

基于 ClamAV 和 RF 的安卓恶意代码检测器

课程作业汇报

第九组：冯驰，姚思悦，何梓欣，韩雨虹
指导老师：刘功申

2022 年 4 月 7 日



上海交通大学

SHANGHAI JIAO TONG UNIVERSITY

目录

- 1 运行环境
- 2 病毒检测
- 3 ClamAV 的修改及应用
- 4 ClamTK 的修改及应用
- 5 总结与演示



目录

- 1 运行环境
- 2 病毒检测
- 3 ClamAV 的修改及应用
- 4 ClamTK 的修改及应用
- 5 总结与演示



系统运行环境

- 操作系统：openEuler20.03-LTS-SP3
- 桌面环境：UKUI

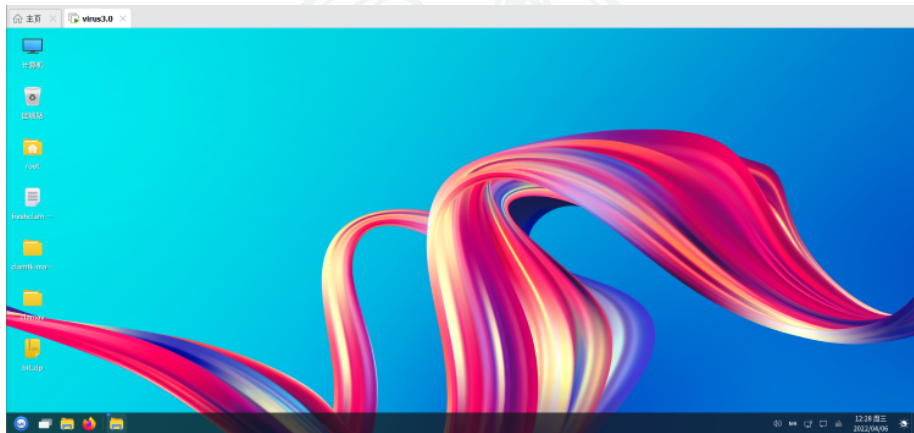


Figure: 操作系统桌面

目录

- 1 运行环境
- 2 病毒检测
- 3 ClamAV 的修改及应用
- 4 ClamTK 的修改及应用
- 5 总结与演示



检测方法概述

• 检测原理：

- 将安卓软件反编译，提取字节编码
- 将字节码分为 MRGITPV 七类，并进行分类映射
- 使用滑动平均窗口提取 3-gram 特征
- 使用随机森林训练分类器

• 检测框架：



Figure: 检测框架

数据集

- 恶意软件：来自 virusShare, 729 个, 共约 1.57G
<https://virusshare.com>
- 良性软件：来自 UNB2015 和 2017 (加拿大网络安全研究所), 686 个, 共约 6.75G
<https://www.unb.ca/cic/datasets>

模块实现——反汇编

- batch_disassemble.py: 将安卓 apk 从指定目录反汇编出对应的字节编码

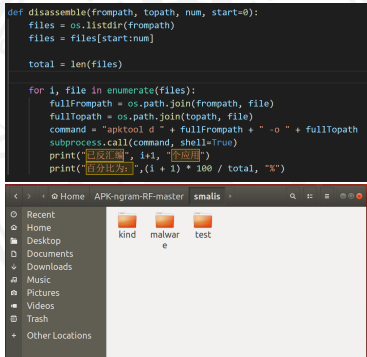


Figure: 存放反汇编文件至指定目录

模块实现——字节编码映射

- `bytecode_extract.py`: 将字节编码取出并映射, 保存映射后的文件

```
def collect(rootdir, isMalware):
    wares = os.listdir(rootdir)
    total = len(wares)
    for i, ware in enumerate(wares):
        warePath = os.path.join(rootdir, ware)
        ware = Ware(warePath, isMalware)
        ware.extractFeature(f)
        print("已提取", i + 1, "个文件的特征")
        print("百分比为:", (i + 1) * 100 / total, "%")

# 1代表良性软件    0代表恶意软件    2代表测试软件
```

