Lecture 10 – Supply Chain Security

[COSE451] Software Security

Instructor: Seunghoon Woo

Spring 2024

Overview

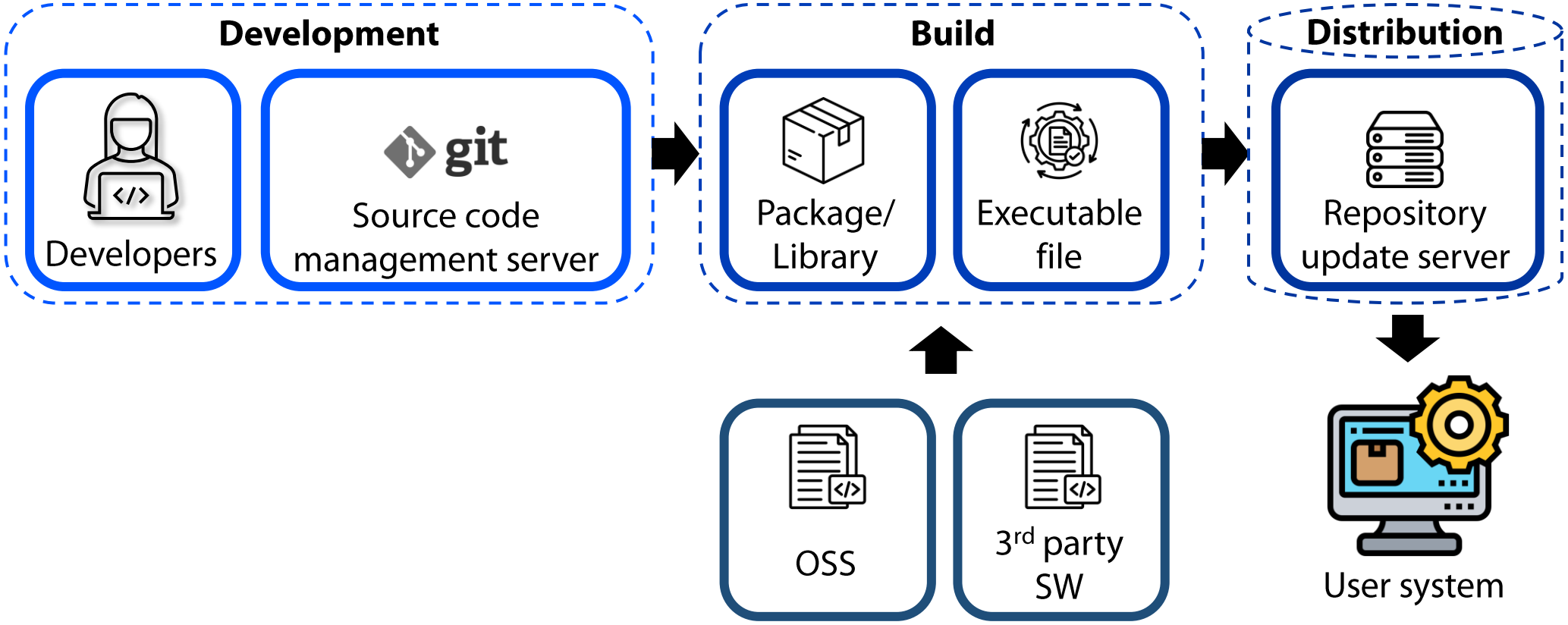
- **Supply chain security**

Software supply chain

- **Software supply chain**
  - Everything that affects or plays a role in a product or application throughout the entire software development life cycle (SDLC)
    - E.g., custom code (in-house components), open source dependencies/libraries (third-party components), development tools, infrastructure that make up the CI/CD process (Continuous Integration/Continuous Deployment), developers, and other related teams

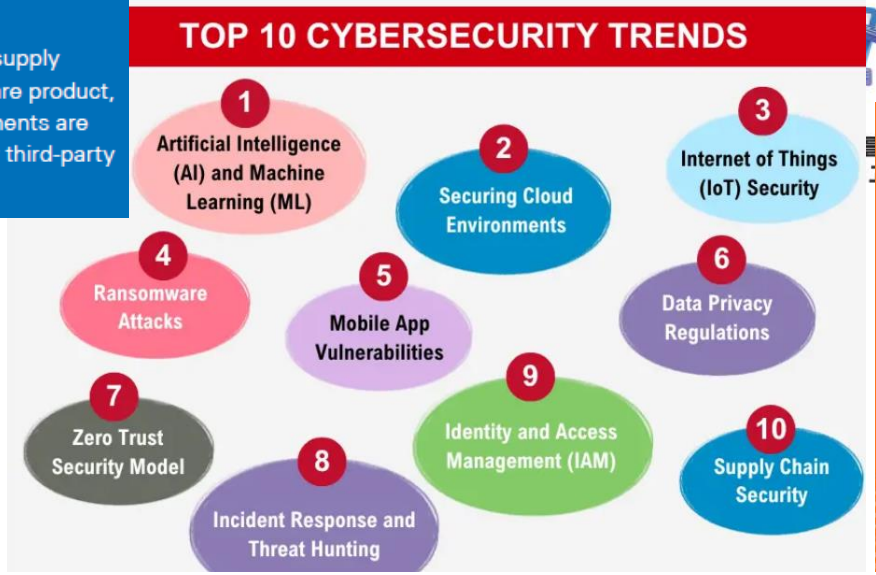
Software supply chain

- **Software supply chain**



Software supply chain

- **Software supply chain**



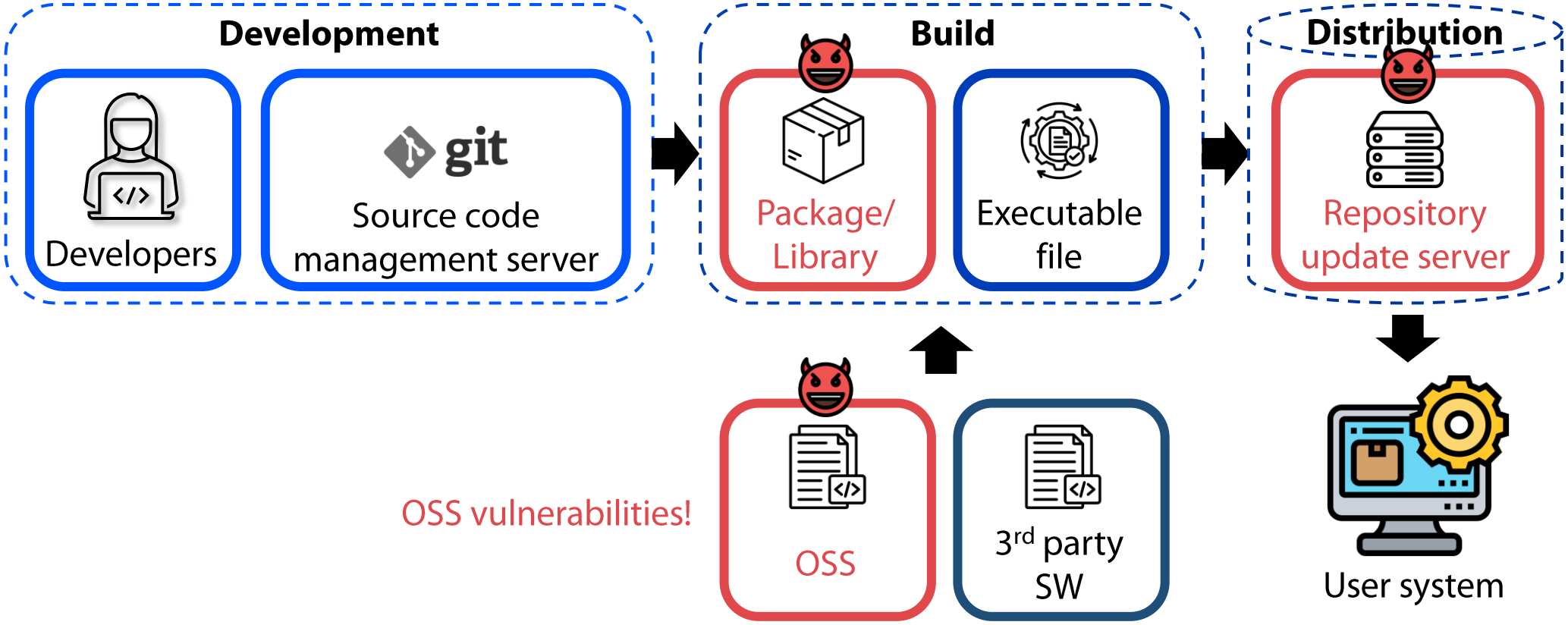
Software supply chain

- **Main types and targets of SW supply chain attacks**

Types / Targets	Description
Vulnerabilities in OSS	Vulnerabilities in OSS can propagate to other software (1-day vulnerabilities)
Third-Party Dependencies	Attackers exploit systems by inserting malicious code into third-party software (commercial SDKs, libraries, or components)
Public Repositories	Uploading malware with names similar to legitimate software packages to well-known repository hosting services like GitHub, targeting developers searching for open-source code
Build Systems	Intrusion into critical code, repositories, containers, and conversion servers on CI/CD for development process automation, replacing them with malicious code.
Hijacking Updates	Attackers interfere with the software update process or hijack admin rights of update servers to insert malicious code
Private Repositories	Intrusion by attackers into code repositories used within a company to insert malicious code

