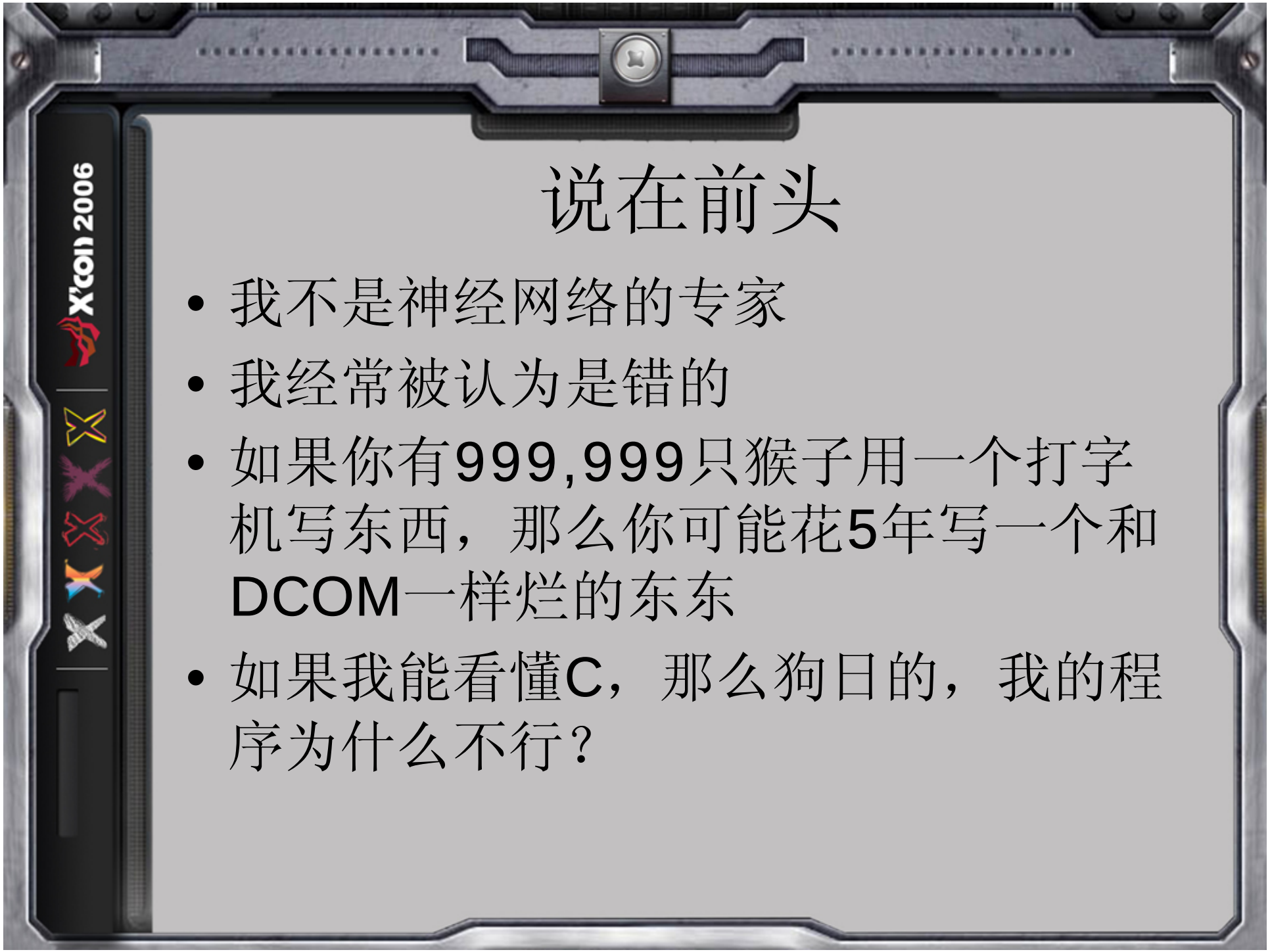
Nhacker: 黑客的神经中枢

如何使用神经网络来增加/减少安全

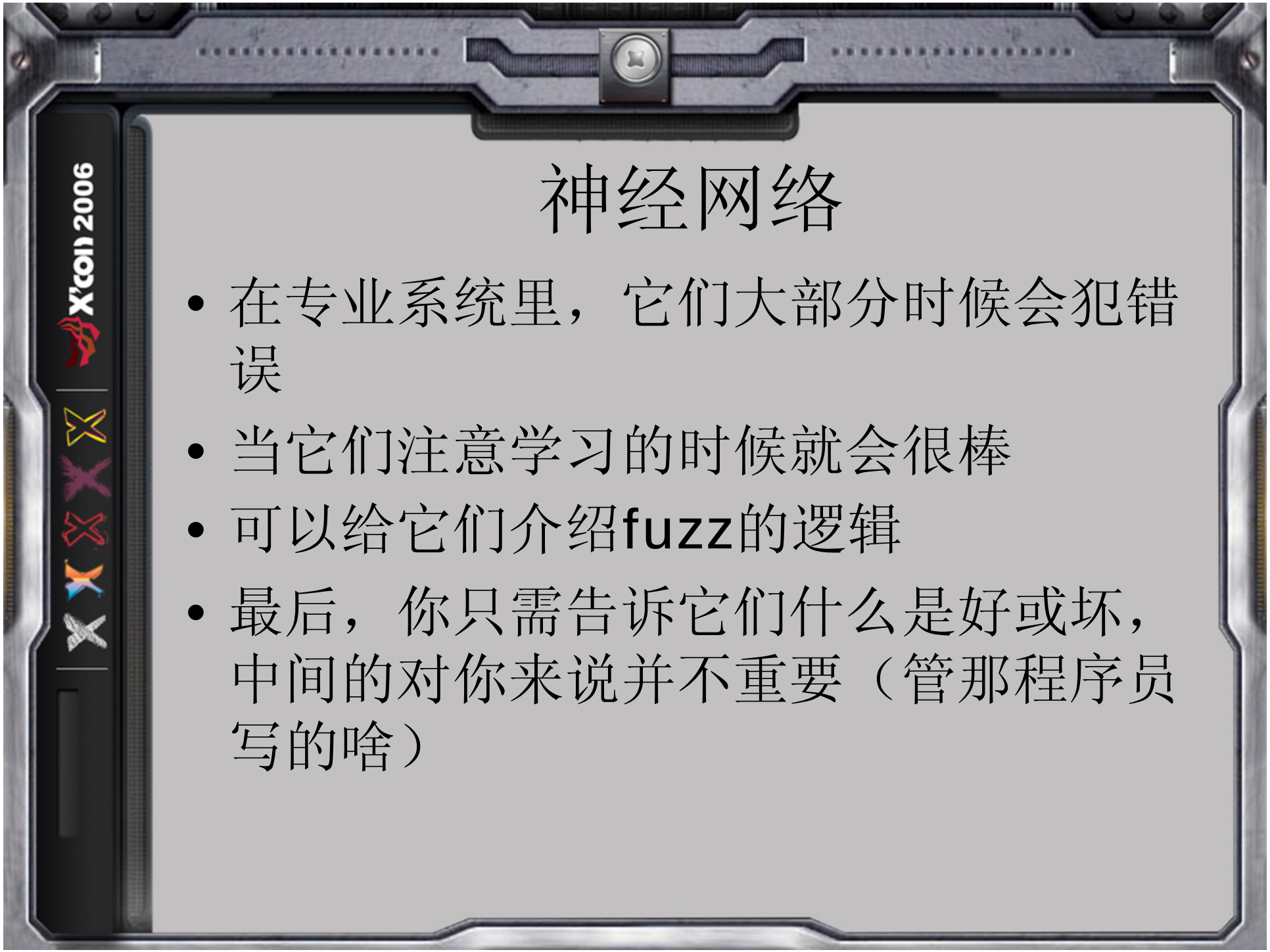
Enrique Alfonso Sanchez Montellano
enrique.sanchez@yaguarete-sec.com

X'Con2006



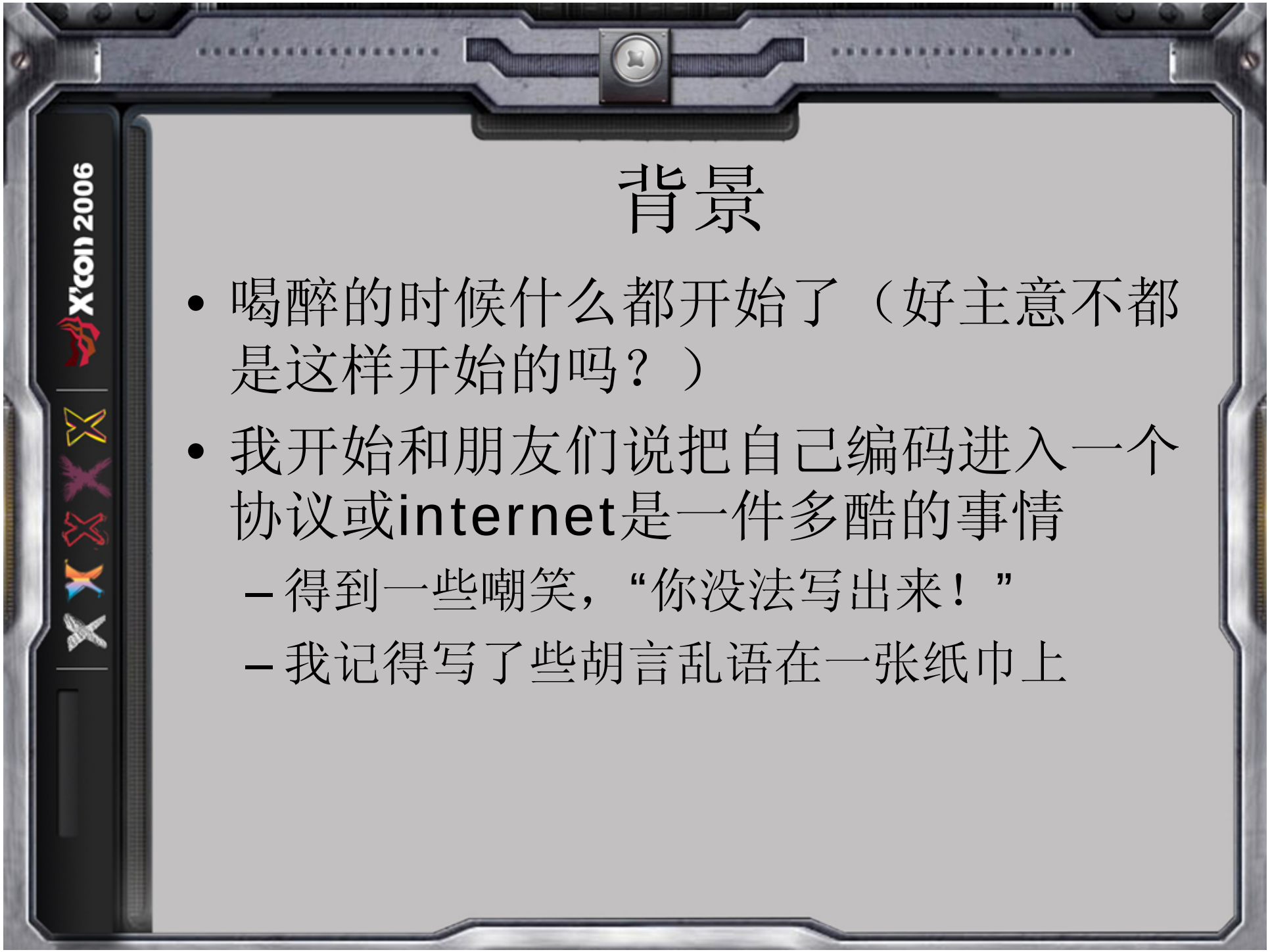
说在前头

- 我不是神经网络的专家
- 我经常被认为是错的
- 如果你有999,999只猴子用一个打字机写东西，那么你可能花5年写一个和DCOM一样烂的东东
- 如果我能看懂C，那么狗日的，我的程序为什么不行？



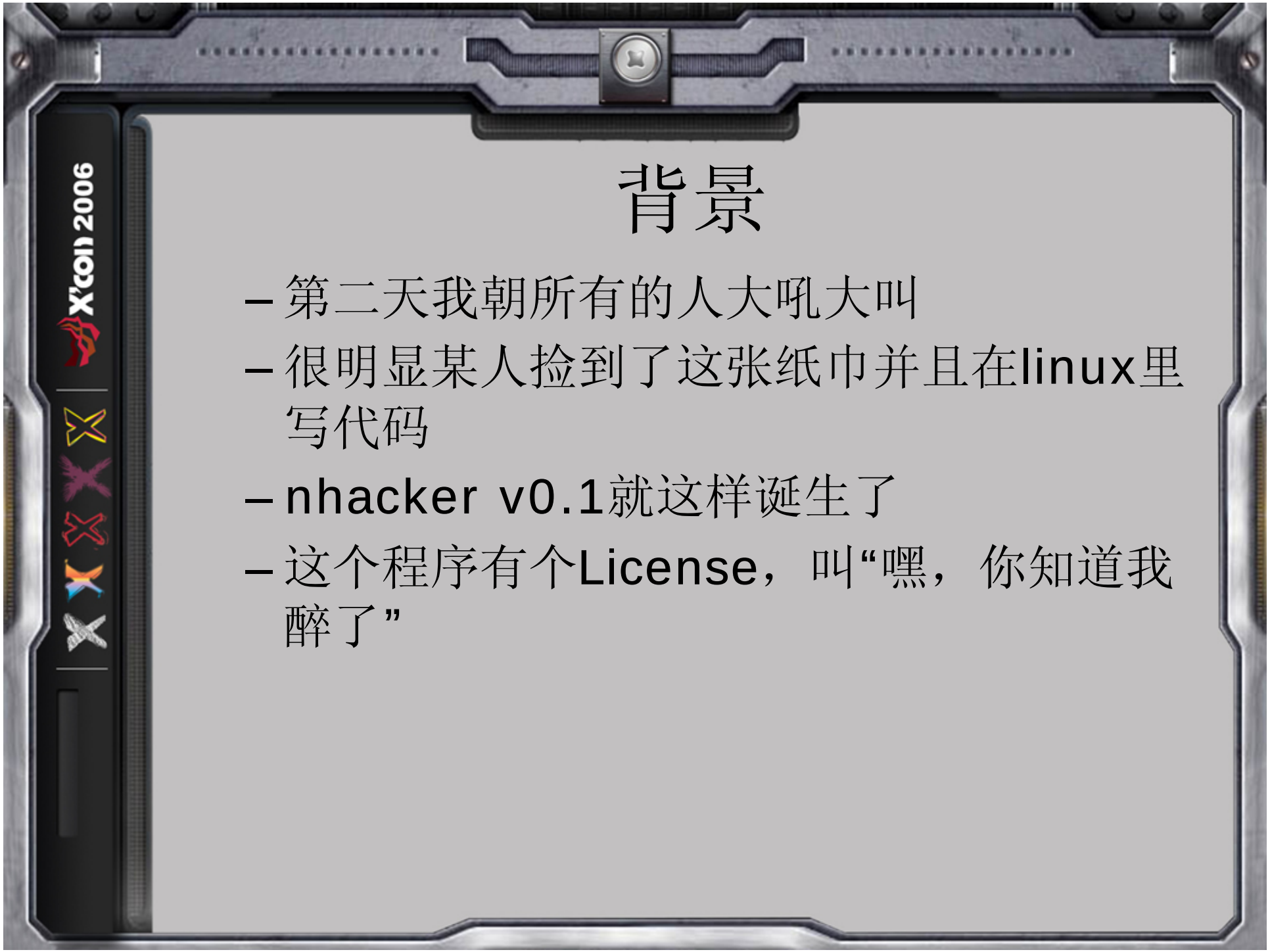
神经网络

- 在专业系统里，它们大部分时候会犯错误
- 当它们注意学习的时候就会很棒
- 可以给它们介绍fuzz的逻辑
- 最后，你只需告诉它们什么是好或坏，中间的对你来说并不重要（管那程序员写的啥）



