Defending Computer Networks
Lecture 2: Vulnerabilities

Stuart Staniford

Adjunct Professor of Computer Science

Logistics

- Still space in class
  - 73 out of 75 taken on Tuesday
  - Will be drop-outs
  - Restriction to CS M.Eng will be lifted shortly
  - Keep attending
- HW1 expected early next week
  - Piazza also early next week
- Website up:
  - http://www.cs.cornell.edu/courses/CS5434/2015fa/

Main Goals for Today

- Understand system() function vulnerabilities
- CVE classification of vulnerabilities
- Outline understanding of buffer overflow vulnerabilities

Junaid Hussain, British Hacker For ISIS, Killed In US Drone Strike In Syria: Sources

A British hacker believed to be a top cyber expert for the Islamic State group has been killed in a U.S. drone strike, sources reportedly said Wednesday. Junaid Hussain, a British citizen from Birmingham, reportedly traveled to Syria in 2013.

The 21-year-old was reportedly killed in the drone strike, which likely involved the U.S. Defense Department, a U.S. source told Reuters. The strike was conducted Tuesday near the Syrian city of Raqqa, a CSO Online report said. However, U.S. authorities have yet to officially announce Hussain's death.

U.S. and European government sources told Reuters that Hussain was believed to be the leader of the CyberCaliphate, a hacking group, which in January attacked a Pentagon Twitter account. The sources could not confirm if Hussain was directly involved in the hack. The hackers took over Twitter and YouTube accounts belonging to U.S. Central Command, or CentCom, often used to provide updates about airstrikes against ISIS.

http://www.ibtimes.com/junaid-hussain-british-hacker-isis-killed-us-drone-strike-syria-sources-2070678

System() Function Vulnerabilities

- Very basic class of C/Unix vulnerability
- “man 3 system”
- Been known for decades
- Still occurs, however.
- Other languages:
  - C++: std::system()
  - Python: os.system()
  - Perl: system LIST
- Let’s work through an example

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <assert.h>
#include <strings.h>
#include <unistd.h>
#include <stdlib.h>

// This code is a very short hack to illustrate server vulnerabilities!!
// Do not write production code like this!!!
int                sockFd;
int                connFd;
unsigned short     port          = 3333;
struct sockaddr_in  serverAddress;
struct sockaddr_in  clientAddress;
```

```
void setupSocket(void)
{
  unsigned clientLen;
  assert( (sockFd = socket(AF_INET, SOCK_STREAM, 0)) >= 0);
  bzero(&serverAddress, sizeof(struct sockaddr_in));
  serverAddress.sin_family      = AF_INET;
  serverAddress.sin_addr.s_addr = INADDR_ANY;
  serverAddress.sin_port        = htons(port);
  assert(bind(sockFd, (struct sockaddr *) &serverAddress, sizeof(struct sockaddr_in)) >= 0);
  assert(listen(sockFd, 5)>=0);
  clientLen = sizeof(struct sockaddr_in);
  assert( (connFd = accept(sockFd, (struct sockaddr *)&clientAddress, &clientLen)) > 1);
}
```

```
int getLineFromSocket(char* buffer, int len)
{
  int n;
  assert(write(connFd, "Type Symbol>", 12) >= 0);
  n = read(connFd, buffer, len);
  buffer[n-2] = '\0';
  return n;
}
```

```
void extractCountFromFile(char* fileName, char* answer)
{
  char buf[256];
  char* start;

  FILE* file = fopen(fileName, "r");
  assert(file);
  fgets(buf, 256, file);
  for(start = buf; *start; start++)
  {
    if(*start == ' ' || *start == '\t')
      continue;
    else
      break;
  }
  if(*start)
  {
    char* end = index(start, ' ');
    if(end)
    {
      *end = '\n';
      *(++end) = '\0';
      strcpy(answer, start);
    }
  }
}
```

```
void processLine(char* buffer, int len)
{
  // line format is "username\n"
  char answer[256];
  char command[256];
  sprintf(command, "ps aux |grep %s |wc > tmp.txt", buffer);
  fprintf(stdout, "Buf %s\n", buffer);
  fprintf(stdout, "About to execute %s\n", command);
  system(command);
  extractCountFromFile("tmp.txt", answer);
  assert(write(connFd, answer, strlen(answer)) >= 0);
}
```

