
从漏洞到利用代码为Metasploit写插件
Saumil Shah
X'Con2000

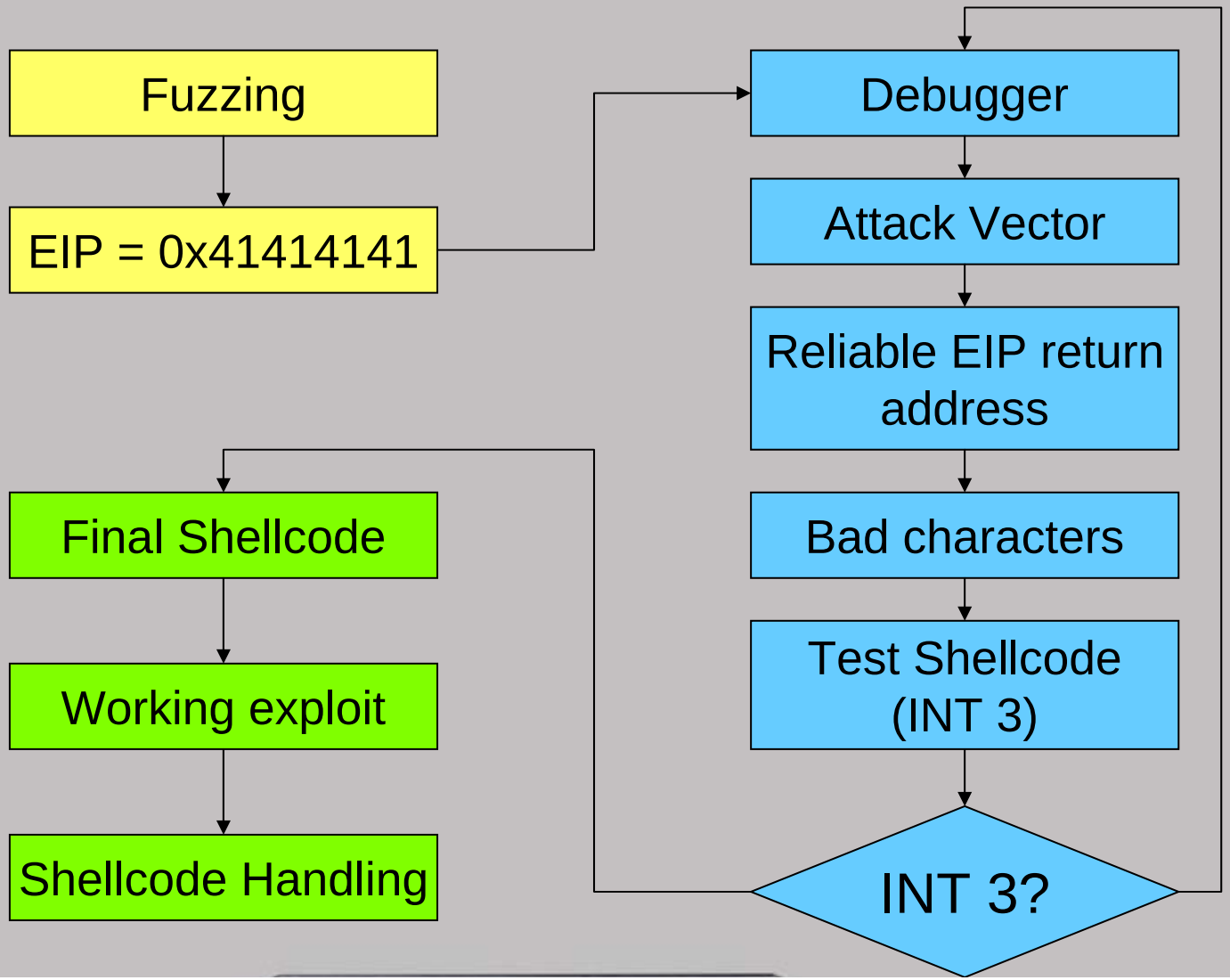
本人简介

```
# who am i
16:08  up  4:26, 1 user, load averages: 0.28 0.40 0.33
USER     TTY      FROM              LOGIN@  IDLE WHAT
saumil   console  -                 11:43    0:05 bash
```

- Saumil Shah - “krafty”
  ceo, net-square solutions
  saumil@saumil.net

  “WEB黑客 – 攻击和防御”的作者

从漏洞到利用


Fuzzing
EIP = 0x41414141
Debugger
Attack Vector
Reliable EIP return address
Bad characters
Test Shellcode (INT 3)
INT 3?
Final Shellcode
Working exploit
Shellcode Handling

CPU寄存器

- Intel 32-bit x86寄存器:

EAX	ESP
accumulator	stack pointer
EBX	EBP
base	base pointer
ECX	ESI
counter	source index
EDX	EDI
data	destination index

EIP

instruction pointer

进程内存映射

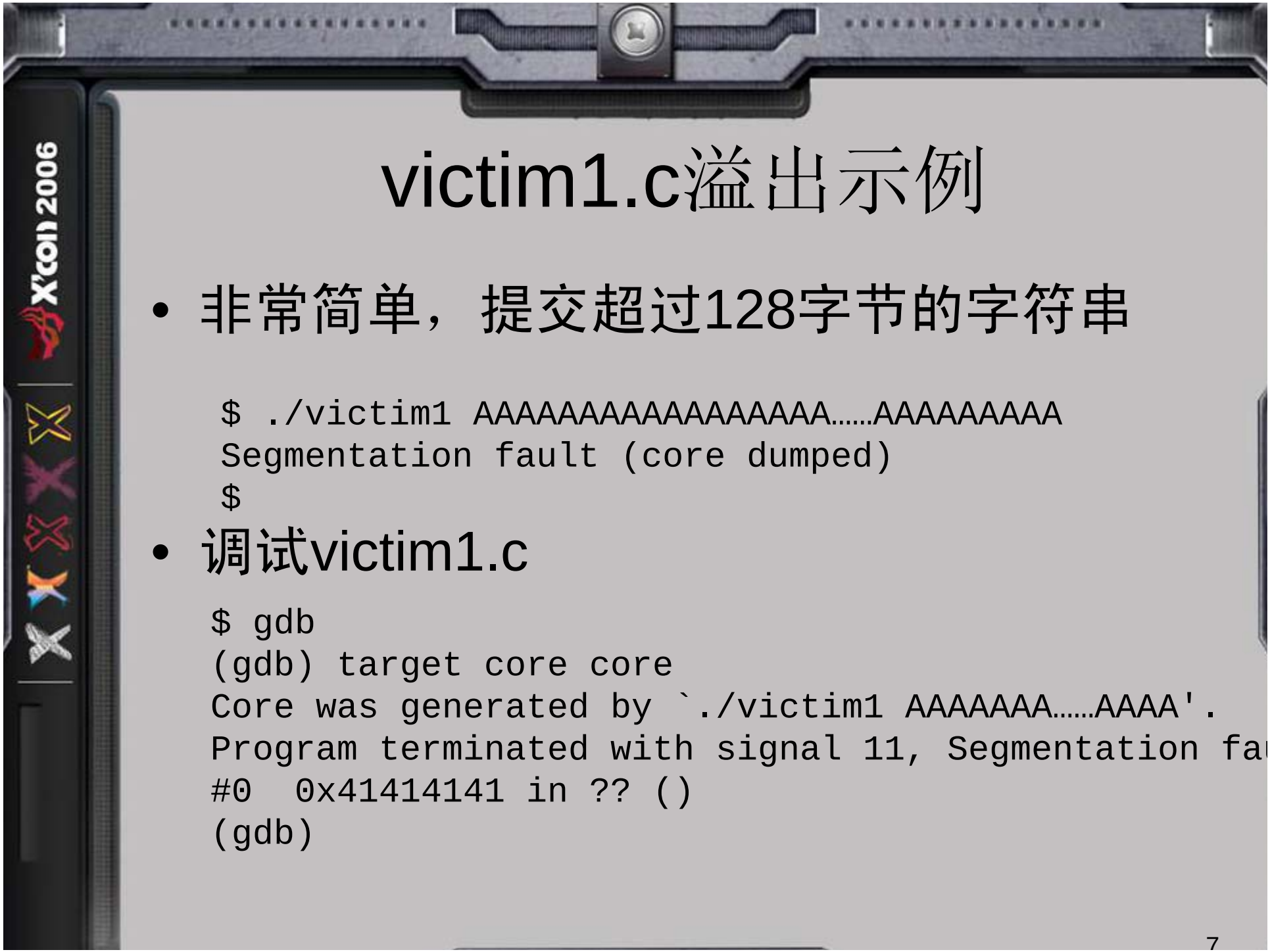

0x08000000
.text
.data
.bss
heap - malloc'ed data
…
v heap
^ stack
…
main() local vars
argc
**argv
**envp
cmd line arguments
environment vars
0xc0000000

栈溢出

- 错误条件：当大块数据尝试写入小的区块时(堆栈上的本地VAR).

```
char buffer[128];
strcpy(buffer, argv[1]);
```

- 如果 “argv[1]” 超过128字节将会发生什么?



victim1.c溢出示例

- 非常简单，提交超过128字节的字符串

```
$ ./victim1 AAAAAAAAAAAAAAAA……AAAAAAAAA
Segmentation fault (core dumped)
$
```

- 调试victim1.c

```
$ gdb
(gdb) target core core
Core was generated by `./victim1 AAAAAAA……AAAA'.
Program terminated with signal 11, Segmentation fa
#0  0x41414141 in ?? ()
(gdb)
```

