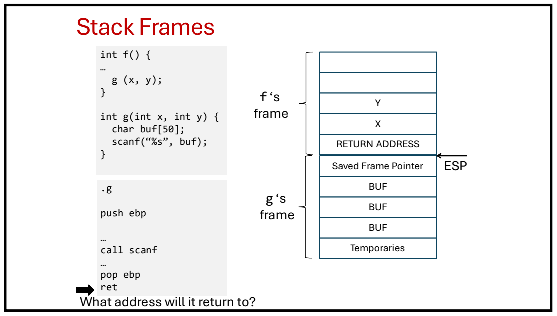
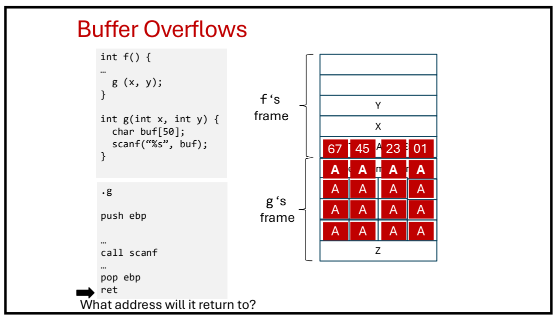
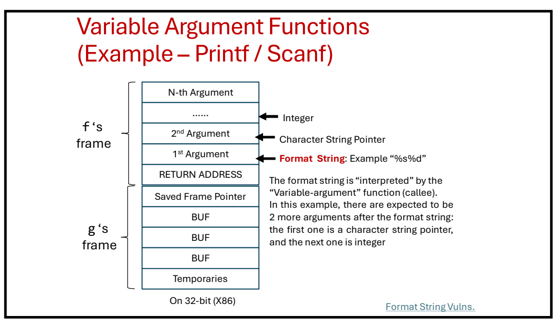

Stack Frames
int f() {
…
g (x, y);
}
int g(int x, int y) {
char buf[50];
scanf("%s", buf);
}
.g
push ebp
…
call scanf
…
pop ebp
ret
What address will it return to?
f 's frame
g 's frame
Y
X
RETURN ADDRESS
Saved Frame Pointer
BUF
BUF
BUF
Temporaries
ESP

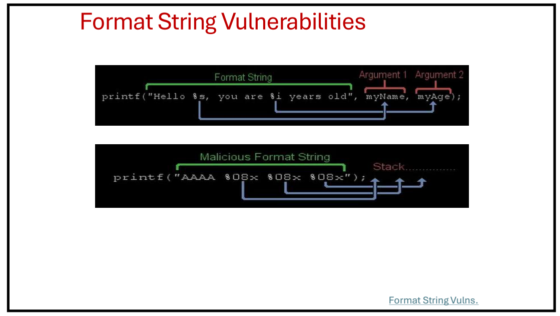
1


Buffer Overflows
int f() {
…
g (x, y);
}
int g(int x, int y) {
char buf[50];
scanf("%s", buf);
}
.g
push ebp
…
call scanf
…
pop ebp
ret
What address will it return to?
f 's frame
g 's frame
Y
X
67 45 23 01
A A A A
A A A A
A A A A
A A A A
Z

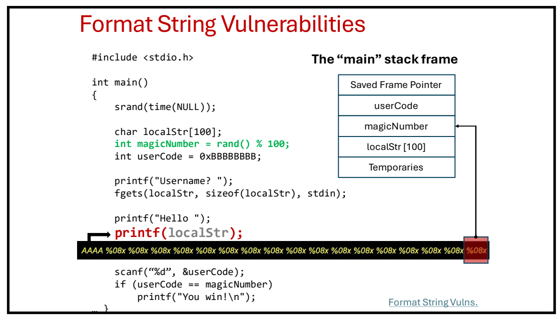
2


Variable Argument Functions
(Example – Printf / Scanf)
f 's frame
g 's frame
N-th Argument
……
2nd Argument
1st Argument
RETURN ADDRESS
Saved Frame Pointer
BUF
BUF
BUF
Temporaries
On 32-bit (X86)
Integer
Character String Pointer
Format String: Example "%s%d"
The format string is "interpreted" by the "Variable-argument" function (callee).
In this example, there are expected to be 2 more arguments after the format string: the first one is a character string pointer, and the next one is integer
Format String Vulns.

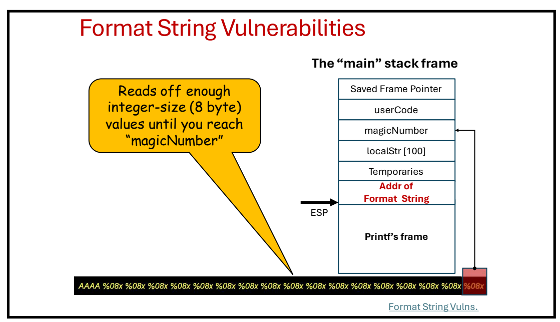
3


Format String Vulnerabilities
Format String
Argument 1
Argument 2
printf("Hello %s, you are %i years old", myName, myAge);
Malicious Format String
Stack...............
printf("AAAA %08x %08x %08x");
Format String Vulns.

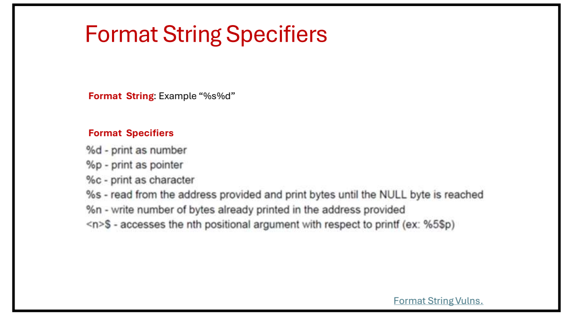
4


Format String Vulnerabilities
#include <stdio.h>
int main()
{
srand(time(NULL));
char localStr[100];
int magicNumber = rand() % 100;
int userCode = 0xBBBBBBBB;
printf("Username? ");
fgets(localStr, sizeof(localStr), stdin);
printf("Hello ");
printf(localStr);
AAAA %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x
scanf("%d", &userCode);
if (userCode == magicNumber)
printf("You win!\n");
… }
The "main" stack frame
Saved Frame Pointer
userCode
magicNumber
localStr [100]
Temporaries
Format String Vulns.

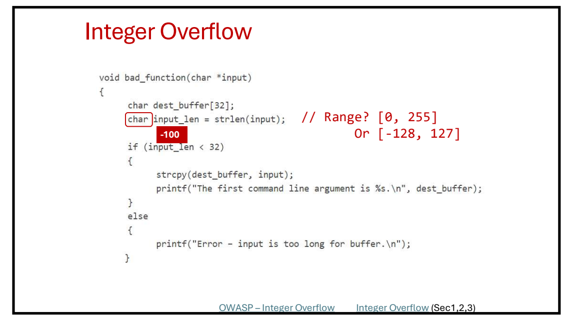
5


Format String Vulnerabilities
Reads off enough integer-size (8 byte) values until you reach "magicNumber"
The "main" stack frame
Saved Frame Pointer
userCode
magicNumber
localStr [100]
Temporaries
Addr of Format String
ESP
Printf's frame
AAAA %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x %08x
Format String Vulns.

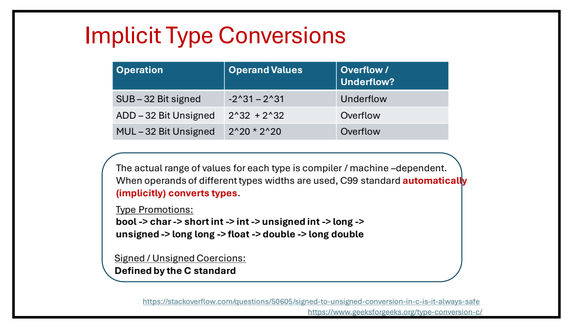
6


Format String Specifiers
Format String: Example "%s%d"
Format Specifiers
%d - print as number
%p - print as pointer
%c - print as character
%s - read from the address provided and print bytes until the NULL byte is reached
%n - write number of bytes already printed in the address provided
<n>$ - accesses the nth positional argument with respect to printf (ex: %5$p)
Format String Vulns.

7


Integer Overflow
void bad_function(char *input)
{
char dest_buffer[32];
char input_len = strlen(input);
-100
// Range? [0, 255]
Or [-128, 127]
if (input_len < 32)
{
strcpy(dest_buffer, input);
printf("The first command line argument is %s.\n", dest_buffer);
}
else
{
printf("Error - input is too long for buffer.\n");
}
OWASP – Integer Overflow
Integer Overflow (Sec1,2,3)

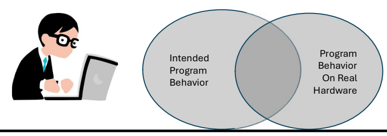
8


Implicit Type Conversions
Operation
Operand Values
Overflow / Underflow?
SUB – 32 Bit signed
-2^31 – 2^31
Underflow
ADD – 32 Bit Unsigned
2^32 + 2^32
Overflow
MUL – 32 Bit Unsigned
2^20 * 2^20
Overflow
The actual range of values for each type is compiler / machine –dependent.
When operands of different types widths are used, C99 standard automatically (implicitly) converts types.
Type Promotions:
bool -> char -> short int -> int -> unsigned int -> long -> unsigned -> long long -> float -> double -> long double
Signed / Unsigned Coercions:
Defined by the C standard
https://stackoverflow.com/questions/50605/signed-to-unsigned-conversion-in-c-is-it-always-safe
https://www.geeksforgeeks.org/type-conversion-c/

9

Summary & Key Takeaways

- Memory Errors / Vulnerabilities
  - Spatial & Temporal
- Worst case: Can give attackers capability to read / write any value anywhere in memory
- Hardware does not give memory safety



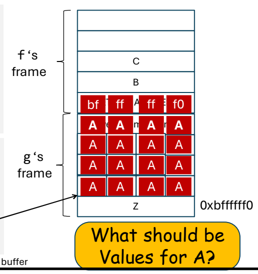
10

Code Injection Example

```
int f() {
…
  g (x, y);
}

int g(int x, int y) {
  char buf[50];
  scanf("%s", buf);
}
```

```
.g
push ebp
…
call scanf
…
pop ebp
ret
```

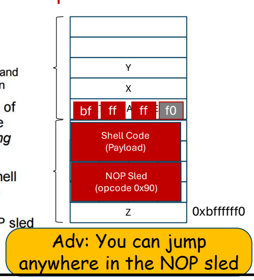
Will return inside attacker's buffer



11

Code Injection Example

- Instruction NOP, No Operation.
  - Tell CPU to do nothing and fetch the next instruction
- Including a large block of NOP instructions in the injected code as *landing area*
- Execution will reach shell code as long as return address pointing to somewhere in the NOP sled