```
int main(int argc, char* argv[])
{
  char buf[256];
  int n;
  if(argc ==2)
  {
   port = atoi(argv[1]);
  }
  setupSocket();
  while(getLineFromSocket(buf, 256))
    processLine(buf, n);
}
```

Live Demonstration of Exploitation

General Point

- When writing a server
  - Task is to mediate access to server's resources
  - Not grant arbitrary access
  - Have to be very careful in channeling
    - Constrained client-server protocol
    - To general-purpose OS/computer
- Attackers are evil/bad/smart/patient
- They are out to get you!


I must not tell lies

Side Note

- SQL Injection Vulnerabilities are closely related
  - Eg ';' passed through to SQL server is a statement separator there too.
- The general issue is failure to properly sanitize input before passing it to general execution engines.

Common Vulnerabilities and Exposures List (CVE)

- Initiative by Mitre Corp to create a common dictionary of known vulnerabilities
  - US govt funded
- Now an industry standard
- Guided by an editorial board from across industry/academia/government
- Excellent Place to Start Looking for Publicly Known Vulnerabilities in something
  - http://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=excel

‘Excel’ Vulnerabilities

Search Results

There are **201** CVE entries that match your search.

Name	Description
CVE-2012-5672	Microsoft Excel Viewer (aka Xlview.exe) and Excel in Microsoft Office 2007 (aka Office 12) allow remote attackers to cause a denial of service (read access violation and application crash) via a crafted spreadsheet file, as demonstrated by a .xls file with battery voltage data.
CVE-2012-4233	LibreOffice 3.5.x before 3.5.7.2 and 3.6.x before 3.6.1, and OpenOffice.org (OOo), allows remote attackers to cause a denial of service (NULL pointer dereference) via a crafted (1) odt file to vcllo.dll, (2) ODG (Drawing document) file to svxcorelo.dll, (3) PolyPolygon record in a .wmf (Window Meta File) file embedded in a ppt (PowerPoint) file to tllo.dll, or (4) xls (Excel) file to scfiltlo.dll.
