

# **Software Security, Implementation Flaws, and Memory Safety**

***CS 161: Computer Security***

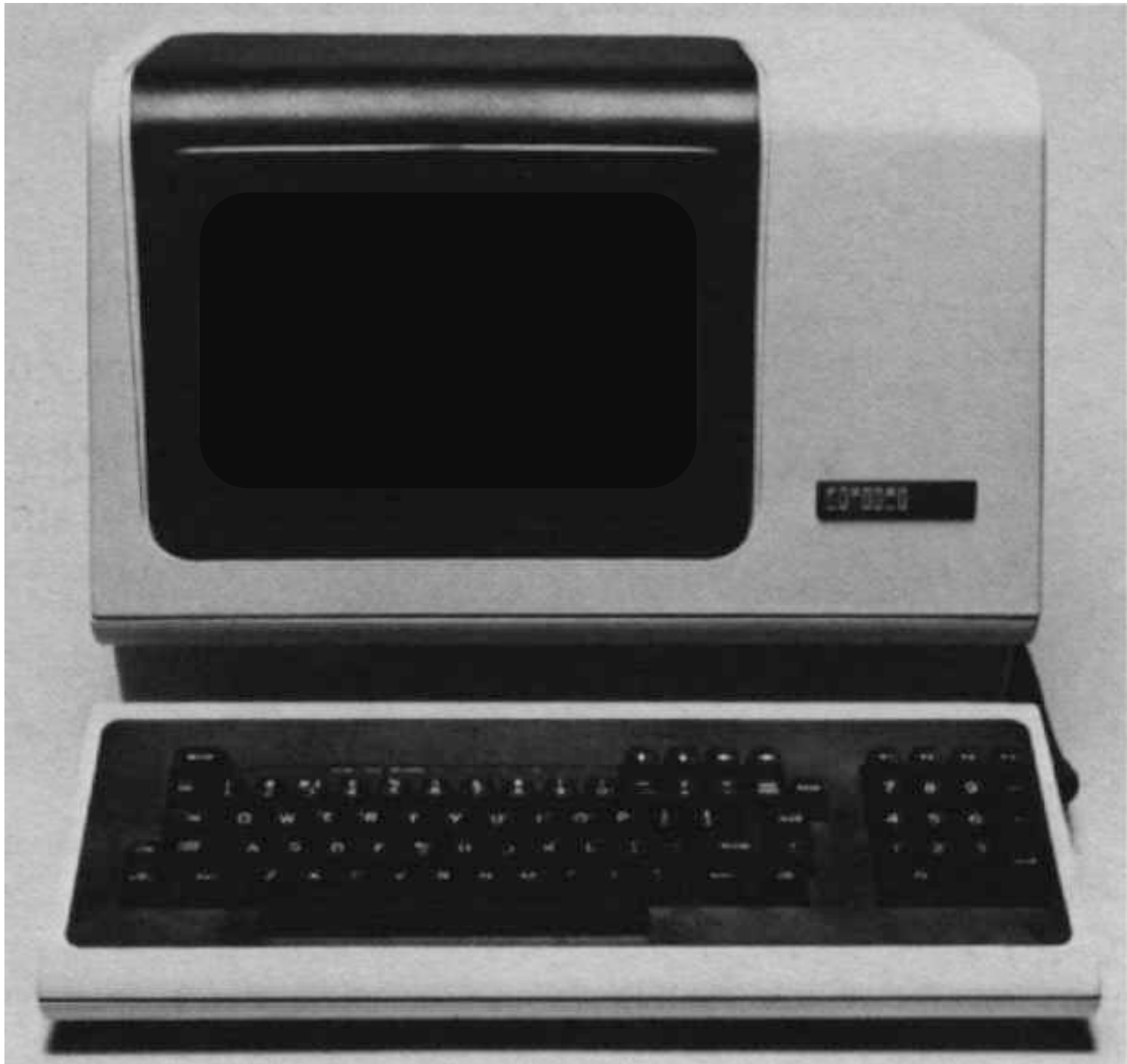
**Prof. Vern Paxson**


**TAs: Devdatta Akhawe, Mobin Javed  
& Matthias Vallentin**

*<http://inst.eecs.berkeley.edu/~cs161/>*

**January 20, 2011**








**#293 HRE-THR 850 1930  
ALICE SMITH  
COACH**

**SPECIAL INSTRUX: NONE**


COACH





**#293 HRE-THR 850 1930**  
**ALICE SMITHHHHHHHHHHH**  
**HHACH**

**SPECIAL INSTRUX: NONE**



**#293 HRE-THR 850 1930**  
**ALICE SMITH**  
**FIRST**

**SPECIAL INSTRUX: GIVE**  
**PAX EXTRA CHAMPAGNE.**

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

# Code Injection

# Excerpt From A Simple Network Server

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . .  
    munch(packet);  
  
    . . .  
}  
  
void munch(char *packet) {  
    int n;  
    char cookie[512];  
  
    . . .  
  
    code here computes offset of cookie in  
    packet, stores it in n  
  
    strcpy(cookie, &packet[n]);  
  
    . . .  
}
```

# Example: Normal Execution

→ `void get_cookie(char *packet) {`

`. . . (200 bytes of local vars) . . . X + 200` →

`munch(packet);`

`. . .`

`}`

`void munch(char *packet) {`

`int n;`

`char cookie[512];`

`. . .`

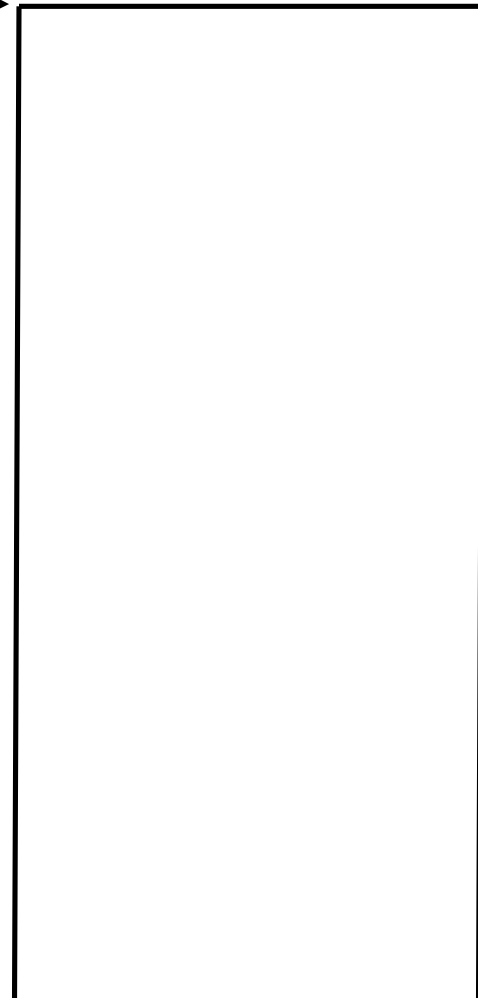
*code here computes offset of cookie in  
packet, stores it in **n***

`strcpy(cookie, &packet[n]);`

`. . .`

`}`

Stack





# Example: Normal Execution

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    → munch(packet);
```

```
    . . .
```

```
}
```