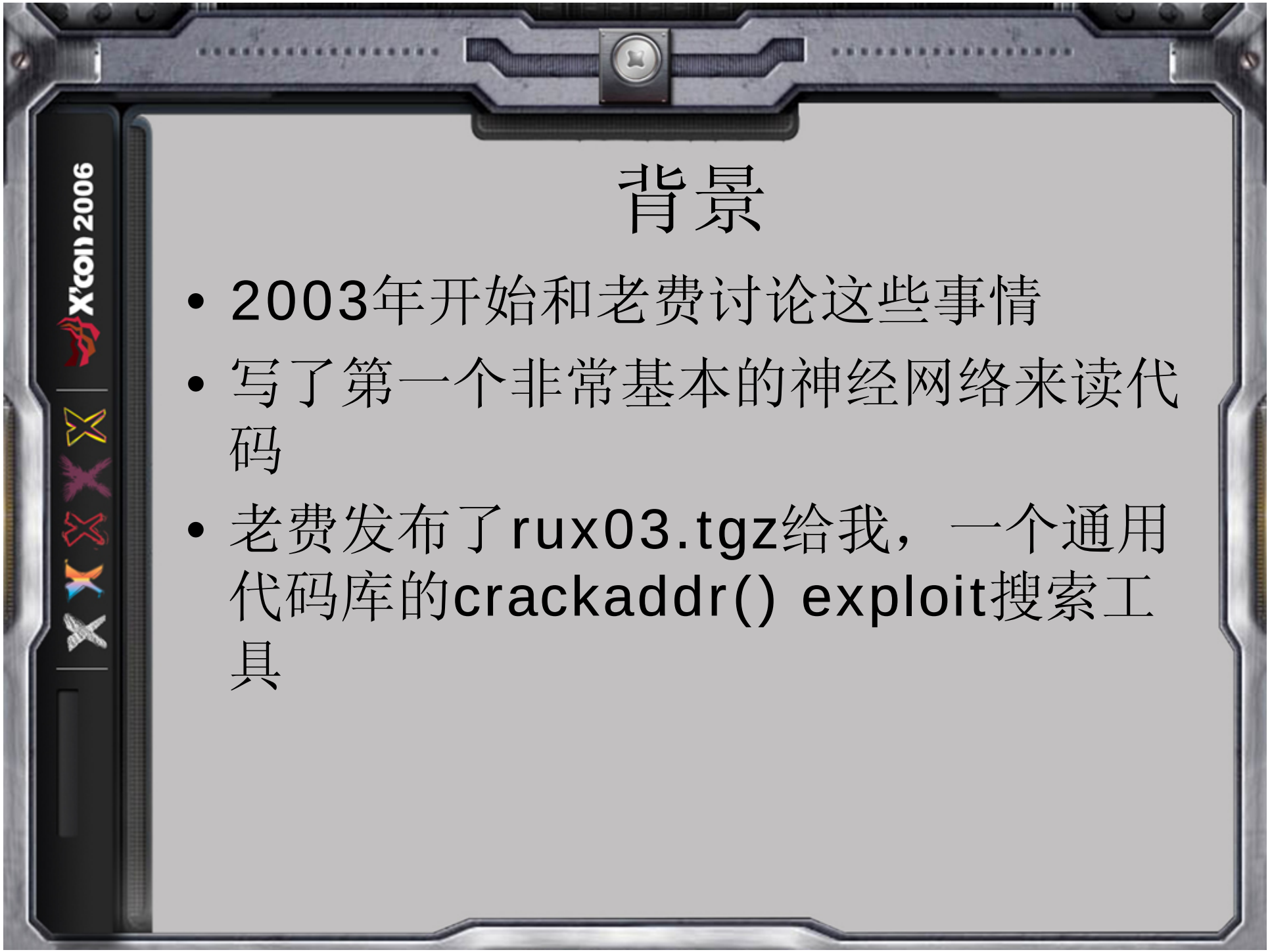
背景

- 喝醉的时候什么都开始了（好主意不都是这样开始的吗？）
- 我开始和朋友们说把自己编码进入一个协议或internet是一件多酷的事情
  - 得到一些嘲笑，“你没法写出来！”
  - 我记得写了些胡言乱语在一张纸巾上



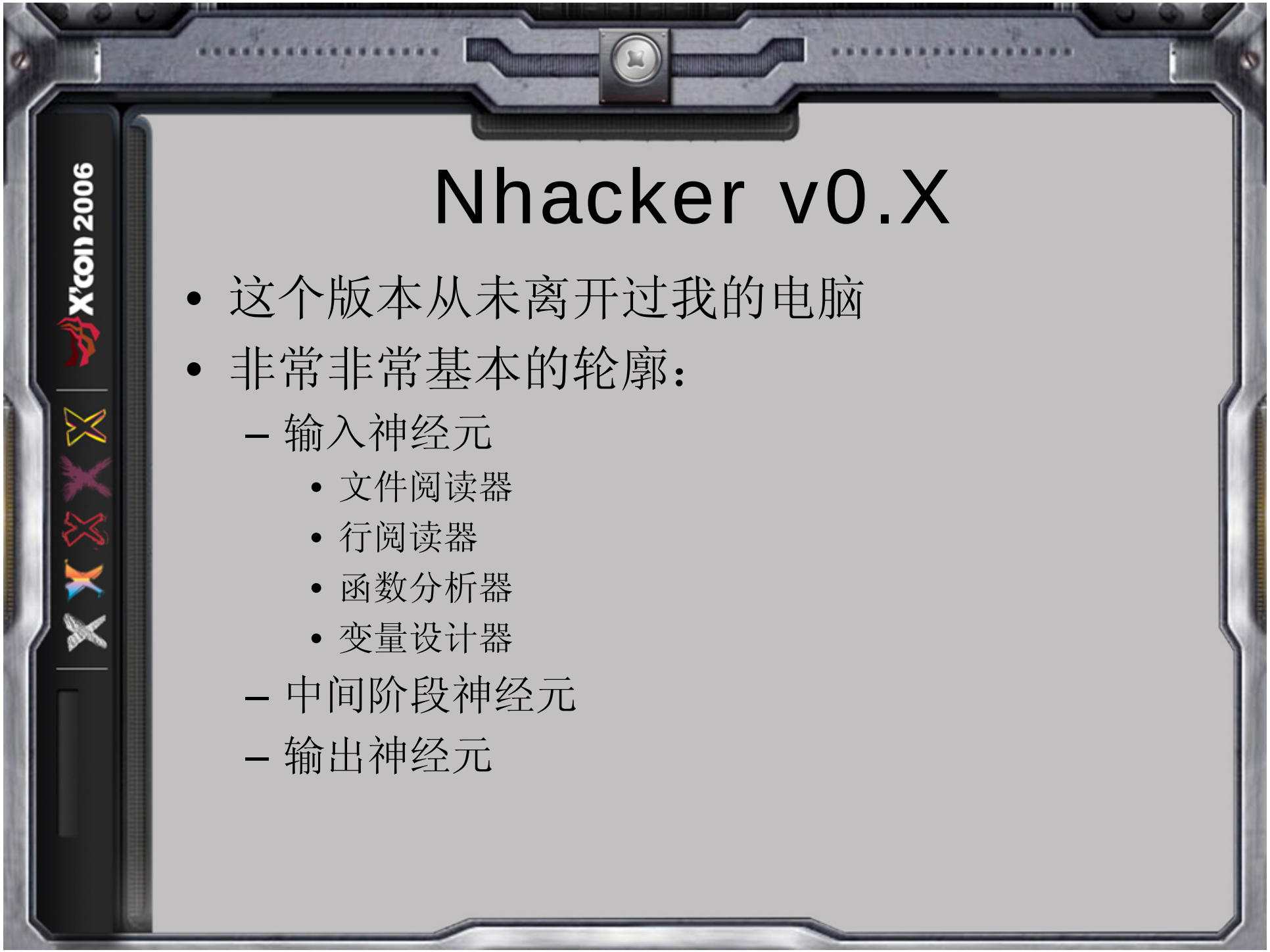
背景

- 第二天我朝所有的人大吼大叫
- 很明显某人捡到了这张纸巾并且在linux里写代码
- nhacker v0.1就这样诞生了
- 这个程序有个License，叫“嘿，你知道我醉了”



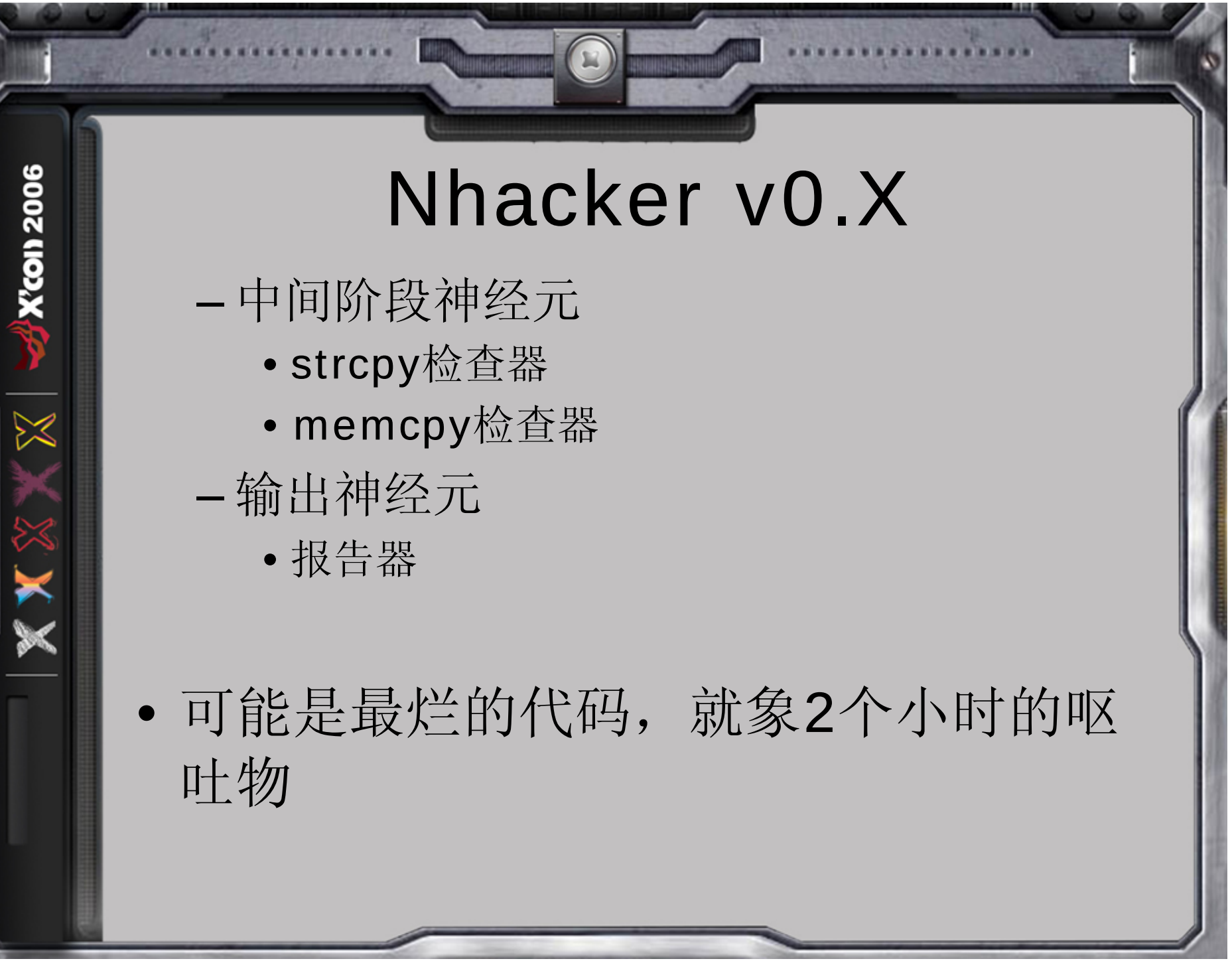
背景

- 2003年开始和老费讨论这些事情
- 写了第一个非常基本的神经网络来读代码
- 老费发布了rux03.tgz给我，一个通用代码库的crackaddr() exploit搜索工具



Nhacker v0.X

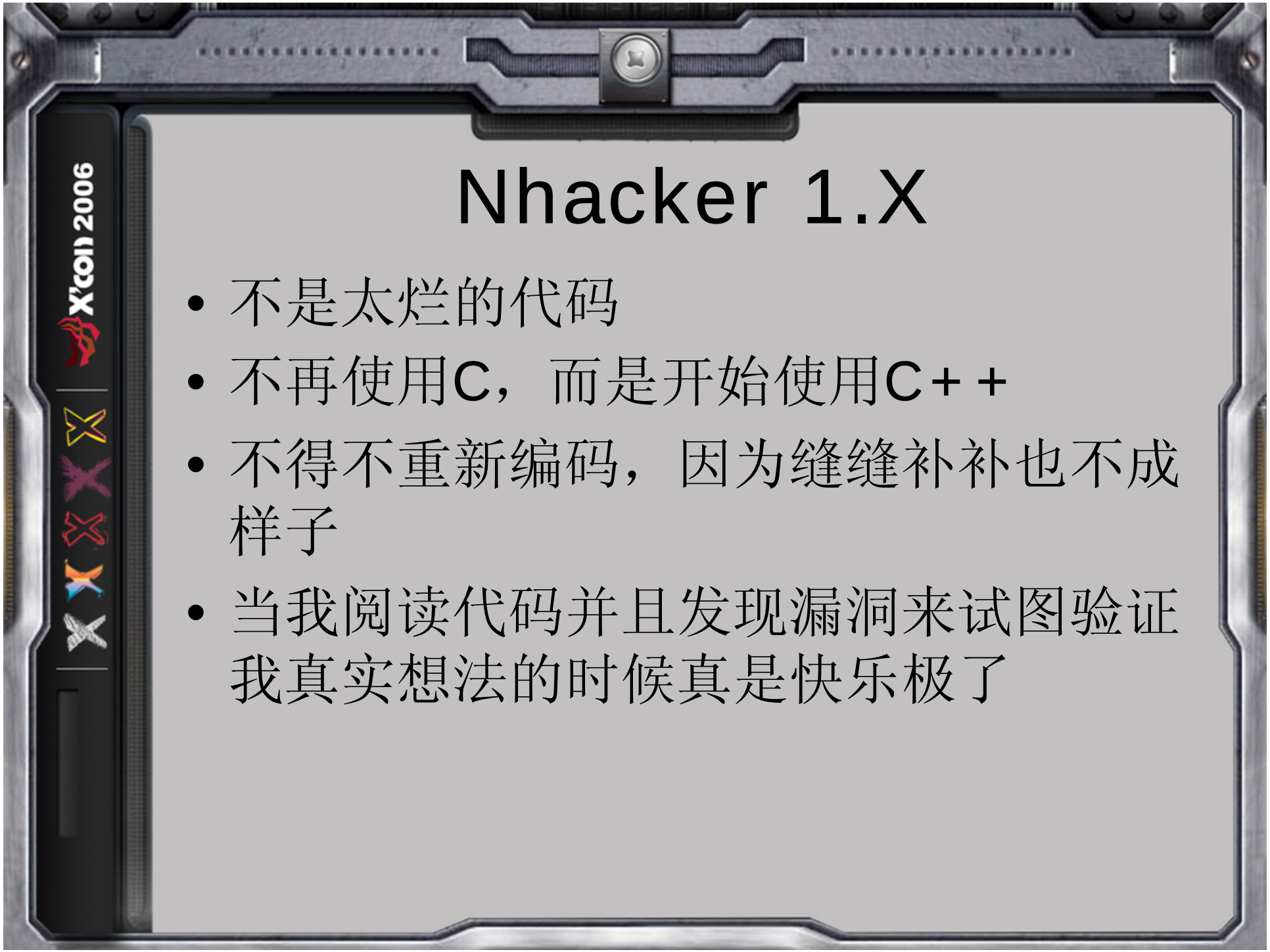
- 这个版本从未离开过我的电脑
- 非常非常基本的轮廓：
  - 输入神经元
    - 文件阅读器
    - 行阅读器
    - 函数分析器
    - 变量设计器
  - 中间阶段神经元
  - 输出神经元



