(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2015-0069844
(43) 공개일자 2015년06월24일

(51) 국제특허분류(Int. Cl.)
G06F 21/14 (2013.01)
(21) 출원번호 10-2013-0156473
(22) 출원일자 2013년12월16일
심사청구일자 2013년12월16일

(71) 출원인
주식회사 에스이웍스
서울 서초구 고무래로10길 26 302(반포동,반도빌딩3층)
(72) 발명자
홍동철
서울 동작구 상도로45길 21-2, 101호 (상도1동)
김동선
부산 해운대구 송정중앙로9번길 60, (송정동)
홍민표
서울특별시 강남구 언주로 116길 6, 101동 402호 (논현동, 동부센트레빌)
(74) 대리인
특허법인유아이피

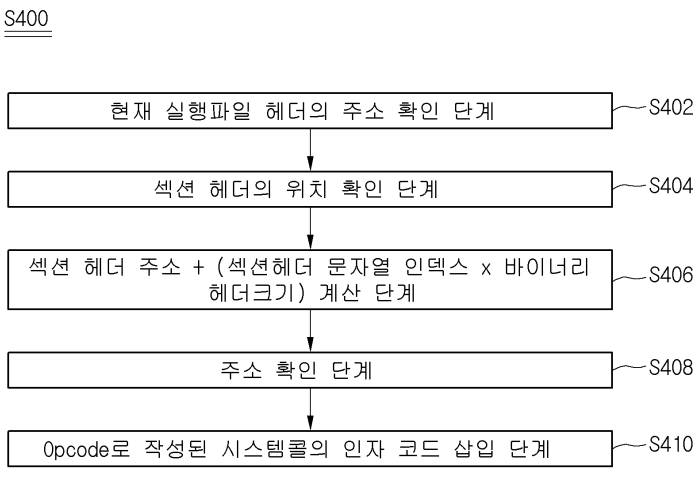
전체 청구항 수 : 총 8 항

(54) 발명의 명칭 **ARM 프로세서 기반의 파일 난독화 방법**

(57) 요 약

본 발명은 ARM 프로세서 기반의 ELF 파일 구조를 갖는 시스템에서의 실행파일과 라이브러리 모듈이 역공학을 통해 프로그램의 흐름을 파악하기 어렵도록 하기 위해, ARM 프로세서를 이용하는 ELF 파일의 실행파일 및 라이브러리 난독화 방법에 있어서, (a) 원본 실행파일 내의 실행 코드가 담겨 있는 섹션의 코드를 암호화하여 고유 섹션에 저장하는 단계; (b) 실행 코드 섹션의 고유 섹션에 암호화 저장 되어 있는 코드를 복호화하고 실행하는 코드를 삽입하는 단계; (c) 실행 시(Runtime) 실행 코드 섹션의 쓰기 권한을 획득하는 코드를 삽입하는 단계; (d) 실행 시(Runtime) 정확한 메모리 주소를 구해 실행코드 섹션의 코드에 덮어쓰는 코드를 삽입하는 단계; 및 (e) 디버깅 툴이 기계 명령어(Opcode)를 해석하기 어렵도록 바이너리의 헤더와 섹션을 조작하는 단계를 포함하는 것을 특징으로 하는 ARM 프로세서 기반의 파일 난독화 방법을 제공한다.

대 표 도 - 도4



명 세 서

청구범위

청구항 1

ARM 프로세서를 이용하는 ELF 파일의 실행파일 및 라이브러리 난독화 방법에 있어서,

(a) 원본 실행파일 내의 실행 코드가 담겨 있는 섹션의 코드를 암호화하여 고유 섹션에 저장하는 단계;

(b) 실행 코드 섹션의 고유 섹션에 저장된 암호화되어 있는 코드를 복호화하고 실행하는 코드를 삽입하는 단계;

(c) 실행 시(Runtime) 실행 코드 섹션의 쓰기 권한을 획득하는 코드를 삽입하는 단계;

(d) 실행 시(Runtime) 정확한 메모리 주소를 구해 실행코드 섹션의 코드에 덮어쓰는 코드를 삽입하는 단계; 및

(e) 디버깅 툴이 기계 명령어(Opcode)를 해석하기 어렵도록 바이너리의 헤더와 섹션을 조작하는 단계를 포함하는 것을 특징으로 하는 ARM 프로세서 기반의 파일 난독화 방법.