CVE-2012-2543	Stack-based buffer overflow in Microsoft Excel 2007 SP2 and SP3 and 2010 SP1; Office 2011 for Mac; Excel Viewer; and Office Compatibility Pack SP2 and SP3 allows remote attackers to execute arbitrary code via a crafted spreadsheet, aka "Excel Stack Overflow Vulnerability."
CVE-2012-1887	Use-after-free vulnerability in Microsoft Excel 2003 SP3, 2007 SP2 and SP3, and 2010 SP1, and Office 2008 and 2011 for Mac, allows remote attackers to execute arbitrary code via a crafted spreadsheet, aka "Excel SST Invalid Length Use After Free Vulnerability."
CVE-2012-1886	Microsoft Excel 2003 SP3, 2007 SP2 and SP3, and 2010 SP1; Excel Viewer; and Office Compatibility Pack SP2 and SP3 allow remote attackers to execute arbitrary code or cause a denial of service (memory corruption) via a crafted spreadsheet, aka "Excel Memory Corruption Vulnerability."
CVE-2012-1885	Heap-based buffer overflow in Microsoft Excel 2003 SP3, 2007 SP2 and SP3, and 2010 SP1; Office 2008 and 2011 for Mac; and Office Compatibility Pack SP2 and SP3 allows remote attackers to execute arbitrary code via a crafted spreadsheet, aka "Excel SerAuxErrBar Heap Overflow Vulnerability."
CVE-2012-1847	Microsoft Excel 2003 SP3, 2007 SP2 and SP3, and 2010 Gold and SP1; Office 2008 and 2011 for Mac; Excel Viewer; and Office Compatibility Pack SP2 and SP3 do not properly handle memory during the opening of files, which allows remote attackers to execute arbitrary code via a crafted spreadsheet, aka "Excel Series Record Parsing Type Mismatch Could Result in Remote Code Execution Vulnerability."
