Review: OS Section

CS461 / ECE422 – UIUC SPRING 2016

By Gene Shiue

Outline

-Application Security

-Malware

-Security Policy/Isolation

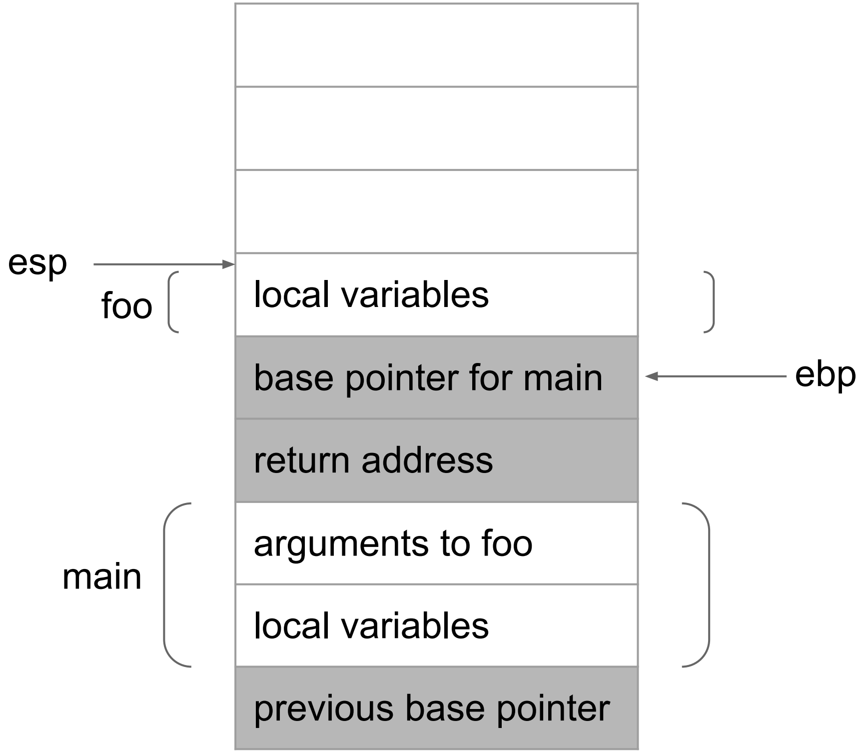
-Web App Security

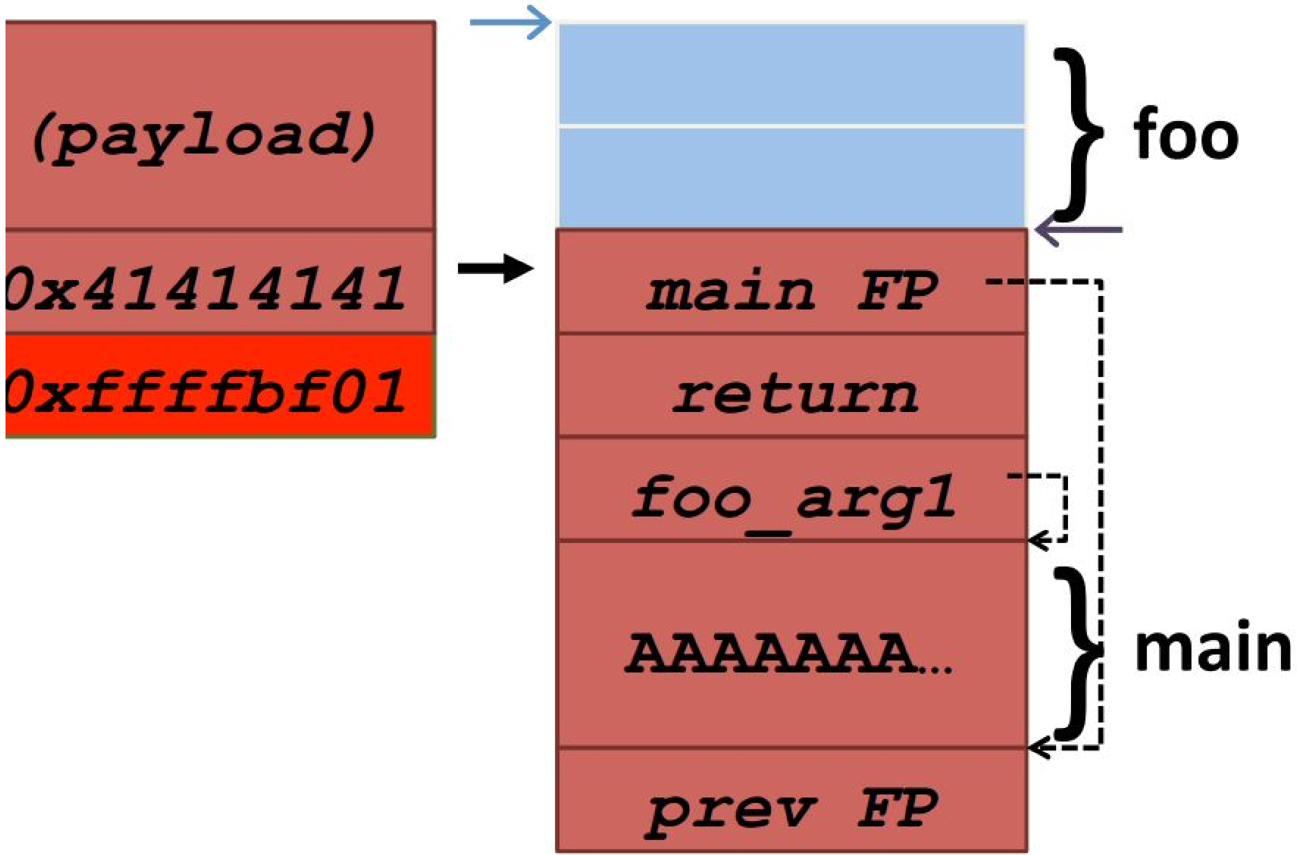
-Authentication

Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*
4. Do stuff in *foo*





(payload)
0x41414141
0xffffbf01
foo
main FP
return
foo_arg1
AAAAAAA...
main
prev FP

What to do?

Bounds checking:

strcpy, gets vs strncpy, fgets

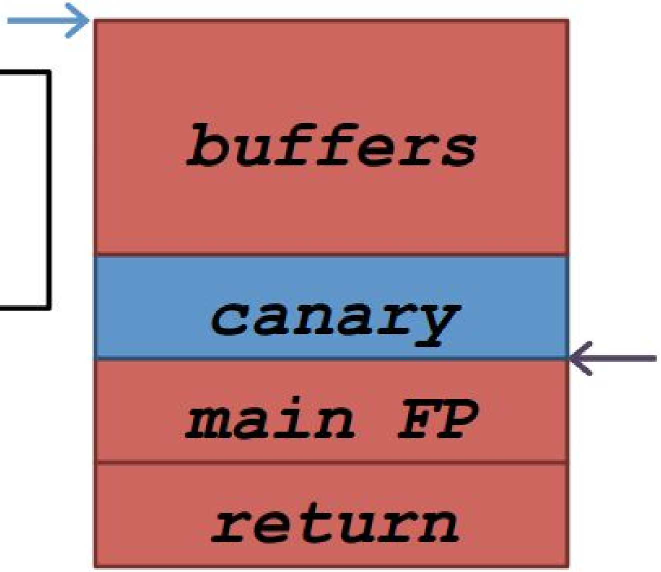
Defenses

Stack Canary

Stack canaries

```
# on function call:

canary = secret
```



Defenses

Stack Canary

DEP

No eXecute (aka W^X aka DEP aka…)

- Mark pages as EITHER
  - Read/write (stack/heap)
  - Executable (.text/code segments)
  - (never both)
- Requires hardware support
- Attacker cannot return to stack

Return-Oriented Programming

```
8057360:    5a       pop    %edx
8057361:    59       pop    %ecx
8057362:    5b       pop    %ebx
8057363:    c3       ret

8055060:    8b 01    mov    (%ecx),%eax
8055062:    89 02    mov    %eax,(%edx)
8055064:    89 d0    mov    %edx,%eax
8055066:    c3       ret
```

(original return addr)	0x8057360
	0xbfff0000(edx)
	0xbfff3230(ecx)
	0x12341234(ebx)
	0x8055060
	Next Gadget

Defenses

Stack Canary

DEP

ASLR

Address Space Layout Randomization

- Virtual Address Space: 4GB+
- Stack/code size: ~10 MB
- Randomize offsets

Some other attacks:

Integer overflow

```
void foo(int *array, int len) {
  int *buf;
  buf = malloc(len * sizeof(int));
  if (!buf)
     return;

  int i;
  for (i=0; i<len; i++) {
     buf[i] = array[i];
  }
}
```

Integer casts

```
void foo(char *array, int len) {
  int buf[100];

  if (len >= 100) {
    return;
  }

  memcpy(buf, array, len);
}
```

1.2.11 Format String Attack

%n

Proto-answer: print malicious_code + padding + ADDR1 + ADDR2 + “%00000x%04$hn%00000x%05$hn”

Malware:

Virus

Trojan Horses

Rootkits

Worm

Adware

Spyware

Defenses Against Malware

Signatures

White/black listing

Heuristic Analysis

Access Control

Mandatory access control - decisions by admin/root

Discretionary access control - decisions by users

Role-based access control

least privilege philosophy

Isolation

Confinement:

- Hardware (different machine)
- Virtual Machine (different OS on same machine)
- Process (system call interposition)

System call interposition

Observation: to damage host system (e.g. persistent changes) app must make system calls:

- To delete/overwrite files: unlink, open, write
- To do network attacks: socket, bind, connect, send

Idea: monitor app's system calls and block unauthorized calls

Implementation options:

- Completely kernel space (e.g. GSWTK)
- Completely user space (e.g. program shepherding)
- Hybrid (e.g. Systrace)

Web App Security:

SQL Injection

- Consider an SQL query where the attacker chooses $city:

```
SELECT * FROM `users` WHERE location='$city'
```

- What can an attacker do?

$city = "Ann Arbor'; DELETE FROM `users` WHERE 1='1"

```
SELECT * FROM `users` WHERE location='Ann Arbor';
DELETE FROM `users` WHERE 1='1'
```

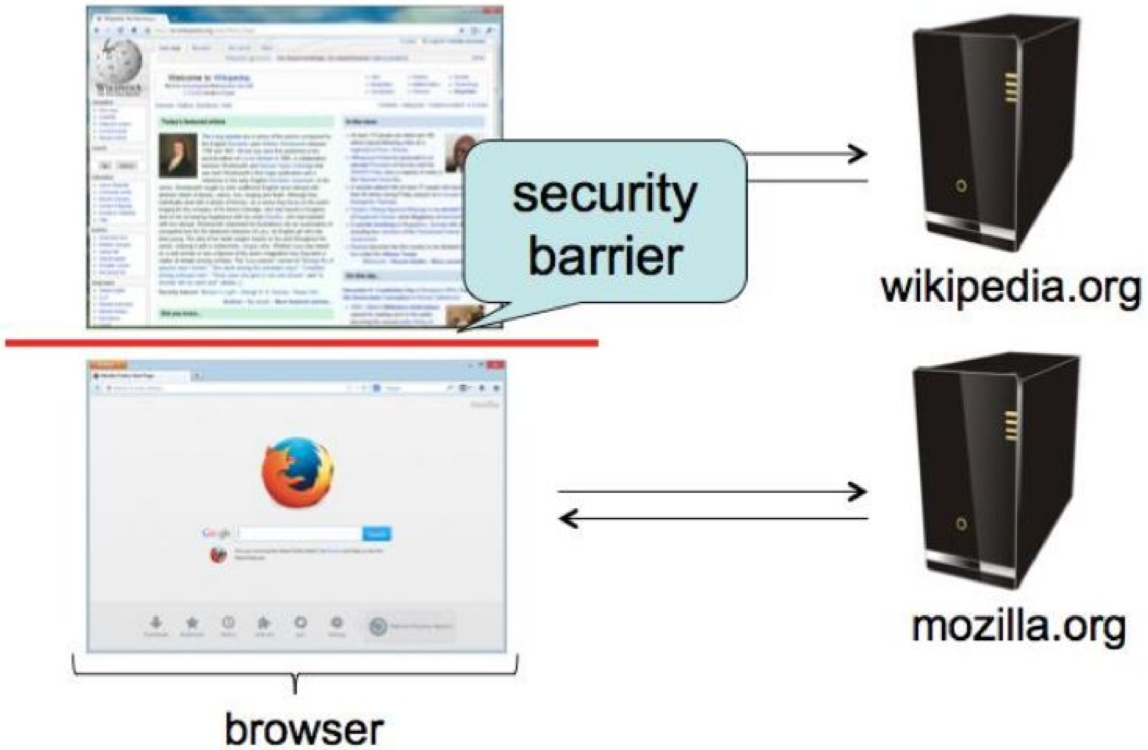
SQL Injection Defense

- Make sure **data** gets interpreted as **data**!
  - Basic approach: escape control characters (single quotes, escaping characters, comment characters)
  - Better approach: Prepared statements – declare what is data!

```
$pstmt = $db->prepare(
 "SELECT * FROM `users` WHERE
location=?");
$pstmt->execute(array($city)); // Data
```