청구항 2

제 1항에 있어서,

상기 단계 (a)는,

특정 섹션을 생성하는 단계;

상기 특정 섹션으로 실행 섹션의 코드를 모두 이동하는 단계; 및

해당 섹션에 담긴 코드들을 암호화와 같은 역공학을 방지하기 위한 알고리즘들을 적용하는 단계를 포함하는 것을 특징으로 하는 ARM 프로세서 기반의 파일 난독화 방법.

청구항 3

제 1항에 있어서,

상기 단계 (b)는,

실행코드 섹션에 고유 섹션의 코드를 복호화와 같은 역공학을 방지하기 위한 알고리즘들을 해제하는 코드를 삽입하는 단계;

특정 세그먼트의 이름을 변경하는 단계;

변경된 이름의 세그먼트 속성을 삽입할 Opcode의 크기에 맞춰 수정하는 단계; 및

현재 Entry Point인 실행 코드 섹션에 변경된 세그먼트에 들어있는 암호화된 코드를 복호화 후 점프 코드를 삽입하는 단계를 포함하는 것을 특징으로 하는 ARM 프로세서 기반의 파일 난독화 방법.

청구항 4

제 1항에 있어서,

상기 단계 (c)는,

메모리 영역의 권한을 설정하는 시스템 콜을 사용하여 실행 코드 섹션의 쓰기 권한을 확보하는 코드를 삽입하는 것을 특징으로 하는 ARM 프로세서 기반의 파일 난독화 방법.

청구항 5

제 4항에 있어서,

상기 시스템 콜은 운영체제가 제공해주는 기능으로 주어진 파일을 현재 프로세스의 메모리 영역에 매핑(Mapping)시켜 주는 것을 특징으로 하는 ARM 프로세서 기반의 파일 난독화 방법.

청구항 6

제 1항에 있어서,

상기 단계 (d)는,

(d-1) 현재 실행파일 헤더의 주소를 구하는 단계;

(d-2) 실행파일 헤더의 정보를 참조해 섹션 헤더의 위치를 구하는 단계;

(d-3) 섹션 헤더의 주소 + (섹션 헤더의 문자열 인덱스 ㅅ 바이너리 헤더의 크기)를 계산하는 단계;

(d-4) 계산한 주소 값이 실행 코드 섹션의 주소인지 확인하는 단계; 및

상기 (d-1) 내지 (d-4)에서 구해진 실행 코드 섹션의 주소를 이용하여 Opcode로 작성된 시스템 콜 인자를 채워주는 코드를 삽입하는 단계를 포함하는 것을 특징으로 하는 ARM 프로세서 기반의 파일 난독화 방법.

청구항 7

제 1항에 있어서,

상기 단계 (e)는,

실행파일 헤더 만큼의 크기를 메모리 영역에 할당하는 단계;

실행파일 헤더의 주소를 이용해 섹션들의 주소를 구하는 단계;

구한 두 가지 섹션을 더미 데이터로 채우는 단계;

매핑한 메모리 맵과 동기화(Synchronize)하는 단계; 및

매핑한 메모리를 해제하는 코드를 삽입하는 단계를 포함하는 것을 특징으로 하는 ARM 프로세서 기반의 파일 난독화 방법.

청구항 8

제 7항에 있어서,

상기 단계 (e)는,

섹션의 이름을 변경하여 디버거에게 혼란을 주기 위해 바이너리 헤더의 주소를 구하는 단계; 및

내부 섹션들을 추출해서 소속된 세그먼트가 변경 대상인지를 확인하고, 변경 대상 세그먼트의 속성을 비정상적으로 변경하는 과정을 반복하는 단계를 더 포함하는 것을 특징으로 하는 ARM 프로세서 기반의 파일 난독화 방법.

