1. *Short answers: must be less than 30 words. In an exam, we will use a format with much shorter answers (eg, multiple-choice).*

- What is difference between *ruid* and *euid* of a process.

  Answer: The ruid does not normally change. The euid initially equals ruid; it changes while it executes a setuid program.

- When a non-root user starts executing a set-root-uid executable, what are the values of its `ruid` and `euid`. Can these change while in the executable.

  Answer: ruid stays the same; euid becomes root uid. They change if the executable does setuid or seteuid.

- When a non-root user starts executing a set-root-uid executable, can it change its `euid` and later restore its `euid`.

  Answer: Yes, using seteuid().

- What is the difference between *worm* and *virus*.

  Answer: Worm attacks a running program. Virus attacks a stored program.

- What is difference between *polymorphic virus* and *metamorphic virus*.

  Answer: Polymorphic virus encrypts itself in storage, but the decrypted code stays the same. Metamorphic virus changes its code at each new infection.

- What is difference between *browser fingerprint* and *cookie*

  Answer: The fingerprint has info about the browser (type, version, OS, architecture, hardware, etc). The server does not ask for it. A cookie is set by the server.

- What is difference between *web storage* and *cookie*

  Answer: Http sends all cookies in the scope of the URL. Web storage info is sent exclusively by client-side script.

- What is *cookie elevation*

  Answer: A website sets a cookie for a anonymous user. When the user logs in, the website continues to use the cookie but gives it more access rights.

- What are the three categories of security design principles.

  Answer: Prevention, mitigation, log/audit

- Can a browser that does not execute scripts be subjected to an XSS attack.

  Answer: No.

- Can a browser that does not execute scripts be subjected to a CSRF attack.

  Answer: Yes. Eg, has page a1.com, which sends a request to a2.com upon load.

- What is framebusting. Is it a defense against clickjacking.

  Answer: Framebusting is where a website includes code in its pages that prevent other pages from framing it. Defends against clickjacking, but code can be stopped.

2

```
int f(int arg) {
    int j3;
    char buf[16];
    FILE *badfile;
s2: badfile = fopen("badfile", "r");
    fread(buf, sizeof(char), 40, badfile);
    return arg+1;
}

int main(int argc, char **argv) {
    int j1;
    int j2[4];
    j1 = 123;
    return f(j1);
}
```

Assume the Ubuntu environment of project 1.

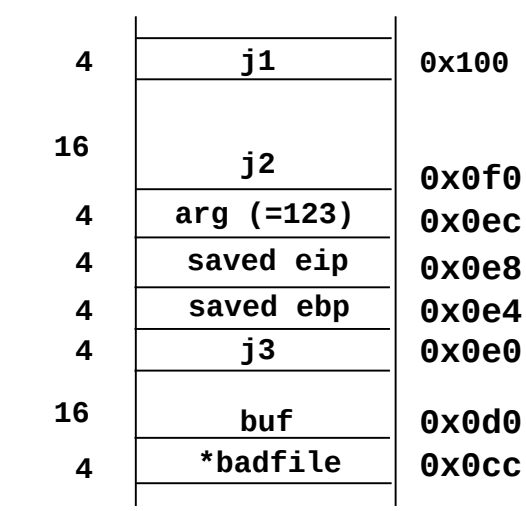
This program is compiled without Stack Guard and executed such that variable `j1` is always at address `0x00000100`.

Below, "draw the stack layout" means indicate, in the provided drawing, the contents of the stack from address `0x100` to the top of stack, and give the addresses (in **hex**) of the contents at the side.
As usual, grow the stack **downward**.

Below, "`func()` returns to `0x4444`" means that when control returns from `func()`, it starts executing at address `0x4444`.

2a Draw the stack layout when control comes to `s2` (i.e., "`badfile = fopen(..)`" is to be executed next).

Solution



Note
- increasing addresses: -1 to -2 pts
- not hex: -1 to -2 pts
- wrong sizes: -1 pts
- wrong order: -2 pts

2b Supply the contents of file `badfile` so that `func()` returns to `0x4444` with argument

Solution 16 (`buf`) + 4 (`j3`) + 4 (`ebp`) = 24. So `badfile` should have: 24 bytes of anything; 4 bytes `0x4444`

2c Supply the contents of file `badfile` so that `func()` returns to `0x4444` with argument `0xfbfbfbfb`.

