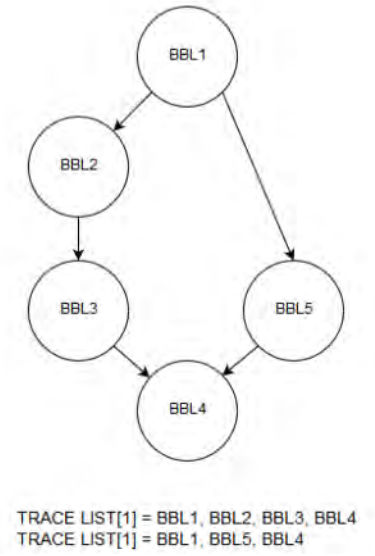
<그림 1> BOFDetector

BOFDetector의 전체적인 구조와 동작은 그림1 과 같다. 첫째, 타겟 프로세스의 시스템 콜, 네트워크 소켓에 들어오는 외부 데이터를 오염된 비 신뢰성 데이터로 간주하여 감시 테이블에 저장한다. 이때 필요한 Dynamic Binary Instrumentation Tool로 Intel사의 PinTool[3]을 활용했다. 둘째, 프로세스의 데이터 이동/복사/연산 명령을 실시간으로 핸들링하여 오염된 데이터의 전파를 추적한다. 단위는 1바이트 크기로 레지스터는 해당 레지스터 크기를 단위로 오염추적을 진행한다. Intel 아키텍처의 명령셋[4] 중 본 연구에서 처리하는 명령어는 표2 와 같다. 셋째, 버퍼 복사/이동과 관련된 BBL(Basic Block)패턴을 감지하여 오염 데이터와의 연계성을 분석하여 결과를 출력한다.

<표 2> 오염 전파 대상 명령어

명령어	구 분
MOV, XCHG	전송 명령
ADD, SUB, MUL, DIV	산술 명령
XOR, AND, OR	2진 명령
PUSH, POP	스택 명령
MOVS, LODS, STOS	버퍼 명령

BBL(Basic Block)이란 코드의 흐름이 if, goto, call 등의 명령에 의해 분기되는 부분까지의 단락이다. 이러한 단락을 BBL이라 표현하고 있으며, BBL들이 연결된 모든 코드의 흐름을 리스트화 한 것을 TRACE 라고 한다.(그림 2) 이러한 BBL의 끝 명령어를 조사하여 분기의 종류를 따지고 목적지가 TRACE의 내부로 향한다면 해당 단락이 반복되는 페이지인지 기계어 수준에서 확인할 수 있다.

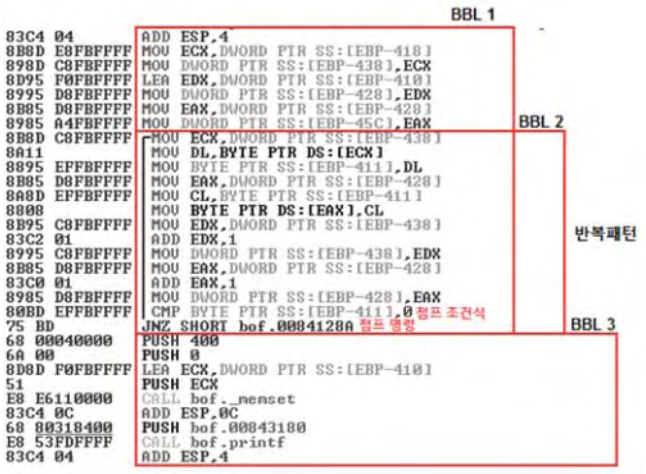


<그림 2> TRACE, BBL

BOFDetector는 PinTool에서 제공하는 BBL관련 API를 활용하여 소프트웨어의 모든 BBL의 끝 명령어를 검사해 반복되는 페이지를 추출한다.

3. 설계 및 구현

PinTool의 Instruction handling 을 이용하면 인텔 아키텍처에서 지원하는 모든 명령에 대한 처리함수를 작성할 수 있다. 본 연구에서는 표2 에서 보인 명령을 대상으로 오염 데이터를 추적한다. 동적 오염 추적의 시작은 PinTool에서 제공하는 Image handling과 Systemcall handling을 활용하여 Nt!ReadFile와 같은 외부데이터를 읽는 시스템콜, 네트워크 소켓 에서 데이터를 받아들이는 함수들 (recv, read)에서 출발한다. 이렇게 받아들이는 데이터의 주소를 최초 감염주소로 판단하며 감염테이블에 등록시키고 위에서 서술한 Instruction handling을 활용하여 오염 추적을 진행한다.