Nhacker v0.X

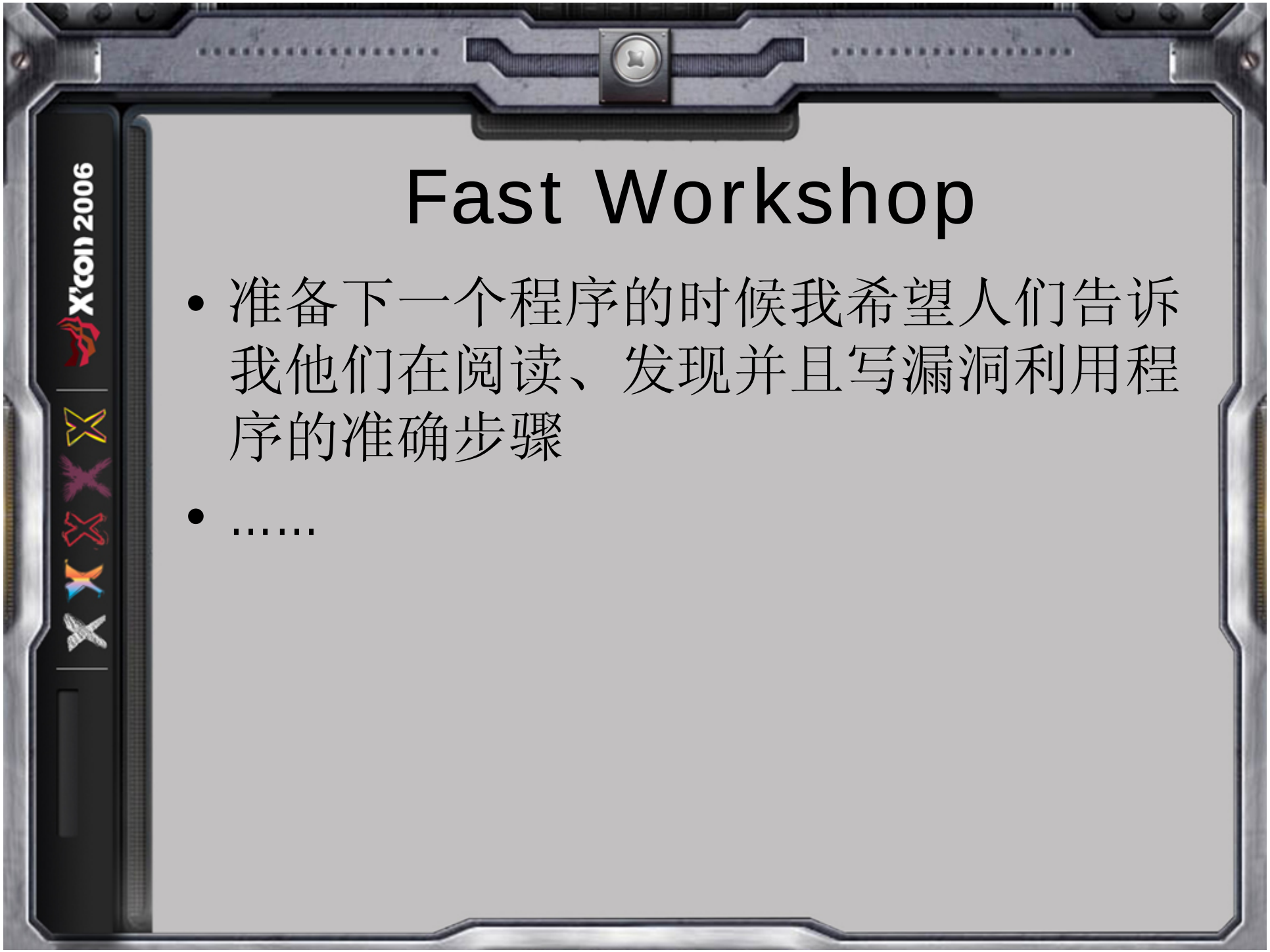
- 中间阶段神经元
  - strcpy检查器
  - memcpy检查器
- 输出神经元
  - 报告器

- 可能是最烂的代码，就象2个小时的呕吐物



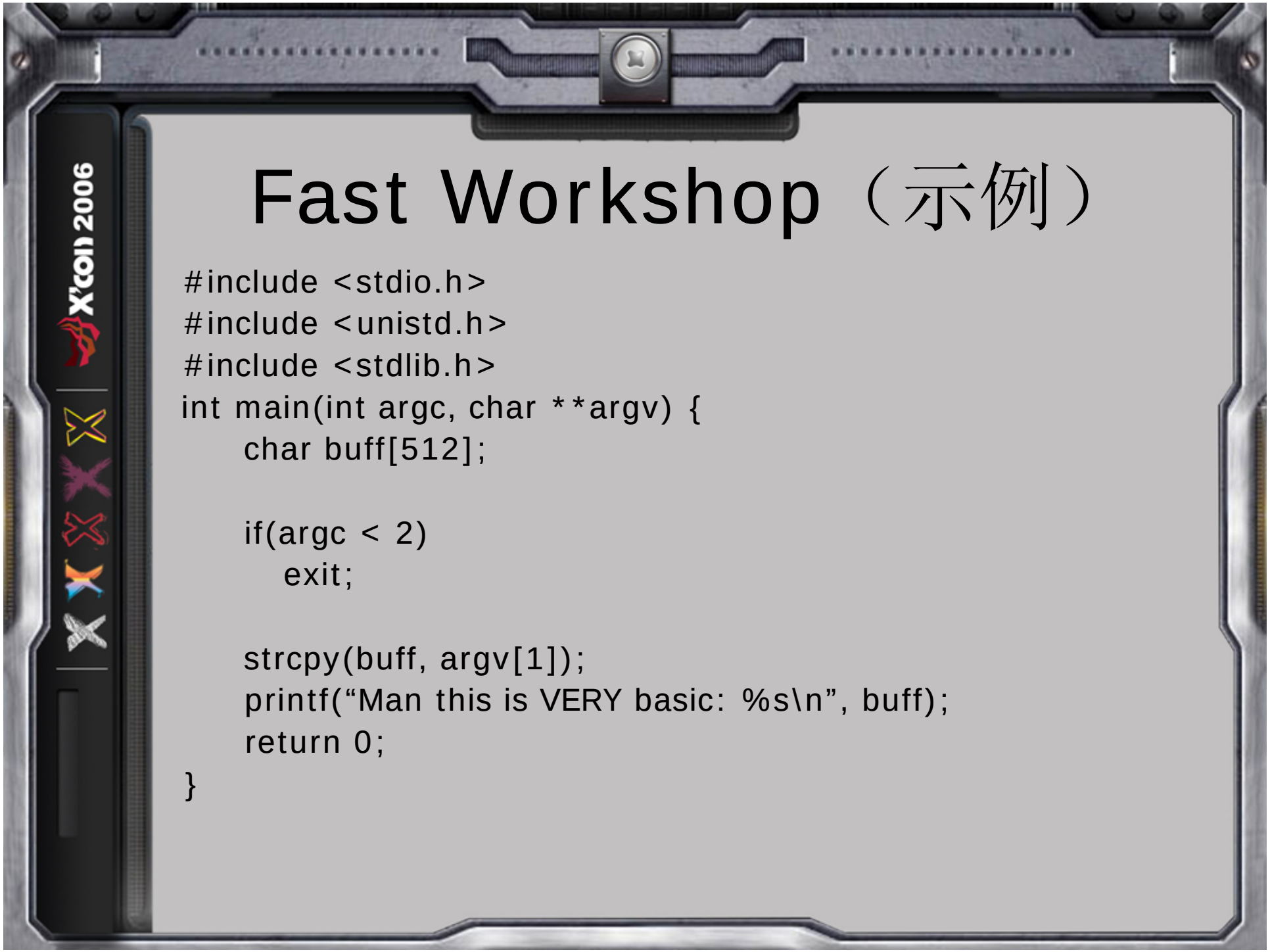
Nhacker 1.X

- 不是太烂的代码
- 不再使用C，而是开始使用C++
- 不得不重新编码，因为缝缝补补也不成样子
- 当我阅读代码并且发现漏洞来试图验证我真实想法的时候真是快乐极了



Fast Workshop

- 准备下一个程序的时候我希望人们告诉我他们在阅读、发现并且写漏洞利用程序的准确步骤
- ……

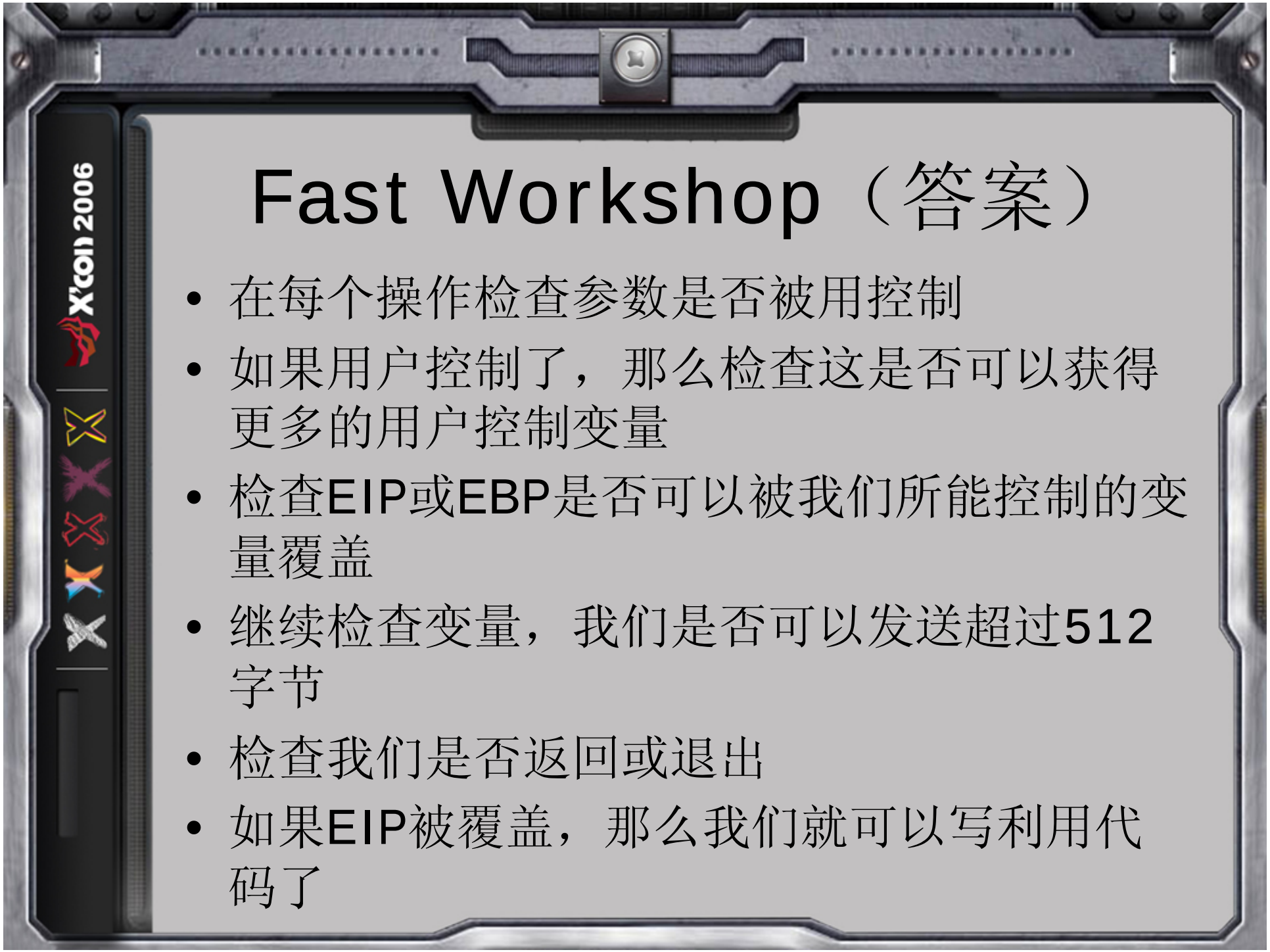


Fast Workshop（示例）

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char **argv) {
    char buff[512];

    if(argc < 2)
       exit;

