

Injection Attacks and Memory Safety

Nicholas Weaver

based on David Wagner's slides from Sp 2016

Administrivia

- You ***really really really*** want to attend discussion section this week
 - More so than normal: This lecture covers the high-level overview of the dark arts of buffer overflow exploitation
 - The discussions cover the low-level details you need: gdb and x86
- Midterms officially scheduled:
 - 9/28 and 10/26 from 8:30-10:00
 - Yes, the time sucks. Blame Econ 1...
- Homework and project (tentative) schedule will be online soon too...
- We are adding more seats in the class

What Properties Do We Want?

- Security is about maintaining one or more critical properties in the face of adversaries...
- For this, the property we want:
The computer **only** executes the program as written
 - No additional code runs, no control flow not present in the program
- Unfortunately for C/C++, this property does not exist in practice
 - Today: The dark arts of exploitation
 - Thursday: The (seemingly futile) DADA of prevention

Nick Hates Being Here... Cuz he's stuck in coach



 **Traveler Information**

Traveler 1 - Adults (age 18 to 64)

To comply with the [TSA Secure Flight program](#), the traveler information listed here must exactly match the information on the government-issued photo ID that the traveler presents at the airport.

Title (optional):	First Name:	Middle Name:	Last Name:
<input type="text" value="Dr."/>	<input type="text" value="Alice"/>	<input type="text"/>	<input type="text" value="Smith"/>

Gender:	Date of Birth:	Travelers are required to enter a middle name/initial if one is listed on their government-issued photo ID.
<input type="text" value="Female"/>	<input type="text" value="01/24/93"/>	

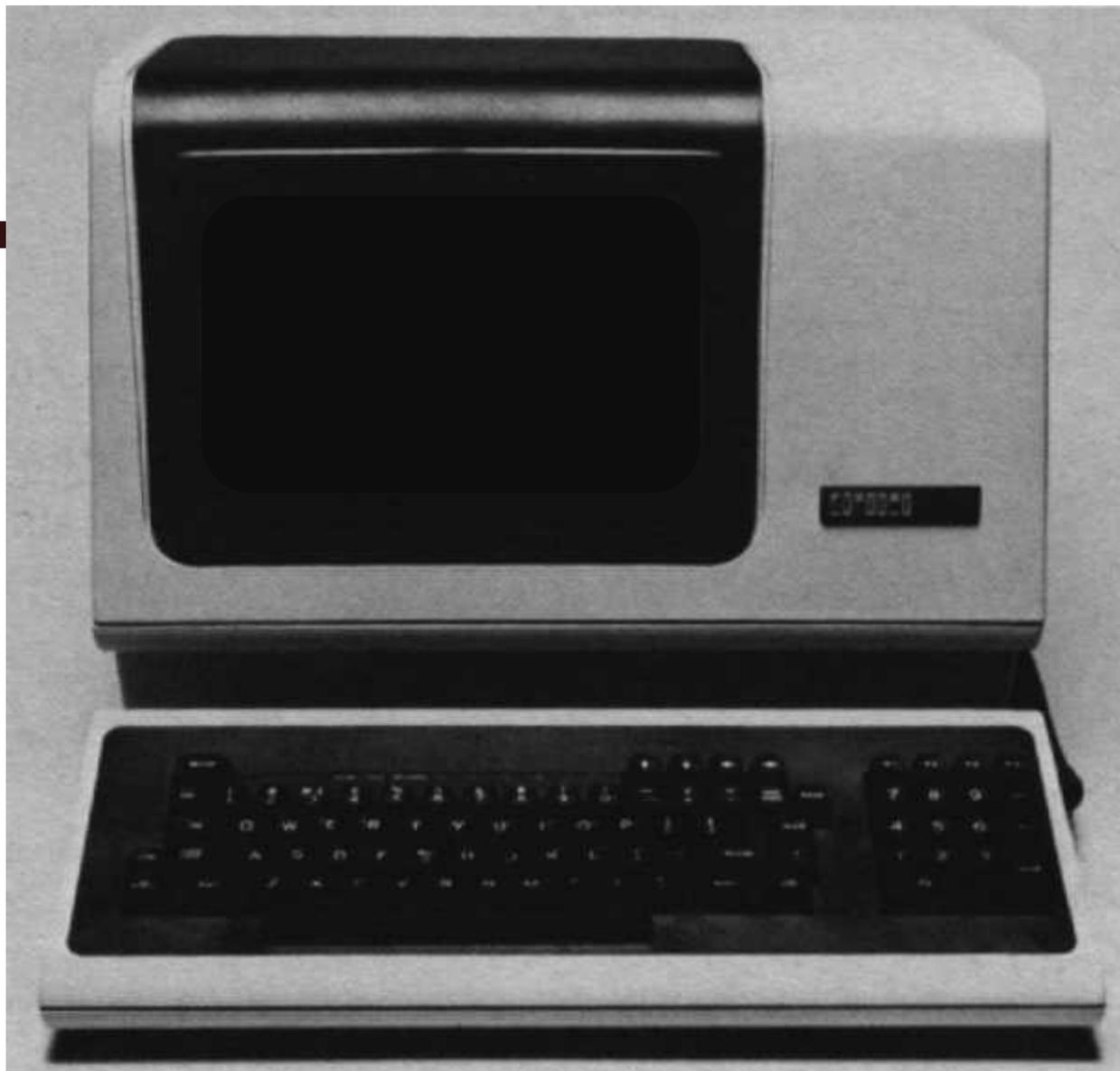
Some younger travelers are not required to present an ID when traveling within the U.S. [Learn more](#)

