(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2017년05월29일
(11) 등록번호 10-1741131
(24) 등록일자 2017년05월23일

(51) 국제특허분류(Int. Cl.)
G06F 11/36 (2006.01) *G06F 11/00* (2017.01)
(52) CPC특허분류
G06F 11/3604 (2013.01)
G06F 11/008 (2013.01)
(21) 출원번호 10-2016-0024583
(22) 출원일자 2016년02월29일
심사청구일자 2016년02월29일
(56) 선행기술조사문헌
"공격 가능한 크래시 탐지를 위한 ARM 바이너리의 정적 분석 방법", 충남대학교 대학원 석사학위논문(2015.02.)*
KR1020100106409 A
*는 심사관에 의하여 인용된 문헌

(73) 특허권자
충남대학교산학협력단
대전광역시 유성구 대학로 99 (궁동, 충남대학교)
(72) 발명자
조은선
대전광역시 유성구 어은로 57, 110동 1504호
전현구
대전광역시 유성구 문화원로 106, 408호
(뒷면에 계속)
(74) 대리인
홍성욱, 심경식

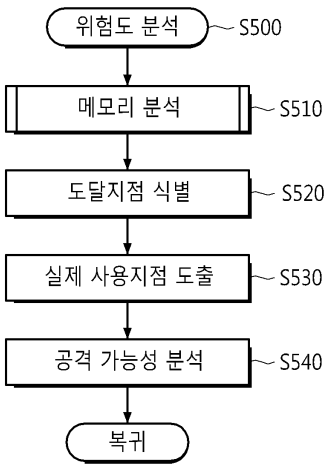
전체 청구항 수 : 총 6 항 심사관 : 이동하

(54) 발명의 명칭 **크래시 위험도 분석 장치 및 그 방법과, 이를 실행하는 프로그램이 기록된 컴퓨터로 읽을 수 있는 기록매체**

(57) 요 약

크래시 위험도 분석 장치를 제공한다. 본 발명의 크래시 위험도 분석 장치는 바이너리 프로그램을 디스어셈블링하여 기계어로 변환하고, 상기 기계어로부터 제어 흐름 정보를 획득하는 디스어셈블러; 상기 기계어를 중간언어로 변환하는 중간언어 변환기; 고의로 크래시를 유발시키기 위해 사전에 제작된 크래시 유발 프로그램에 의해 다수의 크래시들을 발생시키는 크래시 발생기; 및 상기 중간 언어로 변환된 명령어들 중 상기 다수의 크래시들을 발생시킨 명령어들 각각에 의해 영향을 받는 모든 명령어를 정적으로 분석하여 해당 크래시의 위험도를 분석하는 크래시 위험도 분석기를 포함한다.

대 표 도 - 도3



(52) CPC특허분류
G06F 11/3624 (2013.01)
G06F 11/366 (2013.01)

(72) 발명자
목성균
대전광역시 서구 가장로 131-2, 403호
엄기진
대전광역시 서구 용화1길 22
류재철
대전광역시 유성구 어은로 57, 132동 801호

이 발명을 지원한 국가연구개발사업
과제고유번호 1711026515
부처명 미래창조과학부
연구관리전문기관 정보통신기술진흥센터
연구사업명 정보보호핵심원천기술개발사업
연구과제명 안드로이드 신규 취약점 탐지를 위한 모바일 소프트웨어 보안 테스팅 도구 개발
기 여 율 1/1
주관기관 충남대학교 산학협력단
연구기간 2015.03.01 ~ 2016.02.29
공지예외적용 : 있음

명 세 서

청구범위

청구항 1

크래시 위험도 분석 장치에 있어서,

바이너리 프로그램을 디스어셈블링하여 기계어로 변환하고, 상기 기계어로부터 제어 흐름 정보를 획득하는 디스어셈블러;

상기 기계어를 중간언어로 변환하는 중간언어 변환기;

고의로 크래시를 유발시키기 위해 사전에 제작된 크래시 유발 프로그램에 의해 다수의 크래시들을 발생시키는 크래시 발생기; 및

상기 중간 언어로 변환된 명령어들 중 상기 다수의 크래시들을 발생시킨 명령어들 각각에 의해 영향을 받는 모든 명령어를 정적으로 분석하여 해당 크래시의 위험도를 분석하는 크래시 위험도 분석기를 포함하고,

상기 위험도 분석기는

상기 다수의 크래시들을 발생시킨 명령어들 각각에 대하여 상기 명령어들의 도달지점들을 식별하고, 상기 도달지점들 중 해당 명령어가 실제로 사용되는 지점을 찾아낸 후, 그 결과를 사용 지점 그래프로 도출하고,

상기 사용 지점 그래프로부터 프로그램의 제어권을 옮길 수 있는 명령어를 식별한 후, 상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어이거나, 분기(branch) 계열 명령어인 경우 공격 가능성을 분석하고,

상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어인 경우 메모리 영역을 세분화하고, 각 명령어 마다 레지스터 및 메모리에서 가질 수 있는 영역을 분석하는 메모리 분석 과정을 더 수행하고,

상기 메모리 분석 과정을 수행하기 위해, 힙(Heap) 메모리 영역과 스택(Stack) 메모리 영역을 구분하고,

동적 메모리 할당 함수 호출 이후 반환 값이 할당된 힙(Heap) 영역의 주소라는 점을 이용하여 힙(Heap) 메모리를 검출하고, 함수 시작 지점에서 초기화된 레지스터 정보를 이용하여 스택(Stack) 메모리를 검출하는 것을 특징으로 하는 크래시 위험도 분석 장치.

청구항 2

삭제

청구항 3

삭제

청구항 4

제1항에 있어서, 상기 위험도 분석기는

상기 분기(branch) 계열 명령어의 목적 주소(Target Address)가 상기 크래시를 발생시킨 명령어로부터 영향을 받은 경우, 해당 크래시를 공격 가능한 것으로 결정하는 것을 특징으로 하는 크래시 위험도 분석 장치.

청구항 5

삭제

청구항 6

삭제

청구항 7

삭제

청구항 8

크래시 위험도 분석 방법에 있어서,

바이너리 프로그램을 디스어셈블링하여 기계어로 변환하는 단계;

상기 기계어로부터 제어 흐름 정보를 획득하는 단계;

상기 기계어를 중간언어로 변환하는 단계;

고의로 크래시를 유발시키기 위해 사전에 제작된 크래시 유발 프로그램에 의해 다수의 크래시들을 발생시키는 단계; 및

상기 중간 언어로 변환된 명령어들 중 상기 다수의 크래시들을 발생시킨 명령어들 각각에 의해 영향을 받는 모든 명령어를 정적으로 분석하여 해당 크래시의 위험도를 분석하는 단계를 포함하고,

상기 위험도 분석 단계는

상기 다수의 크래시들을 발생시킨 명령어들 각각에 대하여 상기 명령어들의 도달지점들을 식별하는 단계;

상기 도달지점들 중 해당 명령어가 실제로 사용되는 지점을 찾아내고 그 결과를 사용 지점 그래프로 도출하는 단계;

상기 사용 지점 그래프로부터 프로그램의 제어권을 옮길 수 있는 명령어를 식별하여 공격 가능성을 분석하는 단계; 및

상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어인 경우 메모리 영역을 세분화하고, 각 명령어 마다 레지스터 및 메모리에서 가질 수 있는 영역을 분석하는 메모리 분석 단계를 포함하고,

상기 메모리 분석 단계는

힙(Heap) 메모리 영역과 스택(Stack) 메모리 영역을 구분하는 단계를 포함하고,

상기 구분 단계는

동적 메모리 할당 함수 호출 이후 반환 값이 할당된 힙(Heap) 영역의 주소라는 점을 이용하여 힙(Heap) 메모리를 검출하고, 함수 시작 지점에서 초기화된 레지스터 정보를 이용하여 스택(Stack) 메모리를 검출하는 것을 특징으로 하는 크래시 위험도 분석 방법.

