안드로이드 커널 모듈 취약점 탐지를 위한 자동화된 유닛 테스트 생성 기법

김윤호○, 김문주
한국과학기술원 전산학부
kimyunho@kaist.ac.kr, moonzoo@cs.kaist.ac.kr

Automated Unit-test Generation for Detecting Vulnerabilities of Android Kernel Modules

Yunho Kim○, Moonzoo Kim
School of Computing, KAIST

요 약

본 논문에서는 안드로이드 커널 모듈의 취약점을 탐지하기 위한 자동 유닛 테스트 생성 기법을 제안한다. 안드로이드 커널 모듈의 각 함수를 대상으로 테스트 드라이버/스텁 함수를 자동 생성하고 동적 기호 실행 기법을 사용하여 테스트 입력 값을 자동으로 생성한다. 또한 안드로이드 커널 모듈의 함수 포인터와 함수 선행 조건을 고려하지 않은 테스트 생성으로 인한 거짓 경보를 줄이기 위해 정적 분석을 통한 함수 포인터 매칭 기법과 def-use 분석을 사용한 함수 선행 조건 생성 기법을 제안한다. 자동 유닛 테스트 생성 기법을 안드로이드 커널 3.4 버전의 세 모듈에 적용한 결과 기존에 존재하던 취약점을 모두 탐지할 수 있었으며 제안한 거짓 경보 감소 기법으로 평균 44.9%의 거짓 경보를 제거할 수 있었다.

1. 서 론

안드로이드 시스템은 전 세계 스마트폰 시장의 80% 이상을 차지하고 있으며 스마트 폰 이외에도 스마트 TV, 시계, 카메라, 자동차 등 다양한 시스템에 안드로이드 시스템이 탑재되고 있어 그 영향력이 점차 커지고 있다. 이에 따라 안드로이드 시스템을 대상으로 하고 있는 공격이 점차 증가하고 있다.

안드로이드 커널 모듈의 취약점은 공격자가 전체 시스템을 장악할 수 있게 하기 때문에 가장 위험한 취약점이다. 일반 어플리케이션의 취약점은 안드로이드 권한 시스템으로 인해 그 영향력이 제한적이지만 커널 모듈의 취약점은 안드로이드 권한 시스템에서 가장 많은 권한을 갖고 있는 안드로이드 커널을 직접 조작할 수 있기 때문에 권한 시스템의 영향을 받지 않고 공격을 수행할 수 있게 된다.

본 논문에서는 안드로이드 커널 모듈의 취약점을 탐지하기 위해 유닛 테스트를 자동으로 생성하는 기법을 제안한다. 안드로이드 커널 모듈의 각 함수를 대상으로 유닛 테스트를 수행하기 위한 테스트 드라이버/스텁 함수를 자동으로 생성하고 동적 기호 실행 기법[1]을 사용하여 테스트 입력 값을 자동으로 생성한다. 또한 안드로이드 커널 모듈 함수를 테스트할 때 발생할 수 있는 거짓 경보를 제거하기 위한 정적 분석 기법을 개발하였다.

본 논문에서 제안하는 기법이 실제 효과적인지 확인하기 위해 안드로이드 커널 3.4 버전에서 발견된 3개의 취약점을 대상으로 자동 유닛 테스트를 수행하였다. 그 결과 기존에 존재하던 취약점을 모두 탐지하여 취약점 탐지에 효과적임을 확인하였다. 또한 제안한 기법으로 44.9%의 거짓 경보를 제거하여 거짓 경보 감소 기술이 효과적임을 확인하였다.

본 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.13-912-06-003, 안드로이드 신규 취약점 탐지를 위한 모바일 소프트웨어 보안 테스팅 도구 개발)

2. 관련 연구

안드로이드 커널 모듈의 취약점을 탐지하는 관련 연구는 찾기 어려우며 대부분 리눅스 커널 모듈을 대상으로 테스트 및 검증을 수행하였다.

Linux Driver Verification 프로젝트[2], Avinux 프로젝트[3], DDVerify 프로젝트[4]에서는 모델 검증 도구를 사용하여 리눅스 디바이스 드라이버 소스 코드를 정형 검증하고 리눅스 커널 모듈의 버그를 검출하였다. Carburizer[5]는 정적 분석 기법을 활용하여 하드웨어 장애가 발생할 수 있는 지점을 탐지하여 하드웨어 장애로 인한 드라이버 오류를 방지하였다. BitBlaze[6] 는 가상 머신 기반의 플랫폼으로 소스 코드 없이 바이너리 상태에서 리눅스 시스템을 분석할 수 있으며 바이너리 프로그램의 심볼릭 분석을 통해 보안 취약점을 발견하였다.

