The Shellshock Attack (Part I)

Professor Travis Peters
CSCI 476 - Computer Security
Spring 2020

Some slides and figures adapted from Wenliang (Kevin) Du's
Computer & Internet Security: A Hands-on Approach (2nd Edition).
Thank you Kevin and all of the others that have contributed to the SEED resources!

Today

Announcements

- Lab 01 ➔ ***DUE BEFORE CLASS (@3PM) ON THURSDAY***
- ***REMINDER:*** Sign-up for Slack ASAP! *Some people still not signed up……..*
- ***REMINDER:*** RTS = READ THE SYLLABUS!!! (e.g., submitting labs)
- ***REMINDER:*** Post questions to *#labs* (or appropriate channel)
- ***Use the OFFICIAL SEED VM!***

  E.g., Upcoming lab:
  - *Apache webserver, special programs (e.g., vulnerable version of bash: bash_shellshock), etc.*
  - *We simply cannot not support other machines…*

Goals & Learning Objectives

- Tying up loose ends w/ environment variables & set-uid programs
- Understand *Shellshock* and related attacks

Recap: Thoughts from Set-UID Programs + Environment Variables

- Take a few minutes to write down everything you now know about Set-UID Programs & Env. Variables
- Share!
- Anything missing from your list that others shared?
- Questions?

Background: *Shell Functions*

Background: Shell Functions

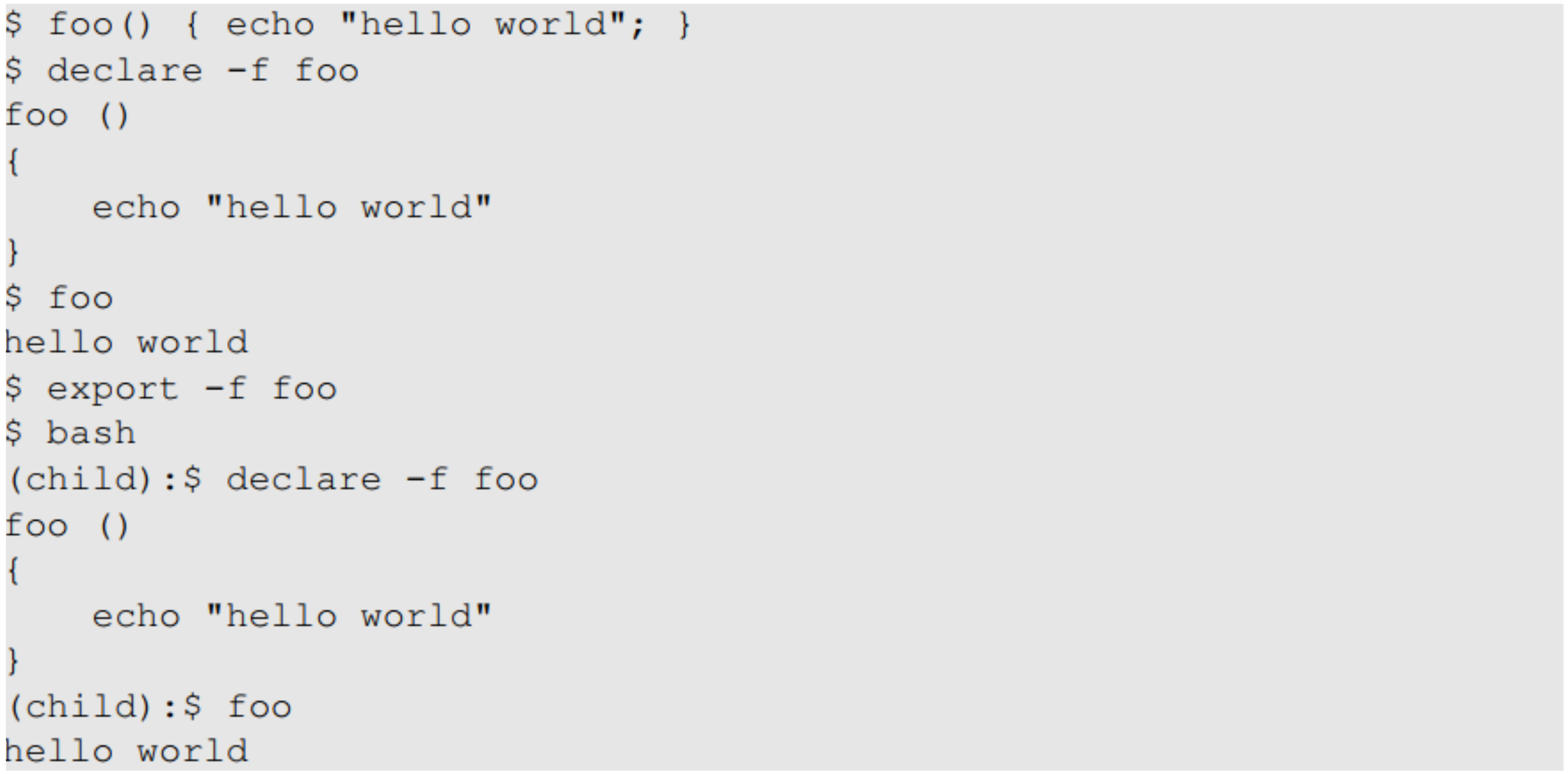
- A shell program is a command-line interpreter
  - Provides an interface between the user and OS
  - There are different types of shell: sh, bash, csh, zsh, Windows powershell, etc.
- The bash shell is one of the most popular shell programs; often used in the Linux OS
- The Shellshock vulnerability results from how **shell functions** and **environment variables** are handled in the bash shell