CVE-2012-0185	Heap-based buffer overflow in Microsoft Excel 2007 SP2 and SP3 and 2010 Gold and SP1, Excel Viewer, and Office Compatibility Pack SP2 and SP3 allows remote attackers to execute arbitrary code via a crafted spreadsheet that triggers incorrect handling of memory during opening, aka "Excel MergeCells Record Heap Overflow Vulnerability."
CVE-2012-0184	Microsoft Excel 2003 SP3, 2007 SP2 and SP3, and 2010 Gold and SP1; Office 2008 and 2011 for Mac; Excel Viewer; and Office Compatibility Pack SP2 and SP3 do not properly handle memory during the opening of files, which allows remote attackers to execute arbitrary code via a crafted spreadsheet, aka "Excel SXLI Record Memory Corruption Vulnerability."
CVE-2012-0143	Microsoft Excel 2003 SP3 and Office 2008 for Mac do not properly handle memory during the opening of files, which allows remote attackers to execute arbitrary code via a crafted spreadsheet, aka "Excel Memory Corruption Using Various Modified Bytes Vulnerability."
CVE-2012-0142	Microsoft Excel 2003 SP3, 2007 SP2 and SP3, and 2010 Gold and SP1; Office 2008 for Mac; Excel Viewer; and Office Compatibility Pack SP2 and SP3 do not properly handle memory during the opening of files, which allows remote attackers to execute arbitrary code via a crafted spreadsheet, aka "Excel File Format Memory Corruption in OBJECTLINK Record Vulnerability."
CVE-2012-0141	Microsoft Excel 2003 SP3, 2007 SP2 and SP3, and 2010 Gold and SP1; Office 2011 for Mac; Excel Viewer; and Office Compatibility Pack SP2 and SP3 do not properly handle memory during the opening of files, which allows remote attackers to execute arbitrary code via a crafted spreadsheet, aka "Excel File Format Memory Corruption Vulnerability."

Example Entry

CVE-ID	
CVE-2012-2543	Learn more at National Vulnerability Database (NVD) • Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings

Description

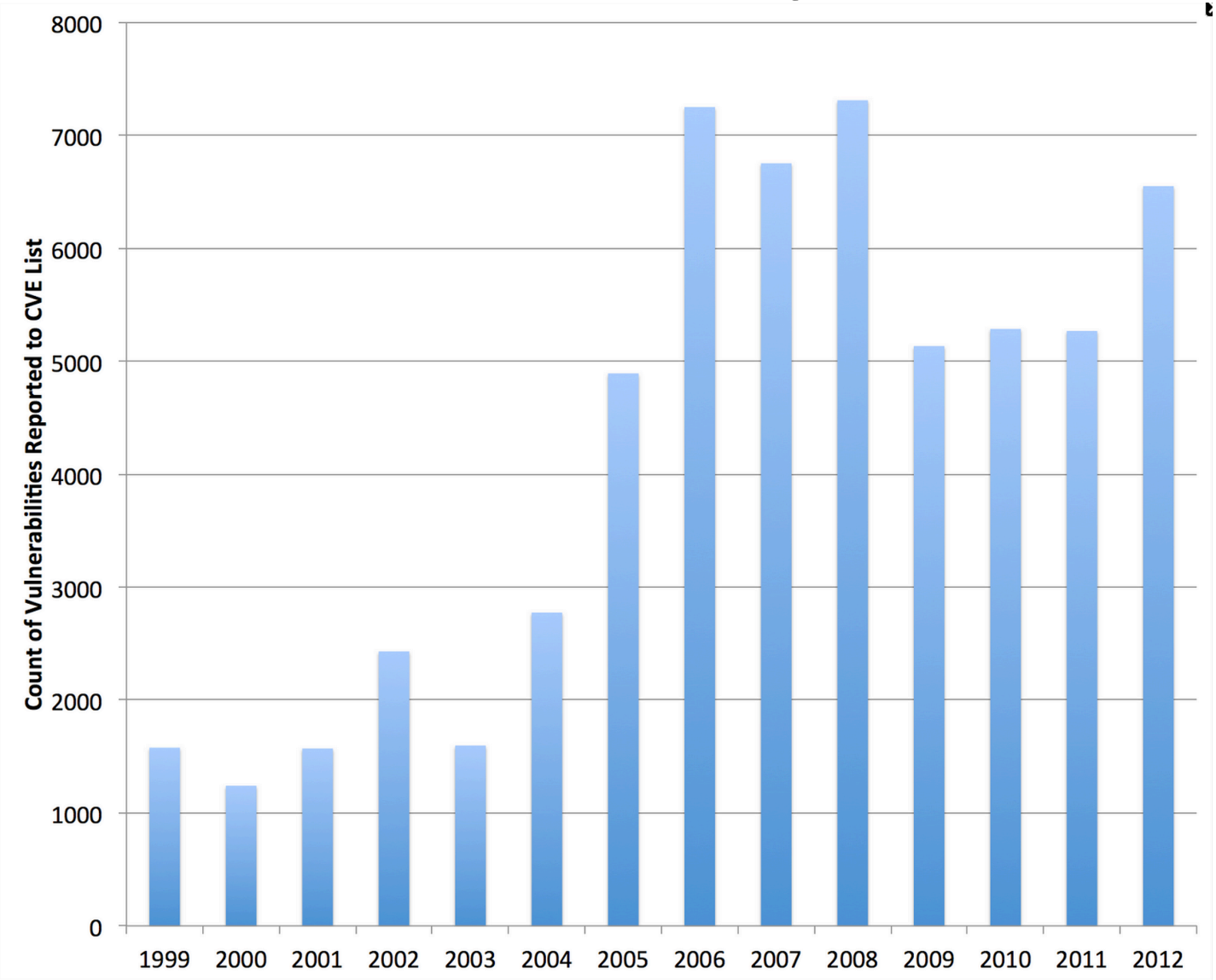
