



(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(11) 공개번호 10-2021-0063677
(43) 공개일자 2021년06월02일

(51) 국제특허분류(Int. Cl.)
G06F 21/57 (2013.01) G06F 11/36 (2006.01)
G06F 9/455 (2018.01)
(52) CPC특허분류
G06F 21/577 (2013.01)
G06F 11/3604 (2013.01)
(21) 출원번호 10-2019-0152085
(22) 출원일자 2019년11월25일
심사청구일자 2019년11월25일

(71) 출원인
세종대학교산학협력단
서울특별시 광진구 능동로 209 (군자동, 세종대학교)
(72) 발명자
이중혁
충청남도 천안시 서북구 북일로 21, 104동 1503(두정동, e편한세상 두정3차)
권순홍
강원도 강릉시 원대로20번길 2-1, 301호
(74) 대리인
특허법인이상

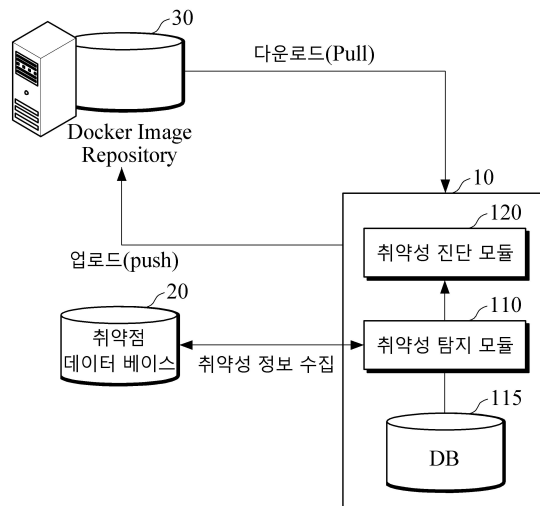
전체 청구항 수 : 총 8 항

(54) 발명의 명칭 도커 이미지 취약성 진단 장치 및 방법

(57) 요약

본 발명은 도커 시스템 기반 클라이언트와 원격 저장소에서의 도커 이미지 취약성을 진단하는 장치에 관한 것으로, 취약점 탐지 모듈과 취약성 진단 모듈을 포함한다. 원격 저장소와 클라이언트는 도커 이미지에 업로드/다운로드시 도커 이미지 취약점 탐지 모듈을 통하여 도커 이미지에 존재하는 패키지의 취약점을 탐지하여 출력하며, 이를 취약성 진단 모듈로 전달한다. 이미지 취약점 탐지 결과를 전달받은 취약성 진단 모듈은 이미지 취약점 탐지 결과로부터 심각도를 파싱 및 카운팅하며, 정의된 심각도별 점수에 따라 이미지 취약점 점수를 계산한다. 계산된 이미지 취약점 점수와 이미지 취약점 임계 점수와 비교하여 초과일 경우, 이미지에 대해 업로드/다운로드를 불가하게 함으로써 클라이언트와 원격 저장소 사이의 안전한 도커 시스템을 수립한다.

대표도 - 도2



(52) CPC특허분류

G06F 9/45508 (2013.01)

이 발명을 지원한 국가연구개발사업

과제고유번호	1711093800
부처명	과학기술정보통신부
과제관리(전문)기관명	(재)기가코리아사업단
연구사업명	범부처GigaKOREA사업(R&D)
연구과제명	(초저지연-총괄/1세부)저지연 융합서비스를 위한 모바일 에지 컴퓨팅 플랫폼 기술
개발	
기 여 율	1/1
과제수행기관명	한국전자통신연구원
연구기간	2019.01.01 ~ 2019.12.31

명세서

청구범위

청구항 1

설정된 주기로 CVE(Common Vulnerabilities and Exposures) 취약점 데이터베이스들로부터 취약점 메타데이터를 수집하여 저장하고, 취약점을 분석할 도커 이미지로부터 수집된 이미지 메타데이터와 저장된 취약점 메타데이터를 이용하여 취약점 탐지 데이터를 생성하여 출력하는 취약점 탐지 모듈; 및

취약점 탐지 데이터를 파싱하여 심각도 정보를 획득하고 심각도 정보를 이용하여 도커 이미지의 취약점 점수를 계산하는 이미지 취약성 진단 모듈;

을 포함하는 도커 이미지 취약성 진단 장치.

청구항 2

제 1 항에 있어서,

취약점 탐지 모듈은 사용자로부터 입력된 취약점 화이트리스트에 포함된 취약점은 승인된 취약점으로 분류하여 취약점 탐지 데이터를 생성하여 출력하고,

취약성 진단 모듈은 승인된 취약점은 제외하고 도커 이미지의 취약점 점수를 계산하는 도커 이미지 취약성 진단 장치.

청구항 3

제 1 항 또는 제 2 항에 있어서,

도커 이미지는 원격 저장소(repository)로부터 다운로드한 이미지이고,

취약성 진단 모듈은 계산된 취약점 점수를 취약점 임계 점수와 비교하여 도커 이미지의 저장 또는 폐기를 결정하는 도커 이미지 취약성 진단 장치.

청구항 4

제 1 항 또는 제 2 항에 있어서,

도커 이미지는 사용자에게 의해 빌드되어 로드된 이미지이고,

취약성 진단 모듈은 계산된 취약점 점수를 취약점 임계 점수와 비교하여 도커 이미지의 원격 저장소로의 업로드 또는 폐기를 결정하는 도커 이미지 취약성 진단 장치.

청구항 5

도커 이미지 취약성 진단 장치에서 실행되는 프로그램 명령어로 적어도 일부가 구현되는 도커 이미지 취약성 진단 방법에 있어서,

설정된 주기로 CVE 취약점 데이터베이스들로부터 취약점 메타데이터를 수집하여 저장하는 단계;

취약점을 분석할 도커 이미지로부터 수집된 이미지 메타데이터와 저장된 취약점 메타데이터를 이용하여 취약점 탐지 데이터를 생성하여 출력하는 단계;

취약점 탐지 데이터를 파싱하여 심각도 정보를 획득하는 단계; 및

심각도 정보를 이용하여 도커 이미지의 취약점 점수를 계산하는 단계;
를 포함하는 도커 이미지 취약성 진단 방법.

청구항 6

제 5 항에 있어서,

취약점 탐지 데이터를 생성하여 출력하는 단계에서 생성되어 출력되는 취약점 탐지 데이터는 사용자로부터 입력된 취약점 화이트리스트에 포함된 취약점은 승인된 취약점으로 분류되어 생성되고,

취약점 점수를 계산하는 단계는 승인된 취약점은 제외하고 도커 이미지의 취약점 점수를 계산하는 도커 이미지 취약성 진단 방법.