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

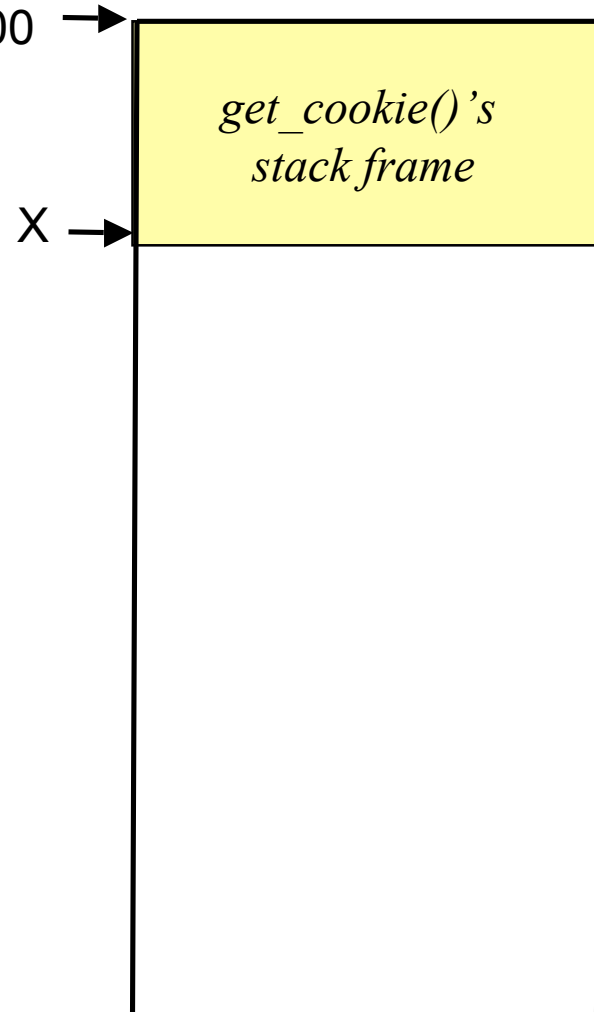
```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

Stack

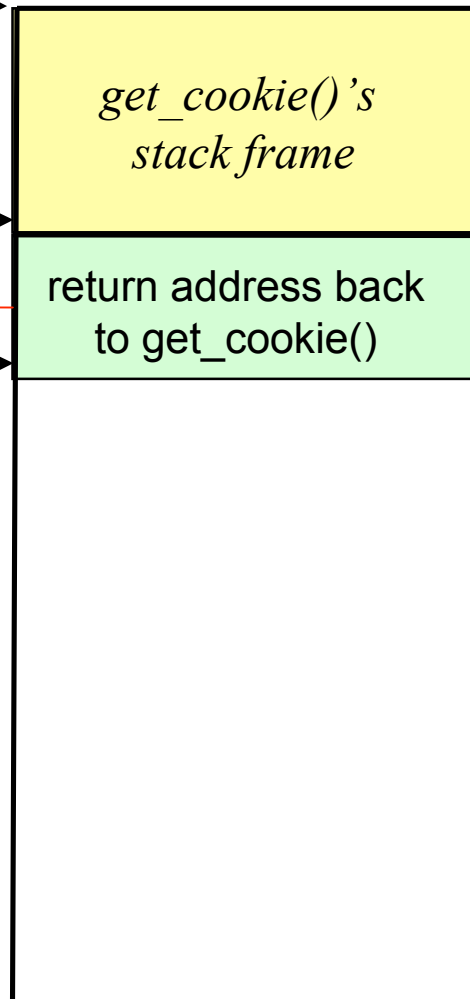


# Example: Normal Execution

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```

Stack



# Example: Normal Execution

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    munch(packet);
```

```
    . . .
```

```
}
```

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

Stack

*get\_cookie()'s  
stack frame*

return address back  
to get\_cookie()

n

cookie

X + 200 →

X →

X - 4 →

X - 8 →

X - 520 →



# Example: Normal Execution

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    munch(packet);
```

```
    . . .
```

```
}
```

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

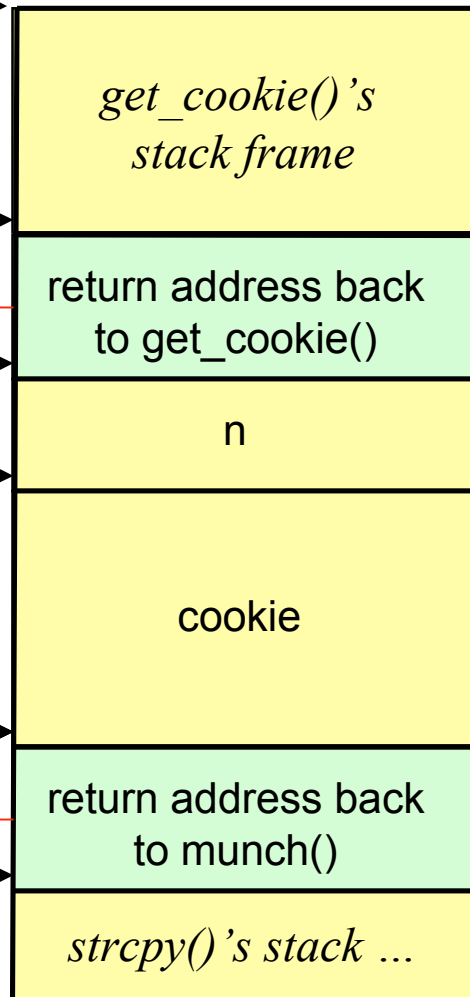
```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

Stack



# Example: Normal Execution

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    munch(packet);
```

```
    . . .
```

```
}
```

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

Stack

*get\_cookie()'s  
stack frame*

return address back  
to get\_cookie()

n

*cookie value read  
from packet*

return address back  
to munch()

X

X - 4

X - 8

X - 520

X - 524

# Example: Normal Execution

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    munch(packet);
```

```
    . . .
```

```
}
```

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

Stack

*get\_cookie()'s  
stack frame*

return address back  
to get\_cookie()

