Cookie 崩溃分析报告

THINCT

December 11, 2023

构建崩溃现场

什么是 cookie 崩溃

分析方法

思考

崩溃复现

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   void func0(char buf[])
5   {
6       for (unsigned int i = 0; i < 300; i++)
7       {
8           buf[i] = i % 256;
9       }
10  }
11
12  void func1(int* pArr)
13  {
14      char buf[32]{0};
15      printf("%x\n", *pArr);
16      func0(buf);
17  }
18
19  int main(int argc, char* argv[])
20  {
21      int* pArr = new int(0x123);
22      func1(pArr);
23
24      printf("%x\n", *pArr);
25
26      system("pause");
27      return 0;
28  }
```

Figure 1: 源码构造浮现过程

崩溃复现

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void func0(char buf[])
5  {
6      for (unsigned int i = 0; i < 300; i++)
7      {
8          buf[i] = i % 256;
9      }
10 }
11
12 void func1(int* pArr)
13 {
14     char buf[32]{0};
15     func0(buf);
16 }
17
18 int main(int argc, char* argv[])
19 {
20     int* pArr = new int(0x123);
21     func1(pArr);
22
23     printf("%x\n", *pArr);
24
25     system("pause");
26     return 0;
27 }
```

```
207         __fastfail(
208     }
209     #pragma warning
210
211 #else
212
213 #if _CONTROL_FLOW_G
214     __declspec(guar
215 #endif
216     __declspec(nore
217     {
218         if (IsProce
219         {
220             __fastf
221         }

volatile UI

// Set up a
// We can't
// exceptio
// exceptio
// __securi
// buffer c

#if defined
// On x86,
// preserve
// the buff
```

未经处理的异常

0x00BA1449 处有未经处理的异常(在 CookieCrashDemo.exe 中): 堆栈 Cookie 检测代码检测到基于堆栈的缓冲区溢出。

显示调用堆栈 | 复制详细信息 | 启动 Live Share 会话...

异常设置

Figure 2: 崩溃结果

构建崩溃现场

什么是 cookie 崩溃

分析方法

思考

cookie 校验

堆栈（Stack）和堆（Heap）的 Cookie 安全检查是一种用于防止缓冲区溢出攻击的技术。这种技术的目的是通过引入一种额外的随机值（通常称为 Cookie）来增加攻击者预测攻击的难度，从而提高系统的安全性。

堆栈 Cookie 安全检查：在堆栈 Cookie 安全检查中，通常是在函数的栈帧中添加一个随机生成的值，称为堆栈 Cookie 或 Canary。这个 Cookie 位于函数局部变量和返回地址之间，当函数返回时，会检查这个 Cookie 是否被破坏。

攻击者如果试图通过溢出缓冲区来修改返回地址，他们也必须修改堆栈 Cookie。由于 Cookie 的值是不可预测的，攻击者需要事先知道这个 Cookie 的值才能成功执行攻击。因此，堆栈 Cookie 提供了一种防范缓冲区溢出攻击的机制。

堆 Cookie 安全检查：堆 Cookie 安全检查与堆栈 Cookie 类似，但是是在动态内存分配的堆上进行的。在动态内存分配时，为分配的内存块添加了一个额外的 Cookie，并将这个 Cookie 存储在内存块的头部或尾部。当释放内存时，系统会检查 Cookie 是否被破坏，从而检测出潜在的堆溢出。

实现原理：添加 Cookie：

在函数调用或内存分配时，系统生成一个随机的 Cookie 值，并将其添加到相关数据结构中（堆栈帧或动态内存块）。检查 Cookie：

在函数返回或内存释放时，系统会检查 Cookie 是否被破坏。如果 Cookie 被破坏，系统就会认为发生了缓冲区溢出，从而触发相应的安全措施。增加攻击者难度：

由于 Cookie 是随机生成的，攻击者必须在进行攻击时预测正确的 Cookie 值，增加了攻击的难度。

cookie 校验

```
void func1(int* pArr)
{
002610C0  push        ebp
002610C1  mov         ebp,esp
002610C3  sub         esp,24h
002610C6  mov         eax,dword ptr [__security_cookie (0263004h)]
002610CB  xor         eax,ebp
002610CD  mov         dword ptr [ebp-4],eax    已用时间 <= 2ms
    char buf[32]{0};
002610D0  xor
002610D2  mov
002610D5  mov
002610D8  mov
002610DB  mov
002610DE  mov
```

