This case is in Cherokee HTTP Server (buffer.c and bogotime.c). It uses the read-write lock in function cherokee_bogotime_try_update to improve the performance. However, this implementation may cause some bugs.

I extract following program from the original source code.

```
1  ret_t cherokee_buffer_add (cherokee_buffer_t *buf, const char *txt, size_t size) {
2          int free = buf->size - buf->len;
3          if ((cuint_t) free < (size+1)) {
4                  if (unlikely (realloc_inc_bufsize(buf, size - free)) != ret_ok)
5                          return ret_nomem;
6          }
7          memcpy (buf->buf + buf->len, txt, size);
8          buf->len += size;
9          buf->buf[buf->len] = '\0';
10         return ret_ok;
11 }
12 void cherokee_buffer_clean (cherokee_buffer_t *buf) {
13         ……
14         buf->len = 0;
15 }
16 static ret_t update_guts (void) {
17         ……
18         newtime = time (NULL);
19         if (cherokee_bogonow_now >= newtime)
20                 return ret_ok;
21         cherokee_bogonow_now = newtime;
22         ……
23         cherokee_buffer_clean (&cherokee_bogonow_strgmt);
24         ……
25         cherokee_buffer_add (&cherokee_bogonow_strgmt, bufstr, szlen);
26         ……
27 }
28 ret_t cherokee_bogotime_try_update (void) {
29         …….
30         unlocked = CHEROKEE_RWLOCK_TRYREADER (&lock);
31         if (unlocked)
32                 return ret_not_found;
33         ret = update_guts();
34         CHEROKEE_RWLOCK_UNLOCK (&lock);
35         ……
36 }
```

Assume that two threads run into function cherokee_bogotime_try_update and get into the critical section in parallel. Although line 18-21 may break some threads in function update_guts, two threads still have chances to reach line 23 at the same time.

Two potential buggy cases are shown as following. I have already produced them by adding

some synchronizations.

(1) Break transaction operation

Thread 1		Thread2	
8	buf->len += size;		
		14	buf->len = 0;
9	buf->buf[buf->len] = '\0';		

The original intention in Thread 1 is that line 9 reads buf->len generated in line 8. However, line 14 may break this intention.

(2) Buffer overflow

Thread 1		Thread2	
14	buf->len = 0;	14	buf->len = 0;
	……	….	
		3	if ((cuint_t) free < (size+1)) {…}
8	buf->len += size;		
		7	memcpy (buf->buf + buf->len, txt, size);

In Thread 2, line 3 determines if the buffer is enough. If the buffer is enough, it does not allocate extra memory. But line 8 in Thread 1 adds buf->len so that (buf->len+size) may be larger than buf->size in line 9 of Thread 2. That may cause buffer overflow.