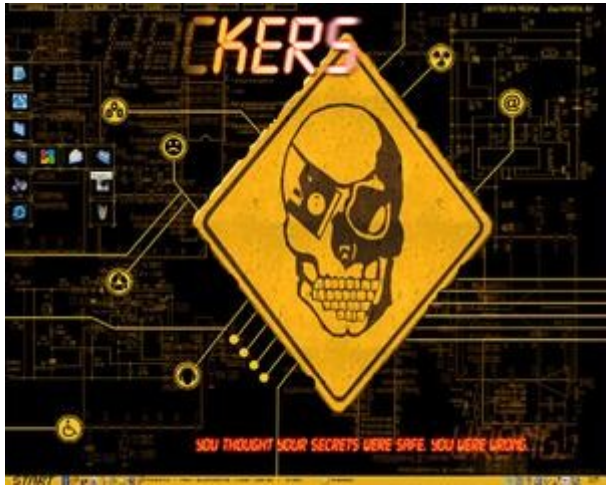
CSCD 303
Fall 2017



Lecture 15
Buffer Overflows

Introduction

Buffer overflow

What's a buffer overflow?

How do attackers exploit buffer overflows?

Potential solutions

Vulnerabilities are Everywhere

- Some vulnerabilities show up repeatedly
  - OS's, Web Applications, Databases, ClientServer applications
  - Which vulnerability is the most important today?
    - Buffer Overflows
      - Have been around since late 70's
      - Most important still most prevalent vulnerability

Buffer Overflow Statistics

Statistics on importance of buffer overflows ...

6/2009 – 9/2009

60% of the CERT advisories dealt with buffer overflows

Client side software at risk, more than OS

Adobe PDF, Quicktime, Flash, MS Office

90% of attacks on Microsoft targets buffer overflows

Specifically, MS08-067 - stack based buffer overflow

http://www.crn.com/security/220000395

Buffer Overflow Statistics

- Statistics on importance of buffer overflows ...

Study looked at 25 years of data – 1988-2012

Buffer overflows were the clear winner when the CVSS data was narrowed to vulnerabilities with **"high"** severity ratings.

Of the almost 16,000 CVSS vulnerabilities rated "high", there were 5528 (about 1/3) buffer overflow vulnerabilities - more than a third of the total.

In contrast, there were only 141 XSS vulnerabilities

https://www.veracode.com/blog/2013/02/at-the-vulnerability-oscars-the-winner-is-buffer-overflow

Buffer Overflow Attacks

- Introduction
  - Overflow attacks that come in from network, still considered an application or host type of attack
    - However, still relevant to understanding both network and host based security
    - Many network problems result of this vulnerability
  - Something that IDS's try to detect
  - Not detected by packet filter firewalls

Buffer Overflow Attacks

- Best known Example through History
  - Internet Worm – Robert Morris – 1988
    - Used a buffer overflow in fingerd on Unix systems

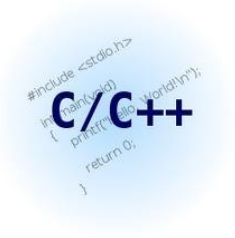
      fingerd - gives information about users
    - Ran by default on all Unix machines
    - Invaded 1000's of machines, about 10% Internet

Famous Buffer Overflow Attacks

- Besides Morris Worm ....
- CodeRed (2001): **overflow in MS-IIS server**
  - 300,000 machines infected in 14 hours
- SQL Slammer (2003): **overflow in MS-SQL server**
  - 75,000 machines infected in **10 minutes** (!!)
- Sasser (2004): **overflow in Windows LSASS**
  - Around 500,000 machines infected
- Conficker (2008-09): **overflow in Windows Server**
  - Around 10 million machines infected
- Is still around ....
  - **Why are buffer overflows still around today?**

Buffer Overflow Attacks

- Still around because ...



  - Foremost
  - Bad language design
    - C and C++, even Java
  - Poor programming practice
    - Programmers still don't know how to correctly obtain input
  - More Applications are ONLINE
    - More applications are Web based and require user input
    - SQL user-based queries, input on Web forms

Buffer Overflow Attacks

- What is the goal for attackers running buffer overflows?
  - In **Unix/Linux**, setuid programs
    - Run with root privileges
    - If regular user runs setuid program, still runs with root privilege
    - If attacker causes buffer overflow in setuid program, gain root privilege

      Examples: xterm, lpr, eject

      all have had buffer overflow attacks

Buffer Overflow Attacks

- Attacker Goals
  - Gain access to your machine
    - Once they are in, run a local buffer overflow to escalate privilege
  - Can then install other means for future access via a backdoor
  - Currently, exporting user info via back doors for money, installing adware and fake virus sites

http://blogs.zdnet.com/security/?p=4388

Buffer Overflow Attack

- History …. When did this attack become popular?
  - Known since 1988, but one paper described it in detail
  - In 1996, Aleph One wrote paper
    - "Smashing the Stack for fun and profit"

      http://insecure.org/stf/smashstack.html
  - Another Beginning tutorial

http://www.thegreycorner.com/2010/01/seh-stack-based-windows-buffer-overflow.html

  - Since then, others have published more details

    Wikipedia Collection of Papers and links

    http://en.wikipedia.org/wiki/Buffer_overflow

What Do you Need to Know
To Write One ?

- Understand C functions and the stack
- Some familiarity with machine code
- Know how system calls are made
  - The exec() system call in particular
- Attacker needs to know which CPU and OS are running on the target machine:
  - Our examples are for x86 running Linux

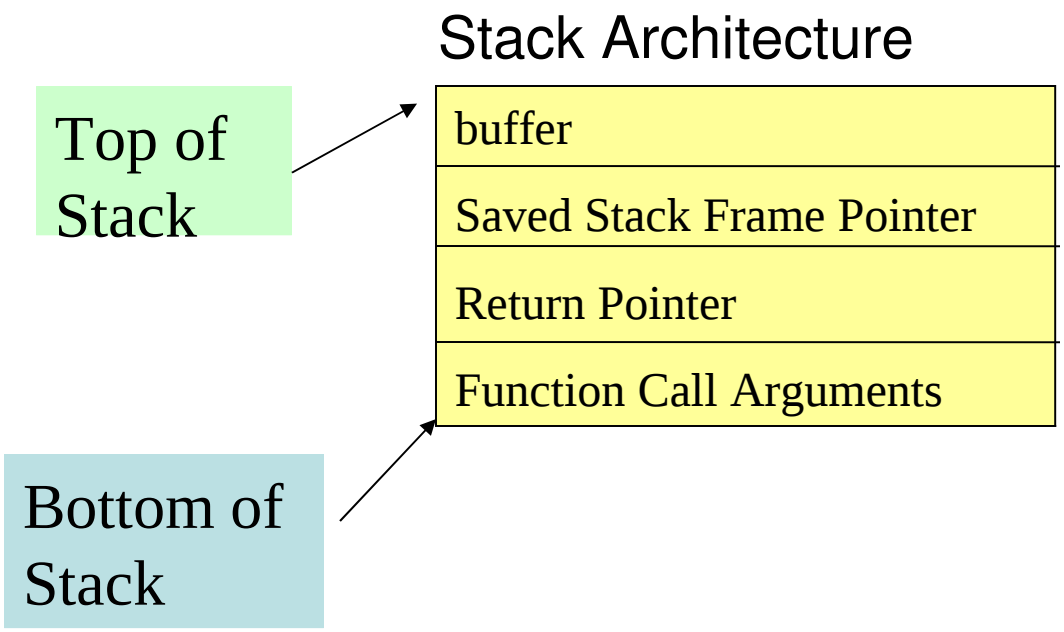
- Details vary slightly between CPUs and OSs:
  - Little endian vs. big endian (x86 vs. Motorola)
  - Stack Frame structure (Unix vs. Windows)
  - Stack growth direction

Buffer Overflow Attacks

- Overflowing the Stack – One way to do it
  - Review of the stack
    - Stack is a LIFO data structure
    - Temporary storage, running processes
    - When calling a function, OS pushes things onto stack so it can remember where it was prior to calling function

    Pushes address, Return Pointer, so it can go back to next instruction in main program
    - Also stores address, Saved Frame Pointer, that keeps track of stack memory, returns frame back to way it was

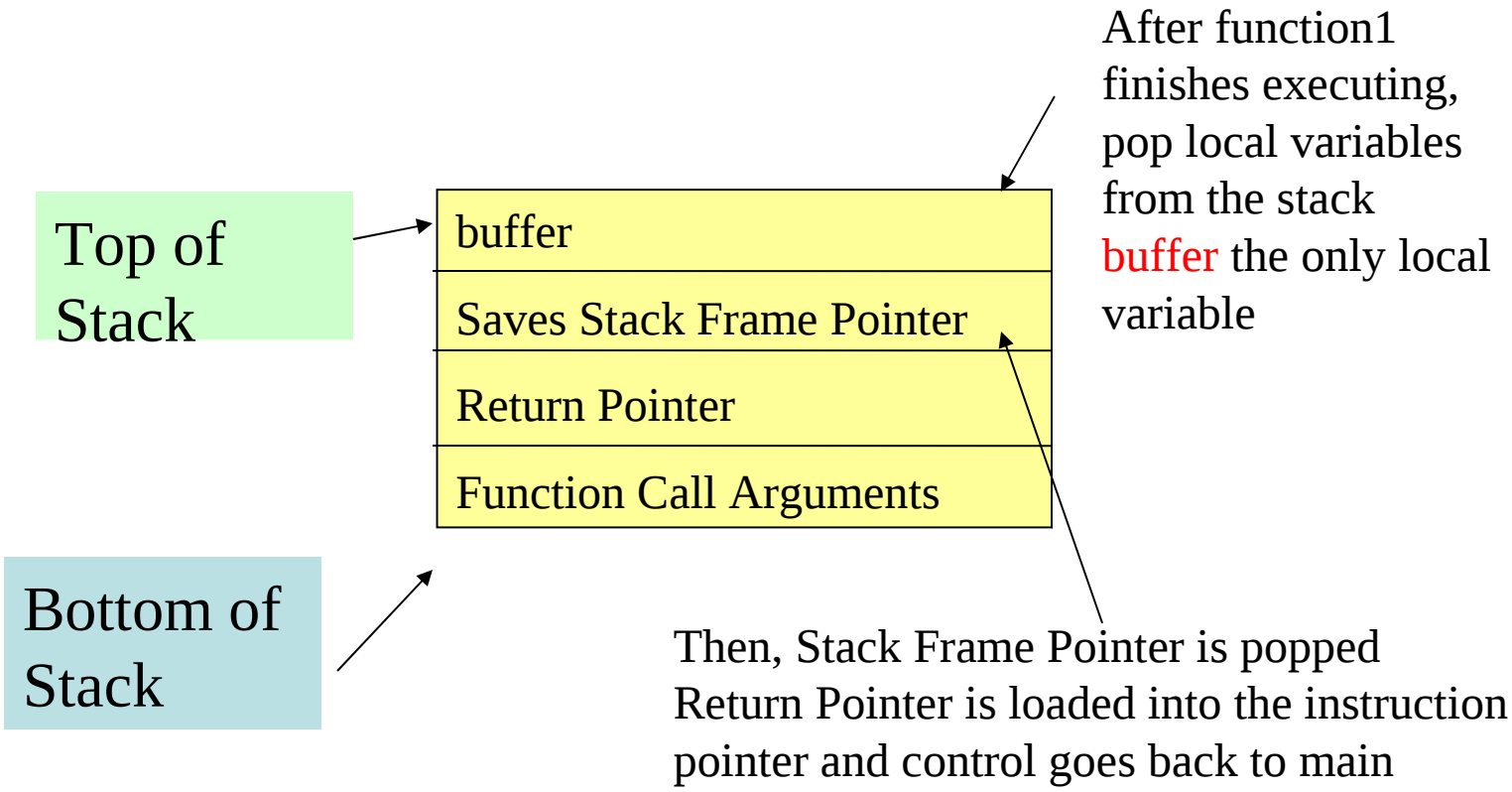
Buffer Overflow Attacks




```
void function1 (void)
{
    char buffer[20];
     printf ("Does nothing");
     return;
}

main ()
{
      function1();
}
```

Buffer Overflow Attacks



Buffer Overflow Attacks

- Question
  - Are contents of buffer zeroed out when it is popped off the stack?

