Defending Computer Networks

Lecture 3: More On Vulnerabilities

Stuart Staniford

Adjunct Professor of Computer Science

Logistics

- Enrollment
  - Send request to cs-course-enroll@cornell.edu
  - Cc me (sgs235@cornell.edu)
- CMS set up
- Piazza set up (will email invites shortly)

Main Goals for Today

- Understand format string vulnerabilities
- Introduction to CWE - classifying vulnerabilities
- The economic/social big picture: why is the Internet riddled with vulnerabilities?
- Understand stack canary defenses against buffer overflows
- Other types of memory vulnerabilities

White House readies cyber sanctions against China ahead of state visit

By **Tal Kopan**, CNN

Story highlights

A timeline for when the U.S. might move on sanctions has yet to be decided, a government official says

The sanctions would go after Chinese entities that steal American business secrets, a practice U.S. companies have long complained of

Washington (CNN)—The White House is preparing a slate of sanctions it could bring against Chinese enterprises in response to cyberattacks against American businesses, a government official familiar with the process told CNN.

The move is a show of force on the issue just a few weeks ahead of a state visit by Chinese President Xi Jinping, and comes amid growing tensions between Washington and Beijing over China's increasingly assertive national security posture.

Preparing the sanctions is the latest in a series of steps the administration has taken to try to show that it takes cyberattacks on U.S. business seriously -- but following through with sanctions would take the issue to a new level.