청구항 7

제 5 항 또는 제 6 항에 있어서, 상기 방법은

계산된 취약점 점수를 취약점 임계 점수와 비교하여 도커 이미지의 저장 또는 폐기를 결정하는 단계;

를 더 포함하고,

도커 이미지는 원격 저장소(repository)로부터 다운로드한 이미지인 도커 이미지 취약성 진단 방법.

청구항 8

제 5 항 또는 제 6 항에 있어서, 상기 방법은

계산된 취약점 점수를 취약점 임계 점수와 비교하여 도커 이미지의 원격 저장소로의 업로드 또는 폐기를 결정하는 단계;

를 더 포함하고,

도커 이미지는 사용자에 의해 빌드되어 로드된 이미지인 도커 이미지 취약성 진단 방법.

발명의 설명

기술 분야

[0001] 도커 시스템을 위한 보안 진단 장치에 관한 것으로, 보다 상세하게는 도커 이미지에 대한 취약점 검사를 통해 취약성을 진단하는 장치 및 방법에 대한 발명이 개시된다.

배경 기술

[0002] 도커(Docker)는 가장 널리 사용되는 컨테이너 기반 기술 중 하나로써, 이미지 형식으로 애플리케이션(예, Apach, MySQL)을 배포한다. 각 도커 이미지에는 대상 응용 프로그램 소프트웨어, 지원 라이브러리 및 구성 파일이 포함되어 있으며, 도커 컨테이너는 동가 런타임 환경과 여러 응용 프로그램 인스턴스를 패키징하고 배포하는 쉬운 방법을 제공하여 널리 사용되고 있다.

[0003] 도커 이미지의 생성 및 공유를 중심으로 하는 커뮤니티가 등장하고 있으며, 2014년 소개된 도커 허브(Docker Hub)가 응용 프로그램 이미지를 공유하기 위한 대표적인 클라우드 레지스트리 서버이다. 이미지는 저장소(Repository)를 사용하여 배포되므로 버전이 지정된 이미지 개발 및 유지 관리가 가능하다. 또한, 저장소는 다른 저장소에서 분기가 가능하다.

[0004] 대표적인 도커 저장소인 도커 허브에서는 옵션으로 이미지 서명/검증을 수행하고 있으나 악의적인 사용자에 의한 악의적인 이미지 배포는 방지하지 못한다. 사용자가 악의적인 도커 이미지를 로컬 도커 이미지 저장소에 다

온로드 받아 컨테이너로 구동할 시, 개인 정보 누출 등의 사고가 유발될 수 있다.

[0005] 도커 이미지가 쉽게 배포되며 다운로드 가능하다는 특징을 활용하는 도커 이미지의 취약점들이 계속해서 알려지고 있다. 최근에는 도커 이미지를 이용해 암호 화폐 채굴 프로그램을 배포하는 사례가 확인되었다. 이 공격의 경우, 공격 대상의 OS에 상관없이 도커를 사용 중이면 모두 공격이 가능함을 보여주었다. 이러한 문제는 사용자가 다운로드 받은 이미지에 대한 별도의 보안성 검증을 하지 않는 점에서 비롯되므로, 도커 이미지의 취약성 진단에 대한 필요성이 대두되고 있다.

발명의 내용

해결하려는 과제

[0006] 본 발명은 클라이언트가 저장소를 통해 다운로드 받은 도커 이미지에 대하여 취약성을 진단하는 방법을 제공하는 것을 목적으로 한다.

과제의 해결 수단

[0007] 본 발명의 일 양상에 따르는 도커 이미지 취약성 진단 장치는 취약점 탐지 모듈과, 취약성 진단 모듈을 포함한다.

[0008] 취약점 탐지 모듈은 설정된 주기로 CVE(Common Vulnerabilities and Exposures) 취약점 데이터베이스들로부터 취약점 메타데이터를 수집하여 저장하고, 취약점을 분석할 도커 이미지로부터 수집된 이미지 메타데이터와 저장된 취약점 메타데이터를 이용하여 취약점 탐지 데이터를 생성하여 출력하고, 취약성 진단 모듈은 취약점 탐지 데이터를 파싱하여 심각도 정보를 획득하고 심각도 정보를 이용하여 도커 이미지의 취약점 점수를 계산한다.

[0009] 본 발명의 또 다른 양상에 따르면, 취약점 탐지 모듈은 사용자로부터 입력된 취약점 화이트리스트에 포함된 취약점은 승인된 취약점으로 분류하여 취약점 탐지 데이터를 생성하여 출력할 수 있다. 이때, 취약성 진단 모듈은 승인된 취약점은 제외하고 도커 이미지의 취약점 점수를 계산할 수 있다.

[0010] 또한, 도커 이미지가 원격 저장소(repository)로부터 다운로드한 이미지인 경우, 다양한 양상의 도커 이미지 취약성 진단 장치의 취약성 진단 모듈은 계산된 취약점 점수를 취약점 임계 점수와 비교하여 도커 이미지의 저장 또는 폐기를 결정할 수 있다.

[0011] 또한, 도커 이미지가 사용자에게 의해 빌드되어 로드된 이미지인 경우, 다양한 양상의 도커 이미지 취약성 진단 장치의 취약성 진단 모듈은 계산된 취약점 점수를 취약점 임계 점수와 비교하여 도커 이미지의 원격 저장소로의 업로드 또는 폐기를 결정할 수 있다.

[0012] 본 발명의 일 실시 예에 따르면, 도커 이미지 취약성 진단 장치에서 실행되는 프로그램 명령어로 적어도 일부가 구현되는 도커 이미지 취약성 진단 방법은 설정된 주기로 CVE 취약점 데이터베이스들로부터 취약점 메타데이터를 수집하여 저장하는 단계와, 취약점을 분석할 도커 이미지로부터 수집된 이미지 메타데이터와 저장된 취약점 메타데이터를 이용하여 취약점 탐지 데이터를 생성하여 출력하는 단계와, 취약점 탐지 데이터를 파싱하여 심각도 정보를 획득하는 단계와, 심각도 정보를 이용하여 도커 이미지의 취약점 점수를 계산하는 단계를 포함한다.

[0013] 또 다른 실시 예에 따르면, 취약점 탐지 데이터를 생성하여 출력하는 단계에서 생성되어 출력되는 취약점 탐지 데이터는 사용자로부터 입력된 취약점 화이트리스트에 포함된 취약점은 승인된 취약점으로 분류되어 생성되고, 취약점 점수를 계산하는 단계는 승인된 취약점은 제외하고 도커 이미지의 취약점 점수를 계산할 수 있다.