范例victim1.c调试

- 栈溢出后的寄存器内容：

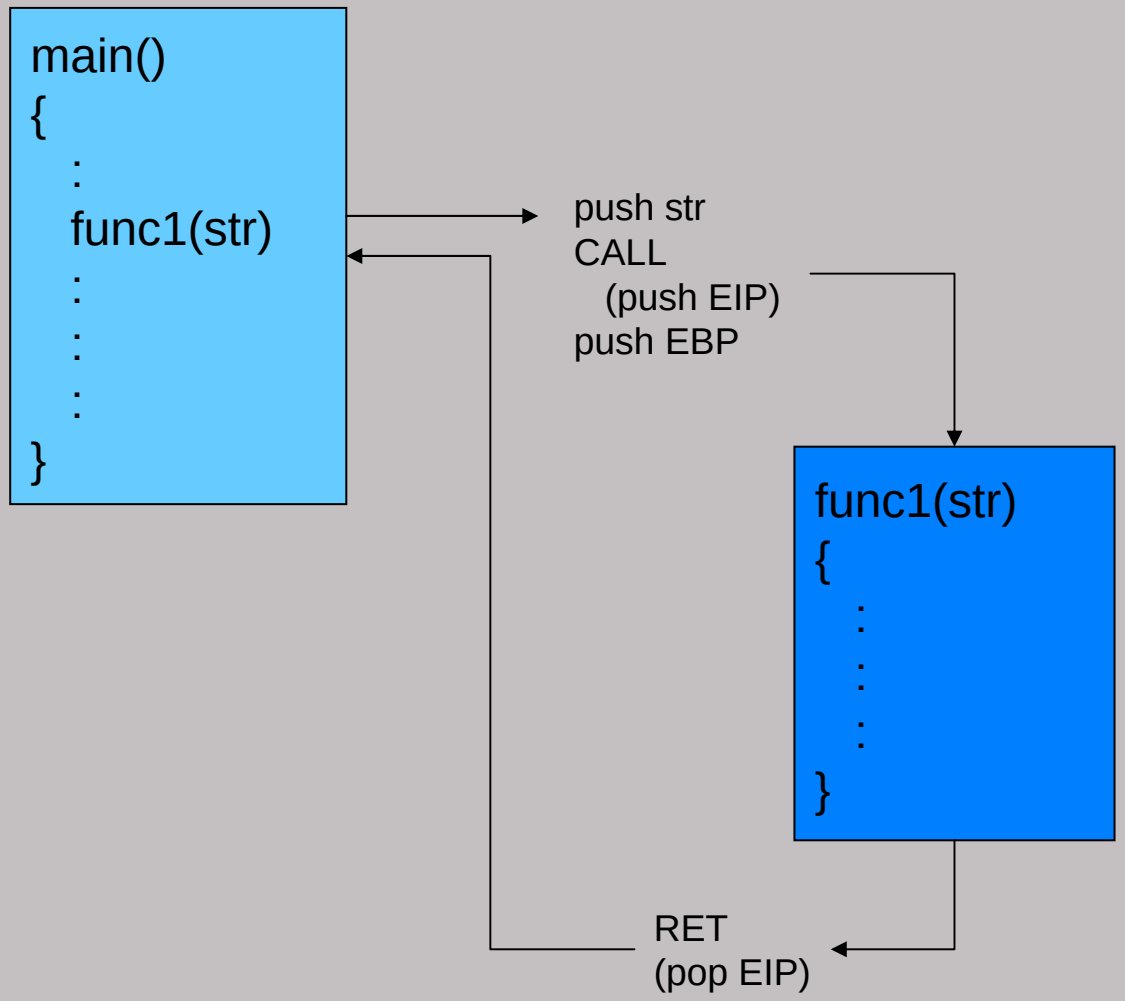
```
(gdb) info registers
esp            0xbffffb24       -1073743068
ebp            0x41414141       1094795585
esi            0x4000ae60       1073786464
edi            0xbffffb74       -1073742988
eip            0x41414141       1094795585
```

- EIP的值是0x41414141，也就是”AAAA”
- EIP被溢出缓冲区中的数据覆盖。

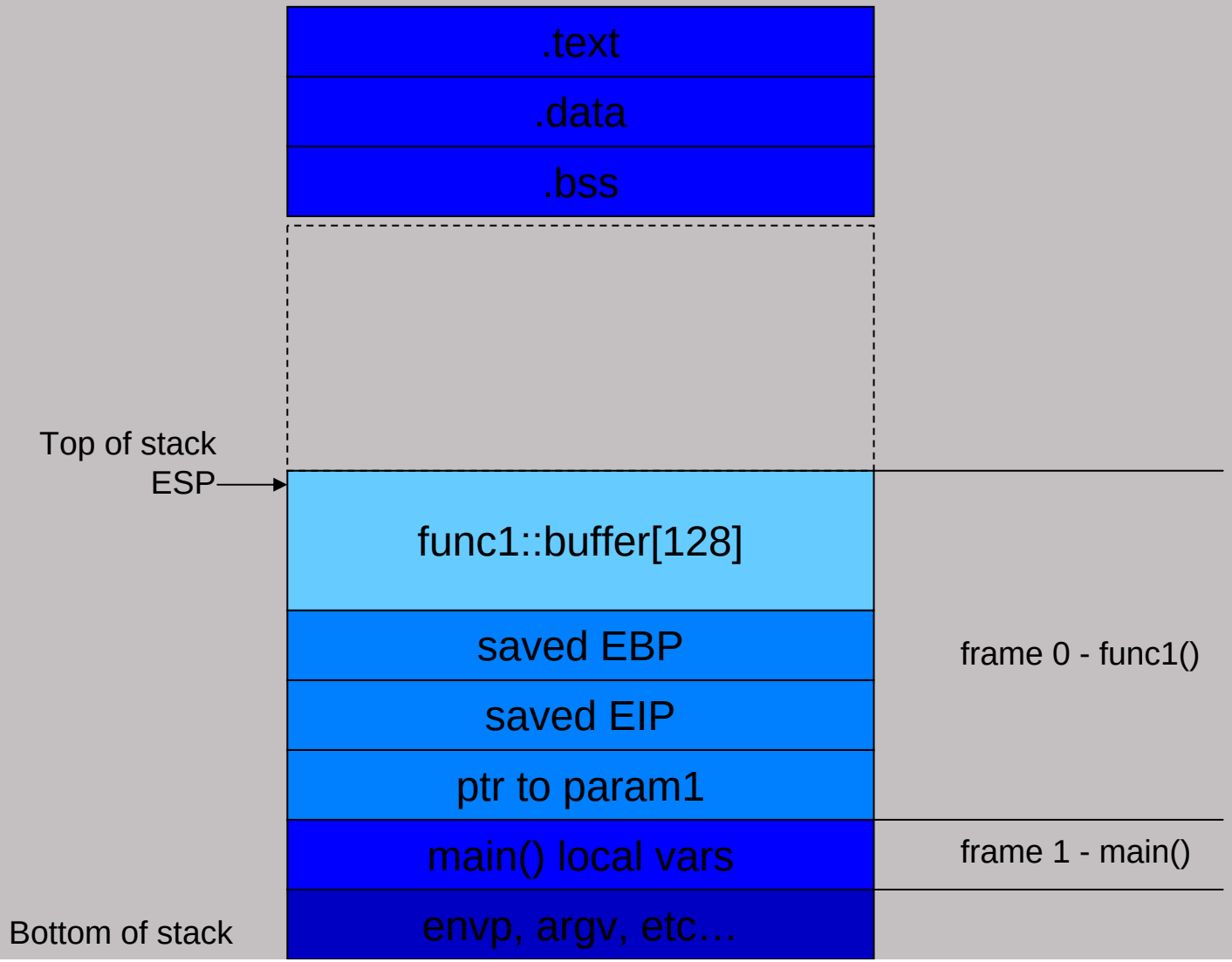
调用函数

- 当一个函数被调用时，以下内容将被压到堆栈中：
  - 函数参数
  - 保存的寄存器值，如EIP和EBP
- 当函数返回时，EIP从堆栈中弹出，恢复正常的程序流程。

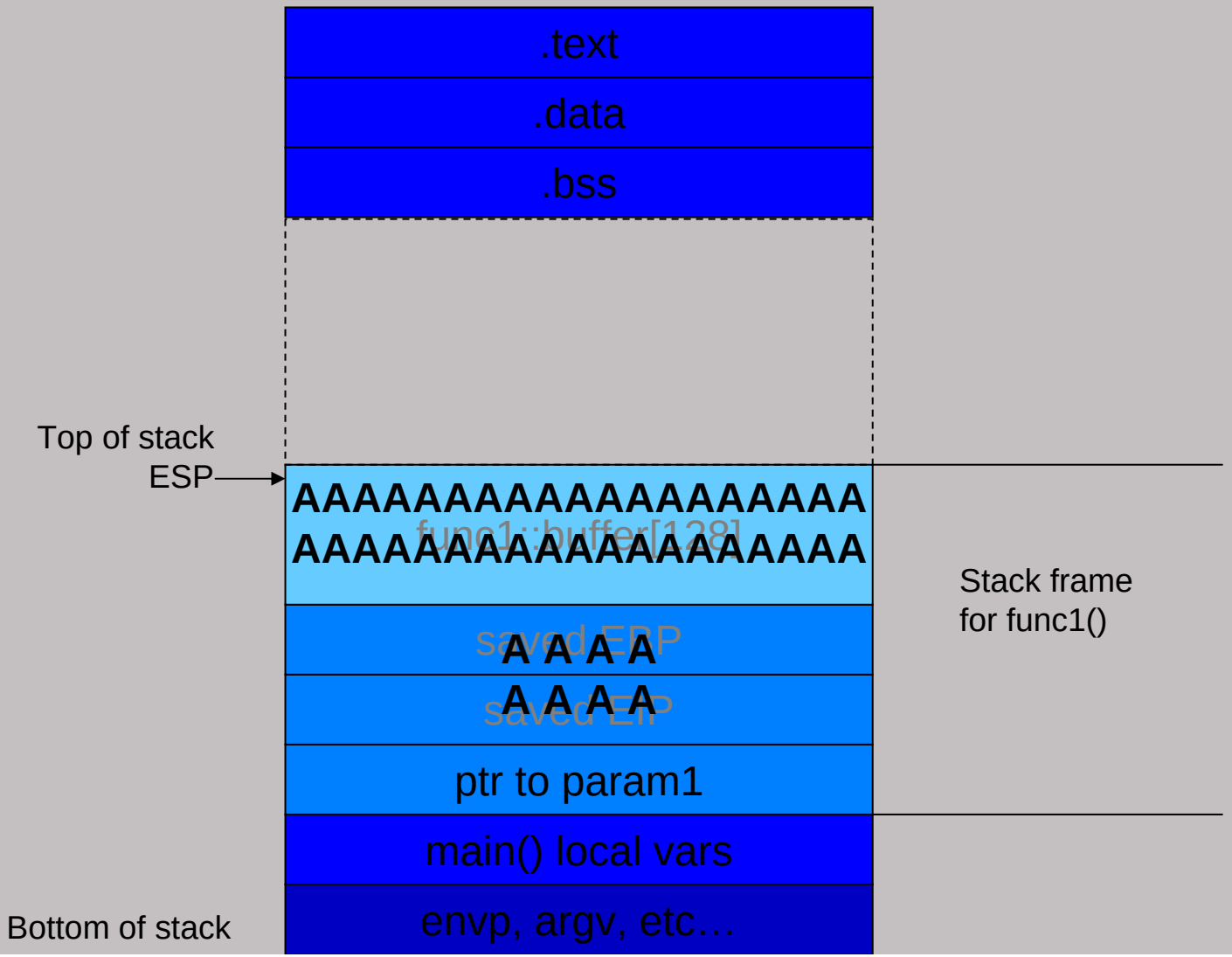
调用一个函数


main()
{
:
func1(str)
:
:
:
}
push str
CALL
(push EIP)
push EBP
func1(str)
{
:
:
:
}
RET
(pop EIP)

