명세서

발명의 명칭: 프로그램 실행 컨텍스트 기반의 빅데이터를 활용한 역공학 방법 및 시스템

기술분야

[1] 아래의 설명은 큰 사이즈의 바이너리 분석을 보다 효율적으로 하는 방법에 관한 것으로서, 더욱 상세하게는 프로그램 실행 컨텍스트를 모두 저장하고 저장한 컨텍스트를 효율적으로 분석하는 역공학 방법 및 시스템, 그리고 컴퓨터와 결합되어 본 발명의 실시예들에 따른 방법을 컴퓨터에 실행시키기 위해 컴퓨터 판독 가능한 기록매체에 저장된 컴퓨터 프로그램과 그 기록매체에 관한 것이다.

배경기술

[2] 임의의 프로그램(바이너리 혹은 소스코드)이 주어졌을 때, 해당 프로그램의 동작 방식을 분석하고 이해하는 것을 '역공학'이라고 부른다. 완성된 제품(바이너리)을 판매하는 기업의 입장에서는 그 제품에 해당 기업의 기술이 그대로 담겨 있기 때문에 제품의 역공학이 어렵게 되는 것을 원한다. 이와 비슷한 입장에 있는 또 다른 생산자의 예시로는 악성코드를 만들고 배포하는 공격자가 있을 수 있다. 반대로 바이너리만 주어진 상태에서 프로그램을 분석해야만 하는 경우가 있다. 프로그램의 취약점을 찾거나 악성코드의 동작 방식을 이해해야 하는 경우가 이에 해당한다. 이처럼 역공학이라는 주제를 놓고 창과 방패의 싸움이 꾸준히 이어져 오고 있으며 특정 방향이 반드시 옳은 방향이라고 단정지을 수 없는 상황이다.

[3] 본 발명은 역공학 기술 중에서도 '동적 분석'이라고 불리는 분야의 한 부분이다. 기존의 동적 분석 방법들 중 가장 기초가 되는 것은 '디버거'라고 불리는 툴을 활용하는 것이다. 디버거의 장점은 광범위한 환경에서 사용 가능하며 사용자가 원하는 프로그램의 지점에서 프로그램의 실행 컨텍스트를 모두 확인할 수 있다는 것이다. 프로그램은 명령어를 담고 있는 코드와 그 코드의 실행 배경이 되는 실행 컨텍스트를 바탕으로 하여 동작한다. 따라서 사용자는 디버거를 활용하여 원하는 프로그램의 지점에서 다음의 실행결과를 예측할 수 있고 이어 프로그램을 이해하는 부분까지 나아갈 수 있다. 또 다른 방법으로는 정적/동적 바이너리 수정이 있다. 일반적으로 이런 수정법은 함수/베이직블럭/명령어 단위로 사용자가 원하는 기능을 삽입/수정하여 프로그램의 행동을 이해하기 위해 이용될 수 있다.

[4] 앞서 설명한 프로그램의 행동을 이해하기 위해 도움을 주는 분석 방법들은 프로그램이 담고 있는 모든 정보들을 사용자에게 주기에 충분하다. 그러나 주어진 프로그램의 크기가 클수록 사용자가 읽고 처리하기 위해 많은 시간이 걸리며 보다 효율적인 분석 방법을 요구한다. 따라서 디버거나 바이너리 수정을

통해 얻은 정보를 효율적으로 가공하는 것이 무엇보다 중요하다고 할 수 있다. 이 가공법이 얼마나 효과적이며 효율적이냐에 따라 분석자가 프로그램을 분석하기 위해 걸리는 시간이 크게 영향을 받는다. 이에 따라 본 발명에서는 실행 컨텍스트의 저장과 효율적인 가공을 제시하고 이 제안이 얼마나 효율적인지를 보인다.

발명의 상세한 설명

기술적 과제

[5] 바이너리의 크기가 아주 크거나 복잡한 경우에 분석자가 다루어야 하는 정보가 아주 많은 상황에서 분석자가 집중해야 하는 부분을 빠르게 찾아 프로그램의 행동을 이해하는 데에 걸리는 시간을 크게 단축시킬 수 있는 역공학 방법 및 시스템, 그리고 컴퓨터와 결합되어 본 발명의 실시예들에 따른 방법을 컴퓨터에 실행시키기 위해 컴퓨터 판독 가능한 기록매체에 저장된 컴퓨터 프로그램과 그 기록매체를 제공한다.

과제 해결 수단

[6] 분석 대상 프로그램을 디버거를 통해 실행하는 단계; 상기 디버거를 이용하여 상기 분석 대상 프로그램의 관심 영역에 포함된 인스트럭션에 브레이크 포인트를 설정하는 단계; 상기 브레이크 포인트가 설정된 인스트럭션에서 발생하는 이벤트에 대해 이벤트 핸들러를 통해 해당 시점에서의 실행 컨텍스트(execution context)를 컨텍스트 데이터베이스에 저장하는 단계; 및 상기 컨텍스트 데이터베이스에 저장된 데이터를 처리하여 상기 분석 대상 프로그램에 대한 정보 분석 결과를 생성하는 단계를 포함하는 역공학 방법을 제공한다.

[7] 일측에 따르면, 상기 정보 분석 결과를 생성하는 단계는, 상기 컨텍스트 데이터베이스에 저장된 데이터를 가공하는 단계; 및 시각화 도구를 활용하여 상기 가공된 데이터에 대한 정보 분석 결과를 리포트하는 단계를 포함하는 것을 특징으로 할 수 있다.

[8] 다른 측면에 따르면, 상기 컨텍스트 데이터베이스에 저장되는 테이블의 컬럼(column)에 상기 실행 컨텍스트가 저장된 순서를 나타내는 인덱스 값이 포함되는 것을 특징으로 할 수 있다.

[9] 또 다른 측면에 따르면, 상기 실행 컨텍스트는 상기 실행 컨텍스트에 대응하는 시점에서의 상기 분석 대상 프로그램에 대한 레지스터 값 및 메모리 상태 중 적어도 하나를 포함하는 것을 특징으로 할 수 있다.

[10] 컴퓨터 장치와 결합되어 상기 역공학 방법을 컴퓨터 장치에 실행시키기 위해 컴퓨터 판독 가능한 기록매체에 저장된 컴퓨터 프로그램을 제공한다.

[11] 상기 역공학 방법을 컴퓨터 장치에 실행시키기 위한 프로그램이 기록되어 있는 컴퓨터에서 판독 가능한 기록매체를 제공한다.