청구항 9

삭제

청구항 10

제8항에 있어서, 상기 공격 가능성 분석 단계는

상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어이거나, 분기(branch) 계열 명령어인 경우 공격 가능성을 분석하는 것을 특징으로 하는 크래시 위험도 분석 방법.

청구항 11

제10항에 있어서, 상기 공격 가능성 분석 단계는

상기 분기(branch) 계열 명령어의 목적 주소(Target Address)가 상기 크래시를 발생시킨 명령어로부터 영향을 받은 경우, 해당 크래시를 공격 가능한 것으로 결정하는 것을 특징으로 하는 크래시 위험도 분석 방법.

청구항 12

삭제

청구항 13

삭제

청구항 14

삭제

청구항 15

제8항, 제10항, 제11항 중 어느 한 항에 기재된 방법을 수행하는 프로그램이 기록된 컴퓨터로 읽을 수 있는 기록매체.

발명의 설명

기 술 분 야

[0001] 본 발명은 크래시 분석 장치 및 그 방법에 관한 것으로서, 특히, 프로그램을 정적 분석하여 크래시가 발생한 경우 취약점이 발생할 수 있는 지에 대한 위험도를 판단하는 크래시 분석 장치 및 그 방법에 관한 것이다.

배 경 기 술

[0002] 최근, 스마트 폰, 태블릿 PC와 같은 모바일 기기의 사용이 급격하게 증가하였다. 이와 함께 개인 정보 유출과 같은 모바일 보안 침해로 인한 피해도 급증하고 있다. 모바일 보안 위협에 대한 최근 보고서에 의하면, PC 환경에 비교될 정도로 모바일 기기를 대상으로 한 악성 코드에 의한 침해가 증가하였다. 이러한 보안 위협은 일반적으로 프로그램 취약점을 이용하여, 악성 행위를 유발시킨다. 따라서 이러한 보안 위협을 적절히 대응하기 위해서, 모바일 코드의 취약점 탐지를 위한 분석 기법에 대한 개발이 요구되고 있다.

[0003] 이러한 모바일 코드 분석을 위해서는 몇 가지 제약 사항을 고려해야 한다. 먼저, 일반적으로 모바일 애플리케이션은 바이너리 형태의 실행 파일로 사용자에게 제공되기 때문에, 바이너리 코드를 대상으로 한 분석 기법이 필요하다. 그리고 대부분의 모바일 기기는 x86 프로세서 보다 전력소모가 적은 에이.알.엠(ARM) 프로세서를 이용하기 때문에, ARM 바이너리 코드에 대한 분석 기법이 연구되어야 한다. 하지만, x86 바이너리 코드에 비해서 ARM 바이너리 코드를 대상으로 한 분석 연구는 상대적으로 활발히 진행되지 않았으며, ARM 바이너리 코드 분석에 적합한 분석 도구도 많지 않다. 따라서 ARM 바이너리 코드를 대상으로 한 정적 분석기가 필요하다.

선행기술문헌

특허문헌

[0004] (특허문헌 0001) 대한민국 공개특허 공개번호 10-2014-52200(애플리케이션 검증을 위한 시스템 및 방법, 에스케이플래닛 주식회사, 2014.05.07 공개)

발명의 내용

해결하려는 과제

[0005] 따라서, 본 발명은 바이너리 프로그램을 중간 언어로 변환한 후 정적 분석하여 바이너리 프로그램에 크래시가 발생한 경우 취약점이 발생할 수 있는 지에 대한 위험도를 판단하는 크래시 분석 장치 및 그 방법을 제공하고자 한다.

[0006] 또한, 본 발명은 해커의 공격 패턴에 따라 바이너리 프로그램을 자동 분석하여 해커의 공격 가능성 여부를 탐지하는 크래시 분석 장치 및 그 방법을 제공하고자 한다.

[0007] 또한, 본 발명은 저장 관련 명령어의 경우 메모리를 세분화하여 분석함으로써, 메모리 분석 결과의 정확도를 높이는 크래시 분석 장치 및 그 방법을 제공하고자 한다.

[0008] 또한, 본 발명은 상기 크래시를 자동으로 분석하는 방법을 실행하는 프로그램이 기록된 컴퓨터로 읽을 수 있는 기록 매체를 제공하고자 한다.

과제의 해결 수단

[0009] 상기 목적을 달성하기 위해, 본 발명에서 제공하는 크래시 위험도 분석 장치는 바이너리 프로그램을 디스어셈블링하여 기계어로 변환하고, 상기 기계어로부터 제어 흐름 정보를 획득하는 디스어셈블러; 상기 기계어를 중간언어로 변환하는 중간언어 변환기; 고의로 크래시를 유발시키기 위해 사전에 제작된 크래시 유발 프로그램에 의해 다수의 크래시들을 발생시키는 크래시 발생기; 및 상기 중간 언어로 변환된 명령어들 중 상기 다수의 크래시들을 발생시킨 명령어들 각각에 의해 영향을 받는 모든 명령어를 정적으로 분석하여 해당 크래시의 위험도를 분석하는 크래시 위험도 분석기를 포함한다.

[0010] 바람직하게는, 상기 위험도 분석기는 상기 다수의 크래시들을 발생시킨 명령어들 각각에 대하여 상기 명령어들의 도달지점들을 식별하고, 상기 도달지점들 중 해당 명령어가 실제로 사용되는 지점을 찾아낸 후, 그 결과를 사용 지점 그래프로 도출할 수 있다.

[0011] 바람직하게는, 상기 위험도 분석기는 상기 사용 지점 그래프로부터 프로그램의 제어권을 옮길 수 있는 명령어를 식별한 후, 상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어이거나, 분기(branch) 계열 명령어인 경우 공격 가능성을 분석할 수 있다.

[0012] 바람직하게는, 상기 위험도 분석기는 상기 분기(branch) 계열 명령어의 목적 주소(Target Address)가 상기 크래시를 발생시킨 명령어로부터 영향을 받은 경우, 해당 크래시를 공격 가능한 것으로 결정할 수 있다.

[0013] 바람직하게는, 상기 위험도 분석기는 상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어인 경우 메모리 영역을 세분화하고, 각 명령어 마다 레지스터 및 메모리에서 가질 수 있는 영역을 분석하는 메모리 분석 과정을 더 수행할 수 있다.

[0014] 바람직하게는, 상기 위험도 분석기는 상기 메모리 분석 과정을 수행하기 위해, 힙(Heap) 메모리 영역과 스택(Stack) 메모리 영역을 구분한 후, 상기 스택(Stack) 메모리 영역을 정교화할 수 있다.