[0014] 또 다른 실시 예에 따르면, 도커 이미지 취약성 진단 방법은 계산된 취약점 점수를 취약점 임계 점수와 비교하여 도커 이미지의 저장 또는 폐기를 결정하는 단계를 더 포함할 수 있고, 이때 도커 이미지는 원격 저장소(repository)로부터 다운로드한 이미지이거나, 사용자에게 의해 빌드되어 로드된 이미지일 수 있다.

발명의 효과

[0015] 본 발명에 의하면 도커 이미지 취약성 진단을 기반으로 하여 악위적인 도커 이미지를 관리함으로써 클라이언트와 저장소간에 이미지 업로드/다운로드에 있어 신뢰할 수 있는 환경을 구축할 수 있는 효과를 달성할 수 있다.

도면의 간단한 설명

[0016] 도 1은 도커 시스템에서 도커 이미지가 업로드/다운로드 및 수정되는 과정을 개념적으로 도시한 것이다.

도 2은 일 양상에 따른 도커 이미지 취약성 진단 장치의 구성을 도시한 블록도이다.

도 3는 도커 이미지 취약성 진단 장치를 이용하여 클라이언트가 저장소에 이미지를 업로드하는 예시적 절차를 도시한 것이다.

도 4는 도커 이미지 취약성 진단 장치를 이용하여 클라이언트가 저장소로부터 이미지를 다운로드하는 예시적 절차를 도시한 것이다.

도 5는 취약성 진단 모듈이 취약점 탐지 데이터를 파싱하는 코드의 예시이다.

도 6은 취약성 진단 모듈이 심각도 정보를 이용하여 취약점 점수를 계산하는 코드의 예시이다.

도 7은 도커 이미지 취약성 진단 방법을 실행하는 예시 스크립트이다.

발명을 실시하기 위한 구체적인 내용

- [0017] 전술한, 그리고 추가적인 양상들은 첨부된 도면을 참조하여 설명하는 실시 예들을 통해 구체화된다. 각 실시 예들의 구성 요소들은 다른 언급이나 상호간에 모순이 없는 한 실시 예 내에서 다양한 조합이 가능한 것으로 이해된다. 블록도의 각 블록은 어느 경우에 있어서 물리적인 부품을 표현할 수 있으나 또 다른 경우에 있어서 하나의 물리적인 부품의 기능의 일부 혹은 복수의 물리적인 부품에 걸친 기능의 논리적인 표현일 수 있다. 때로는 블록 혹은 그 일부의 실체는 프로그램 명령어들의 집합(set)일 수 있다. 이러한 블록들은 전부 혹은 일부가 하드웨어, 소프트웨어 혹은 이들의 결합에 의해 구현될 수 있다.
- [0018] 도커는 리눅스 컨테이너를 기반으로 하는 오픈소스 프로젝트이며, 도커의 등장으로 인해 컨테이너에 대한 연구가 확장되었다. 컨테이너는 기존의 가상 머신과 달리 호스트 운영체제에서 할당 받은 격리된 컨트롤 그룹(control groups, cgroups)과 네임스페이스(namespaces)를 기반으로 동작한다.
- [0019] 네임스페이스(namespaces)는 파일 시스템, 프로세스, 네트워크, IPC, Hostname 등 사용자에게 대한 격리된 공간을 제공해주는 리눅스 커널의 기능이며, 컨트롤 그룹(cgroups)은 리눅스 시스템 자원(예: 메모리, CPU, I/O, 네트워크 등)에 대한 제어를 제공하고 자원에 대한 그룹을 생성해 소유권을 지정할 수 있는 리눅스 커널의 기능이다.
- [0020] 도커는 이미지의 형태로 컨테이너 실행에 필요한 파일과 설정값 등을 포함시킨다. 도커 이미지의 경우, 도커 시스템을 사용하는 사용자들은 누구나 업로드/다운로드가 가능하다.
- [0021] 도 1은 도커 시스템에서 도커 이미지가 업로드/다운로드 및 수정되는 과정을 개념적으로 도시한 것이다. 도커에서는 이미지를 기반으로 하는 컨테이너 환경이 구성된다. 해당 이미지를 원격 저장소(30)에 업로드하여 공유하게 된다. 원격 저장소(30)는 공개 또는 비공개로 구성하는 것이 가능하다. 저장소를 지정하지 않으면, 디폴트로 도커 허브(Docker hub)로 지정되며 파라미터 설정을 통해 저장소를 사설 저장소로 설정하여 업로드가 가능하다.
- [0022] 도 1에 도시된 도커 이미지 업로드/다운로드 및 수정되는 과정에 나타난 바와 같이 클라이언트는 특정 이미지를 업로드/다운로드하기 위해 명령을 수행할 때 도커 엔진을 통해 이미지를 업로드/다운로드 할 수 있다.
- [0023] 하지만, 도 1에서 확인할 수 있듯이, 클라이언트는 원격 저장소(30)에서 로컬 환경으로 도커 이미지를 다운로드하거나 로컬에서 빌드한 도커 이미지를 원격 저장소로 업로드할 경우, 도커 이미지 보안을 위한 별도의 인증 과정을 거치지 않는다.
- [0024] 도 2은 일 양상에 따른 도커 이미지 취약성 진단 장치의 구성을 도시한 블록도이다. 본 발명의 일 양상에 따르는 도커 이미지 취약성 진단 장치(10)는 취약점 탐지 모듈(100)과, 취약성 진단 모듈(110)을 포함하여 도커 이미지에 존재하는 취약점을 탐지하고 도커 이미지의 취약성을 진단한다.
- [0025] 취약점 탐지 모듈(100)은 퍼스널 컴퓨터, 서버 등의 컴퓨팅 장치에서 실행되는 프로그램 명령어로 구현될 수 있으며, 설정된 주기로 CVE(Common Vulnerabilities and Exposures) 취약점 데이터베이스(20)들로부터 취약점 메타데이터를 수집하여 저장하고, 취약점을 분석할 도커 이미지로부터 수집된 이미지 메타데이터와 저장된 취약점 메타데이터를 이용하여 취약점 탐지 데이터를 생성하여 출력한다.
- [0026] 취약점 탐지 모듈(100)은 각 도커 이미지의 보안 취약성을 분석하기 위해 도커 이미지로부터 취약성 분석에 필요한 취약점 메타데이터를 외부에 존재하는 취약점 데이터베이스(20)로부터 수집하여 모듈 내 데이터베이스(115)에 저장한다. 취약점 데이터베이스(20)로는 CVE 취약점 데이터 베이스, Ubuntu CVE Tracker, Debian Security Bug Tracker, Red Hat Security Data와 같은 데이터베이스들이 존재하며 이들 데이터베이스로부터 주

기적으로 도커 이미지에 대한 취약점 정보를 수집한다. 일 예로, 취약점 탐지 모듈(100)은 코어오에스 클레어(CoreOS Clair) 오픈소스 툴 또는 이를 수정하여 개발한 툴을 사용할 수 있고, 이 경우 취약점 탐지 모듈(100)은 Vulnerability Updater를 통해 취약점 데이터베이스(20)로부터 주기적으로 취약점 메타데이터를 수집하여 취약점 탐지 모듈(100)이 사용하는 데이터베이스(115)에 저장한다. 코어오에스 클레어는 하나의 예시로 앵커(Anchore), 앱아머(Apparmor), 다그다(Dagda) 등의 오픈소스 툴 또는 이들을 수정하여 개발한 툴을 사용할 수 있으며, 상용 툴을 사용할 수도 있다.