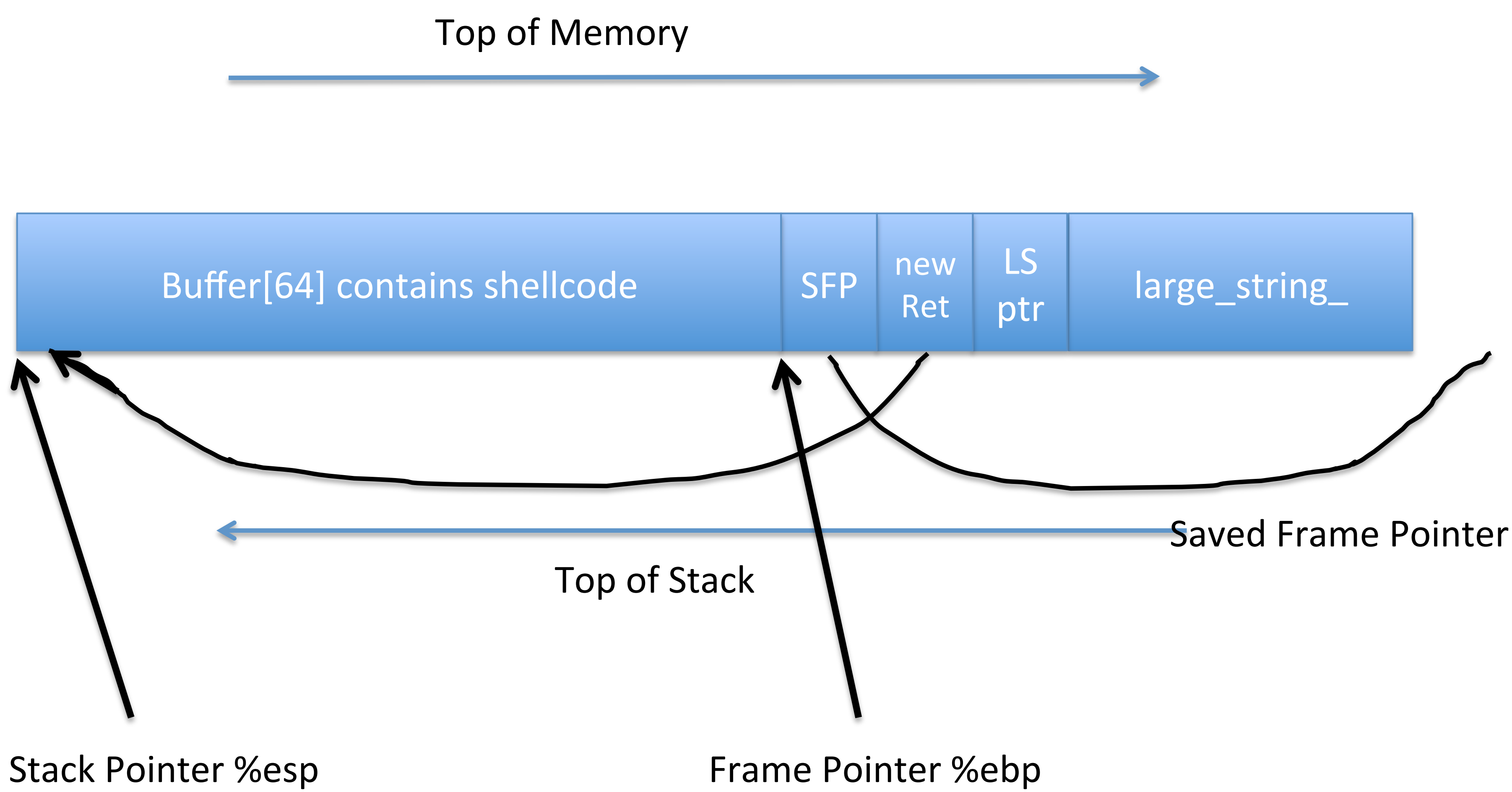
Refresh – System vulnerabilties

- sprintf(command, “blah %s blah”, userInput);
- system(command);

Refresh: 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);
}
```

Refresh: Stack On Exploit



Format String Vulnerabilities

- Class of vulnerabilities in C printf/sprintf/etc
  - Discovered at end of 1990s
  - Almost thirty years after C invented
- Also can affect syslog(), warn(), err()