Solution `badfile` should have: 24 bytes of anything; 4 bytes `0x4444`; 4 bytes `0xfbfbfbfb`.

2d Does your answer for part 2a change if Address Randomization is turned on. Explain briefly.

Solution

Stack layout: the contents are the same but the addresses can change

Badfile: the contents are the same; still returns to 0x4444 (although the code there may have changed).

2e Does your answer for part 2a change if the stack is made Non-Executable. Explain briefly.

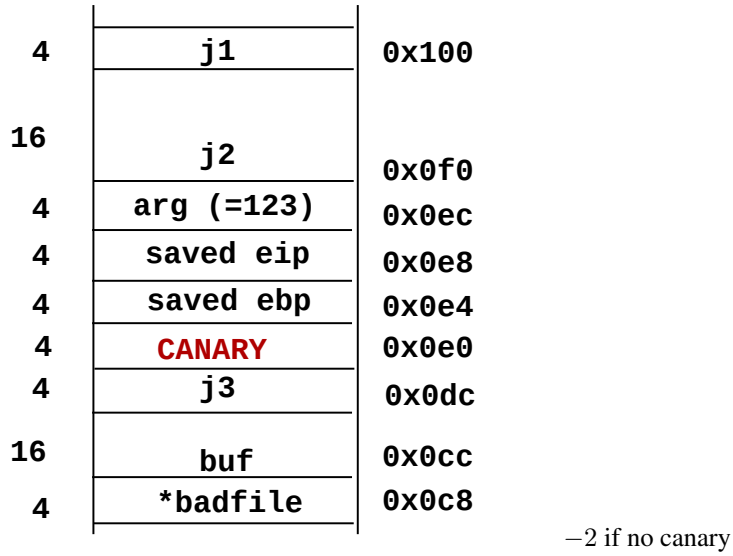
Solution

Stack layout: the contents and addresses are the same

Badfile: the contents are the same; still returns to 0x4444 (because nothing on stack is executed)

2f Now assume the program is compiled with Stack Guard. Draw the stack layout when control comes to s2.

Solution



−2 if no canary

Can file `badfile` be initialized so that `func()` returns to 0x4444. Explain briefly.

Solution

No. For the buffer overflow to modify the return address, it would also have to modify the canary.
This causes the progam to be aborted when the return is executed.

3 Here are two files owned by root:

- `/passwd.txt`: text file that contains user passwords. Root has read-write-execute access. All other users have no access.
- `/chpwd`: executable file that users can run to change their passwords. Root has read-write-execute access. All other users have write-execute access. The setuid bit is set (so it is a set-root-uid file).

Does this configuration allow an ordinary (i.e., non-root) user to delete `passwd.txt`?
If no, explain briefly.
If yes, briefly give the steps of the attack.

Solution

Yes.

- User develops a program, say `xx`, that deletes file `passwd.txt`, e.g.,
  - executable of C program along the following lines:

```
main() {
  remove(/passwd.txt)
}
```

  - bash script along the following lines:

```
#! /bin/bash
rm -f /passwd.txt
```

  - shellcode (to get a root shell from which `passwd.txt` can be deleted).
- User writes `xx` into `/chpwd` (can do this because it has write access).
- User executes `xx`. This deletes `/passwd.txt` because it executes with root privilege.

4 A website maintains an SQL table YY with a row for every user and columns NAME, PWD, AGE and others.

- To change its password, a user sends a GET request with path /chpw.php?a1 = <name>&a2 = <opwd>&a3 = <npwd>.
  The server looks for an entry in YY with NAME = <name> and PWD = <opwd>; if found, it sets PWD field to <npwd>.
- To get a user's age, a user sends a GET request with path
  /getage.php?a1 = <name>.
  The server looks for an entry in YY with NAME = <name>; if found, it returns the entry's AGE field.

The server does no additional checks on these operations.

Among the users are Ted and Bob (these are their NAME entries). Ted does not know Bob's PWD value.

4a Give the path of a GET request that Ted can issue in order to change Bob's password to fqr123.

