软件安全与漏洞分析

2.3 其他溢出漏洞

Previously in Software Security

- 堆的基本构成
- 堆的维护原理
- 堆溢出造成危害的主要方式

其他溢出漏洞

- 本节主题 -- 1. 整数溢出
  - 数值计算的基本原理
  - 整数溢出及其可能的后果
- 本节主题 -- 2. 格式化字符串漏洞
  - 类printf函数簇实现原理
  - 格式化字符串攻击原理及潜在后果

整数溢出

- 整数的表示
  - 类型：short（16 bit）、int（32 bit） 、long（64 bit）
  - 带符号($-2^{n-1} \sim 2^{n-1}-1$) / 无符号($0 \sim 2^{n}-1$)

	signed	unsigned
short	-32768 ~ 32767	0 ~ 65535
int	-2147483648 ~ 2147483647	0 ~ 4294967295
long	-9223372036854775808 ~ 9223372036854775807	0 ~ 18446744073709551615

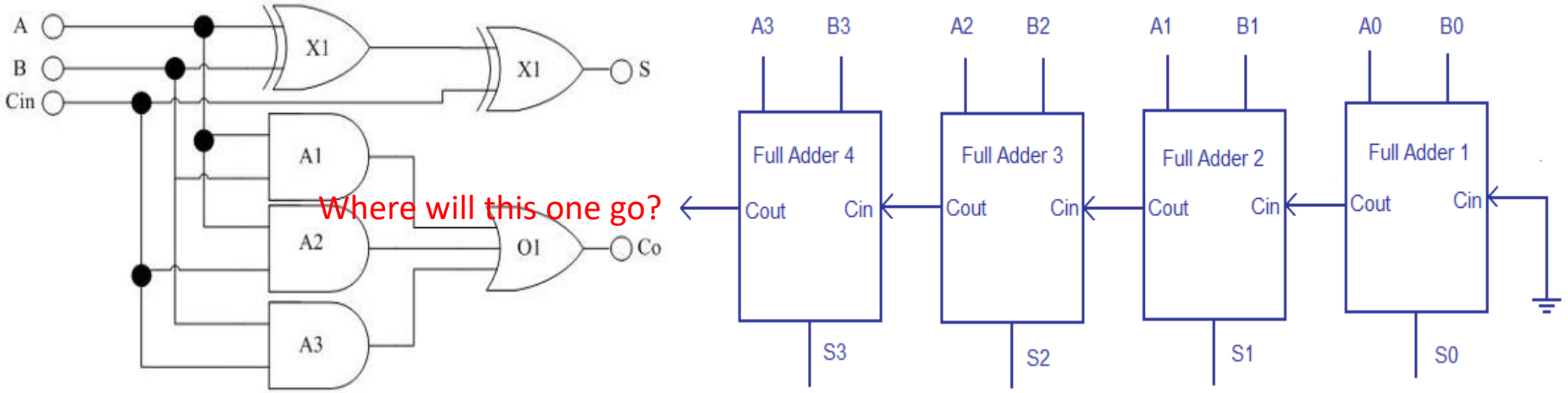
整数溢出

- 带符号整数格式中的补码
  - 带符号整数**统一用补码表示**（而非仅限负数数值）
  - 意义：符号位和数值域统一，加法和减法统一

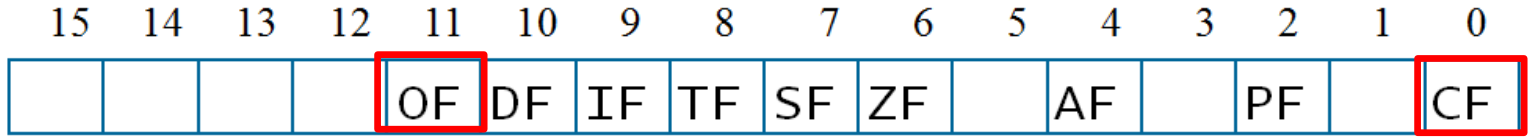
原码表示	反码表示	补码表示
规则：（不解释）	规则：正数不变，**负数按位取反**	规则：正数不变，**负数按位取反再+1**
比较大小需要硬件支持以区分符号位	无需硬件支持整数间的比较	整数间比较同前，且0只有一种表示
0有两种表示	0有两种表示	由于上一条，因此可以额外表示-8