- Core issue is when the format string is (partially) user supplied, not static
  - printf(userData,…) is bad
  - printf("%s", userData,…) doesn't have the issue
- Note that '…' arguments are on the stack
- And format string controls powerful functionality to do stuff with the printf arguments (ie the stack)
- Bypass any compile time checks when user supplied

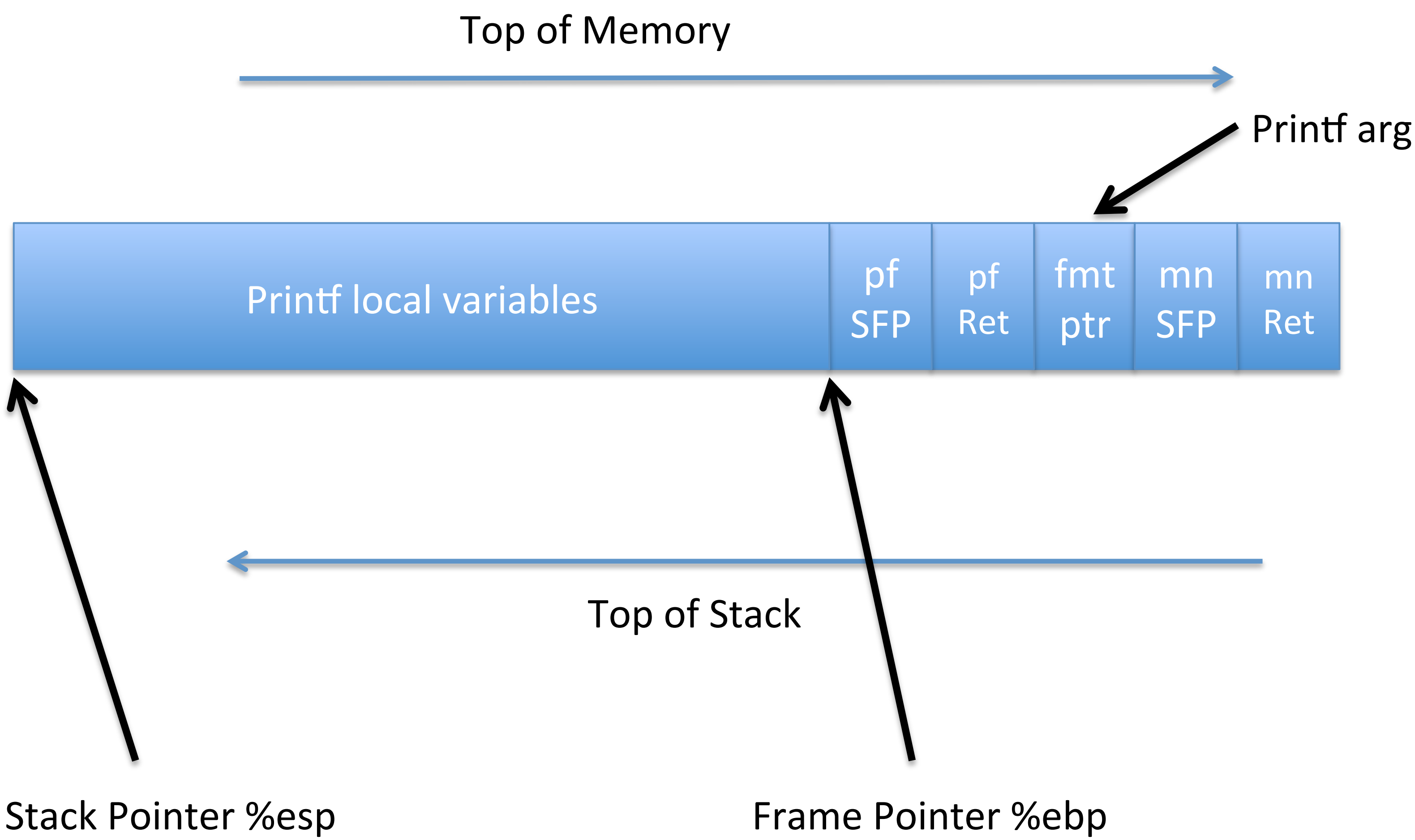
printfVulnerability.c

