Lecture 3: Buffer Overflows

https://cs161.org

Announcements

- Discussion today, tomorrow, Wednesday. Go to any one that isn't full. Please respond to poll on Piazza.
- Expect Homework 1 to be released tonight. Check Piazza.

Security Principles

More security principles

- Use fail-safe defaults
- Consider human factors
- Only as secure as the weakest link
- Don’t rely on security through obscurity
- Trusted path


Leitura de código de barras


Bradesco
Tarifas de
Serviços Bancários
Multi Expresso

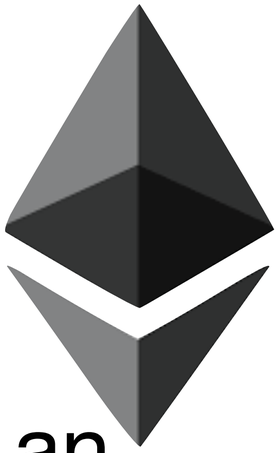
Time of Check to Time of Use Vulnerability: Race Condition


```
procedure withdraw(w)
   // contact central server to get balance
   1. let b := balance

   2. if b < w, abort

   // contact server to set balance
   3. set balance := b - w

   4. dispense $w to user
```

Suppose that *here* an attacker arranges to suspend first call, and calls withdraw again **concurrently**

TOCTTOU = Time of Check To Time of Use

A Hundred Million Dollar TOCTTOU Bug...

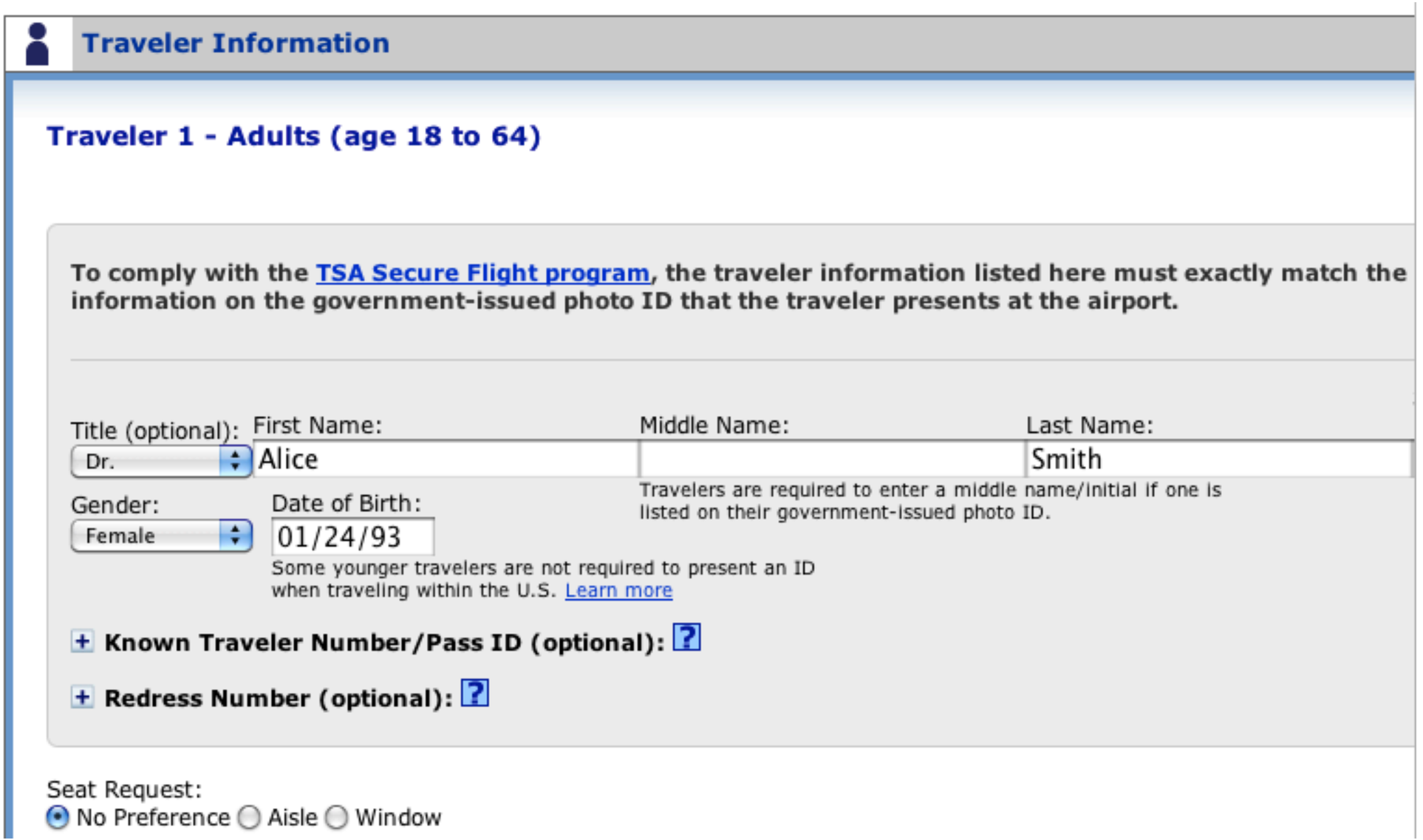


- Ethereum is a cryptocurrency which offers "smart" contracts
  - Program you money in a language that makes JavaScript and PHP look beautiful and sane
- The DAO (Distributed Autonomous Organization) was an attempt to make a distributed mutual fund in Ethereum
  - Participants could vote on "investments" that should be made
- The DAO supported withdrawals as well