발명의 설명

기 술 분 야

[0001] 본 발명은 ARM 프로세서 기반의 파일 난독화 방법에 관한 것으로, 보다 상세하게는 ARM 프로세서 기반의 ELF 파일 구조를 갖는 시스템에서의 실행파일에 역공학을 이용하여 프로그램의 흐름을 파악하기 어렵도록 하기 위한

ARM 프로세서 기반의 파일 난독화 방법에 관한 것이다.

배 경 기 술

[0002] 보안 취약점 분석자들은 각종 보안 문제 분석에 역공학(Reverse engineering) 기술을 적극 활용하고 있다. 역공학은 소스코드 없이 윈도우즈 실행 파일(Portable Executable)이나, 자바 바이트코드 등을 직접 분석해서 프로그램이 어떤 기능을 수행하는지 파악하여 취약점을 찾아내는 기술이다.

[0003] 필요하면 직접 프로그램 바이너리를 수정해 불법적인 일을 수행하게 만들기도 한다. 이에 대한 대응으로 코드를 복잡하게 만들어 알아보기 힘들게 하는 코드 난독화(Code Obfuscation) 기술이 발전하였다.

[0004] 하지만, 이러한 코드 난독화를 이용하여 원본 소스코드를 난독화된 소스코드로 변환하여도 단순히 코드의 복잡도만 상승하였기 때문에 역컴파일로 인한 소스코드 노출이 가능하다.

[0005] 이러한 문제를 해결하기 위하여 대한민국 등록특허공보 10-1097103호(2011.12.22.)에서는 소프트웨어를 구현하기 위한 알고리즘, 구조, 구성 등을 포함하는 소스코드를 난독화하고, 난독화를 위한 복원정보, 처리내역 등을 서버에서 관리토록하여 소스코드의 유출을 방지하기 위한 방법을 개시하고 있다.

[0006] 한편, ARM(Advanced RISC Machine) 프로세서의 실행 파일은 기계어와 1:1 대응되는 코드인 Opcode와 프로그램의 정상적인 작동을 위해 필요한 데이터의 집합으로 구성되어 있다.

[0007] 상기 Opcode는 아키텍처마다 제공하는 Opcode Table과 대조하면 이를 어셈블리 명령으로 번역하는 것이 가능하다.

[0008] 대표적으로 GDB, objdump, IDA와 같은 분석 툴들이 어셈블리 형태로 변환하여 출력해주는 역할을 수행하고 보다 고도화된 툴인 Hex-Rays를 사용하면 어셈블리 형태가 아닌 원본 소스 코드와 거의 유사한 수준의 코드를 복원해내는 역컴파일 기능을 제공한다.

[0009] 이러한 역컴파일을 막기 위한 기존의 기술들은 대부분 윈도우의 실행파일인 PE(Portable Executable) 포맷에 대한 난독화 기법이기 때문에 ELF(Executable and Linking Format) 구조에서의 임베디드 ARM 프로세서의 난독화 기법 개발이 시급한 상황이다.

발명의 내용

해결하려는 과제

[0010] 따라서, 본 발명은 상술한 문제점을 해결하기 위한 것으로, 본 발명의 목적은 ARM 프로세서를 이용하는 ELF 파일의 실행 파일 및 라이브러리 모듈을 난독화하기 위한 ARM 프로세서 기반의 파일 난독화 방법을 제공함에 있다.

[0011] 또한, 본 발명의 목적은 ARM 프로세서 기반의 ELF 파일 구조를 가지고 있는 시스템에서의 실행 파일 및 라이브러리 모듈에 난독화 기법을 적용하여 역공학을 통해 프로그램의 흐름을 파악하기 어렵도록 하기 위한 ARM 프로세서 기반의 파일 난독화 방법을 제공함에 있다.