[0015] 바람직하게는, 상기 위험도 분석기는 동적 메모리 할당 함수 호출 이후 반환 값이 할당된 힙(Heap) 영역의 주소라는 점을 이용하여 힙(Heap) 메모리를 검출하고, 함수 시작 지점에서 초기화된 레지스터 정보를 이용하여 스택(Stack) 메모리를 검출할 수 있다.

[0016] 한편, 상기 목적을 달성하기 위해, 본 발명에서 제공하는 크래시 위험도 분석 방법은 바이너리 프로그램을 디스어셈블링하여 기계어로 변환하는 단계; 상기 기계어로부터 제어 흐름 정보를 획득하는 단계; 상기 기계어를 중간언어로 변환하는 단계; 고의로 크래시를 유발시키기 위해 사전에 제작된 크래시 유발 프로그램에 의해 다수의 크래시들을 발생시키는 단계; 및 상기 중간 언어로 변환된 명령어들 중 상기 다수의 크래시들을 발생시킨 명령어들 각각에 의해 영향을 받는 모든 명령어를 정적으로 분석하여 해당 크래시의 위험도를 분석하는 단계를 포함한다.

[0017] 바람직하게는, 상기 위험도 분석 단계는 상기 다수의 크래시들을 발생시킨 명령어들 각각에 대하여 상기 명령어들의 도달지점들을 식별하는 단계; 상기 도달지점들 중 해당 명령어가 실제로 사용되는 지점을 찾아내고 그 결과를 사용 지점 그래프로 도출하는 단계; 및 상기 사용 지점 그래프로부터 프로그램의 제어권을 옮길 수 있는 명령어를 식별하여 공격 가능성을 분석하는 단계를 포함할 수 있다.

[0018] 바람직하게는, 상기 공격 가능성 분석 단계는 상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어이거나, 분기(branch) 계열 명령어인 경우 공격 가능성을 분석할 수 있다.

[0019] 바람직하게는, 상기 공격 가능성 분석 단계는 상기 분기(branch) 계열 명령어의 목적 주소(Target Address)가 상기 크래시를 발생시킨 명령어로부터 영향을 받은 경우, 해당 크래시를 공격 가능한 것으로 결정할 수 있다.

[0020] 바람직하게는, 상기 위험도 분석 단계는 상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어인 경우 메

모리 영역을 세분화하고, 각 명령어 마다 레지스터 및 메모리에서 가질 수 있는 영역을 분석하는 메모리 분석 단계를 더 포함할 수 있다.

[0021] 바람직하게는, 상기 메모리 분석 단계는 힙(Heap) 메모리 영역과 스택(Stack) 메모리 영역을 구분하는 단계; 및 상기 스택(Stack) 메모리 영역을 정교화하는 단계를 포함할 수 있다.

[0022] 또한, 상기 목적을 달성하기 위해, 본 발명에서는 상기 기재된 방법 중 하나를 수행하는 프로그램이 기록된 컴퓨터로 읽을 수 있는 기록 매체를 제공한다.

발명의 효과

[0023] 본 발명은 중간 언어를 기반으로 프로그램을 정적 분석함으로써, 바이너리 프로그램에 대한 크래시 분석이 가능하다. 따라서, 바이너리 프로그램에 크래시가 발생한 경우 취약점이 발생할 수 있는 지에 대한 위험도를 자동으로 판단할 수 있다. 특히, 본 발명은 바이너리 프로그램에 크래시가 발생한 경우, 해커들의 공격 패턴에 따라 바이너리 프로그램을 자동 분석하여 해커의 공격 가능성 여부를 판단함으로써, 이를 미리 예방할 수 있다. 결과적으로, 본 발명은 모바일 기기를 대상으로 한 악성 코드에 대한 침해를 효과적으로 줄일 수 있는 장점이 있다. 또한, 본 발명은 저장 관련 명령어의 경우 메모리를 세분화하여 분석함으로써, 메모리 분석 결과의 정확도를 높일 수 있는 장점이 있다.

도면의 간단한 설명

[0024] 도 1은 본 발명의 일 실시 예에 따른 크래시 위험도 분석 장치에 대한 개략적인 블럭도이다.

도 2는 본 발명의 일 실시 예에 따른 크래시 위험도 분석 방법에 대한 개략적인 처리 흐름도이다.

도 3은 도 2의 위험도 분석 과정에 대한 개략적인 처리 흐름도이다.

도 4는 도 3의 메모리 분석 과정을 설명하기 위한 도면이다.

도 5는 도 3의 도달 지점 식별 과정을 설명하기 위한 도면이다.

도 6은 도 3의 실제 사용 지점 도출 과정을 설명하기 위한 도면이다.

도 7은 도 3의 메모리 분석 과정에 대한 개략적인 처리 흐름도이다.

도 8은 도 7의 메모리 영역 구분 과정을 설명하기 위한 도면이다.

도 9 내지 도 11은 도 7의 스택 메모리 영역 정교화 과정을 설명하기 위한 도면이다.

도 12는 본 발명의 일 실시 예에 따른 메모리 영역 분석 결과를 출력한 화면 예를 도시한 도면이다.

발명을 실시하기 위한 구체적인 내용

[0025] 아래에서는 첨부한 도면을 참고로 하여 본 발명의 실시 예에 대하여 설명하되, 본 발명이 속하는 기술 분야에서 통상의 지식을 가진 자가 본 발명을 용이하게 실시할 수 있도록 상세히 설명한다. 그러나 본 발명은 여러 가지 상이한 형태로 구현될 수 있으며 여기에서 설명하는 실시 예에 한정되지 않는다. 한편 도면에서 본 발명을 명확하게 설명하기 위해서 설명과 관계없는 부분은 생략하였으며, 명세서 전체를 통하여 유사한 부분에 대해서는 유사한 도면 부호를 붙였다. 또한 상세한 설명을 생략하여도 본 기술 분야의 당업자가 쉽게 이해할 수 있는 부분의 설명은 생략하였다.

[0026] 명세서 및 청구범위 전체에서, 어떤 부분이 어떤 구성 요소를 포함한다고 할 때, 이는 특별히 반대되는 기재가 없는 한 다른 구성 요소를 제외하는 것이 아니라 다른 구성 요소를 더 포함할 수 있는 것을 의미한다.

[0027] 도 1은 본 발명의 일 실시 예에 따른 크래시 위험도 분석 장치에 대한 개략적인 블록도이다. 도 1을 참조하면, 본 발명의 일 실시 예에 따른 크래시 위험도 분석 장치(100)는 디스어셈블러(110), 중간 언어 변환기(120), 크래시 발생기(130) 및 크래시 위험도 분석기(140)를 포함한다.

[0028] 디스어셈블러(Disassembler)(110)는 입력 데이터를 기계어로 변환한다. 특히, 디스어셈블러(110)는 크래시 위험보 분석 대상인 바이너리 프로그램(binary program)(예컨대, ARM 바이너리 프로그램)을 디스어셈블링(disassembling)하여 기계어로 변환하고, 상기 기계어로부터 획득된 제어 흐름 정보(Control Flow Graph)를 출력한다.