+ Known Traveler Number/Pass ID (optional): [?](#)

+ Redress Number (optional): [?](#)

Seat Request:

No Preference Aisle Window







Traveler Information

Traveler 1 - Adults (age 18 to 64)

To comply with the [TSA Secure Flight program](#), the traveler information listed here must exactly match the information on the government-issued photo ID that the traveler presents at the airport.

Title (optional): Dr. ▾	First Name: Alice	Middle Name:	Last Name: Smithhhhhhhhhhhhh
----------------------------	----------------------	--------------	---------------------------------

Gender: Female ▾	Date of Birth: 01/24/93
---------------------	----------------------------

Travelers are required to enter a middle name/initial if one is listed on their government-issued photo ID.

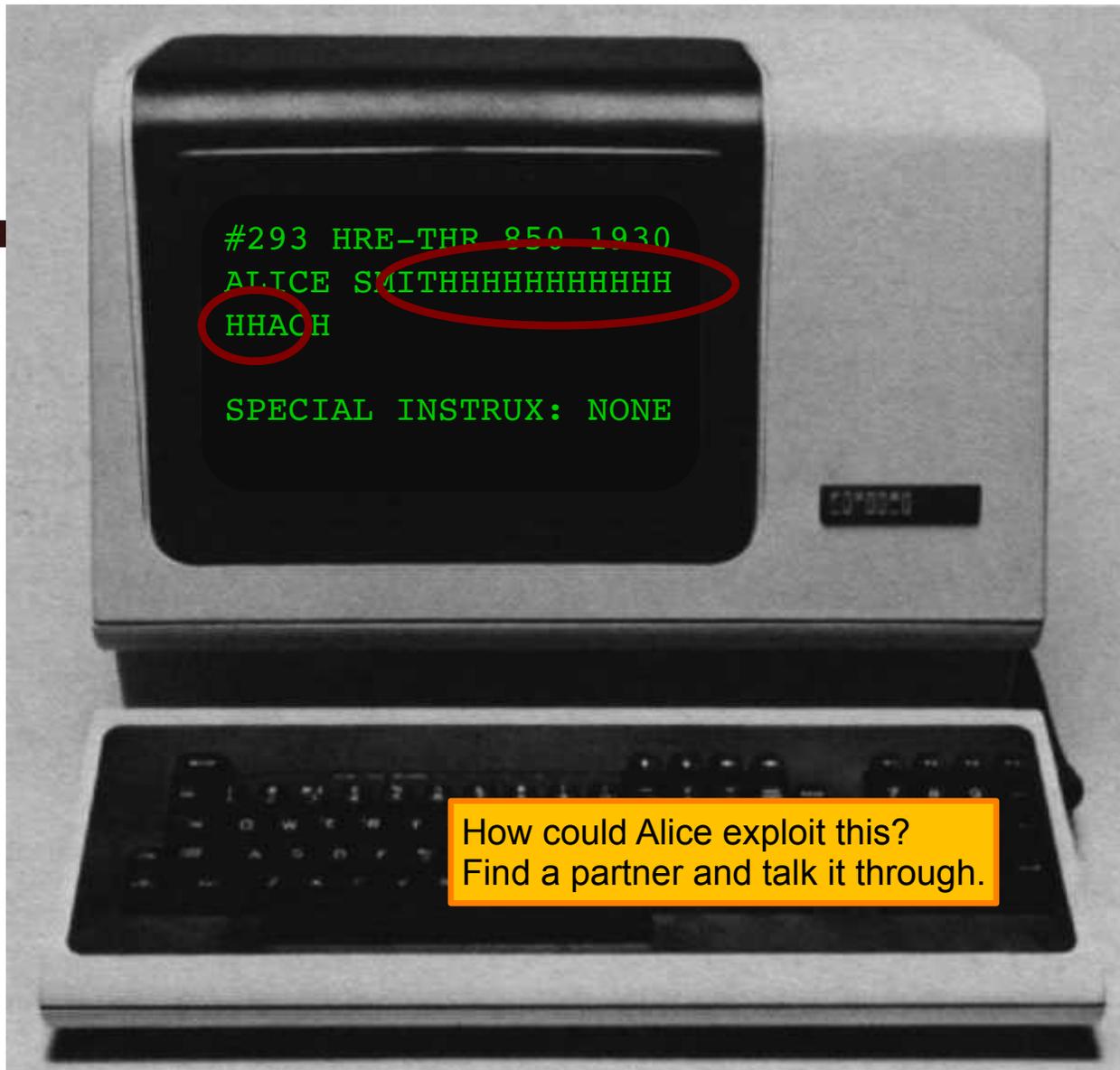
Some younger travelers are not required to present an ID when traveling within the U.S. [Learn more](#)

