(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2015년11월11일
(11) 등록번호 10-1568224
(24) 등록일자 2015년11월05일

(51) 국제특허분류(Int. Cl.)
G06F 21/14 (2013.01)
(21) 출원번호 10-2014-0190954
(22) 출원일자 2014년12월26일
심사청구일자 2014년12월26일
(56) 선행기술조사문헌
JP2012256220 A
JP2011048639 A

(73) 특허권자
고려대학교 산학협력단

(72) 발명자
이희조

리홍제

(뒷면에 계속)

(74) 대리인
특허법인엠에이피에스

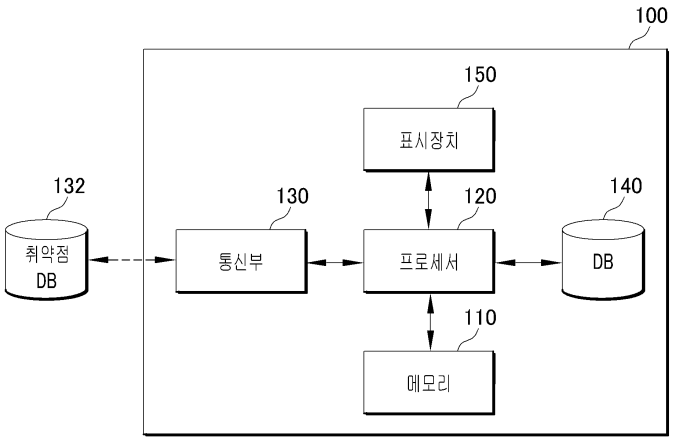
전체 청구항 수 : 총 11 항 심사관 : 박진아

(54) 발명의 명칭 **소프트웨어 취약점 분석방법 및 분석장치**

(57) 요 약

본 발명에서는 소프트웨어 취약점 분석 방법 및 분석 장치가 개시된다. 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석방법은, 임의의 분석대상 소스코드로부터 재사용 소스코드에 해당하는 코드클론을 탐색하는 단계, 탐색된 코드클론 내에서 패치 정보를 기초로 취약 지점 및 취약 지점과 관련된 취약 데이터를 검출하는 단계, 분석대상 소스코드 내에서 검출된 취약 데이터를 역추적하여 입력 지점에 해당하는 소스를 획득하는 단계 및 입력 지점에서 취약 지점까지의 경로를 기초로 컨커릭 테스팅(Concolic testing)을 수행하여 탐색된 코드클론이 상기 분석대상 소스코드 내에서 취약점에 해당하는지 여부를 검증하는 단계를 포함한다.

대 표 도 - 도1



(72) 발명자 **권혁민**

권종훈

명 세 서

청구범위

청구항 1

소프트웨어 취약점 분석 방법에 있어서,

임의의 분석대상 소스코드로부터 재사용 소스코드에 해당하는 코드클론을 탐색하는 단계;

상기 탐색된 코드클론 내에서 패치정보를 기초로 취약 지점 및 상기 취약 지점과 관련된 취약 데이터를 검출하는 단계;

상기 분석대상 소스코드 내에서 상기 검출된 취약 데이터를 역추적하여 입력 지점에 해당하는 소스를 획득하는 단계; 및

상기 입력 지점에서 상기 취약 지점까지의 경로를 기초로 컨커릭 테스팅(Concolic testing)을 수행하여 상기 탐색된 코드클론이 상기 분석대상 소스코드 내에서 취약점에 해당하는지 여부를 검증하는 단계를 포함하는 소프트웨어 취약점 분석 방법.

청구항 2

제 1 항에 있어서,

상기 코드클론을 탐색하는 단계는

상기 분석대상 소스코드 및 상기 분석대상 소스코드의 초기 소스코드를 각각 정규화하는 단계;

상기 정규화된 분석대상 소스코드 및 상기 정규화된 초기 소스코드를 각각 코딩 라인 단위로 토큰화하는 단계;

상기 토큰화된 분석대상 소스코드를 순차적으로 일정 개수의 코딩 라인마다 묶어서 복수의 유닛 소스코드를 구성하는 단계; 및

상기 복수의 유닛 소스코드 중 상기 토큰화된 초기 소스코드를 포함하는 소정의 유닛 소스코드를 상기 코드클론을 포함하는 유닛 소스코드로 판단하는 단계를 포함하는 소프트웨어 취약점 분석 방법.

청구항 3

제 2 항에 있어서,

상기 판단하는 단계는 블룸 필터(Bloom filter)를 통해 판단하는, 소프트웨어 취약점 분석 방법.

청구항 4

제 1항에 있어서,

상기 탐색하는 단계는 상기 분석대상 소스코드와 상기 분석대상 소스코드의 초기 소스코드를 비교하여 탐색하는, 소프트웨어 취약점 분석 방법.

청구항 5

제 1항에 있어서,

상기 탐색하는 단계는 상기 패치 정보를 통해 탐색하는, 소프트웨어 취약점 분석방법.

청구항 6

제 1 항에 있어서,

상기 소스를 획득하는 단계는 코드 구조 그래프(Code Structure Gragh) 기법을 적용하여 상기 검출된 취약 데이터를 역추적하는, 소프트웨어 취약점 분석 방법.

청구항 7

제 1 항에 있어서,

상기 검증하는 단계는

상기 입력 지점에 임의의 초기값을 입력하여 상기 컨커릭 테스팅의 결과를 획득하는 단계; 및

상기 컨커릭 테스팅의 결과로부터 상기 분석대상 소스코드의 스테이트먼트가 유의미한 작업 명령으로서 동작하도록 기정의된 보안 필요조건을 만족하는지 여부를 판단하는 단계를 포함하는, 소프트웨어 취약점 분석 방법.

청구항 8

제 7항에 있어서,

상기 보안 필요조건은 상기 스테이트먼트의 기설정된 함수의 파라미터, 메모리 접속 방식 및 정수 연산 방식 중 적어도 하나에 대한 것인, 소프트웨어 취약점 분석 방법.

청구항 9

소프트웨어 취약점 분석 장치에 있어서,

임의의 분석대상 소스코드로부터 취약점 분석을 수행하는 프로그램이 저장된 메모리;

상기 프로그램을 실행하는 프로세서를 포함하되,

상기 프로세서는 상기 프로그램 실행에 따라,

임의의 분석대상 소스코드로부터 재사용 소스코드에 해당하는 코드클론을 탐색하는 단계;

상기 탐색된 코드클론 내에서 패치정보를 기초로 취약 지점 및 상기 취약 지점과 관련된 취약 데이터를 검출하는 단계;

상기 분석대상 소스코드 내에서 상기 검출된 취약 데이터를 역추적하여 입력 지점에 해당하는 소스를 획득하는 단계; 및

상기 입력 지점에서 상기 취약 지점까지의 경로를 기초로 컨커릭 테스팅(Concolic testing)을 수행하여 상기 탐색된 코드클론이 상기 분석대상 소스코드 내에서 취약점에 해당하는지 여부를 검증하는 단계를 수행하는, 소프트웨어 취약점 분석 장치.

청구항 10

제 9항에 있어서,

상기 탐색하는 단계는 블룸 필터(Bloom filter)를 통해 탐색하고,

상기 탐색된 코드클론을 저장하는 데이터 베이스를 더 포함하는, 소프트웨어 취약점 분석 장치.

청구항 11

제 9항에 있어서,

상기 취약점에 해당하는지 여부를 검증하는 단계의 결과를 화면상에 표시하는 단계를 더 실행하는, 소프트웨어 취약점 분석 장치.

발명의 설명

기 술 분 야

[0001] 본 발명은 소프트웨어 취약점 분석방법 및 분석장치에 관한 것이다.

배 경 기 술

[0002] Symantec사에서 배포하는 Internet Security Threat Report 2014에 따르면, 소프트웨어 취약점 공격은 매년 증

가하는 추세이다. 특히 지난 2013년에는 기존에 알려지지 않은 제로데이 소프트웨어 취약점이23개가 새로 발견되었으며, 이는 2012년 대비 61%가 증가한 수치이다. 국내에서도 소프트웨어 취약점을 이용한 공격이 날로 증가하고 있으며, 이를 통해 악성코드 감염, DDoS 공격, 개인정보 유출 등 다양한 공격이 보고되고 있다.