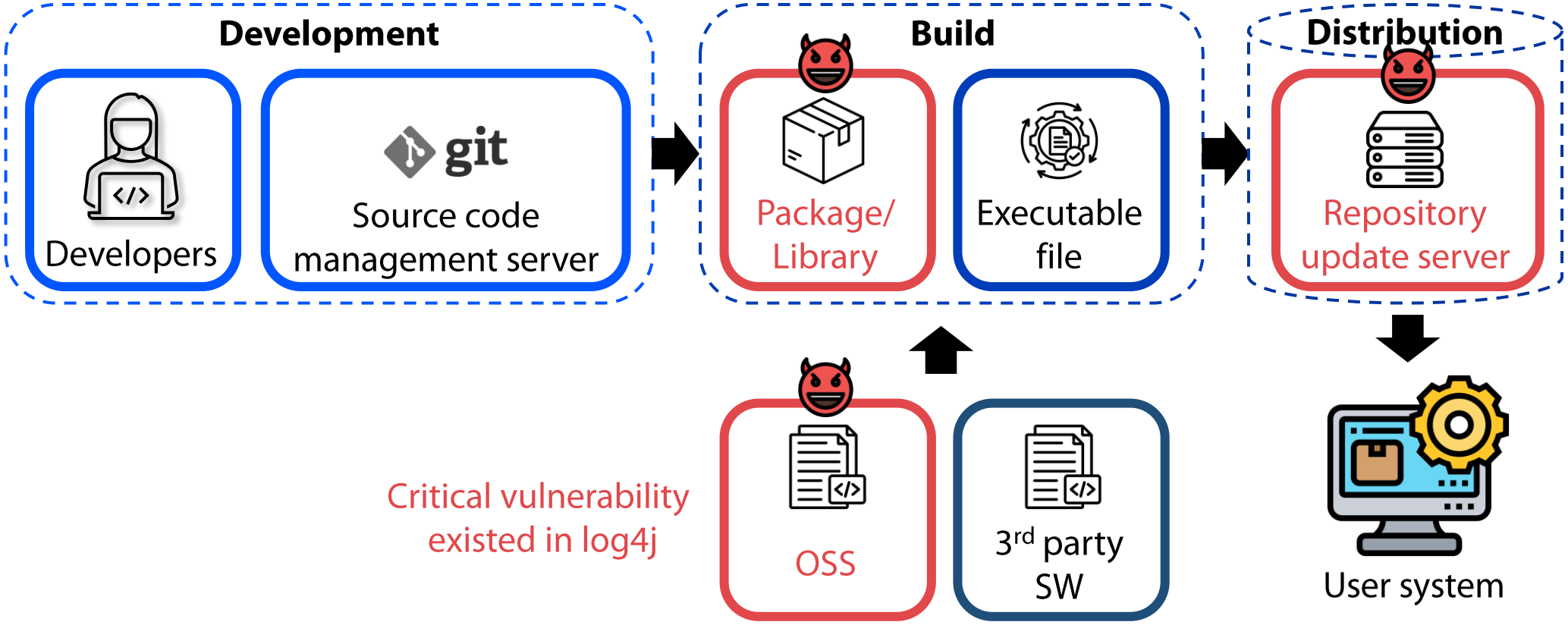
Software supply chain

- **Vulnerabilities in OSS**



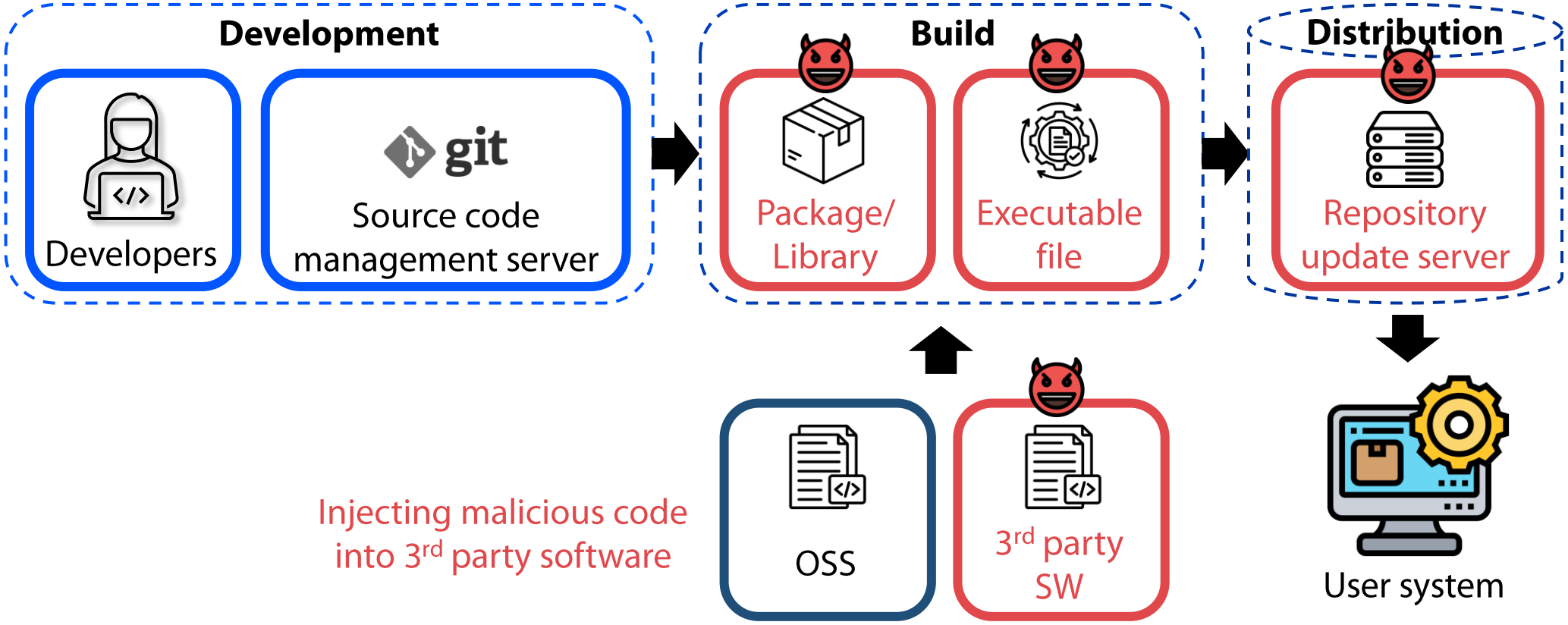
Software supply chain

- **Vulnerabilities in OSS: Log4shell vulnerability**



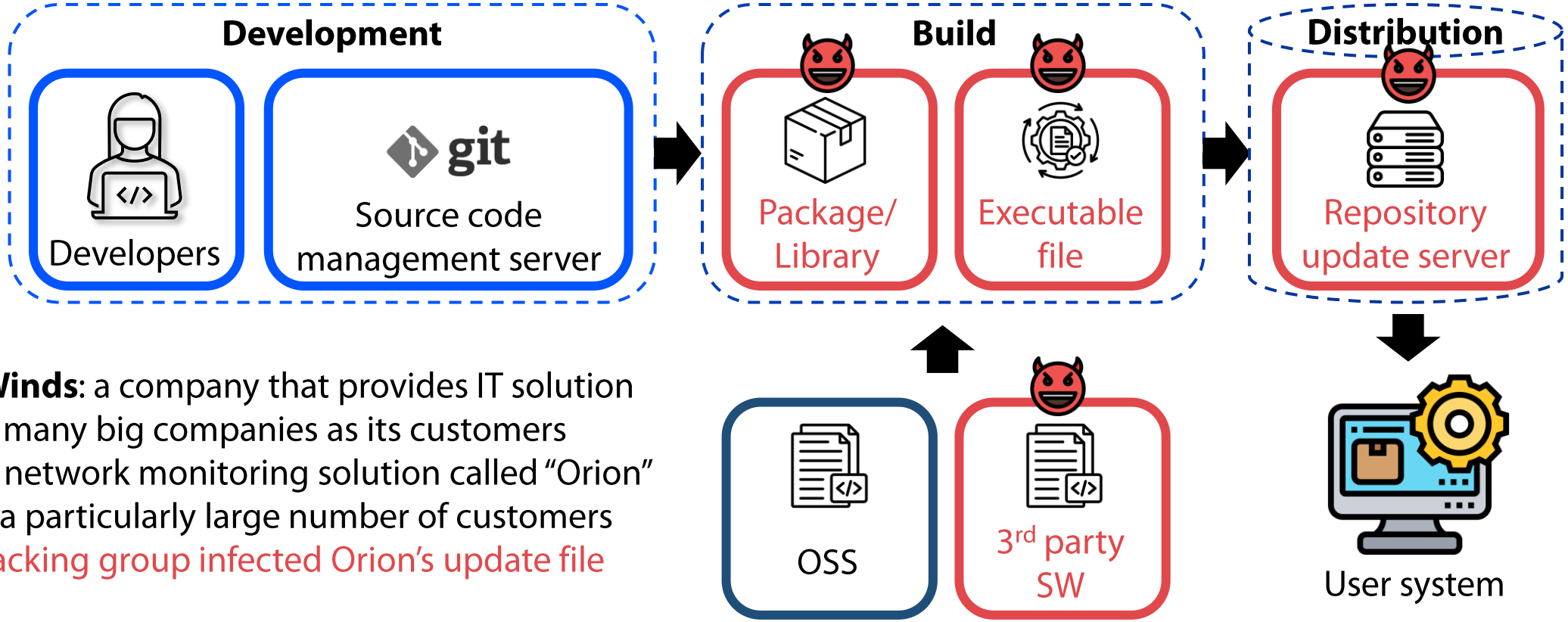
Software supply chain

- 3rd party dependencies



Software supply chain

- 3rd party dependencies: SolarWinds (2021)