Stack-based buffer overflow in Microsoft Excel 2007 SP2 and SP3 and 2010 SP1; Office 2011 for Mac; Excel Viewer; and Office Compatibility Pack SP2 and SP3 allows remote attackers to execute arbitrary code via a crafted spreadsheet, aka "Excel Stack Overflow Vulnerability."

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

- MS:MS12-076
- URL:http://technet.microsoft.com/security/bulletin/MS12-076
- CERT:TA12-318A
- URL:http://www.us-cert.gov/cas/techalerts/TA12-318A.html
- BID:56431
- URL:http://www.securityfocus.com/bid/56431
- SECTRACK:1027752
- URL:http://www.securitytracker.com/id?1027752

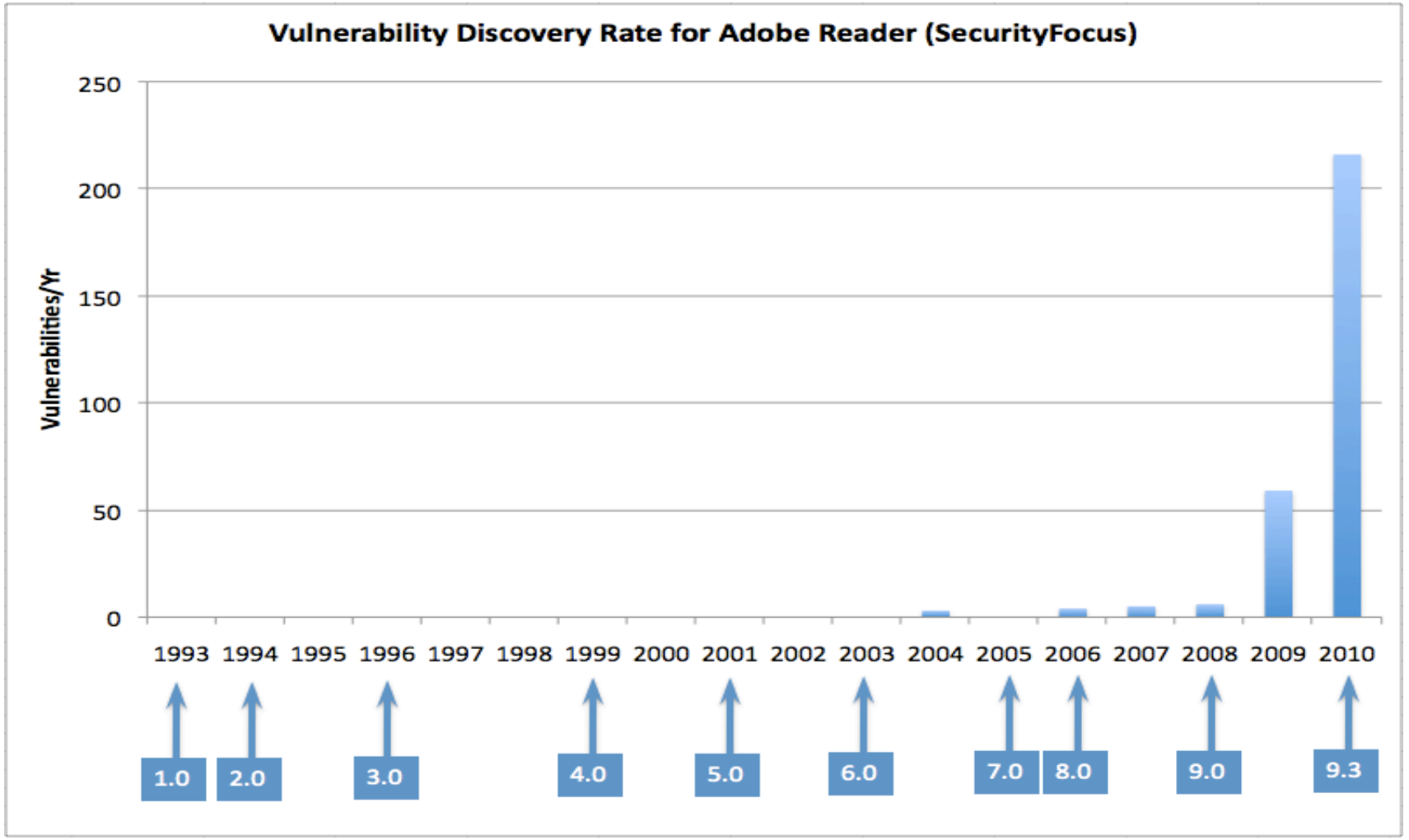
CVE Counts By Year



Is That All The Vulnerabilities?

- No!
  - Anecdote/single datapoint (8000 vs 122)
- Basically, we have no idea how many software vulnerabilities exist in total
  - Probably millions at least,
  - Probably not billions

Example


Vulnerability Discovery Rate for Adobe Reader (SecurityFocus)
Vulnerabilities/Yr
250
200
150
100
50
0
1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
9.3

Buffer Overflow Vulnerabilities

- Most important early class of vulnerabilities
  - Still important
- Will start today, finish in subsequent lecture(s)
- Today, will introduce a "fictionalized" account
  - How things used to be 10-20 years ago
  - Simpler to understand
  - Will not match what happens if you look at output of a modern compiler
    - Modern OS/compilers have numerous defenses
    - Still sometimes vulnerable, more complex to exploit
  - We will expand into more realistic detail next few lectures
- Loosely based on Aleph1 *Smashing Stack for Fun and Profit*.
  - http://www-inst.eecs.berkeley.edu/~cs161/fa08/papers/stack_smashing.pdf

Example 1

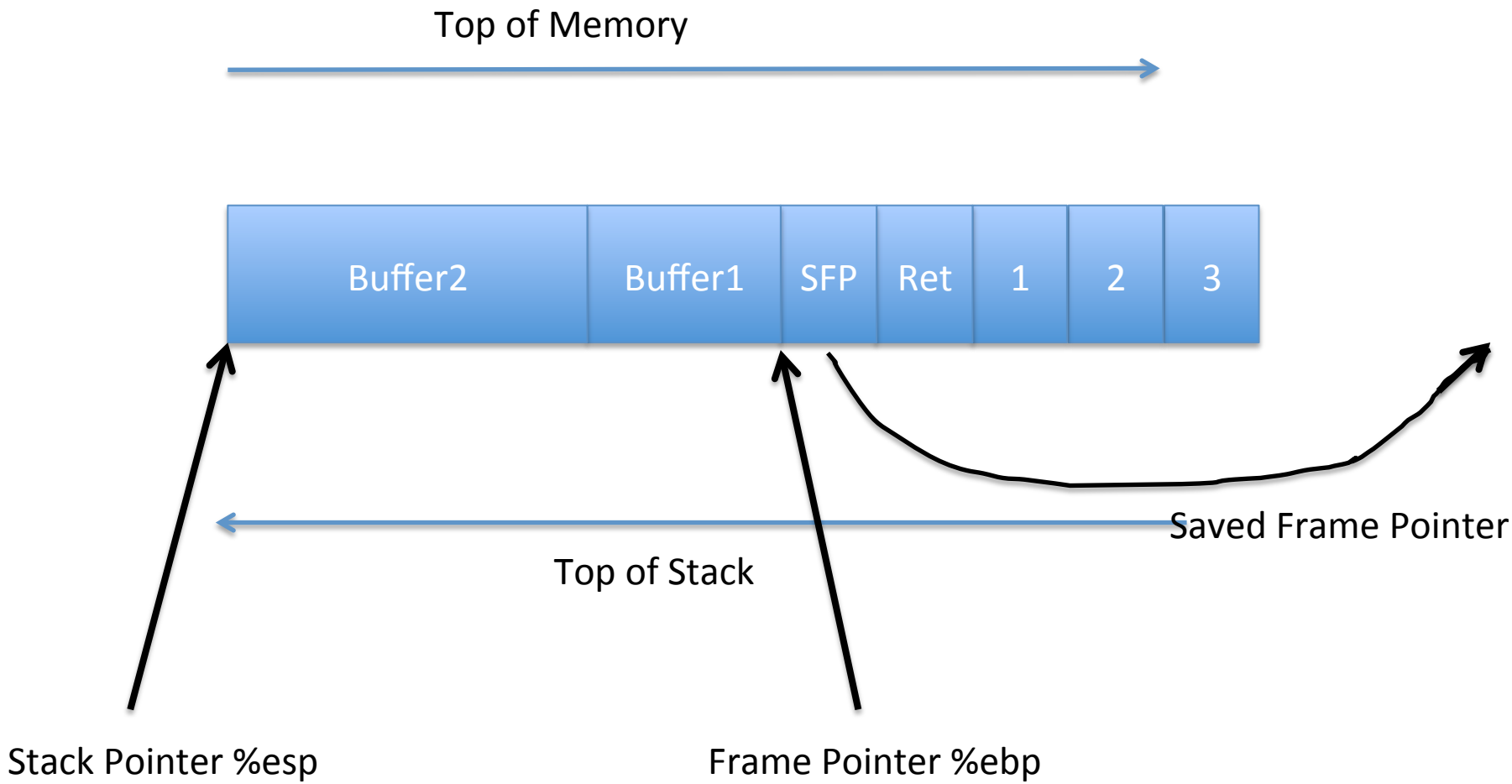
```
void myFunc(int a, int b, int c)
{
  char buffer1[5];
  char buffer2[10];
}