```
#include <stdio.h>

int main(int argc, char* argv[])
{
  printf(argv[1]);
  return 0;
}
```

So let’s try…

- ./printfVulnerability foo
- ./printfVulnerability “foo \n”
- ./printfVulnerability “%08x.%08x.%08x.%08x ”

Stack in printf Vulnerability



Still in the no-defense, old-style, slightly fictionalized view of the world

The Really Big Problem

The ‘%n’ format directive. From ‘man 3 printf’:

```
n       The number of characters written so far is stored into the inte-
        ger indicated by the int * (or variant) pointer argument.  No
        argument is converted.
```

This allows us to write on the stack at a location we can control! by doing ./printfVulnerability “%08x.%08x… %n” we can move around the stack position where the %n will write to.

By doing “aaaaa…%08x.%08x… %n” we can control what number is written into that stack address.

By doing this repeatedly with small width fields, we can write one byte at a time, and overwrite an address (eg a saved IP)

Example CVE Entry

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

Description

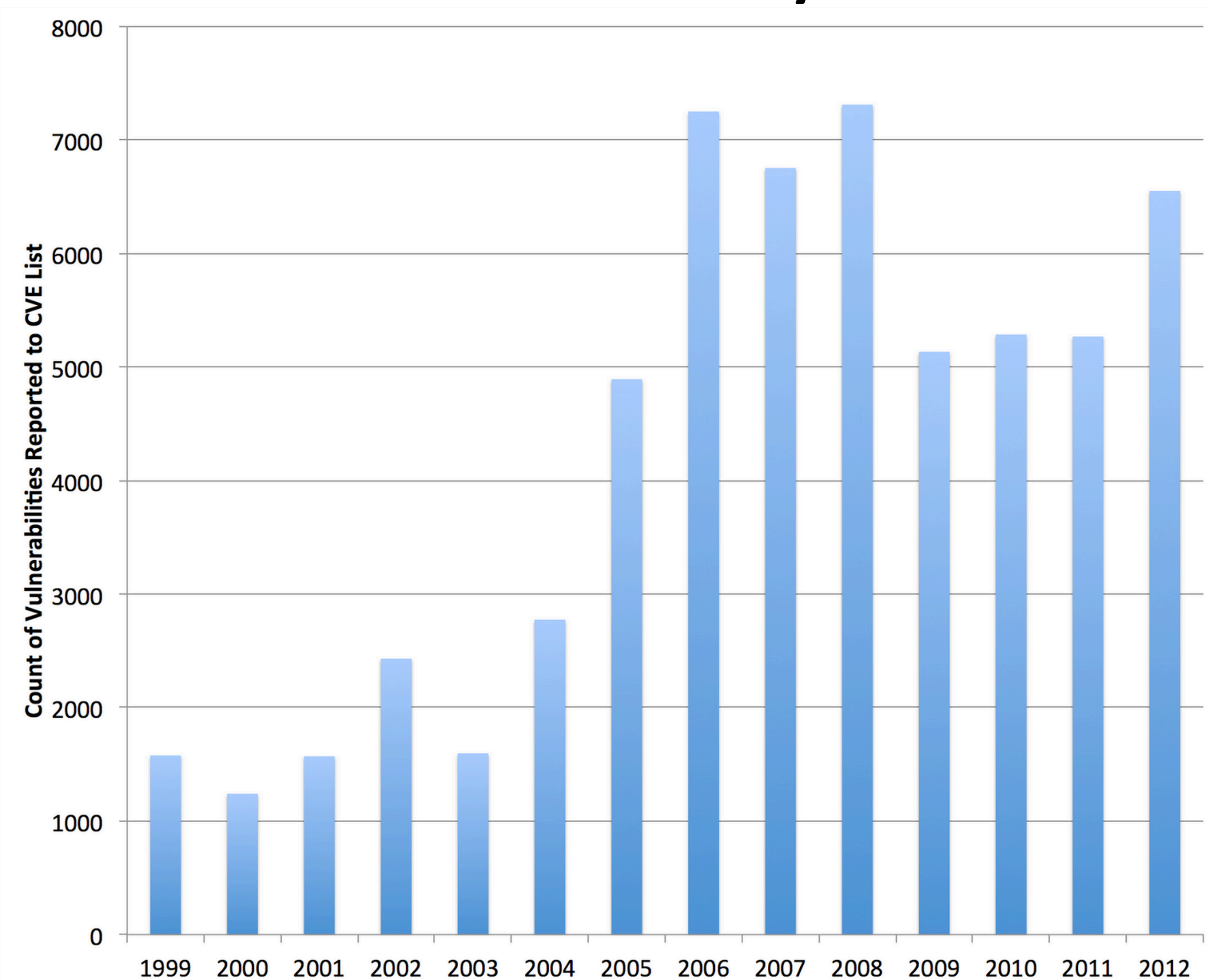
Stack-based buffer overflow in Microsoft Excel 2007 SP2 and SP3 and 2010 SP1; Office 2011 for Mac; Excel Viewer; and Office Compatibility Pack SP2 and SP3 allows remote attackers to execute arbitrary code via a crafted spreadsheet, aka "Excel Stack Overflow Vulnerability."

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



Common Weakness Enumeration(CWE)

- More recent effort by Mitre
  - cwe.mitre.org
- Idea is to classify vulnerabilities into general types
  - See the recurring patterns
  - Prioritize what's most important
  - Learn not to generate more and more of these things
- Excellent place to look for general issues when you get into a new domain

CWE Top 25

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	CWE-250	Execution with Unnecessary Privileges