12

Control-flow Hijacking: Code Injection

- Control-oriented a.k.a control-flow hijacking
- Outcome 1: Code Injection
  - ***Definition:*** *A memory exploit that hijacks control to jump to attacker's data payload*

13

Code Injection: Requirements

- Req 1: Write Attack Payload in memory
- Req 2: Have Attack Payload Be Executable
- Req 3: Divert control-flow to payload

14

Control-oriented Exploits (II): Code Reuse

- Outcome 2: Code Reuse
  - ***Definition:*** *A memory exploit that hijacks control to jump to attacker's controlled code address*
- Requirements for Code Reuse
  - ~~Req 1: Write Attack Payload in memory~~
  - Req 2: Have Attack Payload Be Executable
  - Req 3: Divert control-flow to payload
- Insight: Re-use the existing code as payload

15

Code Reuse: The Idea

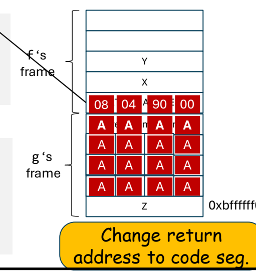
- Attacker hijacks control flow
- Jumps back to the code segment
- Example: Return-to-libc

16

Code Reuse Attack: Return-to-libc

```
.execv <0x8049000>
push ebp
… // read parameter1
… // read parameter2
pop ebp
ret
```

```
.g
push ebp
…
call scanf
…
pop ebp
ret
```

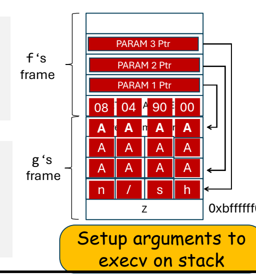


17

Code Reuse Attack: Return-to-libc

```
.execv <0x8049000>
push ebp
… // read parameter1
… // read parameter2
… // read parameter3
pop ebp
ret
```

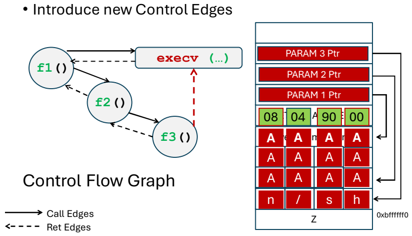
```
.g
push ebp
…
call scanf
…
pop ebp
ret
```



18

Return-to-Libc

- Introduce new Control Edges



Control Flow Graph

19

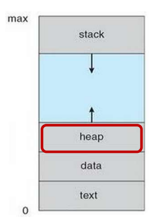
Code Reuse Exploits

- Code Reuse
  - ***Definition:*** *A memory exploit that hijacks control to jump to attacker's controlled* ***code address***
- Requirements for Code Reuse
  - Req 1: Have Attack Payload Be Executable
  - Req 2: Divert control-flow to payload
- Can be even more advanced:
  - Return-oriented programming (optional)

20

Dynamic Memory Management

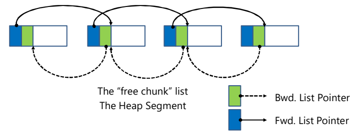
```
int g(int x, int y) {
  char* buf;

  buf = malloc (50);
  scanf("%s", buf);

  free (buf);
}
```



21

Heap Organization

- The glibc (std. C library) manages memory allocations in C
- Stores each **allocated** memory chunk in a linked list
- Stores each **unallocated** memory chunk in a linked list
- Can re-allocate a previously freed chunks
- Requests the OS for VA pages when all memory is allocated

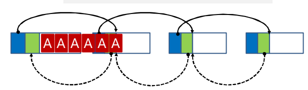


The "free chunk" list
The Heap Segment

22

Heap Overflows

```
int g(int x, int y)
{
  char* buf;

  buf = malloc (50);
  scanf("%s", buf);

  free (buf);
}
```



The 'allocated chunk' list

23

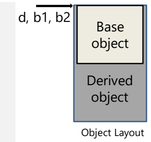
(Remark) Data-oriented Attacks

- Do you need to write to any **control data** for an exploit?
  - No!
- Requirements for Data-oriented attacks
  - ~~Req 1: Write Attack Payload in memory~~
  - ~~Req 2: Have Attack Payload Be Executable~~
  - ~~Req 3: Divert control-flow to payload~~
- Insight: Simply manipulate **non-control data**

24

Background: Upcasting with Single Inheritance

```
class Base {…};
class Derived : public Base {…}

Derived* d = new Derived();

//Implicit cast
Base* b1 = (Base *)&d;

//Explicit cast
Base* b2 = static_cast<Base*>(&d);
```



Always Safe to upcast in C++

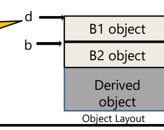
25

Background: Downcasting

```
class B1 {…}; class B2 {…};
class Derived: public B1, public B2 {…}

int foo (B2* b) {
   Derived* d = static_cast<Derived*>(b); //downcast
}
```

You're assuming that b pointed to a Derived object

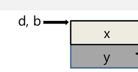
Not always safe!



26

Safe Downcasts

```
class Base {int x};
class Derived : public Base {int y;}
int main() {
   Derived* d1= new Derived(); // Derived type
   foo(d1);
}

int foo (Base* b) {
   Derived* d = static_cast<Derived*>(b); // Ok cast
   d->y = 0;
}
```

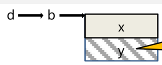


'y' is allocated at offset sizeof (int) from d

27

Bad casts / Type Confusion

```
class Base {int x};
class Derived : public Base {int y;}
int main() {
    Base* b= new Base(); // object of Base type
    foo(b);
}

int foo (Base* b) {
    Derived* d = static_cast<Derived*>(b); //Bad cast
    d->y = 0;
}
```



'y' isn't allocated (Not in Scope)

28

Lifetime & Scope of Variables

- **Scope**:
  - Region of code where a variable can be accessed
  - E.g. Global, Function-local, Heap (dynamic)
- **Lifetime**:
  - Portion of program execution during which storage is guaranteed
  - E.g. Auto vs. static

… are Programming Language Abstractions
  - Not instruction set / hardware abstractions

```
1. int z=0;
2. int g(int x, int y) {
3.     char* buf;
4.     buf = malloc (50);
5.     scanf("%s", buf);
6.     free (buf);
7.      {int w=0; … }
8. }
```

Variable	Scope	Valid Lifetime
z	Global, Line 1-8	Throughout the program runtime
x	Local, Line 3-7	Execution of g
y	Local, Line 3-7	Execution of g
w	Local, Line 7	Execution of block at Line 7
buf	Local, Line 3-7	Execution of g
*buf	Heap, Line 4-7	Line 5-6
"%s"	Constant Literal, Line 5	Execution of Line 5

29

Temporal Memory Error

```
int foo()
{
int *p=NULL;
        {
          int x = 5;
          p = &x;
        }
return *p;
}
```

What will this program return?

Answer:
Undefined behavior as per C11 standard

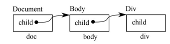
Why?
- The lifetime of x is within the inner block {}
- The compiler can choose to remove the storage for "x"
- The pointer p is in scope at the last line
- The pointed-to object, however, is accessed out of scope!

Temporal Memory Error:
When program accesses memory beyond its valid lifetime!

30

Temporal Memory Errors
Example: Use-after-free

```
class Div: Element;
class Body: Element;
class Document {
  Element* child;
};

// (a) memory allocations
Document *doc = new Document();
Body *body = new Body();
Div *div = new Div();

// (b) using memory: propagating pointers
doc->child = body;
body->child = div;

// (c) memory free: doc->child is now dangled
delete body;

// (d) use-after-free: dereference the dangled pointer
if (doc->child)
    doc->child->getAlign();
```



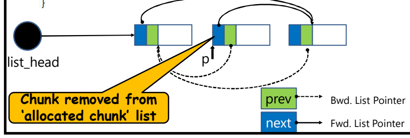
Temporal error:
Used after valid lifetime

Dangling Pointers [NDSS'15]

31

Remove a "chunk" from linked list

```
void delete_from_list (struct chunk * p) {
  if (!p) return NULL;
  if (p->next) p->next->prev = p->prev;
  if (p->prev) p->prev->next = p->next;
  else free_list_head = p->next;
}
```



Chunk removed from 'allocated chunk' list

32

Temporal Memory Error: Double free

```
char* p, q;
p = (char*) malloc (SIZE);
        if (abrt)
        {
            free(p);
        }
q = (char*) malloc (2 * SIZE);
strncpy (q, ext_input, 2*SIZE);
…
free(p);
```

Can point to same chunk as p

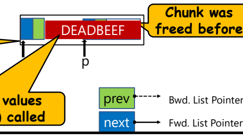
Accessed variable outside the lifetime

33

Double Free Bugs

free (p) twice!

```
void delete_from_list (struct chunk * p) {
  if (!p) return NULL;
  if (p->next) p->next->prev = p->prev;
  if (p->prev) p->prev->next = p->next;
  else free_list_head = p->next;
}
```



p's chunk reallocated to q's chunk

Chunk was freed before

Can have arbitrary values when second free(p) called

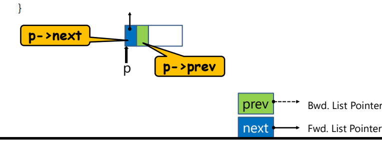
34

Double Free: The Write-Anything-Anywhere Exploit

```
void delete_from_list (struct chunk * p) {
  if (!p) return NULL;
  if (p->next) p->next->prev = p->prev;
  if (p->prev) p->prev->next = p->next;
  else free_list_head = p->next;
}
```

anywhere

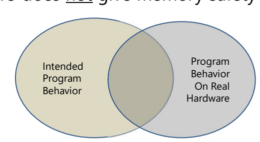
anything



35

Key Takeaways & Summary

- Memory Vulnerabilities: Spatial vs. Temporal
- Exploit Types:
  – Control-flow hijacking vs. Data-oriented
- C/C++/Hardware does not give memory safety



36

Spatial Safety

1. Associate bounds with each pointer (or object)
2. Check each pointer (or object) before access

Recall: Pointers can be incremented, type casted, etc.