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

Passing Shell Functions to Child Processes

Approach 1: Define a function in the parent shell, export it, and then the child process will have it.

Example:



```
$ 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
```

Passing Shell Functions to Child Processes

Approach 2: Define a function as an env. variable; it becomes a function in the child process.

Example:

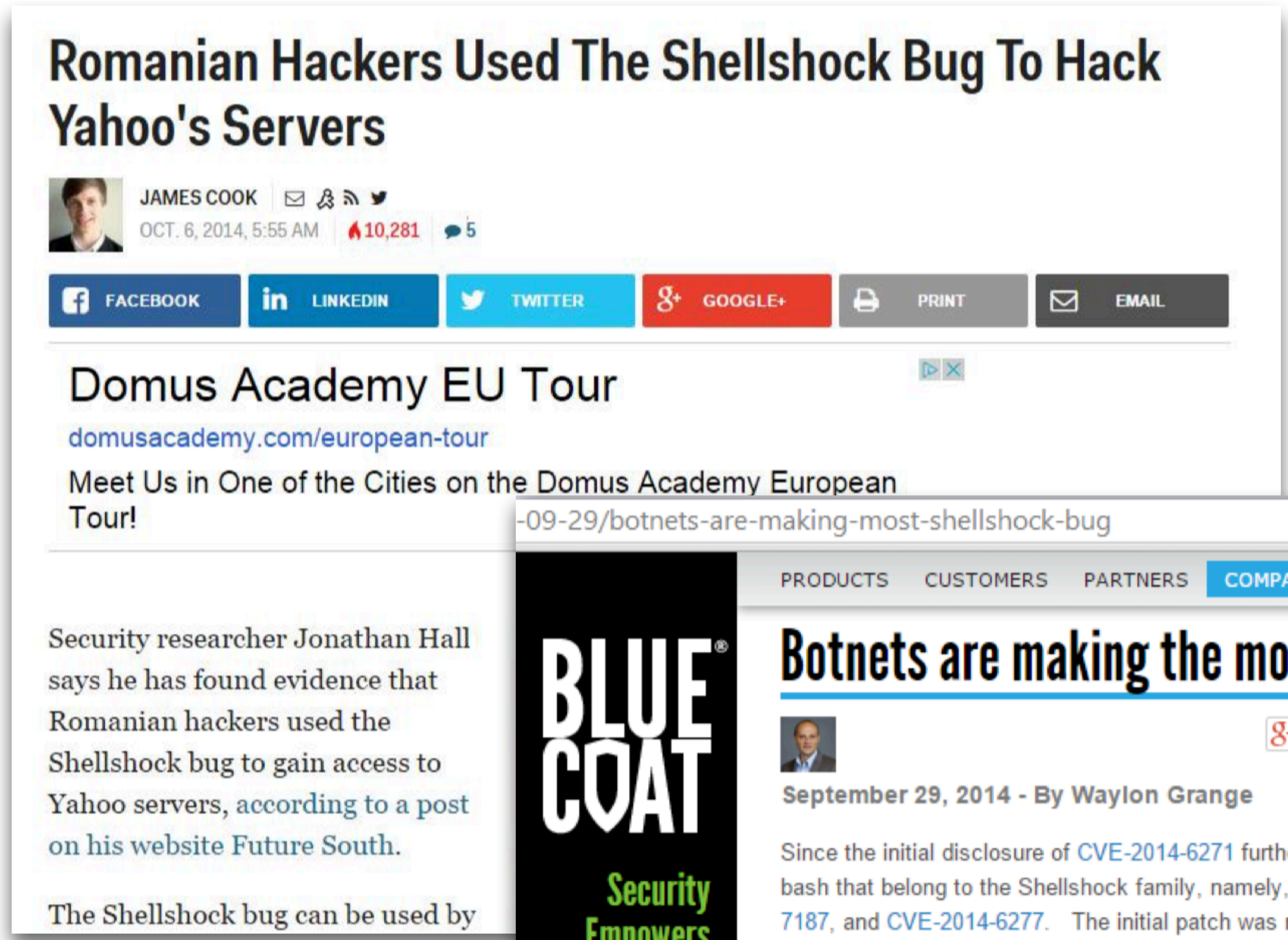
```
$ foo=’() { echo "hello world"; }’
$ echo $foo
() { echo "hello world"; }
$ declare -f foo
$ export foo
$ bash_shellshock    ← Run bash (vulnerable version) in the child
(child):$ echo $foo

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