[12]	70.1	CWE-352	Cross-Site Request Forgery (CSRF)
[13]	69.3	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	CWE-494	Download of Code Without Integrity Check
[15]	67.8	CWE-863	Incorrect Authorization
[16]	66.0	CWE-829	Inclusion of Functionality from Untrusted Control Sphere
[17]	65.5	CWE-732	Incorrect Permission Assignment for Critical Resource
[18]	64.6	CWE-676	Use of Potentially Dangerous Function
[19]	64.1	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
[20]	62.4	CWE-131	Incorrect Calculation of Buffer Size
[21]	61.5	CWE-307	Improper Restriction of Excessive Authentication Attempts
[22]	61.1	CWE-601	URL Redirection to Untrusted Site ('Open Redirect')
[23]	61.0	CWE-134	Uncontrolled Format String
[24]	60.3	CWE-190	Integer Overflow or Wraparound
[25]	59.9	CWE-759	Use of a One-Way Hash without a Salt

Around 1000 CWE's in total – another 39 pages like this

Example From a Different Domain

J2EE Bad Practices: Direct Management of Connections

Weakness ID: 245 *(Weakness Variant)* **Status:** Draft

Description

Description Summary

The J2EE application directly manages connections, instead of using the container's connection management facilities.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

Java

Common Consequences

Scope	Effect
Other	**Technical Impact:** *Quality degradation*

Demonstrative Examples

Example 1

In the following example, the class DatabaseConnection opens and manages a connection to a database for a J2EE application. The method openDatabaseConnection opens a connection to the database using a DriverManager to create the Connection object conn to the database specified in the string constant CONNECT_STRING.

Example Language: **Java** *(Bad Code)*

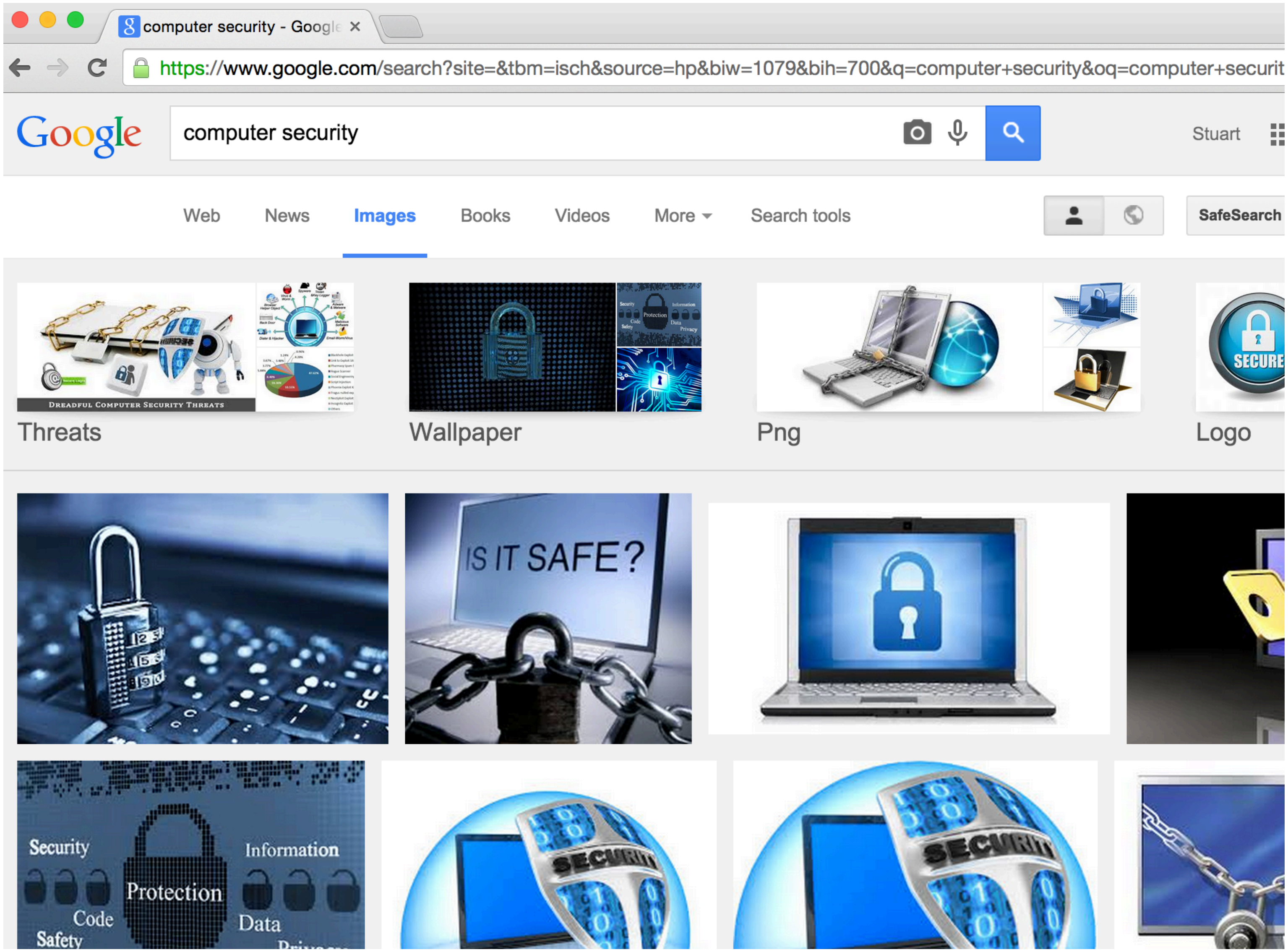
```
public class DatabaseConnection {
  private static final String CONNECT_STRING = "jdbc:mysql://localhost:3306/mysqldb";
  private Connection conn = null;