Buffer Overflow Attacks

- No. Buffer remains unchanged
  - Instruction pointer just moves down the stack
  - This is key to how the buffer overflow exploits work
- How does a buffer overflow happen?
  - Works by stuffing too many items into finite array

- Example Buffer Overflow

```
void sample_function (char *string)
    {
       char buffer[16];
       strcpy (buffer, string);
       return;
    }
void main ()
    {
        char big_buffer[256];
        int i;

        for  (i=1; i < 255; i++)
             big_buffer[i] = 'A';

        sample_function (big_buffer);
    }
```

  - Strcpy doesn't check size of either source or destination string, stops only when it hits a NULL
  - Overflowed characters spill over into stack frame and return pointer , return pointer has a bunch of A's
  - When the return pointer is loaded and executed, it crashes

Example C Program

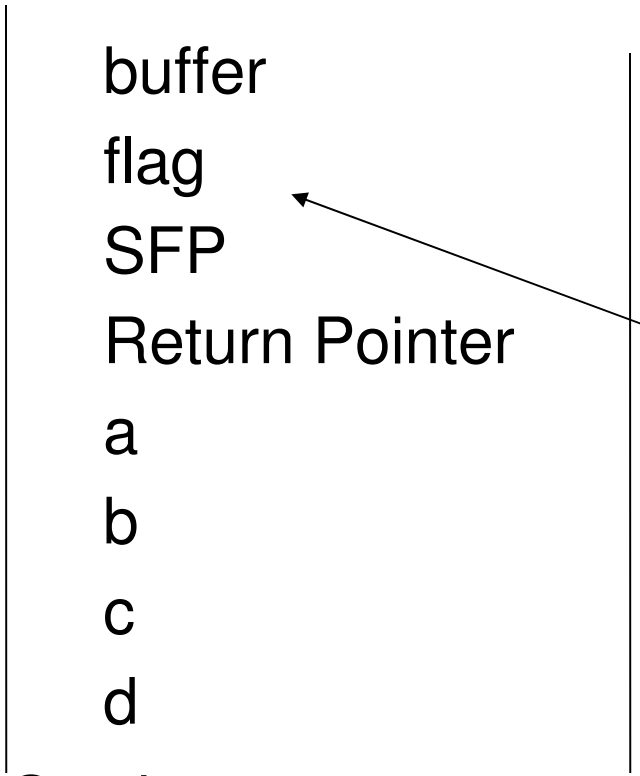
Example

```
void test_function (int a, int b, int c, int d)
{
    char flag;
    char buffer[16];
}
void main ()
{
    test_function (1,2,3,4);
}
```

Stack Memory

- Stack function arguments

Top of Stack



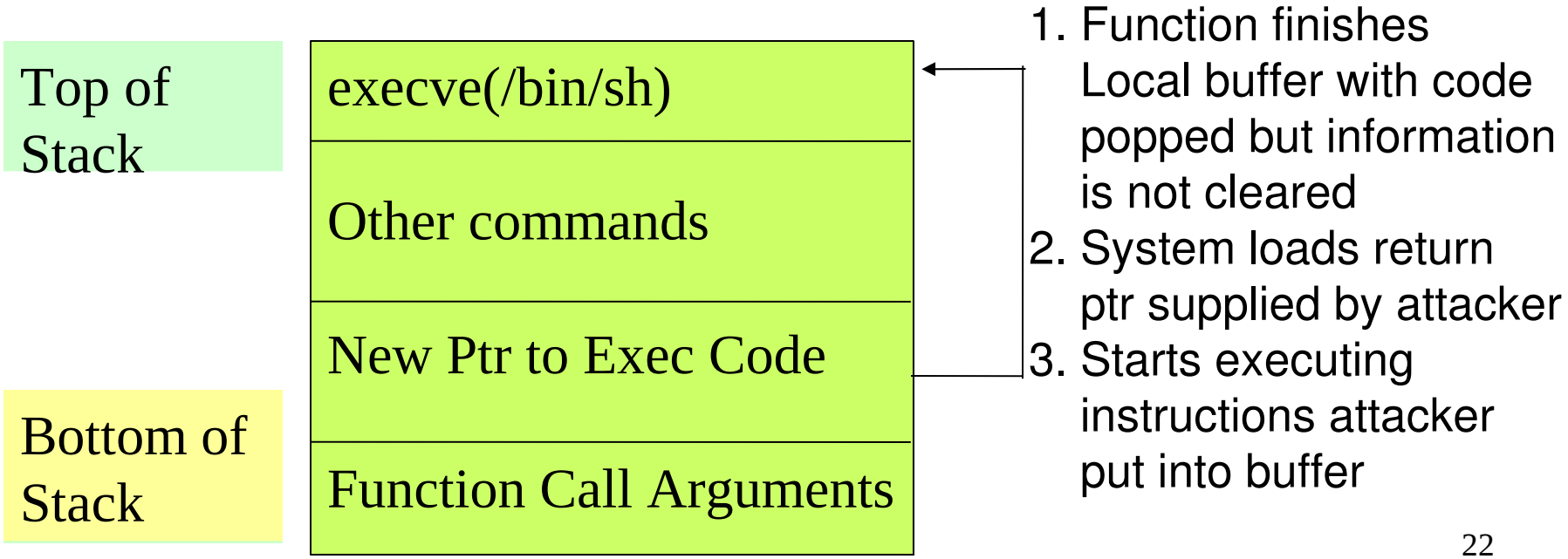
Bottom of Stack

Memory in stack used for functions, local vars

```
void test_function (int a, int b,
    int c, int d)
{
        char flag;
        char buffer[16];
}
void main ()
{
        test_function (1,2,3,4);
}
```

Buffer Overflow Attacks

- How does an attacker craft a buffer overflow?
  - Attacker wants it to do more than crash
  - He needs it to execute some useful instructions
  - Stuffs commands into buffer instead of A's



Buffer Overflow Attacks

- What gets executed?
  - Typically, some type of interactive shell
    - In Unix, /bin/sh
  - Attacker can run most other commands
    - In Windows, trigger a specific DLL

Buffer Overflow Attacks

- How do Buffer Overflows happen in code?
  - Programs using C library functions that allows user to enter data without checking for size of input
  - C language Routines
    - strcat, strcpy, gets, fgets, memcpy, plus many others
  - Programmer allowed to write code by language that does not check for input bounds
  - Sloppy habits allow this to cause problems

Buffer Overflow Attacks

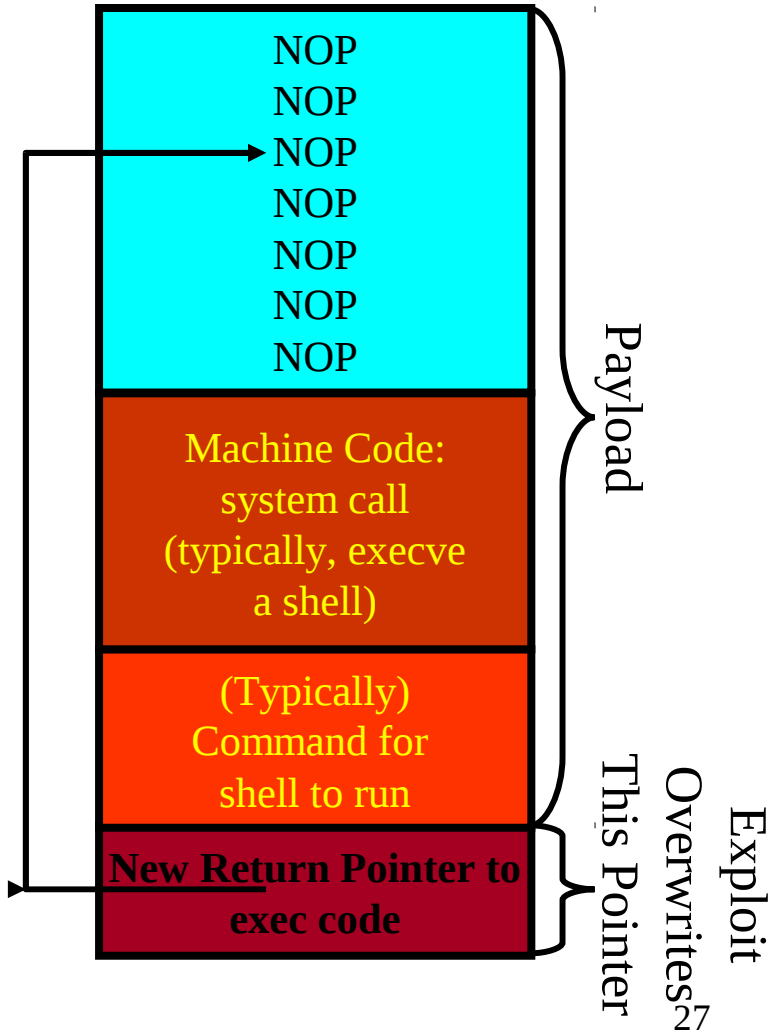
- How are Buffer Overflows Discovered?
  - Have source code, look for vulnerable functions
    - Examine input
    - Manually or use an automated tool
  - Don't have to have source code !!!!
    - Can use debugger to find evidence of these functions
    - Cram data into program looking for program to crash exhibiting a vulnerability
      - Known as "**fuzzing**" input
    - Want to cause an overrun but have your command end up in return pointer
    - Looking for ability to overrun return pointer

Buffer Overflow Attacks

- Crafting the Buffer Overrun
  1. Determine value to insert that points to beginning of executable code
  2. Difficult to find exact location of start of code, stacks are dynamic
  3. Could run program 100's or 1000's to guess exact place to jump back into stack
     But, can fudge value so it doesn't need to be exact
  4. Do this with NOPs - No-Operation instructions
     Place a lot of them before first executable instruction
     Called a NOP sled ... jump onto the sled

Buffer Overflow and Related Exploits

- Building blocks of many exploits include:
  - NOP Sled
  - Code to invoke some system call on the target machine
    - Such as the execve system call to run a program
    - Code for invoking a shell to run on the target – called "Shell code"
    - Instructions for shell to execute
  - New Return pointer, to trigger the whole package
    - Return pointer is set using some exploit, such as a buffer overflow, that overwrites a return pointer on the stack



Buffer Overflow Attacks

- Crafting the Buffer Overrun
  - Many NOP's are a signature for many IDS and IPS systems
  - **Better way to do this**
    - Some NOPS
    - But, other instructions too that “do nothing”
    - Examples
      - Add zero to register
      - Multiply register by 1
      - Jump to next instruction
    - Can sneak by IDS's this way

Buffer Overflow Attacks

- Crafting buffer overflows
  - Easier now than ever
  - Several sites with great GUI's for crafting exploits
    - Other attacks too, not just buffer overruns
  - Bugtrac Archives

    http://www.packetstormsecurity.com
  - Metasploit Project

    http://www.metasploit.com

Buffer Overflow Attacks

- What comes next – after smashing the stack?
  - Best possible result for attacker
    - He/She can run shell as root or administrator
    - If can't locate a program with super-user privileges and some vulnerability from across the network
    - Penetrate a system to gain regular user privilege across the network
      - Then, use a local buffer overflow to escalate their privilege to root ie. run a local setuid progam

Buffer Overflow Attacks

- What comes next – after smashing the stack?
  - Create a backdoor into the system
    1) Changes inetd.conf to spawn an interactive shell with root privileges on a specific port
    2) Use TFTP to copy netcat, a versatile tool that can then export a command shell to attacker's machine for input
  - Attacker will use back door to come back into the machine at their leisure
    - At that point they "own" the machine

Defenses for Buffer Overflows

- Two Categories of Defense
  1. System Administration
     - Management, configuration and security defenses
  2. Software Development
     - Careful programming practices
     - Automated tools help with discovery