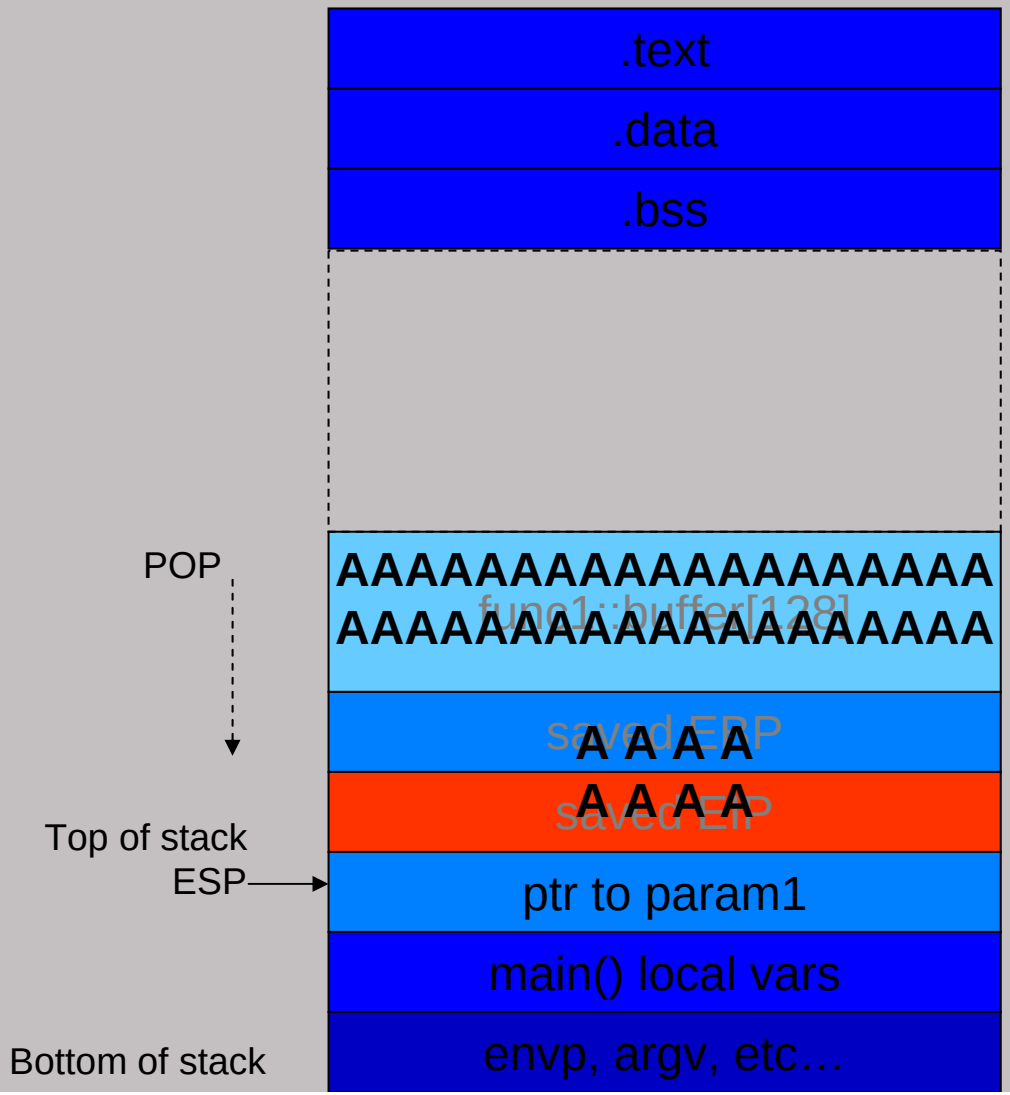
Victim内存映象-拷贝之前



Victim内存映象-拷贝之后



堆栈被溢出



当func1 返回
EIP 会弹出
EIP = 0x41414141
(“AAAA”)

堆栈溢出后的寄存器值

- 当func1() 返回，EIP和EBP从堆栈中弹出

```
(gdb) info registers
esp            0xbffffa24       -1073743324
ebp            0x41414141       1094795585
esi            0x4000ae60       1073786464
edi            0xbffffa74       -1073743244
eip            0x41414141       1094795585
```

- 我们可以控制指令指针

控制EIP

- 漏洞可导致EIP控制
- 我们可以设置指令指针转到任何我们所想的地方…
- 问题是“我们需要使它转到哪个地方”
- 我们可以注入自己的代码，使EIP跳转到指定的代码吗？
- 及我们应该把代码注入到哪里？

介绍Metasploit

- 是一个增加的利用代码研究和开发架构
- WEB地址是http://metasploit.com
- 当前稳定版本：2.6
  - 由PERL编写，运行在Unix和Win32(cygwin)
  - 160+可利用代码，77个攻击负载数据，13编码代码
  - 最新3,0 beta1
    - 完全由Ruby重写

介绍Metasploit

- 生成shellcocode
- Shellcode编码
- Shellcode处理程序
- 扫描程序中的特定指令：
  - 如POP/POP/RET, JMP ESI等
- 可增加定制利用代码，Shellcode，Encoders.
- 及更多

EIP = 0x41414141

- 我们怎么判断哪4个字节会覆盖EIP
- 使用轮转模式作为输入：

  Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1
  Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3
  Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5
  Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5……

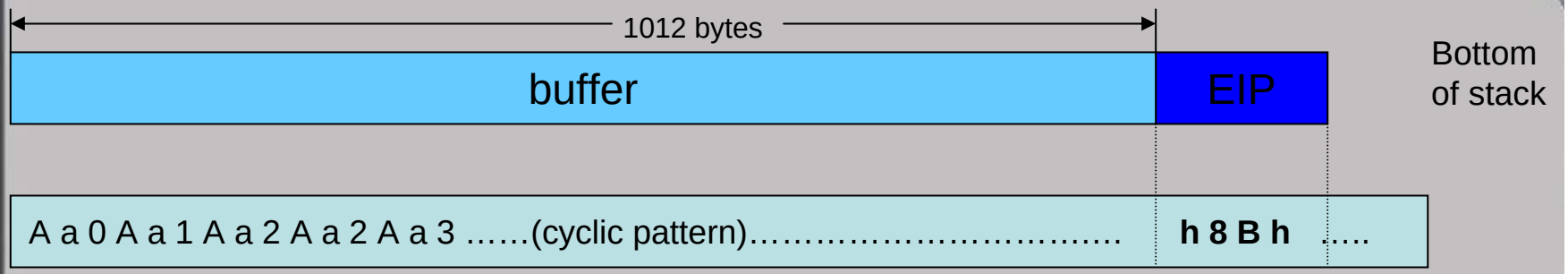
- Metasploit的 Pex::Text::PatternOffset()
- 生成模式，发现子字符串

到EIP的距离处理

- 使用Metasploit的patternOffset.pl

```
krafty:~/metasploit$ perl sdk/patternOffset.pl 0x68423768 2000
1012
```

- 基于EIP被什么值所覆盖，我们可以通过如下模式获得到EIP的距离：



获得对程序计数器的控制

- 栈溢出
  - 直接覆盖程序计数器
  - 异常处理结构覆盖
- 格式字符串错误
- 堆溢出
- 整数溢出

- 通过”是什么”和”在哪里“覆盖PC

输入Shellcode

- 使用CPU的自身指令集编写成的代码
- 作为被溢出缓冲区的一部分注入
- 注入代码最典型的功能是”开出SHELL“，也即是”Shellcode” 。
- 一段包含shellcode的缓冲区数据就一般称为”攻击负载”。

编写Shellcode

- 需要熟悉CPU自身指令集：
  - 如x86 (ia32), x86-64 (ia64), ppc, sparc 等
- 精确有效的汇编语言。
- OS特定的系统调用。
- Shellcode库和生成器。
- MetaSploit架构