[0027] 취약점 탐지 모듈(100)이 이미지별 패키지 취약성 분석을 위해 CVE 취약점 데이터 베이스, Ubuntu CVE Tracker, Debian Security Bug Tracker, Red Hat Security Data 등의 취약점 데이터베이스(20)로부터 수집하는 취약점 메타데이터는 <표 2>에 기재된 항목들을 포함한다.

[0028] 취약점 탐지 모듈(100)이 각 이미지의 보안 취약성을 분석하기 위해 이미지로부터 수집하는 이미지 메타데이터는 <표 1>에 기재된 항목들을 포함한다.

[0029] 취약점 탐지 모듈(100)은 <표 1> 및 <표 2>의 기재된 항목들을 포함하는 메타데이터를 수집한 경우, 도커 이미지에 대한 취약점 탐지 과정을 수행한다. 취약점 탐지 과정은 오픈소스 등으로 공지되어 있으므로 상세한 설명은 생략한다.

[0030] 취약점 탐지 모듈(100)은 취약점 탐지 과정을 통해 분석한 이미지의 레이어 ID, 이미지에 존재하는 총 취약점 개수, 승인되지 않은 취약점 개수, CVE 심각도, 패키지 이름, 패키지 버전, CVE 정보 설명을 포함하는 취약점 탐지 데이터를 출력한다.

[0031] 취약점 탐지 모듈(100)은 사용자로부터 입력된 취약점 화이트리스트(Whitelist)에 포함된 취약점은 승인된 취약점으로 분류하여 취약점 탐지 데이터를 생성하여 출력할 수 있다.

표 1

[0032]

데이터 필드	설명
Image ID	각 고유 이미지에 대해 256비트 길이의 ID
Image Name	특정 이름 정책을 따르는 각 이미지의 식별자
Last Update Time	이미지에 대한 정확한 데이터 및 최종 업데이트 시간
Layer ID	각 레이어의 고유 ID 및 레이어 간의 관계
Commands	이미지 빌드 관련 명령

표 2

[0033]

데이터 필드	설명
Timestamp	취약점 탐지툴(예, Clair)에 의한 정확한 분석 시간
Vulnerability ID	취약성을 식별하기 위한 고유한 CVE 식별자
Severity Ranking	각 취약점의 심각도
Description of CVE	식별된 각 취약점에 대한 설명
Associated Packages	각 취약점과 관련된 패키지의 이름과 정확한 버전
Layer ID	취약점이 존재하는 특정 이미지 layer

[0034] 취약성 진단 모듈(110)은 취약점 탐지 데이터를 파싱하여 심각도 정보를 획득하고 심각도 정보를 이용하여 도커 이미지의 취약점 점수를 계산한다.

[0035] 또한, 취약성 진단 모듈(110)은 취약점 탐지 모듈(100)로부터 출력된 정보를 기반으로 도커 이미지에 대해 업로드 또는 다운로드 가능 여부를 결정하는 모듈이다.

[0036] 취약성 진단 모듈(110)은 이미지 취약점 탐지 모듈(100)로부터 출력된 정보 중 도커 이미지에 포함된 패키지 취약점의 심각도를 기반으로 취약점 점수를 계산하며, 계산된 취약점 점수가 취약점 임계 점수 이상일 경우, 해당 도커 이미지는 업로드/다운로드가 불가하며, 취약점 임계 점수 미만일 경우에만 해당 도커 이미지는 업로드/다운로드가 가능하다. 이때, 취약점 임계 점수는 사용자에게 의해 설정이 가능하다.

수학식 1

$$Img_{score} = \sum_{i=0}^{r_{sv}} (w_{sv} \times \alpha)$$

[0037]

[0038]

sv 는 도커 이미지 패키지 취약점 심각도를 의미하고, r_{sv} 는 해당 심각도의 취약점 수를 의미하고, w_{sv} 는 심각도의 가중치를 의미하고, α 는 화이트리스트에 의한 승인 여부를 의미한다.

[0039]

취약성 진단 모듈(110)에 의한 도커 이미지 취약점 점수 계산은 <수학식1>을 따라 계산되며, 해당 패키지의 특정 취약점이 사용자에게 의해 설정된 화이트리스트에 포함된 경우, 취약점의 승인 여부를 나타내는 값 α 는 0이 된다. 이외에 화이트리스트에 설정되지 않은 이미지에 존재하는 패키지의 취약점의 경우, α 는 1이 된다. 취약성 진단 모듈(110)은 <수학식1>을 따라 도커 이미지 취약점 점수를 계산한 후 취약점 임계 점수와 해당 값을 비

$$Img_{score} > \theta$$

교한다. 도커 이미지 취약점 점수와 취약점 임계 점수를 비교 결과 $Img_{score} > \theta$ 와 같이 도커 이미지 취약점 점수가 취약점 임계 점수(θ)를 초과하는 경우, 해당 이미지를 업로드 또는 다운로드할 수 없다. 즉, 다운로드한 이미지의 경우 설치하지 않고 폐기하고, 업로드되어야 할 이미지의 경우 업로드 하지 않고 폐기한다.

[0040]

본 발명의 일 실시 예에 따르면, 도커 이미지 취약성 진단 장치(10)에서 실행되는 프로그램 명령어로 적어도 일부가 구현되는 도커 이미지 취약성 진단 방법은 설정된 주기로 CVE 취약점 데이터베이스(20)들로부터 취약점 메타데이터를 수집하여 저장하는 단계와, 취약점을 분석할 도커 이미지로부터 수집된 이미지 메타데이터와 저장된 취약점 메타데이터를 이용하여 취약점 탐지 데이터를 생성하여 출력하는 단계와, 취약점 탐지 데이터를 파싱하여 심각도 정보를 획득하는 단계와, 심각도 정보를 이용하여 도커 이미지의 취약점 점수를 계산하는 단계를 포함한다.