Defenses for Buffer Overflows

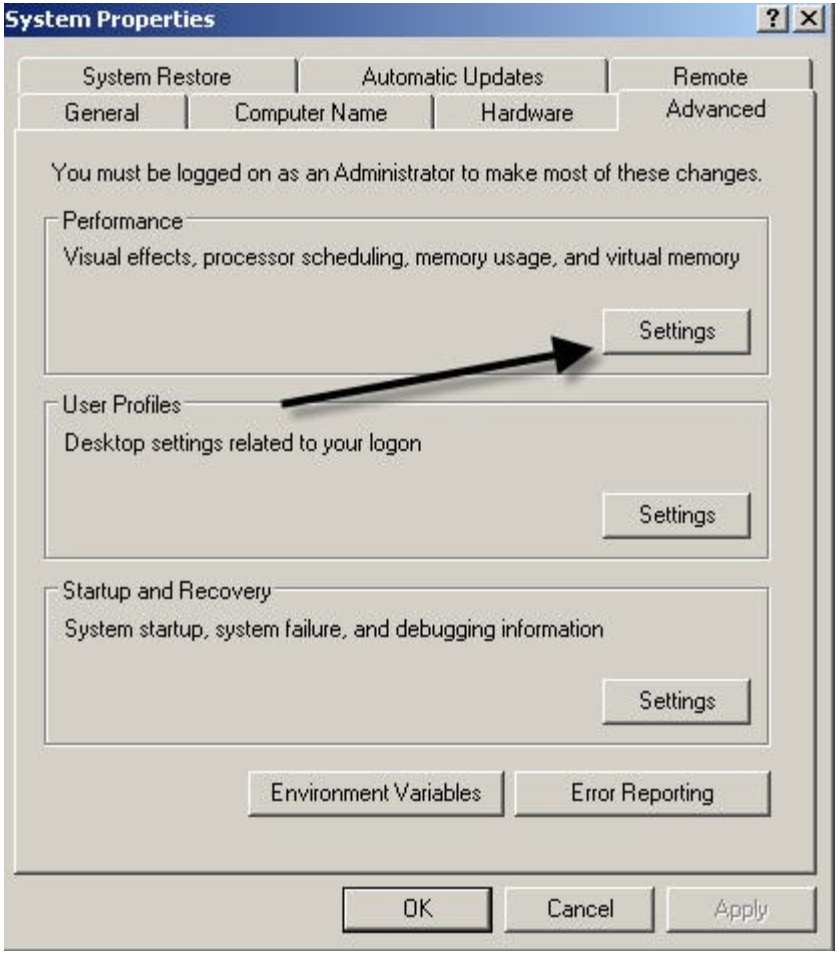
- System Administration
  - Keep systems patched
  - Stay on top of vulnerabilities
    - Bugtraq, US-Cert and
    - Application vendor lists
  - Harden your System
    - Configure systems with minimum number of unnecessary services and software
    - Includes filtering and controlling outgoing traffic

Buffer Overflow Protection

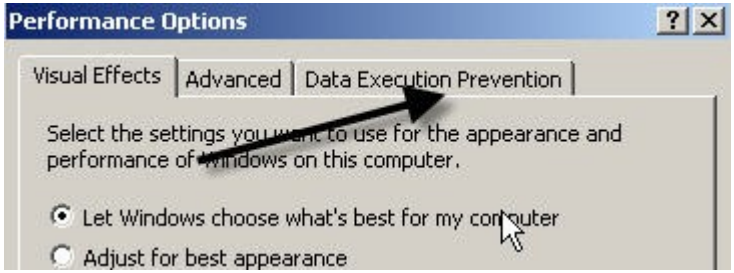
- **Windows Protection**
- **Data Execution Prevention (DEP)**
  - Most modern CPUs support an **NX (No eXecute)** bit to designate part of memory for containing only data
  - DEP will not allow code in memory area to be executed
  - This has been around since Windows XP SP 2
  - Windows Vista allows software developers to enable NX hardware protection specifically for application software that they develop

Windows XP Interface

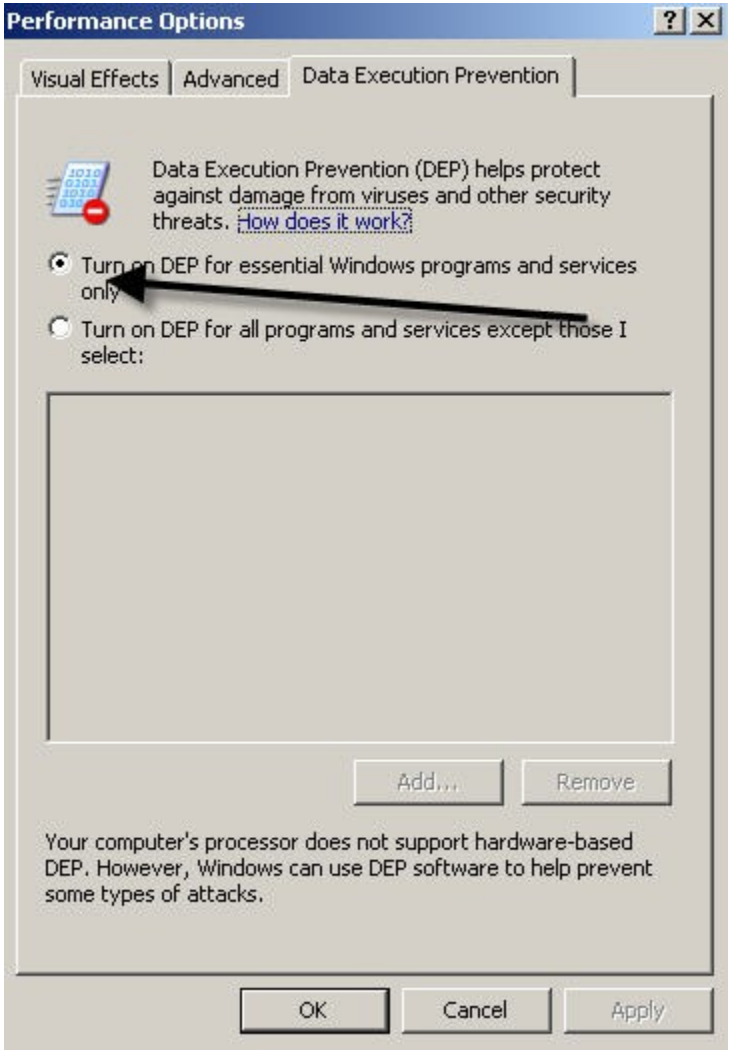
1.



2.



3.



All Newer versions of Windows have this capability
Windows 7 almost the same
Windows 10 enabled by default

DEP Actually Uses Nx

- Prevents the attacker from jumping to data
  - Uses the NX bit in modern CPUs
  - Modes of operation
    - OptIn – protects only apps compiled with /NXCOMPAT
    - Default mode on XP and Vista

    - OptOut – protects all apps unless they opt out
    - Default mode on Server 2003 and 2008

    Blog post that explains this feature pretty well

    http://blog.jalil.org/2008/01/27/dep-nxcompat-and-changes-in-net-framework-20-sp1/

Vista also has ASLR

- Address Space Layout Randomization
- Dramatically lowers exploit reliability
  - Relies on nothing being statically placed
- Several major components
  - Image Randomization
  - Heap Randomization
  - Stack Randomization

Address Space Layout Randomization (ASLR)

- Binaries opted-in to ASLR will be randomized
  - Configurable: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management\MoveImages
- Strategy 1: DLL randomization
  - Random offset from 0x78000000 up to 16M chosen ("Image Bias")
  - DLLs packed together near the top of memory
  - Known DLLs order also mixed up at boot time
- Strategy 2: EXE randomization
  - Random image base chosen within 16M of preferred image base
  - DLLs also use this strategy if "DLL Range" is used up

Address Space Layout Randomization (ASLR)

- Heap randomization strategy: Move the heap base
  - Address where heap begins is selected linearly with NtAllocateVirtualMemory()
  - Random offset up to 2M into selected region is used for real heap base
  - 64K alignment
- Stack randomization strategy: Selecting a random "hole" in the address space
  - Random 5-bit value chosen (X)
  - Address space searched X times for space to allocate the stack
- Stack base also randomized
  - Stack begins at random offset from selected base (up to half a page)

How well do you think this works?

Address Randomization Bypass

- Not a perfect solution ...
- Windows 7,

  http://www.darkreading.com/vulnerability_management/security/attacks/showArticle.jhtml?articleID=224200392

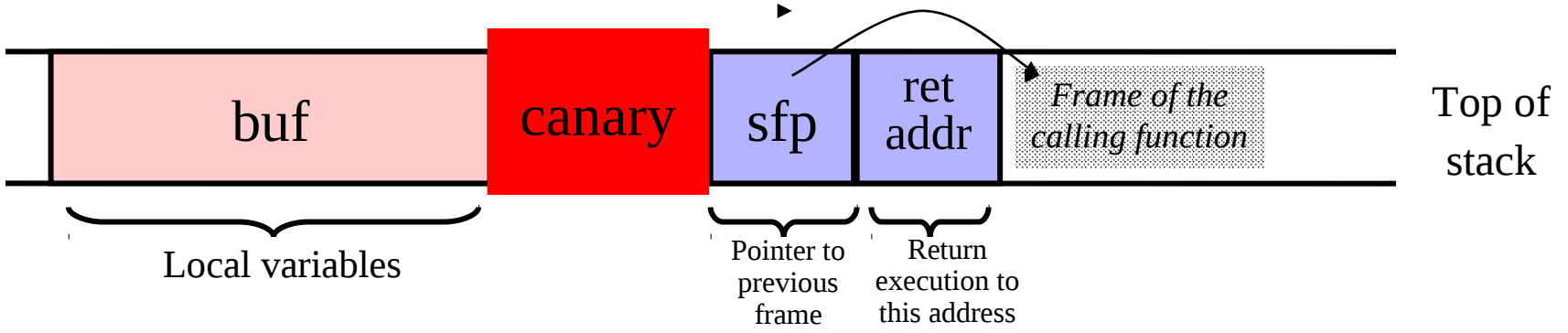
  – Pwn2own contest, CanSecWest in Vancouver

  – Winner used a two-part exploit: First he located a specific .dll file, with a zero day exploit, to evade ASLR, and then used that information to trigger an exploit that disabled DEP

  – He used a heap overflow attack to get the address of the .dll file

Run-Time Checking: StackGuard

- Embed “canaries” in stack frames and verify their integrity prior to function return
  - Any overflow of local variables will damage the canary



- Choose random canary string on program start
  - Attacker can’t guess what the value of canary will be
- Terminator canary: “\0”, newline, linefeed, EOF
  - String functions like strcpy won’t copy beyond “\0

Nice overview and historical perspective

http://courses.cs.washington.edu/courses/cse504/10sp/Slides/lecture3.pdf

StackGuard Implementation

- **StackGuard** requires code recompilation
  Checking canary integrity prior to every function return causes a performance penalty
  For example, 8% for Apache Web server
- **PointGuard** also places canaries next to function pointers and setjmp buffers
  –Worse performance penalty
- StackGuard can be defeated!
  –Phrack article by Bulba and Kil3r
  http://www.phrack.com/issues.html?issue=56&id=5#article

GCC Modification for Stack Protection

- In 2005, GCC implemented Canary style compiled code protection
  - It is a GCC (Gnu Compiler Collection) extension for protecting applications from stack-smashing attacks
  - Protection is realized by buffer overflow detection and the variable reordering feature to avoid the corruption of pointers
- Basic idea of buffer overflow detection comes from StackGuard system

GCC Modification for Stack Protection

What does it do? The novel features are:

(1) the reordering of local variables to place buffers after pointers to avoid the corruption of pointers that could be used to further corrupt arbitrary memory locations,

(2) Copying of pointers in function arguments to an area preceding local variable buffers to prevent the corruption of pointers that could be used to further corrupt arbitrary memory locations, and

(3) Omission of instrumentation code from some functions to decrease performance overhead

Code Defenses for Buffer Overflows

- Software
  - Automated codechecking tools
  - Search for known problems
    - ITS4 – It's the Software Stupid
      - www.cigital.com/its4/
    - RATS – Rough Auditing Tool for Security
      - www.securesw.com/rats/
    - Flawfinder
      - www.dwheeler.com/flawfinder

Code Defenses for Buffer Overflows

- Software
  - Learn to write more secure programs
    - Take our Secure Code Class CSCD437
    - Writing Secure Code – book
      - Howard and Leblanc

        http://www.amazon.com/Writing-Secure-Second-Michael-Howard/dp/0735617228
    - Linux and Unix
      - White paper on developing secure code
      - www.dwheeler.com/secure-programs

Defenses for Buffer Overflows

- Java Code Issues
  - Widely assumed that Java is immune to Buffer Overflow problems
    - To a large extent its true
    - Java has a type-safe memory model
  - However, Java has been subject to buffer overflow attacks. **How?**

Defenses for Buffer Overflows

- Java Code Issues
  - Many Java based services are written in C or C++
  - Integrating Java with support libraries written in C and C++ is done routinely
  - Java also supports loading of DLL's and code libraries
  - Exported functions can be used directly in Java
    - Opens up possibilities buffer overflows may be exploited in support libraries

Buffer Overflow Attacks

- Summary
  - Buffer overflows are an outgrowth of poor memory models compounded by programmer carelessness
  - For each safeguard – StackGuard, non-executable stack, hackers have workarounds
    - One downside of the safeguards
      - People have a false sense of security!!

Buffer Overflow Attacks

- Summary
  - Lots more buffer overflows in other types of content
    - Databases – SQL strings
    - Data files – MP3 files – Header of file for a string
    - HTTP Cookies – Apache HTTPD overflow
    - Environment Variables - $HOME, TERM others
    - Many others ...