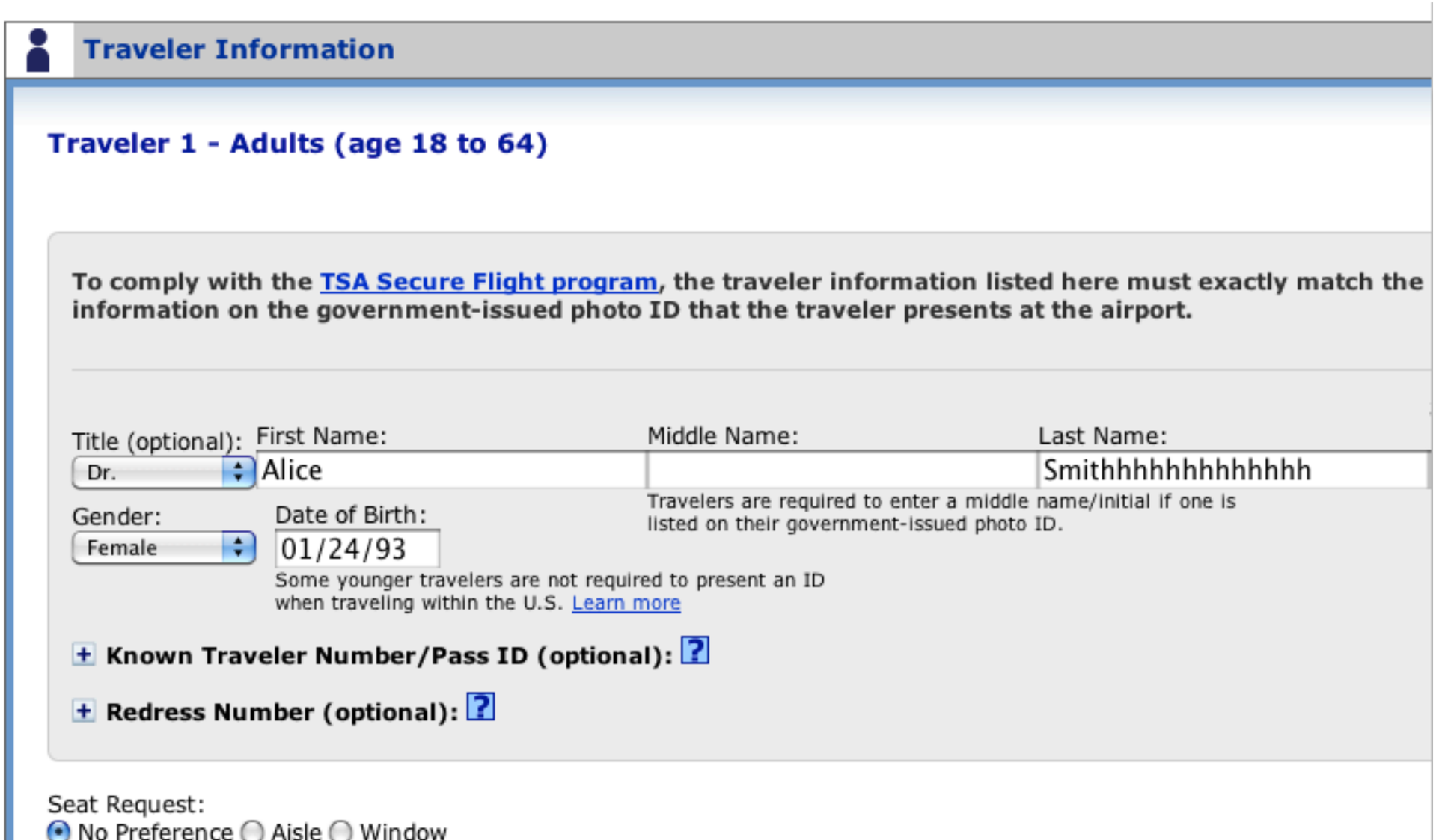
A "Feature" In The Smart Contract

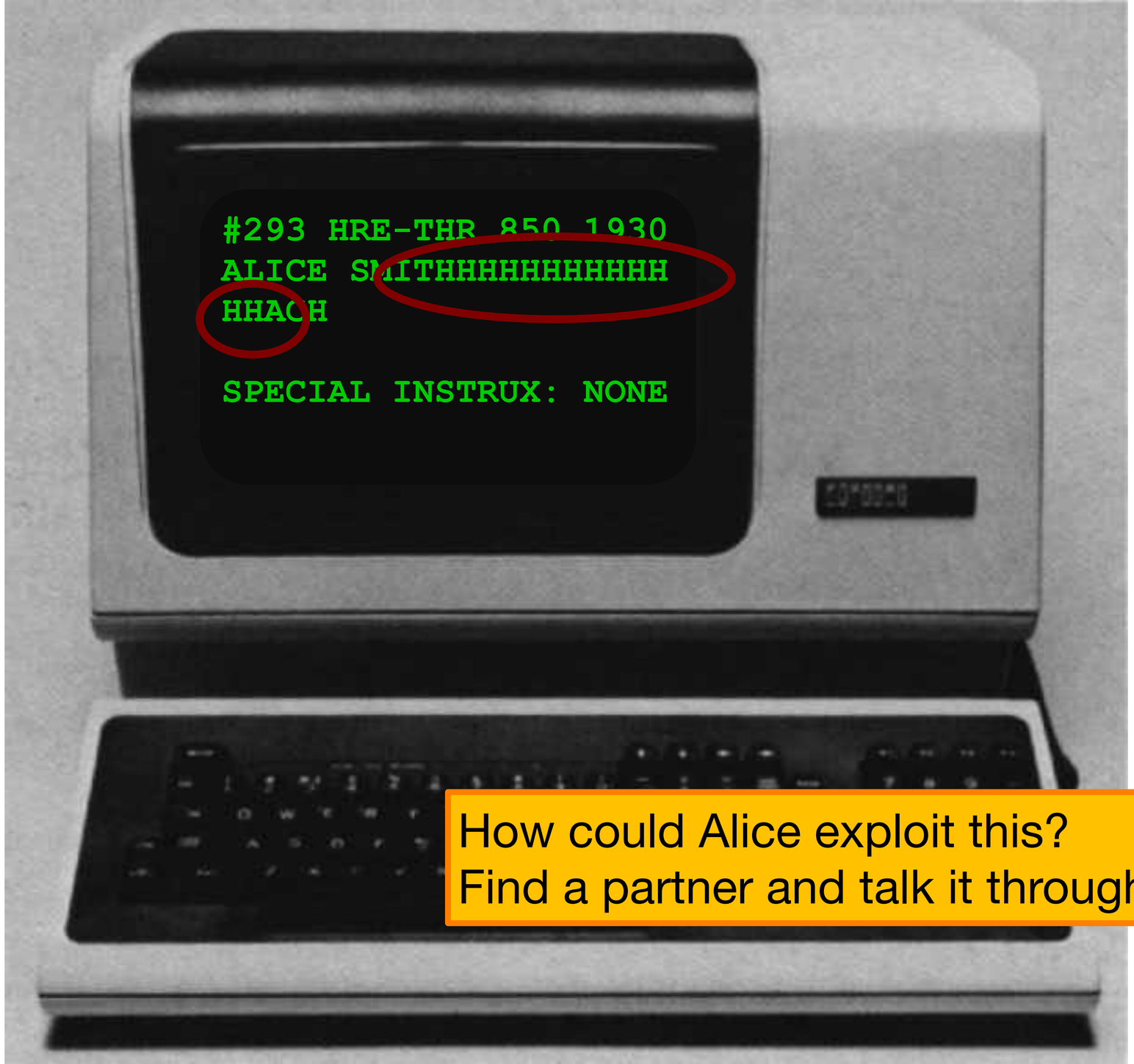
- To withdraw, the code was:
  - Check the balance, then send the money, then decrement the balance
- But sending money in Ethereum can send to ***another program written by the recipient***
- So someone "invested", then did a withdraw to his program
  - Which would initiate another withdraw...