n

*cookie value read  
from packet*

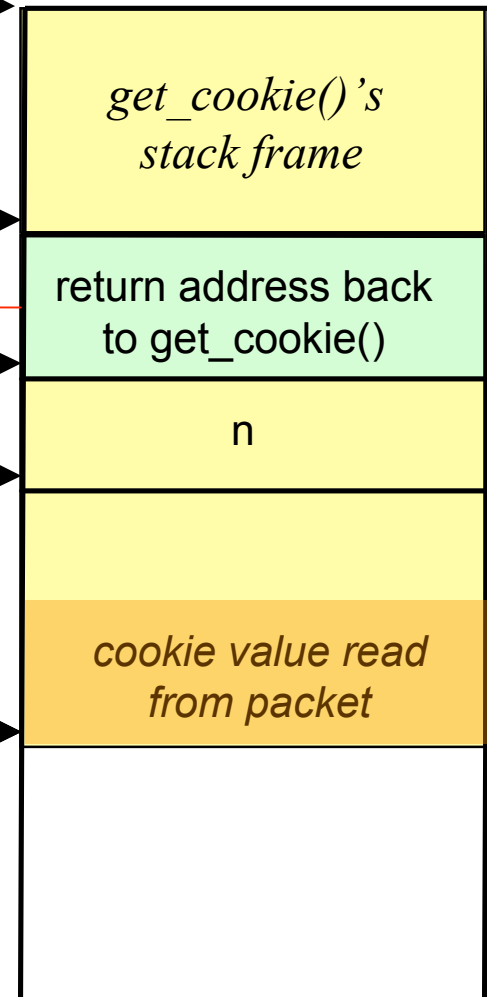
X + 200 →

X →

X - 4 →

X - 8 →

X - 520 →



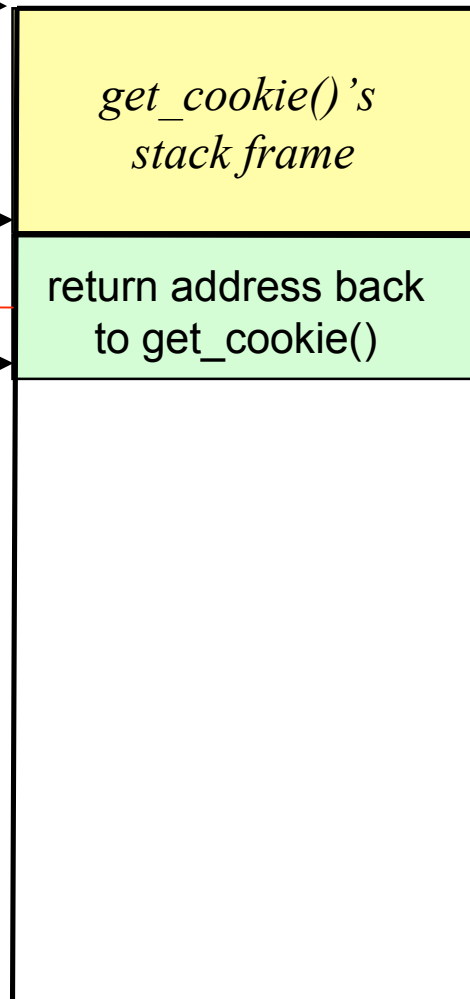


# Example: Normal Execution

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```

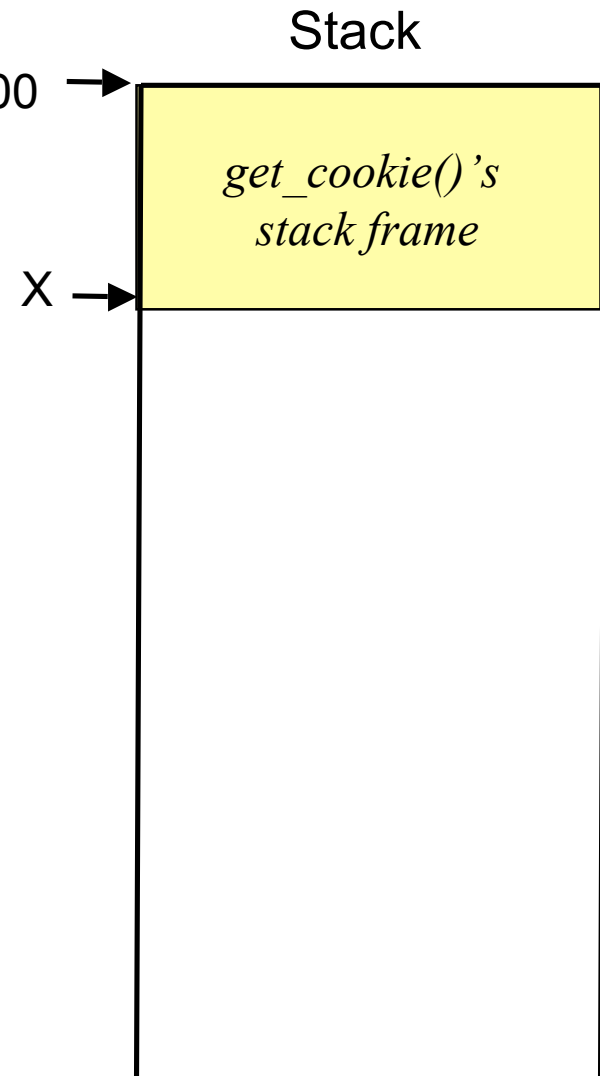
Stack



# Example: Normal Execution

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . .  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```



# Example: Normal Execution

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    munch(packet);
```

```
    . . .
```



```
}  
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

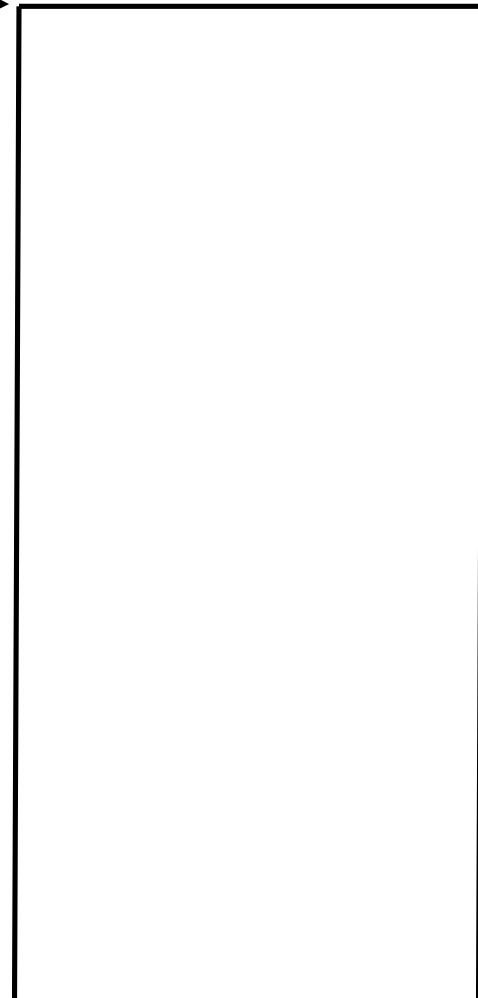
```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

Stack

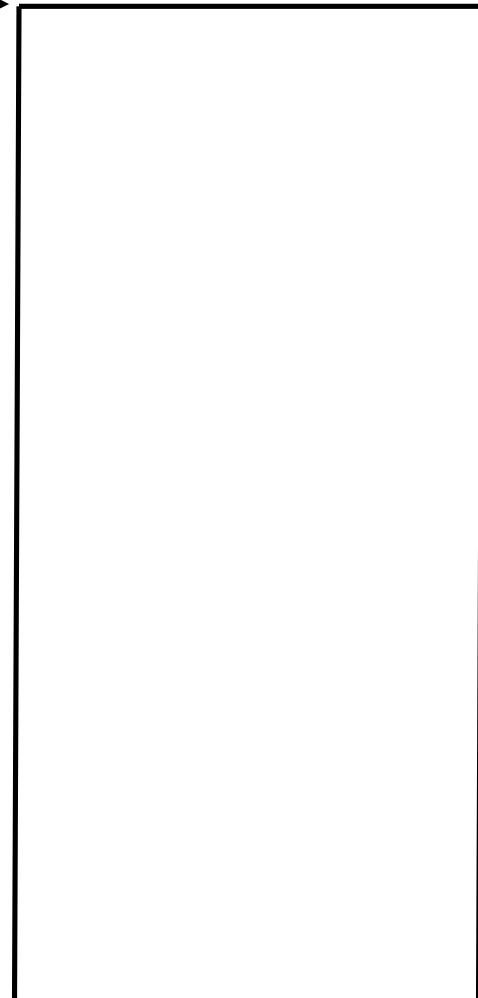


# Example: Buffer Overflow

```
→ void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200 →  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```

Stack



# Example: Buffer Overflow

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    → munch(packet);
```

```
    . . .
```

```
}
```

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

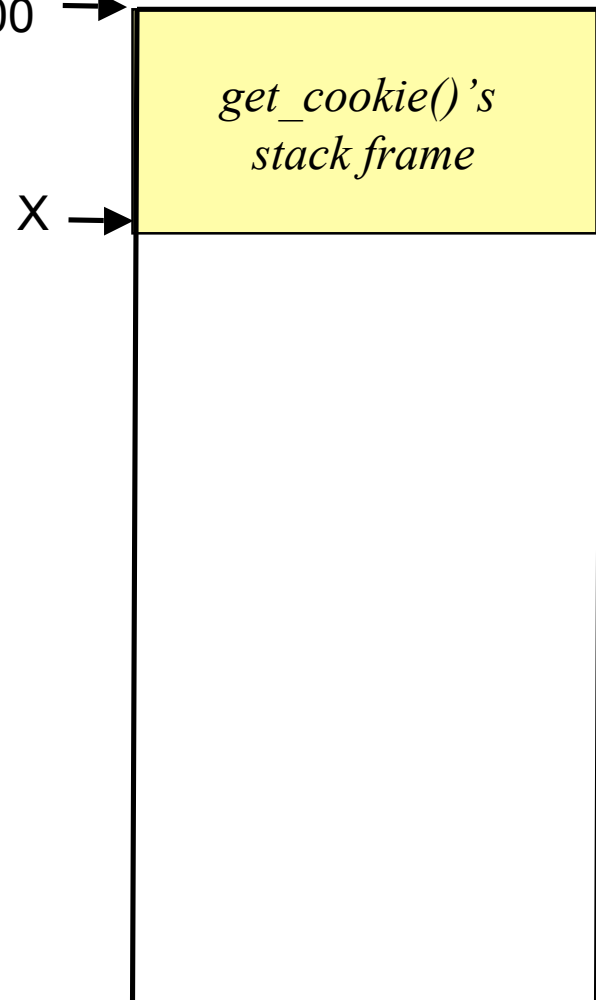
```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