  public DatabaseConnection() {
  }

  public void openDatabaseConnection() {
    try {
      conn = DriverManager.getConnection(CONNECT_STRING);
    } catch (SQLException ex) {...}
  }

  // Member functions for retrieving database connection and accessing database
  ...
}
```

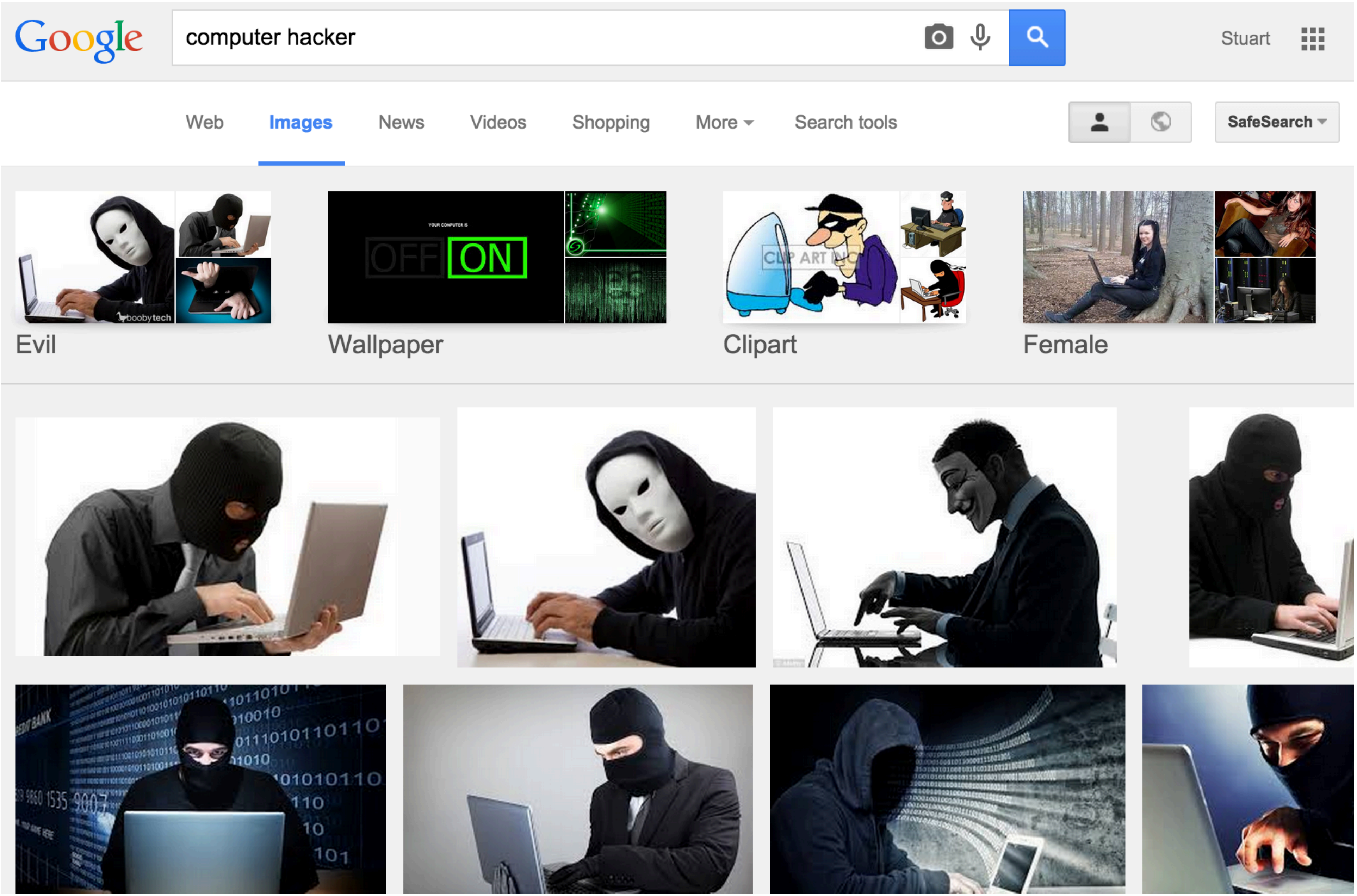
The use of the DriverManager class to directly manage the connection to the database violates the J2EE restriction against the direct management of connections. The J2EE application should use the web application container's resource management facilities to obtain a connection to the database as shown in the following example.

Why So Many Vulnerabilities?

1. Writing secure code is hard
2. Software engineers are poorly trained in security (but not you guys!)
3. Many people don't *like* security.
4. Economic incentives at software companies do not prioritize doing things right

Many people don't *like* security




Google
computer hacker
Stuart
Web
Images
News
Videos
Shopping
More
Search tools
SafeSearch
Evil
Wallpaper
Clipart
Female

Writing Secure Code is Hard

- ~1000 CWEs (and counting)
- Large numbers of different categories of things to do wrong
  - Hard to know about all of them
- Humans are error prone at best
  - Most of us write a noticeable number of bugs/kloc
  - Significant fraction of bugs turn out to be exploitable
  - Testing all code-paths is both theoretically and practically impossible

Software Engineers Untrained in Security

You are here → (#5)

Rank	School name	Score
#1	Carnegie Mellon University Pittsburgh, PA	5.0
#1	Massachusetts Institute of Technology Cambridge, MA	5.0
#1	Stanford University Stanford, CA	5.0
#1	University of California–Berkeley Berkeley, CA	5.0
