|  |  |
| --- | --- |
| 기술문서 | 2020-08-27 |

주요 정보 통신 기반 시설 취약점

분석 평가 점검 항목 보고서

작성자 : 이원석

**개 정 이 력**

|  |  |  |
| --- | --- | --- |
| **개정번호** | **개정 내용 요약** | **개정일자** |
| 1.0 | 리포트 작성 | 2020-08-26 |
| 2.0 | 문서 수정 | 2020-08-27 |
| 3.0 | ELK 연동 부분 추가 작성 | 2020-08-28 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**목차**

|  |
| --- |
| [1. 개요](#_heading=h.gjdgxs)  가. 의의  나. 목적  2. 분석 과정  가. 분석 환경  나. 분석 도구  다. 분석 도구 활용  1) 각 도구 이름  3. 상세분석  가. 항목 별로 제목 및 단계를 지정하여 분석 내용을 작성한다.    4. 결론  5. 참고 문헌 |

**스크립트 결과 및 ELK Service 결과**

* **스크립트의 실행 결과 ( 일반 사용자용 결과 파일 (txt 형식) )**

****

* **ELK 서비스 구축 결과**

1. **구축 사전 준비**

* Main Ubuntu Server 와 ELK VM Server 의 파일 전송을 위해 서로의 ssh-key파일을 등록하였음.
* 추가적으로 스크립트로 손쉬운 파일 전송을 위해 hosts에 서로의 Alias를 등록하였음.

1. **ELK 서비스 구축**

* **LOGSTASH : 수집 CONFIG 파일 (logstash-csv.conf)**

|  |
| --- |
| input {  file {  path => "/home/user/won/\*.csv"  start\_position => "beginning"  incedb\_path => "/dev/null"  }  }  filter {  csv {  separator => ","  skip\_header => true  columns => ["date","ip","os","item","message"]  }  }  output {  elasticsearch {  hosts => "localhost:9200"  index => "vul-won"  }  stdout {}  } |

* **KIBANA : 데이터 분류 및 시각화 CONFIG**

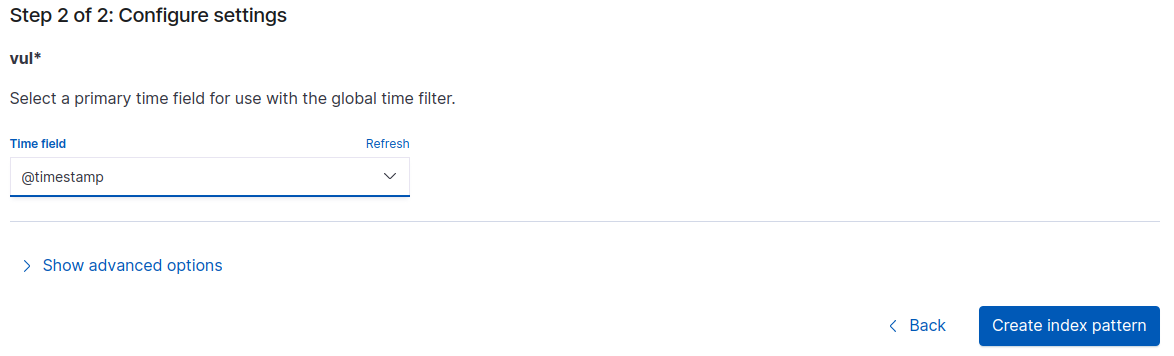
|  |
| --- |
| server.host: "0.0.0.0" |

* **ELK 종합 실행 부분**

|  |
| --- |
| **FIREWALL :** 5601/tcp 해당 포트의 권한 허용  **SERVICE :** KIBANA, Elrasticsearch service start (enable)  **cron :** \*/10 \* \* \* \* \* ./logstash/bin/logstash -f /logstash/config/logstash.yml & : 10분 마다 실행 |

* **ELK 서비스 시각화 확인.**

1. **logstash 에서 cron 을 이용해 주기적으로 받은 데이터를 index 화 시킨다. (Time field : timstamp)**



1. **Kibana - [discover] 탭에서 받아온 로그 시각화가 가능합니다.**



# 1. 개요

## 의의

주요 정보 통신 기반 시설 기술적 취약점 분석 평가 문서를 참고하여 개인, 기업용 UNIX/LINUX SERVER의

해킹, 크래킹등의 방지 및 예방을 위한 서버내의 보안성 검사의 취지를 가지고 있는 프로젝트이다.

즉 운영 서버의 정보 보안 보호 역량 강화를 위한 기술적 안내를 위함이다.

## 목적

▶ 주요 정보 통신 기반 시설 담당자의 기반 시설 보호 역량 강화를 위한 기술적 안내 제공.

▶ 기술적 취약점 분석 평가 문서를 참고로 기술적 취약점 분석 자동화

▶ 즉각적인 로그 데이터 모니터링을 위한 ELK 구축

# 2. 분석 스크립트 개발 과정

## 분석 환경

▶ 분석 환경에 대하여 서술한다.