- Where to keep the bounds metadata?
  - Within **fat** pointers | Pointer | [start, end] |
  - Within **referent** objects | Object | [start, end] |
    - Use shadow memory, i.e., Metadata: $ObjAddr \rightarrow Bounds$ [e.g. JK-Tree], baggy BC [BB]

Assume: Metadata can't be corrupted

37

Spatial Safety: Fat Pointers

- Where to keep the bounds metadata?
  - Within **fat** pointers | Pointer | [start, end] |
- Track bounds on allocation
- Each dereference checks if pointer $\in [start, end]$
- Do **no checks** on pointer arithmetic
- Pointers of different size-types have diff. bounds
- Do **no updates or checks** on <u>unsafe</u> type casts
- **Safe type casts** require update to pointer bounds
  - Compiler sets the bounds based on the target type

Softbound

38

Summary of Fat Pointers

- **Complete** Spatial Safety
- Code compiled for normal pointers will crash when passed fat pointers
- Object Layouts don't change
- Each pointer **dereference** operation needs a check
  - No check on pointer arithmetic
  - No check on type casts

39

Temporal Safety: NULLIFY pointers

1. Track creation and destruction of pointers
2. Ensure: De-allocated pointers are not accessed

- Idea #1: When you free a pointer, set it to NULL

```
char* p, q;
p = (char*) malloc (SIZE);
        if (abrt)
        {
                free(p); p =NULL;
        }
q = (char*) malloc (2 * SIZE);
strncpy (q, ext_input, 2*SIZE);
...
free(p);
```

No double free

40

Temporal Safety: NULLIFY pointers

1. Track creation and destruction of pointers
2. Ensure: De-allocated pointers are not accessed

- Idea #1: When you free a pointer, set it to NULL

```
char* p, q, r;
p = (char*) malloc (SIZE);
r = (char*) p;
        if (abrt)
        {
                free(p);
        }
q = (char*) malloc (2 * SIZE);
strncpy (q, ext_input, 2*SIZE);
free(r);
```

Double free again!

p := NULL

41

Temporal Safety: Lock and Key Mechanism

1. Track creation and destruction of pointers
2. Ensure: De-allocated pointers are not accessed

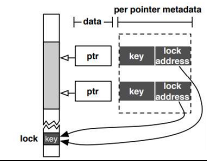
- CETS: Lock-and-key [CETS]
  - Lock & Key values match *<u>if and only if</u>* memory object is temporally safe to access via this pointer

Provides complete temporal safety! (if program has spatial safety)

42

Temporal Safety: Lock and Key Mechanism

- CETS: Lock-and-key [CETS]
  - Associate a (key, lock) pair with each pointer

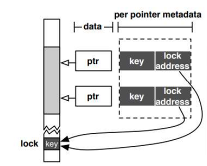


Assume: Metadata can't be corrupted spatially

43

Temporal Safety: Lock and Key Mechanism

- CETS: Lock-and-key [CETS]
  - Associate a (key, lock) pair with each pointer
  - Key: Unique value for each object allocated



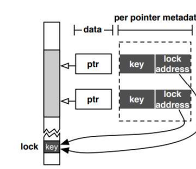

```
ptr = malloc(size);
ptr_key = next_key++;
ptr_lock_addr = allocate_lock();
*(ptr_lock_addr) = ptr_key;
freeable_ptrs_map.insert(ptr_key, ptr);
```

Fresh value

44

Temporal Safety: Lock and Key Mechanism

- CETS: Lock-and-key [CETS]
  - Associate a (key, lock) pair with each pointer
  - Pointer created from another inherit (key, lock) value



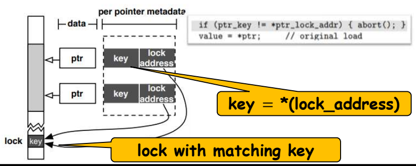

```
newptr = ptr + offset;  // or &ptr[index]
newptr_key = ptr_key;
newptr_lock_addr = ptr_lock_addr;
```

New Pointers Inherit values

45

Temporal Safety: Lock and Key Mechanism

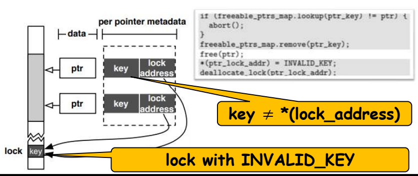
- CETS: Lock-and-key [CETS]
  - Associate a (key, lock) pair with each pointer
  - On **deference**, check pointer's key matches its lock



46

Temporal Safety: Lock and Key Mechanism

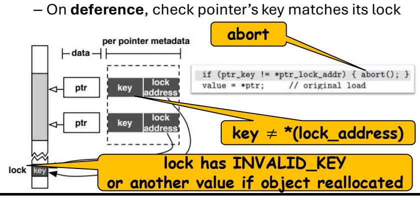
- CETS: Lock-and-key [CETS]
  - Associate a (key, lock) pair with each pointer
  - On **deallocation**, change lock value in the mem. obj.



47

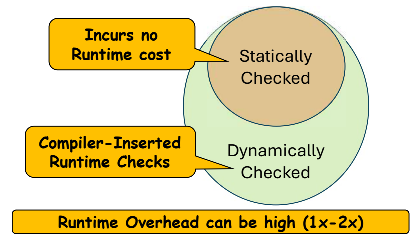
Temporal Safety: Lock and Key Mechanism

- CETS: Lock-and-key [CETS]
  - Associate a (key, lock) pair with each pointer
  - On **deference**, check pointer's key matches its lock



48

Static vs. Dynamic Enforcement



49

Some Unsafe C Lib Functions

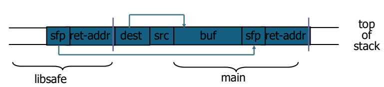
- `strcpy(char *dest, const char *src)`
- `strcat(char *dest, const char *src)`
- `gets(char *s)`
- `sprintf(conts char *format, … )`

- Safe versions:
- `strncpy, strncat, fgets, snprintf`

50

Security Extension -- Libsafe

- Idea: Making unsafe functions safe!
  - Intercepts calls to strcpy (dest, src)
    - Validates sufficient space in current stack frame: |frame-pointer – dest| > strlen(src)
    - If so, does strcpy. Otherwise, terminates application.



51

Code Checking Tools

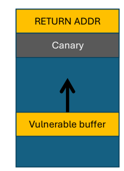
- Tools checking for vulnerabilities using static source code analysis
  - Address Sanitizer
  - ITS4 (It is the Software, Stupid --- Security Scanner)
  - RATS (Rough Auditing Tool for Security)
  - Flawfinder

52

Stack Canaries

- Secret Data values
  - To protect corruption of nearby data.
- Check: the random canary value is OK at ret

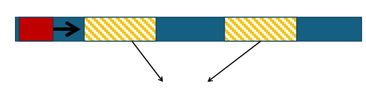
```
.g

push 0x63873633
…
call scanf
…
cmp [esp], 0x63873633
jnz .abort
ret
```



53

Guard Pages

- Defense: Guard Pages
  - Certain pages with NR,NW,NX inserted
- Assumption:
  - Attacker can only write linearly
  - All written values are not used in dereferences



54

Non-executable Data / DEP

- Defense: DEP (a.k.a W ⊕X)
  - Setting regions of memory non-executable
  - Use NX bit
- Defense Goal:
  - Prevents Foreign code Injection
- Blocks Requirement 2 of the Attack
  - "Need to have payload executable"

55

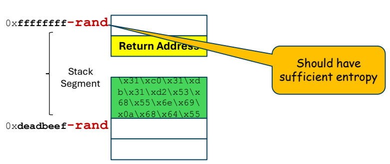
Address Space Layout Randomization (ASLR)

- Assumption:
  - Attacker can write arbitrary places
- Defense Goal:
  - Attacker can't predict location accessed in attack
- Mechanism:
  - At load time, randomize stack, code, bss, etc.
  - Randomize heap location at runtime

56

ASLR: Address Space Layout Randomization

- Goal: "Randomize address space layout"
  - Attacker can't predict location accessed
- Randomizes the base of each segment

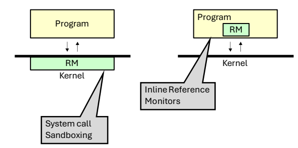


57

Reference Monitors

Reference Monitor: A piece of code that *checks all* **references** to an **object**

Syscall Sandbox: A reference monitor for protecting OS resource objects from an app



Slide from Shmatikov et. al.

58

Policy Design Principle: Allow-listing > Block-listing

- Allow-listing vs. Block-listing in Policies
  - Better to Specify what's allowed
  - Rather than Specify what's not allowed
- Block-listing: E.g. No exec-after-read
- Allow-listing: E.g. `seccomp()` allows 4 syscalls!

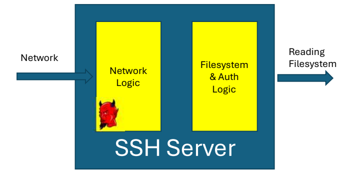
59

3 Security Principles

- Separation of Concerns:
  - Separate the **policy** from its **enforcement**
- Minimize Trusted Code Base (TCB)
  - Reduce what one needs to trust
  - Separate **verifier** from the **enforcement**
- Least Privilege
  - Give each component only the privileges necessary

60

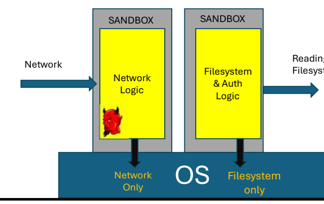
Problem: Bundling of Functionality



61

Principle of Least Privilege

- Each compartment gets the least set of privileges it needs for its function



62

So, what should we do?

- Auto-patching
  - E.g. Google Chrome
- Consider Firefox: Single- process
  - 1 Vulnerability leads to accessing all origins
- Solution: Privilege Separation
  - Compartmentalize & assign least privilege
- Google Chrome
  - Goal: Separate Filesystem from web code

63

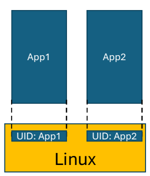
Access Control Primitives

- Definitions:
  - Resource Objects
    "Elements that need to be protected"
  - Authorities or Principals
    "Subjects accessing the resources"
  - Permissions
    "Access Rights"
  - Isolation Environment (or protection domain)
    "A domain in which program executes. It determines what the program will do."

64

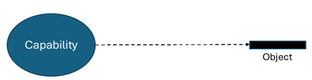
Isolation Environment

- Isolation via OS Processes
- Why is it better?
  - E.g. Apple iOS browser bug
    - Safari exploit [Miller'08]
    - Lead to compromising the Whole phone!
  - On Android, confined to browser app (UID) only!



65

Capabilities



- A **Capability** is a (pointer, metadata) tuple, which
  - Created or modified **only** by querying the security monitor
  - **Sufficient and necessary** to have to access the object
  - Has access rights embedded in it, enforced by monitor
- Capability must be **Unforgeable**:
  - Can't manufacture without explicitly getting it.

