

# Overflows, Injection, & Memory Safety



<https://inst.eecs.berkeley.edu/~cs161/su19/feedback>

# Announcements...

- HW0 due tonight!
- Find project partners
  - Solo or with a partner, it is up to you
- Discussion size  $101 > 102 > 103$

# THE INTERNET of

# THINGS

# SHIT...

# OR NET OF A MILLION SPIES

# Internet of Shit...

- A device produced by the lowest bidder...
  - That you then connect through the network
- This has a very wide ***attack surface***
  - Methods where an attacker might access a vulnerability
- And its often incredibly ***cost sensitive***
  - Very little support after purchase
    - So things don't get patched
  - No way for the user to tell what is "secure" or "not"
    - But they can tell what is cheaper!
    - And often it is ***insanely*** insecure:  
Default passwords on telnet of admin/admin...  
Trivial buffer overflows

# Net Of A Million Spies...

- Device only communicates through a central service
  - Greatly reduces the attack surface but...
- Most of the companies running the service are "Data Asset" companies
  - Make their money from advertising, not the product themselves
    - May actually subsidize the product considerably
  - Some you know about: Google, Amazon
  - Some you may not: Salesforce







## 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:

Smith

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









## 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="Smithhhhhhhhhhhhhhh"/>

Gender:

Date of Birth:

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

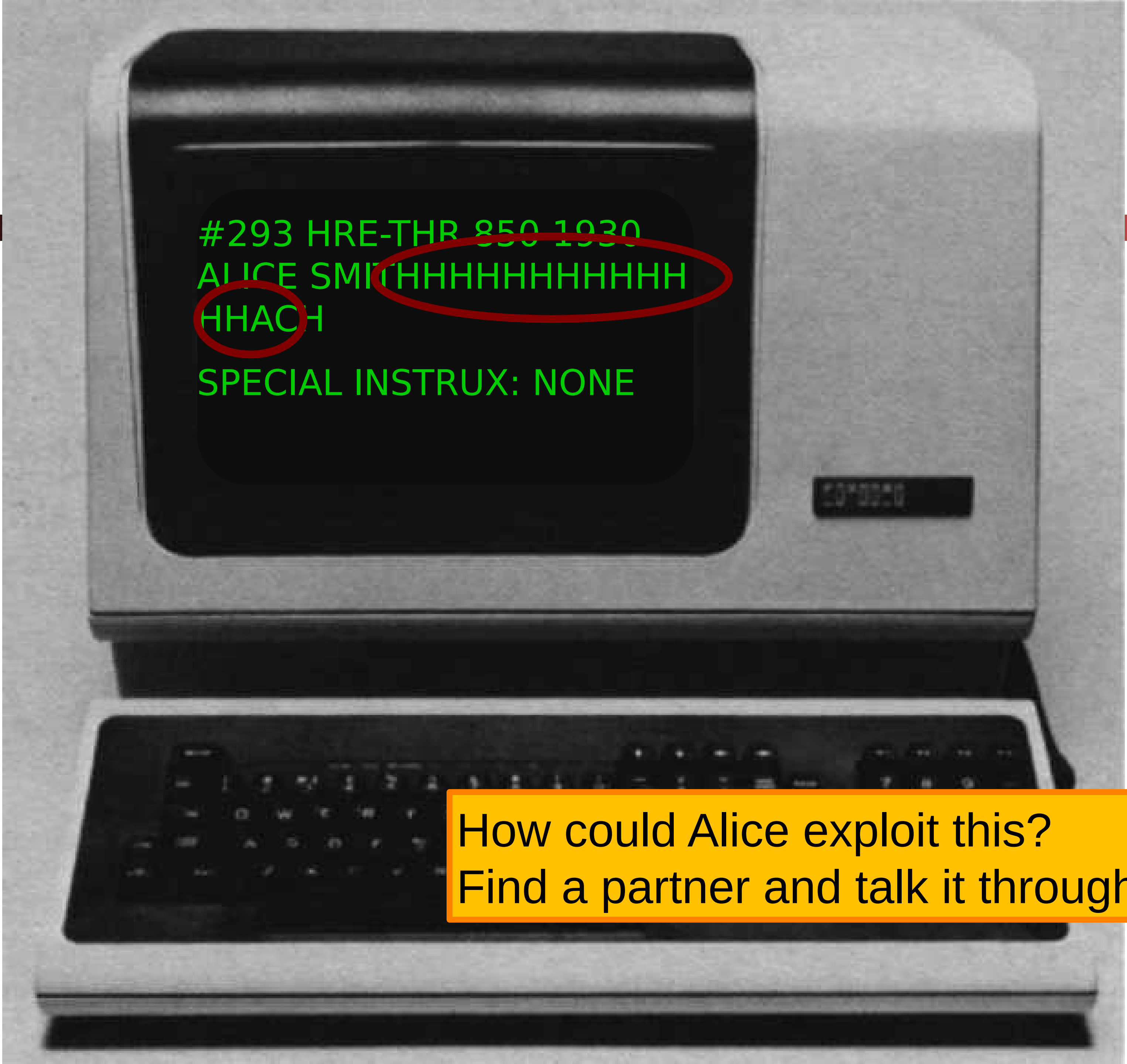
Some younger travelers are not required to present an ID when traveling within the U.S. [Learn more](#)