+ Known Traveler Number/Pass ID (optional): [?](#)

+ Redress Number (optional): [?](#)

Seat Request:

No Preference Aisle Window



How could Alice exploit this?
Find a partner and talk it through.



Traveler Information

Traveler 1 - Adults (age 18 to 64)

To comply with the [TSA Secure Flight program](#), the traveler information listed here must exactly match the information on the government-issued photo ID that the traveler presents at the airport.

Title (optional):	First Name:	Middle Name:	Last Name:
Dr. <input type="button" value="v"/>	Alice		Smith First

Gender:	Date of Birth:
Female <input type="button" value="v"/>	01/24/93

Travelers are required to enter a middle name/initial if one is listed on their government-issued photo ID.

Some younger travelers are not required to present an ID when traveling within the U.S. [Learn more](#)

Known Traveler Number/Pass ID (optional): [?](#)

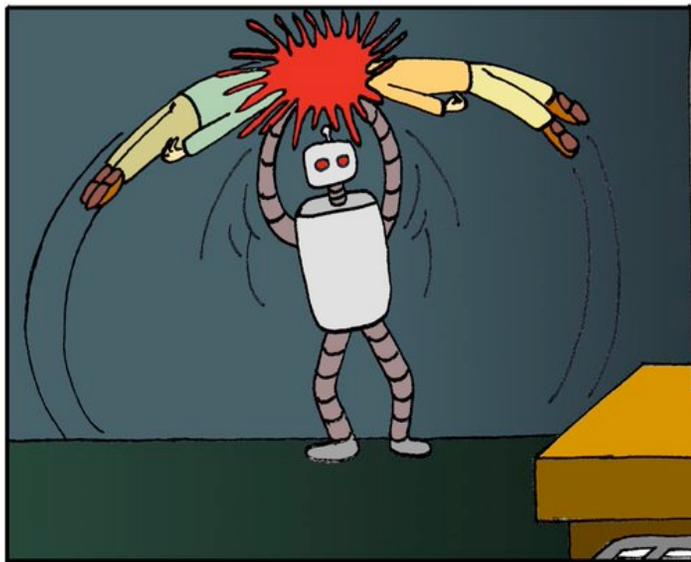
Redress Number (optional): [?](#)

Seat Request:

No Preference Aisle Window







```
static bool isCrazyMurderingRobot = false;  
  
void interact_with_humans (void){  
    if(isCrazyMurderingRobot = true)  
        kill(humans);  
    else  
        be_nice_to(humans);  
}
```