Solution

Here is possible php code for handling the requests. (This is not required for the answer; it just adds some context.)

```
chpw.php:
  $db->sql_query("UPDATE YY SET PWD='npwd' WHERE (NAME='name' AND PWD='opwd');");

getage.php(name):
  $db->sql_query("SELECT AGE FROM YY WHERE (NAME='name');");
```

Possible attack paths ("--" denotes start of comment):

- /chpw.php?a1=bob')/*&opwd=*/--&npwd=fqr123
  // SQL: UPDATE YY SET PWD='fqr123' WHERE NAME='bob'-- ...
- /chpw.php?a1=bob');--&opwd=whocares&npwd=fqr123 // new: a much simpler solution than above
  // SQL: UPDATE YY SET PWD='fqr123' WHERE NAME='bob'-- ...
- getage.php?a1=bob;UPDATE+YY+SET+PWD=fqr123+WHERE+NAME='bob'
  // SQL: UPDATE YY SET PWD='fqr123' WHERE NAME="bob"

Errors in the previous solution to 4a:

- Instead of "chpw.php", the previous solution used "chpw.php(name, opwd, npwd)", which suggests a function call with parameters. But actually, name, opwd, and npwd are "global" variables.
- The SQL query uses single quotes and the attack http request used double quotes (whereas it should use single quotes).
- The SQL query uses matching parentheses (eg, WHERE (Name = 'name' AND PWD = 'opwd'). So the attack http request should have a matching ")", but it does not.

End of solution

4a Now assume the server uses php and the PREPARE construct. Give the server's code that handles a user request.

Solution

```
chpw.php(name, opwd, npwd):
  $stmt = $db->prepare("UPDATE YY SET PWD=? WHERE (NAME=? AND PWD=?);");
  $stmt->bind_param("sss", $npwd, $name, $opwd);
  $stmt->execute();

getage.php(name):
  $stmt = $db->prepare("SELECT AGE FROM YY WHERE (NAME=?)");
  $stmt->bind_param("s", $name);
  $stmt->execute();
```

End of solution

5 This problem concerns a browser `c1`, website `s1`, and attacker website `s2`.

- `c1` clicks `http://s1/p1.html`. In the response, `s1` sets a cookie for domain `s1`.
- Then `c1` clicks `http://s2/p2.html`.
- Then `p1.html` regularly issues POST requests to `s1`. Each POST request contains the cookie value in its data. The server treats a request as valid iff the cookie value (in the request header) matches the value in the data.

For each of the following cases, answer whether `p2.html` can send a POST request to `s1` that the latter treats as valid. Write "YES" if it can, and "NO" if it cannot. (Below, "unguessable" is equivalent to "randomly generated".)

Solution

s1-cookie name	s1-cookie value	your answer
guessable	guessable	YES
guessable	unguessable	NO
unguessable	guessable	YES
unguessable	unguessable	NO

If `s2` knows the cookie value, `p2.html` can have a form element that posts to `s1` with the cookie value in its data. Client `c1` includes the cookie in the header.

6 When `deposit()` is called with `user` pointing to a valid `User` instance, it should update the `user`'s `balance` or return `0` upon error. Does the above function achieve that. If not, fix the function.

```
struct User {
   unsigned int balance;
};

unsigned int deposit(struct User *user, unsigned int amount) {
   user->balance += amount;
   return user->balance;
}
```

Solution No. Integer overflow error if the input is `0xFFFFFFFF`.

Fix: before updating `balance`, check for overflow:
`if (user->balance + amount < user->balance) return 0;`

7 The following function is in a program owned by root and executable by any user. `ustr` points to a string supplied by the user. `secret` points to a string that is stored within the program and should not be disclosed to the user. The function prints a message indicating the result of `strcmp(ustr,secret)`. Can the function expose the `secret` string. If so, fix the function.

```
int f(char* ustr, char* secret) {
   printf(ustr);
   if (strcmp(ustr, secret) <= 0)
      printf(" <= secret");
   else
      printf(" > secret");
}
```

Solution Yes. `printf(ustr)` vulnerability: if `ustr` points to `"%s%s"`, then `*secret` is printed.

Fix: change `printf(ustr)` to `printf("%s", ustr)`.