과제의 해결 수단

[0012] 상기한 본 발명의 목적은, ARM 프로세서를 이용하는 ELF 파일의 실행파일 및 라이브러리 난독화 방법에 있어서, (a) 원본 실행파일 내의 실행 코드가 담겨 있는 섹션의 코드를 암호화하여 고유 섹션에 저장하는 단계; (b) 실행 코드 섹션의 고유 섹션에 저장된 암호화되어 있는 코드를 복호화하고 실행하는 코드를 삽입하는 단계; (c) 실행 시(Runtime) 실행 코드 섹션의 쓰기 권한을 획득하는 코드를 삽입하는 단계; (d) 실행 시(Runtime) 정확한 메모리 주소를 구해 실행코드 섹션의 코드에 덮어쓰는 코드를 삽입하는 단계; 및 (e) 디버깅 툴이 기계 명령어(Opcode)를 해석하기 어렵도록 바이너리의 헤더와 섹션을 조작하는 단계를 포함하는 것을 특징으로 하는 ARM 프로세서 기반의 파일 난독화 방법을 통해서 달성된다.

[0013] 또한 본 발명에 따르면, 상기 단계 (a)는, 특정 섹션을 생성하는 단계; 상기 특정 섹션으로 실행 섹션의 코드를 모두 이동하는 단계; 및 해당 섹션에 담긴 코드들을 암호화와 같은 역공학을 방지하기 위한 알고리즘들을 적용하는 단계를 포함하는 것을 특징으로 한다.

[0014] 또한 본 발명에 따르면, 상기 단계 (b)는, 실행코드 섹션에 고유 섹션의 코드를 복호화와 같은 역공학을 방지하기 위한 알고리즘들을 해제하는 코드를 삽입하는 단계; 특정 세그먼트의 이름을 변경하는 단계; 변경된 이름의 세그먼트 속성을 삽입할 Opcode의 크기에 맞춰 수정하는 단계; 및 현재 Entry Point인 실행 코드 섹션에 변경된 세그먼트에 들어있는 암호화된 코드를 복호화 후 점프 코드를 삽입하는 단계를 포함하는 것을 특징으로 한다.

[0015] 또한 본 발명에 따르면, 상기 단계 (c)는, 메모리 영역의 권한을 설정하는 시스템 콜을 사용하여 실행 코드 섹션의 쓰기 권한을 확보하는 코드를 삽입하는 것을 특징으로 한다.

[0016] 또한 본 발명에 따르면, 상기 시스템 콜은 운영체제가 제공해주는 기능으로 주어진 파일을 현재 프로세스의 메모리 영역에 매핑(Mapping)시켜 주는 것을 특징으로 한다.

[0017] 또한 본 발명에 따르면, 상기 단계 (d)는, (d-1) 현재 실행파일 헤더의 주소를 구하는 단계; (d-2) 실행파일 헤더의 정보를 참조해 섹션 헤더의 위치를 구하는 단계; (d-3) 섹션 헤더의 주소 + (섹션 헤더의 문자열 인덱스 ㅅ 바이너리 헤더의 크기)를 계산하는 단계; (d-4) 계산한 주소 값이 실행 코드 섹션의 주소인지 확인하는 단계; 및 상기 (d-1) 내지 (d-4)에서 구해진 실행 코드 섹션의 주소를 이용하여 Opcode로 작성된 시스템 콜 인자를 채워주는 코드를 삽입하는 단계를 포함하는 것을 특징으로 한다.

[0018] 또한 본 발명에 따르면, 상기 단계 (e)는, 실행파일 헤더 만큼의 크기를 메모리 영역에 할당하는 단계; 실행파일 헤더의 주소를 이용해 섹션들의 주소를 구하는 단계; 구한 두 가지 섹션을 더미 데이터로 채우는 단계; 매핑한 메모리 맵과 동기화(Synchronize)하는 단계; 및 매핑한 메모리를 해제하는 코드를 삽입하는 단계를 포함하는 것을 특징으로 한다.

[0019] 또한 본 발명에 따르면, 상기 단계 (e)는, 섹션의 이름을 변경하여 디버거에게 혼란을 주기 위해 바이너리 헤더의 주소를 구하는 단계; 및 내부 섹션들을 추출해서 소속된 세그먼트가 변경 대상인지를 확인하고, 변경 대상 세그먼트의 속성을 비정상적으로 변경하는 과정을 반복하는 단계를 더 포함하는 것을 특징으로 한다.