Buffer Overflows

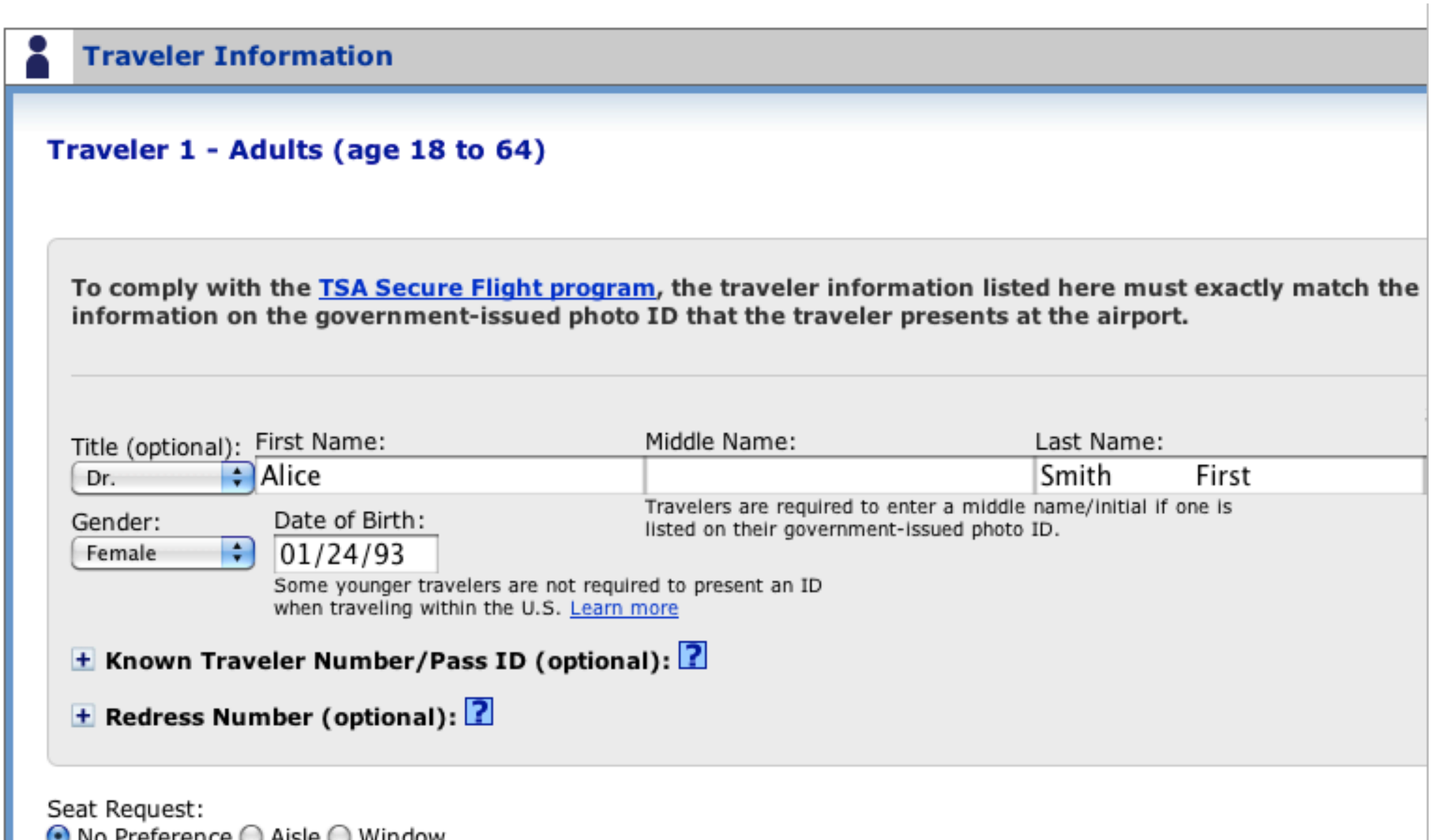

Traveler Information
Traveler 1 - Adults (age 18 to 64)
To comply with the TSA Secure Flight program, the traveler information listed here must exactly match the information on the government-issued photo ID that the traveler presents at the airport.
Title (optional):
Dr.
First Name:
Alice
Middle Name:
Last Name:
Smith
Travelers are required to enter a middle name/initial if one is listed on their government-issued photo ID.
Gender:
Female
Date of Birth:
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


#293 HRE-THR 850 1930
ALICE SMITH
COACH
SPECIAL INSTRUX: NONE


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
Travelers are required to enter a middle name/initial if one is listed on their government-issued photo ID.
Gender:
Female
Date of Birth:
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



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):
Dr.
First Name:
Alice
Middle Name:
Last Name:
Smith
First
Travelers are required to enter a middle name/initial if one is listed on their government-issued photo ID.
Gender:
Female
Date of Birth:
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:


#293 HRE-THR 850 1930
ALICE SMITH
FIRST
SPECIAL INSTRUX: NONE

```
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);
   ...
}
```

The CWE Top 25

Below is a brief listing of the weaknesses in the 2019 CWE Top 25, including the overall score of each.

