Defending Computer Networks
Lecture 4: Exploit Defenses

Stuart Staniford

Adjunct Professor of Computer Science

Logistics

- Course is now official
- Expect an email from Stephanie Meik with your PIN
- Need to enroll by tomorrow (Weds 9/11)
- If you didn't sign list in class, or talk to Stephanie
  - Go talk to her to ensure you get a PIN number

More Logistics

- Let's add a week to homework 1 –
  - now due Weds 9/18 @ 5pm
  - Just now have a TA candidate
  - Still dealing with very basic issues in lab
  - Corollary: my office hrs tomorrow will be normal 1:30-3pm
- Can we use Ubuntu in VM on laptop for hw?
  - Yes, but you're on your own for platform specific issues.

Latest News

N.S.A. Foils Much Internet Encryption

By NICOLE PERLROTH, JEFF LARSON and SCOTT SHANE
Published: September 5, 2013 | 352 Comments

The National Security Agency is winning its long-running secret war on encryption, using supercomputers, technical trickery, court orders and behind-the-scenes persuasion to undermine the major tools protecting the privacy of everyday communications in the Internet age, according to newly disclosed documents.

Enlarge This Image

Associated Press

This undated photo released by the United States government shows the National Security Agency campus in Fort Meade, Md.

The agency has circumvented or cracked much of the encryption, or digital scrambling, that guards global commerce and banking systems, protects sensitive data like trade secrets and medical records, and automatically secures the e-mails, Web searches, Internet chats and phone calls of Americans and others around the world, the documents show.

Many users assume — or have been assured by Internet companies — that their data is safe from prying eyes, including those of the government, and the N.S.A. wants to keep it that way. The agency treats its recent successes in deciphering protected information as among its most closely guarded secrets, restricted to those cleared for a highly classified program code-named Bullrun, according to the documents, provided by Edward J. Snowden, the former N.S.A. contractor.

This story has been reported in partnership among The New York Times, The Guardian and ProPublica based on documents obtained by The Guardian. For The Guardian: James Ball, Julian Borger, Glenn Greenwald. For The New York Times: Nicole Perlroth, Scott Shane. For ProPublica: Jeff Larson.

FACEBOOK
TWITTER
GOOGLE+
SAVE
E-MAIL
SHARE
PRINT
REPRINTS

Enough Said
Coming Soon
Watch Trailer ►

How to remain secure against NSA surveillance

The NSA has huge capabilities – and if it wants in to your computer, it's in. With that in mind, here are five ways to stay safe

Bruce Schneier
theguardian.com, Thursday 5 September 2013 15.06 EDT
Jump to comments (105)

Now that we have enough details about how the NSA eavesdrops on the internet, including today's disclosures of the NSA's deliberate weakening of cryptographic systems, we can finally start to figure out how to protect ourselves.

For the past two weeks, I have been working with the Guardian on NSA stories, and have read hundreds of top-secret NSA documents provided by whistleblower Edward Snowden. I wasn't part of today's story – it was in process well before I showed up – but everything I read confirms what the Guardian is reporting.

At this point, I feel I can provide some advice for keeping secure against such an adversary.

The primary way the NSA eavesdrops on internet communications is in the network. That's where their capabilities best scale. They have invested in enormous programs to automatically collect and analyze network traffic. Anything that requires them to attack individual endpoint computers is significantly more costly and risky for them, and they will do those things carefully and sparingly.

Leveraging its secret agreements with telecommunications companies – all the US and UK ones, and many other "partners" around the world – the NSA gets access to the communications trunks that move internet traffic. In cases where it doesn't have that sort of friendly access, it does its best to surreptitiously monitor communications channels: tapping undersea cables, intercepting satellite communications, and so on.

More…

(TS//SI//NF) The SIGINT Enabling Project actively engages the US and foreign IT industries to covertly influence and/or overtly leverage their commercial products' designs. These design changes make the systems in question exploitable through SIGINT collection (e.g., Endpoint, MidPoint, etc.) with foreknowledge of the modification. To the consumer and other adversaries, however, the systems' security remains intact. In this way, the SIGINT Enabling approach uses commercial technology and insight to manage the increasing cost and technical challenges of discovering and successfully exploiting systems of interest within the ever-more integrated and security-focused global communications environment.