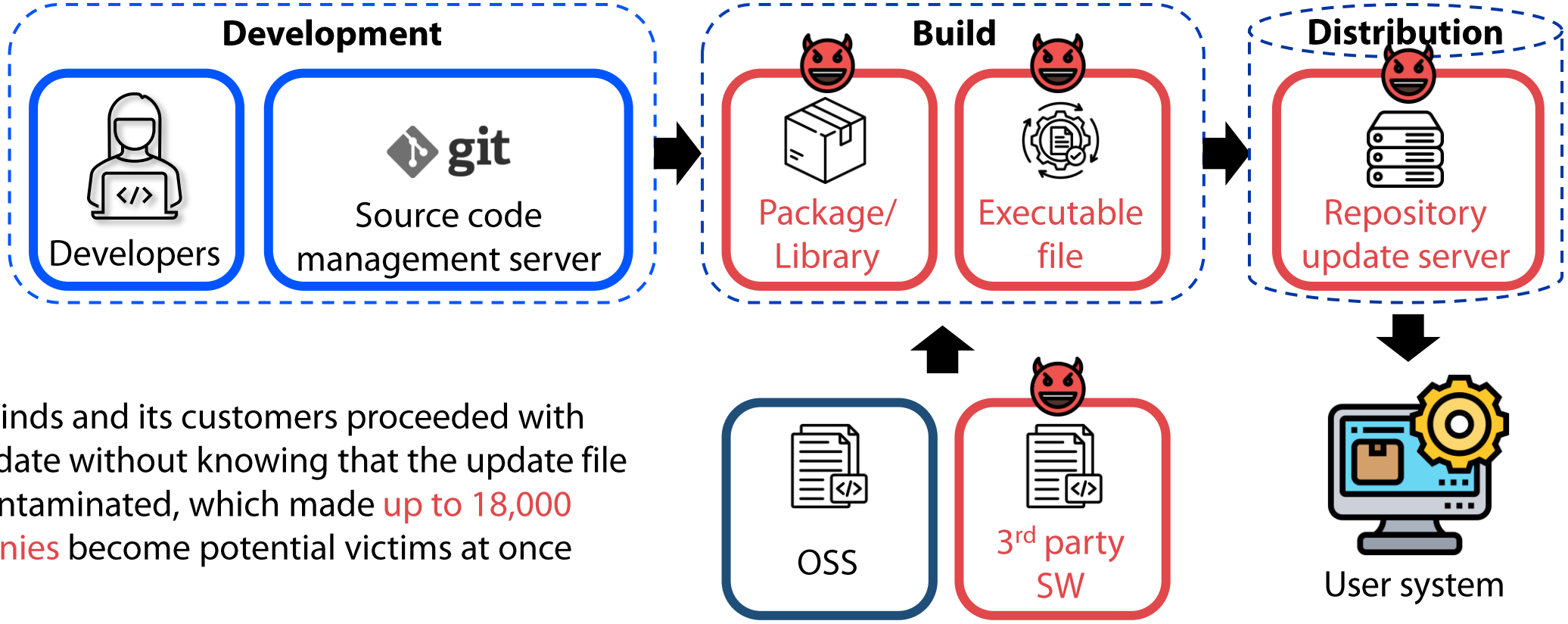


SolarWinds: a company that provides IT solution

- Has many big companies as its customers
- The network monitoring solution called "Orion" has a particularly large number of customers
- A hacking group infected Orion's update file

Software supply chain

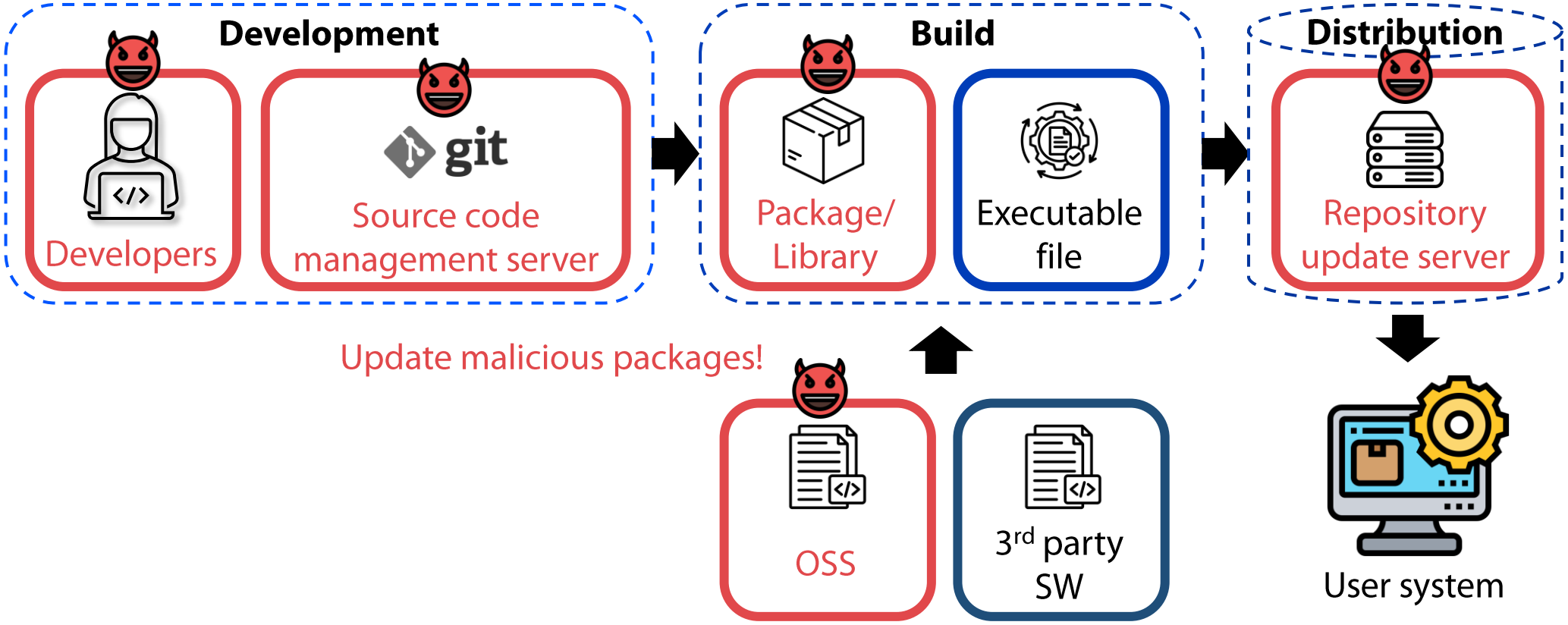
- 3rd party dependencies: SolarWinds (2021)