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

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

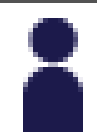
Seat Request:

☒ No Preference ☐ Aisle ☐ Window



```
#293 HRE-THR 850 1930  
ALICE SMITHHHHHHHHHHHHHHHHHHHHH  
HHACH  
SPECIAL INSTRUX: NONE
```

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.	Alice		Smith First

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





#293 HRE-THR 850 1930  
ALICE SMITH  
FIRST

SPECIAL INSTRUX: TREAT  
AS HUMAN.

*Passenger last name:*

"Smith

First

Special Instrux: Treat As Human."

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

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

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

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



```
char line[512];  
char command[] =  
"/usr/bin/finger";  
  
void main() {  
    ...  
    gets(line);  
    ...  
    execv(command, ...);  
}
```

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

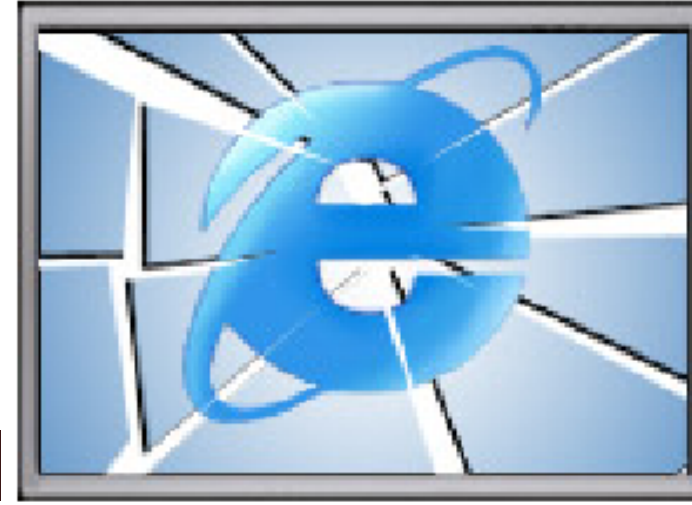
Rank	Score	ID	Name