(TS//SI//REL TO USA, FVEY) This Project supports the Comprehensive National Cybersecurity Initiative (CNCI) by investing in corporate partnerships and providing new access to intelligence sources, reducing collection and exploitation costs of existing sources', and enabling expanded network operation and intelligence exploitation to support network defense and cyber situational awareness. This Project contains the SIGINT Enabling Sub-Project.

(U) Base resources in this project are used to:

- (TS//SI//REL TO USA, FVEY) Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.
- (TS//SI//REL TO USA, FVEY) Collect target network data and metadata via cooperative network carriers and/or increased control over core networks.
- (TS//SI//REL TO USA, FVEY) Leverage commercial capabilities to remotely deliver or receive information to and from target endpoints.
- (TS//SI//REL TO USA, FVEY) Exploit foreign trusted computing platforms and technologies.
- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.
- (TS//SI//REL TO USA, FVEY) Make specific and aggressive investments to facilitate the development of a robust exploitation capability against Next-Generation Wireless (NGW) communications.

http://www.theguardian.com/world/interactive/2013/sep/05/sigint-nsa-collaborates-technology-companies

And more…

C.3. (TS//SI//REL) The fact that NSA/CSS has some capabilities against the encryption in TLS/SSL, HTTPS, SSH, VPNs, VoIP, WEBMAIL, and other network communication technologies	TOP SECRET//SI// REL TO USA, FVEY at a minimum See Remarks.

C.6. (TS//SI//REL TO USA, FVEY) The fact that NSA/CSS develops implants to enable a capability against the encryption used in network communication technologies	TOP SECRET//SI// REL TO USA, FVEY See Remarks.

http://www.theguardian.com/world/interactive/2013/sep/05/nsa-project-bullrun-classification-guide

And still more…

Privacy Scandal: NSA Can Spy on Smart Phone Data

SPIEGEL has learned from internal NSA documents that the US intelligence agency has the capability of tapping user data from the iPhone, devices using Android as well as BlackBerry, a system previously believed to be highly secure.

September 07, 2013 – 06:00 PM

Print | Send

Feedback

Tweet 3,783 Recommend 2.4k +1

TOPIC
NSA Spying Scandal

National Security Agency

Apple

Related SPIEGEL ONLINE links

'Success Story': NSA Targeted French Foreign Ministry (09/01/2013)

Snowden Document: NSA Spied On Al Jazeera Communications (08/31/2013)

