





选题背景与意义



Euler Guardian

Euler Guardian 是为openEuler community开发,面向所有 Linux操作系统的风险评估系统。

Euler Guardian能够基于本地扫描,发现操作系统中安全配置、软件依赖、访问控制等风险,并对风险分级,生成报告。

Euler Guardian能够在Linux服务器受到入侵时,提供快速自动化的应急响应。

01 操作系统安全是信息系统安全的基础

操作系统安全在计算机信息系统的整体安全性中具有 至关重要的作用,没有操作系统提供的安全性,计算 机业务系统的安全性是没有基础的。 02 为什么需要Euler Guardian

通过某种手段监控操作系统及其组件的安全性,是每一个开发者、用户都关心的问题。 我们需要基于 upstream社区报告或主流安全漏洞数据库、开发一款 可扫描openEuler每日构建版本中存在的安全漏洞并 通过合适途径向开发者及用户告警的安全辅助工具。



Euler Guardian 4 modules



风险评估模块

对扫描结果进行风险分级



本地扫描模块

完成安全策略检查、软件依赖CVE检 查等本地扫描



可视化模块

提供CLI 界面, 生成HTML报告



应急响应模块

提供Linux服务器受入侵后的快速自动 化应急响应 能力



应急响应模块



7.可登陆用户

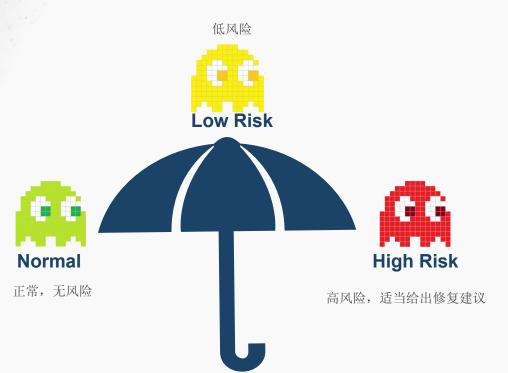
8. 所有用户的上次登录情况

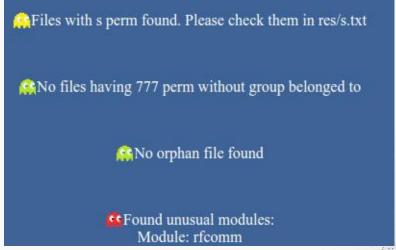
3.检查是否有ssh的

4.root用户口令爆破

风险评估模块





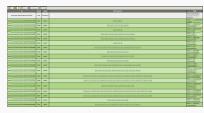


可视化模块









CLI

main report

SSG

OVAL

命令行界面

报告首页

secure configuration报告页面

软件依赖CVE检查页面



不依赖包管理器进行CVE检查

使用包管理器的问题

Linux发行版众多,使用不同的包管理器,例如yum, apt, snap等等。如果软件包版本漏洞匹配基于软件包管理器来做,将会非常繁琐。

如何利用OVAL进行检查

OVAL能够清晰地对与安全相关的检查 点作出描述,且机器可读的,能够直 接应用到自动化的安全扫描中。 根据操作系统的名称和版本,使用对 应的基线库,即可利用oscap进行扫描。









软件依赖的CVE检查

检查软件依赖的CVE,基础的想法是通过包管理器列出软件包版本然后做CPE匹配检查。

例如,使用apt时,可以用apt list -installed来列出已经安装的软件包及其 版本, 通过软件包版本检索其对应的 CVE和是否patch。

OVAL是什么

OVAL,Open Vulnerability Assessment Language,即开放式漏洞评估语言。OVAL由MITRE公司开发,是一种用来定义检查项、脆弱点等技术细节的一种描述语言。OVAL同样使用标准的XML格式组织其内容。

不依赖包管理器进行CVE检查

检查、安装OSCAP

```
function OVALChk() {
       echo -e "\e[1;34mThis device uses apt.\n\033[0m" 2>/dev/null
       if [ "$(oscap -h 2>/dev/null)" ]; then
           echo -e "\e[1;34mNo oscap. Downloading...\n\033[0m" 2>/dev/null
           sudo apt-get install libopenscap8
   elif [ "$(yum --version 2>/dev/null)" ]; then
       echo -e "\e[1;34mThis device uses yum.\n\033[0m" 2>/dev/null
            echo -e "\e[1;34mNo oscap. Downloading...\n\033[0m" 2>/dev/null
           sudo yum install openscap-utils -y
```

确定需要使用的基线库

```
tmpStr=`cat /etc/*-release | grep ^NAME=`
read -ra tmpArr <<<"$tmpStr"
tmpStr1=${tmpArr[1]}
releaseNameStr=${tmpStr1,,}
tmpStr=`cat /etc/*-release | grep VERSION ID 2>/dev/null`
read -ra tmpArr <<<"$tmpStr"
releaseVersionID=${tmpArr[1]}
releaseVersionIDStr=`echo ${releaseVersionID//./}`
targetSSGFile="ssg-${releaseNameStr}${releaseVersionIDStr}-ds.xml"
targetOVALFile="${releaseNameStr}${releaseVersionIDStr}.oval.xml"
```