* ELK 구축 서버 : Centos 7
* 스크립트 작성 서버 : ubuntu 18.04

스크립트 동작 OS : CENTOS 7 이상 , Ubuntu 18.04 이상

## 개발 도구

|  |  |  |
| --- | --- | --- |
| **명칭** | **설명** | **UI** |
| (bash)**Won.sh** | 주요정보통신기반시설의 의거한 취약점 분석 스크립트 |  |
| elrasticseach | logstash를 통해 수신한 데이터를 저장하는 저장소 |  |
| kibana | 데이터를 시각적으로 정리 및 실시간 탐색 및 모니터링 |  |
| logstash | 수집 데이터를 인덱싱하여 elrasticseach에 전달한다. |  |

## 개발 도구 활용

### 각 도구 이름

각 도구의 활용 방법에 대하여 서술한다.

* **ELK 서버**

1. 가상 서버에서 자동으로 로그를 수집 하도록 설정하였음.
2. 대쉬보드에서 시각화된 데이터를 확인이 가능하다.
3. elrasticseach 는 Enable 상태로 항상 구동을 유지 중 이다.. (스크립트 또한 구동을 확인하고 동작)
4. kibana 또한 Enable 상태로 항상 구동을 유지 중.
5. logstash의 경우 일정 시간 별로 crontab을 이용해 동작한다.

* **won.sh**

1. chmod + x won.sh 명령어 입력
2. ./won.sh 실행

# 3. 상세 개발내용

## 항목별로 제목 및 단계를 지정하여 분석 내용을 작성한다.

**1. Root 권한 확인.**

|  |
| --- |
| # check root  if [ "$USER" != "root" ]  then  echo "root 권한으로 스크립트를 실행하여 주십시오."  exit  fi |

**1.1 동작 로직**

|  |
| --- |
| $USER 환경변수를 사용, 현재 로그인 사용자가 “ROOT” 인지 확인하는 코드 |

**2. 취합 데이터 파일명 설정.**

|  |
| --- |
| result=`hostname`"\_result\_"`date +%F\_\_%T`.txt  \*\*\*(중략) >> $result 2>&1 |

**2.1 동작 로직**

|  |
| --- |
| $USER 환경변수를 사용, 현재 로그인 사용자가 “ROOT” 인지 확인하는 코드 |

**3. root 계정 원격 접속 제한 확인.**

|  |
| --- |
| grep ^pts /etc/securetty > /dev/null 2>&1  if [ $? -eq 0 ];then  echo " ==> [취약] root 직접 접속이 허용되었거나, 원격서비스를 사용 중 입니다." >> $result 2>&1  echo " ==> [취약] root 직접 접속이 허용되었거나, 원격서비스를 사용 중 입니다." >> 01.txt 2>&1  echo " ==> [취약] vi /etc/securetty 파일에서 pts 주석처리." >> $result 2>&1  echo " ==> [취약] vi /etc/securetty 파일에서 pts 주석처리." >> 01.txt  else  echo " ==> [안전] 원격 서비스를 사용하지 않고 있거나, 접속이 차단 되어 있습니다." >> $result 2>&1  fi  cat /etc/pam.d/login | grep "^auth required /lib/security/pam\_securetty.so" > /dev/null 2>&1  if [ $? -eq 0 ];then  echo " ==> [안전] pam\_securetty.so 파일을 통해 사용자 인증에 대한 보안이 적용되어 있습니다." >> $result 2>&1  else  echo " ==> [취약] pam\_securetty.so 파일을 통한 사용자 인증에 대한 보안이 적용되어 있지 않>습니다." >> $result 2>&1  echo " ==> [취약] vi /etc/pam.d/login 파일에서 해당 내용을 추가하거나 주석을 해제하십시오." >> $result 2>&1  echo " ==> [취약] auth required /lib/security/pam\_securetty.so" >> $result 2>&1  fi |

**3.1 동작 로직**

|  |
| --- |
| 원격 터미널 사용시 root 계정 직접 접속 허용 여부를 확인하기 위해서  관련 파일들의 컨피그 설정 여부, 관련 모듈 설정을 판별하고 확인한다.  "/etc/securetty” 의 설정 내용을 분석한다. |

**4. 패스워드 복합성 설정(및 정책) 확인.**

|  |
| --- |
| if [ $max = 60 ]  then  echo " ==> [안전] 최대 사용 기간 : $max"일"" >> $result 2>&1  else  echo " ==> [취약] 최대 사용 기간 : $max"일"" >> $result 2>&1  fi  if [ $min = 1 ]  then  echo " ==> [안전] 최소 사용 시간 : $min"일"" >> $result 2>&1  else  echo " ==> [취약] 최소 사용 시간 : $min"일"" >> $result 2>&1  fi  if [ $date = 7 ]  then  echo " ==> [안전] 기간 만료 경고 기간(일) : $date"일"" >> $result 2>&1  else  echo " ==> [취약] 기간 만료 경고 기간(일) : $date"일"" >> $result 2>&1  fi |

**4.1 동작 로직**

|  |
| --- |
| 정보 통신 기반 문서에 명시되어있는 적절한 패스워드 Config 설정에 의거해서  보안적으로 안전한 설정 값이 아닌 Config 설정을 판별하여 확인한다.  보안상 안전한 값  최대 사용 기간 : 60일  최소 사용 시간 : 1시간  기간 만료 경고 기간 : 7일 |