REUTERS

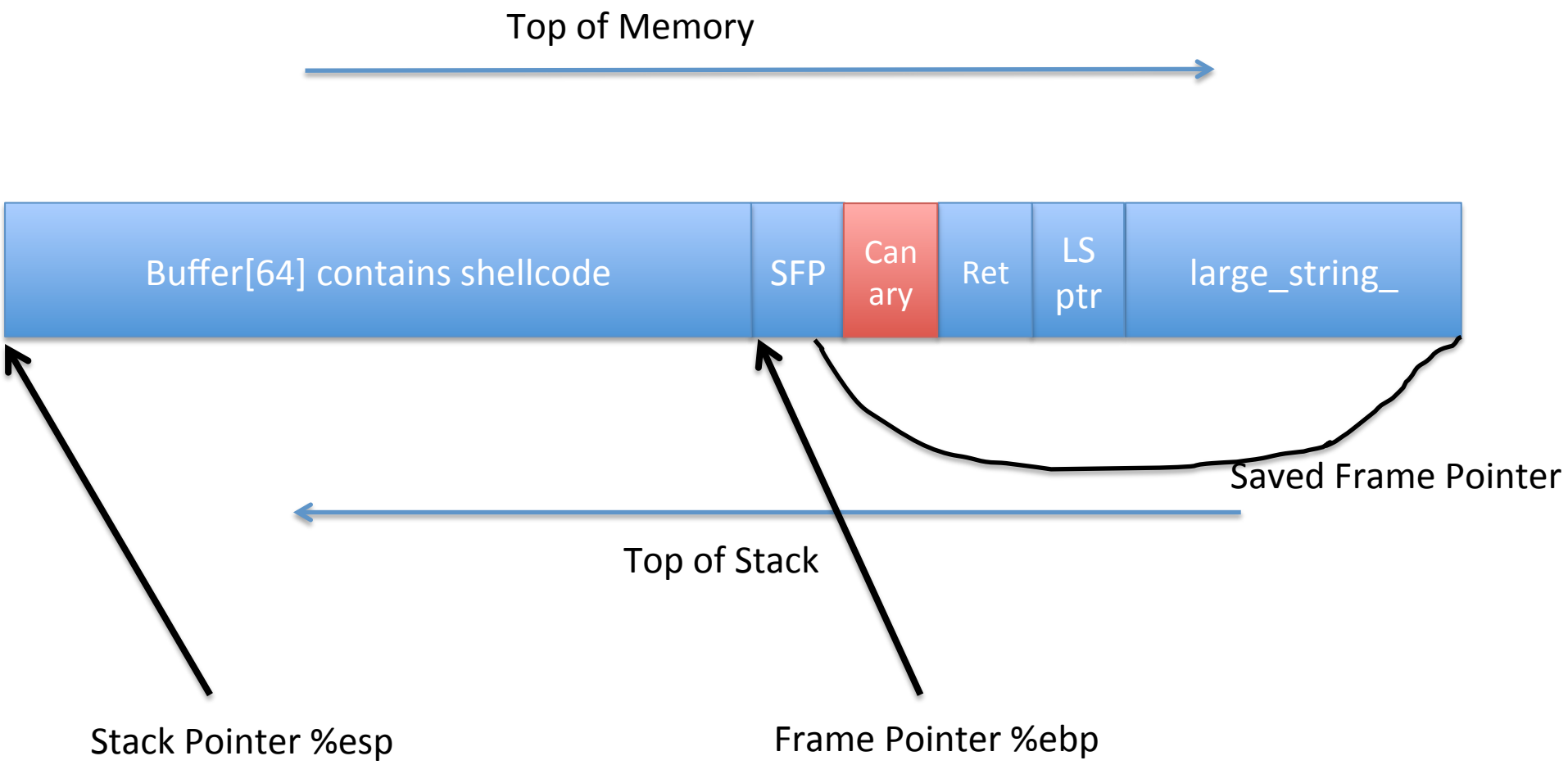
German Chancellor Angela Merkel holds a BlackBerry Z10 smart phone: Will the company face a setback following claims the NSA can spy on its phones?

The United States' National Security Agency intelligence-gathering operation is capable of accessing user data from smart phones from all leading manufacturers. Top secret NSA documents that SPIEGEL has seen explicitly note that the NSA can tap into such information on Apple iPhones, BlackBerry devices and Google's Android mobile operating system.

Main Goals for Today

- Heap/BSS Overflows and Vulnerabilities
  - Just a little taste
- Understand NX defenses against overflows
  - And sketch return oriented programming
- Understand Address Space Randomization
  - And how the dark side can work around it
- Discussion today probably a little sketchier
  - Want to point to various issues
  - But need to move on to other topics

Refresher: Canaries



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.

