Vulner

팀　　명　강 김 홍 차

지도교수　양환석 교수님

팀　　장　김　　솔

팀　　원　강　채　린
차　윤　지
홍　예　진

2024. 11.

중부대학교 정보보호학과

목 차

1. 서론

1.1. 연구 배경

서버 보안의 중요성은 개인정보부터 기업의 중요 데이터까지 다양한 정보를 안전하게 보호해야 한다는 요구사항에서 비롯된다. 최근 리눅스 서버에서 발생한 보안 취약점에 대한 보도는 이 문제의 심각성을 강조하고 있다. 따라서 이러한 상황에서 서버 보안은 기업 및 일반 사용자 모두에게 중요한 이슈가 되었으며, 이를 해결하기 위해 서버 취약점을 진단할 수 있는 도구의 필요성이 대두되고 있다. 이에 기업은 간이 테스트를, 일반 사용자들은 쉽게 취약점을 진단할 수 있는 웹페이지를 제작하고자 한다.

1.2. 연구 목적 및 주제 선정

본 연구의 목적은 서버 취약점 및 모니터링 진단 도구를 개발하고 사용자들이 쉽게 진단을 요청하여 결과를 명확하게 알 수 있는 웹사이트를 제공하는 데 있다.

이를 통해 복잡한 사이버 위협 환경에 대응할 수 있도록, 과학기술정보통신부와 KISA에서 제시한 ‘주요정보통신기반시설 기술적 취약점 분석 평가 방법 상세 가이드’(이하 주통기)를 기반으로 진단 결과와 솔루션을 제공한다. 또한, 본 연구는 진단 프로세스를 자동화하여 주기적인 보안 진단 빈도를 높이는 것을 목표로 한다. 이를 위해, 누구나 쉽게 진단 요청을 할 수 있는 게시판 기능을 제공하고, 자동화된 스크립트를 통해 생성된 결과를 웹 사이트에서 표와 그래프로 시각화하여 가시성을 높였다. 더불어, 진단 결과 파일을 제공함으로써 이전 결과와 비교하여 변화된 사항이나 새로운 보안 문제를 쉽게 파악할 수 있도록 제작하였다.

2. 관련연구

2.1. 사용 기술 및 DBMS

2.1.1. React

UI를 구축하기 위한 JavaScript 라이브러리이다. 이 라이브러리는 코드의 재사용성과 유지 보수성을 높인다. React는 JSX(JavaScript XML)라는 문법을 사용하여 JavaScript 코드 내에서 HTML을 작성할 수 있게 해준다. 또한, 단방향 데이터 바인딩(One-Way Binding) 개념을 통해 데이터 흐름을 명확하게 관리할 수 있다. 가상 돔(Virtual DOM) 기술을 활용해 실제 DOM 변경을 최소화하고 성능을 개선하는 특징이 있다. 메모리 관리와 성능이 뛰어나며 다른 프레임워크, 라이브러리와 혼용이 가능하다는 장점이 있다.

2.1.2. Node.js

서버 사이드에서 JavaScript를 실행할 수 있는 환경을 제공하는 도구이다. JavaScript는 웹 브라우저 내에서만 작동했으나, Node.js를 통해 서버에서도 활용할 수 있게 되었다. npm(Node Package Manager)이라는 패키지 관리자를 통해 다양한 라이브러리와 패키지를 쉽게 설치하고 관리할 수 있어 웹 개발에 자주 사용된다.

2.1.3. Express

Node.js 환경에서 쉽게 개발할 수 있도록 도와주는 웹 프레임워크이다. 특징으로 강력한 라우팅 기능과 미들웨어 구조를 제공하여 요청과 응답을 간편하게 처리할 수 있다. 다양한 HTTP 요청 메서드에 대응하는 라우트를 설정할 수 있고, JSON 데이터 처리에 용이해 개발 생산성을 높이는 특징이 있다.

2.1.4. Express-Session

사용자의 로그인 상태를 관리하는 데 유용한 미들웨어이다. 사용자가 로그인한 경우, 다른 웹사이트로 이동하더라도 로그인 상태를 유지할 수 있도록 도와준다. 이를 위해 로그인 정보를 세션에 저장하고, 세션 ID를 쿠키를 통해 관리한다.

2.1.5. Nodemailer

이메일 전송을 위한 라이브러리이다. 이 라이브러리를 사용하면 Node.js 환경에서 이메일을 쉽게 발송할 수 있다. SMTP(Simple Mail Transfer Protocol) 설정을 통해 이메일을 전송할 수 있으며, Naver, Google 등 다양한 이메일 서비스 제공업체와 호환된다..

2.1.6. Passport, Passport-Local

사용자가 입력한 아이디, 비밀번호를 데이터베이스에 저장되어 있는 값과 비교하여 인증 절차를 진행하는 미들웨어다.

2.1.7. MongoDB

문서 지향 데이터베이스로, 데이터를 JSON 형식으로 저장한다. 이 방식은 개발자가 데이터를 쉽게 읽고 쓸 수 있도록 도와준다. MongoDB는 스키마가 유연하여 데이터 구조가 자주 변경되는 애플리케이션에 적합하다. MongoDB는 강력한 쿼리 기능을 가지고 있으며 검색에 활용하는 인덱싱도 2가지 버전으로 지원해 효율적인 데이터 검색이 가능하도록 한다.

2.1.8. Script

사용자가 작성한 명령어들의 모음을 의미하며, 파일 시스템 관리, 데이터 처리, 네트워크 작업 등 다양한 자동화 작업에 사용한다. 반복적인 작업을 자동화하여 효율성을 높이고 시간을 절약하며 Bash, Python, JavaScript 등 다양한 프로그래밍 언어로 작성할 수 있다. 이를 통해 개발자는 복잡한 작업을 간편하게 수행할 수 있다.

2.1.9. SSH 연결

SSH는 보안 셸로, 네트워크를 통해 원격 시스템에 안전하게 접속할 수 있게 해주는 프로토콜이다. 주로 원격 서버에 안전하게 접근하여 명령어를 실행하거나 파일을 전송하며 인증 후 암호화된 통신을 한다는 특징이 있다. 이를 통해 네트워크에서 발생할 수 있는 여러 위협을 예방할 수 있다. 시스템 관리와 클라우드 환경에서 필수적으로 사용된다.

2.2. OS

2.2.1. Windows

관리자 OS로 사이트 관리, VMware를 통한 페도라 리눅스 위의 MongoDB를 관리한다.

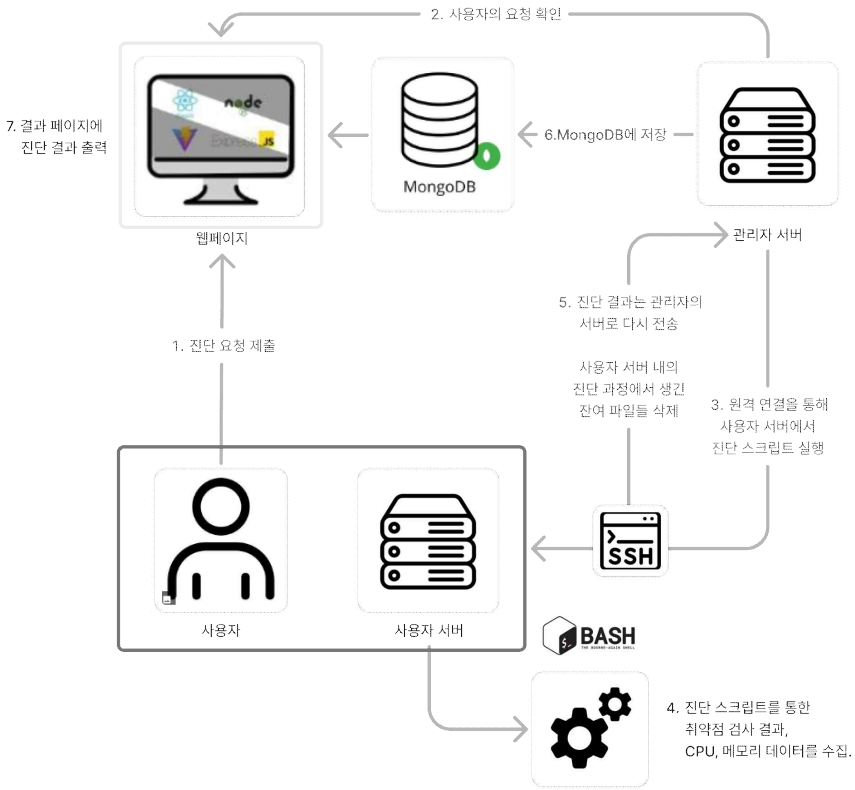
2.2.2. Fedora Linux

취약점 분석을 요청한 client OS와 MongoDB로의 결과 파일 업로드를 위한 관리자 OS에 사용되었다. 앞서 소개하였듯이 관리자 OS에 속하는 리눅스 서버의 경우 윈도우 위의 VMware를 통해 관리된다.

3. 본 론

3.1. 시스템 구성

웹서비스를 통해 고객이 회원가입, 로그인 과정을 거친 후, 취약점 진단을 요청한다. 관리자는 해당 요청 게시글을 확인하고 원격으로 고객의 서버에 접속해 취약점 진단을 진행한다. 관리자는 고객의 취약점 진단 결과를 DB에 업로드하고, 게시글에 PDF 형식으로 정리한 결과 파일을 덧붙인 뒤 고객에게 메일로 알린다. 고객은 메일로 안내된 게시글을 따라 진단 결과를 확인하고, 추가적으로 진행된 자원 모니터링 결과를 웹페이지에서 제공받는다.



[그림 3.1] 구상도

3.2. 프로그램 구성

강김홍차(KKHC)의 Vulner 프로그램은 백엔드 node.js, express 프론트 react, vite를 사용하여 웹사이트를 개발 및 디자인했다.

3.2.1. 서비스 페이지

사용자가 취약점 진단 프로그램에 대한 정보와, 그 사용법을 설명하기 위한 서비스 페이지이다. 대표적으로 사용 된 기술로는 **React**, **React Router DOM**, **Axios**, **js-cookie**가 있다.

[그림3.2.1-1] 메인 홈

메인 홈, 프로그램 정보, 프로그램 가이드, 팀 소개 탭으로 구성되어 있다. 인트로 페이지에 배치된 바로가기 버튼을 통해 프로그램 페이지로 이동할 수 있다.

[메인 홈 주 기능]
1. 상단 헤더 바
2. 프로그램 바로가기 버튼
3. 서비스 페이지 요소 요약(Information, Guide, Team 페이지)

[상단 헤더 바]
1. Home : 메인 홈으로 이동
2. Information : 프로그램에 관한 내용(제공하는 기능, 사용된 기술, 기능 구조

도) 소개

3. Guide : 사용자가 원활히 프로그램을 사용할 수 있도록 로그인 전, 게시글 작성 양식, 로그인 후 할 수 있는 기능들과 사용 순서를 설명

4. Team : 해당 작품을 개발한 개발자들의 정보를 보여주는 페이지로 이동

```
<Router>
  <Routes>
    <Route path="/" element={<Navigate to="/home" />} />
    <Route path="/team" element={<Team />} />
    <Route path="/ExList" element={<ExList />} />
    <Route path="/Information" element={<Information />} />
```

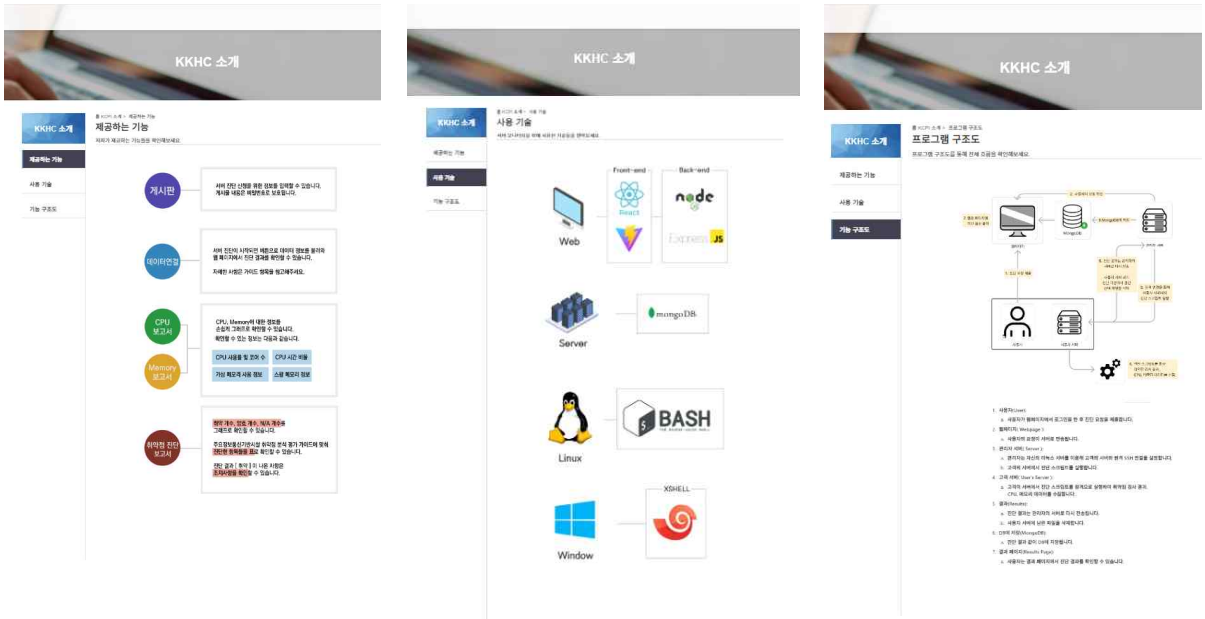
[그림3.2.1-2] 페이지 간 이동 구현을 위한 react-router-dom 코드 일부

[그림3.2.1-3] 메인 홈 - 가이드 슬라이더

React Slick & Slick Carousel , Swiper 를 사용하여 캐러셀(slider)과 모던한 터치 슬라이더를 구현하였다. 더 알아보기 버튼을 누르면 Information 페이지로 리다이렉션 된다.

좌우로 슬라이드 되는 기능이 있으며 하단에 위치한 바(Bar)는 슬라이드의 진행 상태를 확인할 수 있는 기능을 한다.(진행 상태는 색으로 확인할 수 있다.)

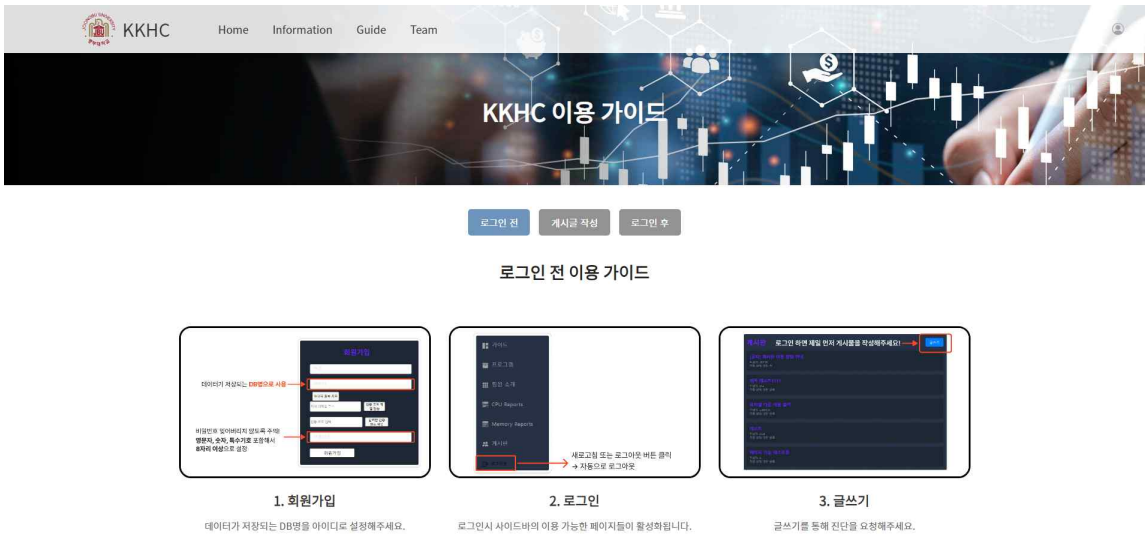
[그림3.2.1-4]메인 홈 - 프로그램 특징 소개



[그림3.2.1-5] Information 페이지

해당 페이지는 제공하는 기능과 그에 대한 설명, 사용되는 기술, 기능 구조도에

관해 안내하며 제공 서비스는 게시판, 데이터 연결 그리고 CPU 보고서, Memory 보고서, 취약점 진단 보고서가 있으며 사용되는 기술은 Web에서 Front-end의 React,Vite, Back-end의 node.js,ExpressJS, 서버에서 [MongoDB], Linux[Bash], Window[XSHELL]이다.



[그림3.2.1-6] Guide 페이지

프로그램 이용 가이드 페이지이다. 로그인 전, 게시글 작성 양식, 로그인 후로 설명을 나누어 사용자가 이용 할 수 있는 기능들과 사용 순서를 설명하는 역할을 한다.

Guide페이지에 처음 접근했을 시 보이는 '로그인 전' 페이지는 로그인 전 이용자가 볼 수 있는 프로그램 페이지는 회원가입, 로그인 페이지, 아이디 찾기 페이지, 비밀번호 변경 페이지이며 로그인을 완료했을 시 글쓰기 기능을 이용할 수 있음을 안내한다.