Rank	ID	Name	Score
[1]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	75.56
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	45.69
[3]	CWE-20	Improper Input Validation	43.61
[4]	CWE-200	Information Exposure	32.12
[5]	CWE-125	Out-of-bounds Read	26.53
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	24.54
[7]	CWE-416	Use After Free	17.94
[8]	CWE-190	Integer Overflow or Wraparound	17.35
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	15.54
[10]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.10
[11]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	11.47
[12]	CWE-787	Out-of-bounds Write	11.08
[13]	CWE-287	Improper Authentication	10.78
[14]	CWE-476	NULL Pointer Dereference	9.74
[15]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.33
[16]	CWE-434	Unrestricted Upload of File with Dangerous Type	5.50
[17]	CWE-611	Improper Restriction of XML External Entity Reference	5.48
[18]	CWE-94	Improper Control of Generation of Code ('Code Injection')	5.36
[19]	CWE-798	Use of Hard-coded Credentials	5.12

```
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.

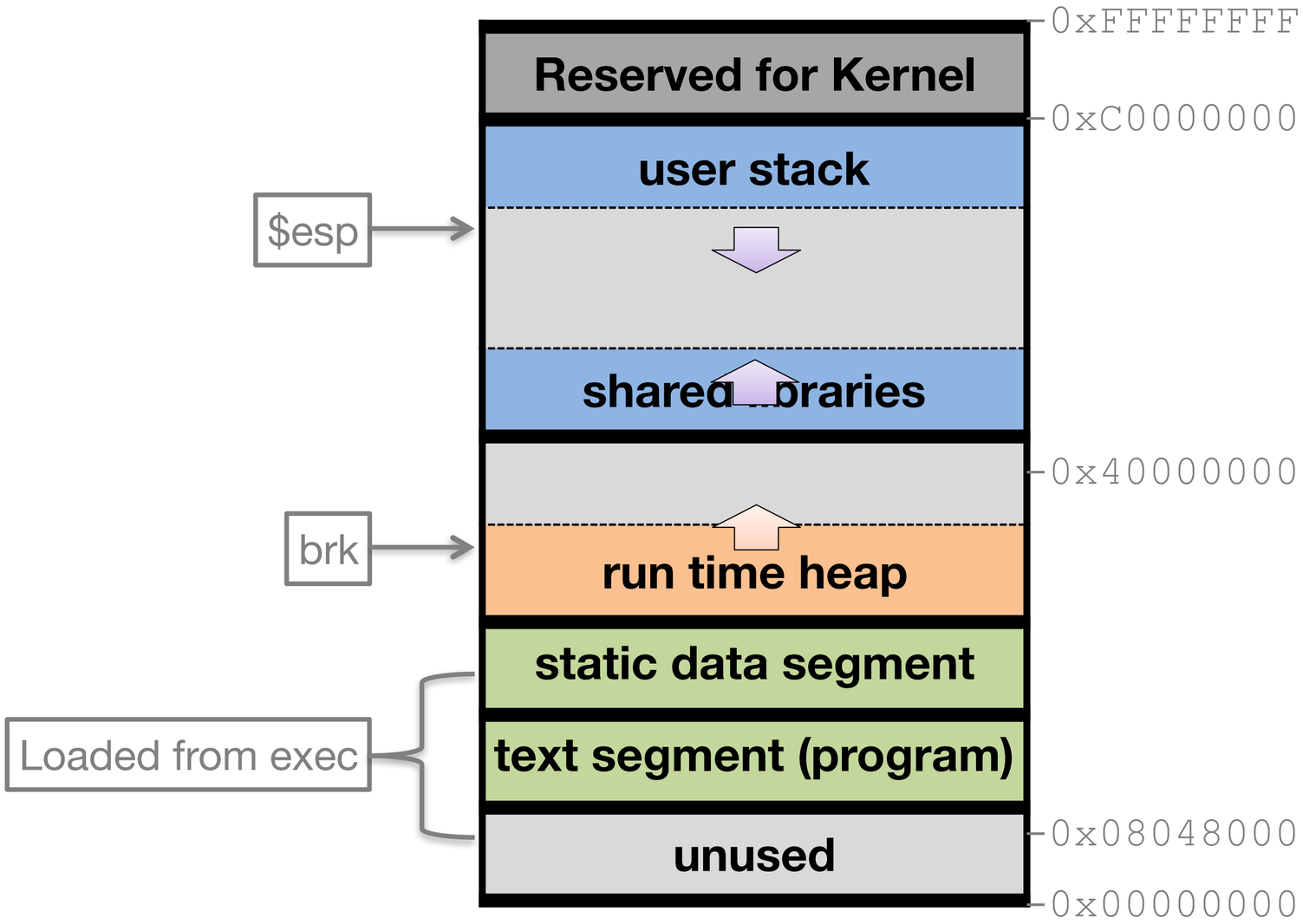
Pointing to the Flaw

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.

Disclaimer: x86-32

- For this class, we are going to use 32-bit x86
