Lecture 13: Environment Variables & Attacks

Security in libc

Xin Liu

Florida State University
xl24j@fsu.edu

CIS 5370 Computer Security
https://xinliulab.github.io/cis5370.html

Outline

We have already learned that an ”executable file” is a data structure that describes the initial state of a process. Through the Funny Little Executable, we explored the compilation, linking, and loading processes involved in generating an executable file.

Today’s Key Question:

- As the software ecosystem evolved, the need for **”decomposing”** software and dynamic linking emerged!

Main Topics for Today:

- Dynamic Linking and Loading: Principles and Implementation
- Security in **libc**

Environment Variables

- A set of dynamically named values.
- Part of the operating environment in which a process runs.
- Affect how a running process behaves.
- Introduced in Unix and later adopted by Microsoft Windows.
- **Example: PATH variable**
  - When a program is executed, the shell process uses the PATH environment variable to locate the program if the full path is not provided.

How to Access Environment Variables

Accessing from the `main` function:

- The environment variables can be accessed via the third argument `envp[]` in `main()`.

Listing 1: Accessing envp from main

```
#include <stdio.h>
void main(int argc, char* argv[], char* envp[])
{
   int i = 0;
   while (envp[i] != NULL) {
      printf("%s\n", envp[i++]);
   }
}
```

A more reliable way: Using the global variable `environ`

- `environ` is a global variable available in most UNIX systems, storing environment variables.
- This method is independent of how `main()` is defined.

Using the Global Variable `environ`

Listing 2: Accessing envp using environ

```c
#include <stdio.h>
extern char** environ;

void main(int argc, char* argv[], char* envp[])
{
   int i = 0;
   while (environ[i] != NULL) {
      printf("%s\n", environ[i++]);
   }
}
```

Key Takeaways:

- The global variable `environ` provides direct access to environment variables.
- Unlike `envp`, `environ` does not rely on `main()` parameters.

How Does a Process Get Environment Variables?

Process can obtain environment variables in two ways:

- If a new process is created using the `fork()` system call, the child process inherits the parent process's environment variables.
- If a process executes a new program, it typically uses the `execve()` system call.
  - In this case, the process's memory space is overwritten, and all previous environment variables are lost.
  - However, `execve()` allows explicitly passing environment variables from one process to another.

Passing Environment Variables When Invoking `execve()`:

Using execve() to Pass Environment Variables

Listing 3: Passing environment variables using execve

```
int execve(const char *filename, char *const argv[],
        char *const envp[]);
```

Key Points:

- The third argument envp[] allows specifying a new environment.
- If envp[] is NULL, the child process inherits the environment of the calling process.
- This mechanism enables controlled execution environments when spawning new processes.

execve() and Environment Variables

Effect of Different Arguments in execve():

```
$ a.out 1 Passing NULL
$ a.out 2 Passing newenv[]
AAA=aaa
BBB=bbb
$ a.out 3 Passing environ
SSH_AGENT_PID=2428
GPG_AGENT_INFO=/tmp/keyring-12UoOe/gpg:0:1
TERM=xterm
SHELL=/bin/bash
XDG_SESSION_COOKIE=6da3e071019f...
WINDOWID=39845893
OLDPWD=/home/seed/Book/Env_Variables
...
```

Key Observations:

- When passing `NULL`, the new process does not inherit environment variables.
- When passing a custom `newenv[]`, only the explicitly defined variables are available.
- When passing `environ`, the child process inherits all environment variables from the parent.

execve() and Environment Variables

Behavior of execve() with Different Environment Variable Arguments

- `execve()` allows specifying environment variables for the new process.
- Three different cases are demonstrated in the example:
  1. Passing `NULL` - No environment variables.
  2. Passing a custom `newenv[]` - Only explicitly defined variables are available.
  3. Passing `environ` - The child process inherits the parent's environment.

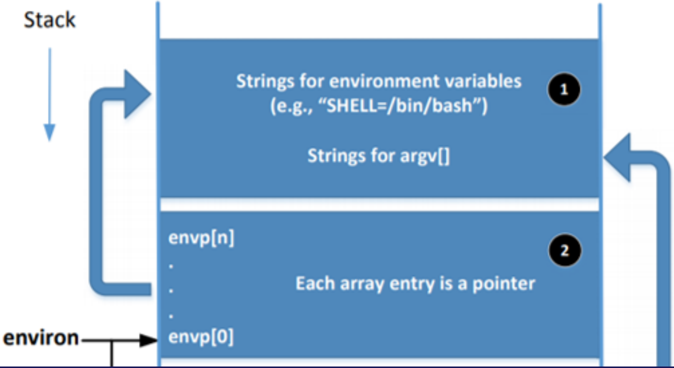
```
$ a.out 1 Passing NULL
$ a.out 2 Passing newenv[]
AAA=aaa
BBB=bbb
$ a.out 3 Passing environ
SSH_AGENT_PID=2428
GPG_AGENT_INFO=/tmp/keyring-12UoOe/gpg:0:1
TERM=xterm
SHELL=/bin/bash
XDG_SESSION_COOKIE=6da3e071019f...
WINDOWID=39845893
OLDPWD=/home/seed/Book/Env_Variables
...
```

Key Takeaways:

Memory Location for Environment Variables

Key Points:

- `envp` and `environ` initially point to the same memory location.
- `envp` is only accessible inside the `main` function, whereas `environ` is a global variable.
- When modifying environment variables (e.g., adding new ones), they may be moved to the heap.
- As a result, `environ` will change, but `envp` remains unchanged.



Stack
Strings for environment variables
(e.g., "SHELL=/bin/bash")
1
Strings for argv[]
envp[n]
.
.
.
envp[0]
2
Each array entry is a pointer
environ

Shell Variables & Environment Variables

Key Points:

- People often mistake shell variables and environment variables to be the same.
- **Shell Variables:**
  - Internal variables used by the shell.
  - Shell provides built-in commands to create, assign, and delete shell variables.
  - Example: Creating and unsetting a shell variable named `FOO`.

```
seed@ubuntu:~$ FOO=bar
seed@ubuntu:~$ echo $FOO
bar
seed@ubuntu:~$ unset FOO
seed@ubuntu:~$ echo $FOO

seed@ubuntu:~$
```

Takeaways: Finding the Core Structure in Complexity

Understanding Dynamic Linking Through Implementation

- By attempting to implement dynamic linking and loading ourselves, we gain deep insights into the process.
- In doing so, we ”invent” key ELF concepts, such as:
  - The **Global Offset Table (GOT)** for resolving addresses dynamically.
  - The **Procedure Linkage Table (PLT)** for indirect function calls.
- This hands-on approach reveals the underlying principles behind complex systems.