The Problem: Writing Off the End of an “Array”

```
char name[20];  
  
void vulnerable() {  
    ...  
    gets(name);  
    ...  
}
```

```
char name[20];  
char instrux[80] = "none";  
  
void vulnerable() {  
    ...  
    gets(name);  
    ...  
}
```

```
char line[512];
char command[] = "/usr/bin/
finger";

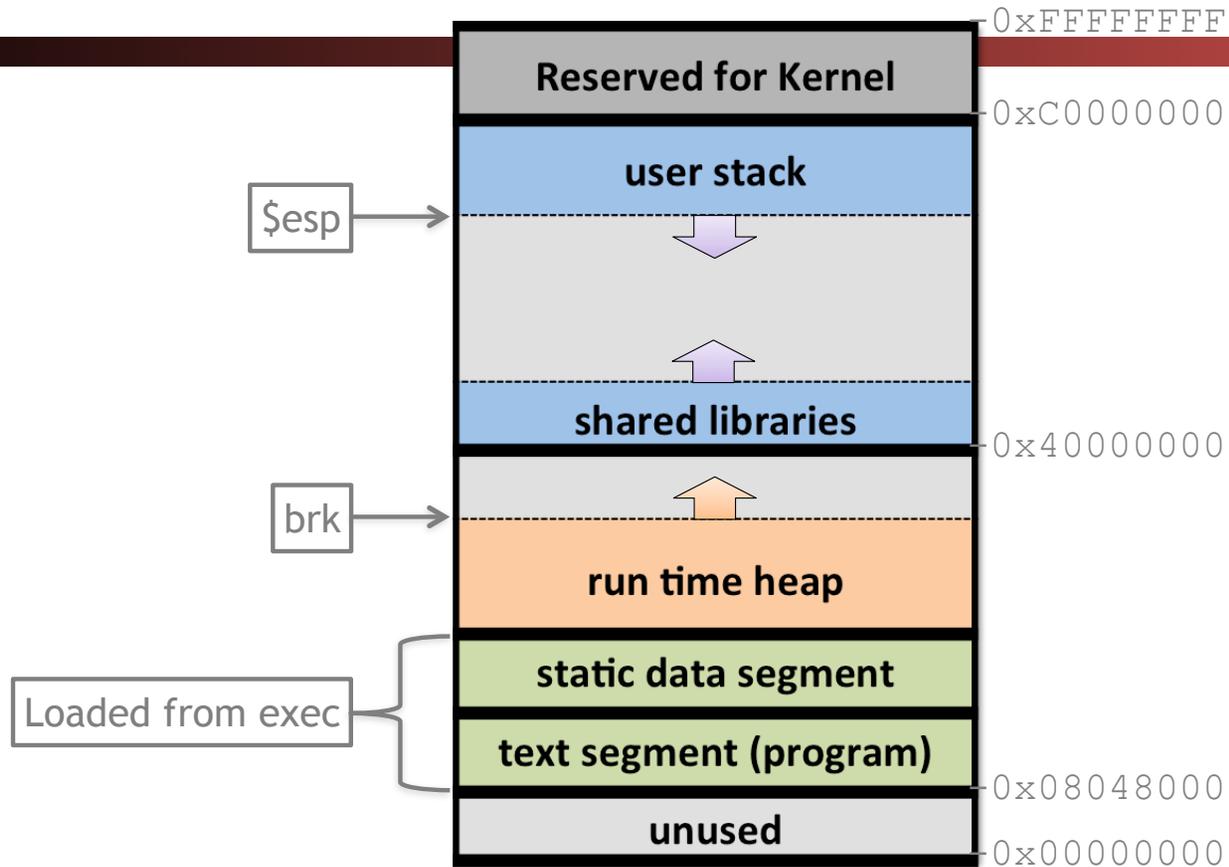
void main() {
    ...
    gets(line);
    ...
    execv(command, ...);
}
```

```
char name[20];  
int  seatinfirstclass = 0;  
  
void vulnerable() {  
    ...  
    gets(name);  
    ...  
}
```