int main(int argc, char* argv[])
{
  myFunc(1,2,3);
}
```

Assembler

- Function Call:
  - pushl $3
  - pushl $2
  - pushl $1
  - call myFunc
- Function Prologue:
  - pushl %ebp
  - movl %esp,%ebp
  - subl $20,%esp

Stack in Example 1

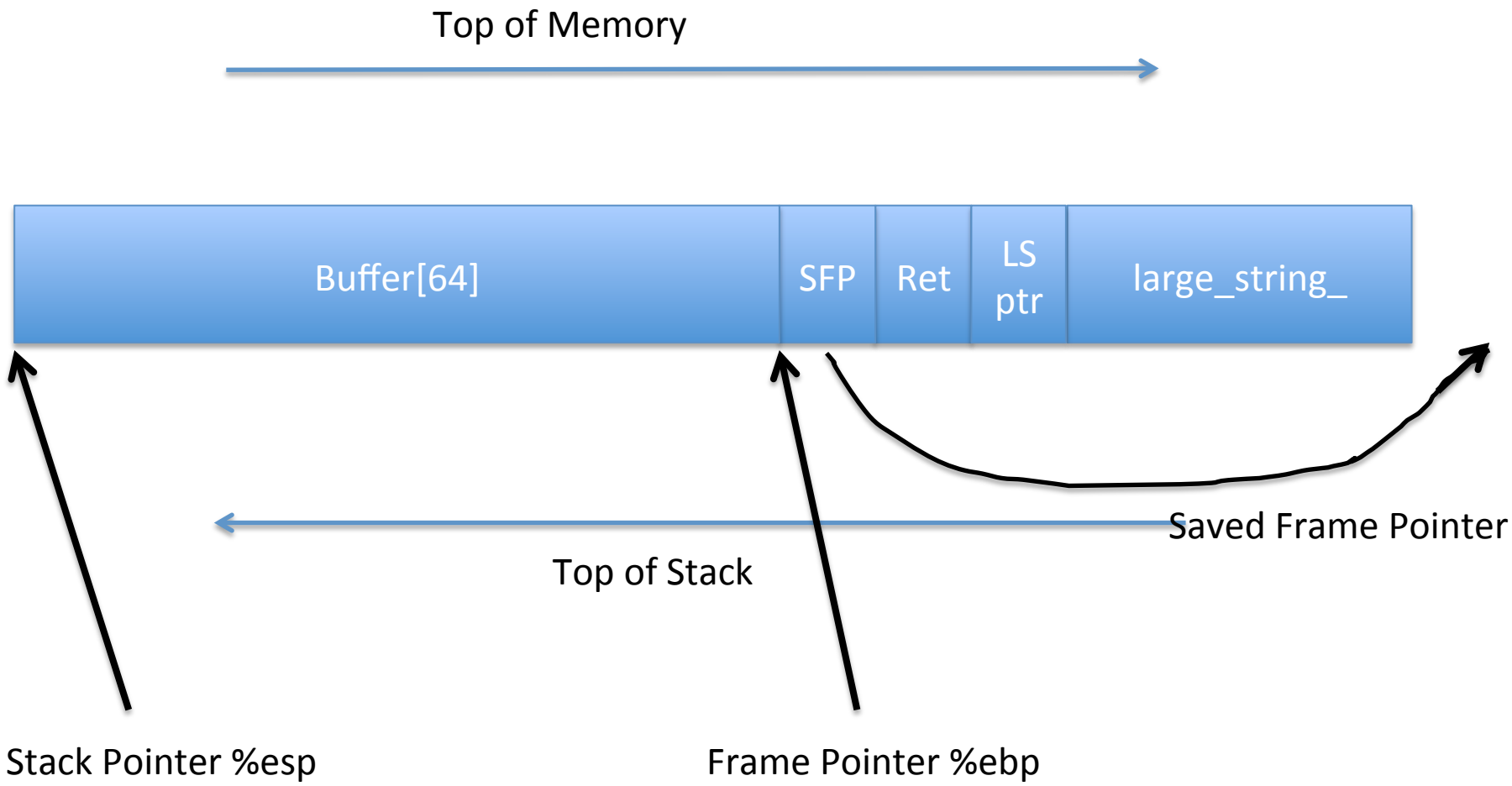


Example 2

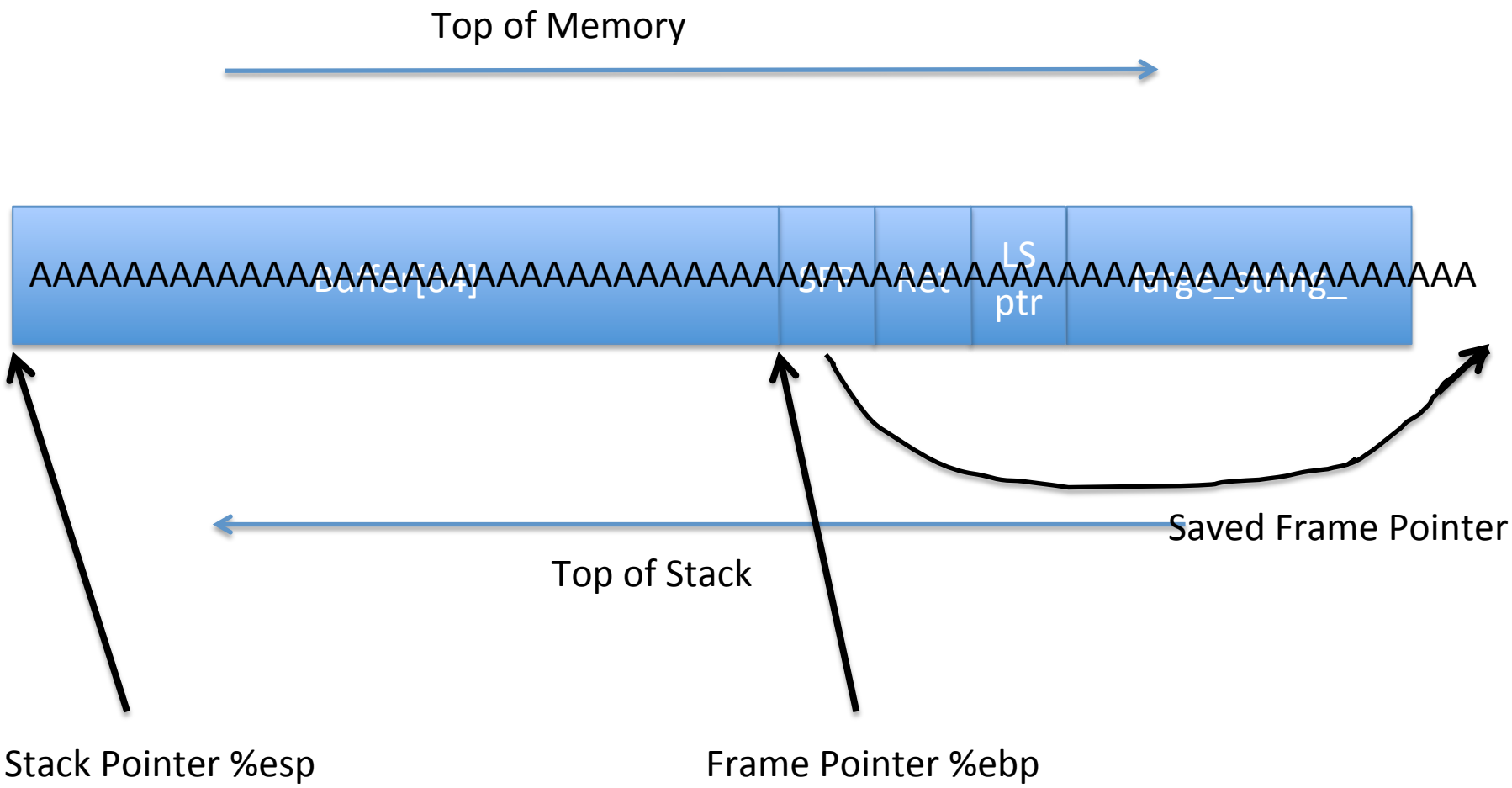
```
void myFunc(char *str)
{
  char buffer[64];
  strcpy(buffer, str);
}