Summary: Passing Shell Functions to Child Processes

- Both approaches are similar—they both use environment variables.
- In the 1st Approach…
  - When the ***parent shell*** creates a new process,
    it passes each exported function definition as an environment variable.
- In the 2nd Approach…
  - Same thing, but the ***parent does not need to be a shell*** process.
- In Both Approaches…
  - If the ***child process*** runs bash,
    the ***bash program will turn the environment variable back to a function definition.***

Takeaway: Any process that needs to pass a function definition to the child (bash) process can simply use environment variables.

The Shellshock Vulnerability



Romanian Hackers Used The Shellshock Bug To Hack Yahoo's Servers

JAMES COOK

OCT. 6, 2014, 5:55 AM 10,281 5

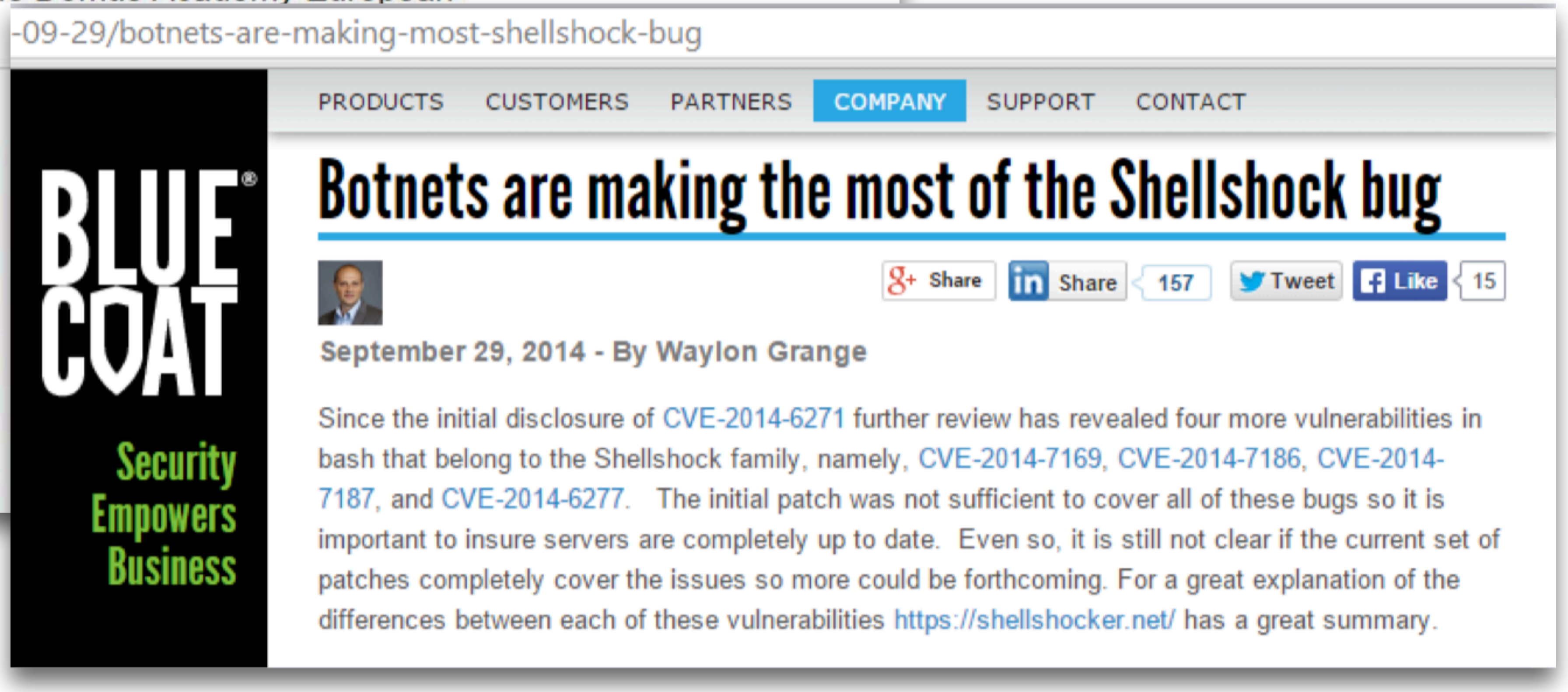
Domus Academy EU Tour

domusacademy.com/european-tour

Meet Us in One of the Cities on the Domus Academy European Tour!

Security researcher Jonathan Hall says he has found evidence that Romanian hackers used the Shellshock bug to gain access to Yahoo servers, according to a post on his website Future South.

The Shellshock bug can be used by



-09-29/botnets-are-making-most-shellshock-bug

Botnets are making the most of the Shellshock bug

September 29, 2014 - By Waylon Grange

Since the initial disclosure of CVE-2014-6271 further review has revealed four more vulnerabilities in bash that belong to the Shellshock family, namely, CVE-2014-7169, CVE-2014-7186, CVE-2014-7187, and