[0029] 중간 언어 변환기(Abstract Interpretation Framework)(120)는 디스어셈블러(110)에서 출력되는 기계어를 중간 언어(Intermediate Language)로 변환한다. 예를 들어, 중간 언어 변환기(120)는 상기 기계어를 REIL(Reverse Engineering Intermediate Language)라는 중간 언어로 변환한다. 이 때, 상기와 같이 기계어를 중간 언어로 변환하는 것은 과다하게 많은 명령어의 수를 줄여서 정적 분석의 편의를 제공하기 위함이다. 예를 들어, 약 300개의 ARM 명령어가 입력된 경우 이를 REIL 이라는 중간 언어로 변환하여 사용할 경우 17개로 줄일 수 있다.

[0030] 크래시 발생기(130)는 고의로 크래시를 유발시킨다. 이를 위해, 크래시 발생기(130)는 다수의 크래시를 고의로 발생시키기 위해 사전에 제작된 크래시 유발 프로그램을 내장하고, 그 크래시 유발 프로그램(예컨대, Fuzzer 프로그램)에 의해 다수의 크래시들을 발생시킨다. 이는 일반적으로 해커들이 퍼저(Fuzzer)와 같은 크래시 유발 프로그램을 이용하여 대상 프로그램에 많은 크래시(crach)를 발생시키고, 그 크래시들을 직접 하나씩 분석한 후 공격 가능성이 있는 경우에 해당 취약점에 맞는 공격 코드를 작성해서 공격하는 공격 패턴을 역 이용하기 위한 것이다. 즉, 상기와 같이 고의로 발생된 크래시들 각각에 대한 공격 가능성(즉, 위험도)을 자동으로 분석하기 위함이다.

[0031] 크래시 위험도 분석기(140)는 중간언어 변환기(120)에서 중간 언어로 변환된 명령어들과, 크래시 발생기(130)에서 발생된 크래시 정보에 기초하여 해당 크래시의 위험도를 분석한다. 즉, 크래시 위험도 분석기(140)는 상기 중간 언어로 변환된 명령어들 중 상기 다수의 크래시들을 발생시킨 명령어들 각각에 의해 영향을 받는 모든 명령어를 정적으로 분석하여 해당 크래시의 위험도를 분석한다. 이를 위해, 위험도 분석기(140)는 상기 다수의 크래시들을 발생시킨 명령어들 각각에 대하여 상기 명령어들의 도달지점(Reaching Definition)들을 식별하고, 상기 도달지점들 중 해당 명령어가 실제로 사용되는 지점(Def-Use Chaining)을 찾아낸 후, 그 결과를 사용 지점 그래프로 도출한다. 그리고, 위험도 분석기(140)는 상기 사용 지점 그래프로부터 프로그램의 제어권을 옮길 수 있는 명령어를 식별한 후, 그 명령어에 대한 공격 가능성을 분석한다. 이 때, 위험도 분석기(140)는 상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어이거나, 분기(branch) 계열 명령어인 경우 공격 가능성을 분석한다. 특히, 위험도 분석기(140)는 상기 분기(branch) 계열 명령어의 목적 주소(Target Address)가 상기 크래시를 발생시킨 명령어로부터 영향을 받은 경우, 해당 크래시를 공격 가능한 것으로 결정한다.

[0032] 한편, 위험도 분석기(140)는 상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어인 경우 메모리 영역을 세분화하고, 각 명령어 마다 레지스터 및 메모리에서 가질 수 있는 영역을 분석하는 메모리 분석 과정을 더 수행한다. 이를 위해, 위험도 분석기(140)는 상기 메모리 분석 과정을 수행하기 위해, 힙(Heap) 메모리 영역과 스택(Stack) 메모리 영역을 구분한 후, 상기 스택(Stack) 메모리 영역을 정교화하되, 동적 메모리 할당 함수 호출 이후 반환 값이 할당된 힙(Heap) 영역의 주소라는 점을 이용하여 힙(Heap) 메모리를 검출하고, 함수 시작 지점에서 초기화된 레지스터 정보를 이용하여 스택(Stack) 메모리를 검출한다.

[0033] 도 2는 본 발명의 일 실시 예에 따른 크래시 위험도 분석 방법에 대한 개략적인 처리 흐름도이다. 도 1 및 도 2를 참조하면, 본 발명의 일 실시 예에 따른 위험도 분석 방법은 다음과 같다.

[0034] 먼저, S100 단계에서는, 디스어셈블러(110)가 바이너리 프로그램을 디스어셈블링하여 기계어로 변환한다. 특히, 디스어셈블러(110)는 크래시 위험보 분석 대상인 바이너리 프로그램(binary program)(예컨대, ARM 바이너리 프로그램)을 디스어셈블링(disassembling)하여 기계어로 변환한다.

[0035] S200 단계에서는, 디스어셈블러(110)가 상기 기계어로부터 제어 흐름 정보를 획득한다. 이 때, 상기 제어 흐름 정보는 이후에 실시되는 위험도 분석 단계(S500)에서 각 명령어들의 도달 지점이 어디까지인지, 또는 실제 사용 지점이 어느 지점인지를 도출하기 위한 참고 자료로 사용될 수 있다.

[0036] S300 단계에서는, 중간 언어 변환기(120)가 상기 기계어를 중간언어로 변환한다. 예를 들어, 중간 언어 변환기(120)는 상기 기계어를 REIL(Reverse Engineering Intermediate Language)라는 중간 언어로 변환한다. 이 때, 상기와 같이 기계어를 중간 언어로 변환하는 것은 과다하게 많은 명령어의 수를 줄여서 정적 분석의 편의를 제공하기 위함이다. 예를 들어, 약 300개의 ARM 명령어가 입력된 경우 이를 REIL 이라는 중간 언어로 변환하여 사용할 경우 17개로 줄일 수 있다.

[0037] S400 단계에서는, 크래시 발생기(130)가 고의로 크래시를 발생시킨다. 이를 위해, 크래시 발생기(130)는 다수의 크래시를 고의로 발생시키기 위해 사전에 제작된 크래시 유발 프로그램을 내장하고, 그 크래시 유발 프로그램(예컨대, Fuzzer 프로그램)에 의해 다수의 크래시들을 발생시킨다. 이는 일반적으로 해커들이 퍼저(Fuzzer)와 같은 크래시 유발 프로그램을 이용하여 대상 프로그램에 많은 크래시(crach)를 발생시키고, 그 크래시들을 직접 하나씩 분석한 후 공격 가능성이 있는 경우에 해당 취약점에 맞는 공격 코드를 작성해서 공격하는 공격 패턴을

역 이용하기 위한 것이다. 즉, 상기와 같이 고의로 발생된 크래시들 각각에 대한 공격 가능성(즉, 위험도)을 자동으로 분석하기 위함이다.

[0038] S500 단계에서는, 크래시 위험도 분석기(140)가 S300 단계에서 중간 언어로 변환된 명령어들과, S400 단계에서 발생된 크래시 정보에 기초하여 해당 크래시의 위험도를 분석한다. 즉, 크래시 위험도 분석기(140)는 상기 중간 언어로 변환된 명령어들 중 상기 다수의 크래시들을 발생시킨 명령어들 각각에 의해 영향을 받는 모든 명령어를 정적으로 분석하여 해당 크래시의 위험도를 분석한다.

[0039] 이 때, S500 단계의 위험도 분석 과정에 대한 보다 상세한 처리 과정의 예가 도 3에 예시되어 있다.

[0040] 도 1 및 도 3을 참조하여, 위험도 분석 과정(S500)을 설명하면 다음과 같다.