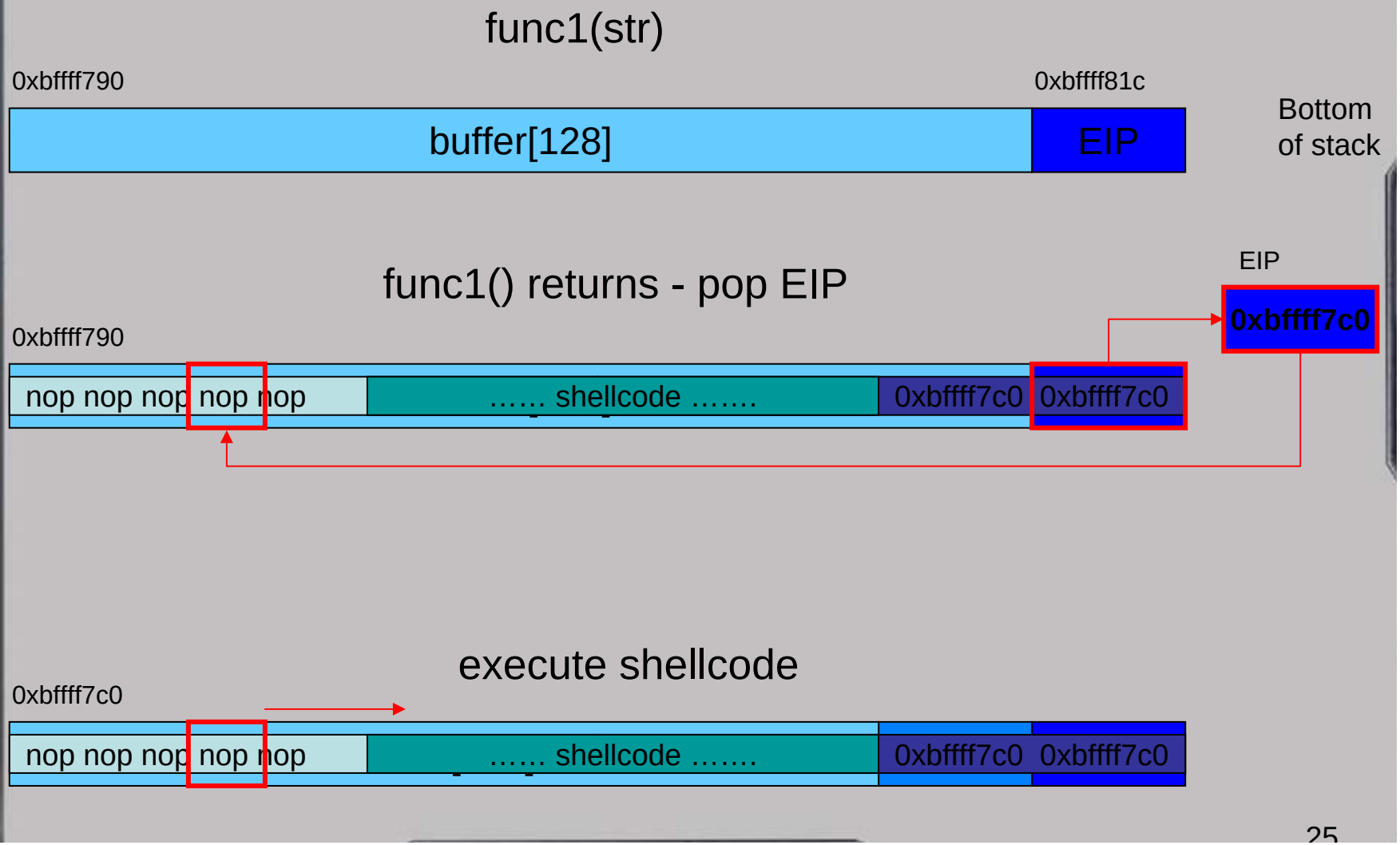
注入Shellcode

- 最方便的方法是直接把Shellcode塞在缓冲区溢出数据自身内。
- 放置在负载数据的某处
- 需要计算出会驻留在目标进程内存中何处

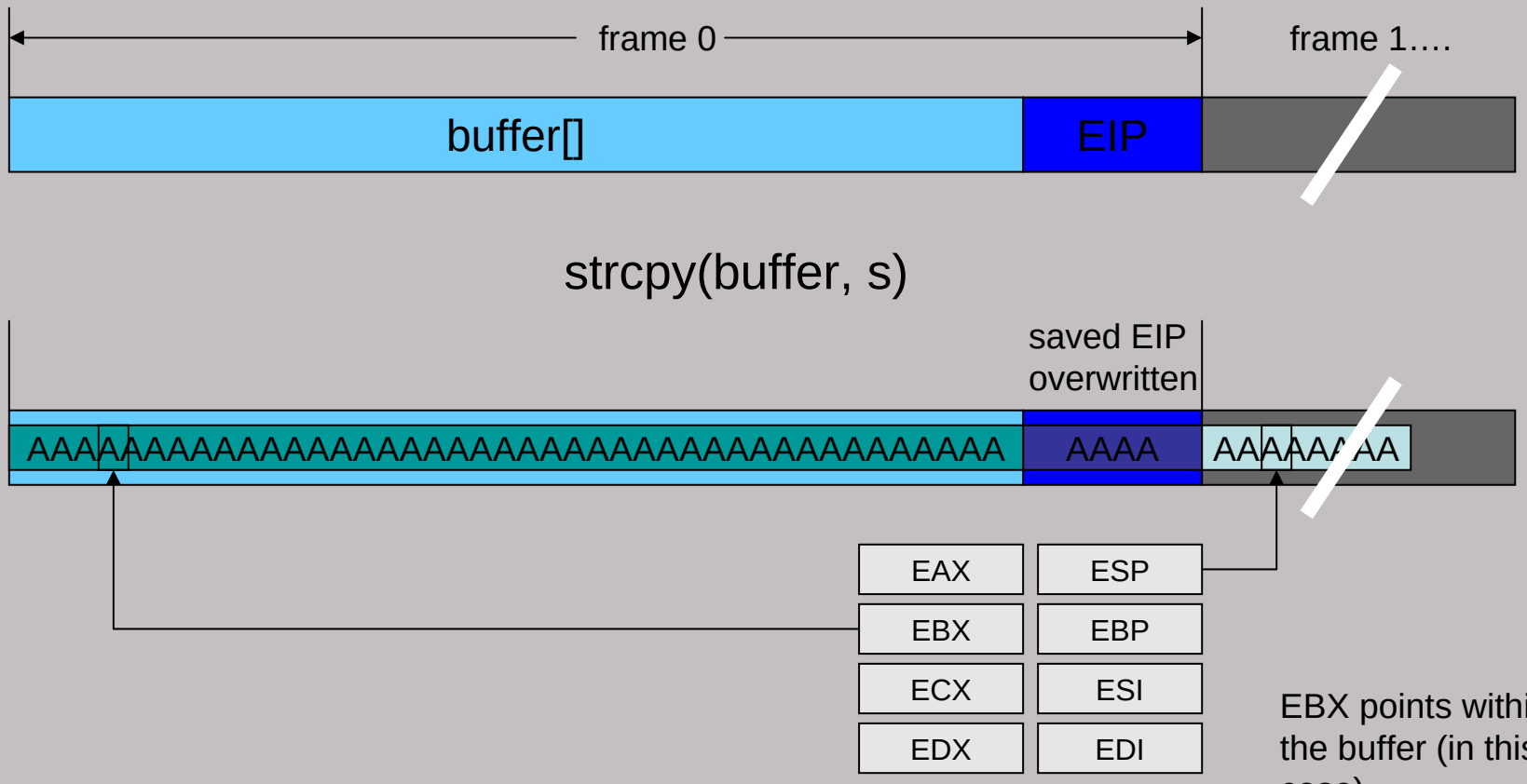
路想怎么走？

- EIP可以这样转向：
  - 返回到堆栈
    直接跳到攻击负载数据中
    （可靠性问题 – 地址不细定， 堆栈保护）
  - 返回到共享库
    通过寄存器跳转
    需要结合部分特定条件
    （比较稳定的技术）

跳回到堆栈


func1(str)
0xbffff790
0xbffff81c
buffer[128]
EIP
Bottom of stack
func1() returns - pop EIP
EIP
0xbffff7c0
0xbffff790
nop nop nop nop nop
…… shellcode …….
0xbffff7c0
0xbffff7c0
execute shellcode
0xbffff7c0
nop nop nop nop nop
…… shellcode …….
0xbffff7c0
0xbffff7c0

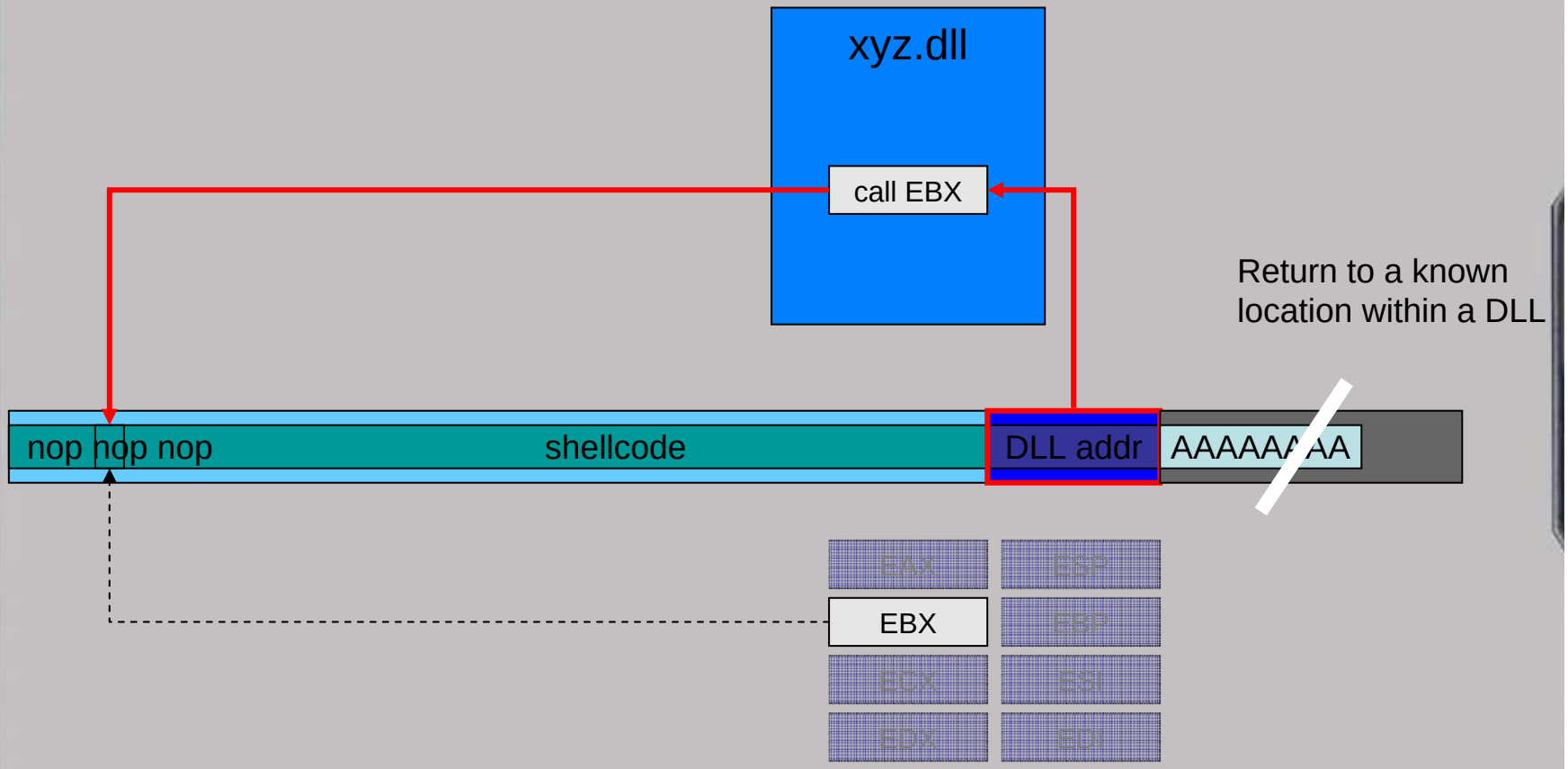
通过寄存器跳转



EBX points within the buffer (in this case)