[12] 컴퓨터 장치에 있어서, 상기 컴퓨터 장치에서 판독 가능한 명령을 실행하도록

구현되는 적어도 하나의 프로세서를 포함하고, 상기 적어도 하나의 프로세서에 의해, 분석 대상 프로그램을 디버거를 통해 실행하고, 상기 디버거를 이용하여 상기 분석 대상 프로그램의 관심 영역에 포함된 인스트럭션에 브레이크 포인트를 설정하고, 상기 브레이크 포인트가 설정된 인스트럭션에서 발생하는 이벤트에 대해 이벤트 핸들러를 통해 해당 시점에서의 실행 컨텍스트(execution context)를 컨텍스트 데이터베이스에 저장하고, 상기 컨텍스트 데이터베이스에 저장된 데이터를 처리하여 상기 분석 대상 프로그램에 대한 정보 분석 결과를 생성하는 것을 특징으로 하는 컴퓨터 장치를 제공한다.

발명의 효과

[13] 바이너리의 분석자가 바이너리의 구조에 대한 이해가 아주 적은 경우에도 프로그램의 구조 분석을 하기 위한 시작점을 찾는 것을 용이하게 할 수 있다. 이는 프로그램 분석에 걸리는 시간을 단축시켜주는 효과를 제공한다.

도면의 간단한 설명

[14] 도 1은 본 발명의 일실시예에 따른 컴퓨터 장치의 예를 도시한 블록도이다.

[15] 도 2는 본 발명의 일실시예에 따른 역공학 과정의 예를 도시한 도면이다.

[16] 도 3은 본 발명의 일실시예에 있어서, 프로그램의 모든 명령어 지점에 브레이크포인트를 설정하고 해당 지점에서의 pc 값을 순차적으로 나타낸 그래프의 예를 도시한 도면이다.

[17] 도 4는 본 발명의 일실시예에 있어서, 문서 작성 프로그램의 화면 예를 도시한 도면이다.

[18] 도 5는 본 발명의 일실시예에 있어서, 메모리 동적 할당자 함수의 에필로그 부분의 예를 도시한 도면이다.

[19] 도 6은 에필로그 부분에 대응하는 지점에서의 esi 레지스터 값들을 도시한 그래프의 예를 도시한 도면이다.

[20] 도 7은 본 발명의 일실시예에 있어서, 문서 작성 프로그램의 화면 예를 도시한 도면이다.

[21] 도 8은 본 발명의 일실시예에 있어서, 메모리 동적 할당자 함수의 에필로그 부분에 대응하는 지점에서의 esi 레지스터 값들을 도시한 그래프의 다른 예를 도시한 도면이다.

[22] 도 9는 본 발명의 일실시예에 있어서, 메모리 주소의 예를 도시한 도면이다.

[23] 도 10은 본 발명의 일실시예에 따른 역공학 방법의 예를 도시한 흐름도이다.

발명의 실시를 위한 최선의 형태

[24] 이하, 실시예를 첨부한 도면을 참조하여 상세히 설명한다.

[25] 본 발명의 실시예들에 따른 역공학 방법은 이후 설명될 컴퓨터 장치를 통해 구현될 수 있다. 이때, 컴퓨터 장치에는 본 발명의 일실시예에 따른 컴퓨터 프로그램이 설치 및 구동될 수 있고, 컴퓨터 장치는 구동된 컴퓨터 프로그램의 제어에 따라 본 발명의 실시예들에 따른 방법을 수행할 수 있다. 상술한 컴퓨터

프로그램은 컴퓨터 장치와 결합되어 본 발명의 실시예들에 따른 방법을 컴퓨터 장치에 실행시키기 위해 컴퓨터 판독 가능한 기록매체에 저장될 수 있다.

[26] 도 1은 본 발명의 일실시예에 따른 컴퓨터 장치의 예를 도시한 블록도이다. 이미 설명한 바와 같이, 본 발명의 실시예들에 따른 역공학 방법은 도 1을 통해 도시된 컴퓨터 장치(100)에 의해 실행될 수 있다.

[27] 이러한 컴퓨터 장치(100)는 도 1에 도시된 바와 같이, 메모리(110), 프로세서(120), 통신 인터페이스(130) 그리고 입출력 인터페이스(140)를 포함할 수 있다. 메모리(110)는 컴퓨터에서 판독 가능한 기록매체로서, RAM(random access memory), ROM(read only memory) 및 디스크 드라이브와 같은 비소멸성 대용량 기록장치(permanent mass storage device)를 포함할 수 있다. 여기서 ROM과 디스크 드라이브와 같은 비소멸성 대용량 기록장치는 메모리(110)와는 구분되는 별도의 영구 저장 장치로서 컴퓨터 장치(100)에 포함될 수도 있다. 또한, 메모리(110)에는 운영체제와 적어도 하나의 프로그램 코드가 저장될 수 있다. 이러한 소프트웨어 구성요소들은 메모리(110)와는 별도의 컴퓨터에서 판독 가능한 기록매체로부터 메모리(110)로 로딩될 수 있다. 이러한 별도의 컴퓨터에서 판독 가능한 기록매체는 플로피 드라이브, 디스크, 테이프, DVD/CD-ROM 드라이브, 메모리 카드 등의 컴퓨터에서 판독 가능한 기록매체를 포함할 수 있다. 다른 실시예에서 소프트웨어 구성요소들은 컴퓨터에서 판독 가능한 기록매체가 아닌 통신 인터페이스(130)를 통해 메모리(110)에 로딩될 수도 있다. 예를 들어, 소프트웨어 구성요소들은 네트워크(160)를 통해 수신되는 파일들에 의해 설치되는 컴퓨터 프로그램에 기반하여 컴퓨터 장치(100)의 메모리(110)에 로딩될 수 있다.

[28] 프로세서(120)는 기본적인 산술, 로직 및 입출력 연산을 수행함으로써, 컴퓨터 프로그램의 명령을 처리하도록 구성될 수 있다. 명령은 메모리(110) 또는 통신 인터페이스(130)에 의해 프로세서(120)로 제공될 수 있다. 예를 들어 프로세서(120)는 메모리(110)와 같은 기록 장치에 저장된 프로그램 코드에 따라 수신되는 명령을 실행하도록 구성될 수 있다.

[29] 통신 인터페이스(130)은 네트워크(160)를 통해 컴퓨터 장치(100)가 다른 장치(일례로, 앞서 설명한 저장 장치들)와 서로 통신하기 위한 기능을 제공할 수 있다. 일례로, 컴퓨터 장치(100)의 프로세서(120)가 메모리(110)와 같은 기록 장치에 저장된 프로그램 코드에 따라 생성한 요청이나 명령, 데이터, 파일 등이 통신 인터페이스(130)의 제어에 따라 네트워크(160)를 통해 다른 장치들로 전달될 수 있다. 역으로, 다른 장치로부터의 신호나 명령, 데이터, 파일 등이 네트워크(160)를 거쳐 컴퓨터 장치(100)의 통신 인터페이스(130)를 통해 컴퓨터 장치(100)로 수신될 수 있다. 통신 인터페이스(130)를 통해 수신된 신호나 명령, 데이터 등은 프로세서(120)나 메모리(110)로 전달될 수 있고, 파일 등은 컴퓨터 장치(100)가 더 포함할 수 있는 저장 매체(상술한 영구 저장 장치)로 저장될 수 있다.