整数溢出

- 整数的加减法

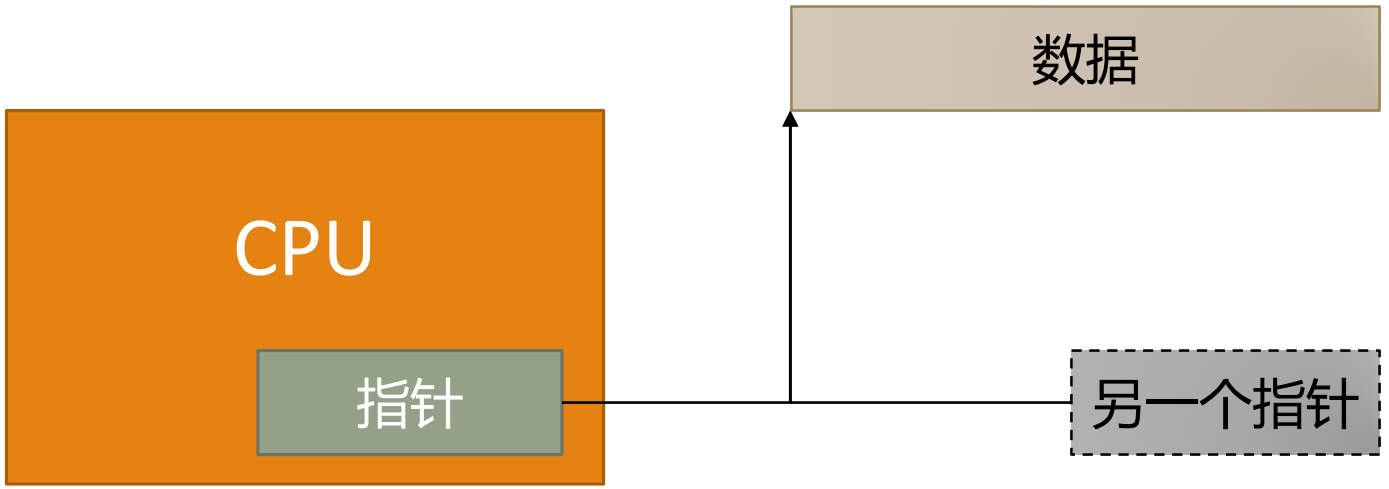


- CPU的FLAG寄存器



整数溢出

- 然而：
  - 检查FLAG这件事**常常被忽略**
  - 对数据的使用还存在如下情况



指针类型不同
=
数据被解读的意义不同
=
存在**损失精度**的风险

整数溢出

- 例如：

```
3.  int  main(int  argc,  char  *argv[])
4.  {
5.          unsigned  short  s;
6.          int  i;
7.          char  buf[100];
8.          i  =  atoi(argv[1]);
9.          s  =  i;
10.         if(s  >=  100)
11. {
12.                 printf("拷贝字节数太大，请退出!/n");
13.                 return  -1;
14.         }
15.         memcpy(buf,  argv[2],  i);
16.         buf[i]  =  '/0';
17.         printf("成功拷贝%d个字节/n",  i);
18.         return  0;
19. }
```

令argv[1] = 65536（1 00000000 00000000 B）

int to unsigned short

实际结果：拷贝了65536个字节

整数溢出

- 此外：
  - 对unsigned整型溢出的C语言规范 --- “以$2^{(8*sizeof(type))}$作模运算”
  - 对signed整型溢出的C语言规范 --- “undefined behavior ”

```
1  unsigned char x = 0xff;
2  printf("%dn", ++x);
```

++x == 0

```
1  signed char x =0x7f;
2  printf("%dn", ++x);
```

++x == -128

整数溢出

- 例如：

```
1  ... ...
2  ... ...
3  short len = 0;
4  ... ...
5  while(len< MAX_LEN) {
6  len += readFromInput(fd, buf);
7  buf += len;
8  }
```