#5	Cornell University Ithaca, NY	4.6
#5	University of Illinois–Urbana-Champaign Urbana, IL	4.6
#7	University of Washington Seattle, WA	4.5
#8	Princeton University Princeton, NJ	4.4
#8	University of Texas–Austin Austin, TX	4.4
#10	Georgia Institute of Technology Atlanta, GA	4.3

Source: http://grad-schools.usnews.rankingsandreviews.com/best-graduate-schools/top-science-schools/computer-science-rankings

Undergraduate Security Course @Top 10

	Available	Required
Carnegie Mellon	Yes	No
MIT	Yes	No
Stanford	Yes	No
Berkeley	Yes	No
Cornell	Yes	No
Illinois/Urbana-Champaign	Yes	No
University of Washington	Yes	No
Princeton	Yes	No
UT Austin	Yes	No
Georgia Tech	Yes	No
Purdue*	Yes	No
UC Davis*	Yes	No

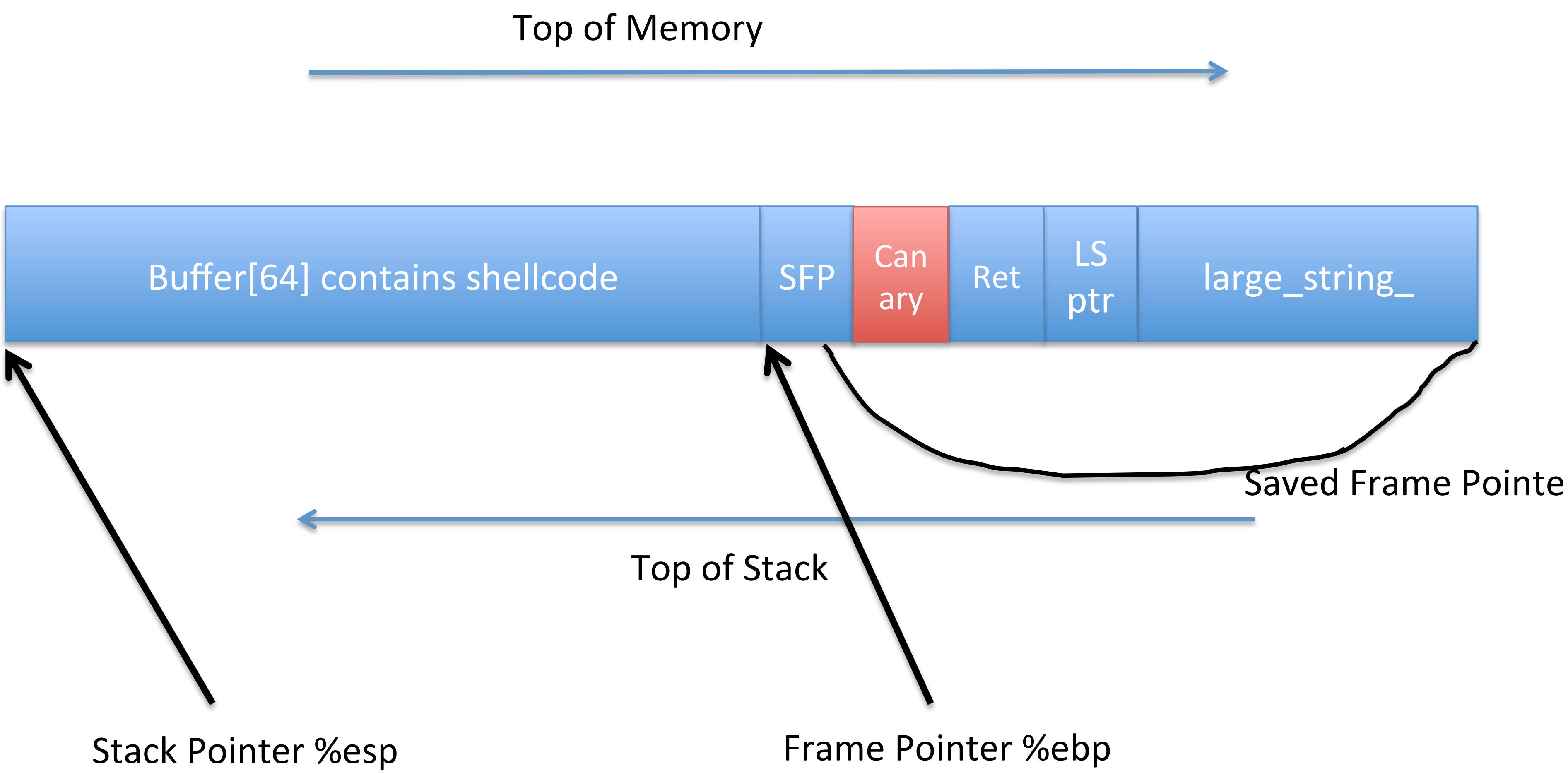
Economic Incentives

- Most tech market categories have a winner-take-all structure
  - Microsoft in operating systems and office suites
  - Google in search
  - Apple/Google in smartphones
  - Oracle in databases
  - Cisco in routers/switches
- Company executives/VCs know this
  - So very strong pressure to ship code fast
    - Dominate the market, then solve problems later

Economic Incentives

- Not writing vulnerabilities in the first place is
  - Hard
  - Therefore expensive and slow
- Finding vulnerabilities takes lots of QA/test
  - Also, expensive and slow
- So companies have a strong incentive to not worry about it **too** much
  - Fix it later, when it becomes a PR problem

Canary-based Overflow Defense



Not invincible. Eg http://phrack.org/issues.html?issue=56&id=5

Heap/BSS Overflows

- Heap is app dynamically allocated memory
  – malloc/new
- BSS is segment for static/global variables.
- Code vulnerabilities are conceptually similar to in stack case, but with heap/bss variables