**5. 계정 잠금 임계값 설정 확인.**

|  |
| --- |
| cat /etc/pam.d/common-auth | grep "^auth required /lib/security/pam\_tally.so" | grep deny > /dev/null 2>&1  if [ $? -eq 0 ]; then  cat /etc/pam.d/common-auth | grep "^account required /lib/security/pam\_tally.so" > /dev/null 2>&1  if [ $? -eq 0 ]; then  echo " ==> [안전] 계정 잠금 임계값이 5 이하의 값으로 설정되어 있는 경우입니다 : " >> $result 2>&1  else  echo " ==> [취약] 계정 잠금 임계값이 설정되어 있지 않거나, 5 이하의 값으로 설정되지 않은 >경우 입니다." >> $result 2>&1  echo " ==> [취약] vi /etc/pam.d/common-auth 확인." >> $result 2>&1  echo " ==> [취약] auth required /lib/security/pam\_tally.so deny=5 unlock\_time=120 no\_magic\_root" >> $result 2>&1  echo " ==> [취약] account required /lib/security/pam\_tally.so no\_magic\_root reset" >> $result 2>&1  else  echo " ==> [취약] 계정 잠금 임계값이 설정되어 있지 않거나, 5 이하의 값으로 설정되지 않은 >경우 입니다." >> $result 2>&1  echo " ==> [취약] vi /etc/pam.d/common-auth 확인." >> $result 2>&1  echo " ==> [취약] auth required /lib/security/pam\_tally.so deny=5 unlock\_time=120 no\_magic\_root" >> $result 2>&1  echo " ==> [취약] account required /lib/security/pam\_tally.so no\_magic\_root reset" >> $result 2>&1  fi |

**5.1 동작 로직**

|  |
| --- |
| 계정 잠금 임계값이 설정되어있는 CONFIG의 설정을 확인 및 분석.  보안적으로 안전한 설정 값 : 5  설정되어있지 않거나 5이하의 값을 판별한다. |

**6. 패스워드 파일 보호 확인.**

|  |
| --- |
| if [ "`cat /etc/passwd | grep "^root" | awk -F: '{print $2}'`" = x ]  then  if test -r /etc/shadow  then  echo " ==> [안전] Shadow 패스워드 시스템을 사용중입니다" >> $result 2>&1  else  echo " ==> [취약] Passwd 패스워드 시스템을 사용중입니다" > $result 2>&1  echo " ==> [취약] Passwd 패스워드 시스템을 사용중입니다" > 04.txt 2>&1  fi  fi |

**6.1 동작 로직**

|  |
| --- |
| 쉐도우 패스워드를 사용하지 않거나, 패스워드를 암호화하여 저장하지 않는 부분을 확인한다.  "/etc/passwd”의 비밀번호가 암호화 되어있는지."  "/etc/shadow”파일을 사용하여 패스워드를 보안화 하였는지 확인. |

**7. root 홈, 패스 디렉터리 권한 및 패스 설정 확인.**

|  |
| --- |
| if echo $PATH | grep "^\." > /dev/null 2>&1 ; then  echo " ==> [취약] PATH 환경변수 값에 '.'이 맨 앞에 포함되어 있습니다." >> $result 2>&1  echo " ==> [취약] vi /etc/profile 파일에서 수정" >> $result 2>&1  echo " ==> [취약] (수정 전) PATH=.:\$PATH:\$HOME/bin" >> $result 2>&1  echo " ==> [취약] (수정 후) PATH=\$PATH:\$HOME/bin:." >> $result 2>&1  elif echo $PATH | grep "::" > /dev/null 2>&1 ; then  echo " ==> [취약] PATH 환경변수 값에 '::'이 포함되어 있습니다." >> $result 2>&1  echo " ==> [취약] vi /etc/profile 파일에서 수정" >> $result 2>&1  echo " ==> [취약] (수정 전) PATH=\$PATH::\$HOME/bin" >> $result 2>&1  echo " ==> [취약] (수정 후) PATH=\$PATH:\$HOME/bin:" >> $result 2>&1  else  echo " ==> [안전] PATH 설정에 문제가 없습니다." >> $result 2>&1  fi |

**7.1 동작 로직**

|  |
| --- |
| $PATH환경 변수에 “. “ 이 맨 앞이나 중간에 포함되어 있는 경우를 체크 및 확인한다.  현재 쉘과 유저에서 설정되어있는 기본 PATH 환경변수를 확인 후 해당 특수 문자들을 판별. |

**8. 파일 및 디렉터리 소유자 설정 확인.**

|  |
| --- |
| find / \( -nouser -o -nogroup \) -xdev -ls > /dev/null 2>&1  if [ "$?" -eq 0 ]  then  echo " ==> [안전] 소유자 혹은 그룹이 없는 파일 및 디렉터리가 존재하지 않습니다" >> $result 2>&1  else  echo " ==> [취약] 소유자 혹은 그룹이 없는 파일 및 디렉터리가 존재합니다" >> $result 2>&1  echo " ==> [취약] 소유자 혹은 그룹이 없는 파일 및 디렉터리가 존재합니다" >> 06.txt 2>&1  fi |