  - Almost everyone in this class has access to an x86 system: Mac, Linux, Windows...
- But these attacks do apply to other microarchitectures

Linux (32-bit) process memory layout

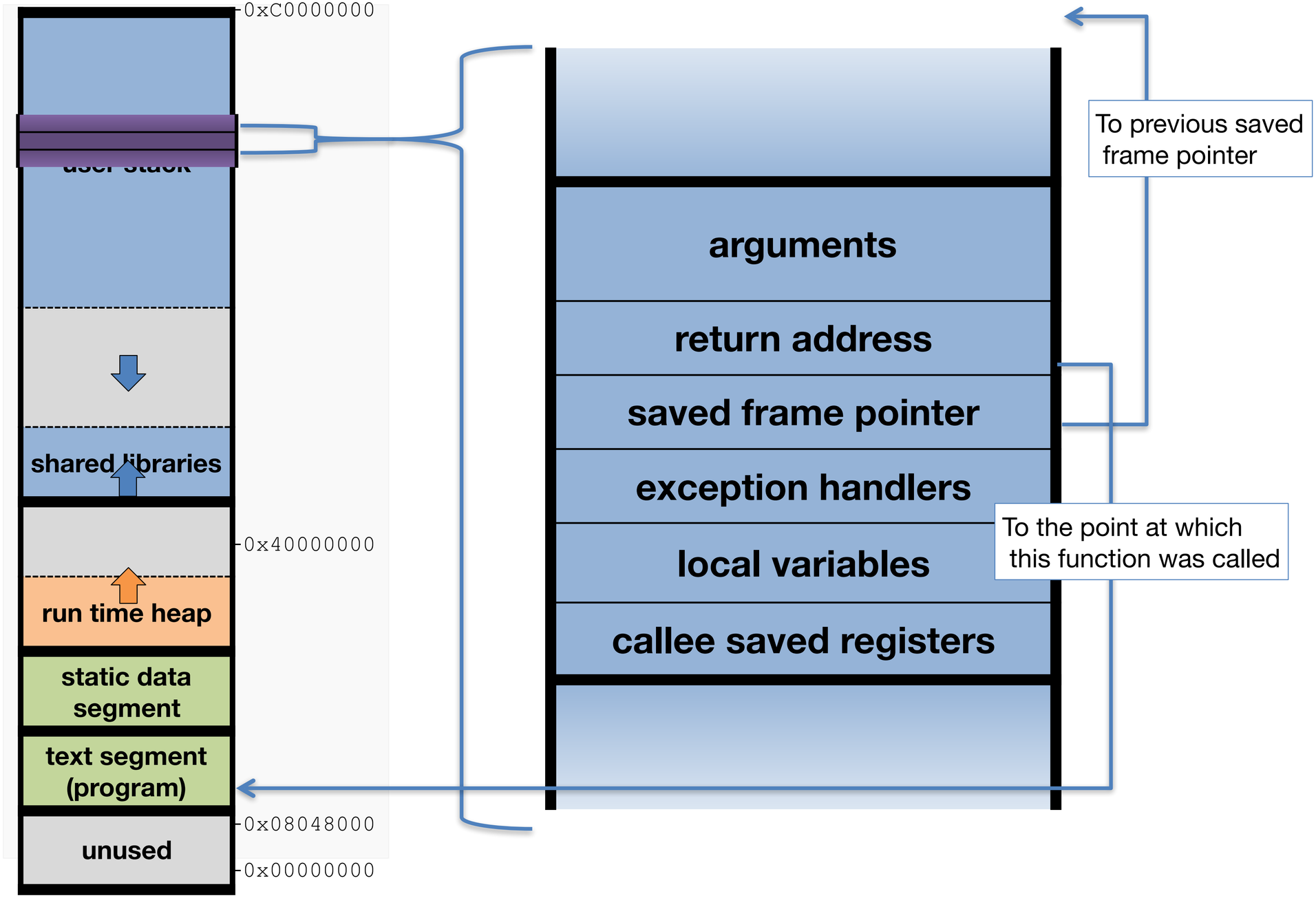


The main x86 registers…

- EAX-EDX: General purpose registers
- EBP: “Frame pointer”: points to the start of the current call frame on the stack
- ESP: “Stack pointer”: points to the current stack
  - PUSH: Decrement the stack pointer and store something there
  - POP: Load something and increment the stack pointer

x86 function calling

- Place the arguments on the stack
- CALL the function
  - Which pushes the return address onto the stack (RIP == Return Instruction Pointer)
- Function saves old EBP on the stack (SFP == Saved Frame Pointer)
- Function does its stuff
- Function restores everything
  - Reload EBP, pop ESP as necessary
- RET
  - Which jumps to the return address that is currently pointed to by ESP
  - And can optionally pop the stack a lot further…


0xC0000000
user stack
shared libraries
0x40000000
run time heap
static data segment
text segment (program)
0x08048000
unused
0x00000000
arguments
return address
saved frame pointer
exception handlers
local variables
callee saved registers
To previous saved frame pointer
To the point at which this function was called

