**Introduction to Race Conditions and Timing Attacks**

Web applications can be vulnerable to various famous attacks such as SQL injection and Cross-Site Scripting. Additionally, web applications can suffer from vulnerabilities not exclusively present in a web context, such as race conditions and timing attacks. Due to a general lack of awareness among developers, these vulnerabilities can be particularly prevalent. Exploiting timing attacks and race conditions can lead to data exposure and loss and business logic bypass, depending on the implementation of the vulnerable web application.

**Timing Attacks**

Generally, [timing attacks](https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/10-Business_Logic_Testing/04-Test_for_Process_Timing) are [side-channel attacks](https://www.rambus.com/blogs/side-channel-attacks/) that exploit differences in computation or processing time in the vulnerable component. As a side-channel attack, timing attacks do not directly attack the core components of web applications, but measure response timing to infer (and therefore exfiltrate) potentially sensitive information. Most typical web vulnerabilities, such as SQL injection and Cross-Site Scripting, involve directly exploiting the web application components such as databases and front-ends. While blind exploitation of such attacks often involves timing (for example, during the exploitation of blind time-based SQL injection vulnerabilities), we will not consider these attacks here but instead focus on ones that result from errors in the business logic of a web application.

**Race Conditions**

Race conditions are vulnerabilities in programs that arise when the timing or sequence of specific actions can influence the outcome unexpectedly and undesirably. Multi-threaded programs are particularly susceptible to race conditions, given that predicting how the different threads affect each other's program flow can be difficult. Since exploiting race condition vulnerabilities may require precise timing, multiple attack attempts may be required before successful exploitation.

As an example, let us consider a classical race condition vulnerability known as [Time-of-check Time-of-use (TOCTOU)](https://cwe.mitre.org/data/definitions/367.html). TOCTOU vulnerabilities are common in filesystem operations and result from a difference in the time of check, i.e., the time when security conditions are checked, and the time of use, i.e., the time when the program actually uses the resource.

As an example, consider the following C-code that is part of a setuid program that reads the file variable as an argument:

Code: c

// access check

if (access(file, W\_OK)) {

return -1;

}

// open file

int fd = open(file, O\_WRONLY);

The call to access checks whether the calling user is allowed to access the specified file. The file is then subsequently opened and operated on. Since the time of check, the call to access, occurs before the time of use, i.e., the call to open, this is a classical TOCTOU vulnerability. To exploit it, we can call the program with a benign file such as /tmp/test and manipulate it after the time of check but before the time of use. We can do so by creating a symlink to a file we are unable to access, such as /etc/shadow:

Introduction to Race Conditions and Timing Attacks

yovecio@htb[/htb]$ rm /tmp/test && ln -s /etc/shadow /tmp/test

We need to get the timing right so that the symlink is created after the call to access and before the call to open. If we succeed, the program now operates on the file /etc/shadow although our user cannot access that file, and thus the access check would cause the program to exit. Since the timing is very precise, we might require multiple exploitation attempts.

Now that we know what race conditions are, let us discuss how they can arise in web applications. In web applications, race conditions typically arise when synchronous actions are assumed, but asynchronous actions are the reality. As an example, consider PHP web applications. PHP does not support any form of multithreading and is, as such, a single-threaded language. However, the situation is different if a web server such as Apache runs the PHP web application. That is because Apache (and other web servers) typically spawn multiple worker threads that run the web application simultaneously to allow for better performance. These cases allow for multi-threaded execution, although PHP itself is single-threaded. Settings like these can cause race condition vulnerabilities that web developers might be unaware of.