[0041] 먼저, S510 단계에서는, 크래시 위험도 분석기(140)가 메모리를 분석한다. 이를 위해, 크래시 위험도 분석기(140)는 메모리 영역을 세분화하고, 각 명령어 마다 레지스터 및 메모리에서 가질 수 있는 영역을 분석하는 일련의 처리과정을 수행하는데, 상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어인 경우에 한하여 상기 S510 단계를 수행한다. 도 4는 도 3의 메모리 분석 과정(S510)을 설명하기 위한 도면으로서, 레지스터와 메모리 영역을 저장하기 위한 구조로서, 각 명령어에서 가질 수 있는 메모리 상태를 저장하는 예를 도시하고 있다. 도 7 내지 도 12는 상기 S510 단계에 대한 보다 상세한 예가 도시되어 있으므로, 도 7 내지 도 12를 참조하여 상기 S510 단계를 다시 설명할 것이다.

[0042] S520 단계에서는, 크래시 위험도 분석기(140)가 상기 다수의 크래시들을 발생시킨 명령어들 각각에 대하여 상기 명령어들의 도달지점(Reaching Definition)들을 식별한다. 이를 위해, S520 단계에서는, Reaching Definition 분석이 적용되며, 상기 Reaching Definition 분석은 변수를 정의하는 명령어가 어디까지 영향을 줄 수 있는가, 즉, 어디까지 도달하는 가를 알아내기 위한 것이다. 도 5는 도 3의 도달 지점 식별 과정(S520)을 설명하기 위한 도면으로서, 상기 Reaching Definition 분석 결과를 블록화하여 나타내고 있다.

[0043] S530 단계에서는, 크래시 위험도 분석기(140)가 상기 도달지점들 중 해당 명령어가 실제로 사용되는 지점을 찾아내고 그 결과를 사용 지점 그래프로 도출한다. 즉, 상기 Reaching Definition 분석 결과로 각 명령어마다 도달한 명령어 리스트가 얻어지면, S530 단계에서는, Def-Use Chaining 분석을 통해, 상기 명령어가 정의한 데이터를 실제 사용하는 지 여부를 결정한다. 이 때, Def-Use Chaining 분석 결과는 그래프 형태로 얻어질 수 있다. 도 6은 상기 실제 사용 지점 도출 과정(S530)을 설명하기 위한 도면으로서, 특정 명령어가 정의한 데이터를 실제 사용하는 명령어를 화살표로 표시하였다.

[0044] 마지막으로, S540 단계에서는, 크래시 위험도 분석기(140)가 상기 사용 지점 그래프로부터 프로그램의 제어권을 옮길 수 있는 명령어를 식별하여 공격 가능성을 분석한다. 특히, 크래시 위험도 분석기(140)는 상기 크래시를 발생시킨 명령어가 저장(store) 계열 명령어이거나, 분기(branch) 계열 명령어인 경우 공격 가능성을 분석한다. 저장(store) 계열 명령어의 경우 직접 실행할 명령어를 가리키고 있는 레지스터의 값을 변경할 수는 없지만, 함수의 리턴(return) 주소나 IAT(Import Address Table), EAT(Export Address Table)에 있는 함수의 주소 등을 변경하여 제어권을 옮길 수 있기 때문이다. 한편, 분기(branch) 계열 명령어는 말 그대로 실행할 명령어를 가리키고 있는 레지스터의 값을 변경하여 실행 흐름을 변경할 수 있기 때문에 제어권을 옮길 수 있다. 이러한 분기(branch) 계열 명령어의 예로 함수콜, 분기, 및 조건 분기 등이 있다.

[0045] 한편, 상기 분기(branch) 계열 명령어의 경우, 목적 주소(Target Address)가 상기 크래시를 발생시킨 명령어로부터 영향을 받으면, 해당 크래시를 공격 가능한 것으로 결정할 수 있다.

[0046] 도 7은 도 3의 메모리 분석 과정에 대한 개략적인 처리 흐름도이다. 도 1 및 도 7을 참조하면, 상기 메모리 분석 단계(S510)는 다음과 같다.

[0047] 먼저, S512 단계에서는, 크래시 위험도 분석기(140)가 힙(Heap) 메모리 영역과 스택(Stack) 메모리 영역을 구분한다. 이를 위해, 크래시 위험도 분석기(140)는 동적 메모리 할당 함수 호출 이후 반환 값이 할당된 힙(Heap) 영역의 주소라는 점을 이용하여 힙(Heap) 메모리를 검출하고, 함수 시작 지점에서 초기화된 레지스터 정보를 이용하여 스택(Stack) 메모리를 검출할 수 있다. 도 8은 도 7의 메모리 영역 구분 과정을 설명하기 위한 도면으로서, 도 8을 참조하면, 로컬 변수인 test에 할당된 heap 영역의 주소를 저장함으로써 스택(Stack) 메모리 영역을 도출하고, *(test+4)인 heap+16 메모리에 0xabcd(10진수로 43981) 값을 저장함으로써 힙(Heap) 메모리 영역을 도출함을 알 수 있다. 이와 같이 스택과 힙 영역이 다르게 설정된 경우 오염된 정보가 서로 전파되지 않음을 알 수 있다.

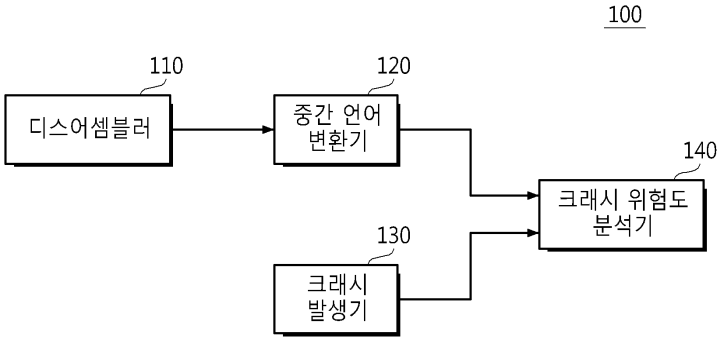
[0048] 한편, S514 단계에서는, 크래시 위험도 분석기(140)가 상기 스택(Stack) 메모리 영역을 정교화한다. 이는 스택의 경우 코드 내용이 매우 짧고, 힙(Heap) 영역에 값을 넣는 간단한 함수에서도 약 12.6:1의 비율로 스택 메모리에 관한 분석이 많기 때문이다. 도 9 내지 도 12는 도 7의 스택 메모리 영역 정교화 과정을 설명하기 위한 도면으로서, 도 9는 구분된 스택(Stack) 메모리 영역(10)과, 힙(Heap) 메모리 영역(20)을 나타내고 있으며, 특히, 스택(Stack) 메모리 영역(10)이 특정 지점1((R1)(11), 특정 지점 2(R2)(12))와 같이 정교화되었음을 나타내고 있다. 도 10은 도 9에 예시된 바와 같이 정교화된 스택(Stack) 메모리 영역(10)의 특정 지점 1(R1)(11)이 오염된(tainted) 경우, 힙(Heap) 메모리 영역(20)은 물론이고 특정 지점2(R2)(12)까지 전파되지 않았음을 알 수 있다. 이와 같이 본 발명에서는 메모리를 구분하고 정교화하여 분석함으로써, 메모리를 하나의 커다란 저장공간으로 보고 분석을 실시하였던 종래의 경우 보다 정확한 분석 결과를 얻을 수 있다. 도 11은 상기 정교화 과정 결과를 메모리 로케이션 테이블(Memory location Table)로 나타난 예를 도시하고 있다. 한편, 도 12는 본 발명의 일 실시 예에 따른 메모리 영역 분석 결과를 출력한 화면 예를 도시한 도면이다.