[그림3.2.1-7] Guide - 게시글 작성

취약점 진단을 위해 작성해야 하는 게시글에 대한 안내를 다루고 있다.

[그림3.2.1-8] Guide - 로그인 후

게시글 작성 이후 취약점 진단이 진행되었을 시 이용할 수 있는 기능들을 소개한다.

각각의 [프로그램], .[게시판], [게시물], [취약점 진단 결과 확인]에 관해 가이드다.

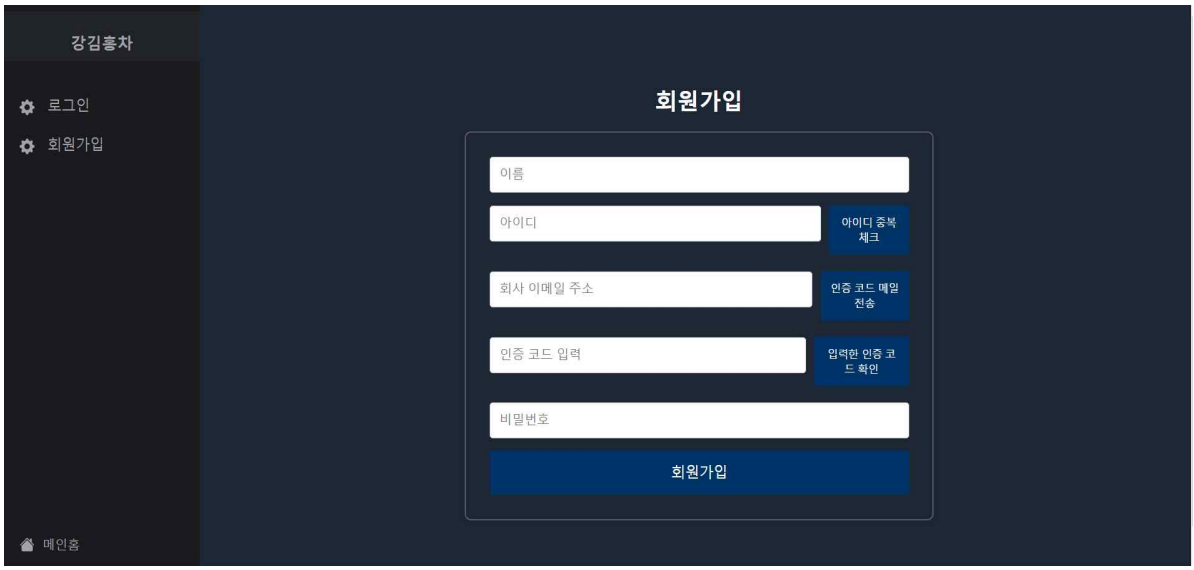
[그림3.2.1-9] Team 페이지

3.2.2. 프로그램 페이지

회원가입 및 게시판에 게시물 작성을 통해 서버 진단 요청을 할 수 있으며, 서버 취약점의 진단 결과 분석과 대응 방안, CPU 메모리 사용률 등의 결과를 진단 결과보기 버튼 및 reports 사이드바 메뉴를 통해 표와 그래프로 제공한다. 취약점 진단 결과는 사용자가 작성한 게시물에 첨부 파일란에서 PDF, TXT 파일로 미리 보기, 다운이 가능하다.

또한 진단 요청 및 결과보기 기능 이외의 추가적인 기능에는 아이디 찾기, 비밀번호 재설정, 게시물 수정, 삭제, DB 설정, 이메일 전송, 자동 로그아웃, 게시물 암호화가 있으며, 쿠키로 로그인 여부를 확인한다. 또, 사용자 ID를 쿠키에 저장해 관리자 여부 판단을 진행하고 이에 따라 유동적인 페이지와 기능을 제공한다.

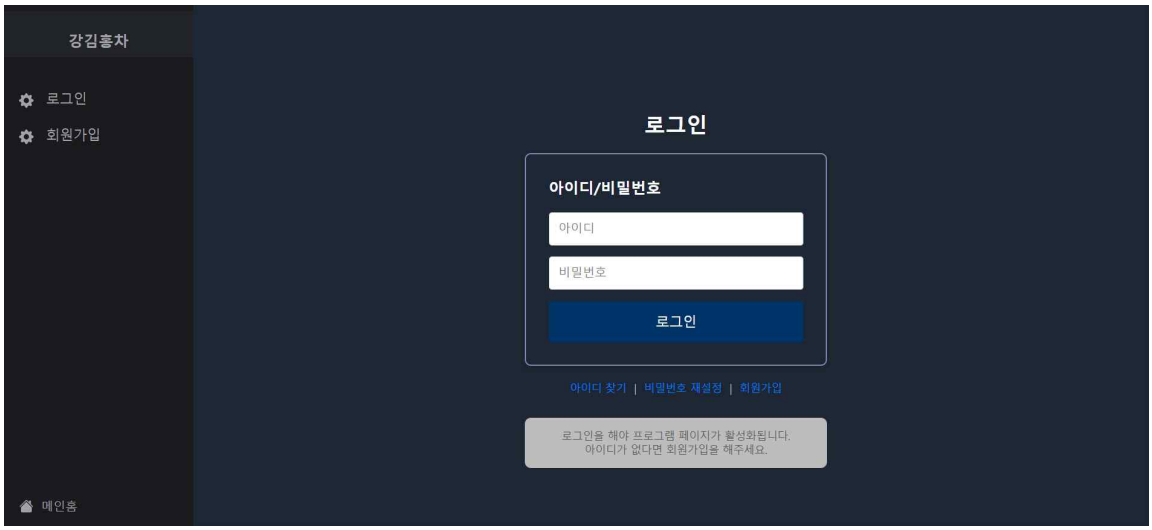
- **유저 기능**



[그림3.2.2-1] 회원가입 페이지

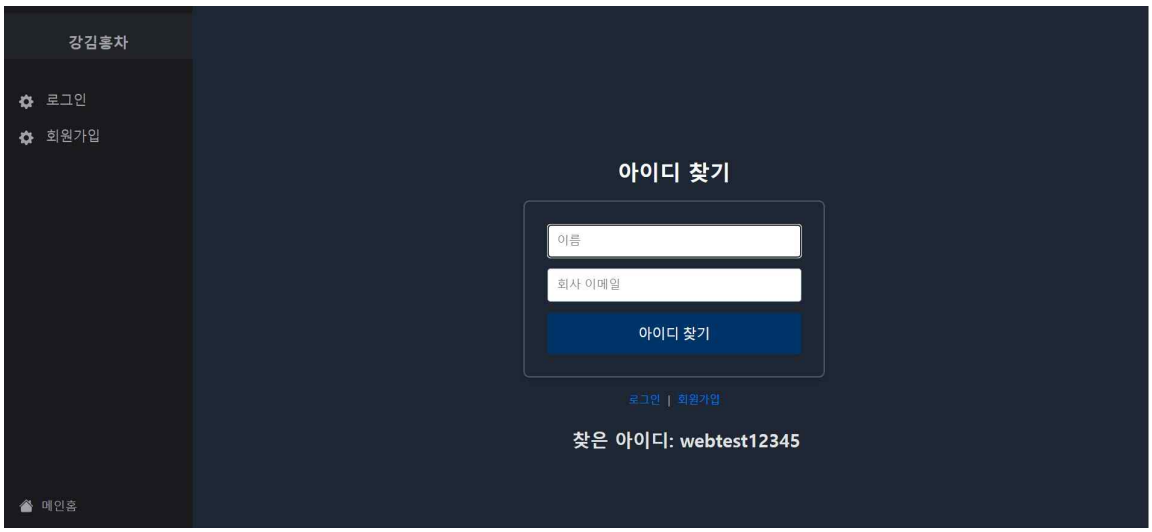
[그림3.2.2-2] DB에 저장된 유저 정보

회원가입을 통해 서버 진단 요청 사이트를 이용할 수 있다. 아이디 중복 체크, 이메일 전송을 통한 회사 이메일 검증, 특수문자, 숫자, 영문자 한 자씩 포함해서 총 8자리 이상이어야지만 회원가입을 할 수 있다. 위 절차를 통해 회원가입이 완료되면 DB에 이름, 아이디, 회사 이메일 주소, 암호화된 비밀번호가 저장된다. 또한 가입한 아이디로 개인 DB가 생성되며 취약점 진단 결과, 모니터링 데이터를 받기 위한 컬렉션이 자동으로 생성된다.



[그림3.2.2-3] 로그인 페이지

아이디와 비밀번호를 입력하면 로그인이 가능하다. 사용자에게 아이디가 없으면 해당 사이트를 이용할 수 없다는 안내와 함께 아이디 찾기, 비밀번호 재설정, 회원가입 기능을 제공하고 있다.



[그림3.2.2-4] 아이디 찾기 페이지

아이디 찾기 페이지에서 가입할 때 기입한 이름, 회사 이메일 주소를 작성하면 아이디가 페이지 하단에 표시된다.

[그림3.2.2-5] 로그인 후 메인 페이지

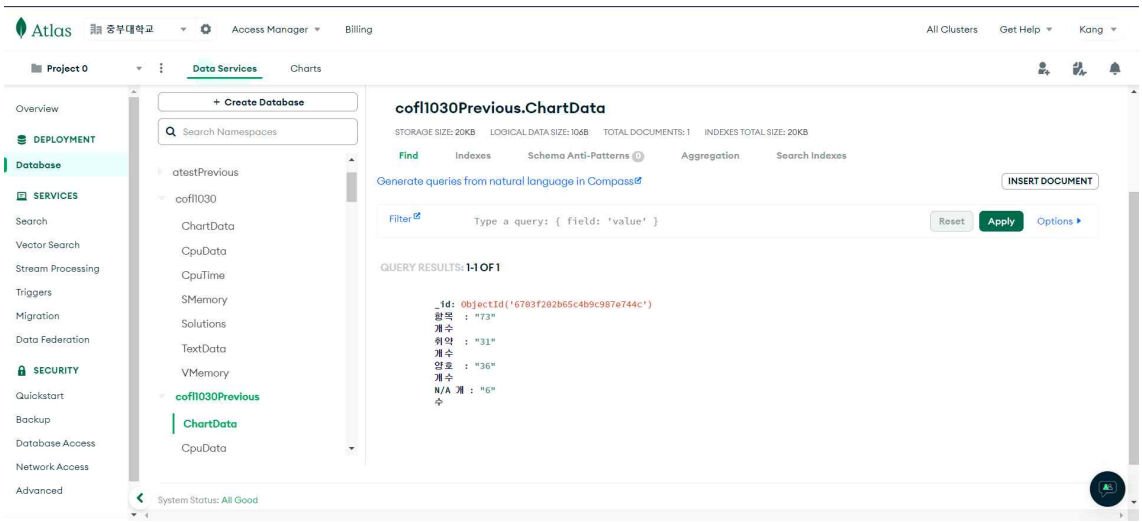
사용자가 로그인을 하면 쿠키에 저장된 userId 값을 확인하여 관리자 여부를 확인한다. 사용자로 판명 나면 메뉴에 프로그램 (DB 설정), Reports (모니터링 메모리 진단 결과 그래프로 확인), 게시판 (서버 진단 요청 게시물 생성 & 진단 결과를 표, 그래프, PDF 및 TXT 파일로 다운 및 확인) 페이지가 활성화된다. 로그아웃은 버튼을 클릭하거나 사용자 접속 시간이 5분이 지나면 자동으로 활성화된다.

[그림3.2.2-6] 서버 진단 요청 게시물 작성 양식

사용자는 양식에 따라 제목, 진단하려는 서버 IP, 관리자 ID, 관리자 PW, 루트 비밀번호, 진단 완료 알림을 받을 이메일, 게시물 비밀번호를 입력해야 한다. 빈칸이 있으면 required를 통해 입력해달라는 알림이 표시되고 다 채울 때까지 새로운 게시물 생성이 불가능하다.

[그림3.2.2-7] 생성된 게시물 목록

게시물이 생성되면 작성자는 자동으로 로그인한 사용자의 ID로 표시된다. 그래서 로그인 할 때 쿠키에 저장한 ID 값과 DB에서 해당 게시물을 작성한 작성자의 ID 값을 비교해서 같은 경우에만 본인이 작성한 게시물로 판단한다. 본인이 작성한 게시물이면 제목, 진단 상태, 작성자를 게시물 목록에서 확인 가능하다. 만약 타인이 작성한 게시물을 선택하면 경우 제목, 진단 상태만 게시물 목록에서 확인 가능하다.



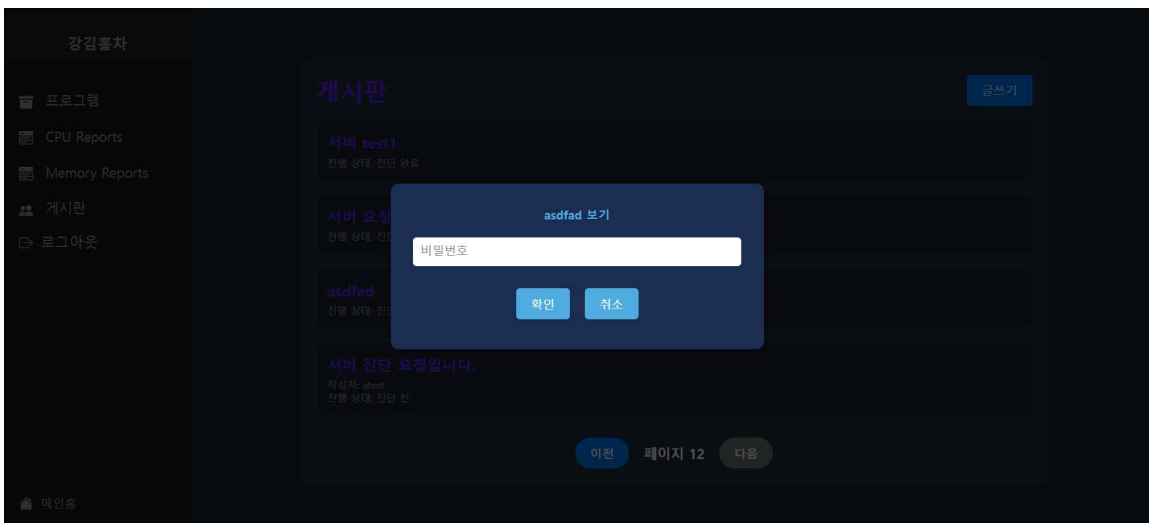
[그림3.2.2-8] 개인 DB와 백업 DB 내용

사용자가 새로운 게시물을 작성할 때마다 이전 데이터가 저장되고 기존 DB는 초기화된다.

백업하는 DB는 유저ID+Previous 이름으로 생성되고 기존 데이터 내용이 전부 저장된다. 개인 DB와 백업 DB의 구조는 취약점 진단 결과를 저장하는 3개의 컬렉션과 모니터링 메모리 데이터를 저장하는 4개의 컬렉션으로 구성되어 있다.

[그림3.2.2-9] 작성한 게시물 내용 확인

본인이 작성한 게시물의 경우 게시물 비밀번호 없이 내용을 확인할 수 있으며 수정이 가능하다. 단, 삭제는 게시물을 생성할 때 입력했던 비밀번호를 입력해야 한다.



[그림3.2.2-10] 타인이 작성한 게시물 진입

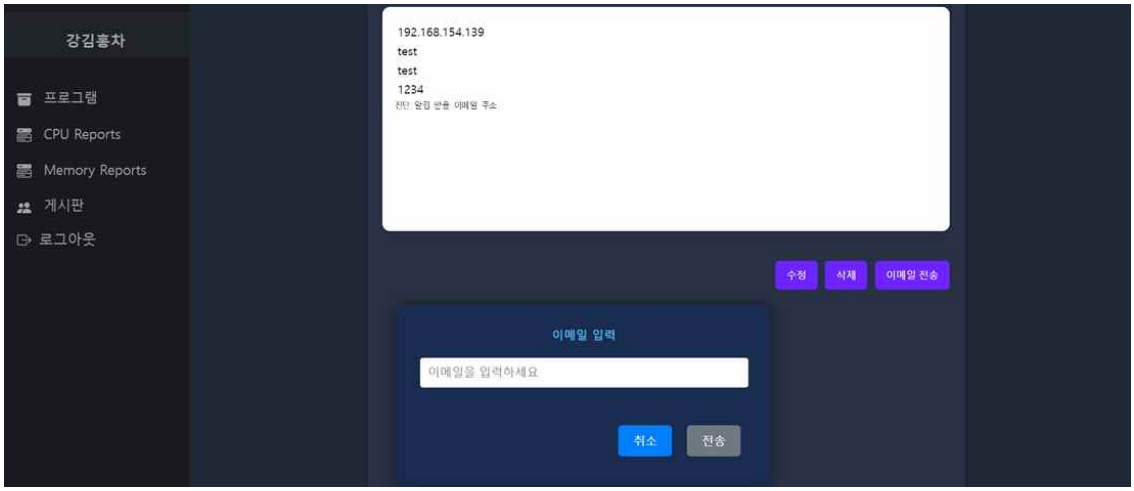
본인이 생성한 게시물이 아닌 경우 게시물 내용을 확인하려면 비밀번호를 입력해야 한다.