[0003] 현재 소프트웨어 취약점의 분석 및 대응을 위하여 심볼릭 테스팅, 퍼지 테스팅 등의 기법을 적용하고 있다. 하지만 분석 시간 및 정확도 측면에서 낮은 성능을 보이는 문제가 있다. 또한, 이러한 기법을 사용하기 위해서는 수준 높은 전문성, 사전 지식 및 경험을 갖춘 인적자원이 필요하기 때문에 소프트웨어 취약점 분석에 있어 접근성이 떨어진다는 문제점이 있다.

[0004] 따라서, 더욱이 발전해가는 IT 산업 규모에 따라 매일 수많은 신규 소프트웨어가 개발되고 있는 현실에 비추어 볼 때, 소프트웨어 취약점 분석의 효율성과 정확도를 높이고, 분석에 필요한 시간 및 자원을 줄이는 기법에 대한 연구가 필수적이다.

[0005] 한편, 한국공개특허 제2013-7015514호 (발명의 명칭: 코드 클론 검출을 이용하는 지능형 코드 디퍼런싱을 수행하는 방법 및 시스템)는 코드 클론 검출 기술을 이용하는 지능형 코드 디퍼런싱용 시스템 및 방법에 관하여 개시하고 있다.

발명의 내용

해결하려는 과제

[0006] 본 발명의 일부 실시예에 따르면, 코드클론 검출 후 소프트웨어 취약점을 분석하는 소프트웨어 취약점 분석 방법 및 분석 장치를 제공하는데 그 목적이 있다.

[0007] 다만, 본 실시예가 이루고자 하는 기술적 과제는 상기된 바와 같은 기술적 과제들로 한정되지 않으며, 또 다른 기술적 과제들이 더 존재할 수 있다.

과제의 해결 수단

[0008] 상술한 기술적 과제를 달성하기 위한 기술적 수단으로서, 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 방법은, 임의의 분석대상 소스코드로부터 재사용 소스코드에 해당하는 코드클론을 탐색하는 단계; 상기 탐색된 코드클론 내에서 패치정보를 기초로 취약 지점 및 상기 취약 지점과 관련된 취약 데이터를 검출하는 단계; 상기 분석대상 소스코드 내에서 상기 검출된 취약 데이터를 역추적하여 입력 지점에 해당하는 소스를 획득하는 단계; 및 상기 입력 지점에서 상기 취약 지점까지의 경로를 기초로 컨커릭 테스팅(Concolic testing)을 수행하여 상기 탐색된 코드클론이 상기 분석대상 소스코드 내에서 취약점에 해당하는지 여부를 검증하는 단계를 포함한다.

[0009] 또한, 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 장치는, 임의의 분석대상 소스코드로부터 취약점 분석을 수행하는 프로그램이 저장된 메모리; 상기 프로그램을 실행하는 프로세서를 포함하되, 상기 프로세서는 상기 프로그램 실행에 따라, 임의의 분석대상 소스코드로부터 재사용 소스코드에 해당하는 코드클론을 탐색하는 단계; 상기 탐색된 코드클론 내에서 패치정보를 기초로 취약 지점 및 상기 취약 지점과 관련된 취약 데이터를 검출하는 단계; 상기 분석대상 소스코드 내에서 상기 검출된 취약 데이터를 역추적하여 입력 지점에 해당하는 소스를 획득하는 단계; 및 상기 입력 지점에서 상기 취약 지점까지의 경로를 기초로 컨커릭 테스팅(Concolic testing)을 수행하여 상기 탐색된 코드클론이 상기 분석대상 소스코드 내에서 취약점에 해당하는지 여부를 검증하는 단계를 수행한다.

발명의 효과

[0010] 전술한 본 발명의 과제 해결 수단 중 어느 하나에 의하면, 대용량의 분석대상 소스코드로부터 재사용 소스코드에 해당하는 코드클론 탐색 후 탐색된 코드클론 결과를 취약점 분석에 이용함으로써, 대용량의 분석대상 소스코드의 취약점 분석에 있어 효율성 및 신속성을 높일 수 있다.

[0011] 또한, 본 발명에 의하면, 공격에 의해 사용될 수 있는 입력 지점에 해당하는 소스를 획득하기 위해서, 코드 구조 그래프 기법을 통해 취약지점 및 취약데이터를 기초로 하여 프로그램 경로를 역추적함으로써, 해당 입력지점의 소스 획득에 있어 효율성 및 정확도를 높일 수 있다.

[0012] 또한, 본 발명에 의하면, 획득된 입력지점 소스를 보안 필요조건을 기초로 하여 컨커릭 테스팅을 수행하여 취약점 여부를 검증함으로써, 해당 입력지점의 소스가 실제로 공격에 활용 가능한(exploitable) 지 여부를 신뢰도

높게 판단할 수 있다.

도면의 간단한 설명

[0013] 도 1은 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 장치의 구성도이다.

도 2는 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 방법에 대한 순서도이다.

도 3은 도 2에 도시된 코드클론을 검색하는 세부 과정을 일 예에 따라 나타낸 순서도이다.

도 4는 일 예에 따른 유닛 소스코드를 설명하기 위한 도면이다.

도 5는 일 예에 따른 취약지점 및 취약데이터 역추적 과정을 설명하기 위한 도면이다.

도 6은 일 예에 따른 코드 구조 그래프를 설명하기 위한 도면이다.

도 7은 도 6에 도시된 코드 구조 그래프의 실행 과정을 설명하기 위한 도면이다.

도 8은 일 예에 따른 테스트용 프로그램을 생성하기 위한 보안 필요조건을 설명하기 위한 도면이다.

도 9는 본 발명에서 제안되는 소프트웨어 취약점 분석방법의 전체 과정을 나타낸 도면이다.

도 10a 내지 도 10d는 본 발명에서 제안되는 소프트웨어 취약점 분석방법에 따라, 취약점을 분석한 결과를 설명하기 위한 도면이다.

발명을 실시하기 위한 구체적인 내용

[0014] 아래에서는 첨부한 도면을 참조하여 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자가 용이하게 실시할 수 있도록 본 발명의 실시예를 상세히 설명한다. 그러나 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시예에 한정되지 않는다. 그리고 도면에서 본 발명을 명확하게 설명하기 위해서 설명과 관계없는 부분은 생략하였으며, 명세서 전체를 통하여 유사한 부분에 대해서는 유사한 도면 부호를 붙였다.

[0015] 명세서 전체에서, 어떤 부분이 다른 부분과 "연결"되어 있다고 할 때, 이는 "직접적으로 연결"되어 있는 경우뿐 아니라, 그 중간에 다른 소자를 사이에 두고 "전기적으로 연결"되어 있는 경우도 포함한다. 또한 어떤 부분이 어떤 구성요소를 "포함"한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성요소를 제외하는 것이 아니라 다른 구성요소를 더 포함할 수 있는 것을 의미하며, 하나 또는 그 이상의 다른 특징이나 숫자, 단계, 동작, 구성요소, 부분품 또는 이들을 조합한 것들의 존재 또는 부가 가능성을 미리 배제하지 않는 것으로 이해되어야 한다.

[0016] 이하의 실시예는 본 발명의 이해를 돕기 위한 상세한 설명이며, 본 발명의 권리 범위를 제한하는 것이 아니다. 따라서 본 발명과 동일한 기능을 수행하는 동일 범위의 발명 역시 본 발명의 권리 범위에 속할 것이다.

[0017] 일반적인 소프트웨어 취약점 분석 기법은 분석대상 소스코드의 전체적인 흐름에 대하여 이루어진다. 하지만 분석대상 소스코드가 대용량 소프트웨어일 경우, 전체적인 흐름을 파악하기 어렵고, 분석을 위한 경우의 수가 많아지기 때문에 시간소요 및 전문지식을 가진 인력자원 확보의 문제가 발생할 수 있다.