Figure: 映射函数

模块实现——3-gram 特征提取

- `n_gram.py`: 将映射后的编码滑动平均提取 `3_gram` 特征

```
def gram():
    n = int(sys.argv[2])
    print("n = ", n)
    origin = pd.read_csv("data.csv")
    mdict = MyDict()
    feature = origin["Feature"].str.split("|")
    total = len(feature)
    for i, code in enumerate(feature):
        mdict.newLayer()
        if not type(code) == list:
            continue
        for method in code:
            length = len(method)
            if length < n:
                continue
            for start in range(length - (n - 1)):
                end = start + n
                mdict.mark(method[start:end])
    print("已完成", i+1, "个应用")
    print("百分比为:", (i + 1) * 100 / total, "%")

result = mdict.dict
result['isMalware']=origin.isMalware
pd.DataFrame(result, index=origin.index)\
    .to_csv("./" + str(n) + " gram.csv", index=False)
```

Figure: 提取 3-gram 特征

模块实现——随机森林分类器

- 1461 个样本（白样本 686，黑样本 729），验证集准确率为 94.17%。

```

Terminal: Local < + v
PS E:\virus\pythonProject1\pythonProject11111\pythonProject1\AndroidMalware-ngram-RF> python RF.py
dataset: 1461
x_train: 892
x_dev: 223

-----
y_dev_pred: [0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 1 0 0 0 0 1 0 0 0 0 0 1
0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 1
0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 1
0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1
0]
y_dev: [0 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 1
0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0
0 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1
0 1 1 1 1 0 1 0 1 1 0 0 1 1 0 1 0 0 0 1 1 1 0 1 1 0 0 1 0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 1 0 0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 2
0]
dev_acc: 0.9417040358744395

-----
x_test_black: 21
y_test_pred_black: [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0]
-----
x_test_white: 25
y_test_pred_white: [0 1 1 1 1 1 1 1 0 0 1 1 0 1 1 1 0 1 1 0 0 0 0 0]
PS E:\virus\pythonProject1\pythonProject11111\pythonProject1\AndroidMalware-ngram-RF>

```

模块实现——随机森林检测器

- 保存训练好的模型
- 构建 python 脚本对每个 apk 文件处理后得到 3-gram 特征
- 加载分类器模型进行检测，可辨别是否为恶意代码软件

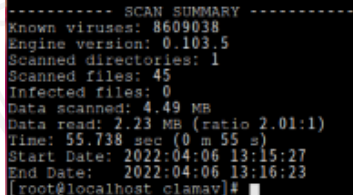
目录

- 1 运行环境
- 2 病毒检测
- 3 ClamAV 的修改及应用
- 4 ClamTK 的修改及应用
- 5 总结与演示



源码安装 ClamAV

- 编译安装 ClamAV
- 遇到的问题：
 - 选择 0.103.1 版本后，系统提示版本较低，最终选择 0.103.5 版本
 - 缺少较多依赖库，需要手动安装
 - 无法识别 freshclam 及 clamscan 指令，需要软连接或者通过路径调用

A terminal window showing the output of a ClamAV scan. The text is as follows:

```
----- SCAN SUMMARY -----  
Known viruses: 8609038  
Engine version: 0.103.5  
Scanned directories: 1  
Scanned files: 45  
Infected files: 0  
Data scanned: 4.49 MB  
Data read: 2.23 MB (ratio 2.01:1)  
Time: 55.738 sec (0 m 55 s)  
Start Date: 2022:04:06 13:15:27  
End Date: 2022:04:06 13:16:23  
[root@localhost clamav]#
```

Figure: 安装完 clamscan 扫描结果

定位 apk 检测

- ClamAV 原生没有 apk 检测的 type

ClamAV Documentation

ClamAV File Types are prefixed with `CL_TYPE_`. The following is an exhaustive list of all current file types.