SolarWinds and its customers proceeded with the update without knowing that the update file was contaminated, which made up to 18,000 companies become potential victims at once

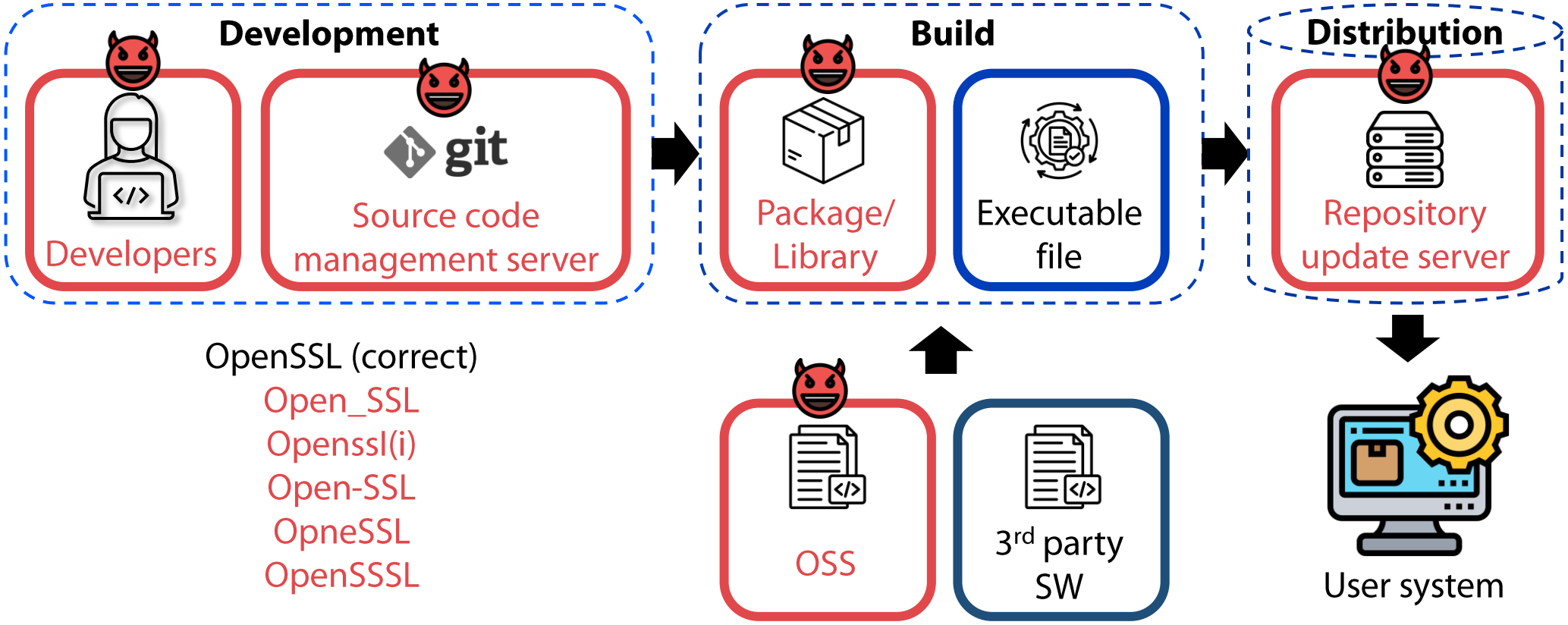
Software supply chain

- **Public Repositories**



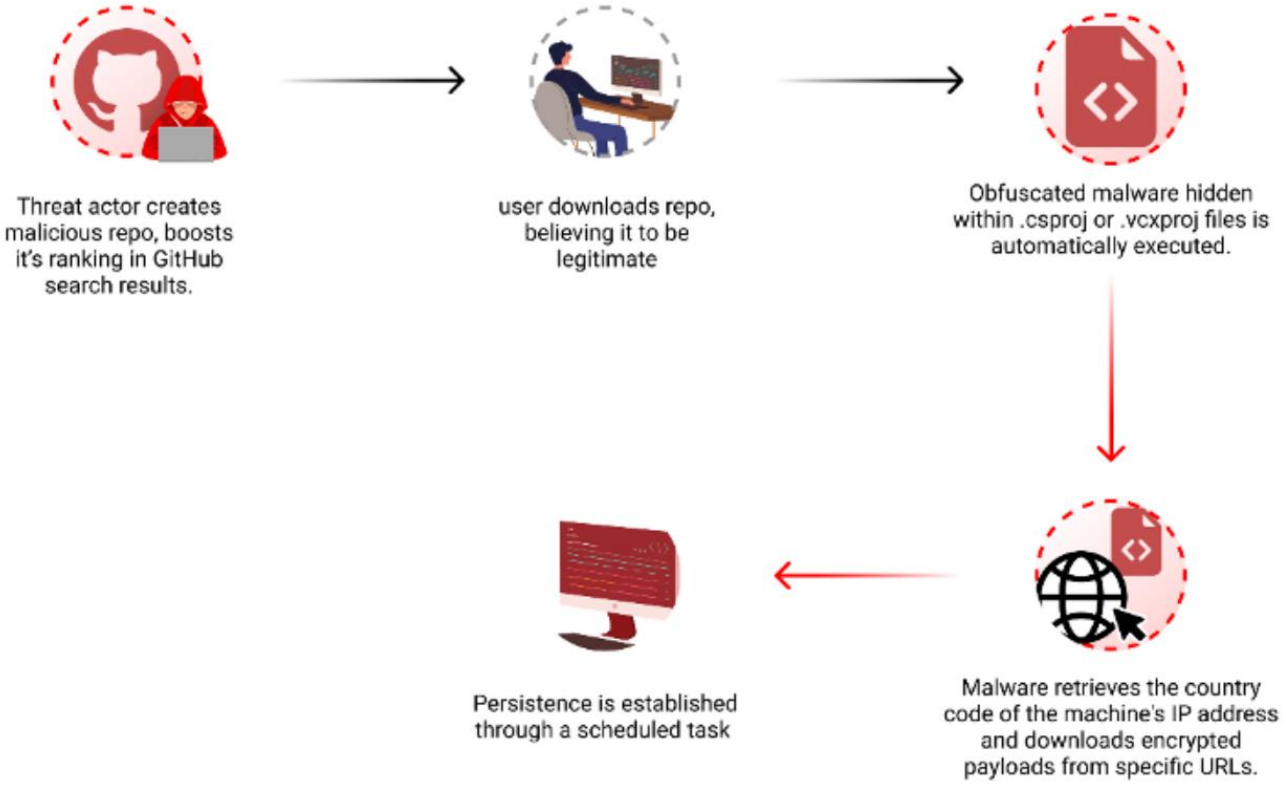
Software supply chain

- **Public Repositories: Typosquatting**



Software supply chain

- **Public Repositories: GitHub manipulation**