[1]	93.8	<a href="#">CWE-89</a>	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	<a href="#">CWE-78</a>	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	<a href="#">CWE-120</a>	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	<a href="#">CWE-79</a>	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	<a href="#">CWE-306</a>	Missing Authentication for Critical Function
[6]	76.8	<a href="#">CWE-862</a>	Missing Authorization
[7]	75.0	<a href="#">CWE-798</a>	Use of Hard-coded Credentials
[8]	75.0	<a href="#">CWE-311</a>	Missing Encryption of Sensitive Data
[9]	74.0	<a href="#">CWE-434</a>	Unrestricted Upload of File with Dangerous Type
[10]	73.8	<a href="#">CWE-807</a>	Reliance on Untrusted Inputs in a Security Decision
[11]	73.1	<a href="#">CWE-250</a>	Execution with Unnecessary Privileges
[12]	70.1	<a href="#">CWE-352</a>	Cross-Site Request Forgery (CSRF)
[13]	69.3	<a href="#">CWE-22</a>	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[14]	68.5	<a href="#">CWE-494</a>	Download of Code Without Integrity Check
[15]	67.8	<a href="#">CWE-863</a>	Incorrect Authorization
[16]	66.0	<a href="#">CWE-829</a>	Inclusion of Functionality from Untrusted Control Sphere

```
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 Science 161 Summer 2019

Dutra and Jawale

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.

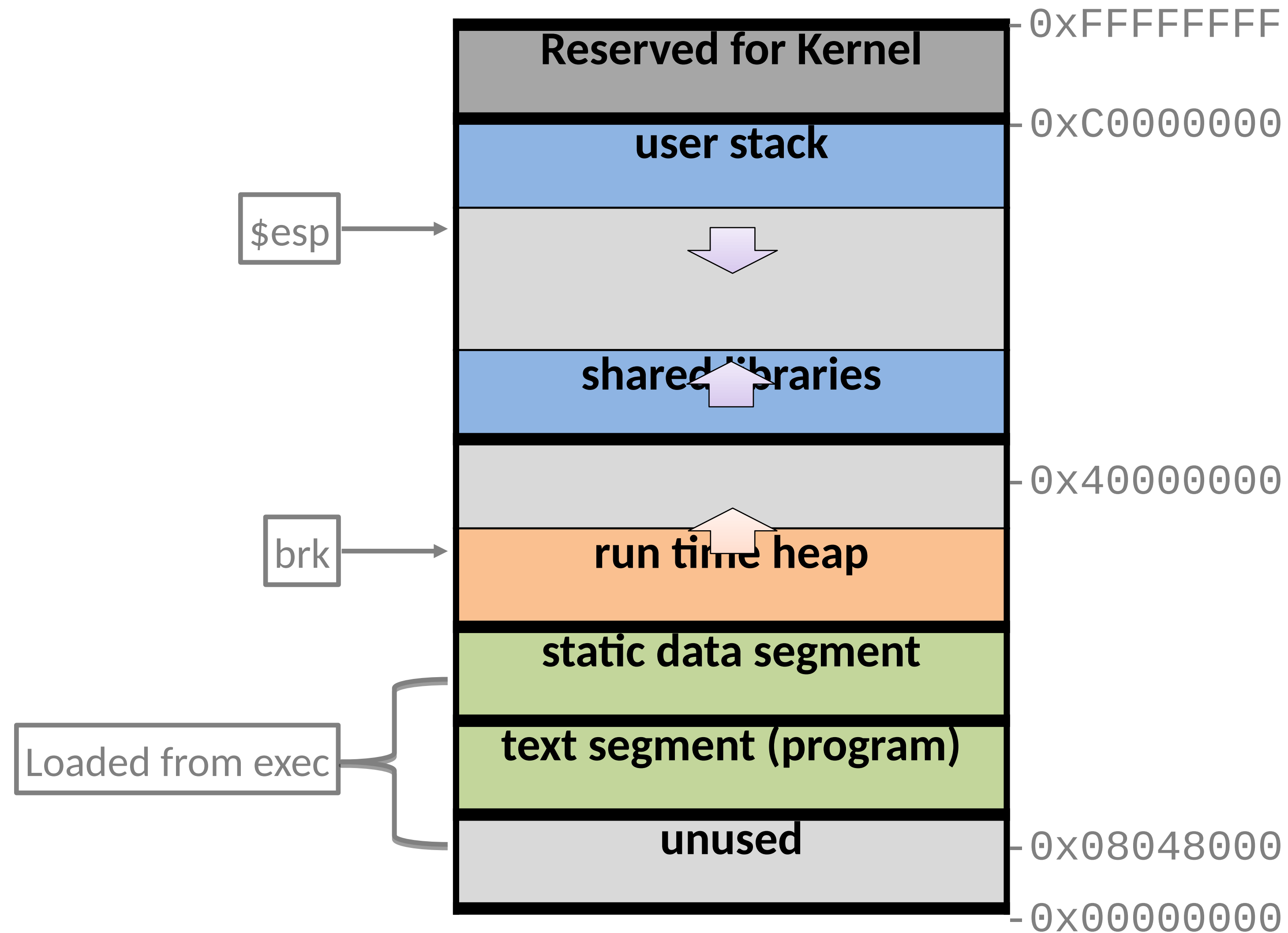
"We're discovering things on an up-to-the-minute basis, and we've seen about a dozen files dropped on infected PCs so far," Dmitri Alperovitch, vice president of research at [McAfee](#) Labs, told TechNewsWorld.

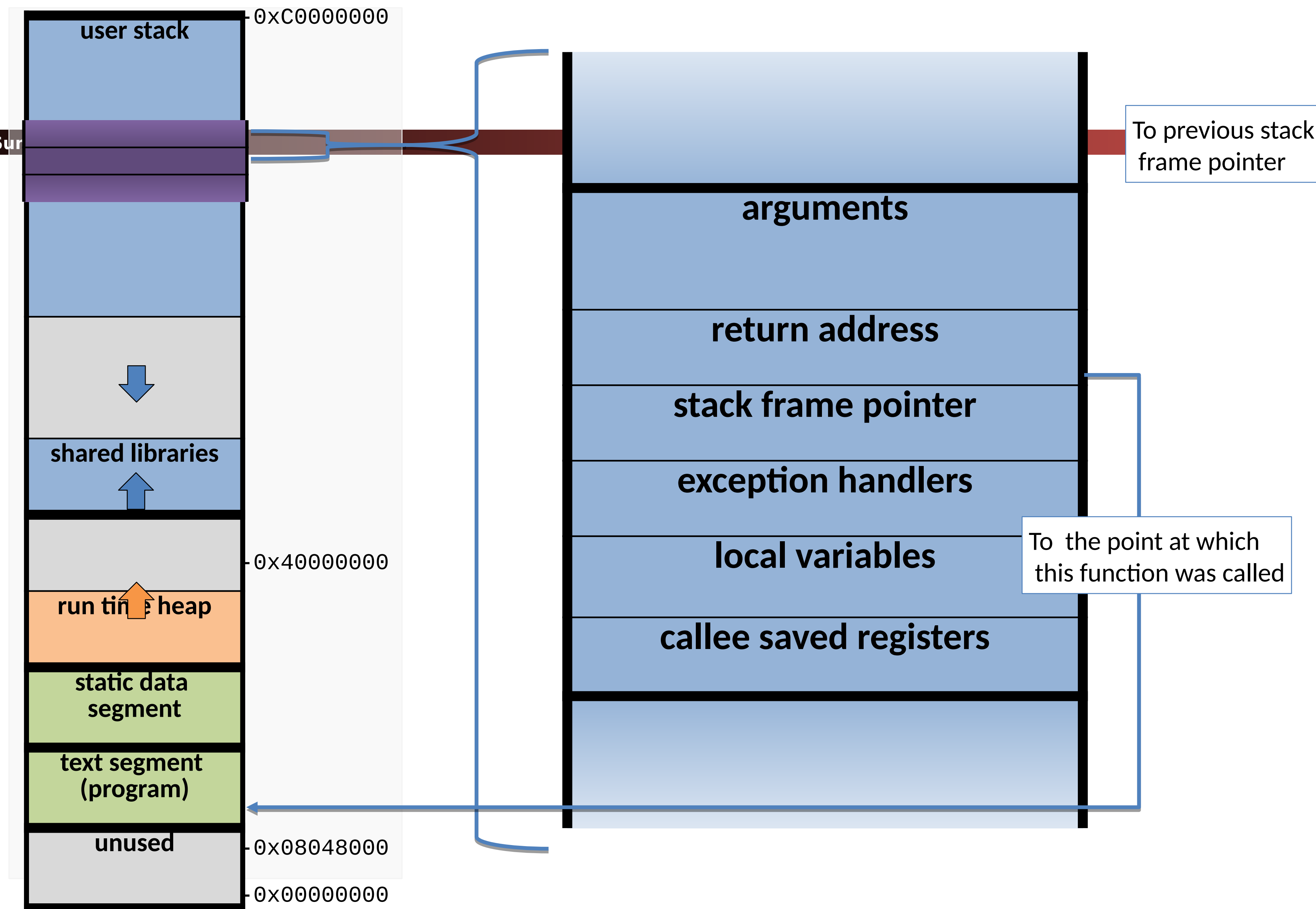
The attacks on Google, which appeared to originate in China, have sparked a feud between the Internet giant and the nation's government over censorship, and it could result in Google pulling away from its business dealings in the country.

## 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.

# Linux (32-bit) process memory layout





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

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



Assume these are both under the control of an attacker.