若设定为32767

同时，假设此处函数返回值总是为2，那么……

实际结果：当len==32766时，

line 6 造成len的值变为-32768，

程序**死循环**

整数溢出

- 又如：abs(-2147483648) < 0
  - 函数abs的功能 --- 对于输入参数为正，返回其本身，否则返回其相反数
  - 然而，对于-2147483648（int类型的最大负数值），**函数abs返回的是其本身**

整数溢出

- 可见，整数溢出至少可能产生以下一些后果
  - 产生逻辑谬误，造成程序在执行中卡死或者出错
  - 为后续的缓冲区溢出充当引信

```
01.  nresp = packet_get_int();
02.
03.  if (nresp > 0) {
04.
05.  response = xmalloc(nresp*sizeof(char*));
06.
07.  for (i = 0; i < nresp; i++)
08.
09.  response[i] = packet_get_string(NULL);
10.
11.  }
```

令等于1073741825（0x 4000 0001）之

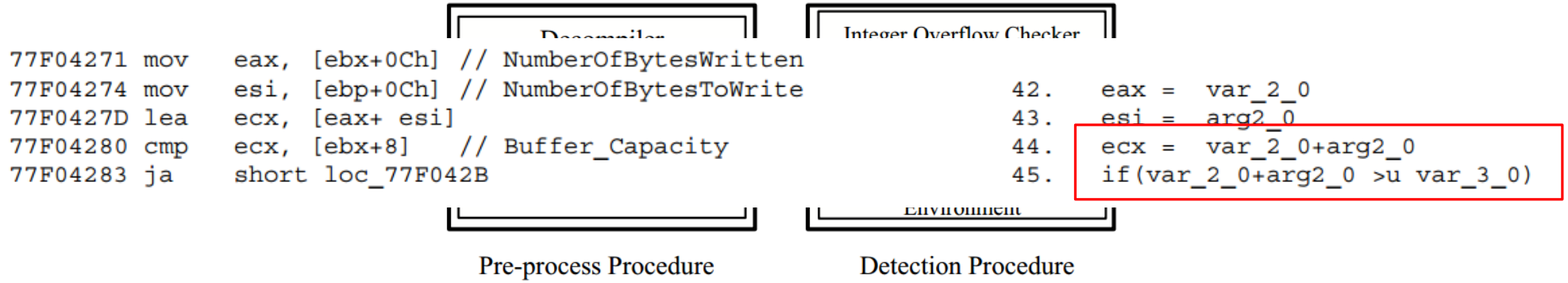
则此处计算得到 0x 1 0000 0004，溢出，被认为是4

参考“OpenSSH Challenge-Response SKEY/BSD_AUTH 远程缓冲区溢出漏洞”

```
3.  int  main(int  argc,  char  *argv[])
4.  {
5.          unsigned  short  s;
6.          int  i;
7.          char  buf[100];
8.          i  =  atoi(argv[1]);
9.          s  =  i;
10.         if(s  >=  100)
11. {
12.                 printf("拷贝字节数太大，请退出!/n");
13.                 return  -1;
14.         }
15.         memcpy(buf,  argv[2],  i);
16.         buf[i]  =  '/0';
17.         printf("成功拷贝了%d个字节/n",i);
18.         return  0;
19. }
```