[RISC-V/ARM CHERI]

66

Removes Ambient Authorities

Capability 1 Capability 2

`$ cp < foo.txt > bar.txt`

The "cp" program has no authority, by default.
It can only use "capabilities" it is given (e.g. UNIX file handles)

If "cp" compromised, only capabilities given to it are affected

Reference: First 20 minutes of Object Capabilities for Security

67

Access Control Lists vs. Capabilities

Access Control	Capabilities
• Pre-specified policy, implemented centrally by the security monitor	• No pre-specified policy of who is allowed to access what, i.e., can follow the natural flow of access rights
• Access rights can change, checks use the latest rights	• Revoke some capabilities when access rights change
• Ambient Authority	• No ambient authority

• Assumption:
  • Complete mediation:
    • No missing access checks
	• Assumption:
	  – Unforgeability
	    • Capabilities must not be leaked or forged

68

Policy vs. Enforcement Mechanism

- Enforcement Mechanisms:
  - Process sandboxing (Last Lecture)
  - Inline Reference Monitors (Did not cover)
  - Capabilities (This lecture)
  - Virtualization
  - Hardware-based isolation / Trusted Execution Env.
- But, what checks to enforce?
  - Access Control Policies (coming up next...)

69

69

Discretionary Access Control

- No fixed policy!
- Each owner decides the access rules
- Example: UNIX File Systems

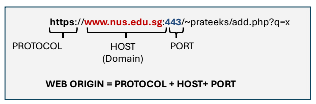
Mandatory Access Control

- Policy fixed by the administrator
- Each owner cannot change access rights of objects created or owned by it

70

Examples of Mandatory AC (I): Same-origin Policy

No direct access between these frames !



1. Same-origin policy [Wikipedia]
2. RFC 6454

71

Examples of Mandatory AC (II): Bell-LaPadula Policy

- Processes & Objects have security labels
- A process of a given label:
  - Cannot read object of higher label (*no-read-up*)
  - Cannot write object of lower label (*no-write-down*)
- Aims to Enforce:
  - Confidentiality

72

Examples of Mandatory AC (III): Biba Policy

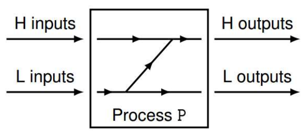
High-Security Label (Trusted)

Low-Security Label (Untrusted)

- Processes & Objects have security labels
- A process of a given label:
  - Cannot write object of higher label (*no-write-up*)
  - Cannot read object of lower label (*no-read-down*)
- Aims to Enforce:
  - Integrity

73

False Positives

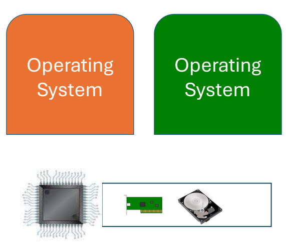


Many apps legitimately:
- Take both H-inputs and L-inputs, and
- Produce both H-outputs and L-outputs

Empirically "Such apps are hard to protect with syscall sandboxing!"

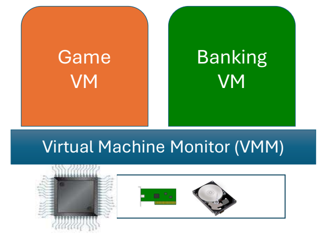
74

Problem: Isolated Computation on Shared CPU



75

Defense(I): Virtualization



76

Assumptions

- Goal: Isolation of Code, data, resources between:
  - Guest VM and Host VMM
  - Between VMs
- Assumptions:
  - Bug-free TCB: Host OS, VMM
  - Malware can affect the guest OS & apps.

77

Security Applications of Virtualization

- Virtual Machine Isolation
  - Red-Green Systems
    - E.g. Banking VM vs. Normal VM
  - Dynamic Analysis / Containment of Malware
- Virtual Machine Introspection
  - E.g. Run an anti-virus in the VMM

78

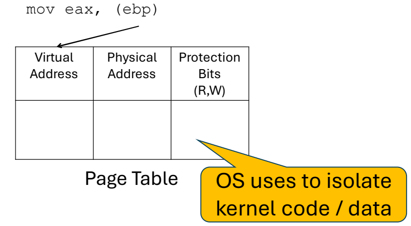
Enforcement Goals for a VMM

- Security VMM Goals:
  - Complete Mediation
  - Trap on all MMU, DMA, I/O accesses
  - Transparency
- Commercial VMM Goals:
  - Performance
  - Compatibility: Run on commodity OSes

79

Compatibility Challenges: An Example

```
mov eax, (ebp)
```



80

VMM Detection: The Red Pill

- Red Pill: "Detects you are virtualized"
- Ways to achieve a "red pill" attack:
  - Commercial VMMs aren't fully transparent
    - E.g. VmWare emulates i440bx chipset (old)
  - Virtualization Timing latencies Measurements
  - Many other measurement channels [HotOS'07]
- Applications of VM Detection:
  - Malware can detect introspection software (e.g. AV)
    - Can utilize "Anti-VM" techniques
  - Benign use: Copy protection by VM duplication

Compatibility is Not Transparency: VMM Detection Myths and Realities

81

The Problem of Covert Channels

- Definition: "An unintended channel of communication between 2 untrusted programs"
- E.g. Shared Cache Latency
  - Sender
    - Send bitval 1: Perform random memory access
    - Send bitval 0: Do nothing
  - Receiver
    - Rcv bitval 1: If long read time for a fixed memory loc.
    - Rcv bitval 0: If short read time for fixed memory loc.
  - Can get 0.02 bits/sec on Amazon EC2 [CCS'09]
  - Many channels: Disk, I/O, Virtualization latency, …

82

Implication on Malware Containment

- In principle, is some containment possible?
  - Yes, When the highest layer of privilege is trusted
  - E.g. the VMM is trustworthy
- Detecting virtualization is easy
  - It can thwart malware analysis (introspection)
- Which containment using VMs is possible: integrity vs. confidentiality?
  - Yes, for Integrity policy – I.e., protecting contained malware from corrupting benign data outside the VM
  - No, for confidentiality, covert channels are a problem

83

Today: The Network Stack

OSI	TCP/IP
Application	Application
Presentation	
Session	Transport
Transport	
Network	Network
Data link	Physical
Physical	

84

Internet Routing

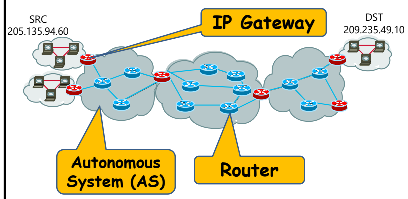


Figure from Boneh et al.

85

Attacker Capabilities

- Strength of the attacker
  - Passive vs. Active
  - Resources: Individual vs. Nation State
- In this lecture, we focus on:
  - **Off-path** attackers
    - Not on the network path between Alice / Bob
  - **On-path** attackers
    - Can access network packets enroute

86

Inter-gateway Routing: BGP

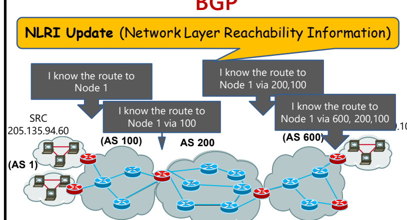


Figure from Boneh et al.

87

Network Attacks (I): Can you spot flaws in BGP?

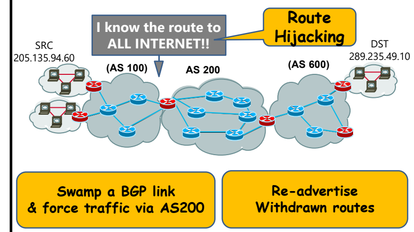


Figure from Boneh et al.

88

How The Internet Routing Works: IP Protocol

Default Route

Default Gateway

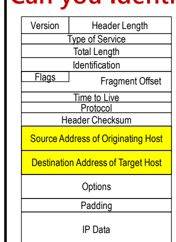
Kernel IP routing table

Destination	Gateway	Genmask	Flags
0.0.0.0	71.46.14.1	0.0.0.0	UG
10.0.0.0	0.0.0.0	255.0.0.0	U
71.46.14.1	0.0.0.0	255.255.255.255	UH
169.254.0.0	0.0.0.0	255.255.0.0	U
172.16.0.0	0.0.0.0	255.240.0.0	U
192.168.0.0	0.0.0.0	255.255.0.0	U
192.168.1.0	192.168.96.1	255.255.255.0	UG
192.168.96.0	0.0.0.0	255.255.255.0	U

Figure from Wikipedia

89

Can you Identify IP Protocol Flaws?



IP Packet Format

- On-path adversary
  - Confidentiality Attacks?
    - Packet sniffing
  - Integrity Attacks
    - IP Data pollution
  - Source IP forgery
    - Replace Source IP with any address
- Off-path adversary
  - Source IP forgery
    - Useful for DDoS
    - Anonymous Infection (e.g. Slammer worm)

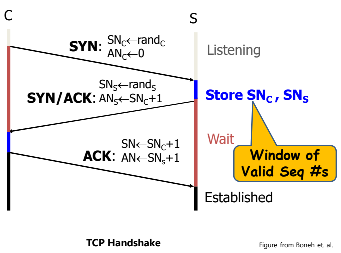
90

How The Internet Works: UDP & TCP

- Unreliable Data delivery over IP: **UDP**
- "Reliable" Data Delivery: **TCP**
  – Connection-oriented, ordered packets

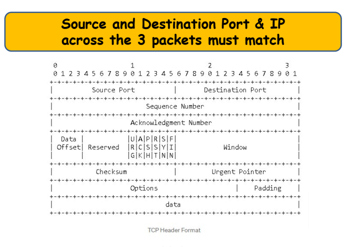
91

TCP Handshake



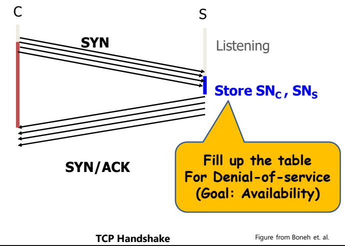
92

TCP Header



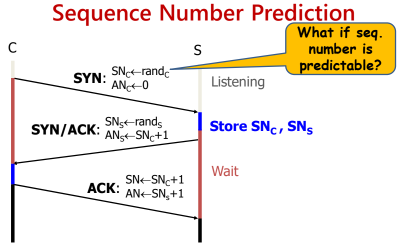
93

Attack (1): SYN Flood



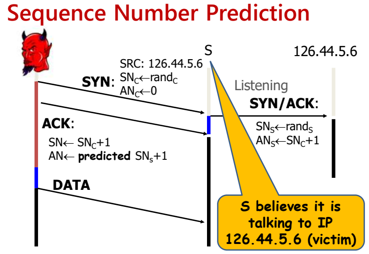
94

Attack (2): Sequence Number Prediction



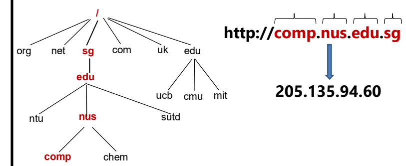
95

Attack (2): Sequence Number Prediction



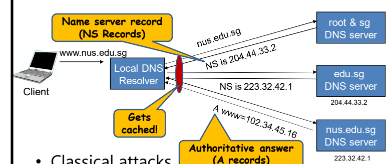
96

Domain Names to IP addresses: DNS



97

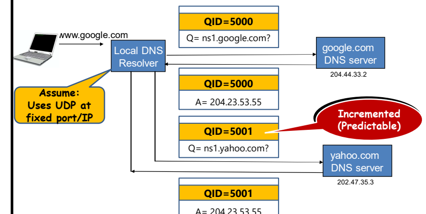
DNS Resolution Protocol



- Classical attacks
  – Modify in-transit DNS responses
  – Find software vulnerabilities in DNS software
- A more advanced (and easy) attack!
- Let's see how you can own *.google.com !