[0041]

또 다른 실시 예에 따르면, 취약점 탐지 데이터를 생성하여 출력하는 단계에서 생성되어 출력되는 취약점 탐지 데이터는 사용자로부터 입력된 취약점 화이트리스트에 포함된 취약점은 승인된 취약점으로 분류되어 생성되고, 취약점 점수를 계산하는 단계는 승인된 취약점은 제외하고 도커 이미지의 취약점 점수를 계산할 수 있다.

[0042]

또 다른 실시 예에 따르면, 도커 이미지 취약성 진단 방법은 계산된 취약점 점수를 취약점 임계 점수와 비교하여 도커 이미지의 저장 또는 폐기를 결정하는 단계를 더 포함할 수 있고, 이때 도커 이미지는 원격 저장소(30, repository)로부터 다운로드한 이미지이거나, 사용자에게 의해 빌드되어 로드된 이미지일 수 있다.

[0043]

도 3는 도커 이미지 취약성 진단 장치를 이용하여 클라이언트가 원격 저장소에 이미지를 업로드하는 예시적 절차를 도시한 것이다. 도 3을 참조하여 설명하면, 취약점 탐지 모듈(100)은 취약점 데이터베이스(20)에 취약점 메타데이터 정보를 요청한다(S1000). 요청을 받은 취약점 데이터베이스(20)는 Ubuntu, CentOS 등의 취약점 메타데이터를 취약점 탐지 모듈(100)에 송신하며(S1020), 취약점 탐지 모듈(100)은 수신한 취약점 메타데이터를 취약점 메타데이터 DB에 저장한다(S1040). S1000 내지 S1040 절차는 설정된 주기에 따라 반복하여 수행된다. 클라이언트가 업로드를 위해 도커 엔진을 통해 빌드한 도커 이미지를 취약성 진단을 위해 취약점 탐지 모듈(100)에 로드한다(S1060). 취약점 탐지 모듈(100)은 로드된 도커 이미지로부터 <표 1>에 기재된 이미지 메타데이터를 추출하여 수집한다(S1080). 업로드할 도커 이미지에 대해 <표 1>의 메타데이터를 추출한 정보를 기반으로 취약점 탐지 모듈(100)은 취약점 메타데이터 DB에서 해당 도커 이미지와 관련한 취약점 메타데이터를 추출하여 취약점 탐지 데이터를 생성하여 출력한다(S1100). 취약점 탐지 모듈(100)은 도커 이미지 취약점 탐지 결과인 취약점 탐지 데이터를 도커 이미지 취약점 점수 계산 및 업로드 가능 여부를 결정하기 위해 취약성 진단 모듈(110)로 송신한다(S1120). 취약점 탐지 데이터를 수신한 취약성 진단 모듈(110)은 취약점 탐지 데이터로부터 심각도 관련 정보를 파싱 및 카운팅하고(S1140), <표 3>에 정의된 심각도별 점수를 반영하여 <수학식1>에 따라 도커 이미지 취약점 점수를 계산하여(S1160) 취약점 임계 점수와 비교를 수행한다(S1180). 만약 계산된 취약점 점수

$$Img_{score}$$

가 취약점 임계 점수를 초과하는 경우, 해당 이미지를 삭제하는 명령을 수행하고 취약성 진단 모듈(110)은 도커 이미지 취약성 진단 결과를 클라이언트에게 송신하여 절차를 종료한다(미도시). 만약 계산된 취약

Img_{score}

점 점수 가 취약점 임계 점수 미만인 경우, 취약성 진단 모듈(110)은 클라이언트에게 도커 이미지 취약성 진단 결과를 송신하고(S1200), 안전한 이미지라는 결과를 수신한 클라이언트는 이미지 업로드 명령을 docker cli에 전달한다(S1220). docker cli는 해당 명령을 docker engine에 post 통신하며(S1240), 해당 명령을 수신한 docker engine은 원격 저장소(30)에 도커 이미지 업로드 요청을 수행한다(S1260). 요청을 수신한 원격 저장소(30)는 이미지 업로드가 완료되었다는 메시지를 클라이언트에게 송신한다.

표 3

Notation	Definition	Example	Example Value
sv	도커 이미지 취약점 심각도	unknown	-
		negligible	-
		low	-
		medium	-
		high	-
		critical	-
r_{sv}	sv 의 수	$r_{unknown}$	-
		$r_{negligible}$	-
		r_{low}	-
		r_{medium}	-
		r_{high}	-
		$r_{critical}$	-
w_{sv}	sv 의 가중치	$w_{unknown}$	0.0
		$w_{negligible}$	0.5
		w_{low}	2
		w_{medium}	5.45
		w_{high}	7.95
		$w_{critical}$	9.5
Img_{score}	도커 이미지 취약점의 총 심각도 점수	-	imiell/bad-dockerfile:178.65
$parse(r_{sv}, result.txt)$	result.txt로부터 r_{sv} 과성	-	-
a	도커 이미지 취약점 승인 여부 값	-	approval value: 0 Unapproval value: 1
θ	도커 이미지 취약점 임계 점수	-	150

도 4는 도커 이미지 취약성 진단 장치를 이용하여 클라이언트가 저장소로부터 이미지를 다운로드하는 예시적 절차를 도시한 것이다. 도 4를 참조하여 설명하면, 취약점 탐지 모듈(100)은 취약점 데이터베이스(20)에 취약점 메타데이터 정보를 요청한다(S2000). 요청을 받은 취약점 데이터베이스(20)는 Ubuntu, CentOS 등의 취약점 메타데이터를 취약점 탐지 모듈(100)에 송신하며(S2020), 취약점 탐지 모듈(100)은 수신한 취약점 메타데이터를 취약점 메타데이터 DB(115)에 저장한다(S2040). S2000 내지 S2040 절차는 설정된 주기에 따라 반복하여 수행된다. 클라이언트는 도커 이미지를 원격 저장소(30)로부터 다운로드하기 위해 도커 이미지 다운로드 명령을 docker cli에 전달한다(S2060). docker cli는 해당 명령을 post 통신을 통해 docker engine에게 송신하며(S2080), 해당 명령을 수신한 docker engine은 원격 저장소에 다운로드 받을 도커 이미지를 요청한다(S2100). 요청을 수신한 원격 저장소(30)는 다운로드 승인을 통해 클라이언트의 docker engine 이미지 저장소에 요청한 이미지를 다운로드한다(S2120). 원격 저장소(30)로부터 다운받은 이미지는 취약성 진단을 위해 docker engine 이미지 저장소로부터 취약점 탐지 모듈(100)에 로드된다(S2140). 취약점 탐지 모듈(100)은 로드된 도커 이미지로부터 <표 1>에 기재된 이미지 메타데이터를 추출하여 수집한다(S2160). 다운로드된 도커 이미지에 대해 <표 1>의 메타데이터를 추출한 정보를 기반으로 취약점 탐지 모듈(100)은 취약점 메타데이터 DB(115)에서 해당 도커 이미지와