발명의 효과

[0020] 본 발명의 ARM 프로세서 기반의 파일 난독화 방법에 의하면, 임베디드 ARM 시스템에서 동작하는 실행 파일 및 라이브러리 모듈을 난독화로 보호할 수 있으며, 스마트기기의 실행 파일 및 라이브러리 모듈을 난독화할 수 있는 효과가 있다.

도면의 간단한 설명

[0021] 도 1은 본 발명의 실시예에 따른 ARM 프로세서 기반의 파일 난독화 과정의 전체 흐름을 나타낸 도면.

도 2는 본 발명의 실시예에 따른 섹션 코드 암호화 및 저장 과정을 나타낸 흐름도.

도 3은 본 발명의 실시예에 따른 난독화를 위한 복호화 및 실행코드 삽입 과정을 나타낸 흐름도.

도 4는 본 발명의 실시예에 따른 난독화를 위한 메모리 주소 덮어쓰기 과정을 나타낸 흐름도.

도 5는 본 발명의 실시예에 따른 난독화를 위한 바이너리 헤더 및 섹션 조작 과정을 나타낸 흐름도.

발명을 실시하기 위한 구체적인 내용

[0022] 본 명세서 및 청구범위에 사용된 용어나 단어는 통상적이거나 사전적인 의미로 한정해서 해석되어서는 아니되며, 발명자는 그 자신의 발명을 가장 최선의 방법으로 설명하기 위해 용어의 개념을 적절하게 정의할 수 있다는 원칙에 입각하여 본 고안의 기술적 사상에 부합하는 의미와 개념으로 해석되어야만 한다.

[0023] 따라서, 본 명세서에 기재된 실시예와 도면에 도시된 구성은 본 발명의 가장 바람직한 일 실시예에 불과할 뿐이고 본 발명의 기술적 사상을 모두 대변하는 것은 아니므로, 본 출원시점에 있어서 이들을 대체할 수 있는 다양

한 균등물과 변형예들이 있을 수 있음을 이해하여야 한다.

[0024] 대부분의 임베디드 시스템과 스마트폰 운영체제는 ARM(Advanced RISC Machine) 프로세서를 이용하는 리눅스 기반의 운영체제를 이용하고 있으며, ARM 프로세서를 이용하는 ELF(Executable and Linking Format) 파일의 실행파일 및 라이브러리는 악의적인 목적의 사용자가 분석을 하기 용이하다.

[0025] 따라서, ARM 프로세서 기반의 ELF 파일 구조를 가지고 있는 시스템에서의 실행파일에 난독화 기법을 적용하여 역공학을 통해 프로그램을 분석하기 어렵도록 하는 방법을 제공하고자 한다.

[0026] 이하 첨부된 도면을 참조하여 본 발명의 바람직한 실시예를 상세히 설명하기로 한다.

[0027] 도 1은 본 발명의 실시예에 따른 ARM 프로세서 기반의 파일 난독화 과정의 전체 흐름을 나타낸 도면이다.

[0028] 도 1에 도시된 바와 같이, 원본 실행파일 내의 실행 코드가 담겨있는 섹션의 코드를 암호화하여 고유 섹션에 저장한다(S100).

[0029] 실행 코드 섹션의 고유 섹션에 저장된 암호화되어 있는 코드를 복호화하고 실행하는 코드를 삽입한다(S200).

[0030] 실행 시(Runtime) 실행 코드 섹션의 쓰기 권한을 얻는 코드를 삽입한다(S300).

[0031] 쓰기 권한을 얻은 후, 런타임에서 정확한 메모리 주소를 구해 실행코드 섹션의 코드에 덮어 쓰는 코드를 삽입한다.(S400).

[0032] 그리고, 바이너리의 헤더와 섹션을 조작하여 디버깅 툴이 기계 명령어(Opcode)를 해석하기 어렵도록 만든다(S500).

[0033] 상기 본 발명의 난독화 과정을 도 2 내지 도 5를 통해 좀 더 상세히 설명하도록 한다.

[0034] 도 2는 본 발명의 실시예에 따른 섹션 코드 암호화 및 저장 과정을 나타낸 흐름도이다.