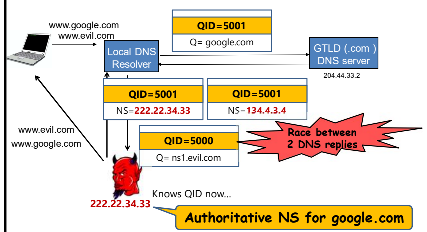
98

DNS Resolution: A Bit of Implementation Detail



99

DNS Cache Poisoning [Kaminsky'08]



100

Firewalls

- Firewalls are tools that control the flow of traffic going between networks.
  - Sitting at border between networks
  - Looking at services, addresses, data, and etc. of traffic
  - Deciding whether a packet should be **allowed or dropped** based on a firewall **policy**
- **Network firewalls** operate at the TCP / IP Level
- **Application-layer firewalls** operate to higher layers
  - Network Intrusion Detections systems (NIDS) and Intrusion Prevention systems (IPS) work on same principle

101

101

Network Firewalls: Stateless Packet Filters

- Applies rules to packets in/out of firewall
  - Based on information in packet header like src/dest IP addr & port, IP protocol, interface
- Typically, a list of rules of matches on fields of n/w packet
  - if match rule says if allow or deny packets
- Decision: **Deny** or **Allow** forwarding of the packets

102

Stateless Packet Filter Rules

- Firewall uses a list of filter rules to decide what to do with a packet
- For a packet, firewall apply the rules starting from the top of the list, and execute the operation specified by the first matching rule
- Sample rule format:

Action	Src Addr	Dst Addr	Protocol	Src Port	Dst Port	Ctrl-Bit

103

103

Stateless Packet Filter Rules: Example

- A company only allows connections to port 80 (HTTP) of external hosts
  - Internal address: 1.2.3.*

Action	Src Addr	Dst Addr	Protocol	Src Port	Dst Port	Ctrl-Bit
Allow	1.2.3.*	*	TCP	*	80	*
Allow	*	1.2.3.*	TCP	80	>1023	ACK
Deny	*	*	*	*	*	*

- Good Design Principle: **Default fail-close** policy
  - Don't open a service to public unless it is necessary

Reference: How to block SQL Slammer Worm with firewall signatures

104

104

Types of Firewalls / NIDS / IPS

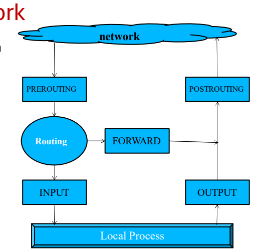
- Traditional / Stateless Packet Filters
  - Applying rules to packets in/out of firewall
  - Based on information in packet header
- Stateful Packet Filters
  - Maintaining a state table of all active connections
  - Filtering packets based on connection states
- Proxy-based or Application Firewalls
  - Understanding application logic
  - Acting as a relay of application-level traffic

105

105

Linux Firewalls: Netfilter Framework

- NetFilter is a packet filtering framework in Linux kernel
- Five hooks to decide the fate of a packet:
  - Prerouting
  - Postrouting
  - Forward
  - Input
  - Output



106

106

Firewall Rules

- Three main components of a firewall rule
  - Hooks to mount the rule
    - Filtering packets for processes on the firewall computer: INPUT, OUTPUT
    - Filtering packets for other computers connected to the firewall: FORWARD
    - Network address translation: PREROUTING, POSTROUTING
  - Conditions
    - IP address, ports, network interface, connection state
  - Actions
    - Drop, reject, change packet information

107

107

The iptables Utility

- The Linux program to maintain firewall rules in the Netfilter framework
- Example: allowing ssh connections to this computer

```
$ sudo iptables -A INPUT -p tcp --dport ssh -j ACCEPT
```

Hook to mount the rule: INPUT

Conditions:
TCP protocol: -p tcp
SSH port: --dport ssh

Action:
Allow

108

108

Firewalls & IDS / IPS: Threat Model

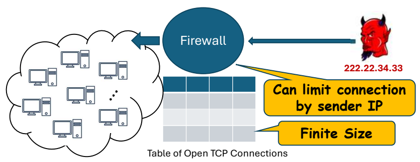
- Goal:
  - Stop attacker's packet from reaching the end application
- Adversary Capability:
  - Adversary can send malicious network packets
  - Adversary is outside the network perimeter
- Assumptions
  - The firewall is uncompromised
  - Defender's policy can tell bad from good traffic by inspecting packet content
  - The firewall sees the same data as end application
  - The network perimeter is correctly defined

109

109

Firewalls & IDS / IPS: Weaknesses of Threat Model

- Defeating firewalls by violating assumptions
- Assumption 1: The firewall is uncompromised
- Attack: **DoS** against the firewall

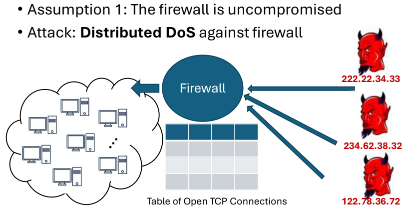


110

110

Firewalls & IDS / IPS: Weaknesses of Threat Model

- Defeating firewalls by violating assumptions
- Assumption 1: The firewall is uncompromised
- Attack: **Distributed DoS** against firewall

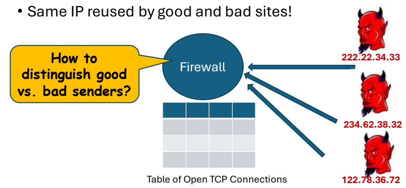


111

111

Firewalls & IDS / IPS: Weaknesses of Threat Model

- Defeating firewalls by violating assumptions
- Assumption 2: Distinguish good vs. bad
- Same IP reused by good and bad sites!



112

112

Firewalls & IDS / IPS: Weaknesses of Threat Model

- Defeating firewalls by violating assumptions
- Assumption 2: Distinguish good vs. bad
- Firewall filters can look packet payloads for
  - **Signatures:** Patterns of known malicious traffic

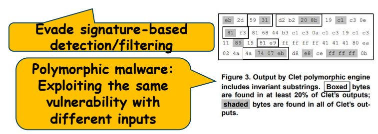


Figure 3. Output by Clet polymorphic engine includes invariant substrings. Boxed bytes are found in at least 20% of Clet's outputs; shaded bytes are found in all of Clet's outputs.

113

Polygraph: Automatically Generating Signatures for Polymorphic Worms (IEEE S&P'05)

113

Firewalls & IDS / IPS: Weaknesses of Threat Model

- Defeating firewalls by violating assumptions
- Assumption 2: Distinguish good vs. bad
- Firewall filters can look packet payloads for
  - **Anomalies:** Patterns that are unusual

Lots of "crud": Corner-case benign traffic that is not malicious

Falsely classified as bad

[Sec 7.2] Bro: A System for Detecting Network Intruders in Real-Time (Computer Networks'99)

114

Firewalls & IDS / IPS: Weaknesses of Threat Model

- What FP rate is considered good?
  - Beware of the "The Base Rate Fallacy"
  - Why does the fallacy arise?
    - Base Rate of True Incidence: 20 / 1,000,000
    - Base Rate of False Incidence: 999,980 / 1,000,000
    - Total TP: $20 * 100\% = 20$
    - Total FP: $999{,}980 * 0.1\% \sim 1000$
    - Of the total alarms, only 1 out of 50 are true incidents

[Sec 6.5] Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 ... (TISSEC'00)

115

Firewalls & IDS / IPS: Threat Model

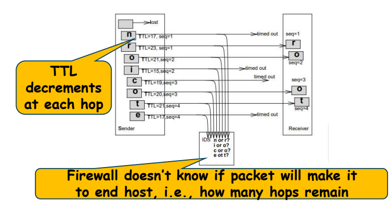
- Assumptions
  - The firewall is uncompromised
  - Defender's policy can tell bad from good traffic by inspecting packet content
  - **The firewall sees the same data as end host**
  - The network perimeter is correctly defined

116

116

Firewalls & IDS / IPS: Weaknesses of Threat Model

- An Example: TCP vs IP streams



Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics

117

117

Firewalls & IDS / IPS: Threat Model

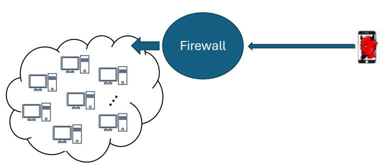
- Assumptions
  - The firewall is uncompromised
  - Defender's policy can tell bad from good traffic by inspecting packet content
  - The firewall sees the same data as end host
  - **The network perimeter is correctly defined**

118

118

Firewalls & IDS / IPS: Weaknesses of Threat Model

- Defeating firewalls by violating assumptions
- Assumption 4: Is the perimeter well defined?
- The "Bring Your Own Device" (BYOD) Problem:



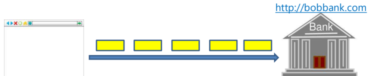
Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics

119

119

Definition: A Secure Channel

http://bobbank.com



A Secure Channel is a data communication protocol established **between 2 programs** which preserves data:
- **C**onfidentiality
- **I**ntegrity
- **A**uthentication

Examples of Secure Channels

- HTTPS
- Encrypted File System
- SSH / VPN
- Others?