More References

- Shellcode References

  http://www.vividmachines.com/shellcode/shellcode.html

  http://www.securiteam.com/securityreviews/5OP0B006UQ.html

  http://www.infosecwriters.com/hhworld/shellcode.txt

  http://shellcode.org/

- Metasploit Framework

  3-part Article

  http://www.securityfocus.com/infocus/1789

  http://www.securityfocus.com/infocus/1790

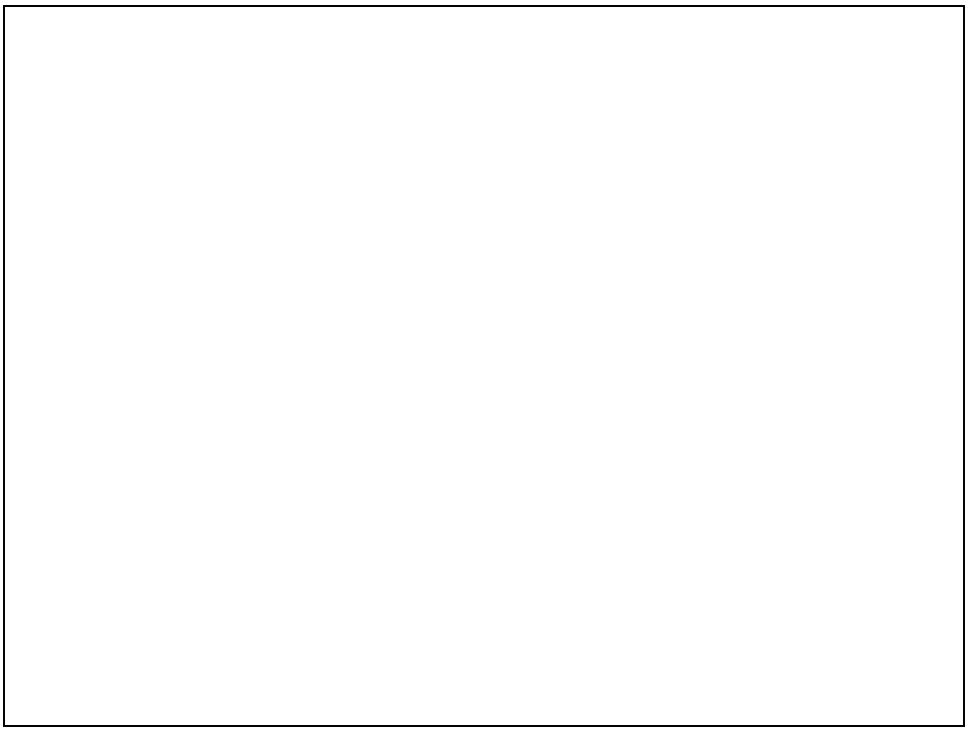
  http://www.securityfocus.com/infocus/1800

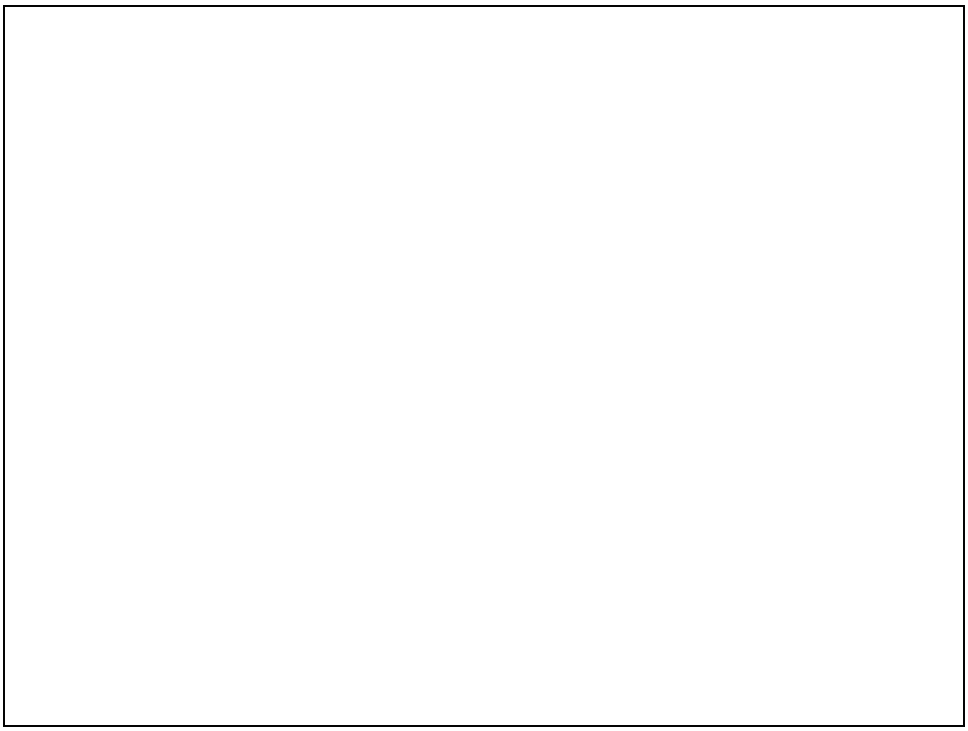
  http://www.syngress.com/book_catalog/327_SSPC/sample.pdf

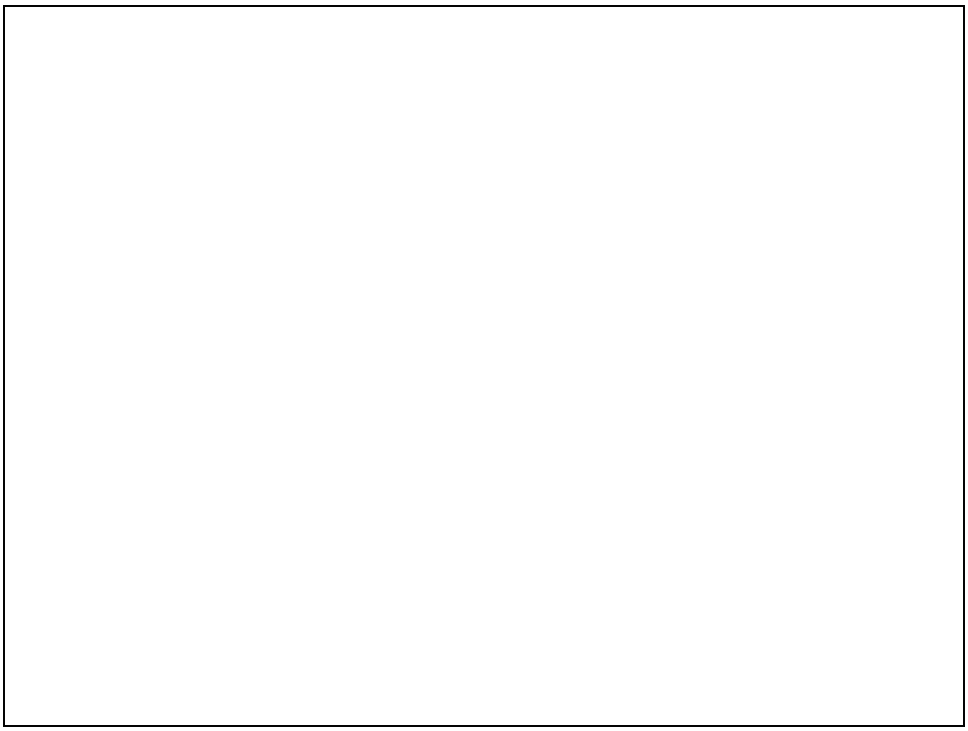
The End

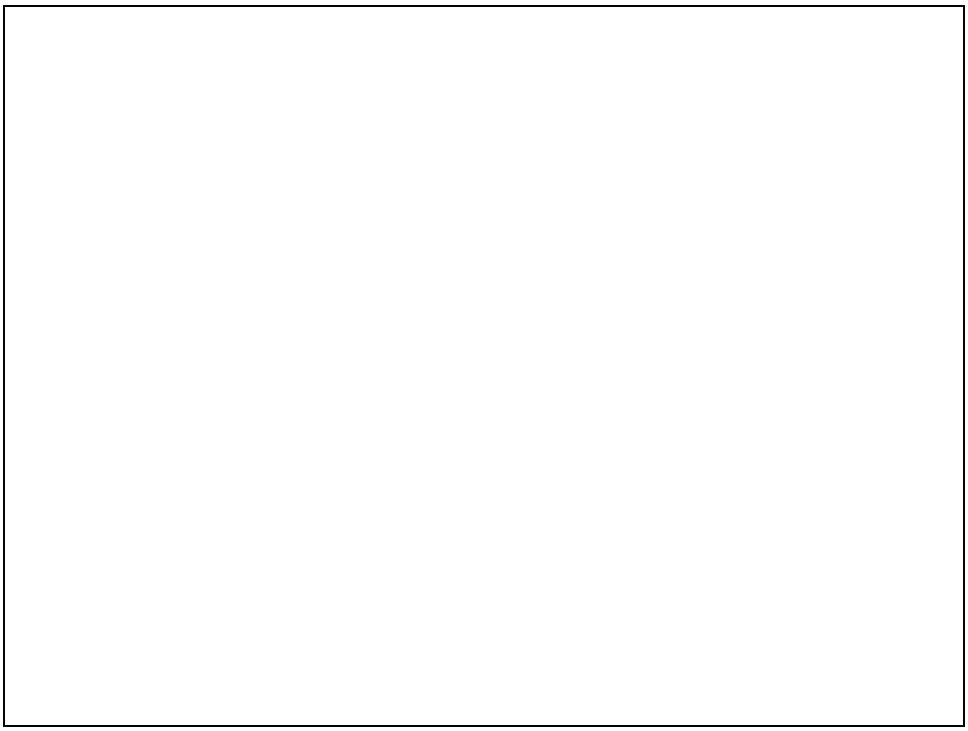
Learn about buffer overflows

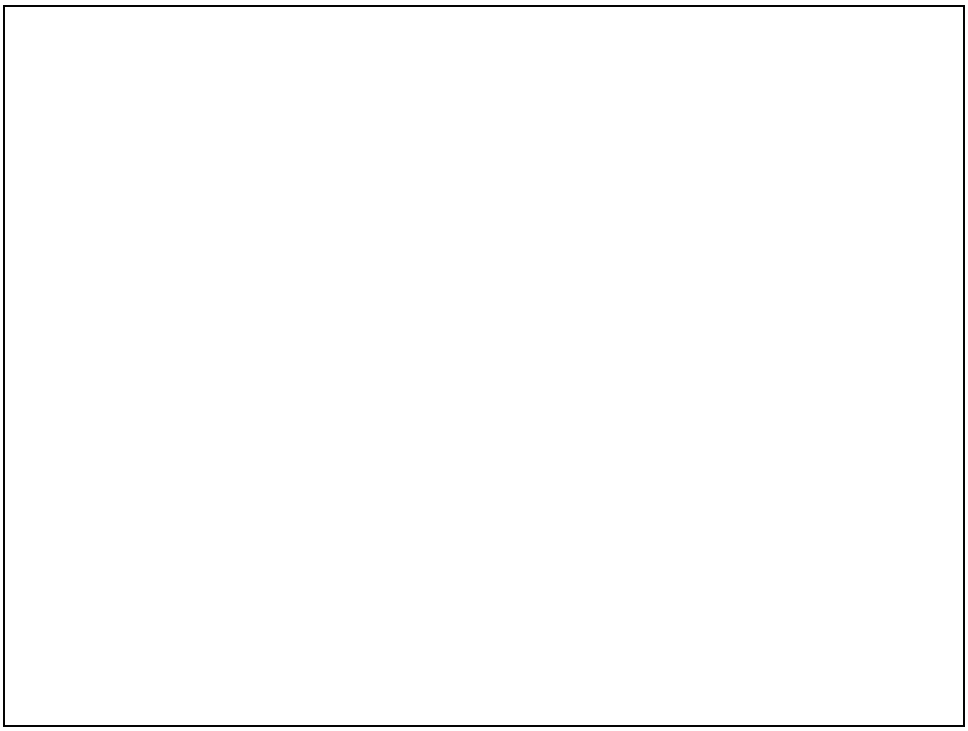
Don't use them for evil !!!