    strcpy(buff, argv[1]);
    printf("Man this is VERY basic: %s\n", buff);
    return 0;
}
```

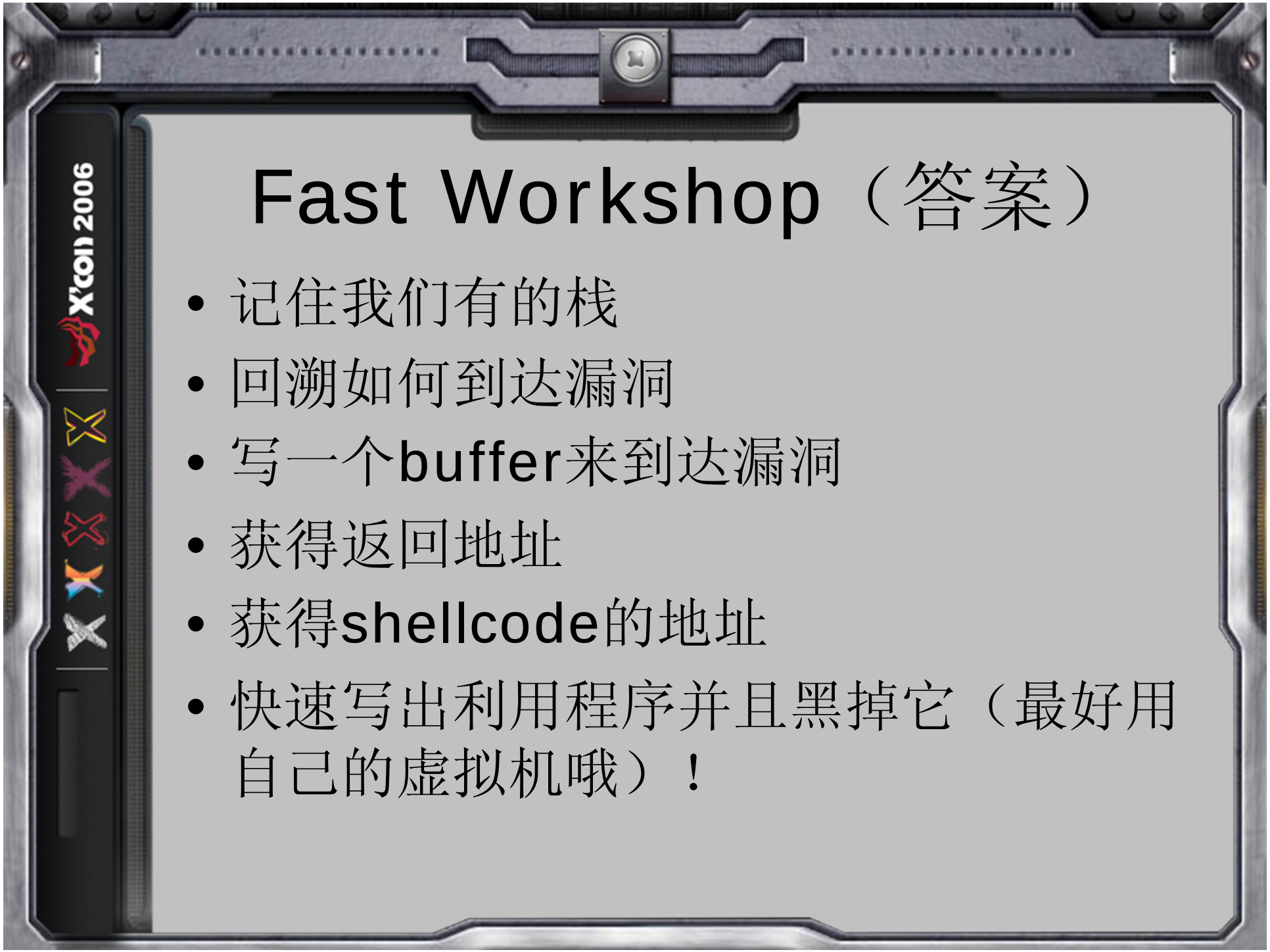
Fast Workshop（答案）

- 下面是（几乎）所有步骤：
  - 寻找文件
  - 检查目录里是否还有更多的文件
  - 打开文件
  - 阅读每一行来查找函数
  - 在每个函数里创建一个栈给它
  - 填充栈并且保留在内存里，当它被调用的时候进入
  - 给变量分配大小



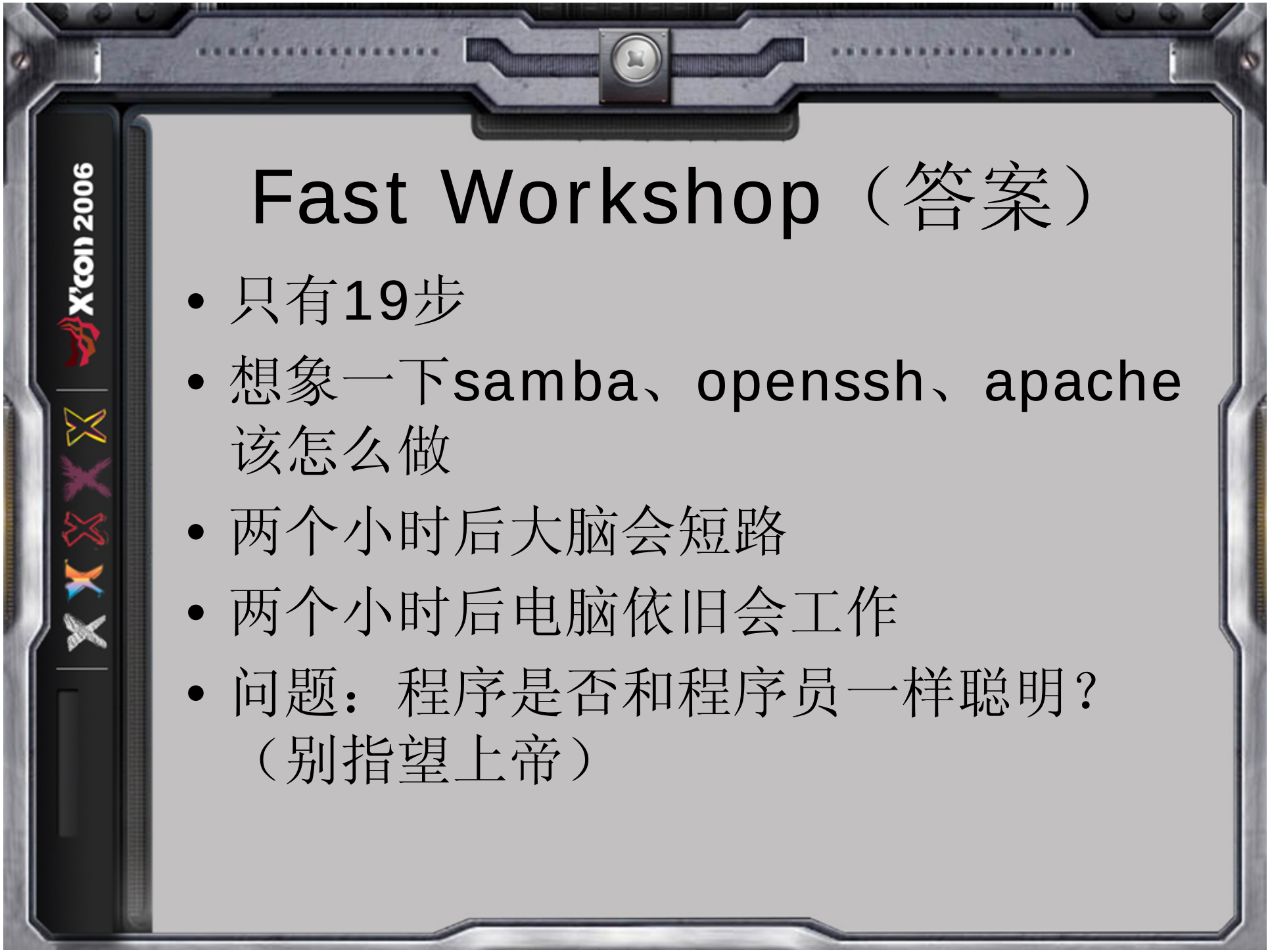
Fast Workshop（答案）

- 在每个操作检查参数是否被用控制
- 如果用户控制了，那么检查这是否可以获得更多的用户控制变量
- 检查EIP或EBP是否可以被我们所能控制的变量覆盖
- 继续检查变量，我们是否可以发送超过512字节
- 检查我们是否返回或退出
- 如果EIP被覆盖，那么我们就可以写利用代码了



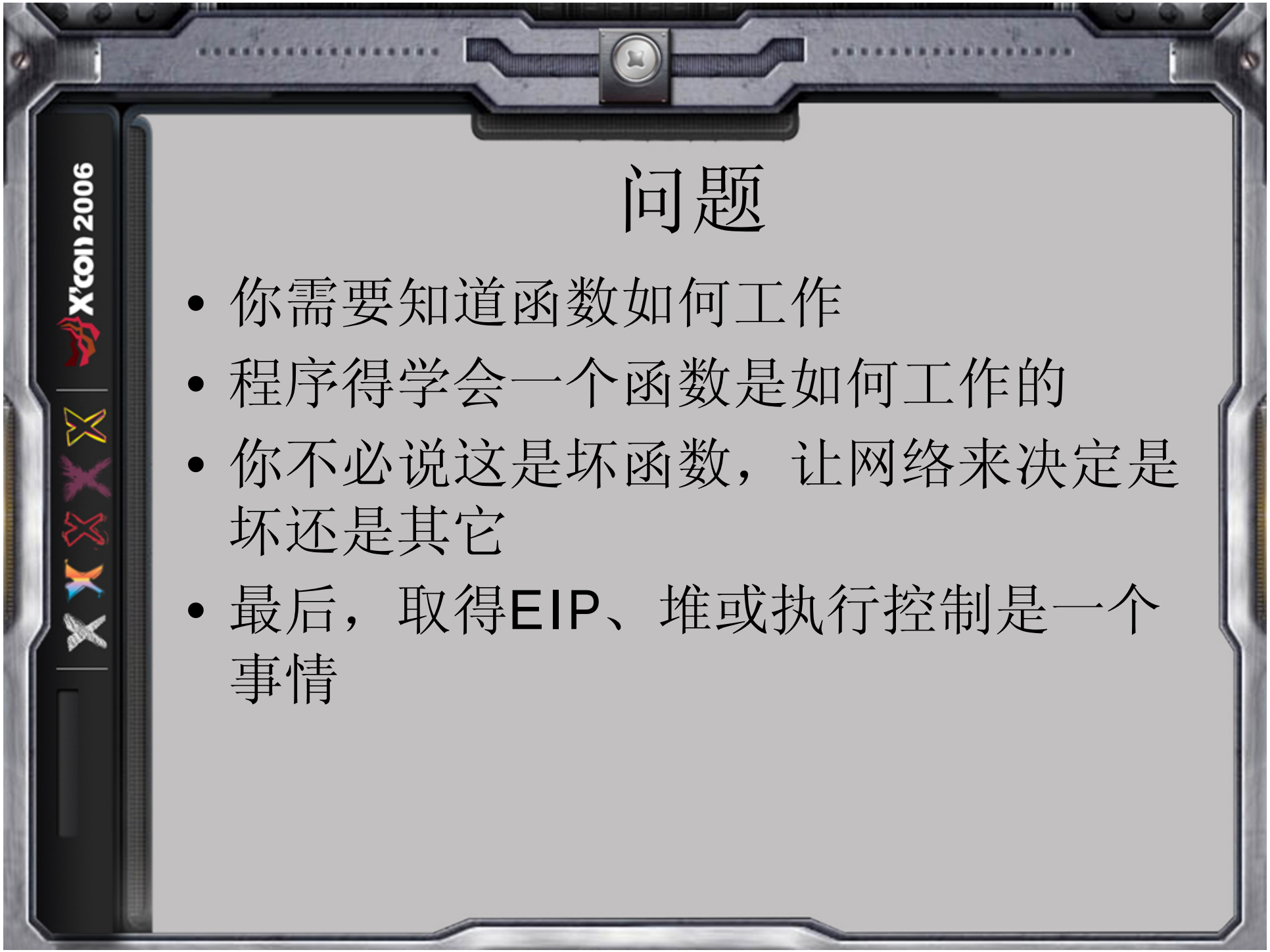
Fast Workshop（答案）

- 记住我们有的栈
- 回溯如何到达漏洞
- 写一个buffer来到达漏洞
- 获得返回地址
- 获得shellcode的地址
- 快速写出利用程序并且黑掉它（最好用自己的虚拟机哦）！



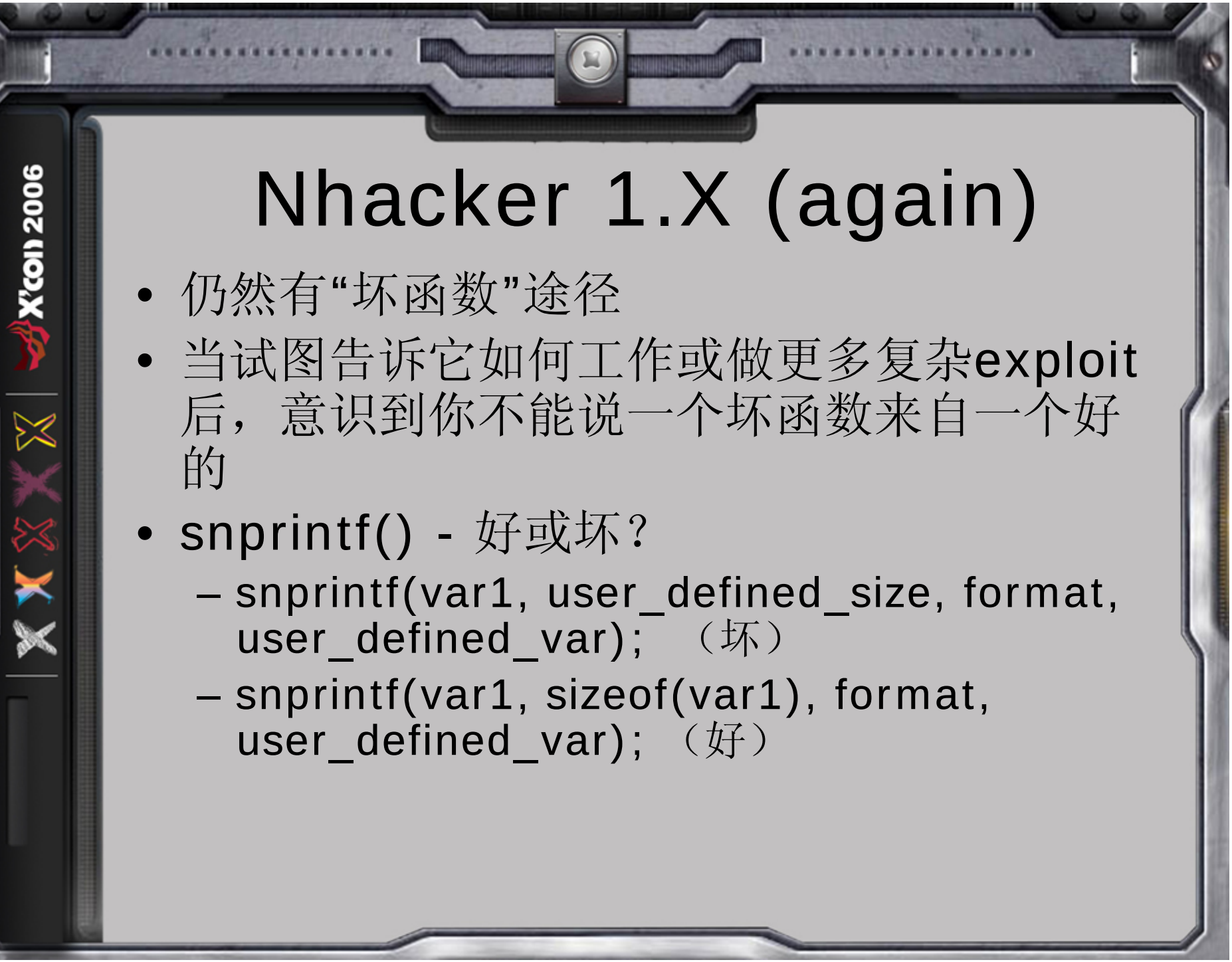
Fast Workshop（答案）

- 只有19步
- 想象一下samba、openssh、apache该怎么做
- 两个小时后大脑会短路
- 两个小时后电脑依旧会工作
- 问题：程序是否和程序员一样聪明？（别指望上帝）