```
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

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.

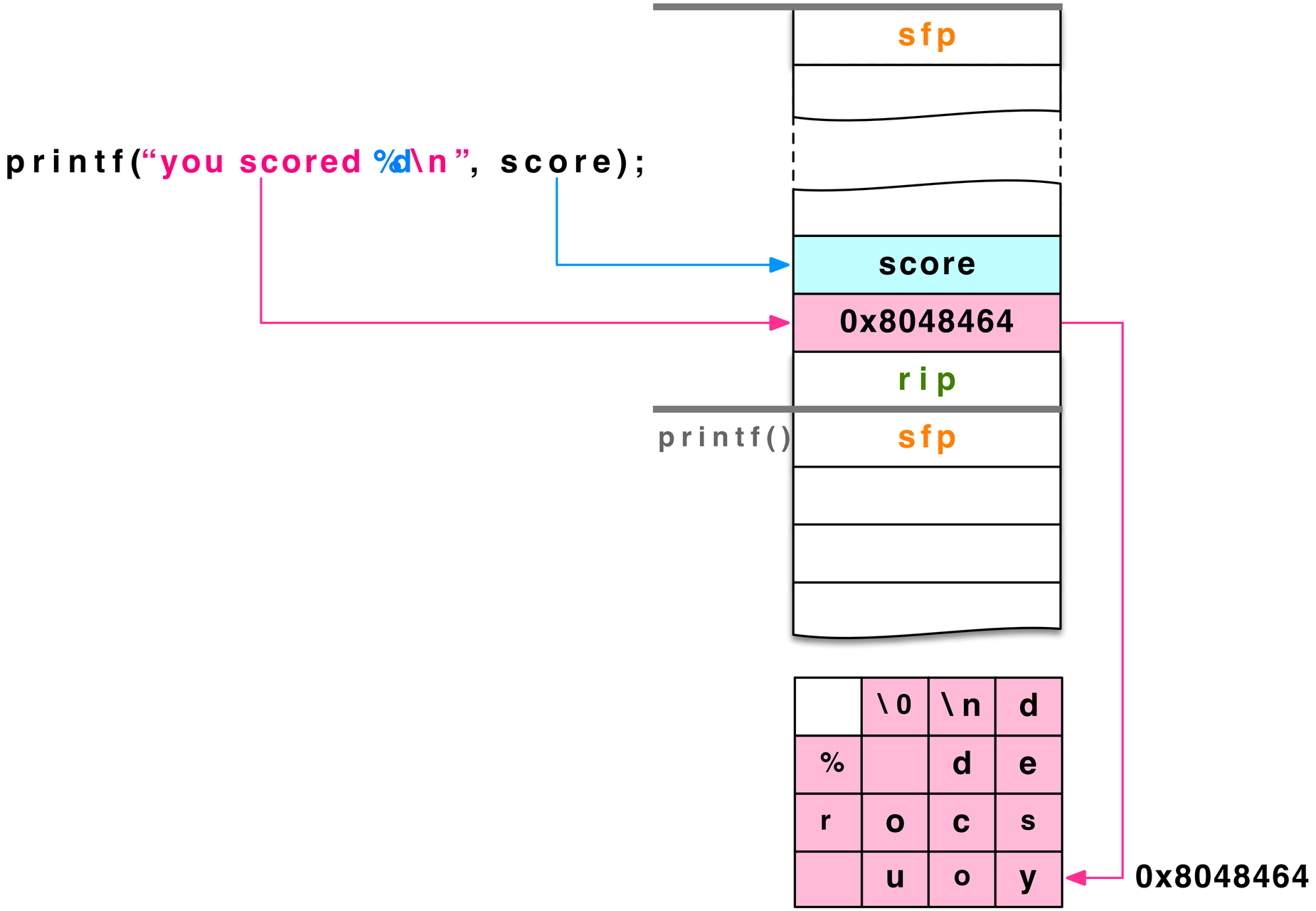
"The software is not geared to count more than 32,000 votes in a precinct. So what happens when it gets to 32,000 is the software starts counting backward," said Broward County Mayor Ilene Lieberman.