一些整数溢出防护技术

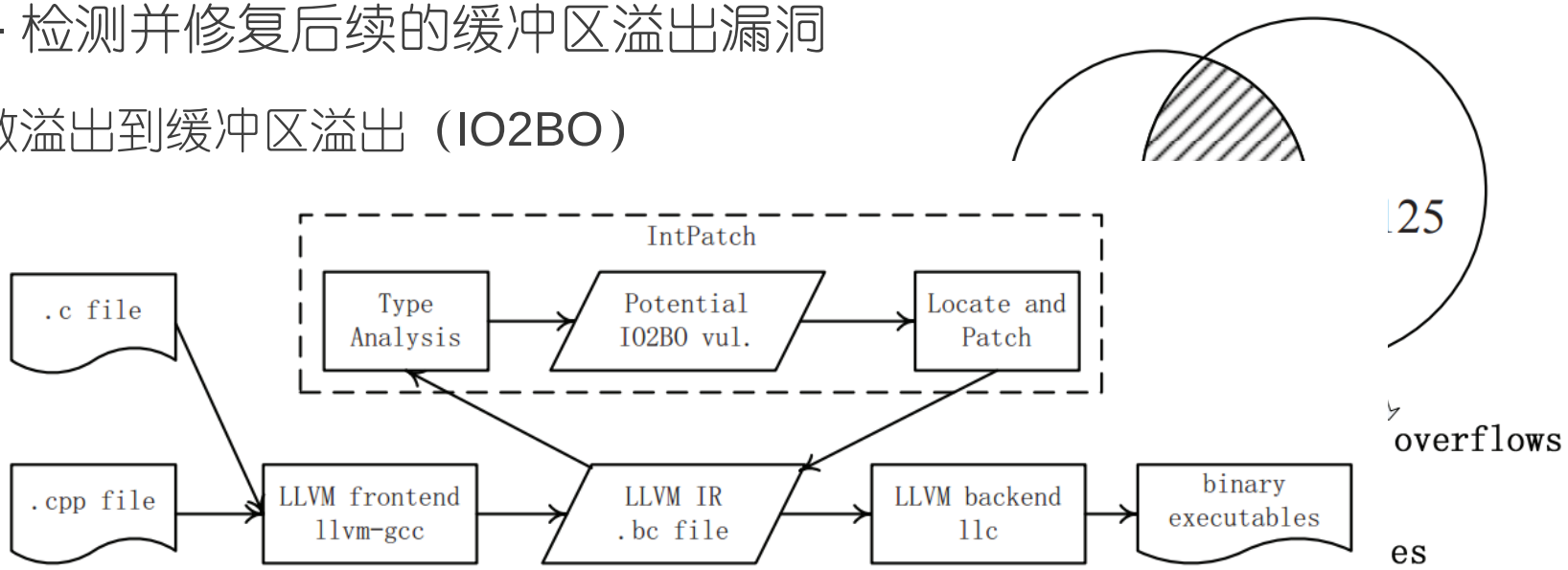
- IntScope --- 检测整数溢出本身
  - 利用符号执行、动态污点跟踪等分析运行时数据流，寻找异常



  - 参考文献：Wang T, Wei T, Lin Z, et al. IntScope: Automatically Detecting Integer Overflow Vulnerability in X86 Binary Using Symbolic Execution[C]//NDSS. 2009.
  - 参考文献：Sidiroglou-Douskos S, Lahtinen E, Rittenhouse N, et al. Targeted automatic integer overflow discovery using goal-directed conditional branch enforcement[C]//ACM SIGPLAN Notices. ACM, 2015, 50(4): 473-486.