**8.1 동작 로직**

|  |
| --- |
| 서버 내에 소유자가 존재하지 않는 파일 및 디렉터리가 존재하는 경우를 판단 및 확인한다.  “fine” 명령을 이용하여 사용자, 그룹이 없는 파일을 전체 경로로 검색하여 판단한다.  해당 명령어의 결과 값에 따라 판별. |

**9. “/etc/passwd “ 파일 소유자 및 권한 설정 확인.**

|  |
| --- |
| if ls -l /etc/passwd | grep '\-rw\-r\-\-r\-\- 1 root' > /dev/null 2>&1 ; then  echo " ==> [안전] /etc/passwd 파일의 소유자는 root, 권한 644로 정상입니다." >> $result 2>&1  else  echo " ==> [취약] /etc/passwd 파일의 소유자 혹은 권한이 잘못 설정되어 있습니다." >> $result 2>&1  echo " ==> [취약] /etc/passwd 파일의 소유자 혹은 권한이 잘못 설정되어 있습니다." >> 07.txt 2>&1  echo " ==> [취약] chown root /etc/passwd" >> $result 2>&1  echo " ==> [취약] chown root /etc/passwd" >> 07.txt 2>&1  echo " ==> [취약] chmod 644 /etc/passwd" >> $result 2>&1  echo " ==> [취약] chmod 644 /etc/passwd" >> 07.txt 2>&1  fi |

**9.1 동작 로직**

|  |
| --- |
| “/etc/passwd” 파일의 소유자가 “root”가 아니거나 권한이 “644” 이하가 아닌 경우를 판단 및 확인한다.  조건으로 파일의 권한을 명시 하였음.  해당 조건에 맞지 않는 경우 취약처리. |

**10. “/etc/shadow” 파일 소유자 및 권한 설정 확인.**

|  |
| --- |
| if test `ls -l /etc/shadow | awk {'print $1'} ` = -r--------.  then  echo " ==> [안전] 권한 : "`ls -l /etc/shadow | awk {'print $1'}` >> $result 2>&1  else  if test `ls -l /etc/shadow | awk {'print $1'} ` = ----------.  then  echo " ==> [안전] 권한 : "`ls -l /etc/shadow | awk {'print $1'}` >> $result 2>&1  else  echo " ==> [취약] 권한 : "`ls -l /etc/shadow | awk {'print $1'}` >> $result 2>&1  fi  fi  if test `ls -l /etc/shadow | awk {'print $3'}` = root  then  echo " ==> [안전] 소유자 : " `ls -l /etc/shadow | awk {'print $3'}` >> $result 2>&1  else  echo " ==> [취약] 소유자 : " `ls -l /etc/shadow | awk {'print $3'}` >> $result 2>&1  fi  if test `ls -l /etc/shadow | awk {'print $4'} ` = root  then  echo " ==> [안전] 그룹 : "`ls -l /etc/shadow | awk {'print $4'}` >> $result 2>&1  else  echo " ==> [취약] 그룹 : "`ls -l /etc/shadow | awk {'print $4'}` >> $result 2>&1  fi |

**10.1 동작 로직**

|  |
| --- |
| “/etc/shadow” 파일의 소유자가 “root”가 아니거나, 권한이 “400”이 아닌 경우를 판별 및 확인한다.  파일의 각각의 소유자와 권한에 해당하는 컬럼을 조건으로 판별하여 안전과 취약을 분리하였다. |

**11. “/etc/hosts” 파일 소유자 및 권한 설정 확인.**

|  |
| --- |
| if ls -l /etc/hosts | grep '\-rw\-\-\-\-\-\-\- 1 root' > /dev/null 2>&1 ; then  echo " ==> [안전] /etc/hosts 파일의 소유자는 root, 권한 600으로 정상입니다." >> $result 2>&1  else  echo " ==> [취약] /etc/hosts 파일의 소유자 혹은 권한이 잘못 설정되어 있습니다." >> $result 2>&1  echo " ==> [취약] /etc/hosts 파일의 소유자 혹은 권한이 잘못 설정되어 있습니다." >> 09.txt 2>&1  echo " ==> [취약] chown root /etc/hosts" >> $result 2>&1  echo " ==> [취약] chown root /etc/hosts" >> 09.txt 2>&1  echo " ==> [취약] chmod 600 /etc/hosts" >> $result 2>&1  echo " ==> [취약] chmod 600 /etc/hosts" >> 09.txt 2>&1  fi |

**11.1 동작 로직**

|  |
| --- |
| /etc/hosts” 파일의 소유자가 “root”가 아니거나, 권한이 “600”이 아닌 경우를 판별 및 확인한다.  파일의 권한과 소유자를 조건으로 취약과 안전을 분류, 판별한다. |