[0035] 도 2에 도시된 바와 같이, 원본 실행파일 내의 실행 코드가 담겨 있는 섹션의 코드를 암호화하여 고유 섹션에 저장하기 위해 특정 섹션을 생성한다(S102).

[0036] 생성된 특정 섹션으로 실행 섹션의 코드를 모두 이동하고(S104), 해당 섹션에 담긴 코드들을 암호화와 같은 역공학을 방지하기 위한 알고리즘들을 적용한다(S106).

[0037] 상기 Runtime 실행 코드 섹션의 쓰기 권한을 획득하는 단계(S300)는, 메모리 영역의 권한을 설정하는 시스템 콜을 사용하여 실행 코드 섹션의 쓰기 권한을 확보하는 코드를 삽입한다.

[0038] 그리고, 상기 시스템 콜은 운영체제가 제공해주는 기능으로 주어진 파일을 현재 프로세스의 메모리 영역에 매핑(Mapping)시켜 준다.

[0039] 도 3은 본 발명의 실시예에 따른 난독화를 위한 복호화 및 실행코드 삽입 과정을 나타낸 흐름도이다.

[0040] 도 3에 도시된 바와 같이, 상기 고유 섹션의 암호화 코드 복호화 및 실행코드 삽입 단계(S200)는, 실행 코드 섹션의 고유 섹션에 암호화되어 저장되어 있는 코드를 복호화하고 실행하기 위한 코드를 삽입해야 한다.

[0041] 이를 위해 실행 코드 섹션에 고유 섹션의 코드를 복호화와 같은 역공학을 방지하기 위한 알고리즘들을 해제하는 코드를 삽입한다(S202).

[0042] 코드 삽입이 완료되면 특정 세그먼트의 이름을 변경하고(S204), 변경된 이름의 세그먼트 속성을 Opcode의 크기에 맞도록 수정한다(S206).

[0043] 수정이 완료되면 현재 Entry Point인 실행 코드 섹션에 변경된 세그먼트에 들어있는 암호화된 코드를 복호화한 후, 점프하는 코드를 삽입한다(S208).

[0044] 도 4는 본 발명의 실시예에 따른 난독화를 위한 메모리 주소 덮어쓰기 단계의 과정을 나타낸 흐름도이다.

[0045] 도 4에 도시된 바와 같이, 실행 코드 섹션 코드에 메모리 주소를 덮어쓰는 단계(S400)는, 실행 코드 섹션의 주소를 구할 때, 모든 디바이스가 같은 Base Address를 가지고 실행되지 않아 매번 실행코드 섹션의 주소가 달라지는 문제를 해결하기 위한 것이다.

[0046] 이 문제를 해결하기 위해 먼저, 현재 실행파일 헤더의 주소를 구한다(S402).

[0047] 다음으로 실행파일 헤더의 정보를 참조해 섹션 헤더의 위치를 구하고(S404), 섹션 헤더의 주소 + (섹션 헤더의

문자열 인덱스 ㅅ 바이너리 헤더의 크기)를 계산한다(S406).

[0048] 그리고, 계산한 주소 값이 실행 코드 섹션의 주소인지 확인한다(S408).

[0049] 상기 단계 S402 ~ S408을 통해 얻어진 실행 코드 섹션의 주소를 이용하여 Opcode로 작성된 시스템 콜의 인자를 삽입하여 채워준다(S410).

[0050] 도 5는 본 발명의 실시예에 따른 난독화를 위한 바이너리 헤더 및 섹션의 조작 과정을 나타낸 흐름도이다.

[0051] 도 5에 도시된 바와 같이, 바이너리 헤더 및 섹션의 조작 단계(S500)는, 디버거가 참조하는 섹션들의 값을 더미 데이터로 채워서 디버거가 섹션 헤더 테이블을 제대로 참조하지 못하게 하여 프로그램 섹션을 구분하는데 혼란을 주도록 하는 것이다.

[0052] 그러기 위해 먼저, 실행파일 헤더 만큼의 크기를 메모리 영역에 할당 한다(S502).

[0053] 이후, 실행파일 헤더의 주소를 이용해 섹션들의 주소를 구하고(S504), 구한 두 가지 섹션을 더미 데이터로 삽입하여 채운다(S506).