一些整数溢出防护技术

- IntPatch --- 检测并修复后续的缓冲区溢出漏洞
  - 针对：整数溢出到缓冲区溢出（IO2BO）

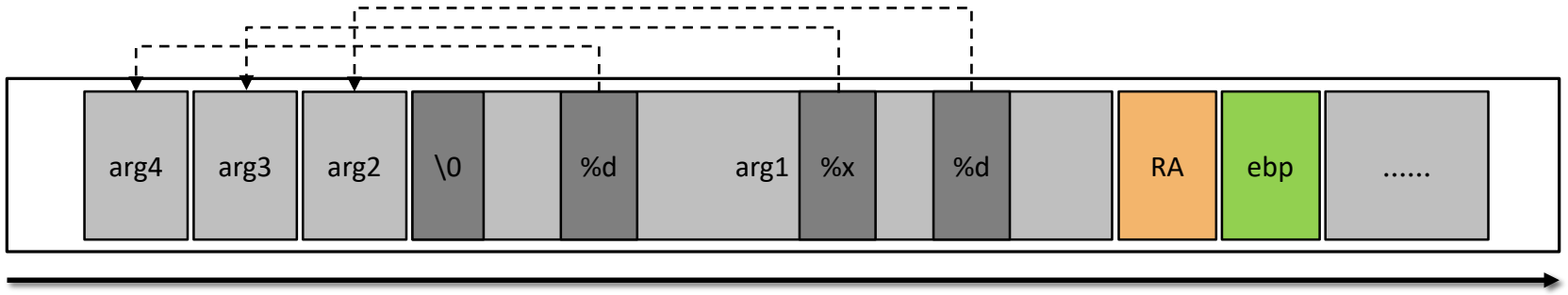


  - 参考文献：Zhang C, Wang T, Wei T, et al. IntPatch: Automatically fix integer-overflow-to-buffer-overflow vulnerability at compile-time[C]//European Symposium on Research in Computer Security. Springer Berlin Heidelberg, 2010: 71-86.

格式化字符串漏洞

- 类printf函数簇实现原理
  - 例：printf("test: i=**%d**; **%x**; j=**%d**;/n", **i**, **0xabcd**, **j**);
  - 特点：**在函数定义时，函数实参的数目和类型均未知**
  - 解决方式：**使用省略号指定参数表**

如printf 的原型（位于stdio.h中）为**extern int** printf(**const char** *format,...);



格式化字符串漏洞

- 类printf函数簇实现原理

字符	对应数据类型	含义
d / i	int	接受整数值并将它表示为有符号的十进制整数，i是老式写法
u	unsigned int	无符号10进制整数
x / X	unsigned int	无符号16进制整数，x对应的是abcdef，X对应的是ABCDEF（不输出前缀0x)
f(lf)	float(double)	单精度浮点数用f,双精度浮点数用lf(尤其scanf不能混用)
e / E	double	科学计数法表示的数，此处"e"的大小写代表在输出时用的“e”的大小写
c	char	字符型。可以把输入的数字按照ASCII码相应转换为对应的字符
s / S	char * / wchar_t *	字符串。输出字符串中的字符直至字符串中的空字符（字符串以'\0‘结尾，这个'\0'即空字符)
p	void *	以16进制形式输出指针
n	int *	到此字符之前为止，一共输出的字符个数，不输出文本
%	无输入	不进行转换，输出字符‘%’（百分号）本身

格式化字符串漏洞

- 类printf函数簇实现原理

字符	说明
-	左对齐，右边填充空格(默认右对齐)
+	在数字前增加符号 + 或 -
0	将输出的前面补上0，直到占满指定列宽为止（不可以搭配使用“-”）
	输出值为正时加上空格，为负时加上负号
#	type是o、x、X时，增加前缀0、0x、0X type是e、E、f、g、G时，一定使用小数点 type是g、G时，尾部的0保留