寄存器

```
EAX = B247866C EBX = 010EB000 ECX = 01829D68 EDX = 01829D68
   ESI = 0182B458 EDI = 0182BD70 EIP = 002610CD
  ESP = 012FF788 EBP = 012FF7AC EFL = 00000286

0x012FF7A8 = 00000004
```

Figure 3: cookie 起点

cookie 校验

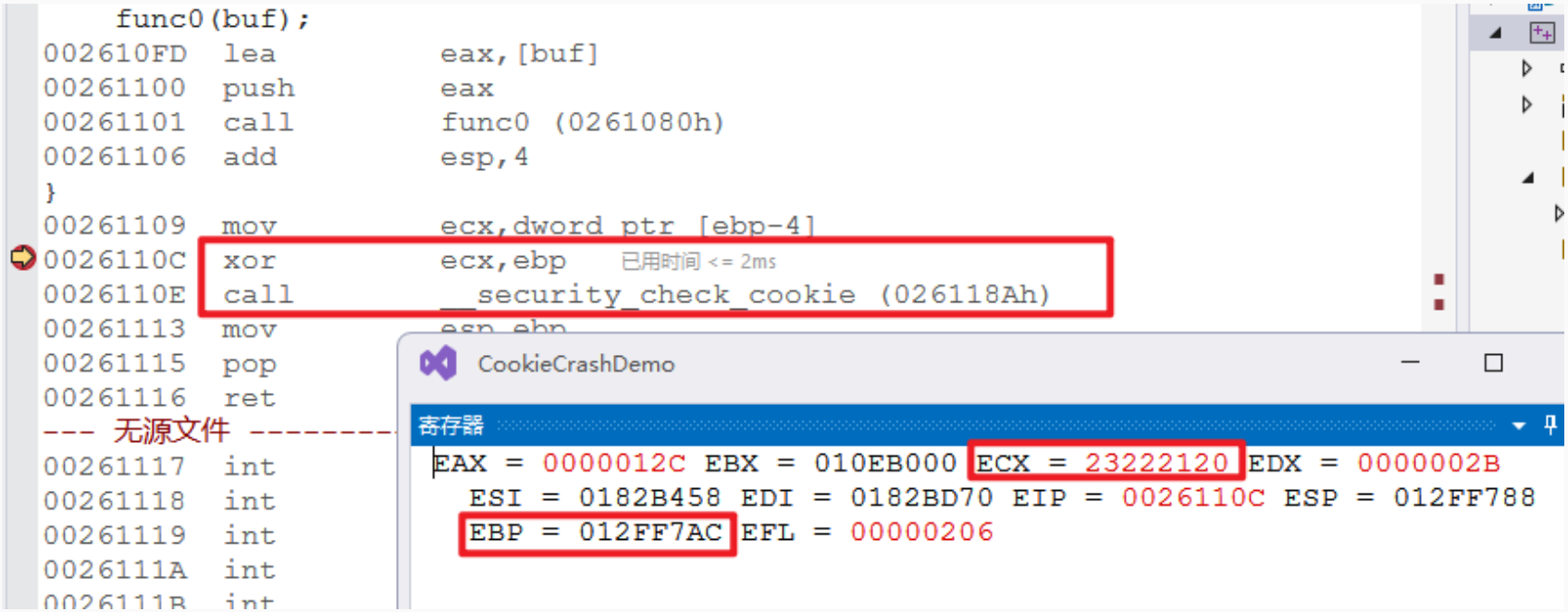

Figure 4: cookie 终点

结论:

参与 cookie 相关的异或运算的两个值， ebp 没有变化, 另外 eax 和 ecx 在正常情况下应该是相同的, 如果不相等, 则说明堆栈出现破坏了.