Stack

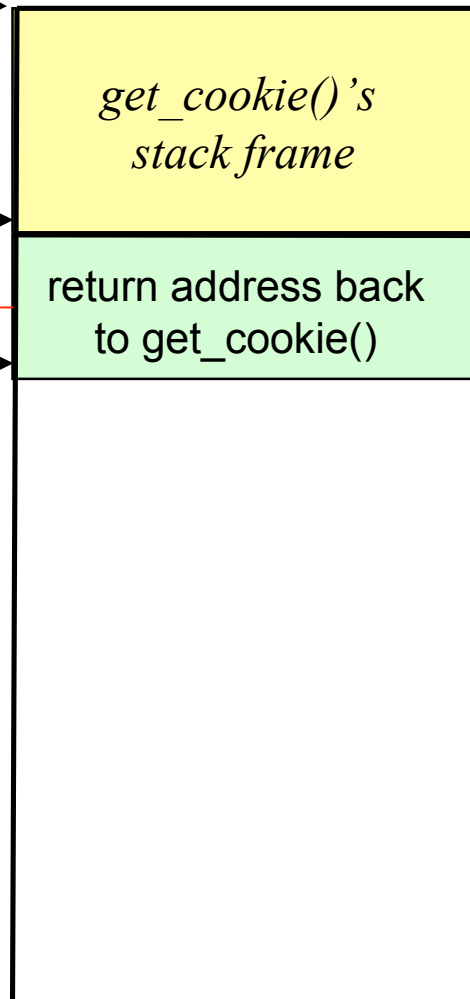


# Example: Buffer Overflow

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```

Stack





# Example: Buffer Overflow

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    munch(packet);
```

```
    . . .
```

```
}
```

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

Stack

*get\_cookie()'*s  
*stack frame*

return address back  
to `get_cookie()`

n

cookie

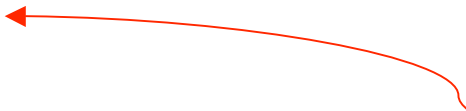
X + 200 →

X →

X - 4 →

X - 8 →

X - 520 →



# Example: Buffer Overflow

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    munch(packet);
```

```
    . . .
```

```
}
```

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

Stack

*get\_cookie()'*s  
*stack frame*

return address back  
to *get\_cookie()*

n

cookie

return address back  
to *munch()*

*strcpy()'*s stack ...