[0049] 상술한 예시적인 시스템에서, 방법들은 일련의 단계 또는 블록으로써 순서도를 기초로 설명되고 있지만, 본 발명은 단계들의 순서에 한정되는 것은 아니며, 어떤 단계는 상술한 바와 다른 단계와 다른 순서로 또는 동시에 발생할 수 있다.

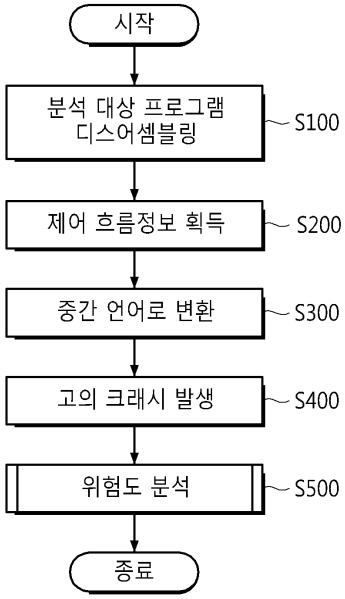
[0050] 또한, 당업자라면 순서도에 나타낸 단계들이 배타적이지 않고, 다른 단계가 포함되거나 순서도의 하나 또는 그 이상의 단계가 본 발명의 범위에 영향을 미치지 않고 삭제될 수 있음을 이해할 수 있을 것이다.

도면

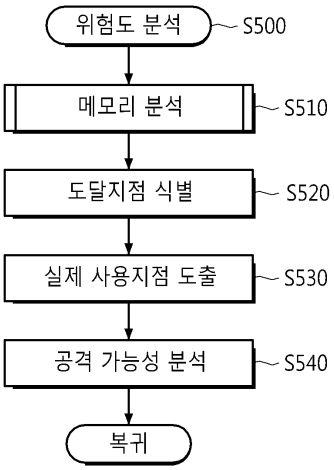
도면1



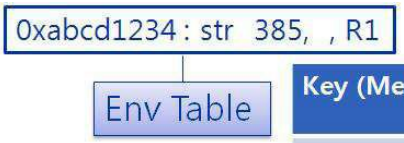
도면2


시작
분석 대상 프로그램 디스어셈블링
S100
제어 흐름정보 획득
S200
중간 언어로 변환
S300
고의 크래시 발생
S400
위험도 분석
S500
종료

도면3

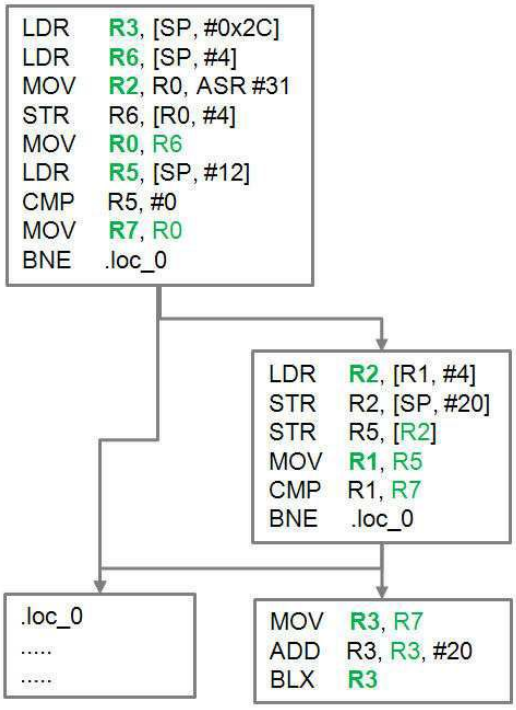

위험도 분석
S500
메모리 분석
S510
도달지점 식별
S520
실제 사용지점 도출
S530
공격 가능성 분석
S540
복귀

도면4

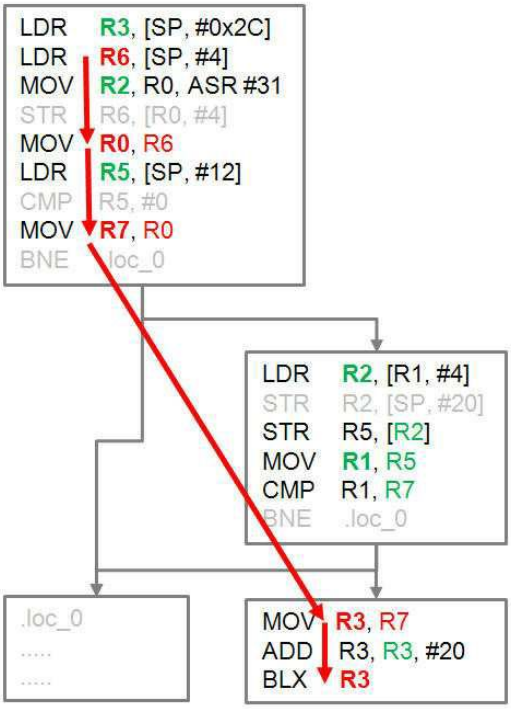


Key (Memory Location)	Value (Memory Location + constant Value)
R1	stack +8
	385
	stack -4
SP	stack -12
LR	retAddr
stack -12	7450
heap +0	1234
...	...