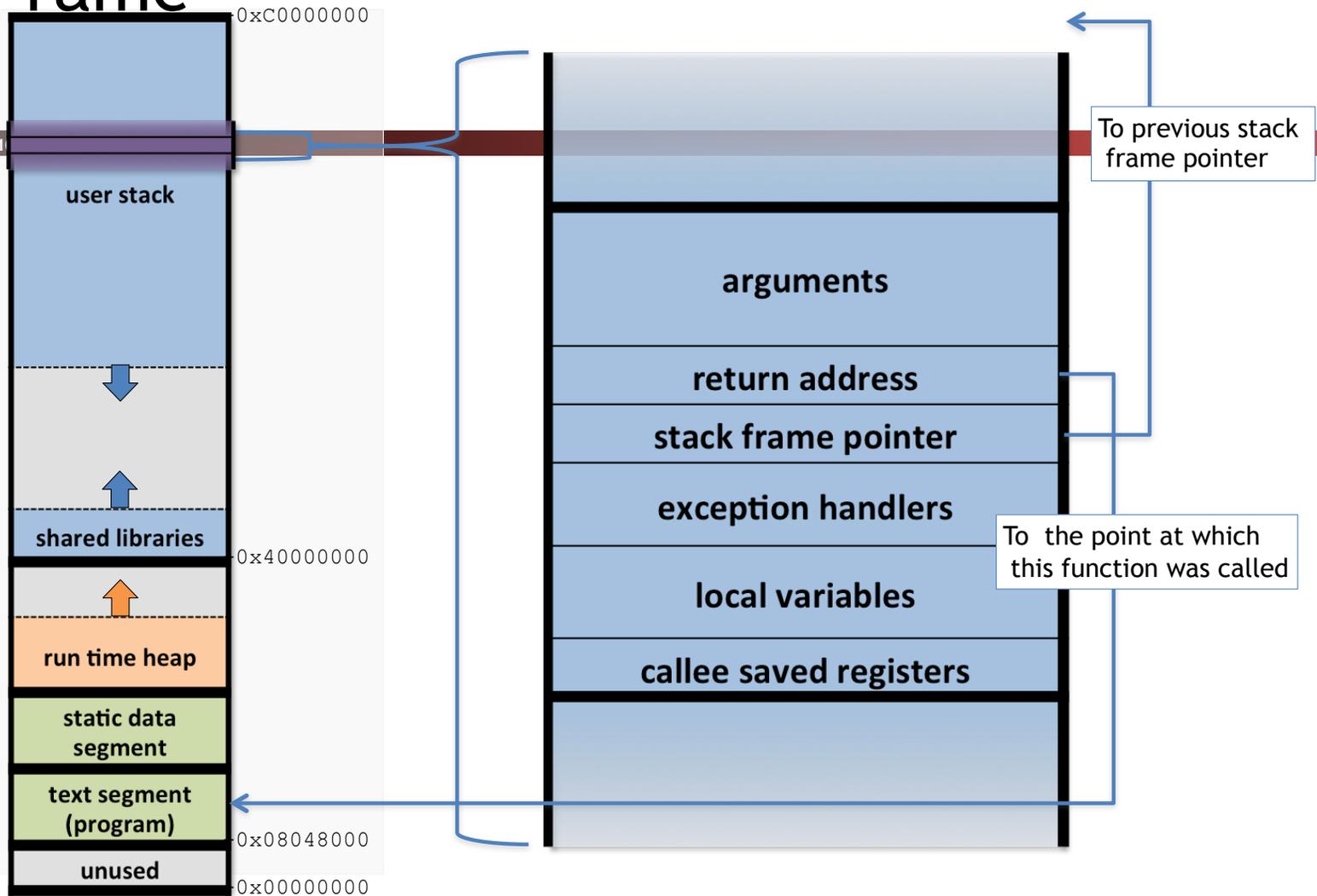
```
char name[20];  
int  authenticated = 0;  
  
void vulnerable() {  
    ...  
    gets(name);  
    ...  
}
```

```
char name[20];  
int (*fnptr)();  
  
void vulnerable() {  
    ...  
    gets(name);  
    ...  
}
```