构建崩溃现场

什么是 cookie 崩溃

分析方法

思考

观察存放 cookie 抑或结果的内存

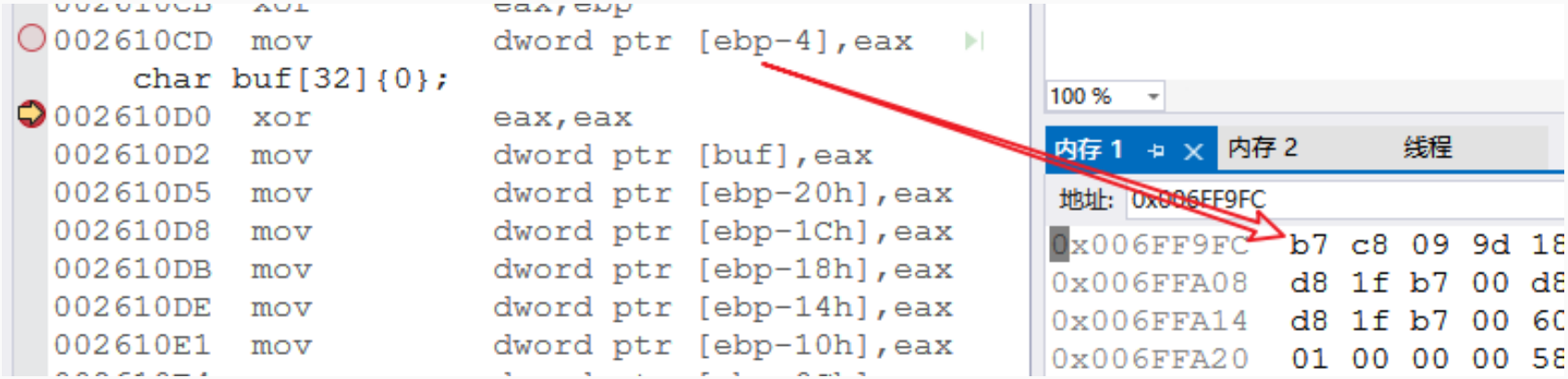

Figure 5: 记住 ebp-4 所指向的内存的值

观察存放 cookie 抑或结果的内存

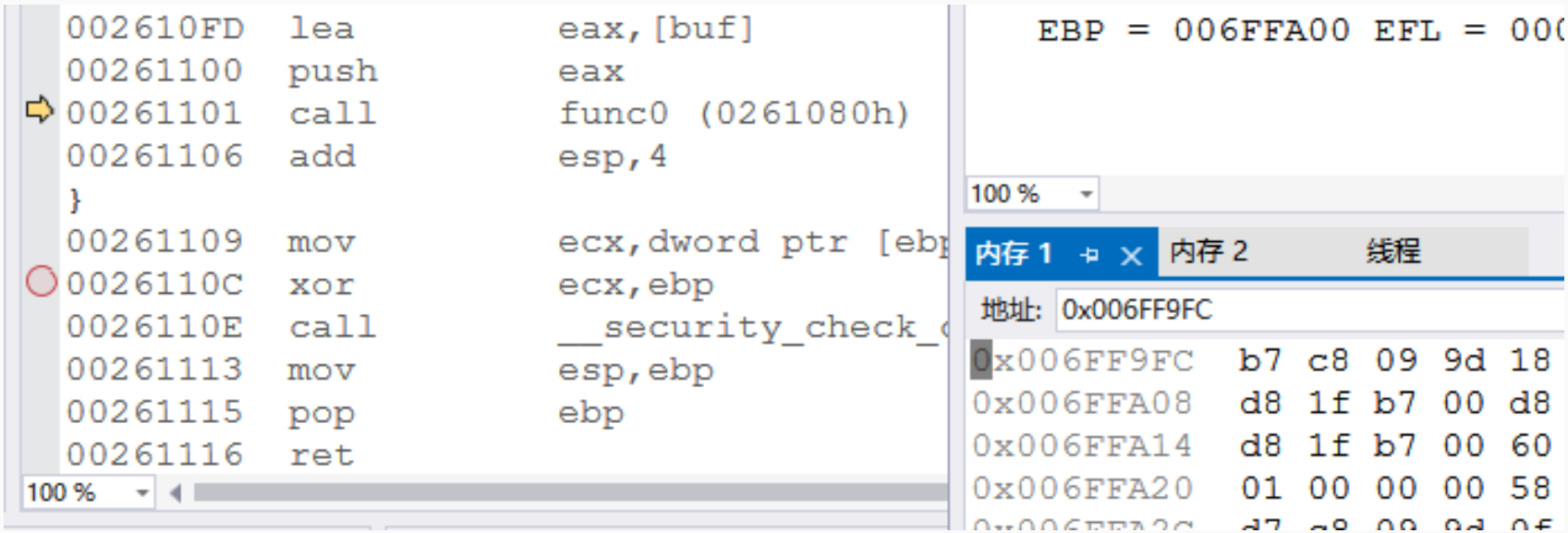

Figure 6: 继续记住 ebp-4 所指向的内存的值