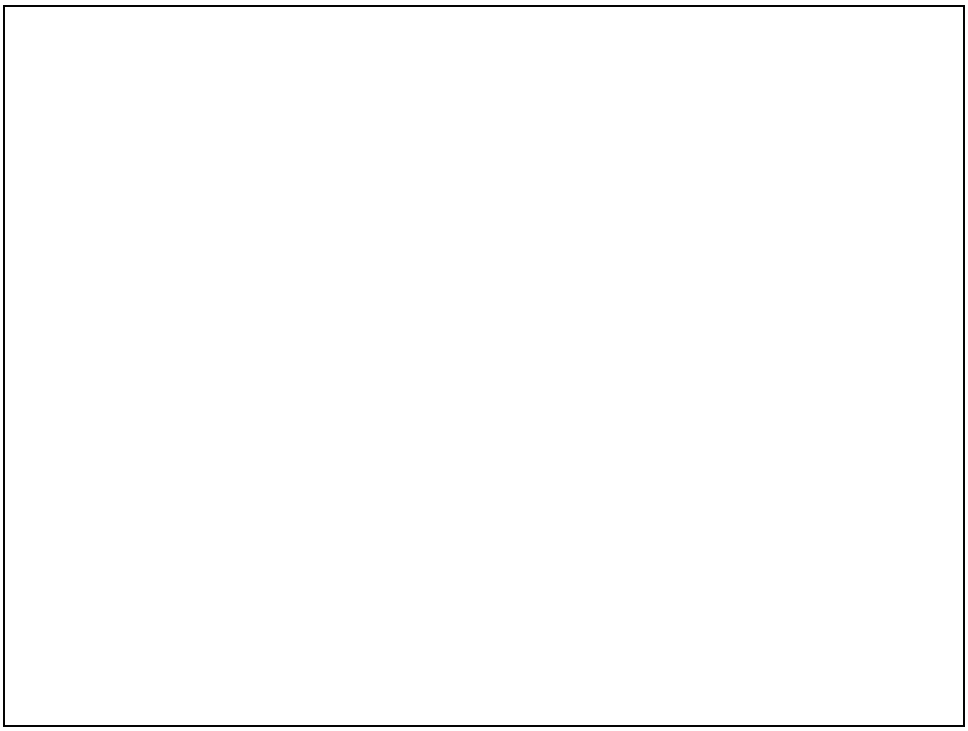


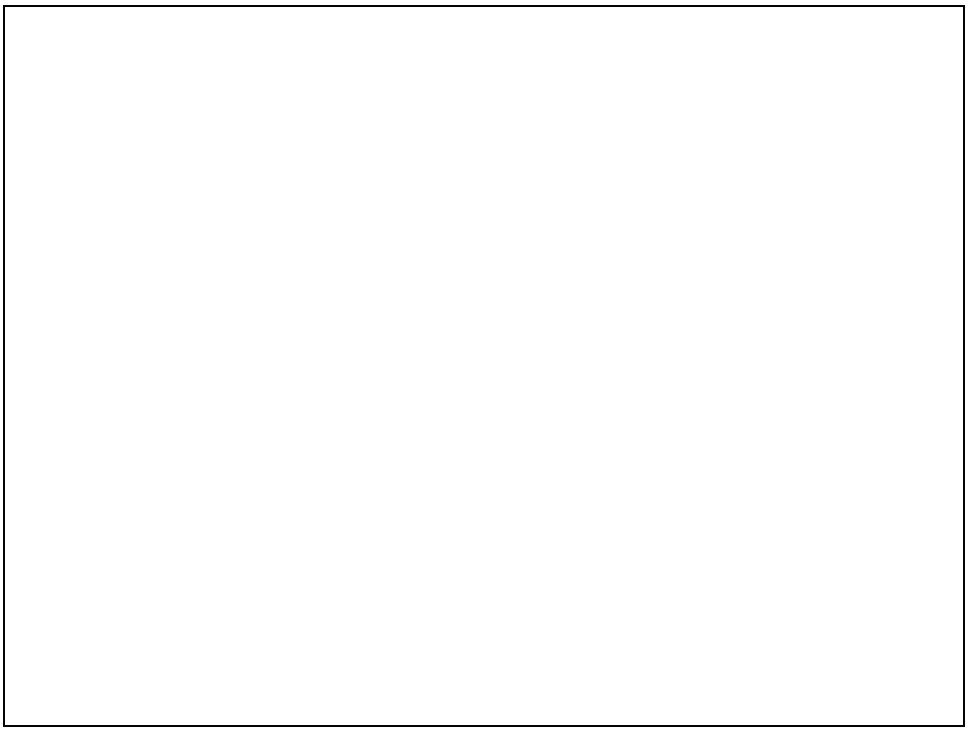


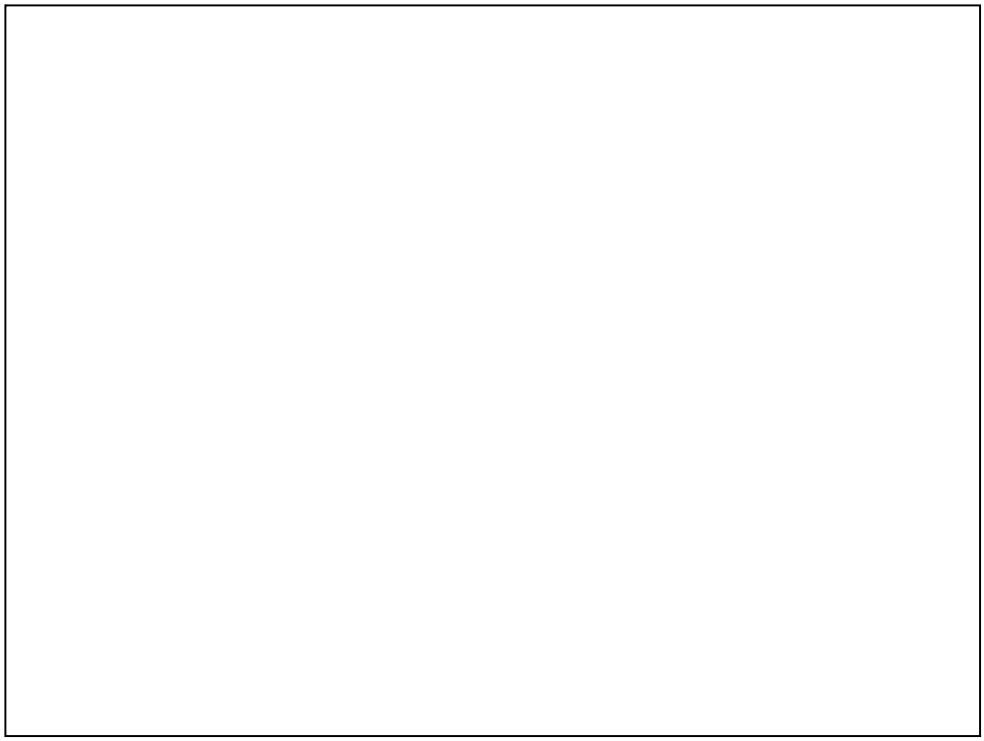


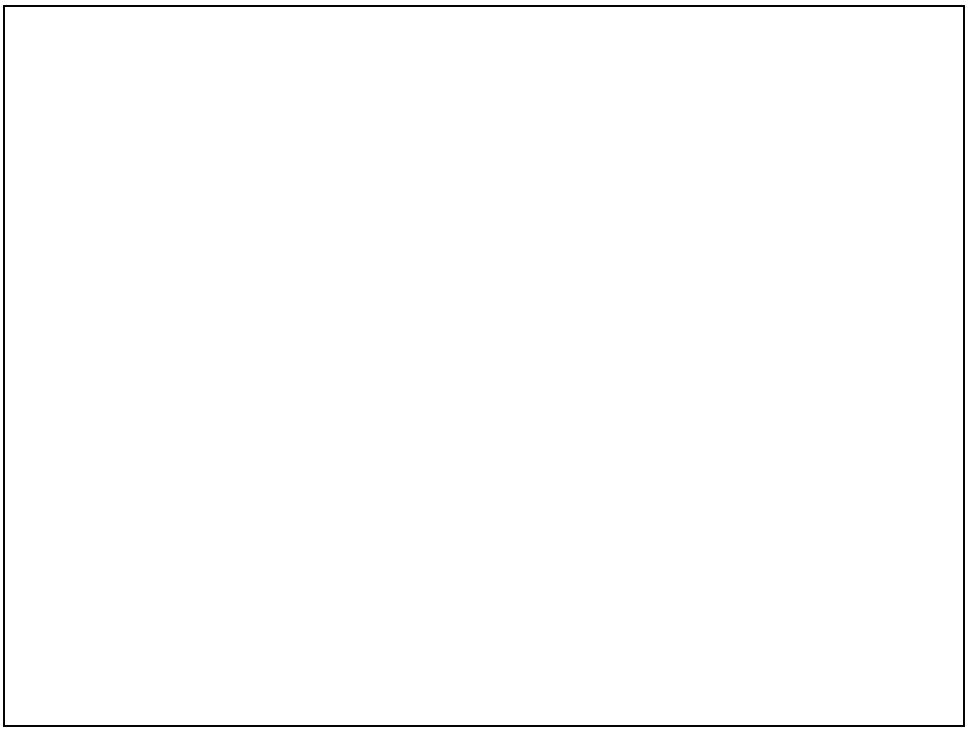


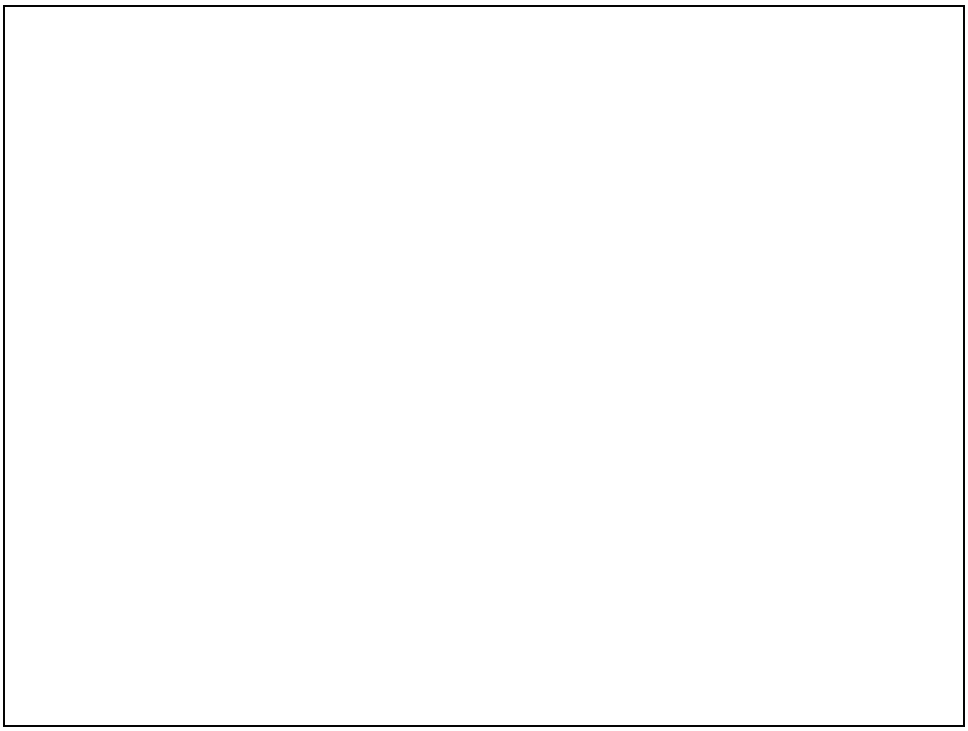


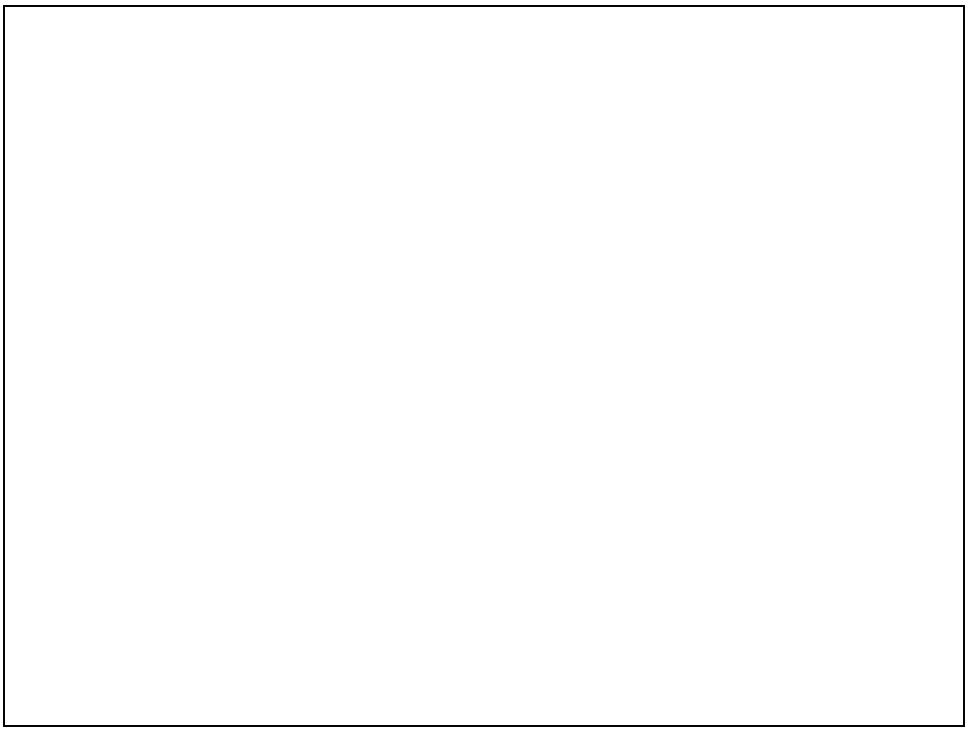


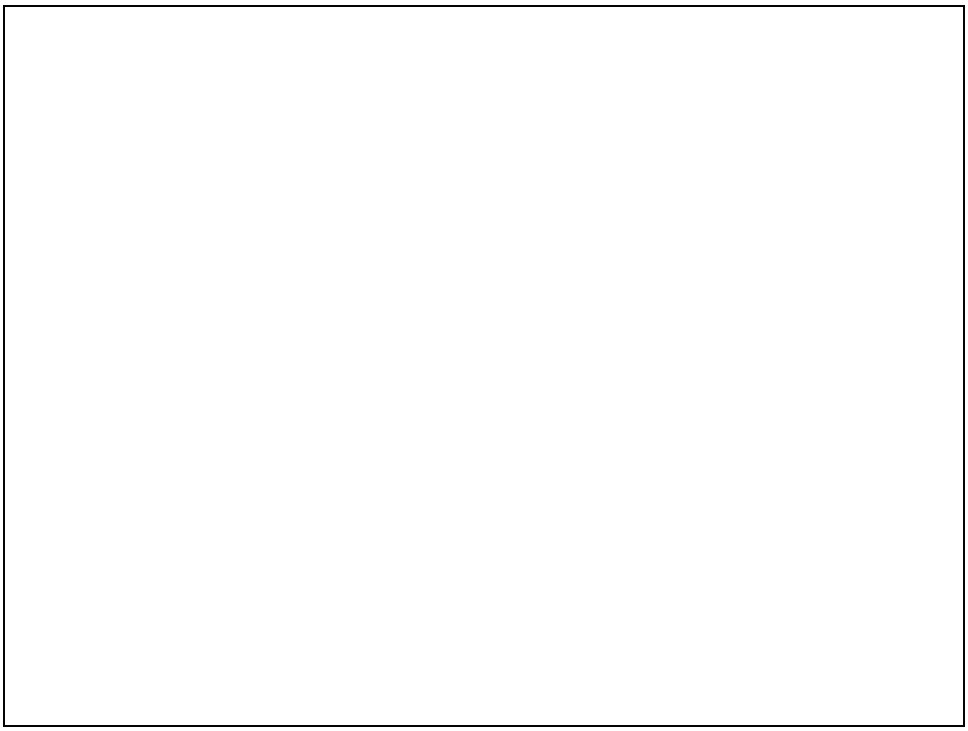








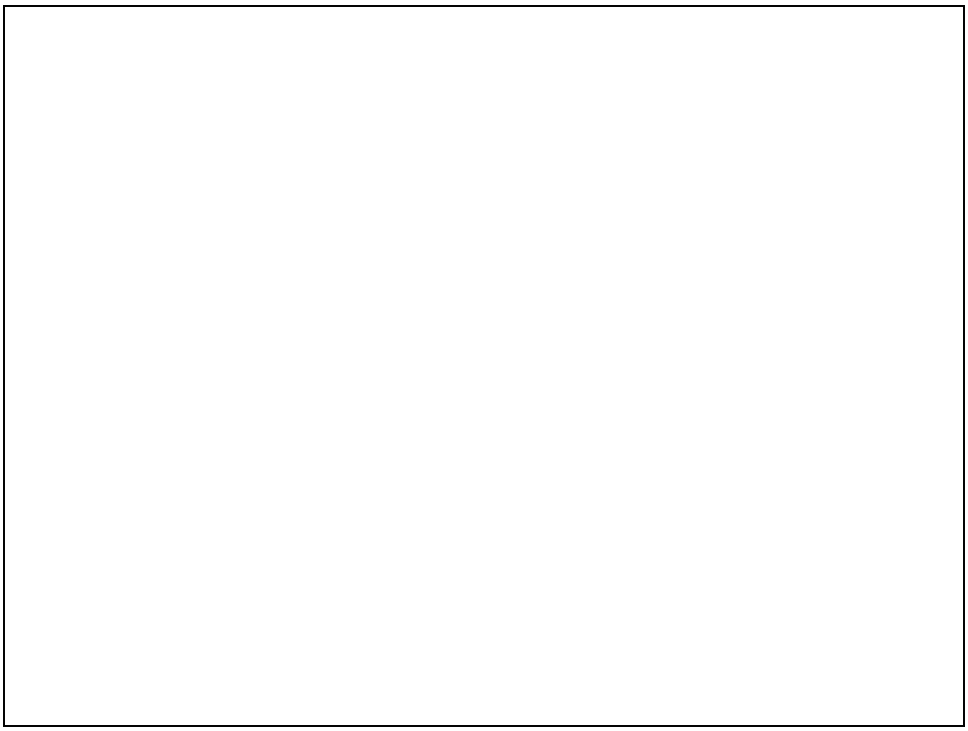


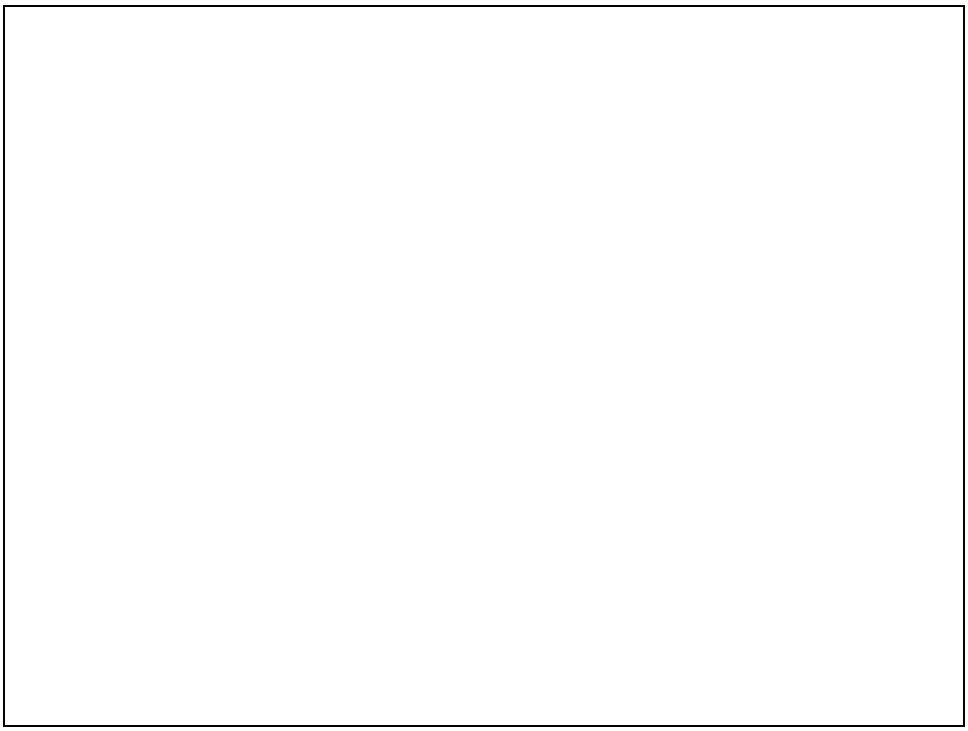