Linux (32-bit) process memory layout

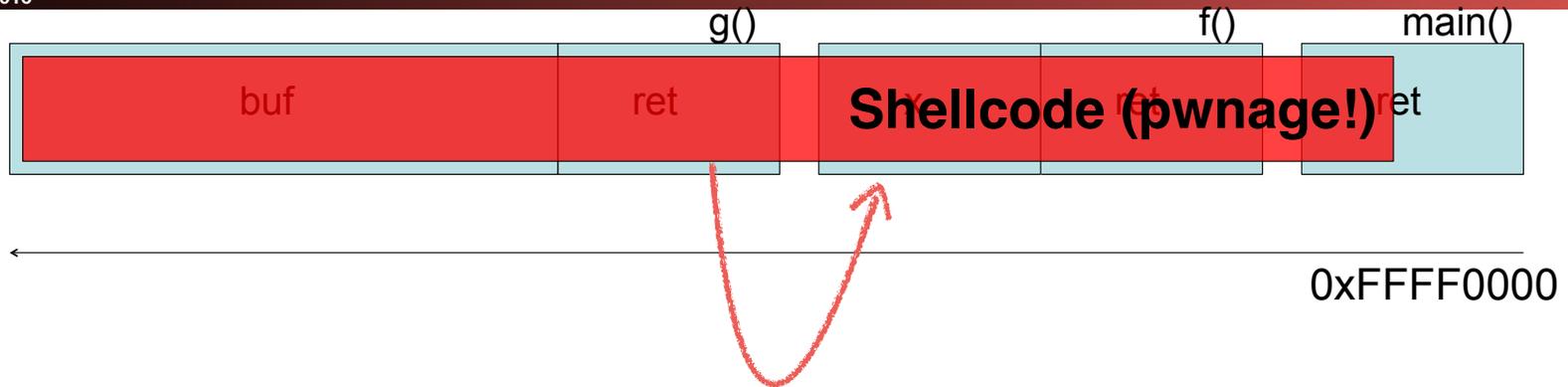


Stack Frame



Code Injection

Code Injection in Action



```
main() {  
    f();  
}
```

```
f() {  
    int x;  
    g();  
}
```

```
g() {  
    char  
    buf[80];  
    gets(buf);  
}
```

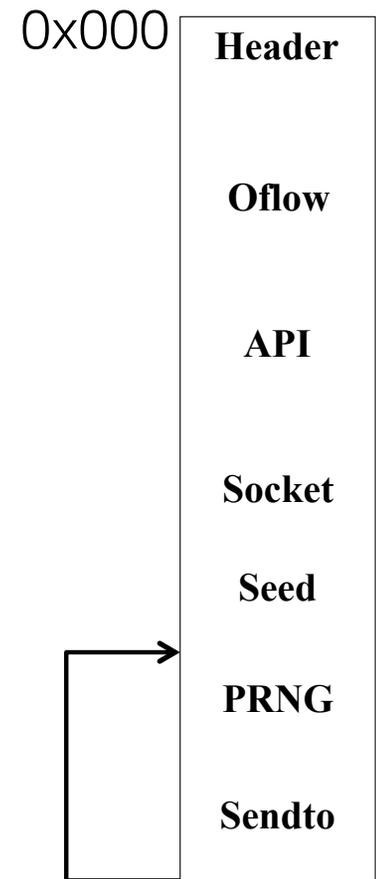
Basic Stack Exploit

- Overwriting the return address allows an attacker to redirect the flow of program control.
- Instead of crashing, this can allow *arbitrary* code to be executed.
- Example: attacker chooses malicious code he wants executed (“shellcode”), compiles to bytes, includes this in the input to the program so it will get stored in memory somewhere, then overwrites return address to point to it.
- Often prepending a "Noop sled": a series of do-nothing instructions to allow the pointer into the shellcode to be imprecise

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

The (2nd) Most Spectacular Buffer Overflow Attack of All Time: Slammer

- Microsoft SQL server received requests on UDP port 1434
 - UDP has no flow, you can just send requests...
- Someone sent a particular buffer overflow attack to a single MSSql server on the Internet...
 - Cleanup from buffer overflow
 - Get API pointers and pointer to self
 - Create socket & packet
 - Seed PRNG with getTickCount()
 - While 1
 - Increment PRNG
 - Send packet to PRNG address as fast as possible
- 404 bytes total
- Worldwide Spread in 10 minutes



The Most Spectacular: Witty...

- ISS security software had a vulnerability like Slammer
 - Simple stack overflow when it receives a magic UDP packet
 - It was fixed on March 17th, 2004...
- But on March 19th, 2004, someone released a worm
 - Just like the Slammer worm, but with a twist...
 - Repeat forever:
 - Send 20,000 attacks...
 - Select a random disk
 - Select a random block on disk
 - Erase that block
- Oh, and much of the initial infection occurred in the US Army!

Oh, and you can still do it Today...

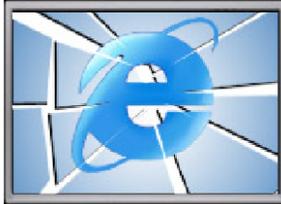
- The NSA screwed up and, two weeks ago, someone released a bunch of NSA tools for attacking firewalls...
- Including several exploits
- **EXTRABACON**, targeting Cisco ASA firewalls
- A single packet, UDP stack overflow in the SNMP code
- You could make "Slammer 3" targeting this vulnerability



```
void vulnerable() {  
    char buf[64];  
    ...  
    gets(buf);  
    ...  
}
```

```
void still_vulnerable() {  
    char buf = malloc(64);  
    ...  
    gets(buf);  
    ...  
}
```

IE's Role in the Google-China War



By Richard Adhikari
TechNewsWorld
01/15/10 12:25 PM PT

The hack attack on Google that set off the company's ongoing standoff with China appears to have come through a zero-day flaw in Microsoft's Internet Explorer browser. Microsoft has released a security advisory, and researchers are hard at work studying the exploit. The attack appears to consist of several files, each a different piece of malware.