[30] 입출력 인터페이스(140)는 입출력 장치(150)와의 인터페이스를 위한 수단일 수 있다. 예를 들어, 입력 장치는 마이크, 키보드 또는 마우스 등의 장치를, 그리고 출력 장치는 디스플레이, 스피커와 같은 장치를 포함할 수 있다. 다른 예로 입출력 인터페이스(140)는 터치스크린과 같이 입력과 출력을 위한 기능이 하나로 통합된 장치와의 인터페이스를 위한 수단일 수도 있다. 입출력 장치(150)는 컴퓨터 장치(100)와 하나의 장치로 구성될 수도 있다.

[31] 또한, 다른 실시예들에서 컴퓨터 장치(100)는 도 1의 구성요소들보다 더 적은 혹은 더 많은 구성요소들을 포함할 수도 있다. 그러나, 대부분의 종래기술적 구성요소들을 명확하게 도시할 필요성은 없다. 예를 들어, 컴퓨터 장치(100)는 상술한 입출력 장치(150) 중 적어도 일부를 포함하도록 구현되거나 또는 트랜시버(transceiver), 데이터베이스 등과 같은 다른 구성요소들을 더 포함할 수도 있다.

[32] 통신 방식은 제한되지 않으며, 네트워크(160)가 포함할 수 있는 통신망(일례로, 이동통신망, 유선 인터넷, 무선 인터넷, 방송망)을 활용하는 통신 방식뿐만 아니라 블루투스(Bluetooth)나 NFC(Near Field Communication)와 같은 근거리 무선 통신 역시 포함될 수 있다. 예를 들어, 네트워크(160)는, PAN(personal area network), LAN(local area network), CAN(campus area network), MAN(metropolitan area network), WAN(wide area network), BBN(broadband network), 인터넷 등의 네트워크 중 하나 이상의 임의의 네트워크를 포함할 수 있다. 또한, 네트워크(160)는 버스 네트워크, 스타 네트워크, 링 네트워크, 메쉬 네트워크, 스타-버스 네트워크, 트리 또는 계층적(hierarchical) 네트워크 등을 포함하는 네트워크 토폴로지 중 임의의 하나 이상을 포함할 수 있으나, 이에 제한되지 않는다.

[33] 본 발명의 실시예들에 따른 역공학 방법에서는 우선 실행하고자 하는 프로그램의 실행 당시의 컨텍스트 수집을 필요로 한다. 일례로, 소프트웨어 브레이크 포인트(break point)와 브레이크 포인트 핸들러를 이용한 방식이 활용될 수 있다.

[34] 도 2는 본 발명의 일실시예에 따른 역공학 과정의 예를 도시한 도면이다. 도 2는 디버거(210), 이벤트 핸들러(220) 및 컨텍스트 데이터베이스(230)를 나타내고 있다.

[35] 일례로, 컴퓨터 장치(100)는 관찰하고자 하는 프로그램에 디버거(210)를 붙여 실행시킬 수 있다. 이때, 프로그램에서 관심있는 부분(또는 프로그램 전체)의 모든 인스트럭션에 브레이크 포인트를 설정해주면, 해당 부분에서 이벤트가 발생하게 되는데, 이때 컴퓨터 장치(100)는 미리 정의해 둔 이벤트 핸들러(220)를 통해 해당 시점에서의 실행 컨텍스트(execution context)를 컨텍스트 데이터베이스(230)에 저장할 수 있다. 추출되는 컨텍스트는 경우에 따라 다를 수 있기 때문에 정확한 DB 테이블의 형태는 중요하지 않으며, 테이블의 컬럼(column)에는 컨텍스트가 저장된 순서를 알 수 있는 인덱스 역할을 위한

값이 포함될 수 있다. 실행 컨텍스트란 프로그램이 실행되는 시점의 레지스터 값이나 메모리 상태를 의미할 수 있다. 프로그램이 같은 명령어를 실행하더라도 실행 컨텍스트가 다르면 그 결과는 달라지는데, 예를 들어 인텔 아키텍처에서 'mov eax, [ebx]' 라는 명령어를 실행할 때 ebx 레지스터가 어떤 값을 가지고 있느냐에 따라 메모리에서 값을 참조하는 위치가 달라지게 되어 프로그램, 혹은 프로그램을 실행하는 머신의 동작 방식이 달라질 수 있다.

[36] 이렇게 구축되는 컨텍스트 데이터베이스(230)는 분석자의 상황에 따라 필요한 정보를 가공하여 적당한 시각화 도구를 통해 충분한 정보를 분석자에게 전달 가능하다.

[37] 도 3은 본 발명의 일실시예에 있어서, 프로그램의 모든 명령어 지점에 브레이크포인트를 설정하고 해당 지점에서의 pc 값을 순차적으로 나타낸 그래프의 예를 도시한 도면이다. 이때, 그래프의 x-축은 값이 증가할수록 나중에 실행된 명령어라는 의미를 갖는 pc(program count)에 대응할 수 있으며, y-축은 그 명령어가 메모리상에서 어떤 주소에 저장되어 있는지를 표현할 수 있다. 이러한 pc(program count) 값들을 추출하는 동안 타겟 프로그램은 점선박스(310)에 나타난 바와 같이 같은 종류의 행동을 5번 수행하였다. 도 3의 그래프를 살펴보았을 때, 프로그램 명령어 중 어느 부분이 그 종류의 행동을 수행한 코드인지는 굉장히 명확하게 드러난다. 이 그래프가 표현하고 있는 부분과 그 종류의 행동을 수행한 코드가 정확하게 일치한다고 단정짓기는 어렵지만, 저 코드의 실행 근방에 그 행동을 수행한 코드가 존재할 확률이 아주 높다고 생각하는 것은 자연스럽다. 도 3의 그래프는 비교적 작고 단순한 프로그램이므로, 보다 크고 널리 이용되는 프로그램을 대상으로 한 예시도 살펴볼 수 있다.

[38] 도 4는 본 발명의 일실시예에 있어서, 문서 작성 프로그램의 화면 예를 도시한 도면이다. 도 4는 임의의 컴퓨터 장치에서 널리 사용중인 문서 작성 프로그램을 이용하여 특정 문서파일을 실행(open)함에 따라 임의의 컴퓨터 장치의 디스플레이 장치에 표시된 화면의 일부(400)를 나타내고 있다. 만약, 분석자가 해당 문서 작성 프로그램을 이용하여 특정 문서파일을 열 때 일어나는 과정에 대해서 분석하고자 하는 경우를 가정할 수 있다. 이때, 임의의 컴퓨터 장치에서 특정 문서파일을 열 때, 임의의 컴퓨터 장치에서 메모리 동적 할당자가 어떠한 크기로 메모리를 동적으로 할당하는가에 대한 정보를 도 3의 방법과 유사한 방법을 통해 얻을 수 있게 된다.

