안전한 소스코드 작성을 위한 자동화 분석 도구의 개발

하경휘*, 김상영*, 최진우*, 우종우*, 김홍철**, 박상서**
*국민대학교 컴퓨터학부
**국가보안기술연구소

A Development of Automatic Analysis tool for Writing Secure Code

KyoungHui Ha*, SangYoung Kim*, JinWoo Choi*, ChongWoo Woo*,
HongChul Kim**, Sangseo Park**
*School of Computer Science, Kookmin University
**National Security Research Institute
E-mail: cwwoo@kookmin.ac.kr*, hckim@etri.re.kr**

요 약

리눅스와 같은 공개 운영 체제의 출현은 많은 프로그램 개발자들로 하여금 시스템 내부적인 측면에 대하여 한층 더 이해를 쉽게 할 수 있는 계기가 되었다. 그러나 프로그램 보안이라는 문제 영역에 관한 심각한 고려가 개발 시부터 병행되지 못함으로 인해 치명적인 프로그램 취약성이 잠재되어 왔다. 따라서 주요 침해 사례들은 상당 수 시스템이 사용하는 일부 알려진 프로세스들의 버그와 같은 프로그램 코드 수준에서의 취약성을 이용한 악의적인 프로그램들이 대부분을 차지한다. 본 논문에서 이러한 문제점을 해결하기 위해 제안하는 바는 다음과 같다. 첫째, 현재까지 널리 알려져 있는 프로그램 관련 취약성에 대한 명확한 이해, 둘째, 프로그램 코드 자체에 잠재되어 있는 취약성들이 내재된 함수들에 관한 분석, 그리고 마지막으로 이러한 분석 기술서를 기반으로 프로그램 취약성 검사 자동화 도구를 제안한다.

1. 서 론

컴퓨터 사용의 급격한 증가와 컴퓨터 응용 기술의 급속한 발전과 함께 응용 프로그램에 대한 사용자의 요구 또한 다양해지고 있다. 이러한 요구에 따라 무수한 개발자들에 의해 응용들이 창출되고 있지만 응용 개발 단계에서부터 안전한 프로그램 개발을 위한 보안 개념이 확립되고 있지 못하는 실정이다. 또한 일반적으로 보안은 상호 연결된 시스템들 사이의 안전한 접속에 관한 문제만이 주요 쟁점이 되어왔으며, 그 원천적인 문제를 해결하고자 하는 노력은 이루어지지 못하고 있다. 이로 말미암아 응용 개발자들은 계속된 응용 유지보수 작업 및 패치(patch)의 개발로 인한 부담과 동시에 시스템 보안 관리자에게도 많은 부담이 가중되고 있다.

보안 관리 사이클(security management cycle)에 있어서, 시스템 보안 관리자들에 의해 수행되는 대부분의 작업들은 각각의 시스템들에서 구동 중인 프로그램들의 결함을 보완하기 위한 것이다.

이를 위해 제공되는 보안 패치(secure patch)들의 설치 작업이 보안 관리 사이클의 대부분을 차지하고 있다. 현재 제공되는 보안 패치들은 이미 우리들에게도 널리 알려진 침해 사례, 예를 들어 버퍼 오버플로우(buffer overflow)와 같은 공격[1][2][3]의 원인이 되는 취약성(vulnerability)을 포함하고 있는 프로그램들의 결함을 보완하기 위한 것들이라는 점이다.

본 논문에서는 이러한 문제들을 원천적으로 해결하기 위하여 프로그램 개발 시 공격의 개연성을 가지는 취약성들에 대한 실질적인 코드들을 조사 및 분석하고 이를 위하여 소스 코드 수준에서 잠재할 수 있는 취약성을 자동으로 분석하고 이에 따른 지침서를 제공하는 자동화 도구의 개발을 제안한다.