- 관리자 기능

로그인할 때 쿠키에 저장한 아이디 값을 DB에 유저 정보들이 기록된 곳에서 검색한다. 유저 정보에 admin이 있을 경우 해당 유저는 관리자로 판단하고 첨부 파일 업로드, 진단 완료 메일 전송, 모든 게시물을 비밀번호 없이 볼 수 있는 기능이 추가된다.

[그림3.2.2-11] 유저가 진단 요청한 게시물에 결과 파일 업로드

관리자 계정으로 로그인하면 수정 기능에 파일 업로드 기능이 추가된다. 유저가 진단 요청을 한 정보를 토대로 취약점 진단을 돌려서 나온 결과 파일들(PDF, TXT)을 Multer 미들웨어를 통해 파일을 업로드 한다. 그리고 MongoDB의 GridFS를 이용해 파일을 DB에 저장해서 사용자가 다운 및 미리보기가 가능하도록 URL을 생성한다.



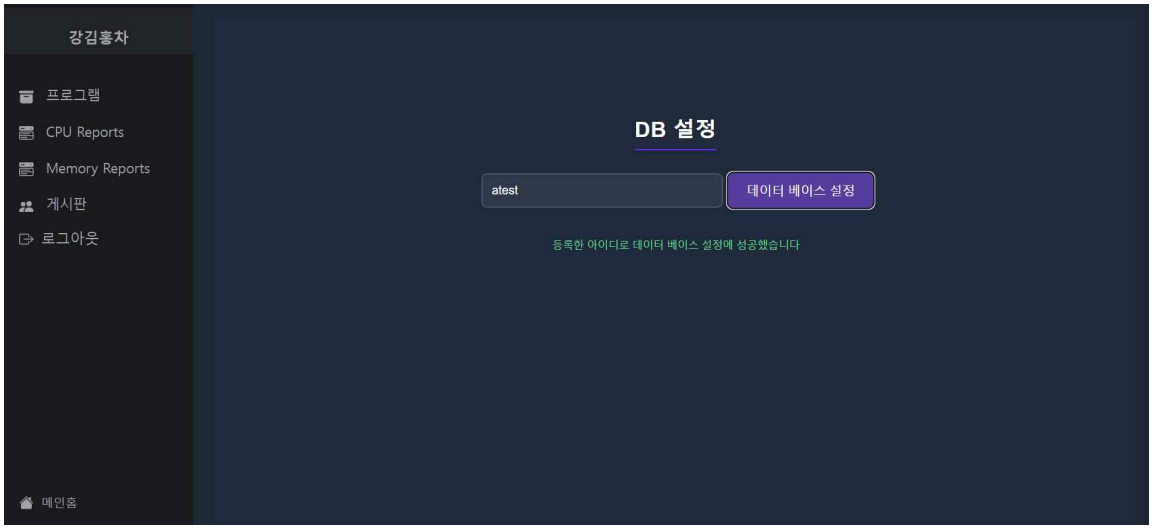
[그림3.2.2-12] 이메일 전송 기능을 통해 진단완료 알림 유저에게 메일로 전송

Nodemailer를 통해 유저가 사전에 입력해놓은 이메일로 진단 완료 알림을 전송한다.

- 유저 기능

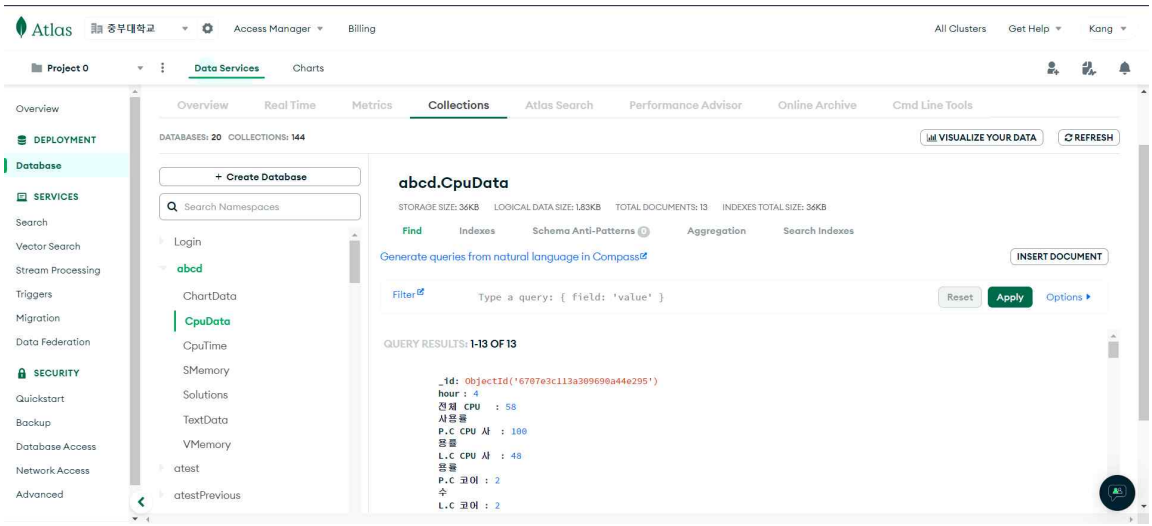
[그림3.2.2-13] 유저가 받은 진단완료 알림 내용

유저가 진단완료 알림을 메일로 받고 사이트에 다시 로그인을 한다. 메일 내용에는 진단 완료 알림, 해당하는 게시물 제목, 사이트 링크가 있다.



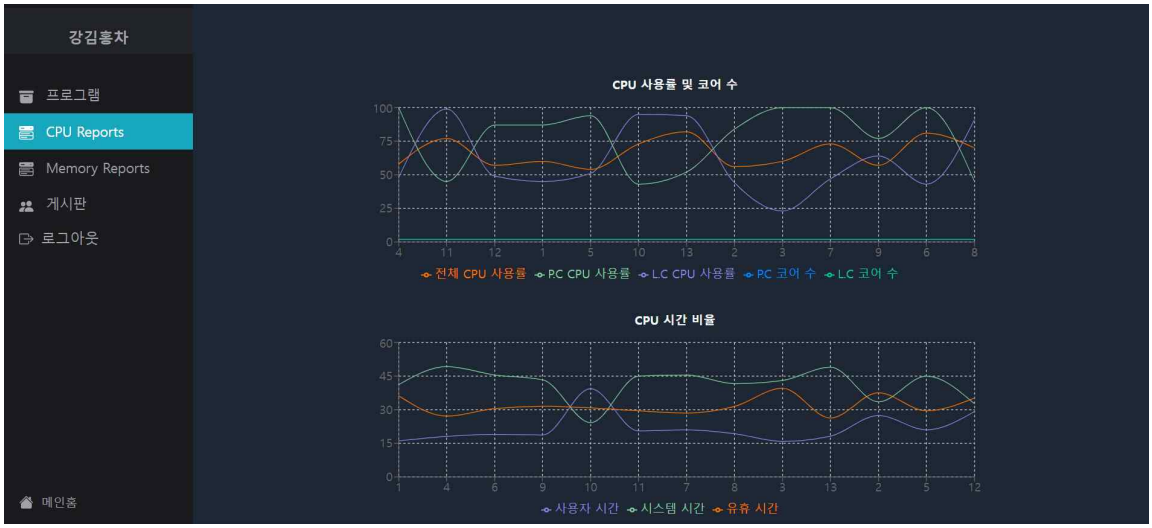
[그림3.2.2-14] 가입한 ID로 DB 설정

취약점 진단 및 모니터링을 통해 수집한 메모리 데이터들은 개인 DB에 저장된다. 이때 DB명은 가입한 아이디이므로 DB 설정 역시 가입한 ID를 기입하고 설정 버튼을 클릭한다.

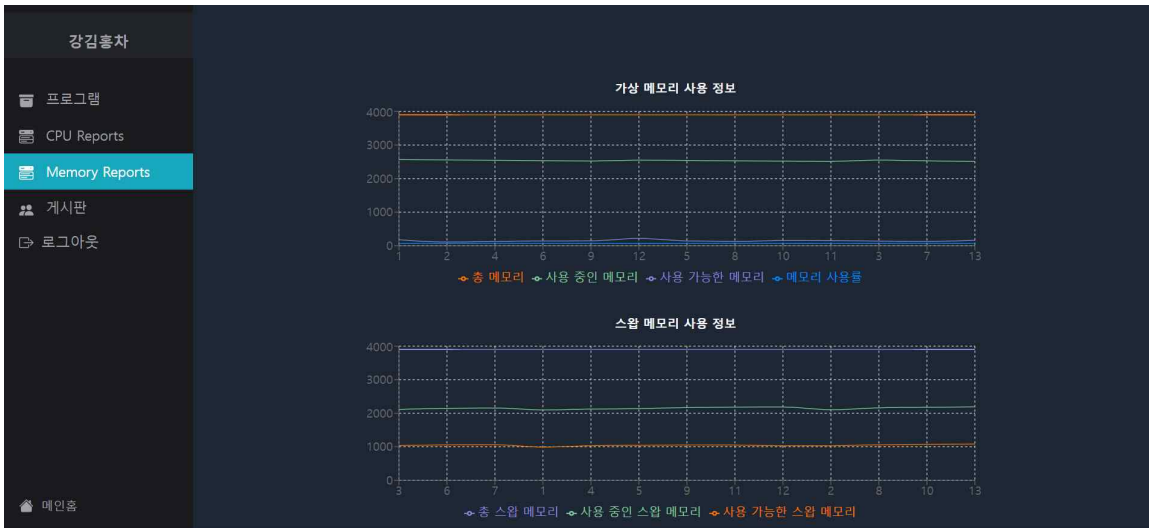


[그림3.2.2-15] DB에 저장된 취약점 진단 결과 및 메모리 사용 정보 등의 모니터링 결과

고객 서버를 진단한 취약점 진단 결과와 모니터링을 통해 측정한 메모리 사용 정보 등의 값은 개인 DB로 올라간다.

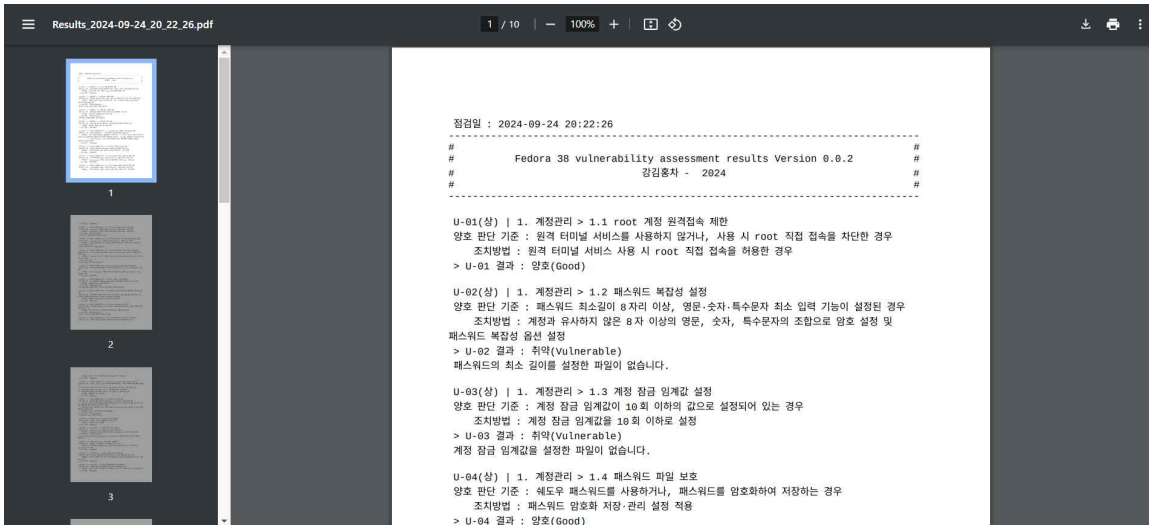


[그림3.2.2-16] CPU 사용률 등의 결과를 그래프로 보여주는 페이지

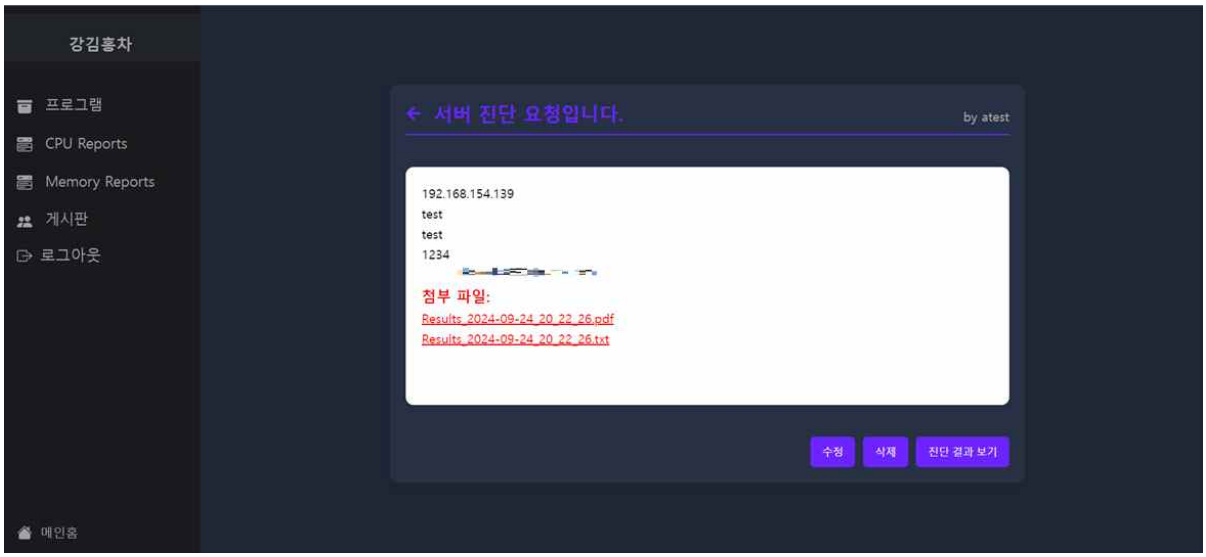


[그림3.2.2-17] 메모리 사용 정보를 보여주는 페이지

프로그램에서 개인 DB (가입한 ID)로 설정하면 DB에 json 형태로 저장된 값들을 그래프로 볼 수 있다. 위 그래프는 모니터링을 돌렸을 때 나오는 결과값들만 나타내었다.



[그림3.2.2-18] 취약점 진단 결과 파일 다운



[그림3.2.2-19] 취약점 진단 결과를 보러 가는 페이지

유저 본인이 작성한 게시물에 다시 들어가면 관리자가 올려준 파일을 확인할 수 있다. 해당 파일을 클릭하면 다운로드 및 미리보기가 가능한 화면으로 이동한다. 또한, 관리자가 파일을 첨부해준 경우 진행 상태가 진단 전 → 진단 완료로 변경되며 진단 결과 보기 버튼이 활성화 된다.

[그림3.2.2-20] 취약점 진단 결과 보여주는 페이지

[그림3.2.2-21] 분류 키워드로 검색

진단 결과 보기 버튼을 통해 들어가게 되면 취약점 진단 결과값을 표와 그래프로 보여준다.

상단의 검색란에서는 id(진단 번호), 분류(계정관리 등), 결과(취약, 양호, N/A)의 값으로 검색하면 해당값만 표시된다. 그래프에서는 전체 항목 개수, 취약 및 양호 개수, 담당자와 상의해야하는 N/A 개수를 표시했다. 양호, N/A는 파란색, 취약은 붉은색으로 표현하여 직관적으로 구분할 수 있도록 하였고, 결과값이 취약인 항목은 클릭하면 해당 항목에 대한 조치 방법을 볼 수 있다.

[그림3.2.2-22] 조치 방법 페이지

표에서 결과값이 취약인 항목을 클릭하면 해당 항목에 대한 조치 방법이 나온다. 진단 번호와 간단하게 조치 방법을 설명해주는 페이지다.

3.2.3. 취약점 진단

취약점 진단 과정은 관리자 주도하에 고객 서버의 보안 취약점을 식별하고 자원 모니터링 정보를 웹사이트에 전송하는 과정이며, 주요정보통신기반 진단 스크립트와 모니터링 데이터 수집 스크립트를 통해 결과 파일을 생성한다. 배시 셸 스크립트로 개발하여 빠른 실행 및 파일 간소화를 목적으로 하였다.

진단에 사용되는 파일은 원격 제어 파일 1개, 진단 후 결과를 DB로 전송하는 파일 1개, 취약점 분석 파일 1개, 모니터링 데이터 수집 파일 5개로 총 8개이며 직접 실행하는 스크립트는 origin_ssh.sh 파일 하나이다.

```
#!/bin/bash

# 사용자로부터 SSH 연결 정보 입력 받기
read -p "연결할 사용자 이름을 입력하세요: " USERNAME
read -p "연결할 호스트의 IP 주소 또는 도메인 이름을 입력하세요: " HOSTNAME

# sudo 비밀번호 입력 받기
echo -n "root 비밀번호를 입력하세요: "
read -s SUDO_PASSWORD
echo # 줄 바꿈

# 전송할 파일 경로
FILE_PATH="testingFile.zip"

# 상대 리눅스에 파일 전송
echo "취약점 검사 및 운영체제 모니터링 파일 전송 중..."
scp "$FILE_PATH" "$USERNAME@$HOSTNAME:~/"

if [ $? -ne 0 ]; then
```

[그림3.2.3-1] origin_ssh.sh 1

origin_ssh.sh 파일을 실행하면 파일의 실행자(관리자)로부터 SSH 연결을 위한 정보(고객 서버의 계정 이름, 고객 서버 호스트 IP). 고객 서버의 권한을 얻기 위한 고객 서버 Root 계정 Password까지 입력받도록 한다.