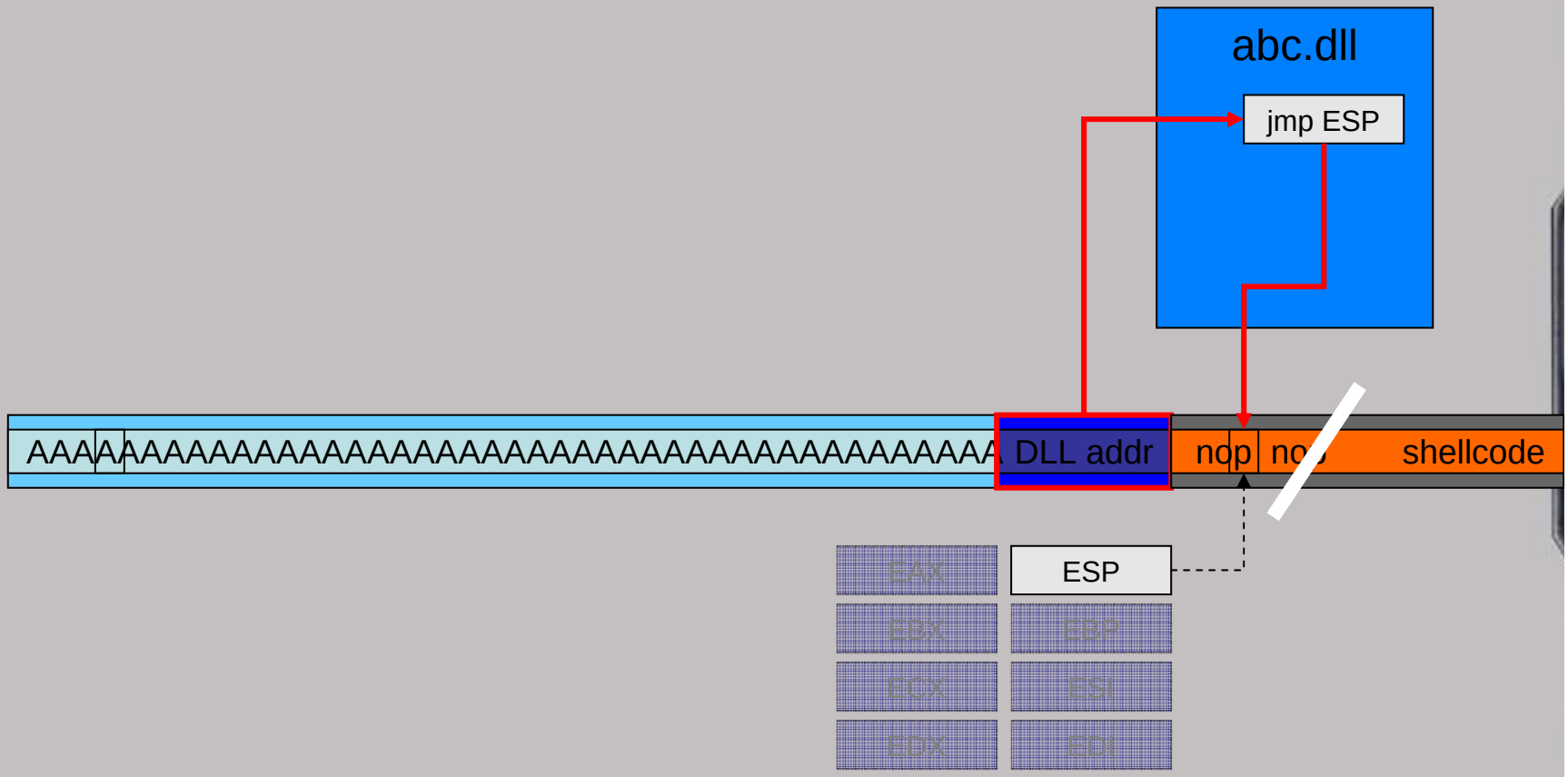
ESP points beyond the saved EIP

通过寄存器跳转



shellcode 在缓冲区开始位置

通过寄存器跳转



shellcode 在缓冲区结尾位置

查找CALL或者JMP指令

- 我们需要从内存中查找包含CALL或者JMP指令所在的固定地址
- 进程内存中共享库一般在固定地址装载
- 理想的情况下是找到CALL，JMP指令
- 我们可以尝试使用调试器使用手工模式使用操作码来搜索
- 或者我们使用msfpescan或msfelfscan

msfpescan, msfelfscan

- 扫描两进制的工具(可执行执行程序或共享库)。
- 支持ELF和PE两进制程序
- 使用metasploit内置的反汇编器
- 可以查找CALL，JMP或者POP/POP/RET指令集。
- 可以用于查找指定规则表达式的指组。

Windows DLLs中使用msfpescan

- 如果我们需要搜索JMP到ESI寄存器：

```
~/framework$ ./msfpescan -f windlls/USER32.DLL -j esi
0x77e11c46    call esi
0x77e121b7    call esi
0x77e121c5    call esi
0x77e1222a    call esi
:         :    :     :
0x77e6ca97    jmp esi
```

- 我们需要把EIP指到任意值...
- 及使它之后执行JMP/CALL ESI

搜索两进制的候选次序

- 最先的是搜索可执行程序自身
  - 独立于内核，服务补丁包，LIB
- 其次是搜索软件自身的共享库或者DLL(如Winamp中的in_mp3.dll)
- 最后一般搜索操作系统包含的默认共享库
  - 如kernel32.dll, libc.so等
  - 或使可利用代码基于操作系统或特定服务包。

示例 - peercast HTTP 溢出

- 1000字节攻击负载
- 开始780字节为AAAA
- 781-784必须包含转到EIP的地址。
- 785字节后数据包含Shellcode.

Shellcode 知识点滴

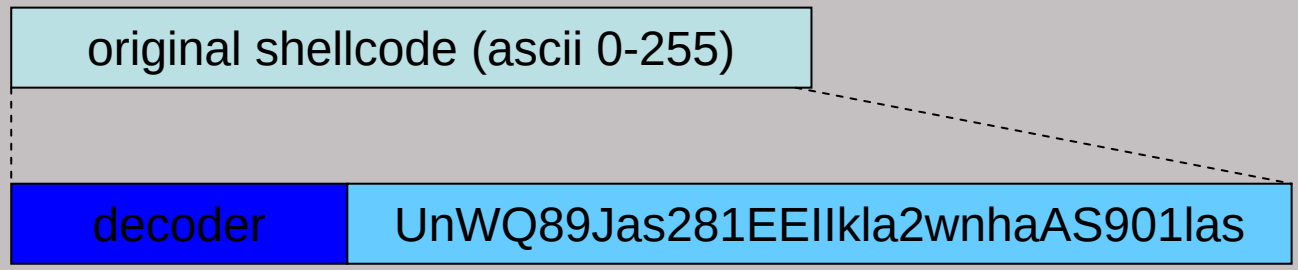
- Shellcode类型：
  - 绑定SHELL
  - 执行命令
  - 反向SHELL
  - 分段SHELL等
- 高级技术：
  - Meterpreter
  - 上传并在进程内运行DLL
  - 等等

负载数据编码

- 负载数据编码根据部分标准建立编码的Shellcode
- 如Alpha2生成仅含字符和数字合成shellcode
- 允许绕过任何协议解析/字节过滤解析
- 增加一个解码小程序在Shellcode开始处
  - 字节可能增加

负载编码

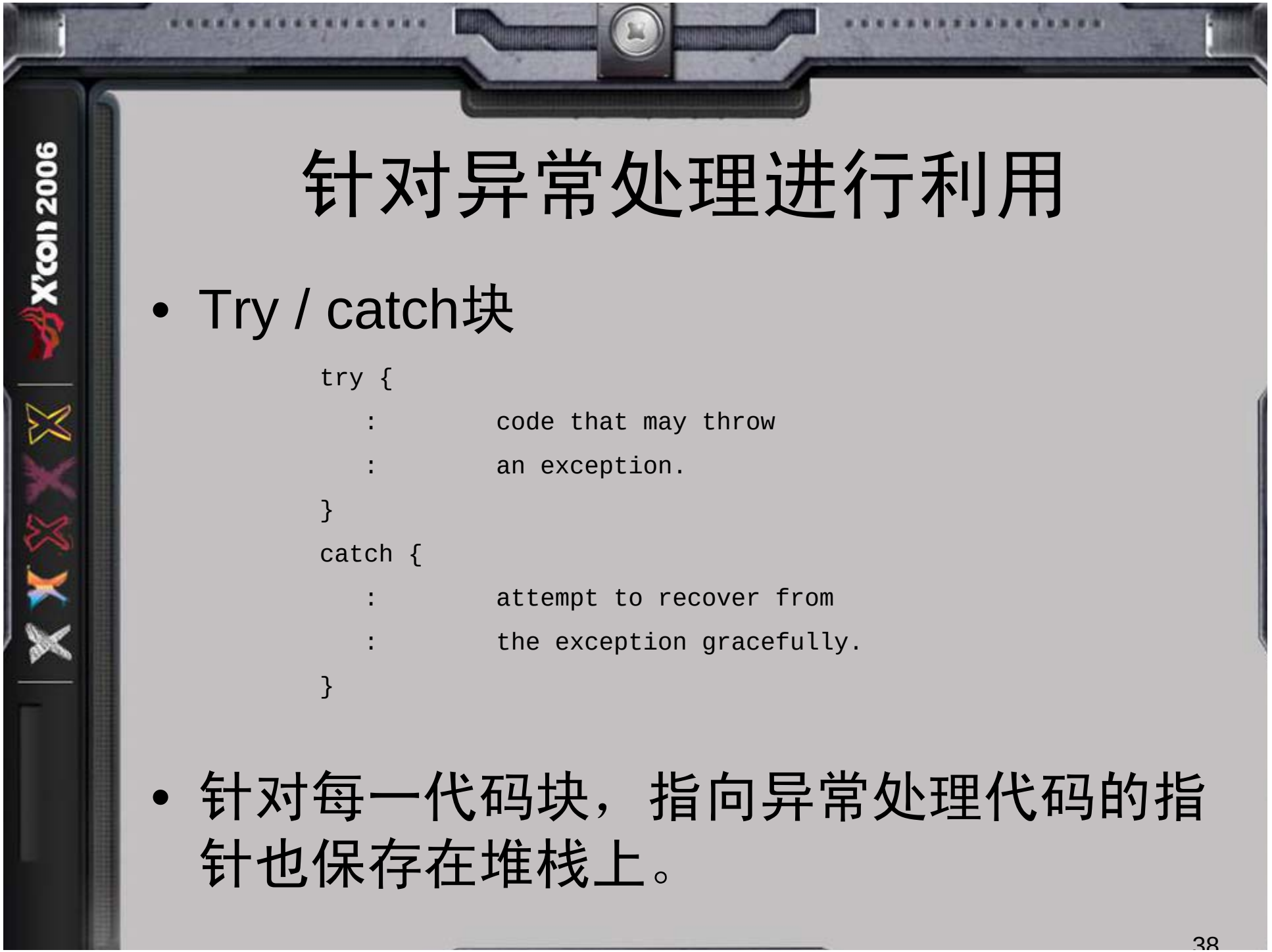
- 例如：Alpha2编码



- 转换原始负载为可见字符式shellcode
- 解码器在内存中解码负载

负载编码器

- Metasploit提供多种类型的编码器
- 能针对协议解析进行工作
  - 如避免CR，LF，NULL
  - toupper(), tolower(),等
- 可绕过IDS检测
  - 多态Shellcode
  - Shikata Ga Nai