<그림 3> BBL단위 코드 페이지

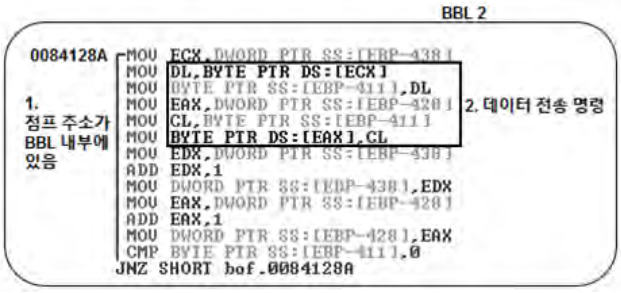
버퍼 복사/이동 명령은 반복 형태를 갖는다. 기계어 수준에서 해당 반복 패턴은 그림3와 같이 조건식과 점프명령으로 나타나게 된다. 이러한 패턴은 BBL단위로 코드페이지를 나눔으로써 시각적으로 쉽게 파악할 수 있다.

PinTool에서 제공하는 Trace handling은 이러한 BBL단위 코드 실행을 감시하는 API를 제공하기 때문에 자동화된 감지모듈을 구현할 수 있다. 본 연구에서 활용한 방법은 다음과 같다.

1) 그림4 (1) 각 BBL과 루틴의 의 시작 주소와 끝 주소를 저장하고 점프 명령이 시작과 끝 주소 사이에 위치하면 반복 되는 코드페이지 라고 판단한다.

2) 그림4 (2) 1)의 코드 페이지에 데이터 전송 명령이 있다면 버퍼 복사/이동 명령셋으로 판단한다.

3) 점프 조건(그림4. CMP 명령의 두 번째 연산자)과 데이터 소스(그림4. CL 레지스터)에 오염된 데이터가 개입된다면 버퍼오버플로우 취약점으로 분류한다.



<그림 3> 버퍼 복사/이동의 반복패턴

4. 실험 및 결과

본 논문에서 제안한 BBL단위 기계어 패턴을 이용한 버퍼 오버플로우 탐지 기법의 탐지율을 측정하기 위해서 표3의 실험 환경과 버퍼 오버플로우 취약 소프트웨어, 상용화된 소프트웨어 총 5개를 대상으로 BOFDetector을 적용하여 실험을 수행하였다. 취약소프트웨어는 Metasploit에 등록되어있는 소프트웨어를 대상으로 진행하였고, 상용화된 소프트웨어는 임의로 선정하였다. 취약한 소프트웨어는 모두 탐지에 성공하였고, 상용화된 소프트웨어 5개중 3개에 대해 탐지에 성공하였다.

<표 3> 실험 환경 및 대상 소프트웨어

CPU Architecture	Intel x86
Operating System	MS Windows 10
소프트웨어	분 류
FTPShell Client	스택 버퍼
Disk Pulse Enterprise	로그인 버퍼
통합 문서뷰어	힙 버퍼
랜섬웨어 백신	힙 버퍼
웹 보안 모듈	전역 버퍼

5. 결론 및 향후 계획

BBL단위 기계어 패턴을 이용한 버퍼 오버플로우 탐지 기법은 기존의 연구들이 갖는 문제점을 보완한다. 본 논문에서 제안하는 방법의 장점은 다음과 같다. 첫째, 바이너리 코드를 대상으로 진행하기 때문에 원본 소스가 필요 없다. 둘째, 소프트웨어에 독립적으로 적용되는 탐지기법이기 때문에 재 빌드가 필요 없다. 셋째, 사용자 정의 인라인 함수, 이름이 변경되는 네이티브 함수, 컴파일러 최적화에 의해 인라인 되는 코드들이 갖는 버퍼 오버플로우 취약점을 탐지할 수 있다. 따라서 본 연구의 결과는 버퍼 오버플로우 취약점 탐지의 사각지대를 없애는 대책이 될 수 있을 것이다. 그러나, 본 연구에서 제안한 방법은 3가지 조건만으로 패턴을 분석하여 탐지하는 것이기 때문에 오탐이 발생할 수 있다. 이러한 예로, 반복되는 코드페이지 내에서 동적으로 메모리를 할당하는 경우 오버플로우는 일어나지 않는다. 이러한 결과는 오탐율에 영향을 주게 되며 오버플로우 탐지 조건들을 추가해야 할 필요성이 된다. 따라서 향후 계획으로 오탐율을 개선하기 위한 연구를 진행할 예정이다.

참고문헌

[1] C. Pasareanu and W. Visser, "A survey of new trends in symbolic execution for software testing and analysis," in ICSE, 2009

[2] Cristian Cadar, Daniel Dunbar, and Dawson Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. In Proceedings of the USENIX Symposium on Operating System Design and Implementation, 2008.

[3] Intel Pin, [www.pintool.org]

[4] Intel® 64 and IA-32 Architectures Software Developer's Manual

[5] James Newsome and Dawn Song. "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software", In Network and Distributed Systems Security Symposium, San Diego, California, USA, 2005.

[6] Michael Sutton, Fuzzing: brute force vulnerability discovery, Addison-Wesley, Jul. 2007.

[7] Patrice Godefroid , Michael Y. Levin , David Molnar, SAGE: whitebox fuzzing for security testing, Communications of the ACM, v.55 n.3, March 2012

[8] 김성호, 박용수 (2014). 동적오염분석과 SAT 해석기를 이용한 상용 소프트웨어 보안 취약점 분석 연구. 한국정보과학회 학술발표논문집, 994-996.