그 후 Client Server OS로 복사할 검사 파일의 경로를 지정하고, 입력받은 서버에 보안 복사(Secure copy)를 진행한다.

```
# SSH 연결 및 명령어 실행
echo "상대와 SSH 연결 중..."
ssh "$USERNAME@$HOSTNAME" << EOF
    echo "--------------------------------"
    echo "검사 파일 압축 해제 중..."
    unzip testingFile.zip -d testingFile  # testingFile로 압축 해제
    cd testingFile  # 압축 해제된 폴더로 이동

    # 비밀번호를 통해 sudo 명령어 실행
    echo "$SUDO_PASSWORD" | sudo -S bash monitor.sh

    echo "--------------------------------"

    # 검사가 완료된 sh 파일 및 여분의 txt 파일 삭제
    rm -rf *.sh && rm -rf *.txt

    # 현재 디렉토리 이름 저장
    current_dir=\$(basename "\$PWD")

    # 압축할 파일 이름
    zip_file="testData.zip"

    # 현재 디렉토리의 모든 파일을 압축
    echo "압축 중: \$zip_file"
    zip -r "\$zip_file" ./*

    # 내 리눅스에 파일 전송 (하드코딩된 IP 및 경로)
    echo "검사 결과 전송 중..."
    sshpass -p "1234" scp "\$zip_file" user1@192.168.154.130:/home/user1/
```

[그림3.2.3-2] origin_ssh.sh 2

```
    # 압축할 파일 이름
    zip_file="testData.zip"

    # 현재 디렉토리의 모든 파일을 압축
    echo "압축 중: \$zip_file"
    zip -r "\$zip_file" ./*

    # 내 리눅스에 파일 전송 (하드코딩된 IP 및 경로)
    echo "검사 결과 전송 중..."
    sshpass -p "1234" scp "\$zip_file" user1@192.168.240.134:/home/user1/

    # 현재 디렉토리 삭제
    echo "검사에 사용된 정크 데이터 삭제 중..."
    cd .. && rm -rf "\$current_dir"*

    echo "SSH 원격 작업 완료"
    echo "--------------------------------"
```

[그림3.2.3-3] origin_ssh.sh 3

입력 받은 정보를 토대로 ssh 연결 후 앞서 전송한 검사 파일을 testingFile이라는 이름으로 압축 해제한다. 압축을 해제한 testingFile로 이동 후 미리 입력한 고객 서버 root 비밀번호를 이용해 sudo 권한을 얻어내면, 취약점 검사 파일과 자원 모니터링 데이터 수집 파일을 관리하는 monitor.sh를 실행한다. 검사 완료 후 결

과 파일들은 현재 디렉토리에 저장되도록 하였다. 또한 진행 중 검사가 완료된 sh 파일 및 여분의 txt 파일을 삭제한다. 결과 파일들로만 이루어진 testingFile은 testData.zip으로 압축되어 보안 복사를 통해 관리자의 서버로 돌아온다. 고객 서버에 남은 testingFile.zip과 testingFile 폴더는 한 번에 삭제된다.

```
    echo "SSH 원격 작업 완료"
    echo "--------------------------------"
EOF

# 검사 결과 압축 해제
unzip testData.zip -d /home/user1/admin/testData
echo "--------------------------------"

# Mongo DB에 데이터 파일 업데이트
read -p "검사 결과를 업로드할 DataBase ID를 입력하세요 : " DB

echo "검사 결과를 DB에 업로드 중..."
bash move_to_DB.sh "$DB"

# Xshell을 이용해 관리자 Windows에 pdf 형태 검사 결과 전송
echo "PDF로 검사 결과를 확인하고 게시글을 업데이트 하세요."
sz testData.zip
rm -f "testData.zip"  # testData.zip 파일 삭제
~
```

[그림3.2.3-4] origin_ssh.sh 4

원격 작업을 완료한 이후, 관리자의 리눅스 서버에서 testData.zip 파일을 압축 해제한다. 이 폴더의 내용물은 자동으로 Mongo DB에 기록되어 있는 고객의 DataBase ID로 업로드 되고, pdf 파일은 관리자가 웹 페이지 게시물에 쉽게 업데이트 할 수 있도록 관리자의 windows로 전달한다. 가장 마지막으로 관리자의 리눅스에서 고객의 취약점 검사 결과 압축 파일을 삭제한다.

origin_ssh.sh 코드를 실행시키고 출력 결과를 살펴보면 [그림3.2.3-2] origin_ssh.sh 2에서 monitor.sh 파일을 실행했을 때 다른 스크립트 파일도 실행되는 것을 알 수 있다. monitor.sh 파일의 코드를 자세히 설명하겠다.

```
#!/bin/bash

# 패키지 목록
packages1=("procps-ng" "sysstat")
packages2=("jq" "sysstat")

# 패키지 업데이트
echo "검사에 필요한 패키지 목록 업데이트 중..."
sudo dnf check-update

# 첫 번째 패키지 설치
echo "설치 중: ${packages1[*]}"
sudo dnf install -y "${packages1[@]}"

# 두 번째 패키지 설치
echo "설치 중: ${packages2[*]}"
sudo dnf install -y "${packages2[@]}"

echo "모든 패키지가 설치되었습니다."

echo "--------------------------------"
```

[그림3.2.3-5] monitor.sh 1

[그림 3.2.3-5]와 같이 스크립트를 살펴보면 bash 선언 후 취약점 점검 및 자원 점검에 필요한 패키지들을 배열로 정의하여 패키지 업데이트 및 설치를 진행한다.

```
# 권한을 -x로 상승시키는 파일 리스트
files=("cpu_usage.sh" "cpu_time_ratio.sh" "vm_usage.sh" "swap_usage.sh" "main_test.sh")

for file in "${files[@]}"; do
    chmod +x "$file"
done

#------------------------------

#main_test.sh 실행
./main_test.sh &
echo -e "[취약점 검사 \033[1;32m시작\033[0m]"       # 키워드 색상 넣어 가시성 높임
echo "잠시만 기다려주세요."

while true
do
        # main_test.sh 실행 확인
        check=$(ps -ef | grep "main_test.sh" | grep -v "grep")

                echo "검사가 진행되는 동안 자원 모니터링 중"
```

[그림3.2.3-6] monitor.sh 2

취약점 검사에 필요한 스크립트 파일들의 실행 권한을 chmod +x를 이용해 추가하고 취약점 검사 파일인 main_test.sh를 백그라운드에서 실행하여 취약점 진단 중 모니터링 정보를 수집할 수 있도록 한다. main_test.sh가 실행 중인지 확인하고 실행 중이라면 자원 모니터링을 반복하도록 작성하였다.

```
# main_test.sh 모니터링
if [ "$check" != "" ]; then

        # 4개의 쉘 스크립트 동시에 실행
        ./cpu_usage.sh &
        pid1=$!

        ./cpu_time_ratio.sh &
        pid2=$!

        ./vm_usage.sh &
        pid3=$!

        ./swap_usage.sh &
        pid4=$!

        # 모든 백그라운드 작업이 완료될 때까지 기다리기
        wait $pid1
        wait $pid2
        wait $pid3
        wait $pid4
```

[그림3.2.3-7] monitor.sh 3

수집 정보는 CPU 사용량, CPU 시간 비율, 가상 메모리 사용량, Swap 메모리 사용량이며 각 스크립트를 백그라운드에서 실행하여 프로세스 ID를 변수에 저장한다.

```
        ./vm_usage.sh &
        pid3=$!

        ./swap_usage.sh &
        pid4=$!

        # 모든 백그라운드 작업이 완료될 때까지 기다리기
        wait $pid1
        wait $pid2
        wait $pid3
        wait $pid4
        echo "--------------------------------"

        # main_test.sh가 여전히 실행 중인 경우 잠시 대기
                        sleep 3
else
        echo -e "[취약점 검사 \033[1;31m종료\033[0m]"    # 키워드 색상 넣어 가시성 높임
        break
fi
```

[그림3.2.3-8] monitor.sh 4

모니터링 무한 루프는 4개의 모니터링 스크립트가 모두 완료될 때까지 wait 명령으로 대기하고, 모두 완료되고도 main_test.sh의 실행이 끝나지 않았다면 다시 모니터링 파일을 실행한다. main_test.sh가 종료되면, 취약점 검사가 완료되었음을 알리고 루프를 종료한다.

취약점 분석 main_test.sh는 주요정보통신기반 가이드에 따른다.

```
resultfile="Results_$(date '+%F_%H:%M:%S').txt"
postdb="TextData.json"
statis="ChartData.json"
action="Solutions.json"


U_01() {
        echo ""  >> $resultfile 2>&1
        echo "[ " >> $postdb 2>&1
        echo " { " >> $postdb 2>&1
        echo "[ " >> $action 2>&1
        echo " { " >> $action 2>&1
        echo " U-01(상) | 1. 계정관리 > 1.1 root 계정 원격접속 제한 "  >> $resultfile 2>&1
        echo "  \"분류\" : \"계정관리\", " >> $postdb 2>&1
        echo "  \"id\" : \"01\", " >> $postdb 2>&1
        echo "  \"id\" : \"01\", " >> $action 2>&1
        echo " 양호 판단 기준 : 원격 터미널 서비스를 사용하지 않거나, 사용 시 root 직접 접속
        echo "     조치방법 : 원격 터미널 서비스 사용 시 root 직접 접속을 허용한 경우" >> $
        echo "  \"조치방법\" : \"원격 터미널 서비스 사용 시 root 직접 접속을 허용한 경우\" (
        if [ -f /etc/services ]; then
```

[그림3.2.3-9] main_test.sh 1

점검 결과를 저장할 파일 resultfile과 데이터베이스 파일인 postdb, statis, action을 변수로 설정한다. 결과 파일명은 실행 시점의 날짜와 시간을 기반으로 동적으로 생성되며 txt 형태로 저장, 데이터베이스 파일은 json 형태로 저장된다. 그 후 주요정보통신기반 가이드 리눅스 항목에 따라 U-01부터 U-72까지 해당 항목을 검사하고, 그 결과와 조치 방법을 기록한다.

```
        echo "] " >> $postdb 2>&1
}

echo ""  > $resultfile 2>&1
echo " 점검일 : `date +'%F %H:%M:%S'`"  >> $resultfile 2>&1
echo "------------------------------------------------------------------------------" >> $resultfile 2>&1
echo "#                                                                            #" >> $resultfile 2>&1
echo "#          Fedora 38 vulnerability assessment results Version 0.0.2          #" >> $resultfile 2>&1
echo "#                                강김홍차 -  2024                              #" >> $resultfile 2>&1
echo "#                                                                            #" >> $resultfile 2>&1
echo "------------------------------------------------------------------------------" >> $resultfile 2>&1

U_01
U_02
U_03
U_04
U_05
U_06
```

[그림3.2.3-10]main_test.sh 2

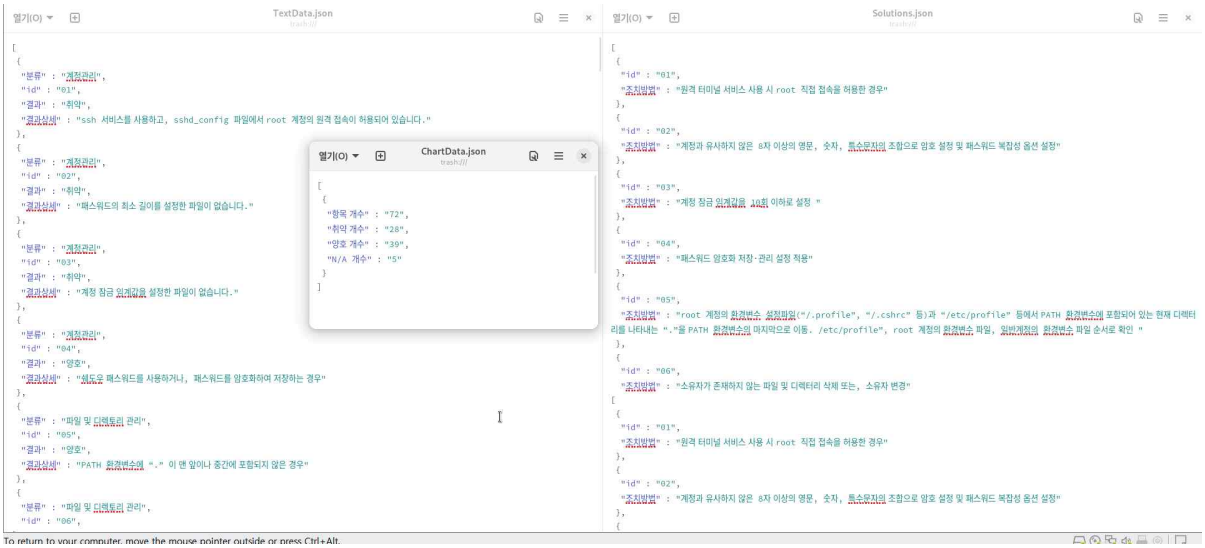
72번 항목까지 함수 지정 후 결과 파일의 헤더 부분을 설정하고 앞서 지정한 함수를 실행하여 각각의 점검 항목이 해당하는 항목의 취약점을 검사하고 그 결

과를 기록한다.

```
U_68
U_69
U_70
U_71
U_72

echo "[" >> $statis 2>&1
echo " {" >> $statis 2>&1
echo "  \"항목 개수\" : \"$(cat $resultfile | grep '결과 : ' | wc -l)\", " >> $statis 2>&1
echo "  \"취약 개수\" : \"$(cat $resultfile | grep '결과 : 취약' | wc -l)\", " >> $statis 2>&1
echo "  \"양호 개수\" : \"$(cat $resultfile | grep '결과 : 양호' | wc -l)\", " >> $statis 2>&1
echo "  \"N/A 개수\" : \"$(cat $resultfile | grep '결과 : N/A' | wc -l)\" " >> $statis 2>&1
echo " }" >> $statis 2>&1
echo "]" >> $statis 2>&1

echo ""  >> $resultfile 2>&1
echo "================================ 진단 결과 요약 ================================" >> $resultfile 2>&1
echo ""  >> $resultfile 2>&1
echo "                              항목 개수 = `cat $resultfile | grep '결과 : ' | wc -l`" >> $resultfile
echo "                              취약 개수 = `cat $resultfile | grep '결과 : 취약' | wc -l`" >> $result
echo "                              양호 개수 = `cat $resultfile | grep '결과 : 양호' | wc -l`" >> $result
echo "                              N/A 개수 = `cat $resultfile | grep '결과 : N/A' | wc -l`" >> $resultfi
echo ""  >> $resultfile 2>&1
echo "================================================================================" >> $resultfile 2>&1
echo ""  >> $resultfile 2>&1
```

[그림3.2.3-11] main_test.sh 3

resultfile에서 각 점검 항목의 결과(취약, 양호, N/A)를 분석하여 JSON 형식으로 저장한다. 항목 개수는 모든 점검 항목의 개수, 취약점으로 나타난 항목의 취약 개수, 양호한 항목의 양호 개수, 해당 사항이 없는 항목의 N/A 개수를 저장하고 점검 결과를 요약하여 resultfile에 추가한다.



[그림3.2.3-12] main_test 결과

main_test.sh 파일 실행 결과 주통기반 취약점 분석 결과(TextData.json), 진단결과 통계(ChartData.json), 취약 조치 방법(Solutions.json), 그리고 진단 결과 텍스트 파일(Results_20xx-xx-xx_oo:oo:oo.txt)과 PDF 파일(Results_20xx-xx-xx_oo:oo:oo.pdf)까지 총 5개가 생성된다.