X + 200 →

X →

X - 4 →

X - 8 →

X - 520 →

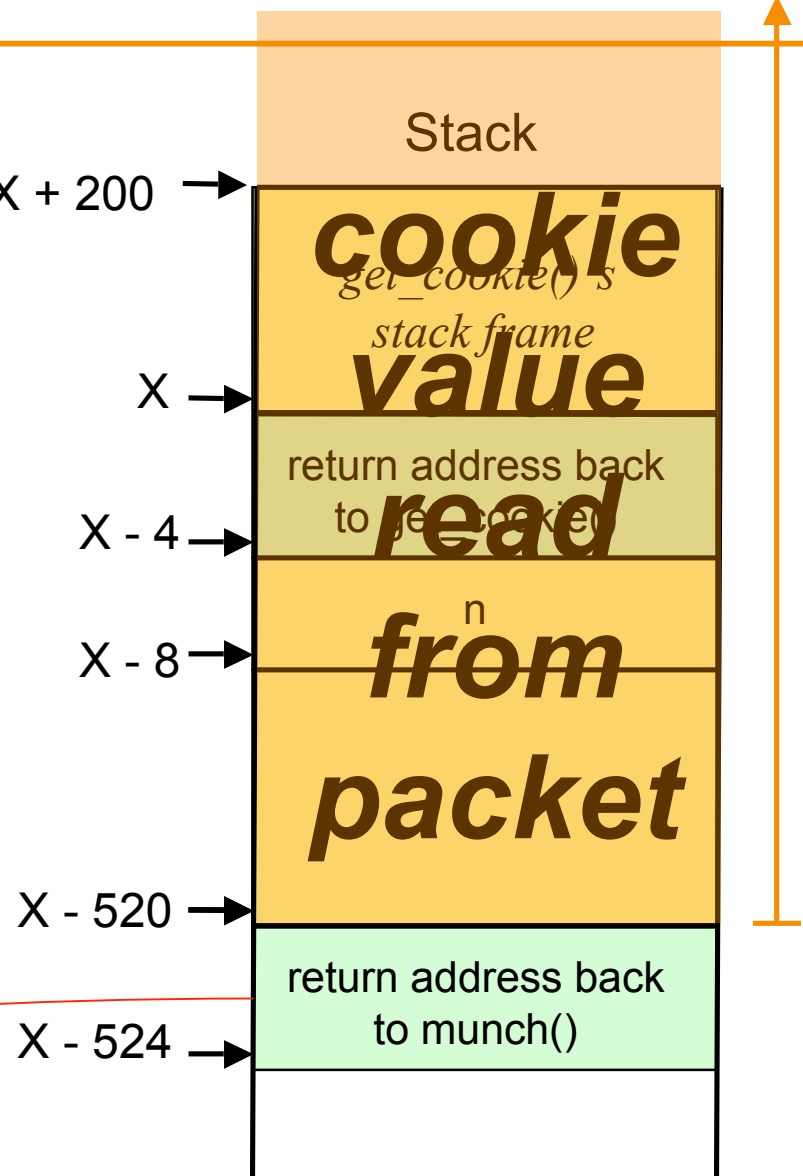
X - 524 →



# Example: Buffer Overflow

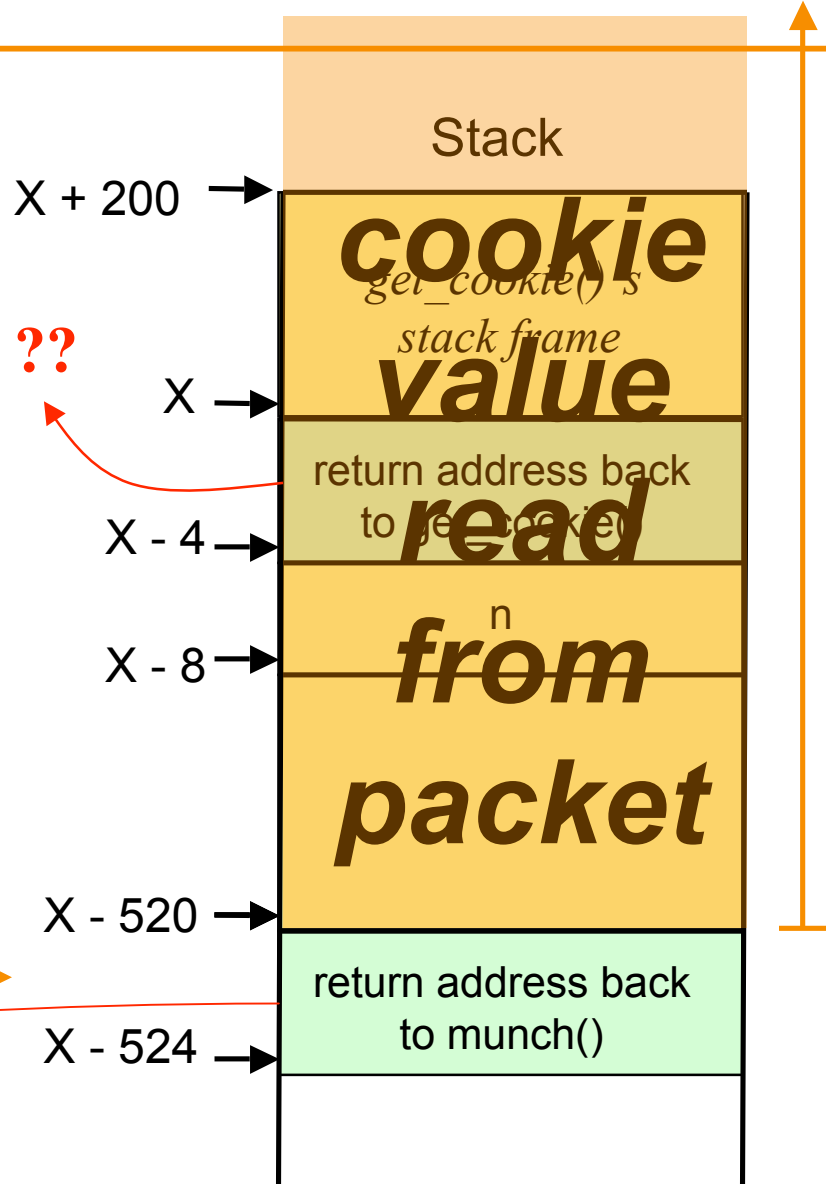
```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```



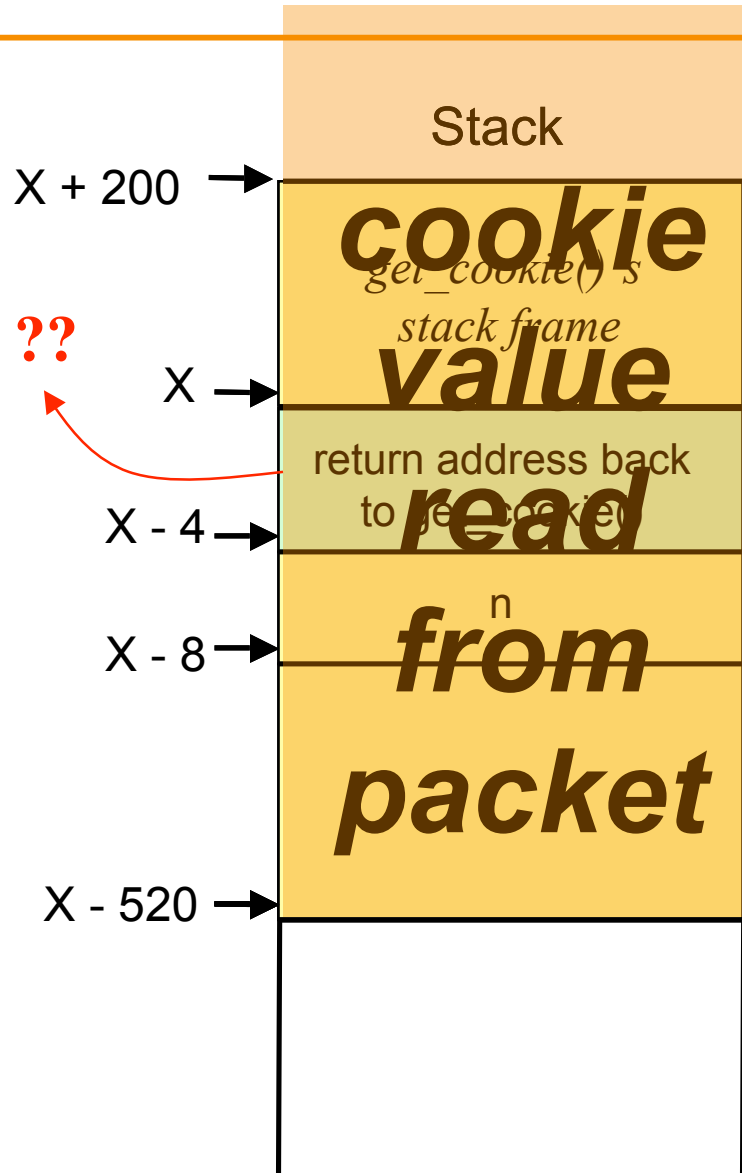
# Example: Buffer Overflow

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . .  
    munch(packet);  
    . . .  
}  
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```



# Example: Buffer Overflow

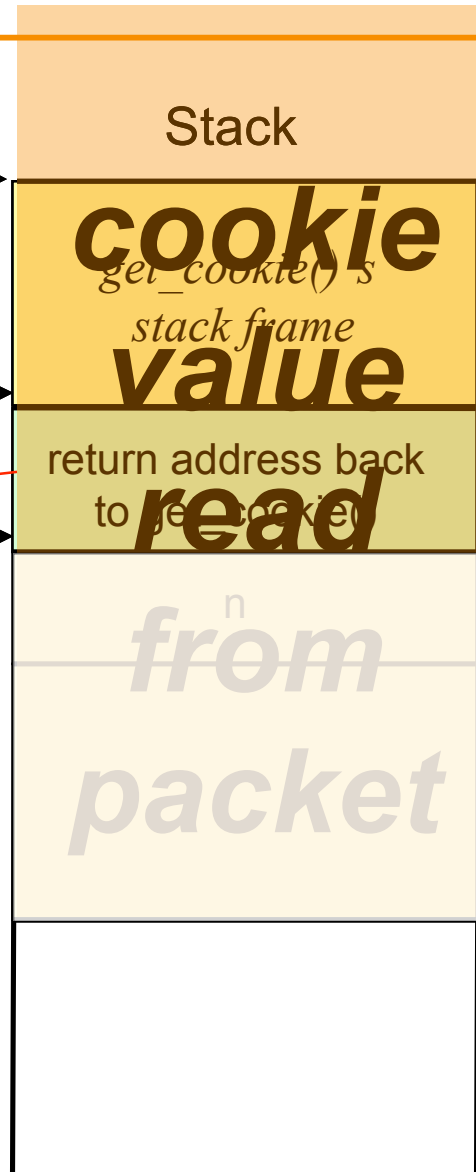
```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . .  
    munch(packet);  
    . . .  
}  
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```



# Example: Buffer Overflow

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . .  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```



# Example: Buffer Overflow

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    memcpy(cookie, packet, n);  
    . . .  
}  
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```



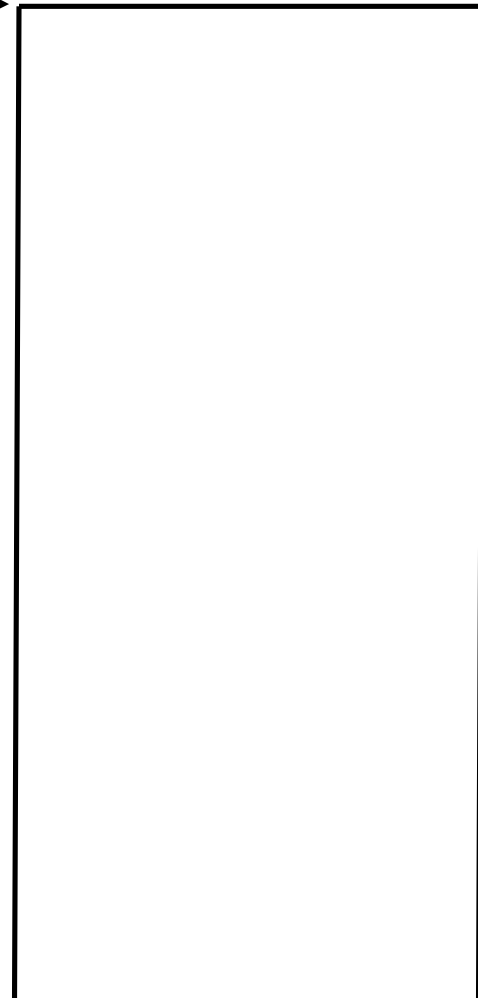
*Crash*

# Example: Code Injection

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . .  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```