[39] 도 5는 본 발명의 일실시예에 있어서, 메모리 동적 할당자 함수의 에필로그 부분의 예를 도시한 도면이고, 도 6은 에필로그 부분에 대응하는 지점에서의 esi 레지스터 값들을 도시한 그래프의 예를 도시한 도면이다. 도 6의 그래프에서 x-축은 값이 증가할수록 나중에 실행된 명령어라는 의미를 갖고 있으며, y-축은 해당 명령어를 실행하는 시점에 esi 레지스터가 가지고 있는 값을 표현하고 있다. 도 6의 그래프는 특히 1000보다 작은 크기(ESI(Extended Source Pointer) 레지스터

값이 100 미만인 크기)의 메모리를 할당한 모습을 나타내고 있다. 여기서, esi 레지스터 값은 메모리 할당 함수가 할당한 메모리의 크기 값을 의미할 수 있다. 이 값은 사용자가 시스템에 요청한 메모리의 크기와는 다소 차이가 있을 수 있다. 이때, 도 6의 그래프에서는 아무런 특징점을 찾을 수 없다.

[40] 도 7은 본 발명의 일실시예에 있어서, 문서 작성 프로그램의 화면 예를 도시한 도면이고, 도 8은 본 발명의 일실시예에 있어서, 메모리 동적 할당자 함수의 에필로그 부분에 대응하는 지점에서의 esi 레지스터 값들을 도시한 그래프의 다른 예를 도시한 도면이다. 도 7에서는 도 4를 통해 설명한 것과 동일한 문서 작성 프로그램을 이용하여 임의의 컴퓨터 장치에서 특정 문서파일을 실행함에 따라 임의의 컴퓨터 장치의 디스플레이 장치에 표시된 화면의 일부(700)를 나타내고 있다. 도 4에서는 특정 문서파일이 단어 "CODEGATE"를 하나만 포함하고 있던 것에 비해 도 7에서는 특정 문서파일이 동일한 단어 "CODEGATE"가 다수 개 포함하고 있는 예를 나타내고 있다. 이때, 도 8의 그래프에서 점선박스(810)는 일정한 값을 연속적으로 표현하고 있다. 이 경우, 분석자는 도 7에 나타난 특정 문서 파일을 문서 작성 프로그램이 다룰 때, 어떤 크기의 메모리를 동적으로 사용하고 있는가에 대해 알 수 있고, 프로세스 전체 메모리에서 해당 크기를 갖고 있는 메모리 공간을 모두 검색함으로써 프로그램이 어떤 단위 혹은 구조체로 이 문서파일을 다루고 있는가에 대한 추가적인 이해를 얻을 수 있게 된다.

[41] 도 9는 본 발명의 일실시예에 있어서, 메모리 주소의 예를 도시한 도면이다. 도 9는 점선박스(910)는 문서에 적어 넣은 문자열이 실제 메모리 주소를 통해 확인될 수 있는 메모리 공간상에 그대로 존재하고 있음을 나타내고 있다.

[42] 도 10은 본 발명의 일실시예에 따른 역공학 방법의 예를 도시한 흐름도이다. 본 실시예에 따른 역공학 방법은 일례로 앞서 설명한 컴퓨터 장치(100)에 의해 수행될 수 있다. 예를 들어, 컴퓨터 장치(100)의 프로세서(120)는 메모리(110)가 포함하는 운영체제의 코드나 적어도 하나의 컴퓨터 프로그램의 코드에 따른 제어 명령(instruction)을 실행하도록 구현될 수 있다. 여기서, 프로세서(120)는 컴퓨터 장치(100)에 저장된 코드가 제공하는 제어 명령에 따라 컴퓨터 장치(100)가 도 10의 방법이 포함하는 단계들(1010 내지 1040)을 수행하도록 컴퓨터 장치(100)를 제어할 수 있다. 여기서 상술한 적어도 하나의 컴퓨터 프로그램은 역공학 방법을 위해 컴퓨터 장치(100)에 설치(install)된 프로그램일 수 있으며, 이후 설명되는 분석 대상 프로그램과는 별도의 프로그램일 수 있다.

[43] 단계(1010)에서 컴퓨터 장치(100)는 분석 대상 프로그램을 디버거를 통해 실행할 수 있다. 디버거는 일례로, 에러를 포함하고 있는 프로그램에 관하여 그 에러의 원인을 색출하기 위하여 보조적으로 이용되는 프로그램으로, 프로그램을 추적하거나 특정 변수 간의 관계식을 프로그램의 구절마다 조사하여 가급적 빠른 시점에 프로그램의 이상 동적을 검출하는데 사용되는 프로그램일 수 있다. 컴퓨터 장치(100)는 이러한 디버거 프로그램을 이용하여

분석 대상 프로그램을 실행시킬 수 있다.

[44] 단계(1020)에서 컴퓨터 장치(100)는 디버거를 이용하여 분석 대상 프로그램의 관심 영역에 포함된 인스트럭션에 브레이크 포인트를 설정할 수 있다. 브레이크 포인트는 소프트웨어 개발에서 프로그램을 의도적으로 잠시 또는 아예 멈추게 하는 위치를 가리키며 디버깅을 목적으로 활용될 수 있다. 브레이크 포인트는 이미 실행중인 프로그램에 대한 정보를 알아내기 위한 수단으로 사용되며, 이를 이용하여 프로그램 실행이 중단되어 있는 상황에서 분석자가 각종 테스트 환경(일반 목적의 레지스터, 메모리, 로그, 파일 등)을 점검하여 프로그램이 예측한대로 기능하고 있는지, 그렇지 않을 경우 문제점이 무엇인지 등을 파악할 수 있다. 실시예에 따라 브레이크 포인트는 원하는 순간에 프로그램 실행을 중단하기 위한 조건을 포함할 수도 있다.

[45] 단계(1030)에서 컴퓨터 장치(100)는 브레이크 포인트가 설정된 인스트럭션에서 발생하는 이벤트에 대해 이벤트 핸들러를 통해 해당 시점에서의 실행 컨텍스트(execution context)를 컨텍스트 데이터베이스에 저장할 수 있다. 이미 설명한 바와 같이, 추출되는 컨텍스트는 경우에 따라 다를 수 있기 때문에 정확한 DB 테이블의 형태는 중요하지 않지만, 컨텍스트 데이터베이스에 저장되는 테이블의 컬럼(column)에는 실행 컨텍스트가 저장된 순서를 나타내는 인덱스 값이 포함될 수 있다. 한편, 실행 컨텍스트는 상기 실행 컨텍스트에 대응하는 시점에서의 상기 분석 대상 프로그램에 대한 레지스터 값 및/또는 메모리 상태를 포함할 수 있다. 일례로, 앞에서 PC 값 및/또는 ESI 레지스터 값을 활용하는 예시들을 설명한 바 있다.