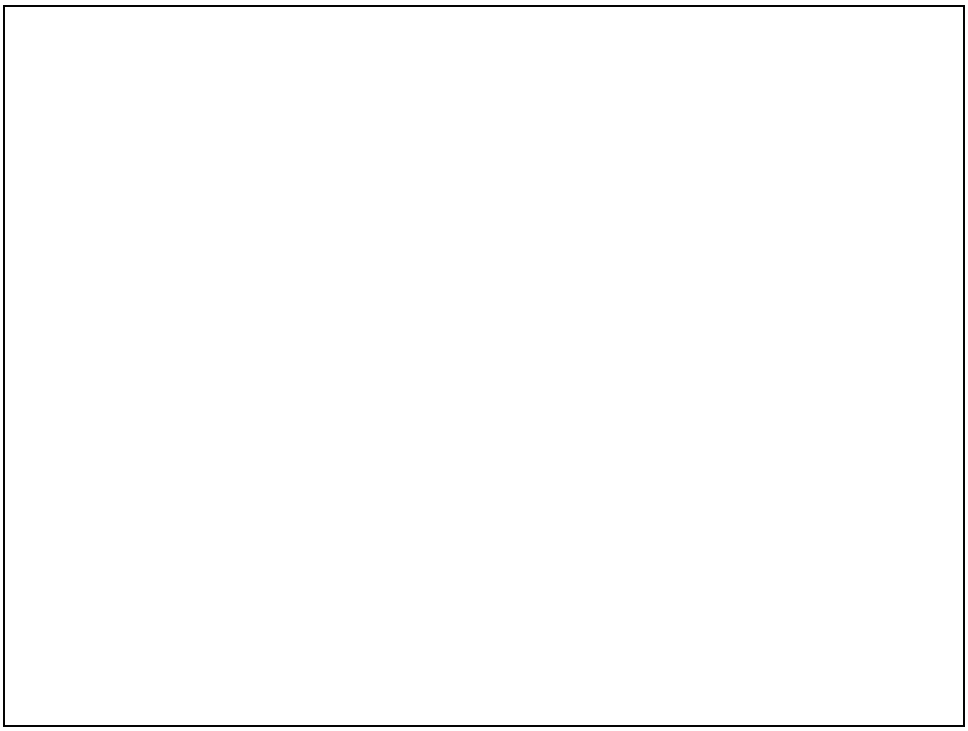
What Do you Need to Know
To Write One ?

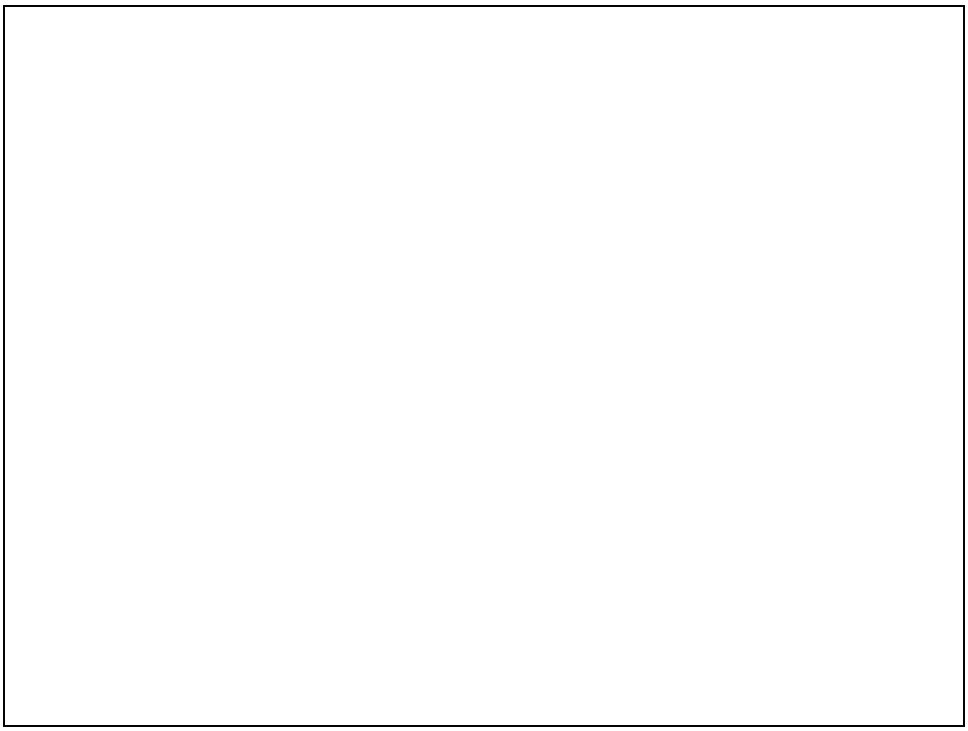
- Understand C functions and the stack
- Some familiarity with machine code
- Know how system calls are made
  - The exec() system call in particular
- Attacker needs to know which CPU and OS are running on the target machine:
  - Our examples are for x86 running Linux