Stack





# Example: Code Injection

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    munch(packet);
```

```
    . . .
```

```
}
```

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

Stack

*get\_cookie()'s  
stack frame*

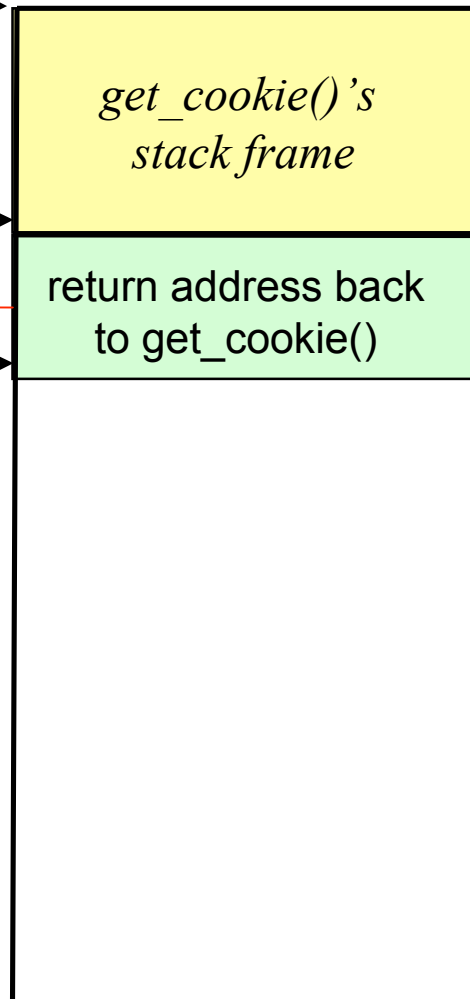
X

# Example: Code Injection

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```

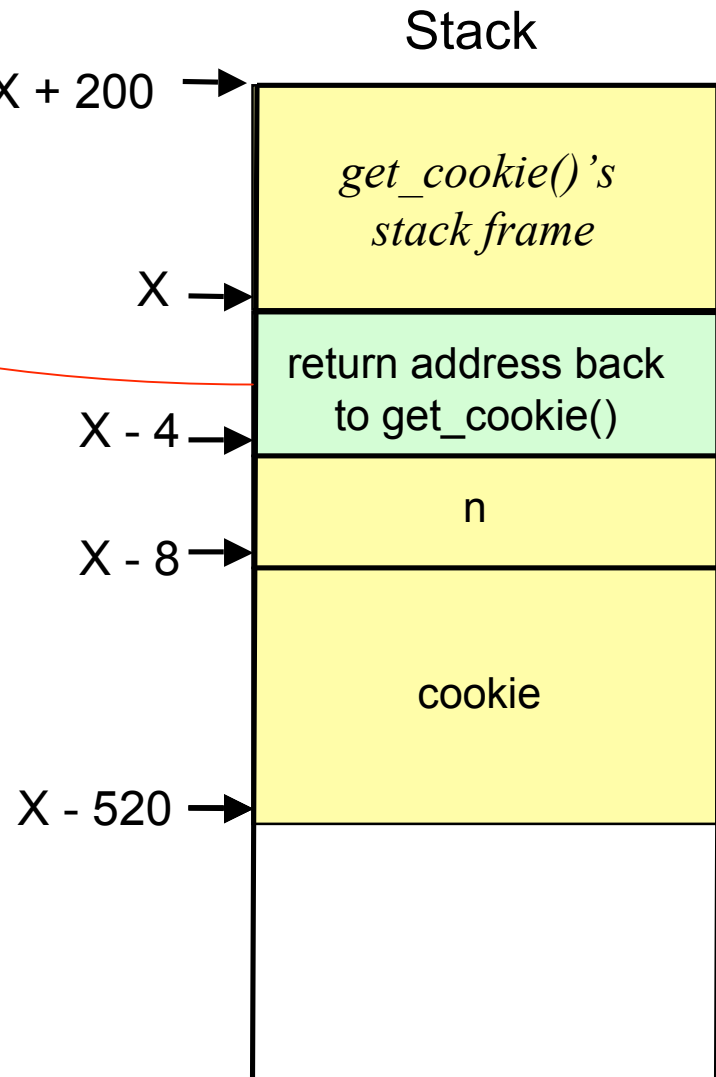
Stack



# Example: Code Injection

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```



# Example: Code Injection

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    munch(packet);
```

```
    . . .
```

```
}
```

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

```
    . . .
```

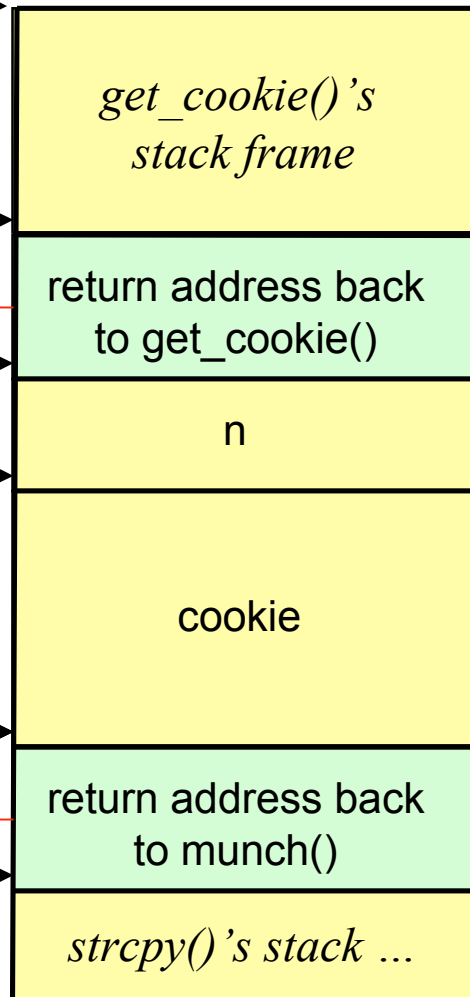
```
    code here computes offset of cookie in  
    packet, stores it in n
```

```
➔ strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```

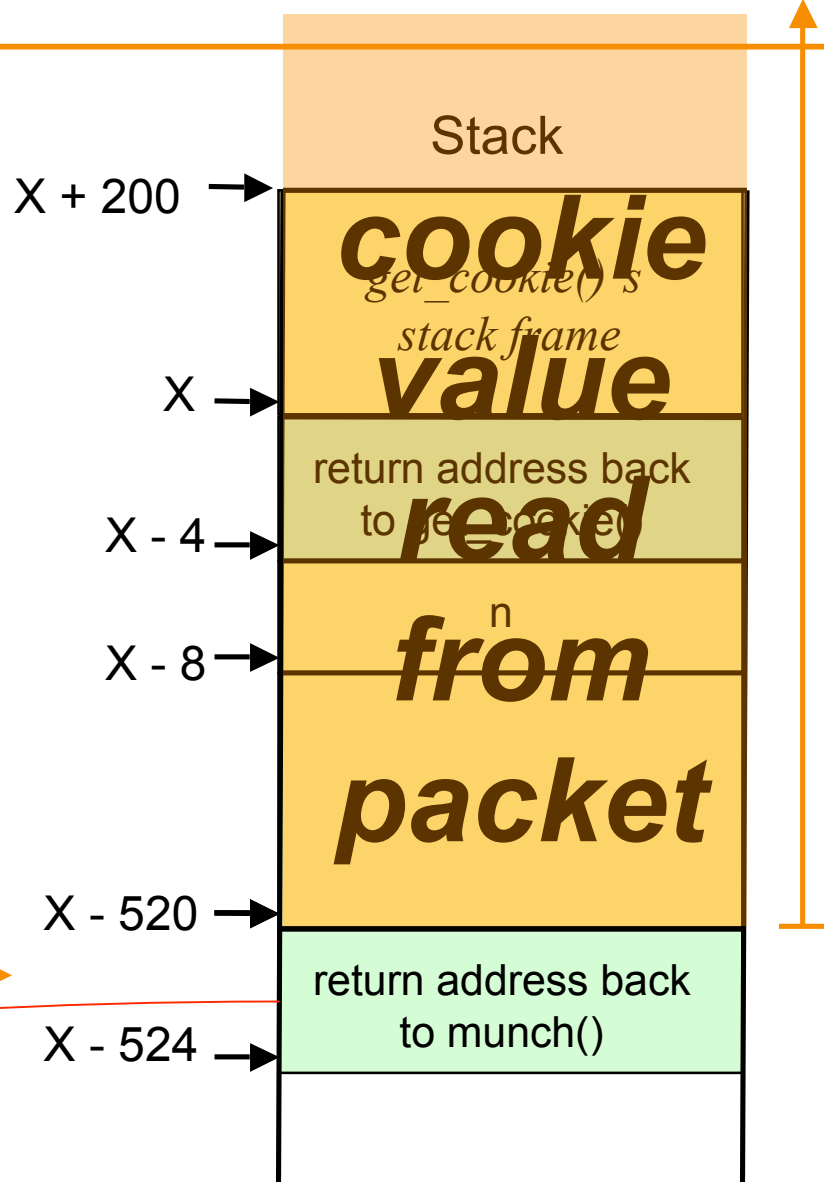
Stack



# Example: Code Injection

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    munch(packet);  
    . . .  
}
```