[46] 단계(1040)에서 컴퓨터 장치(100)는 컨텍스트 데이터베이스에 저장된 데이터를 처리하여 분석 대상 프로그램에 대한 정보 분석 결과를 생성할 수 있다. 이때, 컴퓨터 장치(100)는 정보 분석 결과를 생성하기 위해, 빅데이터 기술 등을 활용하여 컨텍스트 데이터베이스에 저장된 데이터를 가공할 수 있으며, 시각화 도구를 활용하여 가공된 데이터에 대한 정보 분석 결과를 분석자에게 리포트할 수 있다. 리포트되는 정보 분석 결과의 예시는 도 4, 도 6 및 도 8을 통해 설명한 바 있다.

[47] 이처럼 본 발명의 실시예들에 따르면, 바이너리의 분석자가 바이너리의 구조에 대한 이해가 아주 적은 경우에도 프로그램의 구조 분석을 하기 위한 시작점을 찾는 것을 용이하게 함으로써 프로그램 분석에 걸리는 시간을 줄일 수 있다.

[48] 이상에서 설명된 시스템 또는 장치는 하드웨어 구성요소, 또는 하드웨어 구성요소 및 소프트웨어 구성요소의 조합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성요소는, 예를 들어, 프로세서, 콘트롤러, ALU(arithmetic logic unit), 디지털 신호 프로세서(digital signal processor), 마이크로컴퓨터, FPGA(field programmable gate array), PLU(programmable logic unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를

이용하여 구현될 수 있다. 처리 장치는 운영 체제(OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 어플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(processing element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 콘트롤러를 포함할 수 있다. 또한, 병렬 프로세서(parallel processor)와 같은, 다른 처리 구성(processing configuration)도 가능하다.

[49] 소프트웨어는 컴퓨터 프로그램(computer program), 코드(code), 명령(instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로(collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성요소(component), 물리적 장치, 가상 장치(virtual equipment), 컴퓨터 저장 매체 또는 장치에 구체화(embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록매체에 저장될 수 있다.

[50] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 매체는 컴퓨터로 실행 가능한 프로그램을 계속 저장하거나, 실행 또는 다운로드를 위해 임시 저장하는 것일 수도 있다. 또한, 매체는 단일 또는 수개 하드웨어가 결합된 형태의 다양한 기록수단 또는 저장수단일 수 있는데, 어떤 컴퓨터 시스템에 직접 접속되는 매체에 한정되지 않고, 네트워크 상에 분산 존재하는 것일 수도 있다. 매체의 예시로는, 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체, CD-ROM 및 DVD와 같은 광기록 매체, 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical medium), 및 ROM, RAM, 플래시 메모리 등을 포함하여 프로그램 명령어가 저장되도록 구성된 것이 있을 수 있다. 또한, 다른 매체의 예시로, 애플리케이션을 유통하는 앱 스토어나 기타 다양한 소프트웨어를 공급 내지 유통하는 사이트, 서버 등에서 관리하는 기록매체 내지 저장매체도 들 수 있다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다.

발명의 실시를 위한 형태

[51] 이상과 같이 실시예들이 비록 한정된 실시예와 도면에 의해 설명되었으나, 해당 기술분야에서 통상의 지식을 가진 자라면 상기의 기재로부터 다양한 수정 및 변형이 가능하다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다.

[52] 그러므로, 다른 구현들, 다른 실시예들 및 청구범위와 균등한 것들도 후술하는 청구범위의 범위에 속한다.

청구범위

[청구항 1] 분석 대상 프로그램을 디버거를 통해 실행하는 단계;
상기 디버거를 이용하여 상기 분석 대상 프로그램의 관심 영역에 포함된 인스트럭션에 브레이크 포인트를 설정하는 단계;
상기 브레이크 포인트가 설정된 인스트럭션에서 발생하는 이벤트에 대해 이벤트 핸들러를 통해 해당 시점에서의 실행 컨텍스트(execution context)를 컨텍스트 데이터베이스에 저장하는 단계; 및
상기 컨텍스트 데이터베이스에 저장된 데이터를 처리하여 상기 분석 대상 프로그램에 대한 정보 분석 결과를 생성하는 단계
를 포함하는 역공학 방법.

[청구항 2] 제1항에 있어서,
상기 정보 분석 결과를 생성하는 단계는,
상기 컨텍스트 데이터베이스에 저장된 데이터를 가공하는 단계; 및
시각화 도구를 활용하여 상기 가공된 데이터에 대한 정보 분석 결과를 리포트하는 단계
를 포함하는 것을 특징으로 하는 역공학 방법.

[청구항 3] 제1항에 있어서,
상기 컨텍스트 데이터베이스에 저장되는 테이블의 컬럼(column)에 상기 실행 컨텍스트가 저장된 순서를 나타내는 인덱스 값이 포함되는 것을 특징으로 하는 역공학 방법.

[청구항 4] 제1항에 있어서,
상기 실행 컨텍스트는 상기 실행 컨텍스트에 대응하는 시점에서의 상기 분석 대상 프로그램에 대한 레지스터 값 및 메모리 상태 중 적어도 하나를 포함하는 것을 특징으로 하는 역공학 방법.

[청구항 5] 컴퓨터 장치와 결합되어 제1항 내지 제4항 중 어느 한 항의 방법을 컴퓨터 장치에 실행시키기 위해 컴퓨터 판독 가능한 기록매체에 저장된 컴퓨터 프로그램.

[청구항 6] 제1항 내지 제4항 중 어느 한 항의 방법을 컴퓨터 장치에 실행시키기 위한 컴퓨터 프로그램이 기록되어 있는 컴퓨터 판독 가능한 기록매체.

[청구항 7] 컴퓨터 장치에 있어서,
상기 컴퓨터 장치에서 판독 가능한 명령을 실행하도록 구현되는 적어도 하나의 프로세서
를 포함하고,
상기 적어도 하나의 프로세서에 의해,
분석 대상 프로그램을 디버거를 통해 실행하고,
상기 디버거를 이용하여 상기 분석 대상 프로그램의 관심 영역에 포함된 인스트럭션에 브레이크 포인트를 설정하고,

상기 브레이크 포인트가 설정된 인스트럭션에서 발생하는 이벤트에 대해 이벤트 핸들러를 통해 해당 시점에서의 실행 컨텍스트(execution context)를 컨텍스트 데이터베이스에 저장하고,
상기 컨텍스트 데이터베이스에 저장된 데이터를 처리하여 상기 분석 대상 프로그램에 대한 정보 분석 결과를 생성하는 것
을 특징으로 하는 컴퓨터 장치.

[청구항 8] 제7항에 있어서,
상기 적어도 하나의 프로세서에 의해,
상기 컨텍스트 데이터베이스에 저장된 데이터를 가공하고,
시각화 도구를 활용하여 상기 가공된 데이터에 대한 정보 분석 결과를 리포트하는 것
을 특징으로 하는 컴퓨터 장치.