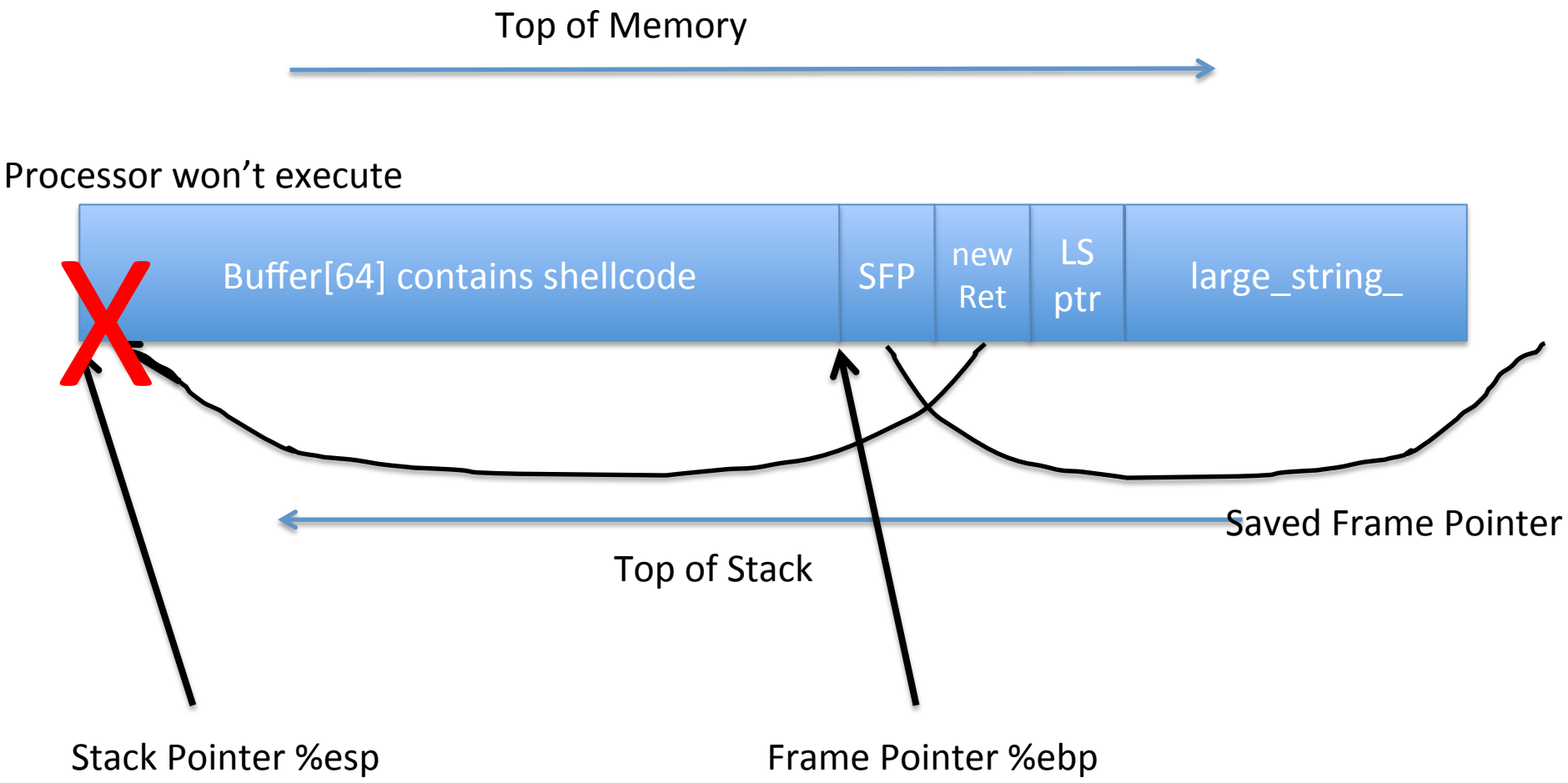
Heap Issues in General

- Often exploitable
- Not nearly as cookie-cutter as stack issues
- Requires more code-analysis from the attacker
  - Each case is different

NX/DEP/W^X

- Related mechanisms with common theme
  - Let's not execute the stack/heap
  - That way, cannot inject shellcode into buffer
  - And then point RIP at buffer contents
  - Ditto in format string attack, cannot put shellcode into buffer
- Requires hardware/OS support
  - But we have that now

NX/DEP/W^X


Top of Memory
Processor won't execute
Buffer[64] contains shellcode
SFP
new Ret
LS ptr
large_string_
Saved Frame Pointer
Top of Stack
Stack Pointer %esp
Frame Pointer %ebp