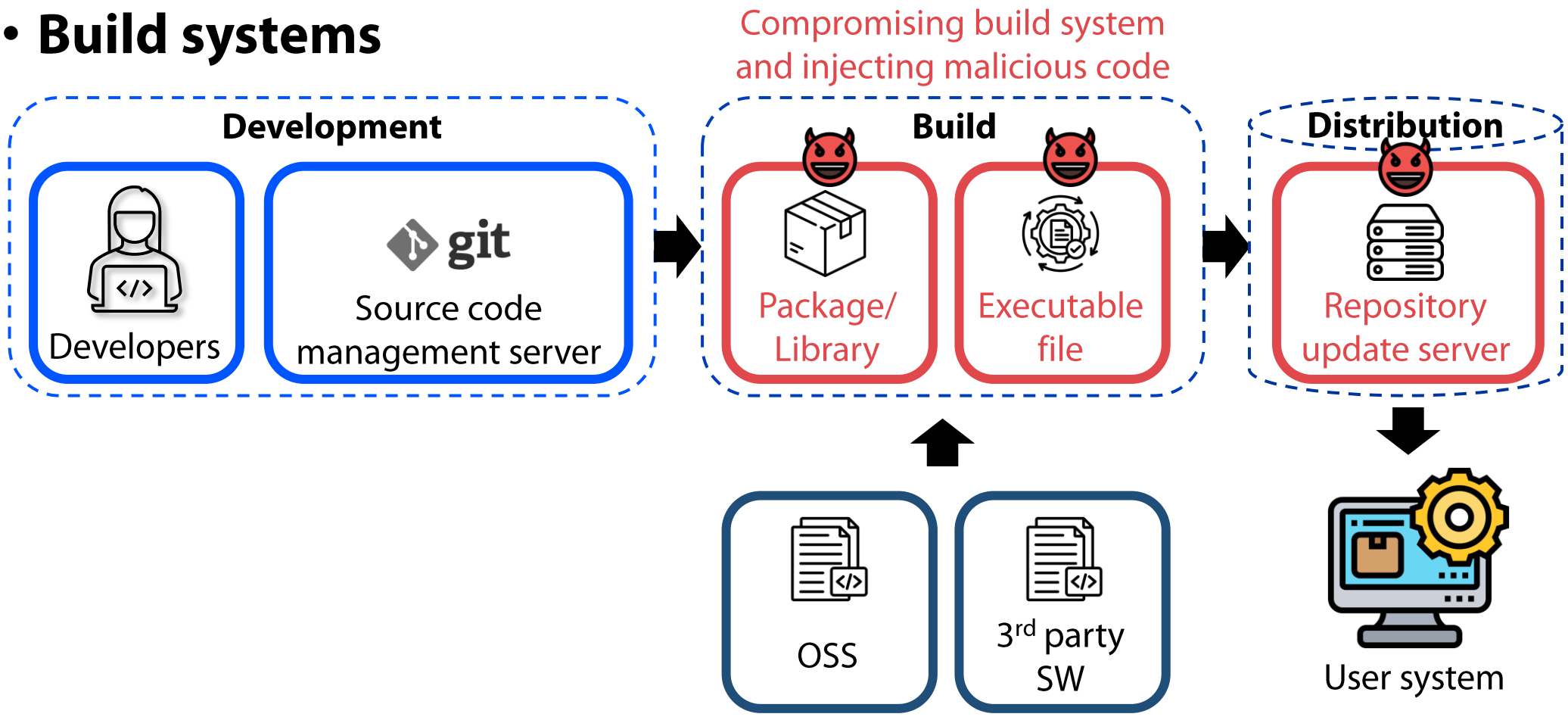


Attackers abuse GitHub's search functionality to trick users into downloading malicious repositories disguised as popular ones!

https://thehackernews.com/2024/04/beware-githubs-fake-popularity-scam.html

Software supply chain

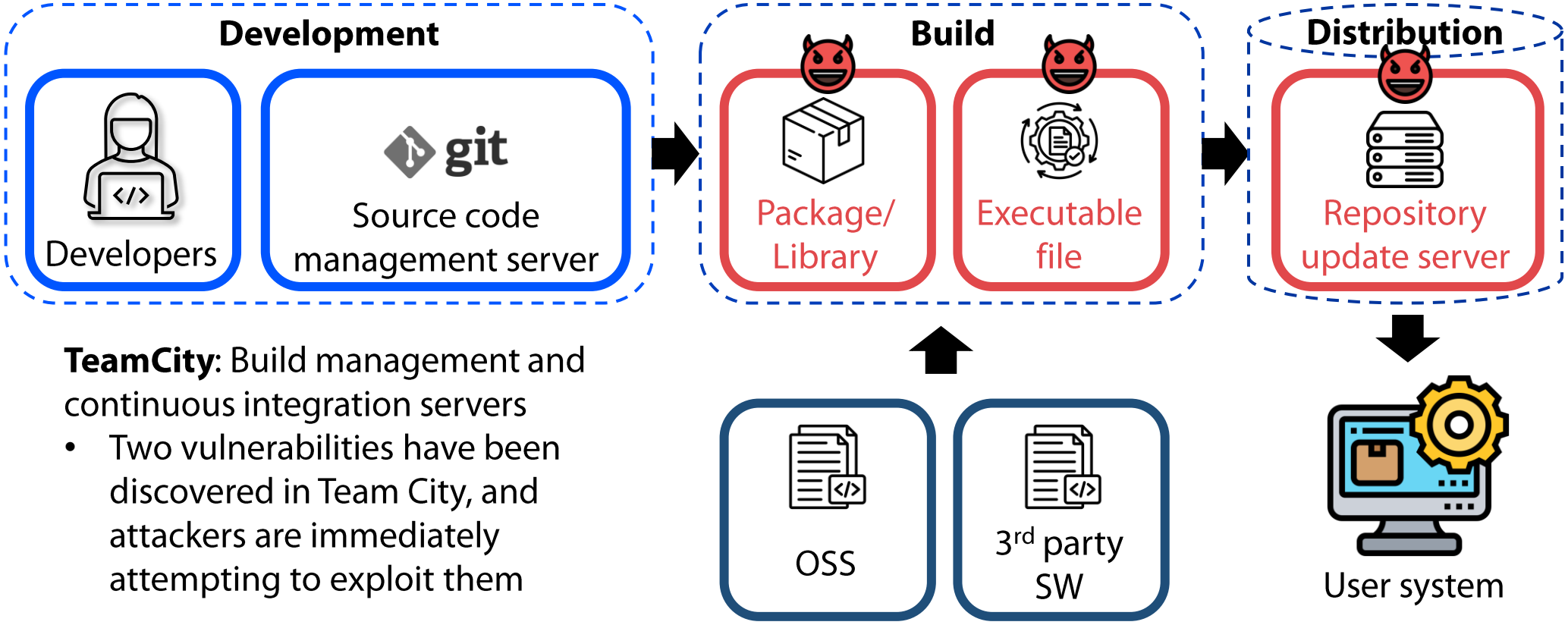
- **Build systems**



https://www.securityweek.com/critical-teamcity-vulnerability-exploitation-started-immediately-after-disclosure/