CL_TYPE	Description
CL_TYPE_7Z	7-Zip Archive
CL_TYPE_7ZSFX	Self-Extracting 7-Zip Archive
CL_TYPE_APM	Disk Image - Apple Partition Map
CL_TYPE_ARJ	ARJ Archive
CL_TYPE_ARJSFX	Self-Extracting ARJ Archive
CL_TYPE_AUTOIT	AutoIt Automation Executable
CL_TYPE_BINARY_DATA	binary data
CL_TYPE_BINHEX	BinHex Macintosh 7-bit ASCII email attachment encoding
CL_TYPE_BZ	BZip Compressed File
CL_TYPE_CABSF	Self-Extracting Microsoft CAB Archive
CL_TYPE_CPIO_CRC	CPIO Archive (CRC)
CL_TYPE_CPIO_NEWC	CPIO Archive (NEWC)
CL_TYPE_CPIO_ODC	CPIO Archive (ODC)
CL_TYPE_CPIO_OLD	CPIO Archive (OLD, Little Endian or Big Endian)
CL_TYPE_CRYPTFF	Files encrypted by CryptFF malware
CL_TYPE_DMG	Apple DMG Archive
CL_TYPE_EGG	ESTSoft EGG Archive, new in 0.102
CL_TYPE_ELF	ELF Executable (Linux/Unix program or library)
CL_TYPE_GIF	GIF Graphics File, new in 0.103
CL_TYPE_GPT	Disk Image - GUID Partition Table
CL_TYPE_GRAPHICS	Other graphics files; BMP, JPEG2000
CL_TYPE_GZ	GZip Compressed File

定位 apk 检测

- 对 apk 的检测会将其碎片化，然后调用 CL_TYPE_MACHO_UNIBIN: 和 CL_TYPE_BINARY_DATA 类型的检测。

```

4621  */
4622  switch (type) {
4623      /* bytecode hooks triggered by a lsig must be a hook
4624       * called from one of the functions here */
4625      case CL_TYPE_TEXT_ASCII:
4626      case CL_TYPE_TEXT_UTF16BE:
4627      case CL_TYPE_TEXT_UTF16LE:
4628      case CL_TYPE_TEXT_UTF8:
4629          perf_nested_start(ctx, PERF_SCRIPT, PERF_SCAN);
4630          if ((DCONF_DOC & DCONF_CONF_SCRIPT) && dettype != CL_TYPE_HTML && (ret != CL_VIRUS || SCAN_ALLMATCHES) && SCAN_PARSE_HTML)
4631              ret = cli_scan_script(ctx);
4632          if (SCAN_PARSE_MAIL && (DCONF_MAIL & MAIL_CONF_MBOX) && ret != CL_VIRUS && (cli_recursion_stack_get_type(ctx, -1) == CL_TYPE_TEXT))
4633              ret = cli_scan_fmapi(ctx, CL_TYPE_MAIL, 0, NULL, AC_SCAN_VIR, NULL, NULL);
4634          }
4635          perf_nested_stop(ctx, PERF_SCRIPT, PERF_SCAN);
4636          break;
4637      /* Due to performance reasons all executables were first scanned
4638       * in raw mode. Now we will try to unpack them
4639       */
4640      case CL_TYPE_MSEXEC:
4641          perf_nested_start(ctx, PERF_PE, PERF_SCAN);
4642          if (SCAN_PARSE_PE && ctx->dconf->pe) {
4643              unsigned int corrupted_input = ctx->corrupted_input;
4644              ret = cli_scan_pe(ctx);
4645              ctx->corrupted_input = corrupted_input;
4646          }
4647          perf_nested_stop(ctx, PERF_PE, PERF_SCAN);
4648          break;
4649      case CL_TYPE_ELF:
4650          perf_nested_start(ctx, PERF_ELF, PERF_SCAN);
4651          ret = cli_unpack_elf(ctx);
4652          perf_nested_stop(ctx, PERF_ELF, PERF_SCAN);
4653          break;
4654      case CL_TYPE_MACHO:
4655      case CL_TYPE_MACHO_UNIBIN:
4656          perf_nested_start(ctx, PERF_MACHO, PERF_SCAN);
4657          ret = cli_unpack_macho(ctx);
4658          perf_nested_stop(ctx, PERF_MACHO, PERF_SCAN);
4659          break;
4660      case CL_TYPE_BINARY_DATA:
4661          ret = cli_scan_fmapi(ctx, CL_TYPE_OTHER, 0, NULL, AC_SCAN_VIR, NULL, NULL);
4662          break;