모니터링

monitor.sh에서 CPU 사용량, CPU 시간 비율, 가상 메모리 사용량, Swap 메모리 사용량을 수집하기 위해 실행하는 스크립트는 cpu.usage.sh, cpu_time_ratio.sh, vm_usage.sh, swap_usage.sh이다

cpu.usage.sh

```
# 각 CPU 코어의 사용률을 저장할 배열
declare -a cpu_usage=()

# 전체 CPU 사용률 계산
total_cpu_usage=$(top -bn1 | grep "Cpu(s)" | awk '{printf("%d\n", 100 - $8)}')

# 각 CPU 코어의 사용률을 배열에 저장
cpu_info=$(mpstat -P ALL 1 1 | awk '$12 ~ /[0-9.]+/ {print $NF}')
cpu_usage=()  # 배열 초기화
while read -r line; do
    cpu_usage+=("$line")
done <<< "$cpu_info"

# P.C와 L.C 사용률 및 코어 수의 기본값 설정
pc_usage=0
lc_usage=0
pc_cores=$(nproc)
lc_cores=$(nproc)

# 사용률이 존재하는 경우에만 값 설정
if [ ${#cpu_usage[@]} -gt 1 ]; then
    # cpu_usage[1]의 %idle 값을 가져와서 CPU 사용률 계산
    idle_value1=$(echo "${cpu_usage[1]}" | awk '{print int($1)}')
    # 100에서 정수값을 뺀다
    pc_usage=$(( 100 - idle_value1 ))
```

[그림 3.2.3-13] cpu.usage.sh 1

```
# JSON 파일 경로
json_file="CpuData.json"

# 기존 JSON 파일에서 데이터를 읽어오기
if [ -f "$json_file" ]; then
    json_content=$(cat "$json_file")
else
    json_content="[]"
fi

# 현재 카운터 값 가져오기
counter=$(echo "$json_content" | jq 'length')

# 현재 시간과 카운터 값을 사용하여 새로운 데이터 추가
new_entry=$(jq --argjson hour "$((counter + 1))" \
               --arg total "$total_cpu_usage" \
               --arg pc_usage "$pc_usage" \
               --arg lc_usage "$lc_usage" \
               --arg pc_cores "$pc_cores" \
               --arg lc_cores "$lc_cores" \
               '. += [{
                   "hour": ($hour | tonumber),
                   "전체 CPU 사용률": ($total | tonumber),
                   "P.C CPU 사용률": ($pc_usage | tonumber),
                   "L.C CPU 사용률": ($lc_usage | tonumber),
                   "P.C 코어 수": ($pc_cores | tonumber),
```

[그림 3.2.3-14] cpu.usage.sh 2

해당 코드는 전체 CPU 사용률을 계산, 각 CPU 코어의 사용률을 배열(cpu_usage)에 저장, P.C(첫 번째 코어)와 L.C(두 번째 코어)의 사용률을 계산한다. 기존의 CpuData.json 파일이 존재하면 내용을 불러오고, 없으면 빈 배열로 초기화하여 새로운 CPU 사용률 데이터를 시간과 함께 추가한다.

cpu_time_ratio.sh

```
# CPU 시간 비율 추출
cpu_time=$(mpstat -P ALL 1 1 | awk '$12 ~ /[0-9.]+/ {print $5, $7, $14}' | head -1)

# 사용자 시간, 시스템 시간, 유휴 시간 추출
user_time=$(echo "$cpu_time" | awk '{print $1}')
system_time=$(echo "$cpu_time" | awk '{print $2}')
idle_time=$(echo "$cpu_time" | awk '{print $3}')

#--------------------------------------
```

[그림 3.2.3-15] cpu_time_ratio.sh 1

```
# 기존 JSON 파일에서 데이터를 읽어오기
if [ -f "$json_file" ]; then
    json_content=$(cat "$json_file")
else
    json_content="[]"
fi

# 현재 카운터 값 가져오기
counter=$(echo "$json_content" | jq 'length')

# JSON 형식으로 새 데이터 추가
new_entry=$(jq --argjson hour "$((counter + 1))" \
               --arg user "$user_time" \
               --arg system "$system_time" \
               --arg idle "$idle_time" \
               '. += [{
                   "hour": $hour,
                   "사용자 시간": ($user | tonumber),
                   "시스템 시간": ($system | tonumber),
                   "유휴 시간": ($idle | tonumber)
               }]' <<< "$json_content")

# JSON 파일에 새로운 데이터 저장
echo "$new_entry" | jq . > "$json_file"
echo "3. Cpu 사용시간 검사 완료"
```

[그림 3.2.3-16] cpu_time_ratio.sh 2

해당 코드는 각 CPU의 시간 비율(사용자 시간, 시스템 시간, 유휴 시간)을 추출하여 기존의 CpuTime.json 파일을 불러와 시간별로 저장한다.

vm_usage.sh

```
# 메모리 사용 정보 추출
memory_info=$(top -bn1 | grep "MiB Mem" | awk '{print $4, $8, $6}')

# 현재 메모리 사용 정보를 수집
total_memory=$(echo "$memory_info" | awk '{print $1}')
used_memory=$(echo "$memory_info" | awk '{print $2}')
available_memory=$(echo "$memory_info" | awk '{print $3}')
memory_usage=$(echo "$memory_info" | awk '{printf("%d\n", $2/$1 * 100.0)}')

# 정수만 가능한 경우 하기 코드로 변환
total_memory=$(echo "$memory_info" | awk '{print int($1)}')
used_memory=$(echo "$memory_info" | awk '{print int($2)}')
available_memory=$(echo "$memory_info" | awk '{print int($3)}')
memory_usage=$(echo "$memory_info" | awk '{printf("%d\n", $2/$1 * 100.0)}')

#----------------------------------------

# JSON 파일의 경로
json_file="VMemory.json"

# 기존 JSON 파일에서 데이터를 읽어오기
if [ -f "$json_file" ]; then
    json_content=$(cat "$json_file")
else
    json_content="[]"
```

[그림 3.2.3-17] vm_usage.sh 1

```
else
    json_content="[]"
fi

# 현재 카운터 값 가져오기
counter=$(echo "$json_content" | jq 'length')

# JSON 형식으로 데이터 생성
new_entry=$(jq --argjson hour "$((counter + 1))" \
               --arg total "$total_memory" \
               --arg used "$used_memory" \
               --arg available "$available_memory" \
               --arg usage "$memory_usage" \
               '. += [{
                   "hour": $hour,
                   "총 메모리": ($total | tonumber),
                   "사용 중인 메모리": ($used  | tonumber),
                   "사용 가능한 메모리": ($available | tonumber),
                   "메모리 사용률": ($usage | tonumber)
               }]' <<< "$json_content")


# JSON 파일에 새로운 데이터 저장
echo "$new_entry" | jq . > "$json_file"
echo "2. Virtual Memory 검사 완료"
```

[그림 3.2.3-18] vm_usage.sh 2

해당 코드는 VMemory.json 파일을 불러와 메모리 정보를 추출하여 저장한다. 가상메모리의 상태를 확인하는 스크립트이며 시간(hour), 총 메모리, 사용 중인 메모리, 사용 가능한 메모리, 메모리 사용률로 저장한다.

swap_usage.sh

```
# 메모리 사용 정보 추출
swap_info=$(top -bn1 | grep "MiB Swap" | awk '{print $3, $7, $9}')

# 현재 스왑 메모리 사용 정보를 수집
total_swap=$(echo "$swap_info" | awk '{print $1}')
used_swap=$(echo "$swap_info" | awk '{print $2}')
available_swap=$(echo "$swap_info" | awk '{print $3}')

#--------------------------------------

# JSON 파일의 경로
json_file="SMemory.json"

# 기존 JSON 파일에서 데이터를 읽어오기
if [ -f "$json_file" ]; then
    json_content=$(cat "$json_file")
else
    json_content="[]"
fi

# 현재 카운터 값 가져오기
counter=$(echo "$json_content" | jq 'length')

# JSON 형식으로 데이터 생성
new_entry=$(jq --argjson hour "$((counter + 1))" \
```

[그림 3.2.3-19] swap_usage.sh 1

```
# 현재 카운터 값 가져오기
counter=$(echo "$json_content" | jq 'length')

# JSON 형식으로 데이터 생성
new_entry=$(jq --argjson hour "$((counter + 1))" \
               --arg total "$total_swap" \
               --arg used "$used_swap" \
               --arg available "$available_swap" \
               '. += [{
                   "hour": $hour,
                   "총 스왑 메모리": ($total | tonumber),
                   "사용 중인 스왑 메모리": ($used | tonumber),
                   "사용 가능한 스왑 메모리": ($available | tonumber)
               }]' <<< "$json_content")

# JSON 파일에 새로운 데이터 저장
echo "$new_entry" | jq . > "$json_file"
echo "1. Swap Memory 검사 완료"
```

[그림 3.2.3-20] swap_usage.sh 2

기존의 SMemory.json 파일을 불러와 시간(hour), 총 Swap 메모리, 사용 중인 Swap 메모리, 사용 가능한 Swap 메모리 정보를 저장한다.

이와 같이 취약점 진단 및 모니터링 정보를 수집한 후 monitor.sh 파일에서 move_to_DB.sh를 실행하게 된다. move_to_DB는 데이터베이스에 진단 결과, 모니터링 정보를 전송한 후 남은 파일들을 삭제한다.

```
# MongoDB Atlas 연결 정보
MONGO_URI="mongodb+srv://admin:NrLJhyWCPsAw2Jcz@cluster0.7anrpml.mongodb.net/$1?retryWrites=true&w=majority

# 인자로 데이터베이스 이름을 받음 (웹에서 )
if [ "$#" -ne 1 ]; then
        echo "Usage: $0 <database_name>"
        exit 1
fi

DATABASE_NAME="$1"

# JSON 파일 경로 (적절히 수정하세요)
BASE_PATH="/home/user1/testData"
JSON_FILES=(
    "$BASE_PATH/ChartData.json"
    "$BASE_PATH/TextData.json"
    "$BASE_PATH/CpuData.json"
    "$BASE_PATH/CpuTime.json"
    "$BASE_PATH/SMemory.json"
    "$BASE_PATH/VMemory.json"
    "$BASE_PATH/Solutions.json"
)

echo -e "[MongoDB 데이터 임포트 \033[32m시작\033[0m]"

# 모든 파일이 성공적으로 업로드되었는지를 추적하는 변수
```

[그림 3.2.3-21] move_to_DB.sh 1

MongoDB Atlas에 연결하기 위한 URI를 변수에 저장한 후 DB에 들어갈 데이터베이스명을 인자로 입력하면, 미리 설정된 업로드할 JSON 파일들의 위치를 따라가 배열로 묶어 각 파일을 차례대로 처리할 수 있도록 한다.

```
# 모든 파일이 성공적으로 업로드되었는지를 추적하는 변수
all_success=true

# 각각의 파일을 해당 컬렉션에 삽입
for FILE in "${JSON_FILES[@]}"; do
    # 파일 이름에서 컬렉션 이름 추출 (예: file1.json -> file1)
    COLLECTION_NAME=$(basename "$FILE" .json)

    # mongoimport 명령어 실행
    mongoimport --uri="$MONGO_URI" --collection="$COLLECTION_NAME" --file="$FILE" --jsonArray

    # 결과 확인
    if [ $? -eq 0 ]; then
        echo -e "$FILE 을 $COLLECTION_NAME 콜렉션에 업로드 \033[32m성공\033[0m"
    else
        echo -e "\033[31m$FILE 임포트에 실패\033[0m"
        all_success=false  # 하나라도 실패하면 false로 변경
    fi

    echo "---------------------------------"

done
echo -e "[MongoDB 데이터 임포트 \033[31m종료\033[0m]"

# 모든 파일이 성공적으로 업로드되었는지 확인
if $all_success; then
```

[그림 3.2.3-22] move_to_DB.sh 2

.json 확장자를 제거한 이름을 MongoDB의 컬렉션 이름으로 사용하기 위해 파일명에서 이름을 추출한 후 DB로 업로드한다.

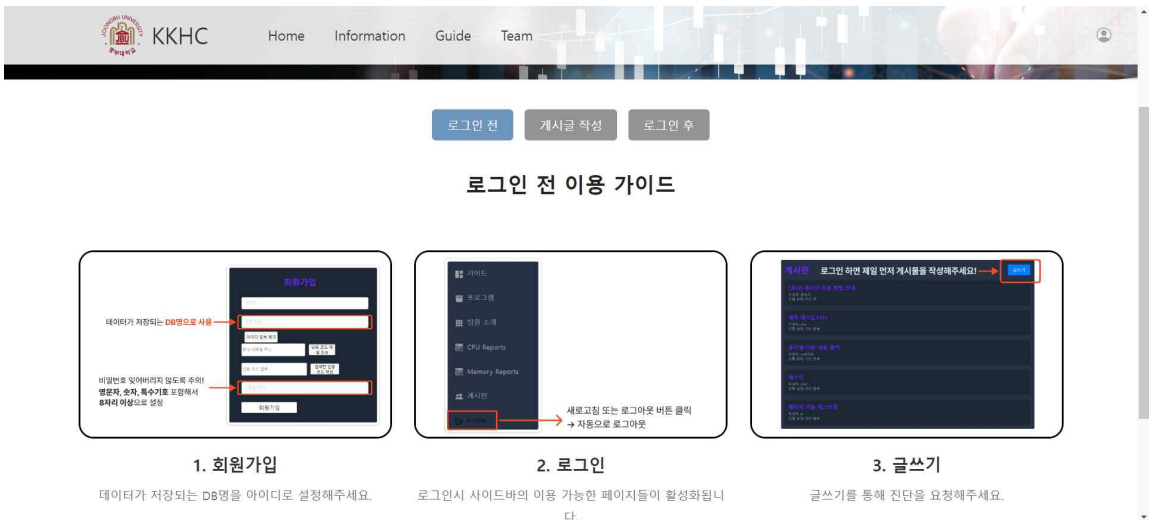
```
    # mongoimport 명령어 실행
    mongoimport --uri="$MONGO_URI" --collection="$COLLECTION_NAME" --file="$FILE" --jsonArray

    # 결과 확인
    if [ $? -eq 0 ]; then
        echo -e "$FILE 을 $COLLECTION_NAME 콜렉션에 업로드 \033[32m성공\033[0m"
    else
        echo -e "\033[31m$FILE 임포트에 실패\033[0m"
        all_success=false  # 하나라도 실패하면 false로 변경
    fi

    echo "---------------------------------"

done
echo -e "[MongoDB 데이터 임포트 \033[31m종료\033[0m]"

# 모든 파일이 성공적으로 업로드되었는지 확인
if $all_success; then
    echo -e "모든 파일의 업로드 \033[32m성공\033[0m"
    echo "testData 파일을 삭제합니다."
    rm -rf testData
else
    echo -e "\033[32m하나 이상의 파일이 업로드에 실패했습니다.\033[0m"
    echo "testData 파일을 열어 수동으로 업로드하세요."
fi
```

[그림 3.2.3-23] move_to_DB.sh 3

실패한 파일이 있으면 수동 업로드 권장 문장을 출력하고, 모든 파일이 성공적으로 업로드되었으면 testData 디렉터리를 삭제한다.

4. 활 용

4.1. 실제 사용



[그림 4.1-1] 서비스 페이지 가이드 메뉴

가이드 메뉴에 들어가면 서비스 페이지와 프로그램 페이지를 어떻게 사용해야 하는지를 확인할 수 있다. 가이드는 크게 로그인 전, 게시글 작성, 로그인 후로 나누어진다.

[그림 4.1-2] 메인 페이지 및 프로그램 페이지로 이동

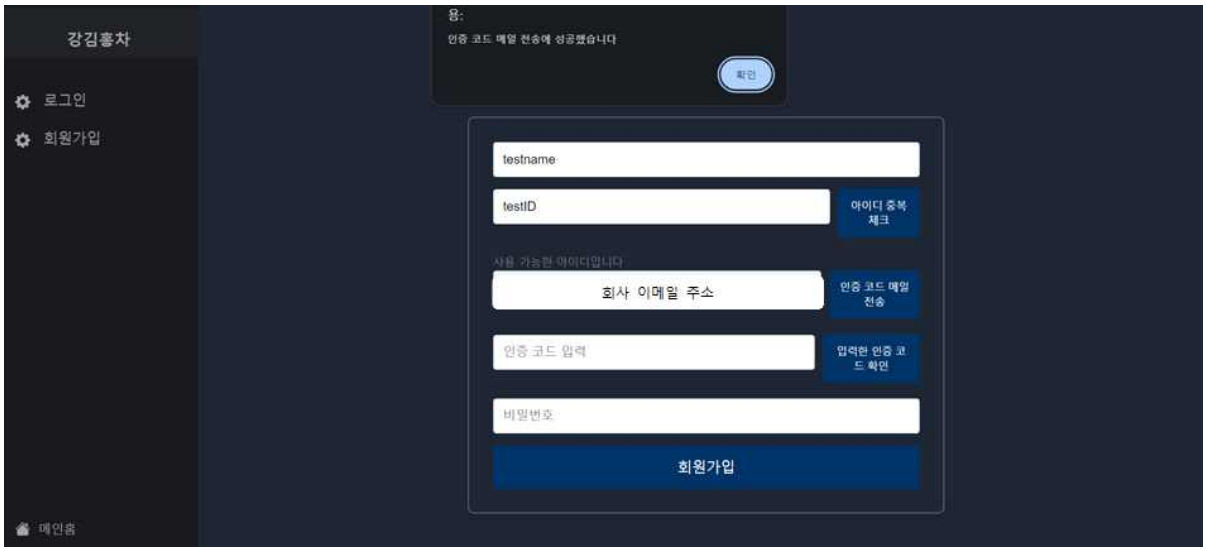
사용자가 접속하면 제일 먼저 보게 되는 메인 페이지이다. 가이드를 확인한 후 바로가기 버튼을 클릭해 서버 진단 요청을 할 수 있는 프로그램 페이지로 이동하게 된다.

[그림 4.1-3] 프로그램 페이지 메인 화면

사용자가 프로그램 페이지에 접속하면 제일 먼저 보게 되는 화면이다. 하단에 메인홈을 통해 가이드, 사용 기술 등이 적혀있는 서비스 페이지로 이동할 수 있다. 프로그램 페이지는 회원가입 및 로그인을 진행해야만 이용할 수 있으므로 회원가입을 기준으로 페이지 이용 방법 및 기능을 소개하고 있다.