```
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);
```

size\_t is *unsigned*:  
What happens if len == -1?

```
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

Computer Science 161

POSTED: 1:34 pm EST November 4, 2004

Dutra and Jawale

**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.



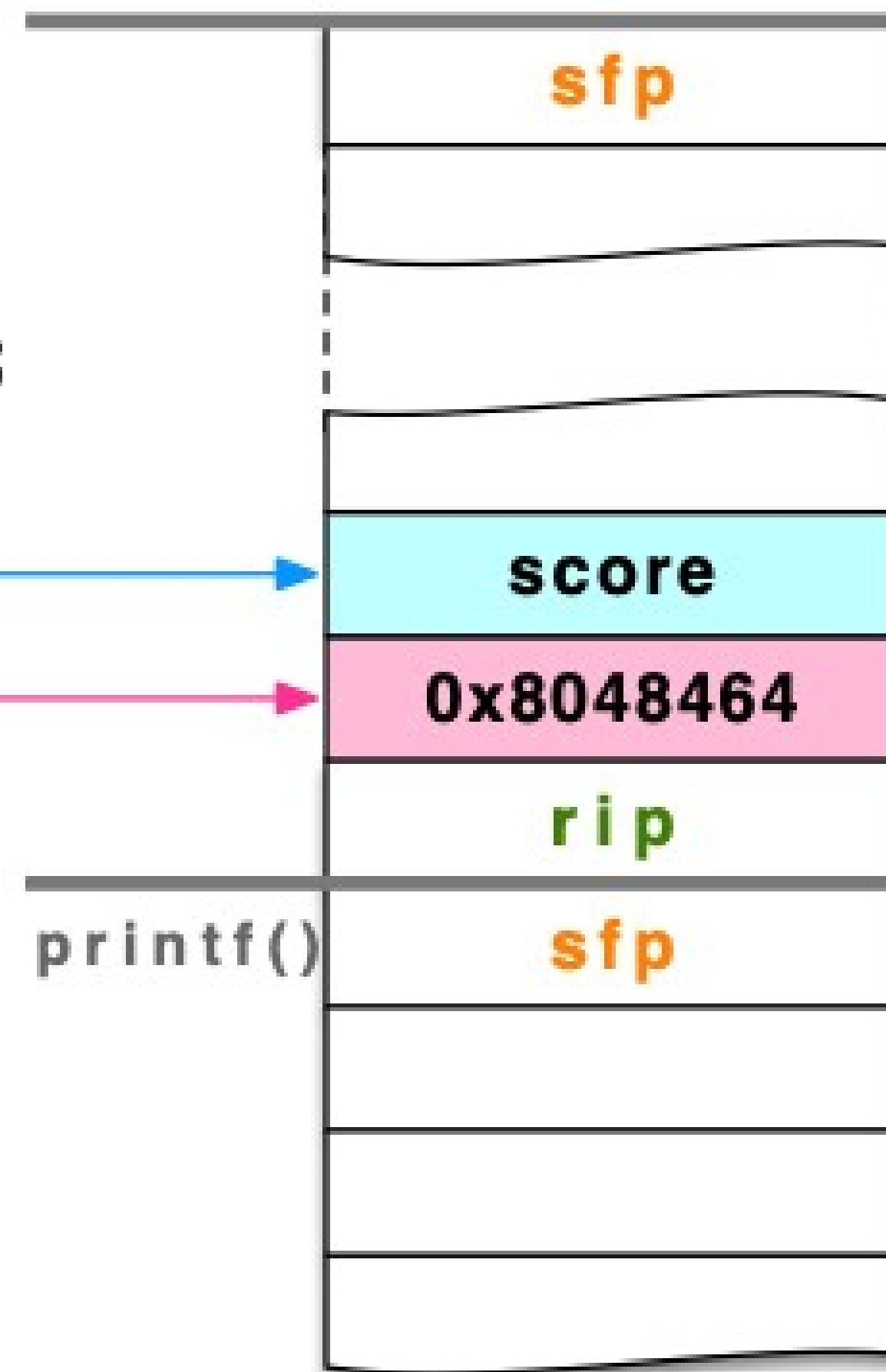
Broward County Mayor Ilene Lieberman says voting counting error is an "embarrassing mistake."