NX Bit

- General term for hardware feature
  - Originally AMD term
  - XD Bit (Intel)
  - XN Bit (ARM)
- Implemented in the page table
  - Bit 63 says this page cannot be executed
  - Hardware will enforce when doing memory lookup on instruction pointers in virtual address space
  - OS needs to manage the bits on the pages
    - Make sure stack and heap pages cannot execute

DEP: Data Execution Prevention

- Term for the OS Level feature
- Particularly on MS Windows
  - Controllable on a process-by-process basis
  - First optionally available on XP SP2 (circa 2004)
  - Default is still only to be available on core OS stuff
  - Optionally turn on for everything
    - Breaks some applications

W^X

- Write XOR Execute
- Extended version of idea
- All virtual pages can be
  - either writeable or executable
  - But not both
- Prevents self-modifying code
- OpenBSD, OS X, some Linux have full W^X

Defeating NX: Return to LIBC

- We can't make RIP point to our code
- But we can make it point to any pre-existing code on system
  - In text segment, so executable
- And we can set up stack beforehand
- Eg call system();
- So NX not invincible by itself

ROP: Return Oriented Programming

- Generalization of return-to-libc idea
- Shacham, 2007
- Idea is to crawl libc (etc), and find a series of "gadgets"
- A gadget is a useful bit of code right before the ret instruction of some function
- Might just be one or two instructions

Libc from a ROP POV

Etc, etc…

More ROP

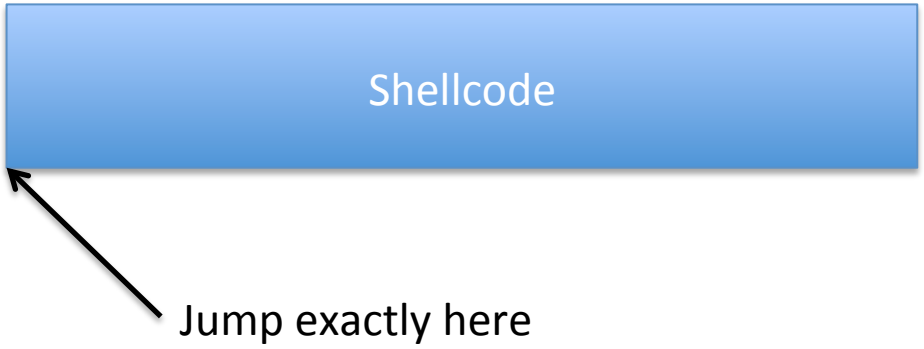
- Then call a bunch of these in sequence
  - from the (scribbled on) stack
- Shachem showed that libc ROP gadgets form a general purpose computation framework that can do anything.
- Very hard to fix this by surgery on libc

NOP Sleds

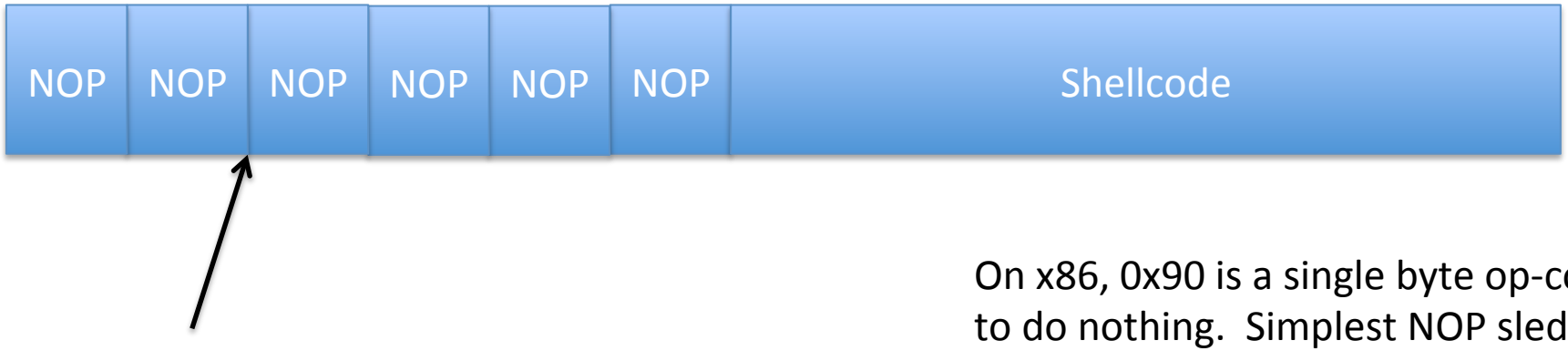
- When we overwrite an address in memory
  - Say a RIP
- We need to know what value to put.
- Simple case:
  - Beginning of shellcode in buffer
- But this is fragile.
  - Any slight difference in code version,
  - Even if it didn't affect the vulnerability
  - Could change the jump address