**12. “/etc/(x)inetd.conf” 파일 소유자 및 권한 설정 확인.**

|  |
| --- |
| if test -f /etc/inetd.conf ;  then  echo " ==> [안전] inetd.conf 파일이 존재합니다" >> $result 2>&1  root=`ls -l /etc/inetd.conf | awk '{print $3}'`  per=`ls -l /etc/inetd.conf | awk '{print $1}'`  if [ $root = root ] ;  then  echo " ==> [안전] inetd.conf 파일 소유자 : " $root >> $result 2>&1  else  echo " ==> [취약] inetd.conf 파일 소유자 : " $root >> $result 2>&1  fi  if [ $per = -rw-------. ] ;  then  echo " ==> [안전] inetd.conf 파일 권한 : " $per >> $result 2>&1  else  echo " ==> [취약] inetd.conf 파일 권한 : " $per >> $result 2>&1  fi  else  echo " ==> [취약] inetd.conf 파일이 존재하지 않습니다" >> $result 2>&1  fi  if test -f /etc/xinetd.conf ;  then  echo " ==> [안전] xinetd.conf 파일이 존재합니다" >> $result 2>&1  xroot=`ls -l /etc/xinetd.conf | awk '{print $3}'`  xper=`ls -l /etc/xinetd.conf | awk '{print $1}'`  if [ $xroot = root ] ;  then  echo " ==> [안전] xinetd.conf 파일 소유자 : " $xroot >> $result 2>&1  else  echo " ==> [취약] xinetd.conf 파일 소유자 : " $xroot >> $result 2>&1  fi  if [ $xper = -rw-------. ] ;  then  echo " ==> [안전] xinetd.conf 파일 권한 : " $xper >> $result 2>&1  else  echo " ==> [취약] xinetd.conf 파일 권한 : " $xper >> $result 2>&1  fi  else  echo " ==> [취약] xinetd.conf 파일이 존재하지 않습니다" >> $result 2>&1  fi |

**12.1 동작 로직**

|  |
| --- |
| xinetd.config 파일의 존재 여부와, 존재했을 경우 파일의 소유자, 권한을 체크 및 확인한다.  inetd.conf, xinetd.config 의 존재여부, 소유자 , 권한을 조건으로 판별한다. |

**13. “/etc/syslog.conf “ 파일 소유자 및 권한 설정 확인.**

|  |
| --- |
| if ls -l /etc/rsyslog.conf | grep '\-rw\-r\-\-r\-\- 1 root' > /dev/null 2>&1 ; then  echo " ==> [안전] /etc/rsyslog.conf 파일의 소유자는 root, 권한 644로 정상입니다." >> $result 2>&1  else  echo " ==> [취약] /etc/rsyslog.conf 파일의 소유자 혹은 권한이 잘못 설정되어 있습니다." >> $result 2>&1  echo " ==> [취약] /etc/rsyslog.conf 파일의 소유자 혹은 권한이 잘못 설정되어 있습니다." >> 11.txt 2>&1  echo " ==> [취약] chown root /etc/rsyslog.conf" >> $result 2>&1  echo " ==> [취약] chown root /etc/rsyslog.conf" >> 11.txt 2>&1  echo " ==> [취약] chmod 644 /etc/rsyslog.conf" >> $result 2>&1  echo " ==> [취약] chmod 644 /etc/rsyslog.conf" >> 11.txt 2>&1  fi |

**13.1 동작 로직**

|  |
| --- |
| “/etc/syslog.conf” 파일의 소유자 및 권한을 확인하여 해당 설정 값이 아닐 경우를 확인한다.  파일의 권한과 소유자를 조건으로 취약과 안전을 판별한다. |

**14. “/etc/services “ 파일 소유자 및 권한 설정 확인.**

|  |
| --- |
| root=`ls -l /etc/services | awk '{print $3}'`  per=`ls -l /etc/services | awk '{print $1}'`  if [ $root = root ]  then  echo " ==> [안전] services 파일 소유자 : " $root >> $result 2>&1  else  echo " ==> [취약] services 파일 소유자 : " $root >> $result 2>&1  echo " ==> [취약] services 파일 소유자 : " $root >> 12.txt 2>&1  fi  if [ $per = -rw-r--r--. ]  then  echo " ==> [안전] services 파일 권한 : " $per >> $result 2>&1  else  echo " ==> [취약] services 파일 권한 : " $per >> $result 2>&1  echo " ==> [취약] services 파일 권한 : " $per >> 12.txt 2>&1  fi |

**14.1 동작 로직**

|  |
| --- |
| “/etc/service” 파일의 소유자와 권한을 확인하여 보안 설정 값이 아닌 경우를 판별하여 확인한다.  파일의 소유자와 권한에 관련된 컬럼을 조건으로 취약 안전 여부를 판별하여 확인한다. |