[0018] 하지만 본 발명의 일 실시예에 따른 소프트웨어 분석방법 및 분석장치는 대용량 분석대상 소스코드의 모든 부분을 커버하기보다, 실제 취약점이 발생할 가능성이 높은 부분(예를 들어, 코드클론)을 우선적으로 탐색하여, 미리 공개된 취약점 정보를 활용하여 분석하는 현실적인 취약점 분석기법이다.

[0019] 이하에서는 첨부된 도면을 통해 본 발명에서 제안하는 소프트웨어 취약점 분석 기술에 대하여 자세히 설명하도록 한다.

[0020] 도 1은 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 장치의 구성도이다.

[0021] 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 장치(100)는 메모리(110), 프로세서(120), 통신부(130), 취약점데이터베이스(132), 데이터베이스(140), 표시장치(150)을 포함한다. 다만 이러한 도 1의 소프트웨어 취약점 분석장치(100)는 본 발명의 일 실시예에 불과함으로 도 1을 통해 본 발명이 한정 해석되는 것은 아니다. 즉, 본 발명의 다양한 실시예들에 따르면 소프트웨어 취약점 분석장치(100)는 사용자 인터페이스장치 등과 같은 구성을 더 포함하여 도 1과 다르게 구성될 수 있다.

[0022] 또한, 소프트웨어 취약점 분석장치(100)는 다양한 종류의 휴대용 단말기 또는 컴퓨터로 구현될 수 있다. 여기

서, 휴대용 단말기는 예를 들어, 휴대성과 이동성이 보장되는 무선 통신 장치로서, PCS(Personal Communication System), GSM(Global System for Mobile communications), PDC(Personal Digital Cellular), PHS(Personal Handyphone System), PDA(Personal Digital Assistant), IMT(International Mobile Telecommunication)-2000, CDMA(Code Division Multiple Access)-2000, W-CDMA(W-Code Division Multiple Access), WiBro(Wireless Broadband Internet) 단말, 스마트 폰(Smart Phone) 등과 같은 모든 종류의 핸드헬드(Handheld) 기반의 무선 통신 장치를 포함할 수 있다. 또한, 컴퓨터는 예를 들어, 웹 브라우저(WEB Browser)가 탑재된 노트북, 데스크톱(desktop), 랩톱(laptop), 테블릿 PC, 슬레이트 PC 등을 포함할 수 있다.

[0023] 도 1을 통해 메모리(110)는 임의의 분석대상 소스코드로부터 취약점 분석을 수행하는 프로그램(이하, '취약점 분석 프로그램' 이라 지칭함)을 저장한다.

[0024] 메모리(110)는 해당 프로그램뿐만 아니라 다양한 프로그램도 함께 저장할 수 있다. 메모리(110)는 캐쉬, ROM(Read Only Memory), PROM(Programmable ROM), EPROM(Erasable Programmable ROM), EEPROM(Electrically Erasable Programmable ROM) 및 플래쉬 메모리(Flash memory)와 같은 비휘발성 메모리 소자 또는 RAM(Random Access Memory)과 같은 휘발성 메모리 소자 또는 하드디스크 드라이브(HDD, Hard Disk Drive), CD-ROM과 같은 저장 매체 중 적어도 하나로 구현될 수 있으나 이에 한정되지는 않는다.

[0025] 프로세서(120)는 메모리(110)에 저장된 취약점 분석 프로그램을 실행하여 분석대상 소스코드의 취약점을 검출한다.

[0026] 본 발명의 일 실시예에 따르면, 프로세서(120)는 취약점 분석 프로그램 실행에 따라, 임의의 분석대상 소스코드로부터 재사용 소스코드에 해당하는 코드클론을 탐색하는 단계, 탐색된 코드클론 내에서 패치정보를 기초로 취약 지점 및 취약 지점과 관련된 취약 데이터를 검출하는 단계, 분석대상 소스코드 내에서 검출된 취약 데이터를 역추적하여 입력 지점에 해당하는 소스를 획득하는 단계 및 입력 지점에서 취약 지점까지의 경로를 기초로 컨커릭 테스팅(Concolic testing)을 수행하여 탐색된 코드클론이 분석대상 소스코드 내에서 취약점에 해당하는지 여부를 검증하는 단계를 수행한다.

[0027] 통신부(130)은 프로세서 처리과정에 있어 필요한 정보를 취약점 데이터베이스(132)로부터 수신받을 수 있다. 또한, 통신부(130)는 네트워크 망을 통해 외부 장치들 및 기타 서버들과도 통신할 수 있다.

[0028] 취약점 데이터베이스(132)는 임의의 분석대상 소스코드, 패치 정보, 임의의 분석대상의 패치되기 전 초기 소스코드(Original source code) 정보 및 공격에 활용될 수 있는 취약점 정보를 저장할 수 있다. 이때, 저장된 정보 또는 패치 정보는 소프트웨어 취약점 분석장치(100)에 제공될 수 있고, 취약점 데이터베이스(132)는 도 1에 도시된 것과 달리 소프트웨어 취약점 분석장치(100)의 일 구성일 수도 있다.

[0029] 일 예에 따른 취약점 데이터베이스(132)는 분석대상의 소스코드, 분석대상의 초기 소스코드, 패치 정보, 취약점 정보를 저장할 수 있다.

[0030] 분석대상의 초기 소스코드는 분석대상 소스코드의 패치되기 이전의 원래의 소스코드를 의미한다. 분석대상의 초기 소스코드는 취약점 데이터베이스를 통해 직접적으로 제공될 수 있거나, 패치 이력 정보를 기초로 하여 분석대상 소스코드로 추출하여 제공될 수도 있다.

[0031] 패치 정보는 패치 버전 및 패치 이력 정보를 포함할 수 있다.

[0032] 예를 들어, 패치 버전 정보 및 패치 이력 정보를 통해 임의의 분석대상 소스코드의 패치 적용 여부가 확인될 수 있다. 이때, 소프트웨어 취약점 분석장치(100)는 패치 정보로부터 취약점 분석에 필요한 일부 정보를 추출하는 별도의 장치를 포함할 수 있다.

[0033] 취약점 정보는 분석대상 소스코드의 기공개된 취약점 정보를 포함할 수 있다.

[0034] 예를 들어, 취약점 정보는 검출된 코드클론 중 공격에 활용가능한 취약점을 가지고있는 코드클론을 압축적으로 찾아내기 위해 사용되는 정보로써, 후술할 취약지점 및 취약데이터를 검출하는 단계를 수행시 입력지점의 소스를 찾기 위한 참고데이터로 사용될 수 있다.

[0035] 데이터베이스(140)는 상술된 프로세서(120)의 프로그램 실행에 따라 코드클론을 탐색하는 단계를 수행시, 블룸 필터(Bloom filter)를 통해 탐색된 코드클론을 저장한다. 데이터베이스(140)는 코드클론뿐만 아니라 프로세서 처리과정에 필요한 다양한 기타 데이터가 저장될 수 있다.

[0036] 표시장치(150)는 상술된 프로세서(120)의 프로그램 실행에 따라 컨커릭 테스팅을 수행하는 단계의 테스팅 결과

를 화면상에 표시한다. 사용자는 이를 통해 코드클론 탐색 결과, 컨커릭 테스팅 검출결과(예를 들어, 오탐확률)를 확인할 수 있다.

[0037] 이하에서는 도 2 내지 도 8을 통해 본 발명의 일 실시예에 따른 취약점 분석방법 및 분석장치에 대해서 구체적으로 설명한다.

[0038] 도 2는 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 방법에 대한 순서도이다. 본 발명의 일 실시예에 따른 소프트웨어 취약점 분석 방법은 코드클론을 탐색하는 단계(S210), 취약 지점 및 취약 데이터를 검출하는 단계(S220), 취약데이터로부터 역추적 및 입력 지점에 해당하는 소스를 획득하는 단계(S230), 보안 필요조건을 기초로 테스트용 프로그램을 생성하는 단계(S240) 및 컨커릭테스팅을 수행하는 단계(S250)를 포함한다.

[0039] S210단계는 임의의 분석대상 소스코드로부터 재사용 소스코드에 해당하는 코드클론을 탐색한다.

