



CTF Workshop

Seshagiri Prabhu

The Stack

Process

CTF Workshop

Seshagiri Prabhu

Day 3

Amrita Vishwa Vidyapeetham
Amritapuri

June 13, 2014



Outline

CTF Workshop

Seshagiri Prabhu

The Stack
Process

1 The Stack

2 Process



The stack

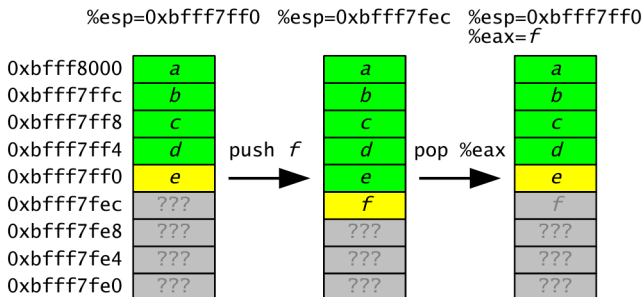
CTF Workshop

Seshagiri Prabhu

The Stack

Process

- The stack grows towards lower memory addresses
- The stack pointer (`%esp`) points to the top of the stack (the lowest valid address)





Frames & function call I

CTF Workshop

Seshagiri Prabhu

The Stack

Process

- The stack is composed of frames
- Frames are pushed on the stack as a consequence of function calls (function prologue)
- The address of the current frame is stored in the Frame Pointer (FP) register
 - On Intel architectures `%ebp` is used
- Each frame contains
 - The function's actual parameters
 - The return address to jump to at the end of the function
 - The pointer to the previous frame
 - Function's local variables
- Compiler optimization may eliminate stack frames



Frames & Func call II

CTF Workshop

Seshagiri Prabhu

The Stack

Process

```
int convert(char *str) {
    int result = atoi(str); return result;
}

int main(int argc, char **argv) {
    int i, sum;
    for (i = 0; i < argc; i++) sum += convert(
        argv[i]);
    printf("sum=%d\n", sum); return 0;
}
```

0xbfff8000	argv	}	pushed by main's caller
0xbfff7ffc	argc		
0xbfff7ff8	return address from main		
0xbfff7ff4	frame pointer before main was called		
0xbfff7ff0	sum	}	pushed by main
0xbfff7fec	i		
0xbfff7fe8	str		
0xbfff7fe4	return address from convert (to main)		
0xbfff7fe0	frame pointer before convert was called (0xbfff7ff4)	}	pushed by convert
0xbfff7fdc	result		
0xbfff7fe0	parameter to atoi		
0xbfff7fdc	return address from atoi (to convert)		



Function prologue

CTF Workshop

Seshagiri Prabhu

The Stack

Process

- Before calling a function the caller prepares the parameters by either setting specific registers or by pushing them on the stack
- The prologue of the called function
 - Pushes the current base pointer on the stack
 - Sets the base pointer to be the current stack pointer
 - Moves the stack pointer onward to make room for local variables

```
push %ebp  
mov %esp, %ebp  
sub $n, %esp
```



Function Epilogue

CTF Workshop

Seshagiri Prabhu

The Stack

Process

- The epilogue of the called function
 - Saves the results (if any) in the `%eax` register
 - Stores the base pointer into the stack pointer (deletes the current stack frame)
 - Pops a value from the stack, restoring the saved base pointer
 - Executes a `ret`
- The second and third operations are equivalent to the `LEAVE` opcode



Example

CTF Workshop

Seshagiri Prabhu

The Stack

Process

```
void function (char *str) {  
    char buffer[8];  
    strcpy(buffer, str);  
    return;  
}
```

```
main:  
    sub $0x4,%esp  
    movl $0x8048560, (%esp)  
    call 0x8048444 <function>  
    ...
```

```
function:  
    push %ebp  
    mov %esp, %ebp  
    sub $0x10, %esp  
    ...  
    leave
```




A Program's Life

CTF Workshop

Seshagiri Prabhu

The Stack

Process

- Design
- Development, usually in a high-level language
- Compilation/translation into binary form (object form) by a compiler or assembler
- Programs in interpreted languages are translated into an intermediate format
- Execution by a process
- Termination



Objects

CTF Workshop

Seshagiri Prabhu

The Stack

Process

- Object files include applications and libraries
- Object files in general contain:
 - The code, in binary format
 - Relocation information about things that need to be fixed once the code and the data are loaded into memory
 - Information about the symbols defined by the object file and the symbols that are imported from different objects
 - Optionally, debugging information



Linking

CTF Workshop

Seshagiri Prabhu

The Stack

Process

- Linking is the process of resolving references that a program has to external objects (variables, functions)
- Static linking is performed at compile-time
- Dynamic linking is performed at run-time



The ELF File Format I

CTF Workshop

Seshagiri Prabhu

The Stack

Process

- The Executable and Linkable Format (ELF) is one of the most used binary object formats
- ELF files are of three types
 - relocatable: need to be fixed by the linker before being executed
 - executable: ready for execution (all symbols have been resolved with the exception of those related to shared libs)
 - shared objects: shared libraries with the appropriate linking information



The ELF File Format II

CTF Workshop

Seshagiri Prabhu

The Stack

Process

- A program is seen as a collection of segments by the loader and as a collection of sections by the compiler/linker
- A segment is usually made of several sections
- The segment structure is defined in the Program Header Table
- The section structure is defined in the Section Header Table



Process Structure

CTF Workshop

Seshagiri Prabhu

The Stack

Process

- Environment/Argument section
 - Used for environment data
 - Used for the command line data
- Stack section
 - Used for local parameters
 - Used for saving the processor status
- Heap section
 - Used for dynamically allocated data
- Data section (Static/global vars)
 - initialized variables (.data)
 - Uninitialized variables (.bss)
- Code/Text section (.text)
 - Usually marked read-only
 - Modifications causes segfaults



PLT and GOT

CTF Workshop

Seshagiri Prabhu

The Stack

Process

- When a shared library function is called by a program the address called is an entry in the Procedure Linking Table (PLT)
- The address contains an indirect jump to the addresses contained in variables stored in the Global Offset Table
- The first time a function is called, the GOT address is a jump back to the PLT, where the code to invoke the linker is called
- The linker does its magic and updates the GOT entry, so next time the function is called it can be directly invoked
- The PLT is read-only, but the GOT is not...
 - By overwriting the contents of a GOT entry it is possible to jump to arbitrary locations



CTF Workshop

Seshagiri Prabhu

The Stack

Process



The examples and slides

```
https://bitbucket.org/seshagiriprabhu/  
binary-exploitation-and-buffer-overflow
```




CTF Workshop

Seshagiri Prabhu

The Stack

Process

Questions ?

Seshagiri Prabhu

seshagiriprabhu@gmail.com