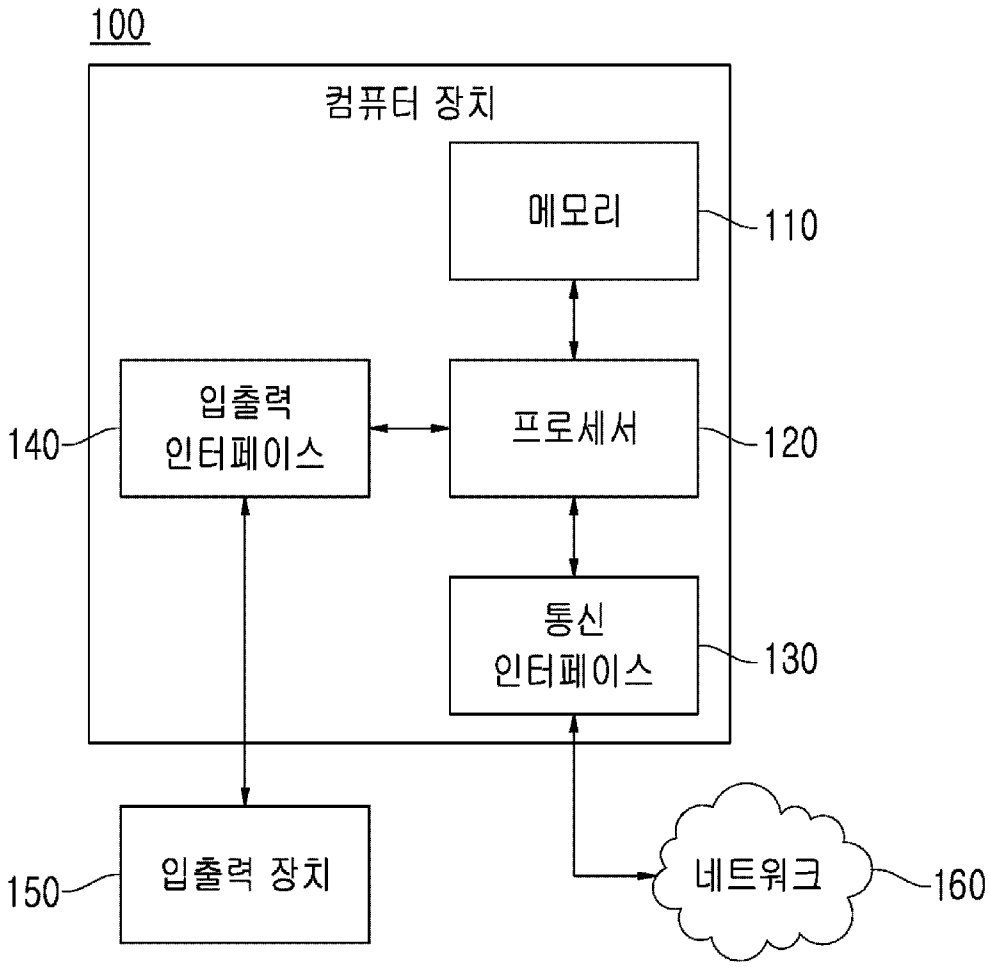
[청구항 9] 제7항에 있어서,
상기 컨텍스트 데이터베이스에 저장되는 테이블의 컬럼(column)에 상기 실행 컨텍스트가 저장된 순서를 나타내는 인덱스 값이 포함되는 것을 특징으로 하는 컴퓨터 장치.

[청구항 10] 제7항에 있어서,
상기 실행 컨텍스트는 상기 실행 컨텍스트에 대응하는 시점에서의 상기 분석 대상 프로그램에 대한 레지스터 값 및 메모리 상태 중 적어도 하나를 포함하는 것을 특징으로 하는 컴퓨터 장치.

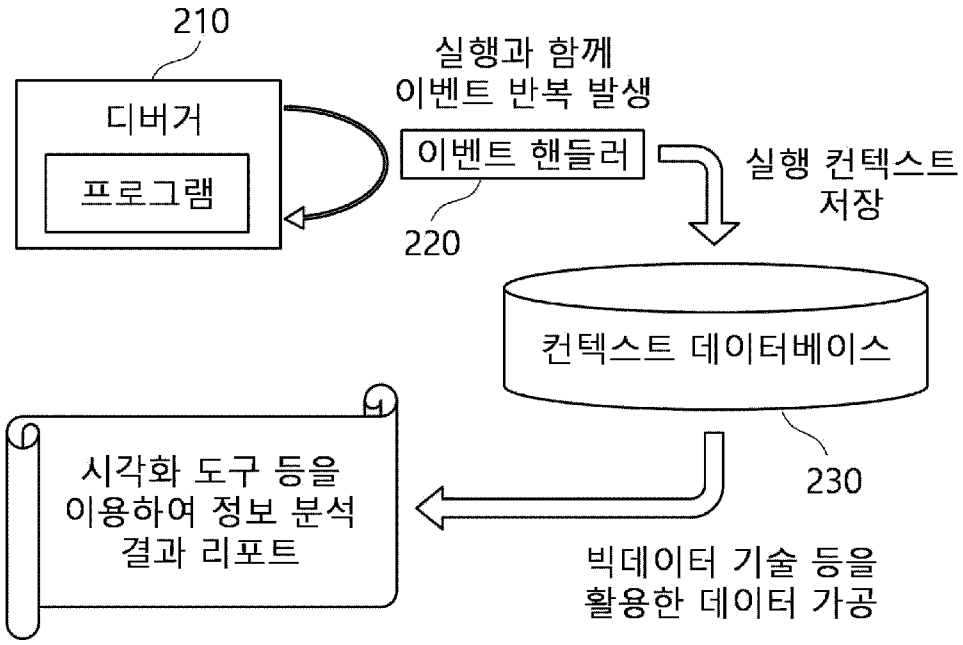
요약서

프로그램 실행 컨텍스트 기반의 빅데이터를 활용한 역공학 방법 및 시스템을 제공한다. 일실시예에 따른 역공학 방법은, 분석 대상 프로그램을 디버거를 통해 실행하는 단계, 상기 디버거를 이용하여 상기 분석 대상 프로그램의 관심 영역에 포함된 인스트럭션에 브레이크 포인트를 설정하는 단계, 상기 브레이크 포인트가 설정된 인스트럭션에서 발생하는 이벤트에 대해 이벤트 핸들러를 통해 해당 시점에서의 실행 컨텍스트(execution context)를 컨텍스트 데이터베이스에 저장하는 단계 및 상기 컨텍스트 데이터베이스에 저장된 데이터를 처리하여 상기 분석 대상 프로그램에 대한 정보 분석 결과를 생성하는 단계를 포함할 수 있다.