[0040] 일 예에 따르면, S210단계는 분석대상 소스코드와 해당 분석대상 소스코드의 초기 소스코드를 직접적으로 비교하여 코드클론을 탐색할 수 있다. 이때, S210 단계에서 분석대상 소스코드 및 해당 분석대상 소스코드의 초기 소스코드가 취약점 데이터베이스(132)로부터 제공될 수 있다.

[0041] 다른 예에 따르면, S210단계는 분석대상 소스코드와 패치 정보를 통해 추출된 분석대상의 초기 소스코드를 활용하여 코드클론을 탐색할 수 있다. 이때, S210 단계에서 분석대상 소스코드 및 패치 정보가 취약점 데이터베이스(132)로부터 제공될 수 있다.

[0042] 도 3은 도 2에 도시된 코드클론을 탐색하는 세부 과정을 일 예에 따라 나타낸 순서도이다.

[0043] 도 3을 자세히 살펴보면, 코드클론을 탐색하는 단계(S210)는 정규화하는 단계(S310), 토큰화하는 단계(S320), 유닛소스코드를 구성하는 단계(S330), 코드클론을 판단하는 단계(S340)를 포함할 수 있다. S310단계는 분석대상 소스코드 및 분석대상의 초기 소스코드를 정규화한다.

[0044] 예를 들어, 분석대상 소스코드 내에 존재하는 비아스키 문자(non-ASCII character), 불필요하게 덧붙여진 빈공간(Redundant Whitespaces), 소문자(lower cases)와 괄호(braces)로 변환된 모든 문자를 제거함으로써 임의의 분석대상 소스코드를 정규화할 수 있다.

[0045] 다음으로, S320단계는 정규화된 분석대상 소스코드 및 정규화된 초기 소스코드를 각각 코딩 라인단위로 토큰화한다.

[0046] 다음으로, S330단계는 토큰화된 분석대상 소스코드를 순차적으로 일정 개수의 코딩 라인마다 묶어서 복수의 유닛 소스코드를 구성한다. 예를 들어 n길이의 윈도우 슬라이딩을 통해 n-token화(n 길이의 토큰화) 함으로써, 복수의 유닛코드를 구성 할 수 있다.

[0047] 도 4는 일 예에 따른 유닛 소스코드를 설명하기 위한 도면이다. 도 4를 자세히 살펴보면, 토큰화되어 있는 토큰파일(400)의 코딩라인의 총 길이(410)를 길이가 4인 윈도우 슬라이딩(window siliding)(412)을 사용하여, 4-token의 복수의 유닛 소스 코드(412)를 구성하고 있다.

[0048] 구체적으로, 복수의 유닛 소스코드를 구성하기 위해서, S320단계를 통해 생성된 총길이가 l 인 토큰화 파일의 파일 $\mathrm{f}(t_1,t_2,t_3,...,t_l)$ 을, 길이가 n 인 윈도우 슬라이딩(window silding)을 사용하여 n-token(재토큰화)하면, 연관관계에 따라 다음의 식 $x = l - n$ 가 구해지며, 복수의 유닛소스(재토큰화파일) $\mathrm{f}(u_1,u_2,u_3,...,u_x)$ 를 구성할 수 있다.

[0049] 다음으로 S340단계는 복수의 유닛 소스코드 중 토큰화된 초기 소스코드를 포함하는 소정의 유닛 소스코드를 상기 코드클론을 포함하는 유닛 소스코드로 판단하는 단계를 포함한다.

[0050] 구체적으로, 기본적으로 '+' 의 prefix를 제거하고, '-' 의 prefix를 추가한 형태의 소정의 유닛 소스코드 f_v 가 만약 토큰화된 분석대상의 초기 소스코드 f (Target source code pool)안에 포함되면, 코드클론이 존재하는 것으로 판단한다.

[0051] 다시 설명하면, 분석대상의 초기 소스코드 f (Target source code pool)내에 n-token 파일

$f(u_1,u_2,u_3,...,u_x)$ 중 소정의 유닛 소스코드 f_v 가 존재하면, 이때의 f_v 를 코드클론이라 판단할 수 있으며, f_v 가 $S=\{u_1,u_2,u_3,...,u_x\}$ 에 대응되는 하나의 유닛소스코드라 가정하면, $S_v \subseteq S$ 일때 f_v 가 f 에 포함되는 관계를 정의할 수 있다.

[0052] 한편, 대용량의 분석대상 소스코드와 분석대상 소스코드의 초기 소스코드와의 비교시 코드클론이 빠르게 탐색되도록 블룸필터(Bloom filter)을 사용할 수 있고, 해당 블룸필터에서는 일반적으로 해쉬함수(Hash function)를 활용한다.

[0053] 본 발명의 일 실시예에 따르면, 블룸필터(Bloom Filter)는 분석대상 소스코드의 n-token파일에 대하여, [1,m]의 변수범위를 가진 k 개의 독립적인 해쉬함수(예를 들어, $Hash(u_1), Hash(u_2), ..., Hash(u_x)$)를 적용할 수 있다. 각각의 해쉬 $h(x)=i$ 를 위해, 비트 벡터의 i번째 비트의 벡터값은 1로 설정되었다. 코드클론 검출(membsership checking)을 위해 n-token 데이터 세트에 대하여 k개의 독립적인 해쉬함수가 적용되는 것처럼, f_v 로부터 n-token에 대해 k 개의 해쉬함수가 적용된다.

[0054] 이와 같은 과정을 통해, 블룸필터는 모든 연관비트가 1로 설정되는가 여부를 체크할 수 있고, 적어도 하나의 해쉬비트가 0인 경우에 코드클론이 존재하지 않는다고 판단할 수 있다.

[0055] S340단계를 통해 코드 클론을 탐색하는 S210단계가 종료되면, S220단계를 수행한다.

[0056] S220단계는 탐색된 코드클론 내에서 패치 정보를 기초로 취약지점 및 해당 취약지점과 관련된 모든 취약 데이터를 검출한다.

[0057] 이때, 취약점 정보는 취약점 데이터 베이스로부터 소프트웨어 취약점 장치의 통신부를 통해 수신된 정보일 수 있다. 또한, 취약점 정보는 공격자의 공격을 위해 이미 사용되었던 소스코드(예를 들어, 함수인자, 함수데이터) 및 공격자의 공격에 사용될 수 있다고 예상되는 소스코드(예를 들어, 함수인자, 함수데이터) 중 적어도 하나를 포함할 수 있고, 상술한 취약지점 및 취약데이터를 검출하기위해 참고될 수 있다.

[0058] 여기서, 취약지점(Security sink)은 임의의 분석대상 소스코드 중 잠재적으로 위험한 방식으로 사용될 수 있는 함수인자가 위치한 지점이고, 취약데이터(sensitive data)는 취약지점을 포함하는 연관된 모든 소스코드 블록 또는 연관된 모든 소스코드 블록을 구성하는 모든 함수인자 데이터를 의미한다.

[0059] 예를 들어, 전형적으로 취약지점 및 취약 데이터로는, 메모리 복사(memory copy)시, 메모리 할당(memory allocation)시, 서식 캐릭터(Format String)사용시, 산술작업(Arithmetic operations)시 사용되는 함수인자 및 함수데이터가 포함될 수 있다.

[0060] 구체적으로 취약지점은 분석대상 소스코드 중 메모리 복사(memory copy)시 사용되는 함수인자(argument) (예를 들어, strcpy, memcpy)가 위치한 지점일 수 있으며, 이 지점을 포함하는 연관된 소스코드 블록(취약데이터)이 공격자에 의해 악용될 경우 버퍼 오버플로우와 같은 심각한 보안 문제가 발생시킬 수 있다.

[0061] 또한, 취약지점은 분석대상 소스코드 중 메모리 할당(memory allocation)시 사용되는 함수인자 (예를들어, malloc, alloca) 가 위치한 지점일 수 있으며, 이지점을 포함하는 소스코드 블록(취약데이터)이 공격자에 의해서 악용될 경우, 시스템에 메모리 부족 문제를 발생시킬 수 있다.