2. 관련연구

프로그램 내의 잠재적인 취약성을 해결하기 위한 방안으로써 원천적으로 프로그램 작성 단계에서 보다 명확한 보안 개념을 가지고 소스 코드를 작성해야 함은 명백한 사실이다. 이를 위해서는 함수 취약성과 관련된 문제점과 대표적인 부류들의 정의 및 각 부류들의 잠재적인 취약성과 그 원인에 대하여 심도

이 연구는 국가보안기술연구소 2003 년 위탁 연구 과제에서 지원 받았음

있게 이해되어야만 한다. 본 장에서는 이를 위하여 잠재적인 취약성이 내재되어 있는 다양한 부류의 함수들에 대하여 기술하고[4][5][6], 이들 취약성들을 검사하는 대표적인 몇몇 도구들에 대하여 기술한다.

2.1 표준 C 프로그램에서의 취약성

프로그램 내부의 잠재적인 취약성으로 인해 초래될 수 있는 위협은 대표적으로 다음과 같이 네 가지로 요약될 수 있다[7].

① 버퍼 오버플로우(buffer overflow) : 버퍼 오버플로우 공격은 입력으로 데이터를 취하고, 이를 저장할 버퍼의 크기가 적합한지를 판별하기 위한 검사가 코드로 구현되지 않은 프로그램들을 그 목표로 삼는다. 그 결과 해당 서비스의 접근 거부, 또는 시스템의 다운 등을 초래한다.

② 경쟁 상태(race condition) : 다중 접근이 적절하게 프로그램 되어지지 않은 경우, 파일 또는 변수와 같은 공유 자원을 포함하도록 하려는 행동이다. 이러한 경쟁 상태는 신뢰 또는 비 신뢰 프로그램에서도 야기되어 질 수 있다.

③ 접근제어(access control) : 대부분의 경우 " setuid bit" 가 설정된 프로그램들에서 취약성이 발견된다. 이들은 특정 그룹 또는 사용자의 권한을 가지고 실행된다. 이러한 유형의 프로그램들은 버퍼 오버플로우와 경쟁 상태뿐만 아니라 잠재적인 공격자에게 그들이 가지지 않은 권한들을 제공하는 것만으로도 손상을 입는다.

④ 포맷 스트링(format string) :비 신뢰 데이터를 입력으로 사용하는 경우로써, 포맷 지시자를 포함한 스트링을 인자로 하는 포맷 함수 계열의 사용에 있어서 발생할 수 있다. 그 결과 공격자는 프로그램의 스택 내부를 이해하여, 프로세스의 사용 메모리 내부의 원하는 위치의 값을 조작할 수도 있다.

2.2 시스템 사례

프로그램 취약성을 검사하는 대표적인 다섯 가지의 도구들에 대하여 설명한다[8].

- Pscan: C 코드 내에서 악용되는 버퍼 오버플로우와 포맷 스트링 공격을 발견하는 도구로써 그 기능이 한정되어 있다. 대표적으로 표준 C 라이브러리로부터 함수 printf와 관련한 일반적인 문제들을 발견한다[9].
- Flawfinder: Pscan 도구와 그 기능이 유사하지만, 보다 많은 유형의 오류들을 발견한다. 포맷 함수 계열과 스트링 조작 함수들 외에 경쟁 상태와 시스템 호출에 관한 문제들을 발견할 수 있다. 그 결과들은 위험 레벨(risk level)에 의해 정렬될 수 있다[10].
- RATS: 여러 개발 언어들를 스캔하는 기능을 가지는 유일한 검색 도구이다. 가능한 언어들로는 C, C++, Perl, PHP, 그리고 Python 소스 코드 등이 가능하며, 가능은 Flawfinder와 유사하다. 실행 결과는 HTML로 생성된 보고서를 제공한다[11].
- Splint: 비교적 그 덩치가 가벼우며, 정적 분석 도구(static analysis tool)에 속한다. 그러나 다른 도구들과는 달리, Splint는 코딩 시 발생하는 실수들, 반드시 요구되어야 하는 구조적 코딩 스타일을 찾아내며, 보안이라는 관점에서는 그 기능이 빈약하다[12].
- MOPS: 다른 도구들처럼 사용법이 직관적이고 간단하지 않으며 사용자들은 FSM(Finite State Machines)을 사용한 모델을 생성하는 방법을 알아야만 한다. 또한 MOPS는 설치나 수행 시에 Java 런-타임 환경을 요구한다[13].