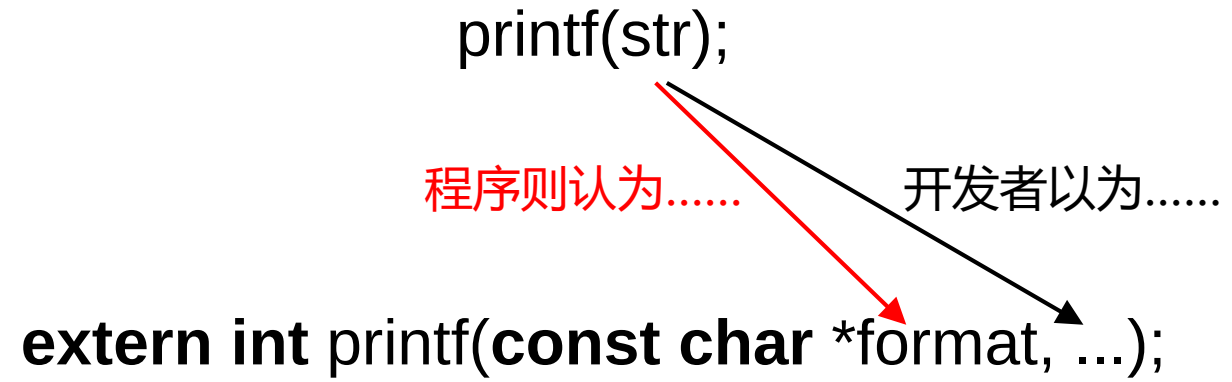
格式化字符串漏洞

- 类printf函数簇实现原理

符号	意义	符号	意义
\a	铃声(提醒)	\b	Backspace
\f	换页	\n	换行
\r	回车	\t	水平制表符
\v	垂直制表符	\’	单引号
\”	双引号	\\	反斜杠
\?	文本问号	\ ooo (例如\024)	ASCII字符(OCX)
\xhh (例如:\x20)	ASCII字符(HEX)	\xhhhh	宽字符(2字节HEX)

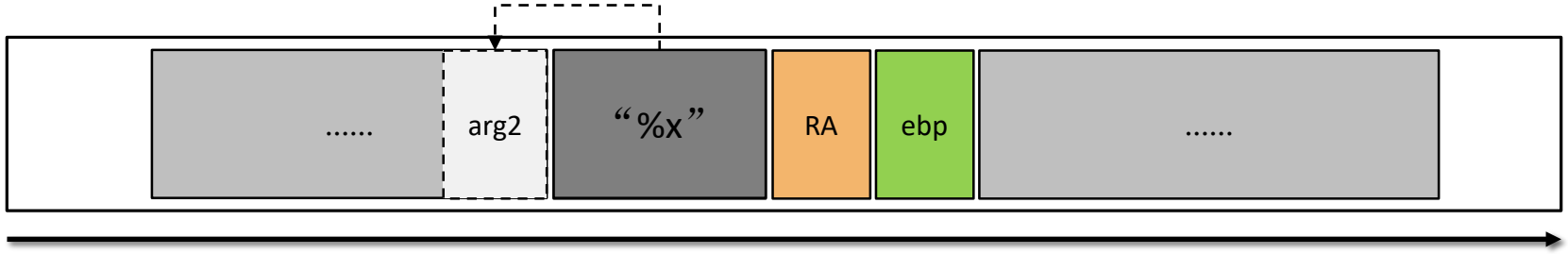
格式化字符串漏洞

- 类printf函数簇的第一个可利用点：**参数不匹配时难以发现**
  - printf()是一个参数长度可变函数，**仅仅看参数数量无法发现问题**
  - 为了查出不匹配，编译器需要了解printf()的运行机制，然而**编译器通常不做这类分析**
  - 若**格式字符串在程序运行期间生成**(如用户输入)，则编译器无法发现不匹配