```


定位 apk 检测

- 向前定位，找到除了以上这些类型之外的文件分类位置。在此添加函数判断其是否为 apk 文件。

```

5366
5367     if (FSTAT(desc, &sb) == -1) {
5368         cli_errmsg("cl_scandesc_callback: Can't fstat descriptor %d\n", desc);
5369         status = CL_ESTAT;
5370         goto done;
5371     }
5372     if (sb.st_size <= 5) {
5373         cli_dbgmsg("cl_scandesc_callback: File too small (" STDu64 " bytes), ignoring\n", (uint64_t)sb.st_size);
5374         status = CL_CLEAN;
5375         goto done;
5376     }
5377     if ((engine->maxfilesize > 0) && ((uint64_t)sb.st_size > engine->maxfilesize)) {
5378         cli_dbgmsg("cl_scandesc_callback: File too large (" STDu64 " bytes), ignoring\n", (uint64_t)sb.st_size);
5379         if (scanoptions->heuristic & CL_SCAN_HEURISTIC_EXCEEDS_MAX) {
5380             if (engine->cb_virus_found)
5381                 engine->cb_virus_found(desc, "Heuristics.Limits.Exceeded.MaxFileSize", context);
5382             status = CL_VIRUS;
5383         } else {
5384             status = CL_CLEAN;
5385         }
5386         goto done;
5387     }
5388
5389     if (NULL != filename) {
5390         (void)cli_basename(filename, strlen(filename), &filename_base);
5391     }
5392
5393     if (NULL == (map = fmap(desc, 0, sb.st_size, filename_base))) {
5394         cli_errmsg("CRITICAL: fmap() failed\n");
5395         status = CL_EMEM;
5396         goto done;
5397     }

```

用 c 调用 python 检测

- 已经封装了随机森林检测病毒脚本。于是在 scanners.c 中用 c 调用，并读取脚本的返回值，用于下一步的判断。

```

136 int exec_process(char *cmd, char *result, int size)
137 {
138     size_t ret;
139     FILE *fp = NULL;
140     if(cmd == NULL)
141     {
142         return -1;
143     }
144     fp = popen(cmd, "r");
145     if(fp == NULL)
146     {
147         printf("exec_process: error exec cmd: %s\n", cmd);
148         return -1;
149     }
150     if(result != NULL && size > 0)
151     {
152         memset(result, 0, size);
153         ret = fread(result, 1, size - 1, fp);
154         if(ferror(fp))
155         {
156             printf("exec_process: error\n");
157             pclose(fp);
158             return -1;
159         }
160         if(ret == 0) {
161             pclose(fp);
162             return -1;
163         }
164     }
165     pclose(fp);
166     return 0;
167 }
168
169 int detect(const char *file_name, const char *file_namebase){
170     char hhhhh[1000] = "python3 /home/group9/test.py ";
171     strcat(hhhhh, file_name);
172     strcat(hhhhh, " ");
173     strcat(hhhhh, file_namebase);
174     char popenret[100000] = {0}; /*执行脚本后的返回值*/
175     memset(popenret, 0, sizeof(popenret));
176     if(exec_process(hhhhh, popenret, sizeof(popenret)) != 0) {

```

调用函数改写

- 检测结果是以 status 存储的, 类型为 CL_VIRUS, CL_CLEAN, 并会在 ClamAV 终端输出返回值。

```
5398     int temp = 0;
5399     if (is_apk(filename)){
5400         temp = detect(filename, filename_base);
5401         if (temp == 1){
5402             status = CL_VIRUS;
5403             engine->cb_virus_found(desc, "Group9 detected virus", context);
5404         }
5405         else if (temp == 2){
5406             status = scan_common(map, filename, virname, scanned, engine, scanoptions, context);
5407         }
5408         else status = CL_CLEAN;
5409     }
5410     else{
5411         status = scan_common(map, filename, virname, scanned, engine, scanoptions, context);
5412     }
5413 }
```

修改后的 apk 文件检测结果

- 重新编译后运行 clamscan 已经可以达到我们想要的检测结果。

```
[root@localhost example]# /usr/local/etc/bin/clamscan
/home/example/virusshare127489hfueoal.apk: Group9 detected virus FOUND
/home/example/virusshare04738921473829.apk: Group9 detected virus FOUND
/home/example/virussharefheuwioyq8979032up.apk: Group9 detected virus FOUND
/home/example/python: Symbolic link
/home/example/air.com.apk: OK
/home/example/tattoo.my.photo.apk: OK

----- SCAN SUMMARY -----
Known viruses: 8609038
Engine version: 0.103.5
Scanned directories: 1
Scanned files: 5
Infected files: 3
Data scanned: 0.00 MB
Data read: 41.87 MB (ratio 0.00:1)
Time: 81.192 sec (1 m 21 s)
```

