**漏洞原理**

shell里可以定义变量，POC中定义了一个命名为x的变量，内容是一个字符串：

() { :;}; echo vulnerable

而根据漏洞信息得知，这个漏洞产生于Shell在处理函数定义时，执行了函数体之后的命令。但这里x的值是个字符串，它是怎么转变成函数的呢?实际这个和Bash实现有关，在Bash中定义一个函数，格式为：

function function\_name() {

body;

}

当Bash在初始化环境变量时，语法解析器发现小括号和大括号的时候，就认为它是一个函数定义：

[lu4nx@lx-pc ~]$ say\_hello='() { echo hello world; }'

[lu4nx@lx-pc ~]$ export say\_hello

[lu4nx@lx-pc ~]$ bash -c 'say\_hello'

hello world

上面代码在新的Bash进程中，say\_hello成了新环境中的一个函数，它的演变过程如下：

1、新的bash在初始时，扫描到环境变量say\_hello出现小括号和大括号，认定它是一个函数定义

2、bash把say\_hello作为函数名，其值作为函数体

typeset命令可以列出当前环境中所有变量和函数定义，我们用typeset看看这个字符串怎么变成函数的。继续上面定义的say\_hello函数：

[lu4nx@lx-pc ~]$ bash -c 'typeset' | fgrep -A 10 say\_hello

say\_hello ()

{

echo hello world

}

这里新启动了个Bash进程，然后执行了typeset，typeset会返回当前环境（新的环境）中所有定义，这里清楚看到say\_hello被变成函数了。

**核心原因**：没有严格限制输入的边界，没有合法化的参数判断。

**解决方案**：安装补丁。在补丁中主要进行了参数的合法性过滤，补丁程序在/builtins/evalstring.c的parse\_and\_execute函数中进行了输入的command进行了合法性的边界检测。（CVE-2014-6271 的修补不够完善，导致CVE-2014-7169。）

**利用方式**

需要以下条件：

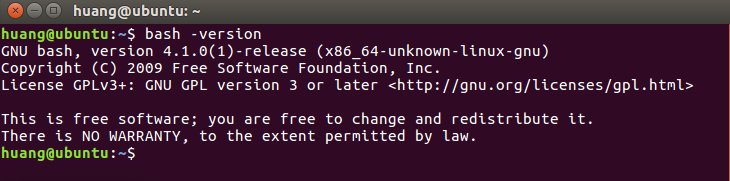
远程服务会调用bash。（创建bash子进程）

远程服务允许用户定义环境变量。

远程服务调用子bash时加载了用户定义的环境变量。

**漏洞复现**

查看bash版本 命令：bash -version



本地验证漏洞