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

```
printf("you scored %d\n", score);
```



`printf("you scored %d\n", score);`



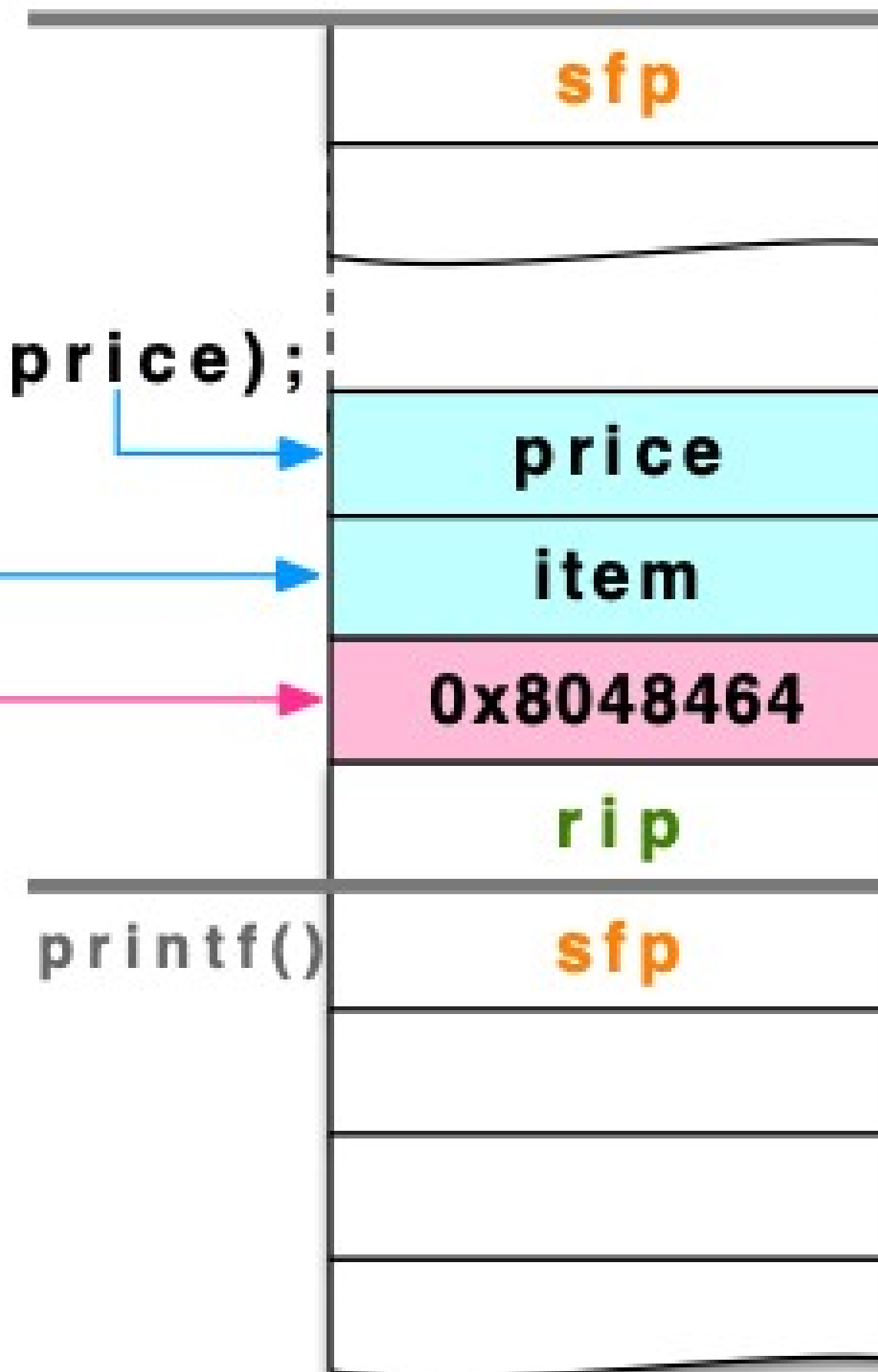
	\0	\n	d
%		d	e
r	o	c	s
	u	o	y

`0x8048464`

```
printf("a %s costs $%d\n", item,  
price);
```



```
printf("a %s costs $%d\n", item, price);
```