2.3. Win32 API에서의 취약성 및 해결안

Win32 API에서 취약성 함수들에 대한 7가지 부류를 알아보고 해당 부류의 함수들의 취약성 원인과 해결방안을 제시하였다.

① 접근위반(access violation) 관련 취약성: 이 부류의 함수들은 일반적으로 문자열을 다루는 함수들이다. 이 부류의 함수들의 경우 NULL 문자를 포함하지 않은 문자열을 적절히 다루지 못하는 문제점을 가지고 있다. 이러한 문제가 발생하면 "Access Violation" 발생 후 프로그램이 비정상적으로 종료될 수 있다. 이를 위해 버퍼를 NULL로써 미리 초기화 해 줌으로 취약성을 예방할 수 있다.

② 버퍼 빅(buffer big) 관련 취약성: 이 부류의 함수들은 버퍼의 문자열을 복사, 비교, 연결 등의 기능을 가지며 사용 시 수행할 바이트 수를 명시하게 되어있다. 그러나, 목적 버퍼의 크기가 명시된 바이트 수 보다 작을 때 문제점이 발생한다. 이를 해결 하기 위해서는 지정된 버퍼의 크기를 충분히 크게 해 주거나 입력값을 버퍼의 크기에 맞게 제한해 주는 방법이 있다.

③ 동적 링크 라이브러리(dll) 로드 관련 취약성: 이 부류에 속하는 함수들은 실행 파일을 로드하기 위해서 사용되는 함수들이다. 이 함수들은 환경 변수 " path" 에 의존적이다. 만약 공격자가 악의 적인 목적의 프로그램을 해당 실행 파일명을 가지고 주요 경로의 우선되는 위치에 놓

아 둔다면 공격자의 의도대로 악의 적인 목적의 프로그램이 실행 될 수 있다. 이를 예방 하기 위해서는 실행 파일명을 사용할 때 전체 경로명을 사용하는 방법이 있다.

④ 임계영역(critical section) 관련 취약성: 이 부류에 속하는 함수들은 임계 영역을 다루는 기능을 수행하는 함수들이다. 일반적인 경우에는 문제가 발생하지 않지만 적은 메모리 상태일 경우 예외가 발생할 수 있다. 각 초기화 함수들에 대해서 대체함수를 사용하거나 예외 처리 구문을 적용하는 방법으로 예방할 수 있다.

⑤ 프로세스 실행에 의한 취약성: 이 부류에 속하는 함수들은 새로운 프로세스를 생성하거나 다른 실행 가능한 프로그램들을 현재 프로그램 내에서 실행하는 기능을 수행한다. 이러한 작업들은 환경 변수 " path" 에 의존적으로 수행된다. 그러나 이러한 방법은 악의 적인 목적을 가진 공격자에게 " Trojan" 을 심을 수 있는 가능성을 제공한다. 이러한 문제는 프로세스 실행을 위한 파일의 절대 경로를 명시함으로써 예방 할 수 있다.

⑥ 실행 프로그램 확장자 관련 취약성: 이 분류에 속하는 함수들은 프로세스 실행 취약성 함수들과 마찬가지로 새로운 프로세스를 생성하거나 다른 실행 가능한 프로그램들을 실행하는 기능을 수행한다. 이 함수들은 실행파일의 확장명 없이 실행 파일의 이름만을 사용하여 실행할 경우가 많은 데 이러한 경우 문제를 발생시킬 수 있다. 실행 파일들의 취약성을 명시함으로써 이러한 문제들을 예방 할 수 있다.

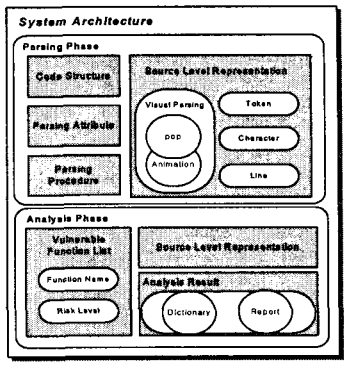
⑦ 포맷 스트링 (format string)에 의한 취약성: 2.1의 포맷 스트링과 동일

3. 자동화 도구의 설계 및 구현

본 논문에서는 소스 코드에서 위와 같은 표준 C 함수와 Win32API에 대한 취약성들을 발견해 내고 이를 해결하기 위한 방안을 제시하기 위한 자동화 도구를 제안한다. 본 장에서는 소스 코드의 분석을 위한 자동화 도구의 전체적인 구조와 개별적인 모듈에 대해 설명하고 이 들의 상호 연관성에 대해서 상세하게 설명한다.