int main(int argc, char* argv[])
{
  char large_string[256];
  int i;
  for( i = 0; i < 255; i++)
    large_string[i] = 'A';
  myFunc (large_string);
}
```

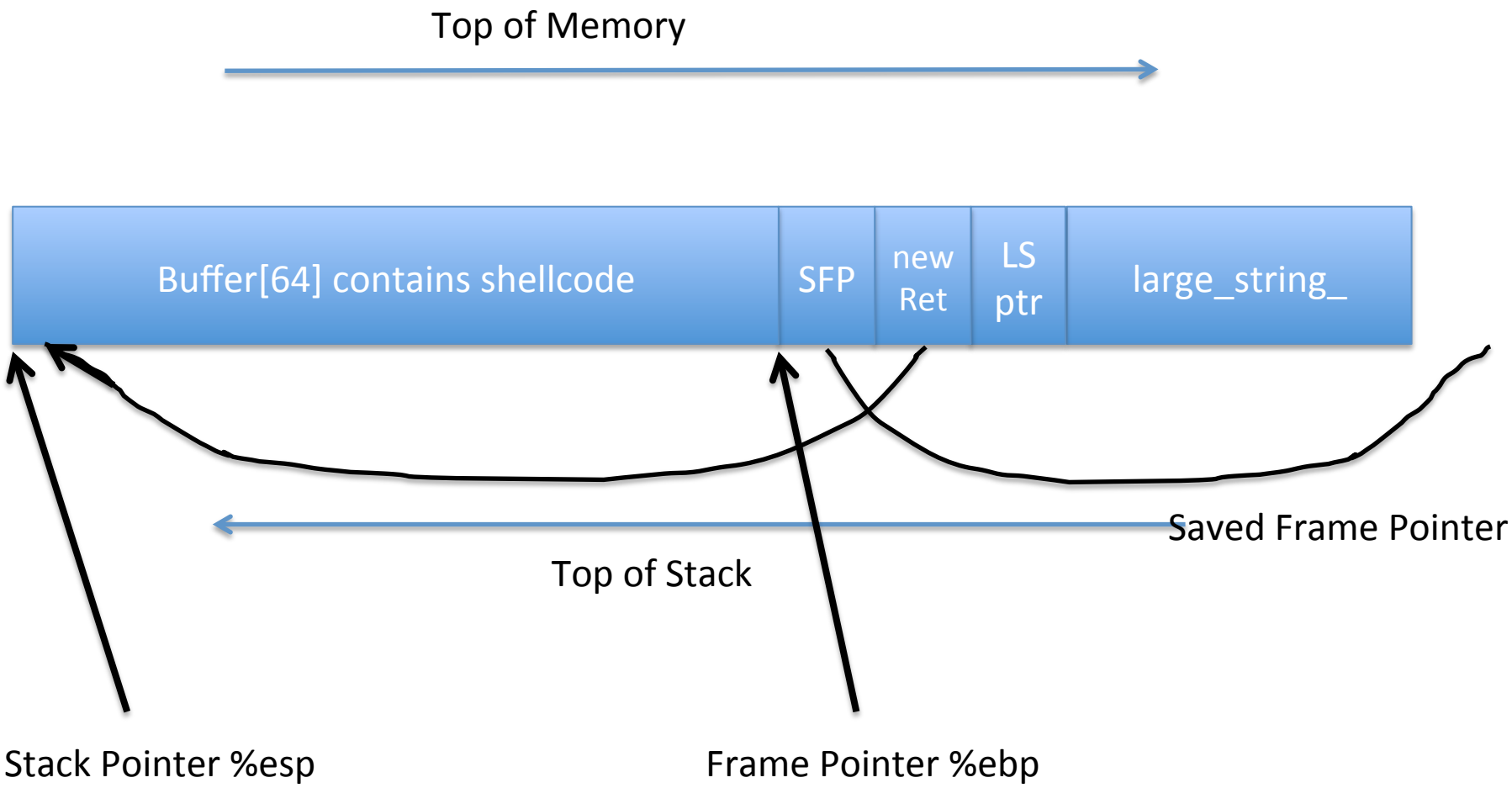
Stack in Example 2
right before strcpy()



Stack in Example 2 right after strcpy()



More Useful Stack for Attacker



Note similarity to System()

- Both cases it's channel mixing
  - ";" mixed with commands in shell language
  - Instruction pointers mixed with data
- Mixing control and data is frequently useful
  - But usually dangerous