Computer security companies are scurrying to cope with the fallout from the Internet Explorer (IE) flaw that led to cyberattacks on [Google](#) (Nasdaq: GOOG) and its corporate and individual customers.

The zero-day attack that exploited IE is part of a lethal cocktail of malware that is keeping researchers very busy.

"We're discovering things on an up-to-the-minute basis, and we've seen about a dozen files dropped on infected PCs so far," Dmitri Alperovitch, vice president of research at [McAfee Labs](#), told TechNewsWorld.

The attacks on Google, which appeared to originate in China, have sparked a feud between the Internet giant and the nation's government over censorship, and it could result in Google pulling away from its business dealings in the country.

Pointing to the Flaw

The vulnerability in IE is an invalid pointer reference, [Microsoft](#) (Nasdaq: MSFT) said in [security advisory 979352](#), which it issued on Thursday. Under certain conditions, the invalid pointer can be accessed after an object is deleted, the advisory states. In specially crafted attacks, like the ones launched against Google and its customers, IE can allow remote execution of code when the flaw is exploited.

```
void safe() {  
    char buf[64];  
    ...  
    fgets(buf, 64, stdin);  
    ...  
}
```

```
void safer() {  
    char buf[64];  
    ...  
    fgets(buf, sizeof(buf), stdin);  
    ...  
}
```

```
void vulnerable(int len, char
*data) {
    char buf[64];
    if (len > 64)
        return;
    memcpy(buf, data, len);
}
```

```
memcpy(void *s1, const void *s2, size_t n);
```

```
void safe(size_t len, char *data) {  
    char buf[64];  
    if (len > 64)  
        return;  
    memcpy(buf, data, len);  
}
```

```
void f(size_t len, char *data) {
    char *buf = malloc(len+2);
    if (buf == NULL) return;
    memcpy(buf, data, len);
    buf[len] = '\n';
    buf[len+1] = '\0';
}
```

Is it safe? Talk to your partner.

Vulnerable!

If `len = 0xffffffff`, *allocates only 1 byte*

Broward Vote-Counting Blunder Changes Amendment Result

POSTED: 1:34 pm EST November 4, 2004

BROWARD COUNTY, Fla. -- The Broward County Elections Department has egg on its face today after a computer glitch misreported a key amendment race, according to WPLG-TV in Miami.

Amendment 4, which would allow Miami-Dade and Broward counties to hold a future election to decide if slot machines should be allowed at racetracks, was thought to be tied. But now that a computer glitch for machines counting absentee ballots has been exposed, it turns out the amendment passed.

"The software is not geared to count more than 32,000 votes in a precinct. So what happens when it gets to 32,000 is the software starts counting backward," said Broward County Mayor Ilene Lieberman.

That means that Amendment 4 passed in Broward County by more than 240,000 votes rather than the 166,000-vote margin reported Wednesday night. That increase changes the overall statewide results in what had been a neck-and-neck race, one for which recounts had been going on today. But with news of Broward's error, it's clear amendment 4 passed.



Broward County Mayor Ilene Lieberman says voting counting error is an "embarrassing mistake."

```
void vulnerable(int len, char
*data) {
    char *buf;
    if (len < 0) len = -len;
    buf = malloc(len + 1);
    memcpy(buf, data, len);
}
```

```
void vulnerable() {  
    char buf[64];  
    if (fgets(buf, 64, stdin) == NULL)  
        return;  
    printf(buf);  
}
```

Fun With Printf...

```
printf("100% dude!");
```

⇒ *prints value 4 bytes above retaddr as integer*

```
printf("100% sir!");
```

⇒ *prints bytes pointed to by that stack entry
up through first NUL*

```
printf("%d %d %d %d ...");
```

⇒ *prints series of stack entries as integers*

```
printf("%d %s");
```

⇒ *prints value 4 bytes above retaddr plus bytes
pointed to by preceding stack entry*

```
printf("100  
% nuke'm!");
```

⇒ ***writes** the value 3 to address pointed
to by stack entry*

Of course strcpy is bad too

```
void iHateC(char *s){  
    char p[80];  
    ...  
    strcpy(s,p);  
    ...  
}
```

Oh, but you think strncpy is *safe*?

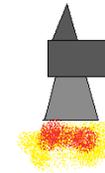
- Insert diabolical laughter here....

```
void iHateC(char *s){
    char p[80];
    ...
    strncpy(s,p,size_of(p));
    ...
    printf("%s", p);
}
```

The Defensive Arms Race...