본 연구는 안드로이드 커널을 직접 타겟하고 있으며 유닛 테스트 기법을 사용하여 기존 정형 검증이나 전체 시스템을 대상으로 적용하는 기법과 달리 확장성(scalability)이 높아 다양한 종류의 커널 모듈에 대해 취약점을 탐지할 수 있는 장점이 있다.

3. 동적 기호 실행 기반 자동화된 유닛 테스트 기법

동적 기호 실행 기반 자동화된 유닛 테스팅 기법[7]은 대상 프로그램의 유닛 테스트 드라이버/스텁 함수를 자동으로

표 1 동적 기호 실행 입력 설정

입력 값 자료형	기호 실행 입력
기본 자료형	해당 자료형에 맞는 기호 실행 입력 값 설정
배열	모든 배열 원소에 대해 배열 원소의 자료형에 맞는 기호 실행 입력 값 설정
구조체	구조체의 모든 필드에 대해 해당 자료형에 맞는 기호 실행 입력 값 설정
포인터	1) 포인터 자료형 T의 기호 실행 입력을 처음 생성하는 경우: T가 가리키는 자료형인 *T의 메모리 크기 만큼 메모리를 할당하고 *T의 동적 기호 실행 입력을 설정 하여 포인터 입력 값으로 할당 2) 이미 포인터 자료형 T의 기호 실행 입력을 생성한 경우: 기존에 생성된 *T를 가리키도록 포인터 입력 값 설정

생성하고 해당 유닛 테스트의 입력 값을 동적 기호 실행 기법을 사용하여 자동으로 생성하는 기법이다. 대상 함수 f()의 모든 파라메터와 f()가 사용하는 모든 전역 변수를 동적 기호 실행의 심볼릭 입력 값으로 설정하고 f()가 호출하는 모든 함수를 심볼릭 값을 리턴하는 심볼릭 스텁 함수로 대체한 후 생성된 테스트 드라이버/스텁 함수에 동적 기호 실행 기법을 적용해 테스트 입력 값을 생성한다. 대상 함수 f()의 파라메터와 전역 변수의 자료형에 따라 표 1과 같이 심볼릭 입력 값을 설정한다.

유닛 테스트 수행 과정에서 취약점을 탐지하기 위해 포인터 참조, 배열 참조 및 나누기 연산에서 널 포인터 참조, 잘못된 배열 범위 참조, 0으로 나누기를 탐지하기 위한 단언문을 삽입하였다.

4 거짓 경보 감소를 위한 정적 분석 기법

4.1 정적 분석을 사용한 함수 포인터 매칭 기법

안드로이드 커널 모듈은 객체 지향 기법을 C로 구현하기 위해서 커널 모듈의 함수를 구조체의 함수 포인터 필드로 선언하여 사용한다

그림 1은 net/ceph 모듈의 함수 포인터 초기화를 보여준다. 1039번째 줄에서 ceph_connection_operations구조체 mon_con_ops를 선언하면서 각각의 함수 포인터 필드에 실제 어떤 함수가 대입되는지 정의한다. 예를 들어 1040번째 줄에서 get 함수 포인터는 ceph_con_get 함수를 가리키고 있으며 1041번째 줄에서 put 함수 포인터는 ceph_con_put 함수를 가리킨다. ceph_mon_init 이라는 ceph 모듈의 초기화 함수가 호출되면 768번째 줄에서 모듈이 사용할 함수 포인터 구조체를 대입하는 역할을 수행하여 커널 모듈의 함수 포인터를 초기화한다.

이와 같이 안드로이드 커널 모듈에는 고유의 초기화 함수가 있고 해당 초기화 함수에서 설정되는 함수 포인터 구조체가

```
net/ceph/mon_client.c
 748 int ceph_monc_init(struct ceph_mon_client *monc, struct ceph_client *cl)
…
 768     monc->con->ops = &mon_con_ops;
…
1039 static const struct ceph_connection_operations mon_con_ops = {
1040     .get = ceph_con_get,
1041     .put = ceph_con_put,
…
1045 };
```

그림 1 안드로이드 커널 net/ceph 모듈의 함수 포인터

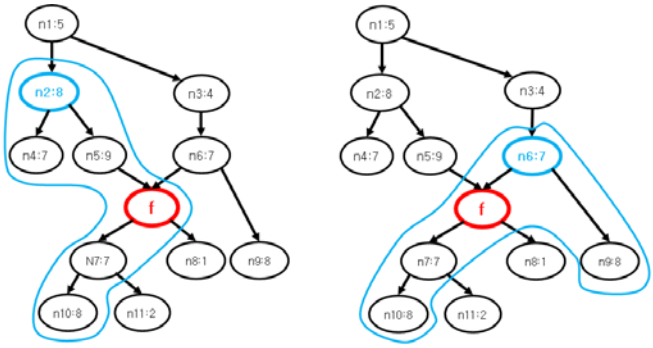
존재한다. 따라서 안드로이드 커널 모듈의 코드를 정적 분석하여 초기화 함수를 찾고 초기화 함수에서 설정하는 구조체를 분석하여 테스트 드라이버 생성시 함수 포인터를 설정할 수 있도록 하여 거짓 경보를 줄였다.