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

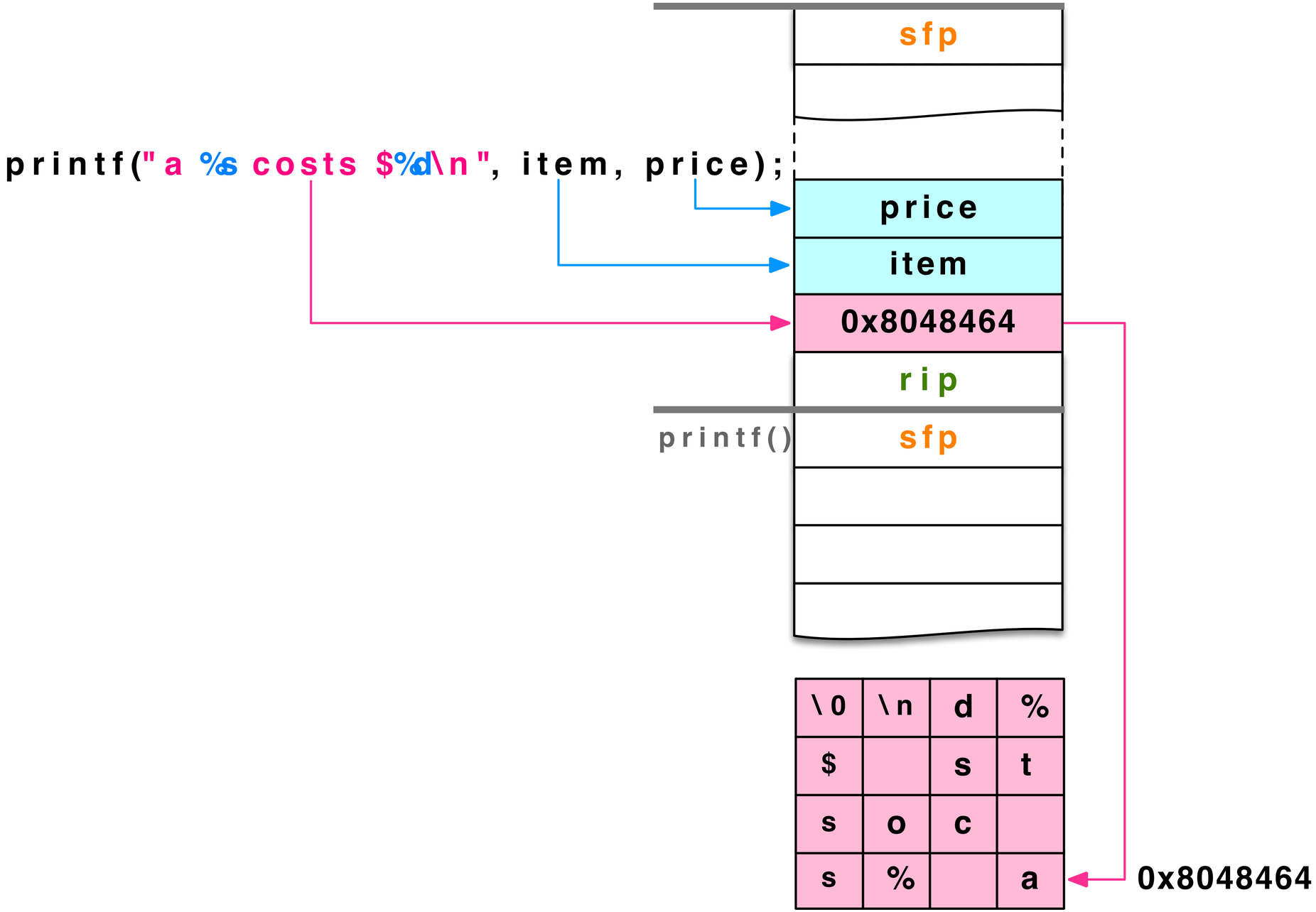
That means that Amendment 4 passed in Broward County by more than 240,000 votes rather than the 166,000-vote margin reported Wednesday night. That increase changes the overall statewide results in what had been a neck-and-neck race, one for which recounts had been going on today. But with news of Broward's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

```
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);