관련한 취약점 메타데이터를 추출하여 취약점 탐지 데이터를 생성하여 출력한다(S2180). 취약점 탐지 모듈(100)은 도커 이미지 취약점 탐지 결과인 취약점 탐지 데이터를 도커 이미지 취약점 점수 계산 및 폐기 여부를 결정하기 위해 취약성 진단 모듈(110)로 송신한다(S2200). 취약점 탐지 데이터를 수신한 취약성 진단 모듈(110)은 취약점 탐지 데이터로부터 심각도 관련 정보를 파싱 및 카운팅하고(S2220), <표 3>에 정의된 심각도별 점수를 반영하여 <수학식1>에 따라 도커 이미지 취약점 점수를 계산하여(S2240) 취약점 임계 점수와 비교를 수행한다

Img_{score}

(S2260). 만약 계산된 취약점 점수가 취약점 임계 점수를 초과하는 경우, 해당 이미지를 삭제하는 명령을 수행하고(S2280) 취약성 진단 모듈(110)은 도커 이미지 취약성 진단 결과를 클라이언트에게 송신하여 절차를 종료한다(S2300).

- [0046] 도 5는 취약성 진단 모듈이 취약점 탐지 데이터를 파싱하는 코드의 예시이다. 도 5에 도시된 예시 코드는 파이썬으로 구현되어 있으나, 구현 언어에 제한이 있는 것은 아니다. 도 5에 도시된 예시 코드를 간략하게 설명하면, Open 함수는 이미지 취약점 탐지 모듈(100)을 통해 얻은 이미지 취약점 결과 값파일을 여는 함수이다.
- [0047] Findall 함수는 취약점 탐지 모듈(100)을 통해 얻은 이미지 취약점 결과인 취약점 탐지 데이터로부터 심각도에 대한 내용을 파싱하는 함수이다. 이를 통해 도커 이미지 취약점 탐지 데이터에 존재하는 심각도 정보(예: unknown, negligible, low, medium, high, critical, defcon)를 파싱한다.
- [0048] frequency.get 함수는 Findall 함수를 통해 파싱한 심각도 별 개수를 카운팅하는 함수이다. 이를 통해 이미지 취약점 탐지 데이터가 저장되어 있는 파일로부터 심각도별 개수를 저장한다.
- [0049] frequency 함수는 frequency.get 함수를 통해 이미지 취약점 탐지 데이터가 저장되어 있는 파일로부터 얻은 심각도별 개수에 각 심각도별 점수를 계산하는 함수이다. 이를 통해 심각도 별 점수가 계산된다.
- [0050] 도 6은 취약성 진단 모듈이 심각도 정보를 이용하여 취약점 점수를 계산하는 코드의 예시이다. 도 5에 도시된 예시 코드는 파이썬으로 구현되어 있으나, 구현 언어에 제한이 있는 것은 아니다. 도 5에 도시된 예시 코드를 간략하게 설명하면, open 함수는 도 6에서 생성한 이미지 취약점 심각도 별 점수가 저장되어 있는 파일을 여는 함수이다.
- [0051] readlines 함수는 open 함수에서 얻은 이미지 취약점 심각도 별 점수가 저장되어 있는 파일을 라인별로 읽어오는 함수이다.
- [0052] total += float(num)은 이미지 취약점 심각도 별 점수에 대해 덧셈 연산을 수행하는 코드이다.
- [0053] if total > θ 는 total += float(num)을 통해 구해진 이미지 취약점 심각도 점수를 이미지 취약점 임계 점수(θ)와 비교하여 임계 점수 초과일 경우, "Delete your docker image"를 출력하고, 임계 점수 미만일 경우, "It's a Safe docker image"를 출력한다.
- [0054] print total은 total+=float(num)을 통해 구해진 이미지 취약점 심각도 점수에 대해 출력한다.
- [0055] 도 7은 도커 이미지 취약성 진단 방법을 실행하는 예시 스크립트이다. 도 7에 도시된 스크립트 코드의 주요 코드를 간략하게 설명한다.
- [0056] docker-compose run -rm clair-scanner \$1> dockerimage.txt는 이미지 취약점 탐지 모듈(100)의 구현 예시인 오픈소스 툴(clair)을 이용하여 클라이언트가 다운로드 받으려는 도커 이미지(\$1)에 대해 취약점 탐지를 수행하여 결과 값인 취약점 탐지 데이터를 dockerimage.txt로 저장하는 명령이다.
- [0057] sed '/Unapproved/!d' dockerimage.txt>result.txt는 dockerimage.txt로 저장된 취약점 탐지 데이터에 대해 클라이언트로부터 승인되지 않은 취약점만을 파싱하여 result.txt로 저장하는 명령이다.
- [0058] while IFS= read -r line은 저장된 파일을 라인별로 읽어 오는 코드이다.
- [0059] if ["\$line" == "Delete your docker image"]; then docker rmi \$1은 계산된 취약점 점수가 취약점 임계 점수를 초과하는 경우에 출력되는 "Delete your docker image" 문자열이 파일에 포함되어 있는 경우, docker rmi \$1(다운받으려는 도커 이미지) 명령을 통해 해당 이미지를 삭제하는 명령이다.
- [0060] 도 7은 도커 이미지 취약성 진단 장치의 취약점 탐지 모듈이 Clair로 구현된 예시이며 취약성 진단 하는 방법의 구현이 도 7의 예시와 같이 스크립트로 작성되어야 하는 것은 아니며, 다른 프로그래밍 언어로 작성될 수 있는

며 스크립트 또한 하나의 예시에 불과하므로 다른 절차로 스크립트가 구성될 수도 있다. 또한, 취약점 탐지 모듈의 구현이 Clair에 한정된 것이 아니며, 다양한 취약점 탐지 툴을 이용하여 구현될 수 있다.

[0061] 일 실시 예에 따른 도커 이미지 취약성 진단 방법은 <표 3>에 정의된 심각도별 점수에 따라 취약성을 진단하였으며, 사용자의 기준에 따라 <표 3>에 정의된 심각도별 점수는 다르게 정의될 수 있다. 또한, 이미지 취약점 임계 점수인 θ 의 경우에도 클라이언트의 기준에 따라 다르게 정의될 수 있다.

[0062] 이상에서 본 발명을 첨부된 도면을 참조하는 실시 예들을 통해 설명하였지만 이에 한정되는 것은 아니며, 이들로부터 당업자라면 자명하게 도출할 수 있는 다양한 변형 예들을 포괄하도록 해석되어야 한다. 특허청구범위는 이러한 변형 예들을 포괄하도록 의도되었다.