```
char* foo = (char*)malloc(64);
if(foo)
  strcpy(foo, user)
```

Simple BSS example

```
int main(int argc, char **argv)
{
  FILE *tmpfd;
  static char buf[BUFSIZE], *tmpfile;
  tmpfile = "/tmp/vulprog.tmp";
  printf("Enter one line of data to put in %s: ", tmpfile);
  gets(buf);
  tmpfd = fopen(tmpfile, "w");
  fputs(buf, tmpfd);
  fclose(tmpfd);
}
```

Adapted from http://netsec.cs.northwestern.edu/media/readings/heap_overflows.pdf

Simple heap example

```
struct myObject
{
  char name[64];
  int (*foo)(int);
 …
}
```

Note that in C++, virtual functions are stored implicitly in object structure as function pointers

Use After Free()

CWE-416: Use After Free

Use After Free

Weakness ID: 416 *(Weakness Base)* **Status:** Draft

Description

Description Summary

Referencing memory after it has been freed can cause a program to crash, use unexpected values, or execute code.

Extended Description

The use of previously-freed memory can have any number of adverse consequences, ranging from the corruption of valid data to the execution of arbitrary code, depending on the instantiation and timing of the flaw. The simplest way data corruption may occur involves the system's reuse of the freed memory. Use-after-free errors have two common and sometimes overlapping causes:

- Error conditions and other exceptional circumstances.
- Confusion over which part of the program is responsible for freeing the memory.

In this scenario, the memory in question is allocated to another pointer validly at some point after it has been freed. The original pointer to the freed memory is used again and points to somewhere within the new allocation. As the data is changed, it corrupts the validly used memory; this induces undefined behavior in the process.

If the newly allocated data chances to hold a class, in C++ for example, various function pointers may be scattered within the heap data. If one of these function pointers is overwritten with an address to valid shellcode, execution of arbitrary code can be achieved.