- The real solution to these problems:
DO NOT USE C OR C++ FOR ANYTHING WHICH MIGHT COME FROM THE OUTSIDE WORLD!!!
- Use Java, Python, Rust, Go, etc etc etc...
- But instead programmers seem to want to keep using C?!?
- Now I know I said last time don't go blindly blaming the users, but I think we can make an exception in this case...
- "Teaching people 'secure' programming in C or C++ is like teaching driver safety in 1920s race cars"

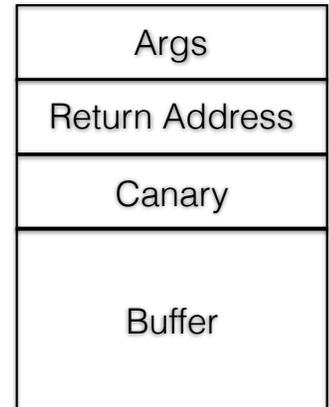
CISCO L.A.R.T
LUSER ATTITUDE
READJUSTMENT TOOL V3



The Lvl. 3 L.A.R.T is an iron bar with a spike on one end and a handle on the other. The L.A.R.T can be swung for heavy piercing OR blunt damage. If that is not sufficient, an ionic thruster can be activated for a cost of 10, 50, 100, or 500 units of power. Each level of power deals +2 piercing damage and +1 AP.

Idea #1: Stack Canaries/Stack Cookies...

- Insert a constant random value just before the return address in each stack frame
 - Before returning, check that the canary with a stored value
- Unfortunately many ways to bypass
 - In Windows could trigger the exception handler instead
 - Can guess the canary
 - And use tricks to reduce the space
 - Vulnerabilities that allow reading memory to to read the cookie
 - Or a vulnerability that allows overwriting the stored copy!



Idea #2:

Non-Executable Stack Space...

- Why should the stack contain executable code?
 - If we remember from 61c, the TLB can specify that a memory address can contain code or just memory
- So why allow code on the stack at all?
 - Apart from some compilers wanting to do that with “trampolines” or “thunks”
- All fine and good, but...
 - Load executable code into the heap
 - Often with a "Heap spray": Seed lots of copies with a large 'noop sled' so you can just get lucky
 - Jump to *that*

Idea #3:

Data Execution Protection/ W^X

- Lets just make ALL the memory either executable or writeable
 - So now you can't write code and run it...
- Solution: Jump into Libc or Return Oriented Programming
 - There is so much other code available
- Either just execute the function you want:
 - Set up the stack to run `exec("create-backdoor")`
 - Set up a series of return statements to execute "gadgets" in the code
 - This is deep, hard to understand voodoo which we won't go into detail but...
 - There are tools to do this for you automatically: ROPgadget
 - Of course its Open Source too!
<https://github.com/JonathanSalwan/ROPgadget/tree/master>

Idea #4:

Lets make that hard to do

- Address Space Layout Randomization...
 - Instead of having all the text segments in a constant arrangement, lets randomize it at runtime...
- OK without W^X , much more powerful with W^X
 - Since bypassing W^X requires only executing existing code, which requires knowing the address of existing codes, but ASLR randomizes where the existing code is...
- Good idea but...
 - Some libraries can't be randomized so if a program uses those ASLR doesn't actually work!
 - Information leakage attacks:
If you can get the address of a single function in a library,
you've defeated ASLR and can just generate your string of ROP gadgets at runtime
 - iPhone "Trident" exploit chain: One vulnerability was an information leakage to enable ASLR bypass

Idea #5:

Write “Secure” code...

- Well, we will get to that next time, but...
 - I’ll argue you can make your C better but you can’t make it secure
- So I still vote for idea #6
 - “If you want to use C or C++ in a new project I will go and whack you upside the head with a 2x4”

Real World Exploitation

- It gets even more complicated once you hit the real world:
 - <http://documents.trendmicro.com/assets/pdf/shell-on-earth.pdf>
 - Evaluates all the 2016 pwn2own exploits
- Need to often chain multiple vulnerabilities together
 - Take over browser, then escape a sandbox, then escalate to owning the entire system
 - It takes 2-3 unpatched exploits to take over a target's iPhone
- Common vulnerability is actually “use-after-free”
 - Often used to enable writing a value somewhere in memory:
Use to overwrite a stored function pointer pointing to your chain of ROP gadgets
 - But again, this is strictly caused by C lack of safety

And if I get to it: Meme of the lecture...

- True: Its called “Machine Learning”