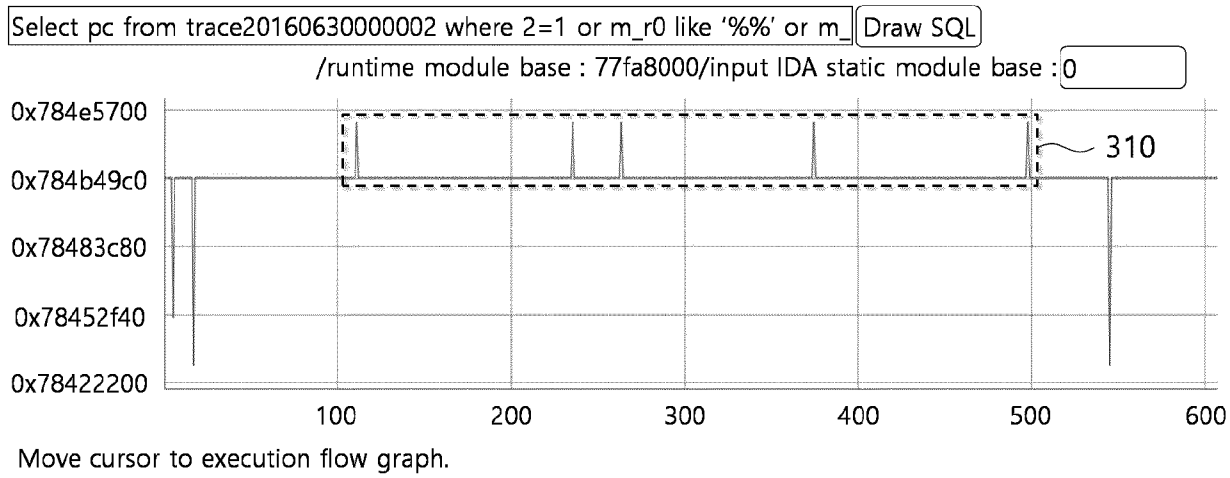
[도1]


100
컴퓨터 장치
메모리
110
입출력
인터페이스
140
프로세서
120
통신
인터페이스
130
입출력 장치
150
네트워크
160

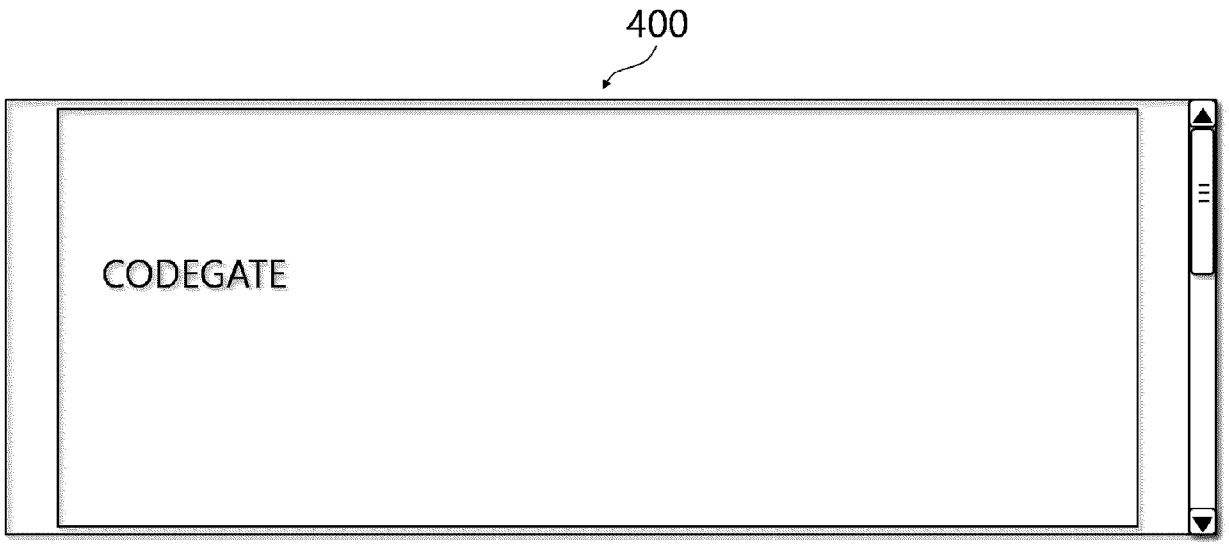
[도2]


210
디버거
프로그램
실행과 함께
이벤트 반복 발생
이벤트 핸들러
220
실행 컨텍스트
저장
컨텍스트 데이터베이스
230
시각화 도구 등을
이용하여 정보 분석
결과 리포트
빅데이터 기술 등을
활용한 데이터 가공

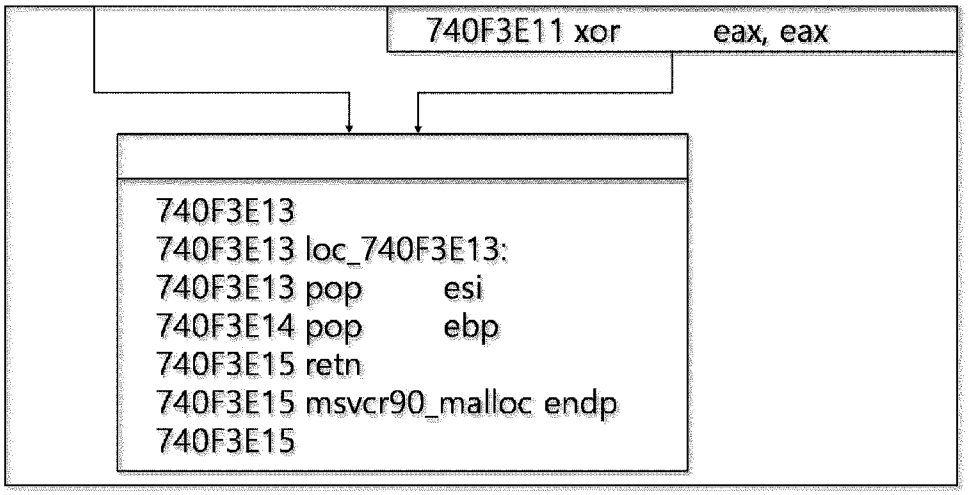
[도3]


Select pc from trace20160630000002 where 2=1 or m_r0 like '%%' or m_
Draw SQL
/runtime module base : 77fa8000/input IDA static module base :
0
0x784e5700
0x784b49c0
0x78483c80
0x78452f40
0x78422200
310
100
200
300
400
500
600
Move cursor to execution flow graph.