Figure: 可以检测出病毒文件

目录

- 
- 1 运行环境
 - 2 病毒检测
 - 3 ClamAV 的修改及应用
 - 4 ClamTK 的修改及应用
 - 5 总结与演示

ClamTK 安装与修改

- 通过源码安装 ClamTK
- 对 GUI.pm 进行修改
 - 将界面标题修改为 “ClamTK Virus Scanner of Group9”
 - 增加名为 “Group9” 的按钮，描述为 “Group9 clamscan”

```
$hb->set_title( _('ClamTk Virus Scanner of Group9') );
$hb->set_decoration_layout( 'menu,icon:minimize,close' );
$hb->set_show_close_button( TRUE );
```

```
sub about {
    my $dialog = Gtk3::AboutDialog->new;
    my $license =
        'ClamTk is free software; you can redistribute it and/or'
        . ' modify it under the terms of either:'
        . ' a) the GNU General Public License as published by the Free'
        . ' Software Foundation; either version 1, or (at your option)'
        . ' any later version, or'
        . ' b) the "Artistic license";'
    $dialog->set_wrap_license( TRUE );
    $dialog->set_position( 'mouse' );

    my $images_dir = ClamTk::App->get_path( 'images' );
    my $icon = "$images_dir/clam.png";
    my $pixbuf = Gtk3::Gdk::Pixbuf->new_from_file( $icon );

    { link'
        => _('Group9'),
        description => _('Group9 clamscan'),
        image
        => 'media-playback-start',
        button
        => FALSE,

    },
}
```

前端修改后界面展示

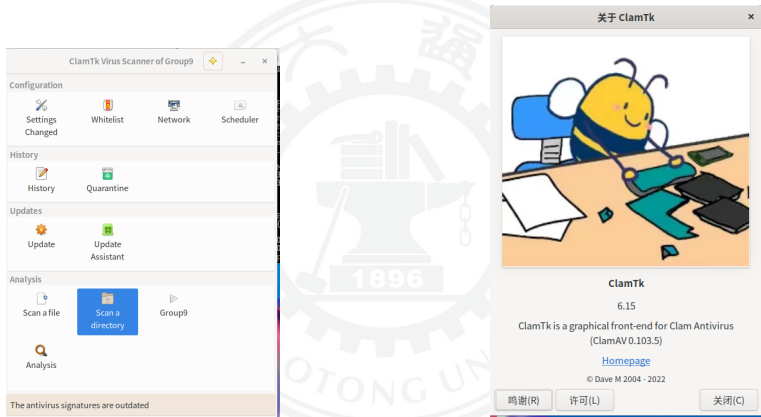


Figure: 修改后的界面

ClamTK 成功运行

- ClamTK 成功调用 ClamAV 扫描 apk 文件

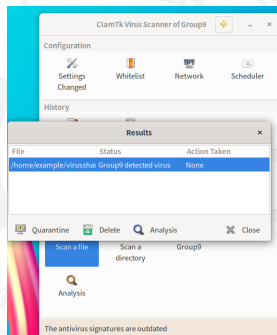


Figure: 扫描一个病毒文件并识别

目录

- 1 运行环境
- 2 病毒检测
- 3 ClamAV 的修改及应用
- 4 ClamTK 的修改及应用
- 5 总结与演示



项目成果与不足

一、 成果：

- 1、 本组在 openEuler 操作系统上，修改 ClamAV 源码，并使用随机森林模型检测 apk 格式的恶意代码文件，检测结果良好。

二、 不足之处与解决方法

- 1、 假阳性概率较高：目前只是用了随机森林来训练，改进可以使用层次结构更复杂的其他神经网络
- 2、 反编译时间较长，clamscan 调用过程时占用内存空间较大：目前使用了基于 java 的 apktool，内存占用高。改进可以使用 c 语言原生的 java 反编译库

Thanks!

谢谢!

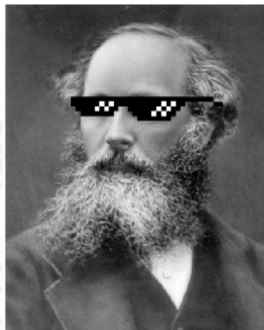
ありがとう!

¡Gracias!

Grazie!

Merci!

Mulțmesc!



分享人:

冯驰

F19 信息安全