Software supply chain

- **Build systems: TeamCity Vulnerability**

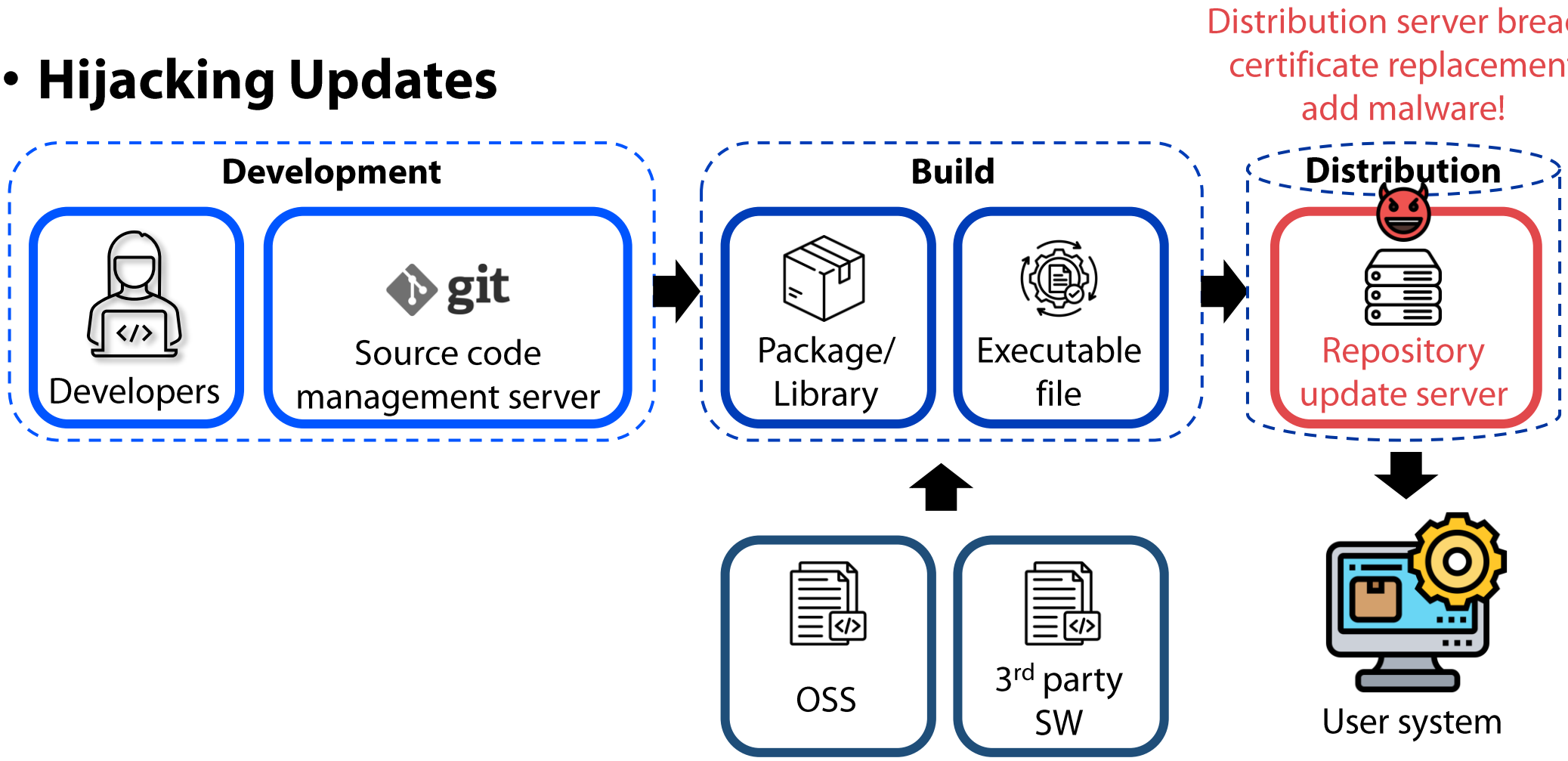


TeamCity: Build management and continuous integration servers

- Two vulnerabilities have been discovered in Team City, and attackers are immediately attempting to exploit them

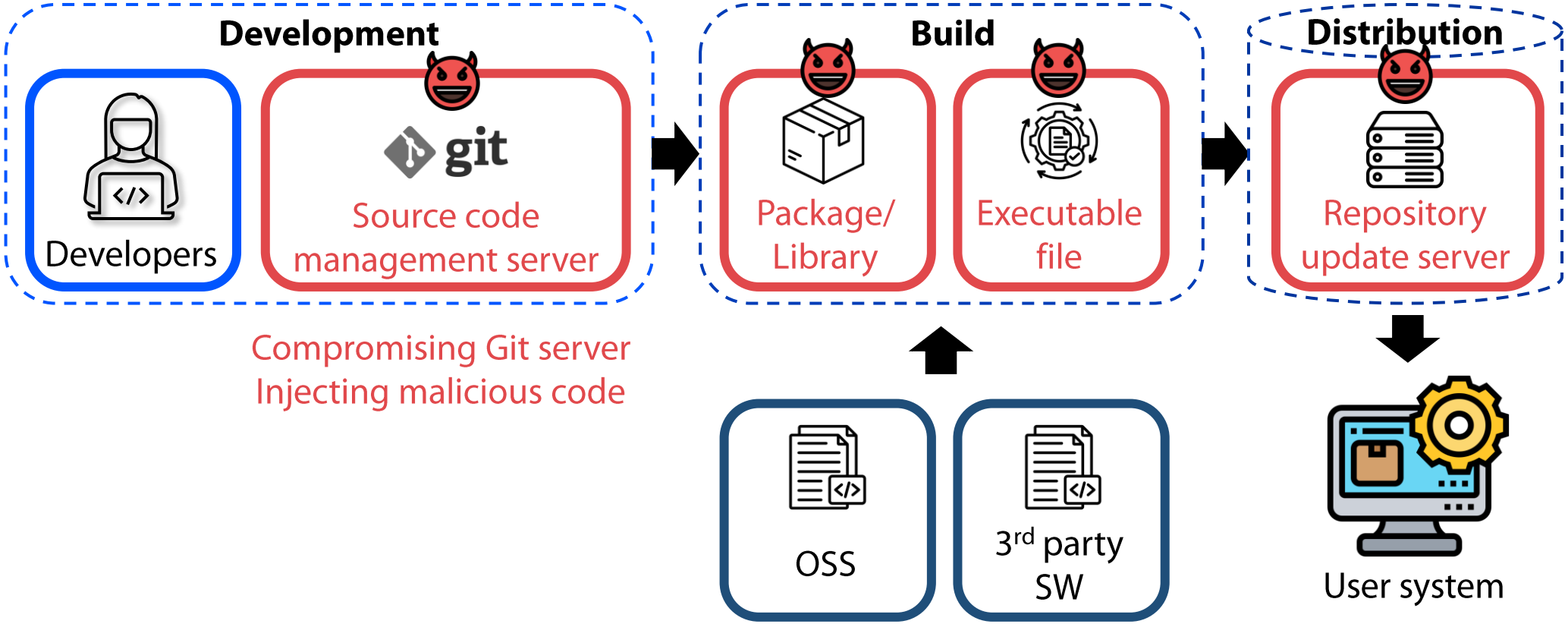
Software supply chain

- **Hijacking Updates**



Software supply chain

- **Private Repositories**



Assignment II

[COSE451] Software Security

Instructor: Seunghoon Woo

Spring 2024

Assignment II

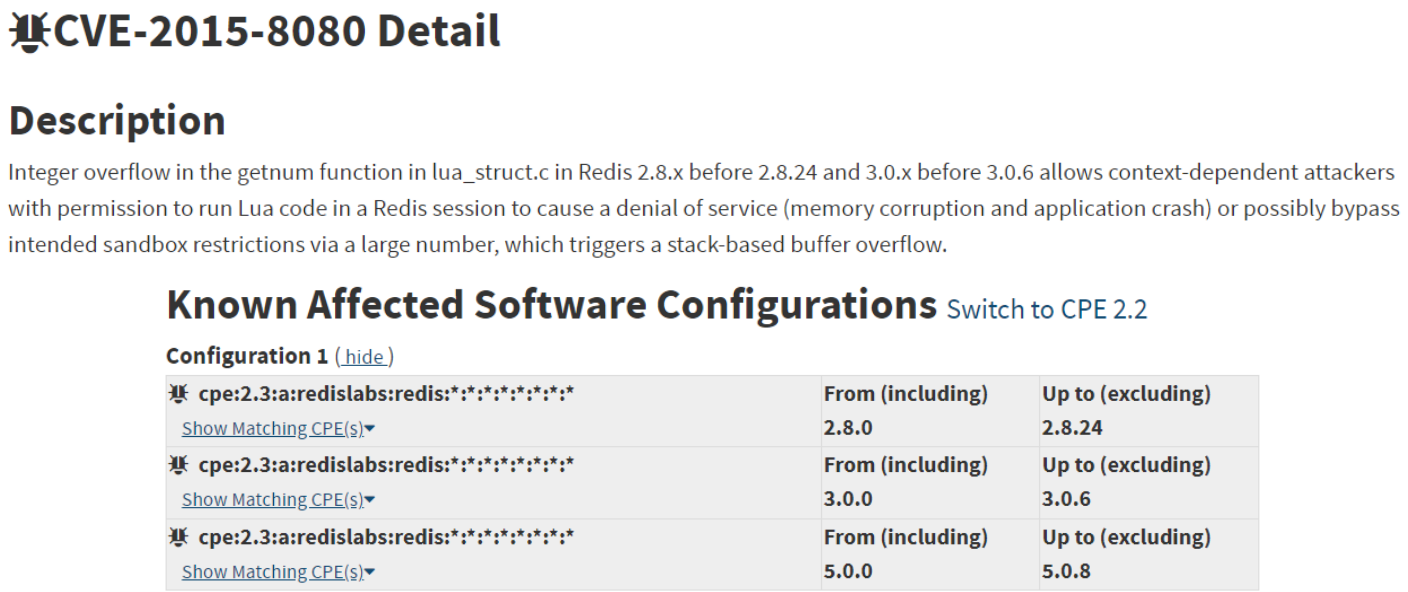
- **Fixing security vulnerability!**
  - **Goal**
    1. Verify the impact of vulnerabilities
       - Check how the actual vulnerability is triggered!
    2. Vulnerability patch practice
       - Try to fix the vulnerability!

Assignment II

- **Fixing security vulnerability!**
  - **Steps**
    1. Check(identify) vulnerable code
    2. Trigger the vulnerability
    3. Apply security patch
    4. Ensure the vulnerability is safely remediated

Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - We target older versions where vulnerabilities exist



CVE-2015-8080 Detail

Description

Integer overflow in the getnum function in lua_struct.c in Redis 2.8.x before 2.8.24 and 3.0.x before 3.0.6 allows context-dependent attackers with permission to run Lua code in a Redis session to cause a denial of service (memory corruption and application crash) or possibly bypass intended sandbox restrictions via a large number, which triggers a stack-based buffer overflow.