**15. SUID, SGID, Sticky bit 설정 및 권한 설정 확인.**

|  |
| --- |
| FILE1=check\_filelist  LOG\_FILE=check\_log  ls -alrt check\_filelist > /dev/null 2>&1  if [ $? = 0 ]; then  if [ -f $LOG\_FILE ]; then  rm $LOG\_FILE  fi  cat $FILE1 | while read FILENAME1  do  `ls -l $FILENAME1 | awk '{print $1,$9}' >> $LOG\_FILE`  done  cat $LOG\_FILE | while read PERM FILENAME2  do  if [ `echo $PERM | egrep '(s|t)'` ]; then  echo " ==> [안전] $FILENAME2은 특수 권한이 설정되어 있습니다." >> $result 2>&1  else  echo " ==> [취약] $FILENAME2은 특수 권한이 설정되어 있지 않습니다." >> $result 2>&1  echo " ==> [취약] $FILENAME2은 특수 권한이 설정되어 있지 않습니다." >> 13.txt 2>&1  fi  done  else  echo " ==> [정보] 파일리스트가 없어 검사를 진행하지 못했습니다." >> $result 2>&1  fi |

**15.1 동작 로직**

|  |
| --- |
| 주요 실행파일의 권한에 SUID, GUID에 대한 설정이 부여되어있는 경우  특정 파일리스트 목록을 가지고 부적절한 설정이 적용되어있는 경우를 판별 및 확인한다. |

**16. 사용자, 시스템 시작 파일 및 환경 파일 소유자 및 권한 설정 확인.**

|  |
| --- |
| ls -lart ~/.\* | sed '/^$/d' |awk 'BEGIN{OFS=";"}{print $1,$4,$9}' | egrep -v "^d|^l" | egrep -v "/|tota  l" | grep -w "r" >> homep.txt  for i in `cat homep.txt`  do  id2=`echo $i | cut -d";" -f2`  a=`echo $i | cut -d";" -f1`  hper=`echo $i | cut -d";" -f1 | cut -c 9`  name=`echo $i | cut -d";" -f3`  if [ "$USER" = "$id2" ] && [ "$hper" != "w" ] ; then  echo " ==> [안전] 소유자, 권한이 안전합니다. "파일 : "$name" >> $result 2>&1  elif [ "$USER" = "$id2" ] ; then  echo " ==> [취약] 소유자는 동일하나 권한이 다릅니다. "권한 : " $a "파일 : "$name" >> $result 2>&1  echo " ==> [취약] 소유자는 동일하나 권한이 다릅니다. "권한 : " $a "파일 : "$name" >> 14.txt 2>&1  else  echo " ==> [취약] 다른 소유자 파일입니다. "소유자 : " $id2 "파일 : "$name " >> $result 2>&1  echo " ==> [취약] 다른 소유자 파일입니다. "소유자 : " $id2 "파일 : "$name " >> 14.txt 2>&1  fi  done  rm -rf homep.txt |

**16.1 동작 로직**

|  |
| --- |
| 홈 디렉터리의 환경 변수 파일 소유자가 “root” 혹은 해당 사용 계정이 아닐 경우  파일의 other의 권한에 “쓰기” 권한이 설정되어 있는 경우를 판별하여 확인한다.  조건을 둘 데이터들을 파일로 저장한 뒤 조건 판별이 끝나고 파일을 삭제한다. |

**17. /dev에 존재하지 않는 device 파일 점검 확인.**

|  |
| --- |
| DF="Device\_file.txt"  find /dev -type f -exec ls -l {} \; > $DF  check=`ls -l Device\_file.txt | awk '{print $5}'`  check2=`cat Device\_file.txt`  if [ $check = 0 ]  then  echo " ==> [안전] 존재하지 않는 파일이 없습니다." >> $result 2>&1  rm -rf $DF  else  rm -rf $DF  echo " ==> [취약] 존재하지 않는 파일이 있습니다 : $check2" >> $result 2>&1  echo " ==> [취약] 존재하지 않는 파일이 있습니다 : $check2" >> 15.txt 2>&1  fi |

**17.1 동작 로직**

|  |
| --- |
| “dev” 에 대한 파일 미점검 또는, 존재하지 않은 “device” 파일을 방치한 경우를 판별하여 확인한다.  "Device\_file.txt” 에 점검 할 파일의 목록을 넣고 스크립트를 실행하게 되면  목록 외의 비정상 디바이스 파일을 판별하여 확인한다. |

**18. /$HOME/.rhosts, hosts.equiv 사용 금지 확인.**

|  |
| --- |
| if [ -f /etc/hosts.equiv ]; then  if ls -l /etc/hosts.equiv | grep '\-rw\-\-\-\-\-\-\- 1 root' > /dev/null 2>&1 ; then  echo " ==> [안전] /etc/hosts.equiv를 사용 중, 소유자 및 권한이 옳바릅니다." >> $result 2>&1  else  echo " ==> [취약] /etc/hosts.equiv를 사용 중, 소유자 및 권한이 옳바르지 않습니다." >> $result 2>&1  echo " ==> [취약] chown root /etc/hosts.equiv" >> $result 2>&1  echo " ==> [취약] chown root /etc/hosts.equiv" >> $result 2>&1  echo " ==> [취약] chmod 600 /etc/hosts.equiv" >> $result 2>&1  fi  else  echo " ==> [안전] /etc/hosts.equiv 파일을 사용하고 있지 않습니다." >> $result 2>&1  fi  if [ -f $HOME/.rhosts ]; then  if ls -l $HOME/.rhosts | grep '\-rw\-\-\-\-\-\-\- 1 root' > /dev/null 2>&1 ; then  echo "==> [안전] $HOME/.rhosts를 사용 중, 소유자 및 옯바릅니다." >> $result 2>&1  else  echo " ==> [취약] $HOME/.rhosts를 사용 중이며, 소유자 혹은 권한이 옯바릅니다." >> $result 2>&1  echo " ==> [취약] chown root \$HOME/.rhosts" >> $result 2>&1  echo " ==> [취약] chmod 600 \$HOME/.rhosts" >> $result 2>&1  fi  else  echo " ==> [안전] $HOME/.rhosts 파일을 사용하고 있지 않습니다." >> $result 2>&1  fi |