CVE-2014-6277. The initial patch was not sufficient to cover all of these bugs so it is important to insure servers are completely up to date. Even so, it is still not clear if the current set of patches completely cover the issues so more could be forthcoming. For a great explanation of the differences between each of these vulnerabilities https://shellshocker.net/ has a great summary.

Very easy to find targets:

- Mass port scanning
- nmap shellshock script
- Metasploit module
- Online scanners

The Shellshock Vulnerability

- "Shellshock" or "bashbug" or "bashdoor" was publicly disclosed on September 24, 2014

  CVE-2014-6271

  https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271

- This vulnerability exploited a mistake made by bash when it converts environment variables to function definitions — ***effectively allows remote command execution*** via bash

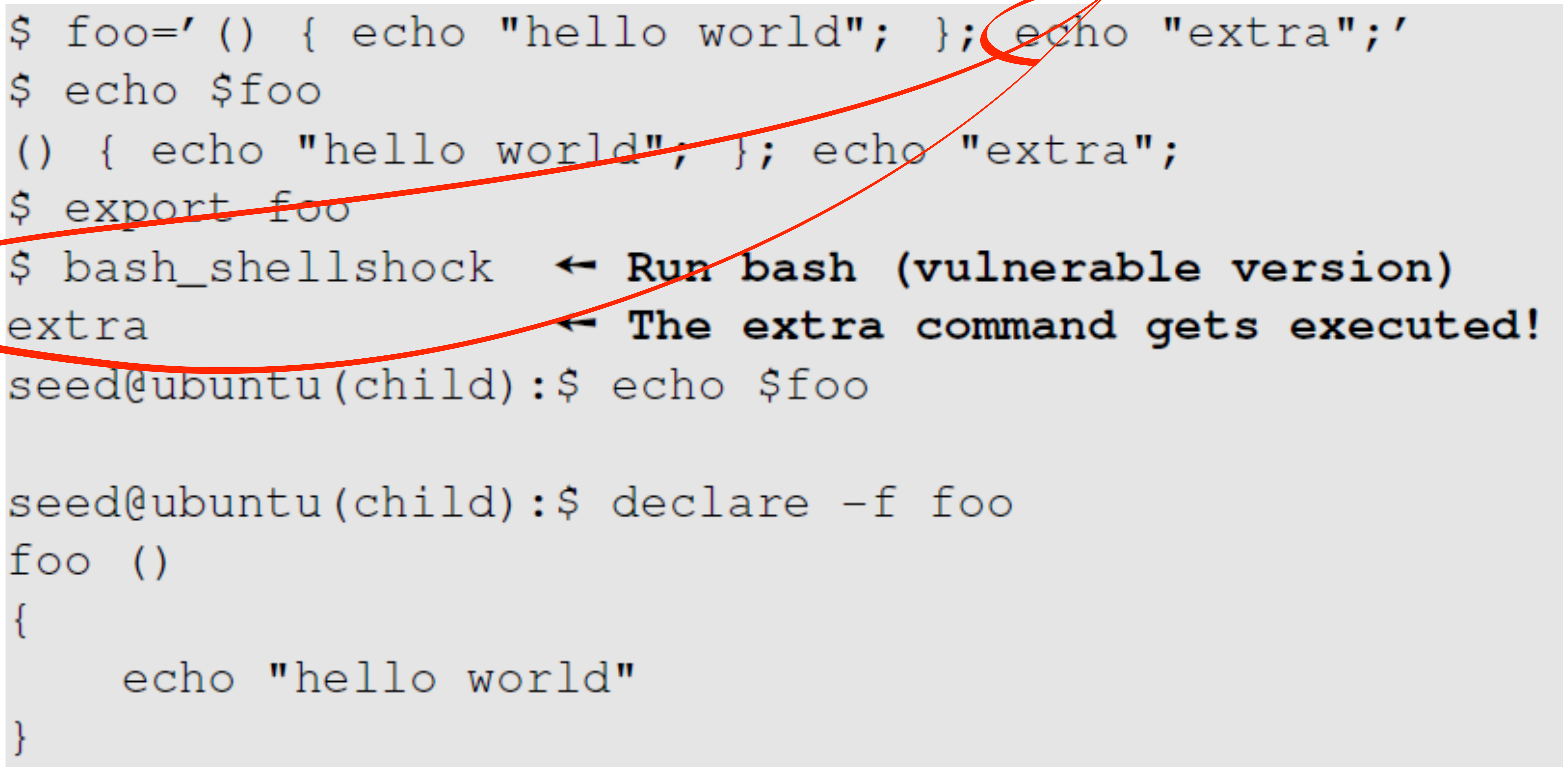
- The bug has existed in the bash source code since August 5th, 1989

  (SINCE BEFORE I WAS EVEN BORN!!!)

- After the official disclosure, several other bugs were found in the bash source code. Shellshock refers to the family of security bugs found in bash

The Shellshock Vulnerability

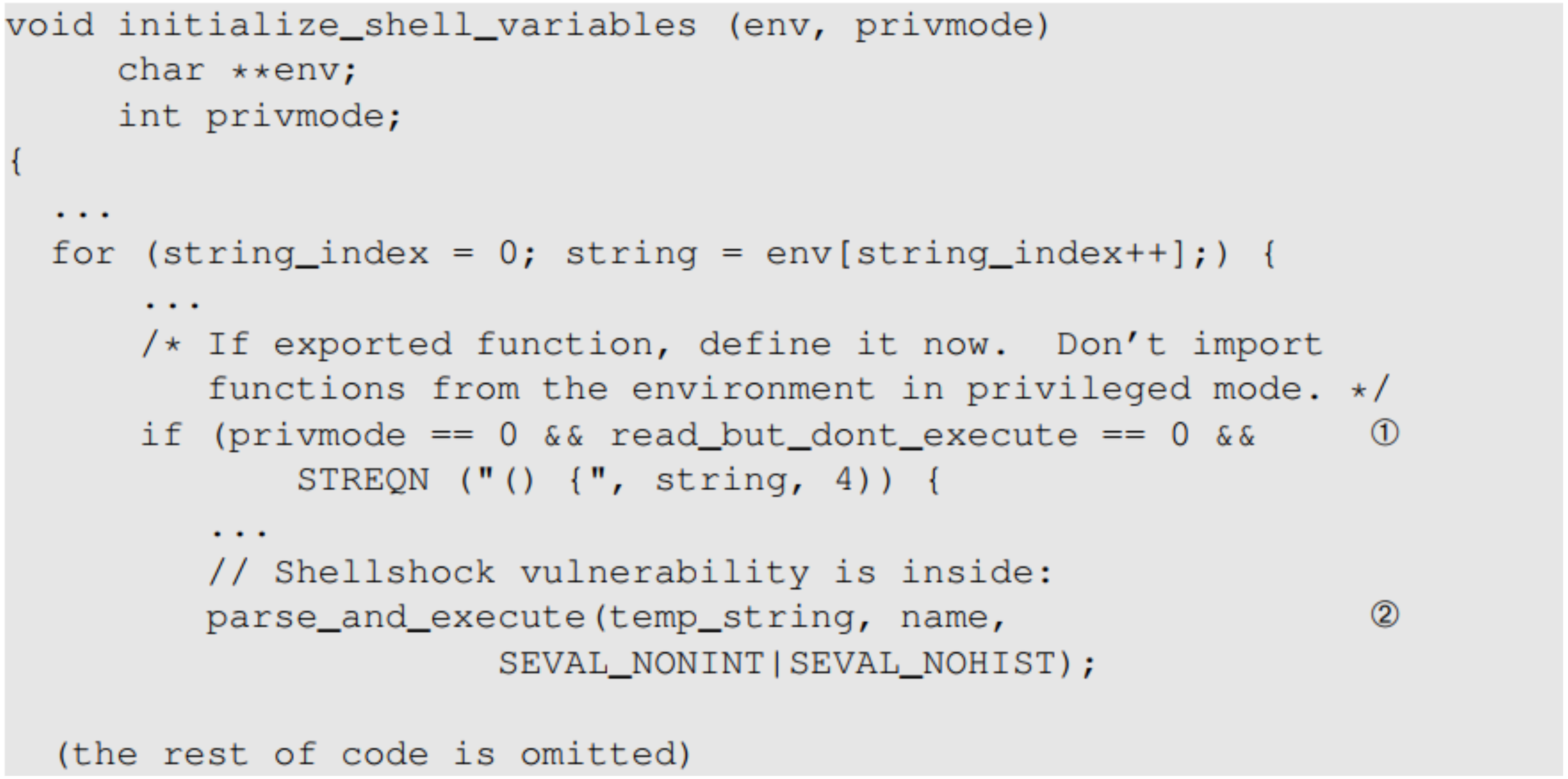
- The parent process can pass a function definition to a child shell process via an environment variable
- Due to a bug in the parsing logic, bash executes **trailing commands** contained in the env. variable