[그림 4.1-4] 회원가입 페이지 - 아이디 중복 체크

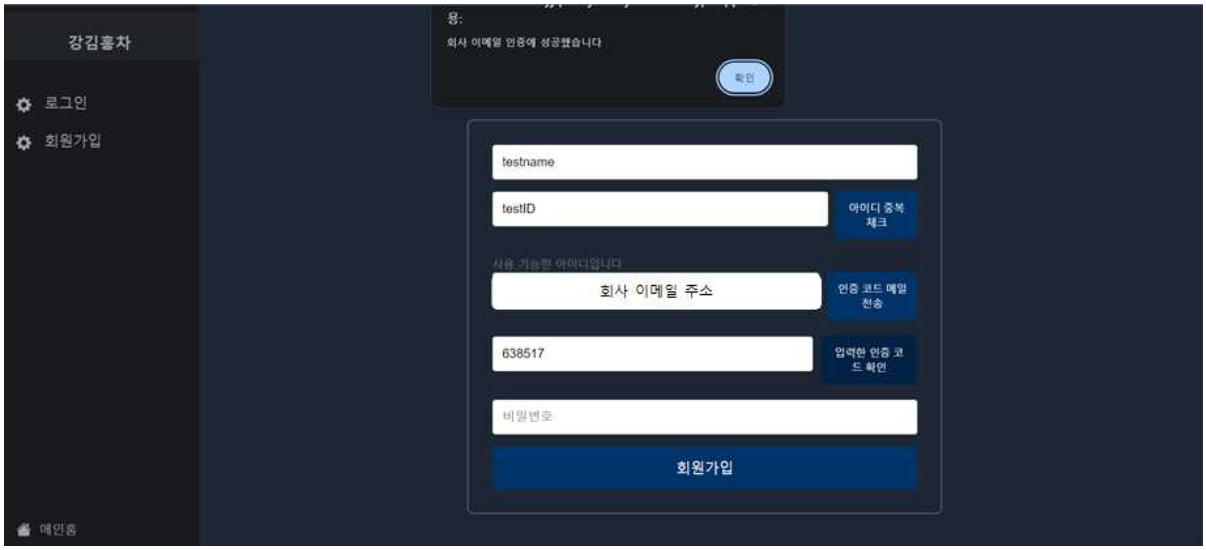
회원가입 페이지다. 회원가입을 하기 위해선 모든 입력란을 채우고 아이디 중복 체크, 회사 이메일 인증 코드 확인, 비밀번호 조건을 충족해야만 한다. 아이디의 경우 이미 데이터 베이스에 저장된 아이디를 사용할 수 없기 때문에 아이디 중복 체크를 통해 '사용 가능한 아이디입니다' 와 같은 알림이 표시되어야만 다음 단계로 넘어갈 수 있다.



[그림 4.1-5] 회원가입 페이지 - 인증 코드 메일 전송

[그림 4.1-6] 회원가입 페이지 - 전송된 메일 내용

사용자가 입력한 회사 이메일 주소로 랜덤으로 생성된 인증코드가 첨부된 메일이 전송된다.

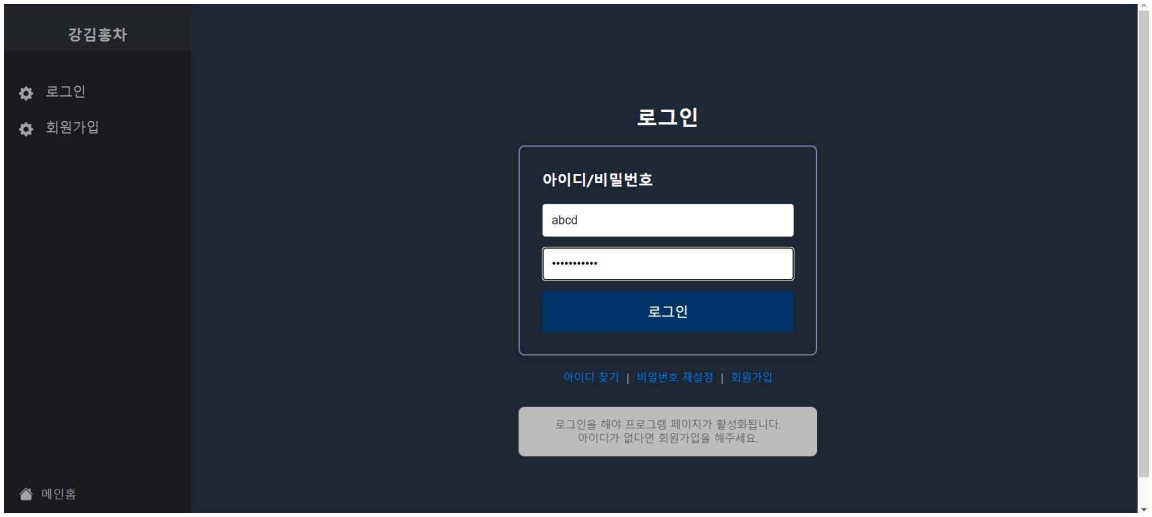


[그림 4.1-7] 회원가입 페이지 - 회사 이메일 유효성 검증

전송된 메일에 있던 인증 번호를 입력한다. 데이터 베이스에 저장되어있는 회사 이메일 주소, 인증코드 값과 일치하면 회사 이메일 유효성 검증이 완료된다. 회사 이메일 주소만 사용자의 개인 정보가 들어있는 데이터 베이스에 보관되고 기존에 회사 이메일 주소 및 인증코드값을 가지고 있던 데이터 베이스는 초기화된다.

[그림 4.1-8] 회원가입 페이지 - 비밀번호 확인

사용자가 영문자, 숫자, 특수문자를 포함해서 총 8자리 이상의 비밀번호를 만들어야지만 회원가입을 진행할 수 있다.



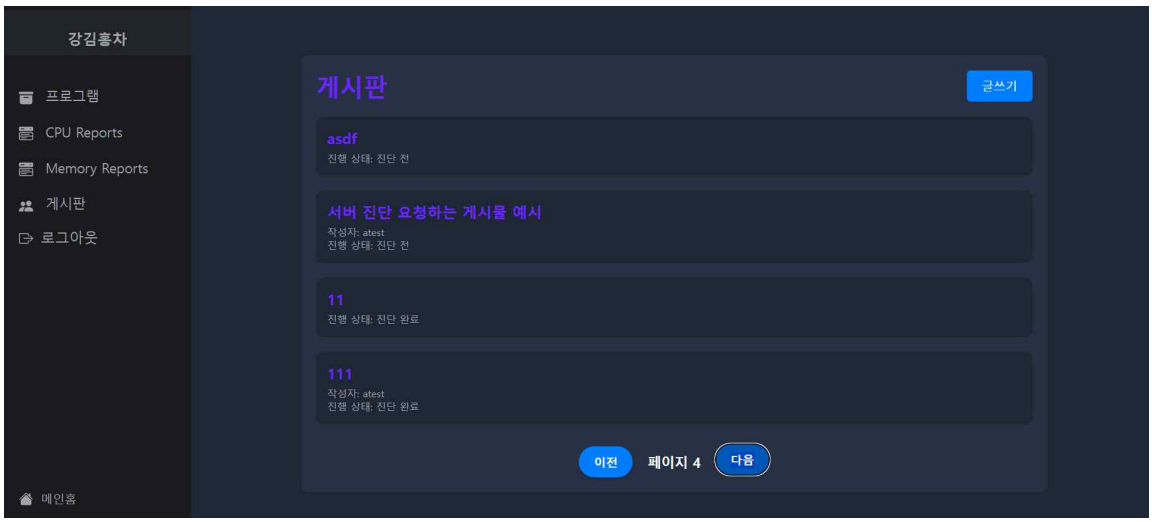
[그림 4.1-9] 로그인 페이지

사용자가 가입할 때 작성한 아이디, 비밀번호를 넣어준다. 아이디가 생각나지 않으면 아이디 찾기에서 이름과 이메일을 입력해 아이디를 찾는다. 로그인에 성공하면 사용가능한 기능 중에는 프로그램, 레포트, 게시판, 로그아웃이 있다. 이중 사용자는 제일 먼저 관리자에게 서버 진단 요청을 해야하므로 게시판에 가서 진단을 요청하는 글을 작성한다.

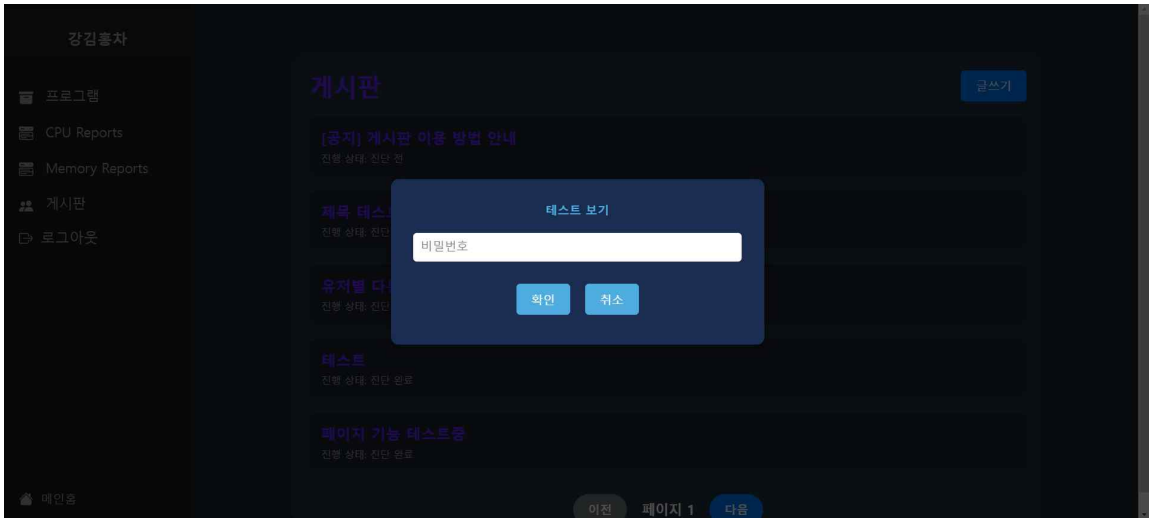
[그림 4.1-10] 게시판 페이지에서 서버 진단 요청 글 양식

[그림 4.1-11] 게시판 페이지에서 글이 새로 생성된 화면

사용자는 양식에 맞춰 서버 진단을 요청하는 글을 작성하게 된다. 빈칸이 있을 경우 글 작성이 완료되지 않는다. 성공적으로 게시물을 생성하면 자신이 쓴 게시물은 게시물 생성할 때 입력했던 비밀번호 없이 볼 수 있다. 수정 기능을 통해 제목, 내용을 수정할 수 있고 삭제는 게시물을 생성할 때 입력했던 비밀번호를 입력해야만 삭제가 가능하다.



[그림 4.1-12] 게시판 페이지에서 게시물 목록 및 진행 상태 확인



[그림 4.1-13] 게시판 페이지에서 타인의 게시물을 클릭했을 때 표시되는 화면

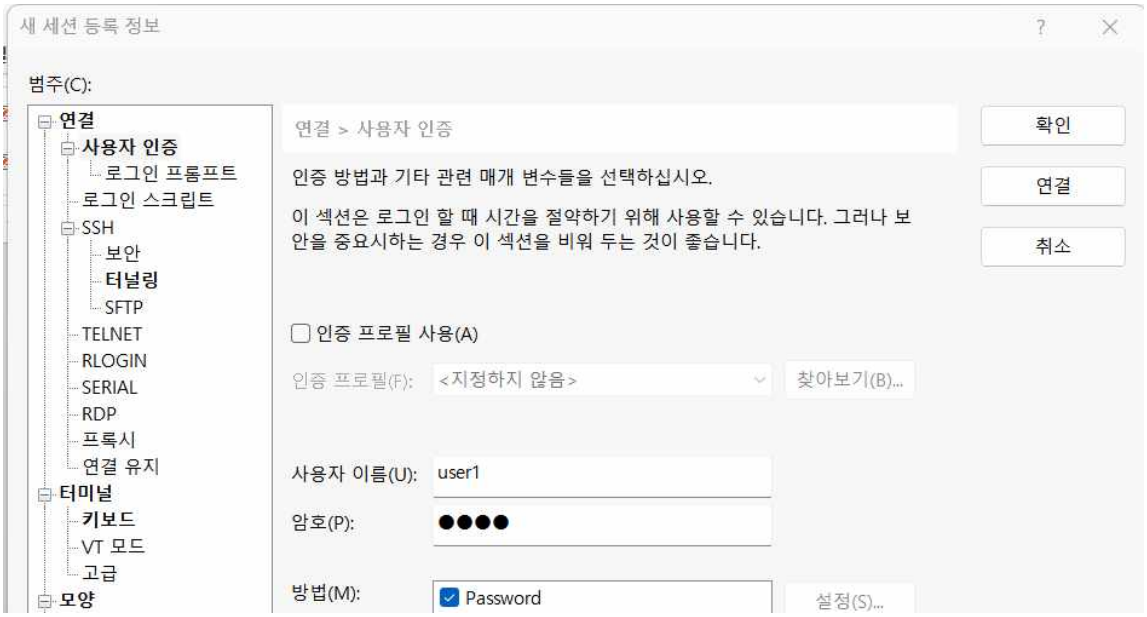
게시물은 본인이 작성한 게시물 외에는 전부 진행 상태만 볼 수 있다. 진행 상태는 진단 전, 진단 완료로 구분된다.

리눅스 과정

관리자 - SSH를 통한 원격 접속

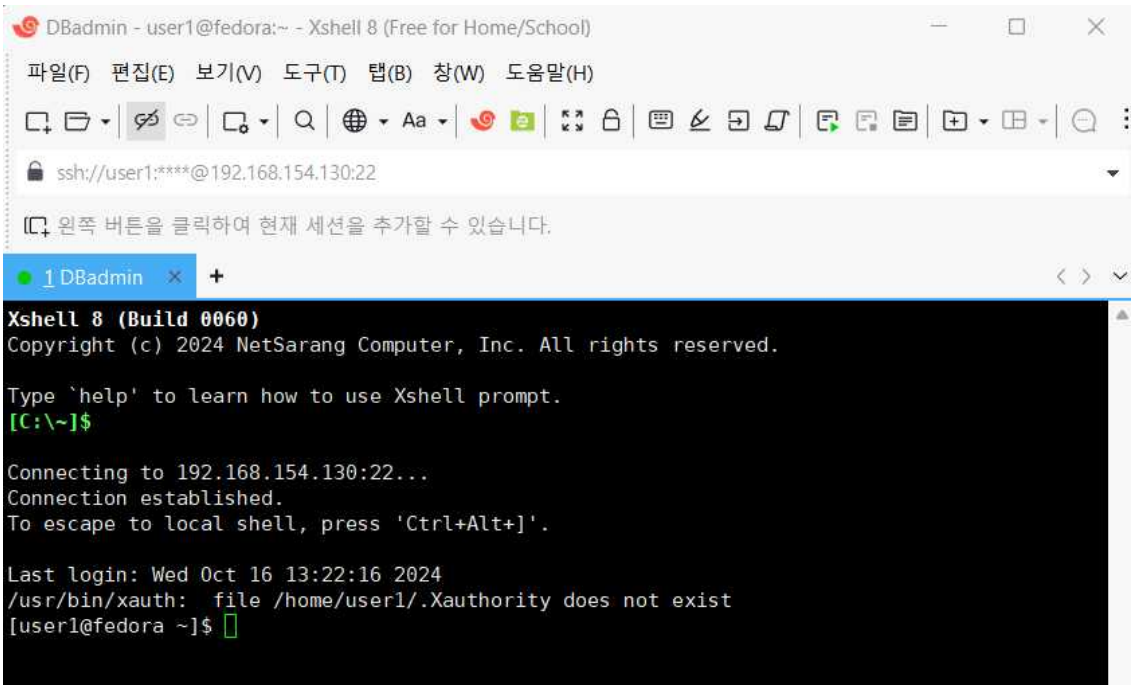
[그림 4.1-26] xshell 세션 등록(연결)

xshell에서 ssh연결을 위해 관리자 리눅스 서버의 호스트명과 프로토콜/포트를 지정한다.



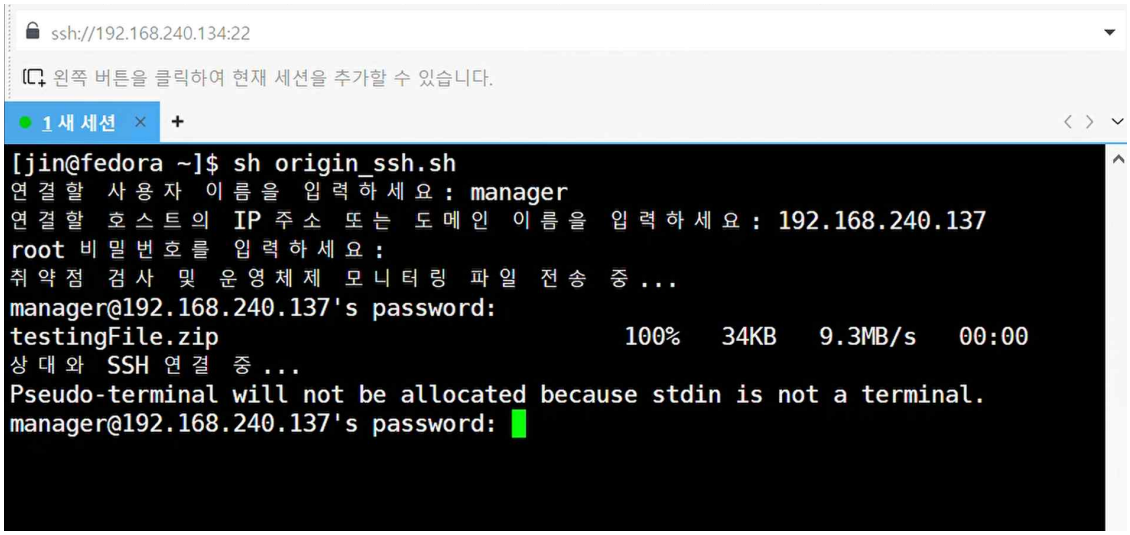
[그림 4.1-27] xshell 세션 등록(연결)

사용자 인증에서 관리자 리눅스 서버의 유저명과 password를 입력 후 세션을 등록한다.