* Note that availability is not a goal. So, denial-of-service attacks are permitted by the threat model.
* Integrity is also referred to as "authenticity" sometimes. Not to be confused with "authentication"

120

Basic Cryptographic Primitives

- **C**onfidentiality → **Encryption**
- **I**ntegrity → **MAC**
- **A**uthentication → **Digital Signature**

~~Is achieved by?~~

- We will study the basic crypto building blocks / primitives later
  - Will define what security means carefully
  - Will clarify the adversary's power

121

Setup

$\mathrm{KeySetup}() \rightarrow K$
$\mathrm{Enc}: M \times K \rightarrow C$
$\mathrm{Dec}: C \times K \rightarrow M$

- Correctness:
  - $\forall m, k, Dec(Enc(m,k),k) = m$

randomized algos., prob. defined over choice of keys

- Security:
  - Will define later

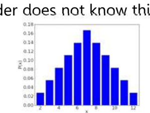
122

Adversary Model

- Algorithms Setup, Enc, Dec are **public**
  - Keys + internal coin flips are **private**

Kerckhoff's principle

- Adversary knows **any** distribution over M from background knowledge
  - Defender does not know this distribution



123

Security Goal

$\mathrm{KeySetup}() \rightarrow K$
$\mathrm{Enc}: M \times K \rightarrow C$
$\mathrm{Dec}: C \times K \rightarrow M$

- Correctness:
  - $\forall m, k, Dec(Enc(m,k),k) = m$
- Security:

$$\boxed{\Pr[Guess = m|c] \leq \frac{1}{2}}$$

What Events? All possible keys

Is this definition a good one?

124

Symmetric Key Encryption: Goal

$\mathrm{Setup}() \rightarrow K$
$\mathrm{Enc}: M \times K \rightarrow C$
$\mathrm{Dec}: C \times K \rightarrow M$

- Correctness:
  - $\forall m, k, Dec(Enc(m,k),k) = m$
- Security Goal: Perfect Secrecy

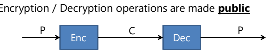
$$\boxed{\Pr[Guess = m|c] = \Pr[Guess = m]}$$

Does not depend on adversary prior knowledge!

125

Modern Cryptosystems

- **Kerckhoffs's principle**:
  - Encryption / Decryption operations are made **public**

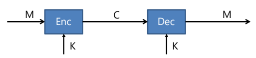


- Cryptosystems are defined w.r.t. to an attacker model.
- Models for Adversary's Capability
  - Ciphertext only Attacker (COA)
    - Given just the ciphertexts, guess the plaintext
  - Known plaintext Attack (KPA)
    - Given ciphertexts for certain plaintexts (not chosen by the adversary)
  - Chosen Plaintext Attacker (CPA)
    - Adversary gets to see ciphertexts for a chosen set of plaintext, then asked to guess for the decryption an unknown ciphertext.
  - Chosen ciphertext Attacker (CCA)
    - Adversary gets to see decryptions of ciphertexts, then asked to guess for plaintexts for an unknown ciphertext

126

One Bit Encryption

- M, K, C are all {0,1}



- How many deterministic functions of Enc type signature exist?
  - 2^4
- Choose keys uniformly at random from {0,1}
- Let's try some familiar candidates for one-bit encryption

127

One Bit Encryption

M	K	C
0	0	0
0	1	0
1	0	0
1	1	1

AND

✖ Correctness
✖ Perfect Secrecy

M	K	C
0	0	0
0	1	0
1	0	1
1	1	1

MESSAGE

✔ Correctness
✖ Perfect Secrecy

M	K	C
0	0	0
0	1	1
1	0	0
1	1	1

KEY

✖ Correctness
✔ Perfect Secrecy

M	K	C
0	0	$
0	1	$
1	0	$
1	1	$

RANDOM

✖ Correctness
✔ Perfect Secrecy

128

One-Time Pad

- Encrypt: c := m⊕ k
  - k is the secret key, chosen uniformly at random
  - k changes for each plaintext message
  - ⊕ is the bitwise XOR operation (addition mod 2)
- Decrypt: m = c ⊕ k

m	k	C = m ⊕ k
0	0	0
0	1	1
1	0	1
1	1	0

129

One-Time Pad is a "perfect" cipher

$\boxed{\Pr[Guess = m|c] = \Pr[Guess = m]}$

Correctness ✔

M	C
0	0
1	1

When $k \leftarrow 0$

M	C
0	1
1	0

When $k \leftarrow 1$

A one-to-one function (bijection)

Perfect Secrecy ✔

Adversary can not distinguish m=0 vs. m=1 (each value of c is equally likely with uniform k)

130

Proof

Let m ∈ {0,1} be with $\mathbf{Pr}[m = 0] = p$ and and $\mathbf{Pr}[m = 1] = (1 - p)$
Note: You can choose p to be any value!
For a one-time pad, what is:
1. $\Pr[\boldsymbol{m} = \mathbf{0} \mid \boldsymbol{c} = \mathbf{0}]$?
2. $\Pr[m = 0 \mid c = 1]$?
3. $\Pr[m = 1 \mid c = 0]$?
4. $\Pr[m = 1 \mid c = 1]$?

Recall: $\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]}$

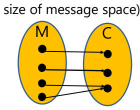
Other 3 cases have the same proof reasoning

Probability of $c = 0$ is defined over choices of key k and of m
- $\Pr[k = 0] = \Pr[k = 1] = \frac{1}{2}$
- $\Pr[\boldsymbol{c} = \mathbf{0}] = \Pr[k = 0 \cap m = 0] + \Pr[k = 1 \cap m = 1] = \frac{p}{2} + \frac{(1-p)}{2} = \frac{1}{2}$
- $\Pr[m = 0 \cap c = 0] = \Pr[m = 0 \cap k = 0] = \frac{p}{2}$
- $\Pr[m = 0|c = 0] = \frac{\Pr[m=0 \cap c=0]}{\Pr[c = 0]} = p$
- Proved that $\underline{\Pr[m = 0|c = 0] = \Pr[m = 0]}$

131

Limitations of Perfect Secrecy

- Looks like we found the perfect cipher…
- Limitations of one-time pad [Shannon'49]:
  - For any cipher to satisfy perfect secrecy
    - $|K| \geq |M|$ (size of key space ≥ size of message space)
- Proof that $|K| \geq |M|$ has **2 parts**:
  - $|M| \leq |C|$
  - $|C| \leq |K|$



- (Part 1): $\boxed{|M| \leq |C|}$
- Why? From **Correctness** property:
  - If we fix any key, then M → C should be invertible
  - So, M → C should be injective (one-to-one).
  - So, $|M| \leq |C|$. (Proof by contradiction)
    - If $|M| > |C|$, figure shows two or more messages map to same c.
    - This contradicts the requirement that M → C is one-to-one

132

Limitations of Perfect Secrecy

- (Part 2) $\boxed{|K| \geq |C|.}$
- From **security** property:
  - Let us fix any $m \in M$, then consider the map K → C**:**
    - If $|K| < |C|$, then K → C cannot be surjective (onto)
      - Not all ciphertexts in C can be mapped to a key
    - If K → C is not surjective, it violates perfect secrecy
      - Let $c' \in C$ be some unmapped ciphertext in K → C.
      - c′ is not a valid ciphertext for m under any possible key
      - Whenever we see c' as a ciphertext produced, we will know that the plaintext can't be m, thereby violating perfect secrecy.
    - So, it must be $|K| \geq |C|$.

133

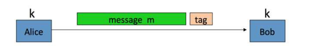
Limitations of Perfect Secrecy

- Completing the proof:
  - (Part 1) From Correctness property: $|M| \leq |C|$
  - (Part 2) From security property: $|C| \leq |K|$
  - Putting it together: $|M| \leq |C| \leq |K|$. QED.
- We have proved that $|K| \geq |M|$
- Implication: Key space at least as large as message space

Can't repeat the pad bits across multiple encryptions!

134

Integrity: Message Authentication Codes (MAC)



Generate tag:
tag ← S(k, m)

Verify tag:
V(k, m, tag) $\stackrel{?}{=}$ \`yes'

Def: **MAC** I = (S,V) defined over (K,M,T) is a pair of algs:
- S(k,m) outputs t in T
- V(k,m,t) outputs \`yes' or \`no'

Slide courtesy: Teodora Baluta (CS3235 Fall 2018)

135

Formal Security Goal for MAC

- **Existential Forgery** under Chosen Message Attack

Chosen message attack: for m_1, m_2,…,m_q attacker get $t1$, $t2$, …,t_q
Existential Forgery: when an attacker can create a tag for another message m

Slide courtesy: Teodora Baluta (CS3235 Fall 2018)

136

Formal Security Goal: Existential Forgery under CMA

- The adversary A wins if:

$$\boxed{\exists m, \Pr_{k}[V(k,m,t) \rightarrow yes\ ] > negl}$$

- A MAC is **EFCMA-secure** if for all adversaries A:

$$\boxed{\forall m, \Pr_{k}[V(k,m,t) \rightarrow yes\ ] \leq negl}$$

137

One Bit MAC

- The tag space size $|T| = 2^n$
- Let's try to create a one-bit MAC, i.e., n=1
- $S\ (m,k) \coloneqq$
  - m
  - $m \oplus k$
- Is existential forgery property true for it?
  - No. Why?
    - The CMA attacker can get the tag for m=0
    - Flip the bit, it's a valid tag for m=1
    - So, attacker wins!

138

Perfectly Secure MAC

$$\boxed{S\ (m,k) \coloneqq (k_a \cdot m + k_b)\ mod\ p}$$

- p is a prime known publicly
- $\mathbb{Z}_p \coloneqq \{0, \dots, p-1\}$
- $k_a \in \mathbb{Z}_p$, and $k_b \in \mathbb{Z}_p$ chosen uniformly
- (k_a, k_b) are the shared secret key k

139

Perfectly Secure MAC in $p = 2$

$$\boxed{S\ (m,k) \coloneqq (k_a \cdot m + k_b)\ mod\ p}$$

- Say you're given $(m,t) \coloneqq (0,1)$
  - You don't know randomly chosen (a,b)
- Can you forge tag for $m = 1$?
  - No
- Why? Intuitively…
  - 2 different values of (a,b) are correct
  - Both equally likely
  - Each gives a different tag for $m = 1$
  - Guessing correct tag has prob. is $\frac{1}{2}$
  - No better than randomly guessing the tag bit

k_a	k_b	m	T= S(m)
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

140

Perfectly Secure MAC in $p = 2$

$$\boxed{S\ (m,k) \coloneqq (k_a \cdot m + k_b)\ mod\ p}$$

- What's making this work?
- **Formally**: $\forall m, m'$ and $\forall t, t'$

$$\boxed{\Pr_{a,b}[S(m) = t \wedge S(m') = t'] = \frac{1}{|T|^2}}$$

- Randomness over choices of (a,b)
- Verify yourself the fact above
  - Each probability is ¼ in example

k_a	k_b	m	T= S(m)
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

Research Challenge (Optional):
Generalize & Prove that probability statement holds for arbitrary prime 'p'

141

Size of Key Space $|\mathrm{K}|$?

- What is minimum bit size for secret keys for n-bit MAC?