**18.1 동작 로직**

|  |
| --- |
| “$HOME/.rhosts”, “/etc/hosts.equiv” 파일의 소유자가 “root”이 아니거나  파일의 권한이 600이 아닌 경우를 판별하여 확인한다  해당 파일들의 존재 여부, 소유자, 권한을 조건으로 취약과 안전을 판별하여 확인한다. |

**19. hosts.\* 설정 체크 확인.**

|  |
| --- |
| if [ -f /etc/hosts.deny ]; then  if grep "ALL:ALL" /etc/hosts.deny > /dev/null 2>&1 ; then  echo " ==> [안전] /etc/hosts.deny 설정이 정상적입니다." >> $result 2>&1  else  echo " ==> [취약] /etc/hosts.deny 설정이 비정상적입니다." >> $result 2>&1  echo " ==> [취약] /etc/hosts.deny 설정이 비정상적입니다." >> 17.txt 2>&1  fi  else  echo " ==> [취약] /etc/hosts.deny 파일이 존재하지 않습니다." >> $result 2>&1  echo " ==> [취약] /etc/hosts.deny 파일이 존재하지 않습니다." >> 17.txt 2>&1  fi |

**19.1 동작 로직**

|  |
| --- |
| “/etc/hosts.deny” 설정 파일의 존재 여부와 외부 접속 허용 IP, PORT 허용 설정을 판별하고 확인한다.  해당 파일의 존재 여부, 설정 컨피그 상의 차단 여부를 조건으로 판단한다. |

**20. finger service 확인.**

|  |
| --- |
| if [ -f /etc/xinetd.d/finger ]; then  if grep "yes" /etc/xinetd.d/finger > /dev/null 2>&1 ; then  echo " ==> [안전] finger service가 꺼져있습니다." >> $result 2>&1  else  echo " ==> [취약] finger service가 켜져있습니다." >> $result 2>&1  echo " ==> [취약] finger service가 켜져있습니다." >> 18.txt 2>&1  fi  else  echo " ==> [안전] finger service가 꺼져있습니다." >> $result 2>&1 fi |

**20.1 동작 로직**

|  |
| --- |
| FINGER 서비스가 활성화되어있는지 config를 통해 판별하고 확인한다.  설정파일의 존재 여부, 설정파일의 서비스 상태 정보를 조건으로 판별한다. |

**21. Anonymous FTP 활성화 여부 확인.**

|  |
| --- |
| if grep "ftp" /etc/passwd > /dev/null 2>&1 ; then  echo " ==> [취약] Anonymous FTP 계정이 존재합니다." >> $result 2>&1  else  echo " ==> [안전] Anonymous FTP 계정이 존재하지 않습니다." >> $result 2>&1  fi |

**21.1 동작 로직**

|  |
| --- |
| “/etc/passwd”의 계정 정보 설정을 확인하여 일반 FTP 계정으로 접속하는지 판별하고 확인한다.  해당 파일의 ftp 계정 존재 여부를 조건으로 취약과 안전을 판별한다. |

**22. r 계열 서비스 활성화 여부 확인.**

|  |
| --- |
| if [ -f /etc/inetd.conf ]; then  if ls -alL /etc/xinetd.d/\* | egrep "rsa|rlogin|rexec" | egrep -v "grep|klogin|kshell|kexec" > /dev/null 2>&1 ; then  echo " ==> [취약] r 계열 서비스가 활성화 되어 있습니다." >> $result 2>&1  else을  echo " ==> [안전] r 계열 서비스가 활성화 되어 있지 않습니다." >> $result 2>&1  fi  else  echo " ==> [안전] r 계열 서비스가 활성화 되어 있지 않습니다." >> $result 2>&1  fi |

**22.1 동작 로직**

|  |
| --- |
| “/etc/inetd.conf” 설정을 확인 후 불필요한 r 계열의 서비스가 활성화 되어 있는 것 판별하고 확인한다.  해당 파일의 존재여부, 서비스의 파일 존재 여부를 조건으로 안전과 취약을 판별한다. |

**23. 데이터 결과 파일 저장 형식 결정.**

|  |
| --- |
| grep -w "\[취약\]" $result | awk -F"==>" '{print $2}' > error\_result.txt  count=`cat error\_result.txt | wc -l`  rm -f error\_result.txt  for i in $list  do  main=`cat $i.txt | grep "$i" | awk -F"." '{print $2}' | sed s/^/\[/g | sed s/$/\]/g`  sul=`cat $i.txt | grep -w "\[취약\]" | awk -F"==>" '{print $2}'`  cc=`cat $i.txt | wc -l`  if [ $cc -le 2 ]; then  echo > /dev/null 2>&1  rm -f $i.txt  else  echo "${YEL}$main\n${RED}$sul${NC}" >> $result 2>&1  echo "-------------------------------------" >> $result 2>&1  count=`cat $i.txt | grep -w "\[취약\]" | awk -F"==>" '{print $2}' | wc -l`  ca=`expr $ca + $count`  rm -f $i.txt  fi |