4.2 Def-use 분석을 사용한 함수 선행 조건 생성 기법

자동화된 유닛 테스트의 거짓 경보를 줄이기 위해 def-use 분석을 활용한 함수 선행 조건 생성 기법을 개발하였다. 자동화된 유닛 테스트 수행 시 테스트 대상 함수 f를 테스트 드라이버가 직접 호출하는 것이 아니라 f를 호출하는 f의 caller 함수[2]를 호출함으로써 f의 선행 조건을 생성할 수 있게 하였다. 함수 f를 호출할 때 함수 f의 선행 조건을 만족하는 입력 값을 생성하거나 선행 조건 만족 여부를 체크하는 것은 주로 함수 f를 호출하기 이전에 실행되는 f의 caller 함수들이기 때문에 f의 caller 함수들을 사용해 f를 호출함으로써 선행 조건을 만족한 상태에서 f를 실행할 수 있다. 함수 f의 모든 caller 함수를 호출할 경우 각 caller 함수마다 유닛 테스트를 수행해야 해서 테스트 수행 시간이 길어지고, 선정할 상위 caller 함수의 단계를 제한하지 않으면 main 함수와 같은 프로그램 시작 함수까지 호출하여 테스트 범위가 커지고 유닛 테스트의 효과가 떨어진다. 따라서 def-use 분석을 사용하여 caller 함수와 테스트 대상 함수 f의 의존도를 계산하고 의존도가 큰 caller 함수만 호출하여 테스트 대상 함수 f의 선행 조건 생성에 큰 영향을 줄 수 있는 함수만 선정하였다.

함수 f와 함수 g의 의존도 $D_f(g)$는 함수 g에서 정의하고 함수 f에서 사용하는 def-use쌍의 수로 정의하였다. 배열의 경우 각각의 원소를 하나의 변수로 간주하였으며 구조체의 경우 각각의 필드 멤버를 하나의 변수로 간주하였다. 포인터 변수 p의 경우 p 자체의 def-use는 일반 변수와 동일하게 계산하였다. *p에 대한 def(혹은 use)가 있는 경우 먼저 포인터 분석을 통해 p와 동일한 메모리를 가리키는 포인터 q, r 등을 찾고 *q, *r을 def(혹은 use)로 하는 모든 def-use의 수를 계산하여 그 합을 def-use의 수로 계산하였다. 테스트 대상 함수 f와 다른 함수의 의존도를 계산하여 평균을 구하고 평균보다 큰 의존도를 갖는 함수를 실제 호출하였다.

[2] 직접 f를 호출하는 함수뿐 아니라 함수 호출 그래프에서 시작 함수 정점부터 함수 f 정점까지 가능한 모든 경로에 포함되는 함수 전부를 포함한다.



(a) 함수 n2가 유닛 테스트의 시작 함수 (b) 함수 n6이 유닛 테스트의 시작 함수

그림 2 유닛 테스트 시작 함수 및 실제 호출 함수 탐색 예제

테스트 대상 함수 f와의 의존도를 사용하여 실제 호출할 함수를 찾는 과정은 다음과 같다. 먼저 유닛 테스트 드라이버가 호출할 시작 함수를 찾는다. 테스트 대상 함수 f에서 시작하여 f의 caller 함수 중 함수 f와 caller 함수의 의존도가 평균 의존도보다 낮은 caller 함수를 만날 때까지 함수 호출 그래프를 탐색한다. 평균 의존도보다 의존도가 큰 가장 상위의 caller 함수 g를 시작 함수로 하고 시작 함수 g에서 출발하여 모든 호출 가능한 함수 정점을 탐색하면서 테스트 대상 함수 f와의 의존도가 평균보다 높은 함수를 찾아 해당 함수를 실제 호출한다. 그림 2의 예제를 살펴보자. 각 정점은 <함수명:테스트 대상 함수 f의 의존도>이고 평균 의존도는 5.7이다. 먼저 (a)의 경우 함수 f에서 왼쪽 caller 함수 n5로 탐색을 시작하는 경우이다. n5와 n5의 caller n2는 평균 의존도보다 의존도가 더 크지만 n5의 caller n1은 의존도가 평균보다 작기 때문에 n2가 시작 함수가 된다. n2가 호출 가능한 함수 중 평균보다 큰 의존도를 갖는 함수는 파란색 실선에 포함되는 함수들이다. (b)의 경우는 오른쪽 caller n6으로 탐색을 시작하는 경우이다. n6의 caller n3는 평균보다 의존도가 작기 때문에 n6이 시작 노드가 되며 n6이 호출 가능한 함수 중 평균보다 의존도가 높은 함수가 파란 실선 안에 포함된다. (a)에서는 n2를 시작함수로 하여 n2, n4, n5, f, n7, n10이 실제 호출되는 함수이며 (b)에서는 n6를 시작 함수로 하여 n6, f, n7, n9, n10이 실제 호출되는 함수이다. 그 외의 함수는 심볼릭 값을 리턴하는 심볼릭 스텁 함수로 대체된다.