Keys should be at least 2n for n-bit perfectly secure MAC

- Proof (sketch) coming up…
- Let's first revisit the definition of existential forgery

$$\boxed{\Pr[Success] = \Pr_{k}[V(k,m,t) \rightarrow yes\ ] \leq negl}$$

- Prob. is for the *best guess* of $\boldsymbol{t}$ any adversary can make for any m
- What should attacker's probability of success be?
  - It should be $\frac{1}{2^n}$ at most. Why?
    - Adversary can always guess right tag with probability $\frac{1}{|T|} = \frac{1}{2^n}$
- **Lemma 0**: $\Pr[Success]\ \leq \frac{1}{2^n}$

142

Proof (cont…)

- $\mathrm{K}(\mathrm{m_0}, \mathrm{t_0}) \coloneqq$ **valid** set of keys for seen msg-tag pair $(\mathrm{m_0}, \mathrm{t_0})$
  - Defn: Valid keys are those which produce tag t for m
- **Lemma 1**:

$$\boxed{|\mathrm{K}(\mathrm{m_0}, \mathrm{t_0})| \geq |\mathrm{T}| = 2^n}$$

K (0,1)

a	b	m	T= S(m)
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

- Why? (Proof)
  - Suppose no, i.e., $|\mathrm{K}(\mathrm{m_0}, \mathrm{t_0})| < |\mathrm{T}|$.
  - Then, adversary can guess the right key with probability $> \frac{1}{2^n}$ and will guess the right tag for the challenge message.
  - The Pr[Success] $> \frac{1}{2^n}$, violating Lemma 0 (A contradiction!)

143

Proof (cont…)

- **Lemma 2**:

$$\boxed{\forall (\mathrm{m}, \mathrm{t}), \frac{|\mathrm{K}(\mathrm{m}, \mathrm{t})|}{|\mathrm{K}|} \leq \frac{1}{2^n}}$$

K (0,1)

a	b	m	T= S(m)
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

- Why? (Proof)
  - Consider what if not: $\exists(m,t)$ for which $|\mathbf{K}\ (m,t)\ | > \frac{|\mathrm{K}|}{2^n}$
    - Then, probability that $S(m,k) = t$ will be $> \frac{1}{2^n}$ (since keys are chosen uniformly at random)
    - Violates Lemma 0, as attacker can always output that tag t for the message m, and $\Pr[Success]$ is better than $\frac{1}{2^n}$

144

Proof (cont…)

- For any (m_0, t_0), how many of keys $|\mathrm{K}(\mathrm{m_0}, \mathrm{t_0})|$ make it valid?
- By Lemma 2, $\boxed{\frac{|\mathrm{K}(\mathrm{m_0},\mathrm{t_0})|}{|K|} \leq \frac{1}{2^n}}$
- **Attack**: Given some (m_0, t_0), adv. outputs tag for m_1 as follows:
  - Uniformly at randomly choose a $k \leftarrow \mathrm{K}\,(\mathrm{m_0}, \mathrm{t_0})$
  - Output $(m_1, S_k(m_1))$

Randomly pick $k \in \mathrm{K}(0,1)$ →
Output for that k:
$(m_1 = \mathbf{1}, S_k(m_1) = \mathbf{0})$

a	b	m	T= S(m)
0	0	0	0
0	1	0	1
1	0	0	0
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0

EXAMPLE

Gets lucky in picking correct key with probability $\frac{1}{|\mathrm{K}(\mathrm{m_0},\mathrm{t_0})|}$

145

Proof (cont…)

- Given some (m_0, t_0), adversary forges tag for m_1 as follows:
  - Uniformly at randomly choose a $k \leftarrow |\mathrm{K}(\mathrm{m_0}, \mathrm{t_0})|$ and output $(m_1, S_k(m_1))$
- By Lemma 2, $\boxed{\frac{|\mathrm{K}(\mathrm{m_0},\mathrm{t_0})|}{|K|} \leq \frac{1}{2^n}}$
- $\Pr[Success] = \frac{1}{|\mathrm{K}(\mathrm{m_0},\mathrm{t_0})|}$ of guessing the key used correctly
- $\frac{1}{|\mathrm{K}(\mathrm{m_0},\mathrm{t_0})|} \geq \frac{2^n}{|\mathrm{K}|}$ (by rearranging terms in Lemma 2)
- $\frac{1}{2^{\mathrm{n}}} \geq \frac{1}{|\mathrm{K}(\mathrm{m_0},\mathrm{t_0})|}$ (by rearranging terms in Lemma 1)
- So, $\frac{1}{2^{\mathrm{n}}} \geq \frac{1}{|\mathrm{K}(\mathrm{m_0},\mathrm{t_0})|} \geq \frac{2^n}{|\mathrm{K}|} \quad \Rightarrow \quad \boxed{|K| \geq 2^{2n}}$

Any one-bit MAC needs at least 2 bit keys

A n-bit MAC needs 2n bit keys

146

Key Takeaways

- Precisely defined Threat Models
- Symmetric Key Constructions are feasible!
- But, perfect security has **impractical** key sizes
  - Perfect Secrecy has $|K| \geq |M|$
  - Perfect MACs have $|K| \geq |T|^2$
- Key lengths as large as message /tag length
  - **Can't reuse key bits** and preserve security claims!

147

Practical Symmetric Encryption

- Use **Computational** Hardness Assumptions

$\boxed{\mathrm{Enc}(\mathrm{k}, \mathrm{m}) \coloneqq \mathrm{PRG}(\mathrm{k}) \oplus \mathrm{m}}$

- If One-way Functions exist, then there exist:
  - Pseudorandom Generators (PRG) or stream ciphers
  - Cryptographic Hash Functions
  - Pseudorandom Permutations (PRP) or block ciphers
- Several constructions from these primitives
- If curious, please see optional lecture notes

148

Computational Hardness

- Assumption: The adversary is "efficient"
  - Or, has limited computation power
- The adversary is an arbitrary algorithm:
  - That can execute in polynomial # of steps
  - That has randomized, non-determ. Execution
  - Bounds queries in CPA/CMA to polynomial in $|K|$
- Such an adversary is still quite powerful
  - Covers all efficient attacks algorithms be definition
  - But, is less powerful than "perfect secrecy"

149

Computational Hardness Assumption: Discrete Log (DLOG)

- For an appropriately chosen group G
  - g is the generator or the group G
  - " $\cdot$ " is the group operator
  - g^a is repeated application of " $\cdot$ ", $g \cdot g \cdots$ (a times)
- Discrete Log (DLOG) Problem:
  - Given $A \in G$ chosen randomly
  - Difficult to find any $a \in \mathbb{Z}$ such that $g^a = A$
- There exist groups G in which DLOG is hard
  - Eg. $\mathbb{Z}_p^* \coloneqq \{1, .., \mathrm{p}-1\}$ with multiplication mod prime p

150

Key Exchange: Diffie Hellman Key Exchange [1976]

Alice
Pick fresh a

Bob
Pick fresh b

$M_1 = g^a \text{ (mod p)}$ →

← $M_2 = g^b \text{ (mod p)}$

$\boxed{\mathrm{K} = (M_2)^a = g^{ba} \text{ (mod p)}}$ $\boxed{\mathrm{K} = (M_1)^b = g^{ab} \text{(mod p)}}$

- Kerckhoff's Principle: (g, p, G) known to all
- Alice and Bob compute the same key K
- K should not be computable by adversary.

New Directions in Cryptography (1976)

151

Passive Adversary (Eve)

- Passive adv. can only see /**eavesdrop** traffic
- What does Eve see?
  - (g, g^a, g^b) for randomly chosen a and b
  - Breaking DHKE requires solving Computational DH:
    - **CDH**: Compute g^{ab} knowing only (g, g^a, g^b);
- **DLOG** must be at least as hard as **CDH**

152

Passive Adversary (Eve)

- Passive adv. can only see /**eavesdrop** traffic
- What does Eve see?
  - (g, g^a, g^b) for randomly chosen a and b
  - Breaking DHKE requires solving Computational DH:
    - **CDH**: Compute g^{ab} knowing only (g, g^a, g^b);
- **DLOG** must be at least as hard as **CDH**

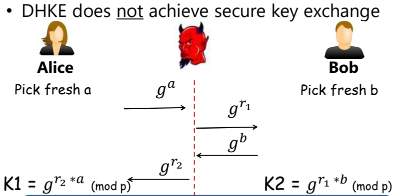
Show that if solving DLOG is easy, then CDH must be easy

- Assuming **CDH** is hard:
  - DH Key Exchange is secure against Eve

153

Active Adversary (Mallory)

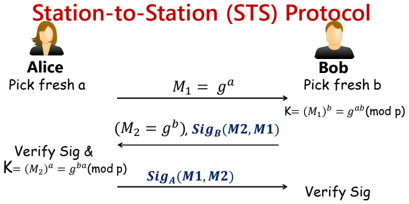
- Active Adversary can see and tamper traffic
- DHKE does not achieve secure key exchange



(K1, K2) are known to the adversary who can man-in-the-middle (MITM) communications freely

154

Adding Signatures to DHE: Station-to-Station (STS) Protocol



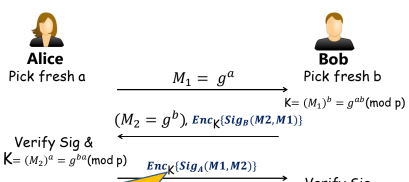
- STS is secure against both Eve and Mallory

Why does the signature help against MITM?

Security Properties: A Hierarchy of Authentication Specifications [Lowe'97]

155

The Full STS Protocol



Why encrypt with symmetric key?