Same-origin policy

- Each site is isolated from all others



Same-origin policy

- Granularity of protection: the *origin*
- Origin = protocol + hostname (+ port)

- Javascript on one page can read, change, and interact freely with all other pages from the same origin

Same-origin policy

- Browsers provide isolation for JS scripts via the Same Origin Policy (**SOP**)
- Simple version:
  - Browser associates web page elements (layout, cookies, events) with a given **origin** ≈ web server that provided the page/cookies in the first place
    - Identity of web server is in terms of its hostname, e.g., bank.com
- SOP = *only scripts received from a web page's origin have access to page's elements*
- **XSS: Subverting the Same Origin Policy**

Cross-site Request Forgery (CSRF)

Click me!!!
http://bank.com/transfer?to=badguy&amt=100

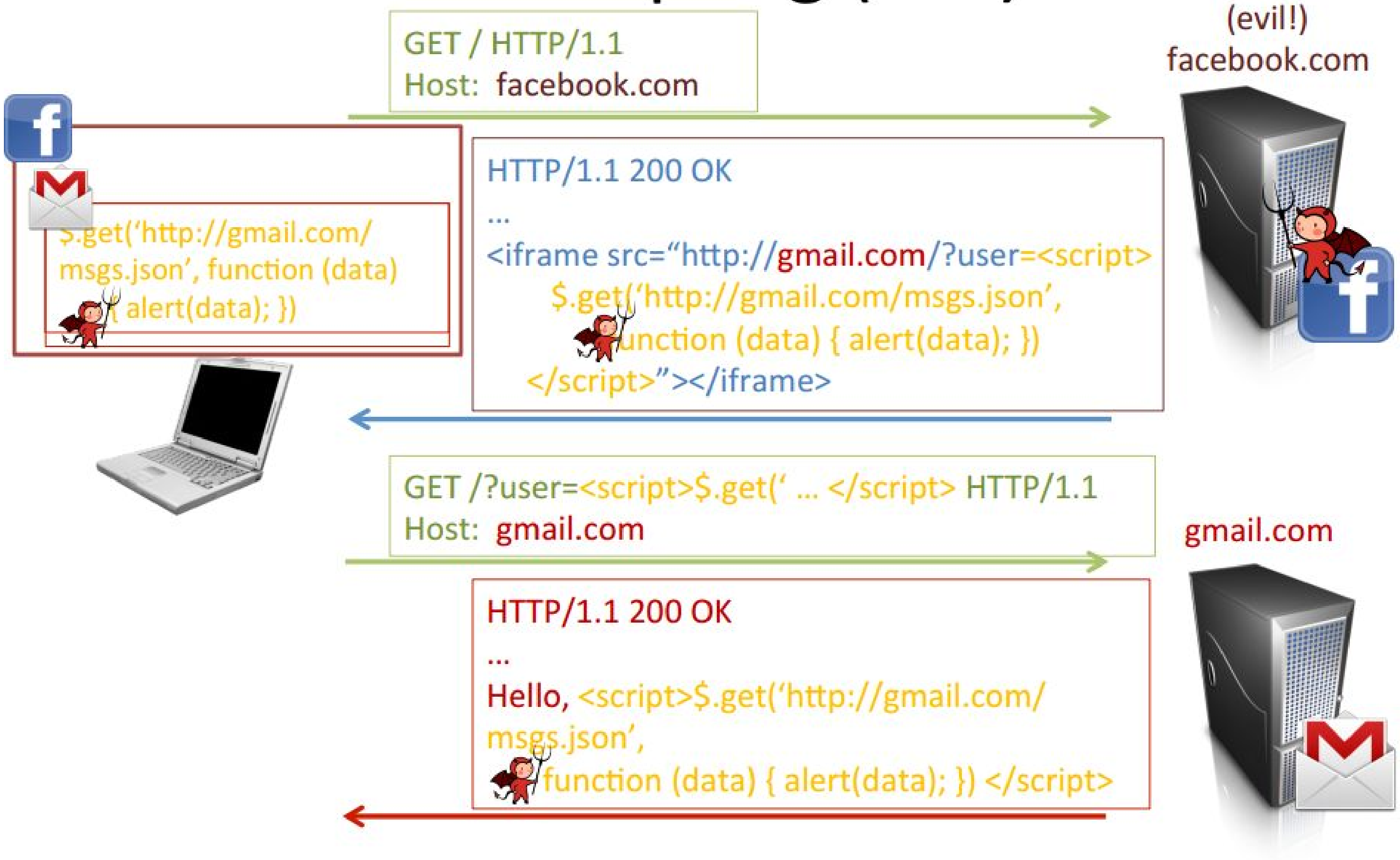
fde874 = bob

bank.com

GET /transfer?to=badguy&amt=100 HTTP/1.1
Host: bank.com
Cookie: login=fde874

HTTP/1.1 200 OK
….
Transfer complete: -$100.00

Cross-Site Scripting (XSS) Attack