5 실험 결과

안드로이드 커널 드라이버 유닛 테스트 자동 생성 도구는 Clang/LLVM 3.4 버전을 사용하여 구현하였다. 동적 기호 실행은 CREST-BV[8]를 사용하여 수행하였다.

본 도구를 사용하여 안드로이드 커널 3.4 버전의 3개 네트워크 모듈 net/ceph(CVE-2013-1059), net/core(CVE-2013-1763), net/sctp(CVE-2014-0101) 모듈에 있는 3개 취약점을 대상으로 테스트를 수행하였다. 각 모듈은 11kLOC~35kLOC의 크기를 갖는다. 실험은 쿼드코어 Core I5-3570K@3.8GHz, 16GB 메모리를 갖는 서버 50대 규모의 클러스터에서 수행하였으며 각 서버당 4개의 유닛 테스트를 동시에 수행하였다. 동적 기호 실행 탐색 기법은 DFS 탐색

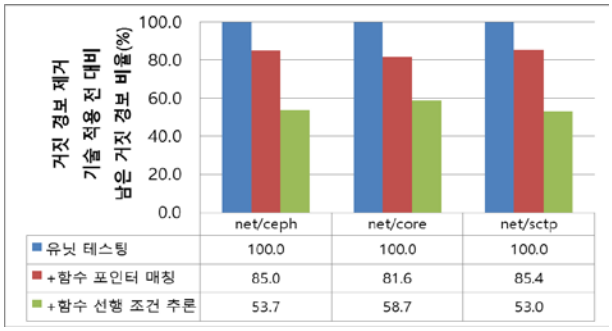


그림 3 거짓 경보 제거 결과

기법을 사용하고 각 유닛 테스트마다 5분의 시간 제한을 설정하였다.

적용 결과 약 56.6분의 시간이 소요되었으며 세 모듈에서 기존에 보고된 취약점을 모두 발견할 수 있었다. 거짓 경보 제거 결과는 그림 3과 같이 평균 44.9%의 거짓 경보를 제거할 수 있었다.

6 결론 및 향후 연구

본 논문에서는 안드로이드 커널 모듈의 취약점을 찾기 위한 자동 유닛 테스트 기법을 제안하고 커널 모듈 유닛 테스트의 거짓 경보를 제거하기 위한 기법을 제안하였다. 실제 안드로이드 커널 취약점을 대상으로 적용한 결과 효과적으로 취약점을 탐지할 수 있었으며 제안한 거짓 경보 제거 기술은 44.9%의 거짓 경보를 효과적으로 제거할 수 있었다.

향후 단순 크래시 외에 권한 상승 등 더 다양한 취약점을 탐지하고 거짓 경보를 줄이기 위한 기술을 추가 개발할 것이다.

참고문헌

[1] Godefroid et al., "DART: Directed Automated Random Testing", PLDI, 2005

[2] Khoroshilov et al., "Configurable toolset for static verification of operating systems kernel modules", Programming and Computer Software, Vol. 41, No. 1, 2015

[3] Post et al., "Avinux: Towards Automatic Verification of Linux Device Drivers", ProVeCS, 2007

[4] Witkowski et al., "Model Checking Concurrent Linux Devvice Drivers", ASE, 2007

[5] Kadav et al., "Tolerating hardware Device Failures in Software", SOSP, 2009

[6] Song et al., "BitBlaze: A New Approach to Computer Security via Binary Analysis", ICISS, 2008

[7] Kim et al., "Automated unit testing of large industrial embedded software using concolic testing", ASE, 2013

[8] 김윤호 외 2인, "CREST-BV: 임베디드 소프트웨어를 위한 Bitwise 연산을 지원하는 향상된 Concolic 테스팅 기법", 정보과학회 논문지, Vol. 40, No. 2, 2013