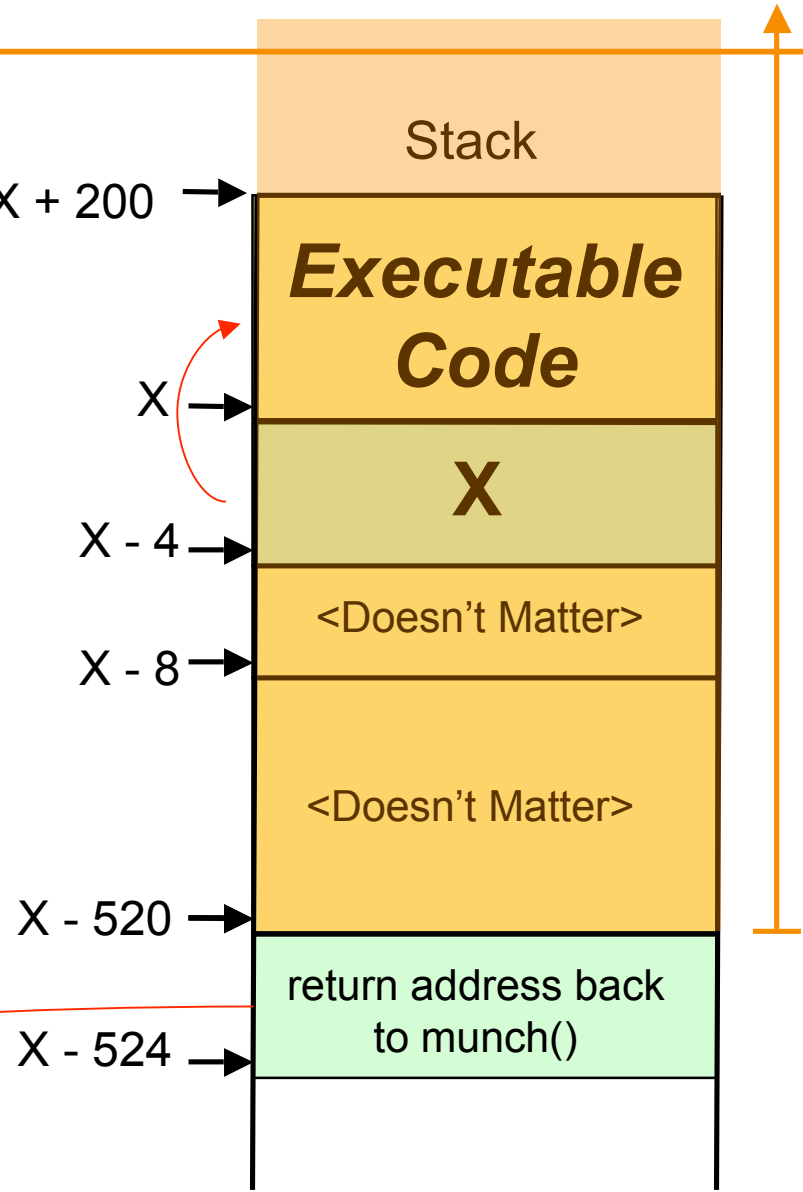
```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```



# Example: Code Injection

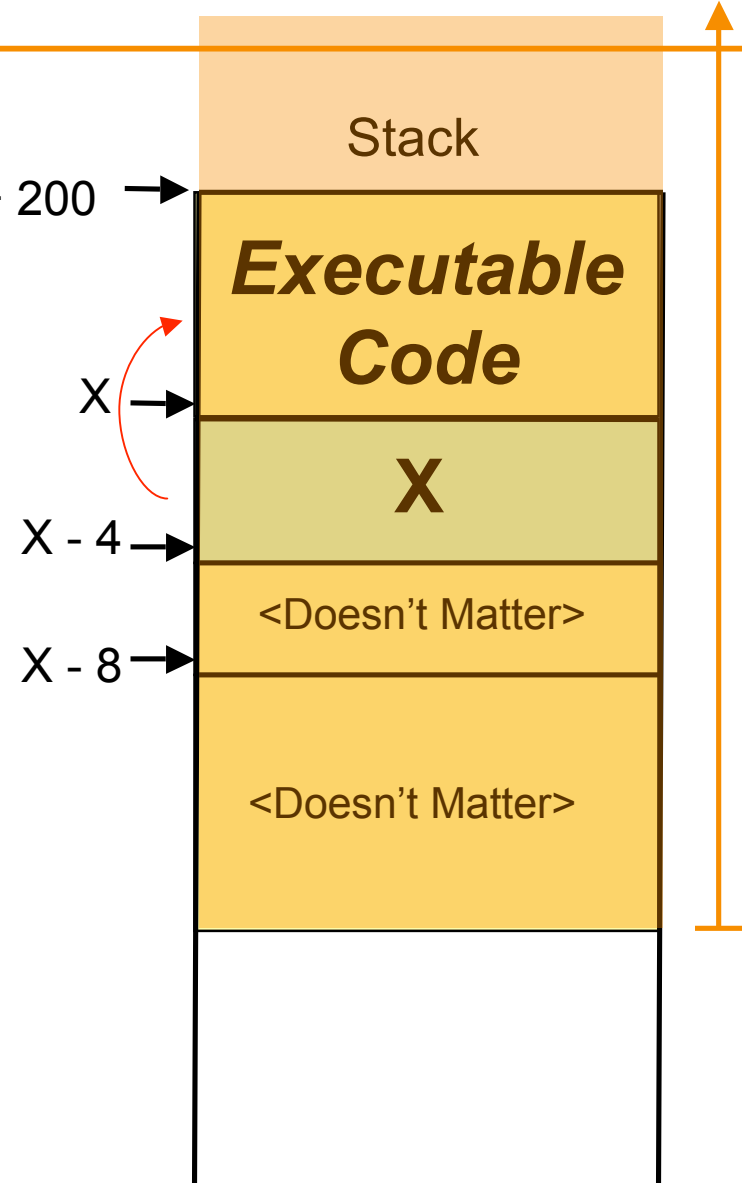
```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```