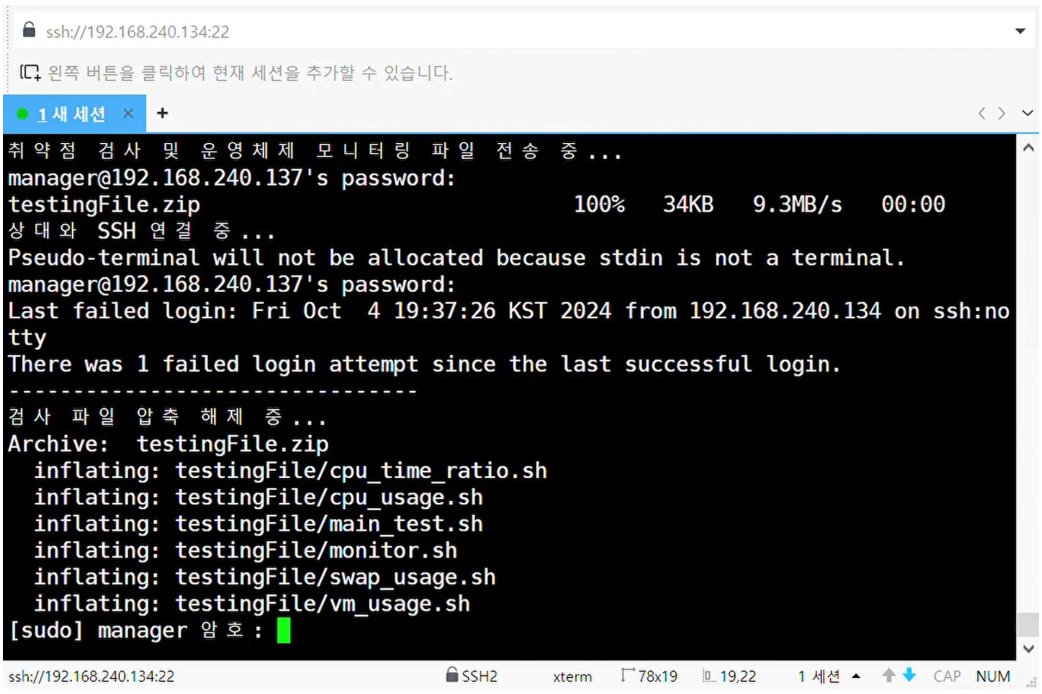
[그림 4.1-28] xshell 관리자 서버 접속

해당 세션으로 들어가면 관리자 리눅스 서버와 연결된 것을 확인할 수 있다.



[그림 4.1-29] xshell Client 서버 접속

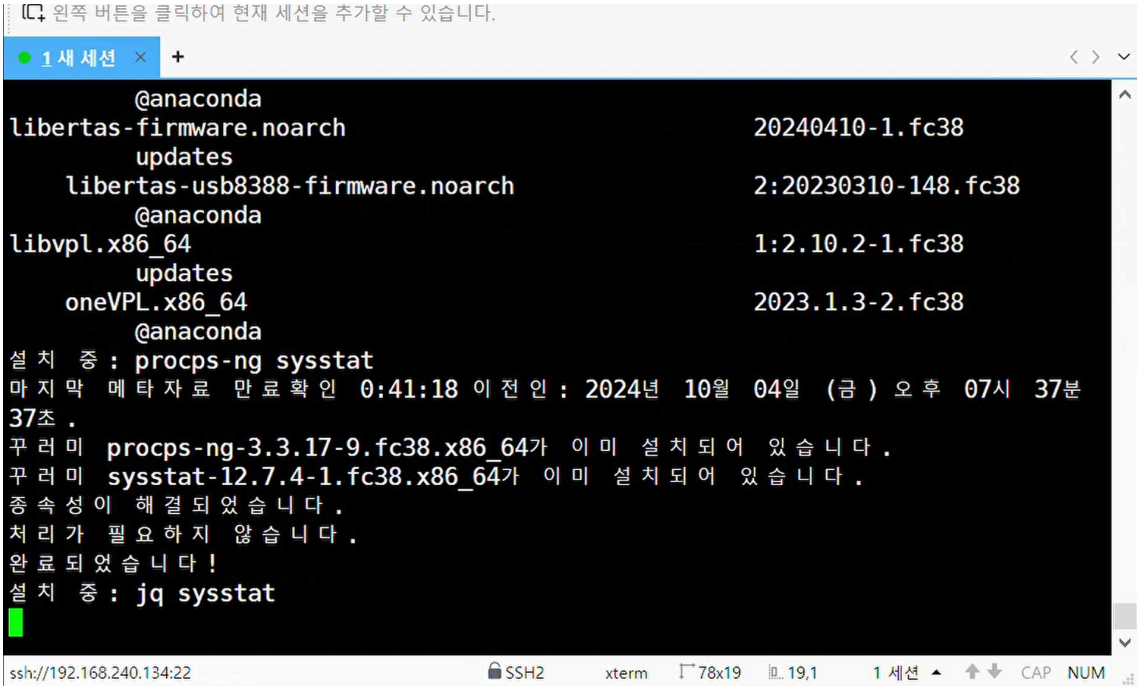
이후 앞서 코드 설명에서와 같이 관리자 리눅스 서버에 저장되어 있는 origin_ssh.sh 파일을 실행해 고객 서버와 연결을 시도한다. 연결할 사용자(고객)의 이름과 IP, root 계정 비밀번호를 입력하고 나면 검사 파일을 전송하고, SSH 연결을 시도한다.



[그림 4.1-30] xshell 진단 파일 압축 해제

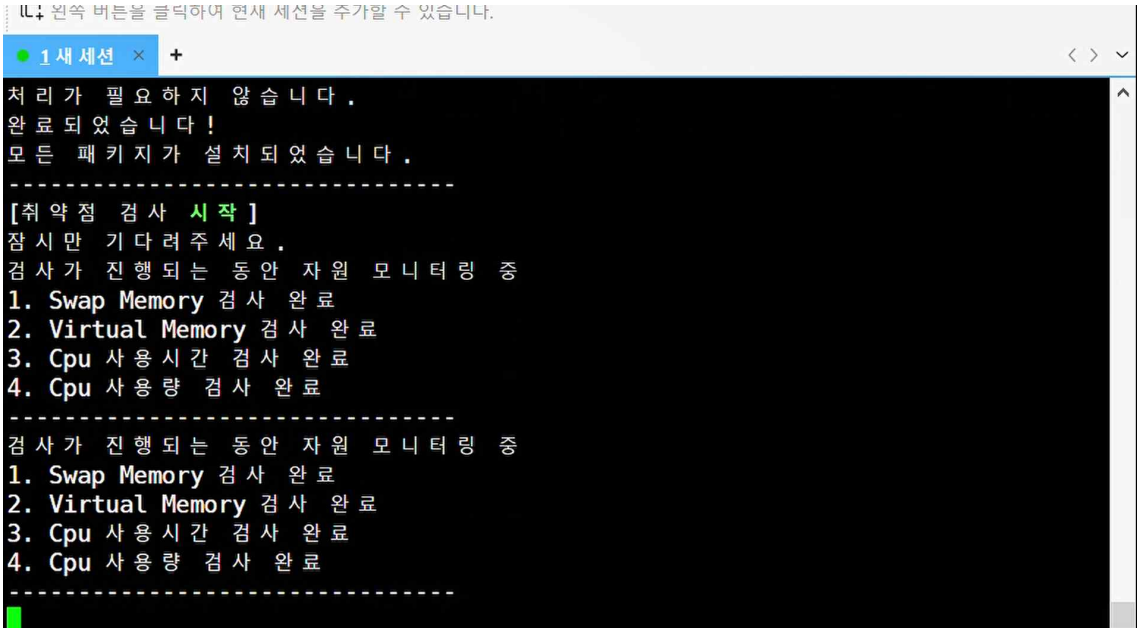
무사히 원격 연결에 성공하면 검사 파일의 압축을 해제하고 본격적인 검사를

시작한다.



[그림 4.1-31] xshell 진단 패키지 설정

우선 검사에 필요한 패키지를 업데이트 한다.

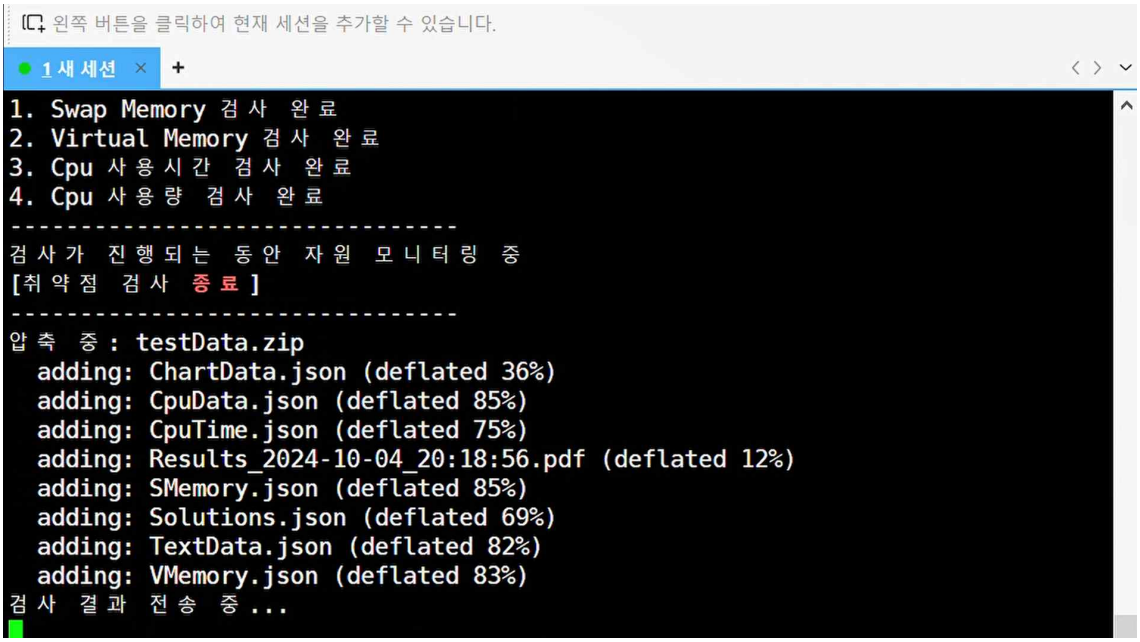


[그림 4.1-32] xshell 진단

패키지 설치가 완료되면 위 사진과 같이 취약점 검사 시작을 알리고, 자원 모니

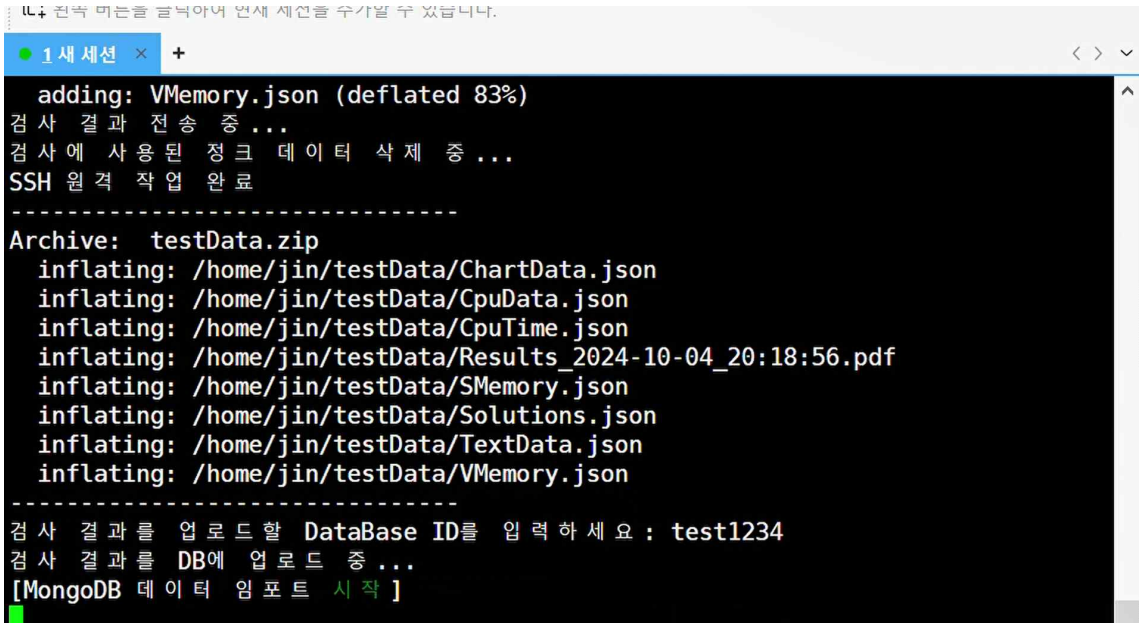
터링이 반복되는 것도 출력한다.

.



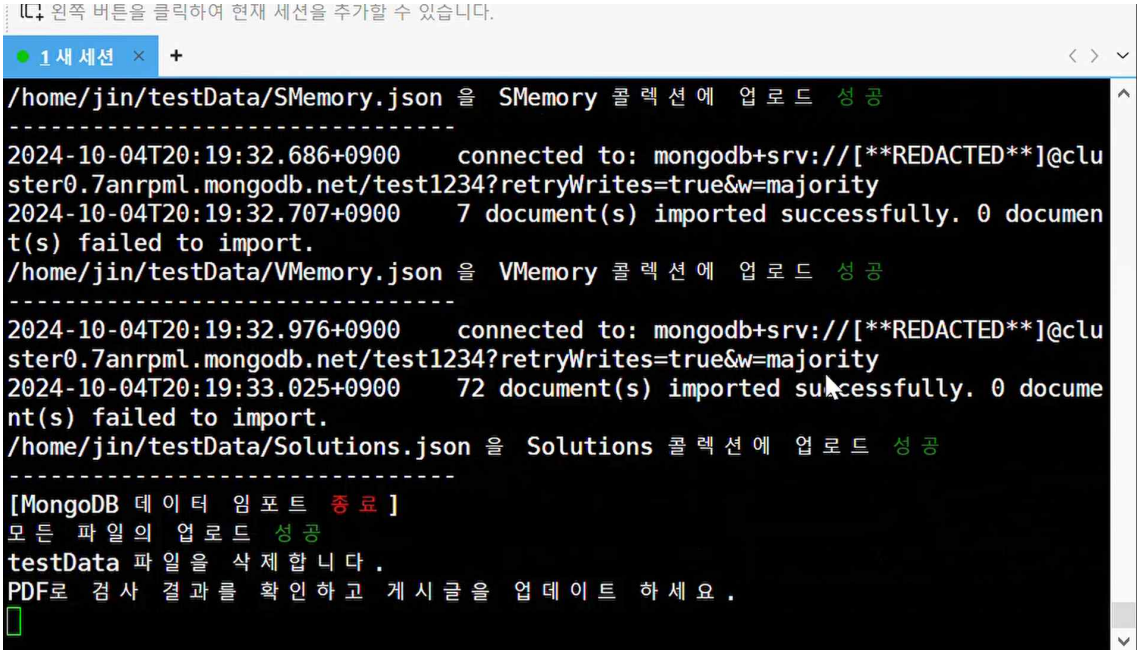
[그림 4.1-33] xshell 진단 결과 전송

검사가 끝나면 결과 파일 압축 과정을 출력하고, 코드에 미리 입력된 관리자 리눅스 서버의 정보를 토대로 검사 결과 zip 파일을 전송한다.



[그림 4.1-34] xshell 진단 결과 DB에 업로드

고객의 서버에 생성되었던 모든 파일들을 삭제한 후 원격 연결을 종료한다. 이후 DataBase ID를 입력받아 move_to_DB.sh 파일을 실행한다.