```
$ 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"
}
```

The Mistake in the Bash Source Code

- The Shellshock bug starts in the `variables.c` file in the bash source code
- The following code snippet that highlights the mistake:



```
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)
```

The Mistake in the Bash Source Code *(cont.)*

- At ①, bash checks if there is an exported function by checking whether the value of an env. variable starts with "`() {`" or not. Once found, bash replaces the "`=`" with a space.

- Bash then calls the function `parse_and_execute()` (②) to parse the functions definition. Unfortunately, this function can parse other shell commands, not just the function definition!

- If the string is a function definition ~~> parse it but don't execute it

- If the string contains a shell command ~~> execute it

```
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)
```

The Mistake in the Bash Source Code *(cont.)*

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

- bash identifies Line A as a function because of the leading "`() {`" and converts it to Line B
- We see that the string now becomes two commands
- Now, `parse_and_execute()` will execute *both* commands!

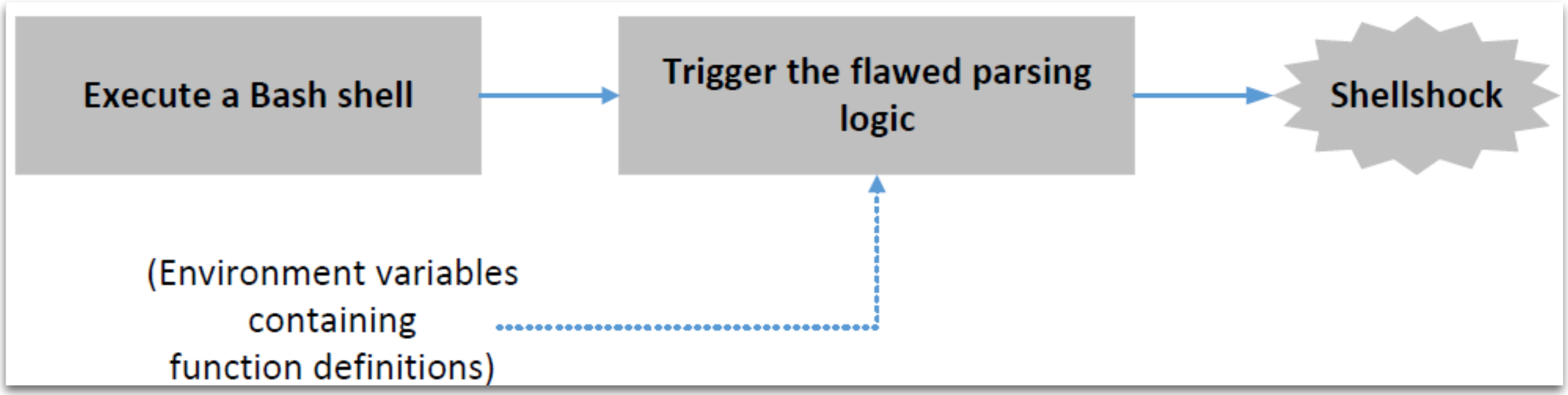
Consequences

- Attackers can get a process to run their commands
- If the target process is a server process or runs with elevated privileges, a security breach can occur

Exploiting the Shellshock Vulnerability

Two conditions are needed to exploit the vulnerability:

- The target process should run **bash**
- The target process should get **untrusted user inputs via env. variables**


Execute a Bash shell
Trigger the flawed parsing logic
Shellshock
(Environment variables containing function definitions)