观察存放 cookie 抑或结果的内存

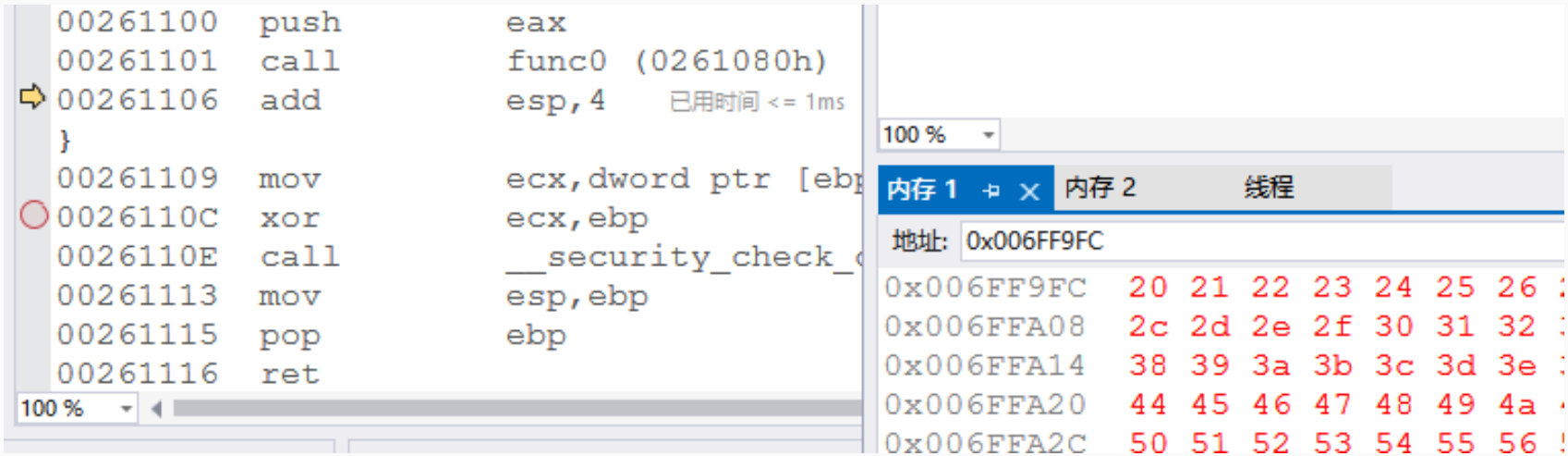

Figure 7: 直到发生变化

观察存放 cookie 抑或结果的内存

总结:

只要这个哨兵位置的内存发生变化, 就是发生了堆栈溢出.

构建崩溃现场

什么是 cookie 崩溃

分析方法

思考

你听懂了吗?

调试类似堆栈破坏的难点在哪里呢?