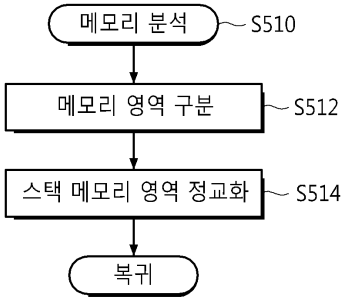
도면5



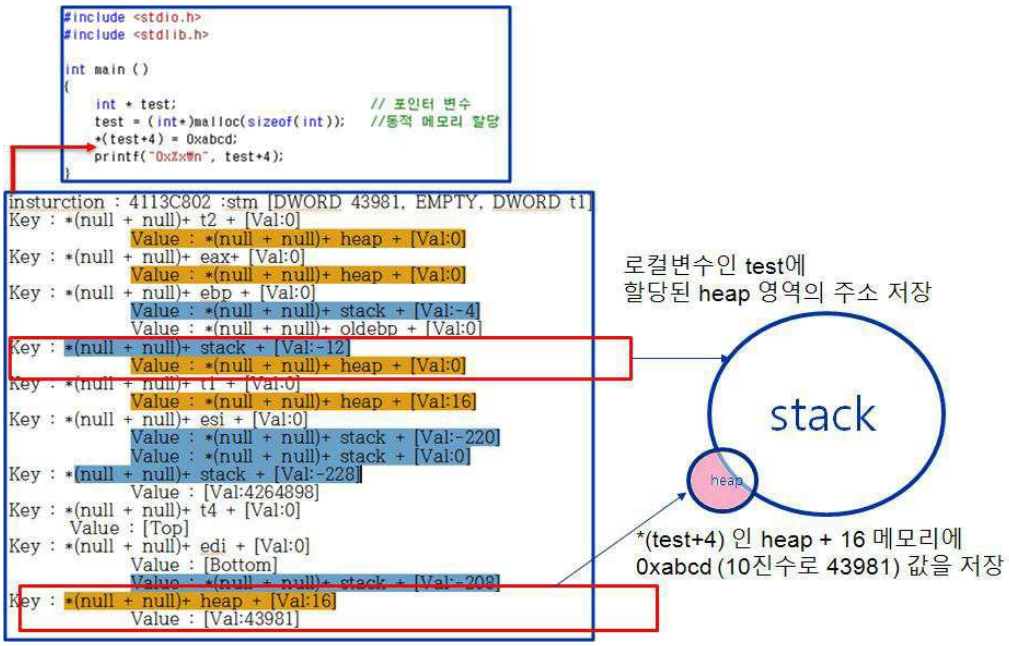
도면6


LDR R3, [SP, #0x2C]
LDR R6, [SP, #4]
MOV R2, R0, ASR #31
STR R6, [R0, #4]
MOV R0, R6
LDR R5, [SP, #12]
CMP R5, #0
MOV R7, R0
BNE .loc_0
LDR R2, [R1, #4]
STR R2, [SP, #20]
STR R5, [R2]
MOV R1, R5
CMP R1, R7
BNE .loc_0
.loc_0
.....
.....
MOV R3, R7
ADD R3, R3, #20
BLX R3

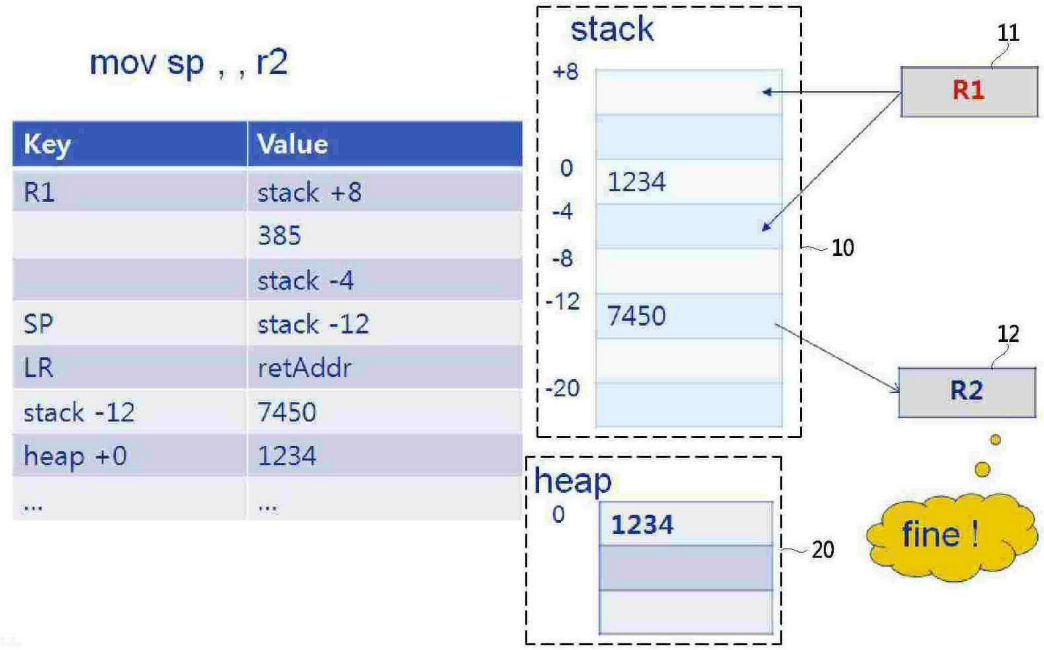
도면7


메모리 분석
S510
메모리 영역 구분
S512
스택 메모리 영역 정교화
S514
복귀

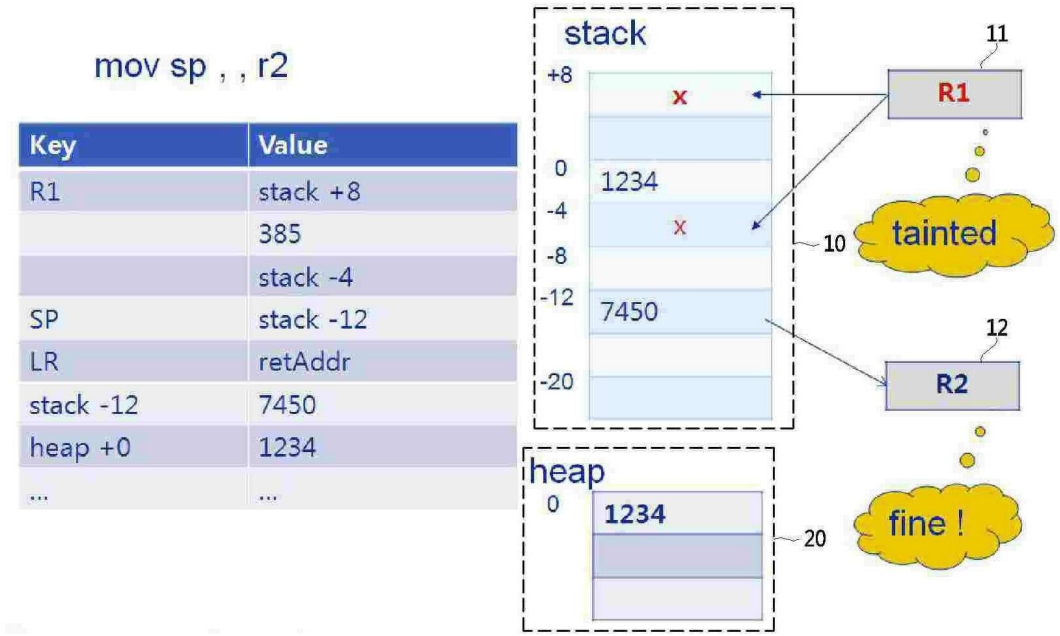
도면8


#include <stdio.h>
#include <stdlib.h>
int main ()
{
int * test; // 포인터 변수
test = (int*)malloc(sizeof(int)); //동적 메모리 할당
*(test+4) = 0xabcd;
printf("0x%x\n", test+4);
}
insturction : 4113C802 :stm [DWORD 43981, EMPTY, DWORD t1]
Key : *(null + null)+ t2 + [Val:0]
Value : *(null + null)+ heap + [Val:0]
Key : *(null + null)+ eax+ [Val:0]
Value : *(null + null)+ heap + [Val:0]
Key : *(null + null)+ ebp + [Val:0]
Value : *(null + null)+ stack + [Val:-4]
Value : *(null + null)+ oldebp + [Val:0]
Key : *(null + null)+ stack + [Val:-12]
Value : *(null + null)+ heap + [Val:0]
Key : *(null + null)+ t1 + [Val:0]
Value : *(null + null)+ heap + [Val:16]
Key : *(null + null)+ esi + [Val:0]
Value : *(null + null)+ stack + [Val:-220]
Value : *(null + null)+ stack + [Val:0]
Key : *(null + null)+ stack + [Val:-228]
Value : [Val:4264898]
Key : *(null + null)+ t4 + [Val:0]
Value : [Top]
Key : *(null + null)+ edi + [Val:0]
Value : [Bottom]
Key : *(null + null)+ heap + [Val:16]
Value : [Val:43981]
로컬변수인 test에
할당된 heap 영역의 주소 저장
stack
heap
*(test+4) 인 heap + 16 메모리에
0xabcd (10진수로 43981) 값을 저장