[0062] 또한, 취약지점은 분석대상 소스코드 중 서식 캐릭터를 위해 사용되는 함수인자(예를 들어, printf, sprintf)가 위치한 지점일 수 있으며, 이 지점을 포함하는 소스코드 블록(취약데이터)이 공격자에 의해서 악용될 경우, 시스템의 제어 문제를 발생시킬 수 있다.

[0063] 또한, 취약지점은 분석대상 소스코드 중 산술연산을 위해 사용되는 인자(예를들어, 정수)가 위치한 지점일 수 있으며, 이 지점을 포함하는 소스코드 블록(취약데이터)이 공격자에 의해서 악용될 경우, 정수 오버플로, 언더플로, 또는 제로 문제를 발생시킬 수 있다.

[0064] 다시 말해, S220단계는 공격자에 의해 악용될 수 있는 소스코드 함수인자가 위치한 지점(취약지점)과 공격자에 의해 악용될 수 있는 함수인자가 포함된 소스코드 블록(취약데이터)를 검출한다.

[0065] 다음으로 S230단계는 분석대상 소스코드 내에서 검출된 취약 데이터를 역추적하여 입력 지점에 해당하는 소스(최종 함수소스)를 획득한다.

[0066] S230단계는 도 5 내지 도 7을 통해 자세히 설명한다.

[0067] 도 5는 일 예에 따른 취약지점 및 취약데이터 역추적 과정을 설명하기 위한 도면이다. 도 5를 자세히 살펴보면, 취약 지점a(510)로부터 취약 데이터a(510)->b(514)->c(514)의 순서로 역추적하여 입력지점에 해당하는 소스(516)를 획득한다.

[0068] 입력지점(516)은 분석대상 소스코드의 취약지점 및 취약데이터와 관련이 있는 최종함수 시작지점이다. 해당 입력지점(516)의 소스를 획득하기 위해 코드 구조 그래프(CSG: Code Structure Gragh)기법을 적용하여 검출된 취약 데이터를 통해 역추적한다. 코드 구조 그래프(CSG) 기법을 위해 소스코드 실행 순서 정보 및 동시에 실행될 수 없는 동등한 수준의 병렬 코드 블록에 대한 정보가 선행적으로 정의될 수 있다.

[0069] 도 6은 일 예에 따른 코드 구조 그래프를 설명하기 위한 도면이다.

[0070] 도 6을 자세히 살펴보면, 어떠한 소스코드(600) 블록 내에서 소스코드가 실행되는 순서(610)를 확인할 수 있다. 라인 번호를 따라 어떠한 소스코드(600)의 실행 순서를 살펴보면 첫 번째 경로는 (1->3->4->5->7->8)의 순서로 실행되고, 두 번째 경로는 (1->10->12->13->15)의 순서로 실행되는 것을 알 수 있다. 또한, 5번 라인과 10번 라인은 동시에 실행될 수 없는 병렬 코드 블록이라고 할 수 있다.

[0071] 따라서, 본 발명의 일 실시예에 따른 역추적 순서는 (15->13->12->8->7->4->3->1)일 수 있다.

[0072] 도 7은 도 6에 도시된 코드 구조 그래프의 실행 과정을 설명하기 위한 도면이다. 도 7을 자세히 살펴보면, 임의의 분석대상 소스코드(710)은 역추적 코딩(712)을 통해 문자열 복사 함수인자를 사용하는 취약지점(712)으로부터, 실질적으로 복사하고자하는 대상과 연관된 상위 레벨의 매개변수를 포함하는 소스블락(714), 그 다음 상위 레벨의 매개변수를 포함하는 소스블록(716)을 역추적하여 입력지점이 검출 과정을 보여주고 있다.

[0073] 이와 같은 역추적 과정을 수행하는 역추적 코드(720)를 자세히 살펴보면, 취약 지점 및 취약데이터를 나타내는 변수(v)와 코드 구조 그래프 변수(G)를 사용하는 재귀함수(recursive function)를 사용하여, 최종 입력지점의 소스를 찾기 위한 경로(유의미한 스테이트먼트)를 찾기 위한 과정을 while문을 통해 반복 수행하고 있음을 확인할 수 있다.

[0074] S230단계를 통해 얻어진 입력지점에 따른 경로는 S240단계를 통해 컨커릭 테스팅 된다. 다시 말해, S240 단계는 입력지점에서 취약 지점까지의 경로를 기초로 컨커릭 테스팅을 수행하여 탐색된 코드클론이 분석대상 소스코드 내에서 취약점에 해당하는 지 여부를 검증한다.

[0075] 다시 말해, 코드클론 검출결과 내에서 취약지점 및 취약 데이터를 역추적하여 얻어진 입력 지점으로부터의 소스 경로에 따른 경로식을 입력하는 프로그램이 생성되면, 생성된 테스트용 프로그램에 대하여 경로마다 랜덤 초기 입력 값을 적용하는 동적 테스팅을 적용하는 컨커릭 테스팅을 수행한다.

[0076] 예를 들어, probe를 삽입함으로써, 실제 수행을 통해 취약지점으로부터 입력지점 소스까지의 경로(심볼릭 경로)를 추출할 수 있다. 이렇게 추출된 경로에 관한 수식은 해당 취약점이 실제로 공격에 활용 가능한(exploitable) 취약점인지 아니면 단순히 프로그램의 크래쉬를 야기시키는 버그인지를 판단할 수 있게 해준다.

[0077] 이때, 입력 지점에 임의의 초기값을 입력(동적테스팅)하여 컨커릭 테스팅의 결과물을 획득하는 단계를 수행할 수 있다.

[0078] 또한, 컨커릭 테스팅의 결과로부터 분석대상 소스코드의 스테이트먼트가 유의미한 작업 명령으로서 동작하도록 기정의된 보안 필요조건을 만족하는지 여부를 판단하는 단계를 수행할 수 있다.

[0079] 보안 필요조건은 기설정된 보안 취약 함수 파라미터, 메모리 접속 및 정수형 산술 중 적어도 하나와 관련된 것일 수 있다.

[0080] 여기서, 보안 필요조건은 정상적인 소스코드(취약점이 없는 소스코드)에서 반드시 만족해야하는 조건을 의미하고, 분석대상 소스코드가 취약점을 가지는지 여부를 판단하기 위한 하나의 조건으로 사용될 수 있다.

[0081] 도 8은 일 예에 따른 테스트용 프로그램을 생성하기 위한 보안 필요조건을 설명하기 위한 도면이다. 도 8을 자세히 살펴보면, 각 함수 파라미터에 대한 보안 필요조건 사항이 표시되어 있다.

[0082] 예를 들면, 문자열 복사함수strcpy(801), strncpy(802)의 보안 필요조건 사항은 복사 문자열의 길이가 버퍼의 용량을 초과하면 안되며, 문자열 조작함수strcat(803)는 조작 문자열의 길이가 버퍼의 용량을 초과하면 안되고, 문자 출력함수 printf(804)는 format의 개수와 parameter개수에서 format의 개수를 뺀만큼의 개수가 동일하도록 구성되어야한다.

[0083] 지금까지 설명한 취약점 분석방법 및 분석장치를 통해 취약점 분석이 효율적으로 신뢰도 높게 이루어질 수 있다.

[0084] 전술한 본 발명의 과제 해결 수단 중 어느 하나에 의하면, 대용량의 분석대상 소스코드로부터 재사용 소스코드에 해당하는 코드클론 탐색 후 탐색된 코드클론 결과를 취약점 분석에 이용함으로써, 대용량의 분석대상 소스코드의 취약점 분석에 있어 효율성 및 신속성을 높일 수 있다.

[0085] 또한, 본 발명에 의하면, 공격에 의해 사용될 수 있는 입력 지점에 해당하는 소스를 획득하기 위해서, 코드 구조 그래프 기법을 통해 취약지점 및 취약데이터를 기초로 하여 프로그램 경로를 역추적함으로써, 해당 입력지점의 소스 획득에 있어 효율성 및 정확도를 높일 수 있다.