[0054] 매핑한 메모리 맵과 동기화(Synchronize)하고(S508), 매핑한 메모리를 해제한다(S510).

[0055] 그리고, 섹션의 이름을 변경하여 디버거에게 혼란을 주기 위해 바이너리 헤더의 주소를 구한 후(S512), 내부 섹션들을 추출해서 소속된 세그먼트가 변경 대상인지를 확인한다(S514).

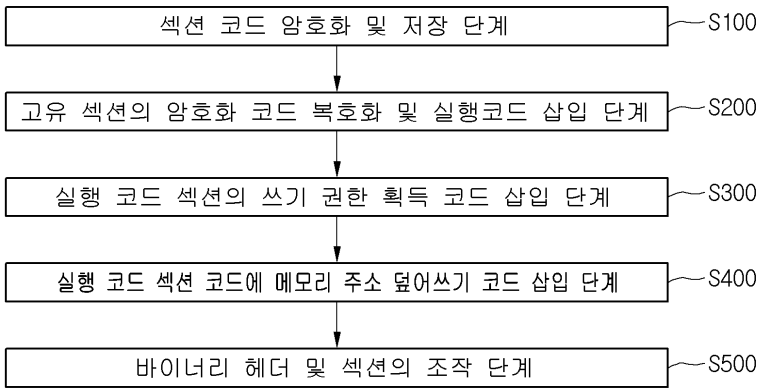
[0056] 마지막으로, 변경 대상 세그먼트의 속성을 비정상적으로 변경하는 과정을 반복한다(S516).

[0057] 상기와 같이, 실행 코드 섹션의 고유 섹션에 암호화 저장 되어 있는 코드를 복호화하고 실행하는 코드를 삽입하고, 원본 실행파일 내의 실행 코드가 담겨 있는 섹션의 코드를 암호화하여 고유 섹션에 저장하고, 실행 시(Runtime) 실행 코드 섹션의 쓰기 권한을 획득하고, 런타임에서 정확한 메모리 주소를 구해 실행코드 섹션의 코드에 덮어쓰고, 디버깅 툴이 기계 명령어(Opcode)를 해석하기 어렵도록 바이너리의 헤더와 섹션을 조작하여 ARM 프로세서 기반의 ELF 파일 구조를 갖는 시스템에서의 실행파일에 난독화 기법을 적용하여 역공학을 통해 프로그램을 분석하기 어렵도록 한다.

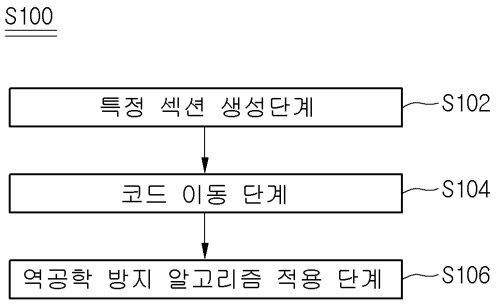
[0058] 상기의 과정을 거쳐 생성된 난독화된 ARM ELF 파일(Obfuscated ARM ELF File)은 변환 구역(Transformation Layer), 암호화된 코드(Encrypted Code) 및 파일 조작(File format Manipulation)이 된 실행파일 및 라이브러리 모듈을 갖게 된다.