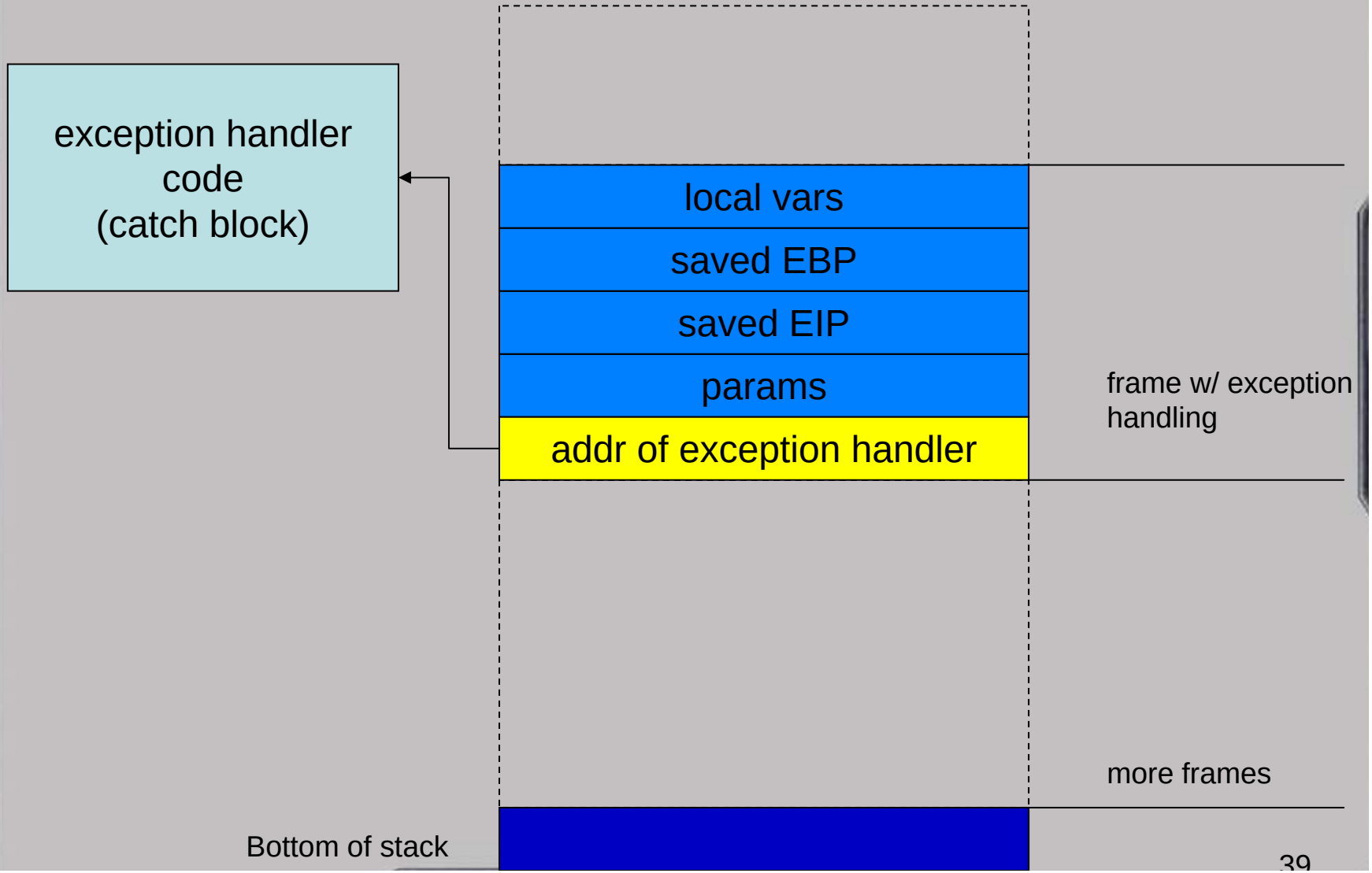
针对异常处理进行利用

- Try / catch块

```
try {
   :        code that may throw
   :        an exception.
}
catch {
   :        attempt to recover from
   :        the exception gracefully.
}
```

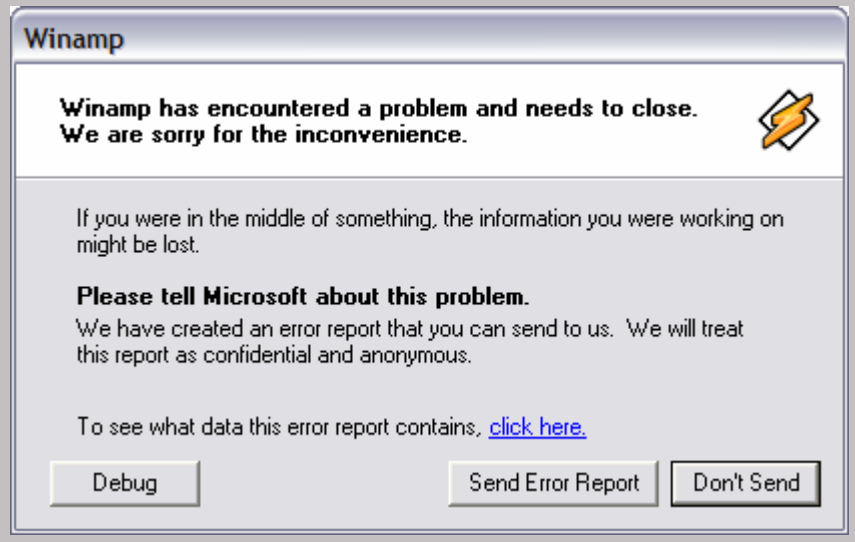
- 针对每一代码块，指向异常处理代码的指针也保存在堆栈上。

异常处理…实现


exception handler
code
(catch block)
local vars
saved EBP
saved EIP
params
addr of exception handler
frame w/ exception
handling
more frames
Bottom of stack

Windows SEH

- SEH – 结构化异常处理
- Windows弹出对话窗口：



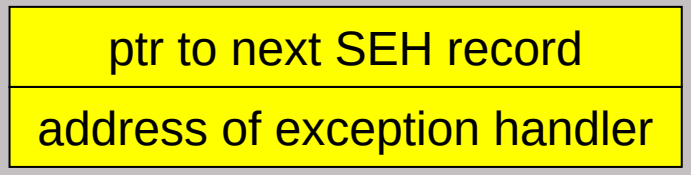
- 由默认处理机制处理

自定义异常处理

- 默认SEH必须作为最后一着处理
- 多种语言包括C++提供异常处理编码功能
- 编译器生成链接并依照操作系统调用异常处理代码
- 在Windows中，异常处理程序在堆栈中形成链接的列表链