\0	\n	d	%
\$		s	t
s	o	c	
s	%		a

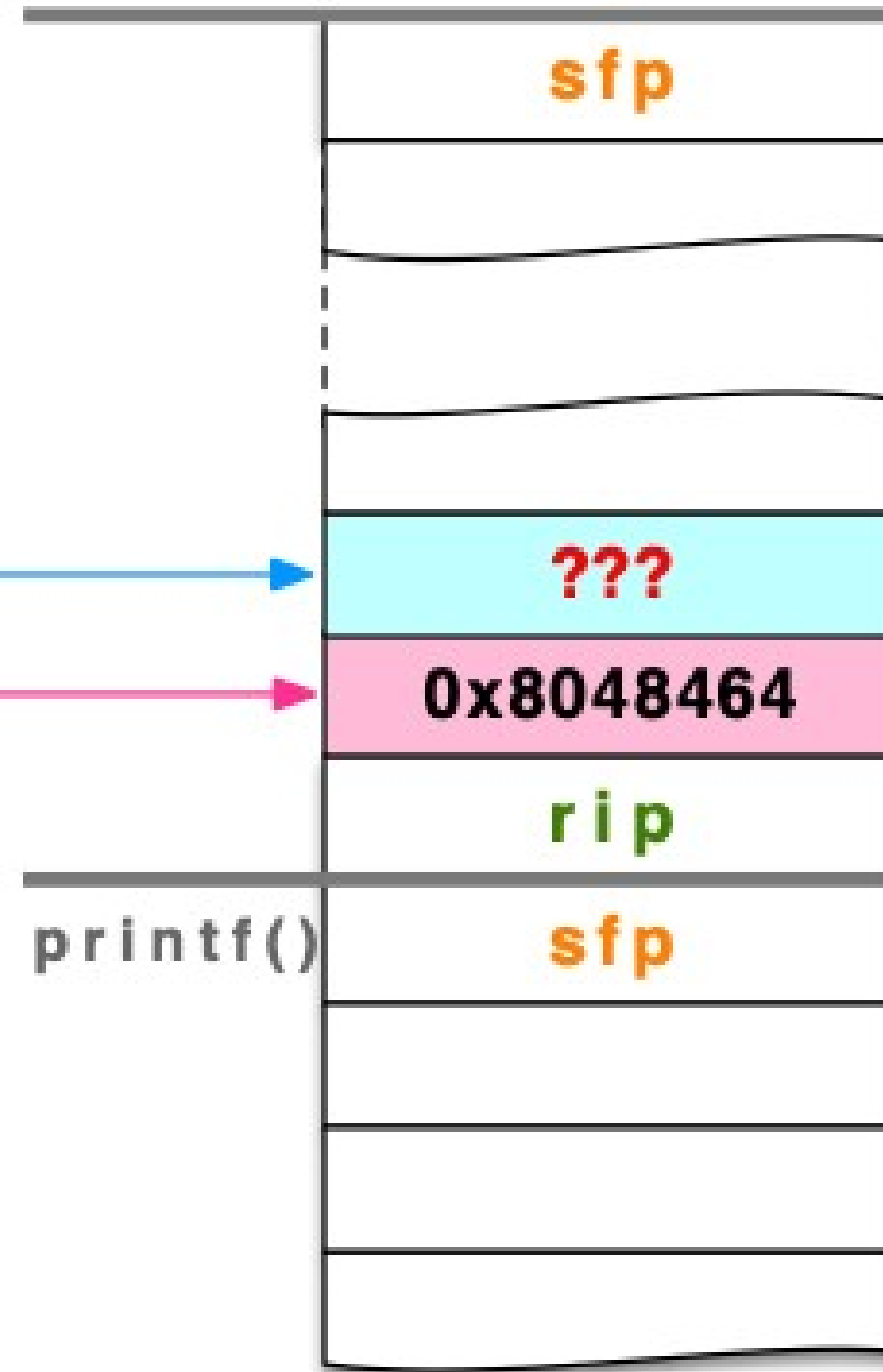
0x8048464

# Fun With `printf` format strings...

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

Format argument is missing!

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



	\0	!	e
d	u	d	
%	0	0	1

`0x8048464`

# More Fun With `printf` format strings...

```
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!");
```

What does the `%n` format do??

`%n` *writes* the number of characters printed so far into the corresponding format argument.

```
int report_cost(int item_num, int price) {  
    int colon_offset;  
    printf("item %d:%n $%d\n", item_num,  
          &colon_offset, price);  
    return colon_offset;  
}
```

`report_cost(3, 22)` prints "item 3: \$22"  
and returns the value 7

`report_cost(987, 5)` prints "item 987: \$5"  
and returns the value 9

**%hn** is the same, but writes a **short** instead of **int**

*%n* writes the number of characters printed so far into the corresponding format argument.

```
int report_cost(int item_num, int price) {  
    short int colon_offset;  
    printf("item %d:%hn $%d\n", item_num,  
          &colon_offset, price);  
    return colon_offset;  
}
```

report\_cost(3, 22) prints "item 3: \$22"  
and returns the value 7

report\_cost(987, 5) prints "item 987: \$5"  
and returns the value 9

# Fun With `printf` format strings...

```
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 the address pointed to by stack entry***

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



# And Now: Lets Walk Through A Stack Overflow

- Idea: We override a buffer on the stack...
  - In the buffer we place some code of our choosing
    - "Shellcode"
  - Override the return address to point to code of our choosing
- Lets step through the process on an x86...