Euler Guardian

openEuler操作系统风险评估系统

汇报人：Anaïs Huang

目录》

CONTENTS

01
PART

选题背景与意义

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。

02
PART

项目架构设计

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

本地扫描模块


local scan
本地扫描模块
1
PreOp 预操作
1. 检查current id, 判断是否有root权限
2.检查SetUID
3.删除之前留下的s.txt
2
System information check
系统信息检查
2.1
SysInfoChk系统信息检查
检查系统基本信息
2.2
SecCheck安全策略检查
1. 检查SELinux是否开启
2. 检查资源的限制情况
2.3
AuditChk
操作系统安全审计
3
User information and access control check
用户信息和访问控制检查
3.1
UserInfoChk用户信息检查
1. 检查hostname, id
2. 检查口令是否以hash存储
3. 检查上一次登录用户
3.2
UserIdenChk用户身份检查
1. 口令配置的时效与复杂度
2. 空口令用户检查
4
FileChek文件检查
1. 查找系统中所有含s权限的文件
2.查找无属组的777权限文件
3.查找孤儿文件
4.查找不常见但载入内核的模块
5
OVALChk OVAL检查
1. 检查secure configuration
2. 检查软件依赖CVE

应急响应模块

基本检查

1.防火墙规则
2.开放端口
3.init.d services
4.$PATH

文件检查

1.加载到内核的不常见module
2.被打开的已删除文件(除浏览器)
3.7天内改变过的文件

进程分析

1.检查进程使用CPU是否过高
2.检查隐藏进程，并按升序排列

后门检查

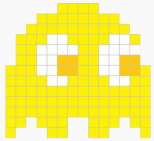
1.root的crontab files检查和cron后门检查
2.webshell检查(asp, jsp, php)

用户和log检查

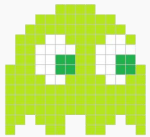
1.检查history中wget
2.检查history中ssh
3.检查是否有ssh的
4.root用户口令爆破
5.检查有root权限的用户是否为root
6.检查空口令用户
7.可登陆用户
8.所有用户的上次登录情况

风险评估模块

低风险

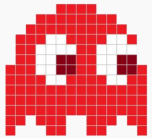


Low Risk



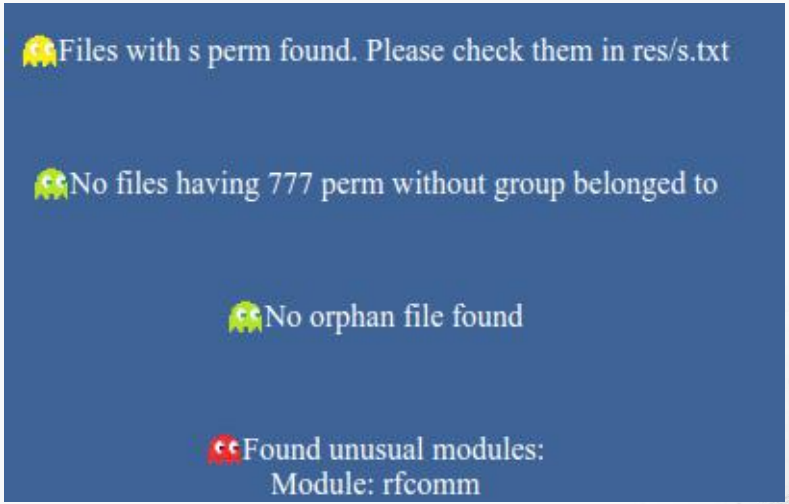
Normal

正常，无风险



High Risk

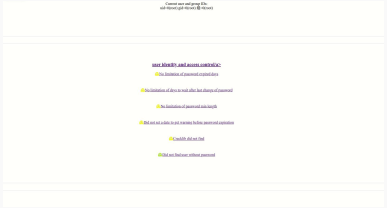
高风险，适当给出修复建议



可视化模块

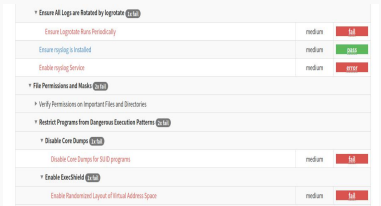
CLI

命令行界面



main report

报告首页



SSG

secure configuration报告页面

OVAL

软件依赖CVE检查页面

03
PART

关键技术问题

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() {
    # install oscap
    if [ "$(apt -v 2>/dev/null)" ]; then
        # use apt
        echo -e "\e[1;34mThis device uses apt.\n\033[0m" 2>/dev/null
        # if oscap is installed
        if [ "$(oscap -h 2>/dev/null)" ]; then
            :
        else
            echo -e "\e[1;34mNo oscap. Downloading...\n\033[0m" 2>/dev/null
            sudo apt-get install libopenscap8
        fi
    elif [ "$(yum --version 2>/dev/null)" ]; then
        # use yum
        echo -e "\e[1;34mThis device uses yum.\n\033[0m" 2>/dev/null