使用对应基线库或通用基线库进行检查

-

```
hasSSGFile=`ls ssg | grep ${targetSSGFile} 2>/dev/null`
   echo -e "\e[1;34mSSG file found:\e[00m\n$targetSSGFile\033[0m"
   oscap xccdf eval --profile xccdf org.ssgproject.content profile standard --results ./report/sec conf ${timeStamp}.xml --report ./report/sec con
    echo -e "\e[1;34mNo SSG file found. Use centos7.\033[0m"
    oscap xccdf eval --profile xccdf org.ssgproject.content profile standard --results ./report/sec conf ${timeStamp}.xml --report ./report/sec con
fi
hasOVALFile=`ls ssg | grep ${targetOVALFile} 2>/dev/null`
   echo -e "\e[1;34mOVAL file found:\e[00m\n$targetOVALFile\033[0m"
   oscap oval eval --results ./report/comp vuln ${timeStamp}.xml --report ./report/comp vuln ${timeStamp}.html ./ssg/${targetOVALFile}
    targetOVALFile="${releaseNameStr}.oval.xml"
   hasOVALFile=`ls ssg | grep ${targetOVALFile} 2>/dev/null`
    if [ "$hasOVALFile" ]; then
       echo -e "\e[1;34mOVAL file found:\e[00m\n$targetOVALFile\033[0m"
        oscap oval eval --results ./report/comp vuln ${timeStamp}.xml --report ./report/comp vuln ${timeStamp}.html ./ssg/${targetOVALFile}
        echo -e "\e[1;34mNo OVAL file found. Use rhel7.\033[0m"
        oscap oval eval --results ./report/comp vuln ${timeStamp}.xml --report ./report/comp vuln ${timeStamp}.html ./ssg/rhel7.oval.xml
```

隐藏进程检查及排序

diff文件

一般在运维时, 会将两个结果输出 到文件,然后diff文件差异

利用shell完成比较和排序

hiddenPID=`echo \${psPIDList[@]} \${procPIDList[@]} | tr ' ' '\n' | sort -n | uniq -u`



隐藏进程检查的基本思路是: 比较 ps输出的PID和proc/下的PID差异。

考虑到应用场景,希望减少文件操作。 使用数组的数据结构,比较数组长度以判断是否有隐藏进程。 利用shell本身的特性,构建数组。

for eachPID in \$psPID; do psPIDList[\${#psPIDList[*]}]=\$eachPID

隐藏进程检查及排序

```
echo -e "\n\e[1;34mChecking hidden processes.\033[0m"
psPIDList=()
psPID=`ps -ef 2>/dev/null | awk 'NR>1{print $2}'`
for eachPID in $psPID; do
    psPIDList[${#psPIDList[*]}]=$eachPID
procPIDList=()
procPID=`ls /proc/ | grep ^[0-9]`
for eachPID in $procPID; do
    procPIDList[${#procPIDList[*]}]=$eachPID
    echo -e "\e[1;32mNormal. No hidden process found.\033[0m"
    hiddenPID=`echo ${psPIDList[@]} ${procPIDList[@]} | tr ' ' '\n' | sort -n | uniq -u`
    for eachPID in $hiddenPID; do
        echo -e "\e[1;31mHigh risk. Found hidden process, PID: $eachPID\033[0m"
```

权衡实用性和交互性



第一版

高交互,需要用户 老手不需要,新手 自定义安全策略 不会用

考虑使用场景

- 1. 制定较为详细的安全策略
- 2.使用参数进行交互

第二版

安全策略详细, 自 动化扫描

未来

- 1.支持参数定义扫 描范围 2.支持用户自定义
- 安全策略,以 profile的形式存在



项目总结

项目完成质量

本项目基本满足了使用场景中的使用需求, 兼容性较好,可在多种Linux操作系统中运行。 具有较为完善的安全策略,能够自动化地完 成对操作系统的安全性分析,并CLI输出风 险分级信息或生成较为完善的HTML报告。

开源项目的项目管理

项目管理包括策划、进度计划和维护组成项目的活动的进展。

本项目是我第一次作为maintainer的 开源项目,也使我第一次接触项目管 理,我认识到了项目管理的重要性。



项目不足

由于时间缘故,仍有部分功能需要加以完善(见方案进度部分),在比赛结束后,将长期维护这个项目,对这些部分进行完善,对各位同仁在使用中发现的问题进行修复。

对操作系统安全的思考

"安全"是一个相对的概念,没有绝对的安全。操作系统安全性具有短板效应,只能追求在攻击者之前发现漏洞,提高系统的攻击难度,增加安全性。

本项目主要参考等保2.0《GBT25070-2019信息 安全技术网络安全等级保护安全设计技术要求》 和Minimum Security Requirements for Multi-User Operating Systems两份文件