So allow for some imprecision

Instead of



Do



Jump somewhere in here. Now our exploit is less fragile. Assuming we have the space.

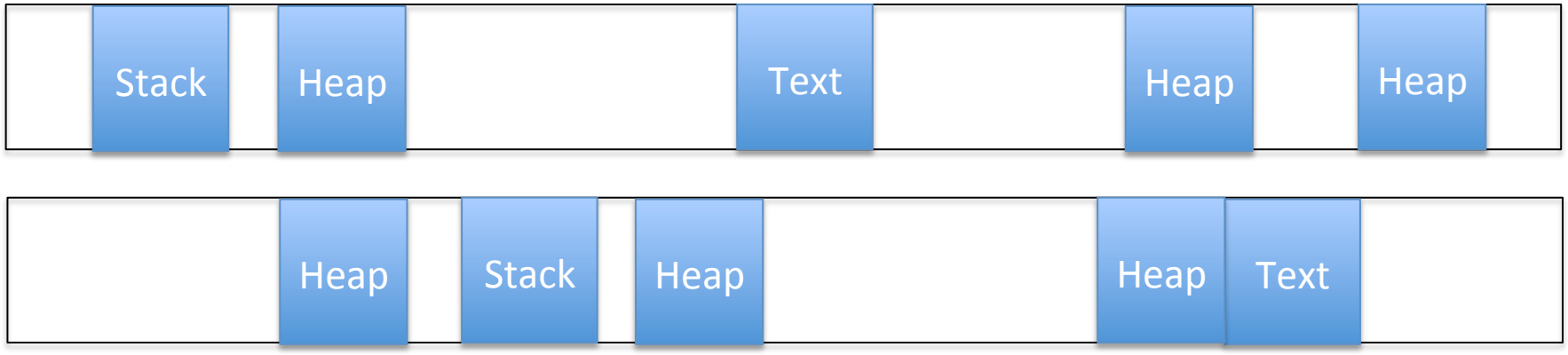
On x86, 0x90 is a single byte op-code to do nothing. Simplest NOP sled.

Address Space Layout Randomization

Basic insight is to make it really hard to figure out what address to jump to. Put key parts of the program in random places in memory

Instead of loading program into memory the same way every time:

Randomize:



So Now

- When we overwrite RIP, we don't know where to point, not even close
  - If we could get close, could use a NOP-sled

ASLR

- Now available on all major OS's
- Not all legacy code is compiled this way
  - May not be position-independent
- But increasingly becoming standard
- So a fully modern exploit must get past
  - Canaries
  - NX/DEP
  - ASLR
- All at the same time…
- But let's look at ASLR in isolation for a moment

Brute Forcing ASLR

- Loading at run time, we can't do fine-grained randomization
  - Code, for example, has all kinds of internal jumps that must be known, so code can't easily be jumbled up at the micro scale.
- Limited to moving around big chunks (stack, text, etc)
  - Consider an 8MB stack in 4GB (32 bit machine)
  - 512 possible positions. Not outrageous to guess
  - Much more difficult on 64 bit machines

Leaking Addresses

- Anything that allows us to see an address,
  - lets us get a handle on where that kind of thing lives
  - Eg format string vulnerability allows us to inspect the stack before doing our attack
    - We can quickly figure out where everything lives
      - Text pointers in RIPs
      - Stack pointers in stack frame bases
      - Heap pointers in local variable pointers to heap buffers