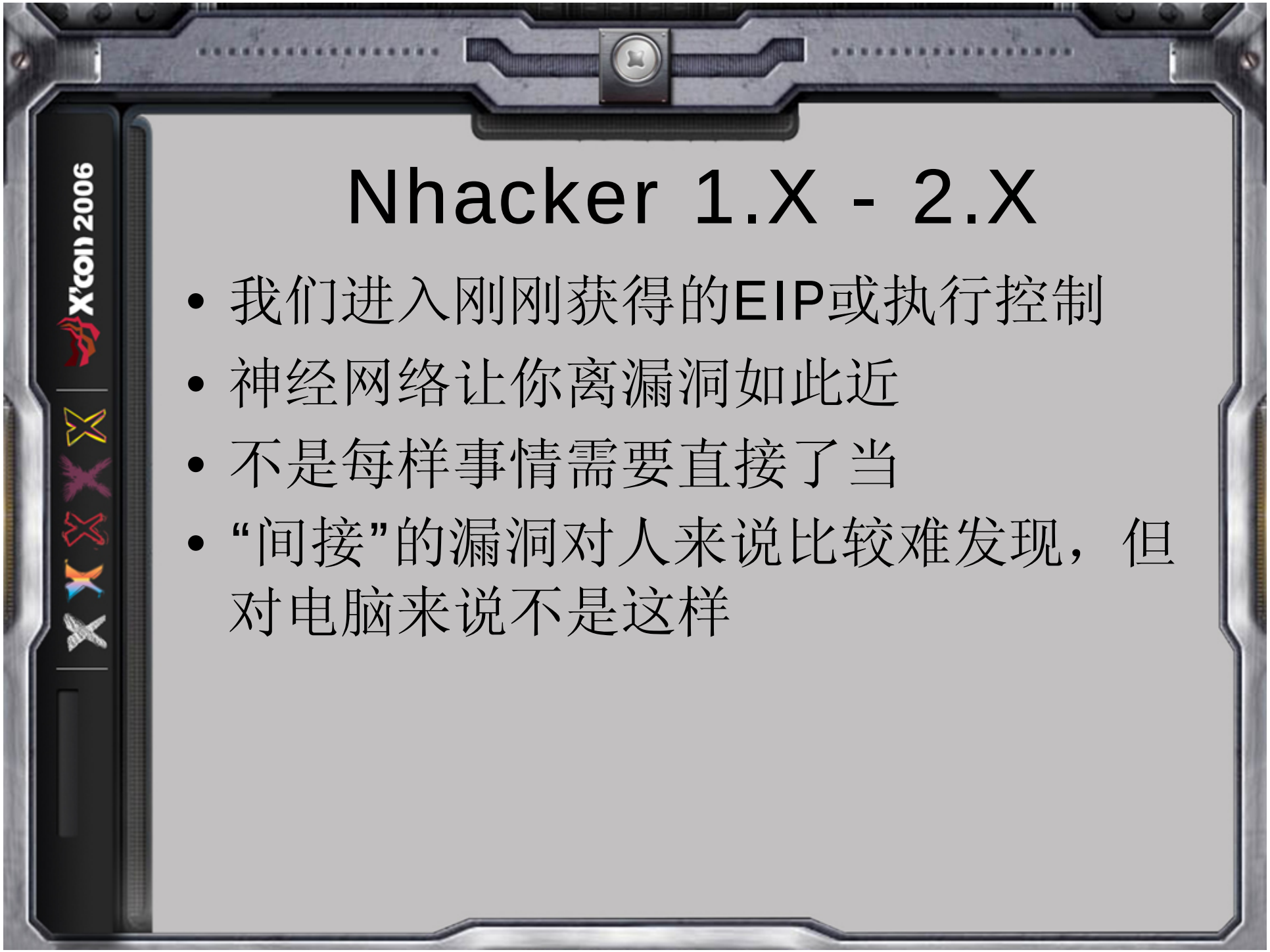
问题

- 你需要知道函数如何工作
- 程序得学会一个函数是如何工作的
- 你不必说这是坏函数，让网络来决定是坏还是其它
- 最后，取得EIP、堆或执行控制是一个事情



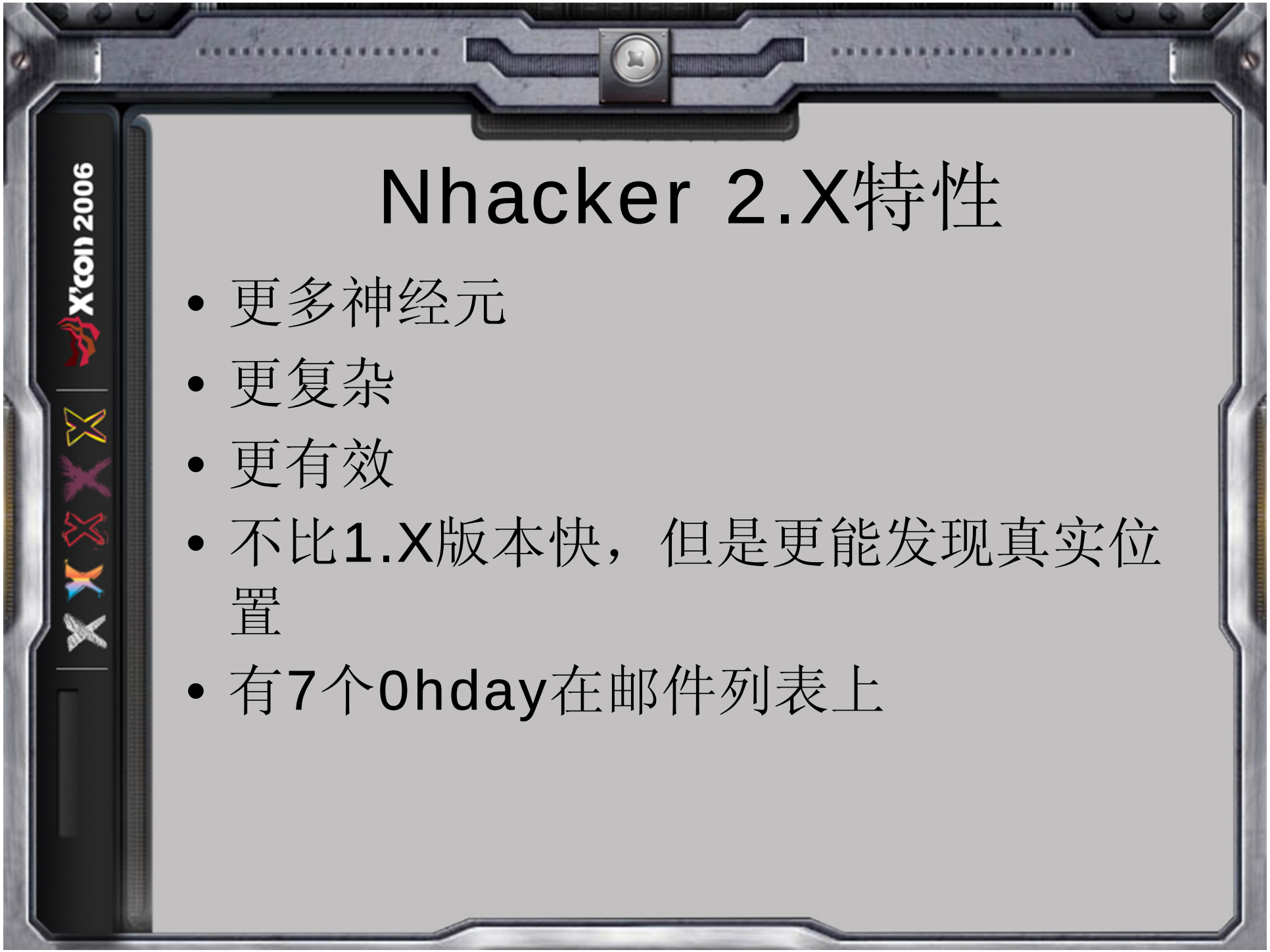
Nhacker 1.X (again)

- 仍然有“坏函数”途径
- 当试图告诉它如何工作或做更多复杂exploit后，意识到你不能说一个坏函数来自一个好的
- snprintf() - 好或坏？
  - snprintf(var1, user_defined_size, format, user_defined_var); （坏）
  - snprintf(var1, sizeof(var1), format, user_defined_var); （好）



Nhacker 1.X - 2.X

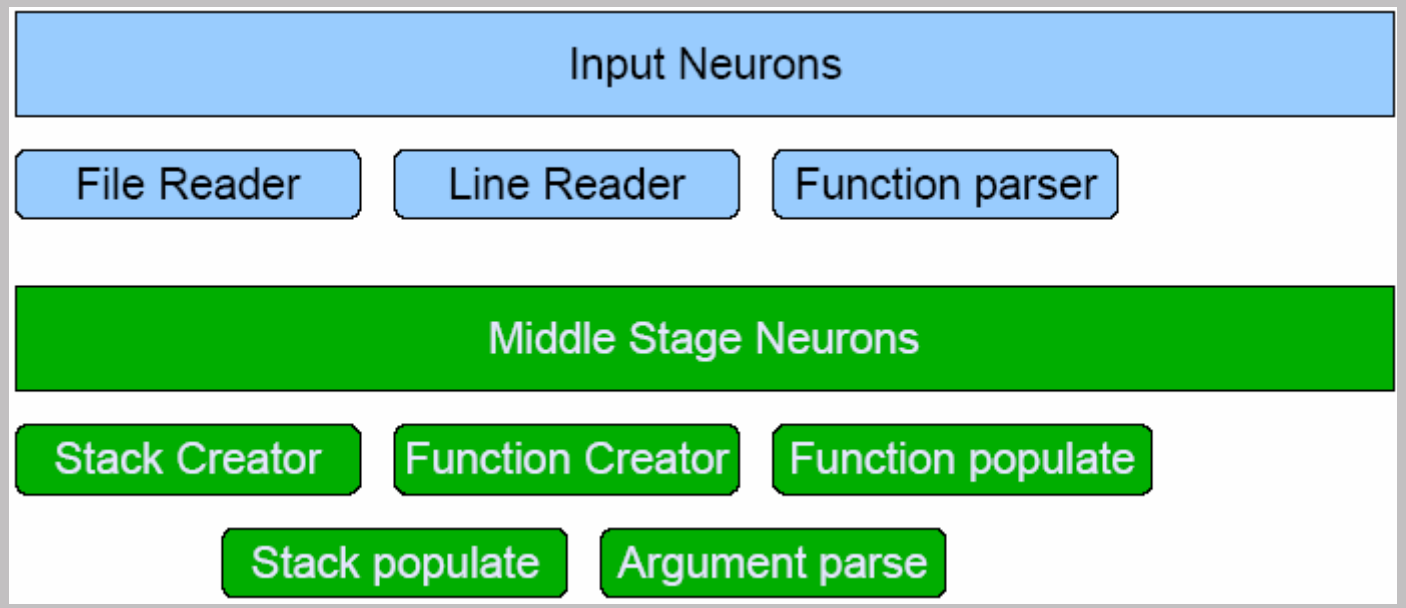
- 我们进入刚刚获得的EIP或执行控制
- 神经网络让你离漏洞如此近
- 不是每样事情需要直接了当
- “间接”的漏洞对人来说比较难发现，但对电脑来说不是这样



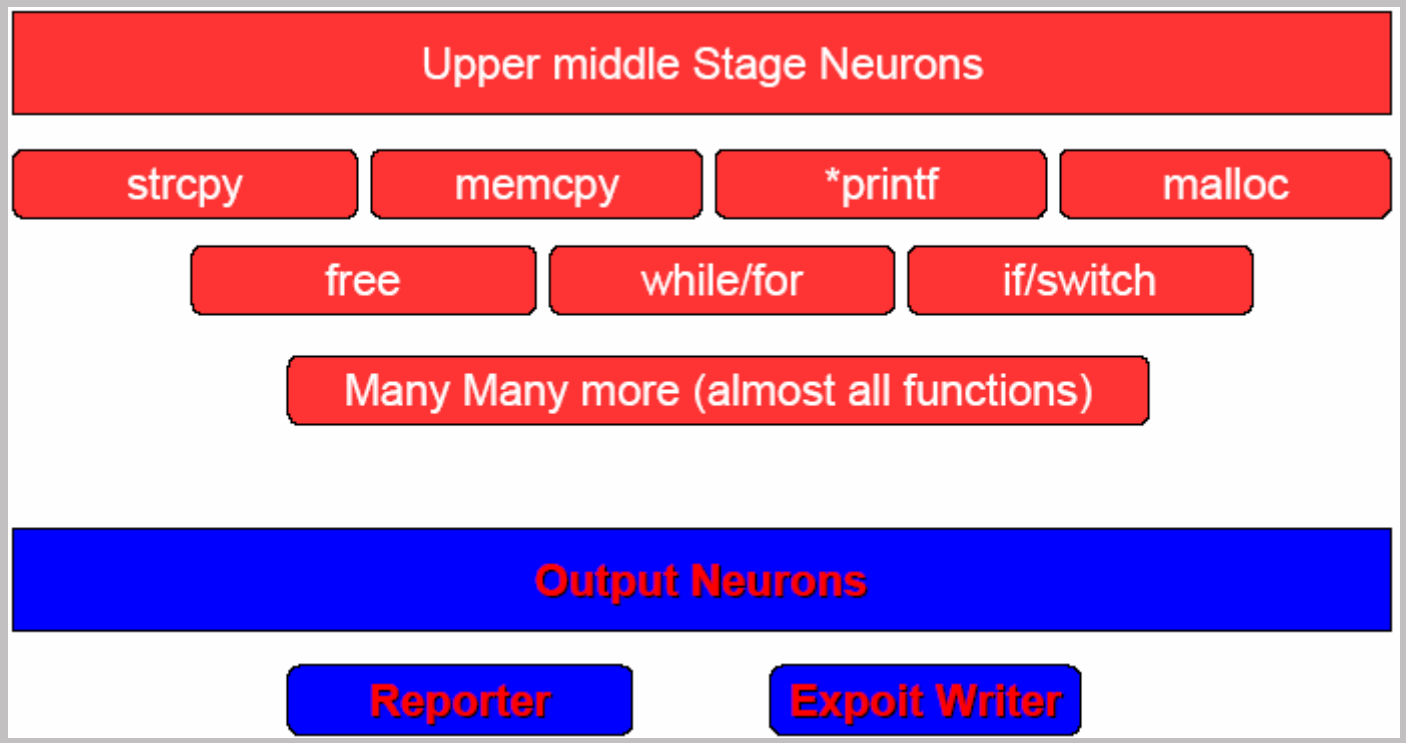
Nhacker 2.X特性

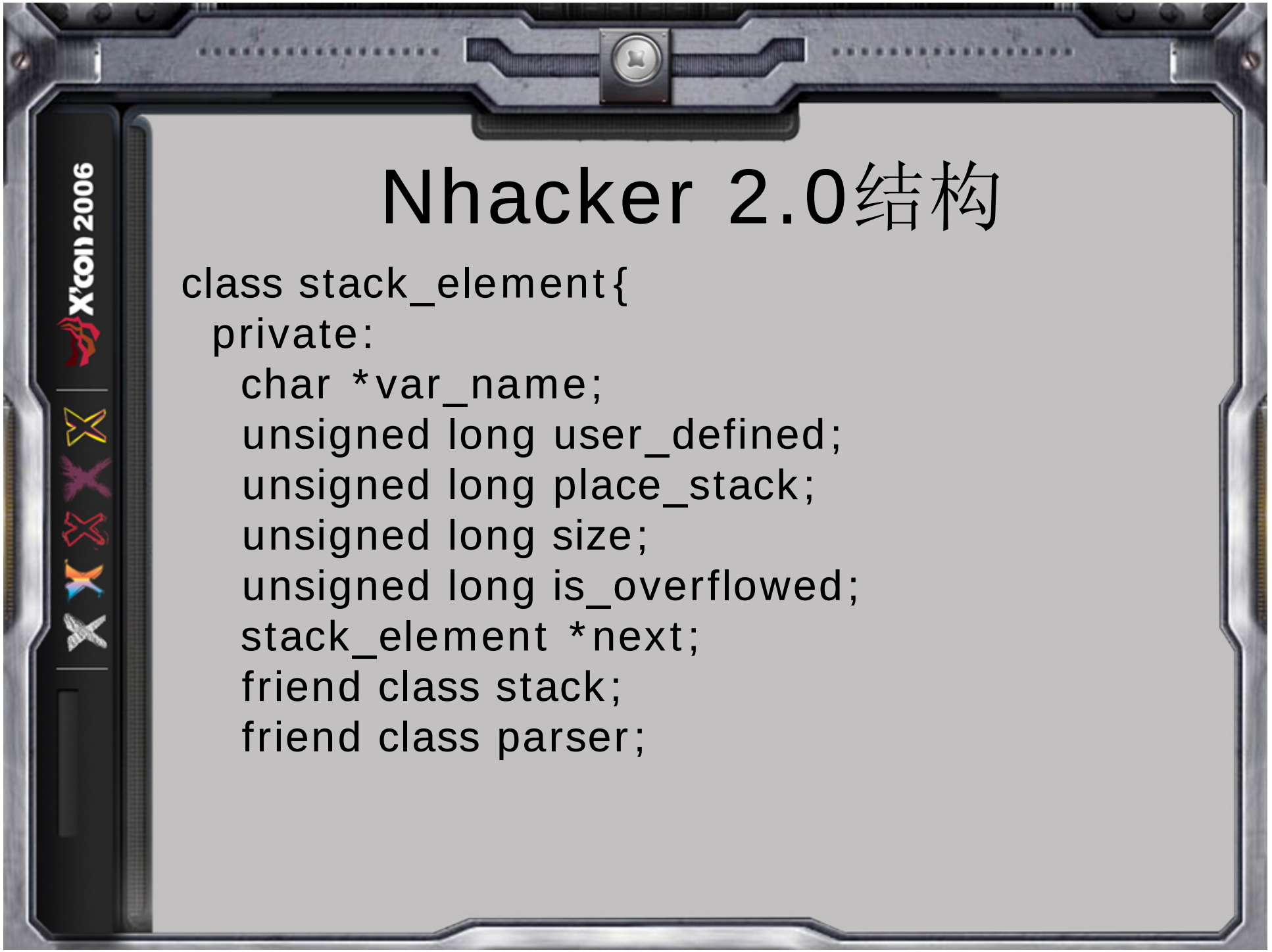
- 更多神经元
- 更复杂
- 更有效
- 不比1.X版本快，但是更能发现真实位置
- 有7个0hday在邮件列表上

Nhacker 2.0图表



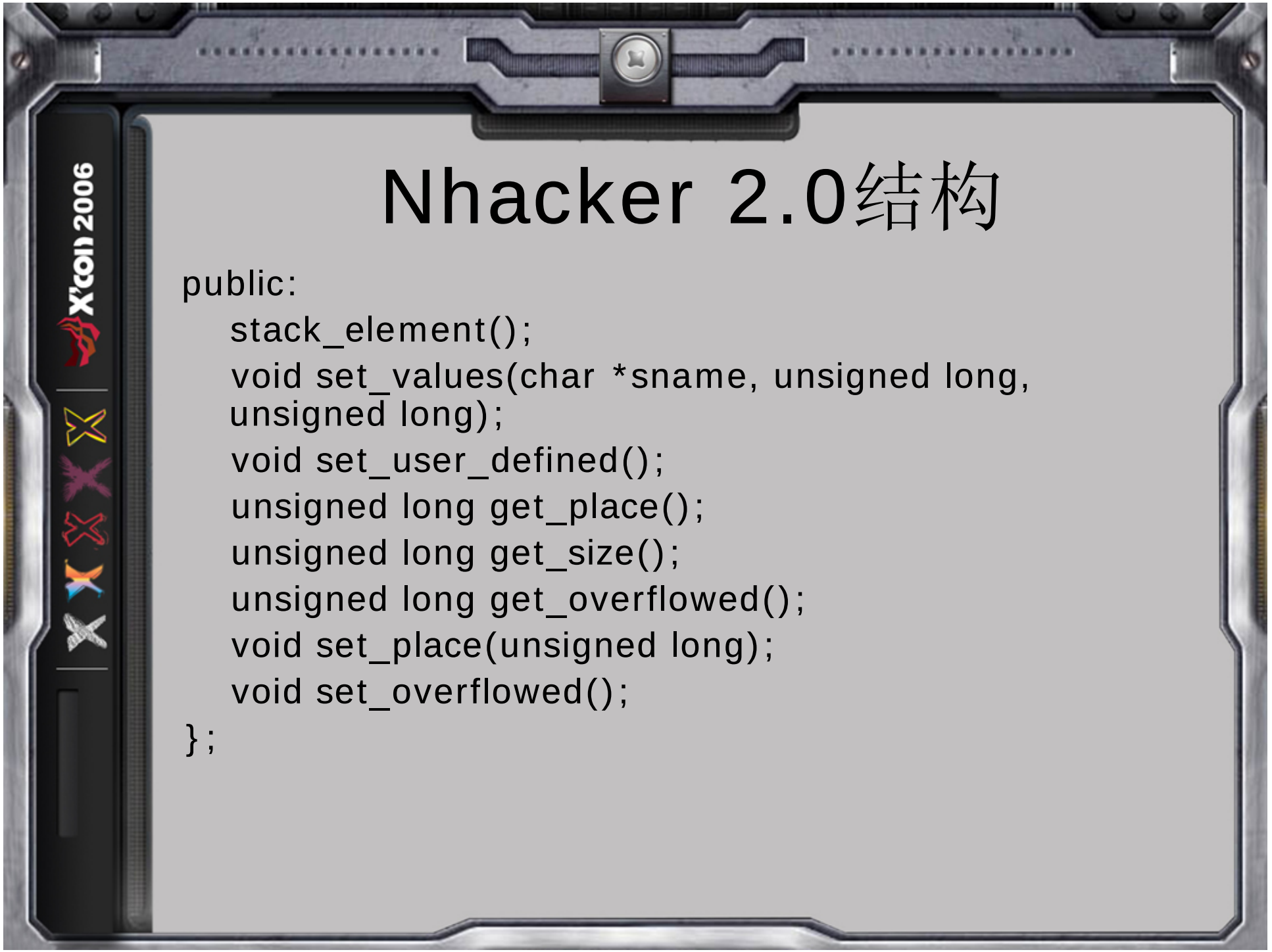
Nhacker 2.0图表





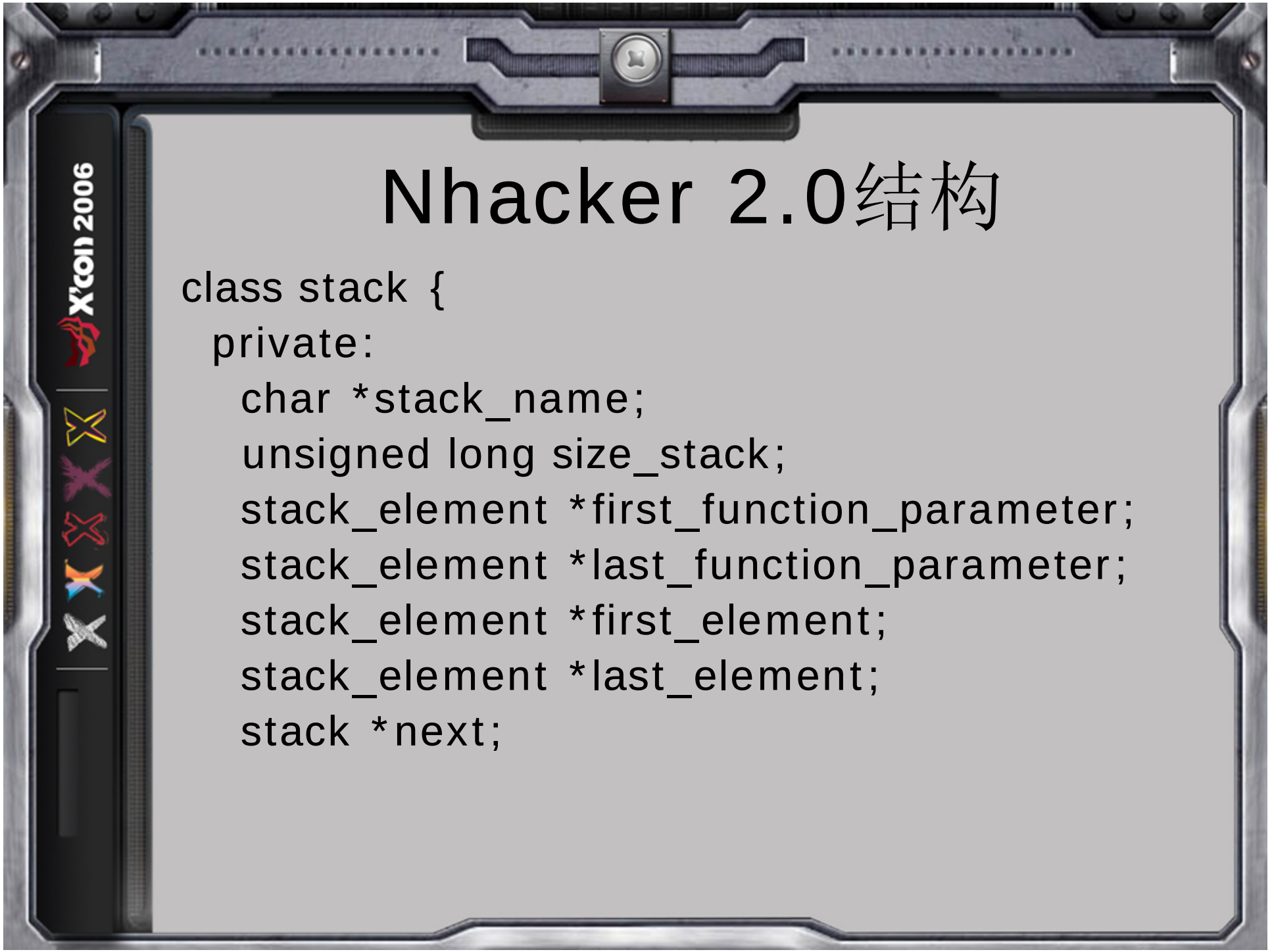
Nhacker 2.0结构