[Not important] Additional Privacy: Hides that Alice and Bob are talking (signing)

Security Properties: A Hierarchy of Authentication Specifications [Lowe'97]

156

Takeaways (I)

- Without computational assumptions, secret key cryptography has impractical key sizes
- With computational hardness assumptions
  - Digital Signatures exist
  - DHKE Key Exchange secure against Eve
  - DHKE + Signatures ⇒ Authenticated Key Exchange
  - Authenticated Key Exchange ⇒ fresh symmetric keys

157

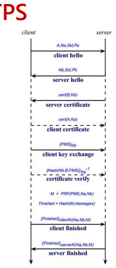
Takeaways (II)

- Why do we need KE protocols?
  - **A way to establish fresh shared secrets**
  - Even if we have pre-established secrets, we want to use refresh "keys" per session
- Remark: (Perfect) Forward Secrecy
  - Protect Encrypted information, even **if long-term key (for client and server) is compromised**
  - Idea: Generate session keys, and throw away messages used to generate sessions keys…

158

Secure Channel on the Internet: SSL + HTTP = HTTPS

- Used in HTTPS, SMTP, fax, …
- Originated in Netscape SSL 2.0 [1993]
- Revised name to TLS
- Many versions:
  - TLS 1.2 [2008]
  - TLS 1.3 [2018]



Paulson JSC'98

159

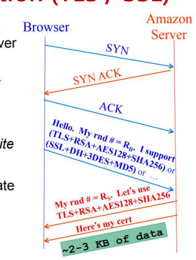
Chain Of Trusted Certificates

- Root CAs (e.g. GeoTrust)
  - Can designate Intermediate CA
    - E.g. Google Internet Authority
    - Restricted to signing certs for its subdomains
- Where does the browser start trusting?
  - Root CA's certificates are baked in your browser
  - ~50
- Who are the root CAs for the web?
  - Symantec (GeoTrust) – 38%
  - Comodo – 20%
  - GoDaddy – 13%, GlobalSign – 10%

160

HTTPS Connection (TLS / SSL)

- Browser (client) connects via TCP to Amazon's **`HTTPS`** server
- Client picks 256-bit random number R_B, sends over list of crypto protocols it supports
- Server picks 256-bit random number R_S, selects *cipher suite* to use for this session
- Server sends over its certificate
- (all of this is in the clear)
- ***Client now validates cert***

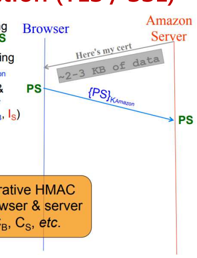


Slide Courtesy (thanks!): Vern Paxson

161

HTTPS Connection (TLS / SSL)

- For RSA, browser constructs long (368 bits) "Premaster Secret" **PS**
- Browser sends PS encrypted using Amazon's public RSA key K_{Amazon}
- Using PS, R_B, and R_S, browser & server derive symm. *cipher keys* (C_B, C_S) & MAC *integrity keys* (I_B, I_S)
  - One pair to use in each direction



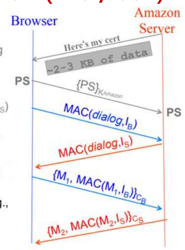
PS is used as the key for iterative HMAC invocations on R_B || R_S. Browser & server use the output to generate C_B, C_S, *etc.*

Slide Courtesy (thanks!): Vern Paxson

162

HTTPS Connection (TLS / SSL)

- For RSA, browser constructs long (368 bits) "Premaster Secret" **PS**
- Browser sends PS encrypted using Amazon's public RSA key K_{Amazon}
- Using PS, R_B, and R_S, browser & server derive symm. *cipher keys* (C_B, C_S) & MAC *integrity keys* (I_B, I_S)
  - One pair to use in each direction
- Browser & server exchange MACs computed over entire dialog so far
- If good MAC, Browser displays
- All subsequent communication encrypted w/ symmetric cipher (e.g., AES128) cipher keys, MACs
  - Messages also numbered to thwart replay attacks

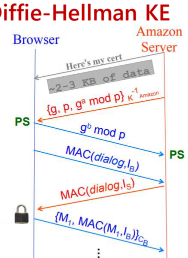


Slide Courtesy (thanks!): Vern Paxson

163

HTTPS Connection (TLS / SSL): Alternative via Diffie-Hellman KE

- For Diffie-Hellman, server generates random a, sends public params and g^a mod p
  - Signed with server's public key
- Browser verifies signature
- Browser generates random b, computes **PS** = g^{ab} mod p, sends to server
- Server also computes **PS** = g^{ab} mod p
- Remainder is as before: from PS, R_B, and R_S, browser & server derive symm. *cipher keys* (C_B, C_S) and MAC *integrity keys* (I_B, I_S), etc…



Slide Courtesy (thanks!): Vern Paxson

Please read SSL-handshake.pdf on Canvas

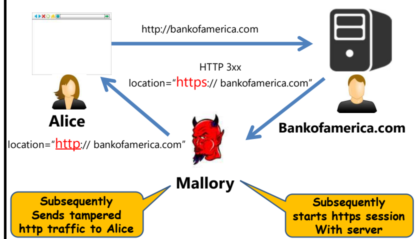
164

Assumptions in the threat model

- User is using a secure channel
- Crypto primitives are secure
- TLS protocol design is secure
- TLS protocol implementation is secure
- Certificate issuers are uncompromised
- Users check browser UI correctly
- Alice & Bob's secrets are secure
- Entities are authenticated correctly

165

HTTP Downgrade



166

Defense Against HTTP Downgrade

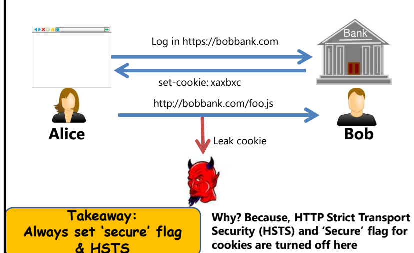
- HSTS: **HTTP Strict Transport Security**
- Idea: Server supplies a header over HTTPS

```
Transport-Security: max-age=31536000; includeSubDomains; preload
```

- Browser never issues any HTTP request to this site if it receives this header

167

Insecure Cookies



168

Secure Channel For Web Cookies?

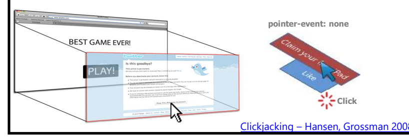
- Does the web have a secure channel for cookies?
- Confidentiality – Yes!
  - Over HTTPS only using 'Secure' keyword
    - Won't be sent over HTTP
  - Can be read by JS via DOM API
- Integrity – No!
  - Can be written by HTTP requests
    - E.g. Set-cookie: SID=bad; secure
    - It will override the previously set Secure cookie
  - Can be written / deleted via JavaScript
    - **evil.example.com** can set cookies for **example.com**

Cookies Lack Integrity: Real-World Implications

169

Clickjacking

- Pages can embed iframes from 3rd-party
  - Any site can host another in <iframe>
  - Frames can overlap
  - CSS controls the transparency, location of frames
- How to trick users?
  - E.g. `opacity : 0.1`, or `pointer-events: none`



Clickjacking – Hansen, Grossman 2008

170

Mixing HTTP and HTTPS

- Mixed Content
  - HTTP resources in HTTPS pages

https://example.com

`<script src=http://example.com/lib.js>`

Is this safe?

Attacker can corrupt JS and include payload

- What do browsers do for mixed content?
  - Legacy: Ignore, No security warning.
  - Recent: Block and present new UI indicators

171

Defenses Against Compromised Certs

- How to Detect If Being Served Bad Cert
  - Certificate Pinning
  - Certificate Revocation
  - Certificate Transparency
- Certificate Pinning
  - Browser "pins" or caches certain certificates after the first visit (e.g. Gmail.com)
  - Issues: How many and which certs to pin?

172

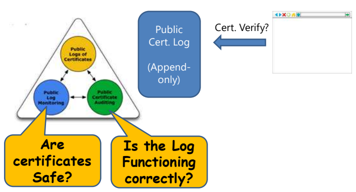
Certificate Revocation

- Idea: CA can revoke compromised certs.
- Supported by OCSP
  - CA signs a revocation list
  - Problems?
    - Time windows after compromise
    - Privacy
    - Implementation bugs (replay attacks)
  - Improvements: OCSP stapling (see Wikipedia)
    - Network costs increase

173

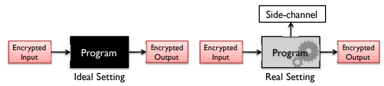
A Mitigation for Compromised Certs: Certificate Transparency

- Idea: Publicly audit all SSL certs.



174

What is a side-channel?



Side-channel: *A side-channel is an unintended source of information leakage that is not designed as the primary means of communication*

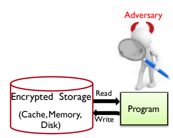
Courtesy: Shruti Tople

175

Types of side-channels

Attacker's knowledge in encrypted computation
- Program logic is public

Side-Channels
- Size of data
- Timing Channel
- Data Access Patterns
- Power Channel
- Sound
- Electromagnetic radition



Courtesy: Shruti Tople

176

Timing Channel

Cryptographic protocols: Exponentiation is implemented via square-and-multiply
RSA has $y = g^k \mod N$

Algorithm 1 RSA - Left-to-Right Binary Algorithm

Inputs: $g, k = (k_{t-1}, \cdots, k_0)_2$ **Output:** $y = g^k$
Start:
$R_0 \leftarrow 1; R_1 \leftarrow g$
for $j = t-1$ *downto* 0 **do**
  $R_0 \leftarrow (R_0)^2$
  if $k_j = 1$ **then** $R_0 \leftarrow R_0 R_1$ **end if**
end for
return R_0

Always executes (1 CPU cycle)

Executes conditionally (1 CPU cycle)

Leaks key via timing channel

Courtesy: Shruti Tople

177

Fixing the Timing Channel

Same computation on both the branches

Is there any other leakage channel?
- YES

Memory access patterns reveal key bits
- Order of accessing R_0 and R_1
- Need to be fixed using deterministic address patterns, or randomization

Algorithm 2 Montgomery Power Ladder Algorithm

Inputs: $g, k = (k_{t-1}, \cdots, k_0)_2$ **Output:** $y = g^k$
Start:
$R_0 \leftarrow 1; R_1 \leftarrow g$
for $j = t-1$ *downto* 0 **do**
  if $k_j = 0$ **then** $R_1 \leftarrow R_0 R_1;\ R_0 \leftarrow (R_0)^2$
  else $R_0 \leftarrow R_0 R_1;\ R_1 \leftarrow (R_1)^2$
  end if
end for
return R_0

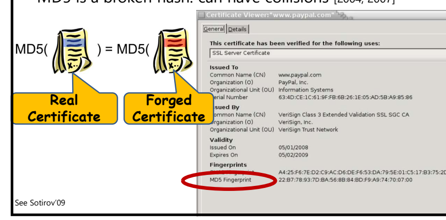
Leakage via access patterns

Courtesy: Shruti Tople

178

Broken Crypto Primitives: Broken MD5 leads to Forged Certs.

- Can attack the cryptographic signing [Sotirov et al.]
- MD5 is a broken hash: can have collisions [2004, 2007]



See Sotirov'09

179

Improper Use of Crypto Primitives

- MAC => integrity, Enc => confidentiality
- Which of these is a secure MAC+Enc scheme?

E (m) | MAC (m)

- Example 1: SSH
  - Encrypt – **and**- MAC
  - Clearly Insecure! (Why?)

E (MAC (m))

- Example 2: SSL (Used in HTTPS)
  - **MAC – then – encrypt**
  - Can be insecure
  - Encryption is malleable!
  - If curious, see [Padding Oracle Attacks]

MAC (E (m))

- Example 3: IPSec
  - **Encrypt - then - MAC**
  - Provably Secure

180