Defeating ALSR/DEP combined

- Any non-ALSR code can be analyzed for ROP.
  - Still sometimes libraries/code lying around. Eg
    - https://blogs.technet.com/b/srd/archive/2013/08/12/mitigating-the-ldrhotpatchroutine-dep-aslr-bypass-with-ms13-063.aspx

The bypass takes advantage of a predictable memory region known as SharedUserData that exists at a fixed location (0x7ffe0000) in every process on every supported version of Windows. On 64-bit versions of Windows prior to Windows 8, this region contains pointers to multiple functions in the 32-bit version of NTDLL that is used by WOW64 processes as shown below:

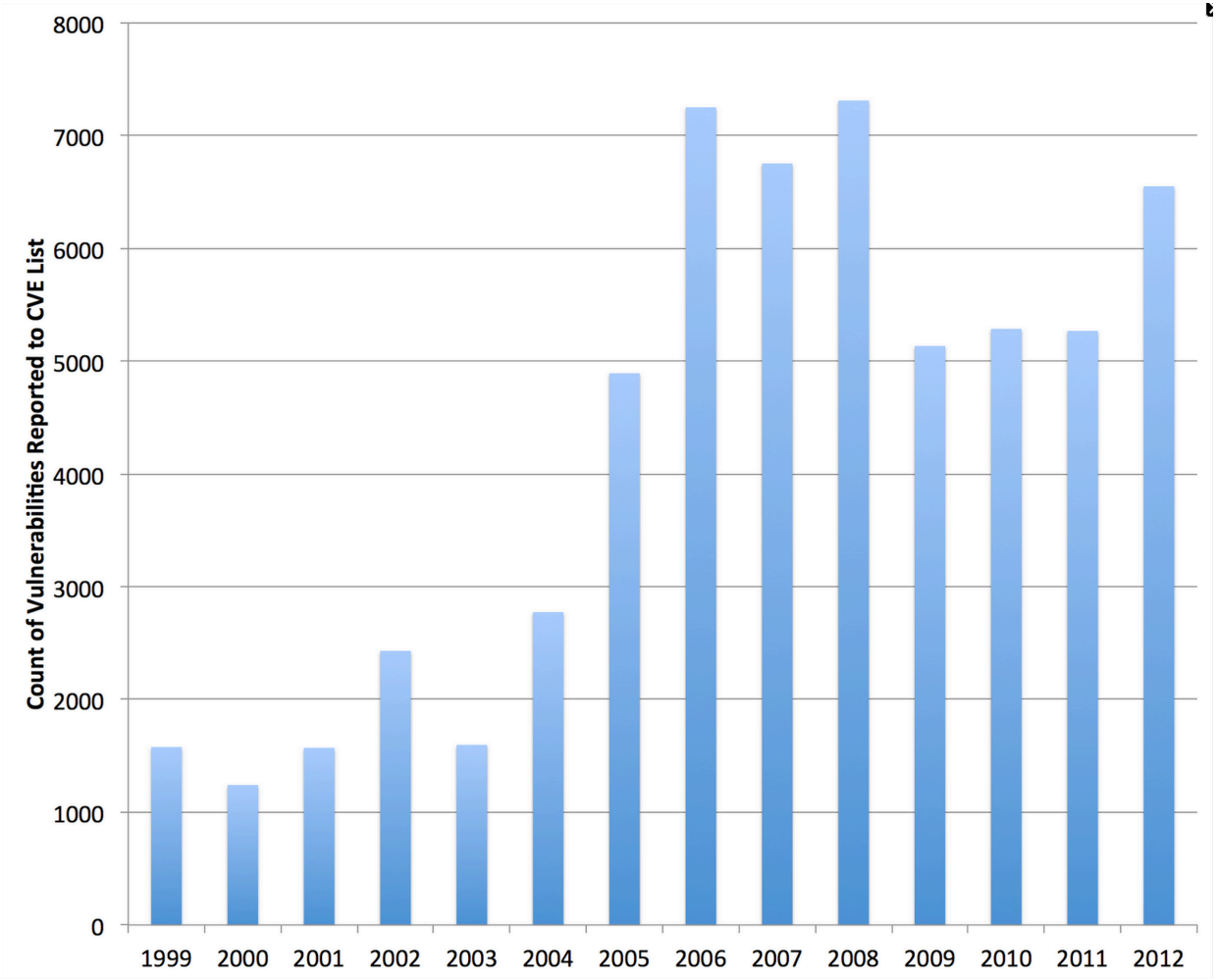
```
0:000> dds 7ffe0340 Lc
00000000`7ffe0340  77829ce9 ntdll32!LdrInitializeThunk
00000000`7ffe0344  77800100 ntdll32!KiUserExceptionDispatcher
00000000`7ffe0348  77800028 ntdll32!KiUserApcDispatcher
00000000`7ffe034c  778000b8 ntdll32!KiUserCallbackDispatcher
00000000`7ffe0350  7788f8d4 ntdll32!LdrHotPatchRoutine
00000000`7ffe0354  77822551 ntdll32!ExpInterlockedPopEntrySListFault
00000000`7ffe0358  7782251b ntdll32!ExpInterlockedPopEntrySListResume
00000000`7ffe035c  77822553 ntdll32!ExpInterlockedPopEntrySListEnd
00000000`7ffe0360  77800190 ntdll32!RtlUserThreadStart
00000000`7ffe0364  77892dfd ntdll32!RtlpQueryProcessDebugInformationRemote
00000000`7ffe0368  778517d9 ntdll32!EtwpNotificationThread
00000000`7ffe036c  777f0000 ntdll32!CsrServerApiRoutine
```