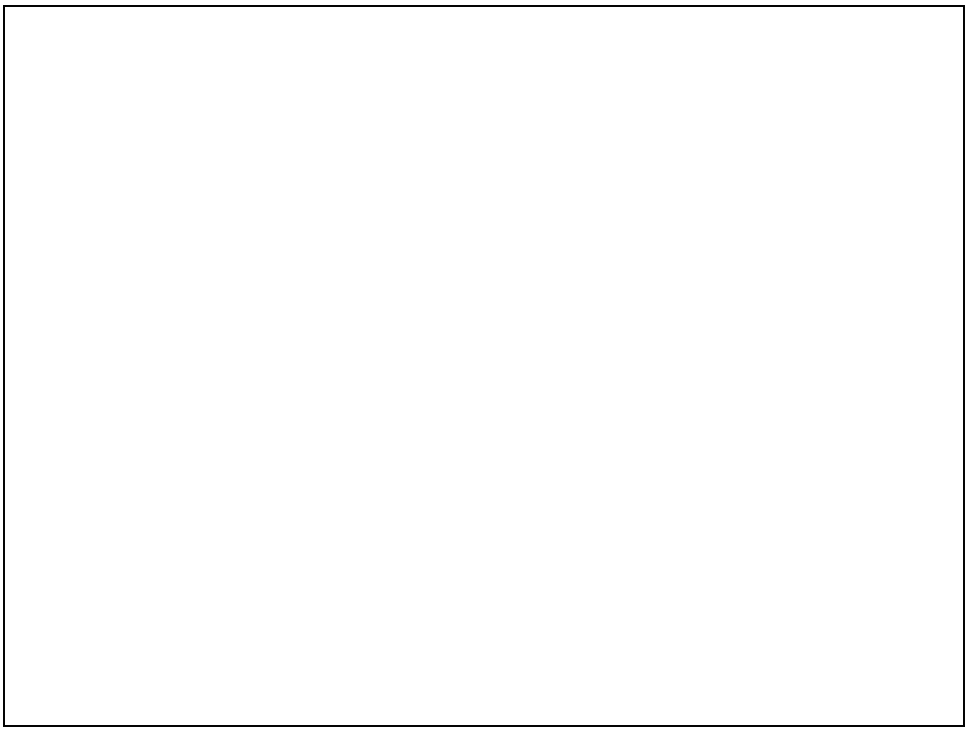
- Details vary slightly between CPUs and OSs:
  - Little endian vs. big endian (x86 vs. Motorola)
  - Stack Frame structure (Unix vs. Windows)
  - Stack growth direction