```
class stack_element{
  private:
    char *var_name;
    unsigned long user_defined;
    unsigned long place_stack;
    unsigned long size;
    unsigned long is_overflowed;
    stack_element *next;
    friend class stack;
    friend class parser;
```



Nhacker 2.0结构

```
public:
    stack_element();
    void set_values(char *sname, unsigned long,
    unsigned long);
    void set_user_defined();
    unsigned long get_place();
    unsigned long get_size();
    unsigned long get_overflowed();
    void set_place(unsigned long);
    void set_overflowed();
};
```



Nhacker 2.0结构

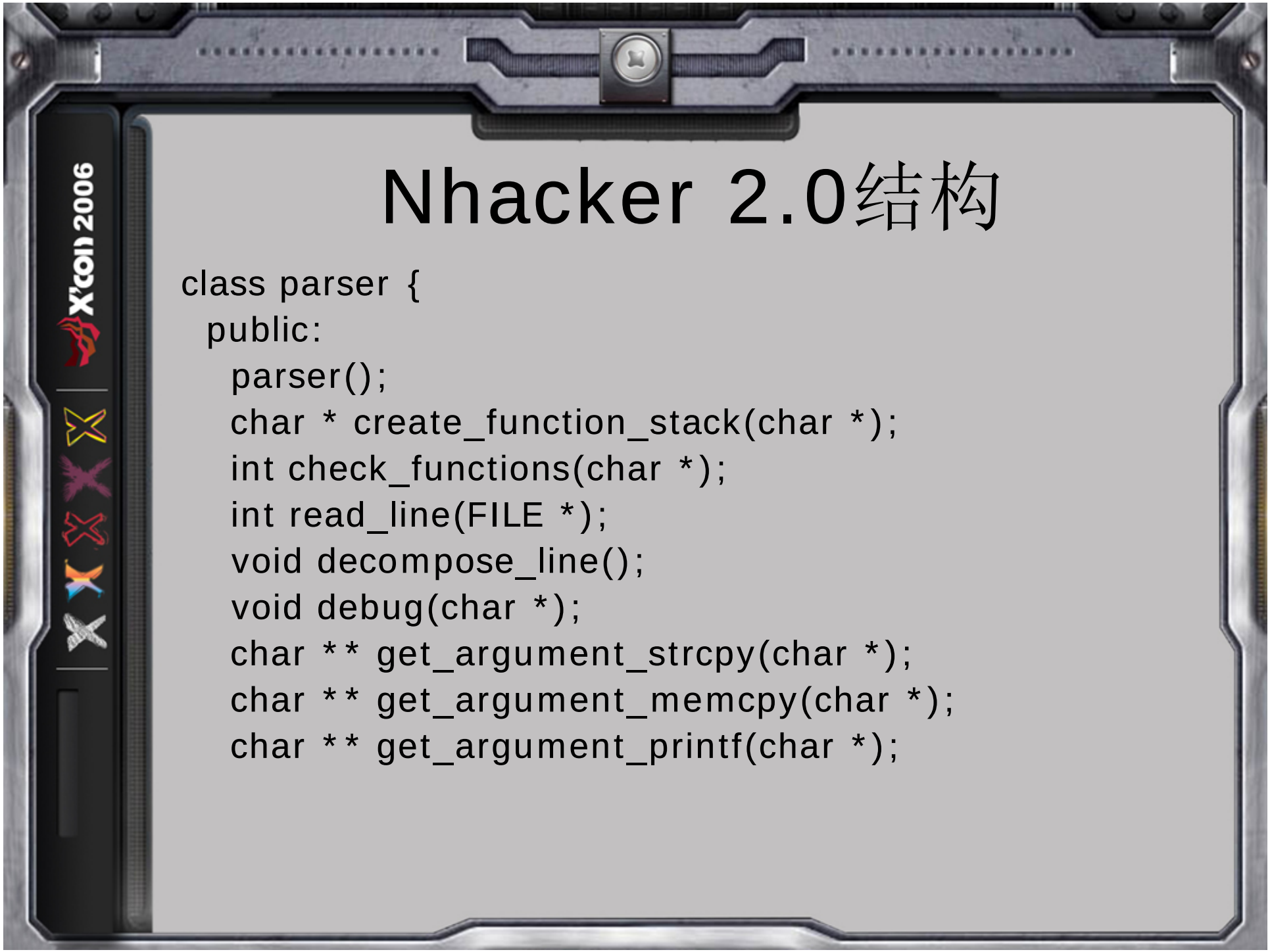
```
class stack {
  private:
    char *stack_name;
    unsigned long size_stack;
    stack_element *first_function_parameter;
    stack_element *last_function_parameter;
    stack_element *first_element;
    stack_element *last_element;
    stack *next;
```

Nhacker 2.0结构

```
public:
    stack(char *);
    int get_stack_number();
    void pop_stack();
    int pop_stack(char *);
    void print_stack();
    void add_2_stack(stack_element *);
    void add_2_parameters(stack_element *);
    stack_element * search_in_stack(char *);
    unsigned long stack_size_by_name(char *);
};
```

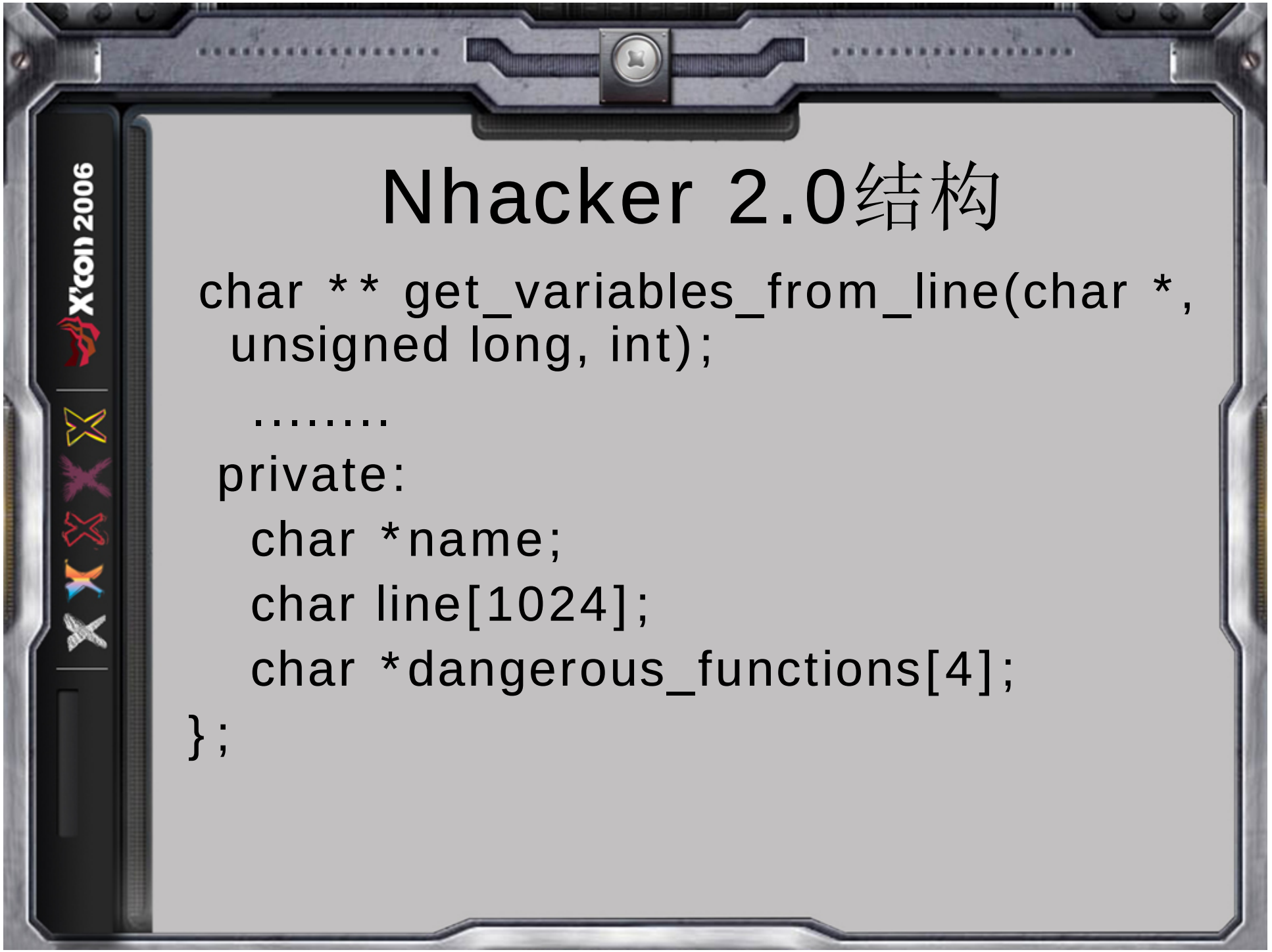
Nhacker 2.0结构

```
class stack_list {
  private:
    char *name;
    stack *first_stack;
    stack *last_stack;
  public:
    stack_list(char *);
    ~stack_list();
    int add_2_stack_list(char *);
};
```



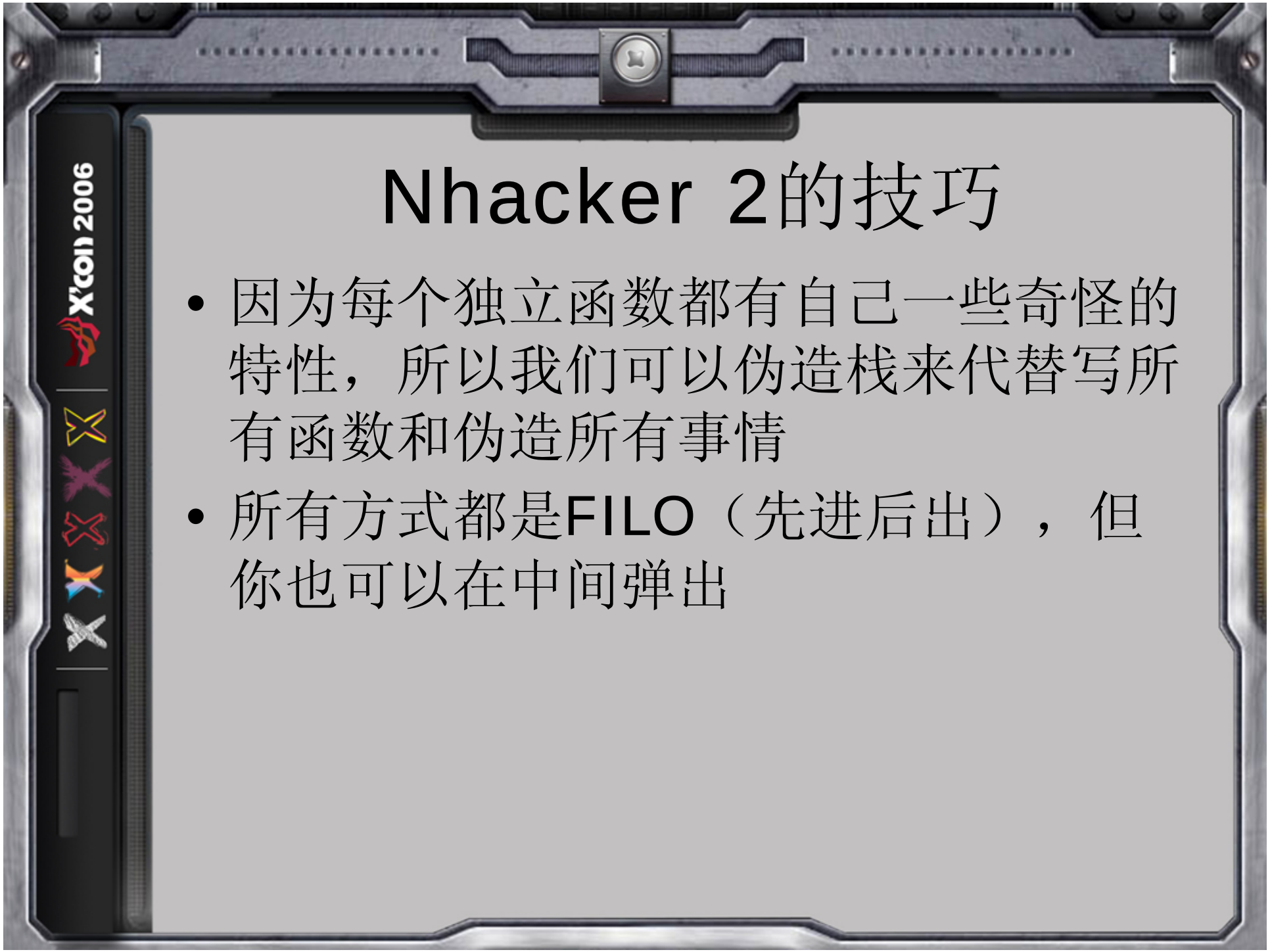
Nhacker 2.0结构