[0086] 또한, 본 발명에 의하면, 획득된 입력지점 소스를 보안 필요조건을 기초로 하여 컨커릭 테스팅을 수행하여 취약점 여부를 검증함으로써, 해당 입력지점의 소스가 실제로 공격에 활용 가능한(exploitable)지 여부를 신뢰도 높게 판단할 수 있다.

[0087] 이하에서는, 실제로 구현된 예를 참고하여 보다 구체적으로 해당 장치를 설명하도록 한다. 다만, 이와 같이 본 발명이 한정되는 것이 아님을 쉽게 이해할 수 있을 것이다.

[0088] 도 9는 본 발명에서 제안되는 소프트웨어 취약점 분석방법의 전체 과정을 나타낸 도면이다. 본 발명에서 일 실시예에 따르는 소프트웨어 취약점 분석장치는 취약점 분석 방법에 따라, 코드 클론 검출부(910), 프로그램생성부(920), 취약점 검증부(930)를 포함하여 구현될 수 있다.

[0089] 코드클론 검출부(910)는 임의의 분석대상의 소스코드와 임의의 분석대상의 초기 소스코드를 비교하여 코드클론을 검출할 수 있다. 이때, 사용되는 초기 소스코드는 취약점 데이터 베이스로부터 직접적으로 제공되거나, 취약점 데이터베이스로부터 제공된 패치 정보를 이용하여 추출될 수도 있다.

[0090] 코드클론을 검출하기 이전에, 임의의 분석대상 소스코드와 분석대상의 초기 소스코드 각각은 일련의 정규화 및 토큰화 과정을 수행할 수 있다.

[0091] 코드클론 검출은, 정규화 및 토근화된 분석대상 소스코드는 해쉬함수로 재구성되고, 블룸필터를 통해 정규화 및 토큰화된 초기소스코드와의 비교(membership checking)를 통해 이루어질 수 있다. 이때, 검출된 코드클론 결과는 별도의 데이터베이스에 저장될 수 있다.

[0092] 프로그램 생성부(920)는 검출된 코드클론 중 취약점 데이터베이스로부터 제공된 취약점 정보를 활용하여, 취약지점 및 취약 데이터를 기초로 입력지점에 해당하는 소스를 획득하기 위한 역추적 프로그램을 생성한다. 이때, 생성된 역추적 프로그램을 통해 입력 지점에 해당하는 소스를 획득할 수 있다.

[0093] 이때, 코드 구조 그래프(Code Structure Gragh) 기법을 적용하여 검출된 취약 데이터를 역추적할 수 있다. 또한, 역추적을 통해 얻어지는 입력지점에 대한 경로 및 취약데이터를 획득할 수 있다..

[0094] 취약점 검증부(930)는 획득된 입력지점에 대 대하여 실제로 취약한지 여부를 검증한다. 즉, 입력지점으로부터 취약지점까지의 경로에 대해 컨커릭 테스팅을 수행하고, 랜덤 초기값 입력(Random initial input)을 적용한 동적 테스팅이 진행된다. 이때, 보안 필요조건은 취약점 여부 판단의 일부 기준으로 사용될 수 있다.

[0095] 테스팅 결과 취약점이 있다고 판단되면, 소프트웨어 취약점 분석장치 내 표시장치를 통해 사용자에게 취약성을 보고한다. 예를 들어, 사용자에게 보고되는 정보는 사용된 패치정보, 검출된 코드클론 목록, 검출된 코드클록의 개수, 검출된 코드클론의 경로 및 컨커릭 테스팅 결과인 오탐율 등일 수 있다.

[0096] 테스팅 결과 취약점이 없다고 판단되면, 취약점 발견시까지 또는 사용자에 의해 정해진 임계값 도달시까지 테스팅을 반복하며, 그래도 취약성이 없다고 판단되면, 다음으로 취약하다고 판단할 수 있는 입력지점에 대하여 테스팅을 수행할 수도 있다.

[0097] 도 10a 내지 10d는 본 발명에서 제안되는 소프트웨어 취약점 분석방법에 따라, 취약점을 분석한 결과를 설명하기 위한 도면이다. 즉, 도 10a 내지 10d 는 소프트웨어 취약점 분석 장치 내 표시장치에 의해 사용자에게 제공

되는 정보의 형태일 수 있다.

[0098] 도 10a는 코드클론 검출 결과를 나타내는 도면으로, 도 10a를 통해 사용자는, 취약점 분석이 진행된 프로그램 목록(Target)과, 취약점 검출에 사용된 패치목록(CVE patches), 분석대상 소스코드목록(Target src pool), 취약점을 검출하기 위해 분석된 분석파일수(# of files), 검출된 코드클론의 수(# of reported code clones), 코드클론 검출시까지 걸린시간(Execution time)을 알 수 있다.

[0099] 예를 들어, 도 10b와 같은 형태로 검출결과 정보가 표시장치에 제공될 수 있는데, 도 10b를 자세히 살펴보면, 취약점 분석이 진행된 프로그램 목록(Target) 중 Src pool-1에 대한 63개의 코드클론 중 실제적으로 공격에 활용가능한 취약점들의 정보를 제공하고 있으며, 각 프로그램 목록 및 목록에 대한 CVE패치 목록, 그리고 각 취약점이 발견되는 경로의 정보등을 파악할 수 있다.

[0100] 10c내지 10 d는 역추적 프로그램 및 동적 테스팅을 적용한 컨커릭 테스팅의 결과를 나타내는 도면이다.

[0101] 10c 내지 10 d를 통해 사용자는, 취약점 분석이 진행된 분석대상의 소스코드 목록에 대한 컨커릭 테스팅 결과를 확인할 수 있다. 이때 제공되는 정보를 통해 사용자는 실제 검출된 코드클론이 실제로 취약성을 갖는지 여부에 따른 오탐의 확률을 확인할 수도 있다.

[0102] 상기 소프트웨어 취약점 분석 방법 및 분석장치는 컴퓨터에 의해 실행되는 프로그램 모듈과 같은 컴퓨터에 의해 실행 가능한 명령어를 포함하는 기록 매체의 형태로도 구현될 수 있다. 컴퓨터 판독 가능 매체는 컴퓨터에 의해 액세스될 수 있는 임의의 가용 매체일 수 있고, 휘발성 및 비휘발성 매체, 분리형 및 비분리형 매체를 모두 포함한다. 또한, 컴퓨터 판독가능 매체는 컴퓨터 저장 매체 및 통신 매체를 모두 포함할 수 있다. 컴퓨터 저장 매체는 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈 또는 기타 데이터와 같은 정보의 저장을 위한 임의의 방법 또는 기술로 구현된 휘발성 및 비휘발성, 분리형 및 비분리형 매체를 모두 포함한다. 통신 매체는 전형적으로 컴퓨터 판독가능 명령어, 데이터 구조, 프로그램 모듈, 또는 반송파와 같은 변조된 데이터 신호의 기타 데이터, 또는 기타 전송 메커니즘을 포함하며, 임의의 정보 전달 매체를 포함한다.

[0103] 전술한 본 발명의 설명은 예시를 위한 것이며, 본 발명이 속하는 기술분야의 통상의 지식을 가진 자는 본 발명의 기술적 사상이나 필수적인 특징을 변경하지 않고서 다른 구체적인 형태로 쉽게 변형이 가능하다는 것을 이해할 수 있을 것이다. 그러므로 이상에서 기술한 실시예들은 모든 면에서 예시적인 것이며 한정적이 아닌 것으로 이해해야만 한다. 예를 들어, 단일형으로 설명되어 있는 각 구성 요소는 분산되어 실시될 수도 있으며, 마찬가지로 분산된 것으로 설명되어 있는 구성 요소들도 결합된 형태로 실시될 수 있다.

[0104] 본 발명의 범위는 상세한 설명보다는 후술하는 특허청구범위에 의하여 나타내어지며, 특허청구범위의 의미 및 범위 그리고 그 균등 개념으로부터 도출되는 모든 변경 또는 변형된 형태가 본 발명의 범위에 포함되는 것으로 해석되어야 한다.