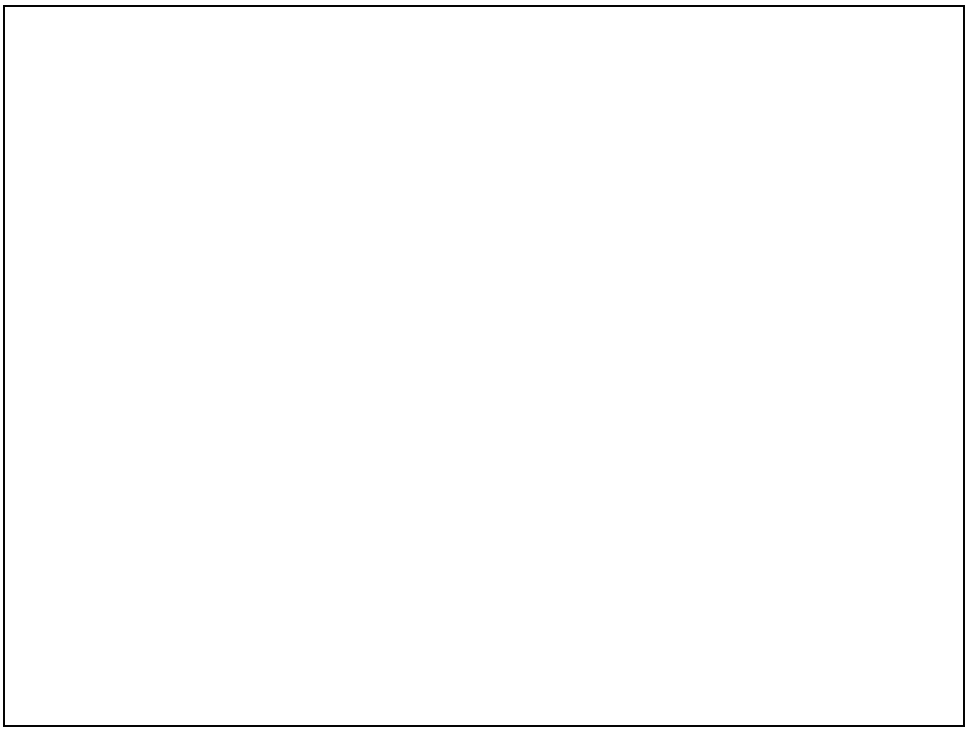


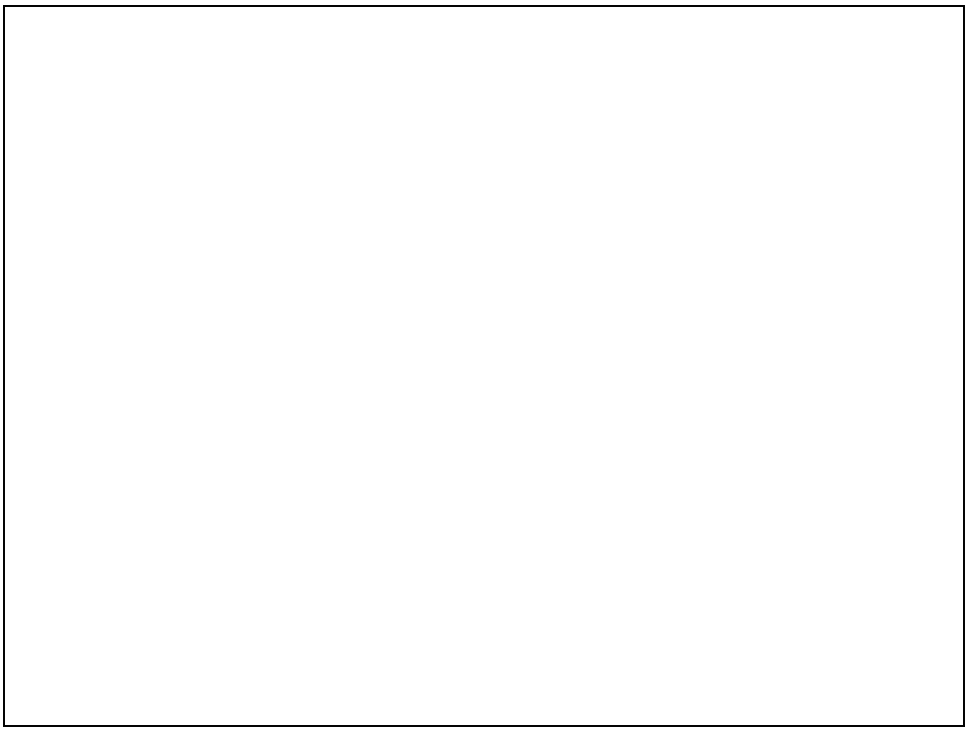


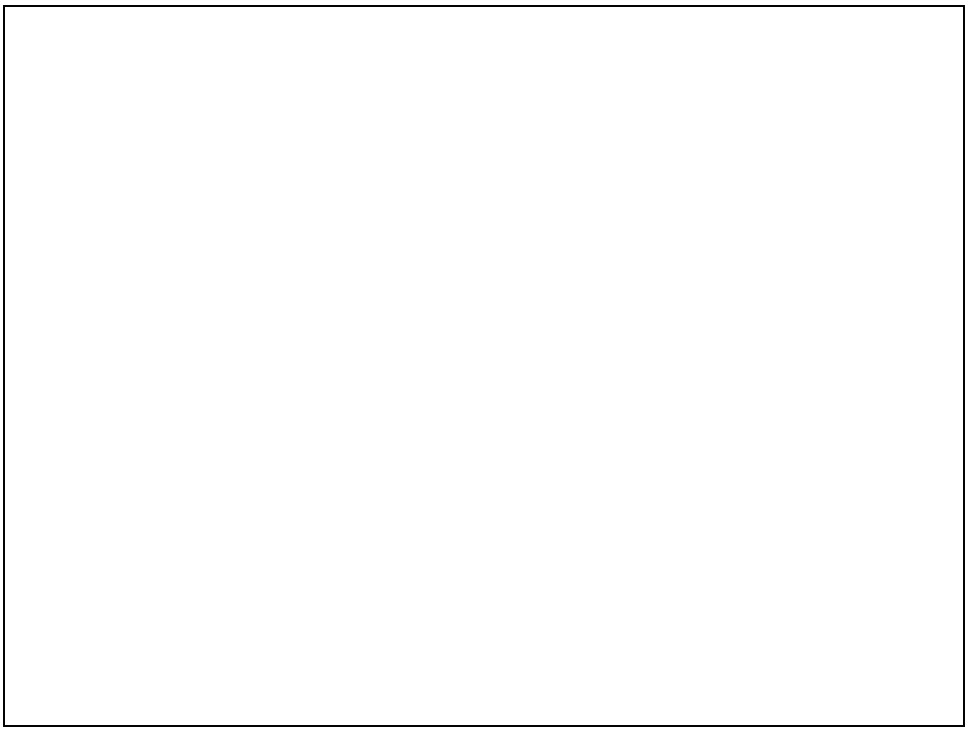


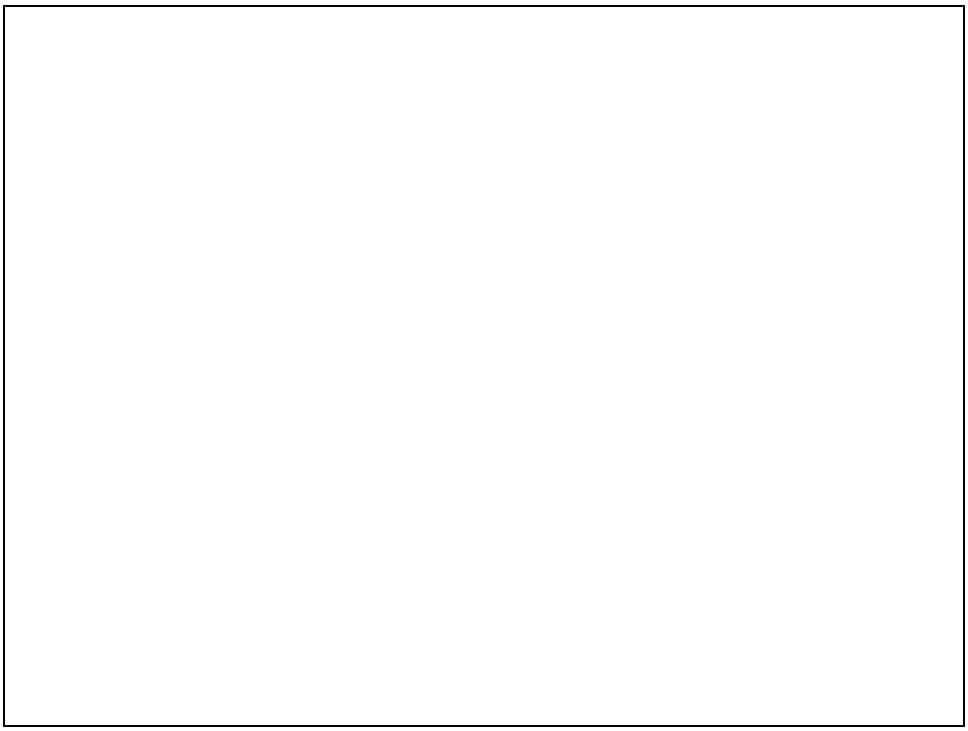


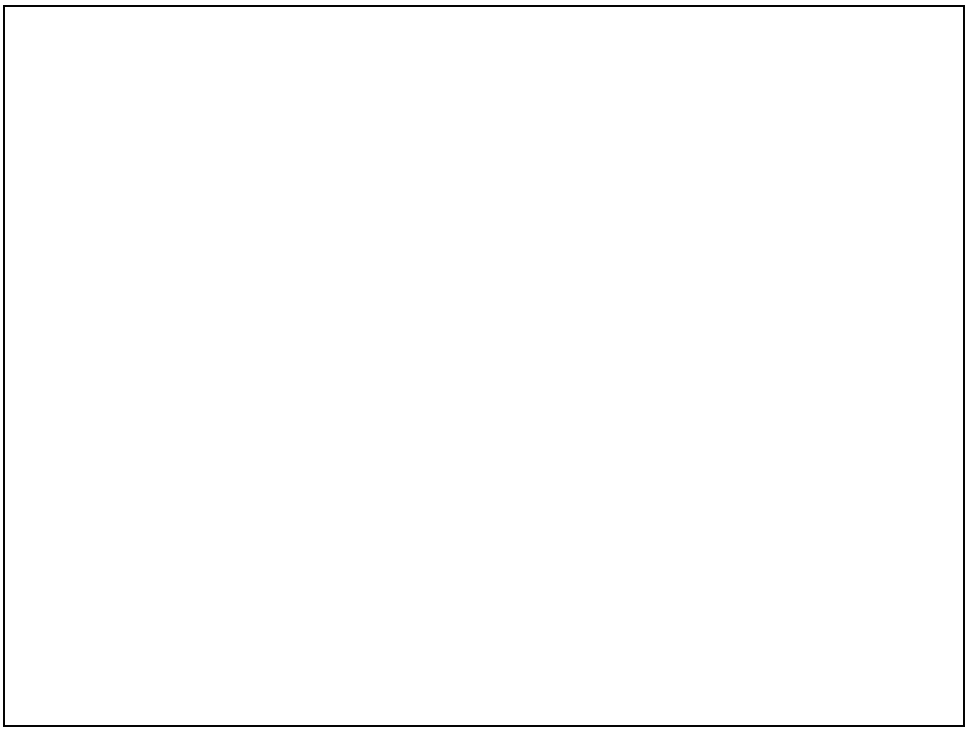


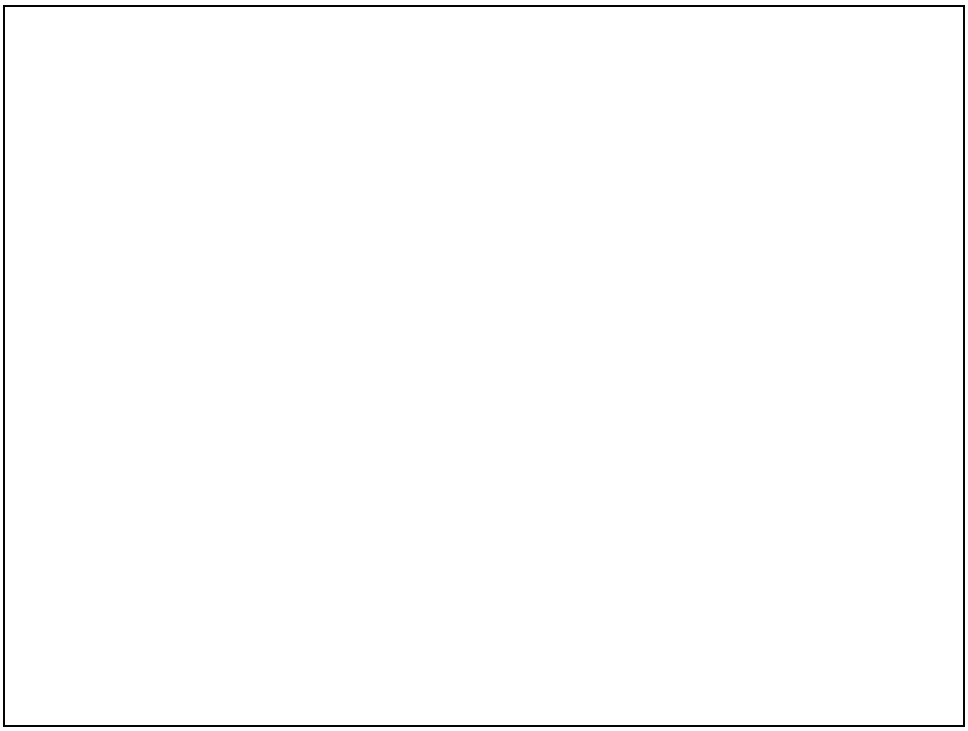


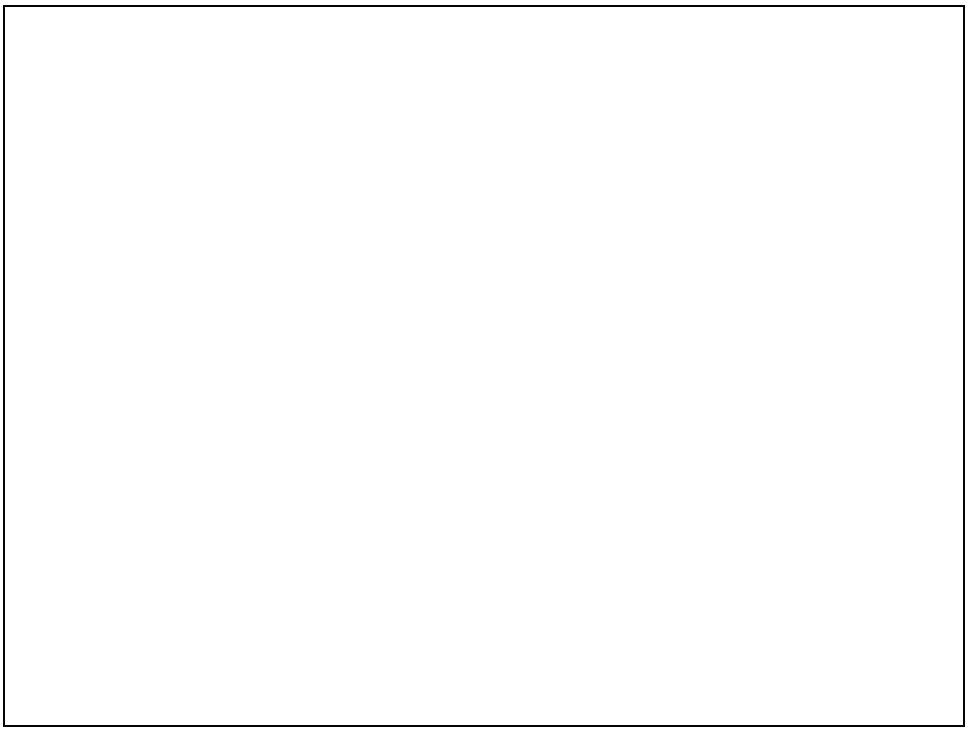


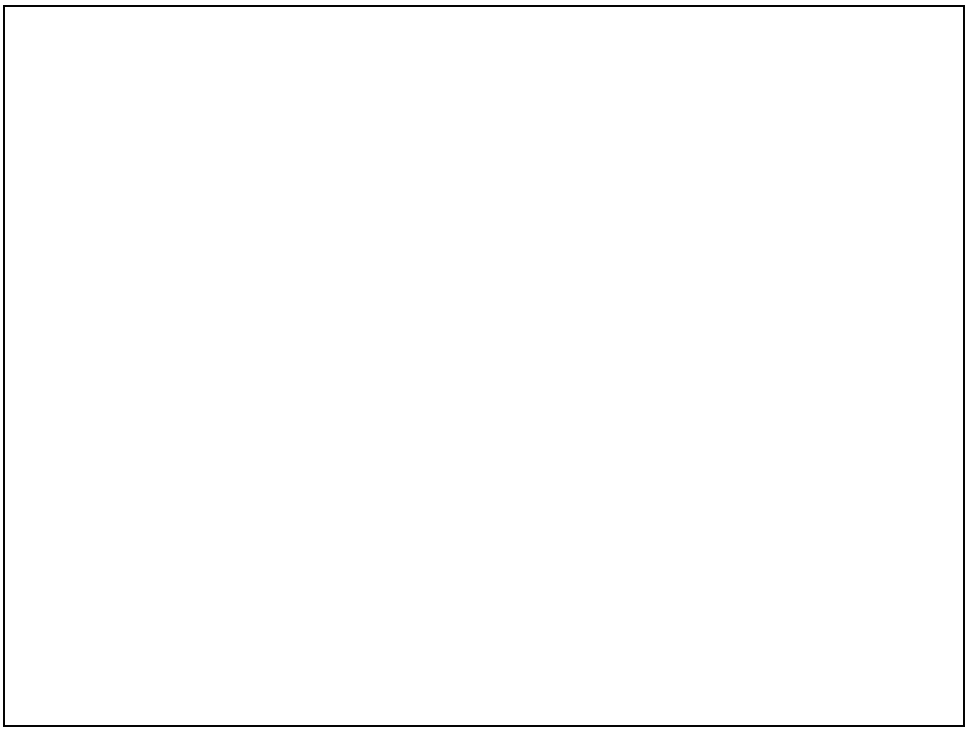


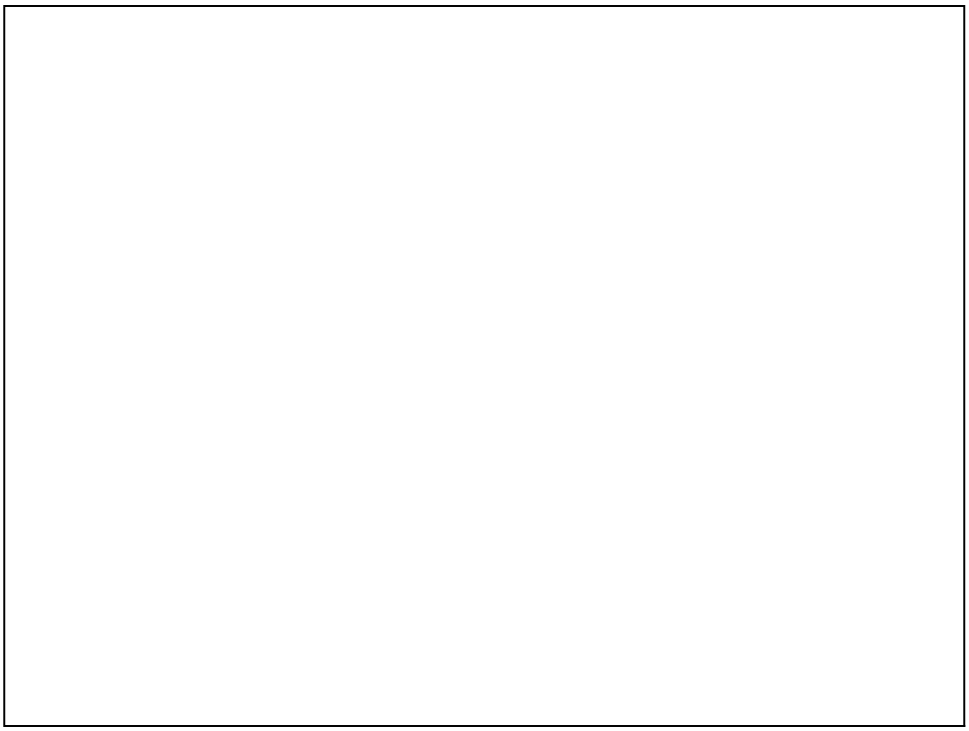












So, the fundamental building block of many exploits, including stack-based, heap-based, and BSS-based buffer oveflows, as well as many format string attacks and other exploits includes the following elements:

- The NOP Sled, which is used to help improve the odds that the return pointer will hit valid code.
- Code to invoke some system call on the target machine. This code must be written in machine language for a given processor type (x86, SPARC, etc.) and tailored for a given Operating System type (Windows, Linux, Solaris, etc.) Typically, some system call associated with executing a program (such as the Linux execve system call) will be activated.
- (Typically) Code for invoking a shell to run on the target. Attackers usually invoke a shell (such as the Unix/Linux /bin/sh or Windows cmd.exe) on the target. Shells are nice, because attackers can feed them commands to execute.
- (Typically) Instructions for that shell to execute. This is the command the attacker wants to run on the victim. It could involve installing a backdoor, or attaching a shell to an active TCP connection, or a variety of other items.
- A return pointer, to trigger the whole package. This pointer aims execution flow back into the memory location to get the exploit to run. This return pointer is set using some exploit, such as a buffer overflow that overwrites a return pointer on the stack or a format string exploit that lets the attacker change values on the stack.

The NOP sled, Machine code, and command are collectively called the “Payload”. Code that overwrites the return pointer is called the “Exploit”. Sometimes, people refer to the Payload and Exploit together as simply “the Exploit”.