[그림 4.1-35] xshell 진단 pdf 결과

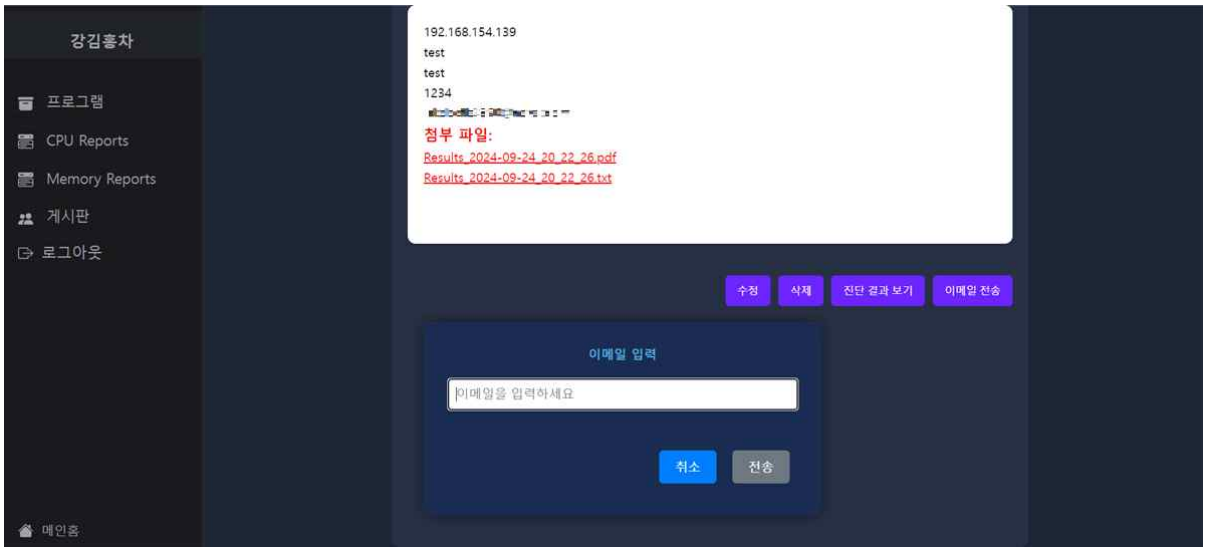
데이터 베이스에 결과 파일 업로드가 완료되면 모든 파일의 업로드 결과에 따라 성공과 실패를 나누어 출력하며, 실패할 경우에는 수동으로 업로드하라는 안내를 출력한다. 이후 웹페이지의 게시물을 업데이트하기 위해 검사 결과 파일을 관리자의 윈도우로 전달한다.

관리자는 전달받은 파일을 압축 해제하고 웹 페이지에 관리자로 로그인한다.

[그림 4.1-36] 관리자로 로그인하여 진단 결과 파일을 올리는 화면

관리자로 로그인 후 진단 전 상태인 게시물에 들어가서 작성된 내용을 바탕으

로 리눅스에서 취약점 진단, 모니터링 스크립트를 돌릴 수 있도록 데이터를 전달한다. 이후 진단 결과를 정리한 파일인 PDF, TXT 파일을 사용자가 작성한 게시물에 파일 업로드 기능을 통해 추가해준다.



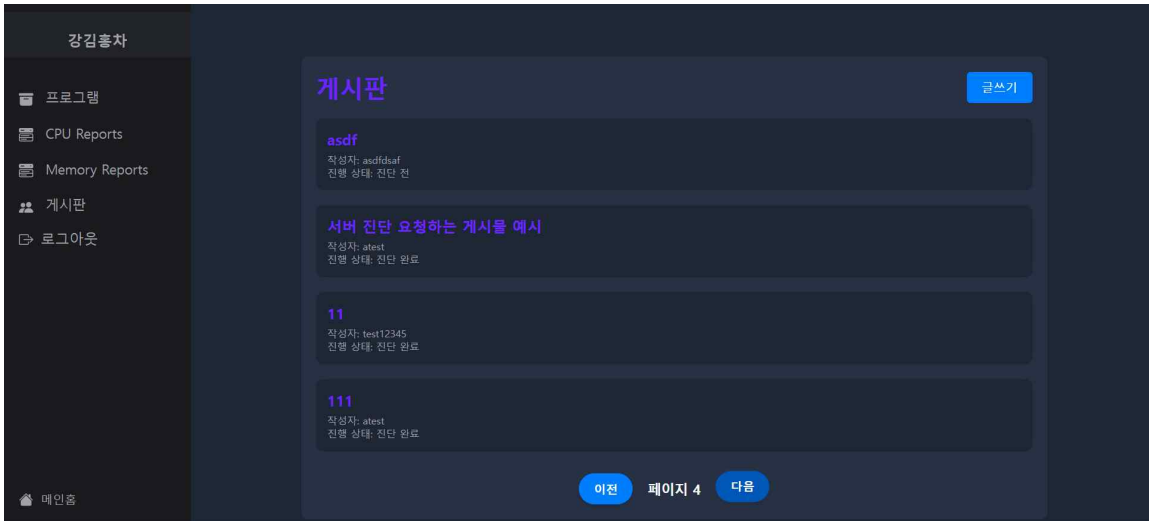
[그림 4.1-37] 관리자가 유저가 입력해준 이메일 주소에 진단 완료 알림 보내는 화면

관리자가 파일을 첨부한 후 게시물을 작성한 작가에게 제목 등의 정보를 제공한다.

[그림 4.1-38] 관리자가 전송한 진단 완료 메일 내용

진단 완료 메일 내용에는 진단이 완료된 게시물의 제목, 프로그램 페이지 주소

등이 들어가 있다.

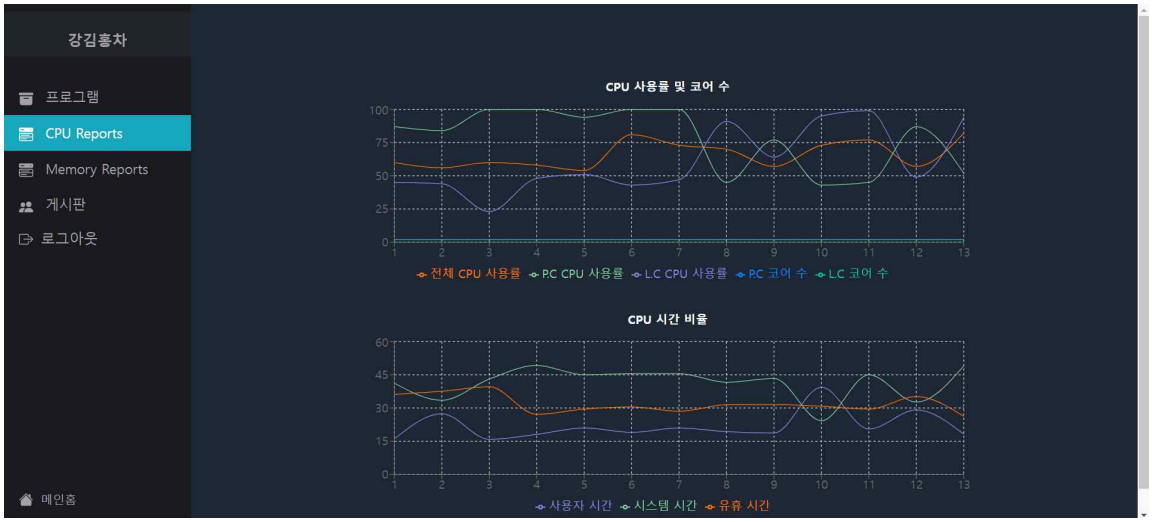


[그림 4.1-39] 관리자로 로그인하여 진단 결과 파일을 올리는 화면

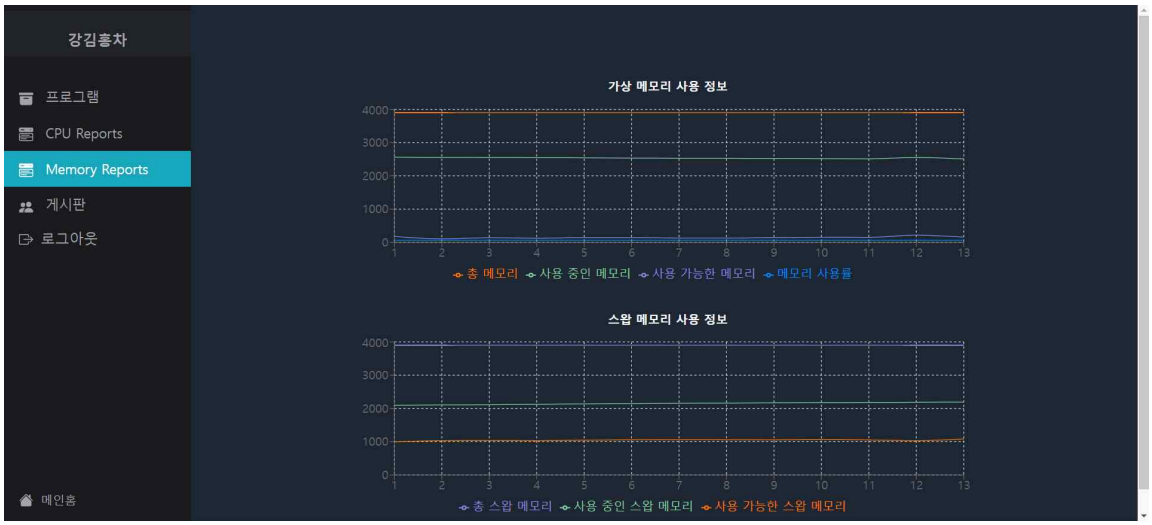
관리자가 올린 파일이 정상적으로 올라가면 진행 상태가 진단 완료로 바뀐다.

[그림 4.1-40] 프로그램 페이지에서 데이터 베이스 설정하는 화면

진단 완료 알림을 받은 사용자가 프로그램 페이지로 접속한다. 로그인 후 프로그램 메뉴에서 가입할 때 입력한 ID로 개인 DB 설정을 완료한다. 개인 DB에는 리눅스에서 돌린 취약점 진단 결과와 모니터링 결과가 저장되어 있다.

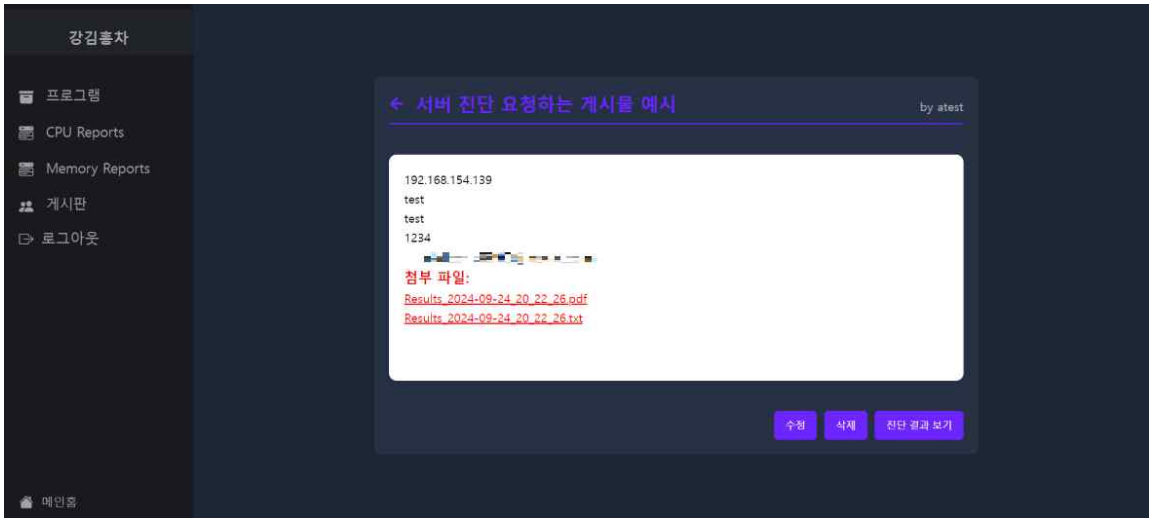


[그림 4.1-41] 모니터링 결과값중 CPU 부분 결과 그래프로 확인

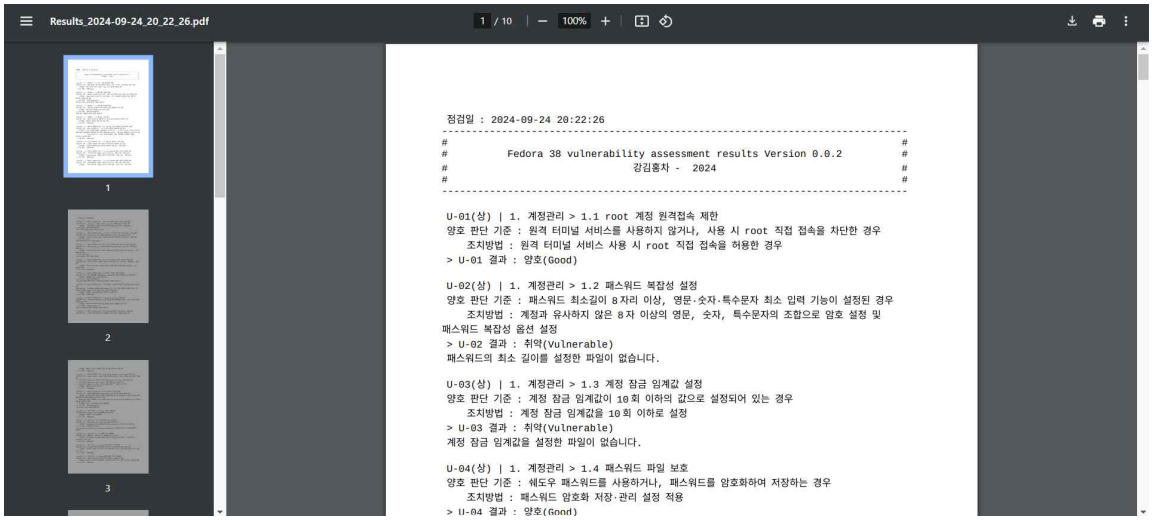


[그림 4.1-42] 모니터링 결과값중 메모리 부분 그래프로 확인

CPU Reports에서는 CPU 사용률 및 코어수, CPU 시간 비율을 그래프로 확인할 수 있다. Memory Reports에서는 Swap 메모리와 가상 메모리의 사용률 등을 그래프로 확인할 수 있다.



[그림 4.1-43] 유저가 작성한 게시물에서 결과 파일 다운



[그림 4.1-44] 진단 결과 파일 내용

관리자가 올려준 진단 결과 파일들을 본인이 작성한 게시물에 가면 다운 및 미리보기가 가능하다.

[그림 4.1-45] 취약점 진단 결과값을 표, 그래프로 나타내는 화면

본인이 작성한 게시물에서 진행 상태가 진단 완료로 변경되면 진단 결과 자세히 보기 버튼이 활성화된다. 클릭하면 취약점 진단 결과값을 표와 그래프로 확인할 수 있다. 그래프에서는 전체 항목 개수, 취약 개수, 양호 개수, 담장자와 상의가 필요한 N/A 항목 개수가 총 몇 개인지 그래프로 나타내었다.

[그림 4.1-46] 취약점 진단 결과 검색 기능

표는 취약점 진단 결과의 값 전체를 표현했다. 결과값이 양호, N/A는 파란색으로, 취약만 붉은색으로 표시했다. 표에서는 id(진단 항목), 분류 (계정관리), 결과 (취약 등)로 검색이 가능하다.

[그림 4.1-47] 취약 항목별 조치 방법

결과가 취약인 값을 클릭하면 해당 항목에 대한 조치 방법만 출력된다.

5. 결 론

5.1. 결론

서버의 취약점 진단 결과와 CPU와 메모리 사용률 등을 동시에 확인할 수 있어 현재 사용하고 있는 서버의 상태와 이상이 있는 부분을 파악하고 점검해볼 수 있다. 또한 단순히 진단 결과만 제공하는 것이 아닌 문제가 발생한 항목에 대한 조치 방법도 같이 제공하기 때문에 사용자가 문제점을 쉽게 개선할 수 있다. 취약점 진단은 주기적으로 점검하는 것이 좋기 때문에 PDF, TXT 파일을 제공하여 이전 점검기록과 비교하여 개선된 점 및 새로운 취약점을 비교하기 용이하고, 현재 상황을 쉽게 볼 수 있도록 그래프와 표로 제공하여 사용자가 편리하게 이용하도록 하였다. 마지막으로 사용자가 진단 요청 및 점검 파일을 언제든지 다운받을 수 있도록 게시판을 운영하여 효율적으로 사이트를 관리할 수 있도록 하였다.

5.2. 기대 효과

취약점 진단은 주기적으로 진행하는 것이 좋지만 시간도 소모되고 파일 형식으로 제공되기 때문에 검사를 진행하는 것은 물론, 검사 결과를 보고 이해하기 쉽지 않다는 문제점이 있다. 따라서 Vulner은 사용자에게 비교적 친숙한 게시판 형태로 서버 진단 요청을 하고 결과물을 기존 방식인 파일로 받아볼 수 있는 것뿐만 아니라 그래프와 표로 가시성을 높여 사용자에게 취약점에 대한 이해도와 수정해야 할 부분을 명확하게 알려줄 수 있다. 또한 진단 결과뿐만 아니라 취약한 항목에 대한 조치 방법도 표와 그래프로 표현하므로 취약점 확인, 해당 취약점 수정, 서버 취약점 재진단 요청 및 현재 서버 상태 확인을 한 페이지에서 할 수 있어 사용자가 편리하게 이용할 수 있고, 이는 목표 중 하나였던 주기적인 서버 취약점 점검 유도의 달성을 기대해 볼 수 있다.

6. 별 첨

- Vulner 프로젝트 진단 서비스 (github 소스코드)

- 웹 사이트 (github 소스코드)

- 웹 사이트 링크

시연 영상

https://youtu.be/nC-XKStVy0g

팀원 소개

강채린 92113437 web front & backend

차윤지 92113877 web front & backend

김솔 92113542 리눅스 스크립트 & 모니터링 개발

홍예진 92113944 리눅스 스크립트 & 모니터링 개발

소개 자료

발표 자료(별도 첨부)
소개 팜플릿(별도 첨부)