부호의 설명

[0063] 10: 도커 이미지 취약성 진단 장치

100: 취약점 탐지 모듈

110: 취약성 진단 모듈

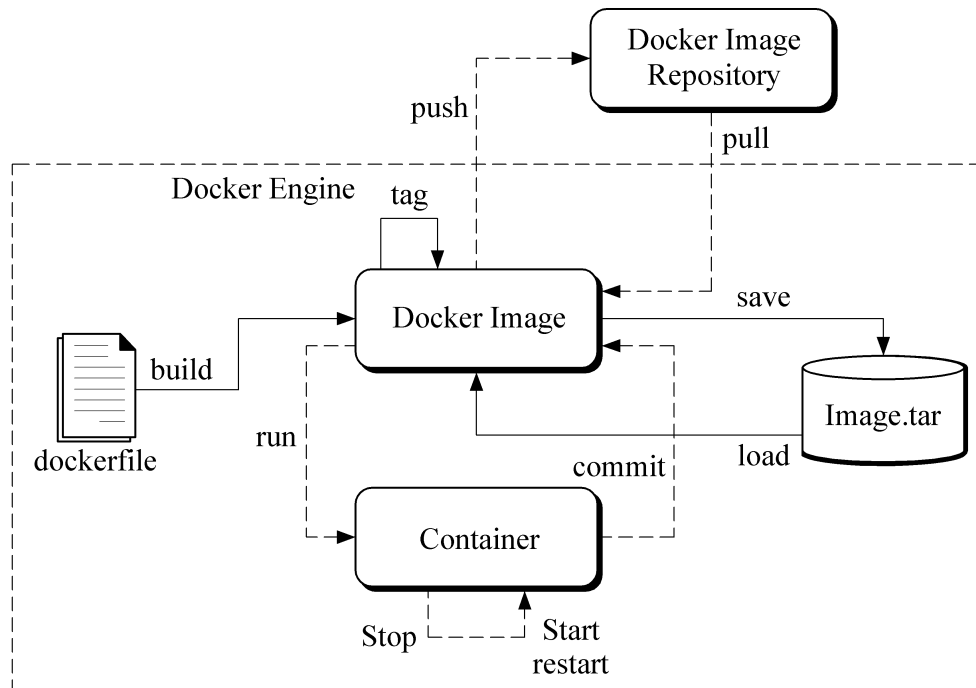
115: 취약점 메타데이터 데이터베이스

20: 취약점 데이터베이스

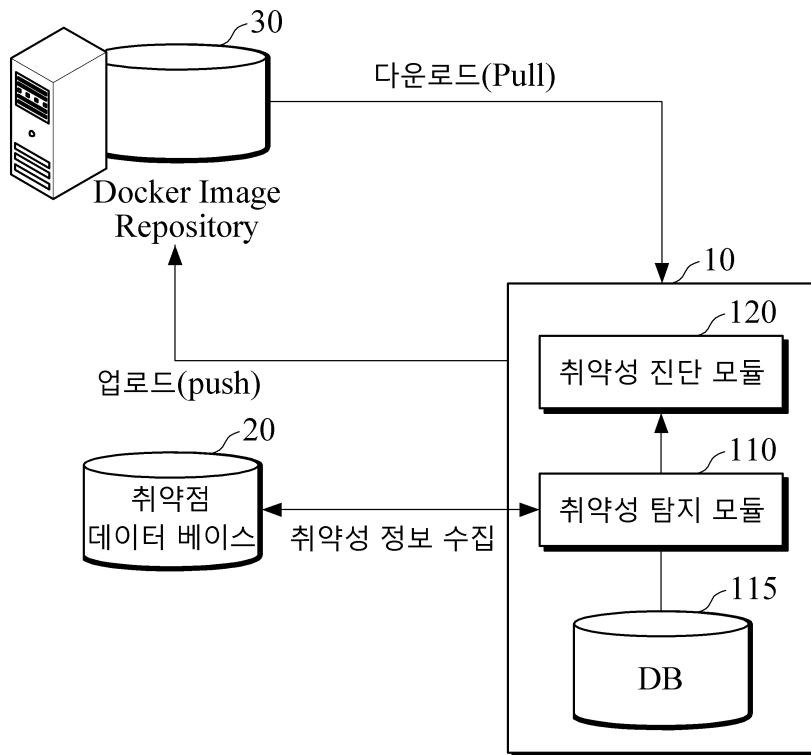
30: 원격 저장소

도면

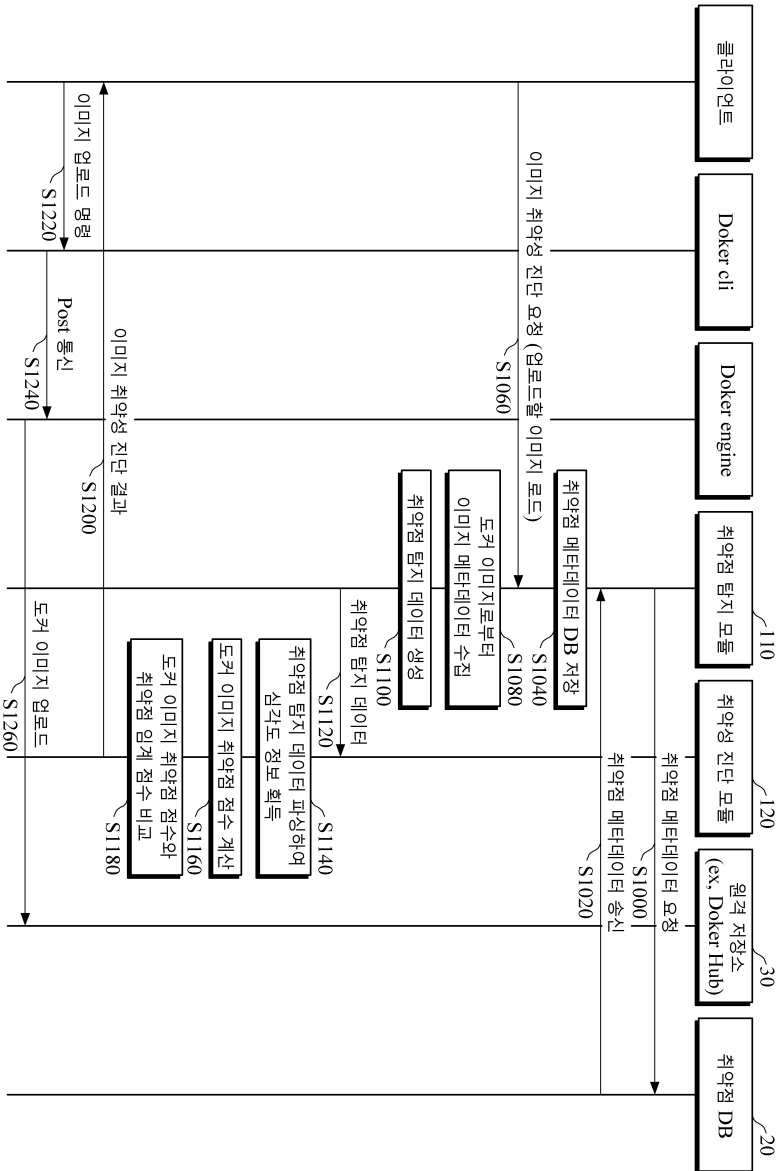
도면1



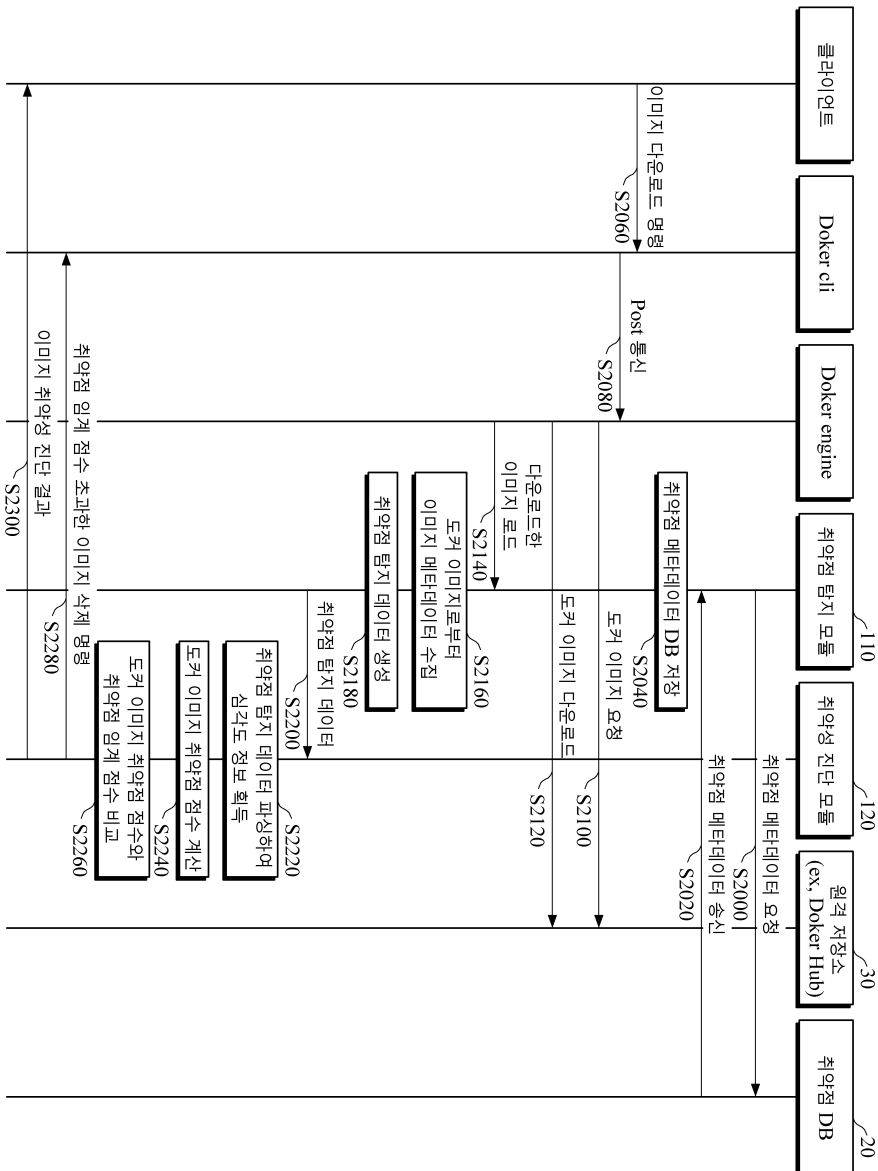
도면2



도면3



도면4



도면5

```
import re
import string

frequency = {}
document_text = open('result.txt', 'r')
text_string = document_text.read().lower()

match_pattern = re.findall(r'\b[unknown]{7,10}\b', text_string)
match_pattern1 = re.findall(r'\b[negligible]{10}\b', text_string)
match_pattern2 = re.findall(r'\b[low]{3,10}\b', text_string)
match_pattern3 = re.findall(r'\b[medium]{6,10}\b', text_string)
match_pattern4 = re.findall(r'\b[high]{4,10}\b', text_string)
match_pattern5 = re.findall(r'\b[critical]{7,10}\b', text_string)
match_pattern6 = re.findall(r'\b[defcon]{6,10}\b', text_string)

for word in match_pattern:
    count = frequency.get(word,0)
    frequency[word] = count + 1 * 0.0 * 1
for word1 in match_pattern1:
    count1 = frequency.get(word1,0)
    frequency[word1] = count1 + 1 * 0.5 * 1
for word2 in match_pattern2:
    count2 = frequency.get(word2,0)
    frequency[word2] = count2 + 1 * 2 * 1
for word3 in match_pattern3:
    count3 = frequency.get(word3,0)
    frequency[word3] = count3 + 1 * 5.45 * 1