3.1 설계

[그림 1]은 자동화 도구의 전체적인 시스템 구조를 도식화 한 것이다. 이 도구는 크게 두 가지 측면에서 소스 코드 분석 작업을 수행한다. 첫째는 소스 코드에 대한 파싱과 관련된 기능을 수행하는 파싱부(Parsing Phase) 이고 둘째는 파싱된 결과를 가지고 함수의 취약성을 분석하는 기능을 수행하는 분석부(Analysis Phase)이다.



[그림 1] 시스템 구조

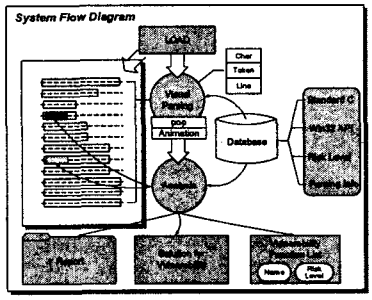
[그림 2]는 전체적인 시스템의 흐름을 나타내는 다이어그램을 나타낸다. 순차적인 시스템의 흐름은 다음과 같다.

① 소스 코드를 읽어 들여 파싱 모듈에 의해서 시각적인 특징이 강조된 파싱 과정에 들어가간다.

② 파싱 방법이 결정되면 파싱 모듈은 데이터베이스에서 파싱과 관련된 정보를 가져온다. 이렇게 가져온 데이터를 기반으로 소스 코드에 대한 파싱 과정이 수행되고 진행되는 결과들이 소스 코드의 에디팅 영역과 부가적인 다이얼 로그 창에 적용된다.

③ 파싱 과정이 종료 된 후 사용자의 요청에 의해서 분석 작업이 수행된다.

④ 분석 작업이 시작되면 분석 모듈은 데이터베이스에 있는 취약성 함수 정보와 위험도와 관련된 정보를 가져온다.



[그림 2] 시스템 흐름 다이어그램

⑤ 이 정보를 기반으로 소스 코드에서 발견된 취약성 함수에 대해서 소스 코드 에디트 영역에 위험도에 따라 다른 색상으로 표기 한다.

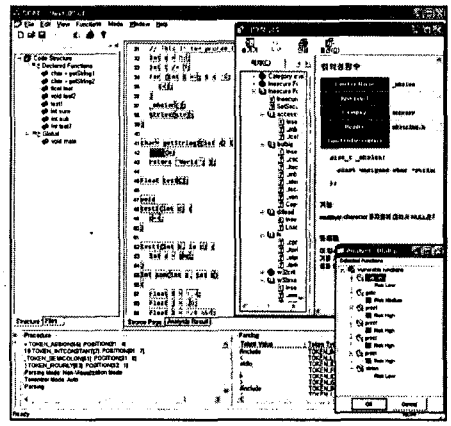
⑥ 각각의 발견된 취약성 함수와 위험성 정보에 대해서 "Vulnerability Function List"에 나타내고, 분석된 결과에 대해서 별도의 탭에 표시한다.

⑦ 각 취약성 함수에 대해서 사용자의 요청에 의해

서 해결 방안을 제시해 주는 기능을 수행한다.

3.2 구현

본 절에서는 소스 코드 분석 자동화 도구의 구현에 대해서 기술한다. 본 자동화 도구는 Windows 계열의 OS에서 동작하며 표준 C 함수와 Win32 API 함수들의 취약성을 발견하고 이에 대한 적절한 해결 방안을 제시하는 기능을 수행한다.