부호의 설명

[0105] 100: 소프트웨어 취약점 분석장치

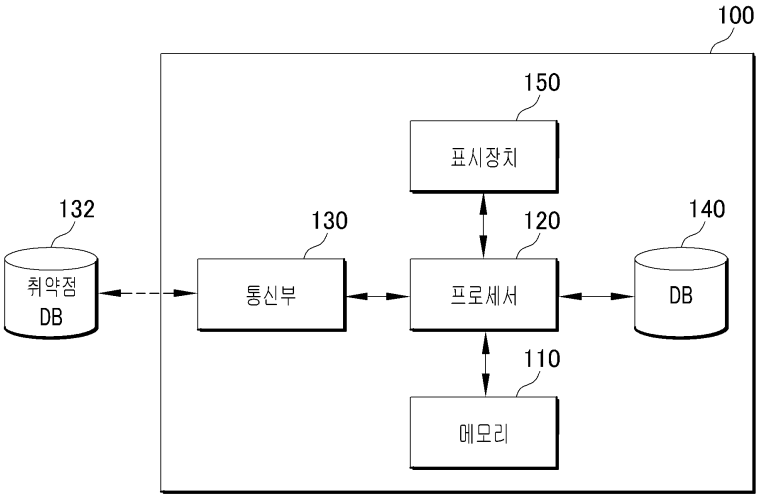
110: 메모리 120: 프로세서

130: 통신부 132: 취약점 데이터베이스

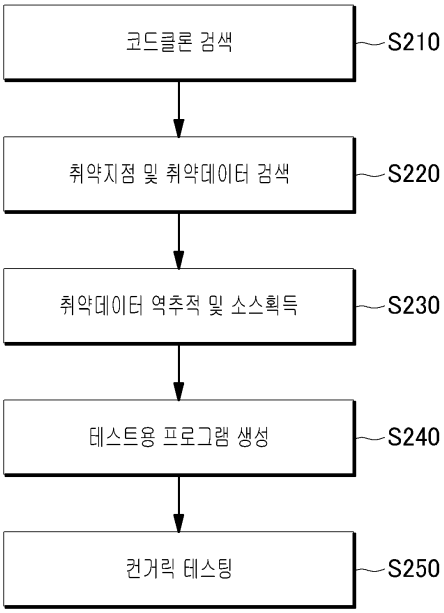
140: 데이터베이스 150: 표시장치

도면

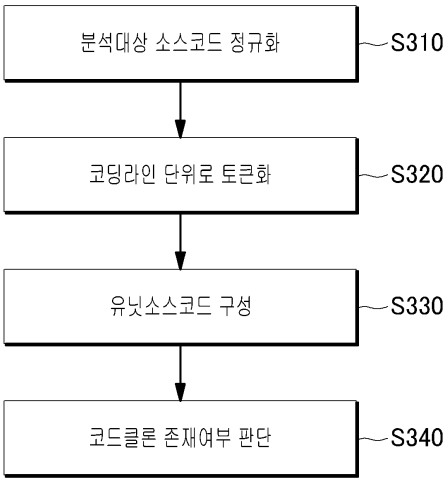
도면1


100
150
표시장치
132
130
120
140
취약점
DB
통신부
프로세서
DB
110
메모리

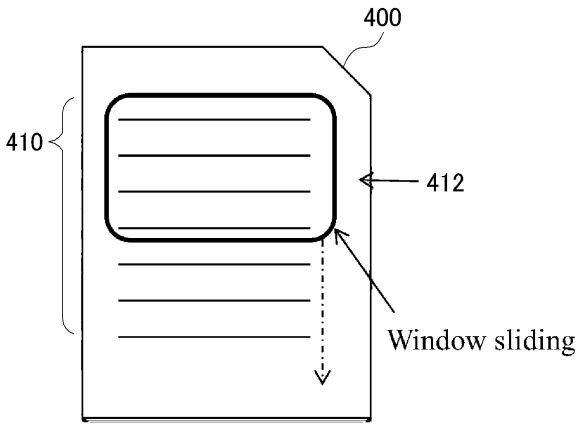
도면2


코드클론 검색
S210
취약지점 및 취약데이터 검색
S220
취약데이터 역추적 및 소스획득
S230
테스트용 프로그램 생성
S240
컨거릭 테스팅
S250

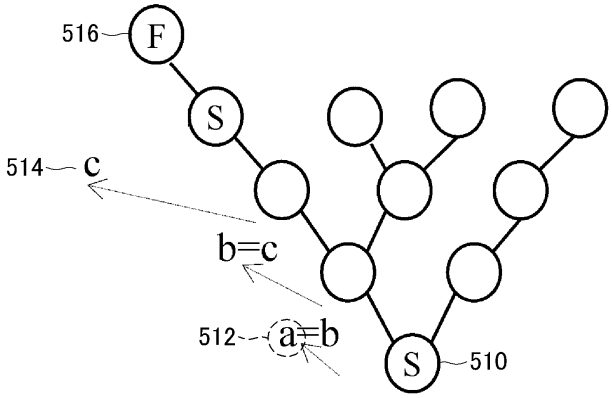
도면3


분석대상 소스코드 정규화
S310
코딩라인 단위로 토큰화
S320
유닛소스코드 구성
S330
코드클론 존재여부 판단
S340

도면4


400
410
412
Window sliding

도면5

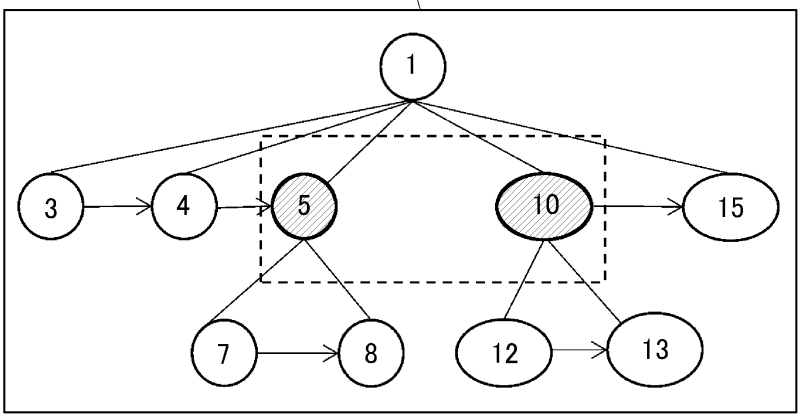


도면6

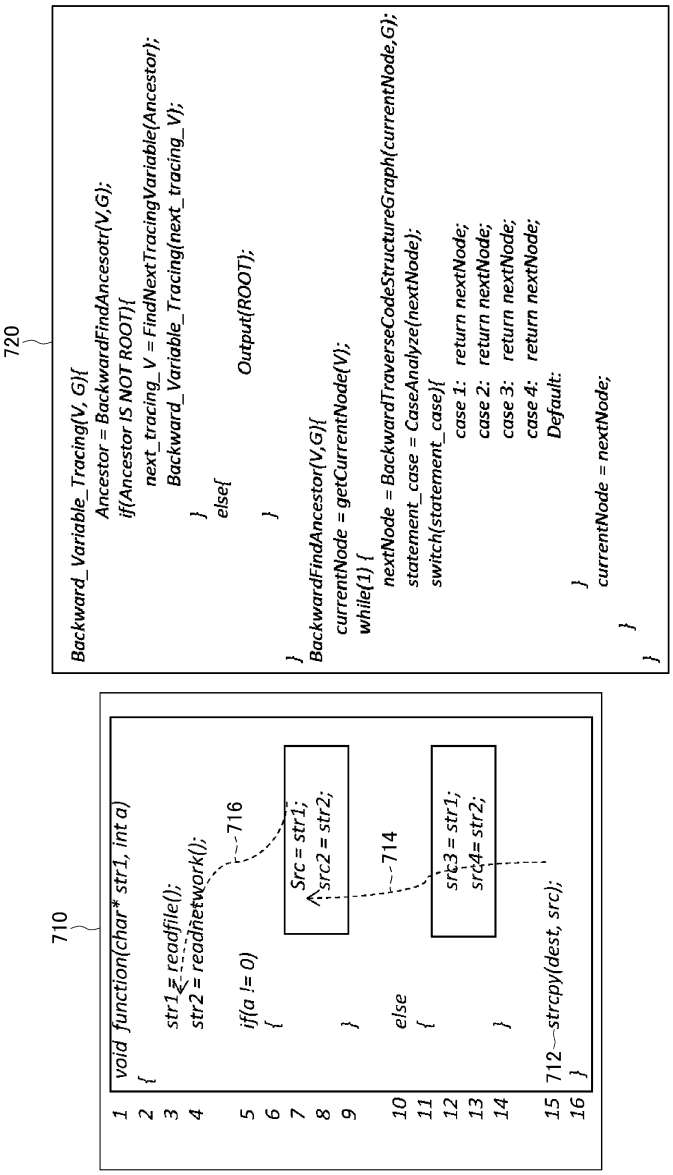
600

```
1   void function(char* str1, int a)
2   {
3           str1 = readfile();
4           str2 = readnetwork();
5           if(a != 0)
6           {
7                   Src = str1;
8                   src2 = str2;
9           }
10          else
11          {
12                  src3 = str1;
13                  src4= str2;
14          }
15          strcpy(dest, src);
16  }
```