sfp
score
0x8048464
rip
printf()
sfp
\0 \n d
% d e
r o c s
u o y
0x8048464

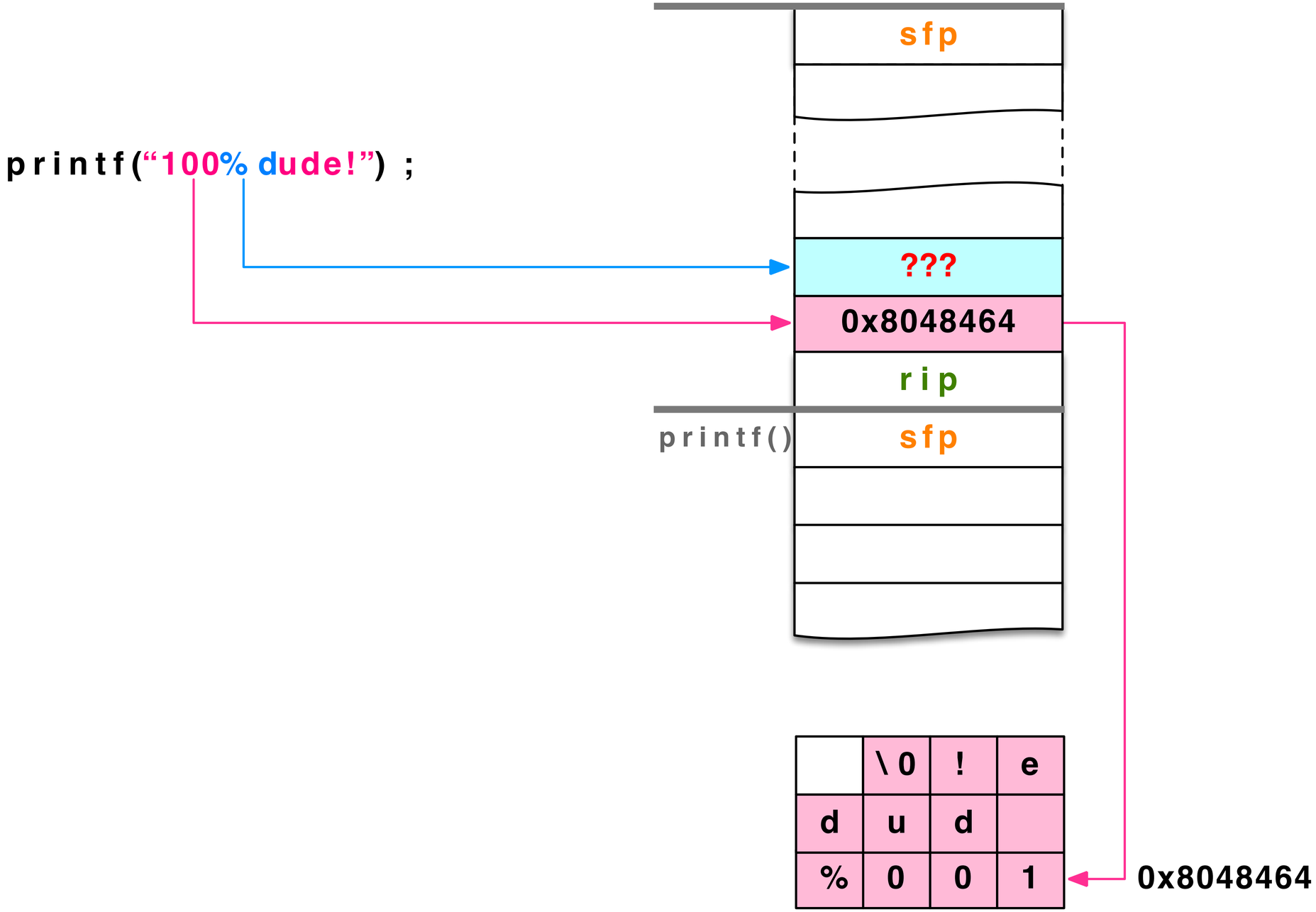
```
printf("a %s costs $%d\n", item, price);
```


printf("a %s costs $%d\n", item, price);
sfp
price
item
0x8048464
rip
printf()
sfp
\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!") ;
sfp
???
0x8048464
rip
printf()
sfp
\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

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);
}
```