for word4 in match_pattern4:
    count4 = frequency.get(word4,0)
    frequency[word4] = count4 + 1 * 7.95 * 1
for word5 in match_pattern5:
    count5 = frequency.get(word5,0)
    frequency[word5] = count5 + 1 * 9.5 * 1
for word6 in match_pattern6:
    count6 = frequency.get(word6,0)
    frequency[word6] = count6 + 1 * 10 * 1

frequency_list = frequency.keys()

for words in frequency_list:
    print frequency[words]
```

도면6

```
import re
import string

allNums = []
total = 0
with open(r"severity.txt", "r+") as f:
    data = f.readlines()
    for line in data:
        allNums += line.strip().split(" ")
    for num in allNums:
        total += float(num)
    if total > 150:
        print "Delete your docker image"
    else:
        print "It's a authorized docker image"
    print total
```

도면7

```
#!/bin/bash

if [ $? -eq 0 ];then
    docker pull $1
    if [ $? -eq 0 ];then
        docker-compose run --rm clair-scanner $1>docker_imagetest.txt
        sed '/Unapproved/!d' "$docker_imagetest.txt">result.txt
        python confirm.py>severity.txt
        python tx.py>score.txt
        file="score.txt"
        while IFS= read -r line
        do
            if [ "$line" == "Delete your docker image" ]; then
                docker rmi $1
            fi
        done < "$file"
    fi
fi
```