**23.1 동작 로직**

|  |
| --- |
| 위의 스크립트가 동작함에 따라 산출된 결과물 파일의 내용의 취약한 로그만 따로 분류한 뒤  실제 사용자가 쉽게 이해 할 수 있는 로그의 형식으로 바꿔준다. |

**24. 데이터 결과 Elrasticsearch 용 csv 파일 저장 형식 결정.**

|  |
| --- |
| # elrasticsearch  DATE\_NOW=$(date +"%Y-%m-%d-%H-%M-%S")  VUL\_ITEM=$1 # U-01/U-02  MESSAGES=$3 # error message  HOSTNAME=$(hostname)  OS\_NAME=$(sed -ne "{ /^PRETTY\_NAME[ \t]\*=/ p; }" /etc/\*release\* | awk -F= '{ print $2 }' | sed s/\"//g)  IP\_ADDR=$(hostname -I | sed s/\ /,/g | sed s/,$//)  elra=`date +%F`-elra.csv  …  … (중략)  sul2=`cat $i.txt | grep -w "\[취약\]" | awk -F"==>" 'BEGIN{ORS=" "}{print $2}'`  echo "$DATE\_NOW,$IP\_ADDR,$OS\_NAME,$main,$sul2" >> $elra 2>&1  sleep 3  scp $elra elk:/home/user/won/ |

**24.1 동작 로직**

|  |
| --- |
| 스크립트가 동작함에 따라 산출된 결과물 파일의 내용의 취약한 로그만 따로 분류한 뒤  Elrasticsearch, logstash Config 파일에 선언해놓은 설정의 형식 대로 로그 파일을 새롭게 만든다.  스크립트 동작이 모두 완료되면 약 3초간의 유예기간을 갖고 자동으로 ELK 서버의 경로로 복사한다. |

# 4. 결론

|  |
| --- |
| UNIX 서버 취약점 분석-평가 항목의 "계정관리", "파일 및 디렉터리 관리". "서비스관리"  항목들에 해당하여 분석을 자동화하여 산출물을 추출 할 수 있다.  추가적으로 ELK를 구축하여 사용자가 데이터를 주기별로 확인 할 수 있고, 기존 보다 보기 쉽게 시각화 하였다. |

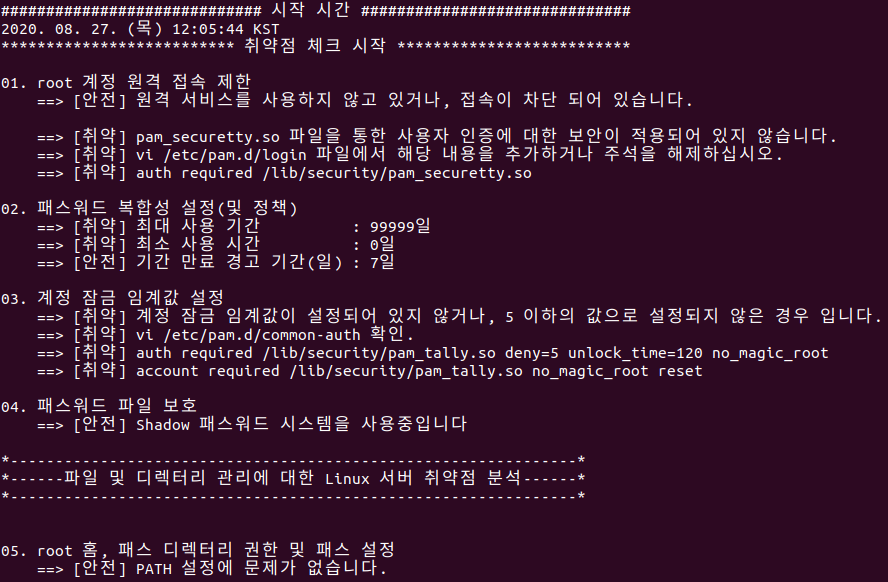
**4.1 산출물**

* **스크립트를 실행하게 되면 아래와 같은 두 종류(txt, csv)의 파일을 남긴다.**

|  |
| --- |
| **2020-08-28-elra.csv** : ELK 서버에서 시각화 할 csv 형식의 데이터 파일  **cccr\_result\_2020-08-20\_\_09:32:04.txt** : 스크립트 실 사용자가 바로 확인 할 수 있는 결과 데이터 파일 |

**4.2 산출물 결과**

* **cccr\_result\_2020-08-20\_\_09:32:04.txt : 사용자용 파일의 일부분만 캡쳐한 이미지.**



* **2020-08-28-elra.csv : Elrasticsearch 전용 csv 파일의 일부분만 캡쳐한 이미지.**