        # if oscap is installed
        if [ "$(oscap -h 2>/dev/null)" ]; then
            :
        else
            echo -e "\e[1;34mNo oscap. Downloading...\n\033[0m" 2>/dev/null
            sudo yum install openscap-utils -y
        fi
    fi
```

确定需要使用的基线库

```
# check OVAL file
# release id
tmpStr=`cat /etc/*-release | grep ^NAME=`
IFS='"'
read -ra tmpArr <<<"$tmpStr"
tmpStr1=${tmpArr[1]}
releaseNameStr=${tmpStr1,,}
#echo "$releaseNameStr"

# release version id
tmpStr=`cat /etc/*-release | grep VERSION_ID 2>/dev/null`
IFS='"'
read -ra tmpArr <<<"$tmpStr"
releaseVersionID=${tmpArr[1]}
releaseVersionIDStr=`echo ${releaseVersionID//./}`
#echo "$releaseVersionIDStr"

#default ssg:centos7 oval:redhat7
targetSSGFile="ssg-${releaseNameStr}${releaseVersionIDStr}-ds.xml"
targetOVALFile="${releaseNameStr}${releaseVersionIDStr}.oval.xml"
```

不依赖包管理器进行CVE检查

使用对应基线库或通用基线库进行检查

```
hasSSGFile=`ls ssg | grep ${targetSSGFile} 2>/dev/null`
if [ "$hasSSGFile" ]; then
    # ssg check
    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
else
    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 ${targetOVALFile} 2>/dev/null`
if [ "$hasOVALFile" ]; then
    #echo "has oval file"
    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}
else
    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}
    else
        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
    fi
fi
```

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

```
# compare differences of 2 lists and sort
hiddenPID=`echo ${psPIDList[@]} ${procPIDList[@]} | tr ' ' '\n' | sort -n | uniq -u`
```

隐藏进程检查基本思路

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。

使用数组剪枝

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
done
```

隐藏进程检查及排序

```
function HiddenProc() {
    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
    done
    #echo "psPID num: ${#psPIDList[*]}"

    procPIDList=()
    procPID=`ls /proc/ | grep ^[0-9]`
    for eachPID in $procPID; do
        procPIDList[${#procPIDList[*]}]=$eachPID
    done
    #echo "procPID num: ${#procPIDList[*]}"

    if [[ ${#psPIDList[*]} -eq ${#procPIDList[*]} ]]; then
        echo -e "\e[1;32mNormal. No hidden process found.\033[0m"
    else
        # compare differences of 2 lists and sort
        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"
        done
    fi
}
```

权衡实用性和交互性

第一版

高交互，需要用户自定义安全策略

缺点

老手不需要，新手不会用

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

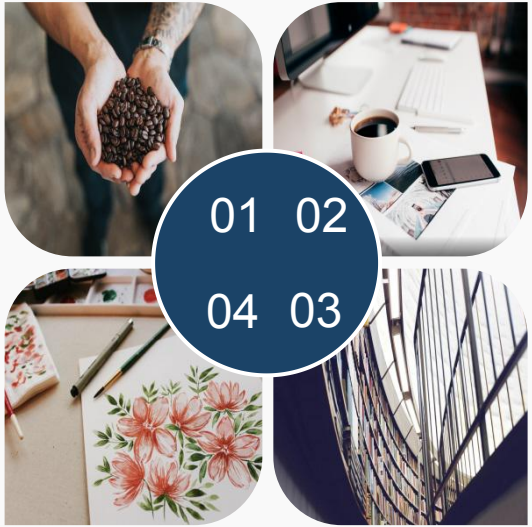
04
PART

项目总结

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。



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。

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。

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

项目展望

1.威胁情报共享

（1）upstream漏洞信息同步，在原有基线库的基础上，同步新出现的漏洞信息，进行检查

（2）向运维人员以邮件等形式告警系统上漏洞（场景：运维人员将本工具放在服务器上，定期自动化检测）

2.测试组件的插件化

很有“低耦合，高内聚”的想法，我将会逐步改进，并把针对特定常见漏洞patch的PoC或EXP加入测试组件，将风险分级细化

3.用户自定义模板

支持用户自定义模板，定义测试项

4.多国标准

添加多国安全性标准作为风险检测项相应标准

5.网络扫描

添加网络扫描模块

THANKS