Defeating ALSR/DEP

- Getting harder – Microsoft will pay $100k for any new methods of doing it on Windows
  - http://www.microsoft.com/security/msrc/report/bypass_bounty.aspx

Bottom Line: Defenses Help

But not a panacea yet:



Problem Still Not Fully Solved

Microsoft » Windows 8 : Vulnerability Statistics

Vulnerabilities (**38**) CVSS Scores Report Browse all versions Possible matches for this product Related Metasploit Modules

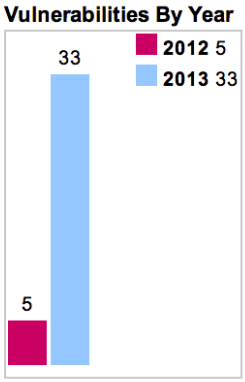
Related OVAL Definitions : Vulnerabilities (38) Patches (0) Inventory Definitions (2) Compliance Definitions (0)

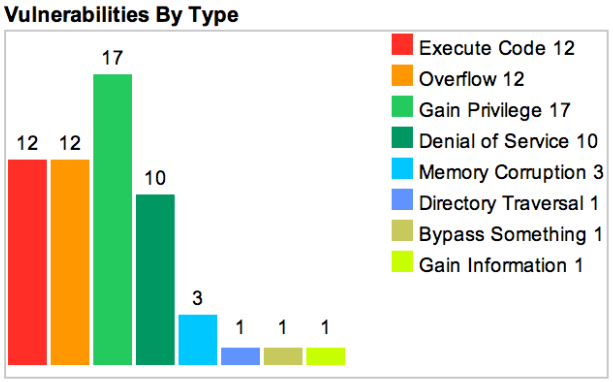
Vulnerability Feeds & Widgets

Vulnerability Trends Over Time

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2012	5		3	2								2			
2013	33	10	9	10	3			1		1	1	15			2
Total	38	10	12	12	3			1		1	1	17			2
% Of All		26.3	31.6	31.6	7.9	0.0	0.0	2.6	0.0	2.6	2.6	44.7	0.0	0.0	

Warning : Vulnerabilities with publish dates before 1999 are not included in this table and chart. (Because there are not many of them and they make the page look bad; and they may not be actually published in those years.)





http://www.cvedetails.com/product/22318/Microsoft-Windows-8.html