SHE记录

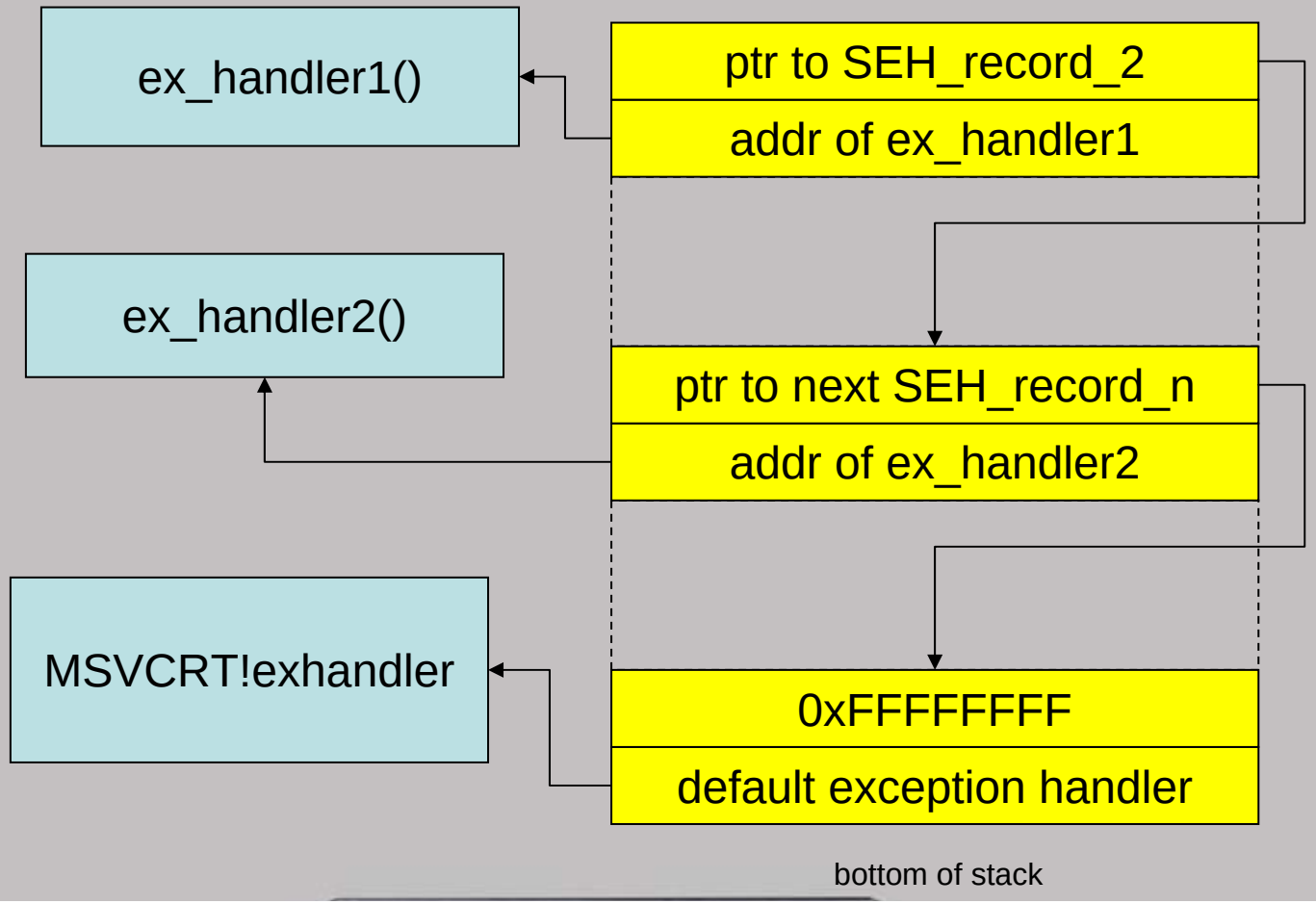
- 每个SHE记录为8字节



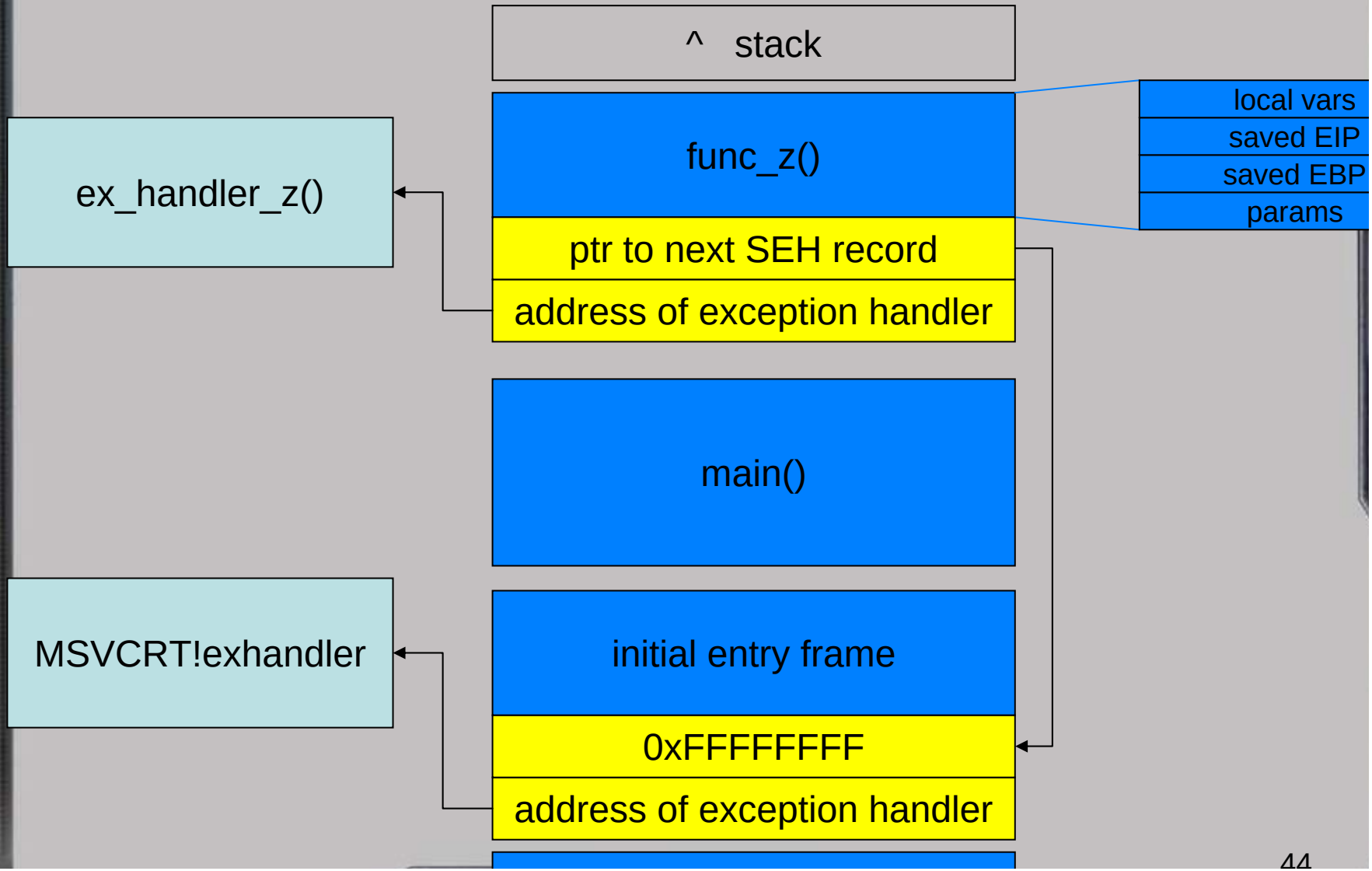
- 这些SHE记录基于堆栈
- 在函数依次调用后，散布在函数(块)帧中
- WinDBG命令 - !exchain

SHE 链

- 每个SHE记录为8字节



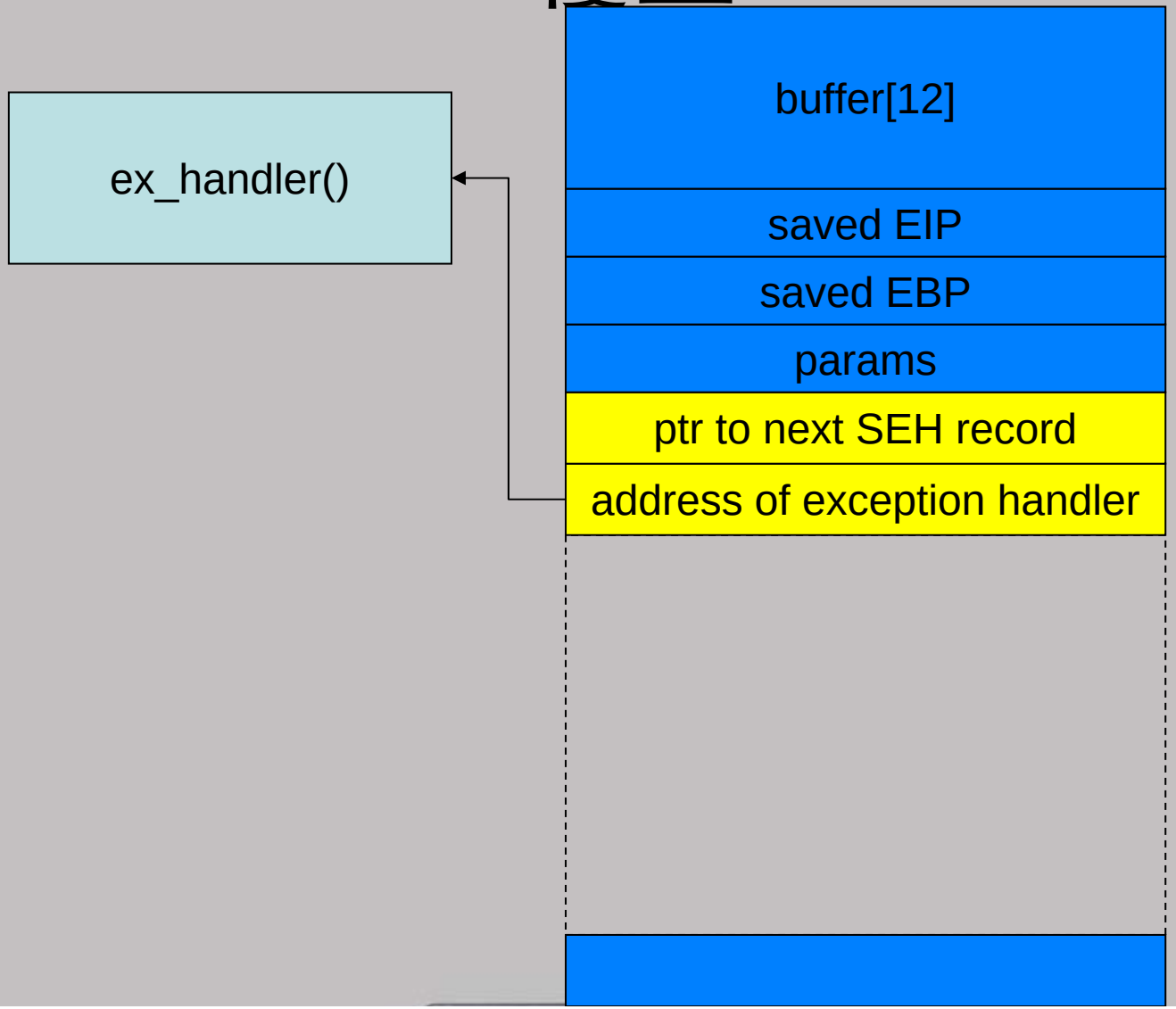
堆栈中的SEH


^ stack
func_z()
local vars
saved EIP
saved EBP
params
ex_handler_z()
ptr to next SEH record
address of exception handler
main()
MSVCRT!exhandler
initial entry frame
0xFFFFFFFF
address of exception handler

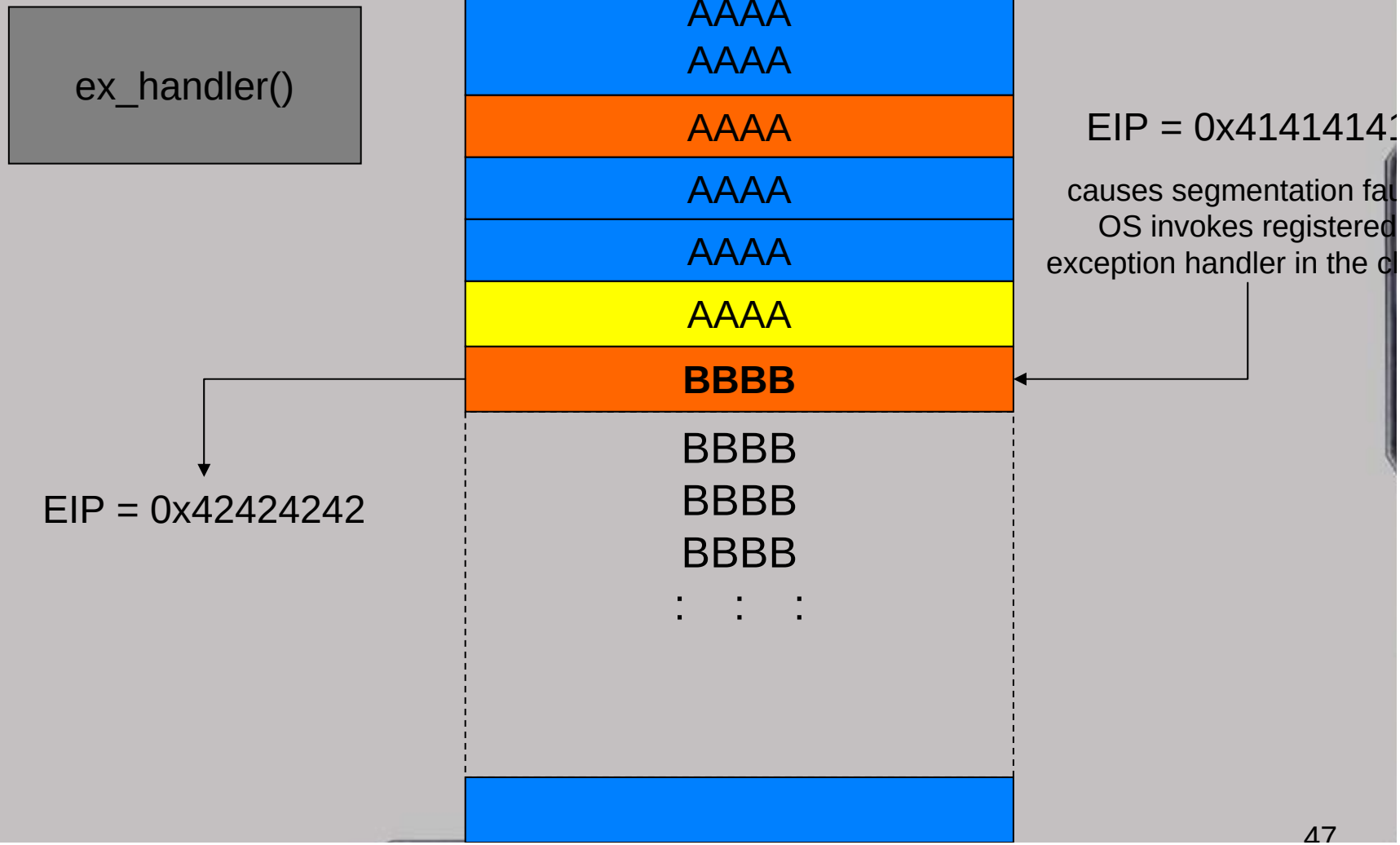
还有另一个方法获得EIP

- 覆盖寄存的异常处理器的其中一个地址...
- 及使进程抛出异常
- 如果没有自定义异常处理器注册，覆盖默认SHE
- 或许必须往下回溯堆栈...
- 但这样做需要比较长的缓冲区

覆盖SEH


ex_handler()
buffer[12]
saved EIP
saved EBP
params
ptr to next SEH record
address of exception handler

覆盖SEH


ex_handler()
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
BBBB
BBBB
BBBB
BBBB
EIP = 0x41414141
causes segmentation fau
OS invokes registered
exception handler in the c
EIP = 0x42424242