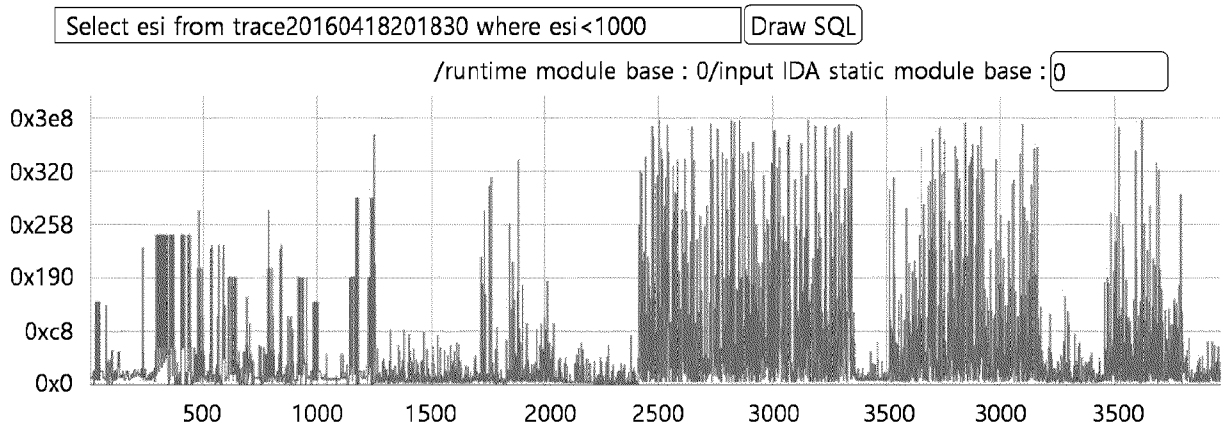
[도4]


400
CODEGATE

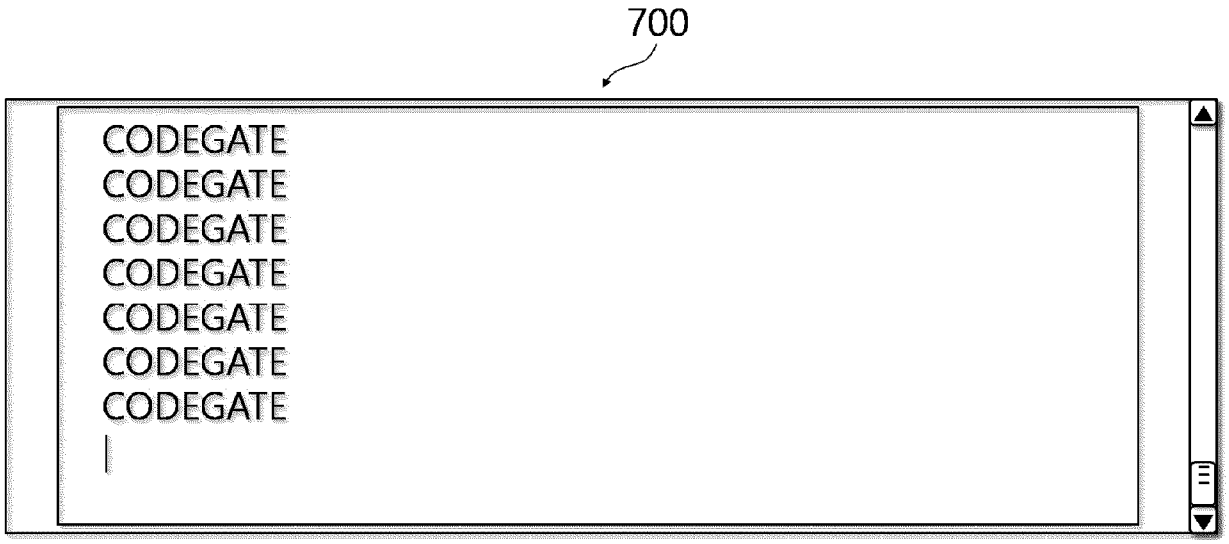
[도5]


740F3E11 xor eax, eax
740F3E13
740F3E13 loc_740F3E13:
740F3E13 pop esi
740F3E14 pop ebp
740F3E15 retn
740F3E15 msvcr90_malloc endp
740F3E15

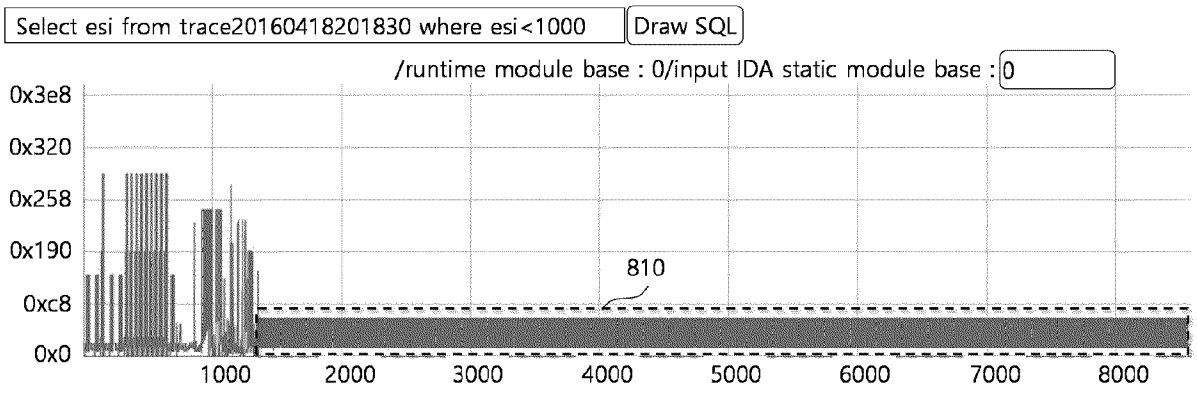
[도6]


Select esi from trace20160418201830 where esi<1000
Draw SQL
/runtime module base : 0/input IDA static module base : 0
0x3e8
0x320
0x258
0x190
0xc8
0x0
500
1000
1500
2000
2500
3000
3500
3000
3500

[도7]


700
CODEGATE
CODEGATE
CODEGATE
CODEGATE
CODEGATE
CODEGATE
CODEGATE

[도8]


Select esi from trace20160418201830 where esi<1000
Draw SQL
/runtime module base : 0/input IDA static module base : 0
0x3e8
0x320
0x258
0x190
0xc8
0x0
810
1000
2000
3000
4000
5000
6000
7000
8000
Move cursor to execution flow graph.

[도9]

06A5CF80	01 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	. . . . .
06A5CF90	00 00 00 00 00 00 00 00	01 00 00 00 E8 03 00 00	. . . . .
06A5CFA0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	. . . . .
06A5CFB0	01 00 00 00 00 00 00 00	BE 00 F5 4D 00 00 00 00	. . . . . M . . . .
06A5CFC0	43 00 4F 00 44 00 45 00	47 00 41 00 54 00 45 00	C . O . D . E . G . A . T . E .
06A5CFD0	0D 00 00 00 00 00 00 00	00 19 00 00 E8 03 00 00	. . . . .
06A5CFE0	E8 03 00 00 52 03 00 00	58 02 00 00 00 00 00 00	. . . . R . . . X . . . . .

910

[도10]