# Example: Code Injection

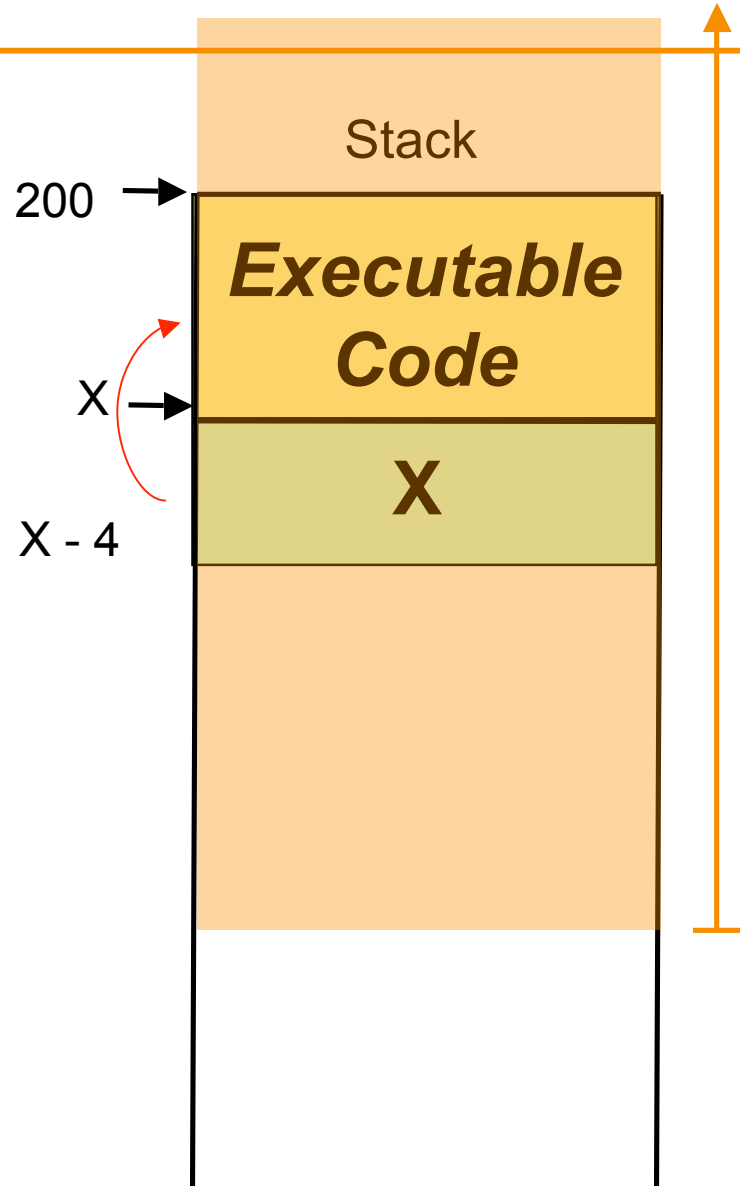
```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    munch(packet);  
    . . .  
}  
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```



# Example: Code Injection

```
void get_cookie(char *packet) {  
    . . . (200 bytes of local vars) . . . X + 200  
    munch(packet);  
    . . .  
}
```

```
void munch(char *packet) {  
    int n;  
    char cookie[512];  
    . . .  
    code here computes offset of cookie in  
    packet, stores it in n  
    strcpy(cookie, &packet[n]);  
    . . .  
}
```





# Example: Code Injection

```
void get_cookie(char *packet) {
```

```
    . . . (200 bytes of local vars) . . . X + 200
```

```
    munch(packet);
```

*Now branches to code **read in from the network***

```
void munch(char *packet) {
```

```
    int n;
```

```
    char cookie[512];
```

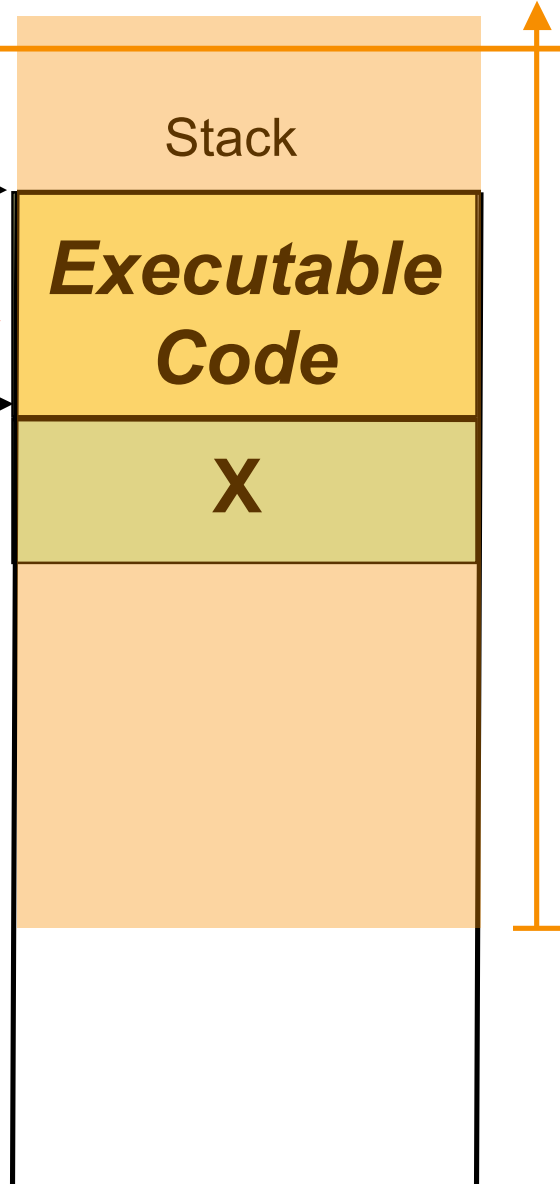
***From here on, machine falls under the attacker's control***

*code here computes offset of cookie in packet, stores it in n*

```
    strcpy(cookie, &packet[n]);
```

```
    . . .
```

```
}
```



Rank	Score	ID	Name
[1]	346	<a href="#">CWE-79</a>	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[2]	330	<a href="#">CWE-89</a>	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[3]	273	<a href="#">CWE-120</a>	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	261	<a href="#">CWE-352</a>	Cross-Site Request Forgery (CSRF)
[5]	219	<a href="#">CWE-285</a>	Improper Access Control (Authorization)
[6]	202	<a href="#">CWE-807</a>	Reliance on Untrusted Inputs in a Security Decision
[7]	197	<a href="#">CWE-22</a>	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
[8]	194	<a href="#">CWE-434</a>	Unrestricted Upload of File with Dangerous Type
[9]	188	<a href="#">CWE-78</a>	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[10]	188	<a href="#">CWE-311</a>	Missing Encryption of Sensitive Data
[11]	176	<a href="#">CWE-798</a>	Use of Hard-coded Credentials
[12]	158	<a href="#">CWE-805</a>	Buffer Access with Incorrect Length Value
[13]	157	<a href="#">CWE-98</a>	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP File Inclusion')
[14]	156	<a href="#">CWE-129</a>	Improper Validation of Array Index
[15]	155	<a href="#">CWE-754</a>	Improper Check for Unusual or Exceptional Conditions
[16]	154	<a href="#">CWE-209</a>	Information Exposure Through an Error Message

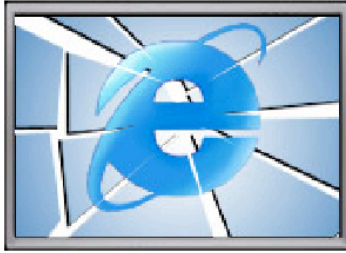
**5 Minute Break**

**Questions Before We Proceed?**

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

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

## 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."