도면

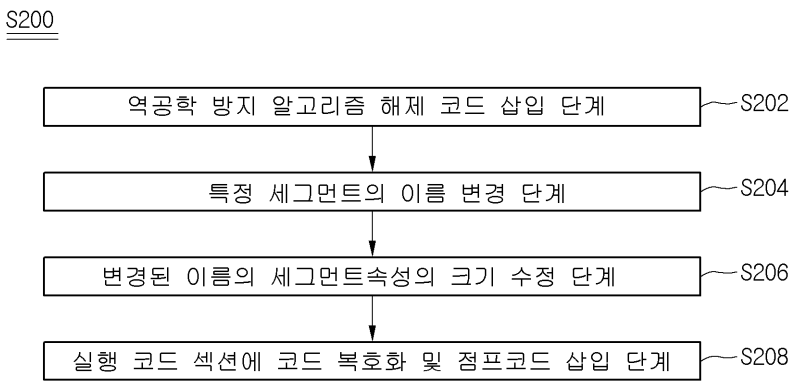
도면1



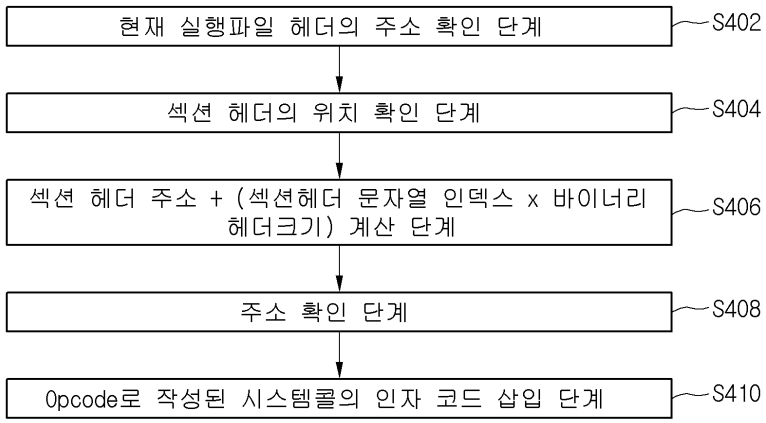
도면2


S100
특정 섹션 생성단계
S102
코드 이동 단계
S104
역공학 방지 알고리즘 적용 단계
S106

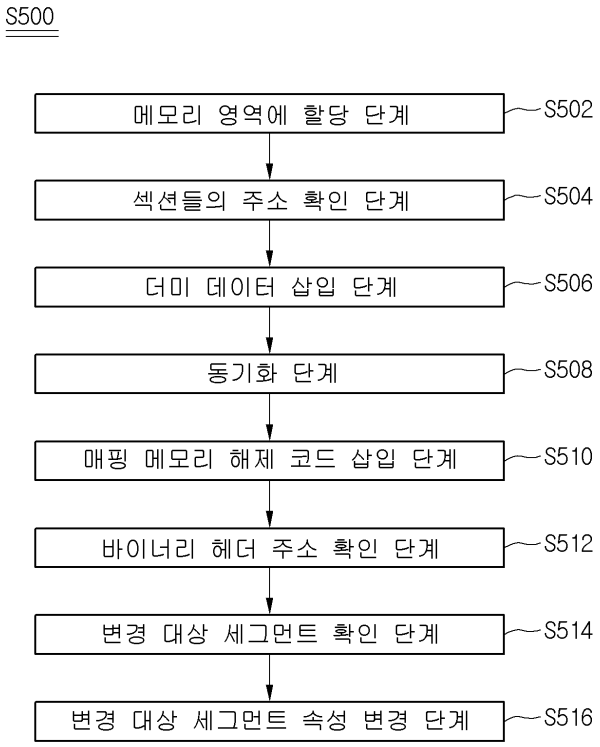
도면3


S200
역공학 방지 알고리즘 해제 코드 삽입 단계
S202
특정 세그먼트의 이름 변경 단계
S204
변경된 이름의 세그먼트속성의 크기 수정 단계
S206
실행 코드 섹션에 코드 복호화 및 점프코드 삽입 단계
S208

도면4


S400
현재 실행파일 헤더의 주소 확인 단계
S402
섹션 헤더의 위치 확인 단계
S404
섹션 헤더 주소 + (섹션헤더 문자열 인덱스 x 바이너리 헤더크기) 계산 단계
S406
주소 확인 단계
S408
0pcode로 작성된 시스템콜의 인자 코드 삽입 단계
S410

도면5


S500
메모리 영역에 할당 단계
S502
섹션들의 주소 확인 단계
S504
더미 데이터 삽입 단계
S506
동기화 단계
S508
매핑 메모리 해제 코드 삽입 단계
S510
바이너리 헤더 주소 확인 단계
S512
변경 대상 세그먼트 확인 단계
S514
변경 대상 세그먼트 속성 변경 단계
S516