Known Affected Software Configurations Switch to CPE 2.2

Configuration 1 (hide)

cpe:2.3:a:redislabs:redis:*:*:*:*:*:*:*:* Show Matching CPE(s)	From (including) 2.8.0	Up to (excluding) 2.8.24
cpe:2.3:a:redislabs:redis:*:*:*:*:*:*:*:* Show Matching CPE(s)	From (including) 3.0.0	Up to (excluding) 3.0.6
cpe:2.3:a:redislabs:redis:*:*:*:*:*:*:*:* Show Matching CPE(s)	From (including) 5.0.0	Up to (excluding) 5.0.8

https://nvd.nist.gov/vuln/detail/CVE-2015-8080

Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - We target older versions where vulnerabilities exist

Hyperlink	Resource
http://lists.opensuse.org/opensuse-updates/2016-05/msg00126.html	Mailing List, Third Party Advisory
http://rhn.redhat.com/errata/RHSA-2016-0095.html	Third Party Advisory
http://rhn.redhat.com/errata/RHSA-2016-0096.html	Third Party Advisory
http://rhn.redhat.com/errata/RHSA-2016-0097.html	Third Party Advisory
http://www.debian.org/security/2015/dsa-3412	Third Party Advisory
http://www.openwall.com/lists/oss-security/2015/11/06/2	Mailing List, Third Party Advisory
http://www.openwall.com/lists/oss-security/2015/11/06/4	Mailing List, Third Party Advisory
http://www.securityfocus.com/bid/77507	Third Party Advisory, VDB Entry
https://github.com/antirez/redis/issues/2855	Exploit, Issue Tracking, Patch, Third Party Advisory
https://raw.githubusercontent.com/antirez/redis/2.8/00-RELEASENOTES	Release Notes, Third Party Advisory
https://raw.githubusercontent.com/antirez/redis/3.0/00-RELEASENOTES	Release Notes, Third Party Advisory
https://security.gentoo.org/glsa/201702-16	Third Party Advisory

Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - We target older versions where vulnerabilities exist

`getnum()` can be tricked into an integer wraparound with a large size number as input, thus returning a negative value. `optsize()` has no lower bound/negative check; moreover, there is an implicit `int` -> `size_t` promotion, yielding a very large (unsigned) size value.

This, plus further `int` / `size_t` confusion in the whole module, results in stack-based buffer overflows in other places, eg. `putinteger()` reachable in LUA via `struct.pack()`.

Simple PoC as follow:

```
EVAL "struct.pack('>I2147483648', '10')" 0
```

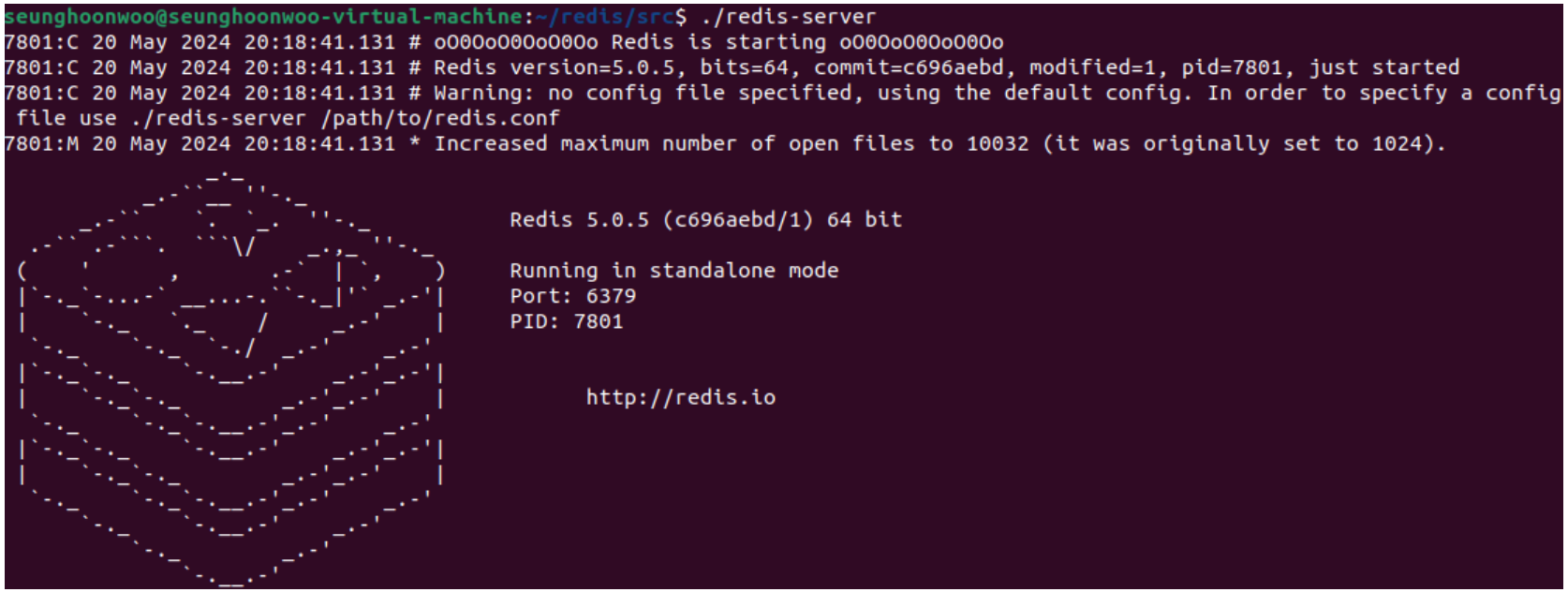
Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - Build Redis v5.0.5

```
seunghoonwoo@seunghoonwoo-virtual-machine:~/redis$ sudo make
cd src && make all
make[1]: Entering directory '/home/seunghoonwoo/redis/src'
    CC quicklist.o
    CC server.o
    CC sds.o
    ...
    LINK redis-cli
    CC redis-benchmark.o
    LINK redis-benchmark
    INSTALL redis-check-rdb
    INSTALL redis-check-aof

Hint: It's a good idea to run 'make test' ;)
```

Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - Execute Redis v5.0.5



Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - PoC test!

```
seunghoonwoo@seunghoonwoo-virtual-machine:~/redis/src$ ./redis-cli
127.0.0.1:6379> EVAL "struct.pack('>I2147483648', '10')" 0
Could not connect to Redis at 127.0.0.1:6379: Connection refused
not connected>
```

Assignment II

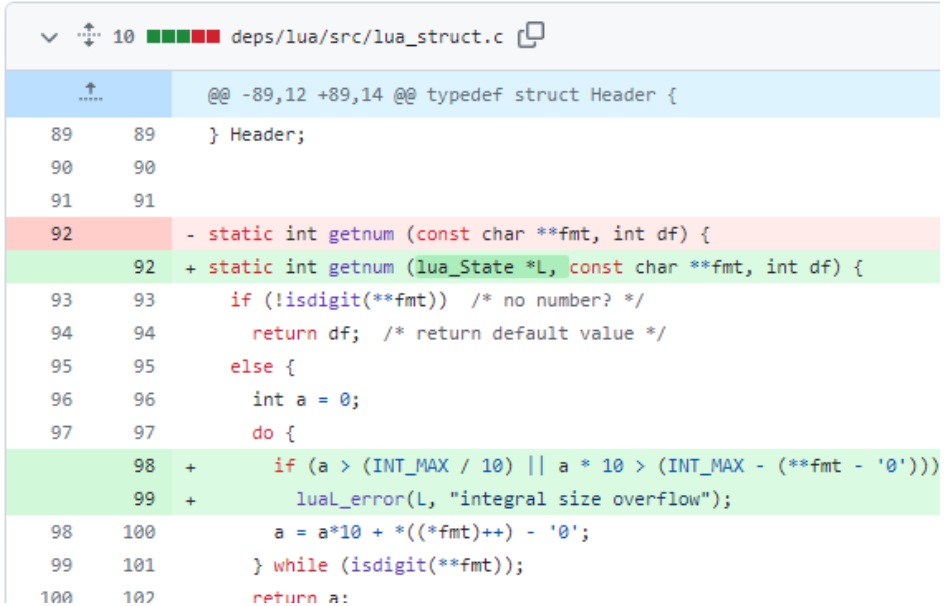
- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - PoC test!

```
seunghoonwoo@seunghoonwoo-virtual-machine:~/redis/src$ ./redis-cli
127.0.0.1:6379> EVAL "struct.pack('>I2147483648', '10')" 0
Could not connect to Redis at 127.0.0.1:6379: Connection refused
not connected>
```

```
Fast memory test PASSED, however your memory can still be broken. Please run a memory test for several hours if possible.

------ DUMPING CODE AROUND EIP ------
Symbol: (null) (base: (nil))
Module: ./redis-server *:6379 (base 0x5f89ca458000)
$ xxd -r -p /tmp/dump.hex /tmp/dump.bin
$ objdump --adjust-vma=(nil) -D -b binary -m i386:x86-64 /tmp/dump.bin
------

=== REDIS BUG REPORT END. Make sure to include from START to END. ===

       Please report the crash by opening an issue on github:

           http://github.com/antirez/redis/issues

  Suspect RAM error? Use redis-server --test-memory to verify it.

Segmentation fault (core dumped)
```

Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - Fix the vulnerability



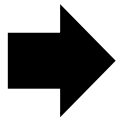
10 deps/lua/src/lua_struct.c

```
@@ -89,12 +89,14 @@ typedef struct Header {
89  89  } Header;
90  90
91  91
92      - static int getnum (const char **fmt, int df) {
    92  + static int getnum (lua_State *L, const char **fmt, int df) {
93  93    if (!isdigit(**fmt))  /* no number? */
94  94      return df;  /* return default value */
95  95    else {
96  96      int a = 0;
97  97      do {
    98  +       if (a > (INT_MAX / 10) || a * 10 > (INT_MAX - (**fmt - '0')))
    99  +         luaL_error(L, "integral size overflow");
98  100       a = a*10 + *((*fmt)++) - '0';
99  101     } while (isdigit(**fmt));
100 102     return a;
```

Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - Fix the vulnerability

```
static int getnum (const char **fmt, int df) {
  if (!isdigit(**fmt))  /* no number? */
    return df;  /* return default value */
  else {
    int a = 0;
    do {
      a = a*10 + *((*fmt)++) - '0';
    } while (isdigit(**fmt));
    return a;
  }
}
```



```
static int getnum (lua_State *L, const char **fmt, int df) {
  if (!isdigit(**fmt))  /* no number? */
    return df;  /* return default value */
  else {
    int a = 0;
    do {
     if (a > (INT_MAX / 10) || a * 10 > (INT_MAX - (**fmt - '0')))
        luaL_error(L, "integral size overflow");
      a = a*10 + *((*fmt)++) - '0';
    } while (isdigit(**fmt));
    return a;
  }
}
```

Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - Build again..

```
seunghoonwoo@seunghoonwoo-virtual-machine:~/redis$ sudo make
cd src && make all
make[1]: Entering directory '/home/seunghoonwoo/redis/src'
    CC quicklist.o
    CC server.o
    CC sds.o
```

```
    LINK redis-cli
    CC redis-benchmark.o
    LINK redis-benchmark
    INSTALL redis-check-rdb
    INSTALL redis-check-aof

Hint: It's a good idea to run 'make test' ;)
```

Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - PoC test!

```
seunghoonwoo@seunghoonwoo-virtual-machine:~/redis$ ./src/redis-cli
127.0.0.1:6379> EVAL "struct.pack('>I2147483648', '10')" 0
(error) ERR Error running script (call to f_0ba5d6867f8a0d59c13d2ee49dc170ebdb28
89d7): @user_script:1: user_script:1: integral size overflow
```

Assignment II

- **Case 1) Targeting real-world OSS project**
  - Example: Redis case (v5.0.5, released in 2019)
    - PoC test!

```
seunghoonwoo@seunghoonwoo-virtual-machine:~/redis$ ./src/redis-cli
127.0.0.1:6379> EVAL "struct.pack('>I2147483648', '10')" 0
(error) ERR Error running script (call to f_0ba5d6867f8a0d59c13d2ee49dc170ebdb28
89d7): @user_script:1: user_script:1: integral size overflow
```

Proceed with this vulnerability trigger & patch process and submit a report

Assignment II

- **Case 1) Targeting real-world OSS project**
  - CVEs that are easy to detect PoC and trigger/patch the vulnerability
    - CVE-2018-19210, CVE-2016-10269, CVE-2016-10270, CVE-2017-5225 (LibTIFF)
      - LibTIFF vulnerabilities are generally easy to verify
    - CVE-2019-9169 (Glibc)
    - CVE-2016-3705 (LibXML2)
    - CVE-2017-0700 (LibGDX, Godot Engine)
    - CVE-2018-20330 (LibJPEG)
    - CVE-2019-17371 (Gif2png)
    - You can select any CVE (even if it is not displayed on this page)

Assignment II

- **Case 1) Targeting real-world OSS project**
  - CVEs that are easy to detect PoC and trigger/patch the vulnerability
    - CVE-2018-19210, CVE-2016-10269, CVE-2016-10270, CVE-2017-5225 (LibTIFF)
      - LibTIFF vulnerabilities are generally easy to verify
    - CVE-2019-9169 (Glibc)
    - CVE-2016-3705 (LibXML2)
    - CVE-2017-0700 (LibGDX, Godot Engine)
    - CVE-2018-20330 (LibJPEG)
    - CVE-2019-17371 (Gif2png)
    - You can select any CVE (even if it is not displayed on this page)

But this is a big big challenge for some students..

Assignment II

- **Case 2) Targeting toy example**
  1. Create a small vulnerable software based on vulnerabilities learned in class
     - Create a new one exclude code that appeared in class materials or assignments
  2. Show that the vulnerability can be triggered
  3. Try patching vulnerabilities (e.g., using input validation)
  4. Now you need to show that the vulnerability is not triggered! (i.e., fixed)

Assignment II

- **Case 2) Targeting toy example**
  - Example..

```c
#include <stdio.h>

int main(int argc, char * argv[]) {
    int valid = 0;
    char str1[8] = "START";
    char str2[8];

    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = 1;

    printf("Buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);

}
```

Assignment II

- **Case 2) Targeting toy example**
  - Example..

```c
#include <stdio.h>

int main(int argc, char * argv[]) {
    int valid = 0;
    char str1[8] = "START";
    char str2[8];

    gets(str2);
    if (strncmp(str1, str2, 8) == 0)
        valid = 1;

    printf("Buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);

}
```

```
seunghoonwoo@seunghoonwoo-virtual-machine:~$ ./overflow
BADINPUTBADINPUT
Buffer1: str1(BADINPUT), str2(BADINPUTBADINPUT), valid(1)
```

Assignment II

- **Case 2) Targeting toy example**
  - Example..

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char * argv[]) {
    int valid = 0;
    char str1[8] = "START";
    char str2[8];

    gets(str2);
    if (strlen(str2) > 8){
        printf("OVERFLOW!!!!!\n");
        return 0;
    }

    if (strncmp(str1, str2, 8) == 0)
        valid = 1;

    printf("Buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);

}
```

Assignment II

- **Case 2) Targeting toy example**
  - Example..

```c
#include <stdio.h>
#include <string.h>

int main(int argc, char * argv[]) {
    int valid = 0;
    char str1[8] = "START";
    char str2[8];

    gets(str2);
    if (strlen(str2) > 8){
        printf("OVERFLOW!!!!!\n");
        return 0;
    }

    if (strncmp(str1, str2, 8) == 0)
        valid = 1;

    printf("Buffer1: str1(%s), str2(%s), valid(%d)\n", str1, str2, valid);

}
```

```
seunghoonwoo@seunghoonwoo-virtual-machine:~$ ./overflow
TEST
Buffer1: str1(START), str2(TEST), valid(0)
seunghoonwoo@seunghoonwoo-virtual-machine:~$ ./overflow
START
Buffer1: str1(START), str2(START), valid(1)
```

```
seunghoonwoo@seunghoonwoo-virtual-machine:~$ ./overflow
BADINPUTBADINPUT
OVERFLOW!!!!!
```

Assignment II

- **Scoring**
  - Due to the significant difference in difficulty between cases 1 and 2, the final scores will also reflect this difference
    - For real-world OSS cases: a maximum of 100 points
    - For toy example cases: a maximum of 80 points

Assignment II

- **Due date:** June 14th 11:59 PM
- **To be submitted:**
  - Please compress the following three items into a single file (.zip) and submit it

1. Source code with vulnerabilities
   - For real-world OSS, only submit the files containing vulnerabilities
2. Source code with patches applied
3. Report
   - As shown in this material, you must include the following information (e.g., using screenshots)
     - where the vulnerability was located
     - how the vulnerability was triggered
     - how it was patched
     - confirmation that the vulnerability is no longer triggered

Next Lecture

- **Supply chain security**
- **Software Bill of Materials (SBOM)**