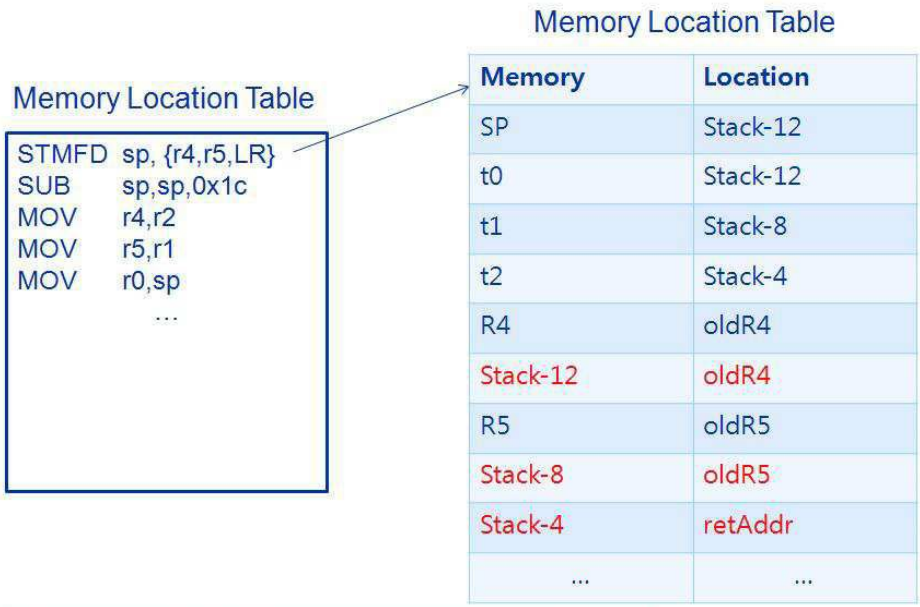
도면9


mov sp , , r2
Key
Value
R1
stack +8
385
stack -4
SP
stack -12
LR
retAddr
stack -12
7450
heap +0
1234
...
...
stack
+8
0
-4
-8
-12
-20
1234
7450
11
R1
10
12
R2
heap
0
1234
20
fine !

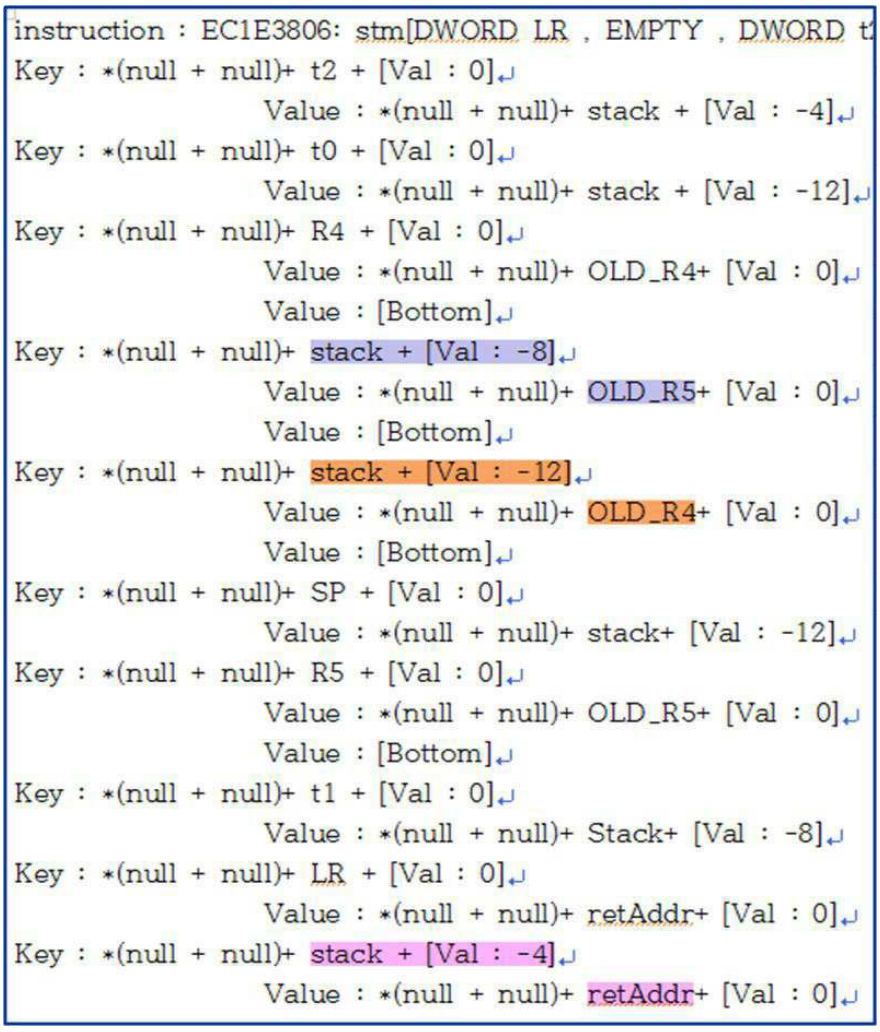
도면10


mov sp , , r2
Key
Value
R1
stack +8
385
stack -4
SP
stack -12
LR
retAddr
stack -12
7450
heap +0
1234
...
stack
+8
x
0
1234
-4
x
-8
-12
7450
-20
heap
0
1234
11
R1
10
tainted
12
R2
20
fine !

도면11


Memory Location Table
STMFD sp, {r4,r5,LR}
SUB sp,sp,0x1c
MOV r4,r2
MOV r5,r1
MOV r0,sp
...
Memory Location Table
Memory
Location
SP
Stack-12
t0
Stack-12
t1
Stack-8
t2
Stack-4
R4
oldR4
Stack-12
oldR4
R5
oldR5
Stack-8
oldR5
Stack-4
retAddr
...
...

도면12


instruction : EC1E3806: stm[DWORD LR , EMPTY , DWORD t
Key : *(null + null)+ t2 + [Val : 0]
Value : *(null + null)+ stack + [Val : -4]
Key : *(null + null)+ t0 + [Val : 0]
Value : *(null + null)+ stack + [Val : -12]
Key : *(null + null)+ R4 + [Val : 0]
Value : *(null + null)+ OLD_R4+ [Val : 0]
Value : [Bottom]
Key : *(null + null)+ stack + [Val : -8]
Value : *(null + null)+ OLD_R5+ [Val : 0]
Value : [Bottom]
Key : *(null + null)+ stack + [Val : -12]
Value : *(null + null)+ OLD_R4+ [Val : 0]
Value : [Bottom]
Key : *(null + null)+ SP + [Val : 0]
Value : *(null + null)+ stack+ [Val : -12]
Key : *(null + null)+ R5 + [Val : 0]
Value : *(null + null)+ OLD_R5+ [Val : 0]
Value : [Bottom]
Key : *(null + null)+ t1 + [Val : 0]
Value : *(null + null)+ Stack+ [Val : -8]
Key : *(null + null)+ LR + [Val : 0]
Value : *(null + null)+ retAddr+ [Val : 0]
Key : *(null + null)+ stack + [Val : -4]
Value : *(null + null)+ retAddr+ [Val : 0]