[그림 3] 전체적인 시스템 구성

[그림 3]은 파싱과 분석이 끝난 후의 시스템의 전체적인 모습을 보여 준다. 크게 세 가지 다이얼로그의 형태로 구분하여 볼 수 있는데, 첫 번째 다이얼로그는 소스 코드와 이를 파싱하는 과정과 결과를 나타내는 낸다. 두 번째 다이얼로그는 선택한 취약성 함수에 대해서 발생할 수 있는 문제점과 이에 대한 해결 방안을 제시한다. 세 번째 다이얼 로그에서는 소스 코드에서 발생한 취약성 함수들의 리스트와 위험도를 나타낸다.

4. 결론

본 논문에서는 위에 언급한 문제들과 관련하여 표준 C 함수 및 Win32 API 함수에 대해서 보안에 취약한 함수들을 조사 및 분류하고 그러한 취약성들이 발생하는 원인들 해결 방안에 관하여 분석하였다. 분석 내용은 다음과 같이 요약해 볼 수 있다.

첫째, 시스템 취약성 관리에 있어서 대부분의 시스템 관리가 패치에 의존하고 있으며 보안 정책이 없이 패치에 의존한 시스템 관리는 패치 개발의 지연 시 위험 부담을 증가시키게 된다. 이러한 취약성을 야기시키는 결점들이 발생하는 위치와 시점은 프로그램의 개발 초기 단계에서 발생한다.

둘째, 프로그램 취약성을 위한 해결 방안은 원천적으로 프로그램 작성 단계에서 보다 명확한 보안 개념을 가지고 소스 코드를 작성해야 한다는 것이다. 따라서 프로그램을 위한 보안 구조를 확립하고 그에 따른 개념을 가진 프로그램을 작성해야 한다.

셋째, 취약성과 관련된 문제점과 관련하여 표준 C 함수와 Win32 API 함수의 취약성에 관한 대표적인 부류들의 정의와 각 부류들에서의 발생 가능한 잠재적인 취약성과 그 원인에 대하여 기술하고 안전한 프로그래밍을 위해 고려 해야 할 점에 대하여 기술하였다. 마지막으로 개발 단계에서의 소스 코드 수준에서 공격자들의 목표가 될 수 있는 취약점들, 예를 들어 잠재적인 취약성을 가지는 함수들을 분석하고 이에 대한 데이터베이스를 구축하였다. 데이터베이스는 표준 C 함수의 경우 70여 개의 함수들과 Win32 API 함수의 경우 80여 개의 함수들로 구성되었다. 그리고 견고한 보안 구조의 확립을 가능케 하기 위하여, 즉시 개발자에게 발견된 취약성에 대하여 보고하고, 이에 대한 지침서를 제공하는 자동화된 도구를 설계하고 개발하였다.

참 고 문 헌

[1] A. One, "Smashing the stack for fun and profit", Phrac 7(49), November 1996.

[2] A.baratloo and N.Singh, "Transparent Run-Time Defense Against Stack Smashing Attacks," USENIX Annual Technical Conference.

[3] Mudge, "How to write Buffer Overflows," available to http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html.

[4] M. Conover, "w00w00 on Heap Overflows," available at http://www.w00w00.org/files/articles/heaptut.txt.

[5] C. Cowan, C. Pu, H. Hinton, J. Walpole, "StackGuard: Automatic Adaptive Detection and Prevention of Buffer – Overflow Attacks," 7 th USENIX Security Conference, 1998.

[6] D. A. Wheeler, "Secure programming for Linux and Unix HOWTO," available at http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html, 2003.

[7] M. Howard and D. LeBlanc, "Writing Secure Code", Microsoft Press, ISBN 0-7356-1588-8, 2002.

[8] T. La, "Secure Software Development and Code Analysis Tools", available at http://www.sans.org/rr/paper.php?id=389, 2003.

[9] Pscan, available at http://www.striker.ottawa.on.ca/~aland/pscan/.

[10] Flawfinder, available at http://www.dwheeler.com/flawfinder.

[11] RATS, available at http://www.securesoftware.com.

[12] Splint, available at http://lclint.cs.virginia.edu/.

[13] MOPS, available at http://www.cs.berkeley.edu/~daw/mops.