```
class parser {
  public:
    parser();
    char * create_function_stack(char *);
    int check_functions(char *);
    int read_line(FILE *);
    void decompose_line();
    void debug(char *);
    char ** get_argument_strcpy(char *);
    char ** get_argument_memcpy(char *);
    char ** get_argument_printf(char *);
```



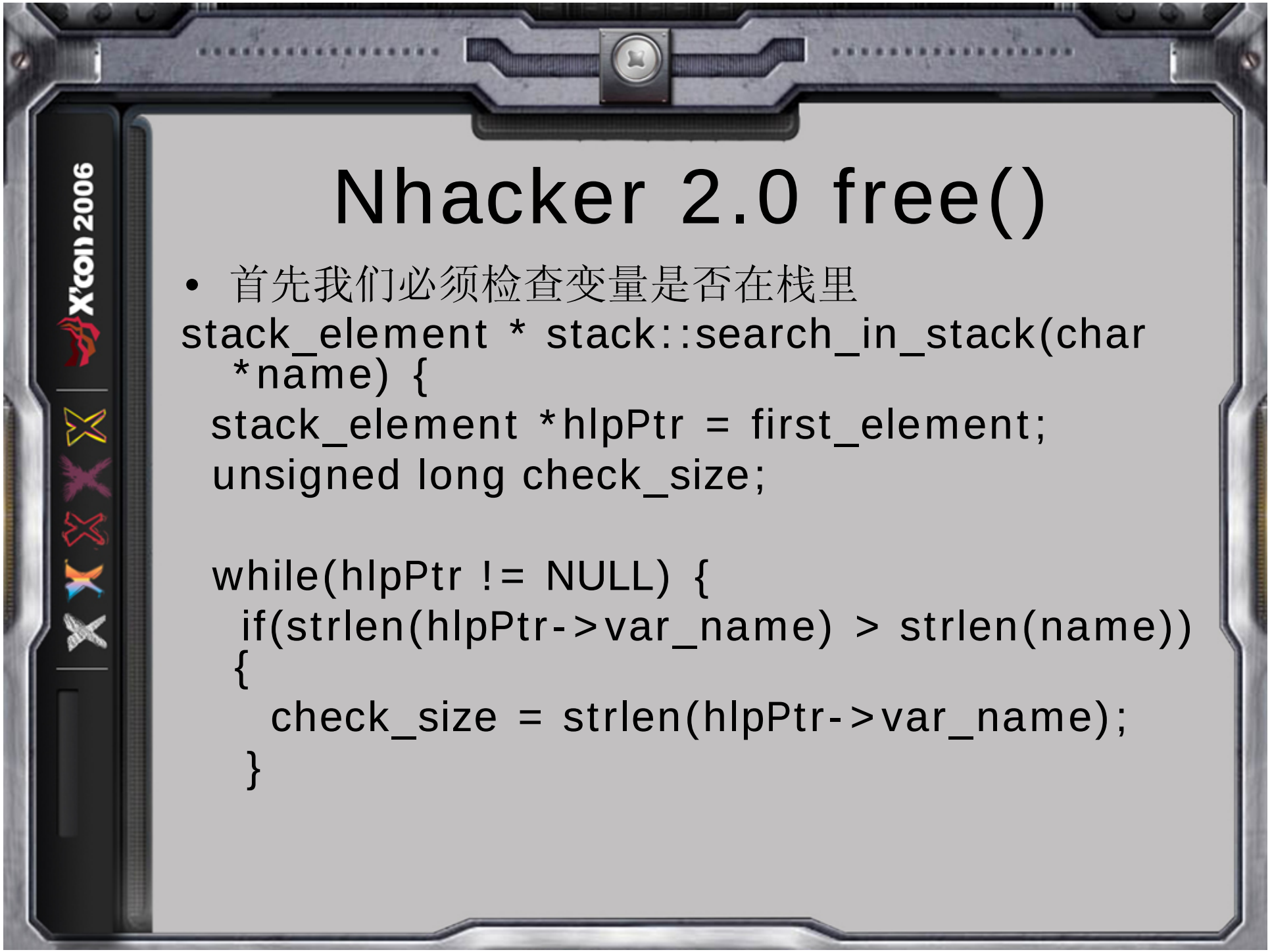
Nhacker 2.0结构

```
char ** get_variables_from_line(char *,
 unsigned long, int);
   ........
 private:
   char *name;
   char line[1024];
   char *dangerous_functions[4];
};
```



Nhacker 2的技巧

- 因为每个独立函数都有自己一些奇怪的特性，所以我们可以伪造栈来代替写所有函数和伪造所有事情
- 所有方式都是FILO（先进后出），但你也可以在中间弹出



Nhacker 2.0 free()

- 首先我们必须检查变量是否在栈里

```
stack_element * stack::search_in_stack(char
   *name) {
  stack_element *hlpPtr = first_element;
  unsigned long check_size;

  while(hlpPtr != NULL) {
   if(strlen(hlpPtr->var_name) > strlen(name))
   {
     check_size = strlen(hlpPtr->var_name);
   }
```

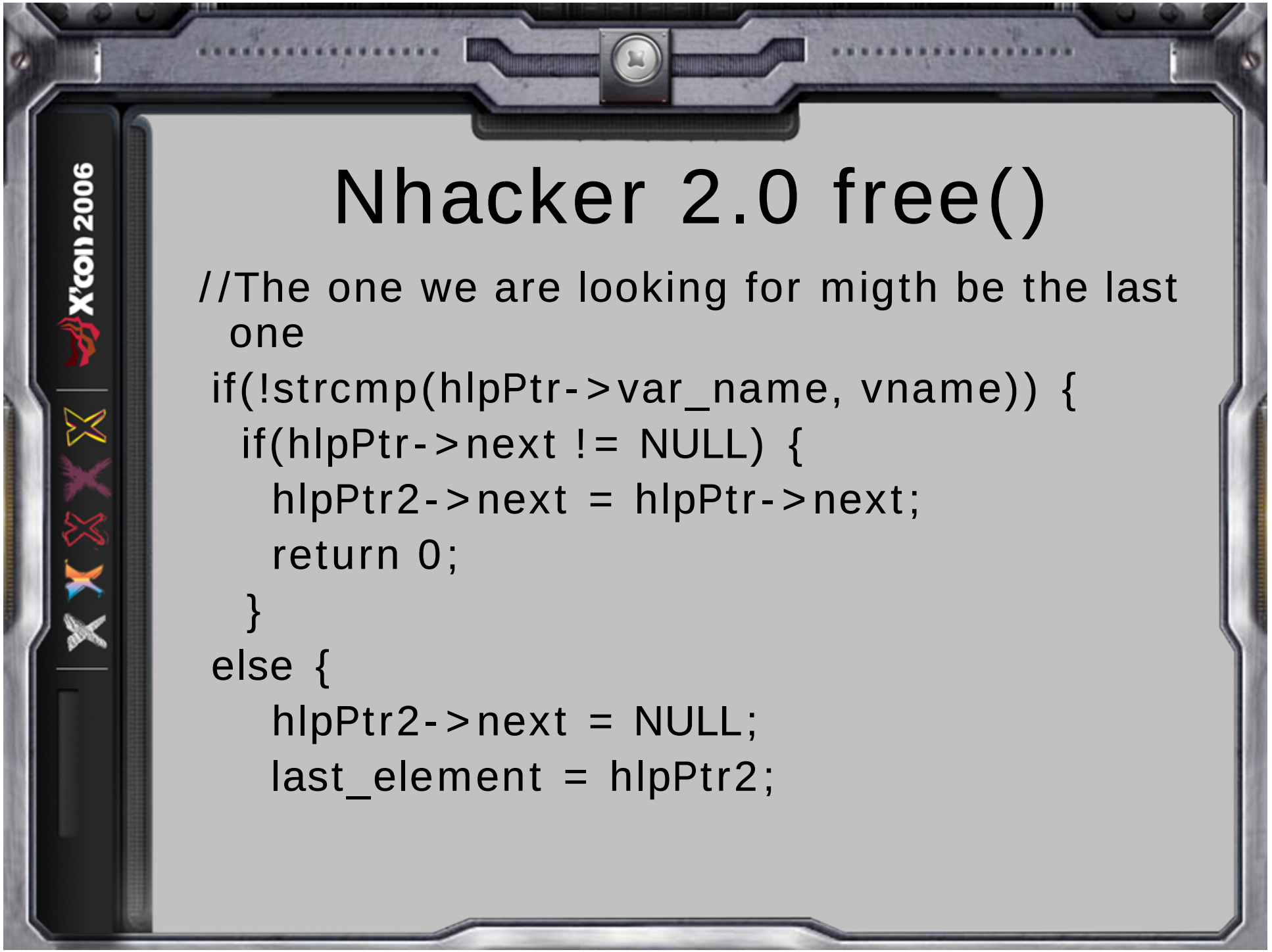
Nhacker 2.0 free()

```
 else {
    check_size = strlen(name);
  }

  if(!(strncmp(hlpPtr->var_name, name, check_size)))
  {
    return hlpPtr;
  }

  hlpPtr = hlpPtr->next;
 }

 return NULL;
}
```

Nhacker 2.0 free()

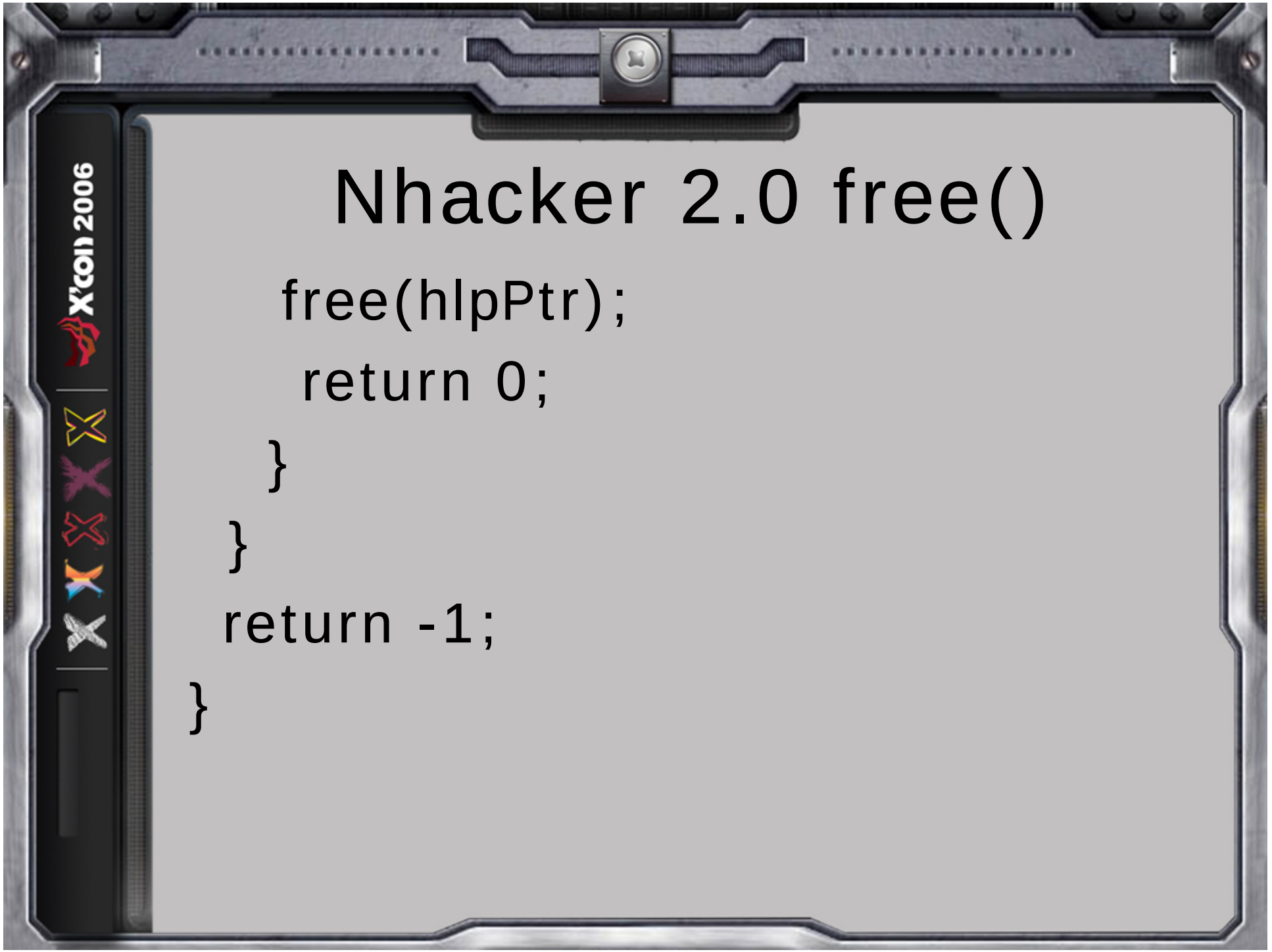
- 现在我们“释放”变量（如果变量在栈中）

```
int stack::pop_stack(char *vname){
  stack_element *hlpPtr = first_element;
  stack_element *hlpPtr2 = hlpPtr;

  while((hlpPtr->var_name != vname) ||
   (hlpPtr->next != NULL)) {
    hlpPtr2 = hlpPtr;
    hlpPtr=hlpPtr->next;
  }
```



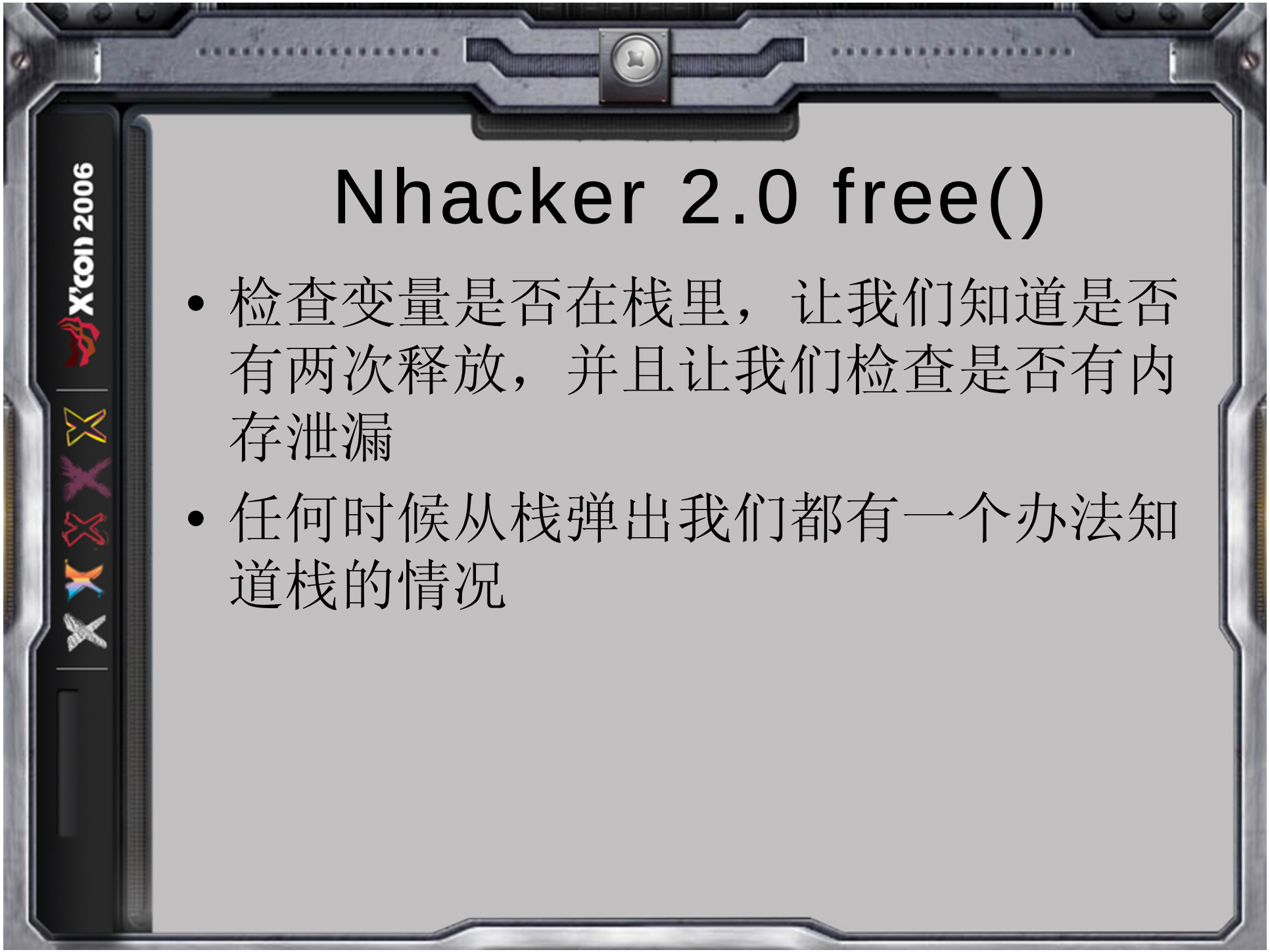
Nhacker 2.0 free()