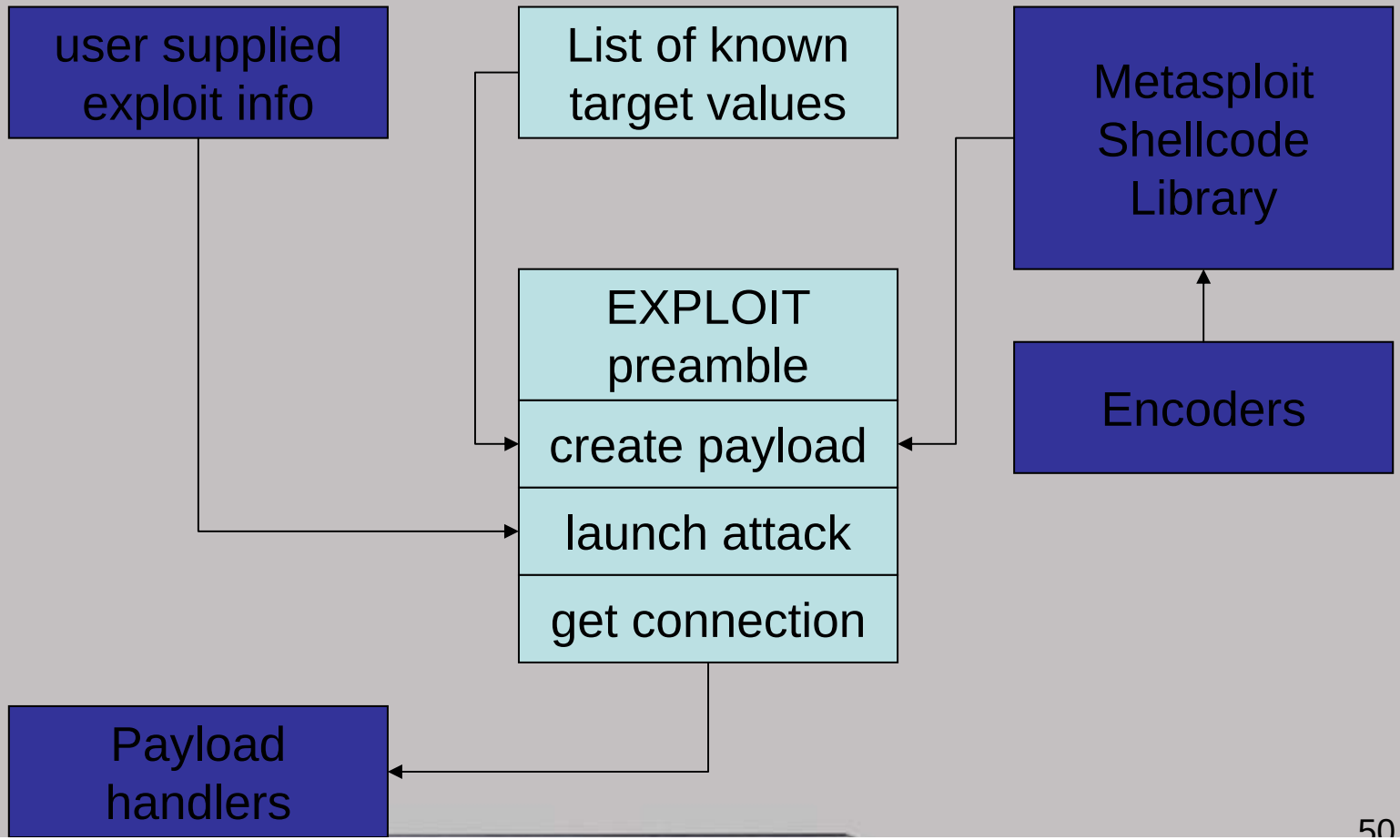
范例学习 – sipXtapi CSeq溢出

- sipXtapi库 – 流行的开放源代码VoIP库
- 使用在多个软电话中
  - AOL Triton软电话程序使用sipXtapi
- 在CSeq SIP头数据中有24字节缓冲区溢出。
- 此缓冲区目前来说可实用shellcode来讲太小
- 我们可以通过覆盖SHE来处理

Metasploit可利用代码模块编写

- 集成在Metasploit架构中
- 支持多个目标系统
- 动态攻击负载选择。
- 动态攻击负载编码。
- 内置攻击负载处理器。
- 可以使用更先进的攻击负载。
- ..可移植，扩展性好

Metasploit怎样运行一个利用代码



编写一个Metasploit利用代码

- Perl 模块 (2.6), Ruby模块(3.0)
- 预置数据结构
  - %info, %advanced
- 构造器
  - sub new {...}
- 利用代码
  - sub Exploit {...}

Perl模块下的利用代码结构

```
package Msf::Exploit::name;
use base “Msf::Exploit”;
use strict;
use Pex::Text;

my $advanced = { };

my $info = { };

sub new {

}

sub Exploit {

}
```

information block

constructor
return an instance of our exploit

exploit block

%info 数据

- Name
- Version
- Authors
- Arch
- OS
- Priv
- UserOpts
- Payload
- Encoder
- Refs
- DefaultTarget
- Targets
- Keys

Metasploit Pex

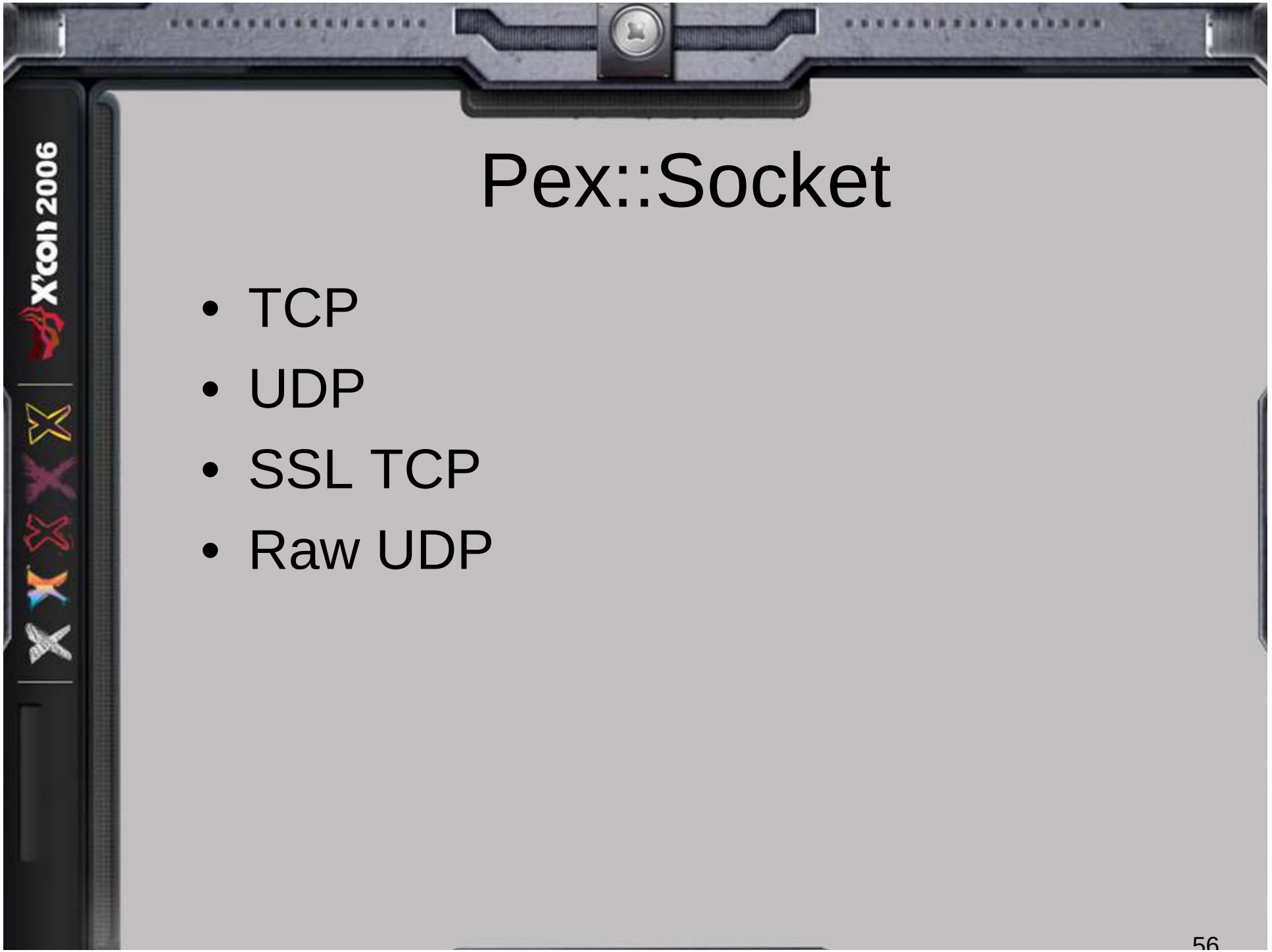
- Perl扩展

  <metasploit_home>/lib/Pex.pm

  <metasploit_home>/lib/Pex/
- 文本处理函数
- 套接字管理函数
- 特定协议函数
- 以上这些及更多的函数可供我们编写可利用代码

Pex::Text

- 编码和解码(如Base64)
- 模式生成
- 随机文本生成(逃避IDS检测)
- 额外数据填补
- 等等



Pex::Socket

- TCP
- UDP
- SSL TCP
- Raw UDP

Pex – 特定协议工具

- SMB
- DCE RPC
- SunRPC
- MSSQL
- …etc

Pex – 各种杂项工具

- Pex::Utils
- 数组和哈希操作
- 位轮循
- 读和写文件
- 格式串生成
- 建立Win32 PE文件
- 建立Javascript数组
- 及其他！

metasploit_skel.pm

- 这是个skeleton可利用代码生成模块
- 接但介绍
- 可以使用skeleton编写可利用代码模块
- 可把完成的可利用模块放置到如下位置
  <path_to_metasploit>/exploits/

已经完成模块范例

- my_peercast.pm
- my_sipxtapi.pm

部分Metasploit命令行工具

- Msfcli
  - Metasploit命令行接口
  - 可在非交互行为中使用metasploit架构的操作
- msfpayload
  - 使用特定选项生成攻击负载
- msfencode
  - 编码生成攻击负载

更多Metasploit命令行工具

- msfweb
  - Metasploit架构的WEB接口
- Msfupdate
  - Metasploit架构在线升级

新版本3.0中的命令行工具

- Msfd
  - Metasploit守护进程，允许客户端-服务器之间操作。
- Msfopcode
  - Metasploit在线操作码数据库的命令行接口
- Msfwx
  - 使用wxruby的GUI接口

新版本3.0的其他特性

- 增加了新的攻击负载，新的编码器
- Ruby扩展 – Rex (类似Pex)
- NASM shell.
- 后台数据库支持
- 及一些其他不错的特性和工具

感谢！

Saumil Shah

saumil@saumil.net

http://net-square.com

+91 98254 31192