Implementing Bungle

In checkpoint 1 you will

- Construct database to store user and search history information
- Write code which processes user input to SQL queries (connecting frontend and backend)

→ You will use prepared statements to protect against SQL injection

- Implement input sanitization against XSS
- Implement token validation against CSRF

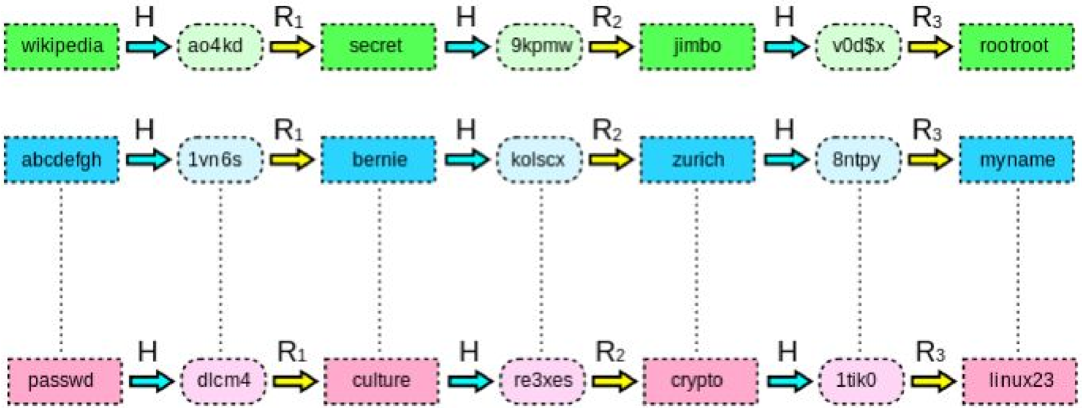
Passwords

From user: key loggers, phishing attack, network attacks

From website: database (plaintext? yes or no?)

Rainbow tables

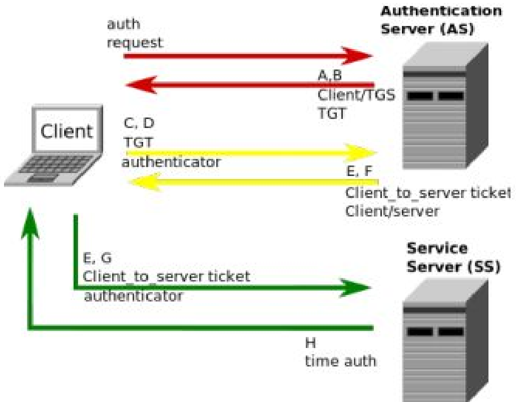
- Similar to a lookup table
- Attacker(s) can trade-off disk-space vs. CPU time
  - Recovered **90%** of **6.5M** LinkedIn passwords in **6 days**



Defense: add salt!!

Network authentication

- User sends password
  - Hopefully over encrypted channel (TLS/SSH)
- Challenge-based authentication
  - Server sends challenge (nonce)
  - User sends response (H(password, nonce))
- Kerberos



Multi-factor Authentication

Something you know, something you have, something you are

Examples?