```
//The one we are looking for migth be the last
  one
 if(!strcmp(hlpPtr->var_name, vname)) {
   if(hlpPtr->next != NULL) {
     hlpPtr2->next = hlpPtr->next;
     return 0;
   }
 else {
     hlpPtr2->next = NULL;
     last_element = hlpPtr2;
```



Nhacker 2.0 free()

```
    free(hlpPtr);
     return 0;
   }
  }
  return -1;
}
```

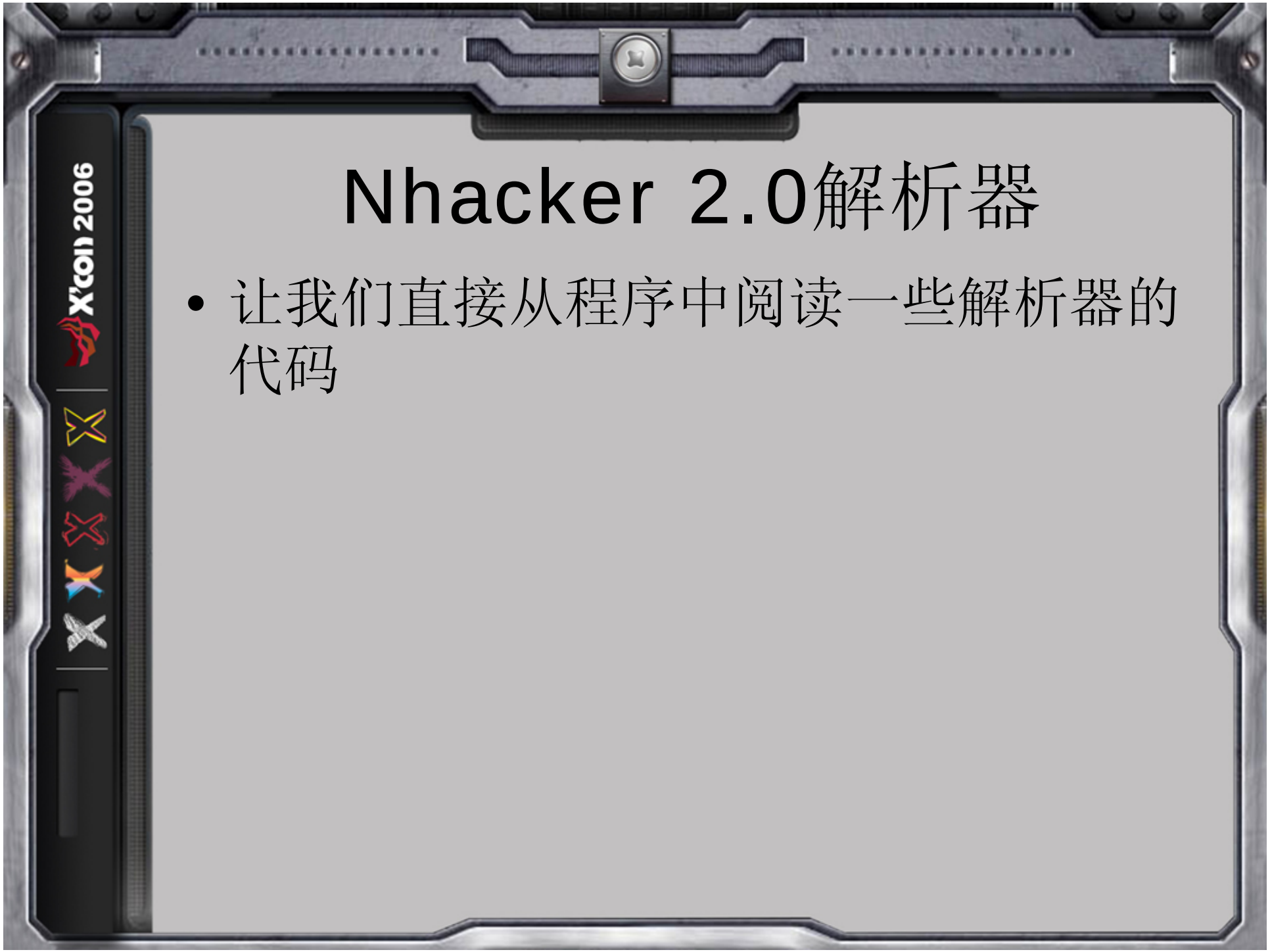


Nhacker 2.0 free()

- 检查变量是否在栈里，让我们知道是否有两次释放，并且让我们检查是否有内存泄漏
- 任何时候从栈弹出我们都有一个办法知道栈的情况

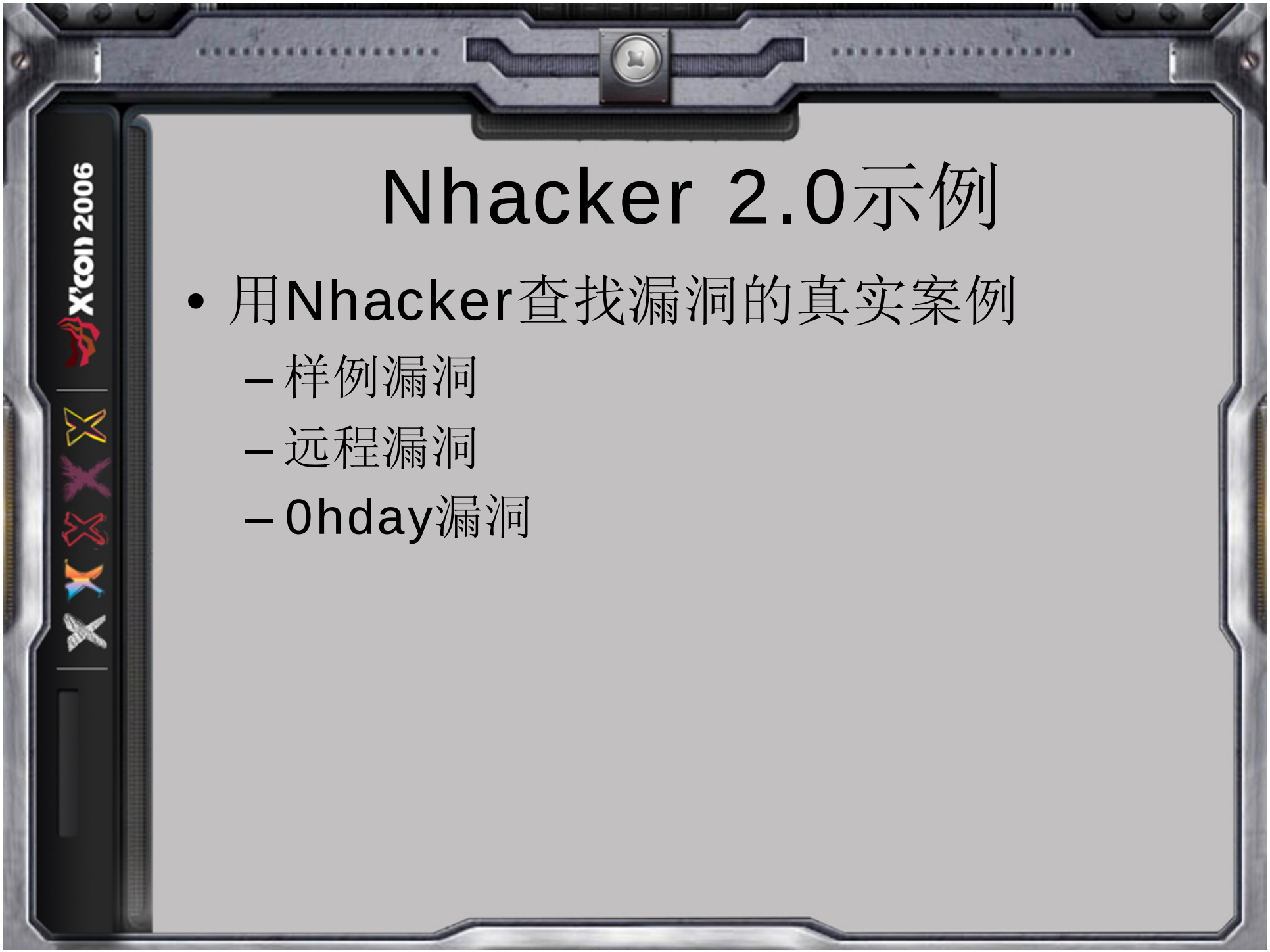
Nhacker 2.0解析器

- 你所见过最垃圾的代码
- 比openssl和sendmail都要糟糕
- 不过它能工作！



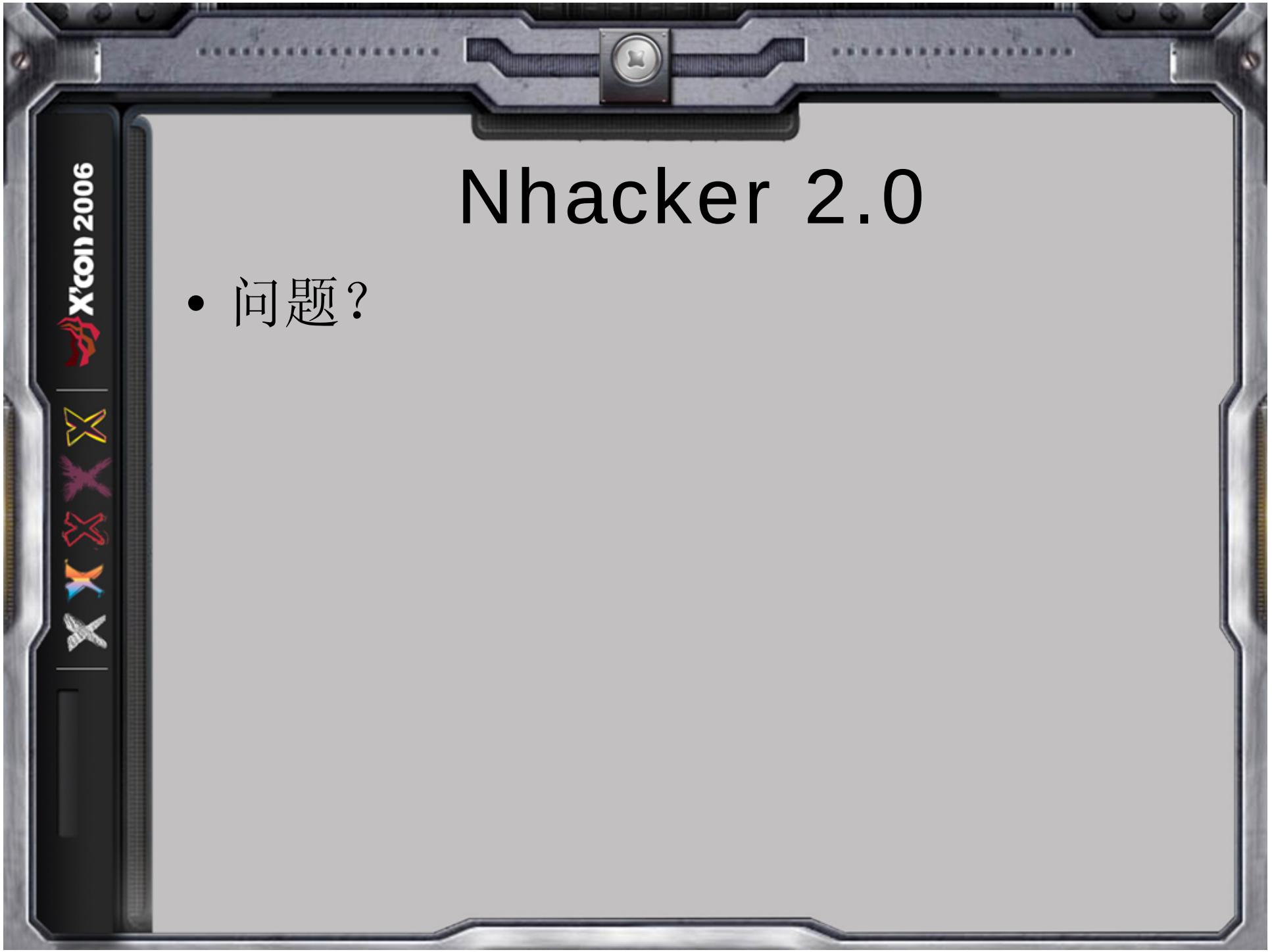
Nhacker 2.0解析器

- 让我们直接从程序中阅读一些解析器的代码



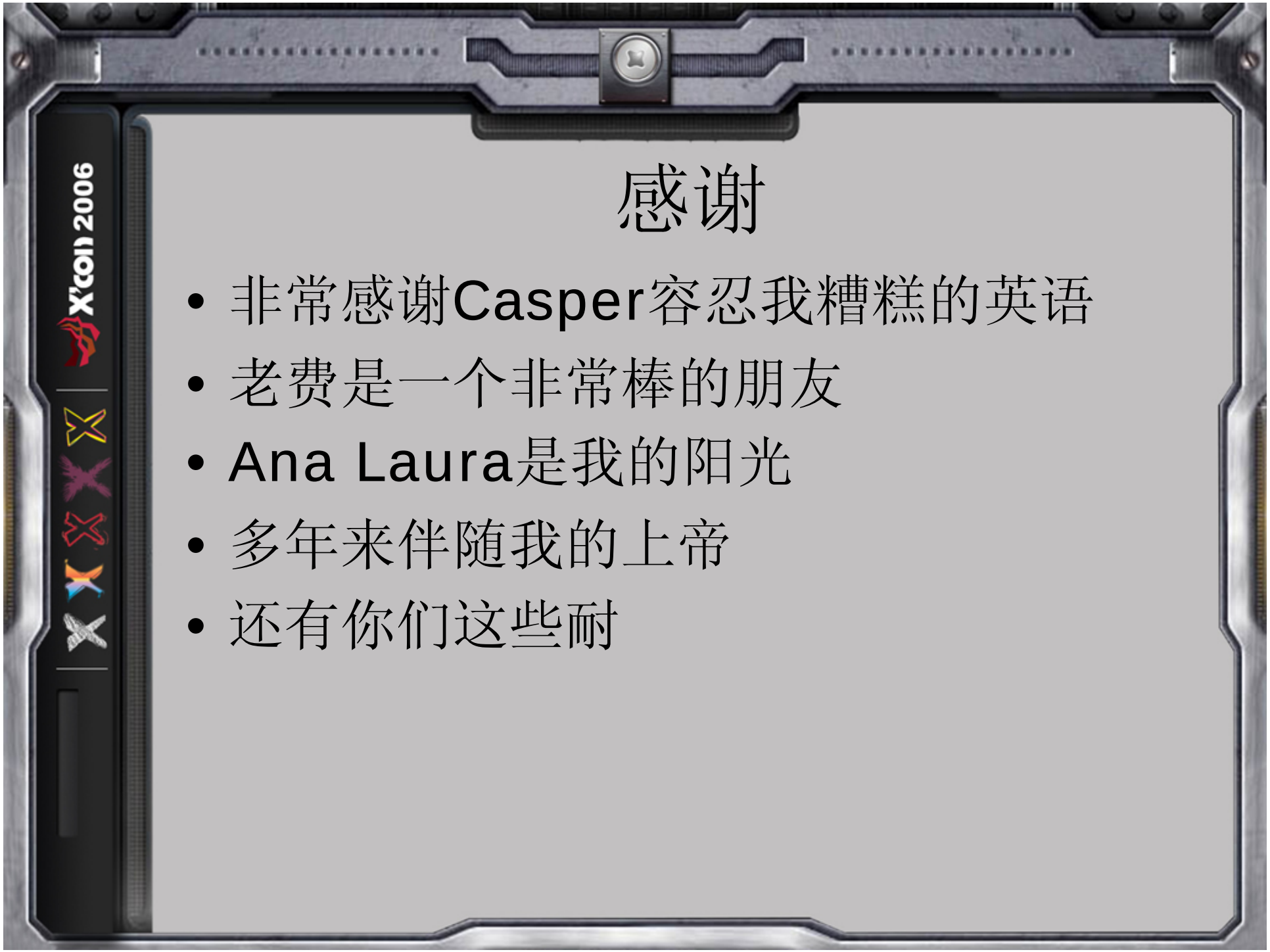
Nhacker 2.0示例

- 用Nhacker查找漏洞的真实案例
  - 样例漏洞
  - 远程漏洞
  - 0hday漏洞



Nhacker 2.0

- 问题？



感谢

- 非常感谢Casper容忍我糟糕的英语
- 老费是一个非常棒的朋友
- Ana Laura是我的阳光
- 多年来伴随我的上帝
- 还有你们这些耐