610



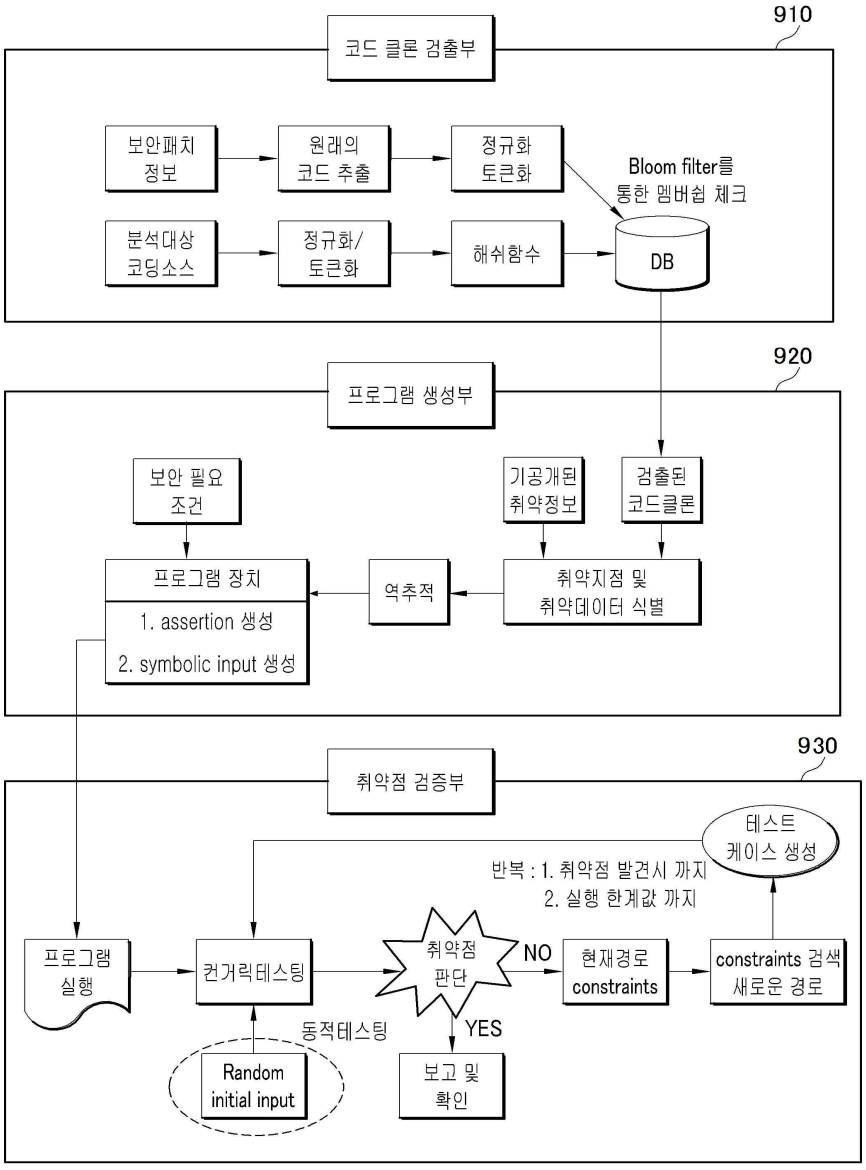
도면7



도면8

보안 취약 함수	보안 필요 조건
strcpy(dst,src)	dst.space > src.strlen
strncpy(dst,src,n)	(dst.space ≥ n) ∧ (n ≥ 0)
strcat(dst,src)	dst.space > dst.strln + src.strlen
printf(format, ...)	# formats = # parameters-1

도면9


910
코드 클론 검출부
보안패치 정보
원래의 코드 추출
정규화 토큰화
Bloom filter를 통한 멤버쉽 체크
분석대상 코딩소스
정규화/ 토큰화
해쉬함수
DB
920
프로그램 생성부
보안 필요 조건
기공개된 취약정보
검출된 코드클론
프로그램 장치
1. assertion 생성
2. symbolic input 생성
역추적
취약지점 및 취약데이터 식별
930
취약점 검증부
테스트 케이스 생성
반복 : 1. 취약점 발견시 까지
2. 실행 한계값 까지
프로그램 실행
컨거릭테스팅
취약점 판단
NO
현재경로 constraints
constraints 검색 새로운 경로
YES
동적테스팅
Random initial input
보고 및 확인

도면10a

Target	CVE patches	Target src pool	# of files	# of reported code clones	Execution time
Src pool-1	CVE patch pool (2010-2014, for C code)	Ubuntu 14.04 OS distribution	259346	63	24812.5 sec (7 hours)
Src pool-2	CVE patch pool (2010-2014, for C code)	Httpd-2.2.23 to 2.4.6	7820	14	738.6 sec (12.31 min)
Src pool-3	CVE patch pool (2010-2014, for C code)	Rsyslog-5.8.13 to 8.2.1	1692	7	274.7 sec (4.57 min)

도면10b

program	CVE patch	Location of the vulnerability
Cmake-2.8.12.2	CVE-2010-0405.patch	/Utilities/cmbzip2/decompress.c:381
Firefox-28.0+build2	CVE-2010-0405.patch	/modules/libbz2/src/decompress.c:381
Thunderbird-24.4.0+build1	CVE-2010-0405.patch	/plugins/pmrfc3164sd/pmrfc3164sd.c:381
rsyslog-7.4.4	CVE-2011-3200.patch	/plugins/pmrfc3164sd/pmrfc3164sd.c:272
gegl-0.2.0	CVE-2012-4433.patch	/operations/external/ppm-load.c:87
linux-3.13(Linux kernel)	CVE-2014-2581.patch	/net/ipv4/ping.c:250
httpd-2.4.7(Apache)	CVE-2011-3368.patch	/server/protocol.c:625

도면10c

Version	# LOC	# of reported code clones	# of vulnerability found/verified	# of false positives
Httpd-2.2.23	350145	1	1	0
Httpd-2.2.24	350256	1	1	0
Httpd-2.3.6	209369	1	1	0
Httpd-2.3.8	210564	1	1	0
Httpd-2.3.11-beta	219427	1	1	0
Httpd-2.3.15-beta	226497	0	0	0
Httpd-2.4.1	223050	1	1	0
Httpd-2.4.2	223265	1	1	0
Httpd-2.4.3	223921	1	1	0
Httpd-2.4.4	226000	1	1	0
Httpd-2.4.6	233330	1	1	0

도면10d

Version	# LOC	# of reported code clones	# of vulnerability found/verified	# of false positives
Rsyslog-5.8.13	78937	1	1	0
Rsyslog-5.10.0	78259	1	1	0
Rsyslog-5.10.1	77811	1	1	0
Rsyslog-6.6.0	92448	1	1	0
Rsyslog-7.4.0	105324	1	1	0
Rsyslog-7.6.3	111218	1	1	0
Rsyslog-8.2.1	112711	1	1	0