格式化字符串漏洞

- 于是，如果让str="%x"，则会发生什么？
  - %x是printf规定的一种输出类型，unsigned int，输出无符号16进制数

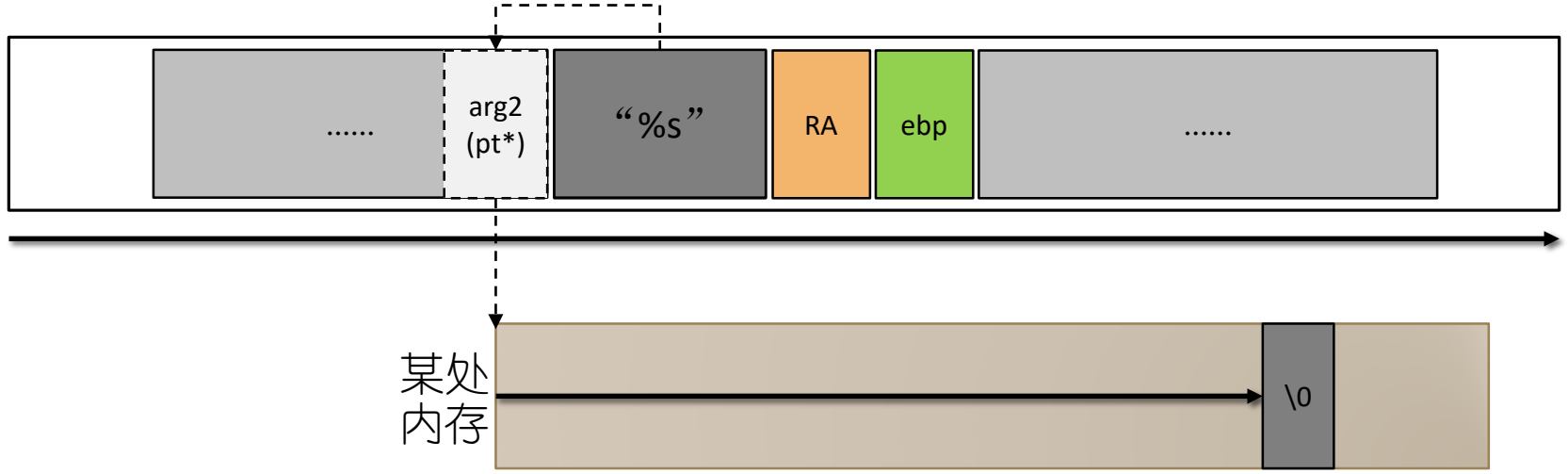


  - 结果：通过构造异常字符串，可以实现**对程序栈结构的任意读取**

注：格式化字符串中，输出"%"本身需要由连续的%%来表示

格式化字符串漏洞

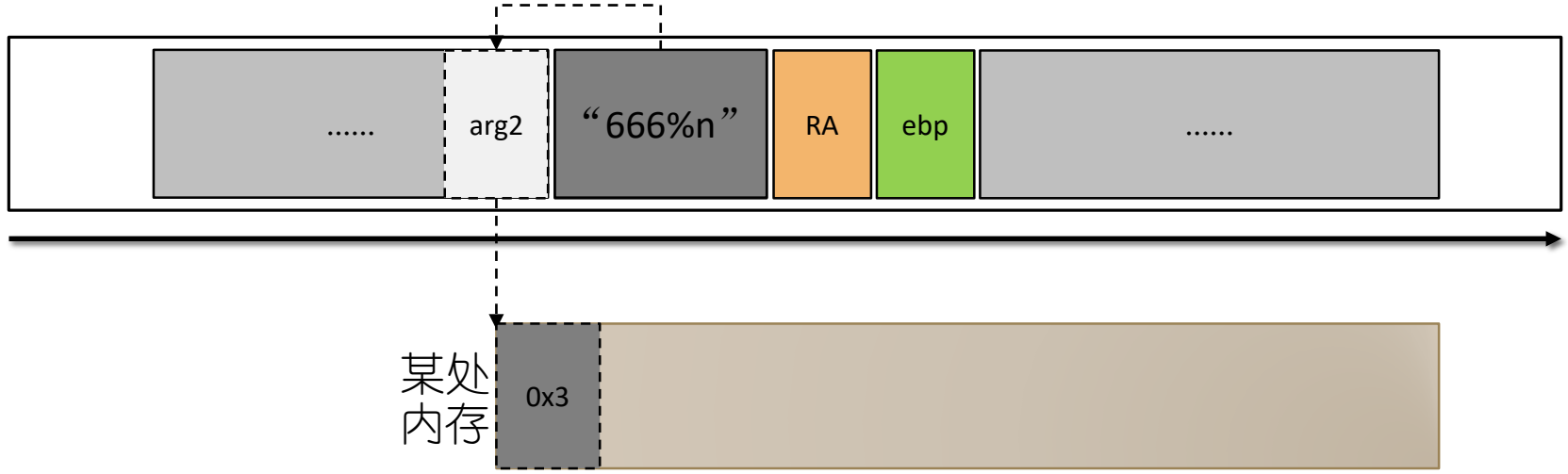
- 更进一步：
  - %s，printf规定的另一种输出类型，**char*/wchar_t***，输出字符串



  - 结果：通过构造异常字符串，还有可能实现**对任意内存的（大面积）读取**

格式化字符串漏洞

- 再进一步：
  - %n，特殊printf输出类型，int*，将此前已输出的串长度**写入指针所指位置**



  - 结果：通过构造异常字符串，可以实现**对任意内存的任意改写**

What’s next?

- 其他类似溢出型的漏洞
  - SQL注入
  - 数组越界